Introduction to Trigger and Data AcQuisition

1/3 – Base concepts

Bologna, mar 2021 Mauro Villa

Plan

- Basic DAQ concepts
- TDAQ Infrastructure & FOOT examples
- Advance TDAQ & FOOT TDAQ specific

Acknowledgment

- Lecture based on ISOTDAQ school material
  - Material and ideas taken from: Andrea Negri, Wainer Vandelli, Roberto Ferrari, Clara Gaspar, Niko Neufeld, Lauren Tompkins, ...
- Errors and flaws are mine

Introduction

- Aim of this lesson is to introduce the **basic DAQ concepts** avoiding as many technological details as possible
  - The following lectures will cover these aspects

Outline

- Introduction
  - What is DAQ?
  - Overall framework
- Basic DAQ concepts
  - Digitization, Latency
  - Deadtime, Busy, Backpressure
  - De-randomization
- Scaling up
  - Readout and Event Building
  - Buses vs Network

What is DAQ? [Wikipedia]

- **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
- Components:
  - **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? [Wikipedia]

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

What is acquired?

An acquired value is nothing without a context

- The 5W rule of journalism apply here!
  - What, Who, When, Where, Why
  - What: value actually recorded
  - Who: which sensor has been read?
    very important if you have millions of channels…
  - When: timestamping!!
    very important if you run for days at kHz frequency…
  - Where: you have to know the external conditions
  - Why: never forget your final goal

TDAQ

What is DAQ?

- DAQ is an **heterogeneous** field
  - 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 a permanent storage
- What does it mean interesting?
  - When does this happen?
- We need a trigger

Trigger

- Either selects interesting events or rejects boring ones, in real time
  - **Selective**: efficient for “signal” and resistant to “background”
  - **Simple** and **robust**
  - **Quick**

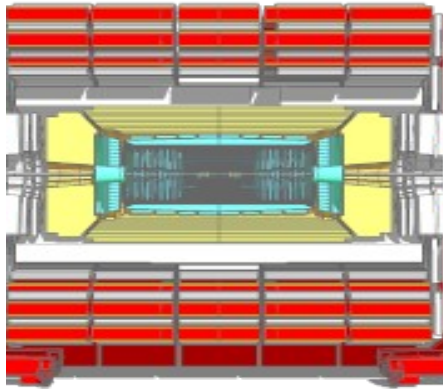
- With minimal *controlled* **latency**
  - time it takes to form and distribute its decision
- The trigger system generates a prompt signal used to start the data-acquisition processes
  - To be distributed to front end electronics

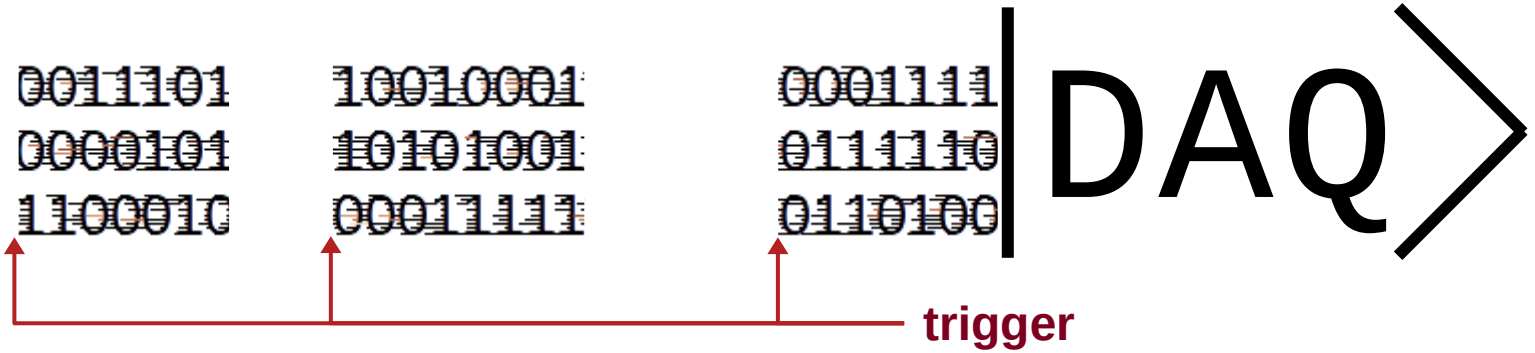
Double paths

- **Trigger** path
  - From dedicated detectors to trigger logic
- **Data** path
  - From all the detectors to storage
  - On positive trigger decision

Trigger(less)

- **Triggered (data pull)**: data is readout from detector only when a trigger signal is raised

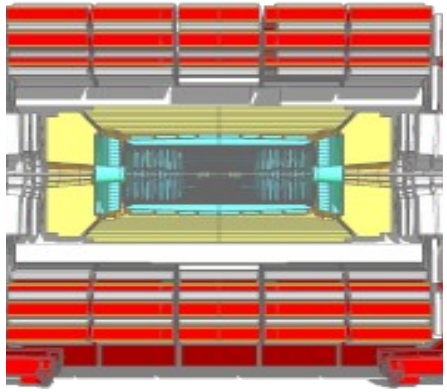


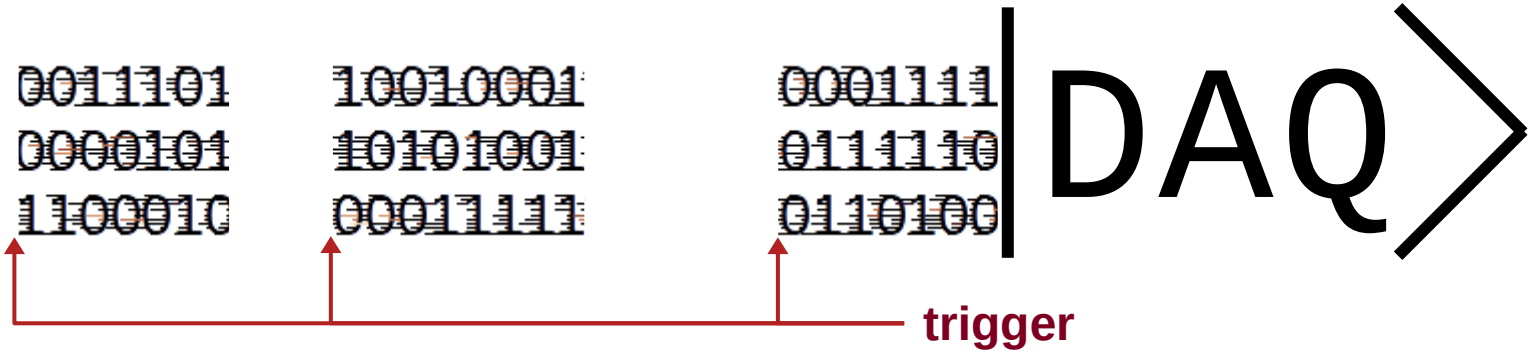


- **Triggerless (data push)**: the detector push data at its speed and the downstream daq must keep the pace

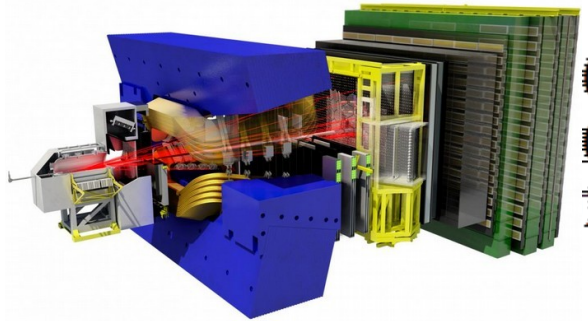
Trigger(less)

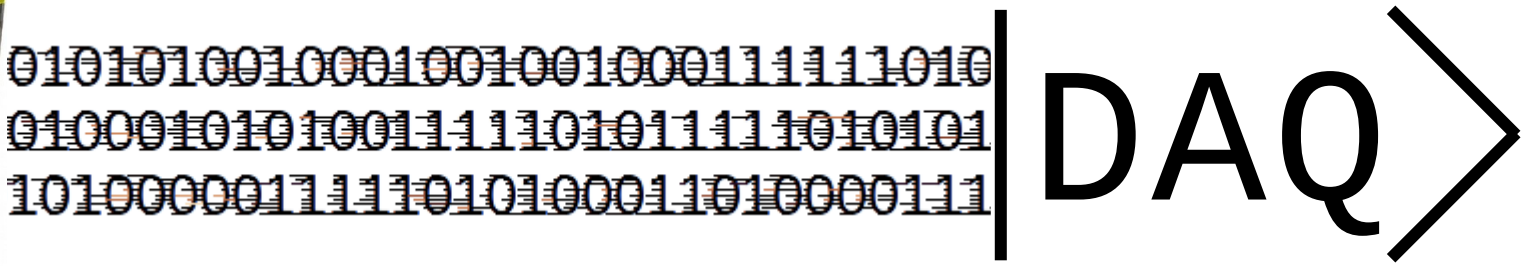
- **Triggered (data pull)**: data is readout from detector only when a trigger signal is raised





- **Triggerless (data push)**: the detector push data at its speed and the downstream daq must keep the pace





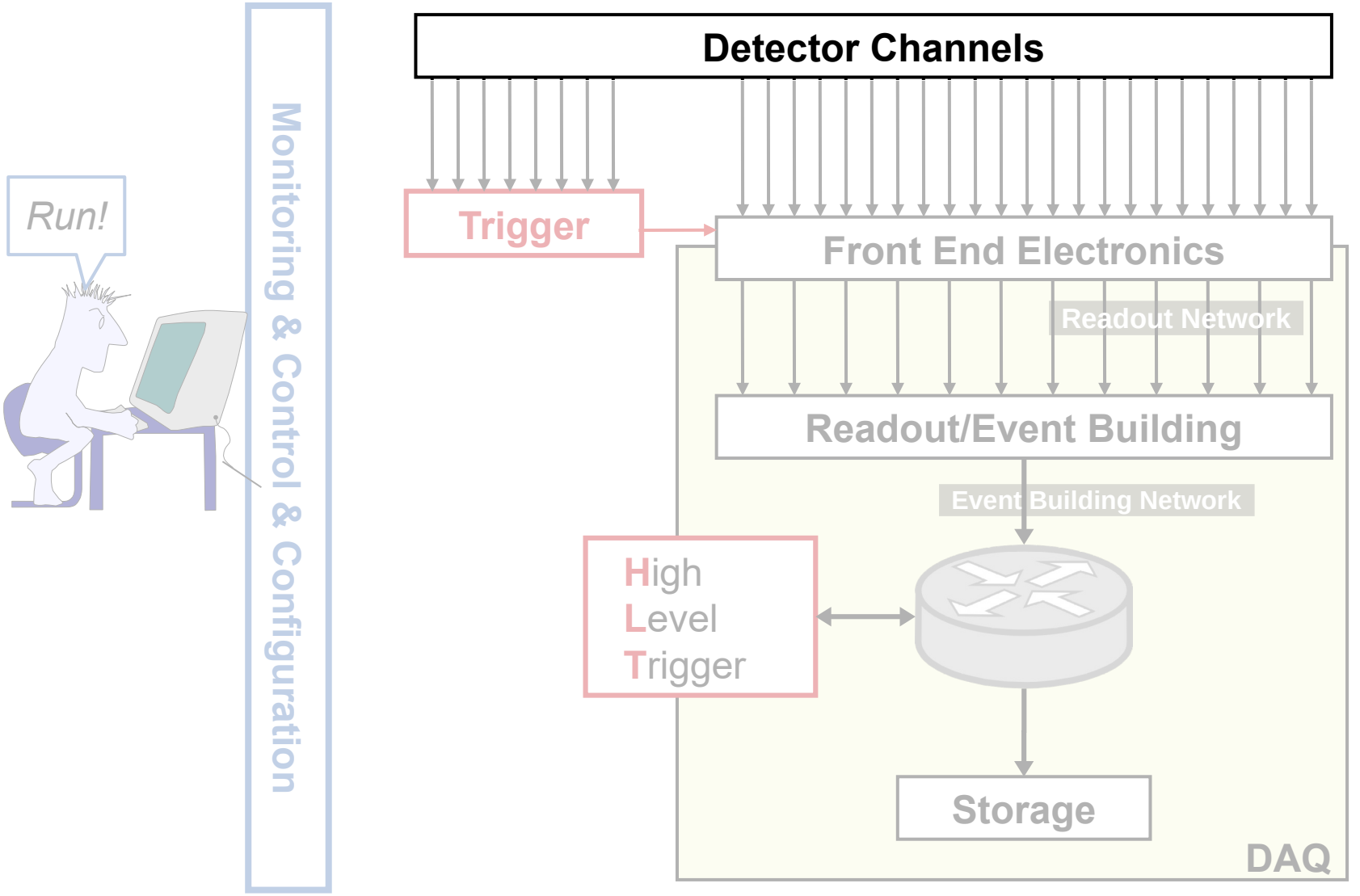
DAQ duties

- Gather data produced by detectors
  - **Readout**
- Form complete events
  - **Data Collection** and **Event Building**
- Possibly feed other trigger levels
  - **H**igh **L**evel **T**rigger
- Store event data
  - **Data Logging**

Data Flow

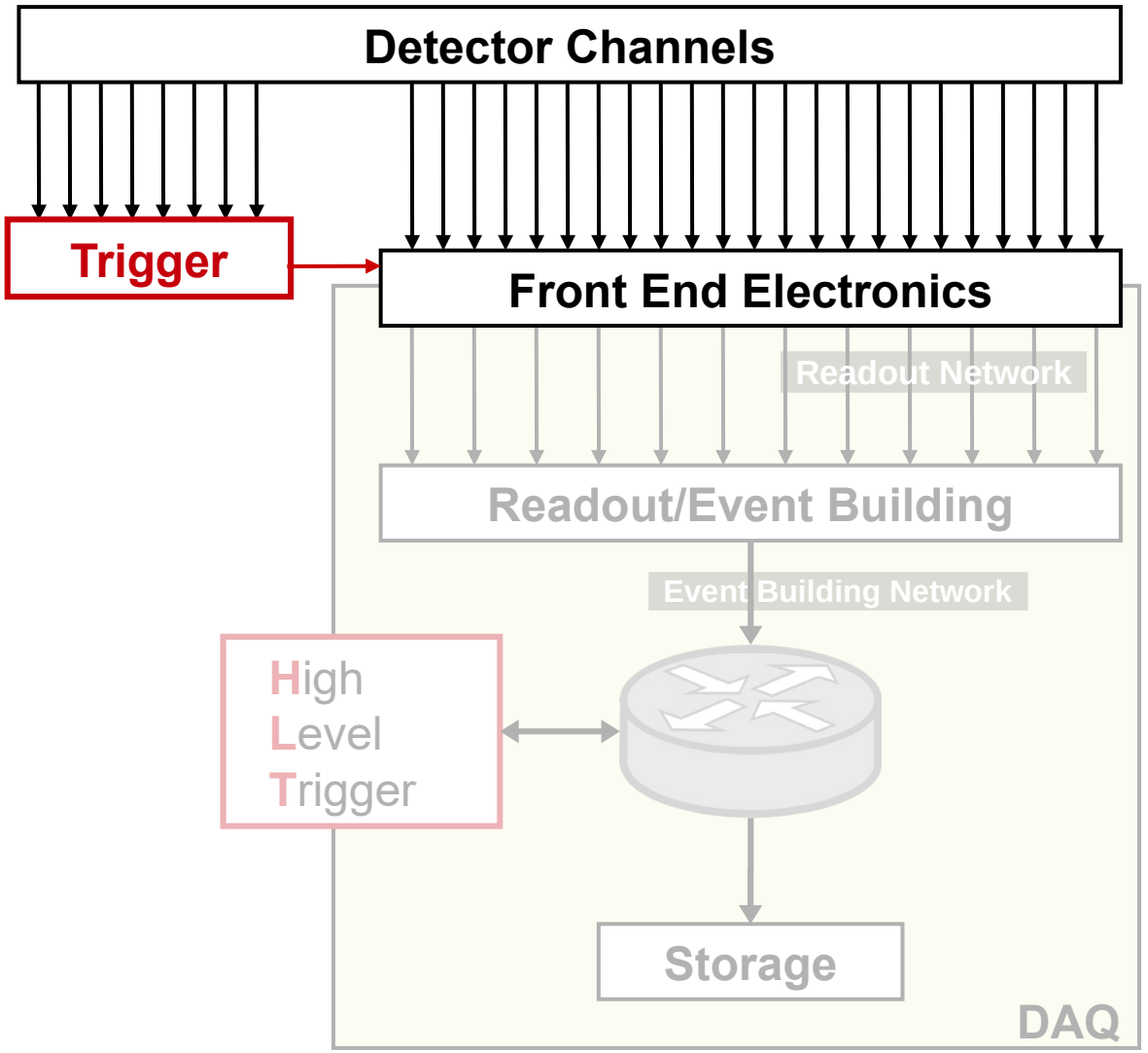
- Manage the operations
  - **Run Control**, **Configuration**, **Monitoring**

T-DAQ


Run!
Monitoring & Control & Configuration
Detector Channels
Trigger
Front End Electronics
Readout Network
Readout/Event Building
Event Building Network
High
Level
Trigger
Storage
DAQ

T-DAQ

Monitoring & Control & Configuration



T-DAQ

Monitoring & Control & Configuration

Detector Channels

Trigger

Front End Electronics

Readout Network

Readout/Event Building

Event Building Network

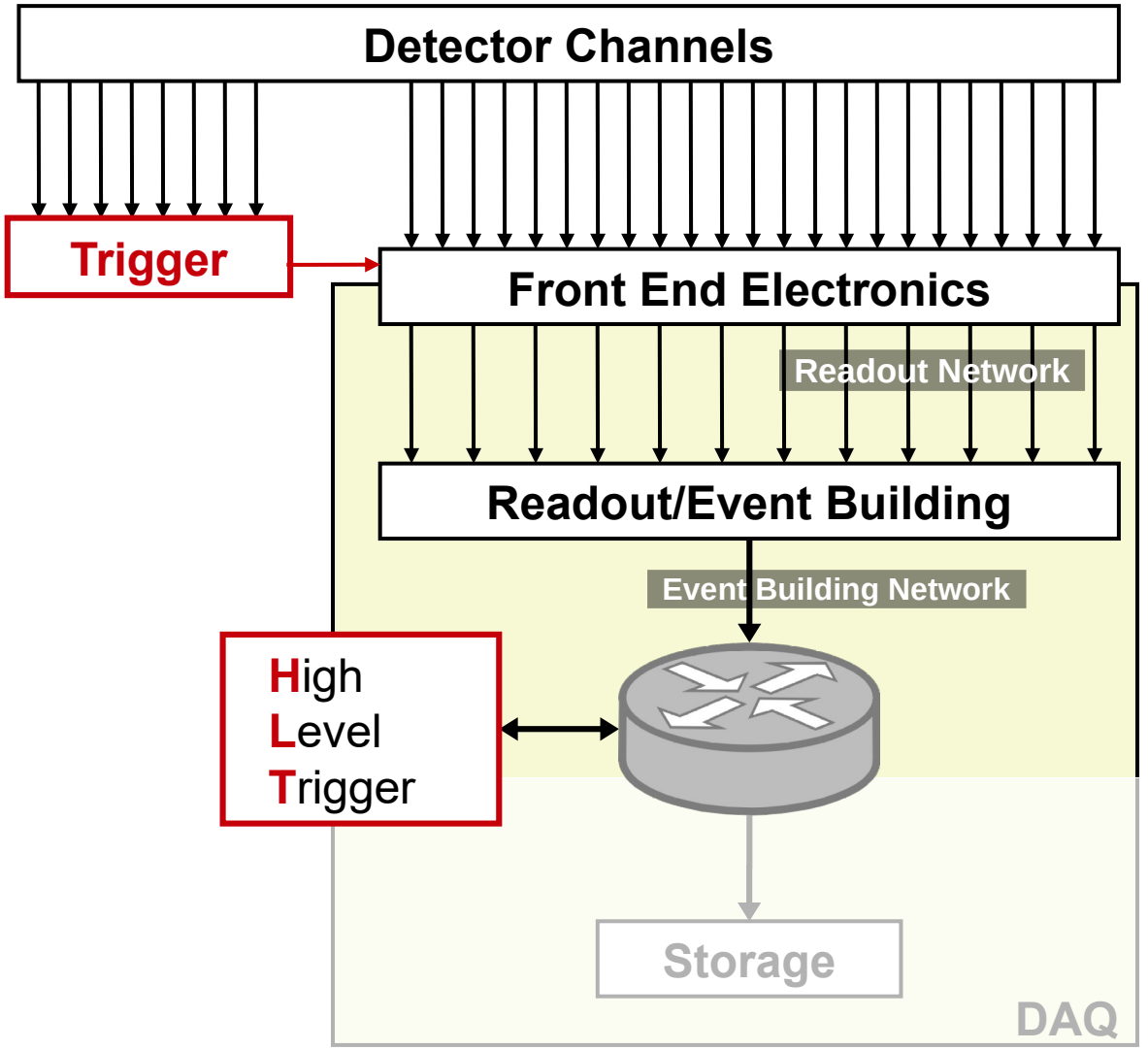
High
Level
Trigger

Storage

DAQ

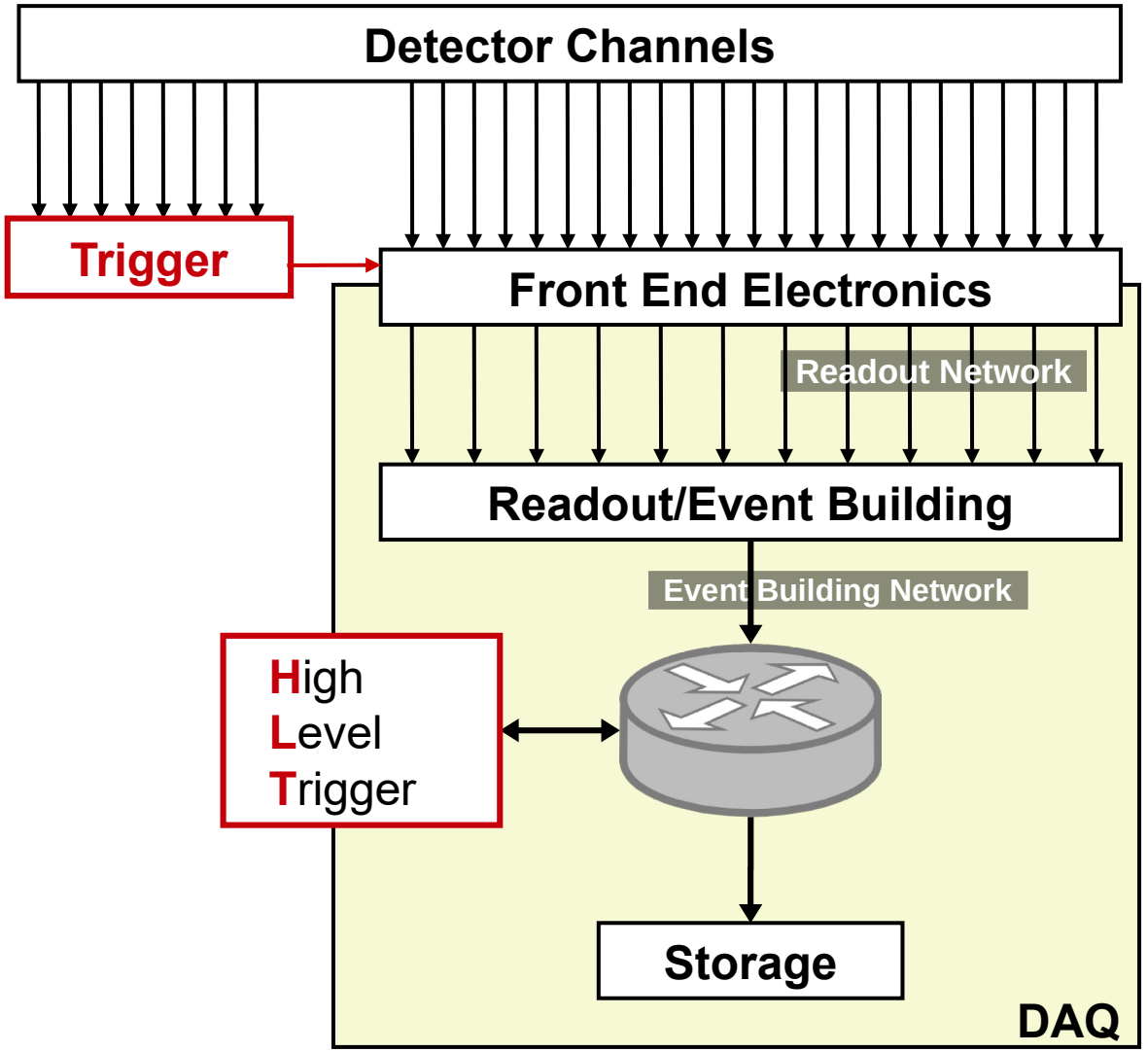
T-DAQ

Monitoring & Control & Configuration

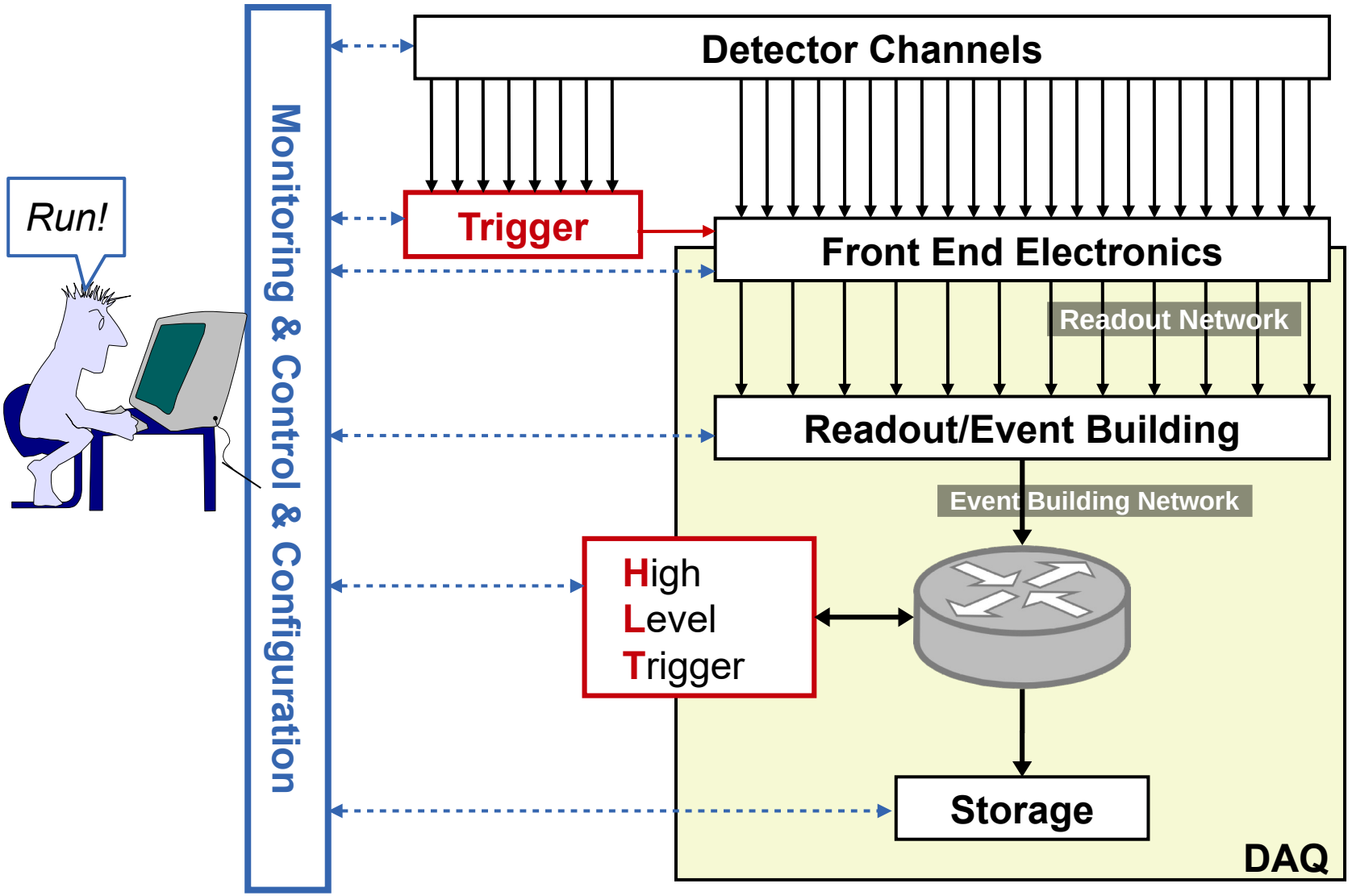


T-DAQ

Monitoring & Control & Configuration

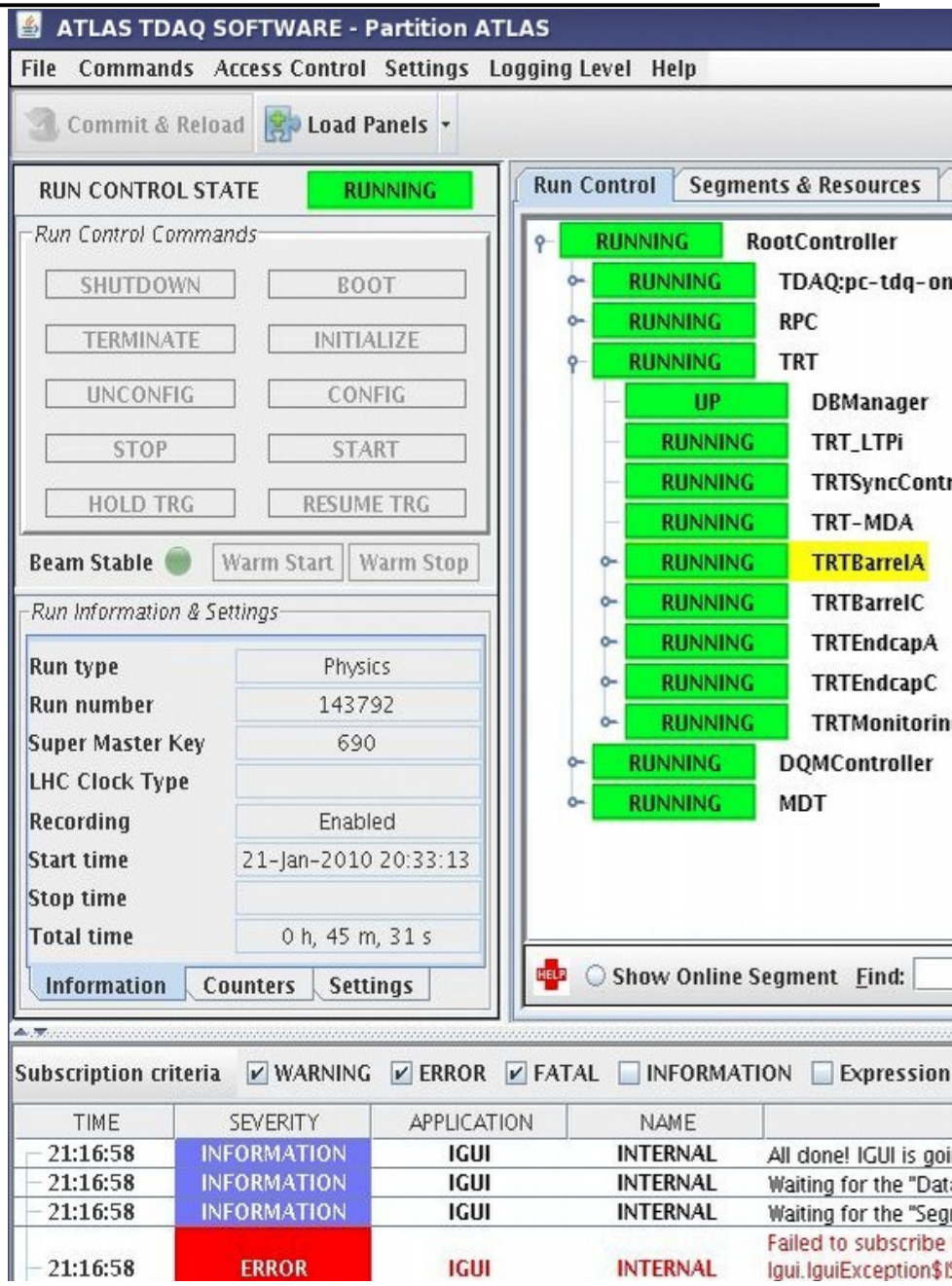


T-DAQ


Run!
Monitoring & Control & Configuration
Detector Channels
Trigger
Front End Electronics
Readout Network
Readout/Event Building
Event Building Network
High
Level
Trigger
Storage
DAQ

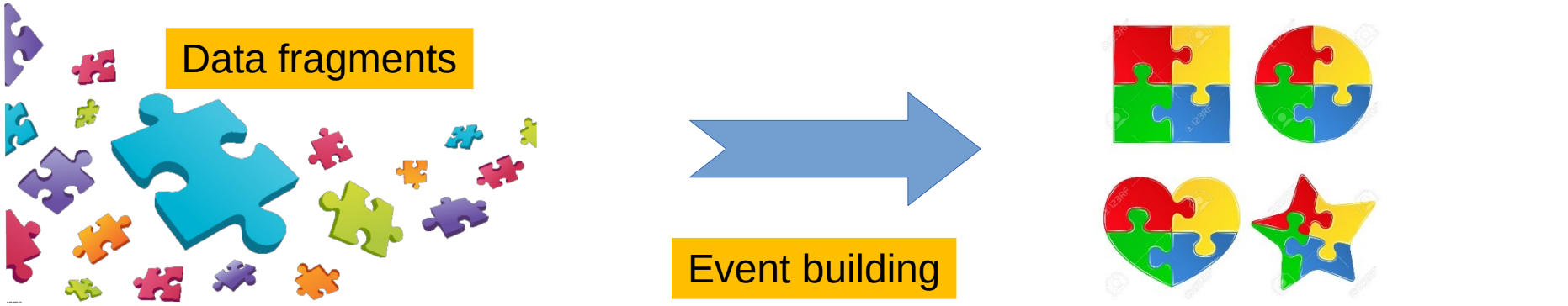
The glue of your experiment

- Configuration
  - The data taking setup
- Control
  - Orchestrate applications participating to data taking
  - Via distributed **F**inite **S**tate **M**achine
- Monitoring
  - Of data taking operations
  - What is going on?
  - What happened? When? Where?



Data fragments, events

- DAQ systems can be very complex
- Many sources of data (many subdetectors)
- **Data fragment**: collection of data of the same detector (uniform) related to a *given trigger event and/or of the same time-tag*
- **DAQ Event**: collection of data fragments from different detectors related to a *given trigger event and/or of the same time-tag*



Outline

- Introduction
  - What is DAQ?
  - Overall framework
- Basic DAQ concepts
  - Digitization, Latency
  - Deadtime, Busy, Backpressure
  - De-randomization
- Scaling up
  - Readout and Event Building
  - Buses vs Network

Via a toy model

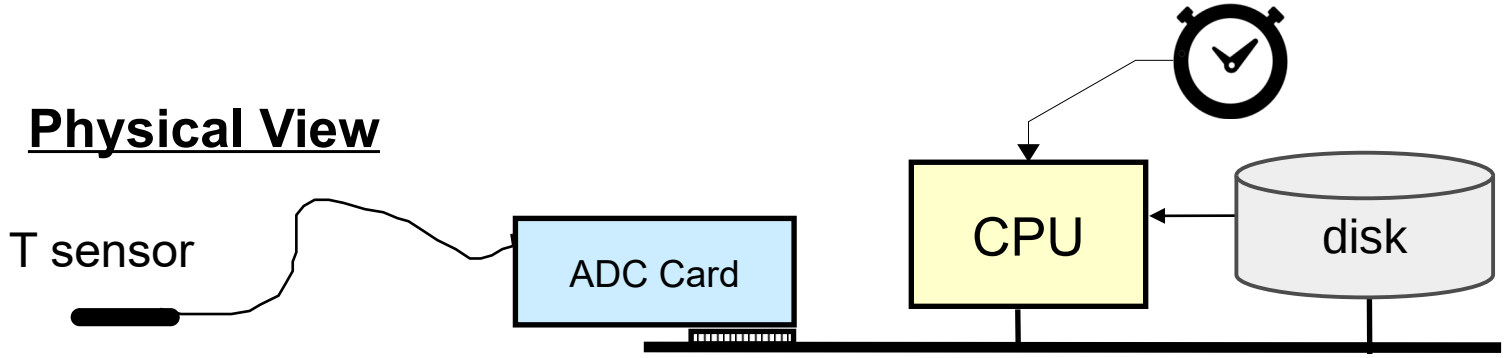
Two flavors of DAQ

- Periodic sampling
  - Periodic trigger (deterministic)
  - Domain of loggers & data push architectures

- Triggered sampling
  - A trigger condition is needed to acquire data
    - Directly from the detector
    - During the processing of the analog signal
    - On the digitized signals
    - On a fraction of the full event
  - Data pull architectures

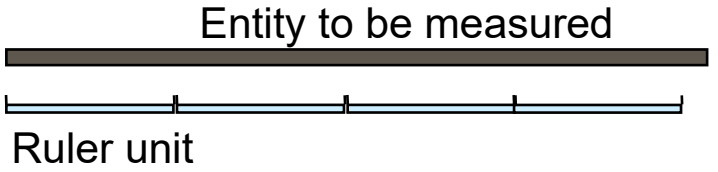
Basic DAQ: periodic trigger

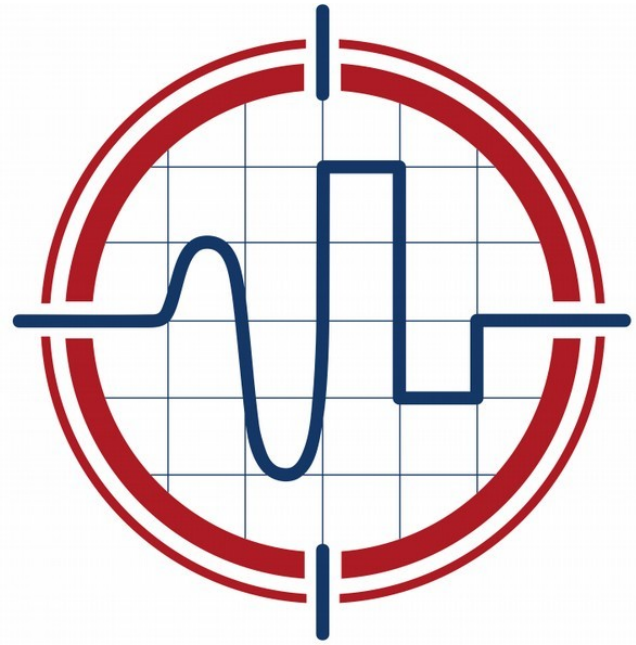
- Eg: measure temperature at a fixed frequency
  - Clock trigger
- ADC performs analog to digital conversion, **digitization** (our front-end electronics)
  - Encoding analog value into binary representation
- CPU does
  - Readout, Processing, Storage



Digitization

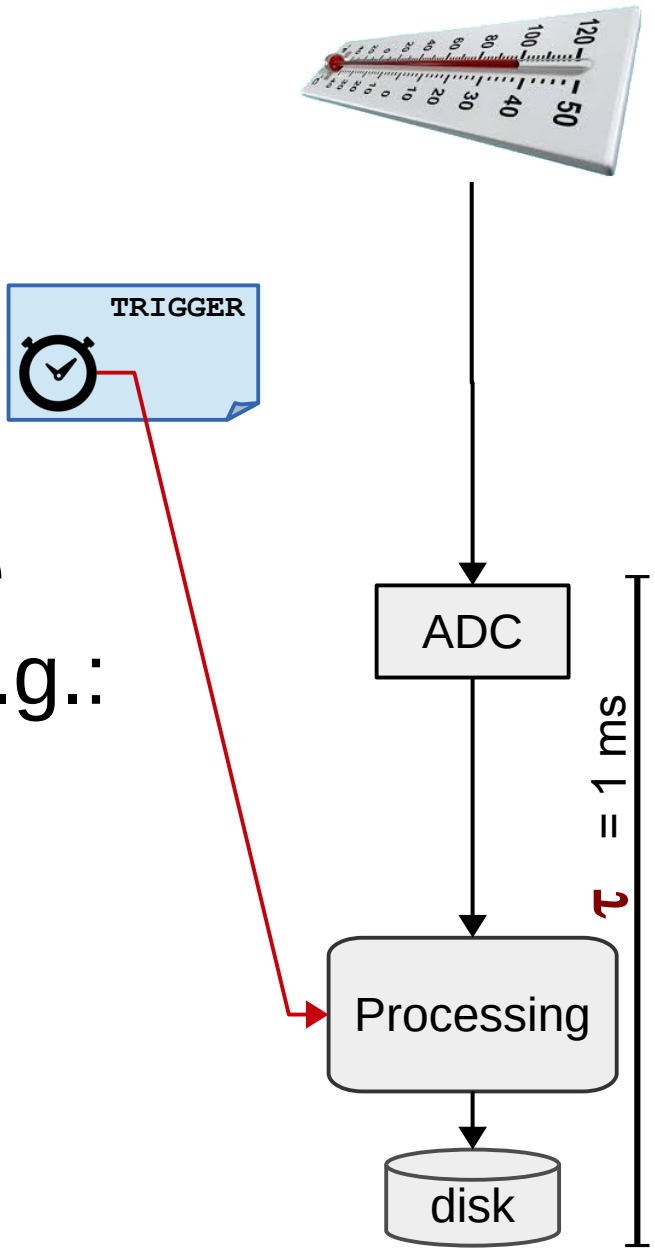
- Encoding an analog value into binary representation
  - Comparing entity with a ruler
- Different basic digitization
  - **ADC**: Analog to Digital Converter
    - (voltages or currents)
    - FADC: Flash ADC for waveforms
  - **QDC**: Charge to Digital Converter
  - **TDC**: Time to Digital Converter

