



# Electronics, Trigger and Data Acquisition

Summer Student Programme 2015, CERN

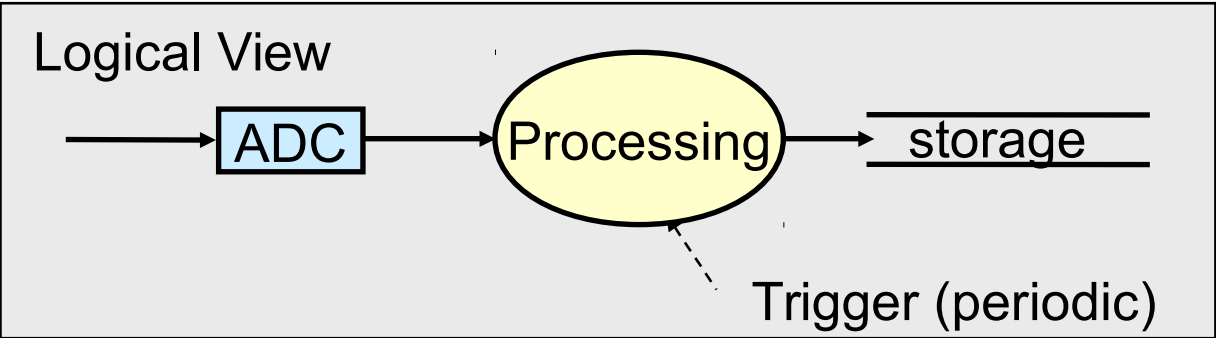
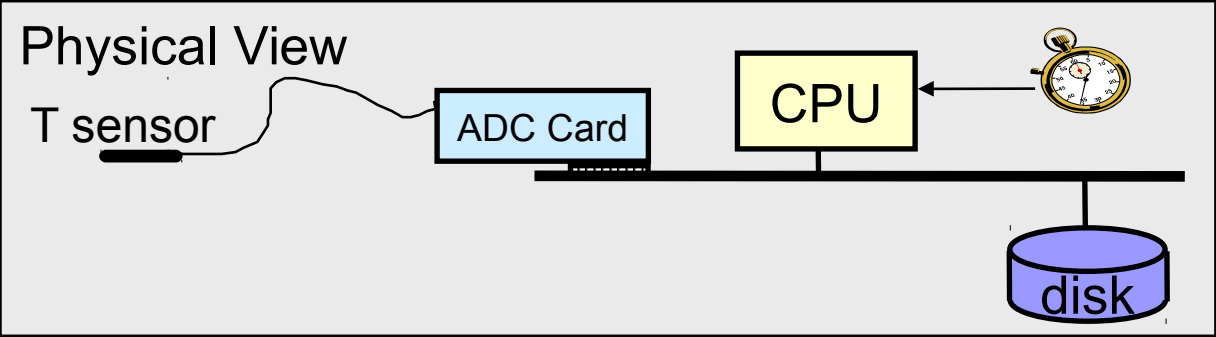
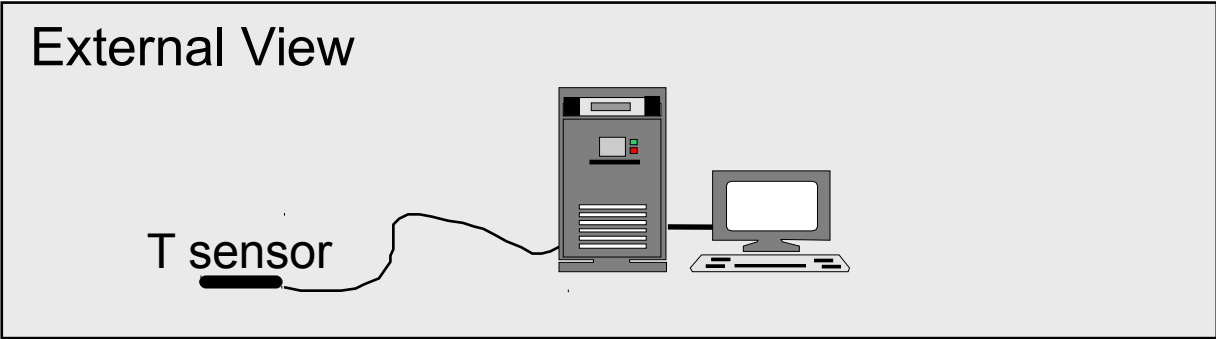
Part 2

W.Vandelli CERN/PH-ADT  
Wainer.Vandelli@cern.ch



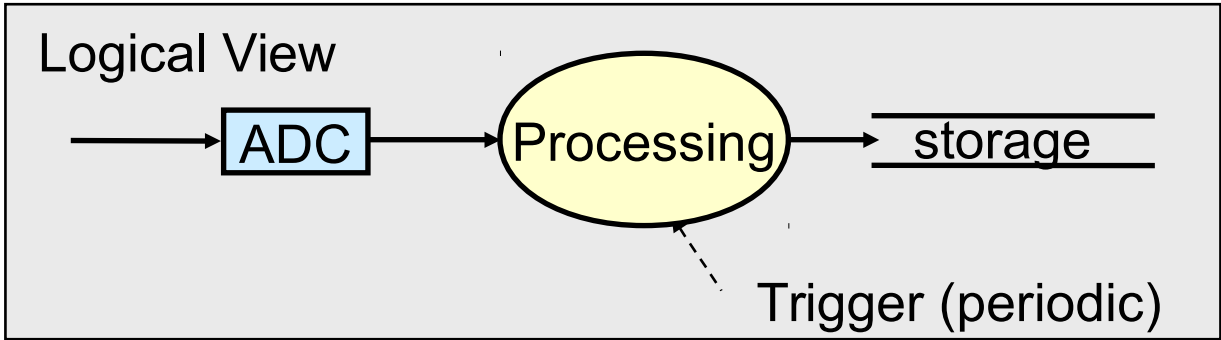
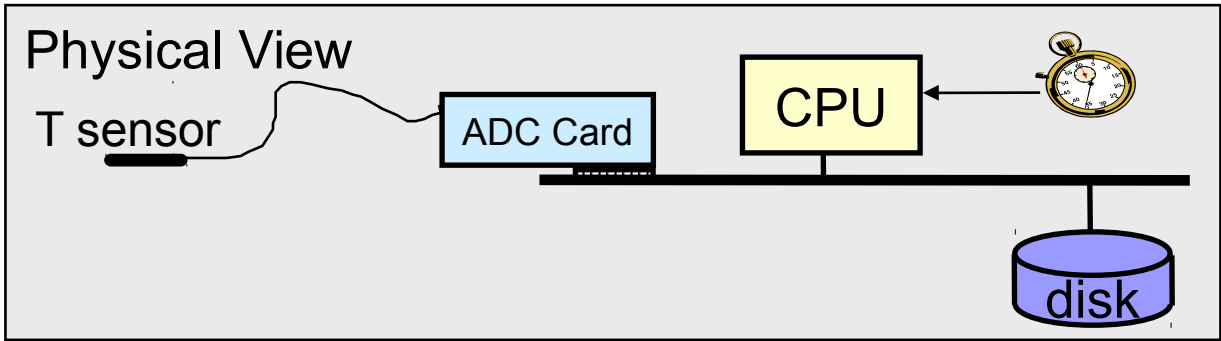
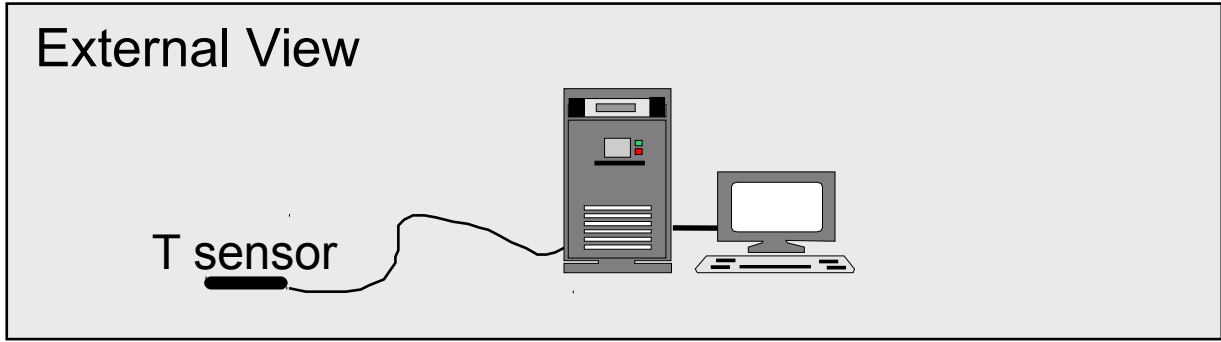
# DAQ & Trigger

# Basic DAQ: periodic trigger



- Measure temperature at a fixed frequency
- ADC performs analog to digital conversion
  - our front-end electronics
- CPU does readout and processing

# Basic DAQ: periodic trigger



→ Measure temperature at a fixed frequency

→ The system is clearly limited by the time to process an “event”

→ Example  $\tau=1\text{ms}$  to

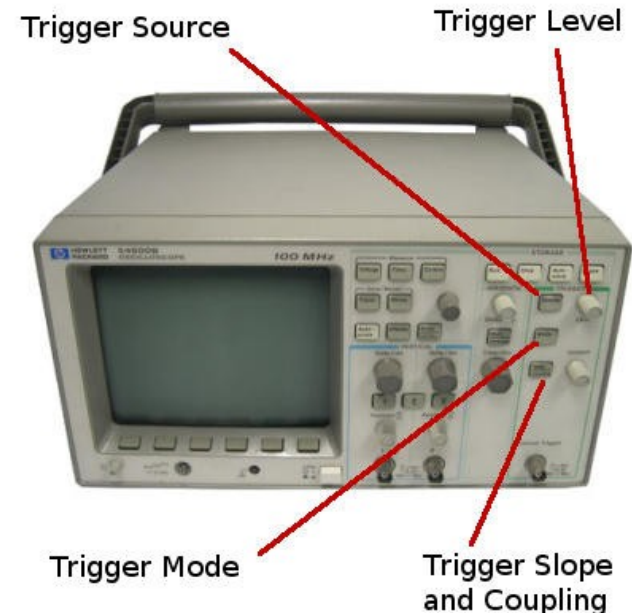
- ADC conversion
- +CPU processing
- +Storage

