Introduction to FPGAs

(An incomplete talk by Jahred Adelman)

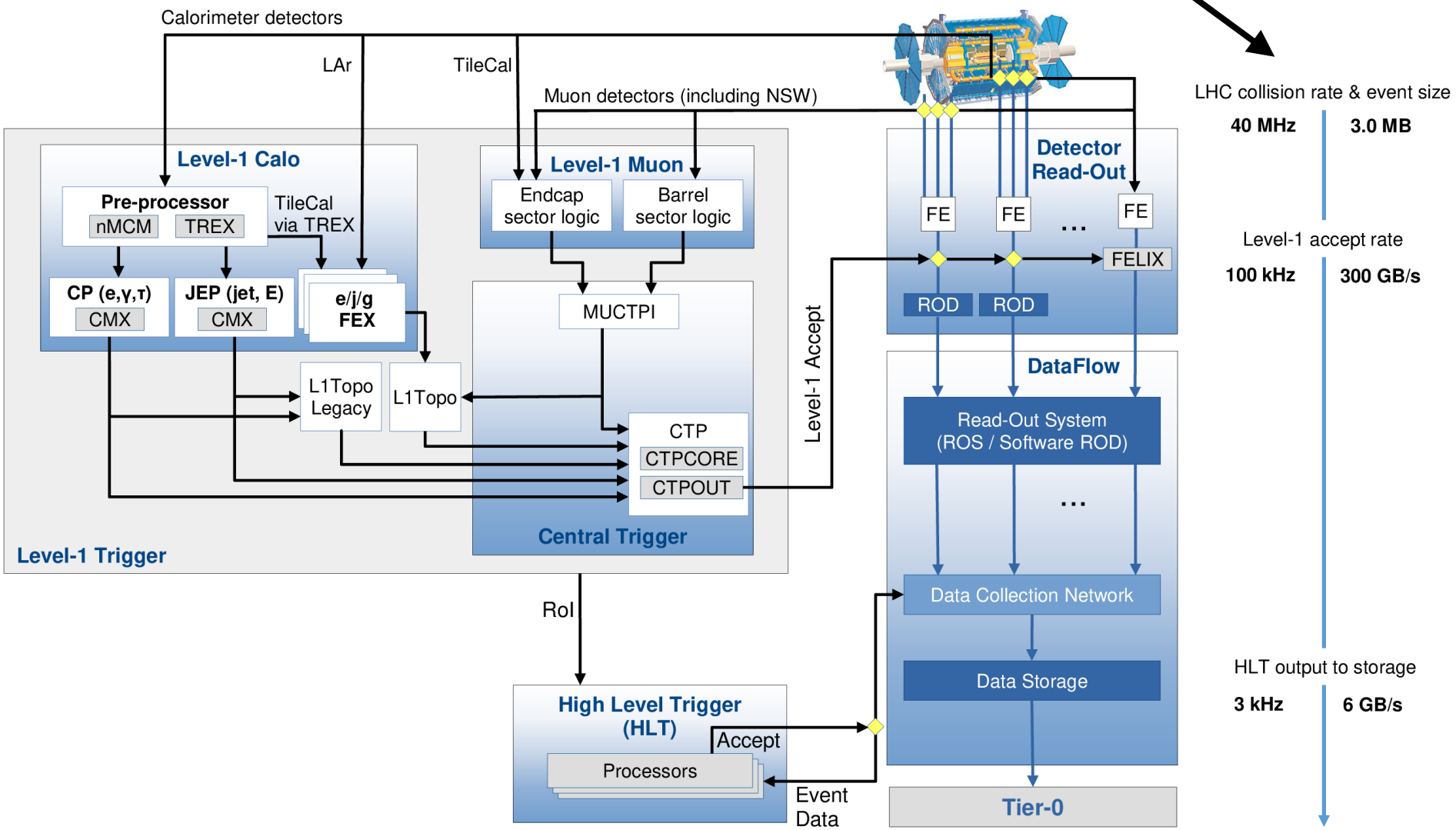
A lot of other material taken from the TAC-HEP lecture series on FPGAs: https://tac-hep.org/training-modules/uw-gpu-fpga

Electronics in modern particle physics?

Reminder - where do we use electronics for physics?
Answer: Almost everywhere!
Here is an (incomplete!) and very brief set of examples

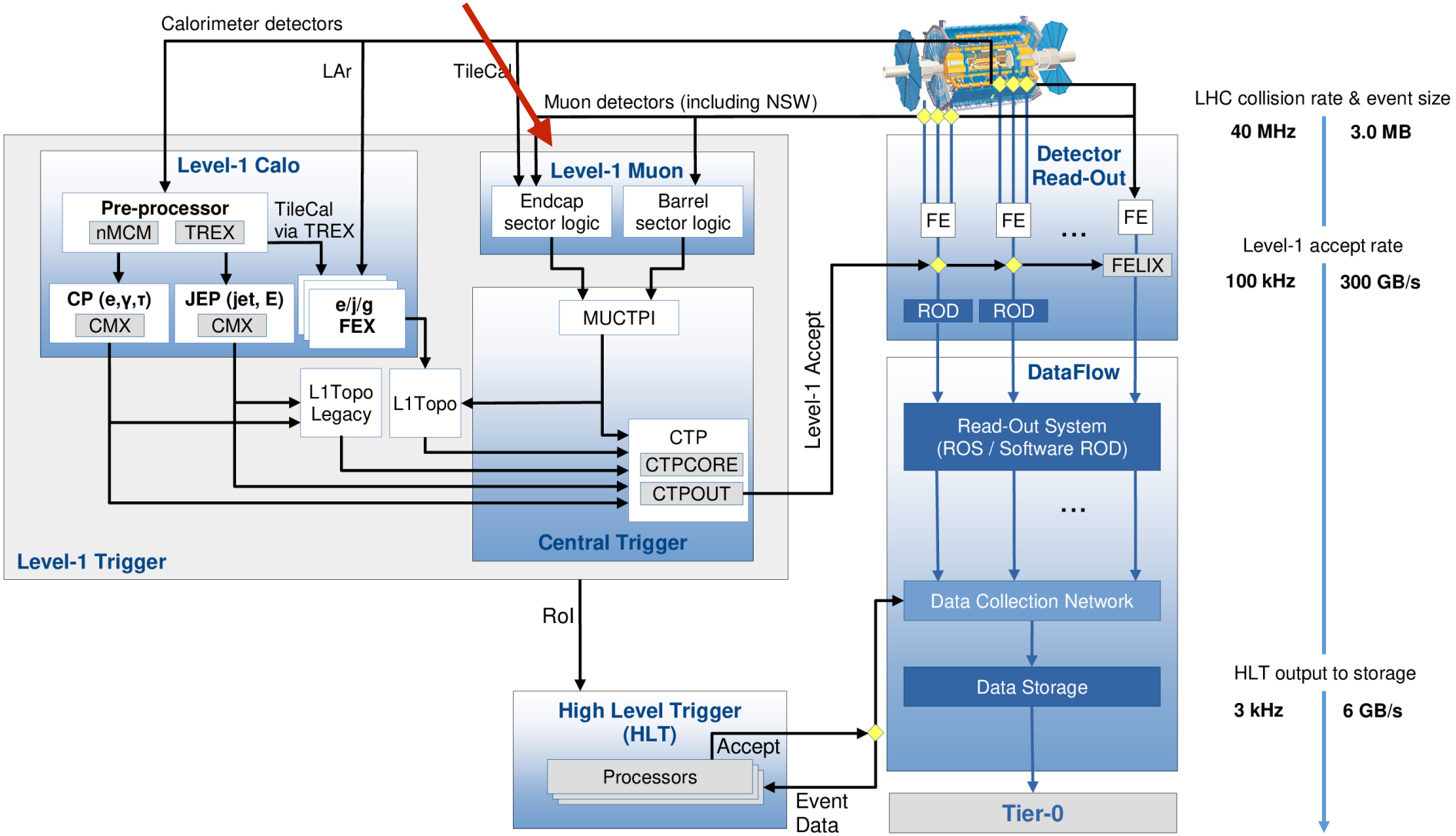
Detector readout

Bunch crossing rate of **40 MHz**. Need to **read out** portions of the detector to decide if something interesting happened



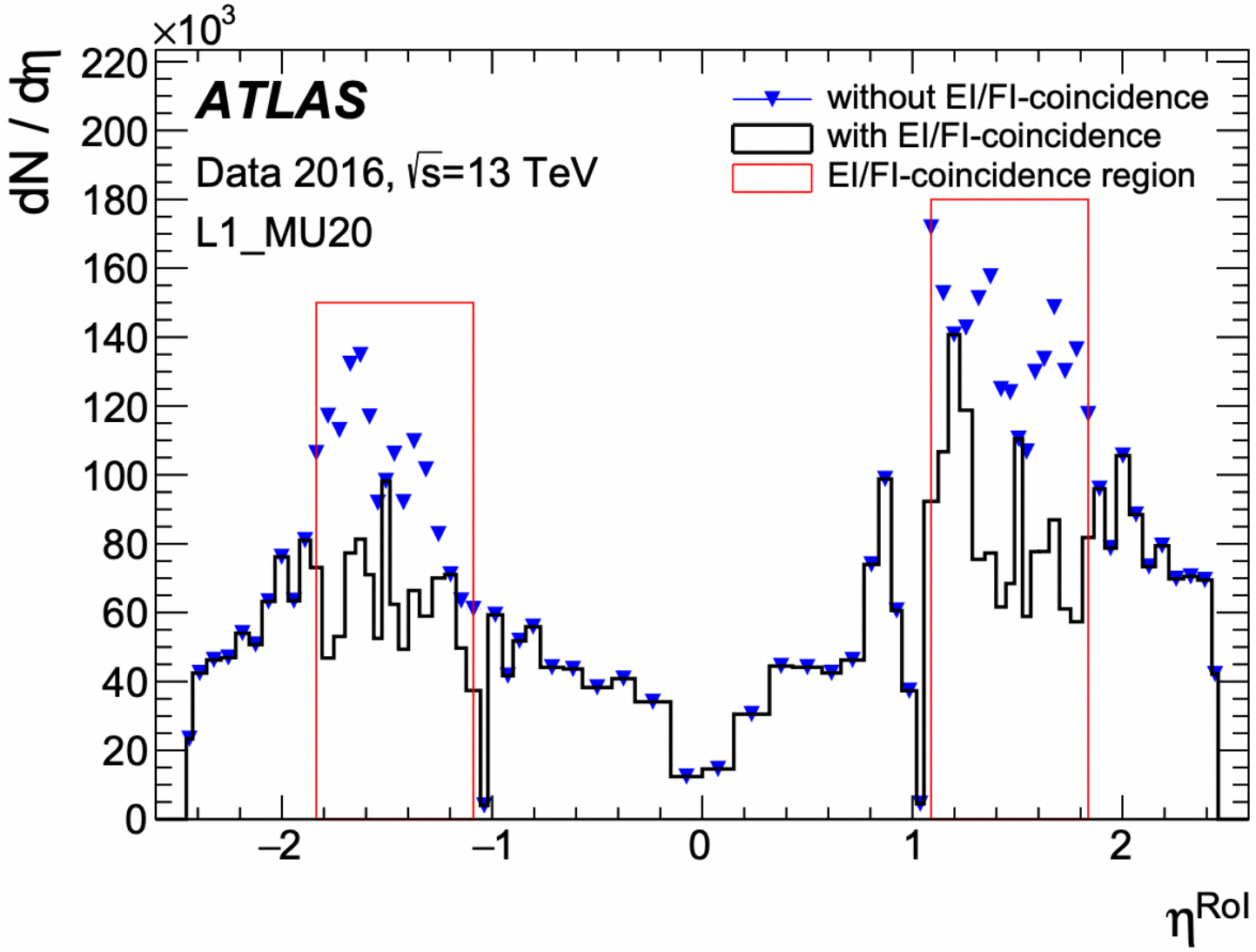
Level 1 muon trigger

Level 1 muon system makes crude attempts to find potential muons traversing the detector using information from the muons and calorimeter systems. “Simple” coincidences of hits assuming infinite p_T muons.