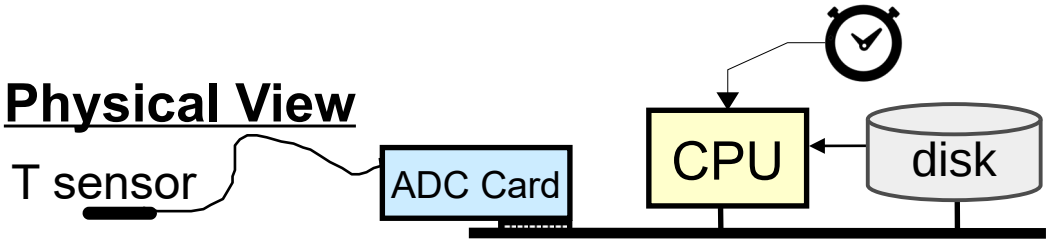




Basic DAQ: periodic trigger

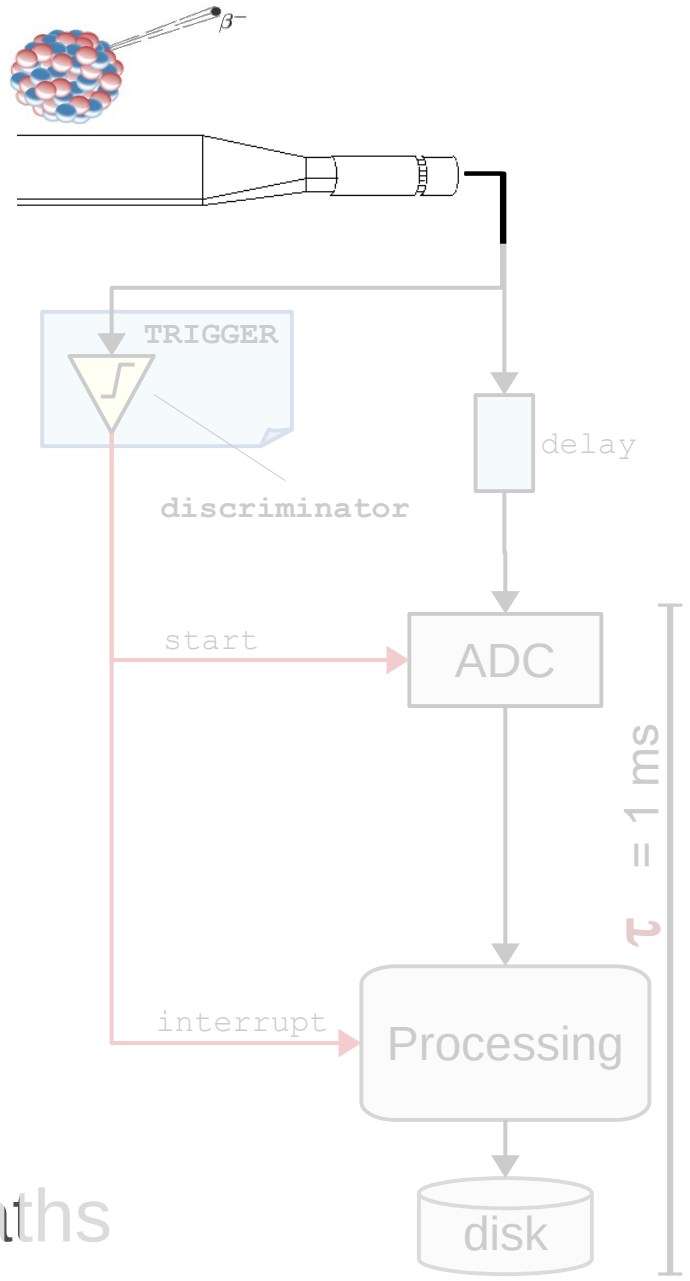
- System clearly limited by the time τ to process an "event"
  - ADC conversion + CPU processing + Storage
- The DAQ maximum sustainable rate is simply the inverse of τ, e.g.:
  - E.g.: τ = 1 ms $\rightarrow$ R = 1/τ = 1 kHz





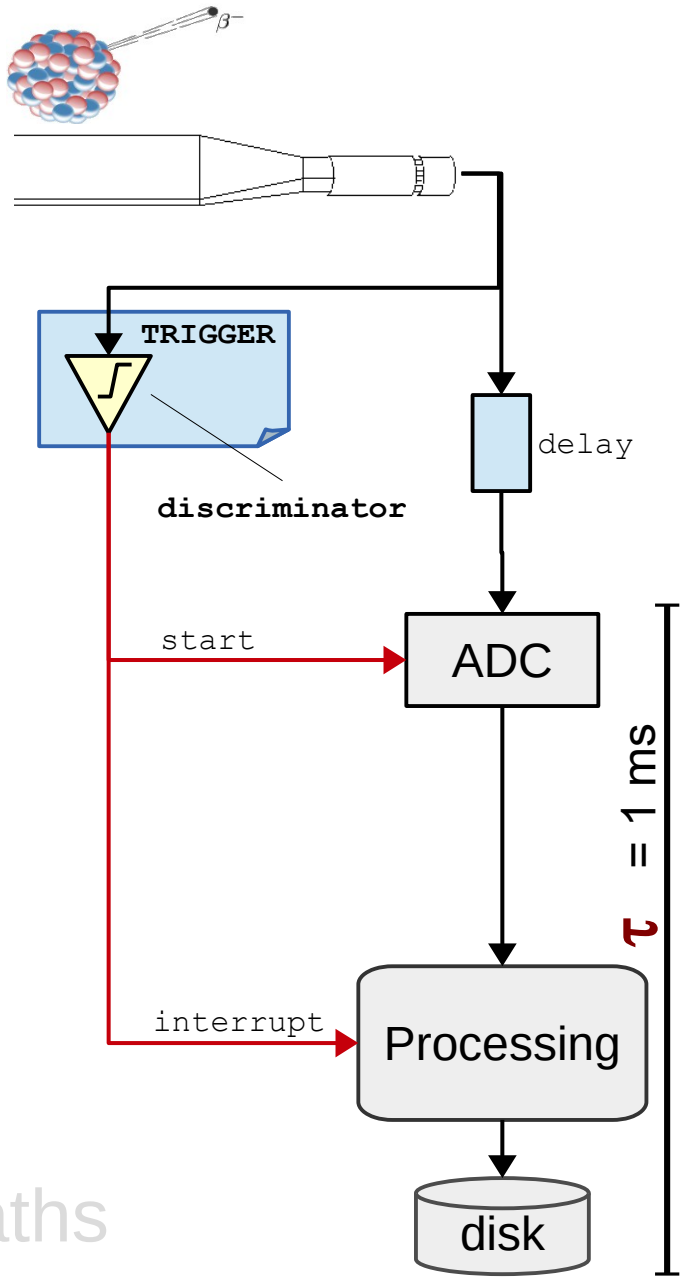
Basic DAQ: “real” trigger

- 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



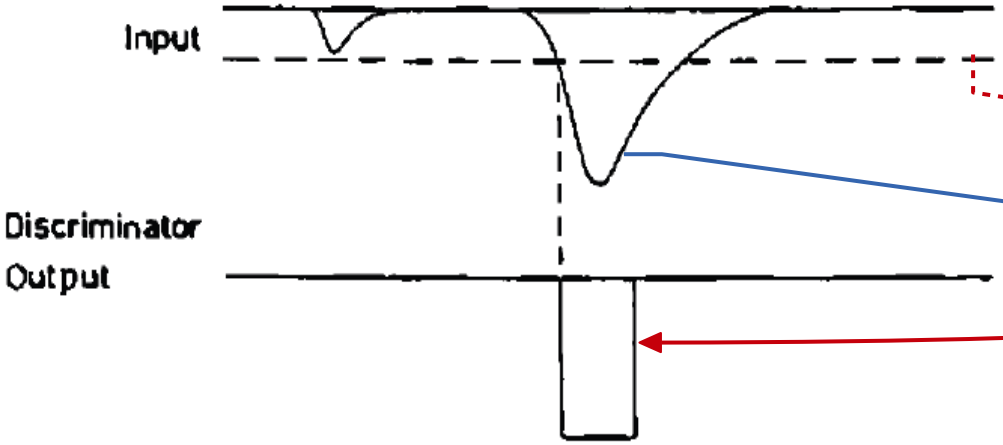
Basic DAQ: “real” trigger

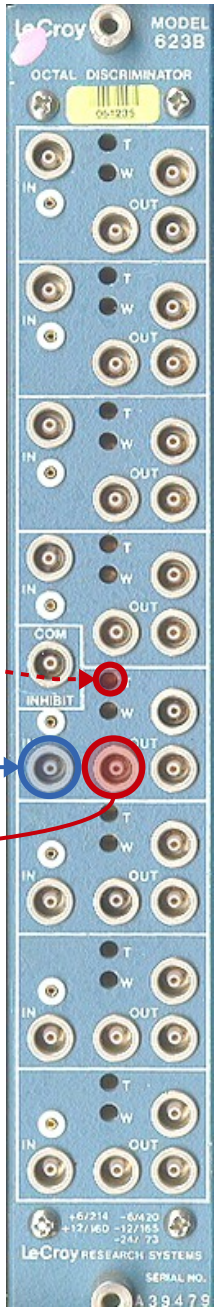
- Events asynchronous and unpredictable
  - E.g.: beta decay studies
- A physics trigger is needed
  - **Discriminator**: generates an output digital signal if amplitude of the input pulse is greater than a given threshold
- NB: delay introduced to compensate for the **trigger latency**
  - Signal split in trigger and data paths

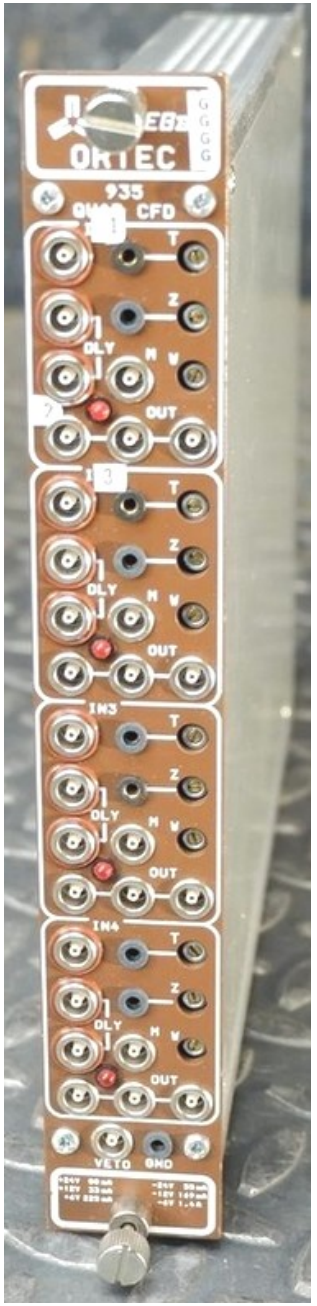


Discriminator

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

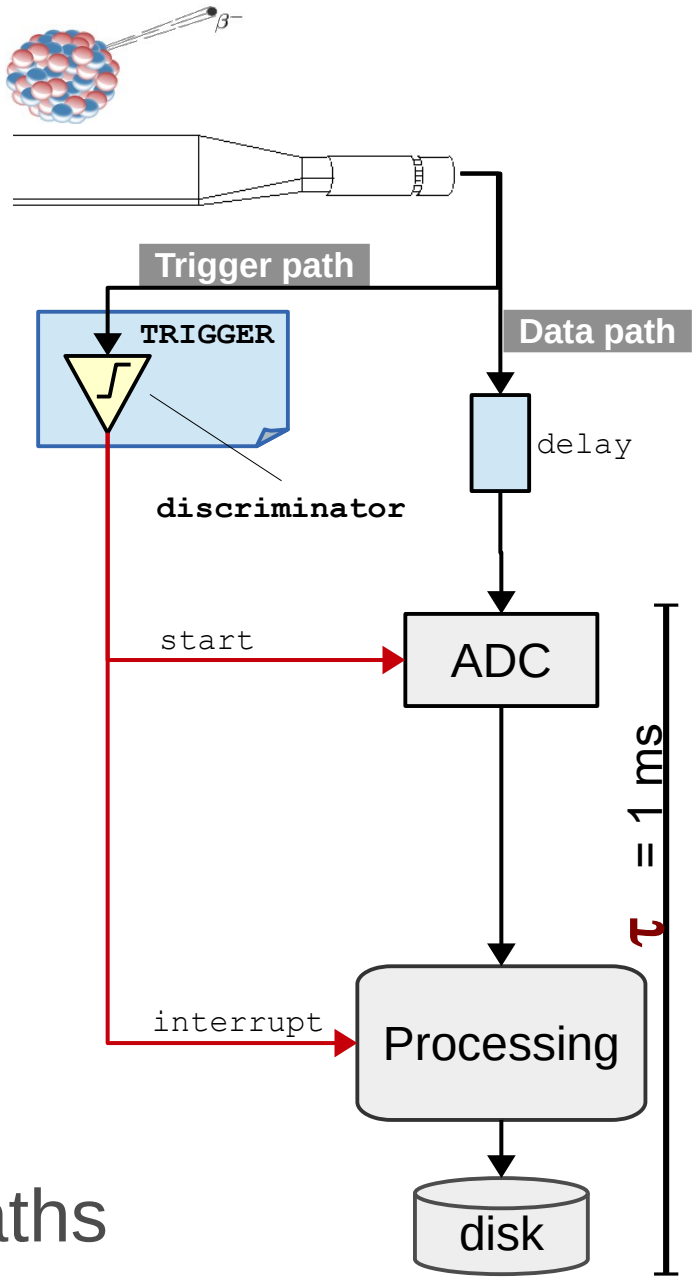






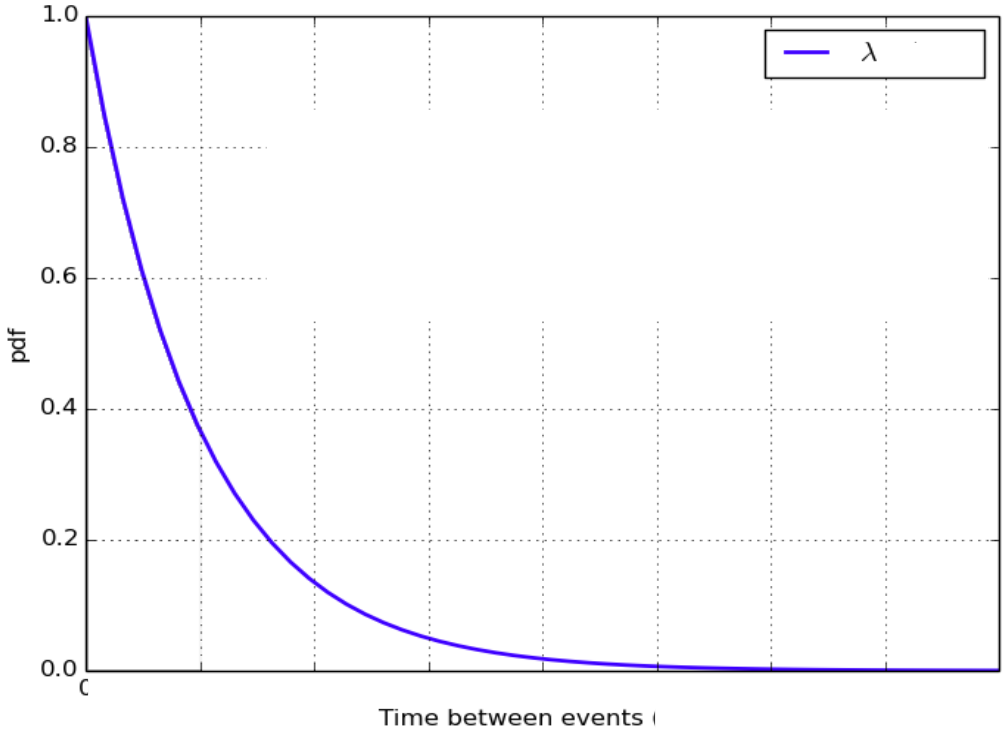
Basic DAQ: “real” trigger

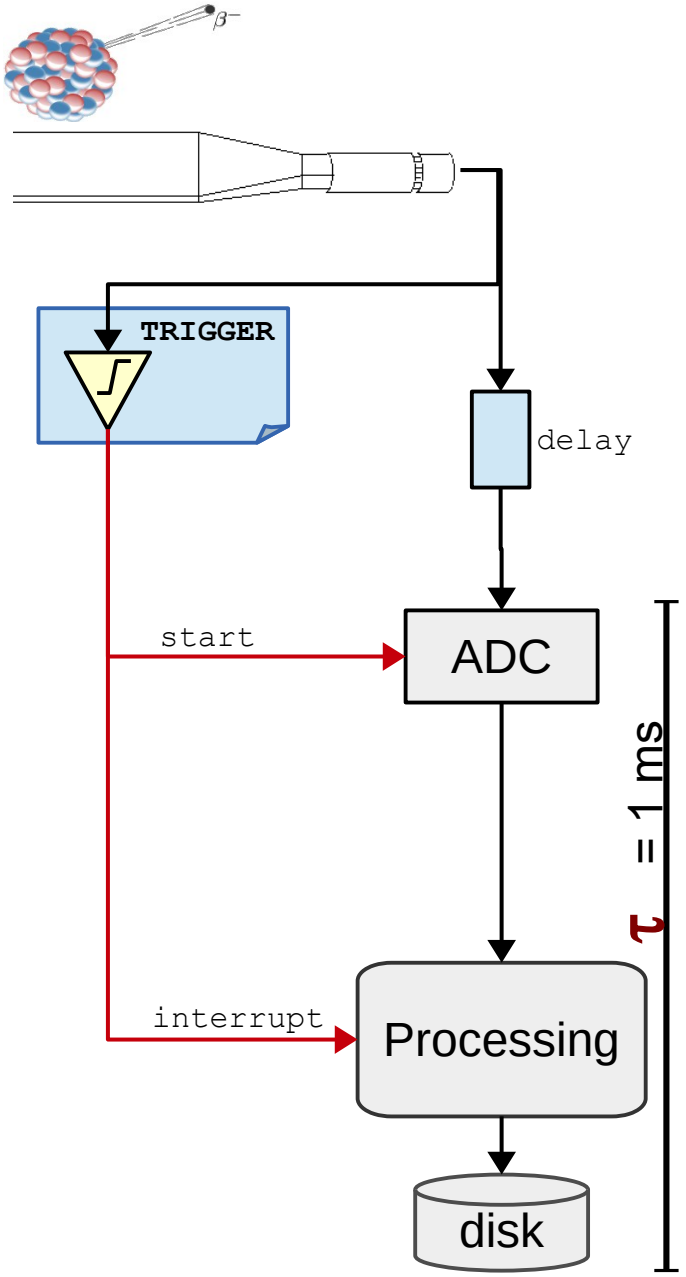
- Events asynchronous and unpredictable
  - E.g.: beta decay studies
- A physics trigger is needed
  - **Discriminator**: generates an output digital signal if amplitude of the input pulse is greater than a given threshold
- NB: delay introduced to compensate for the **trigger latency**
  - Signal split in trigger and data paths



Basic DAQ: “real” trigger

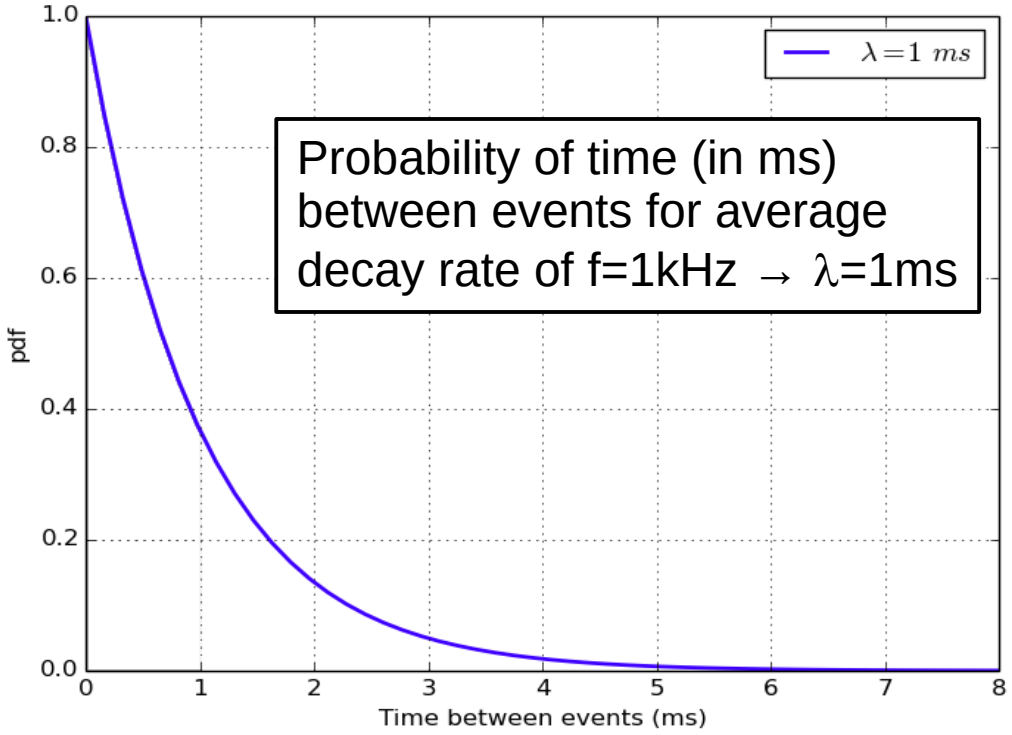
- Stochastic process
  - Fluctuations in time between events
- Let's assume for example
  - physics rate f = 1 kHz, i.e. λ = 1 ms
  - and, as before, τ = 1 ms

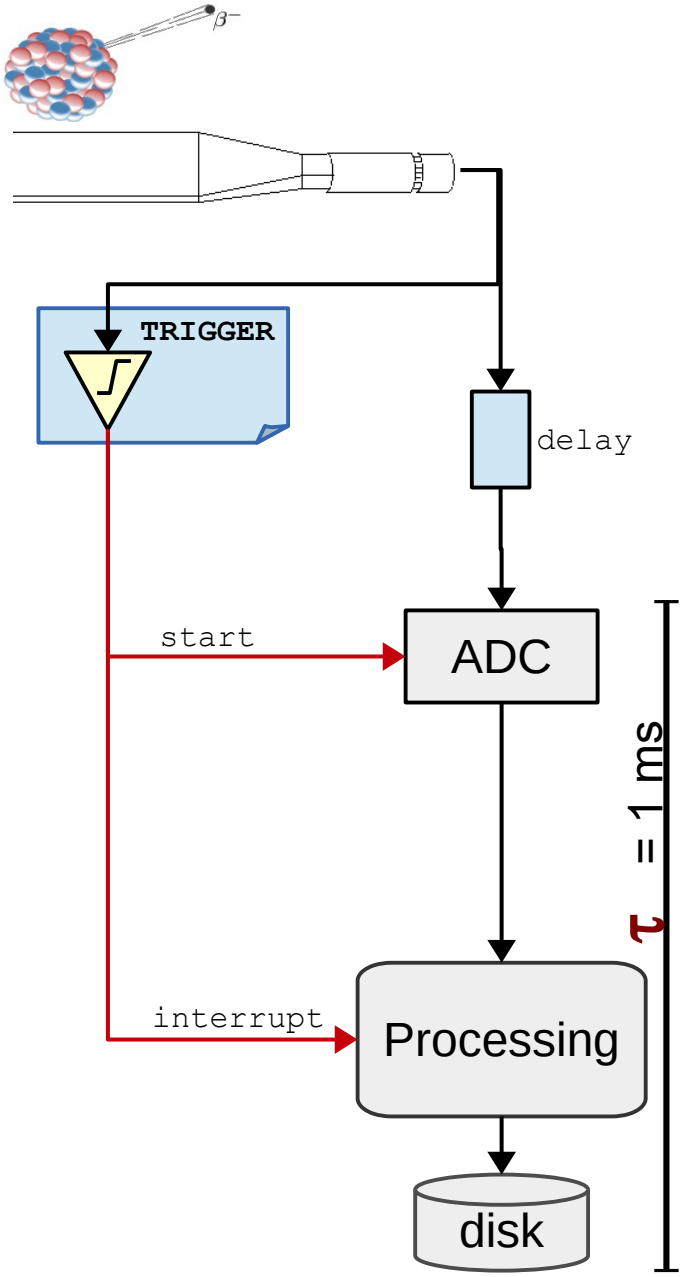




Basic DAQ: “real” trigger

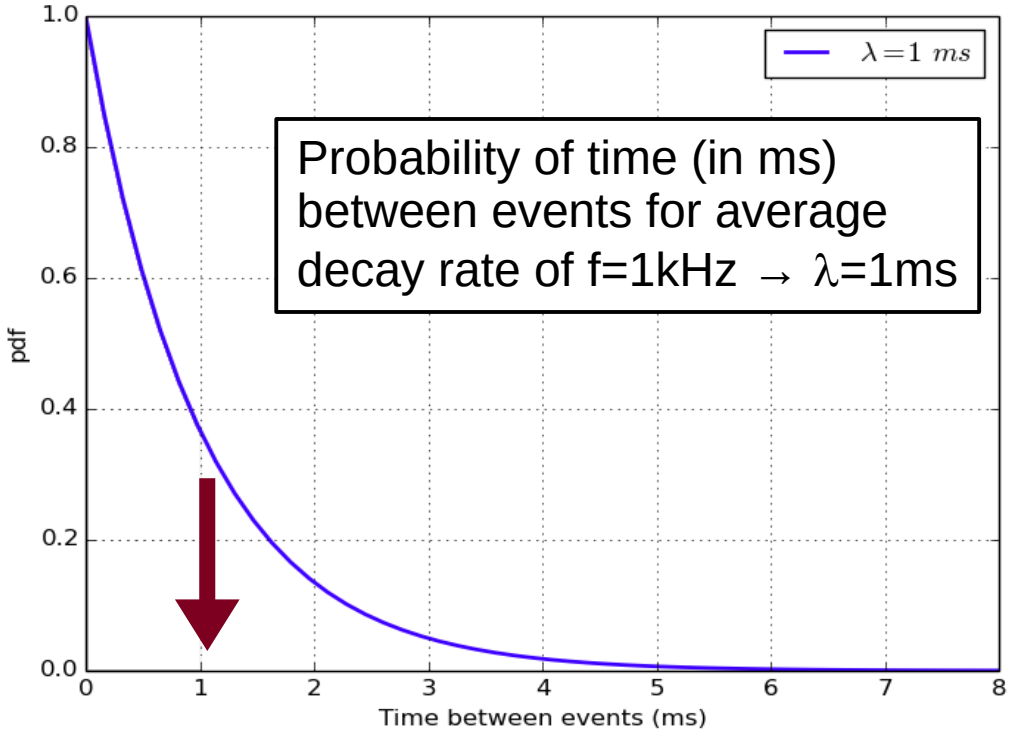
- Stochastic process
  - Fluctuations in time between events
- Let's assume for example
  - physics rate f = 1 kHz, i.e. λ = 1 ms
  - and, as before, τ = 1 ms

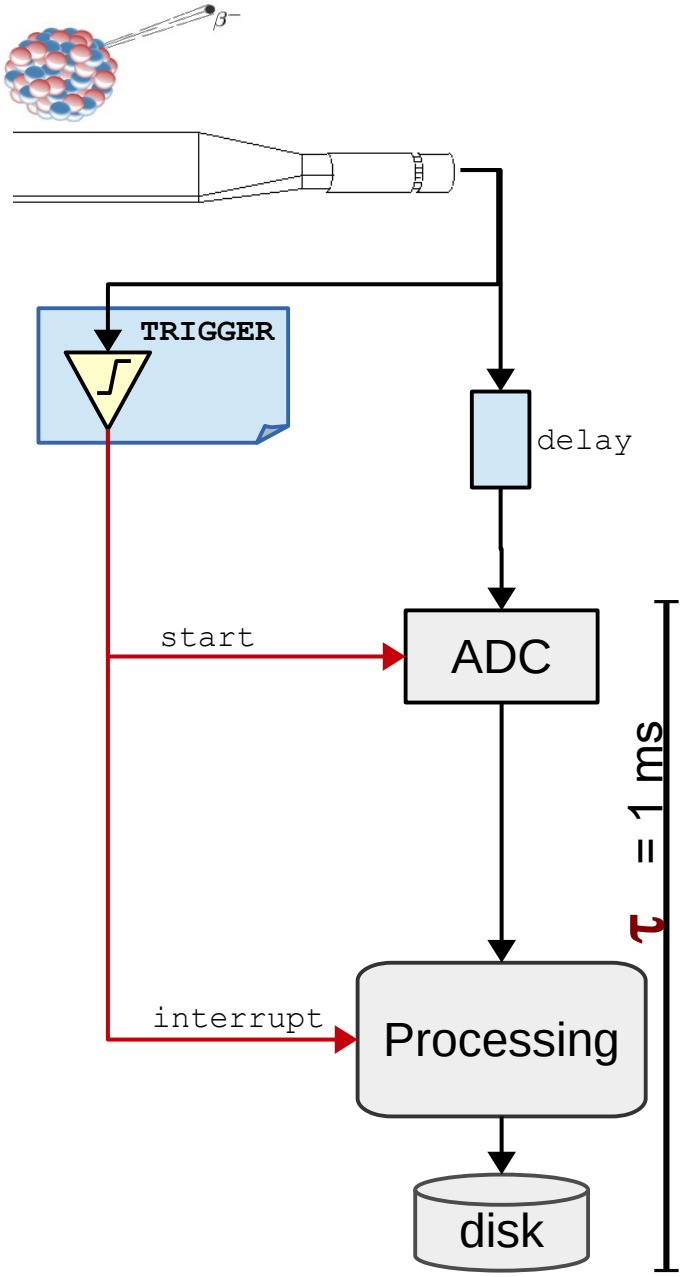




Basic DAQ: “real” trigger

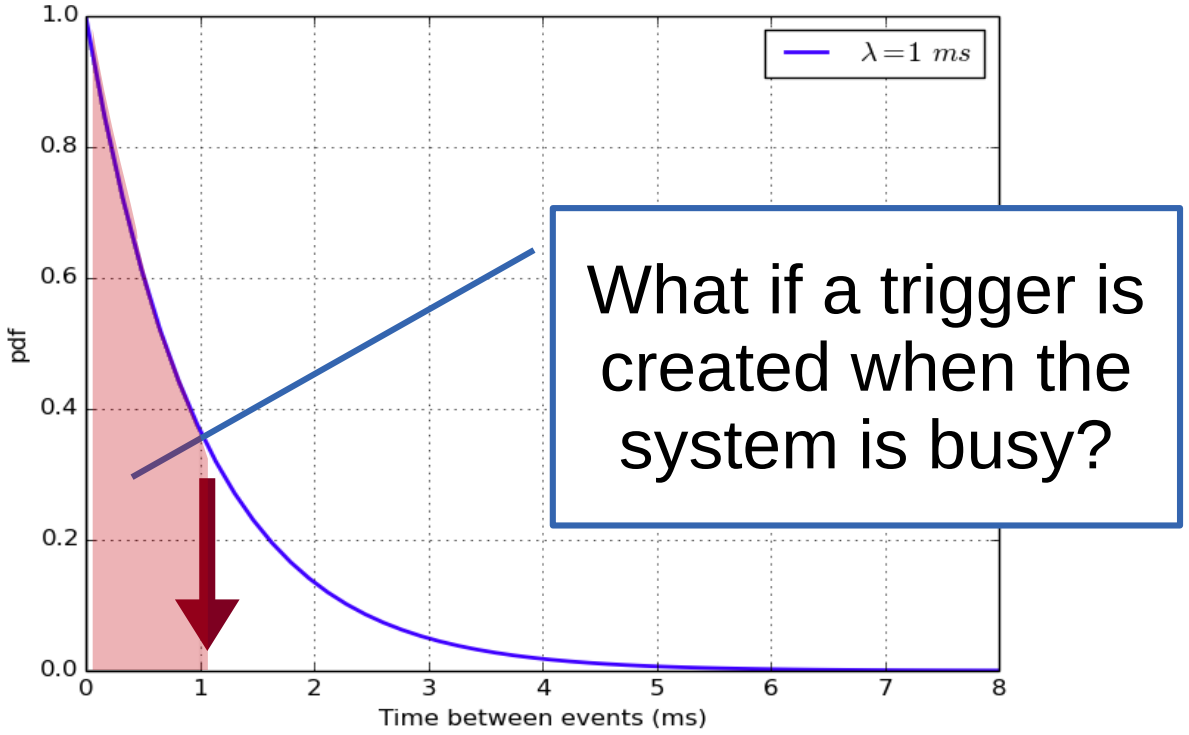
- Stochastic process
  - Fluctuations in time between events
- Let's assume for example
  - physics rate f = 1 kHz, i.e. λ = 1 ms
  - and, as before, τ = 1 ms



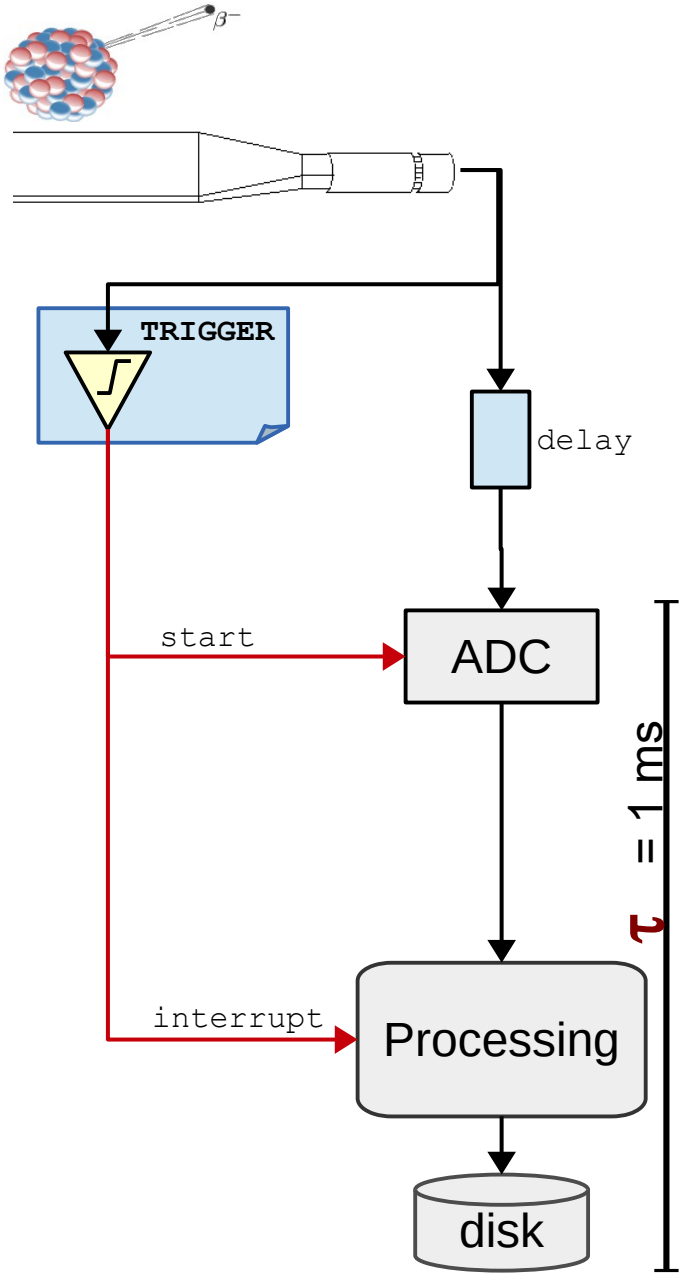


Basic DAQ: “real” trigger

- 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

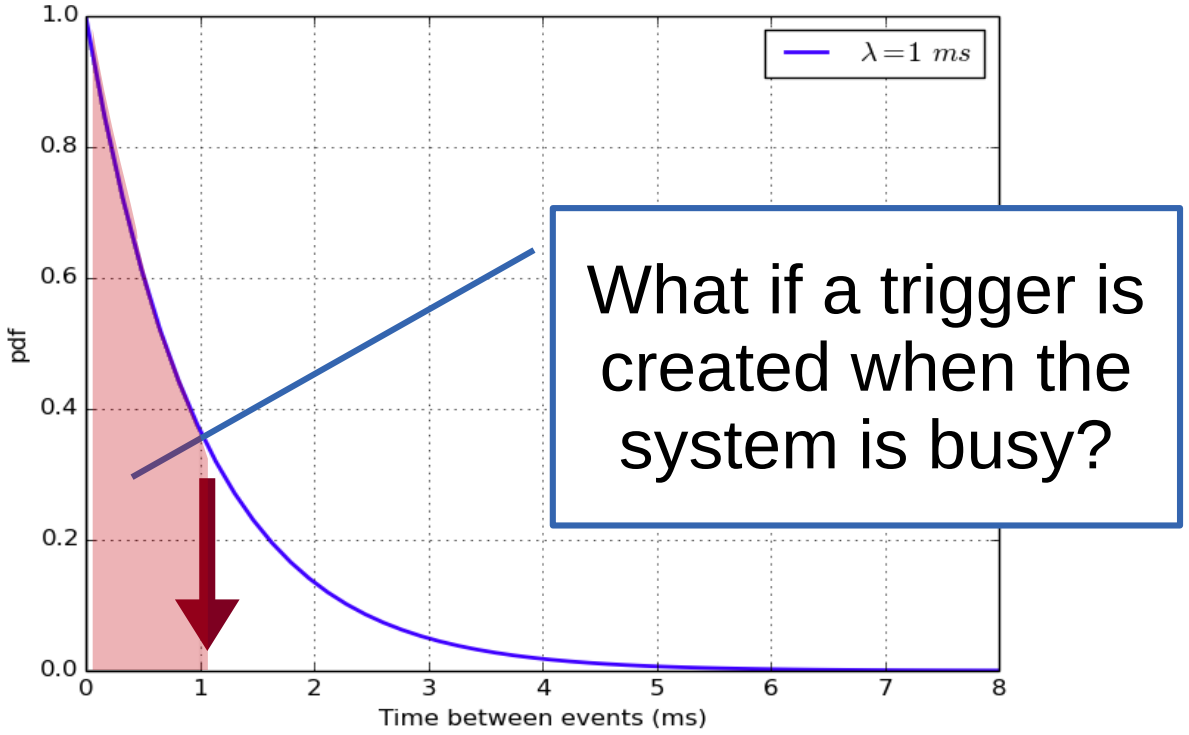


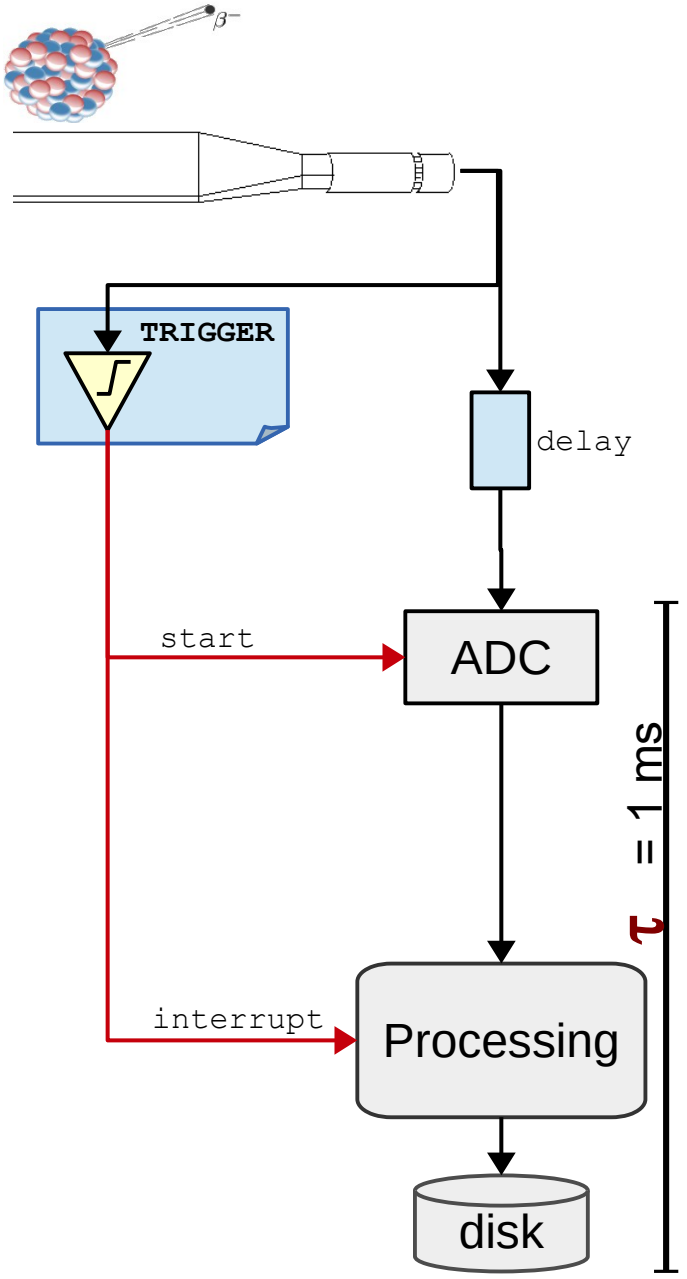
What if a trigger is created when the system is busy?