→ Sustain  $\sim 1/1\text{ms}=1\text{kHz}$  **periodic trigger** rate

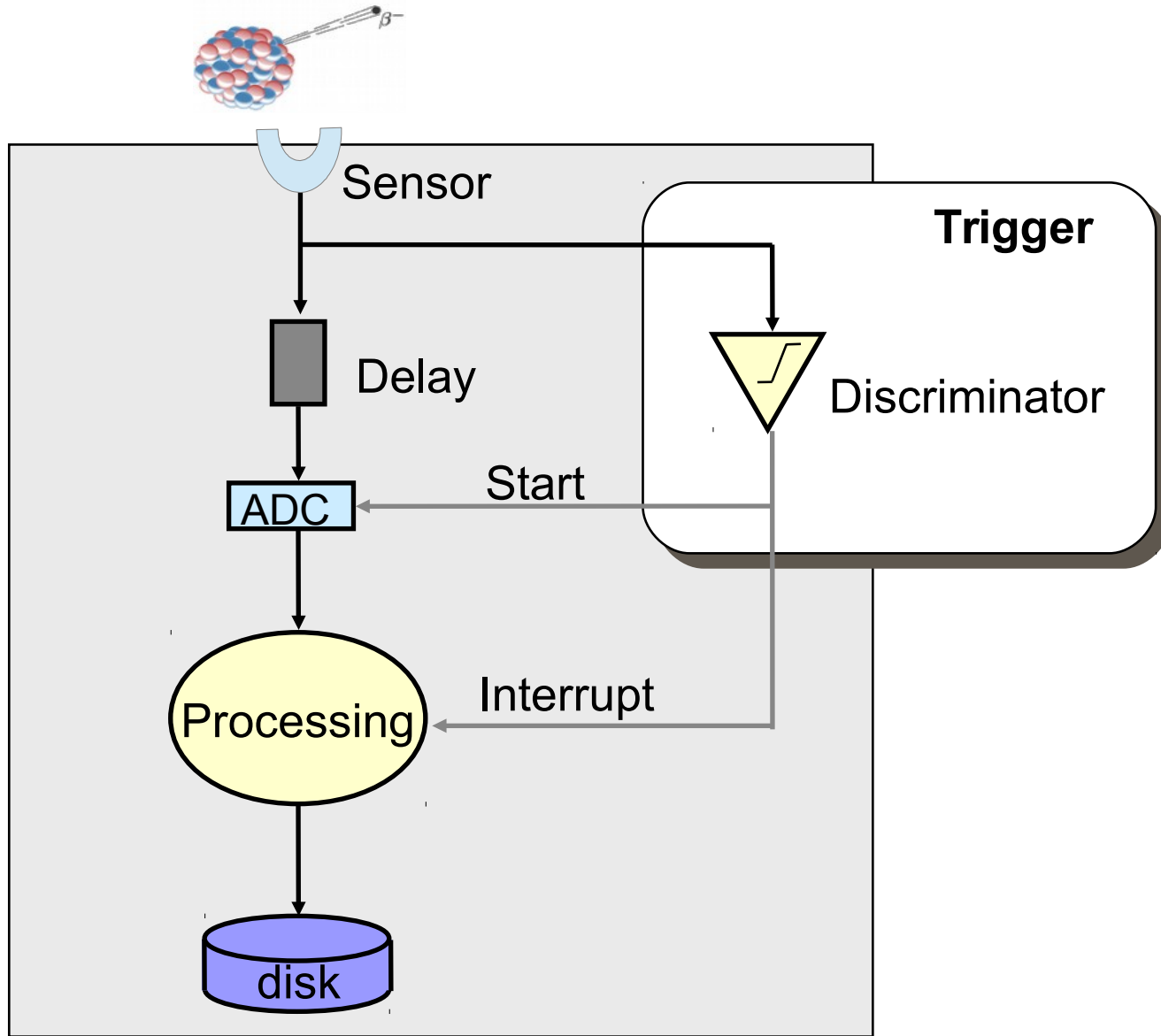
# What is a “trigger”?

- A “trigger” is a system that rapidly decides, based on “simple” criteria, if an interesting event took place, initiating the data-acquisition process
- Simple, rapid, selective are the trigger keywords
- Relative parameters that depend on the operating conditions
  - in a multi-level trigger system the last level is normally way slower and more complex than the first one

The oscilloscope trigger does exactly this. Informs the instrument to initiate the internal signal acquisition and visualization

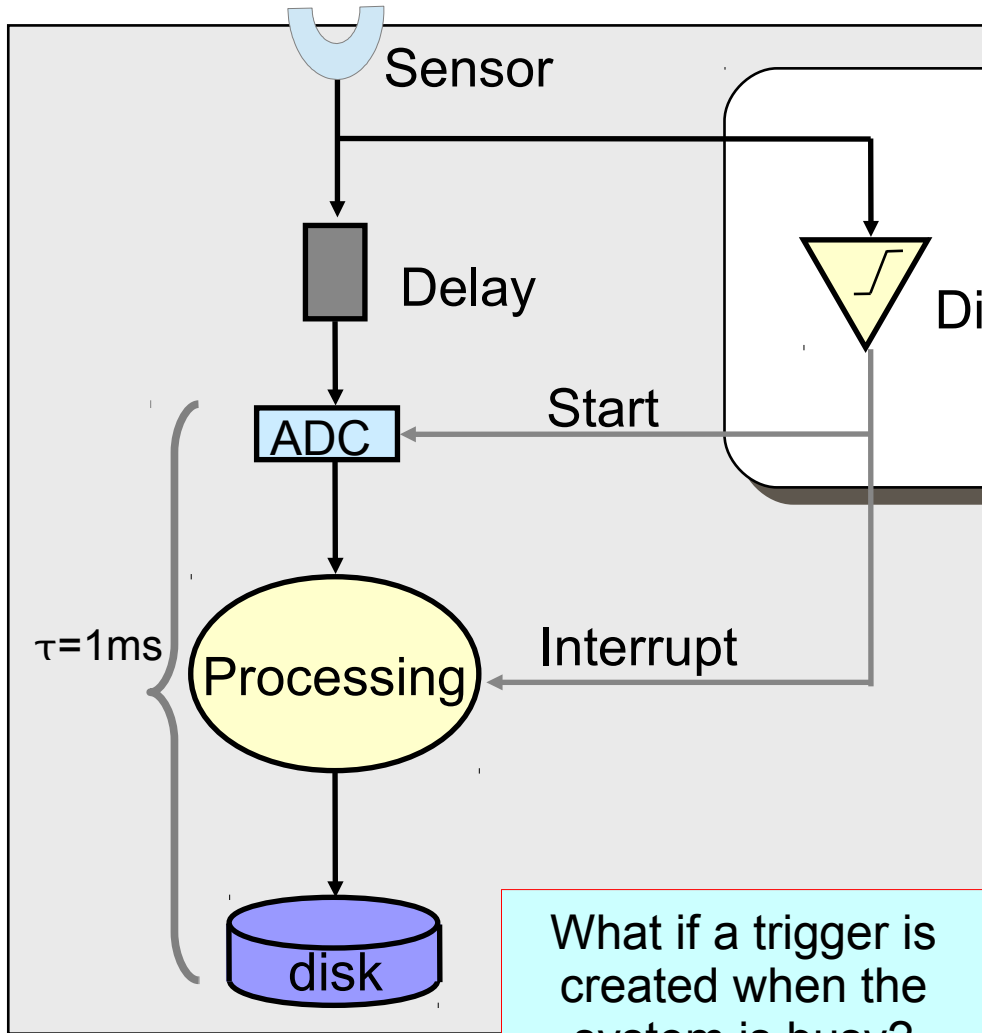
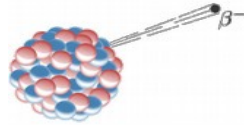


# Basic DAQ: physics trigger



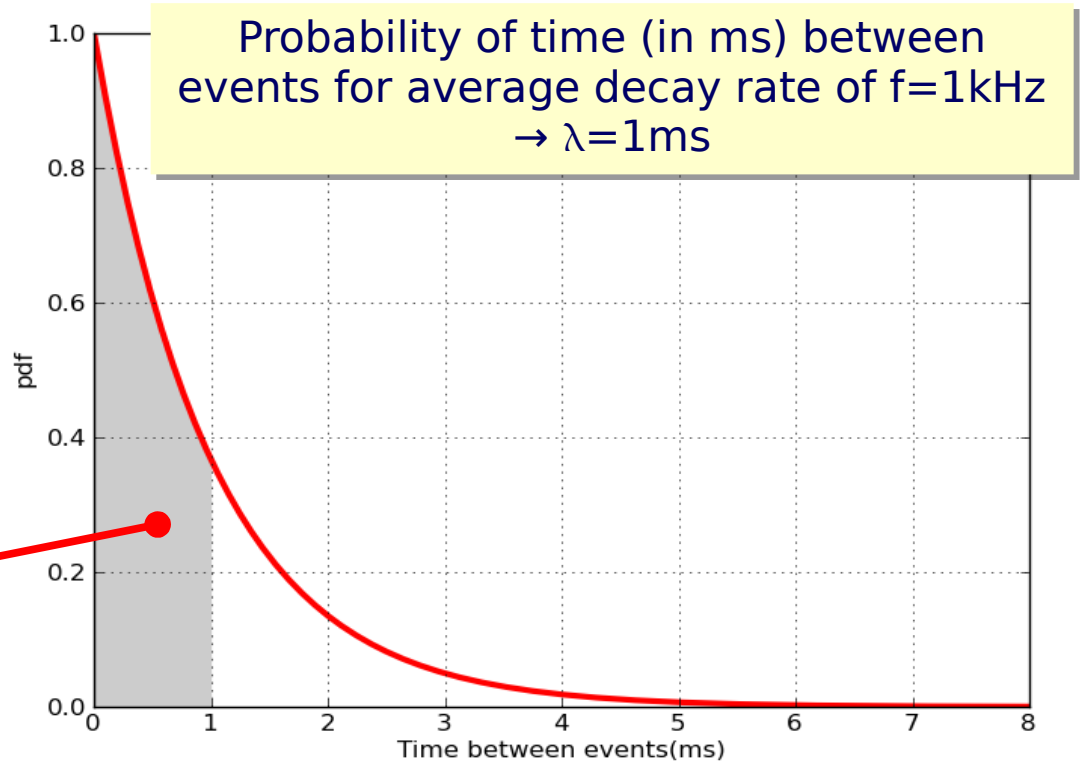
- Measure  $\beta$  decay properties
- Events are asynchronous and unpredictable
  - need a **physics** trigger
- Delay compensates for the **trigger latency**
  - time needed to reach a decision

# Basic DAQ: real trigger



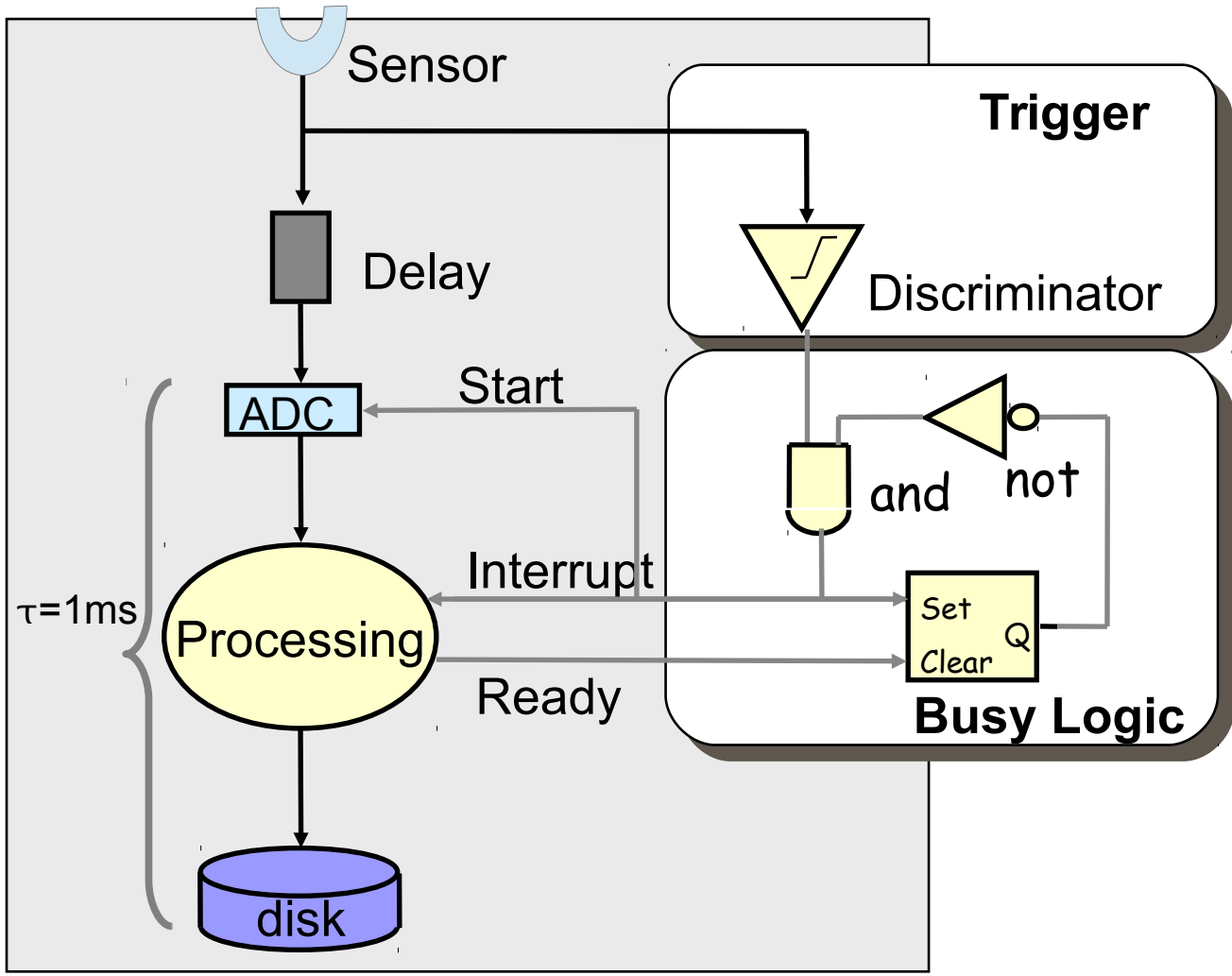
What if a trigger is created when the system is busy?

