

Monika Wielers

Rutherford Appleton
Laboratory

- Lecture 1

- Introduction to data acquisition (DAQ) systems

- Lecture 2

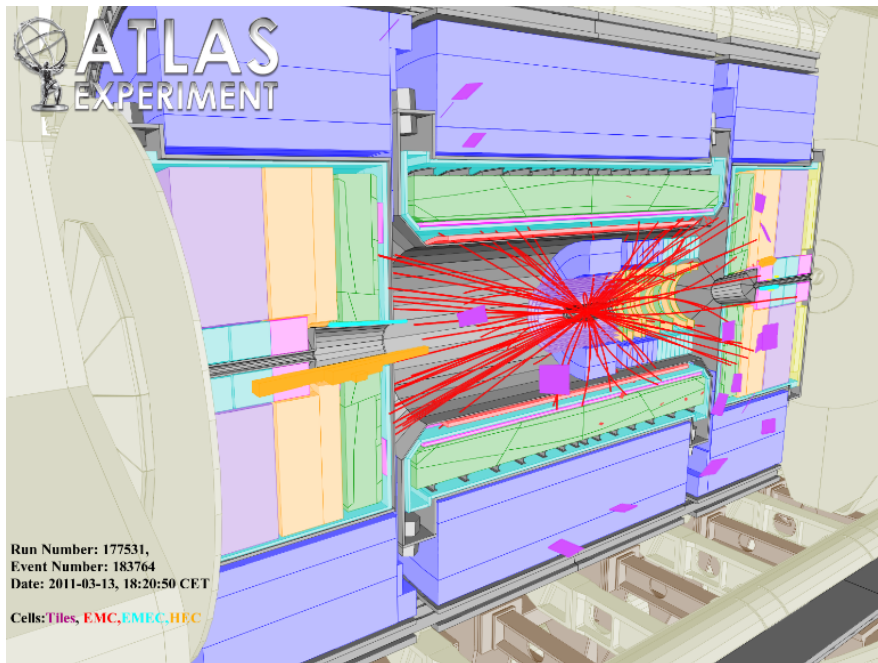
- DAQ for LHC experiments

- Lecture 3

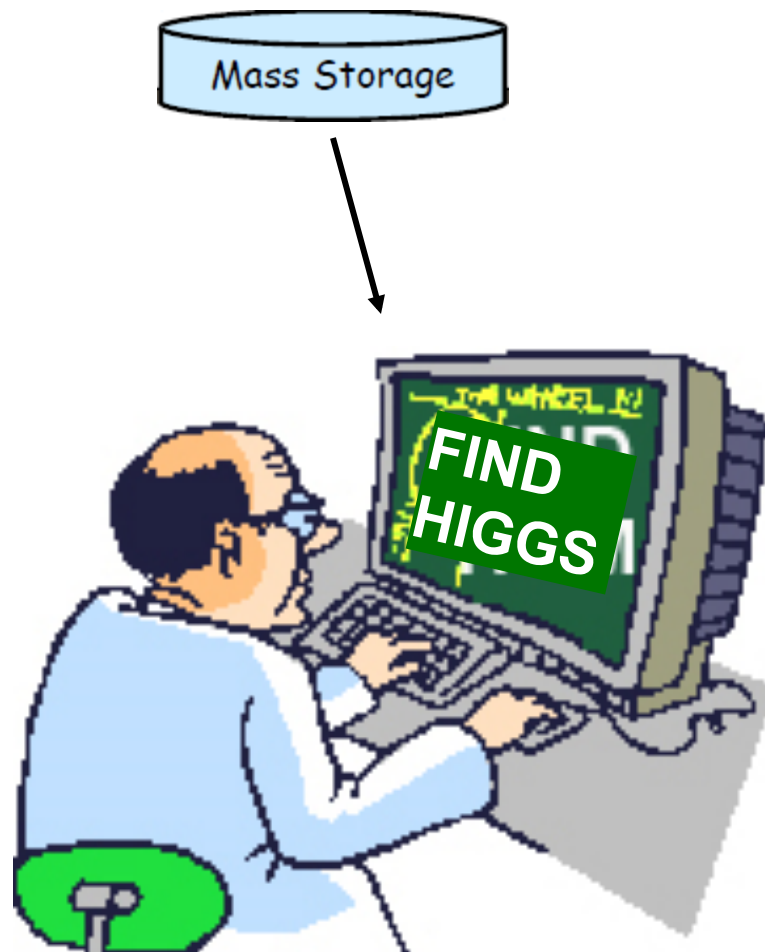
- Trigger

What is it about...

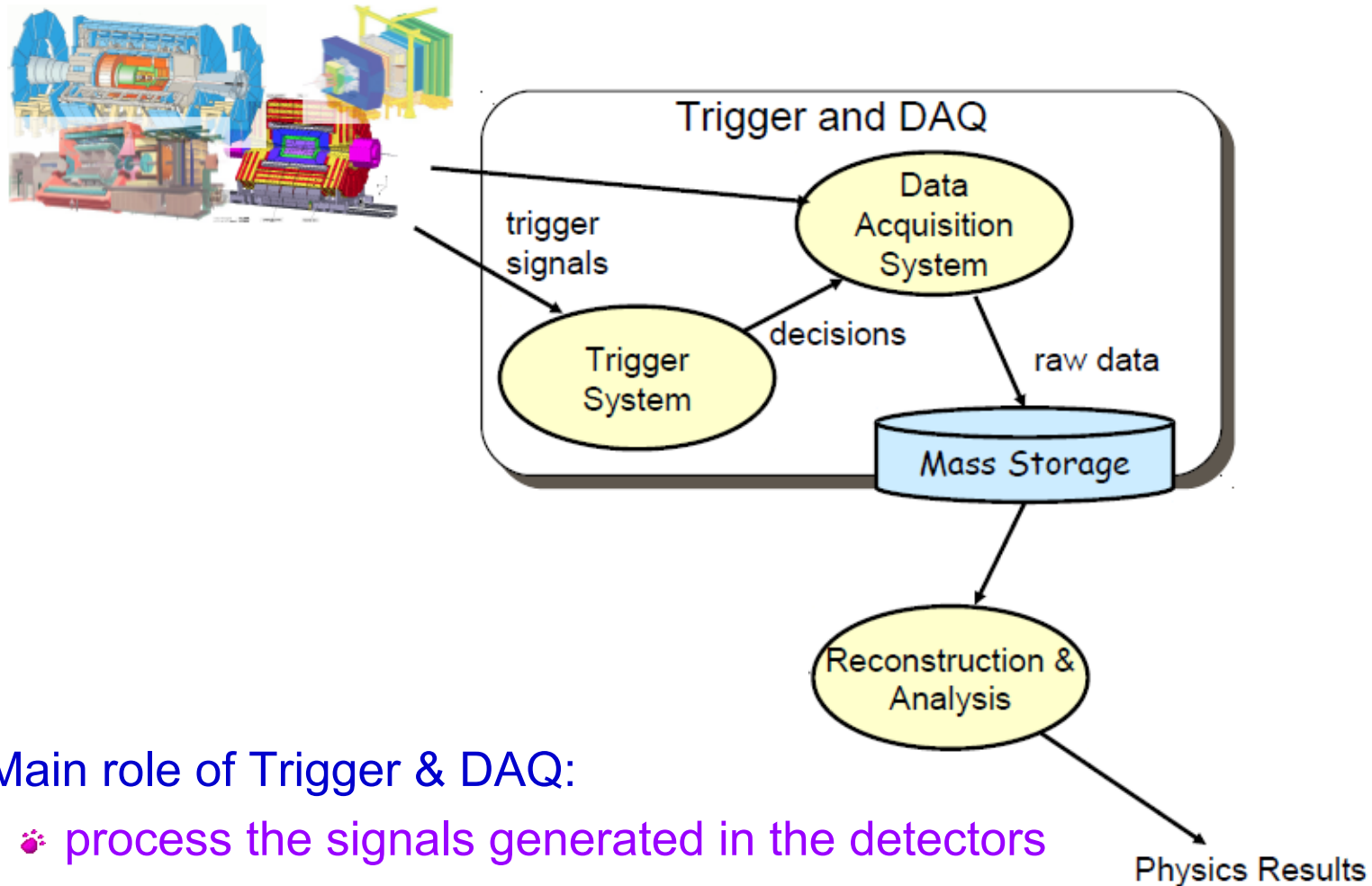
How to get from



to



Or



❖ Main role of Trigger & DAQ:

- ❖ process the signals generated in the detectors
- ❖ Select the 'interesting' events and reject the 'boring' ones
- ❖ save interesting ones on mass storage for physics analysis

 **Heartbeat of the experiment!**

DAQ

- Abbreviation for Data Acquisition System
- Wikipedia:
 - process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer.
- In HEP it consists mainly of electronics, computer science, networking and quite some physics
- Tasks
 - Gathers the data produced by the detectors (Readout)
 - Possibly feeds (several) levels of deciding to keep the collision (called typically **event** in the following)
 - Forms complete events (Event Building)
 - Stores event data (Data logging)
 - Provides control, configuration and monitoring facilities

Trigger

- That's one

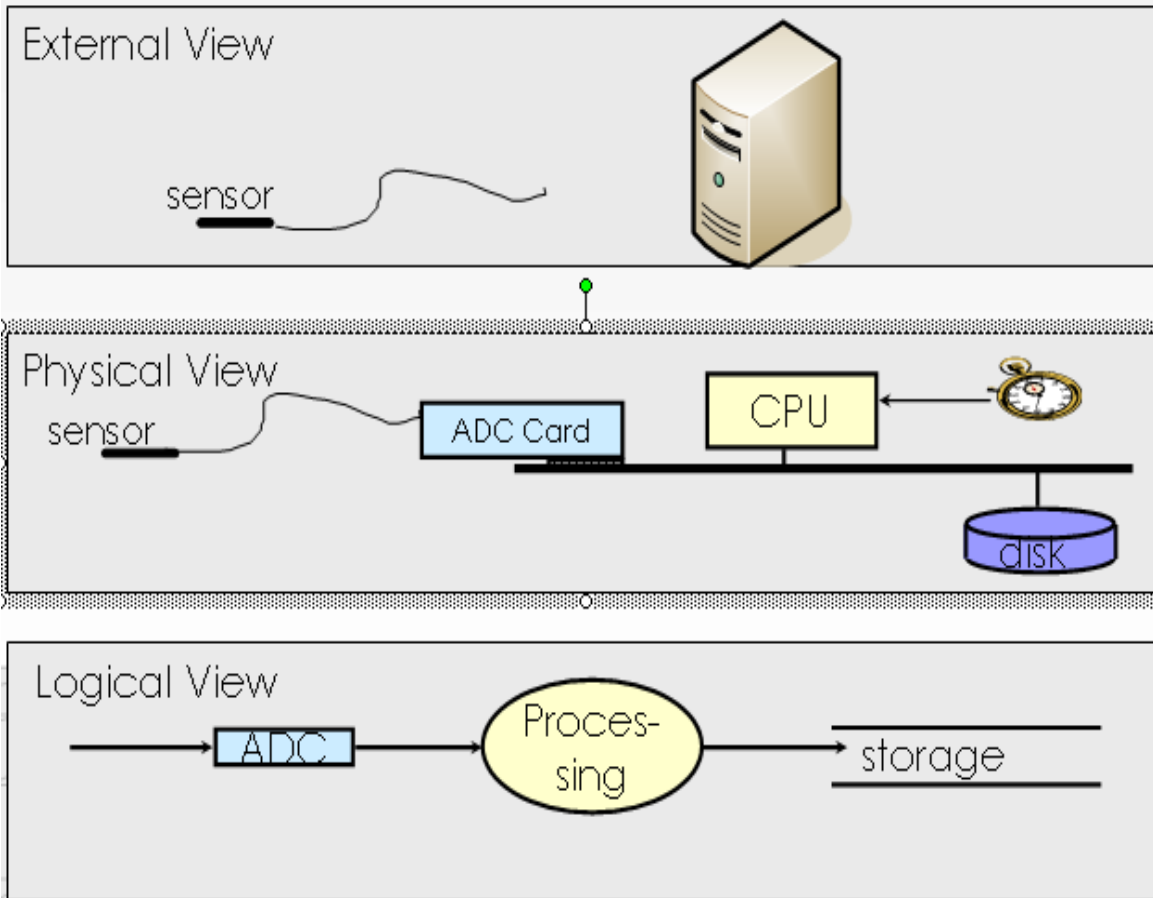


- but that's not what we want to discuss here
- Trigger = in general something which tells you when is the “right” moment to take your data
- Trigger = process to very rapidly decide if you want to keep the data if you can't keep all of them. The decision is based on some ‘simple’ criteria
- This can happen in several stages, if needed
- Note, DAQ and Trigger often are not two separate issues, but are ‘interwoven’

Goals of this lecture

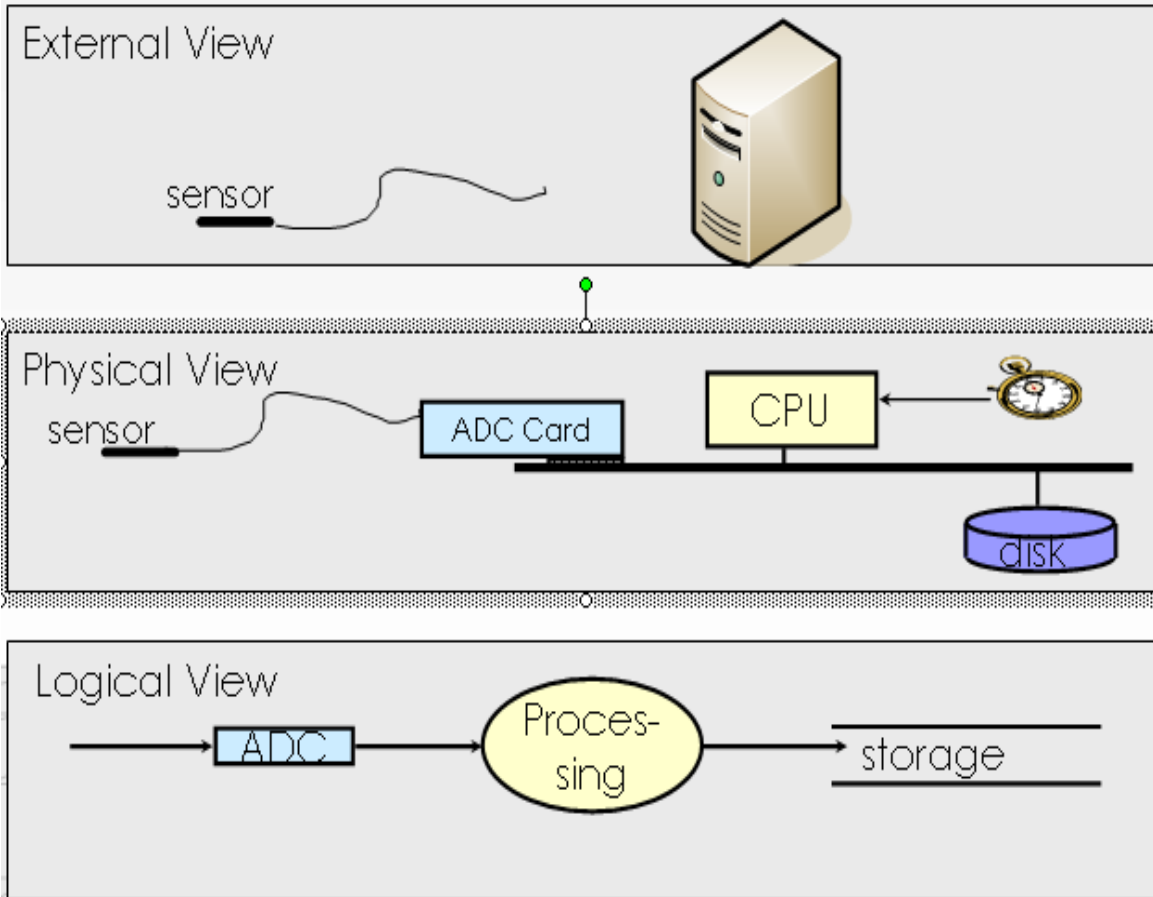
- Understand how data acquisition is devised
 - Start with simple example and then get more complex
- Introduce the terms you will hear when you hear about data acquisition in a particle physics experiment
- All this might be a bit technical but might help you later during your Ph.D. and it is actually also quite some fun!