System 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



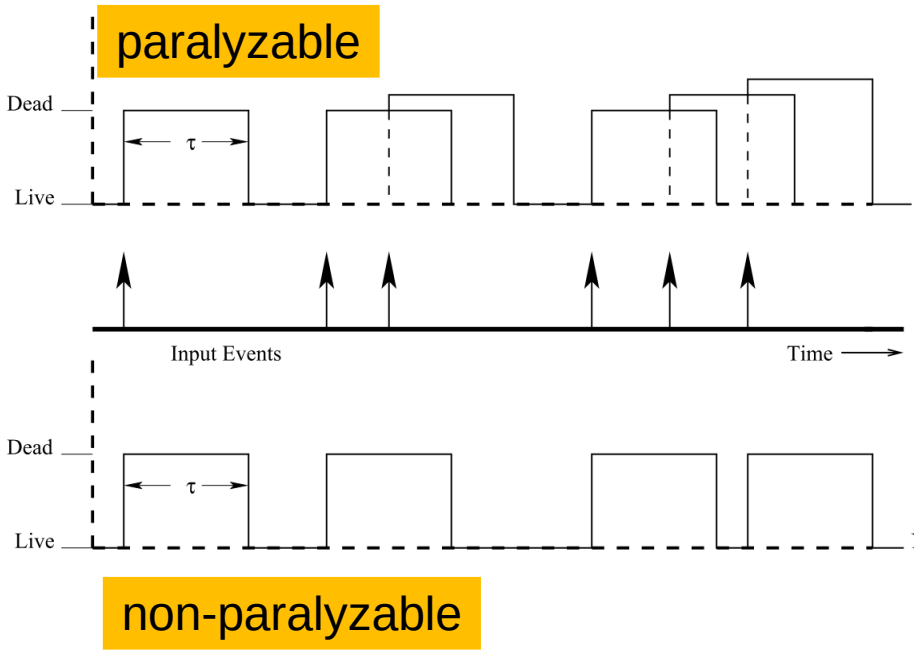


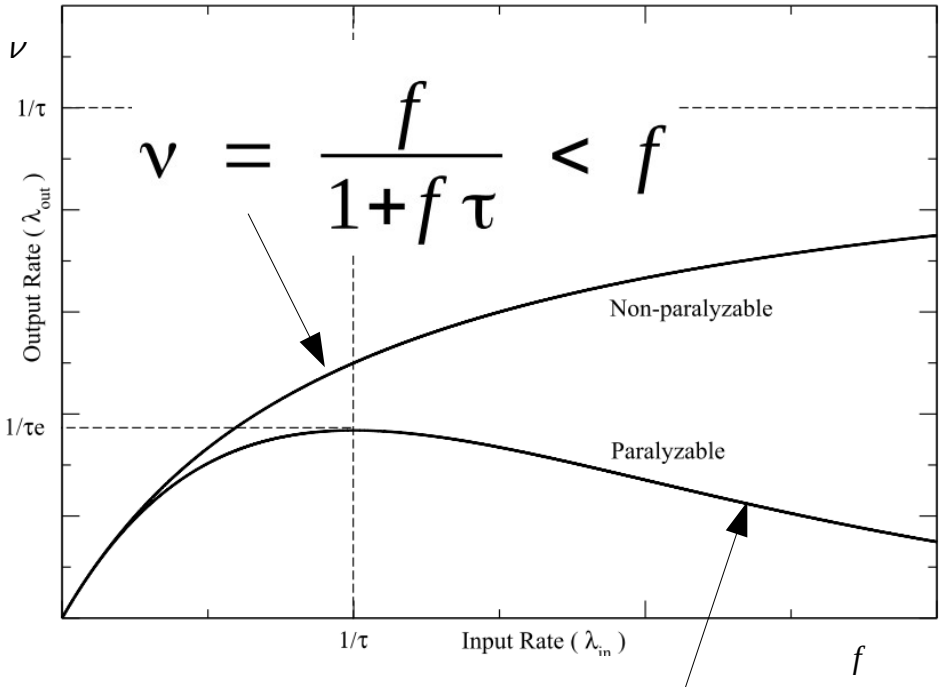
Pause to regroup

- 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**

Two detector models

- Paralyzable vs non paralyzable detectors





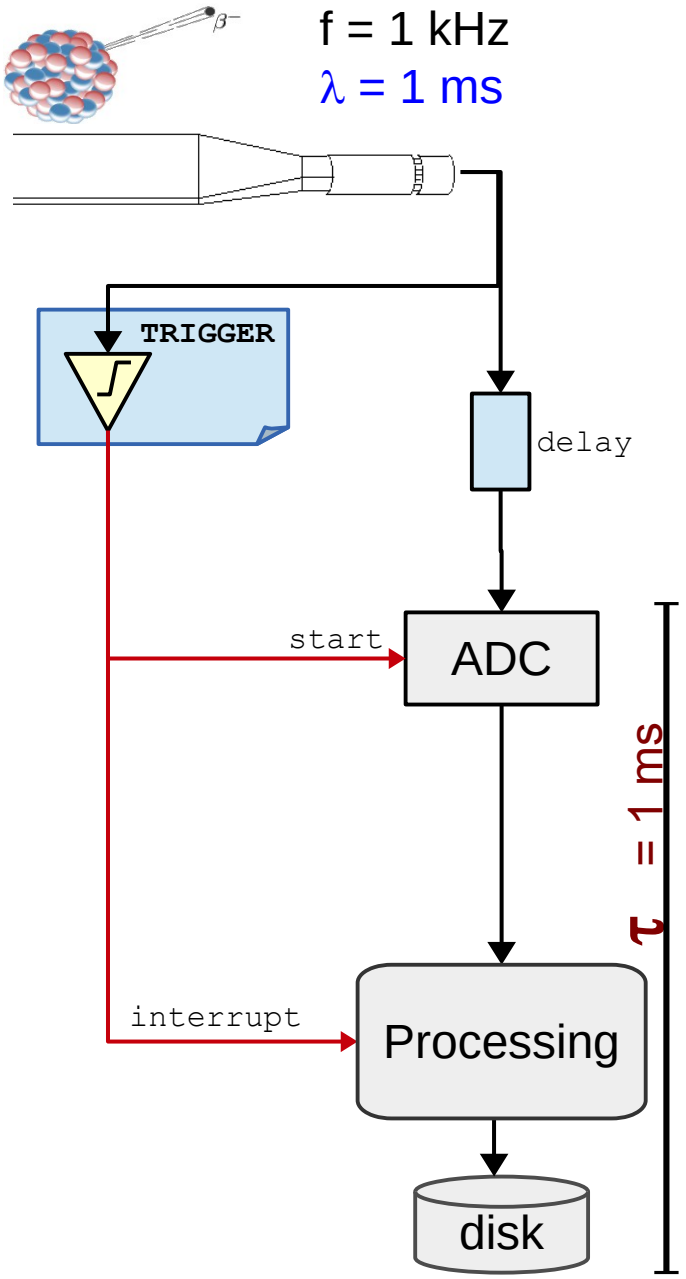
$$\nu = \frac{f}{1+f\,\tau} < f$$

$$\nu = f \exp(-f\,\tau)$$

All detectors we will be dealing with will be of the type non-paralyzable

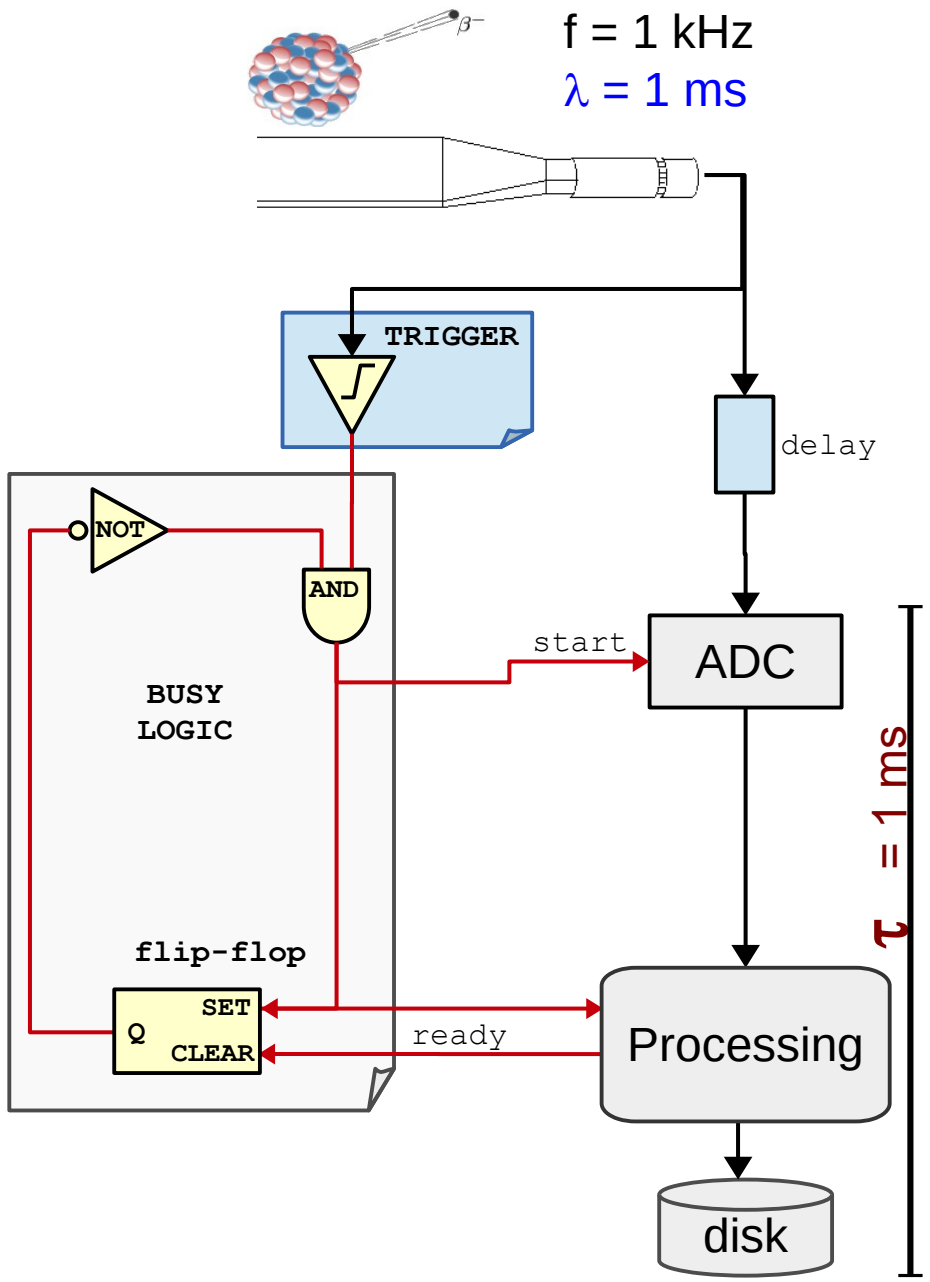
Busy logic

- 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**



Busy logic : nomenclature

Same concepts:

- BUSY: refers to the system (or a subsystem) that is processing a (trigger) signal and cannot accept other signals
- INHIBIT: refers to the trigger (or the DAQ) that cannot be raised since the system is not free
- VETO: refers to a condition that prevents the generation of another trigger

Contrary:

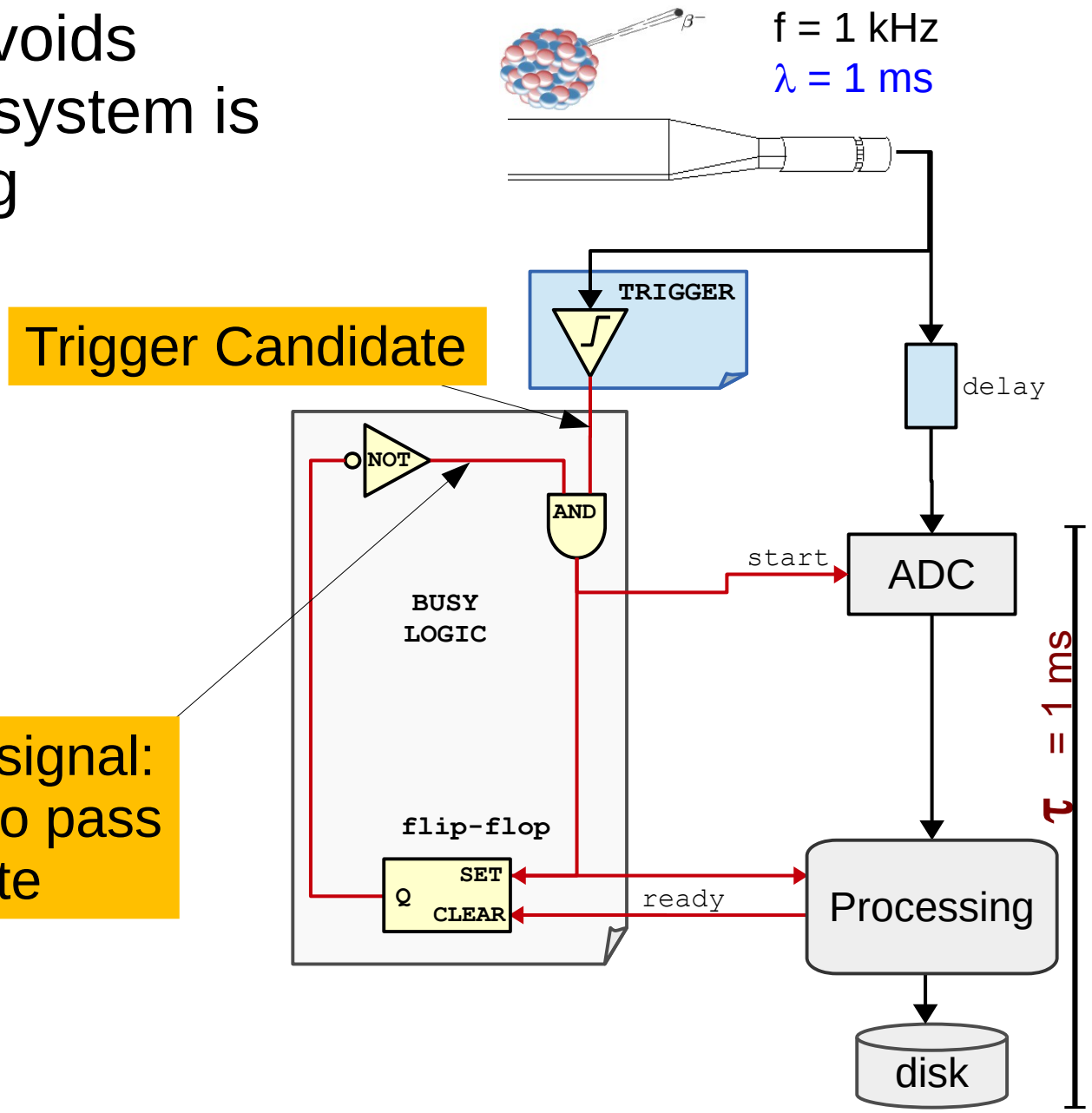
- ENABLE: refers to a condition that allows the generation of a trigger

For all practical purposes:

- BUSY = INHIBIT = VETO = .not. ENABLE

Busy logic

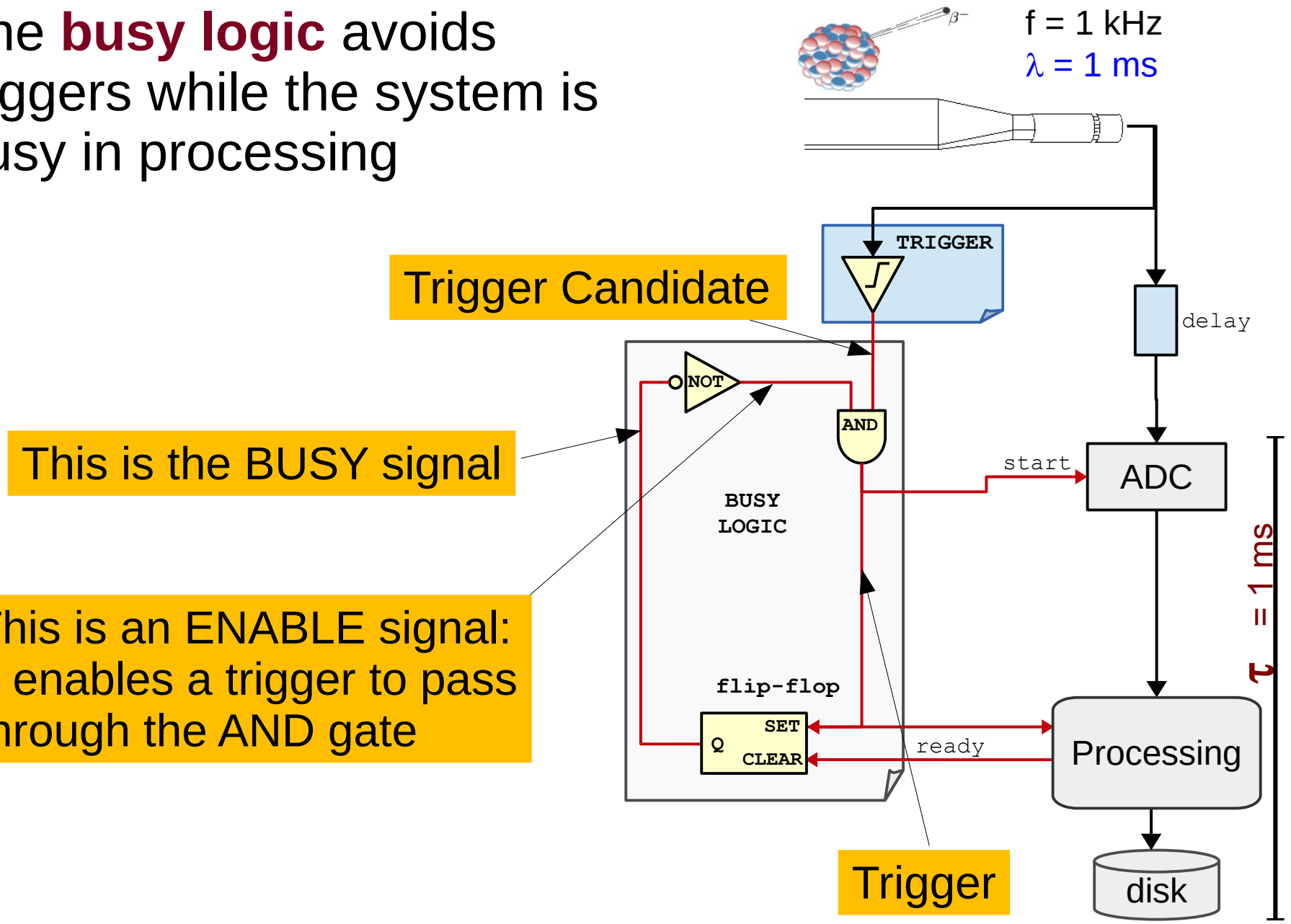
- The **busy logic** avoids triggers while the system is busy in processing



This is an ENABLE signal: It enables a trigger to pass through the AND gate

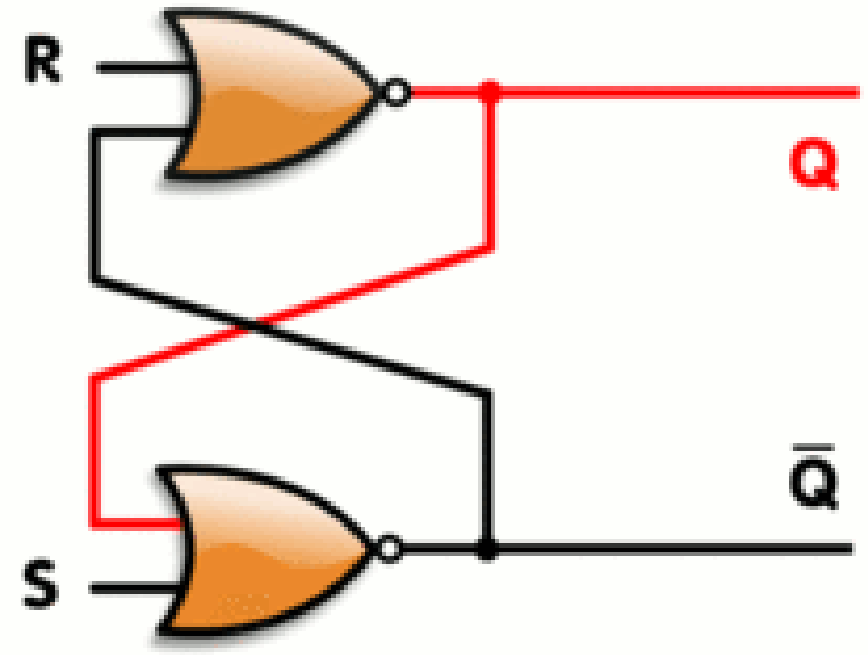
Busy logic

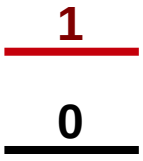
- The **busy logic** avoids triggers while the system is busy in processing



Flip Flop 1/5

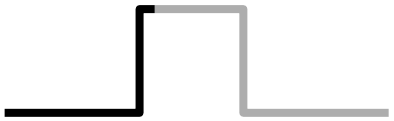
- Flip-flop: two NOR gates: .NOT. (a.OR.b)
  - a **bistable** circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)
- Before: stable state, Q up and $\overline{Q}$ down

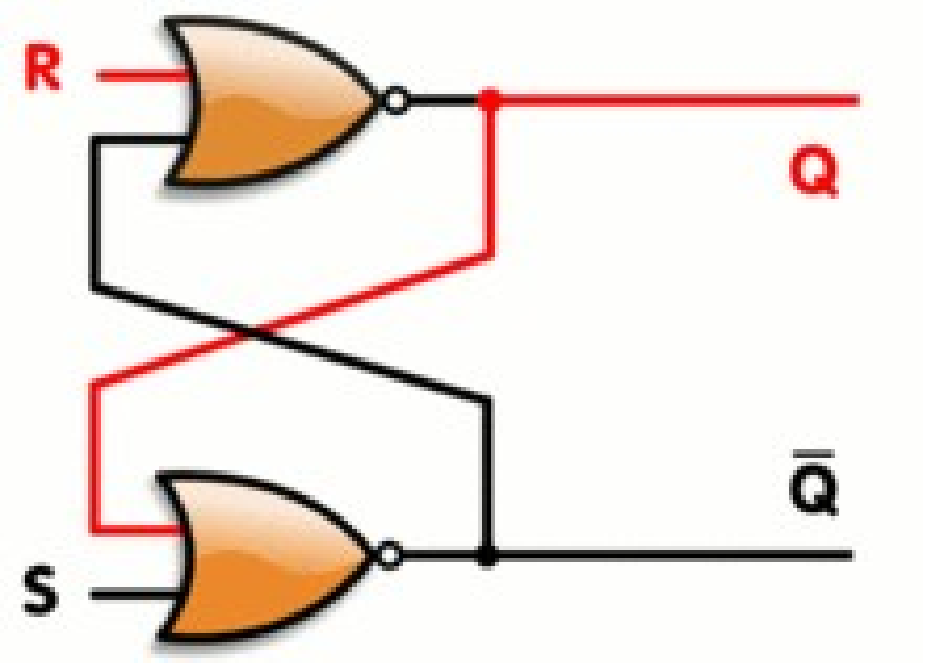


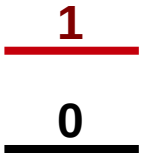


Flip Flop 2/5

- Flip-flop: two NOR gates: .NOT. (a.OR.b)
  - a **bistable** circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)
- At some point, signal injected in R

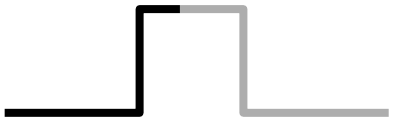


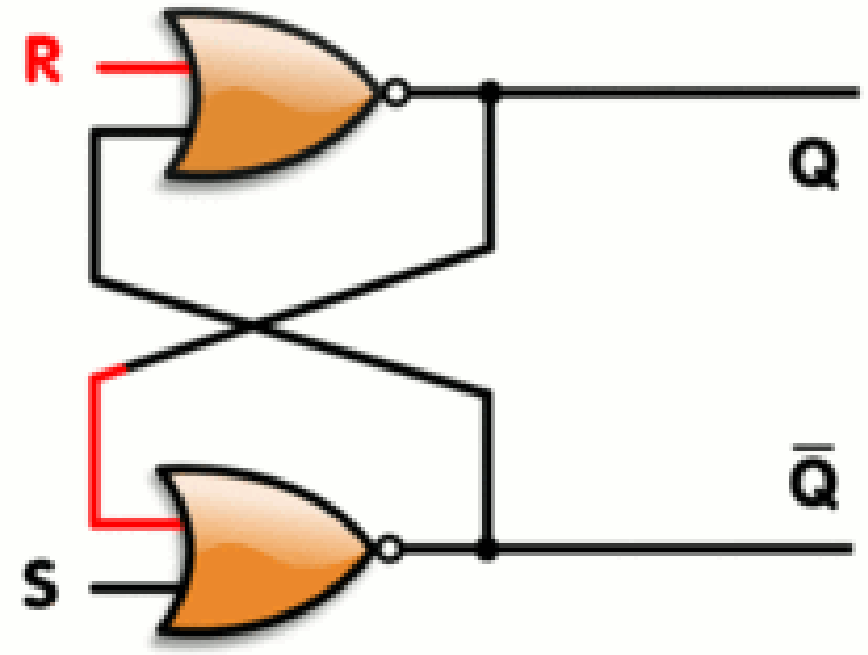


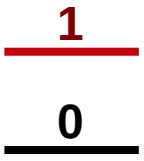


Flip Flop 3/5

- Flip-flop
  - a **bistable** circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)
- At some point, signal injected in R
  - Q switched down and the feedback travels to S

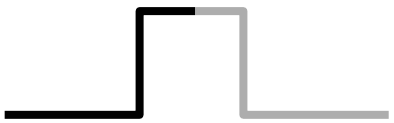


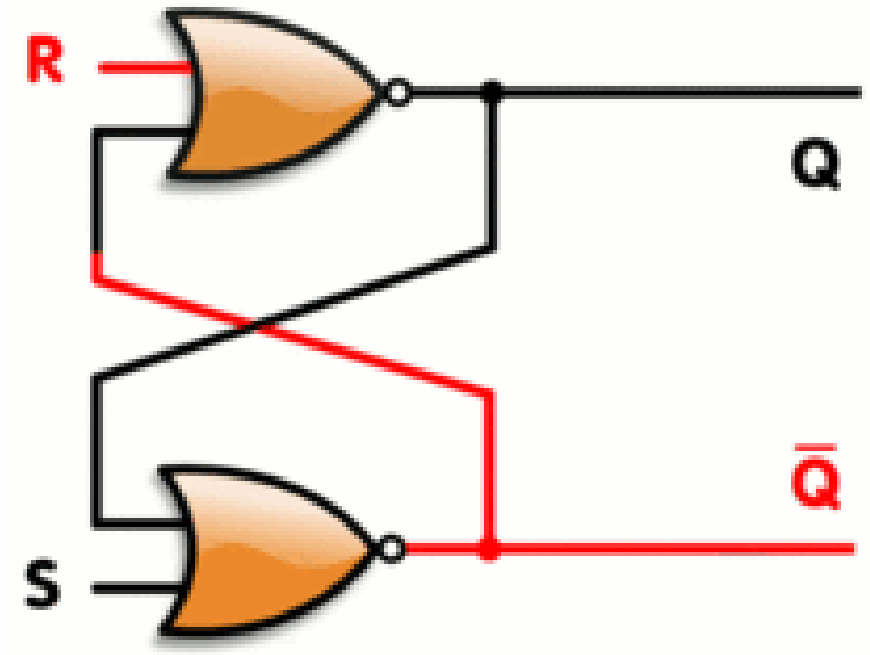


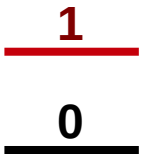


Flip Flop 4/5

- Flip-flop
  - a **bistable** circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)
- At some point, signal injected in R
  - $\overline{Q}$ becomes up and the feedback travels to R

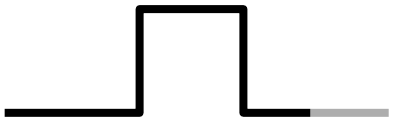


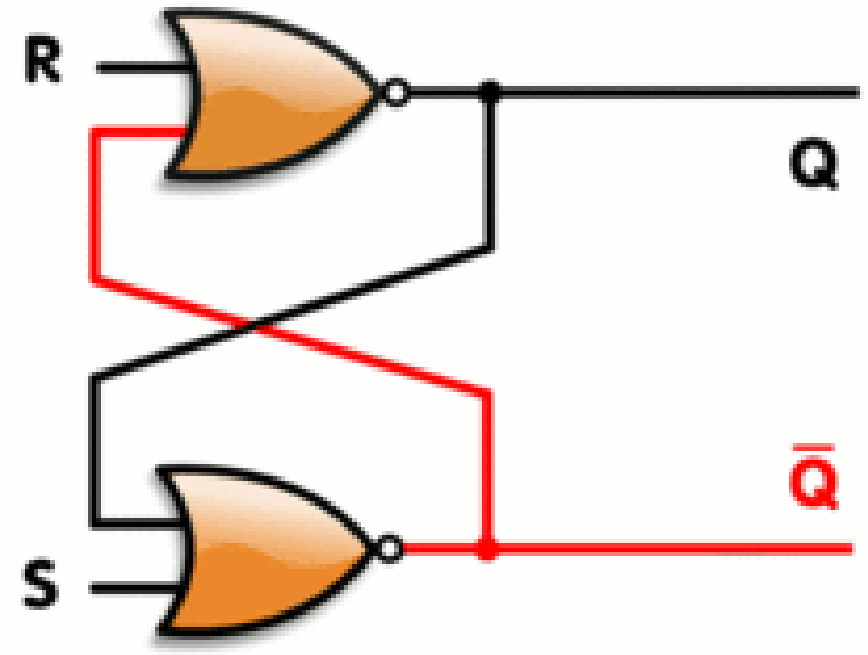


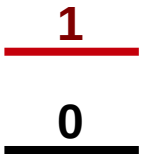


Flip Flop 5/5

- Flip-flop
  - a **bistable** circuit that changes state (Q) by signals applied to the control inputs (SET, CLEAR)
- After: stable state, Q down and $\overline{Q}$ up:
  - End of pulse





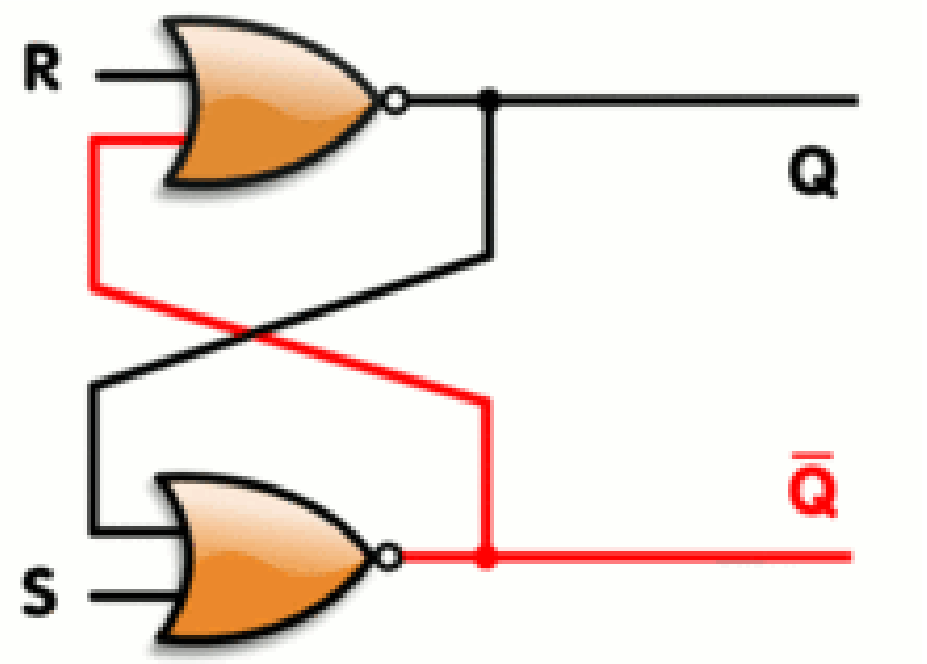


Flip Flop Truth table

- Flip-flop

  a **bistable** circuit that changes state (Q) by signals applied to the control inputs (S=SET, R=CLEAR, RESET)

Q out	S=0	S=1
R=0	Previous Q	1
R=1	0	undefined

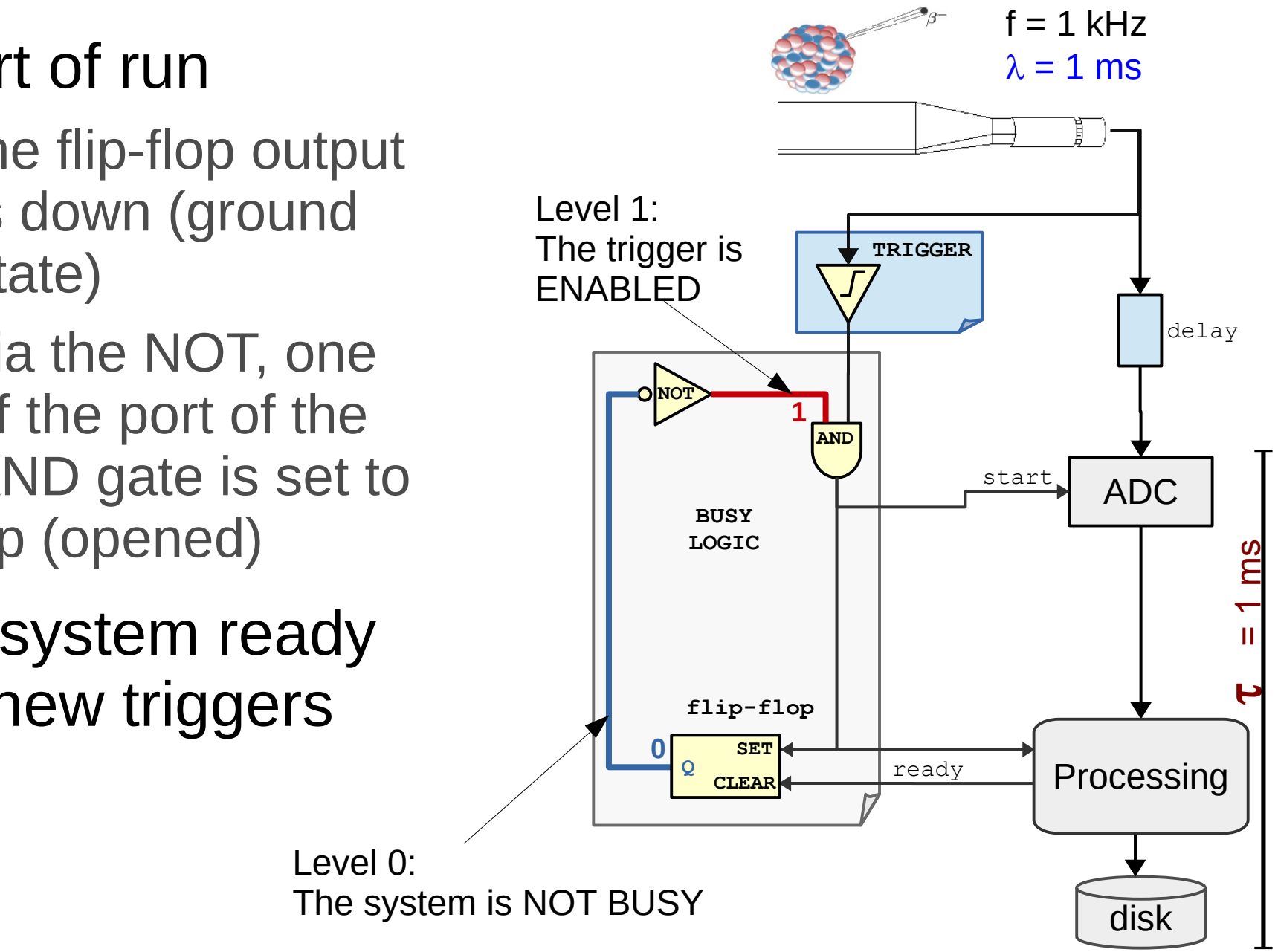


1

0

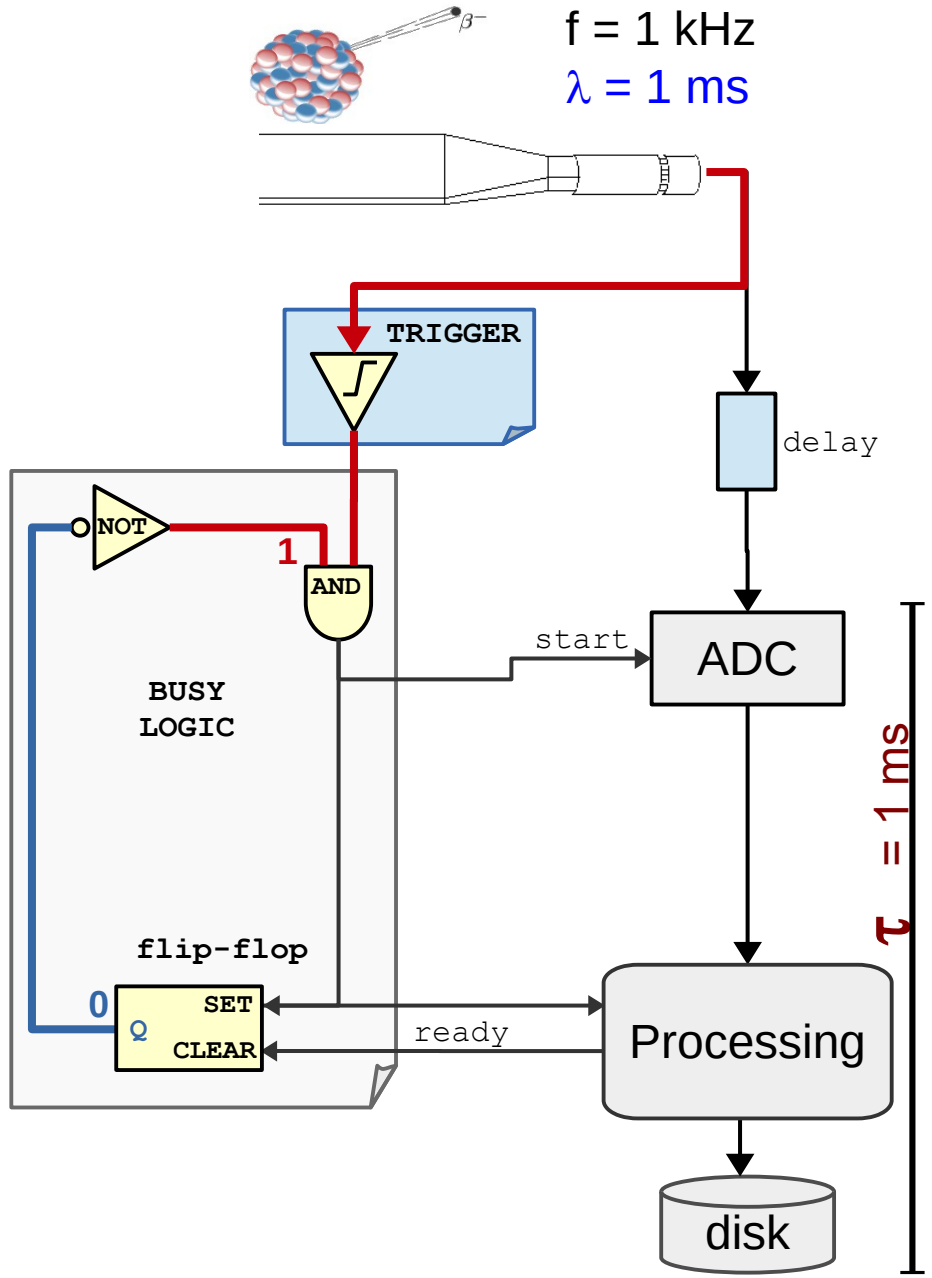
Busy logic

- 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



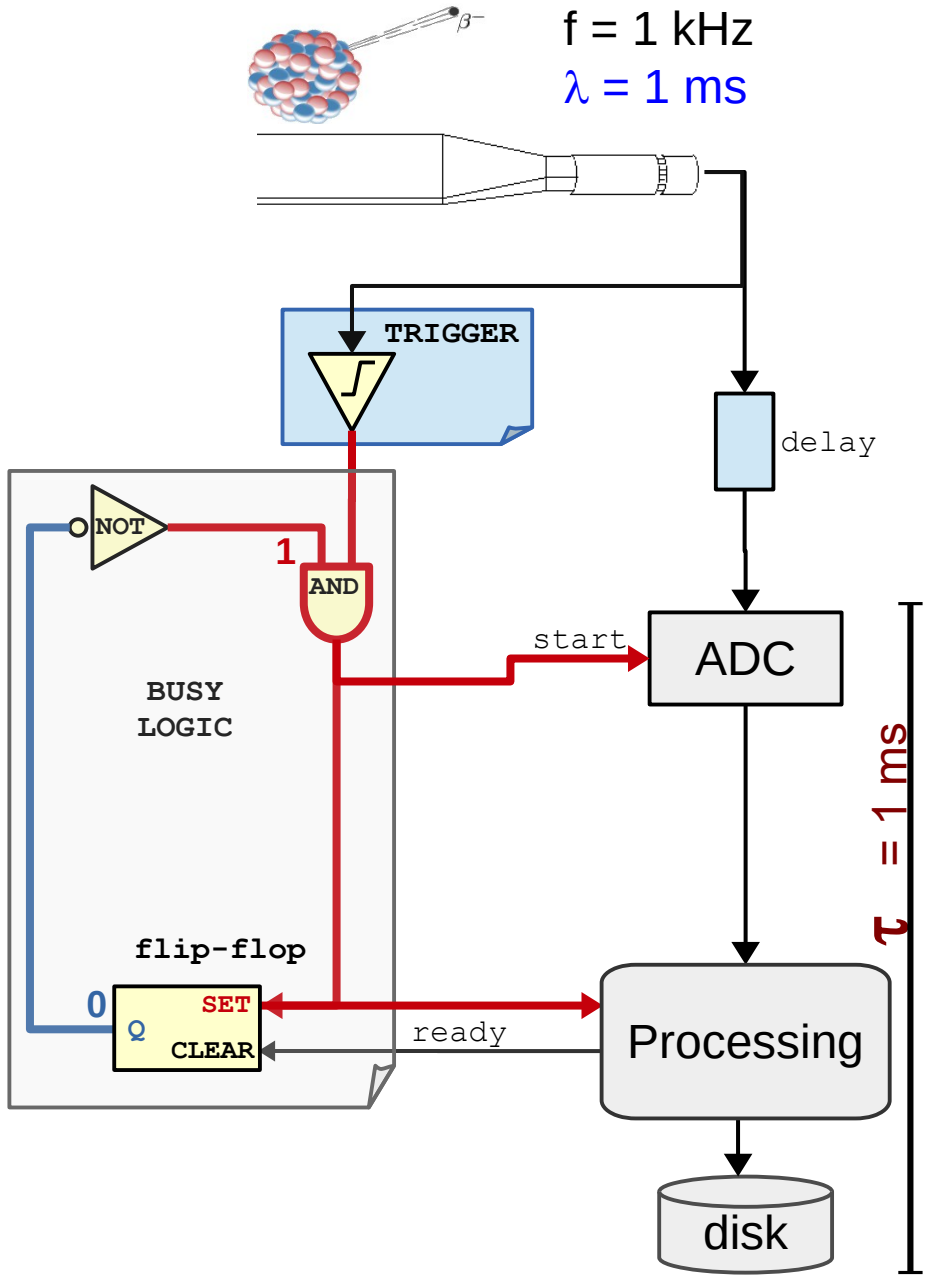
Busy logic

- 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)