- ➔ Measure  $\beta$  decay properties
  - need a **physics** trigger
- ➔ Stochastic process
  - fluctuations



# Basic DAQ: real trigger & busy logic

$f=1\text{kHz}$   
 $1/f=\lambda=1\text{ms}$



- ➔ Busy logic avoids triggers while processing
- ➔ Which (average) DAQ rate can we achieve now?
  - reminder:  $\tau=1\text{ms}$  was sufficient to run at 1kHz with a clock trigger





# DAQ Deadtime & Efficiency (1)

Define  $\nu$  as average DAQ frequency

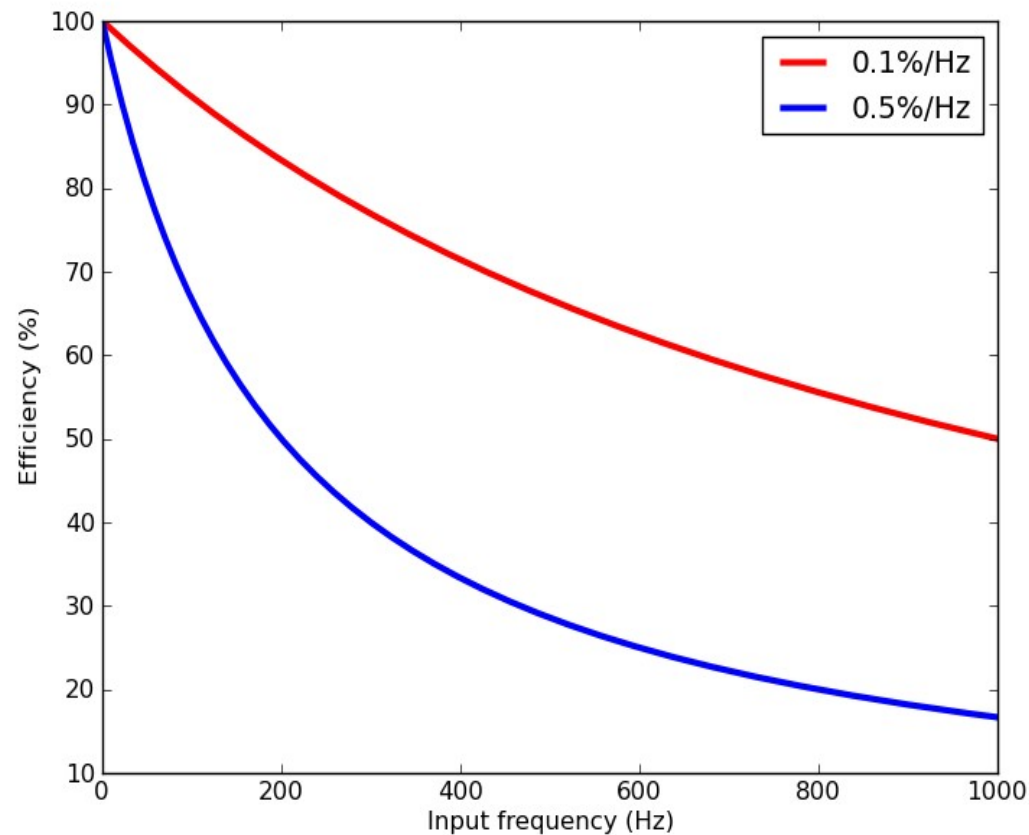
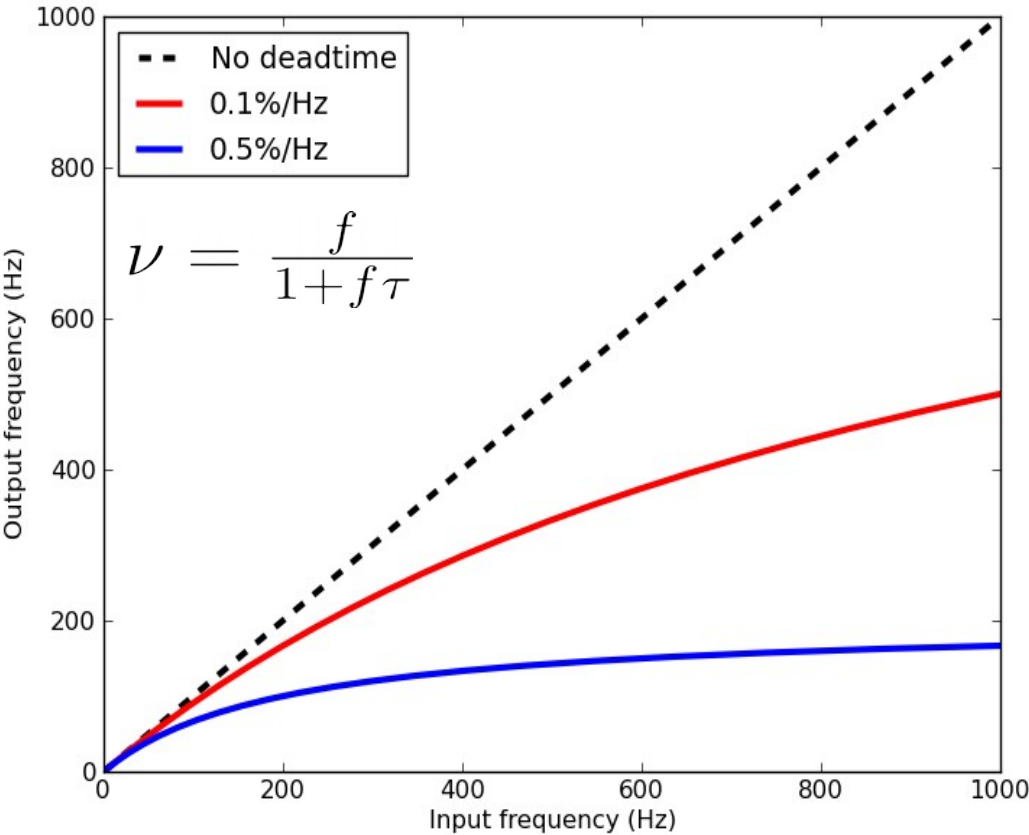
$\nu\tau \rightarrow$  DAQ system is busy -  $(1 - \nu\tau) \rightarrow$  DAQ system is free

$$f(1 - \nu\tau) = \nu \rightarrow \nu = \frac{f}{1 + f\tau} < f$$

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

- Define DAQ deadtime (d) as the time the system requires to process an event, without being able to handle other triggers. In our example  $d=0.1\%/Hz$
- Due to the fluctuations introduced by the stochastic process the efficiency will always be less 100%
  - in our specific example,  $d=0.1\%/Hz$ ,  $f=1kHz \rightarrow \nu=500Hz$ ,  $\epsilon=50\%$

# DAQ Deadtime & Efficiency (2)



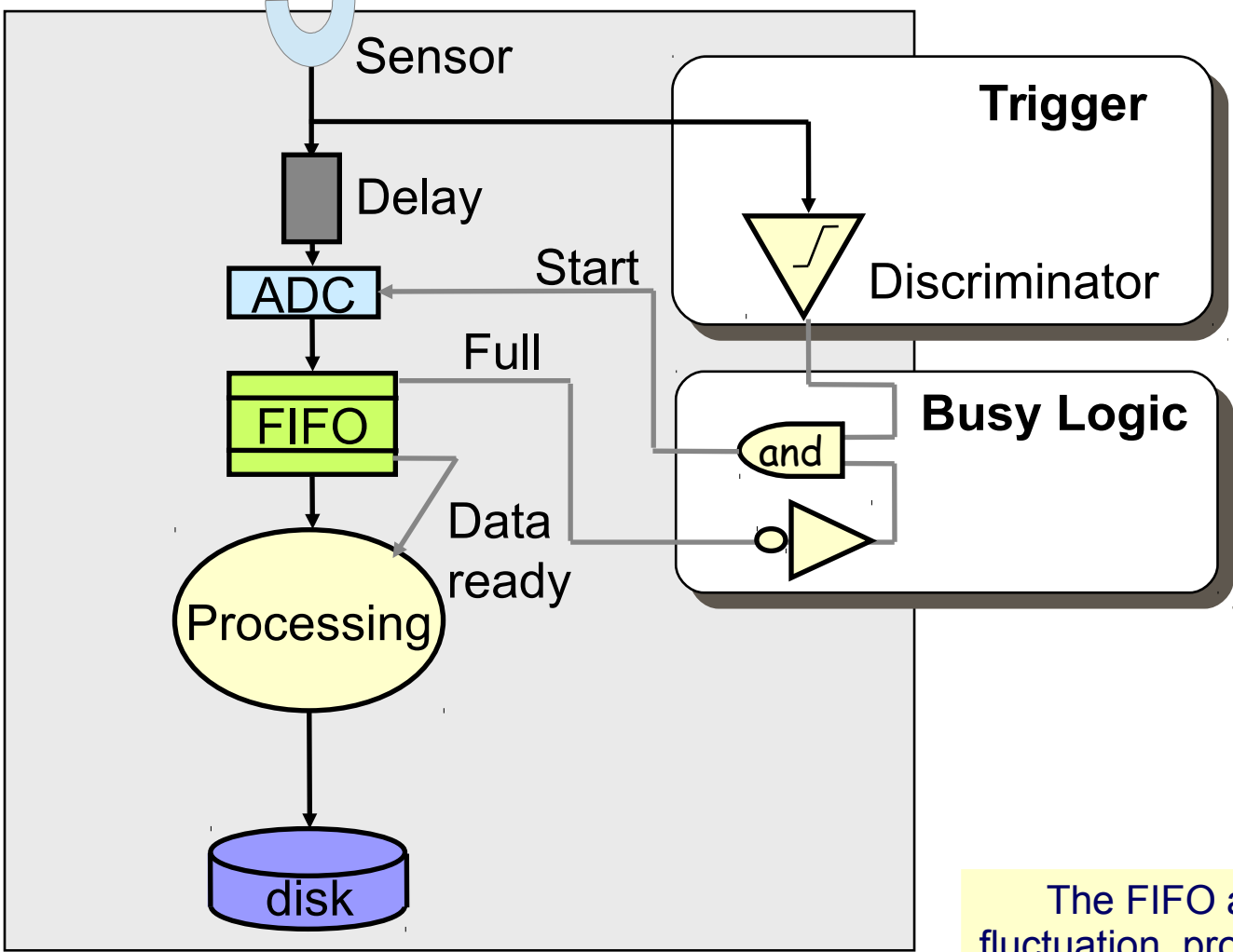
➔ If we want to obtain  $\nu \sim f$  ( $\epsilon \sim 100\%$ )  $\rightarrow f\tau \ll 1 \rightarrow \tau \ll \lambda$

-  $f=1\text{kHz}, \epsilon=99\% \rightarrow \tau < 0.1\text{ms} \rightarrow 1/\tau > 10\text{kHz}$

➔ In order to cope with the input signal fluctuations, we have to over-design our DAQ system by a factor 10. This is very inconvenient! Can we mitigate this effect?