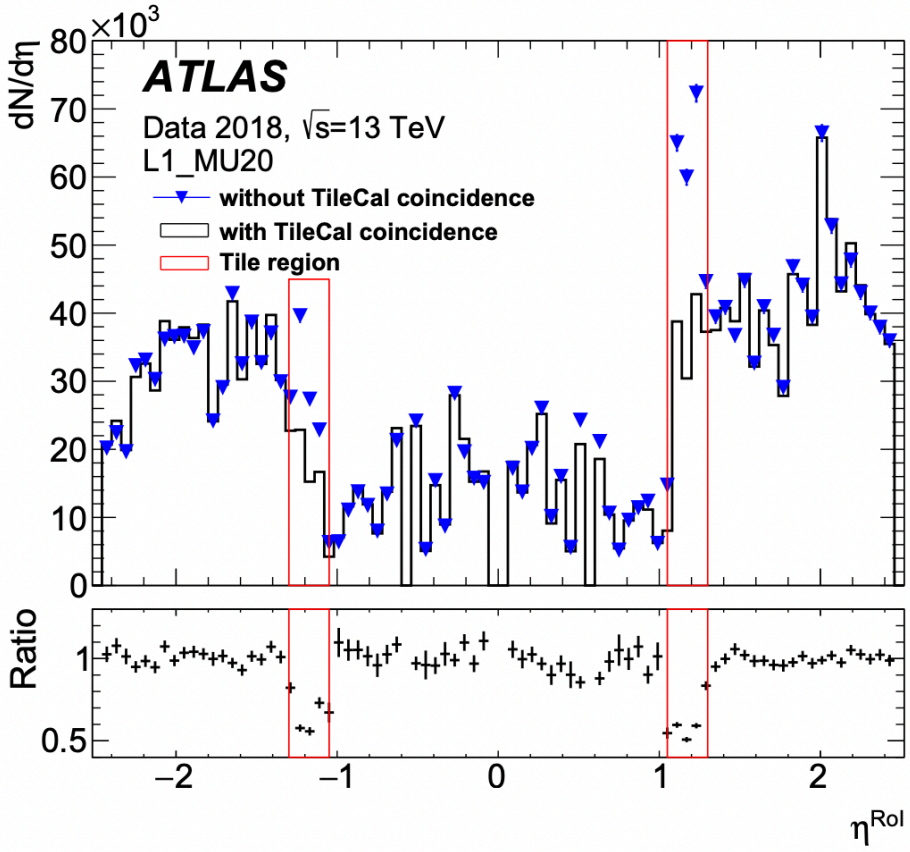
Level 1 muon trigger

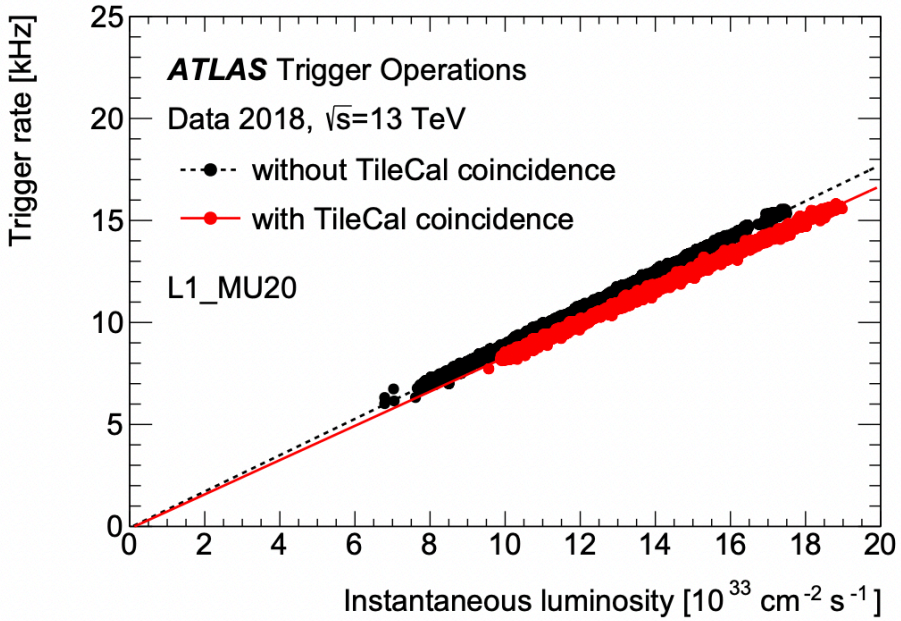
What if we want to add extra coincidences?



Level 1 muon trigger

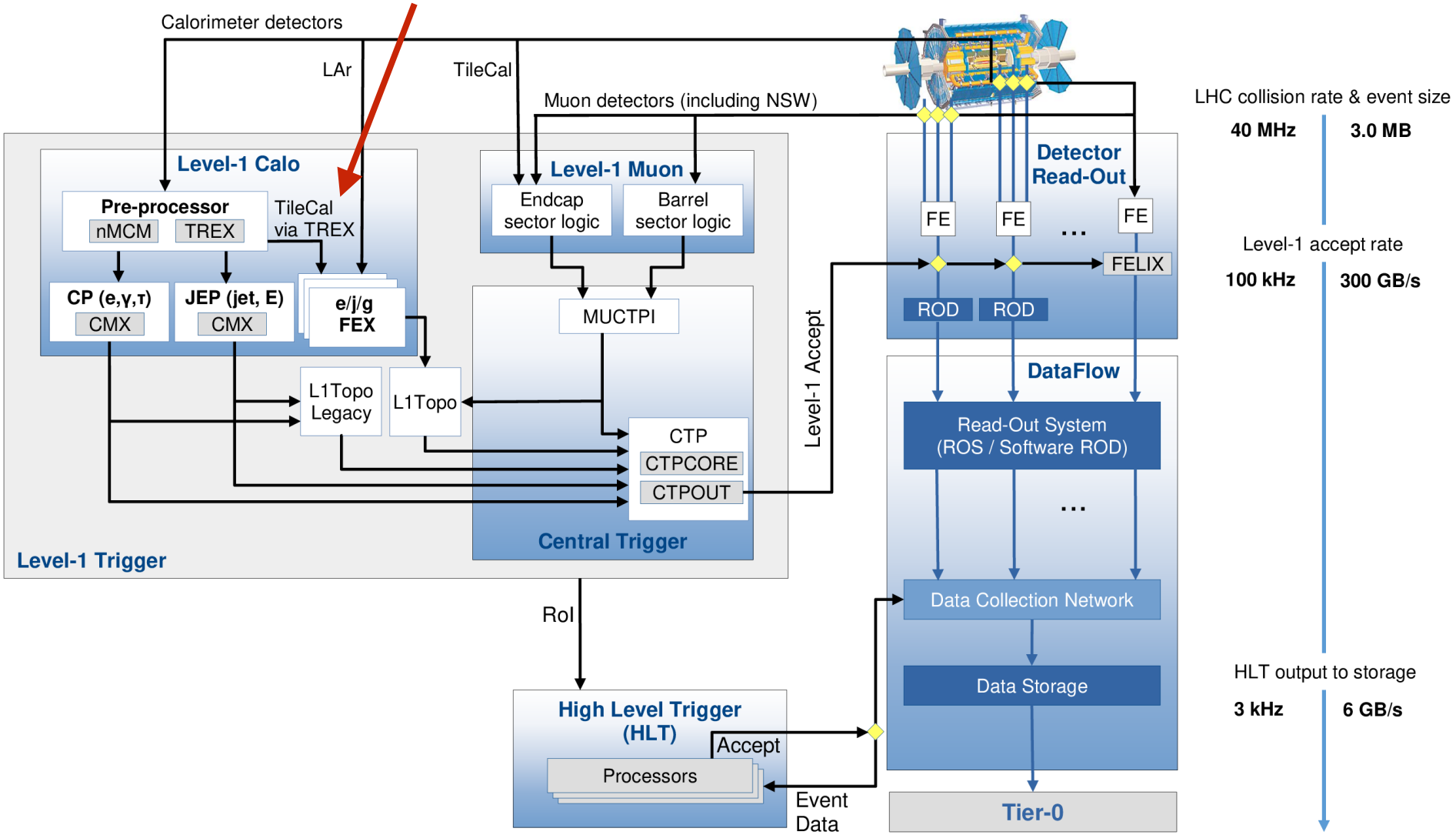
What if we want to use information from other detectors? Or what if some muon chambers are offline or noisy?





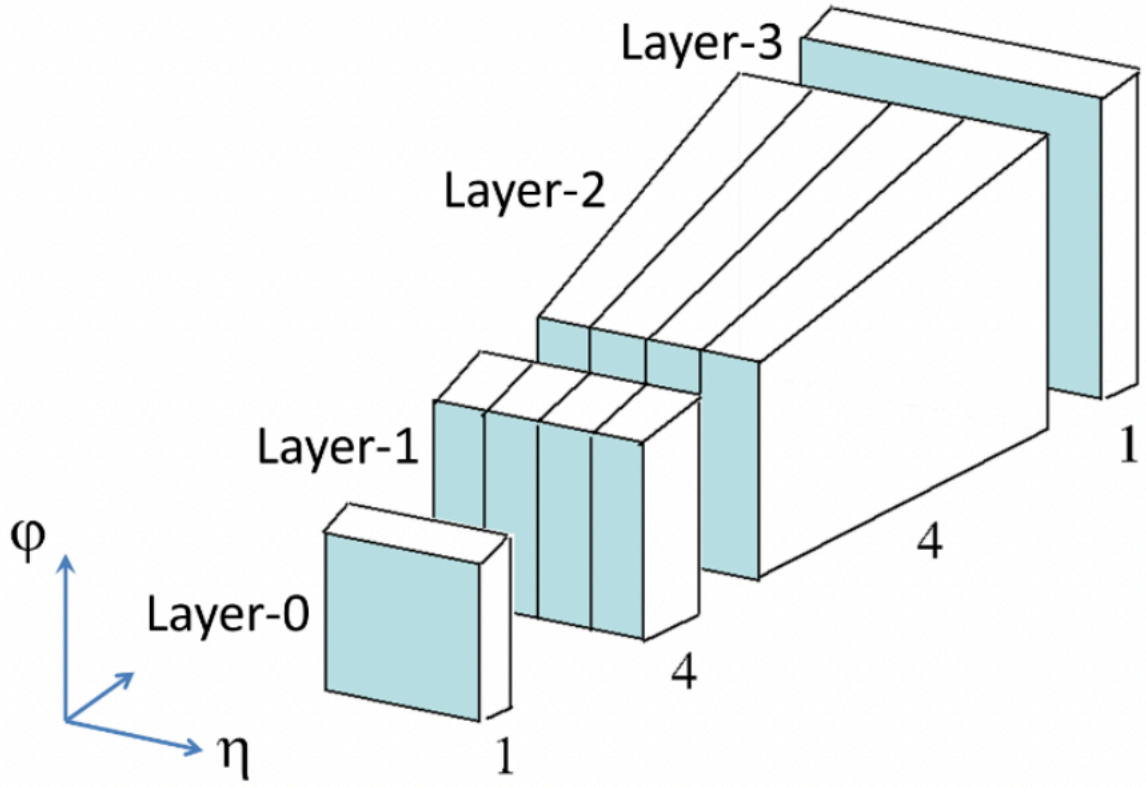
Level 1 calo trigger

Level 1 calo system does crude clustering of calorimeter data into geographic regions to look for potential electrons, muons, taus or jets passing minimum energy thresholds. We may need calibration for the detector!