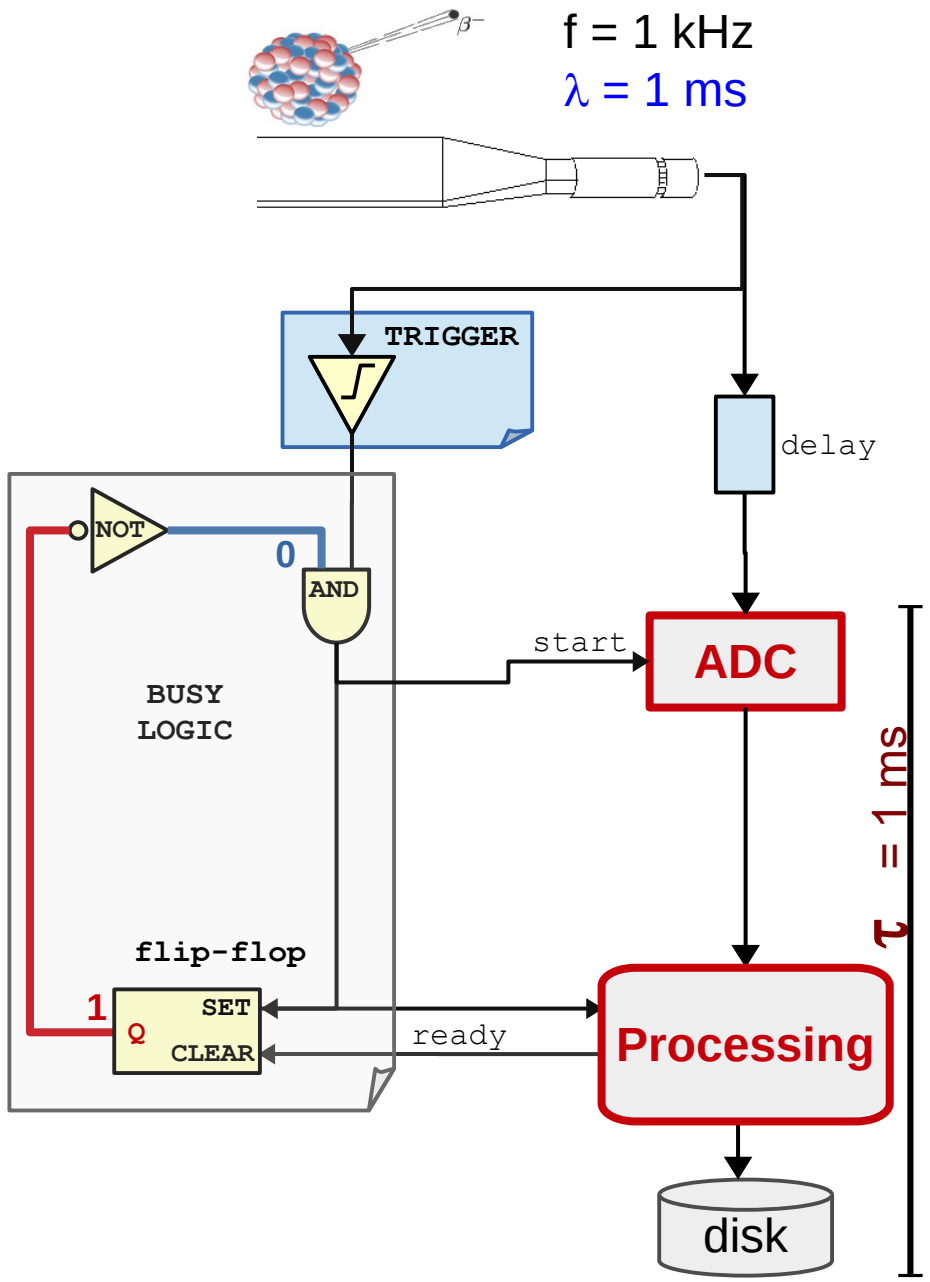
Busy logic

- 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)



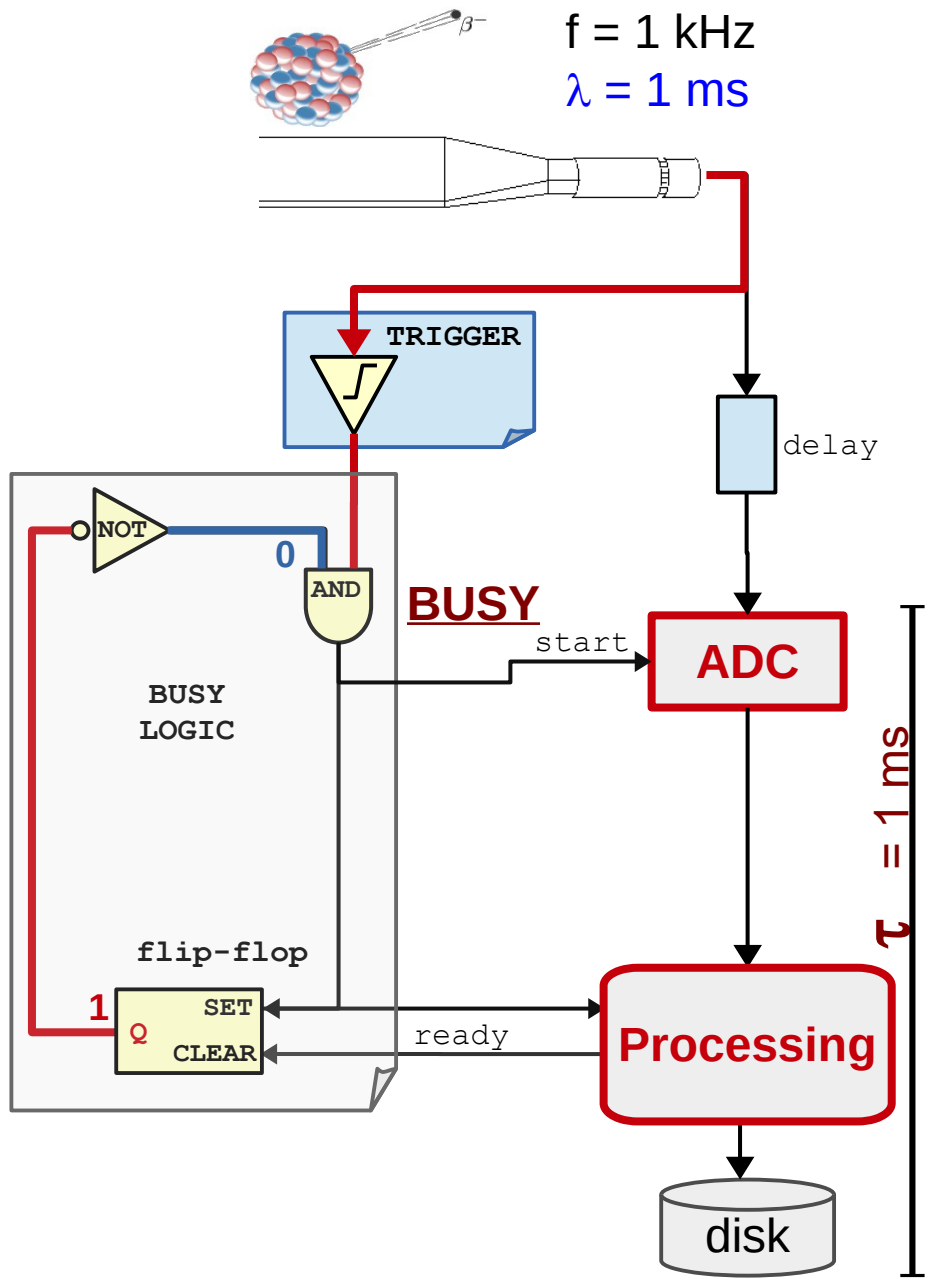
Busy logic

- 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)



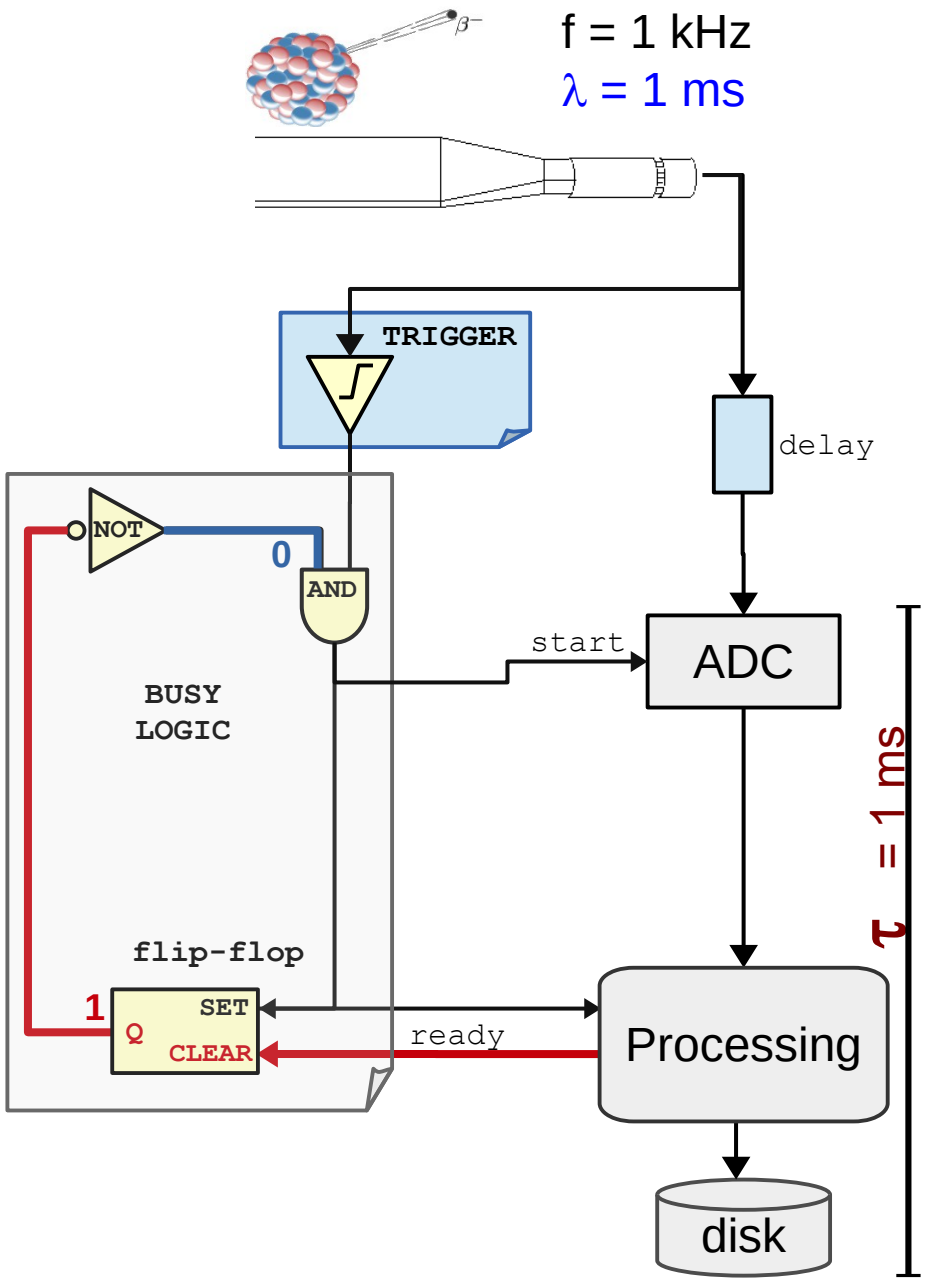
Busy logic

- 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)



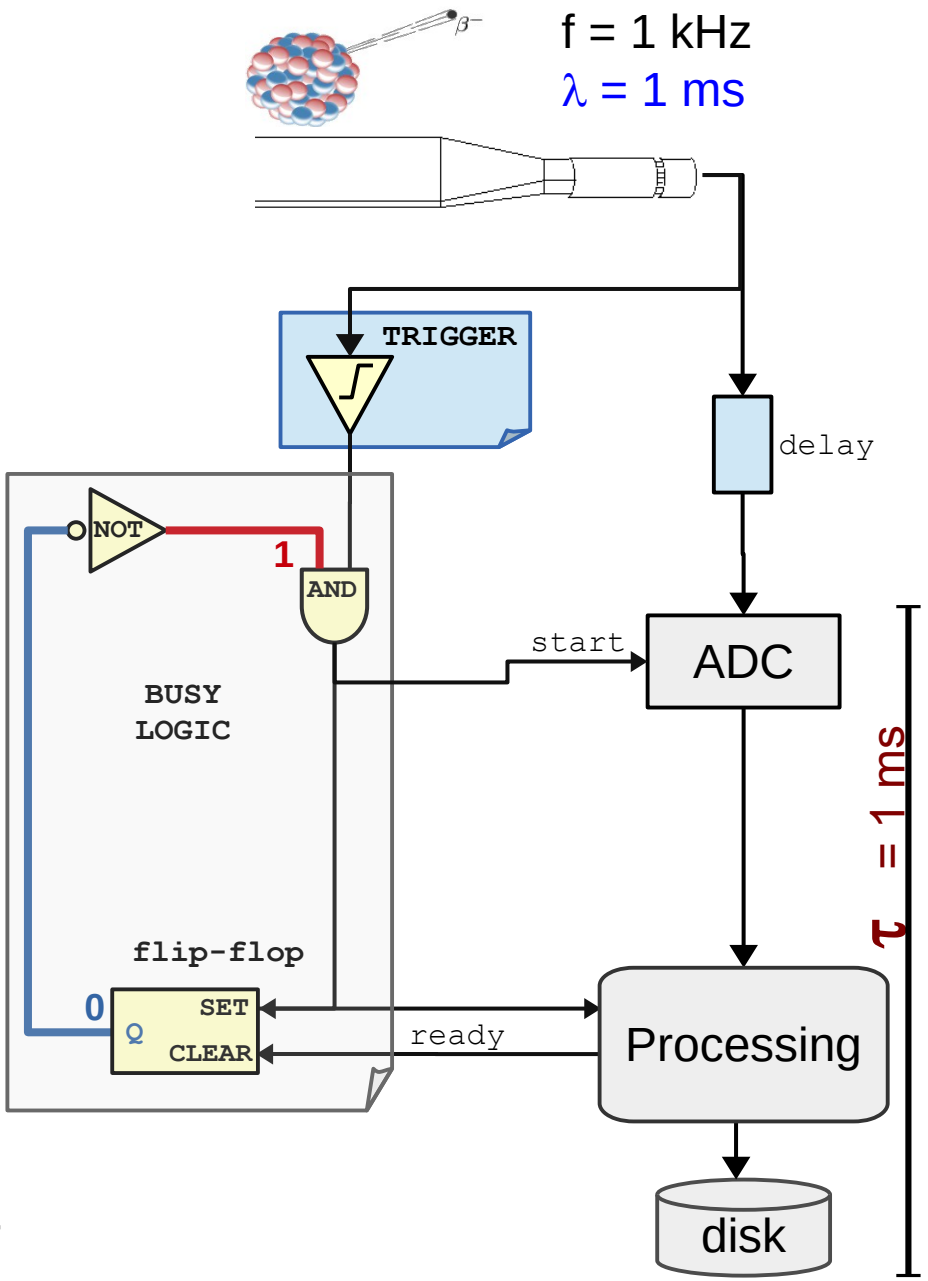
Busy logic

- 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



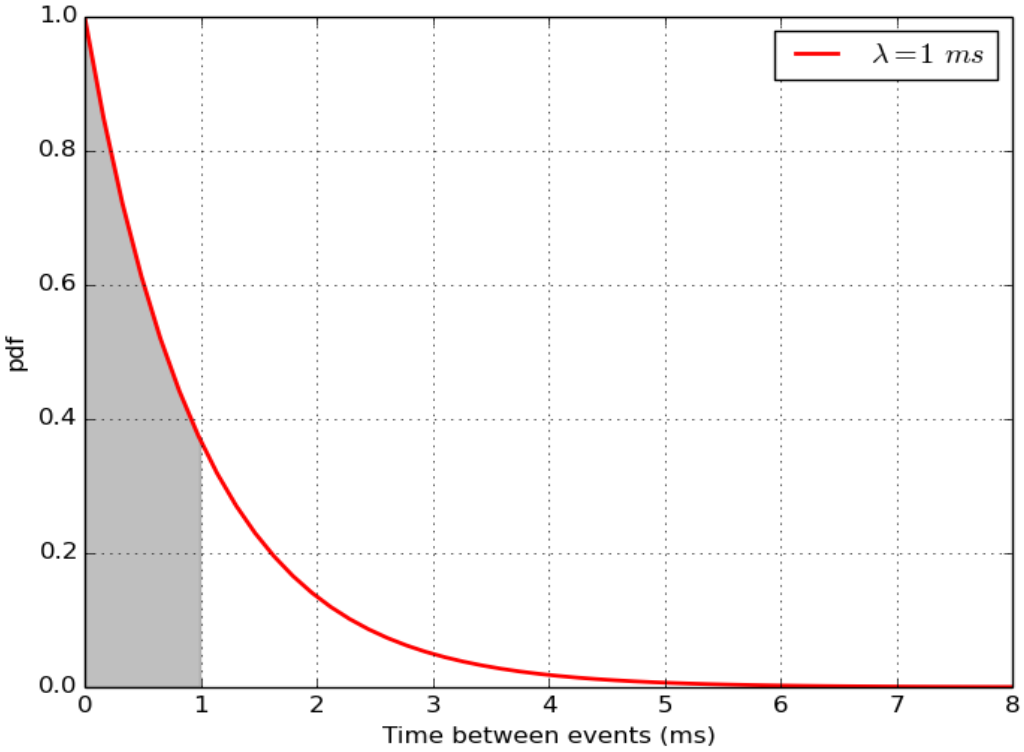
Busy logic

- 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

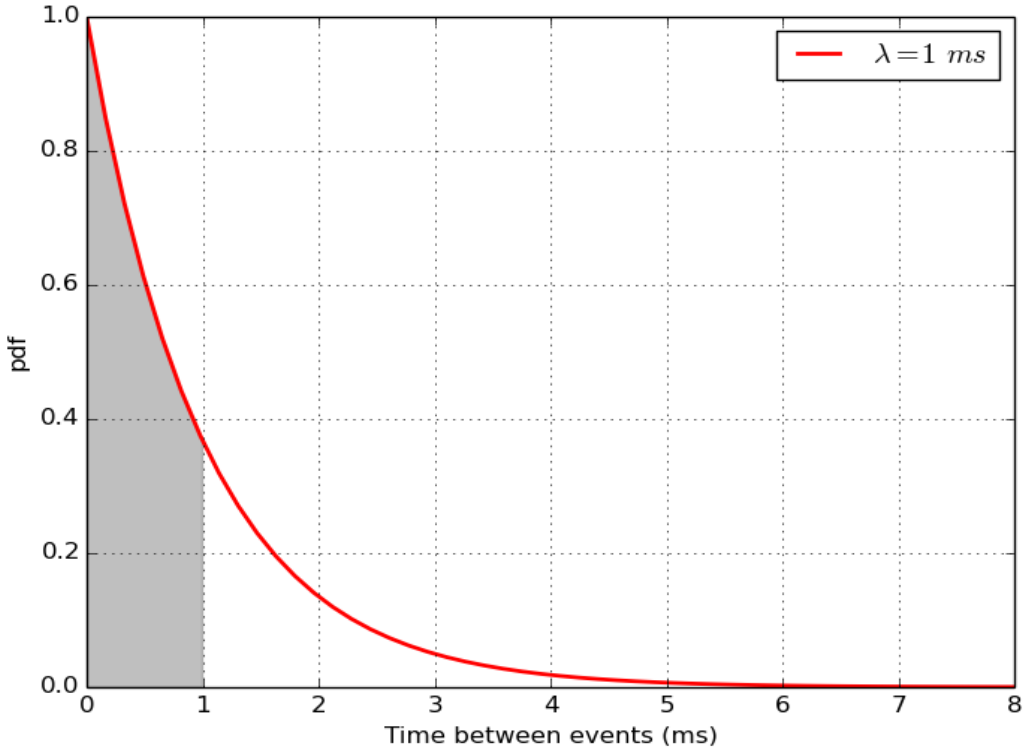
- So the busy mechanism protects our electronics from unwanted triggers
  - New signals are accepted only when the system in ready to process them



- Which (average) DAQ rate can we achieve now?
  - How much we lose with the busy logic?
  - Reminder: with a clock trigger and τ = 1 ms the limit was 1 kHz

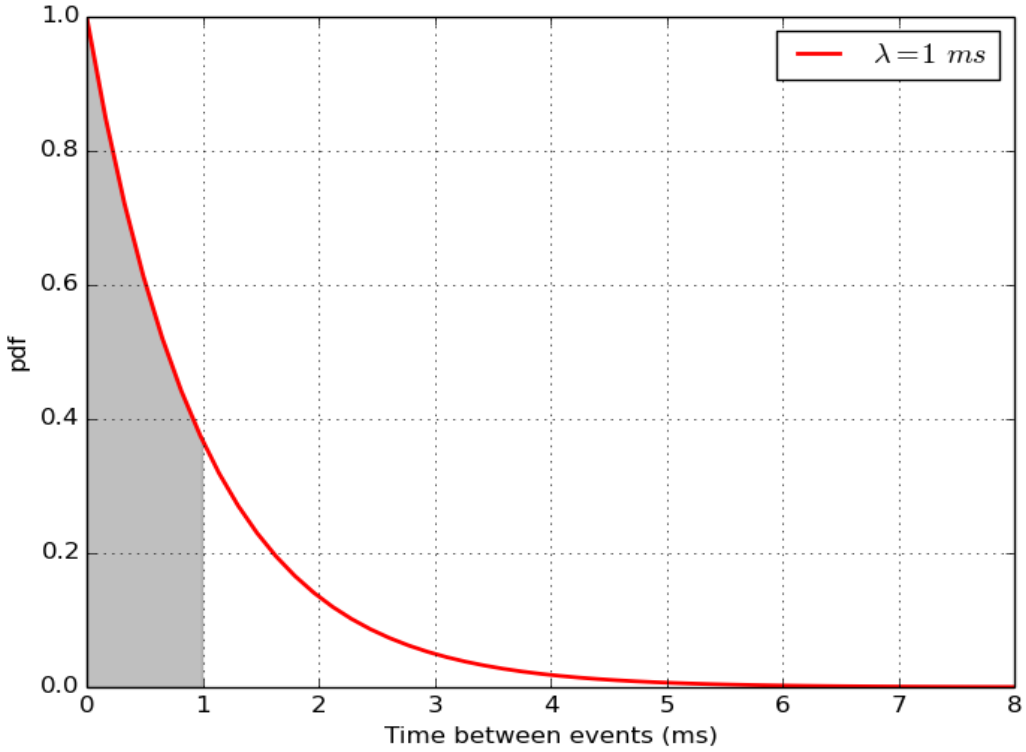
Deadtime and efficiency

- 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] = ν τ; P[free] = 1 - ν τ
- Therefore:



Deadtime and efficiency

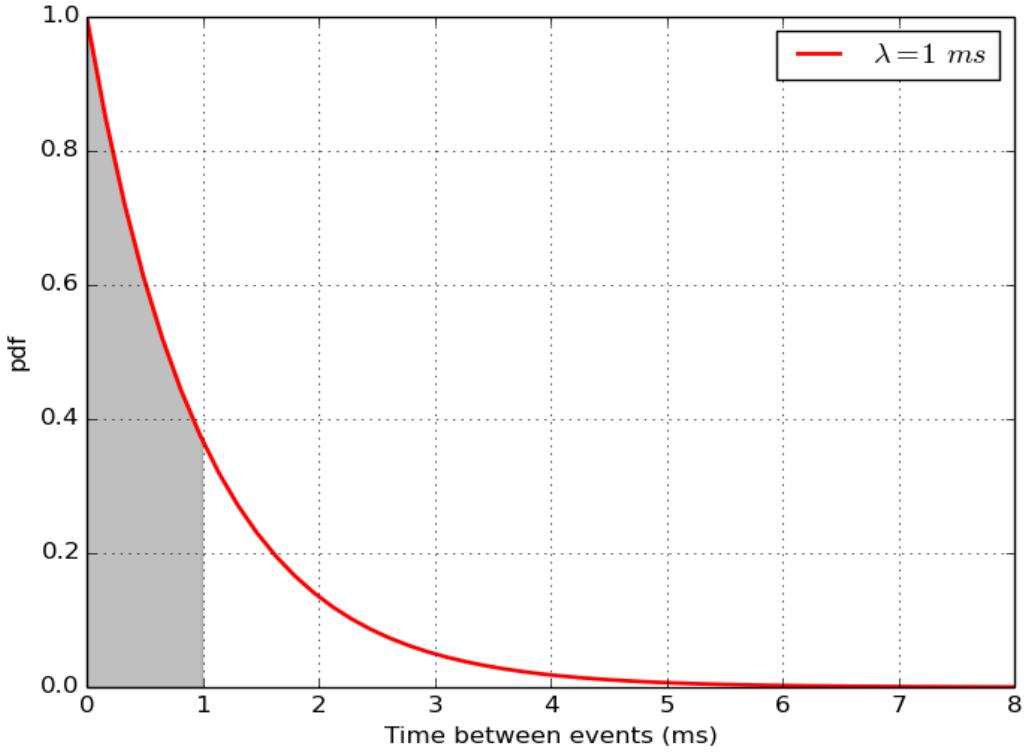
- Definitions
  - **f**: average rate of physics (input)
  - **ν**: average rate of DAQ (output)
  - **τ: deadtime**, needed to process an event, without being able to handle other triggers
  - probabilities: P[busy] = ν τ; P[free] = 1 - ν τ



- Therefore:

Deadtime and efficiency

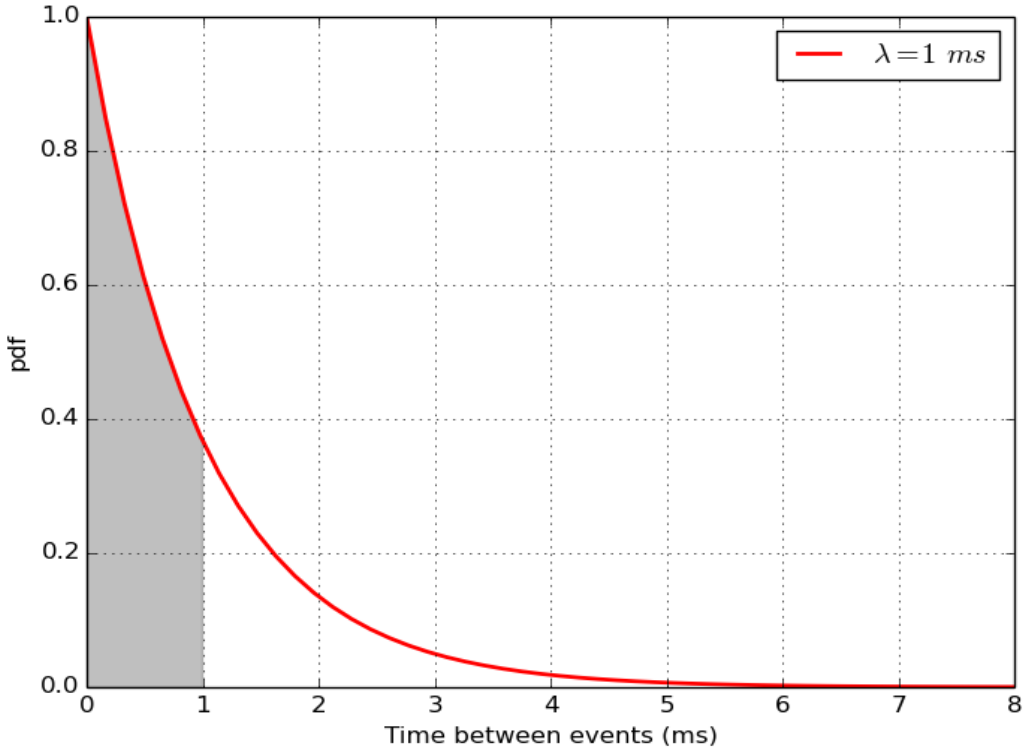
- Definitions
  - **f**: average rate of physics (input)
  - **ν**: average rate of DAQ (output)
  - **τ: deadtime**, needed to process an event, without being able to handle other triggers
  - probabilities: P[busy] = ν τ; P[free] = 1 - ν τ



- Therefore:

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] = ν τ; P[free] = 1 - ν τ



- Therefore:

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

Deadtime and efficiency

- Due to stochastic fluctuations
  - DAQ rate always < physics rate $\nu = \frac{f}{1+f\,\tau} < f$
  - Efficiency always < 100% $\epsilon = \frac{N_{saved}}{N_{tot}} = \frac{1}{1+f\,\tau} < 100\,\%$

- So, in our specific example
  - Physics rate 1 kHz
  - Deadtime 1 ms

$$\begin{matrix} f = 1\,kHz \\ \tau = 1\,ms \end{matrix} \rightarrow \begin{matrix} \nu = 500\,Hz \\ \epsilon = 50\,\% \end{matrix}$$

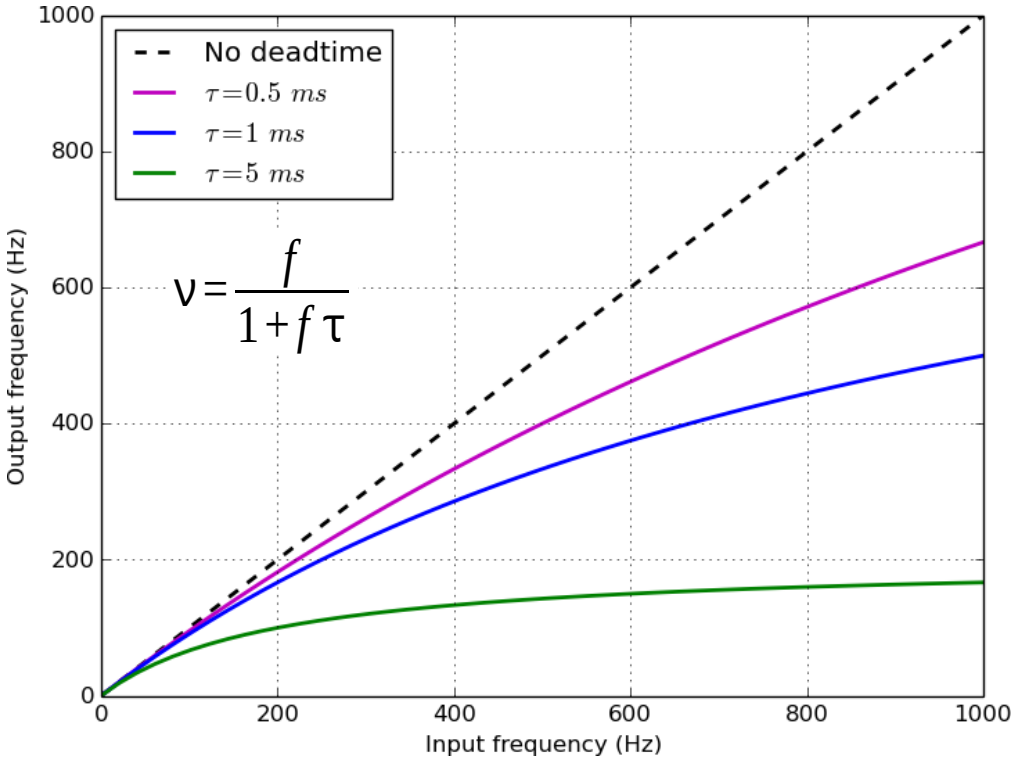
Deadtime and efficiency

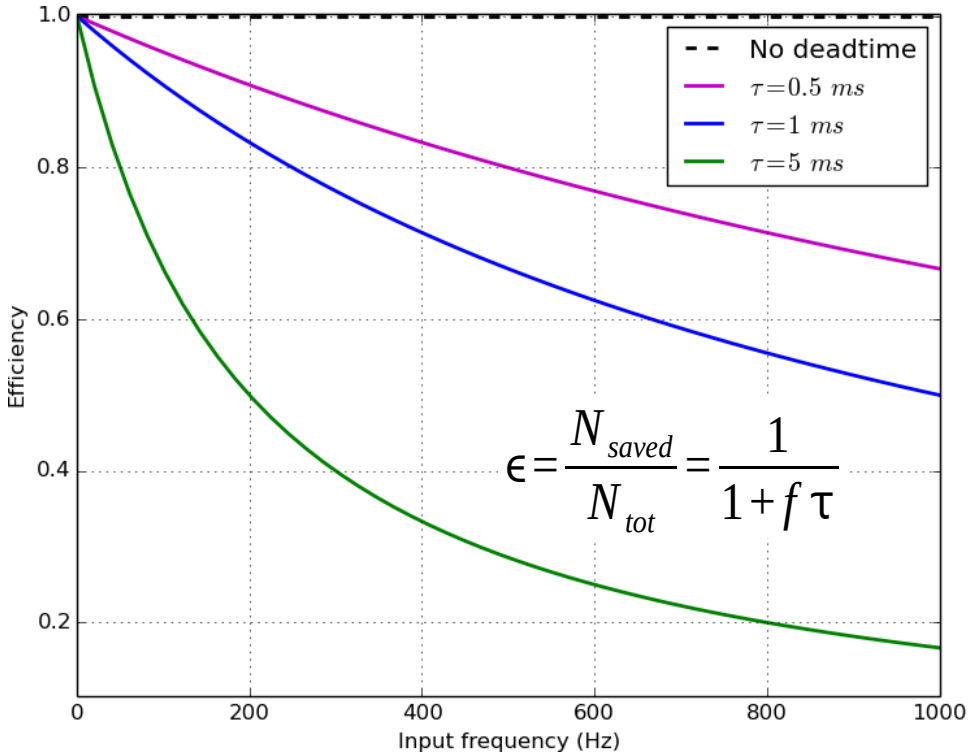
- Due to stochastic fluctuations
  - DAQ rate always < physics rate $\nu = \frac{f}{1+f\,\tau} < f$
  - Efficiency always < 100% $\epsilon = \frac{N_{saved}}{N_{tot}} = \frac{1}{1+f\,\tau} < 100\,\%$

- So, in our specific example
  - Physics rate 1 kHz
  - Deadtime 1 ms

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

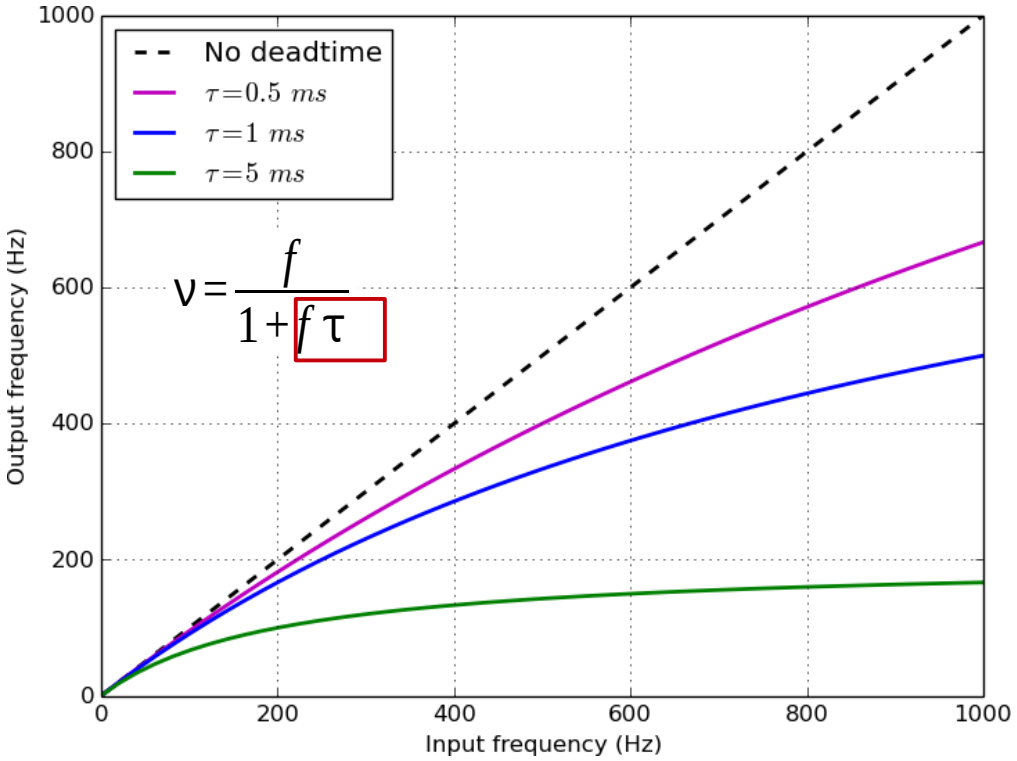
Deadtime and efficiency

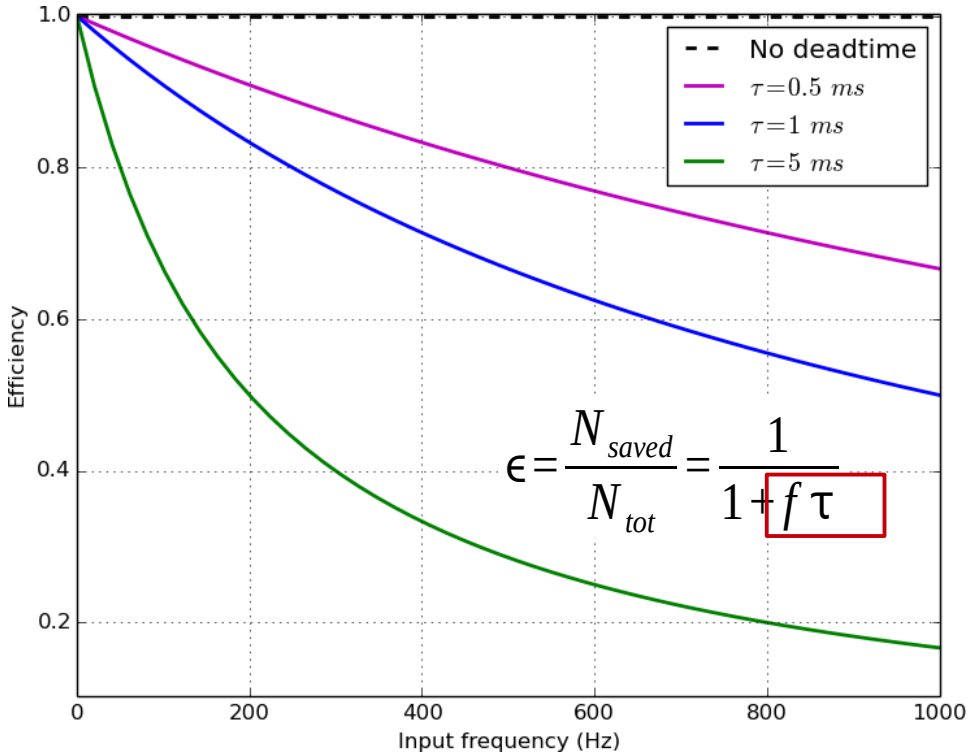




- In order to obtain $\varepsilon \sim 100\%$ (i.e.: $\nu \sim f$) $\rightarrow f\tau << 1 \rightarrow \tau << \lambda$
  - E.g.: $\varepsilon \sim 99\%$ for f = 1 kHz $\rightarrow$ $\tau < 0.01$ ms $\rightarrow 1/\tau > 100$ kHz
  - To cope with the input signal fluctuations, we have to **over-design** our DAQ system **by a factor 100!**
- How can we mitigate this effect?

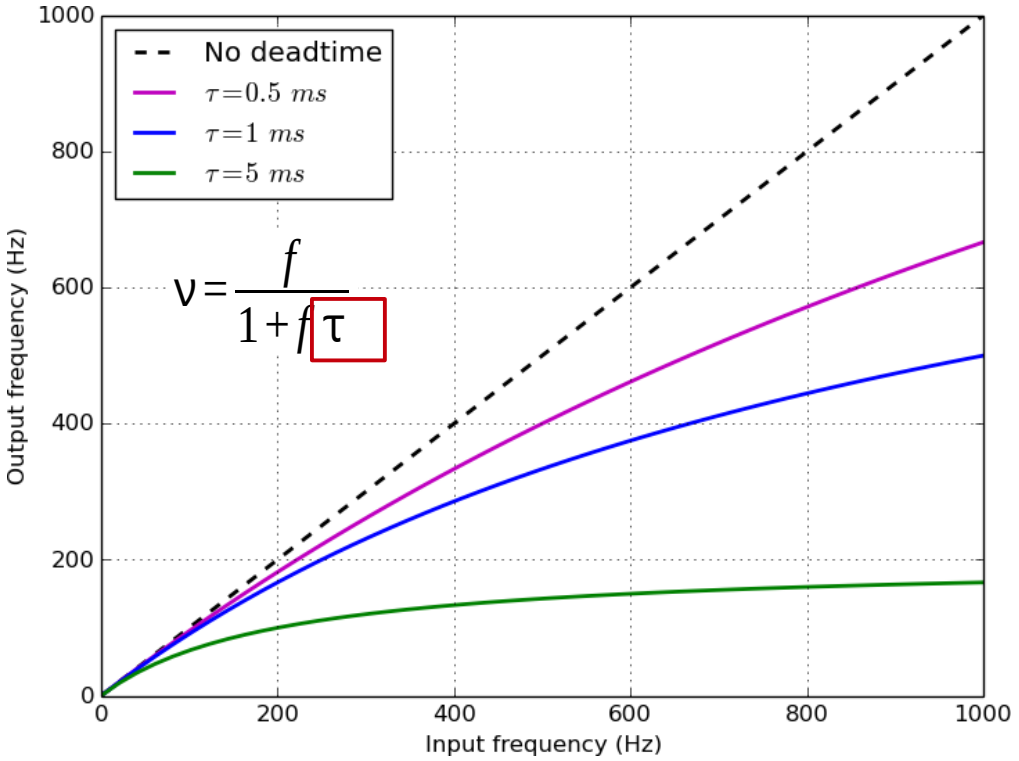
Deadtime and efficiency





- In order to obtain ε~100% (i.e.: ν~f) $\rightarrow f\tau << 1 \rightarrow \tau << \lambda$
  - E.g.: ε~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?