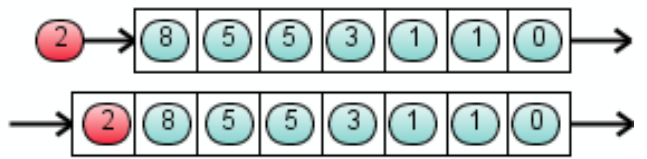
# Basic DAQ: De-randomization

$f=1\text{kHz}$   
 $1/f=\lambda=1\text{ms}$



## → First-In First-Out

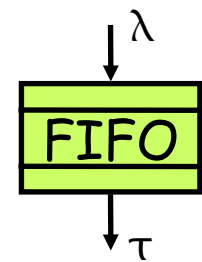
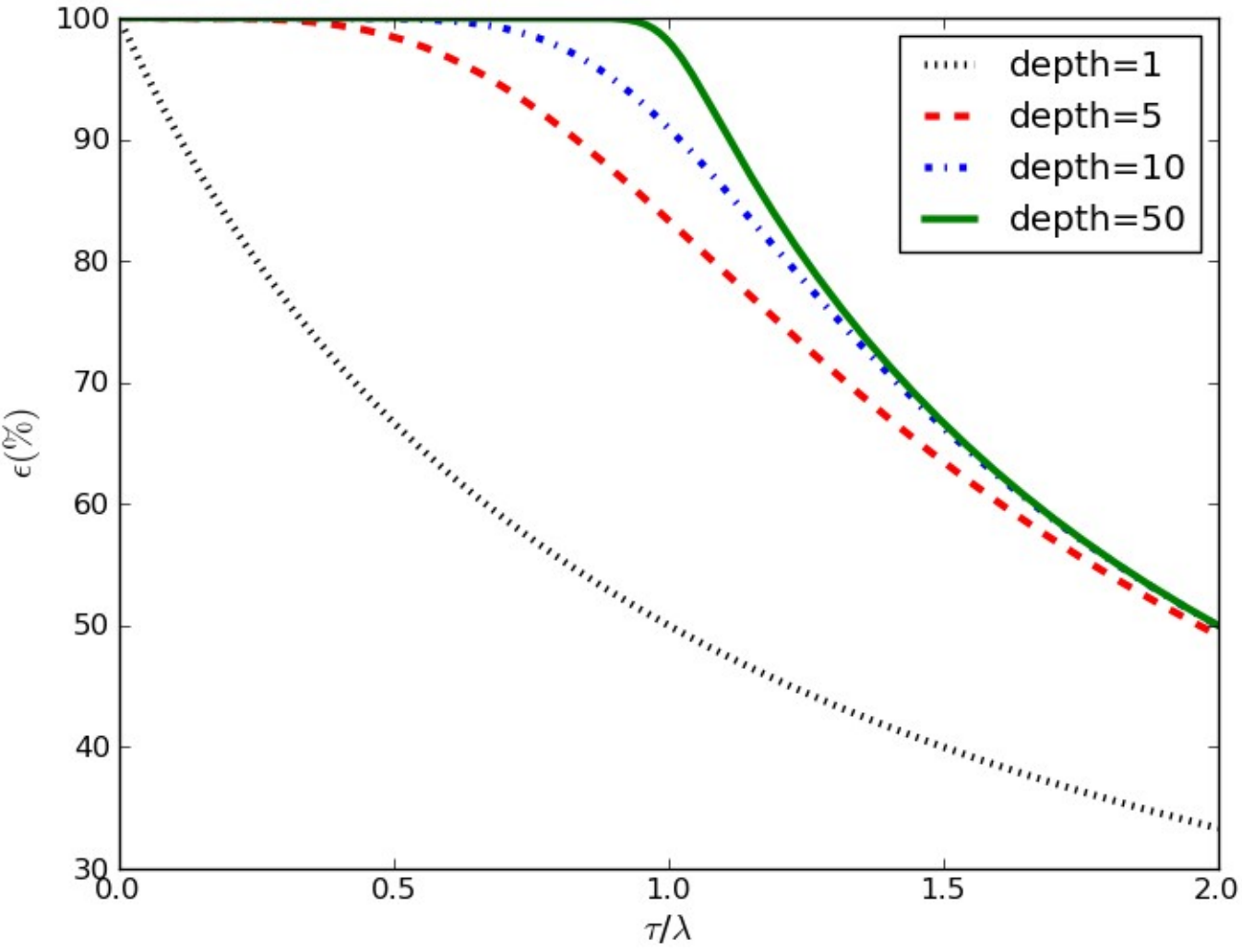
- buffer area organized as a queue
- depth: number of cells
- implemented in HW and SW



## → FIFO introduces an additional latency on the data path

The FIFO absorbs and smooths the input fluctuation, providing a ~steady (**De-randomized**) output rate

# De-randomization: queuing theory

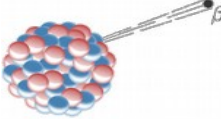


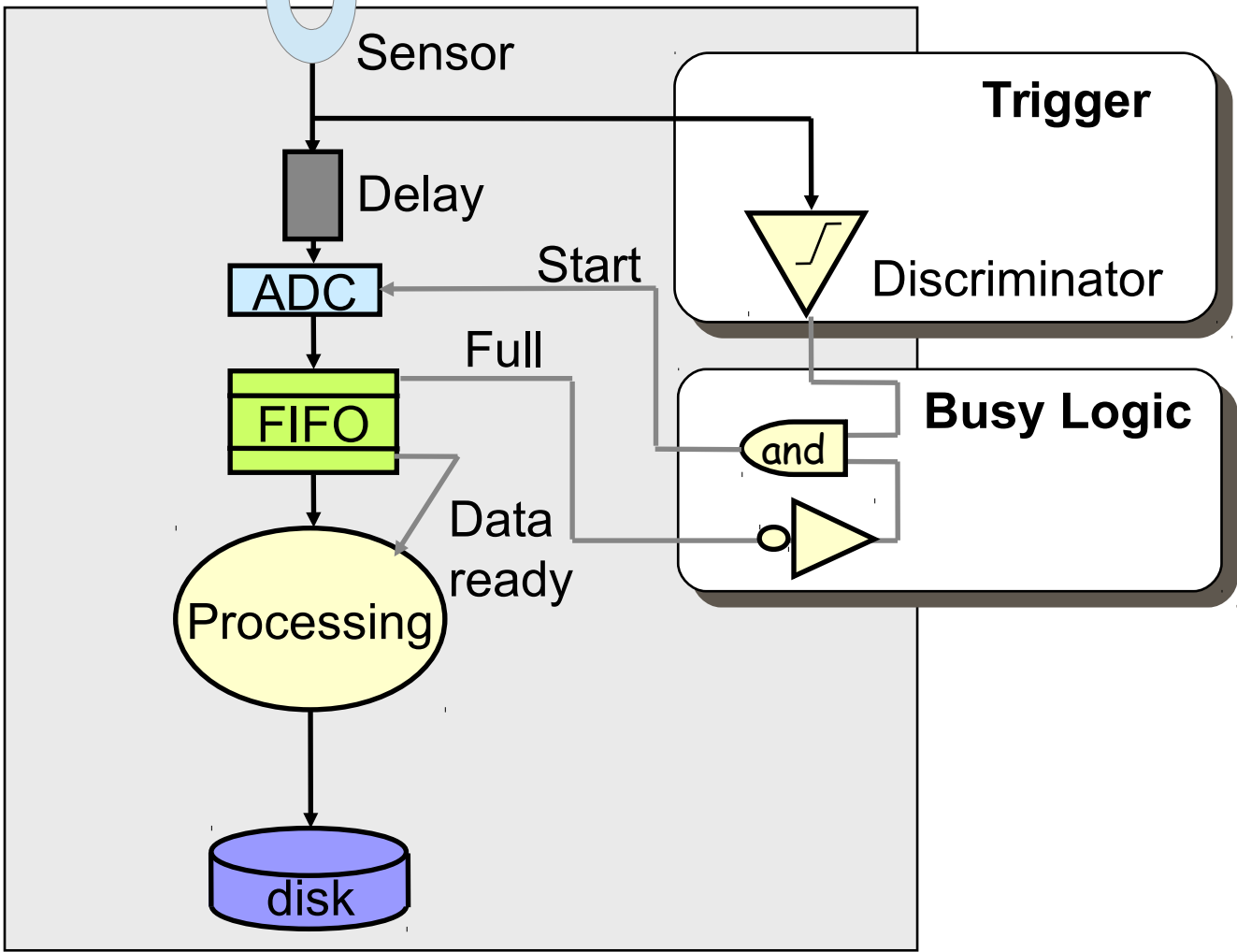
→ We can now attain a FIFO efficiency ~100% with  $\tau \sim \lambda$

- moderate buffer size

Analytic calculation possible for very simple systems only. Otherwise simulations must be used.

# De-randomization: summary

  $f=1\text{kHz}$   
 $1/f=\lambda=1\text{ms}$



→ Almost 100% efficiency and minimal deadtime are achieved if

- ADC is able to operate at rate  $\gg f$
- data processing and storing operates at  $\sim f$

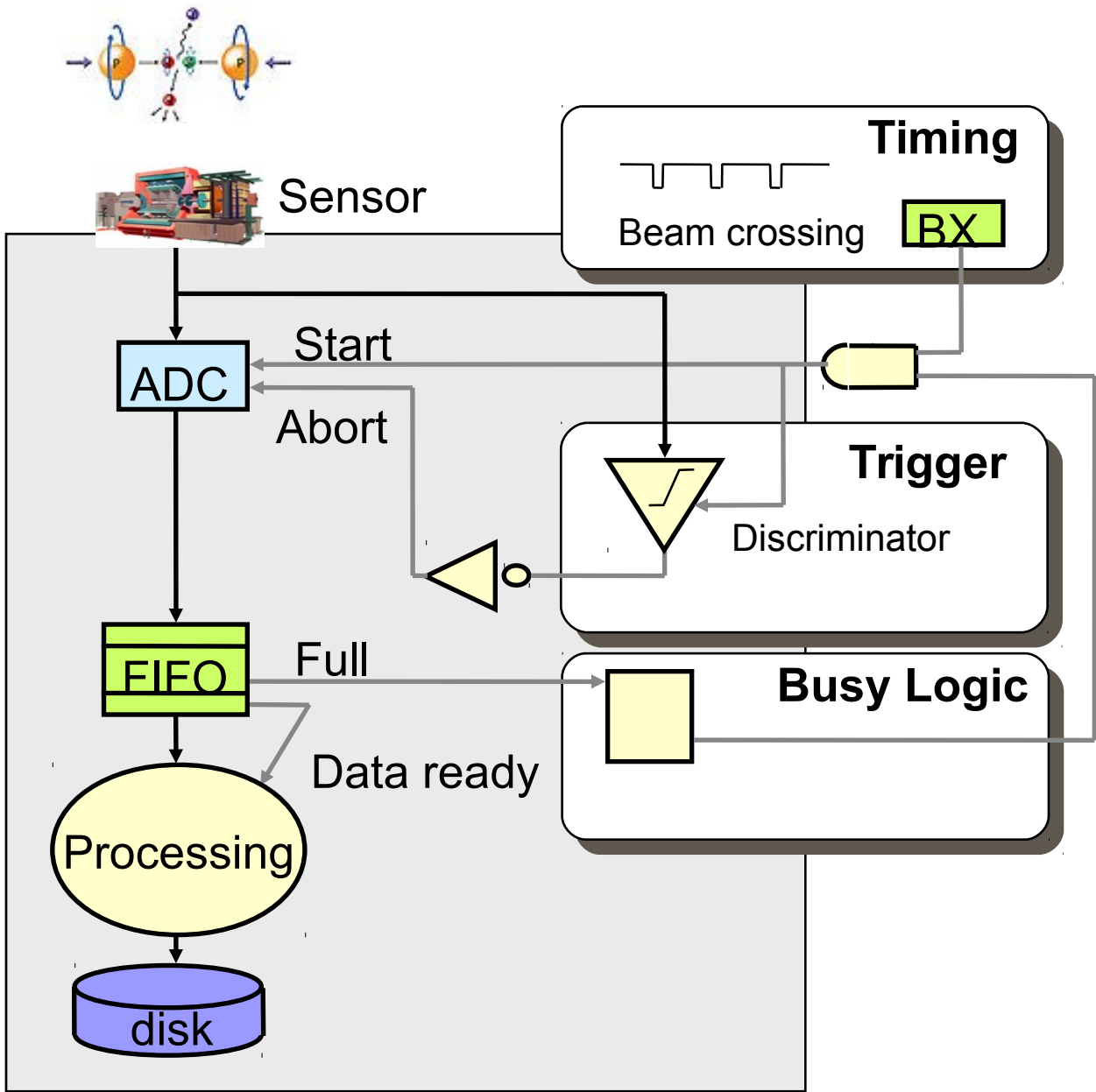
→ The FIFO decouples the low latency front-end from the data processing

- minimize the amount of “unnecessary” fast components

→ Could the delay be replaced with a “FIFO”?