Trivial DAQ (periodic trigger)



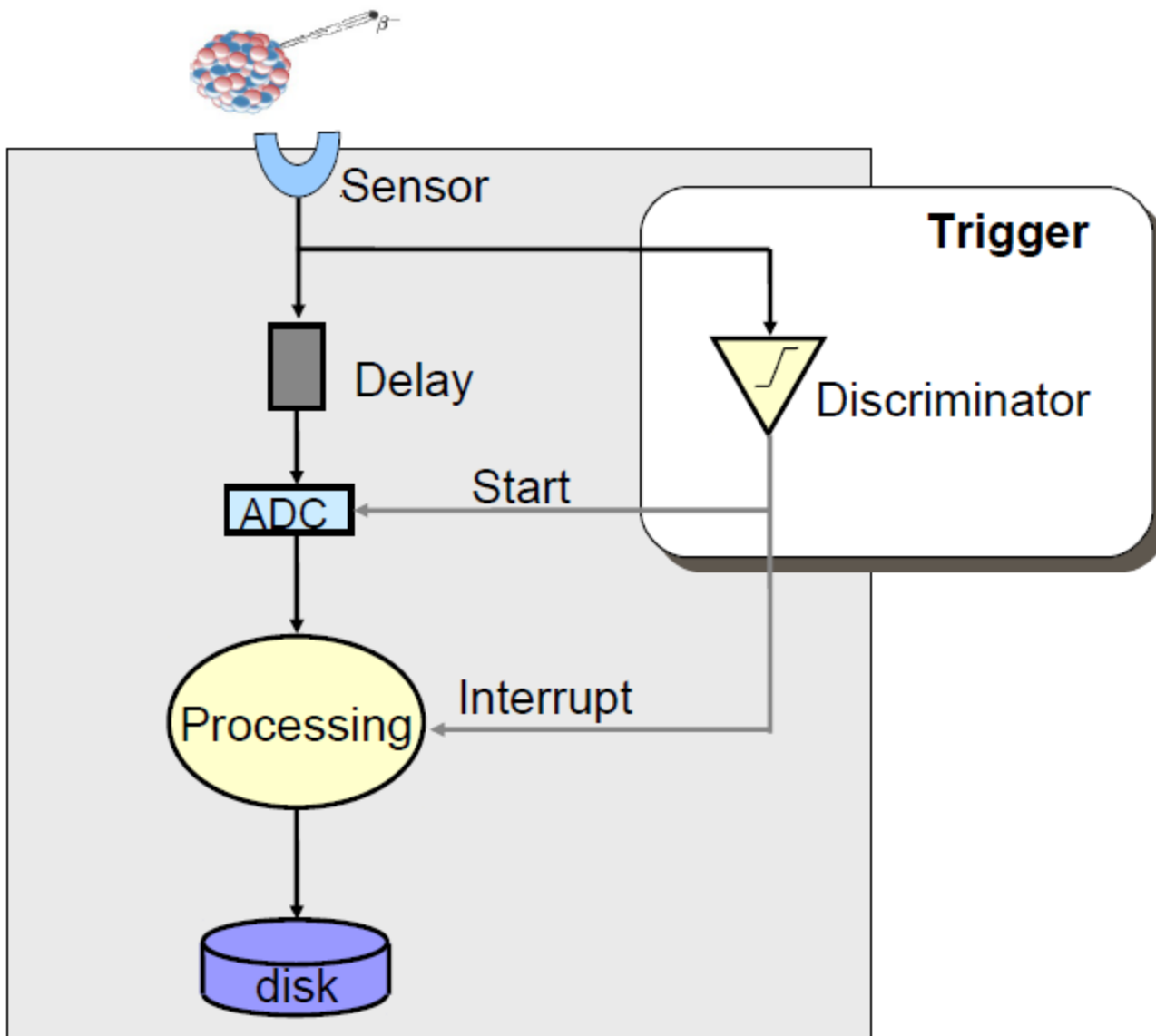
- Measure temperature at a fixed frequency
- ADC performs analog to digital conversion (digitisation)
 - Our frontend electronics
- CPU does readout and processing

Trivial DAQ (periodic trigger)



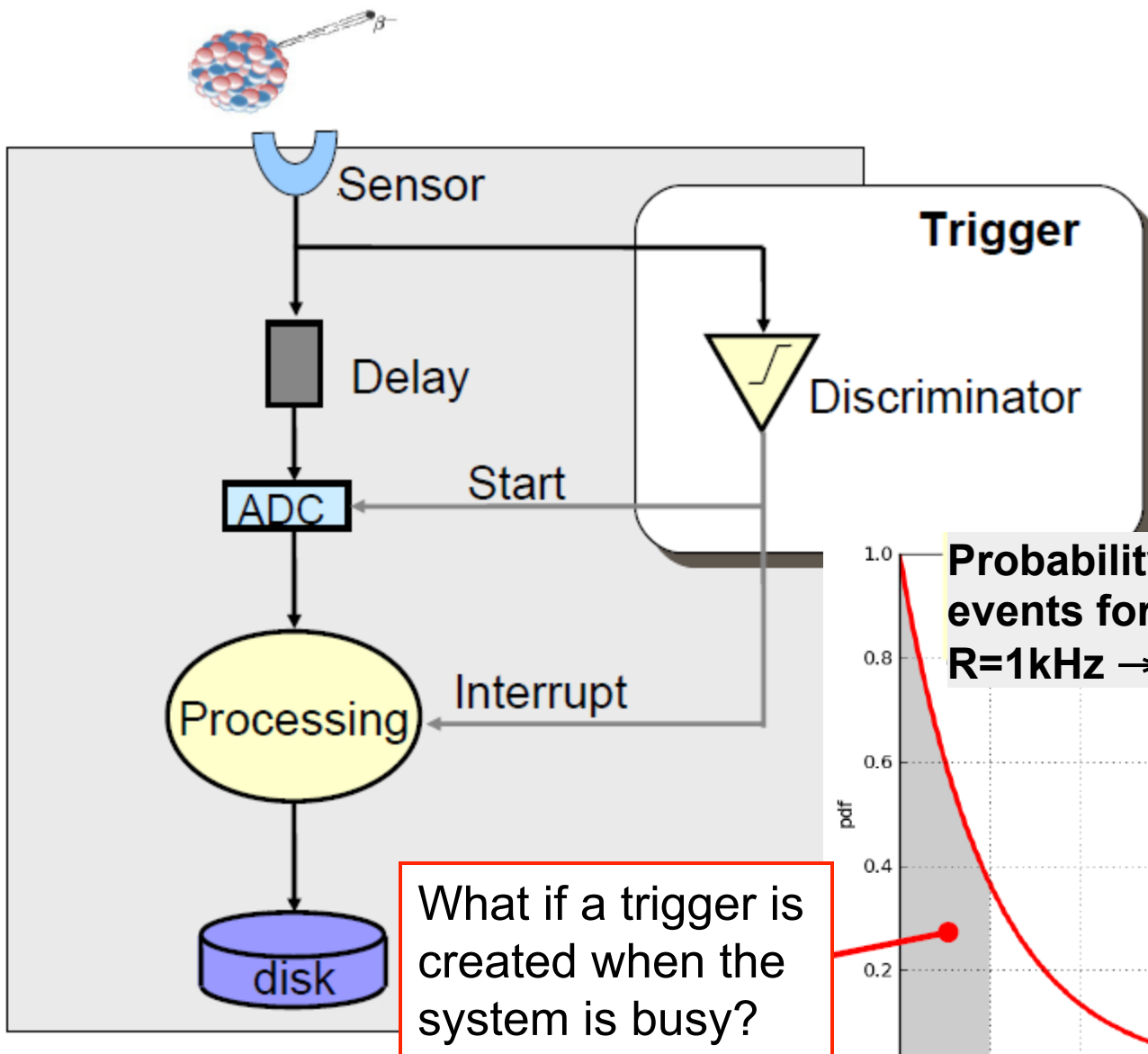
- Measure temperature at a fixed frequency
- The system is clearly limited by the time to process a measurement (or event)
- Example $\tau=1\text{ms}$ to
 - ADC conversion
 - +CPU processing
 - +Storage
- Sustain $\sim 1/1\text{ms}=1\text{kHz}$
periodic trigger rate

Trivial DAQ with a trigger

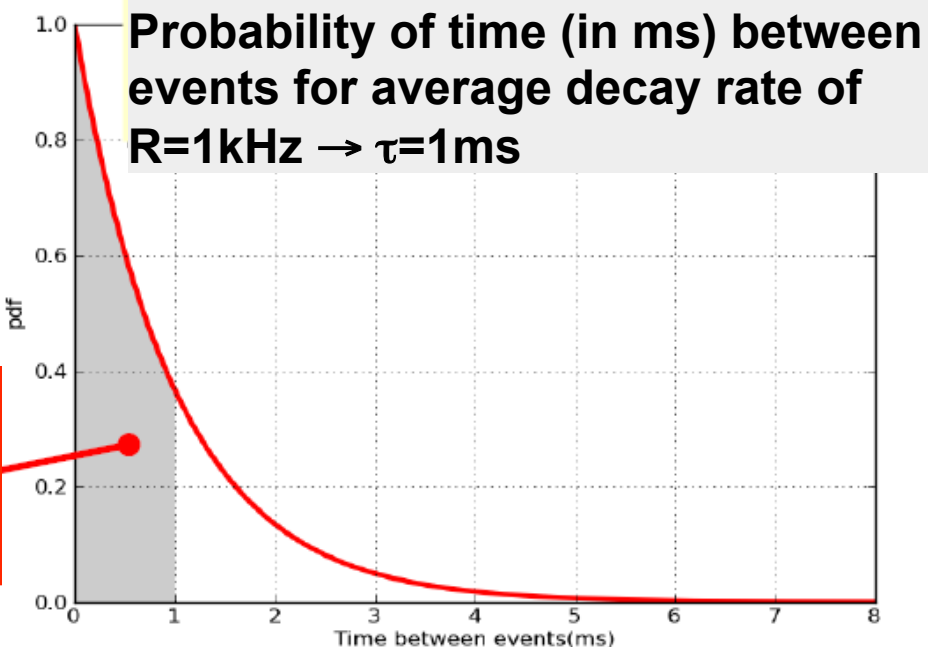


- Example: measure β decay properties
- Our events are asynchronous and unpredictable
 - Need a **physics** trigger
- Delay compensates for the trigger latency

Trivial DAQ with a trigger



- Example: measure β decay properties
- Need a **physics** trigger



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

Deadtime and Efficiency

- **Deadtime** (%): ratio between the time the DAQ is busy and the total time (Dead time / trigger = τ sec.)

- $\nu \cdot \tau \rightarrow$ DAQ is busy — $(1 - \nu \cdot \tau) \rightarrow$ DAQ is free, $\nu =$ output rate

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

f input rate
 τ readout time

- **Efficiency:** $N_{\text{saved}}/N_{\text{tot}} = 1/(1 + f \cdot \tau)$

- Note, due to the fluctuations introduced by the stochastic process the efficiency will always be less 100%

- In our specific example, $d=0.1\%/Hz$, $f=1kHz \rightarrow \nu=500Hz$, $\epsilon=50\%$

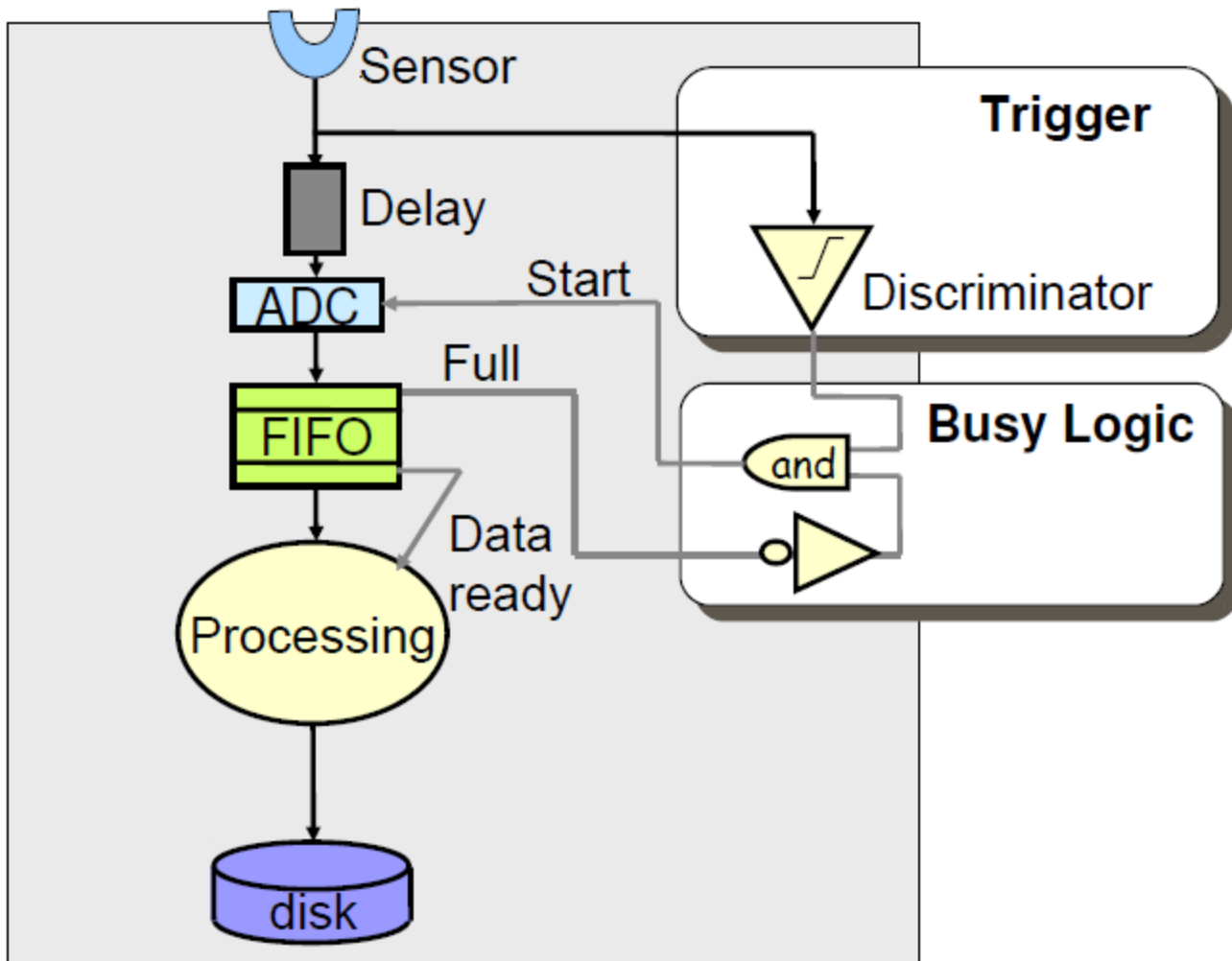
- If we want to obtain $\epsilon \sim 100\% \rightarrow f \cdot \tau \ll 1$

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

- In order to cope with the input signal fluctuations, we would need to overdesign our DAQ system by a factor 10. hmmm...