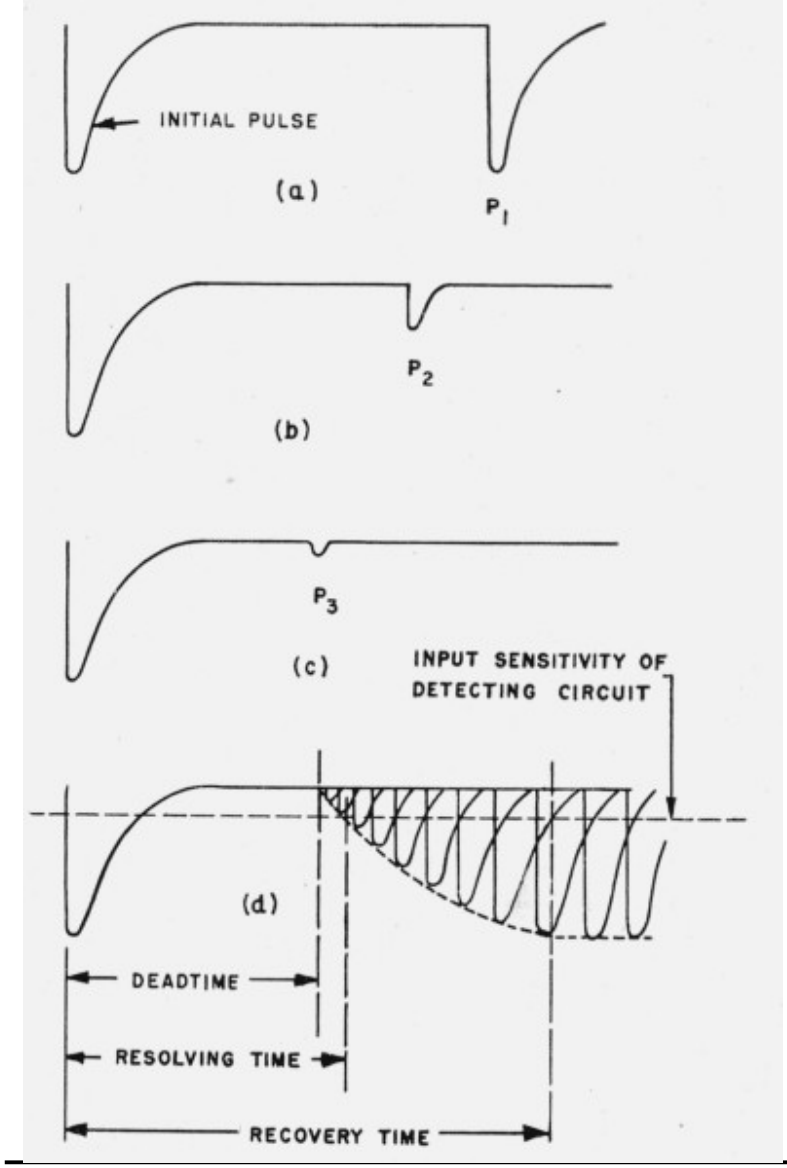
Deadtime and efficiency



- In order to obtain ε~100% (i.e.: ν~f) $\rightarrow f\tau << 1 \rightarrow \tau << \lambda$
  - E.g.: ε~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?

Length of the BUSY

- Lesson from Geiger-Muller counter



Detector deadtime: no 2nd signal period

Resolving time: minimal time between two pulses

Recovery time: time after which the detector becomes fully efficient

BUSY length: depends on the application, but it is better to have it *greater* than the detector recovery time, so that all measurements are taken in the same detector conditions

Live time / total time

- For cross section measurements you need to know the “live” beam intensity (used beam). This is the beam intensity when the trigger is enabled.
- **Total time (T_{tot})**: time elapsed from the first enable of the trigger to the stop of the DAQ
- **Live time (T_{live})**: sum of the periods where the trigger is enabled

For stable beams

$$I_{live} = I_{beam} \frac{T_{live}}{T_{tot}}$$

Decision time

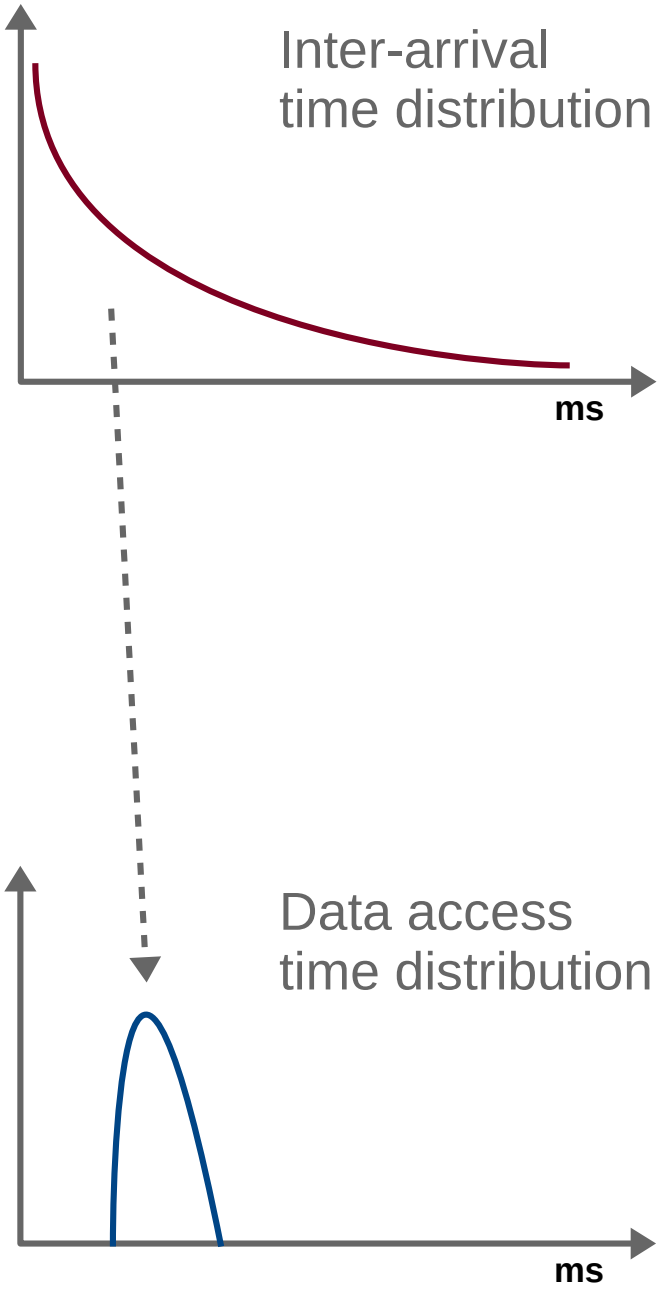
- What can we do to improve the efficiency of the system?
- **Very low dead time**: is expensive!
- **Shortest possible busy**: efficiencies not under control

DAQ aims to take data in a **controlled** fashion

(money & efficiency)

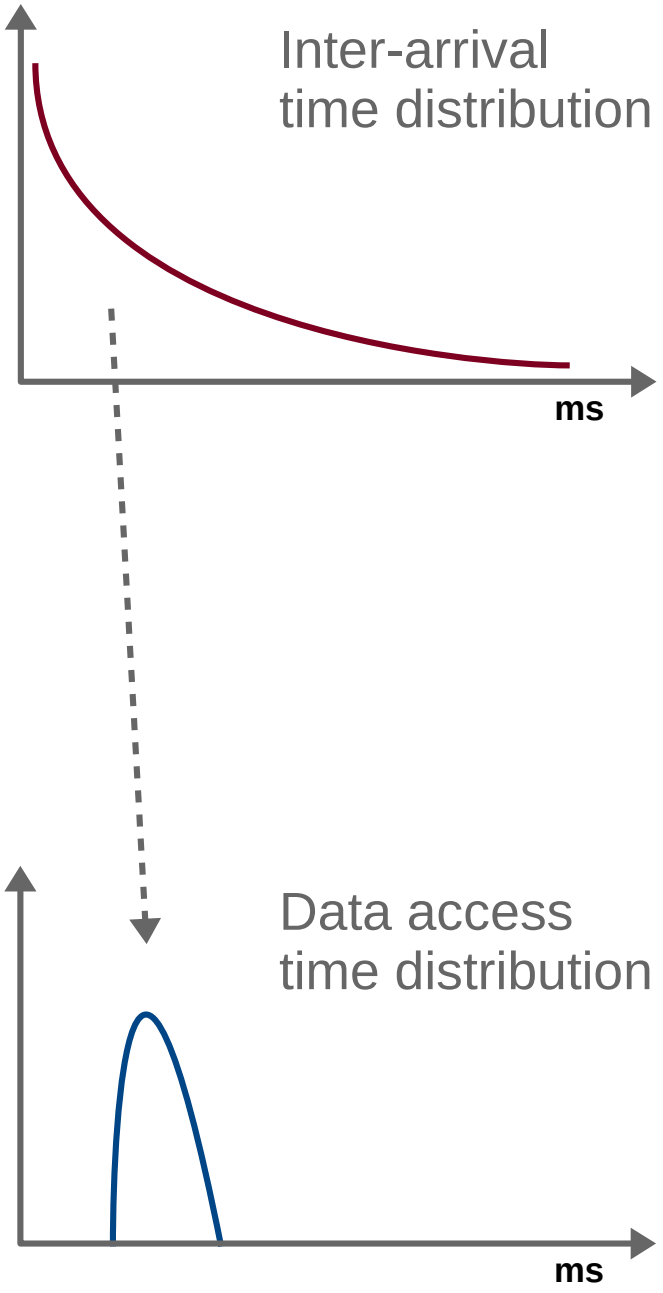
De-randomization

- 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?



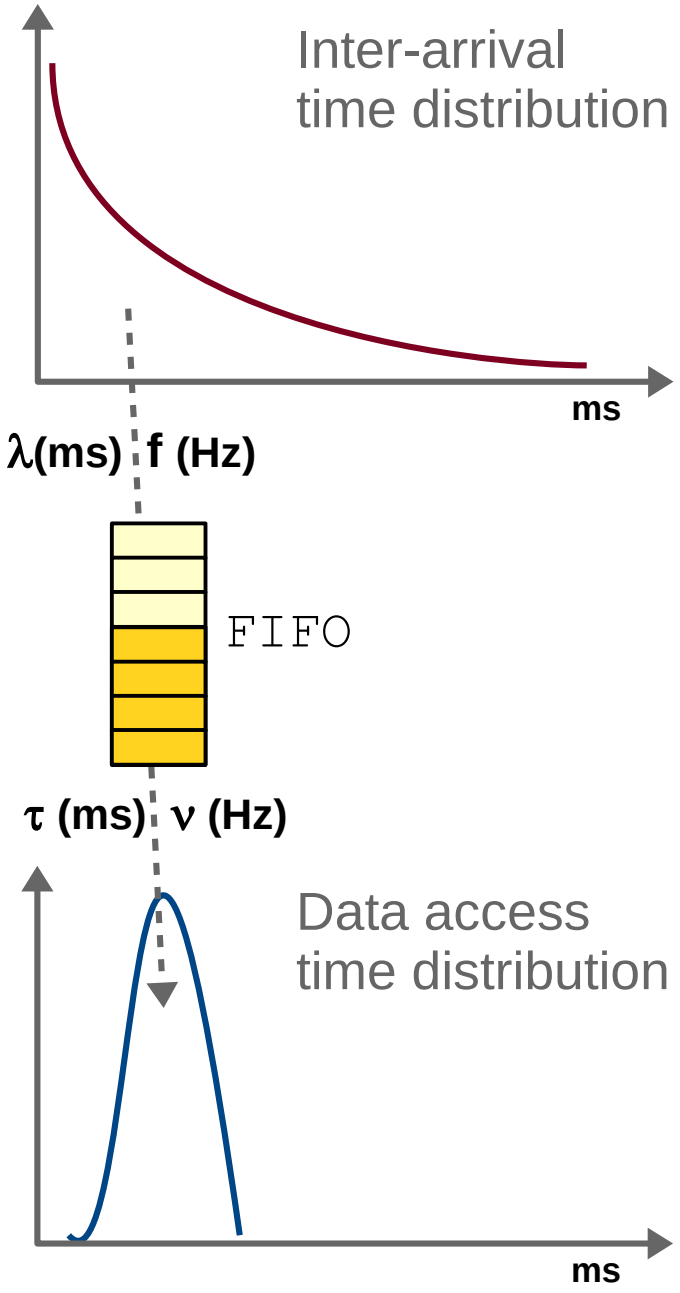
De-randomization

- 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?



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 we can slot it up to be processed)

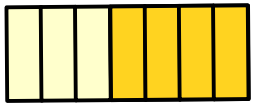


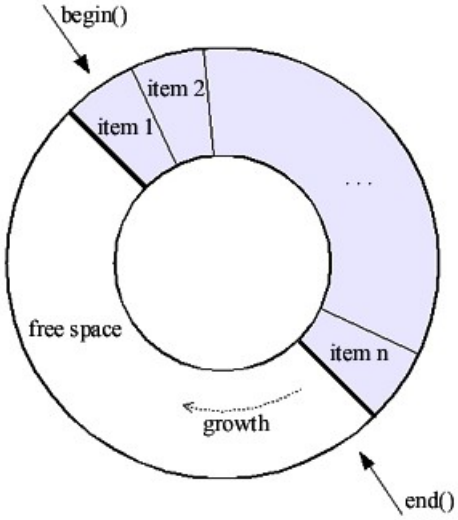
FIFOs, Circular buffers, deque

- Same concept with different names
- FIFO: First In, First Out
- FIXED storage area, with a writing and a reading pointer and control signals (empty, full, almost full)

anItem= deque.pop_front();

FIFO:



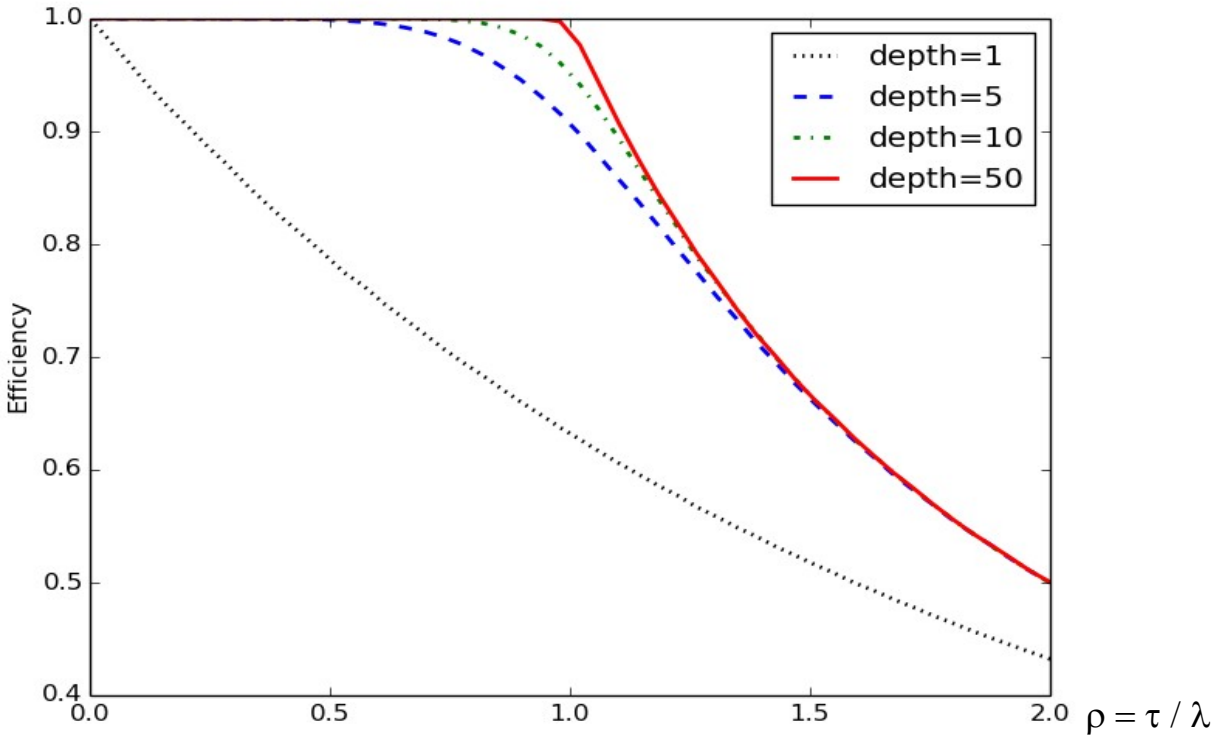


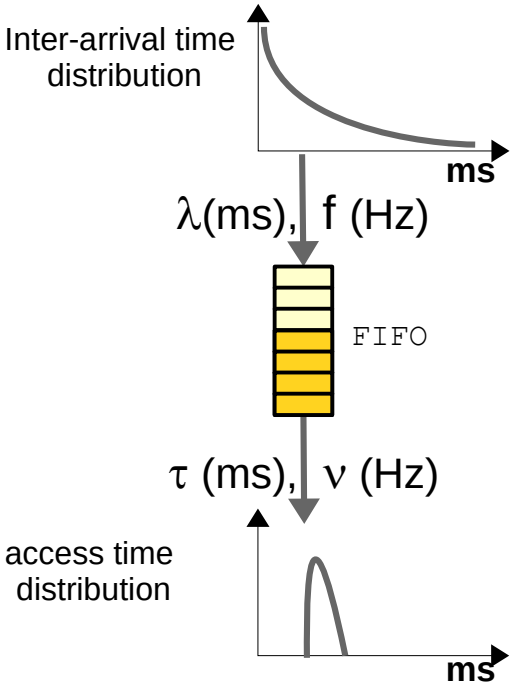
C++ stl sequence container:

deque.push_back(anItem);

```
std::deque<int32_t> myCircularBuffer(1000);
```

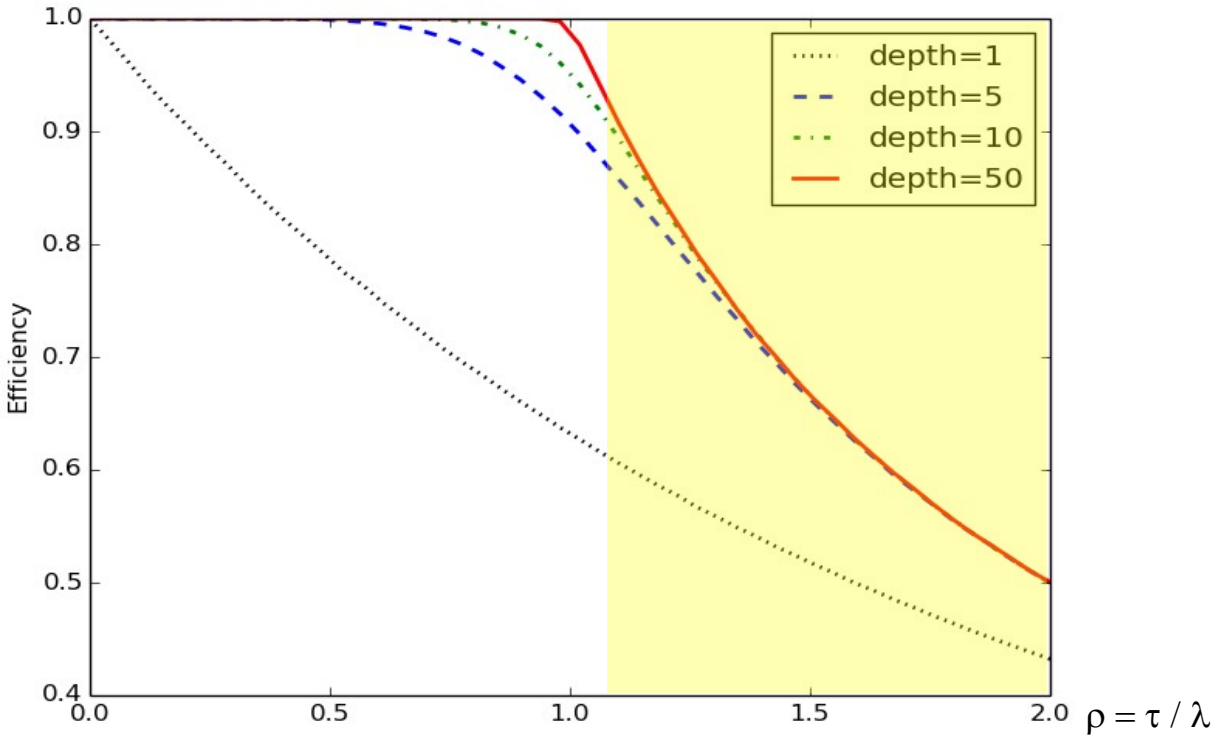
Queuing theory

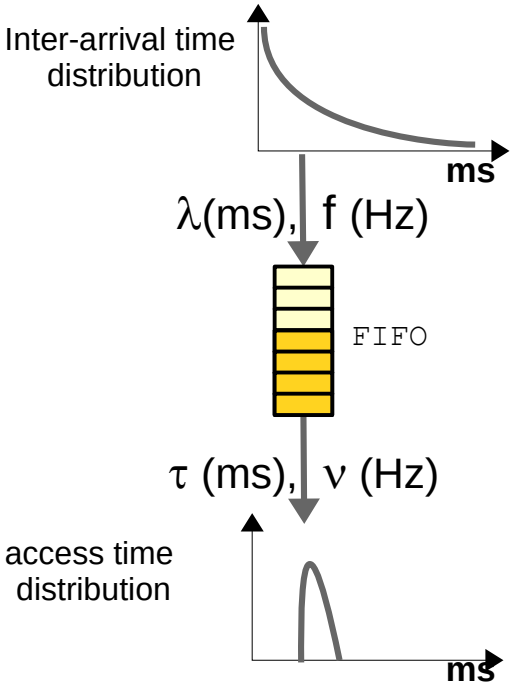




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - $\rho > 1$: the system is overloaded ($\tau > \lambda$)
  - $\rho << 1$: the output is over-designed ($\tau << \lambda$)
  - $\rho \sim 1$: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

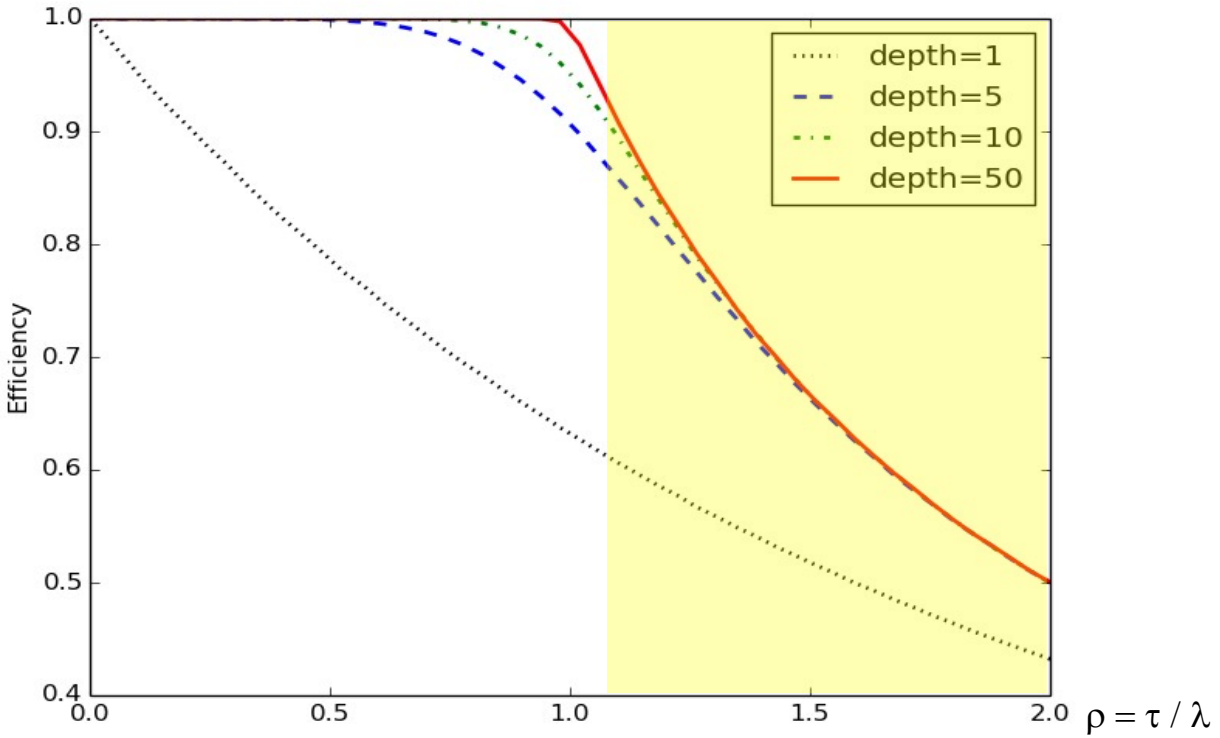
Queuing theory

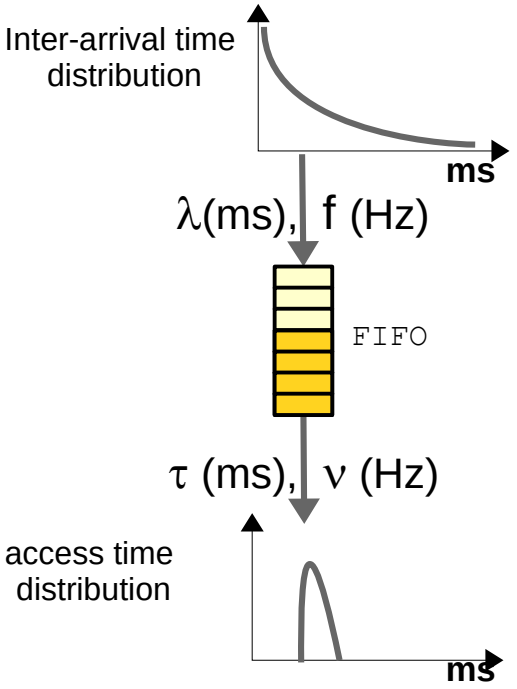




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - **$\rho > 1$**: the system is overloaded ($\tau > \lambda$)
  - **$\rho << 1$**: the output is over-designed ($\tau << \lambda$)
  - **$\rho \sim 1$**: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

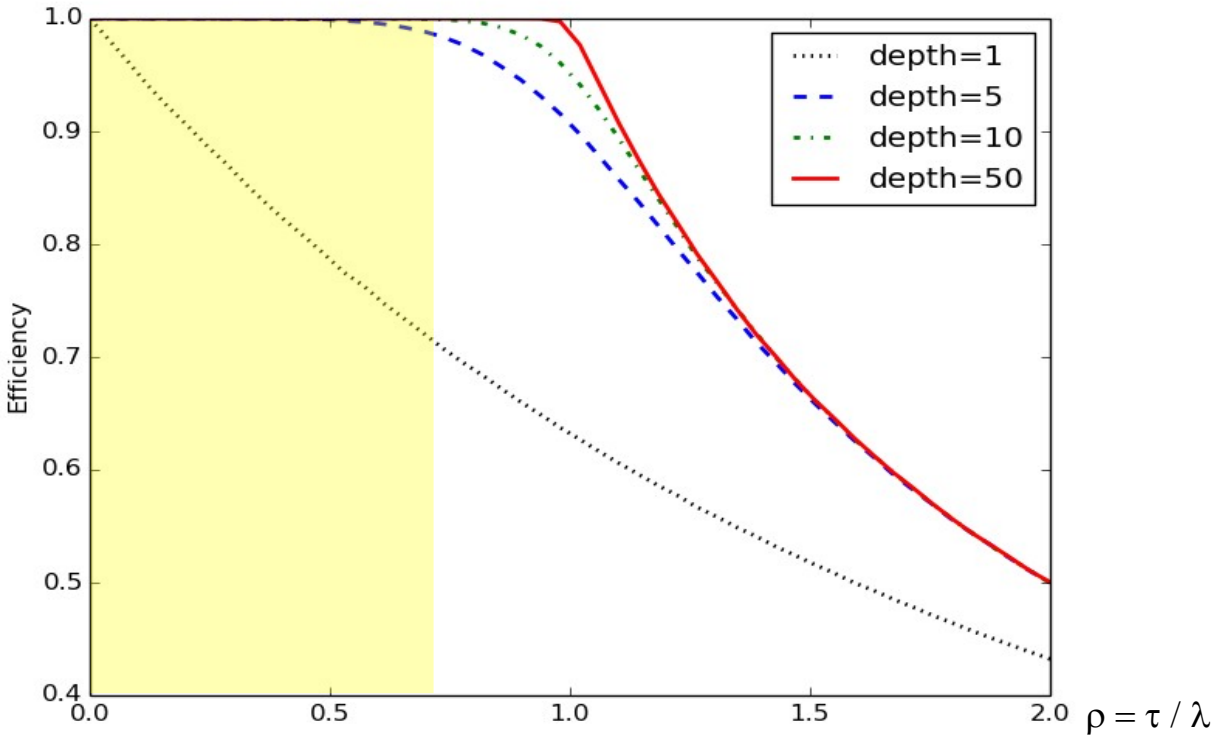
Queuing theory

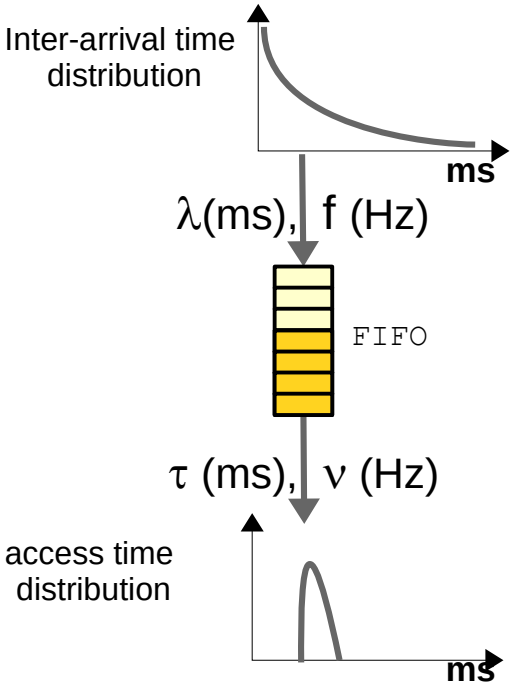




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - **$\rho > 1$**: the system is overloaded ($\tau > \lambda$)
  - **$\rho << 1$**: the output is over-designed ($\tau << \lambda$)
  - **$\rho \sim 1$**: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

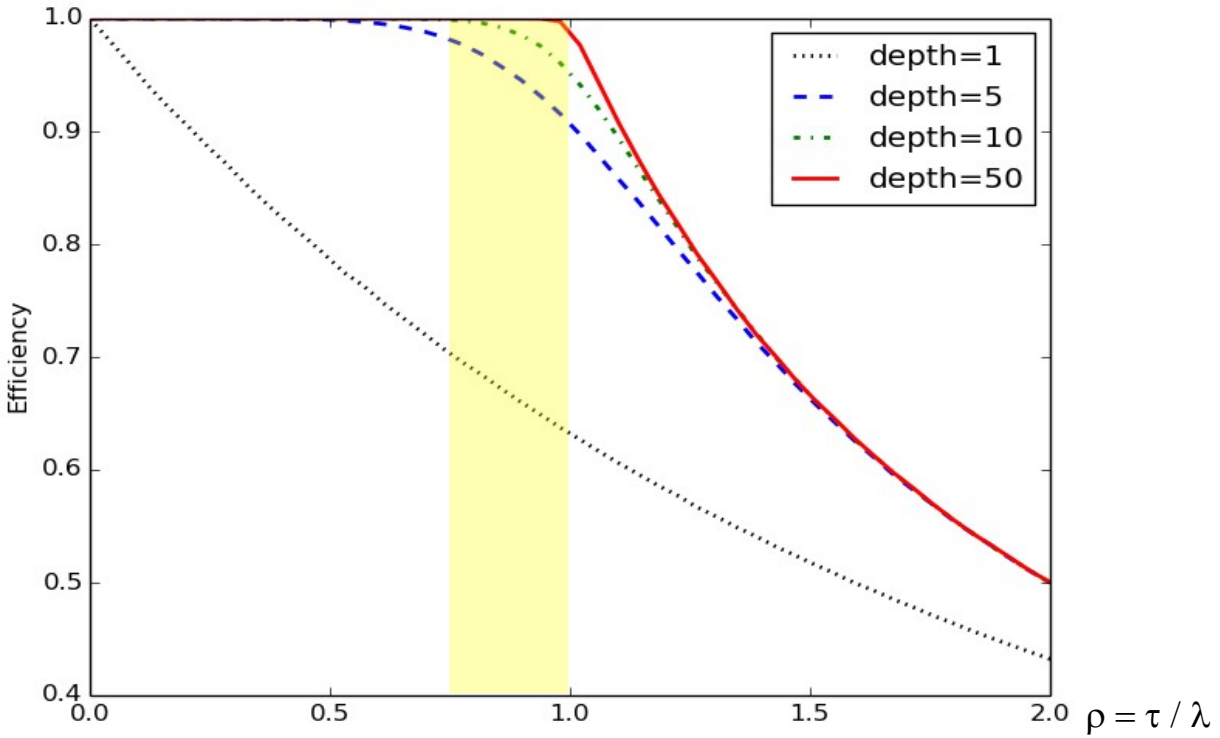
Queuing theory

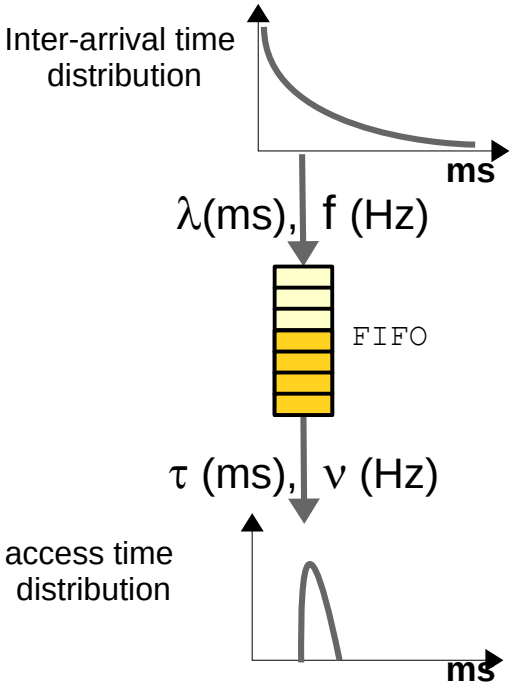




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - **$\rho > 1$**: the system is overloaded ($\tau > \lambda$)
  - **$\rho << 1$**: the output is over-designed ($\tau << \lambda$)
  - **$\rho \sim 1$**: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

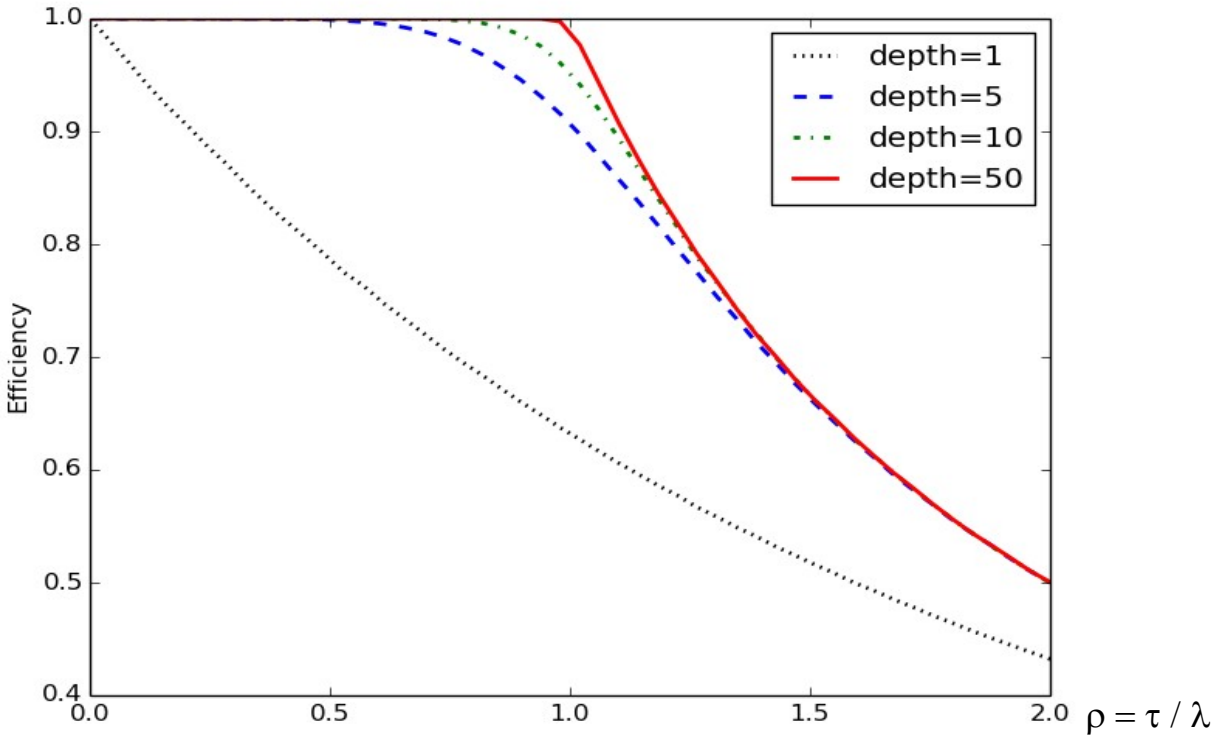
Queuing theory

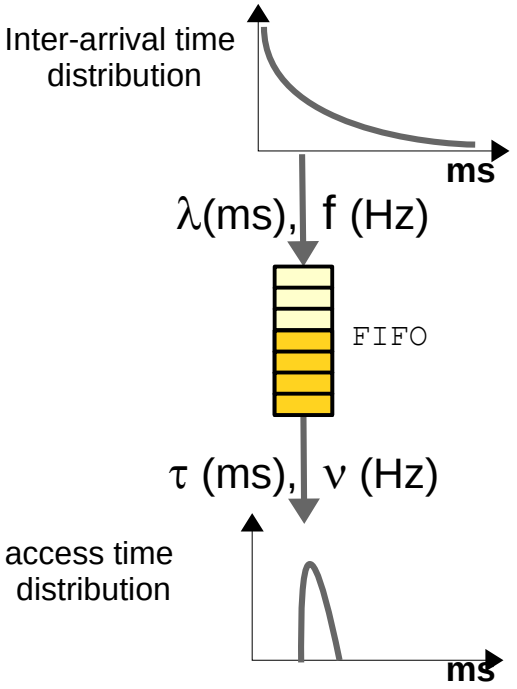




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - **$\rho > 1$**: the system is overloaded ($\tau > \lambda$)
  - **$\rho << 1$**: the output is over-designed ($\tau << \lambda$)
  - **$\rho \sim 1$**: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

Queuing theory

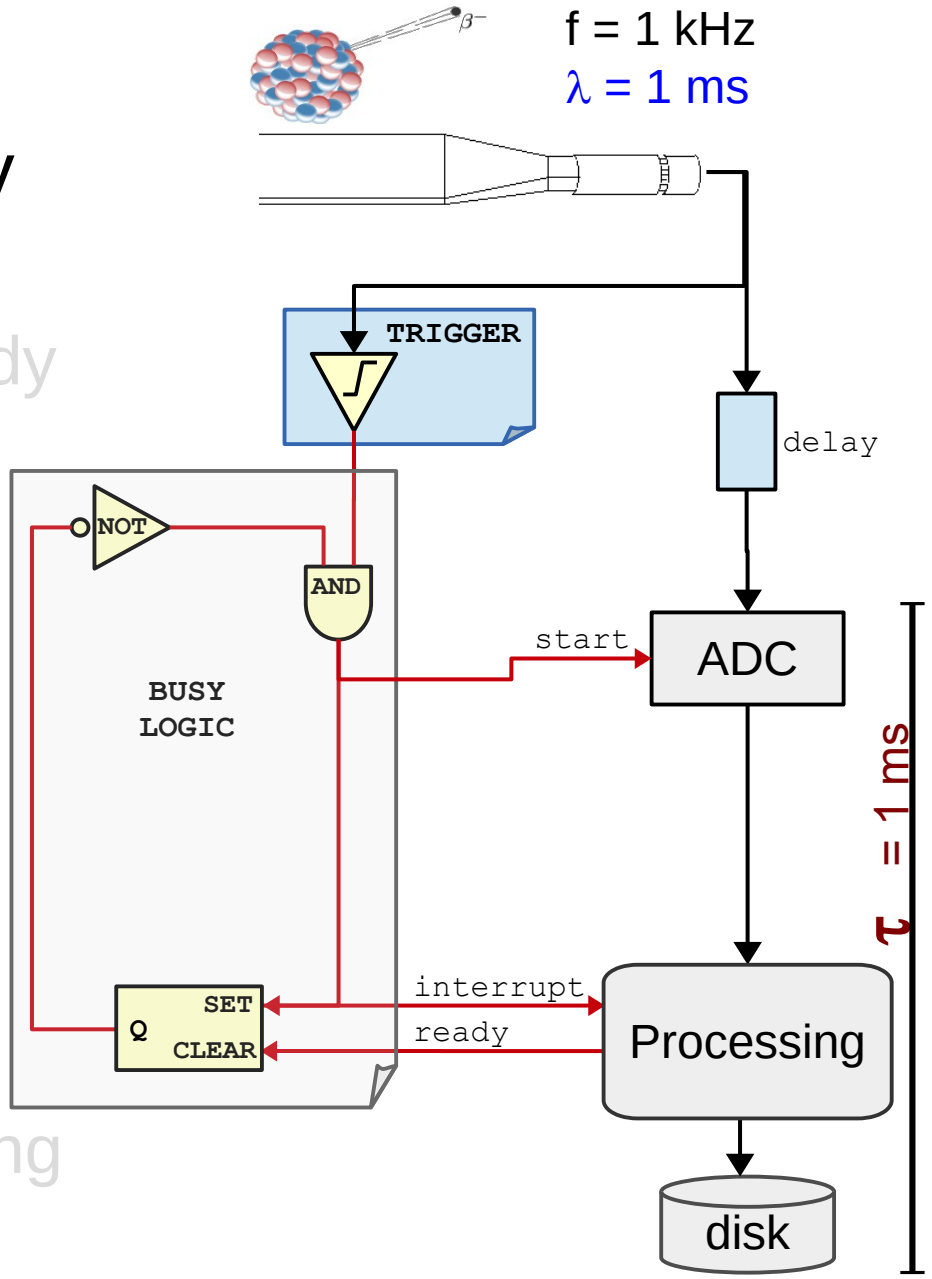




- Efficiency vs traffic intensity ($\rho = \tau / \lambda$) for different queue depths
  - $\rho > 1$: the system is overloaded ($\tau > \lambda$)
  - $\rho << 1$: the output is over-designed ($\tau << \lambda$)
  - $\rho \sim 1$: using a queue, high efficiency obtained even w/ moderate depth
- Analytic calculation possible for very simple systems only
  - Otherwise MonteCarlo simulation is required