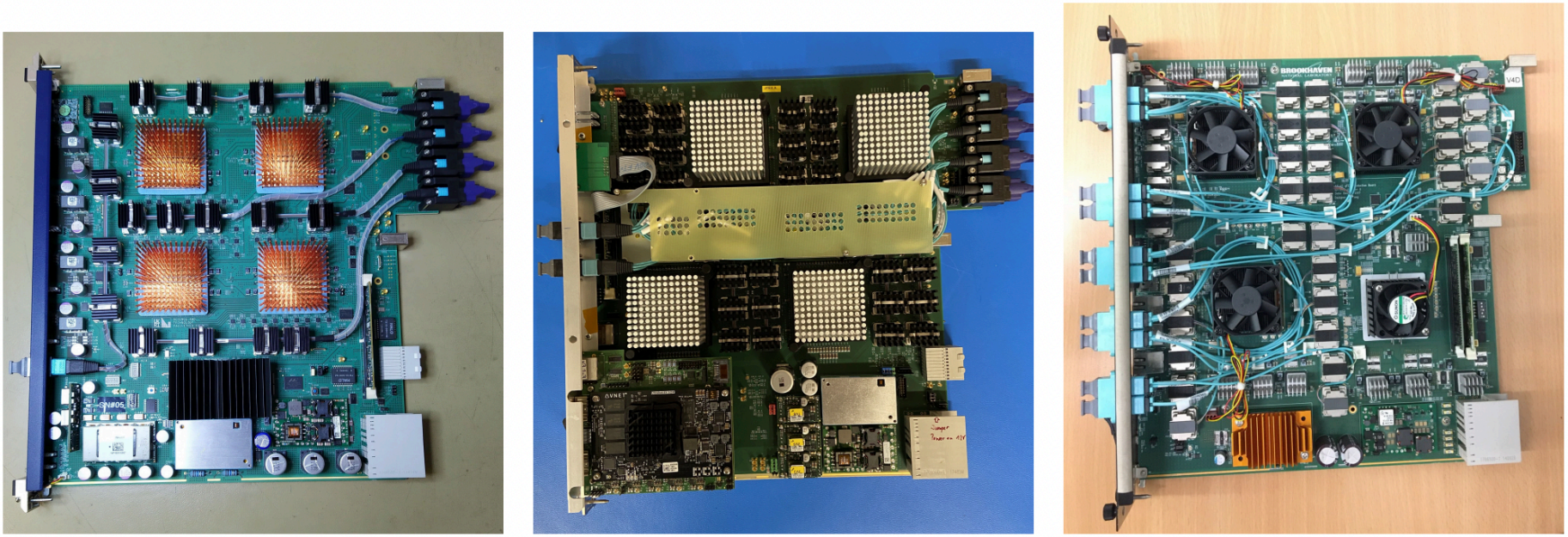
Level 1 calo trigger

Level 1 calo system does crude clustering of calorimeter data into into geographic regions to look for potential electrons, muons, taus or jets passing minimum energy thresholds. We may need calibration for the detector!



Level 1 calo trigger

Level 1 calo system does crude clustering of calorimeter data into into geographic regions to look for potential electrons, muons, taus or jets passing minimum energy thresholds. We may need calibration for the detector!



(a) (b) (c)

Figure 7.3: (a) A production eFEX module. (b) A production jFEX module. (c) A production gFEX module.

eFEX finds electron, photon and tau candidates!

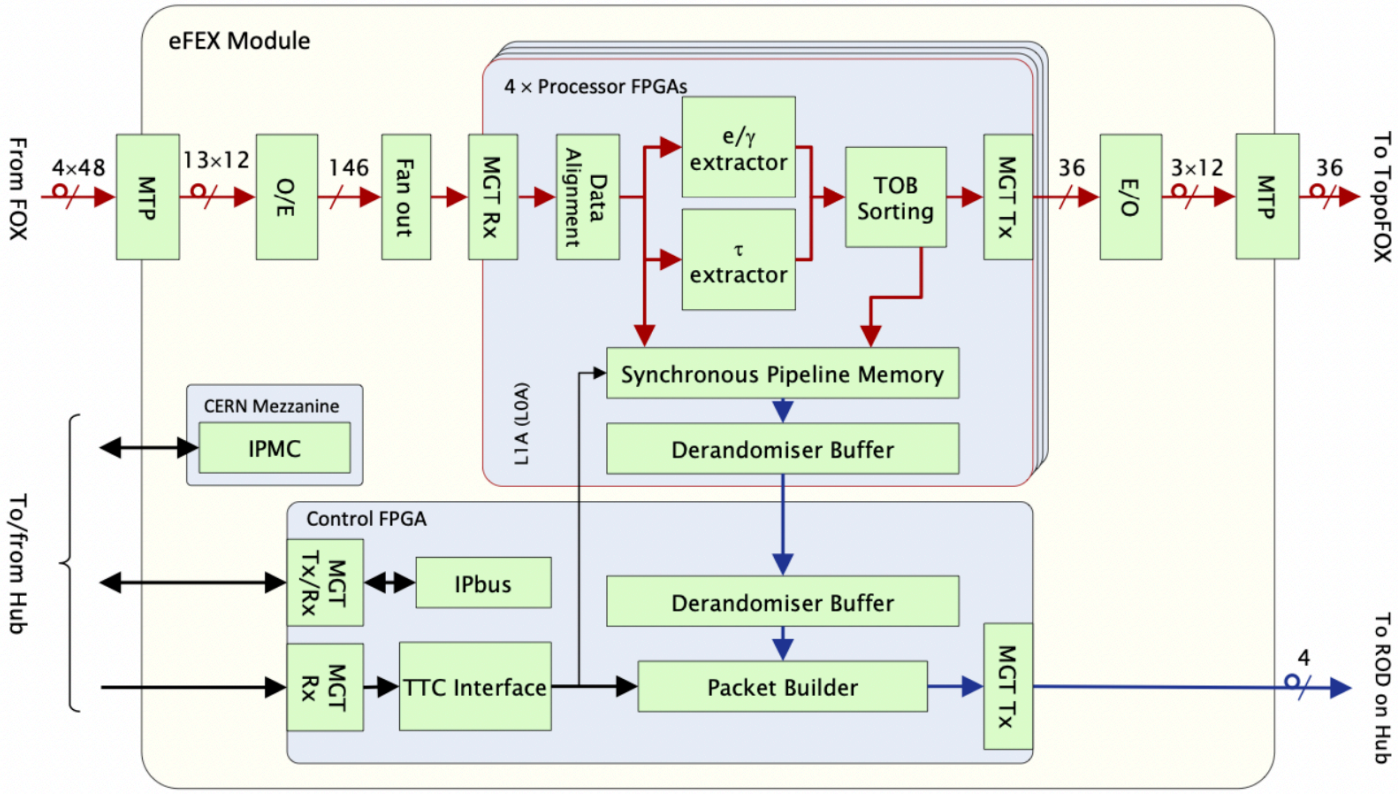


Figure 7.5: Block diagram of an eFEX module, illustrating the real-time and readout paths. Control and monitoring signals, except for the L1A, are not shown.

E/Gamma algorithm. Once a seed is found can compute many shower shape variables (shower widths, hadronic energy fraction, etc)

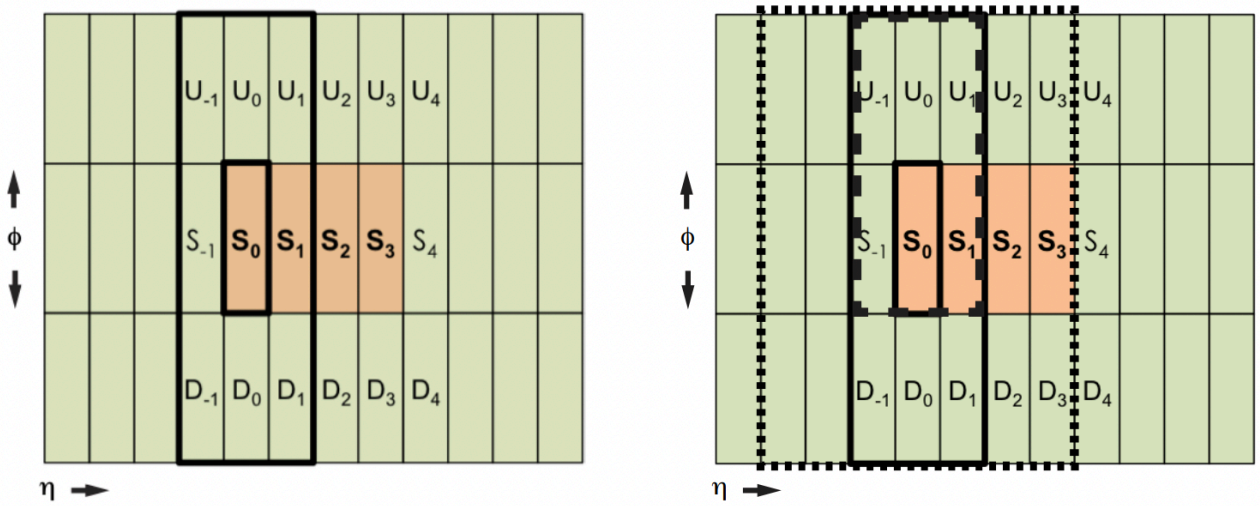


Figure 7.6: (a) The L1Calo eFEX seed finder algorithm. The four potential seeds (S) in the algorithm core are compared in order to determine the local energy maximum. The directionality of the cluster, upward (U) or downward (D) is determined by finding the largest energy deposit among the Super Cells surrounding the seed. (b) The dashed rectangle shows of 3×2 Super Cells used to compute the cluster energy, assuming that the local energy maximum was at cell S_0 and that the most energetic directly neighbouring Super Cells to S_0 was U_0. The dotted rectangle shows the 7×3 Super Cells area used for the calculation of the shower width in the second calorimeter layer.