Trivial DAQ with Derandomisation

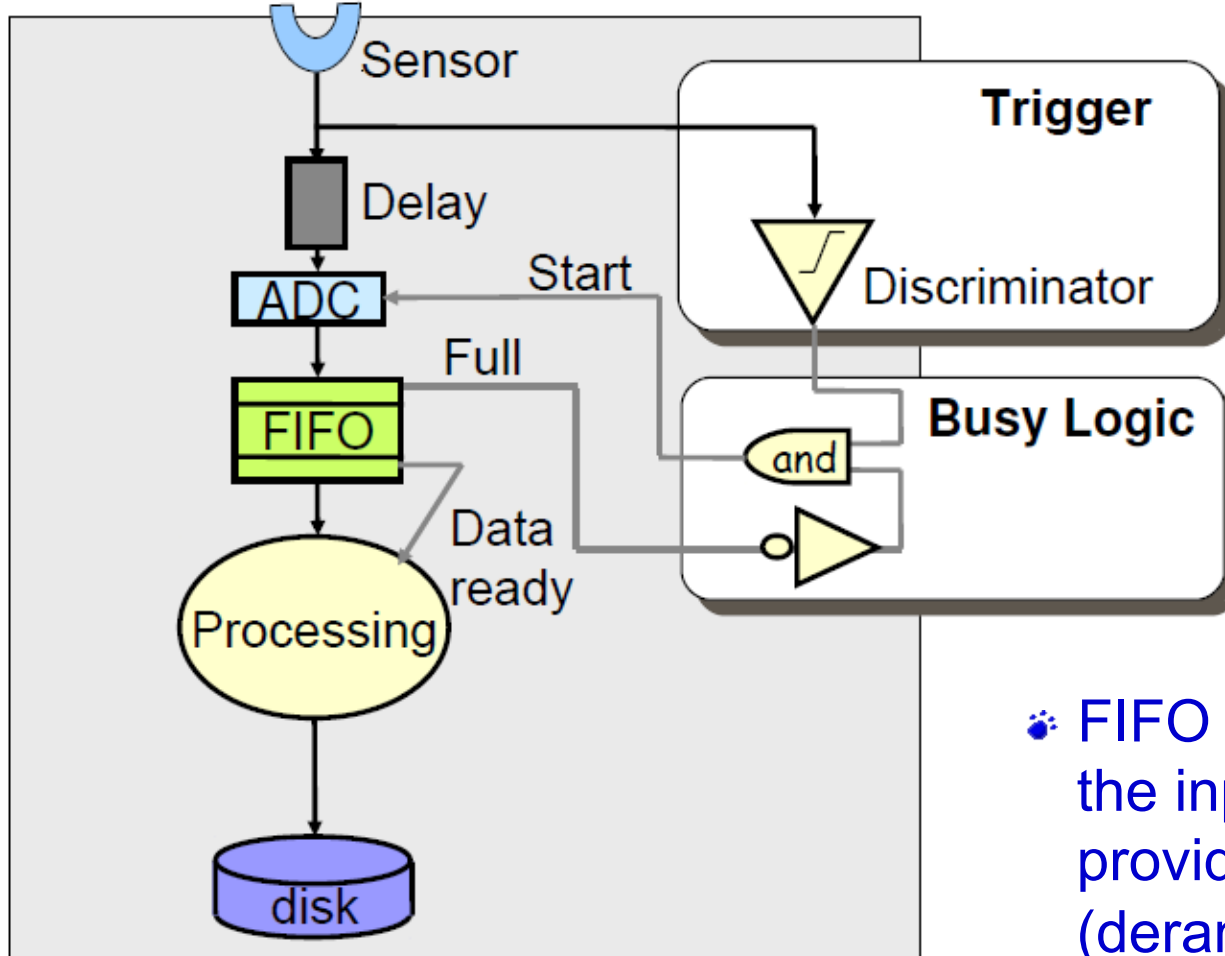
$R=1\text{ kHz}$
 $\tau = 1\text{ ms}$



- Buffers are introduced which hold temporarily the data.
- They decouple the data production from the data processing → Better performance

Trivial DAQ with Derandomisation

$R=1\text{ kHz}$
 $1/\tau = 1\text{ ms}$



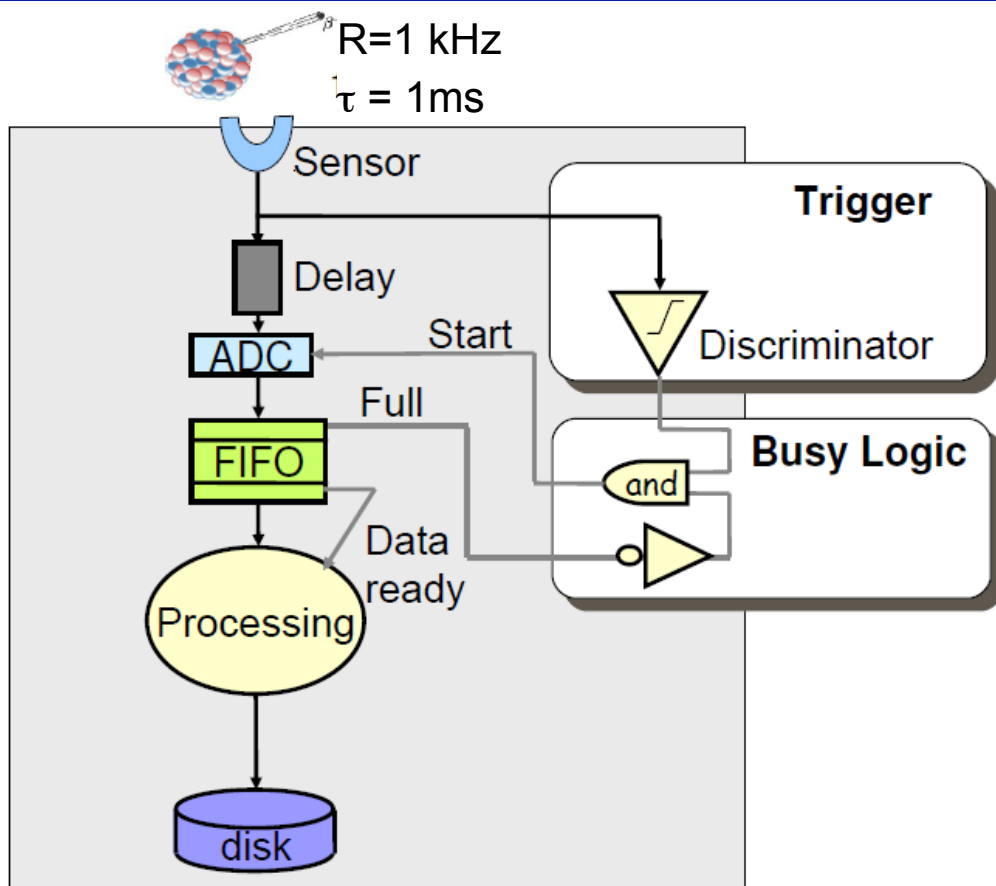
First In First Out

- Buffer area organized as a queue
- Depth: number of cells
- Implemented in HW and SW



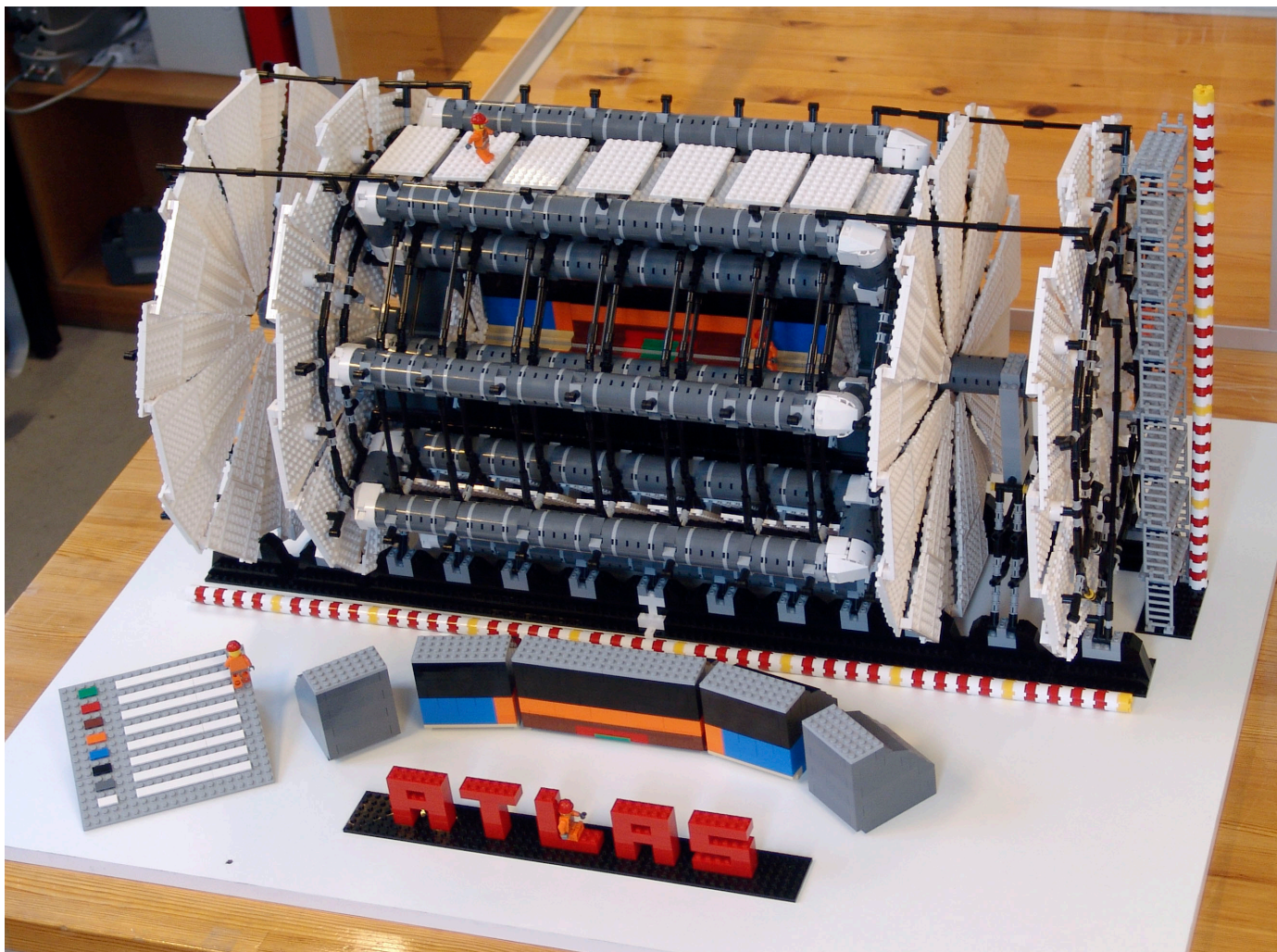
- FIFO absorbs and smoothes the input fluctuations, providing a ~steady (derandomized) output rate
- introduces an additional latency on the data path