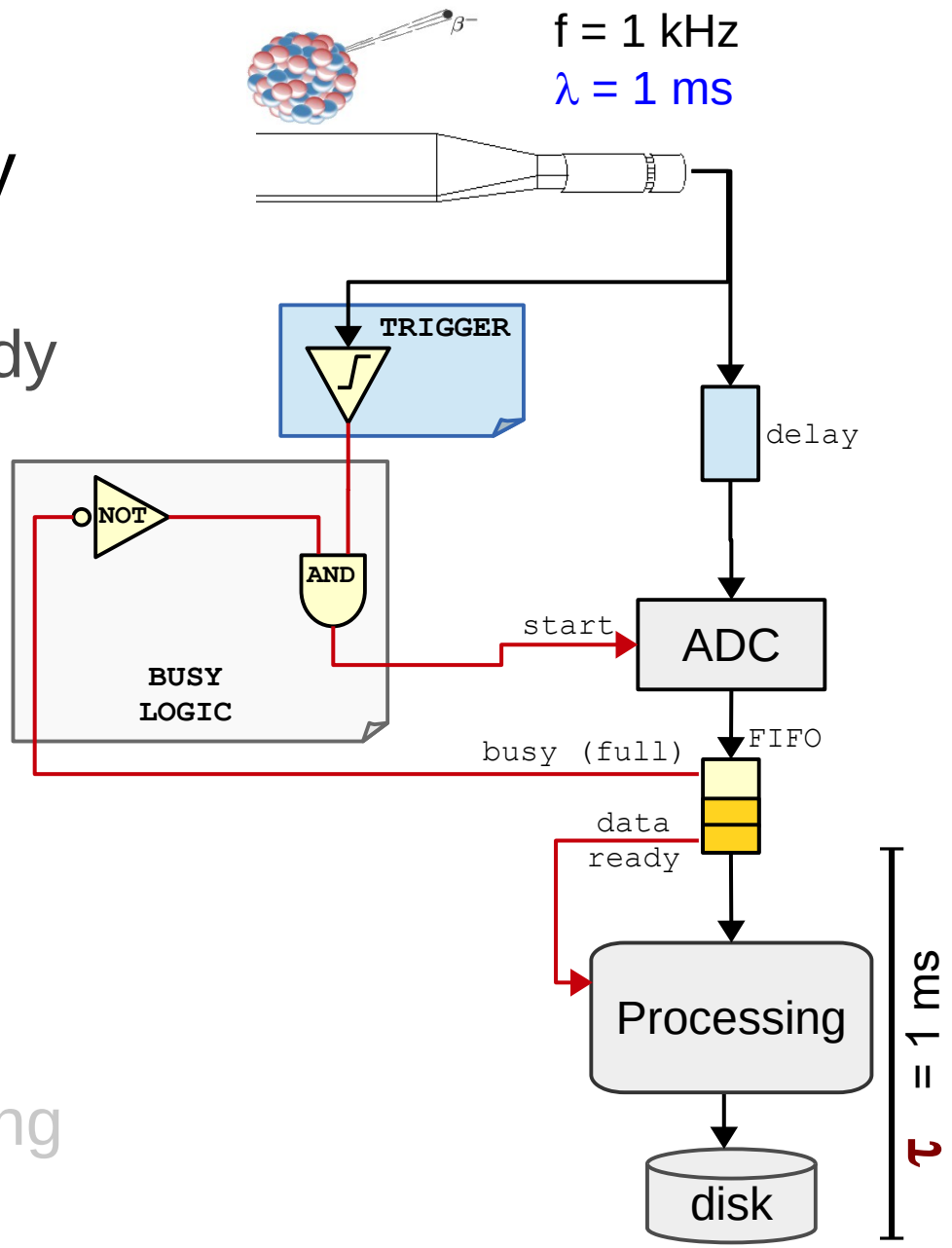
De-randomization

- 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



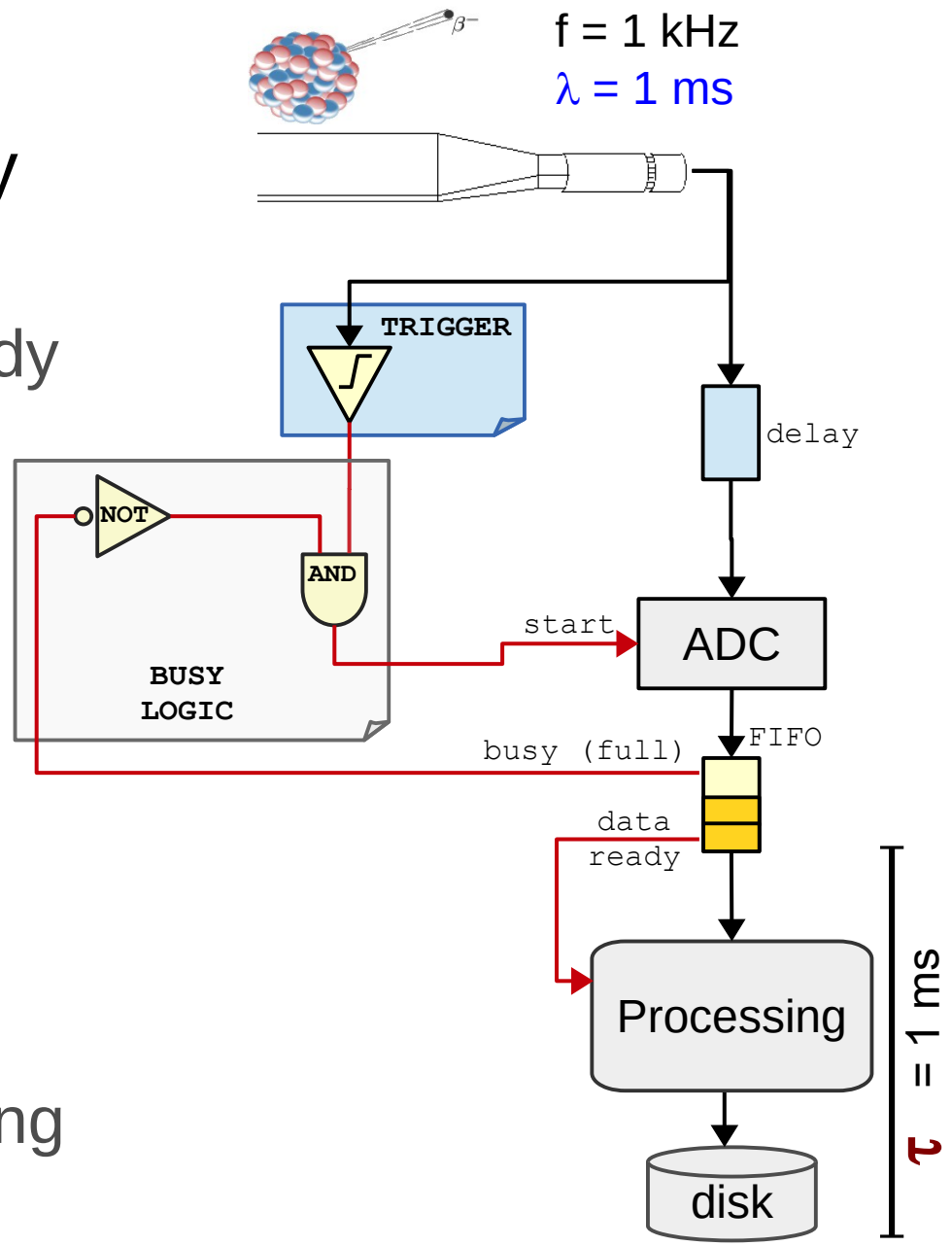
De-randomization

- 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



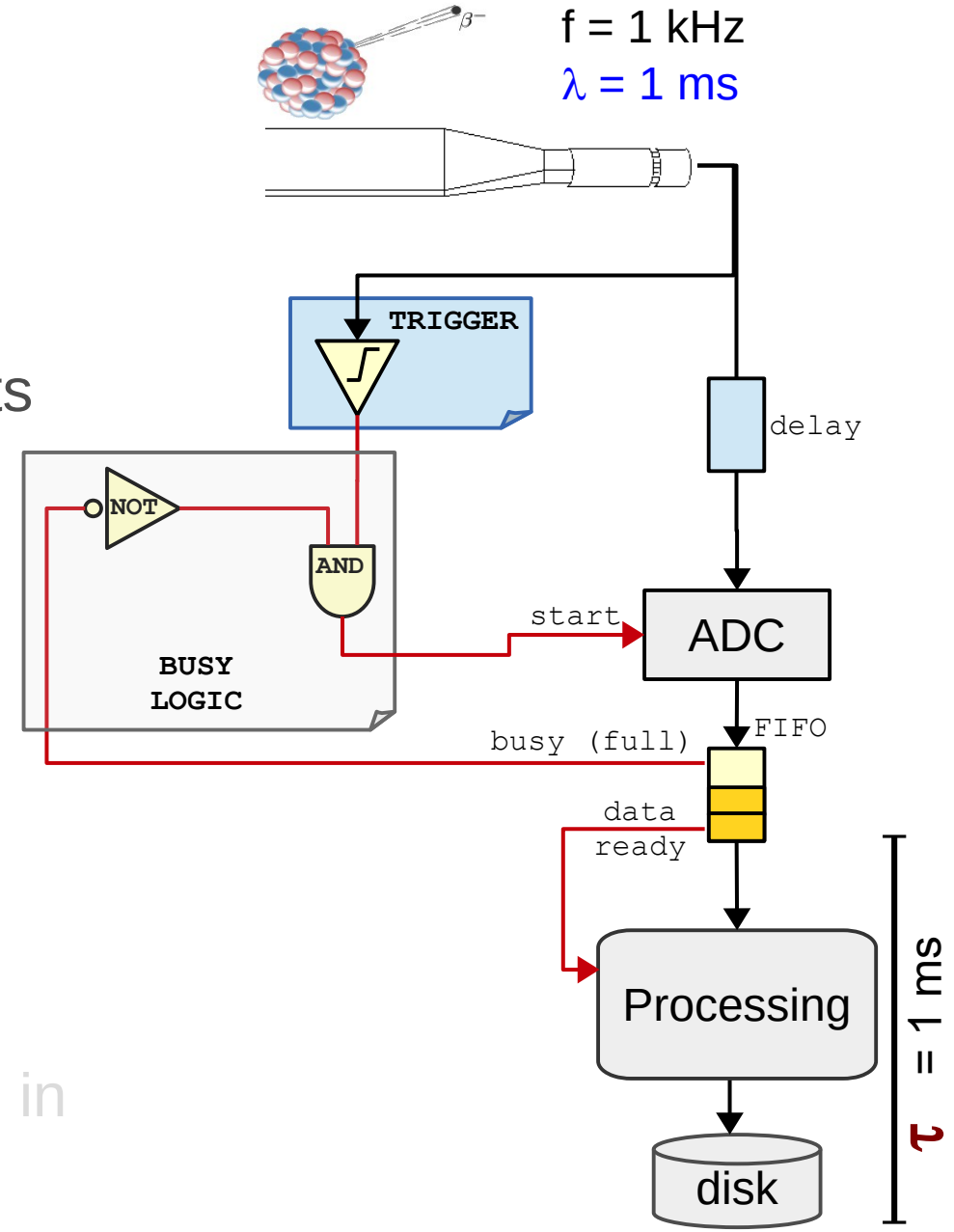
De-randomization

- 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



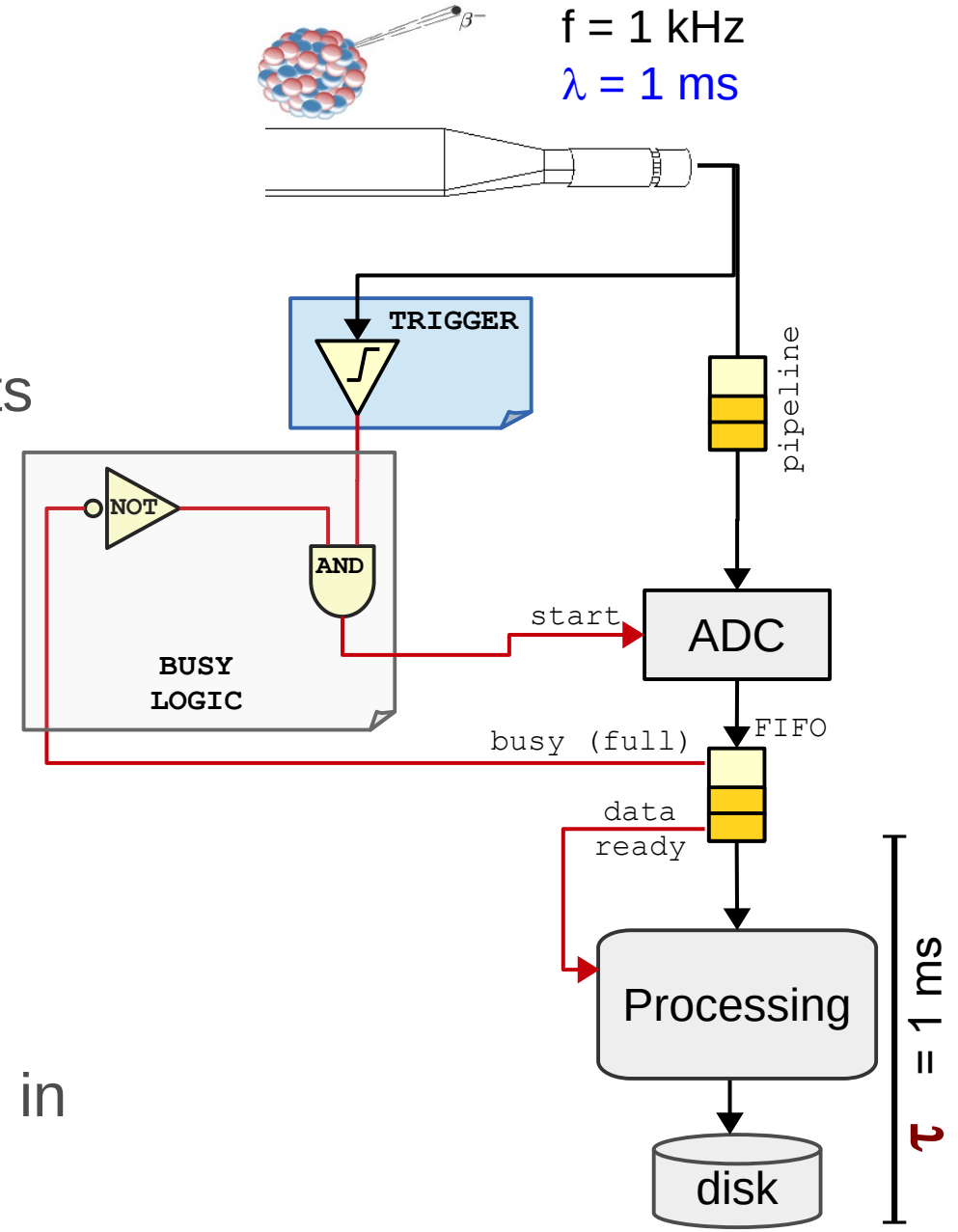
De-randomization summary

- 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 >> f
  - Data processing and storing operate at a rate ~ f
- Could the delay be replaced with a “FIFO”?
  - Analog pipelines, heavily used in LHC DAQs



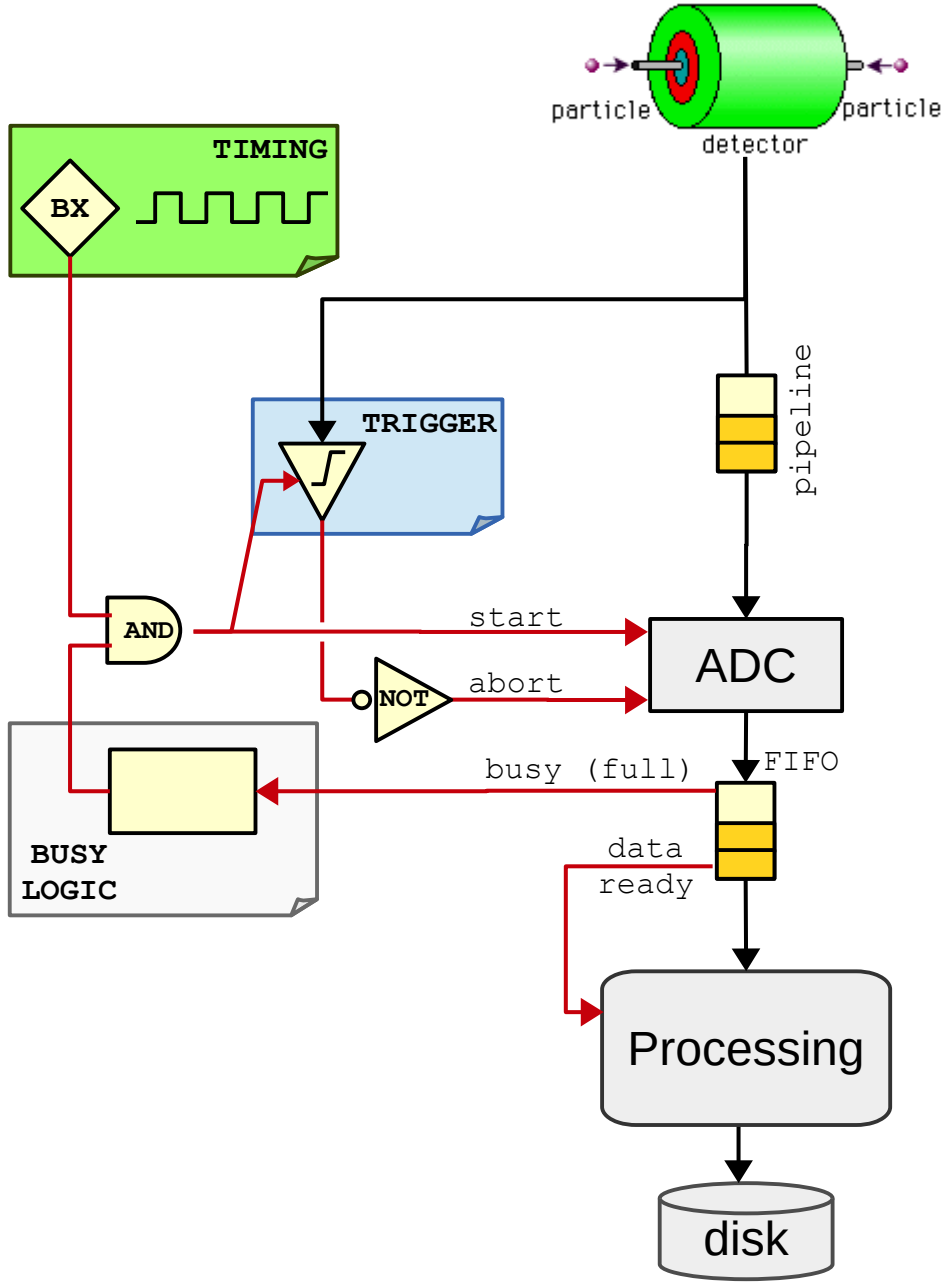
De-randomization summary

- 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 >> f
  - Data processing and storing operate at a rate ~ f
- Could the delay be replaced with a “FIFO”?
  - Analog pipelines, heavily used in LHC DAQs



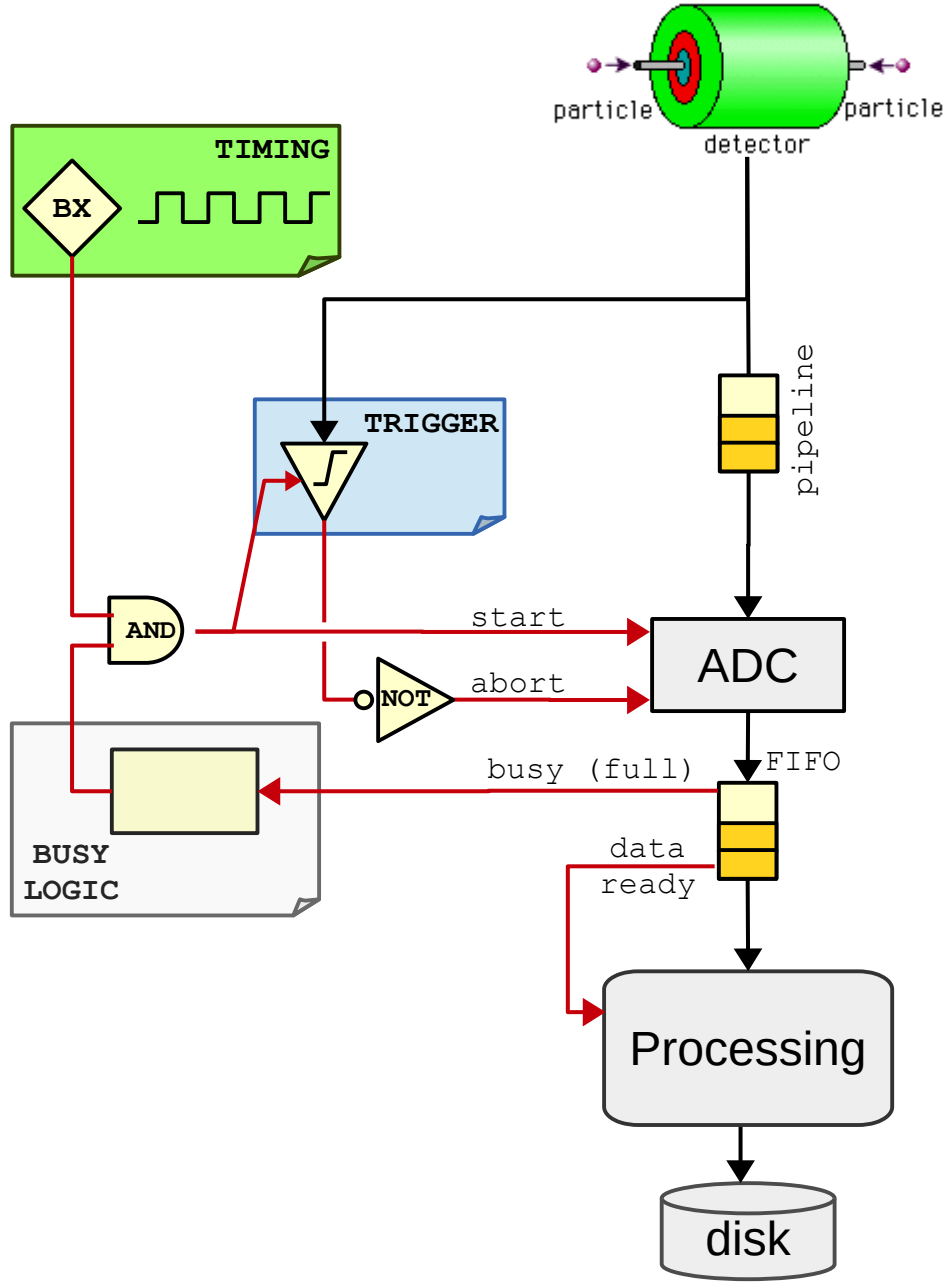
Collider setup

- Do we need de-randomization buffers also in collider setups?
  - Particle collisions are synchronous
  - But the time distribution of triggers is random: good 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: good 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

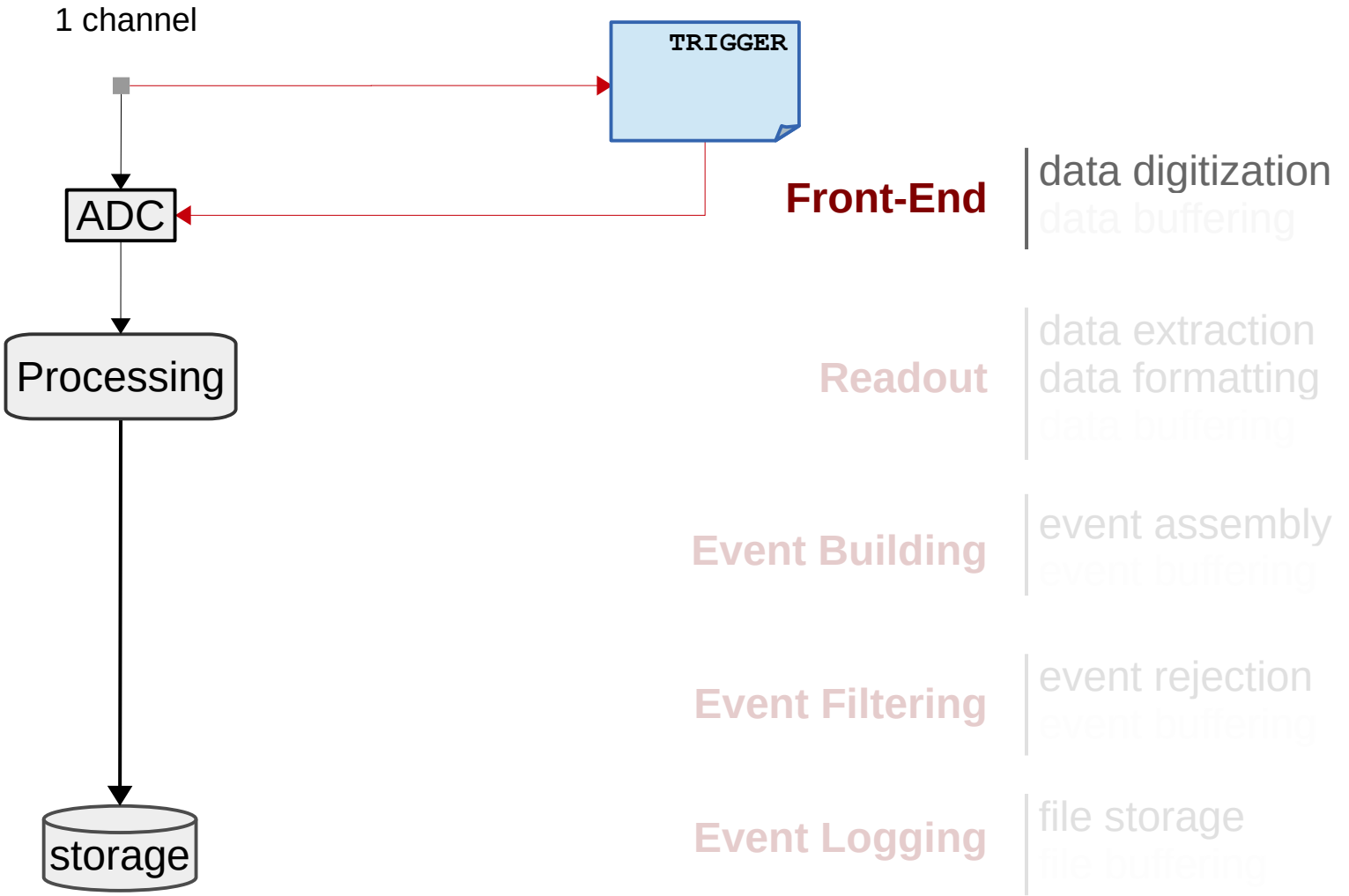


Outline

- Introduction
  - What is DAQ?
  - Overall framework
- Basic DAQ concepts
  - Digitization, Latency
  - Deadtime, Busy, Backpressure
  - De-randomization
- Scaling up
  - Readout and Event Building
  - Buses vs Network
- Data encoding

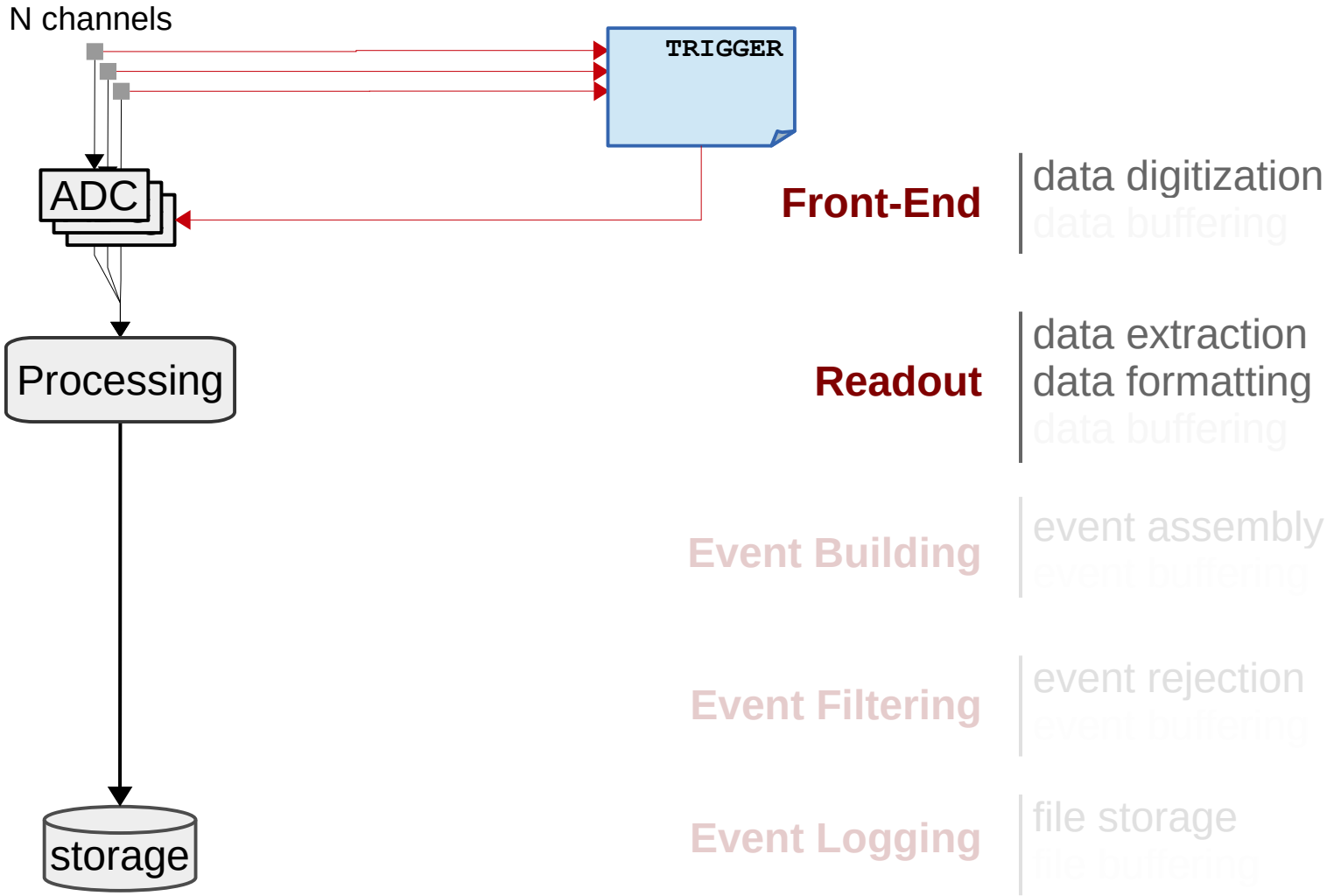
Adding more channels

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



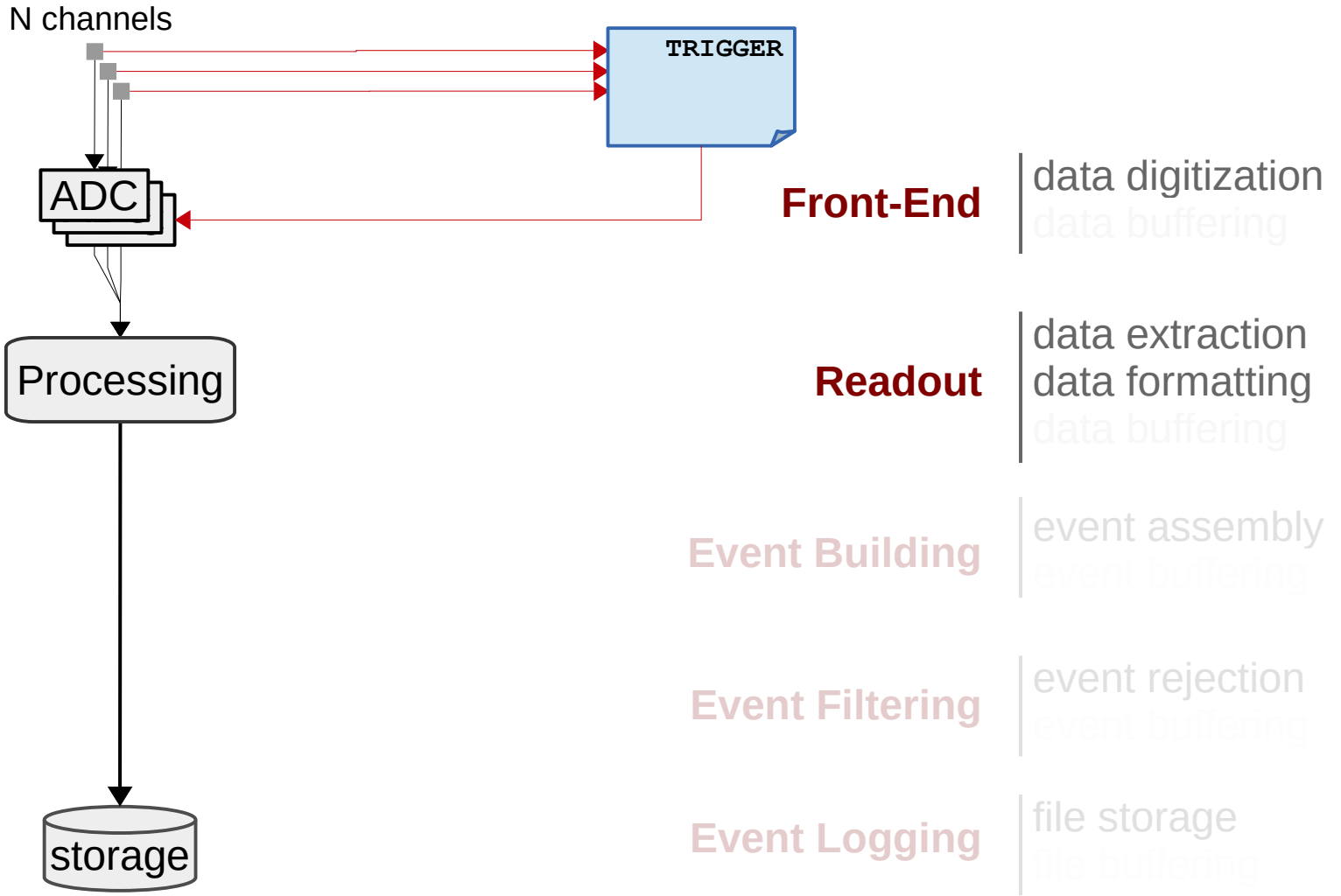
Adding more channels

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



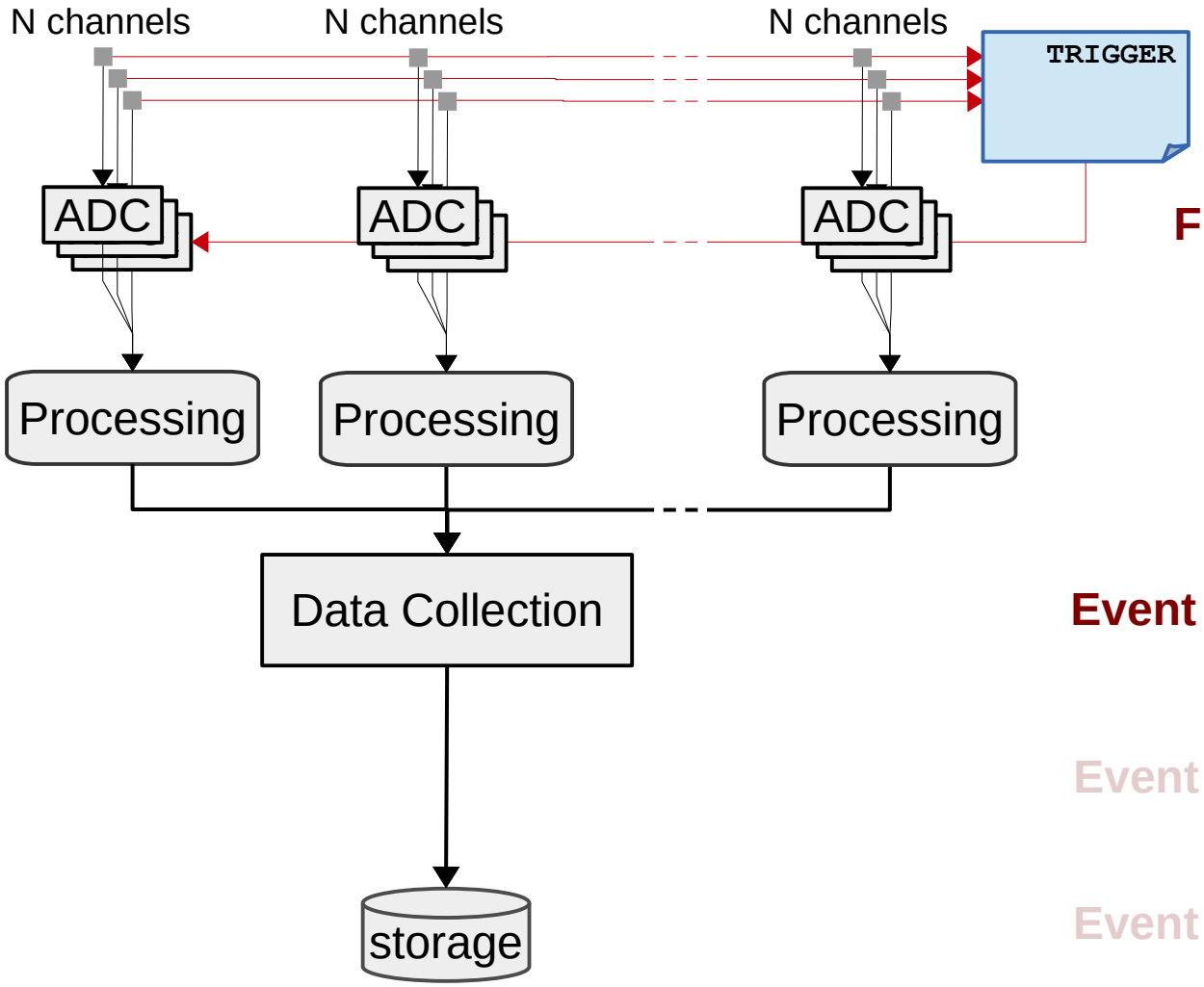
Adding more channels

- 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



Front-End — data digitization, data buffering

Readout — data extraction, data formatting, data buffering

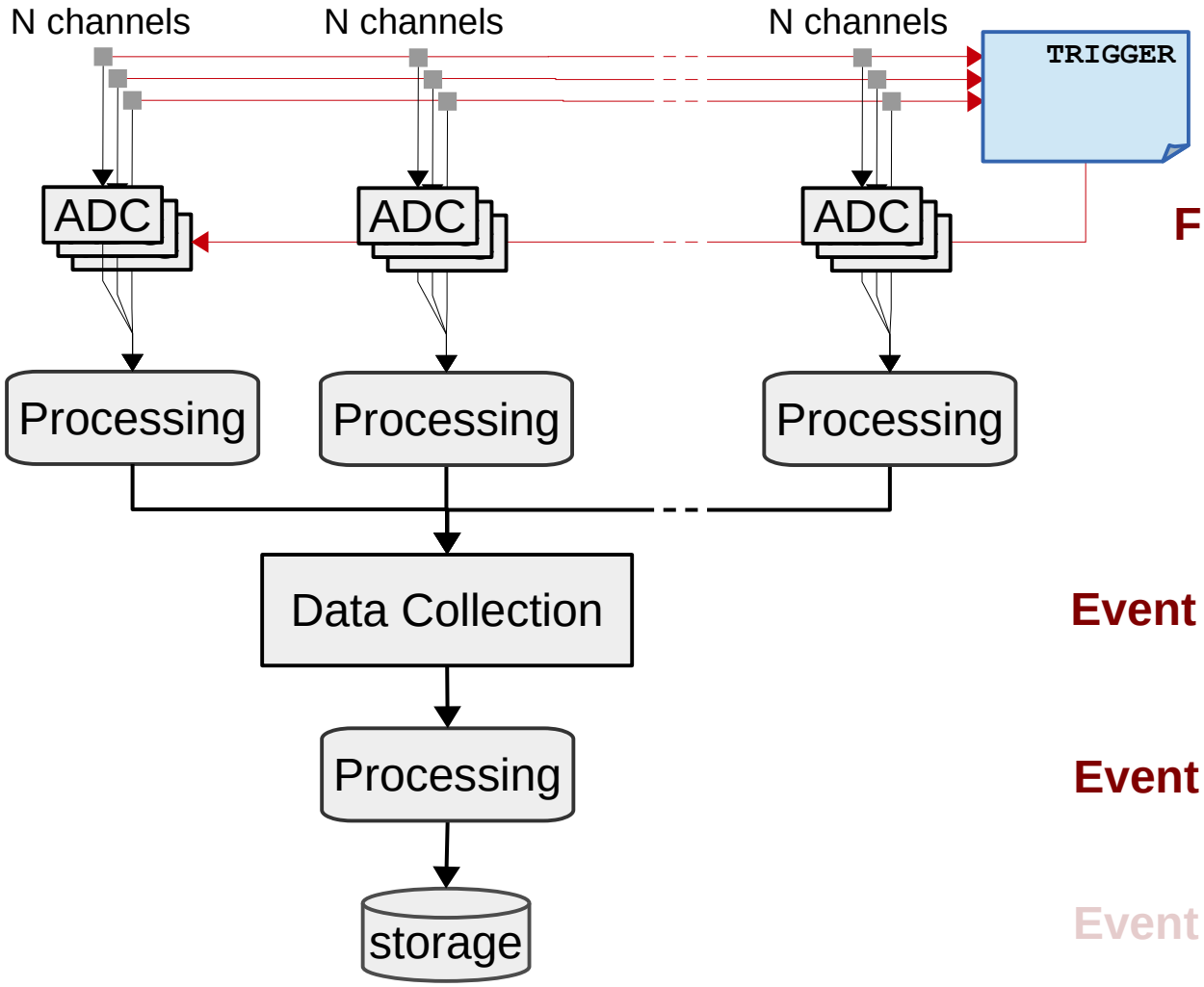
Event Building — event assembly, event buffering