Tau algorithm. Once a seed is found can compute many shower shape variables (shower widths, hadronic energy fraction, etc)

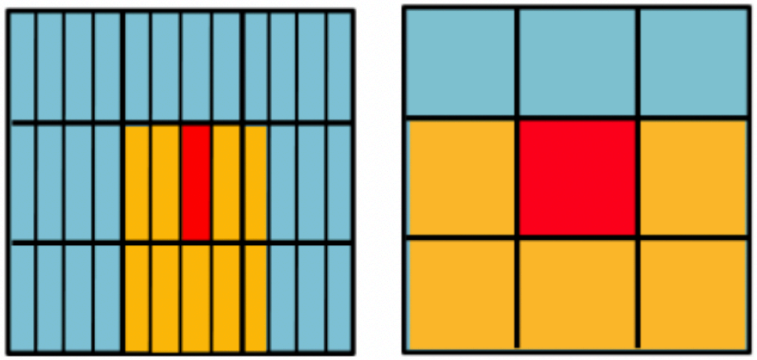


Figure 7.7: (a) The L1Calo eFEX tau cluster, as defined in the first and second EM calorimeter layers. The seed Super Cell is shown in red, while the other Super Cells making up the cluster are shown in orange. The surrounding environment, which is not included in the cluster energy computation, is shown in blue. (b) The L1Calo eFEX tau cluster, as defined in the presampler and third EM calorimeter layers, as well as in the hadronic layer.

jFEX identifies small-R jets, large-R jets, taus, MET and calculates H_T. It can do pileup subtraction (compute pileup density and subtract it from trigger towers), and also noise suppression for individual towers

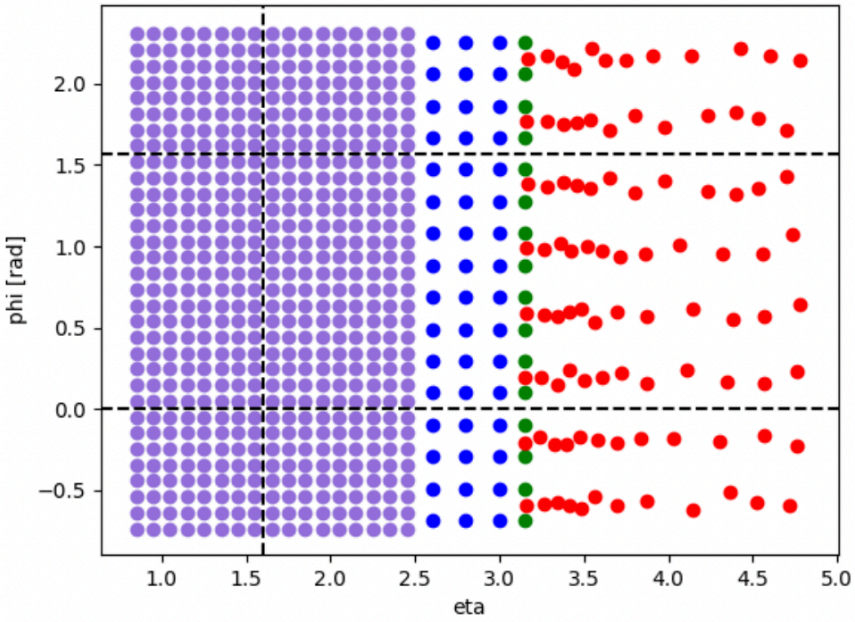


Figure 7.9: Granularity of trigger tower ($|\eta| < 3.1$) and Super Cell ($3.1 < |\eta| < 4.9$) inputs to the jFEX. The different colours serve to highlight the different granularities, as described in the text. The dashed lines illustrate the boundaries between the core region of a particular FPGA ($1.6 < |\eta| < 4.9, 0 < \phi < \pi/2$) and the surrounding environment.

Jet-finding uses a sliding window seeding process. Tau finding is done by computing isolation in a energy rings around jet seeds from small-R jets

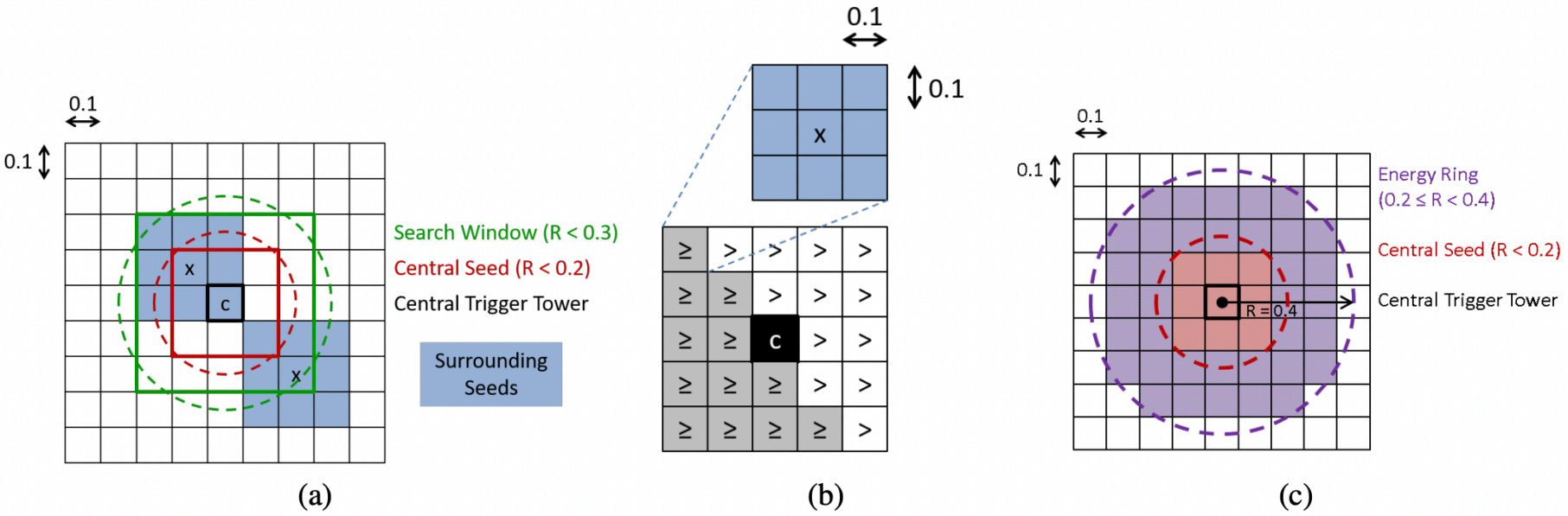


Figure 7.10: (a) The seeding process by which local maxima are identified in the jFEX. (b) Comparative operators used to identify local maxima in a given search window. (c) A small-radius jet as defined by the jFEX.

gFEX identifies global quantities about the bunch crossing such as MET and boosted objects

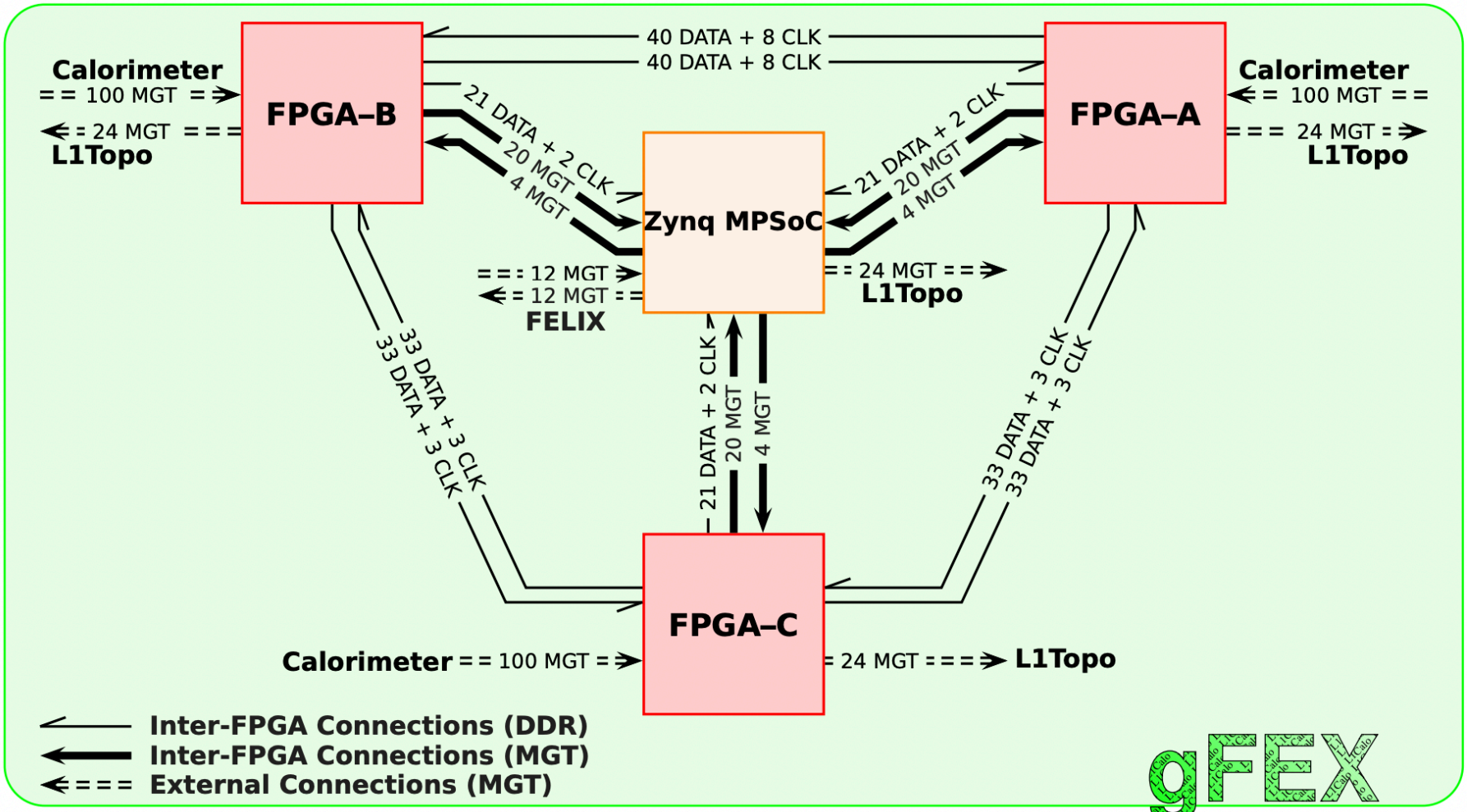
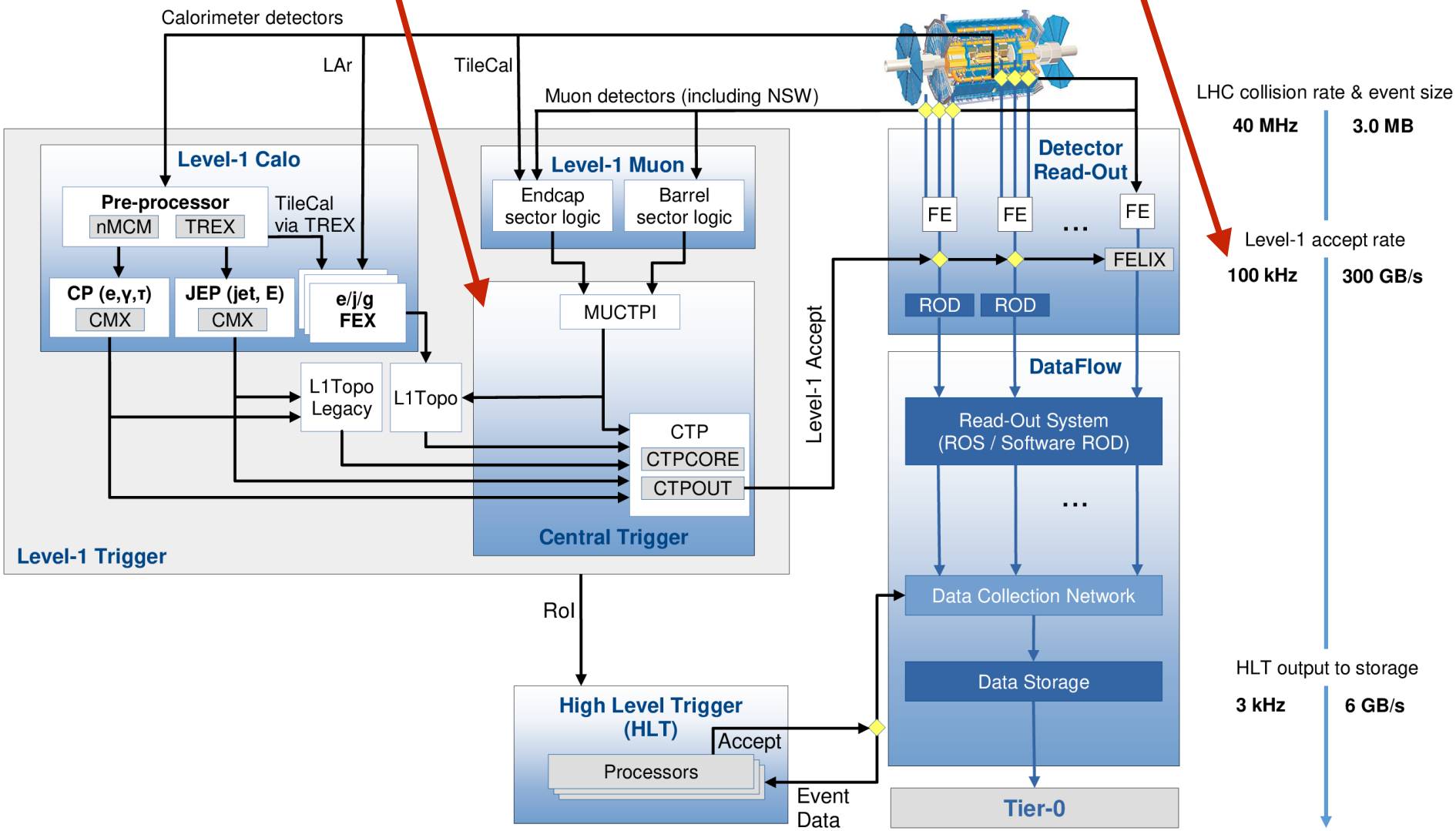