Trivial DAQ with Derandomisation

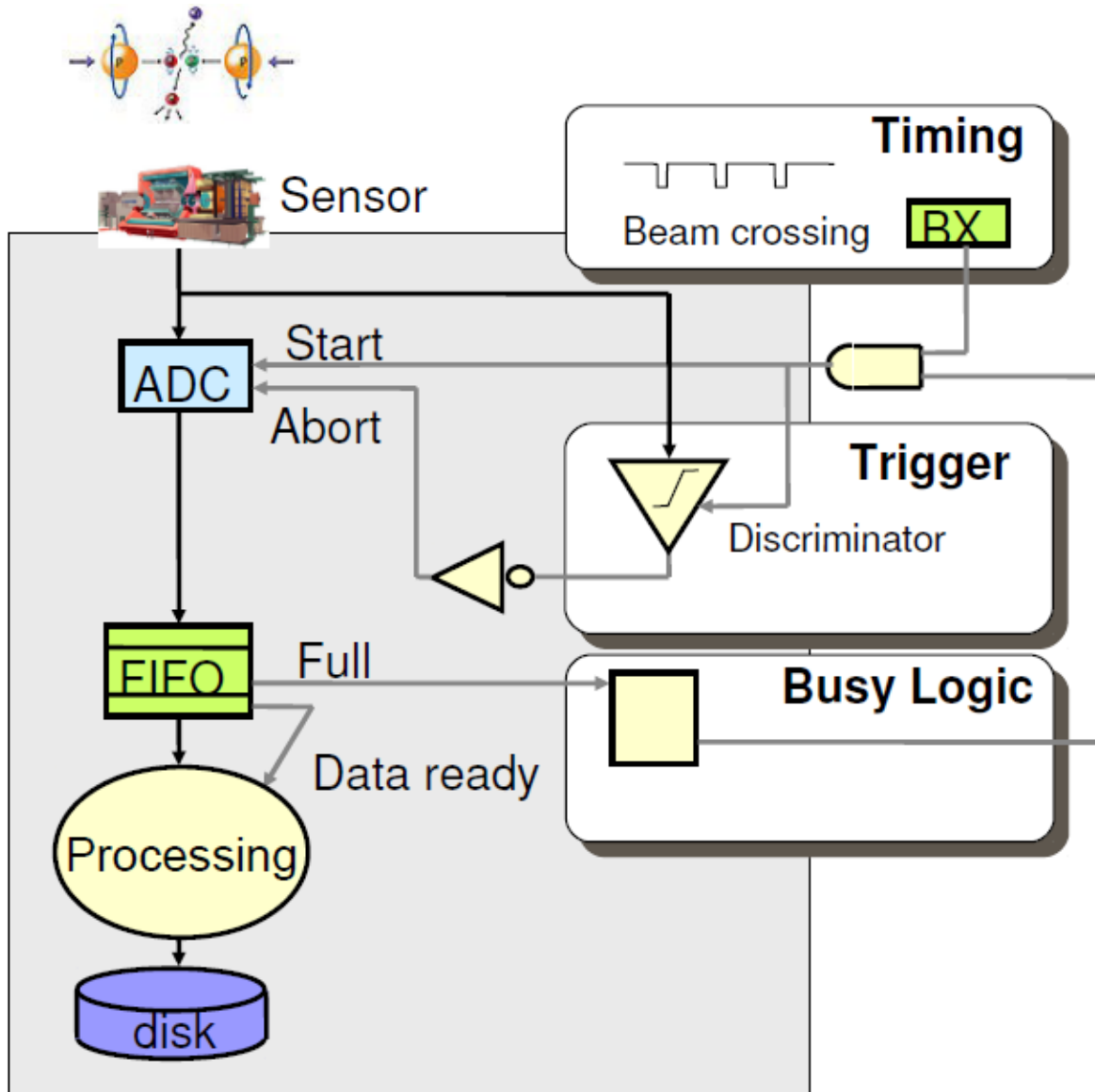


- Almost 100% efficiency and minimal deadtime if
 - ADC is able to operate at rate $\gg f$
 - Data processing and storing operates at $\sim f$
- Minimises the amount of “unnecessary” fast components
- Could the delay be replaced with a “FIFO”?
 - Analog pipelines \rightarrow Heavily used in LHC DAQs

- Let's have a closer look at the DAQ at a collider

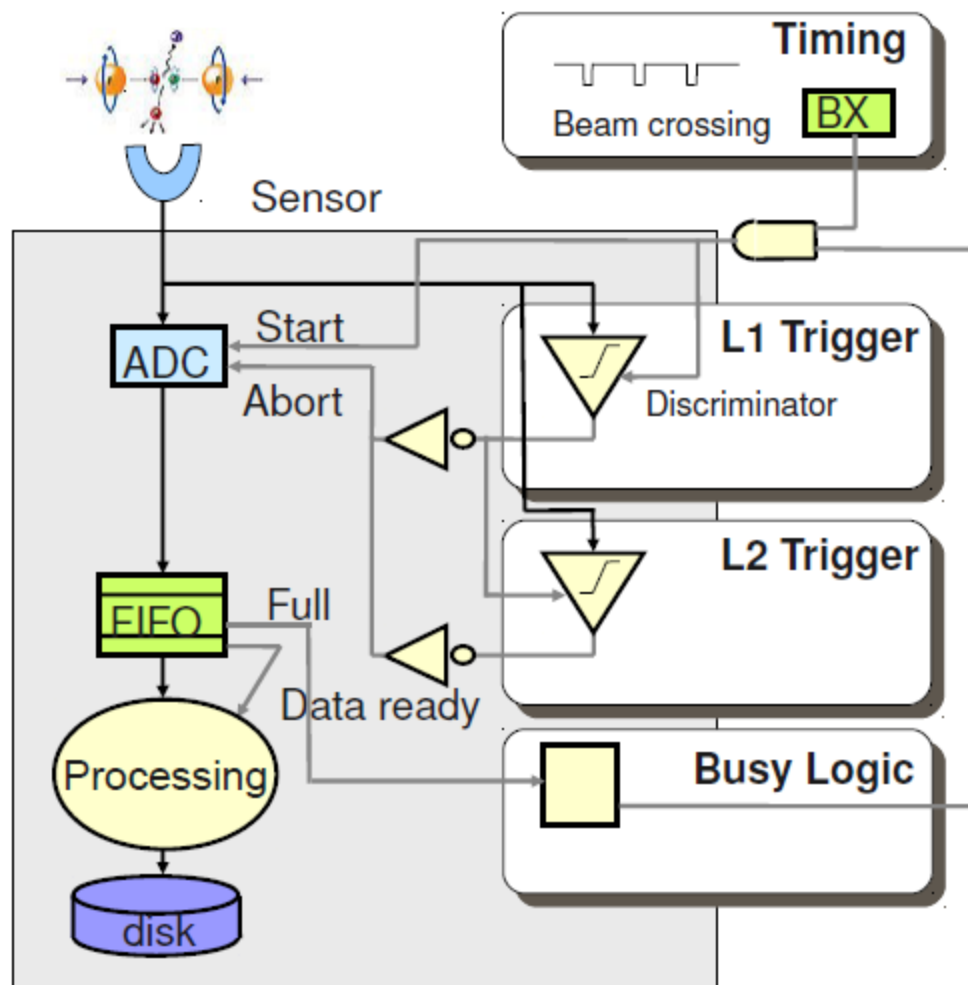


DAQ: Collider mode



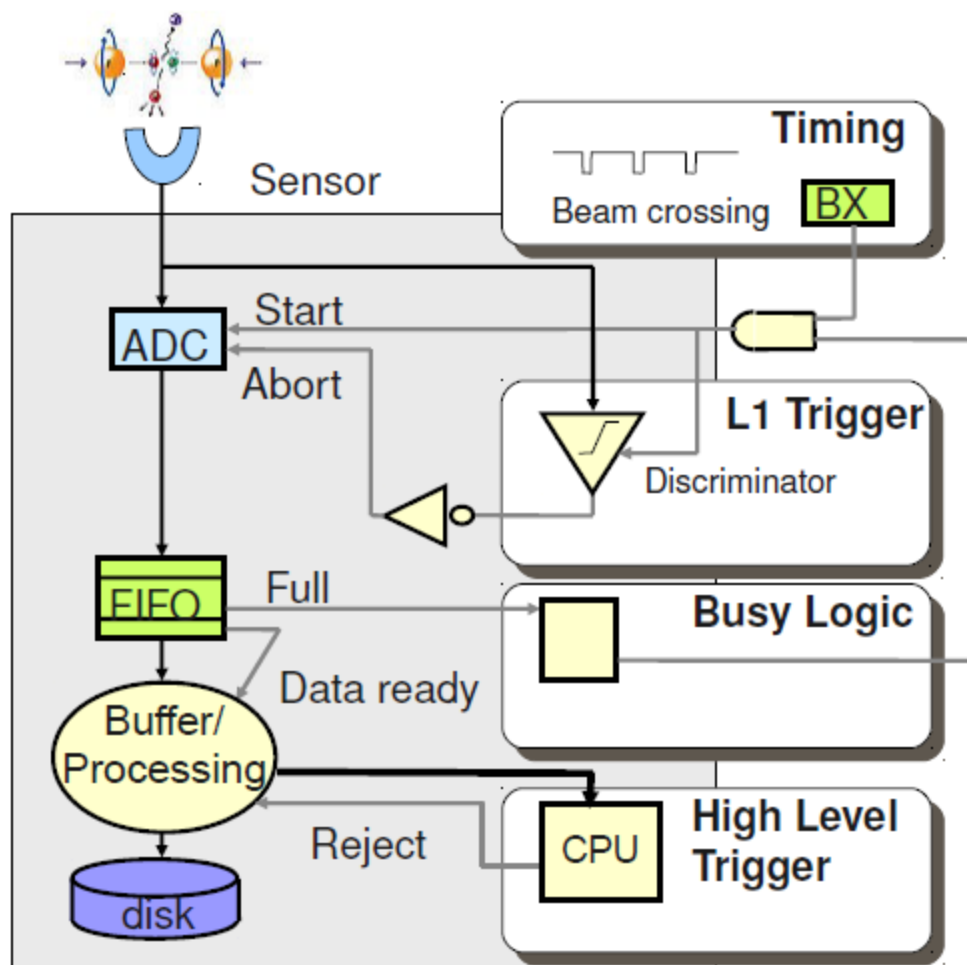
- Particle collisions are synchronous
- Trigger rejects uninteresting events
- Even if collisions are synchronous, the triggers (interesting events) are unpredictable
- Derandomisation is still needed
- No trigger deadtime if trigger latency below time between two beam crossings

Multi-Level Trigger



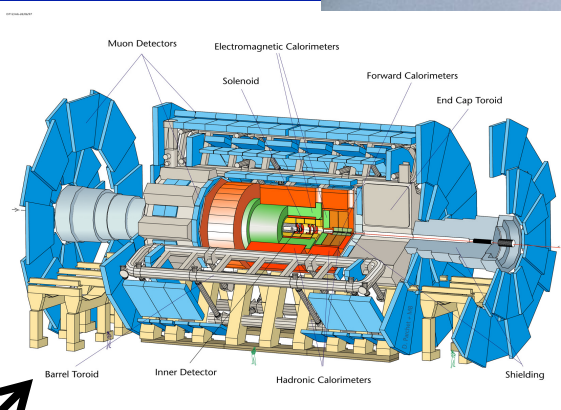
- For complicated triggers latency can be long
 - if $\tau_{\text{trig}} > \tau_{\text{BX}}$,
deadtime > 50%
- Split trigger in several levels with increasing complexity and latency
- All levels can reject events
 - with $\tau_{L1} < \tau_{\text{BX}}$, trigger
deadtime only
 $\forall L1 \cdot \tau_{L2}$

Multi-Level Trigger

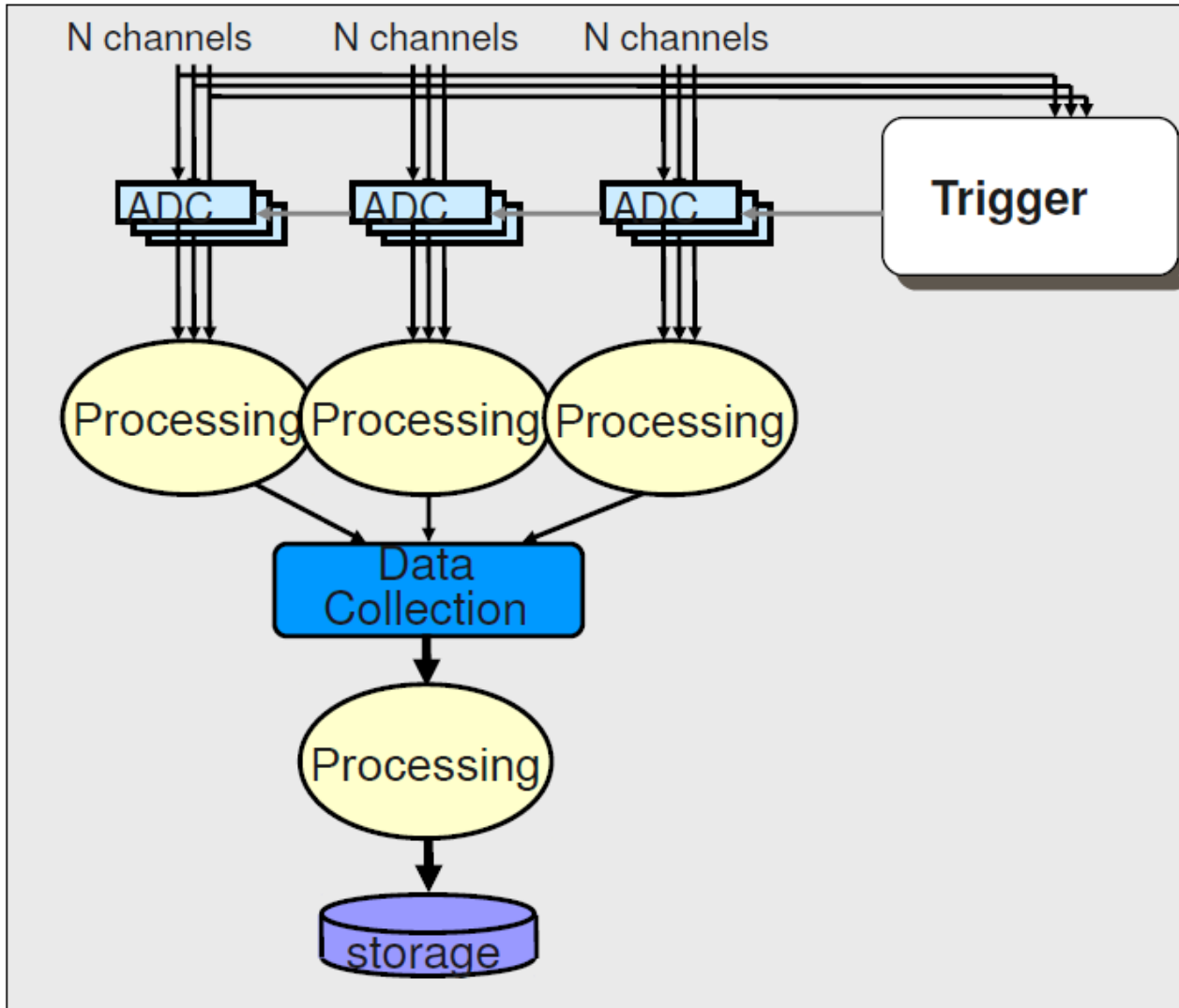


- For optimal data reduction can add trigger level between readout and storage (High-level trigger)
- Has access to some/all processed data