- analog pipelines → Heavily used in LHC DAQs

# Basic DAQ: collider mode



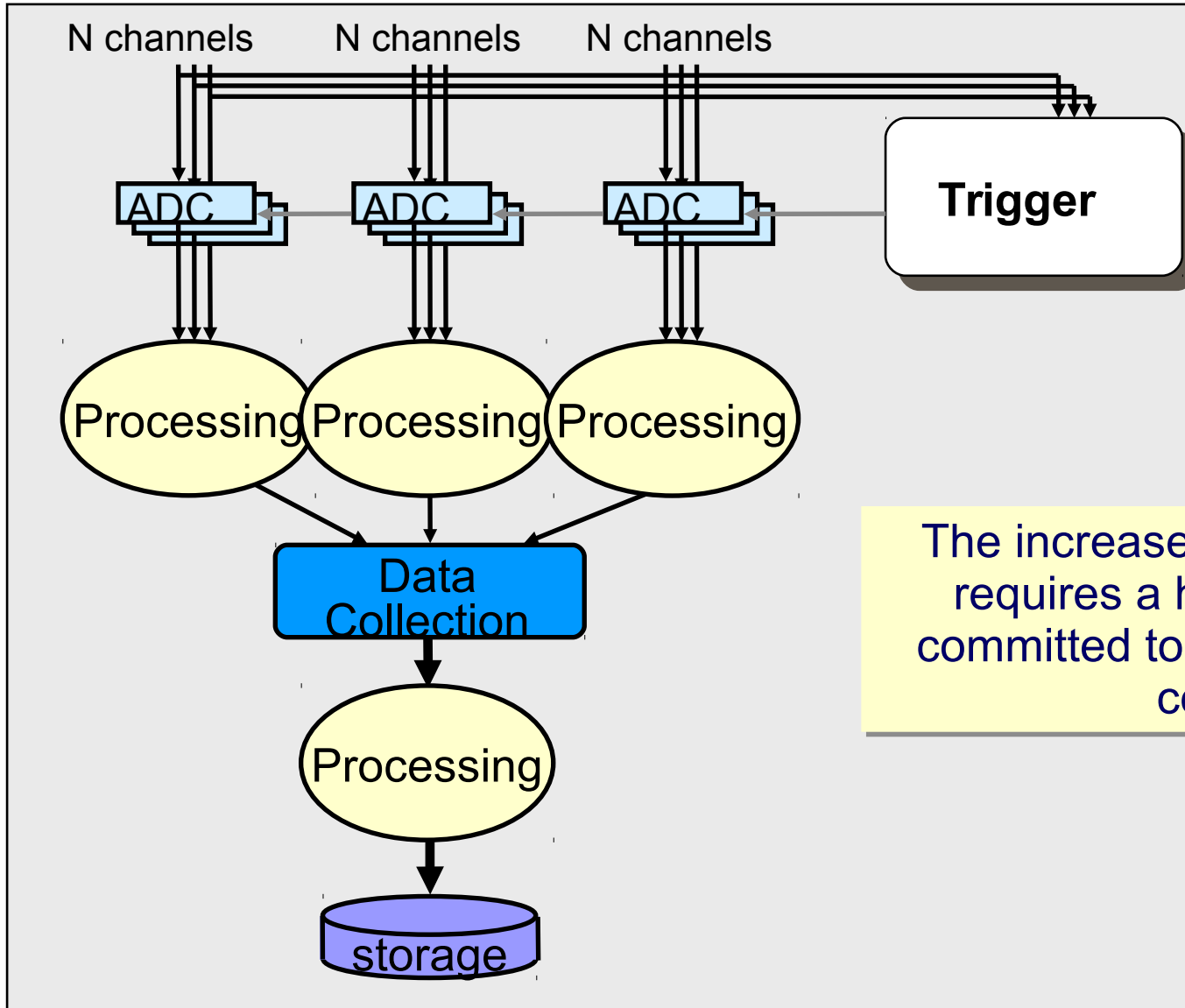
- Particle collisions are synchronous
- Trigger rejects uninteresting events
- Even if collisions are synchronous, the triggers (i.e. good events) are **unpredictable**
- **De-randomization is still needed**



# Scaling up: Network & Bus



# Basic DAQ: more channels

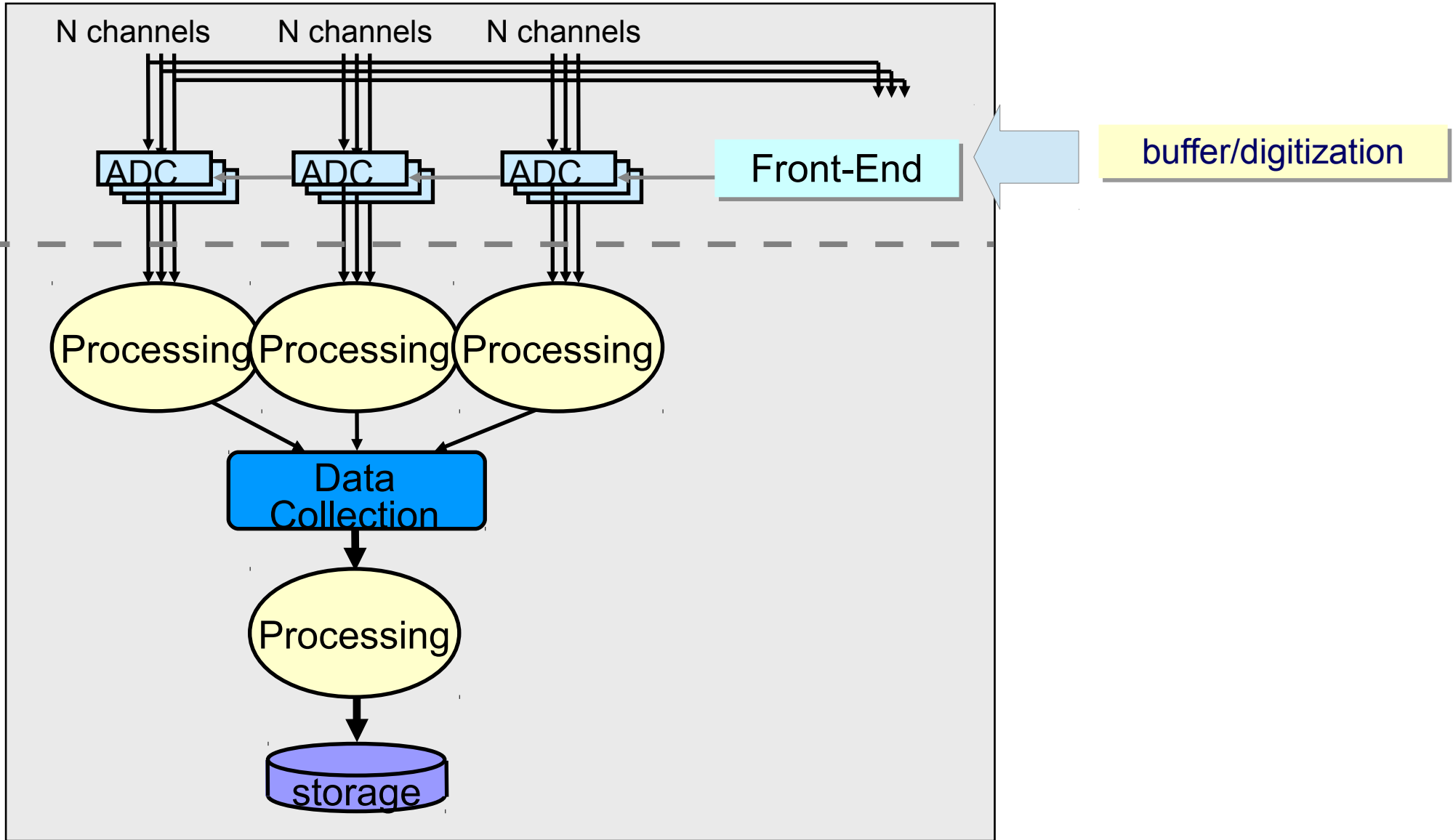


The increased number of channels requires a hierarchical structure committed to the data handling and conveyance