Figure 7.11: Block diagram of the L1Calo global Feature EXtractor (gFEX).

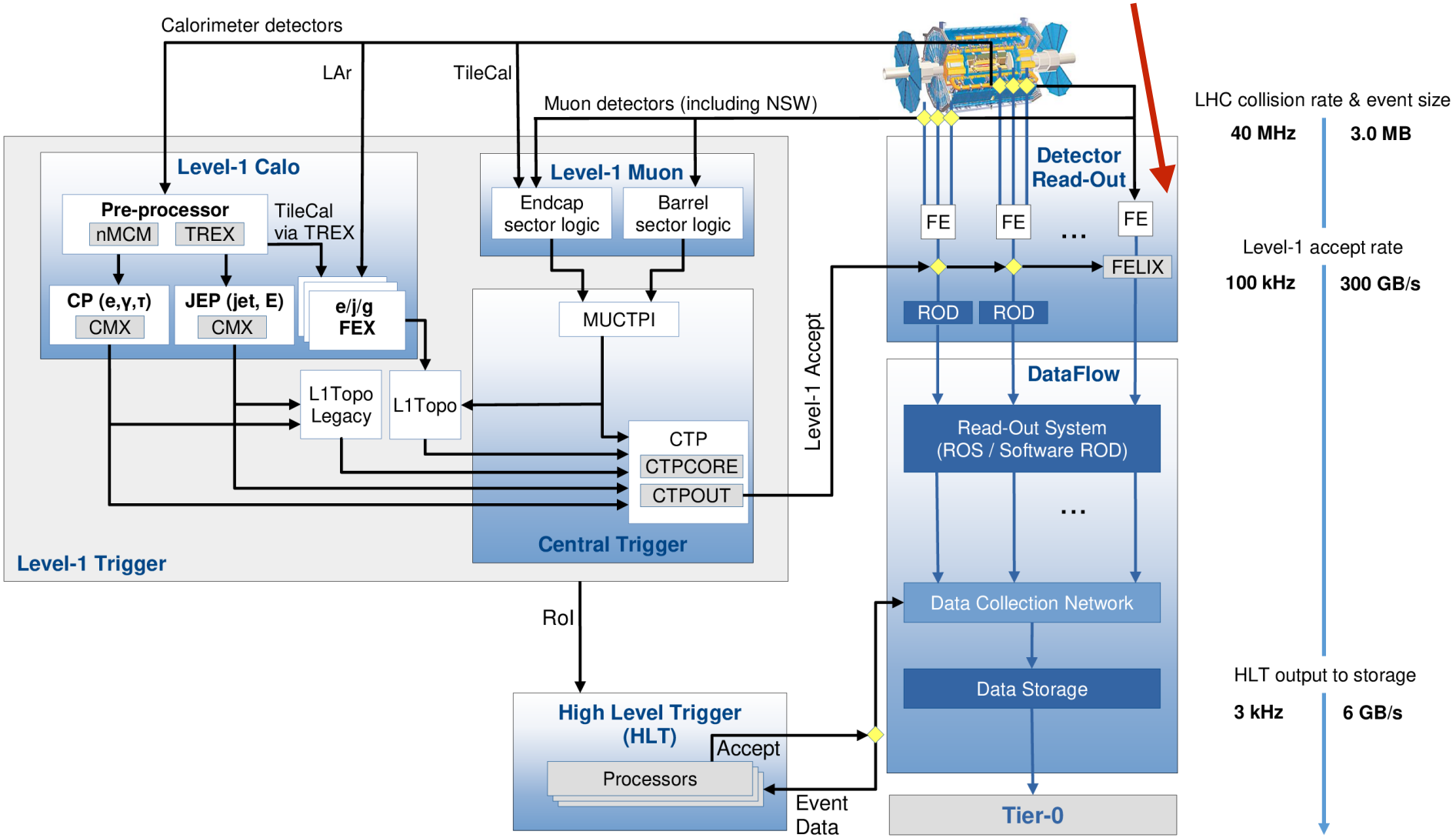
Level 1 trigger decision

Level 1 trigger decision made by the CTP combines information and decides if we should investigate this bunch cross further. 40 MHz → 100 kHz.

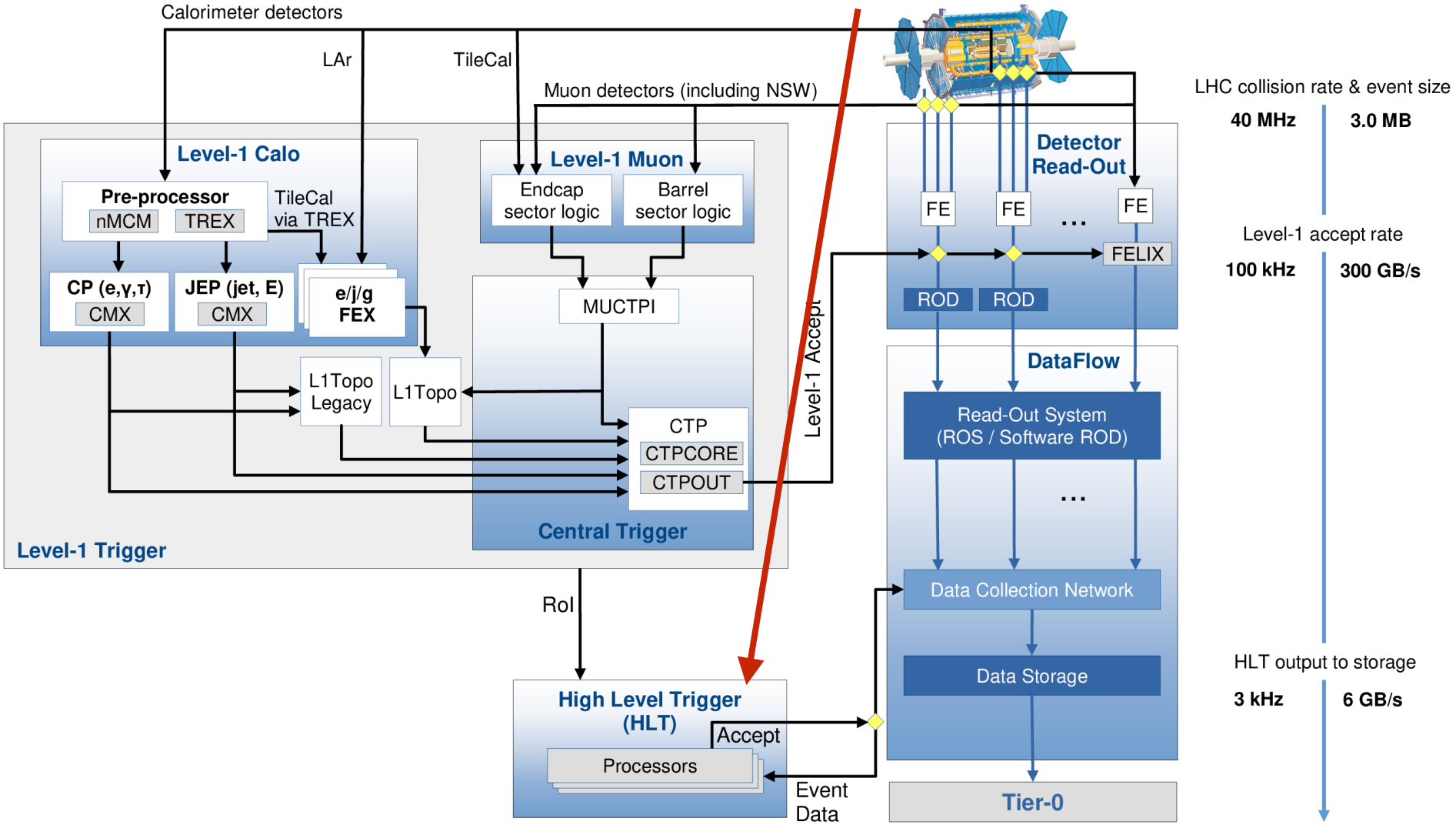


Level 1 trigger accept

If an event was selected by Level 1 trigger, the remaining parts of the detector need to be **read out** (we have since had other bunch crossings so we need to save and correlate this information along the way)