Scaling up

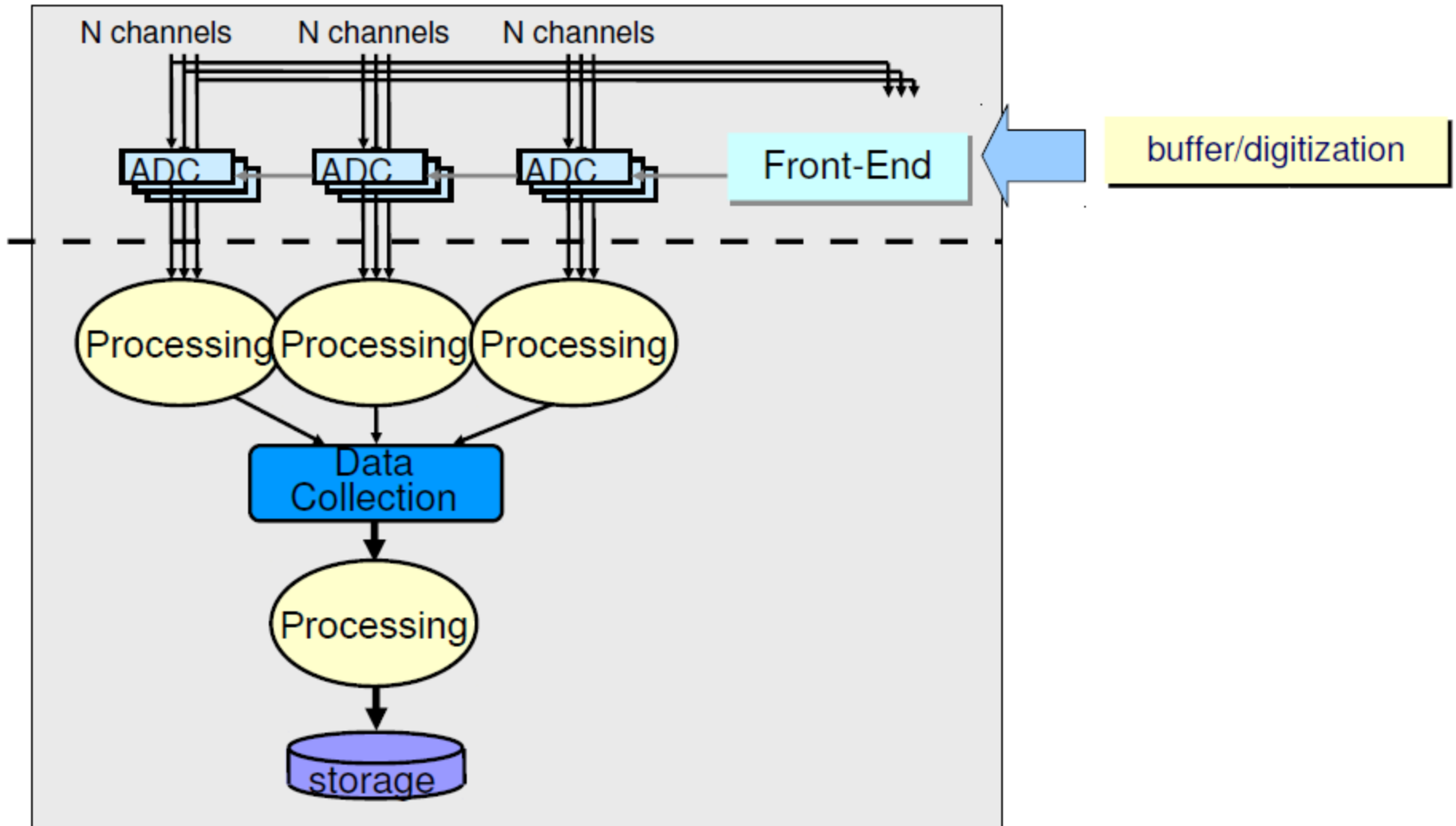


A bit more complicated....

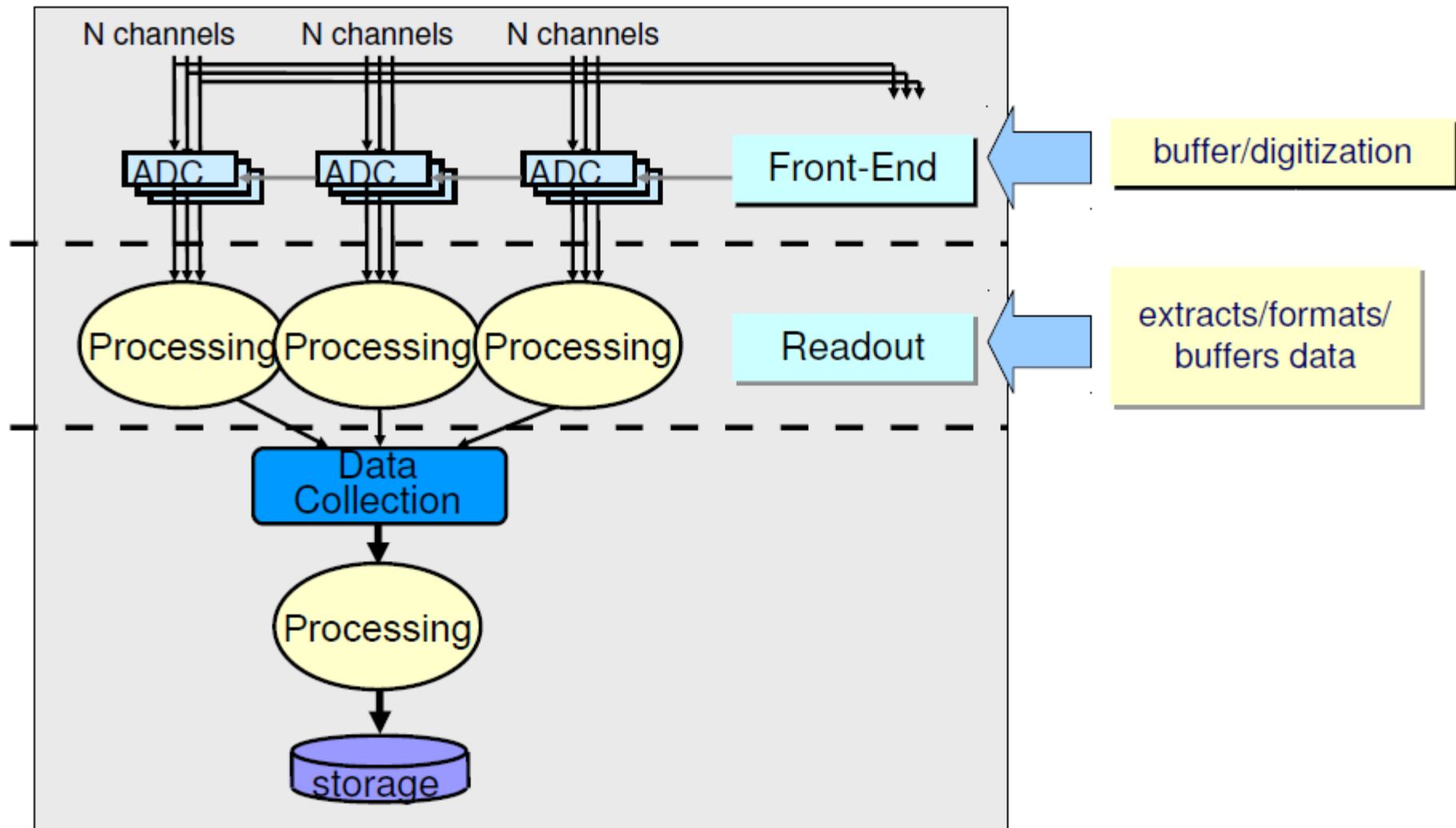


- The increased number of channels require hierarchical structure with well defined interfaces between components