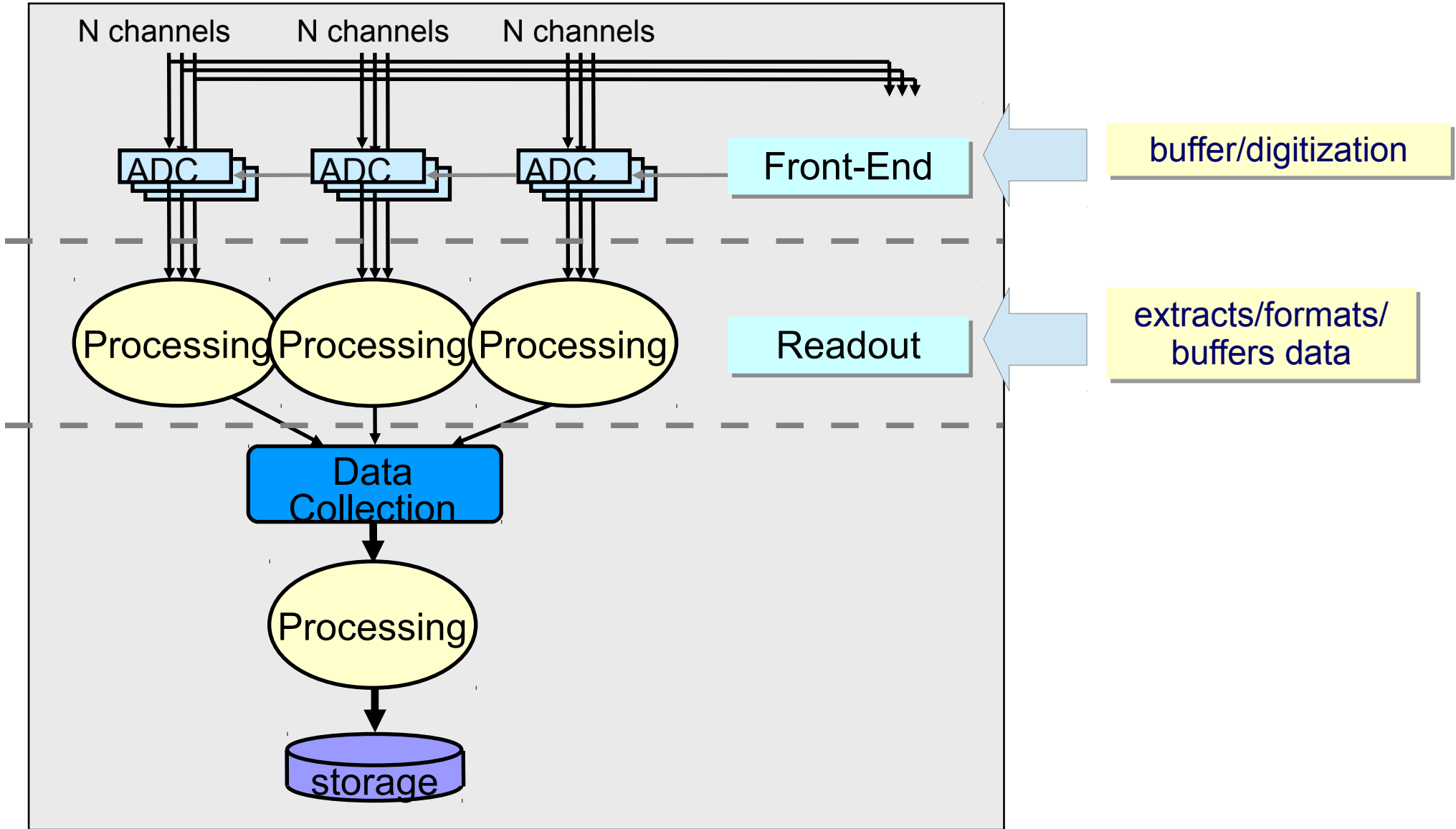


# Large DAQ: Constituents



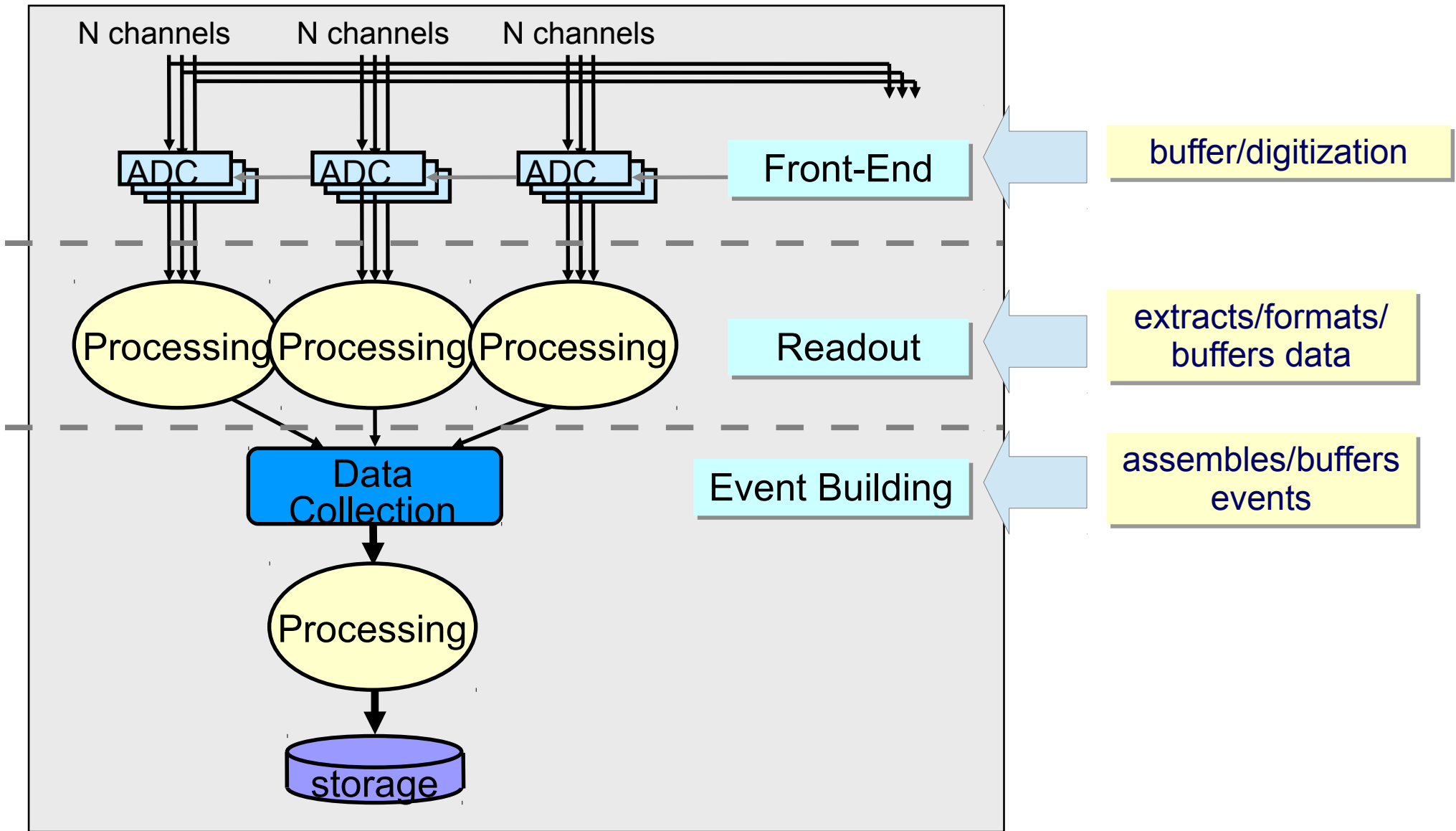


# Large DAQ: Constituents



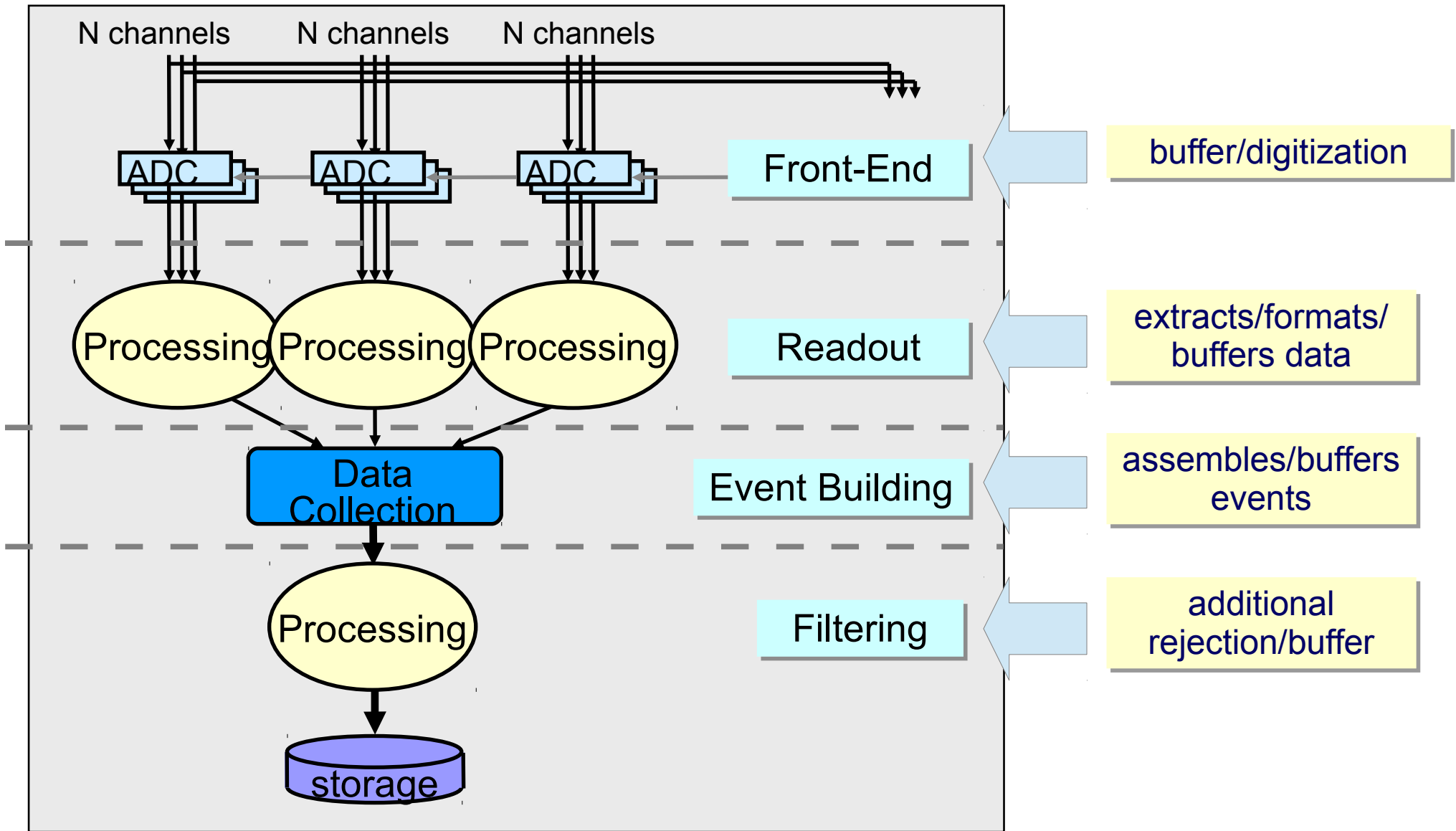


# Large DAQ: Constituents



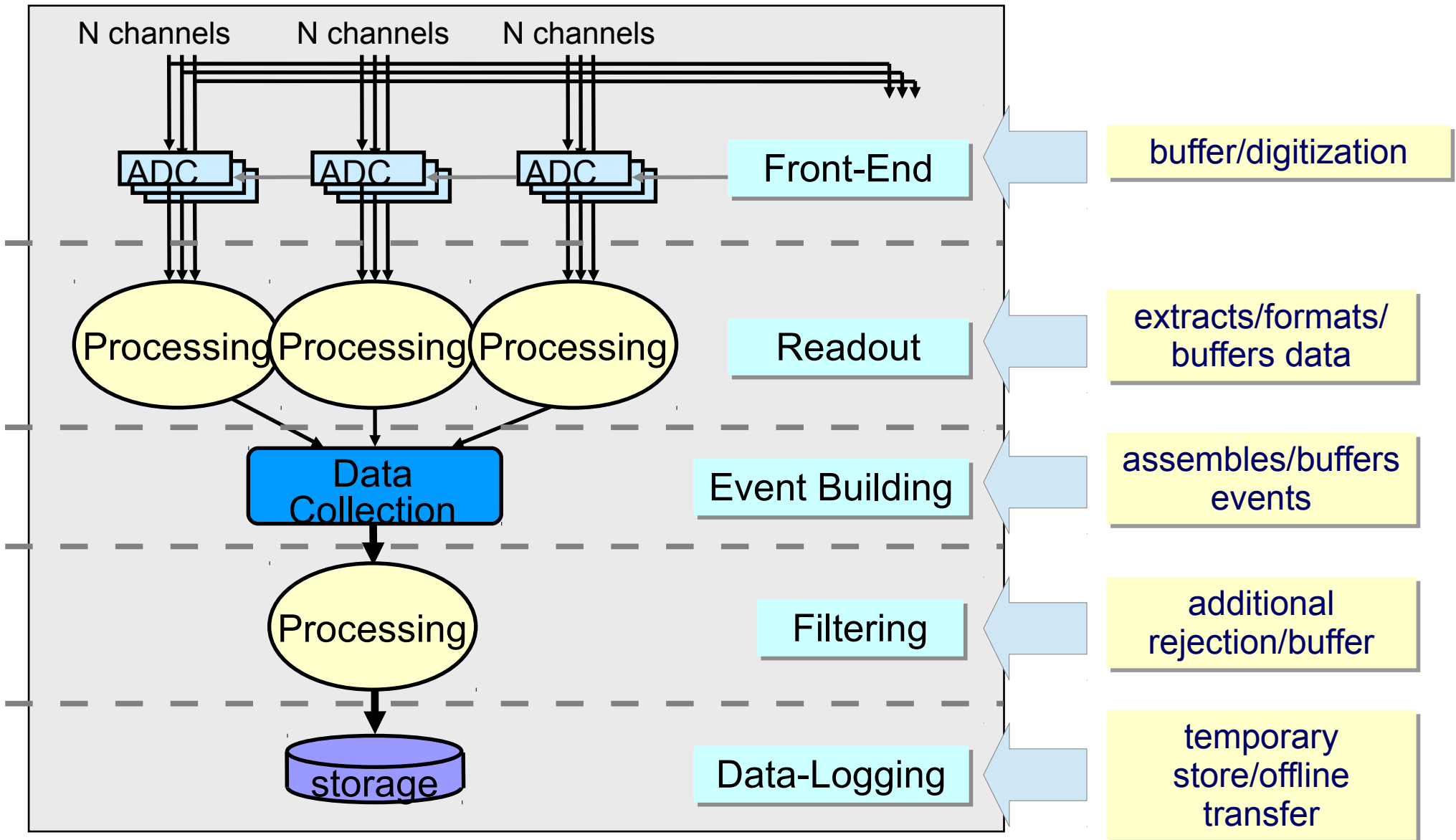


# Large DAQ: Constituents





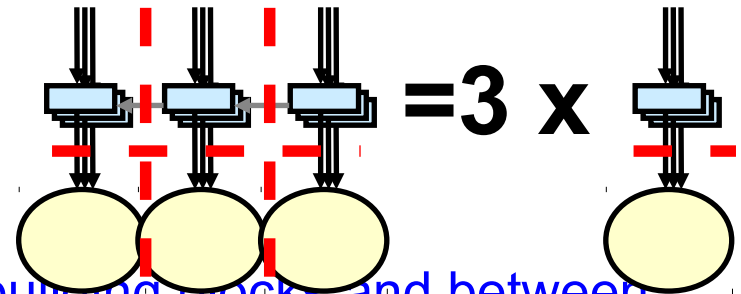
# Large DAQ: Constituents



# Readout Topology

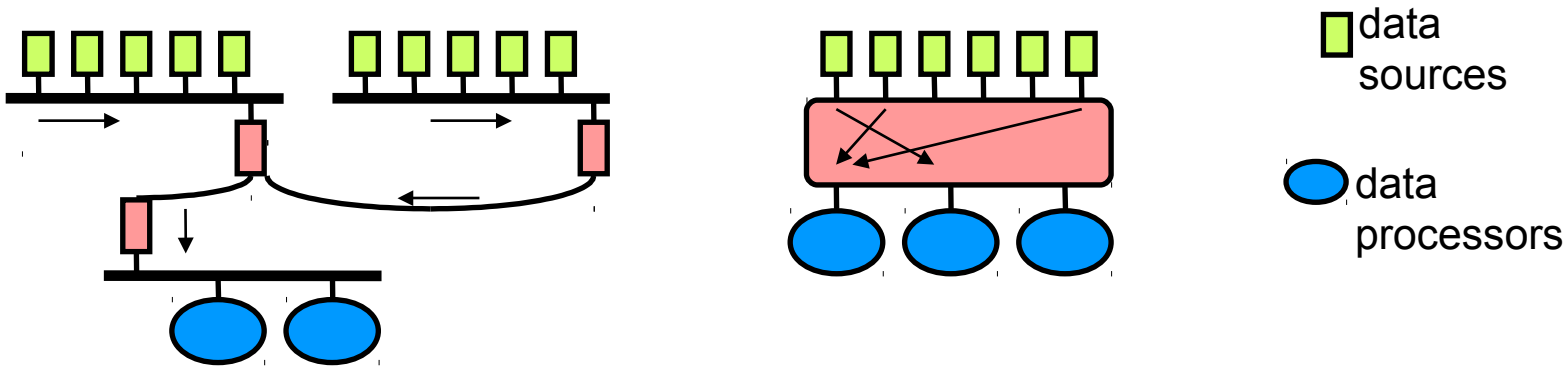
- Reading out or building events out of many channels requires many components
- In the design of our hierarchical data-collection system, we have better define “building blocks”

- example: readout crates, event building nodes, ...



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

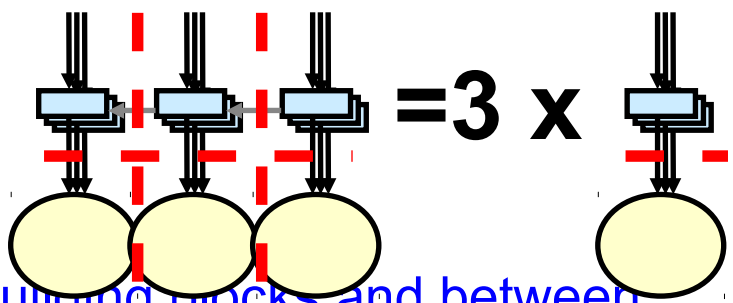
- Two main classes: bus or network



# Readout Topology

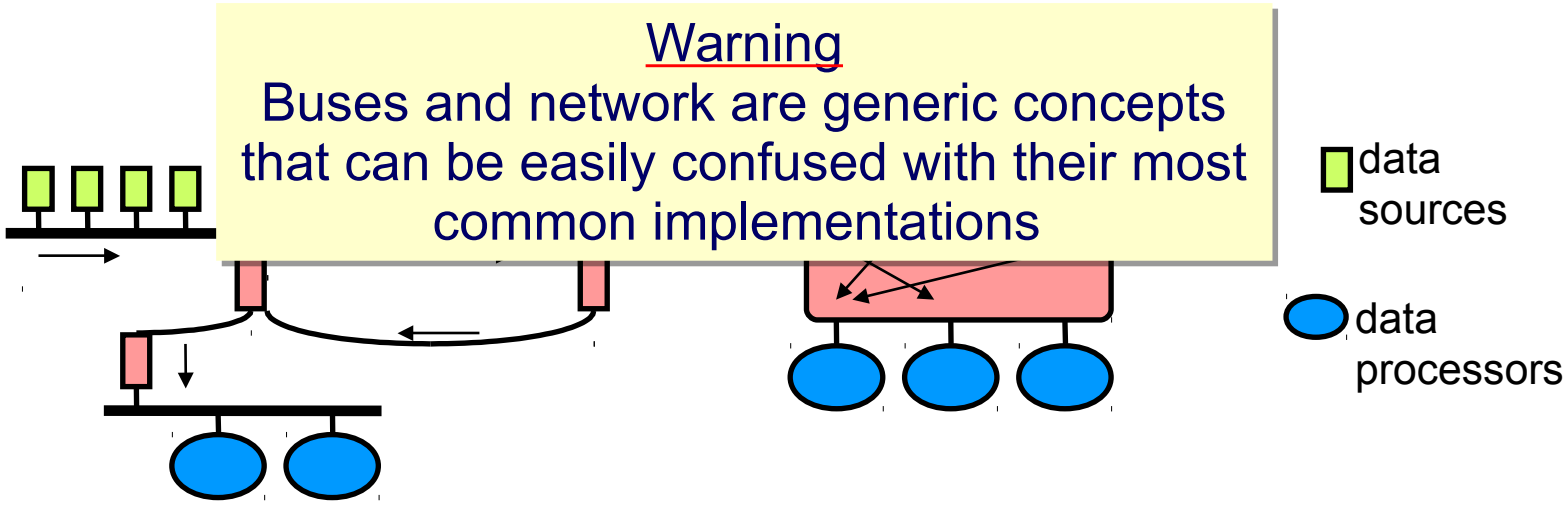
- Reading out or building events out of many channels requires many components