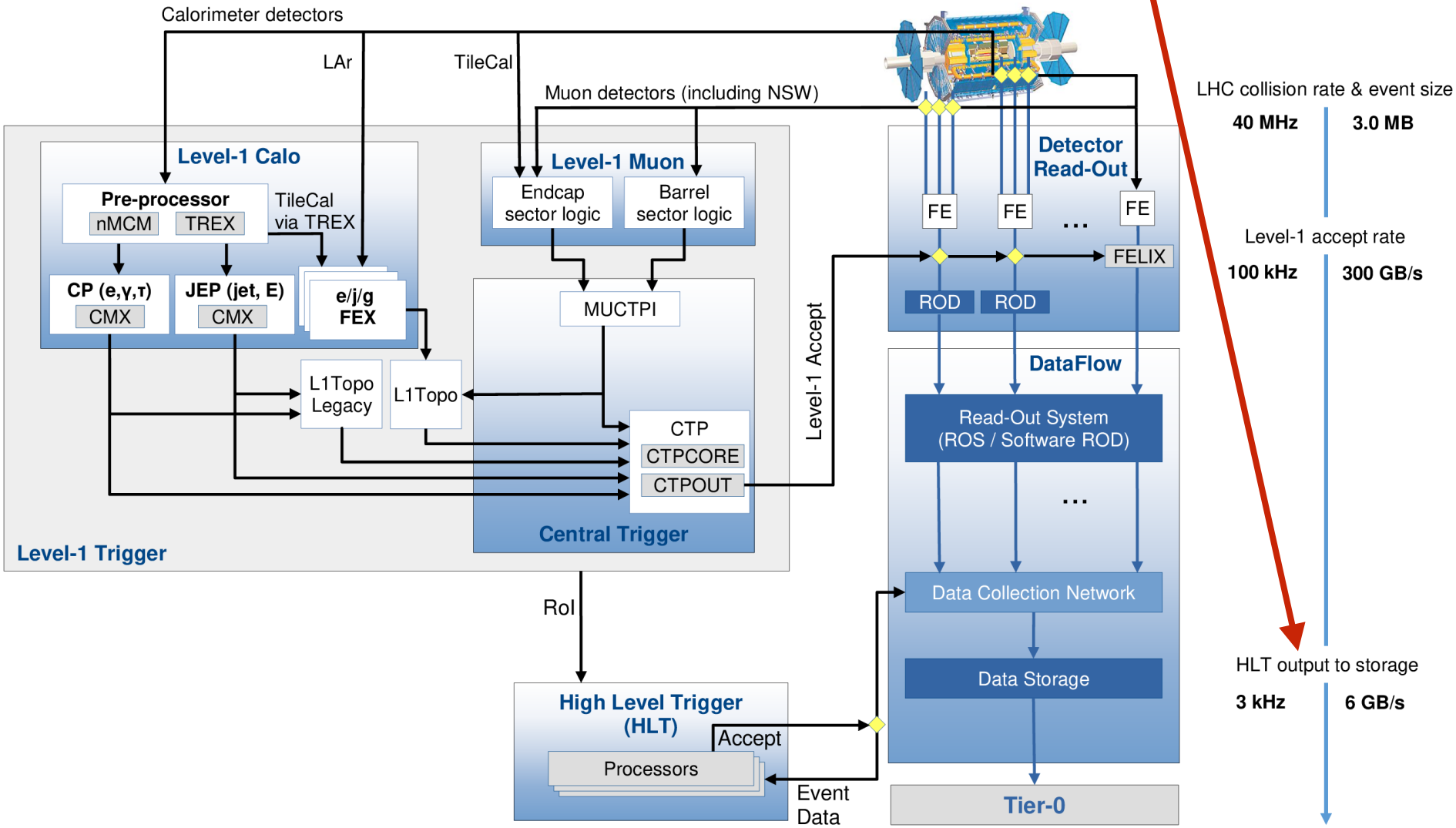


High Level Trigger (**HLT,** a massive computing farm, potentially with accelerators) takes data from either full detector or a region of interest in the detector (potentially using FPGAs in the future!), does event reconstruction, reduces rates to 3 kHz



Output

We again needed to **save the event data** while HLT did its processing, and then send output to tape



The above chain

Each piece needed some dedicated electronics. Each has complicated algorithms, many have large bandwidth rates. An example from ATLAS but not uncommon in the field

One way to do the above chain

ASICs: Application Specific Integrated Circuit. These are custom circuits that are going to give the best performance but …

ASICs are not something you can easily reconfigure. That means you can't easily make small modifications to reuse them to vary with detector regions, for different purposes on the experiment or across experiments. They are stubborn and fixed 😕

ASICs

ASICs are also custom objects. So there is a massive one-time overhead to creating them. HEP doesn't purchase millions of pieces of the same electronics, so ASICs are really expensive for us.

Sometimes we create custom boards with FPGAs on them ($$$), but other times we can use FPGAs that just plug into commodity computers!

What we'd like

It would be great to be able to tell some circuits what to do and to modify our instructions later on. Maybe our algorithms improve. Maybe the data we look at changes. Maybe we add new features. Or maybe different parts of the detector need slightly different configurations and instructions (detectors age, parts closer to the beam line may have different needs, etc)

Enter the FPGA

Field Programmable Gate Array (FPGA): Hardware that we can program, even after fabrication (literally we can rearrange the circuitry!) It has a 2D or even 3D array of logic gates that can be modified via software programming to fit user needs.

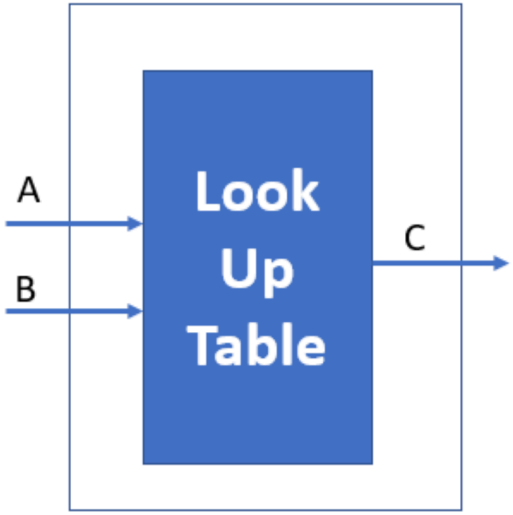
Less flexible than GPUs, but more flexible than ASICs, less power usage than GPUs, more than ASICs

More practically, what's inside an FPGA?

LUT (Lookup Table) is just a very boring table that generates digital outputs based on digital inputs. These function as logic gates!

Input A	Input B	Output C
0	0	0
0	1	0
1	0	0
1	1	1

This serves as an AND gate. We'll hear more about such things from Sasha shortly

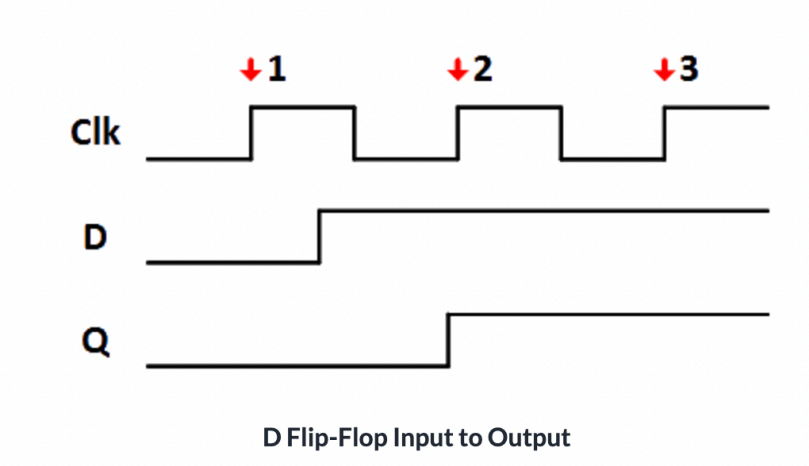


https://fpgainsights.com/fpga/overview-of-lookup-tables-lut-in-fpga-design/

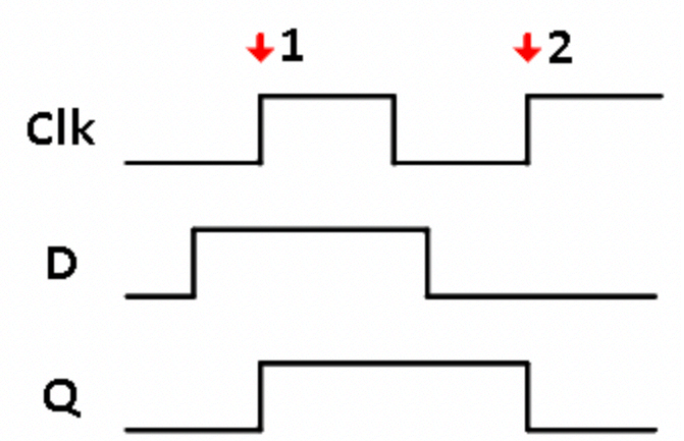
More practically, what's inside an FPGA?

2) Flip-flops are needed to store previous information about the state of the device. this *latching* is done on the edge of a periodic clock. Again, Sasha will talk more about this

https://nandland.com/lesson-5-what-is-a-flip-flop/



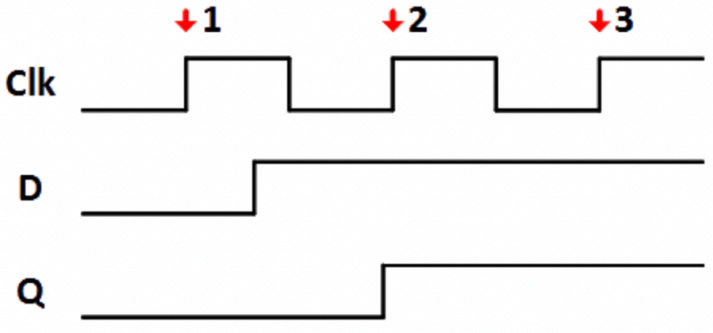
D Flip-Flop Input to Output



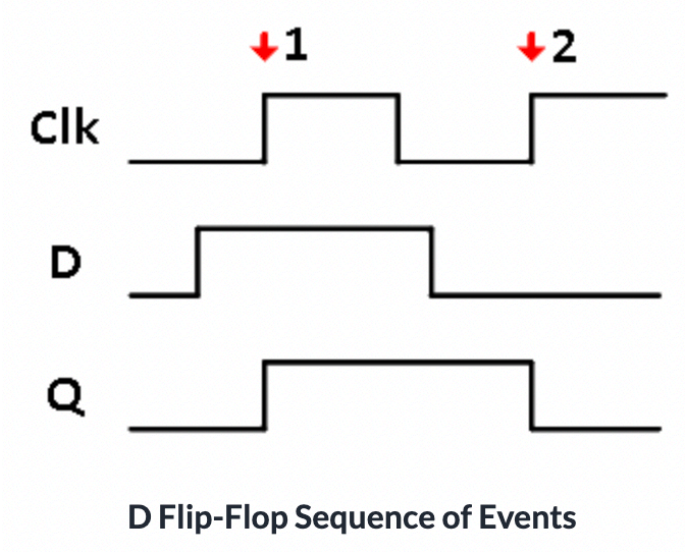
D Flip-Flop Sequence of Events

More practically, what's inside an FPGA?

2) LUTs + Flip-Flops = Configurable Logic Block (CLB, the repeating elements on an FPGA)



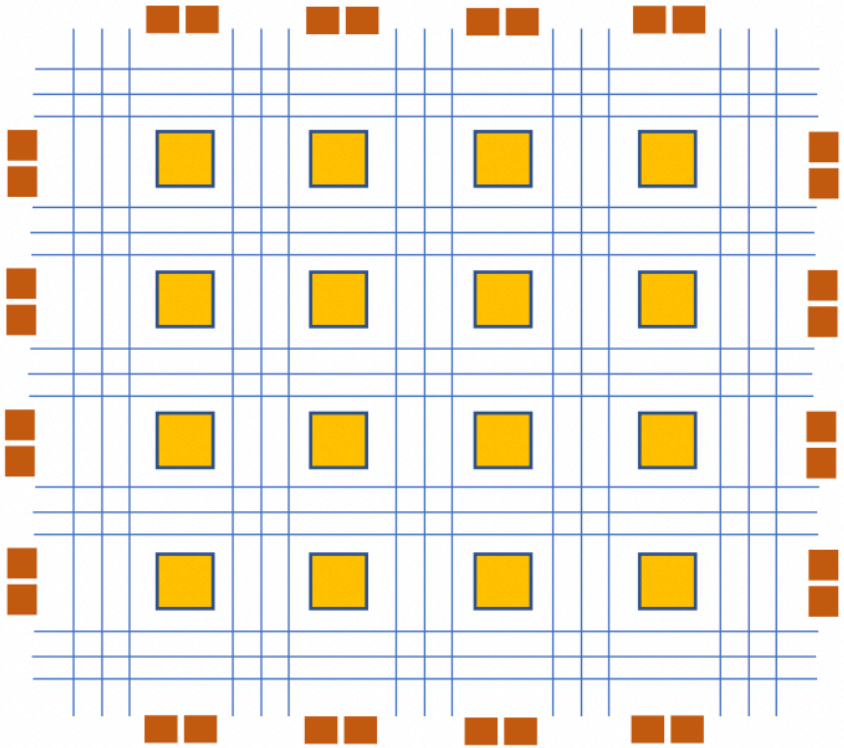
D Flip-Flop Input to Output



D Flip-Flop Sequence of Events

More practically, what's inside an FPGA?