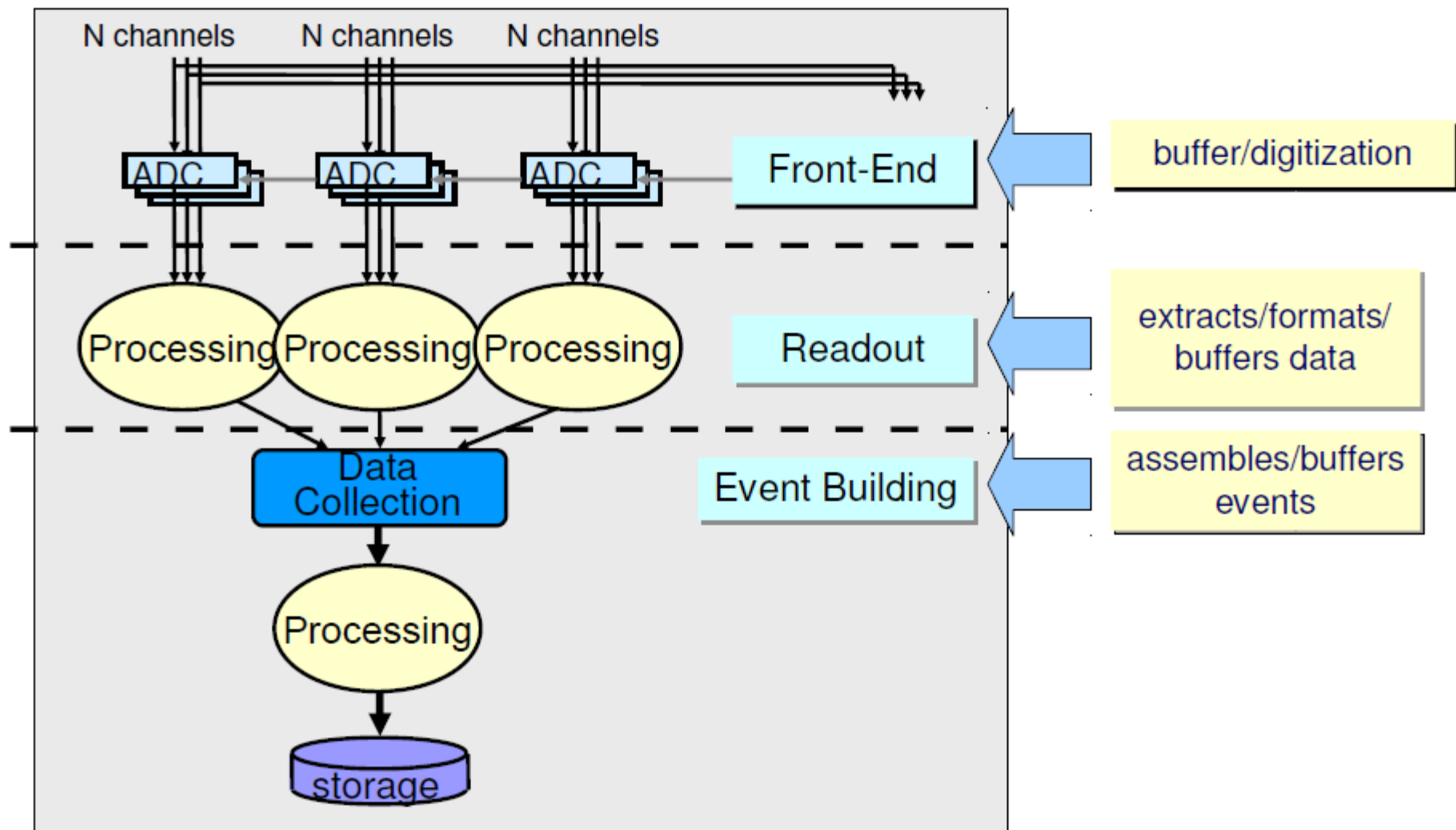
Large DAQ: constituents



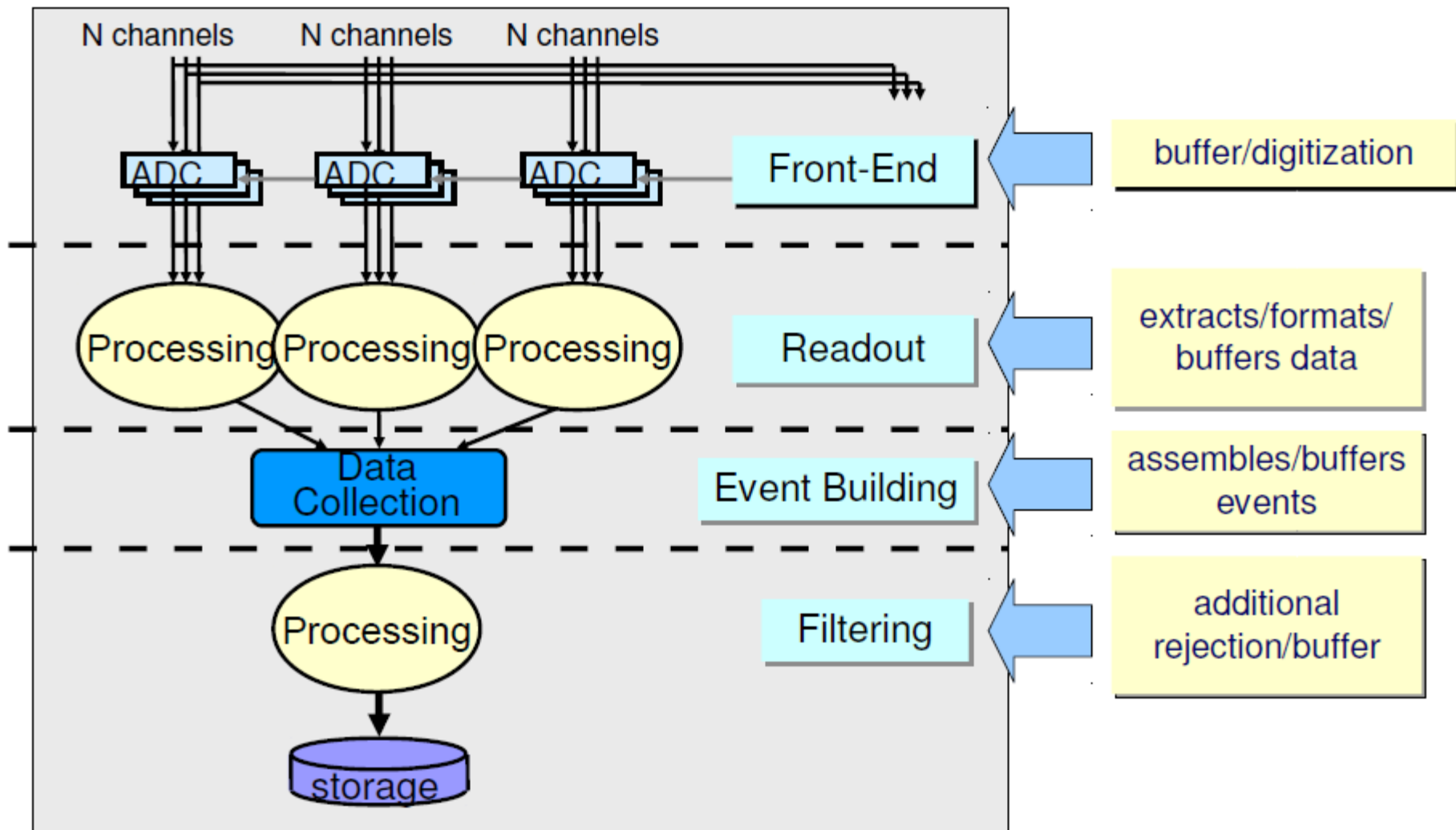
Large DAQ: constituents



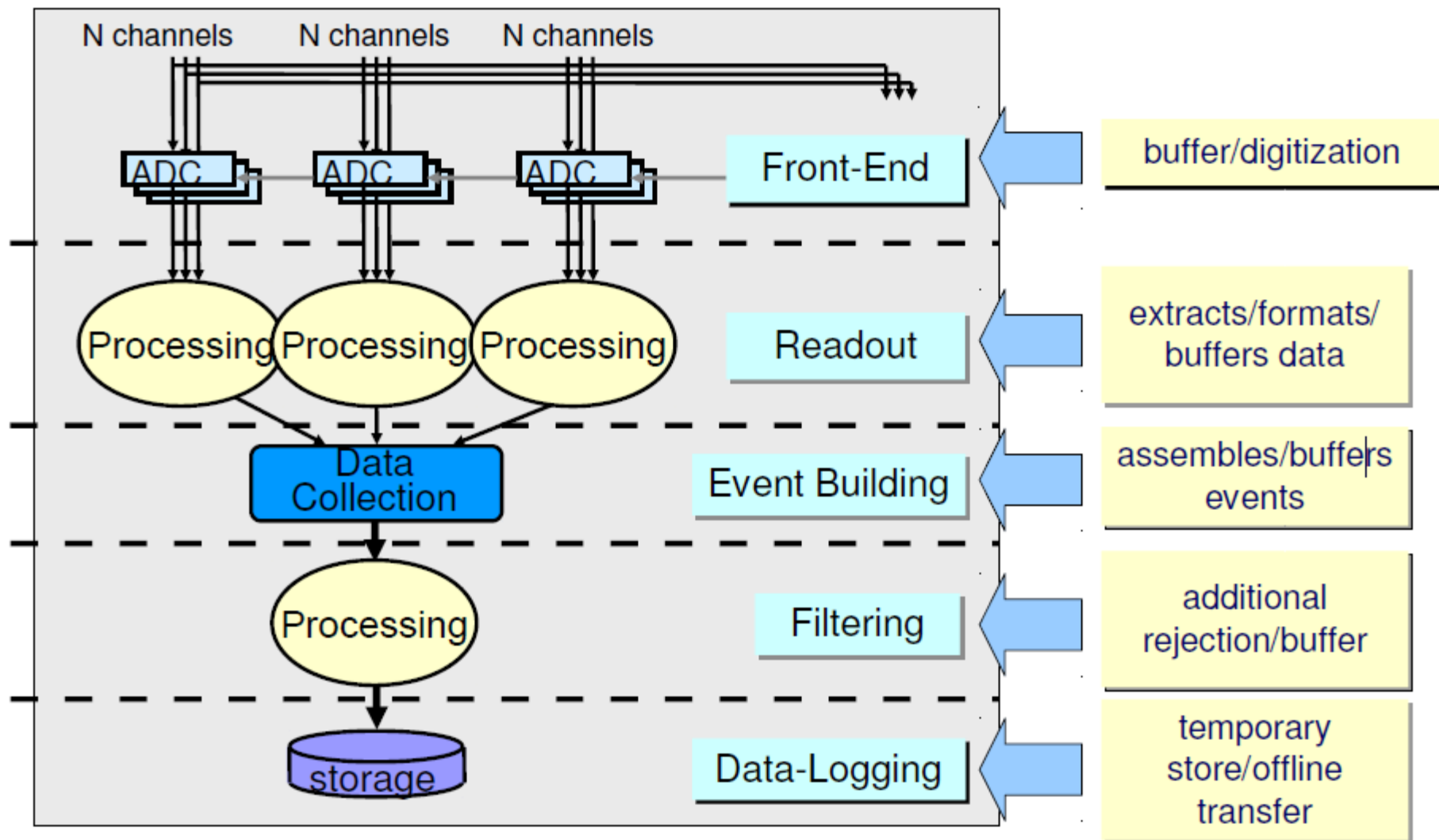
Large DAQ: constituents



Large DAQ: constituents

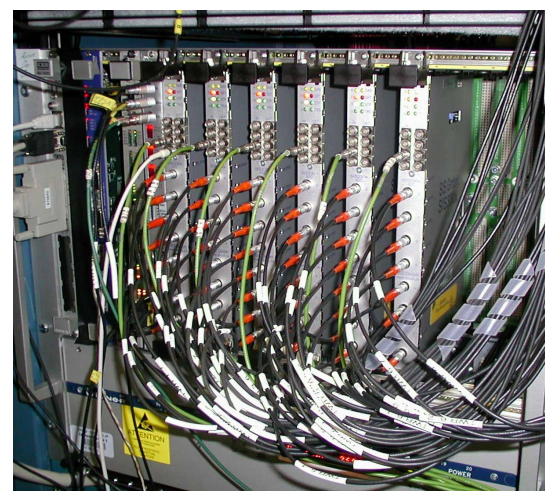
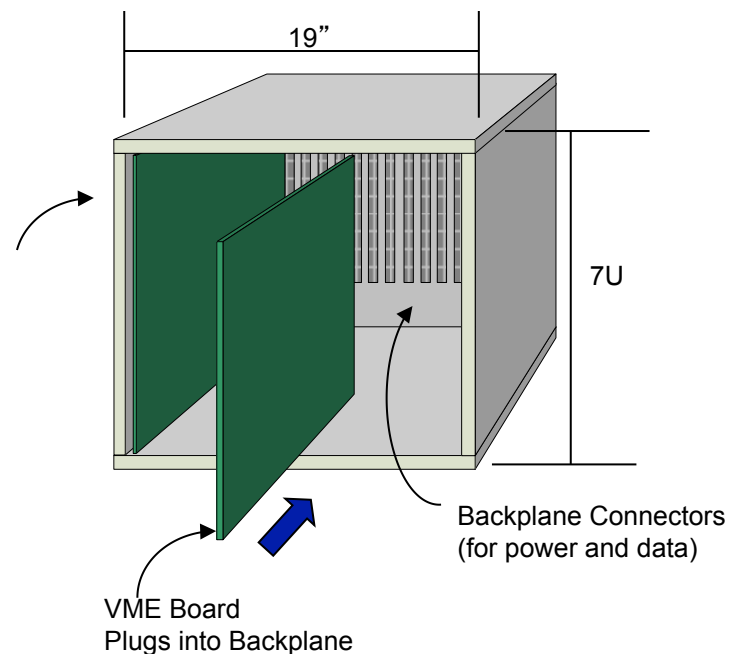


Large DAQ: constituents



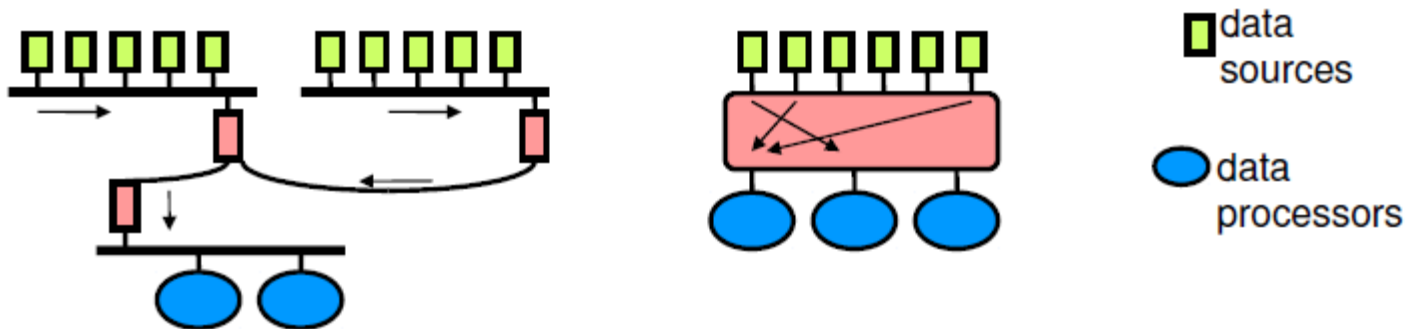
Read-out Topology

- ❖ Reading out = building events out of many detector channels
- ❖ We define “building blocks”
 - ❖ Example: readout crates, event building nodes, ...
- ❖ Crate: many modules put in a common chassis which provides
 - ❖ Mechanical support
 - ❖ Power
 - ❖ A standardised way to access the data
 - ❖ Provides signal and protocol standard for communication
- ❖ All this is provided by standards for (readout) electronics such as **VME** (IEEE 1014)



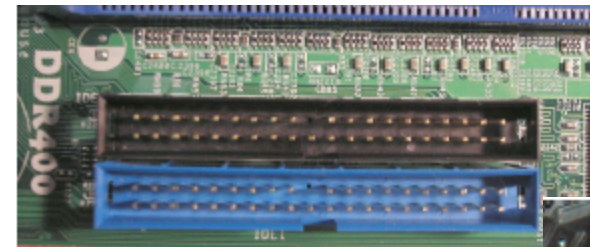
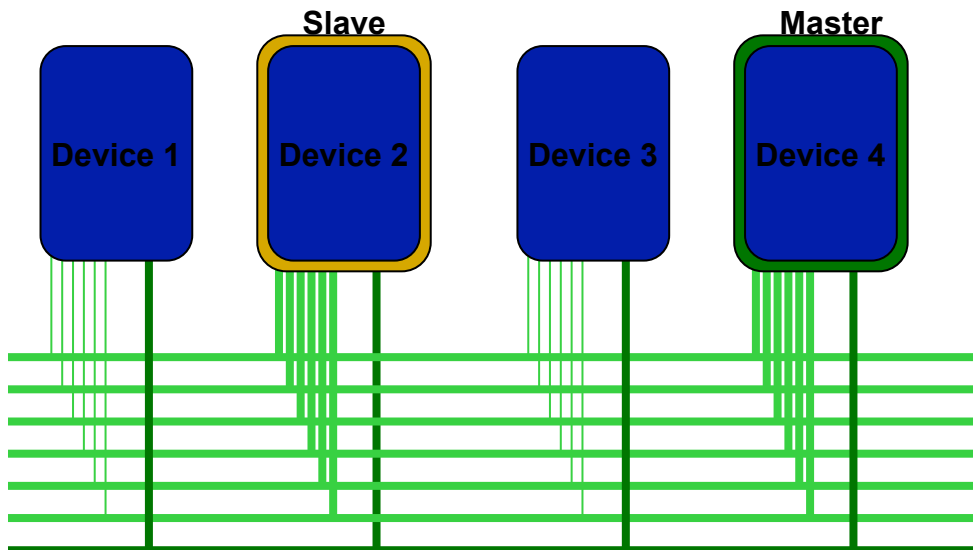
Read-out Topology

- How to organize the interconnections inside the building blocks and between building blocks?
- Two main classes: **bus** or **network**
 - Both of them are very generic concepts



Bus

- A bus connects two or more devices and allows them to communicate
 - Bus → group of electrical lines
- Examples: VME, PCI, SCSI, Parallel ATA, ...
- The bus is **shared** between all devices on the bus → arbitration is required
- Devices can be **masters** or **slaves** (some can be both)
- Devices can be uniquely identified ("**addressed**") on the bus



Data Lines

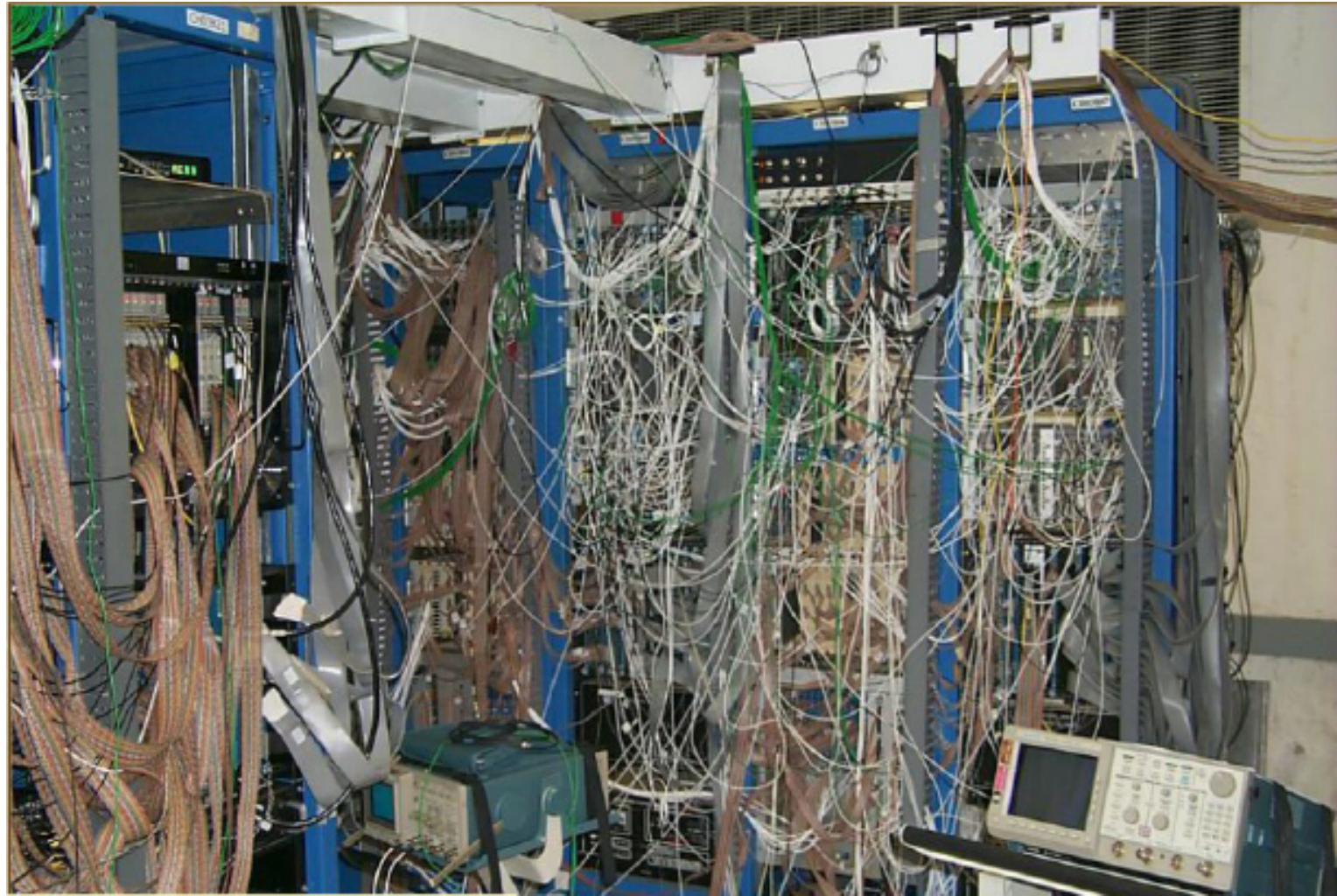
Select Line

Bus

- ☺ Relatively simple to implement
 - 🐾 Constant number of lines
 - 🐾 Each device implements the same interface
 - ➔ Easy to add new devices
- ☹ Scalability issues
 - 🐾 Number of devices and physical bus-length is limited
 - 🐾 Each new active device slows everybody down as bus bandwidth* shared among all the devices
 - 🐾 Maximum bus width is limited (128 bit for PC-system bus)
 - 🐾 Maximum bus frequency (number of elementary operations per second) is inversely proportional to the bus length
- 🐾 Typical buses have a lot of control, data and address lines (e.g. SCSI cable (Small Computer System Interface))
- 🐾 Buses are typically useful for systems $< 1 \text{ GB/s}$