- In the design of our hierarchical data-collection system, we have better define “building blocks”

- example: readout crates, event building nodes, ...



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

- Two main classes: bus or network



# Buses

→ Examples: VME, PCI, SCSI, Parallel ATA, ...

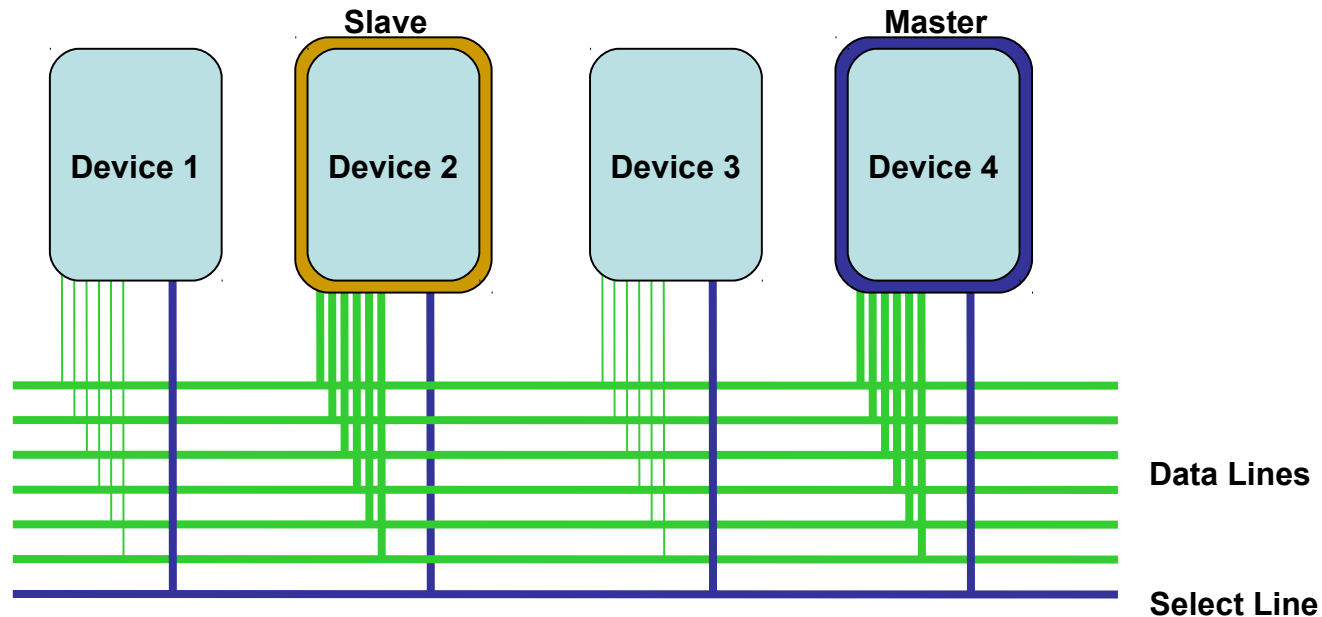
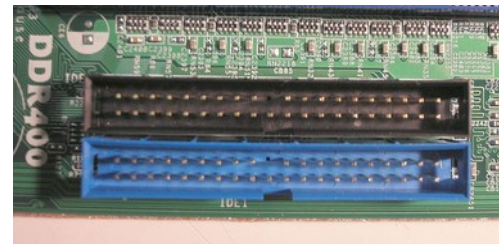
- local, external, crate, long distance

→ Devices are connected via a **shared bus**

- bus → group of electrical lines
- sharing implies arbitration

→ Devices can be **master or slave**

→ Device can be addresses (uniquely identified) on the bus

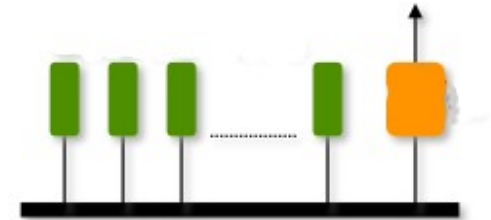




# Bus facts

## → 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 bus frequency is inversely proportional to the bus length
- maximum number of devices depends on the bus length