3) Wires to connect all these pieces! And this interconnect routing can be reprogrammed!



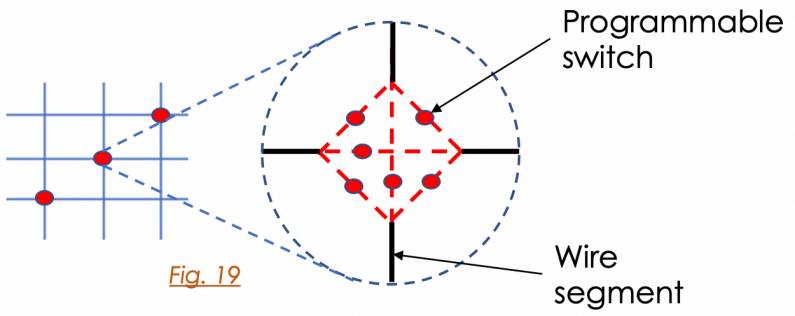
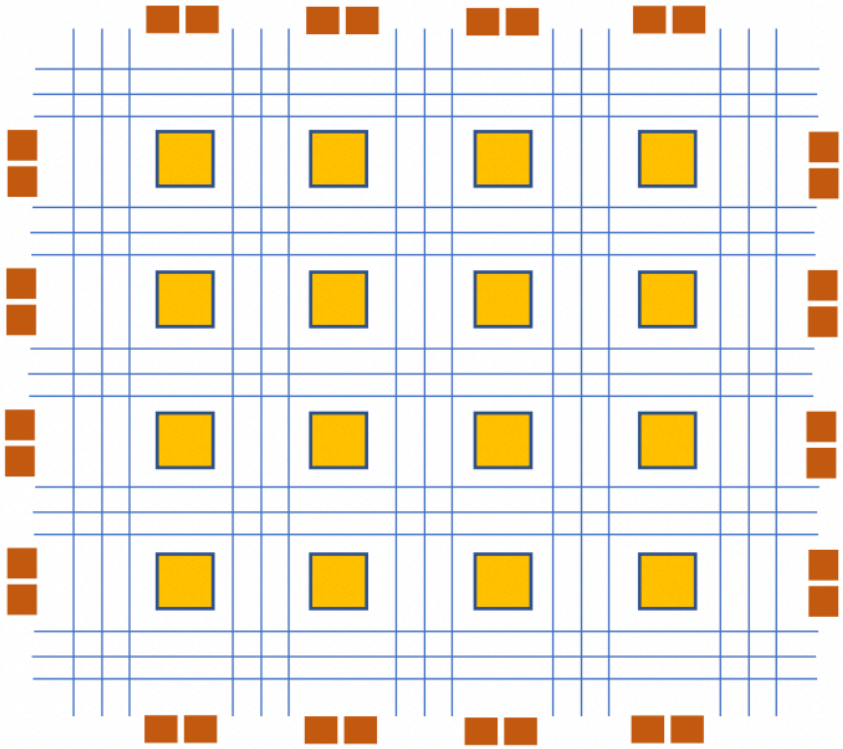


Fig. 19

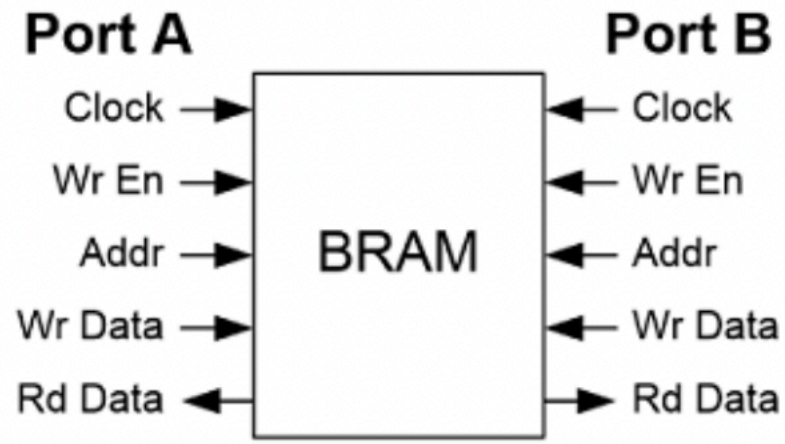
More practically, what's inside an FPGA?

4) I/O pads to connect to the outside world (after all, we need to get data in and out of the FPGA). Want to get data in and send it out quickly, with high bandwidth as needed



More practically, what's inside an FPGA?

5) BRAM (Block RAM) because sometimes we just need more memory to store useful information. Need to be able to read/ write it. BRAM is a precious resource!



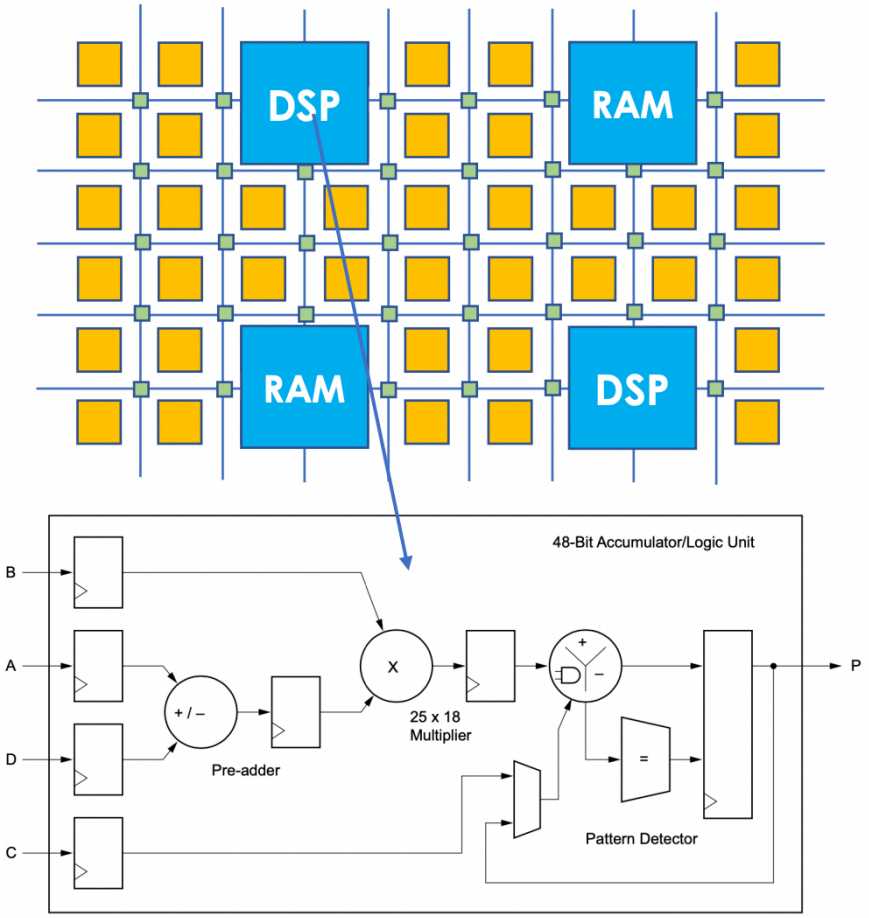
https://nandland.com/lesson-15-what-is-a-block-ram-bram/

More practically, what's inside an FPGA?

Note - your on-chip discrete (ie: finite and distinct from CLB) memories can have different names and different types depending on the FPGA you are targeting. Example: URAM (Ultra RAM), which is larger in capacity than BRAM but has different access ports and latency

More practically, what's inside an FPGA?

6) Digital Signal Processor (DSP) blocks: Because sometimes you want to do real math (ie y=A+Bx) and having specialized units for that is more efficient. Also a precious resource!



https://nandland.com/lesson-5-what-is-a-flip-flop/

Example FPGAs to set the scale

Costs rise rapidly to the right of this table!

Product Table

	XCVU3P	XCVU5P	XCVU7P	XCVU9P	XCVU11P	XCVU13P
System Logic Cells (K)	862	1,314	1,724	2,586	2,835	3,780
DSP Slices	2,280	3,474	4,560	6,840	9,216	12,288
Memory (Mb)	115.3	168.2	230.6	345.9	341	455
GTY/GTM Transceivers (32.75/58 Gb/s)	40/0	80/0	80/0	120/0	96/0	128/0
I/O	520	832	832	832	624	832

https://www.amd.com/en/products/adaptive-socs-and-fpgas/fpga/virtex-ultrascale-plus.html#product-table

So what is the real advantage to an FPGA?

Parallelism! Can have different pieces of the chip working on different tasks (or the same copy of one task) for a given clock cycle. *Can use registers and memory to run the FPGA on different pieces of data simultaneously!*