Bandwidth = amount of data transferred / per unit of time (measured in Bytes/h)

Bus: another limitation



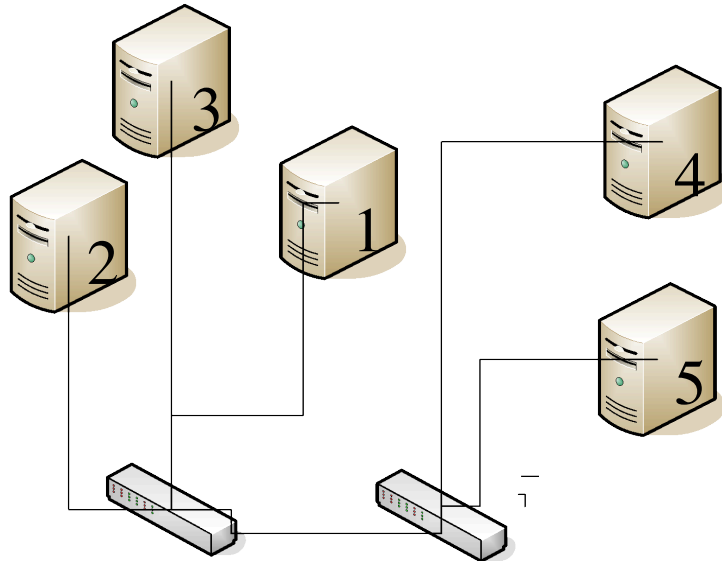
Network based DAQ

- ❖ In large (HEP) experiments we typically have thousands of devices to read, which are sometimes very far from each other
→ *buses can not do that*
- ❖ Network technology solves the scalability issues of buses
 - ❖ Examples: Ethernet, Telephone, Infiniband, ...
 - ❖ Devices are equal ("peers")
 - ❖ They communicate directly with each other by sending messages
 - ❖ no arbitration necessary
 - ❖ bandwidth guaranteed
 - ❖ data and control use the same path
 - ❖ much fewer lines (e.g. in traditional Ethernet only two)
 - ❖ On an network a device is identified by a **network address**
 - ❖ eg: phone-number, MAC address
 - ❖ At the signaling level buses tend to use parallel copper lines. Network technologies can be also optical or wire-less



Switched Networks

- Modern networks are *switched with point-to-point links*
- Each node is connected either to another node or to a **switch**
- Switches can be connected to other switches
- A path from one node to another leads through 1 or more switches
- Switches move messages between sources and destinations
 - Find the right path
 - Handle “congestion” (two messages with the same destination at the same time)



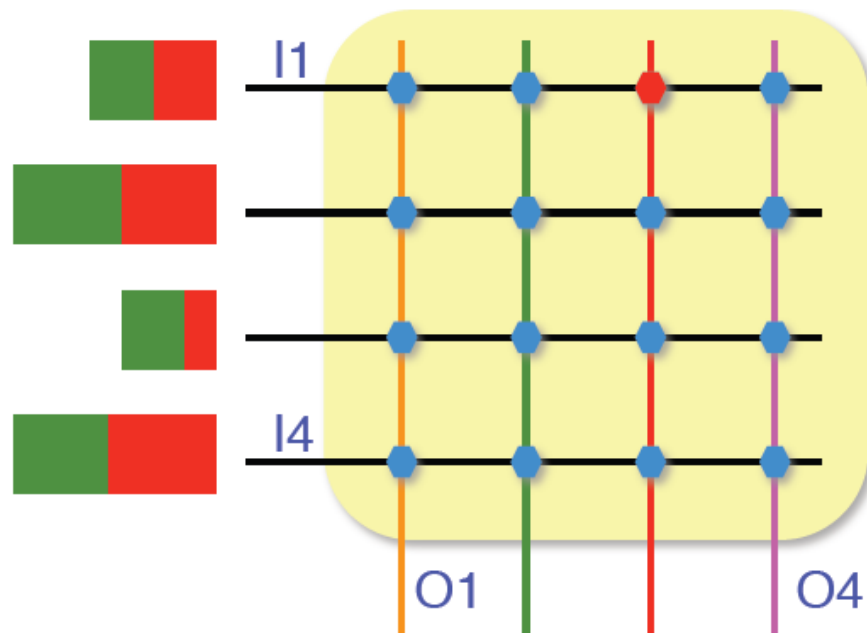
• Example

- While 2 can send data to 1 and 4, 3 can send at full speed to 5
- 2 can distribute the bandwidth between 1 and 4 as needed

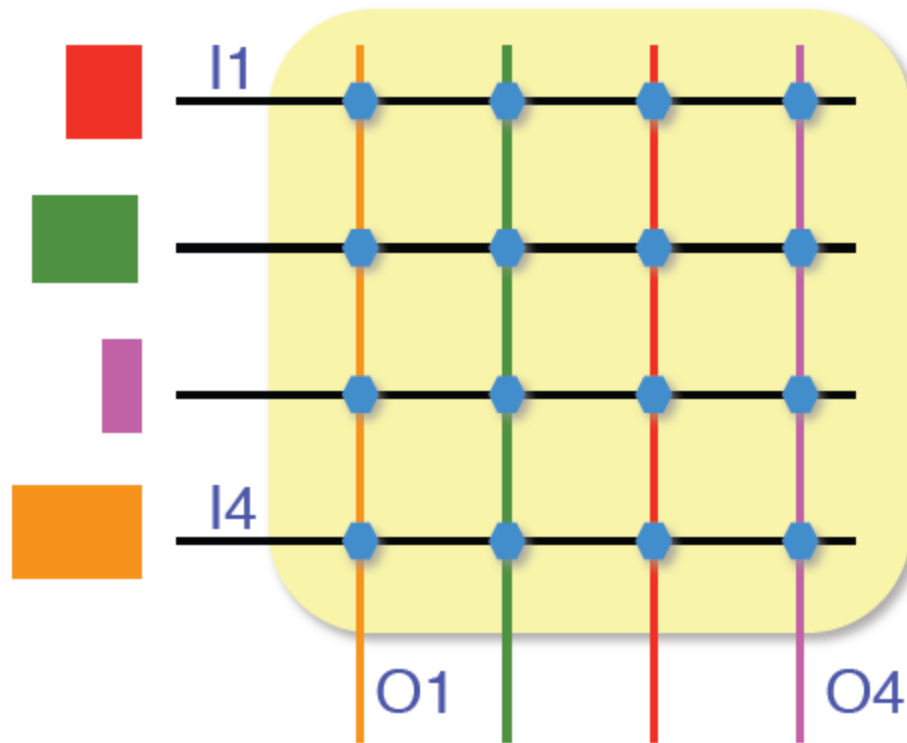
Switched Network

Challenge

- Find the right path
- Handle “congestion” (two messages with the same destination at the same time)



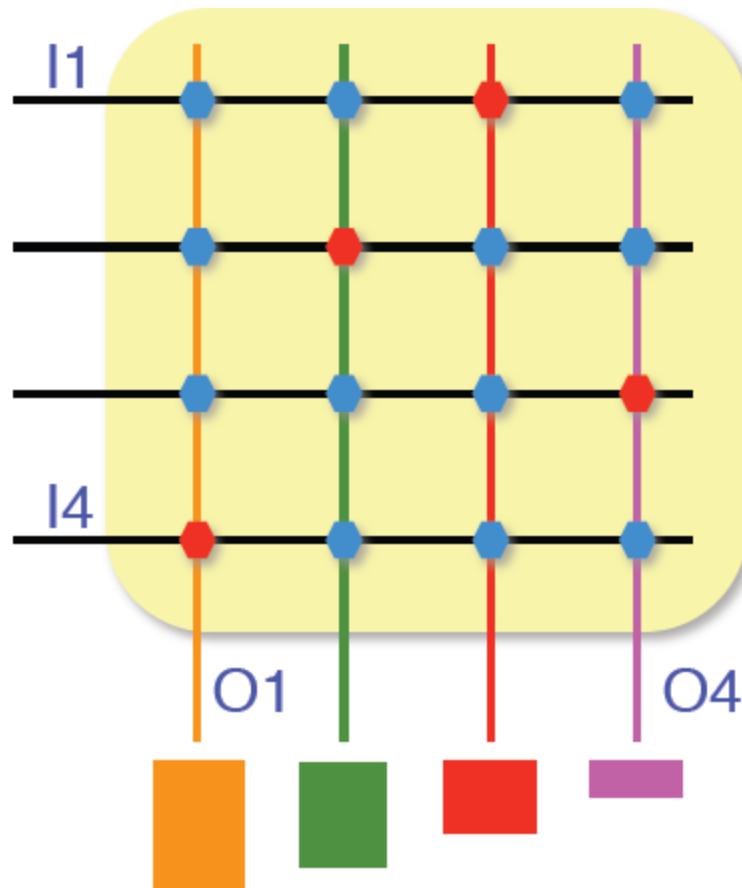
Switch implementation: cross-bar



😊 Paradise scenario:

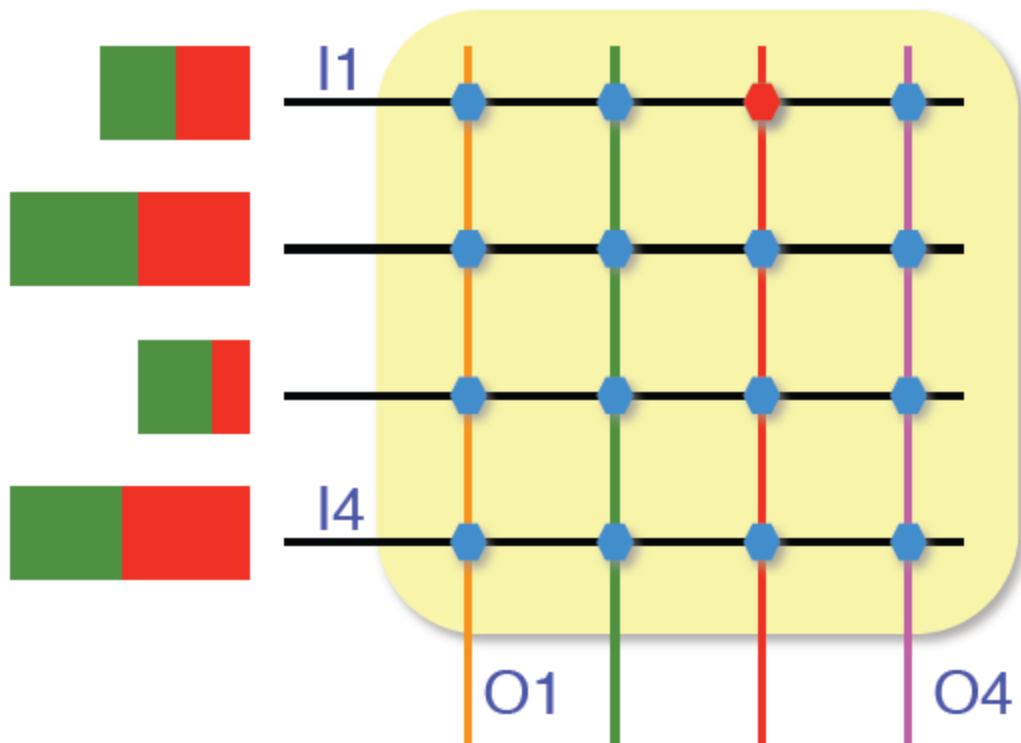
- 🐾 No congestion, since every data package finds a free path through the switch.

Switch implementation: cross-bar



- 😊 Paradise scenario:
 - 🐾 No congestion, since every data package finds a free path through the switch.

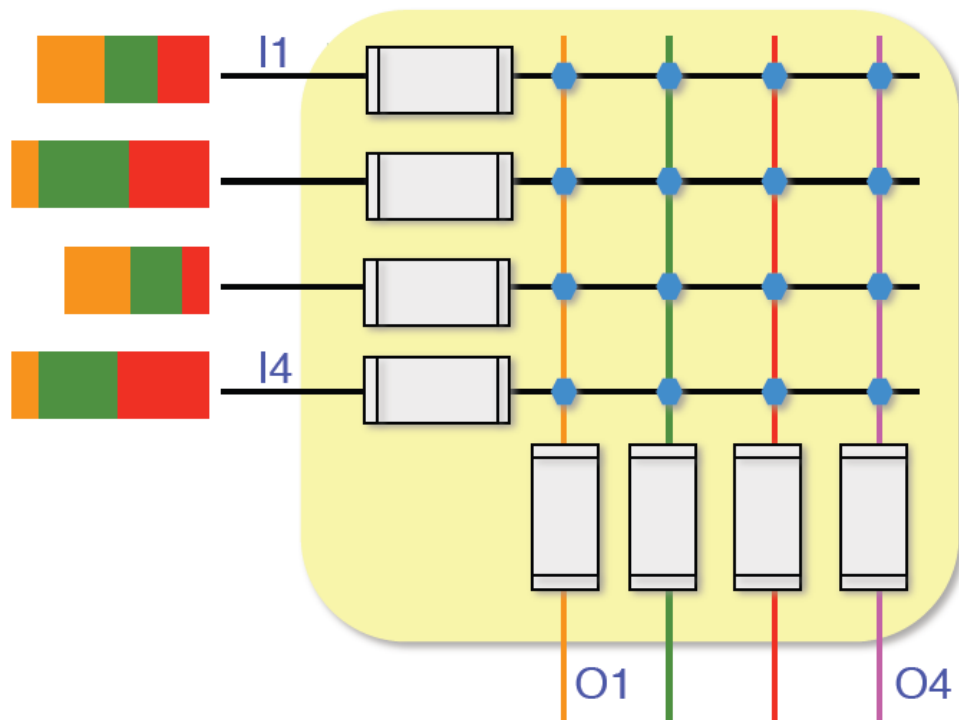
Switch implementation: cross-bar



☹ Nightmare scenario:

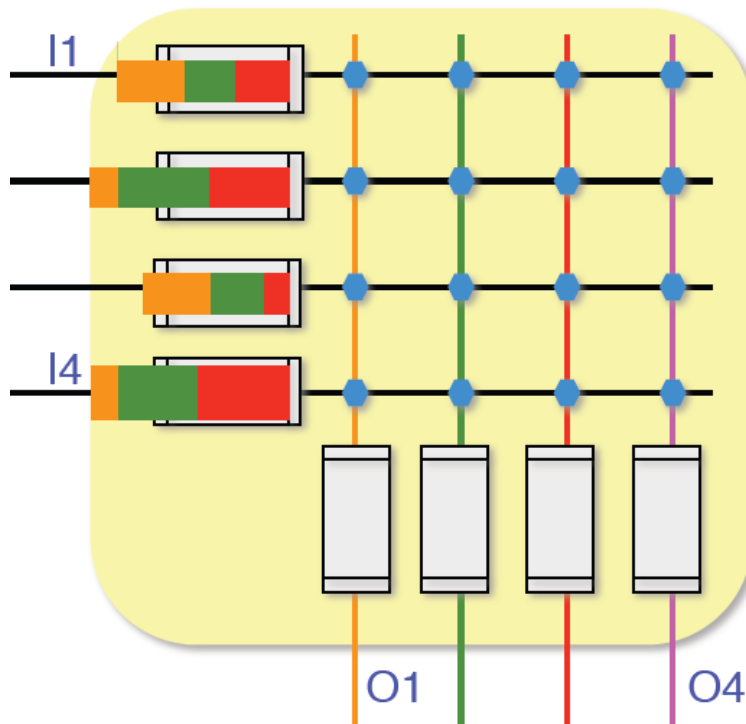
- 🐾 Only one packet at a time can be routed to the destination.
Congestion!
- 🐾 How can we avoid this?