# Bus facts

## → Simple ✓

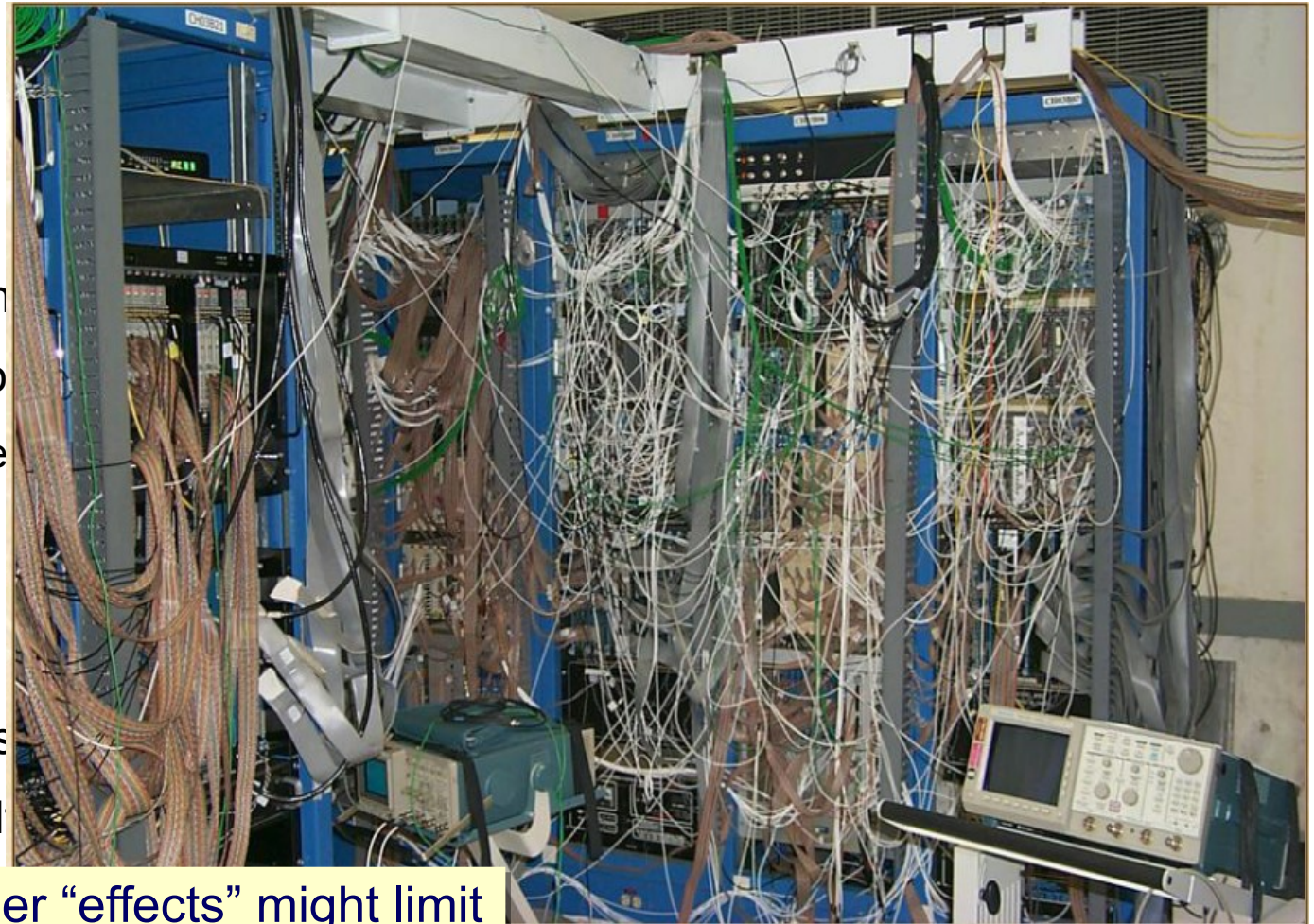
- fixed number of lines
- devices have to follow the bus
  - Mechanical, electrical

## → Scalability issues ✗

- bus bandwidth is shared
- maximum bus width is limited

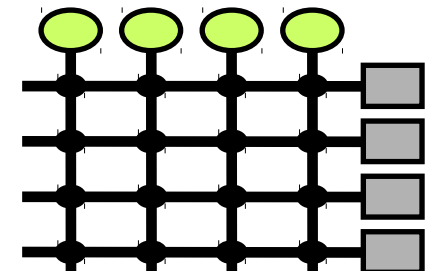
On the long term, other “effects” might limit the scalability of your system

the bus length



# Network

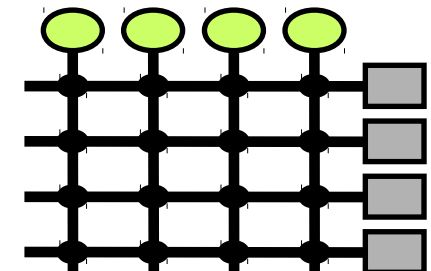
- Examples: Ethernet, Telephone, Infiniband, ...
- All devices are **equal**
- Devices **communicate directly** with each other
  - no arbitration, simultaneous communications
- Device communicate by sending messages
- In switched network, **switches** move messages between sources and destinations
  - find the right path
  - handle “congestion” (two messages with the same destination at the same time)
    - would you be surprised to learn that buffering is the key?





# Network

- Examples: Ethernet, Telephone, Infiniband, ...
- All devices are **equal**
- Devices **communicate directly** with each other
  - no arbitration, simultaneous communications
- Device communicate by sending messages
- In switched network. **switches** move messages between sources and destinations. Thanks to these characteristics, **networks do scale** well. They are the backbones of LHC DAQ systems
  - find the right path
  - handle “congestion” (two messages with the same destination at the same time)
    - would you be surprised to learn that buffering is the key?





# Modular Electronics

# Modular Electronics

→ Standard electronics “functions” implemented in well-defined “containers”

- re-use of generic modules for different applications
- limit the complexity of individual modules → reliability & maintainability
- easy to upgrade to newer versions
- profit from commercially available “functions”

→ “Containers” are normally well-defined standards defining mechanical, electrical, ... , interfaces

- “easy” design and integrate your own module

→ Historically, in HEP, modular electronics was bus-based

- currently in a mixed phase ...

**Allow building your own data-acquisition system just connecting predefined functions → Fast & Efficient**



# NIM

## → NIM (1964)

- “Nuclear Instrumentation Modules”

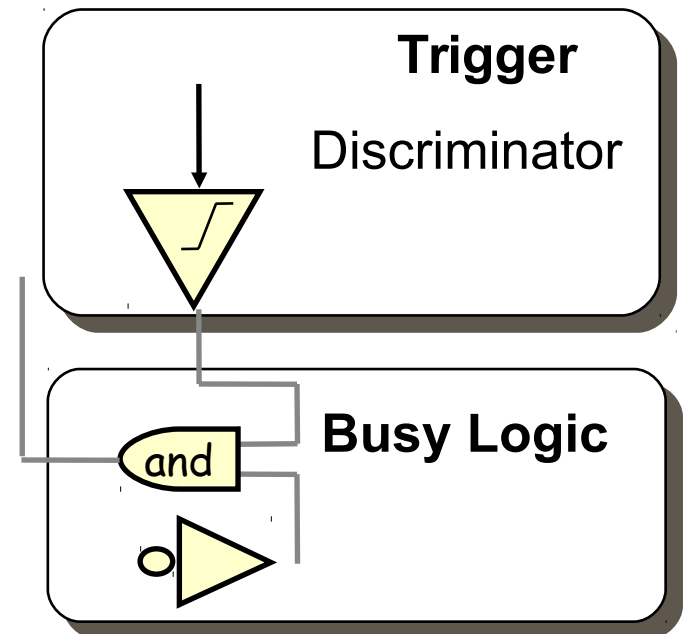
## → NIM modules usually

- do not need software, are not connected to PCs
- implement logic and signal processing functions
  - Discriminators, Coincidences, Amplifiers, Logic gates, ...

## → Typically implement basic Trigger and Busy system



New modules still appear on the market  
Very diffused in medium-sized HEP experiments  
Found in the counting rooms of LHC experiments







# VMEbus

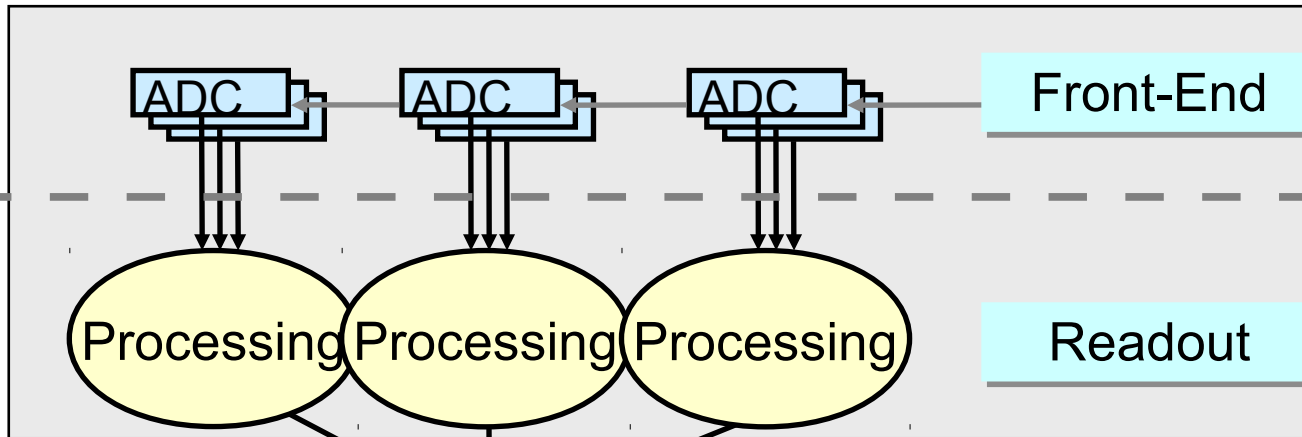
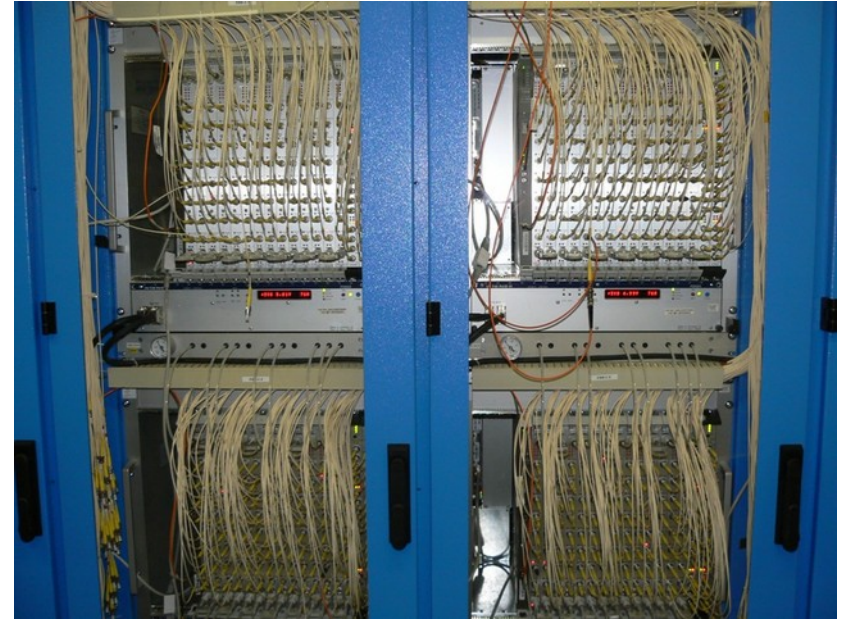
→ VMEbus: modules communicate via a “backplane”

- electrical, mechanical and communication protocols

→ Choice of many HEP experiments

- relatively simple protocol
- a lot of commercially available functions

→ More than 1000 VMEbus crates at CERN





# Other (arising) standards

→ PCI-based



→ We know buses have limited scalability. Can we have “network-based” modular electronics?

→ VXS → essentially VME plus switched interconnectivity

→ ATCA and derivatives

- standard designed for telecom companies
- high-redundancy, data-throughput, high power density
- **being used for LHC upgrade programs**