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

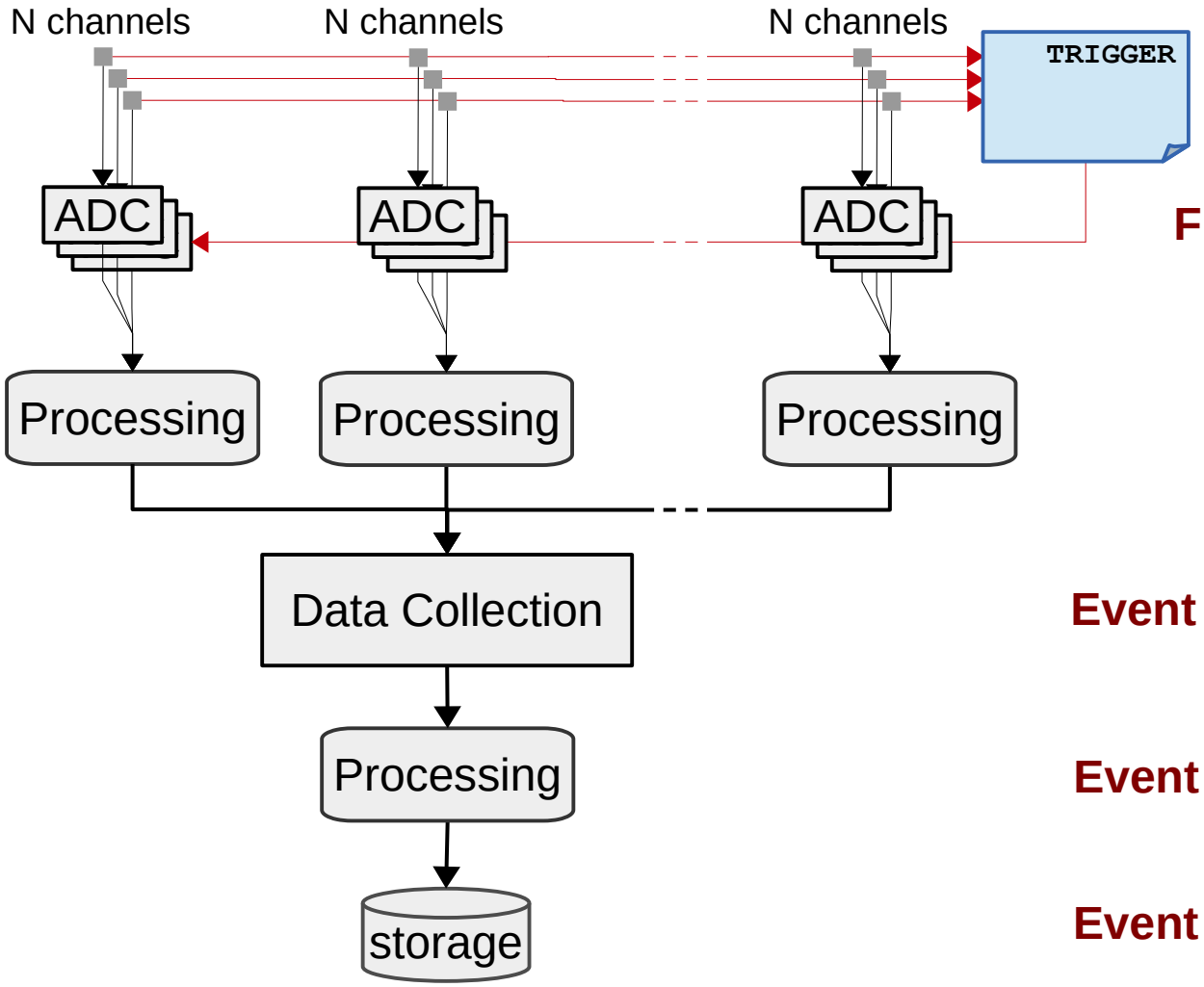
Event Building — event assembly, event buffering

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

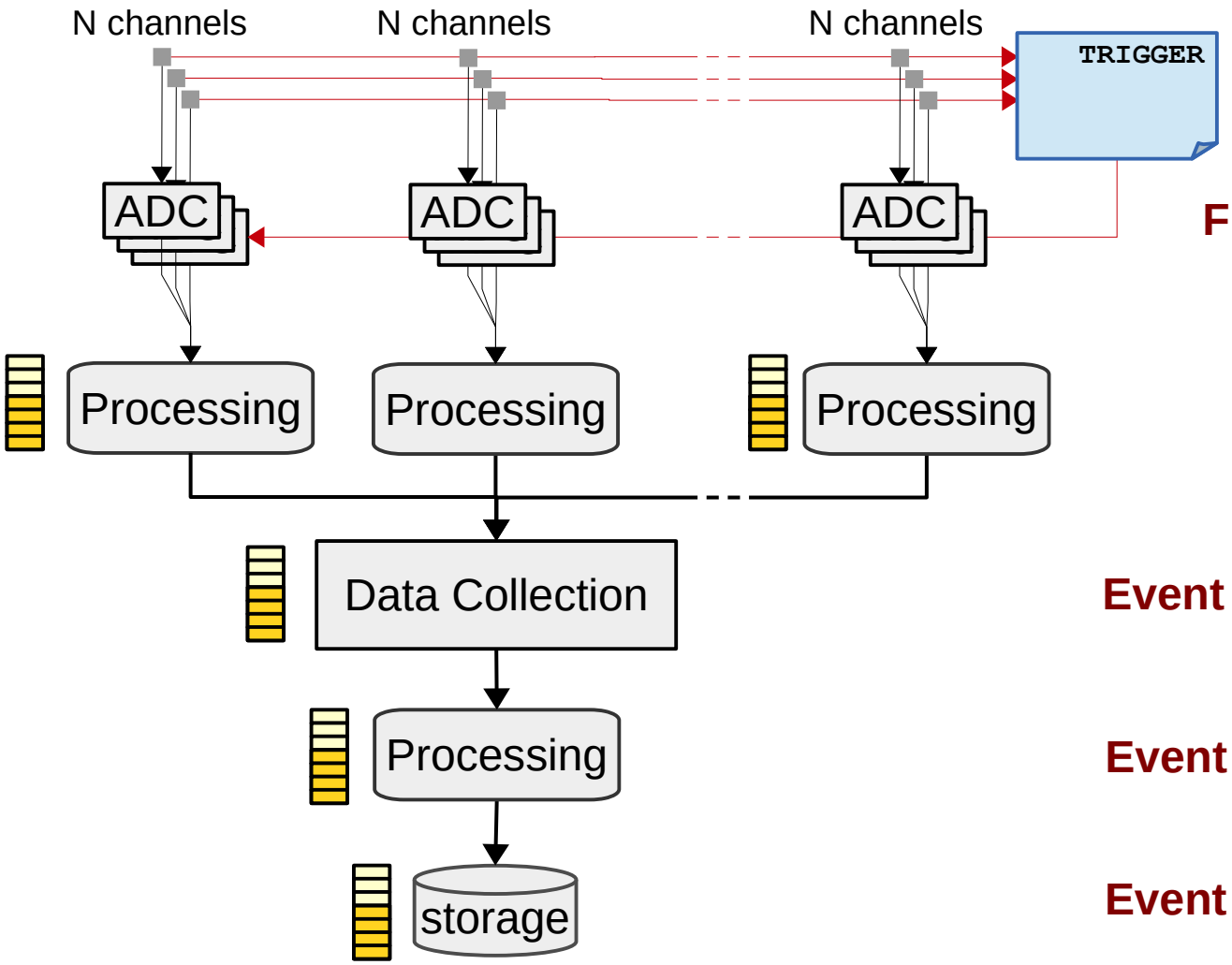
Event Building — event assembly, event buffering

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**

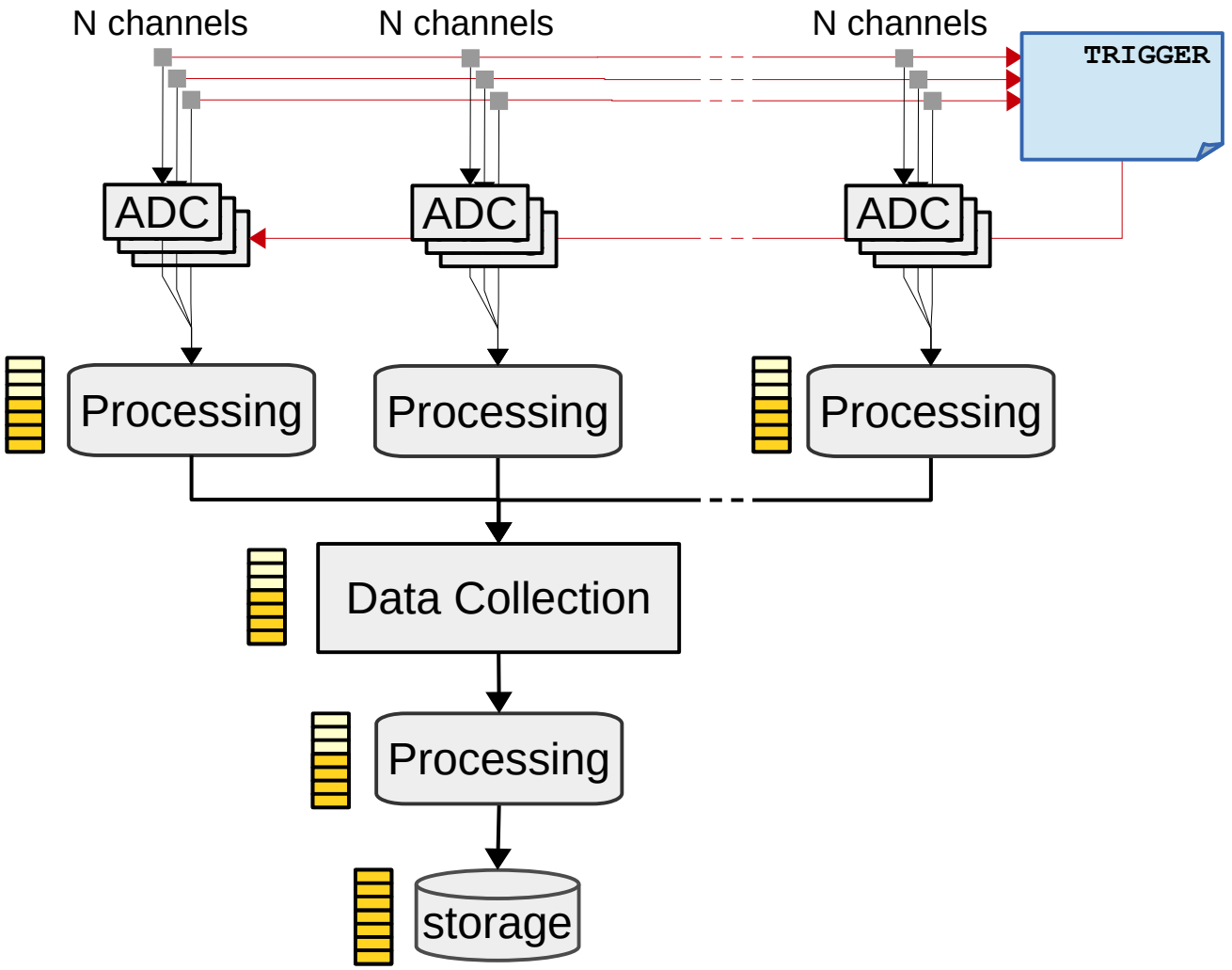
Event Building
event assembly
event **buffering**

Event Filtering
event rejection
event **buffering**

Event Logging
file storage
file **buffering**

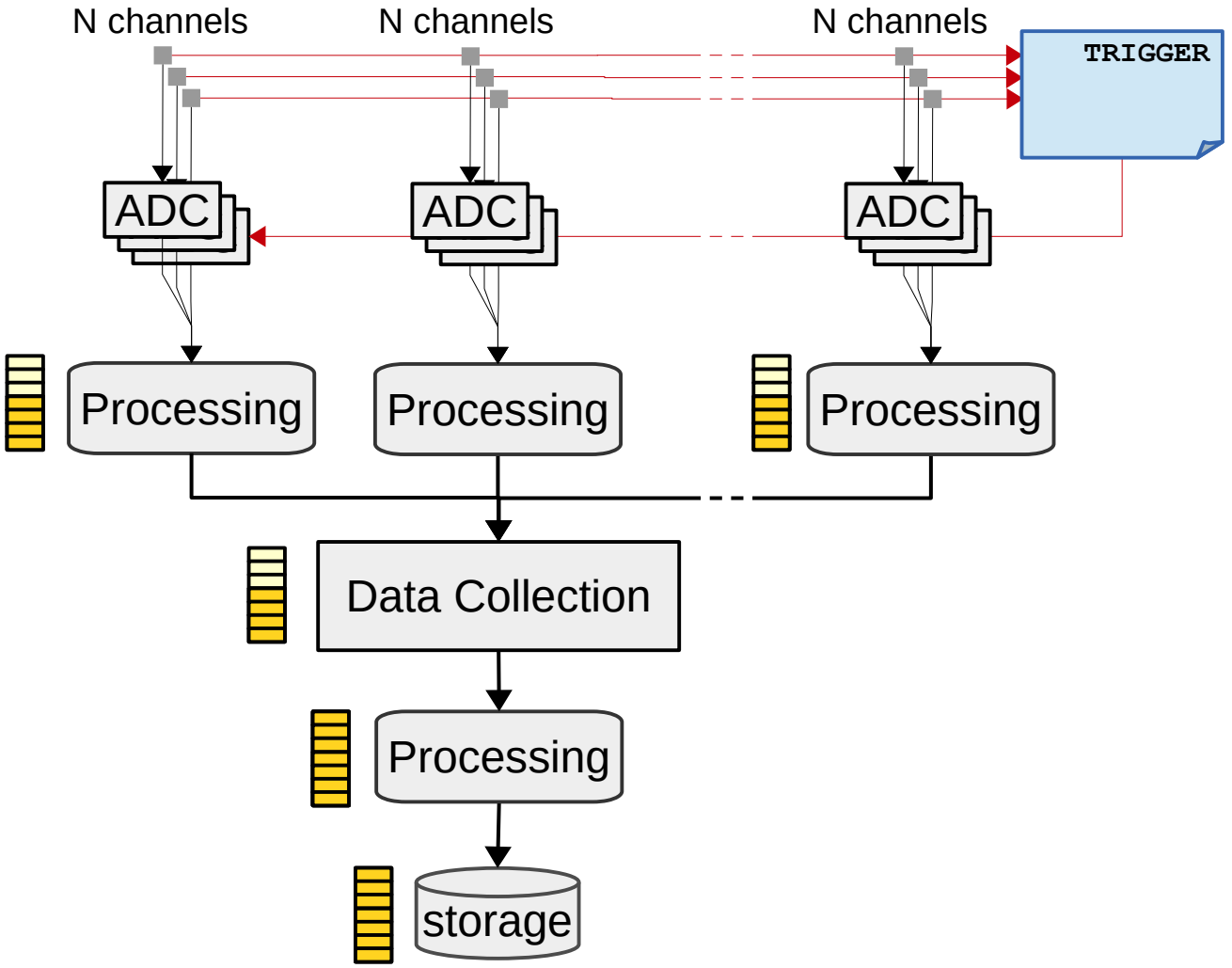
Backpressure

- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)



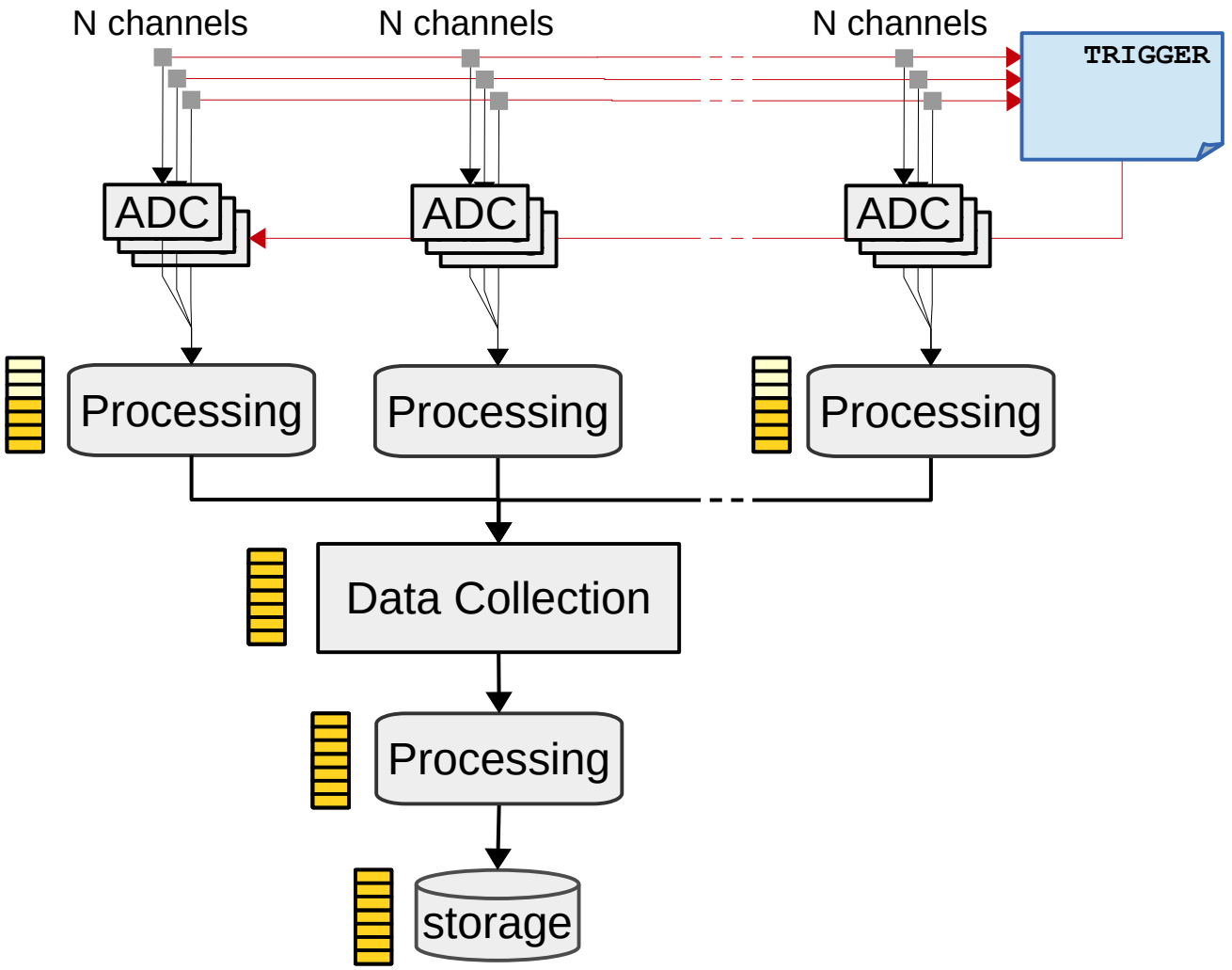
Backpressure

- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)



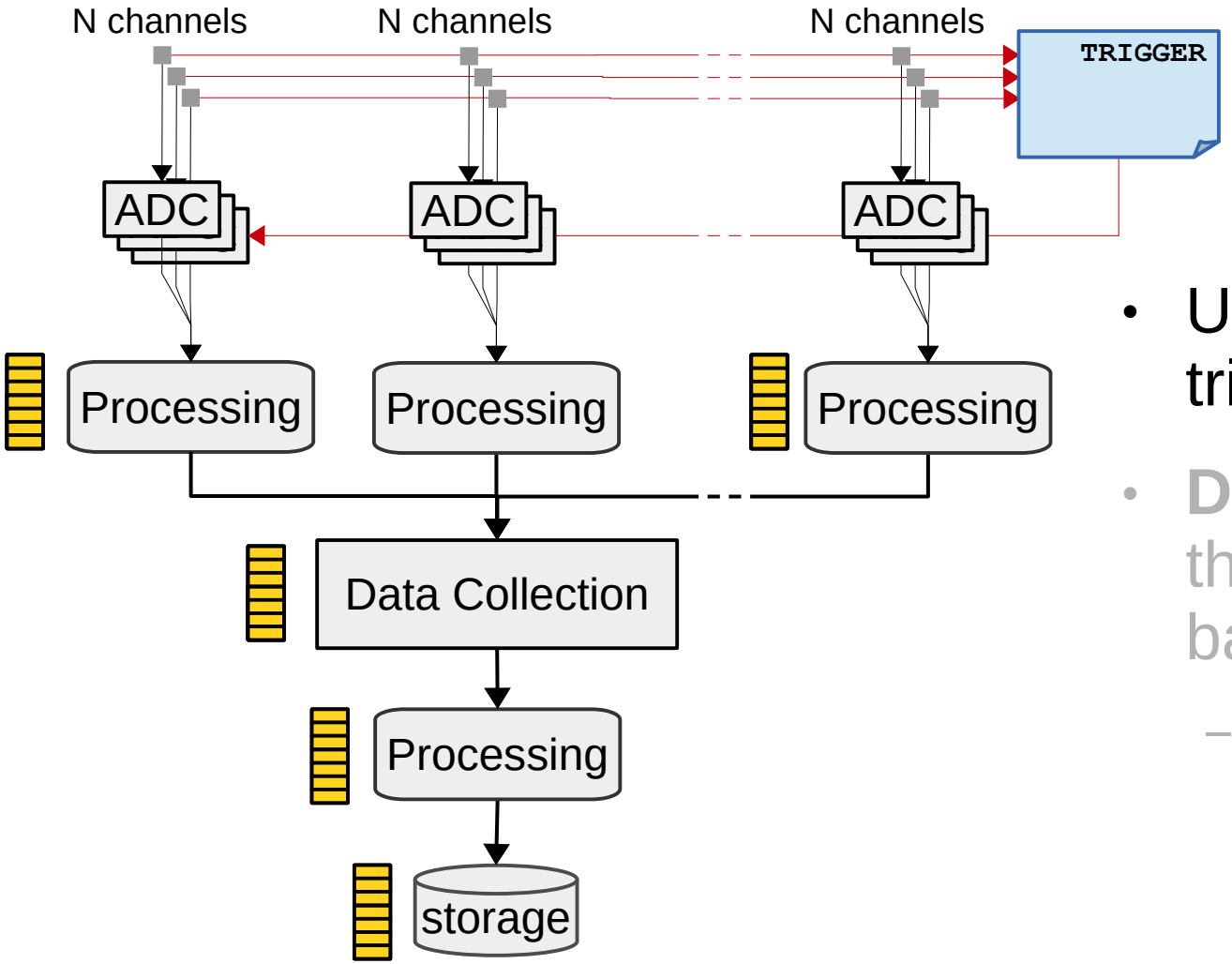
Backpressure

- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)



Backpressure

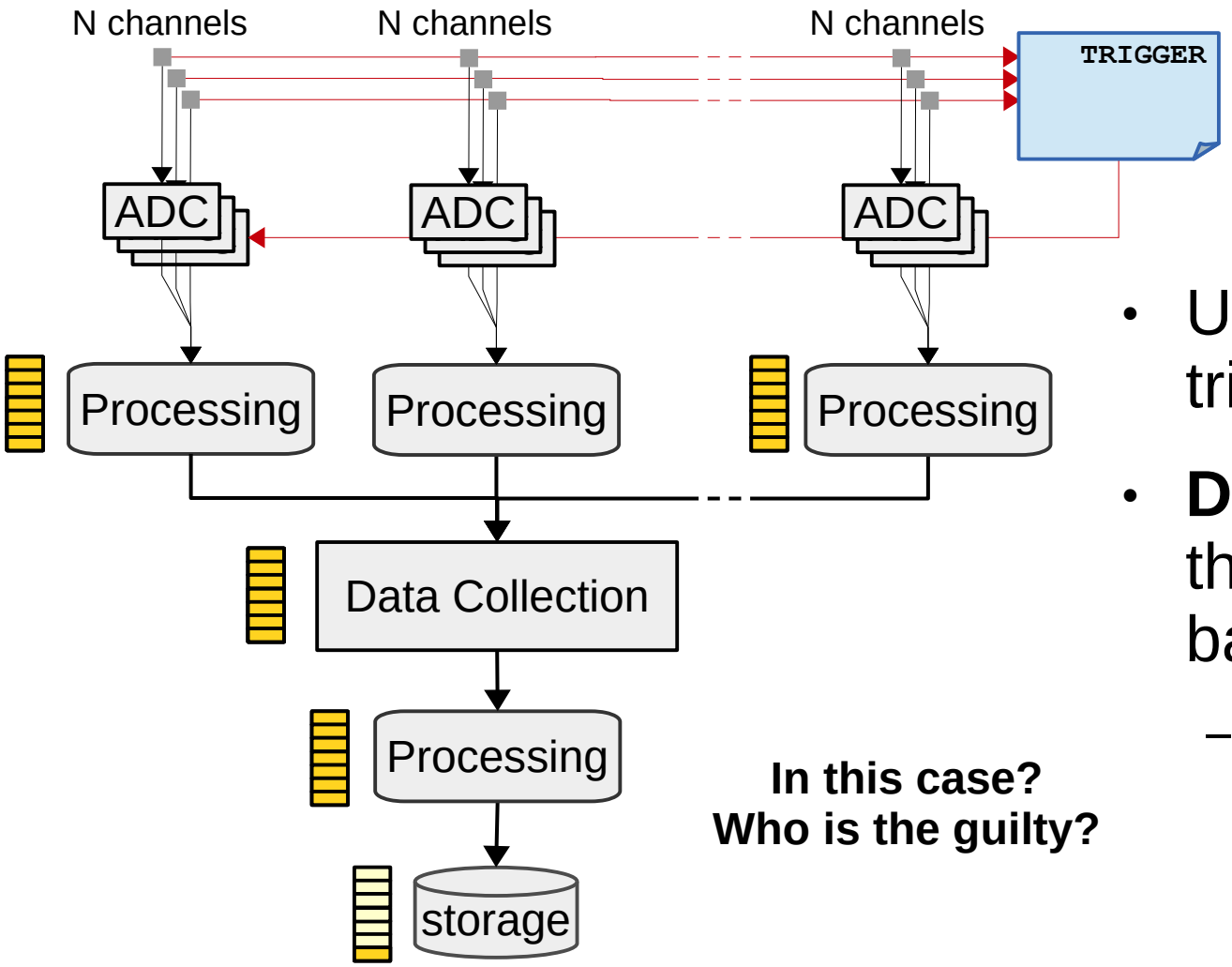
- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)



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

Backpressure

- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)

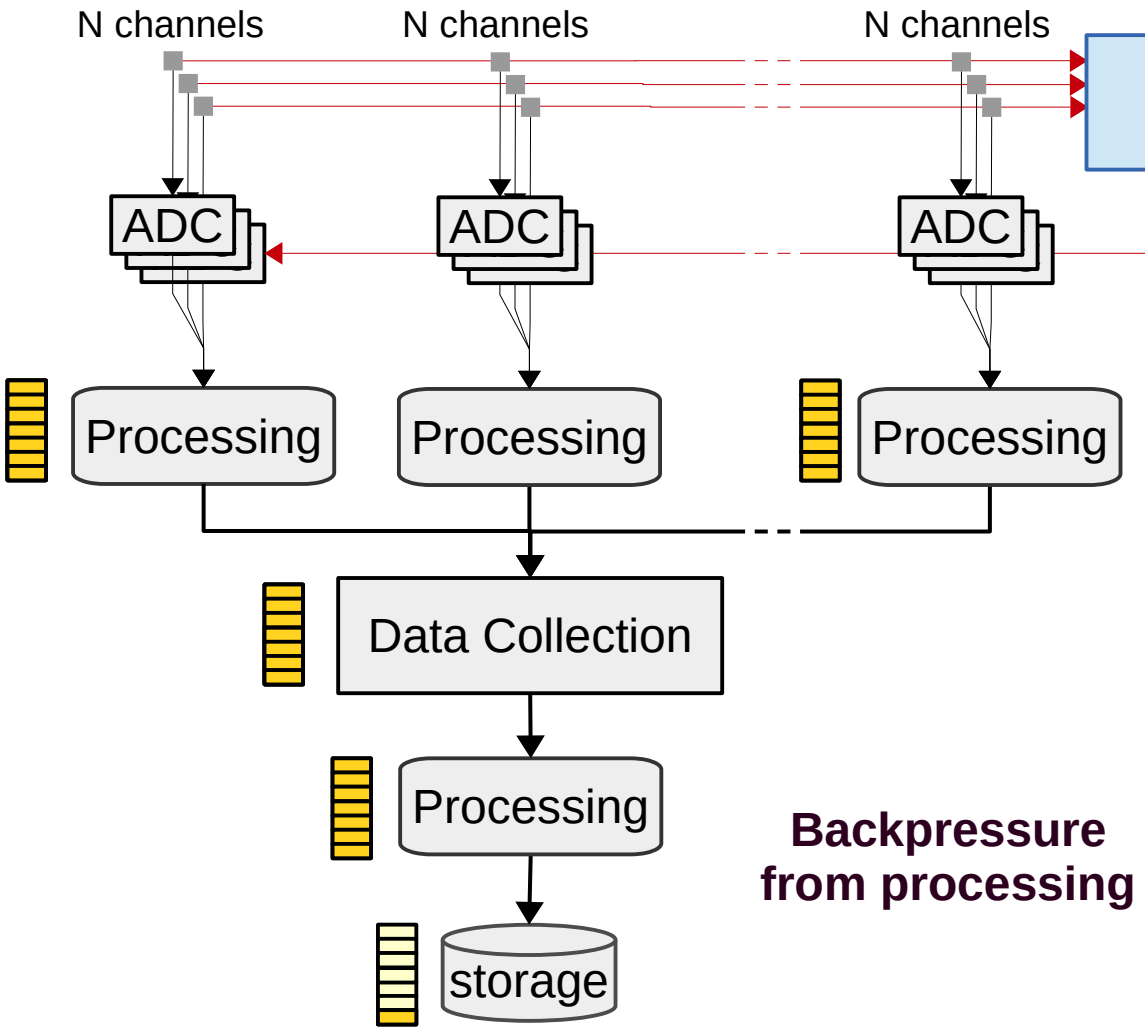


In this case?
Who is the guilty?

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

Backpressure

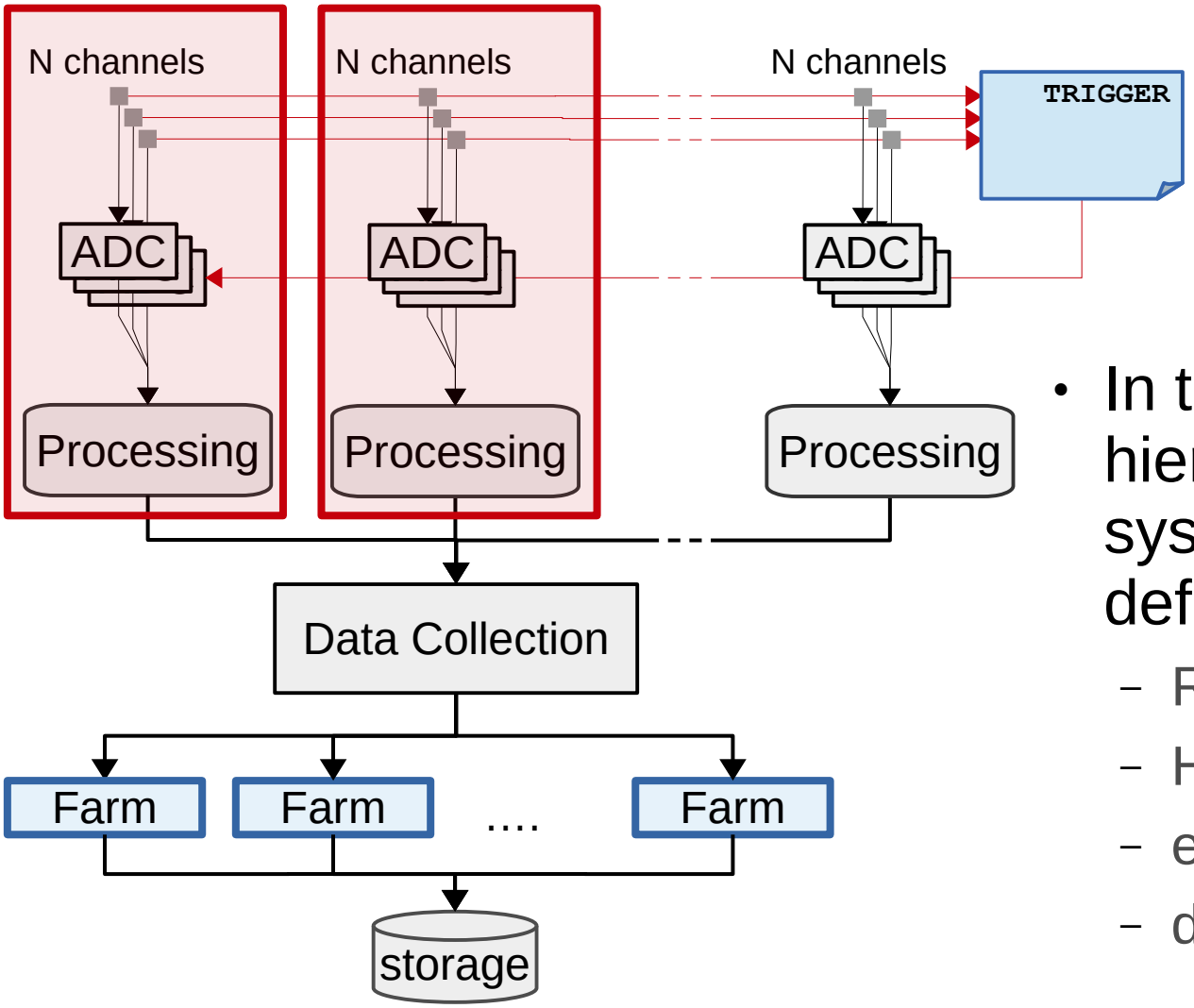
- If a system/buffer gets saturated
  - the “pressure” is propagated upstream (**back-pressure**)



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

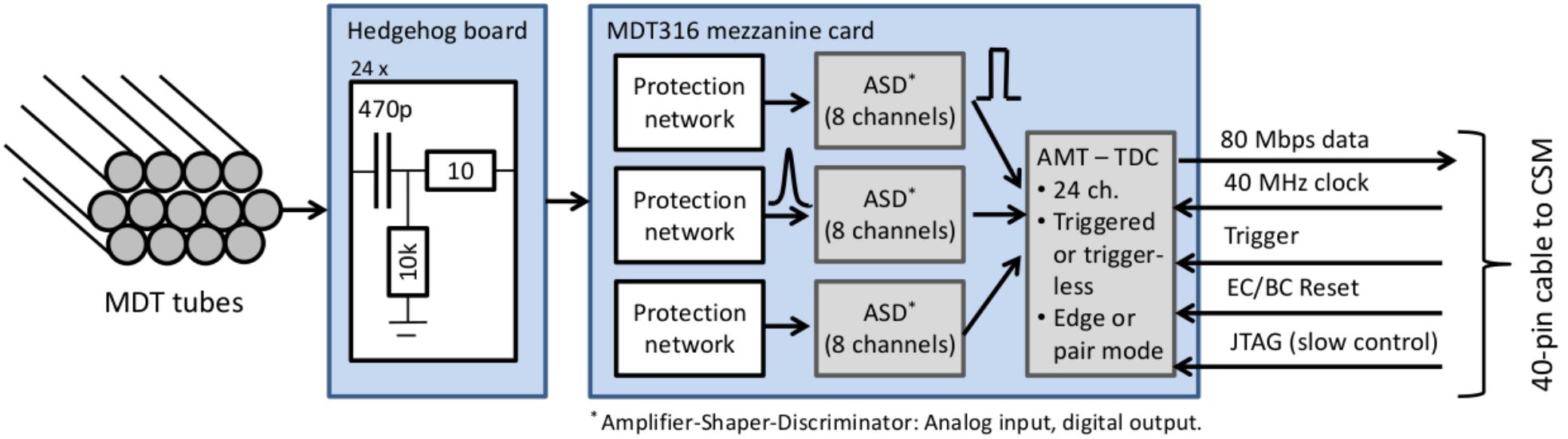
Building blocks

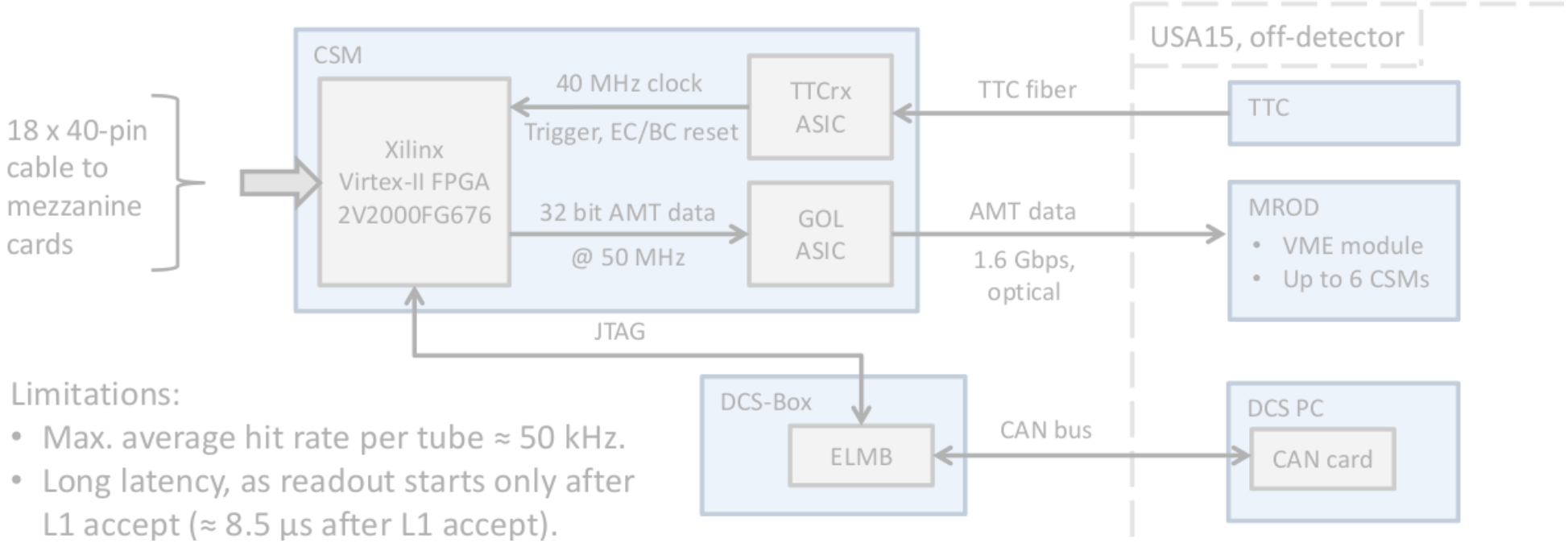
- Reading out data or building events out of many channels requires many components



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

Front End electronics

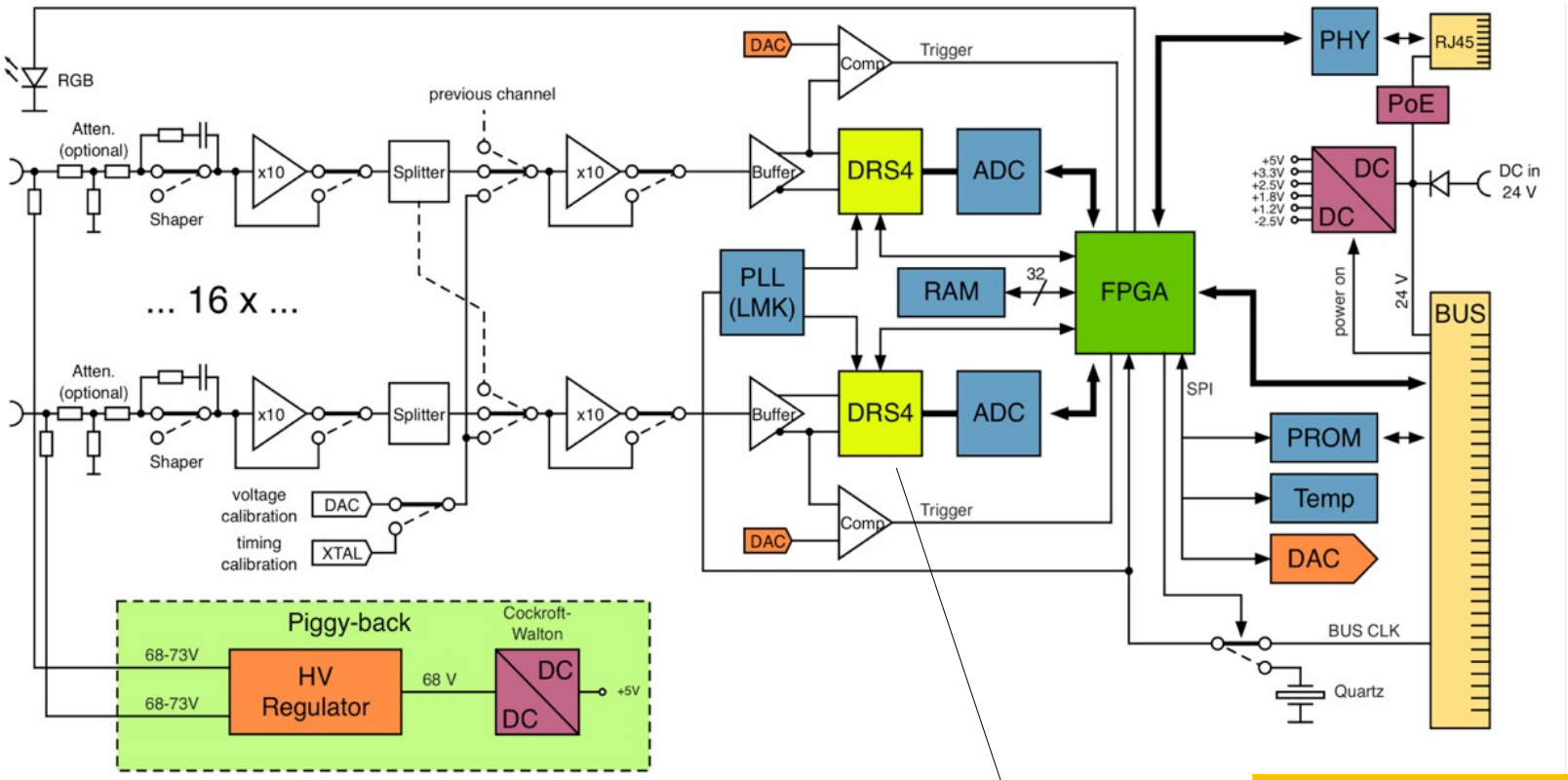




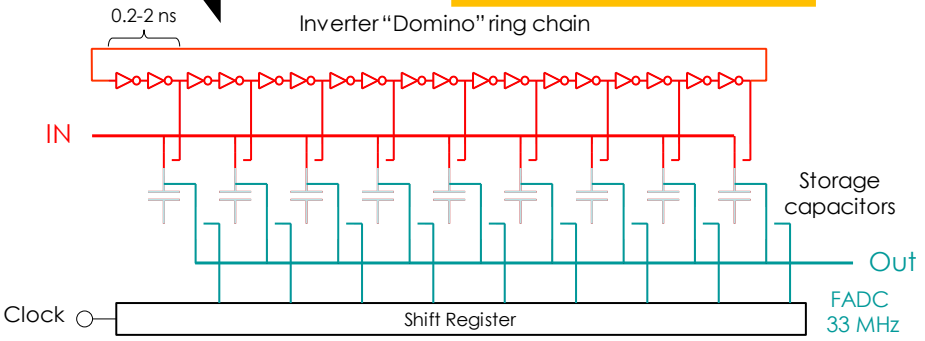
Limitations:

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

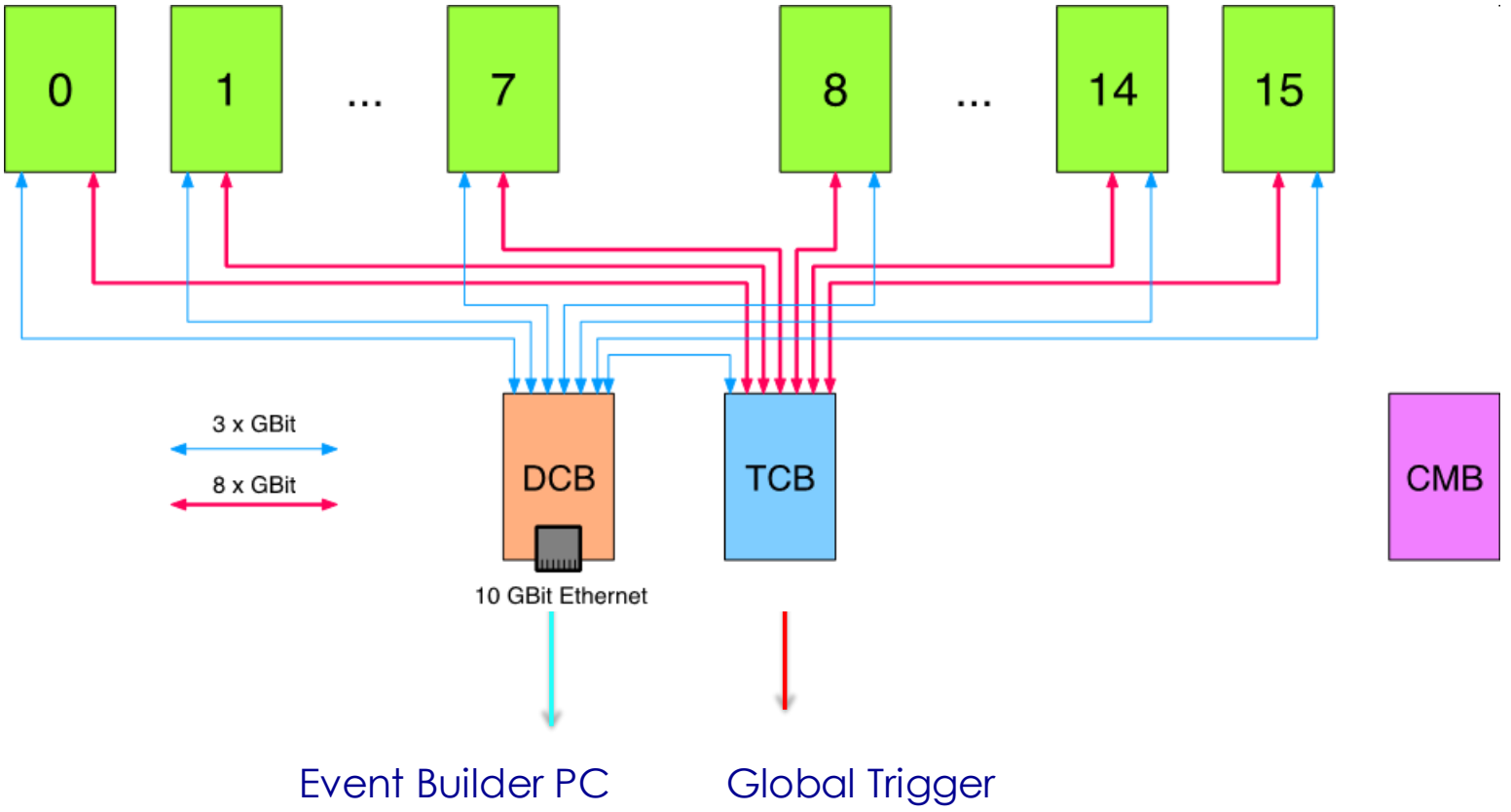
Front-end electronics (WaveDAQ)



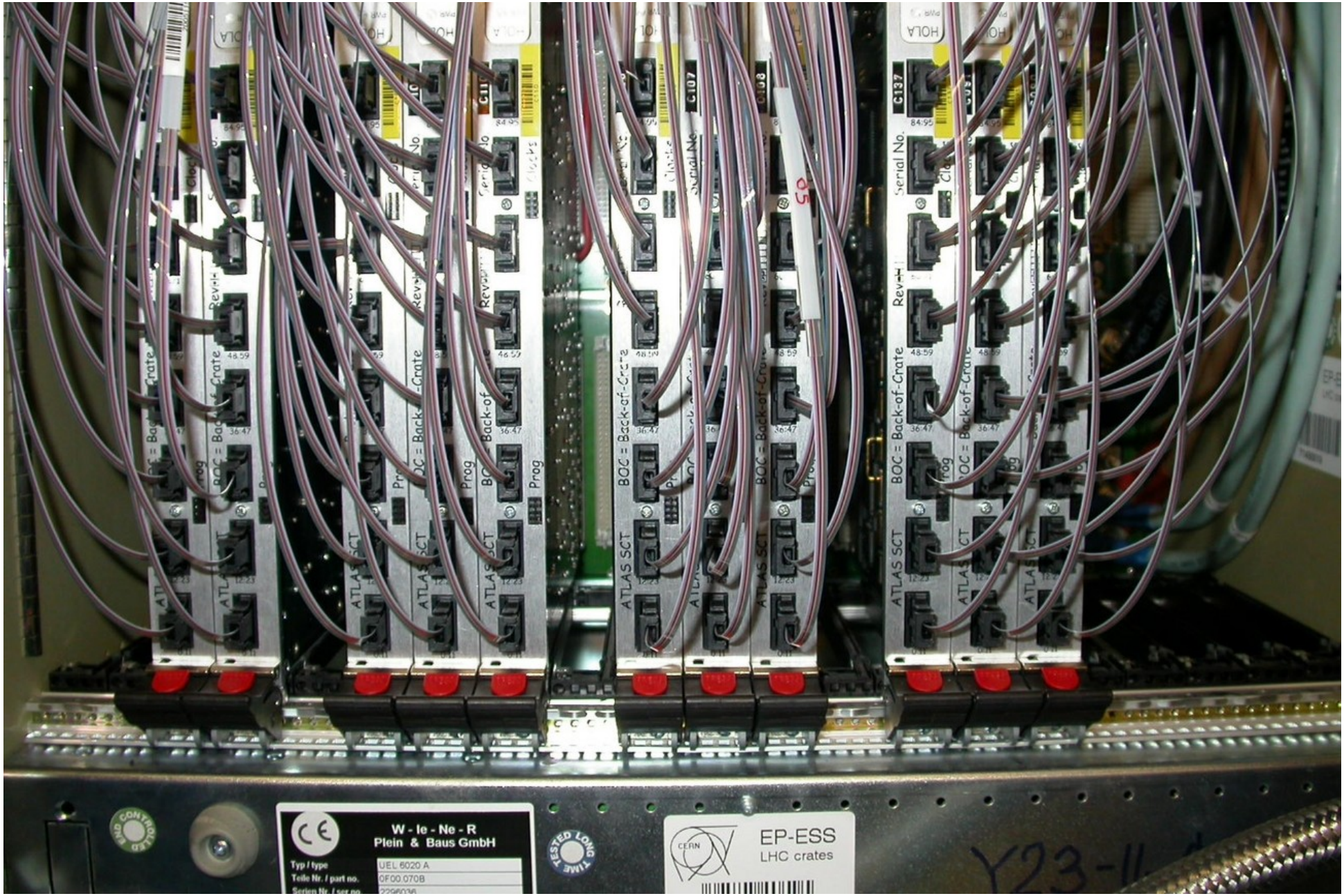
Analog pipeline



DATA and TRIGGER paths

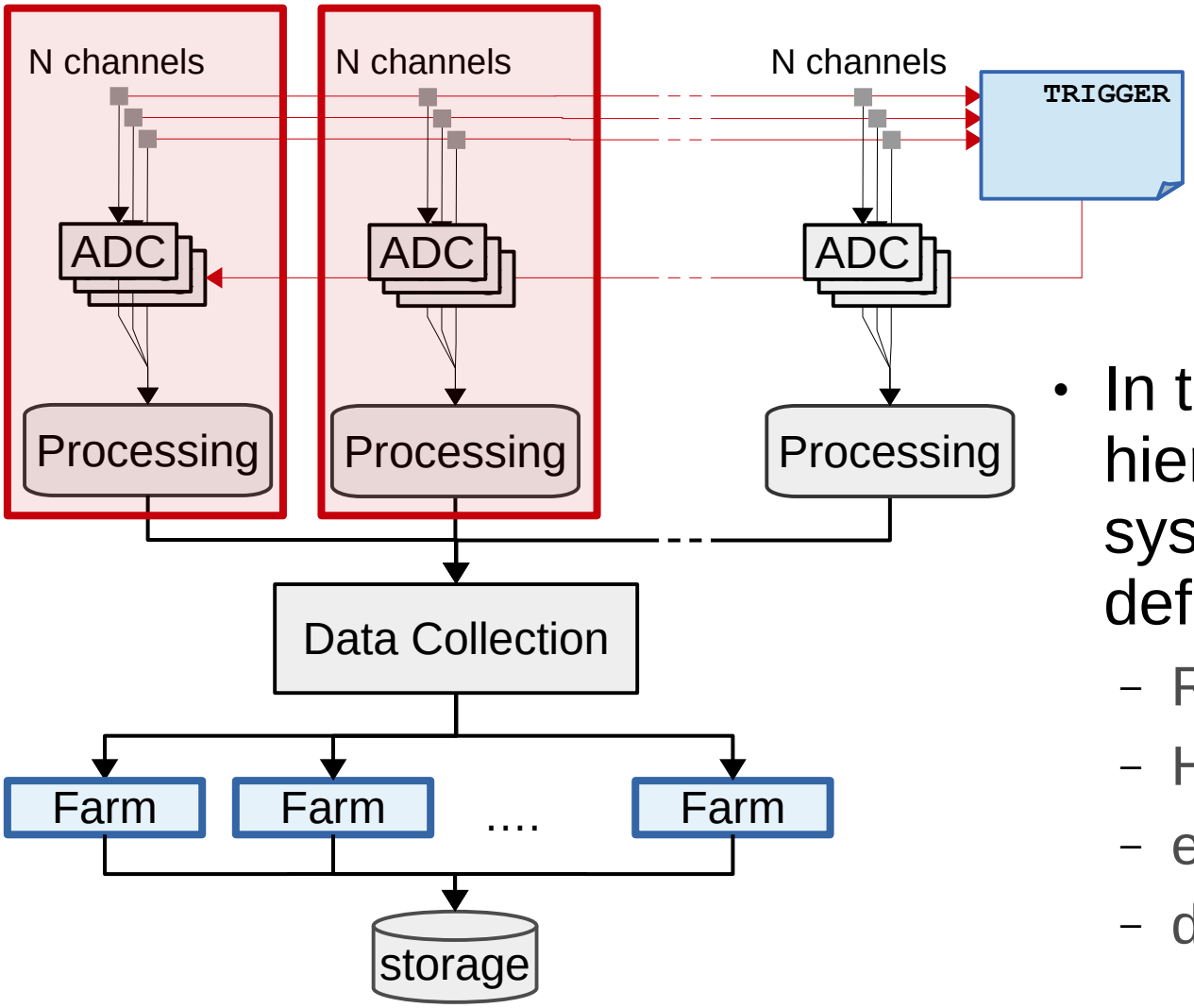


Readout Boards (Counting Room)



Building blocks

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

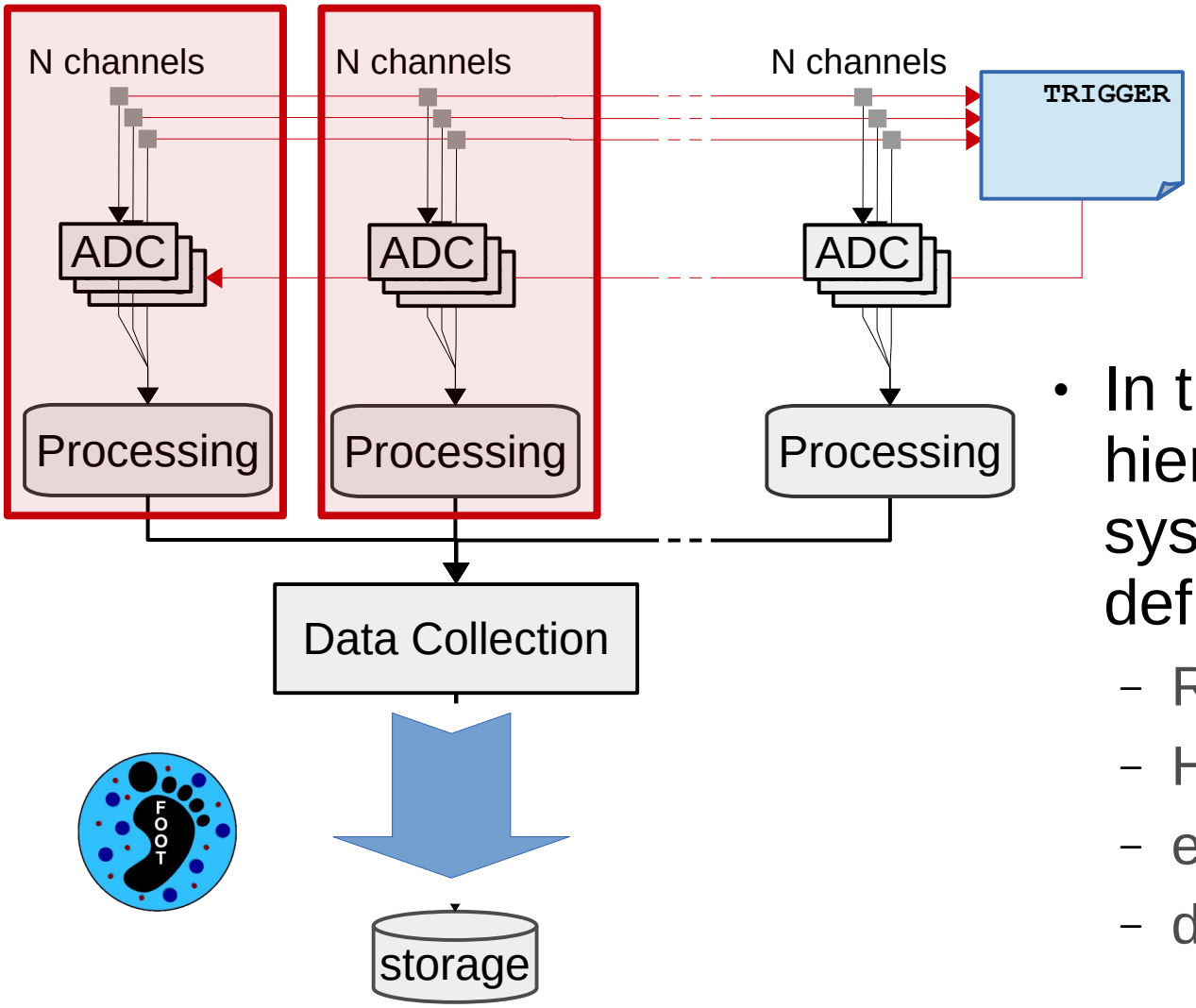


- In the design of our hierarchical data-collection system, we have better define "**building blocks**"
  - Readout crates
  - HLT racks
  - event building groups
  - daq slices

ATLAS Farm (@surface)

Building blocks

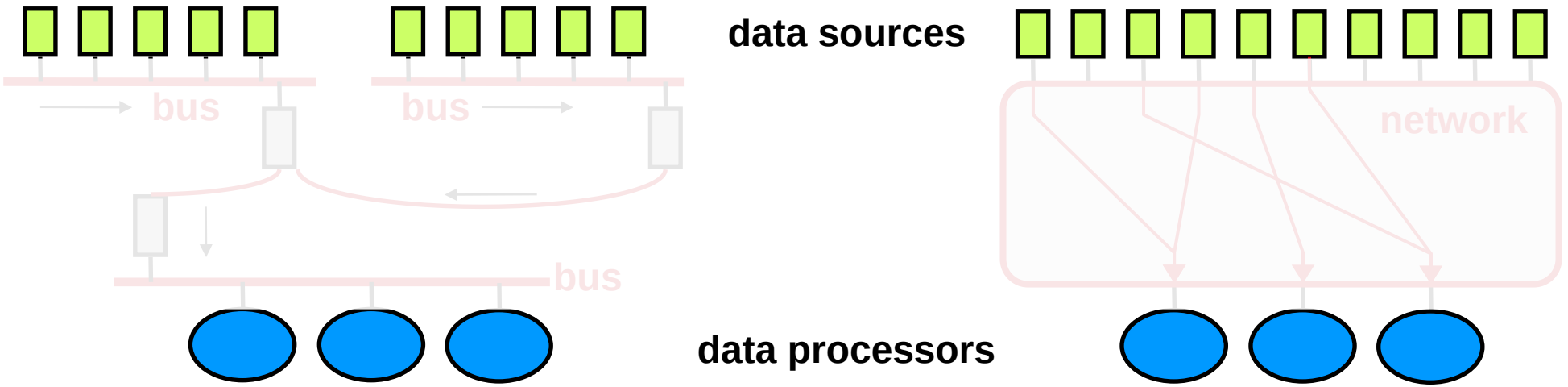
- Reading out data or building events out of many channels requires many components



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

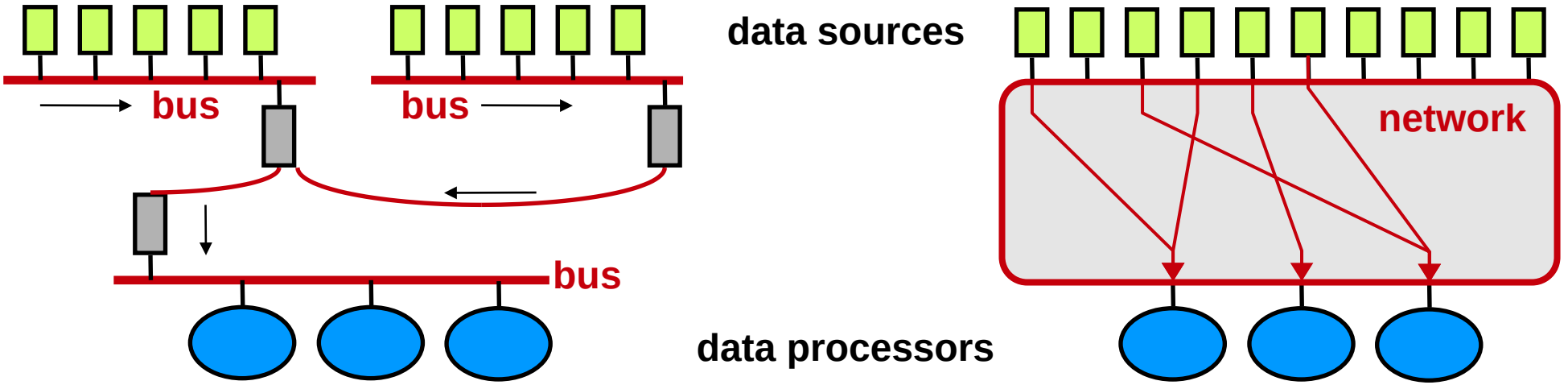
Readout Topology

- How to organize the interconnections inside the building blocks and between building blocks?
  - How to connect data sources and data destinations?
  - Two main classes: **bus** or **network**



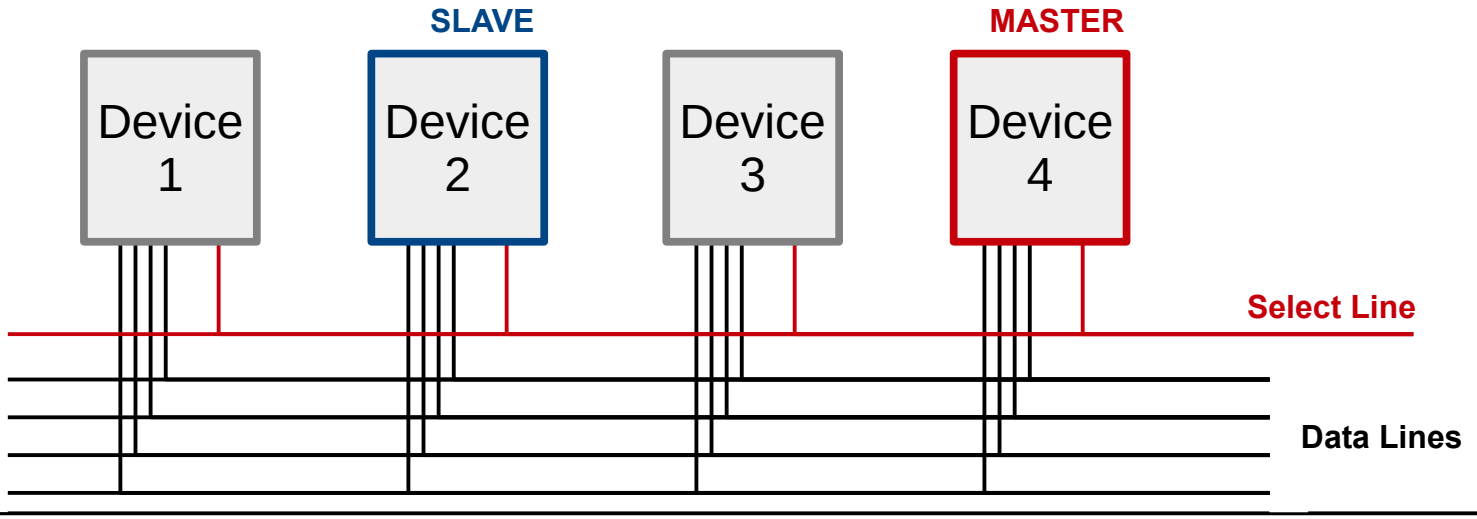
Readout Topology

- How to organize the interconnections inside the building blocks and between building blocks?
  - How to connect data sources and data destinations?
  - Two main classes: **bus** or **network**

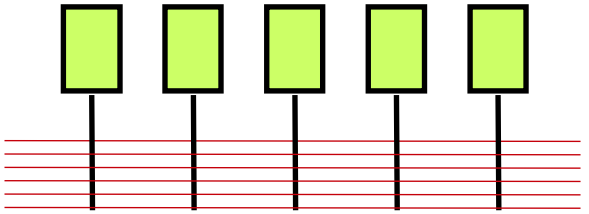


Buses

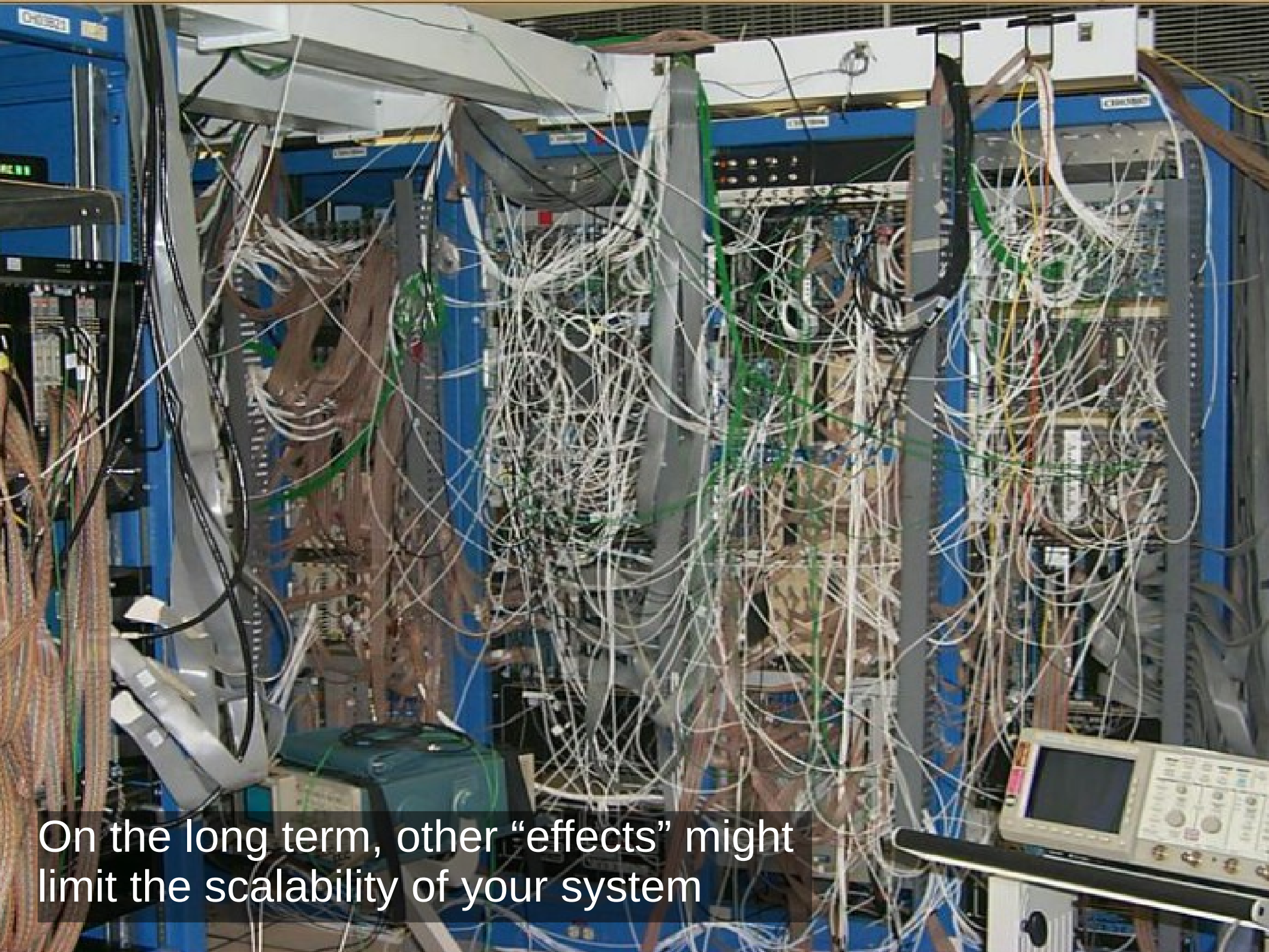
- Devices connected via a **shared bus**
  - Bus → group of electrical lines
- Sharing implies **arbitration**
  - Devices can be **master** or **slave**
  - Devices can be addresses (uniquely identified) on the bus
- E.g.: SCSI, Parallel ATA, VME, PCI …
  - local, external, crate, long distance, ...



Bus facts

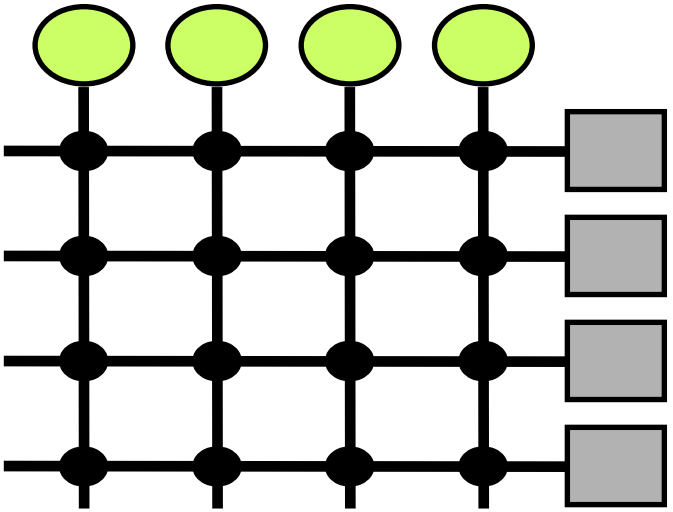


- Simple :-)
  - Fixed number of lines (bus-width)
  - Devices have to follow well defined interfaces
    - Mechanical, electrical, communication, ...
- **Scalability** issues :-(
  - Bus bandwidth is shared among all the devices
  - Maximum bus width is limited
  - Maximum number of devices depends 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

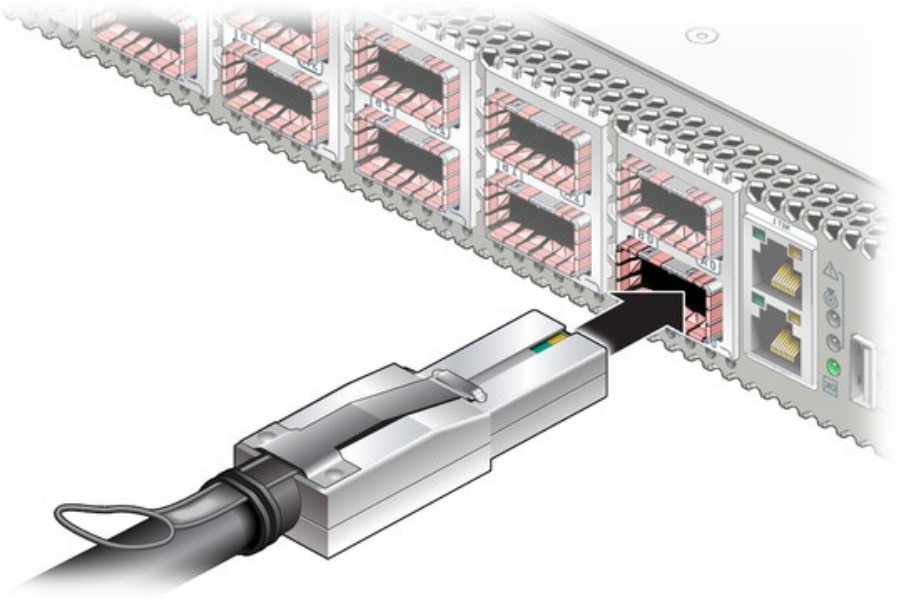

On the long term, other “effects” might limit the scalability of your system

Network

- All devices are **equal**
  - They communicate directly with each other via messages
  - No arbitration, simultaneous communications

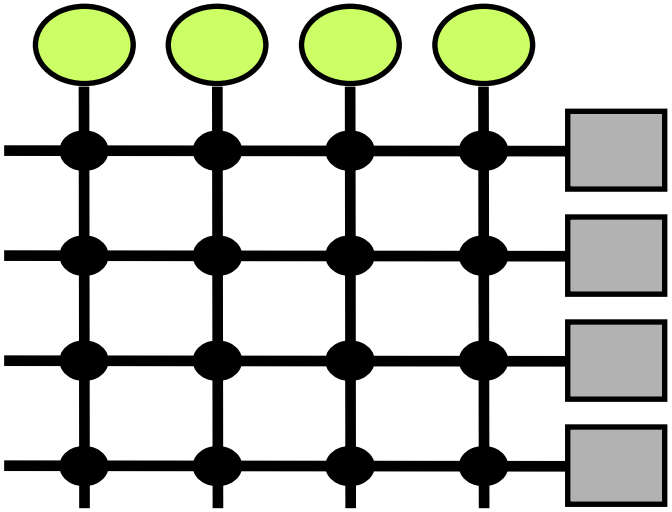


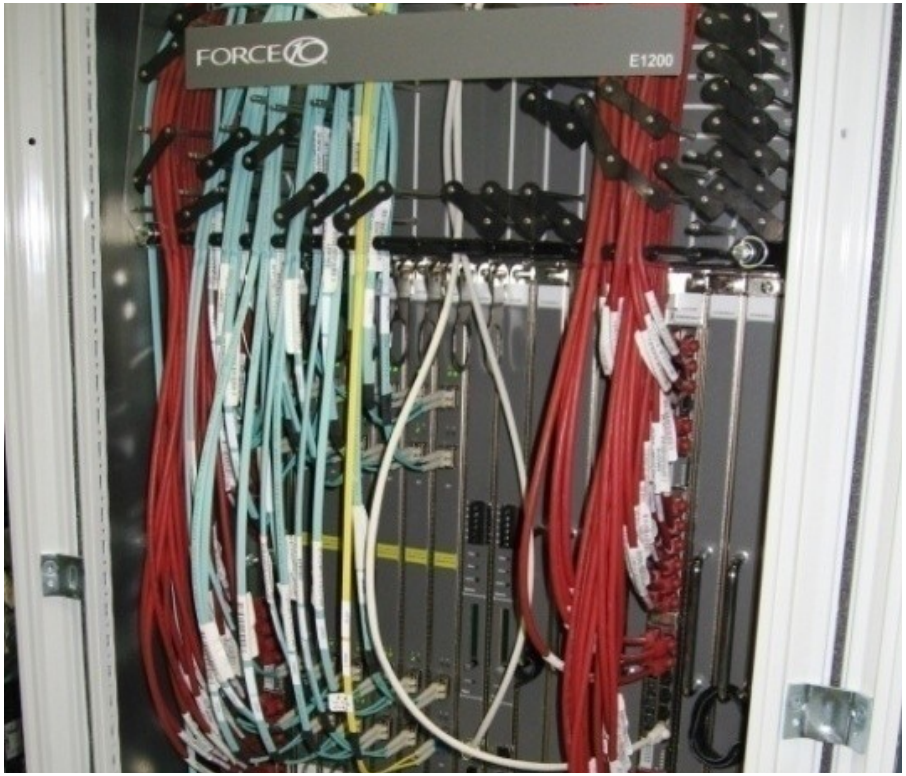
- Eg: Telephone, Ethernet, Infiniband, …



Network

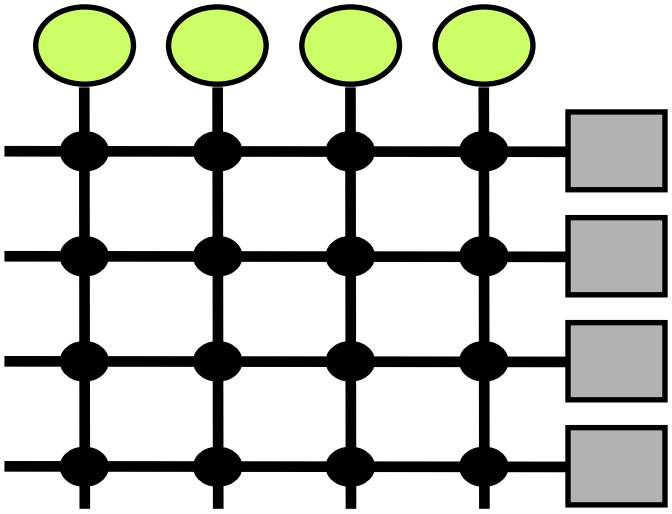
- In switched networks, **switches** move messages between sources and destinations
  - Find the right path
- How **congestions** (two messages with the same destination at the same time) are handled?
  - The key is ...

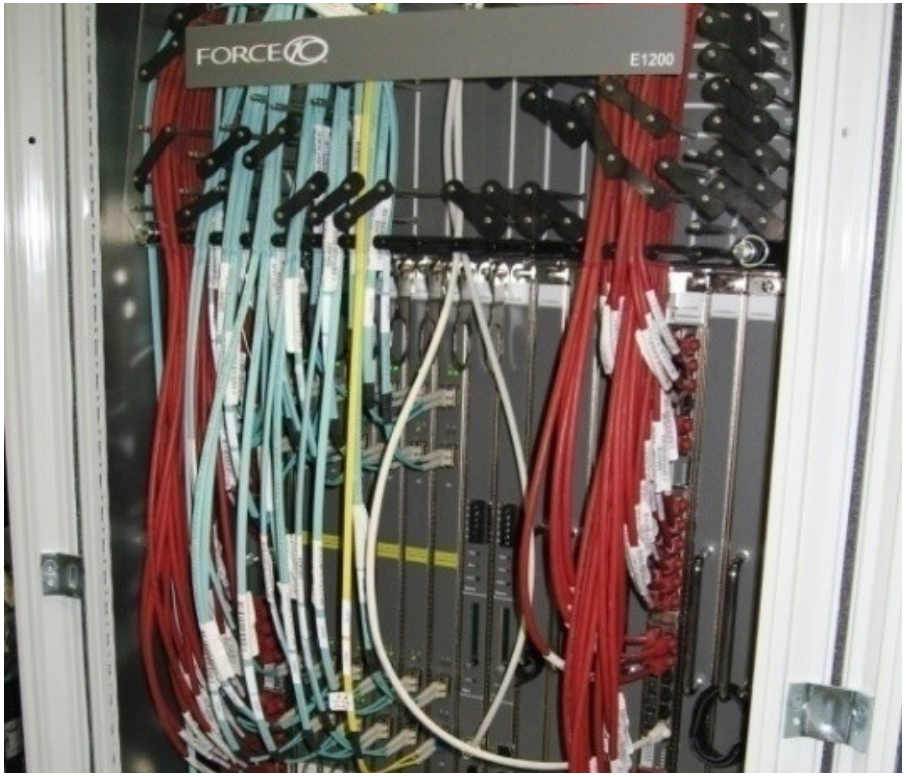




Network

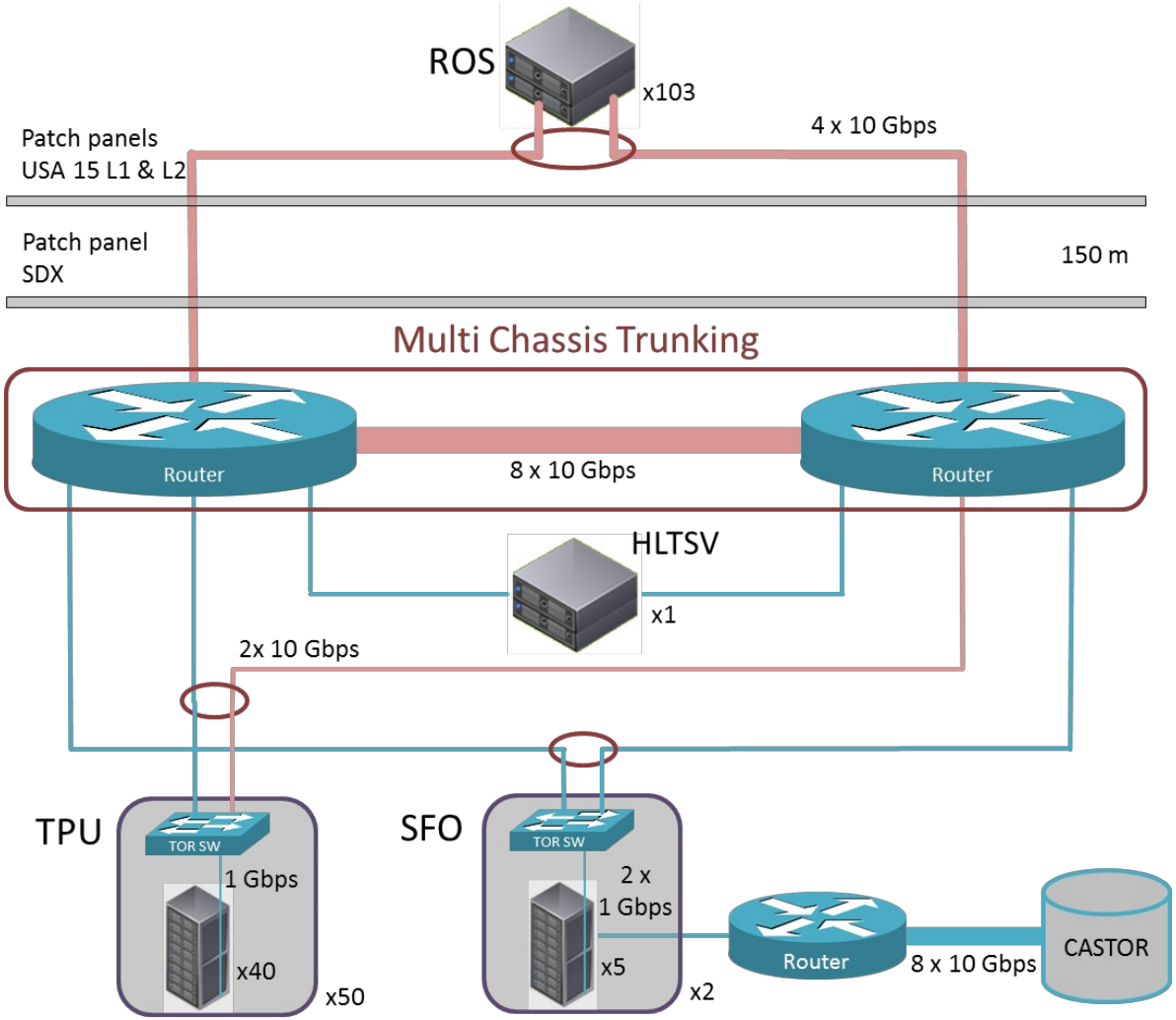
- In switched networks, **switches** move messages between sources and destinations
  - Find the right path
- How **congestions** (two messages with the same destination at the same time) are handled?
  - The key is .... **buffering**





Network

- Networks scale well (and allow redundancy)
  - They are the backbones of LHC DAQ systems
-



DAQ concepts

DAQ Mentoring

- Study the **trigger** properties
  - Periodic or stochastic, continuous or bunched
- Consider the needed **efficiency**
  - Good to keep operation margins, but avoid over-sizing
- Identify **fluctuation** sources and size adequate buffering mechanisms
  - NB: there are many source of fluctuations: multi-threaded sw, network, ...
- Adequate **buffer** is not a huge buffer
  - Makes your system less stable and responsive, prone to oscillations
  - Overall it decreases reliability

DAQ Mentoring

- **Keep it simple**: keep under control the number of free parameters without losing flexibility
  - Have you ever heard about SUSY phase-space scans? Do you really want something like that for your DAQ system?
- Problems require **perseverance**
  - Be careful, a rare little glitch in your DAQ might be the symptom of a major issue with your data
- In any case, …

DAQ Mentoring

- **Keep it simple**: keep under control the number of free parameters without losing flexibility
  - Have you ever heard about SUSY phase-space scans? Do you really want something like that for your DAQ system?
- Problems require **perseverance**
  - Be careful, a rare little glitch in your DAQ might be the symptom of a major issue with your data
- In any case, … DON'T PANIC

and **enjoy** the DAQ