Switch implementation



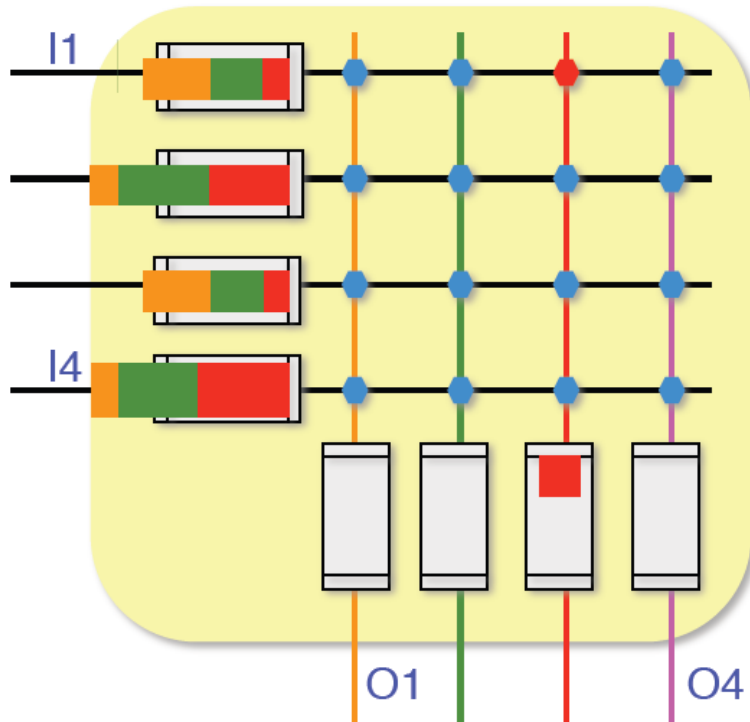
- Use the old “trick”
 - Add buffer
- FIFOs can “absorb” congestion ...until they are full.

Switch implementation

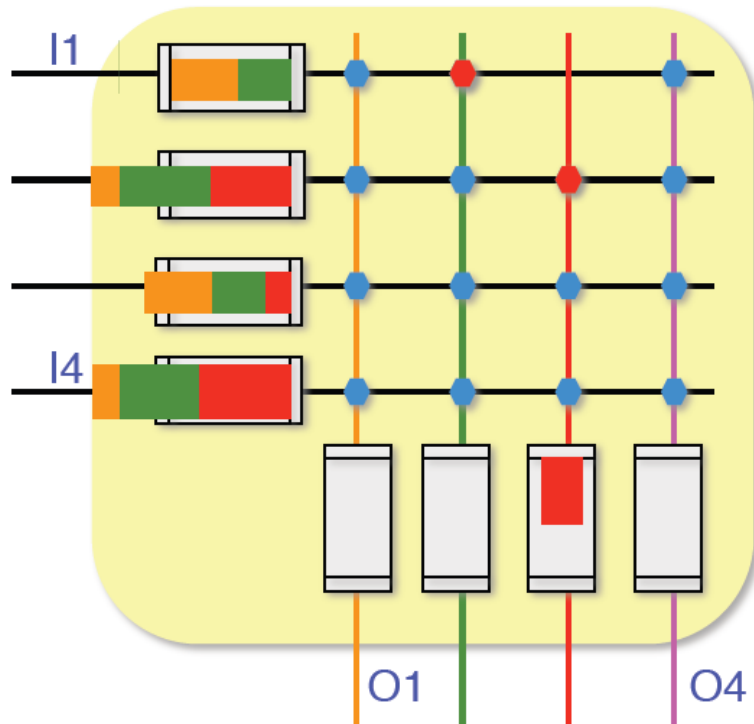


- Use the old “trick”
 - Add buffer
- FIFOs can “absorb” congestion ...until they are full.

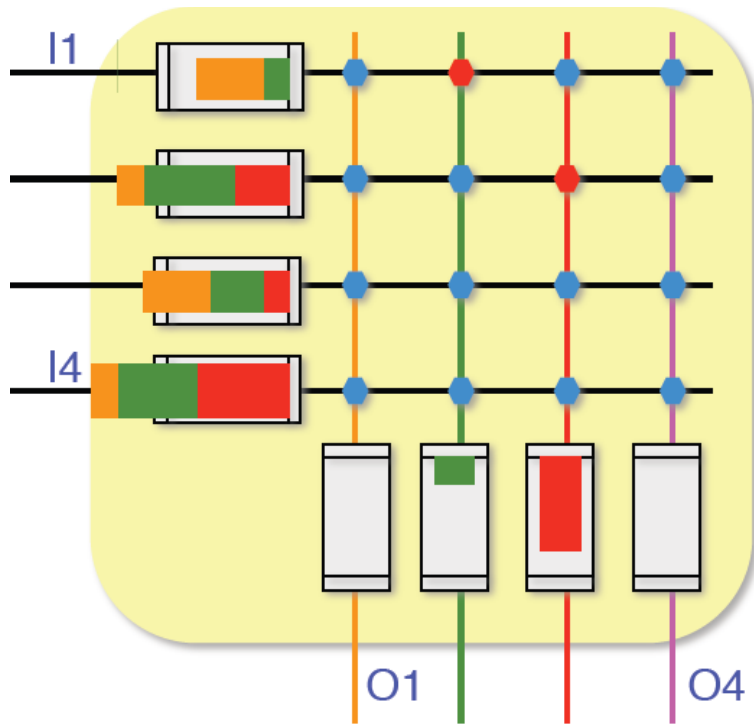
Switch implementation



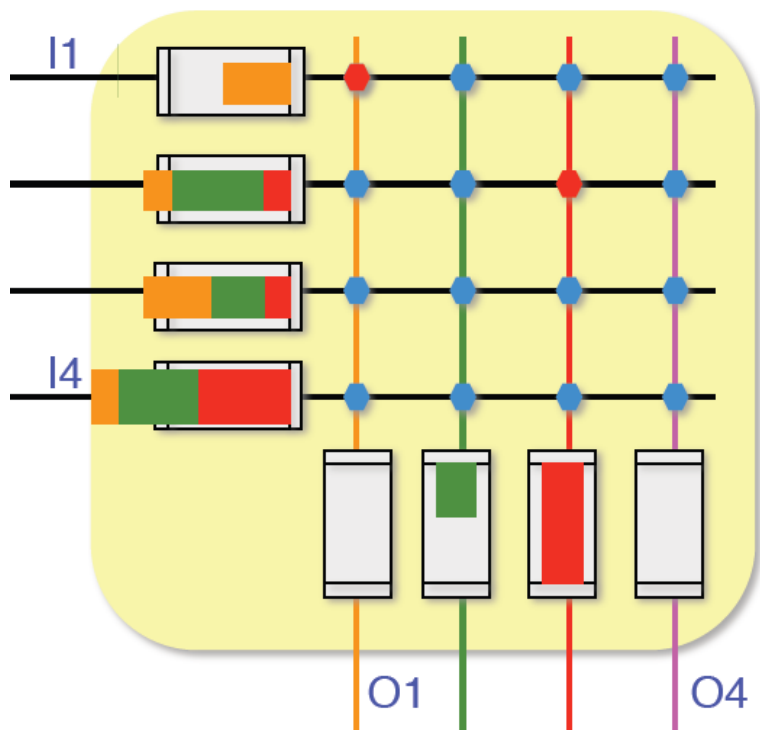
Switch implementation



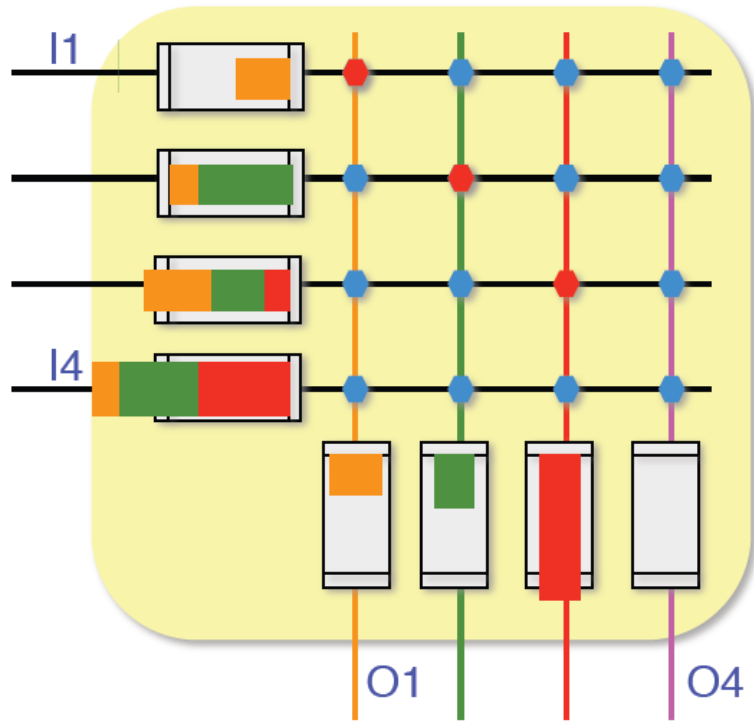
Switch implementation



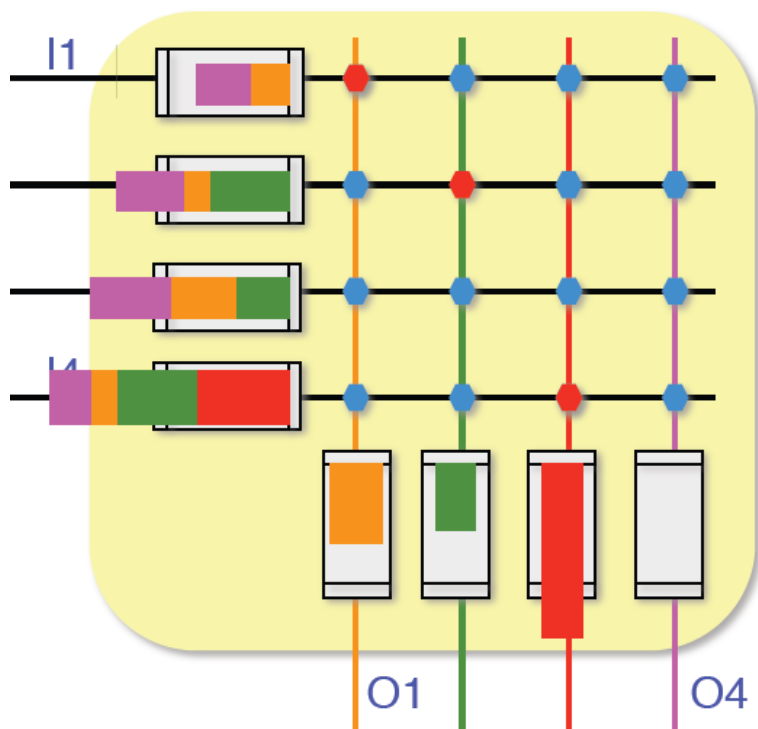
Switch implementation



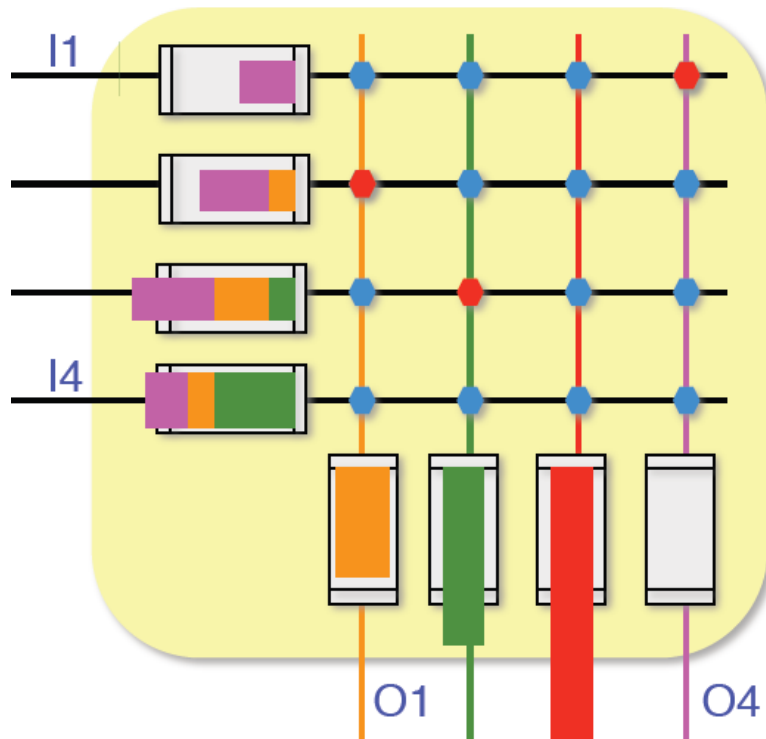
Switch implementation



Switch implementation



Switch implementation



- **Still problematic:**
- Input FIFOs can absorb data fluctuations until they are full. How good it works depends on:
 - FIFOs capacity
 - data distribution
 - Internal speed of the switch
- traffic: blocking problem remains to some extent

What we have learnt so far

- The principle of a simple data acquisition system
- Introduction to some basic elements: trigger, derandomiser, FIFO, busy logic
- How data is transported
 - Bus versus network
- In the next lecture we will look in more detail at the DAQ used by the experiments at LHC