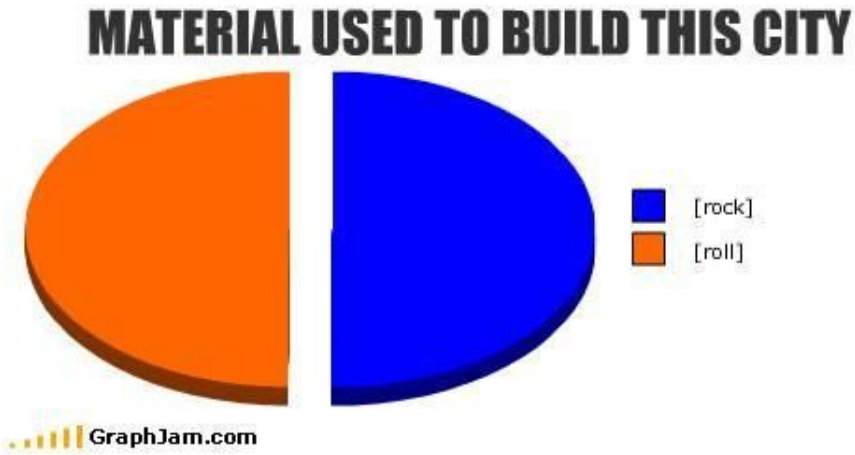
Great visual from TAC-HEP slides

Fig. 22: Like a production line for data…

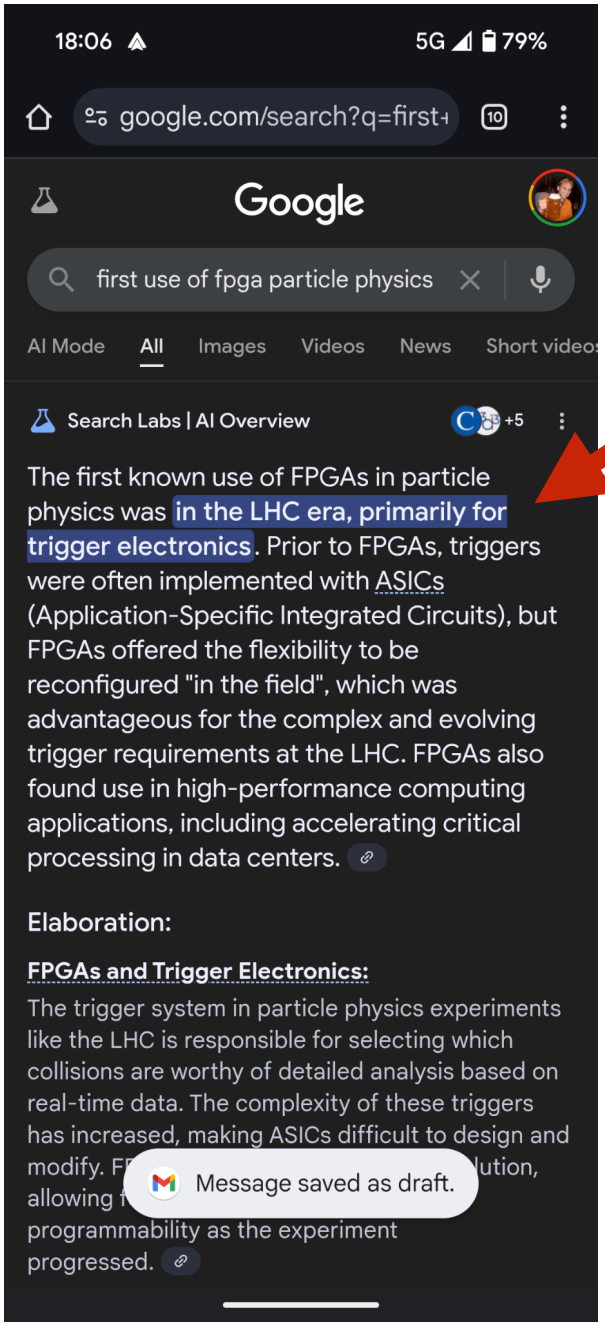
FPGAs: A brief history

1985: Xilinx produced the first commercially viable FPGA, the XC2064, with a whopping 64 CLBs!

Also in 1985, Starship's debut single We Built This City reached Number 14 on the Billboard Year-End Hot 100 chart



Google AI: A brief history of false information



This is just blatantly false!

I developed firmware for FPGAs on the CDF experiment in the early 2000s!

FPGAs were used even earlier than that in the mid-to-late 90s for Belle, CLEO III and HERA-B and also at the Tevatron

FPGAs: A brief history

Gates [edit]

- 1987: 9,000 gates, Xilinx[6]
- 1992: 600,000, Naval Surface Warfare Department[3]
- Early 2000s: millions[8]
- 2013: 50 million, Xilinx[12]

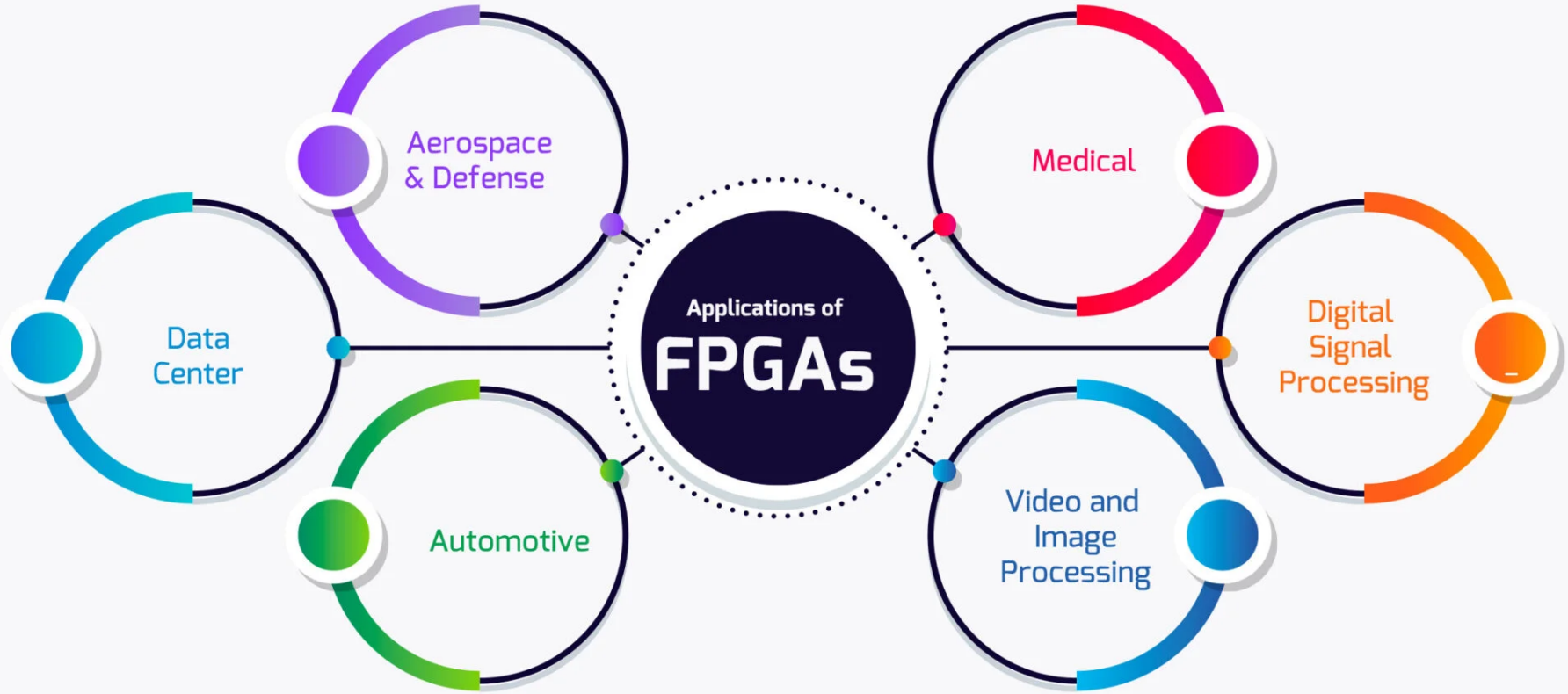
Market size [edit]

- 1985: First commercial FPGA : Xilinx XC2064[5][6]
- 1987: $14 million[6]
- c. 1993: >$385 million[6][*failed verification*]
- 2005: $1.9 billion[13]
- 2010 estimates: $2.75 billion[13]
- 2013: $5.4 billion[14]
- 2020 estimate: $9.8 billion[14]
- 2030 estimate: $23.34 billion[15]

https://en.wikipedia.org/wiki/Field-programmable_gate_array#Growth

Considerable growth!

Where to find FPGAs?



Everywhere! (Plus HEP)

https://www.logic-fruit.com/blog/fpga/fpga-design-architecture-and-applications/

Clear FPGA use cases

Finance: Can write custom algorithms to buy and sell stocks and other financial objects quickly. Over time, it was found that extremely small speed advantages lead to financial edges. But what if you come up with a new algorithm? Or a new stock to purchase? Or a new idea to implement? Reprogram the FPGA!

Clear FPGA use cases

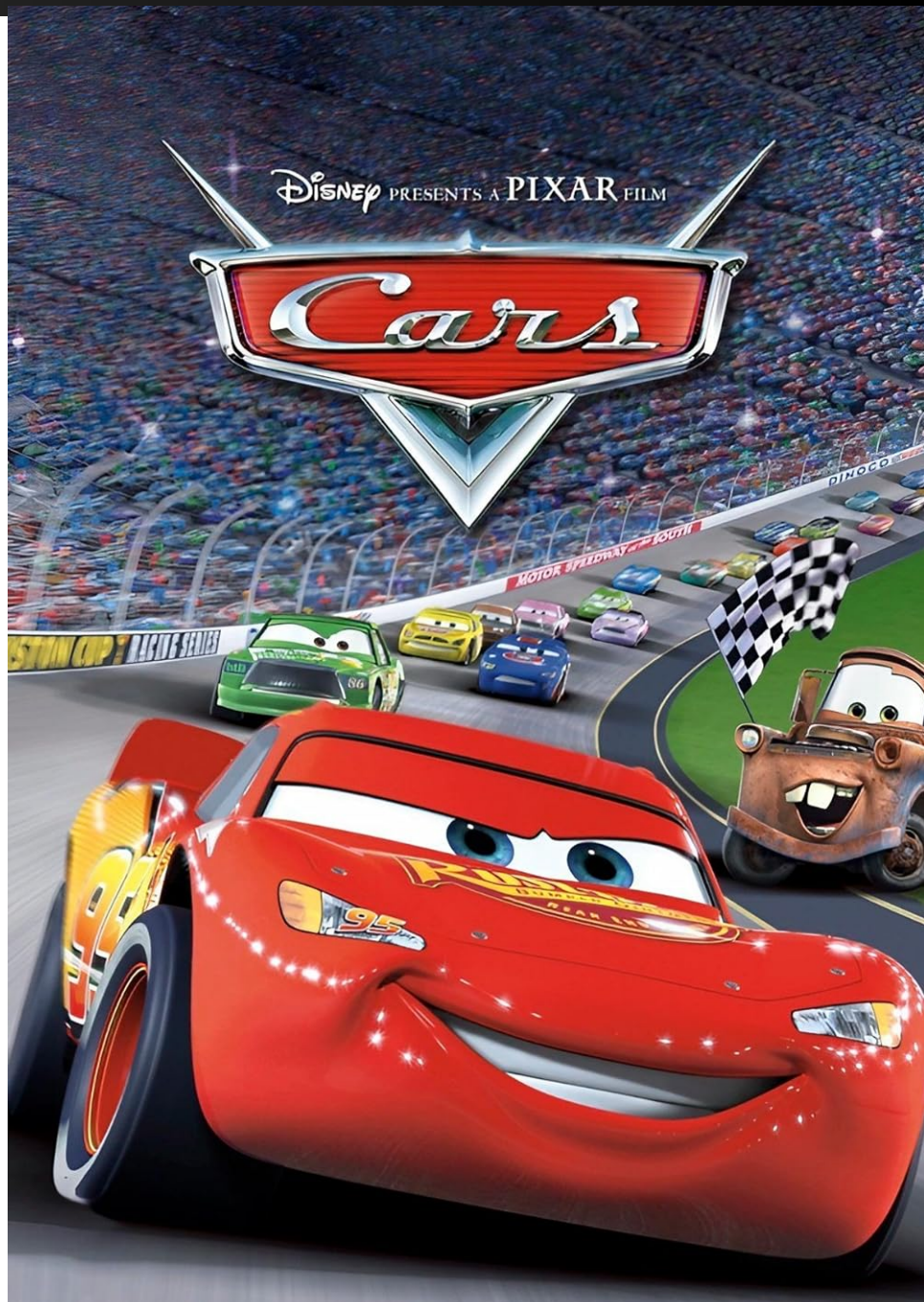
Telecommunications: Want to route and filter data and communications. Need to do it in parallel and quickly but also to be able to update your algorithm and configuration!

Clear FPGA use cases

Military: Radar processing and night vision. But want to be able to update the tools as needed

Clear FPGA use cases

Cars: Driver assistance tools like cruise control and lane keeping (need to process RADAR, LIDAR, cameras), controlling car powertrains



Stepping back: Addition in a CPU

Think about simple CPU operations such as **addition**. **Not trivial**!

- **Locate the two values** somewhere (cache? DDR RAM? Disk?)
- **Load** those two values into memory
- Do the **actual addition bitwise**
- **Store** the result
  - Note that we cannot always do two additions "at the same time" - often the disk and memory have limitations on this, even if two threads can do the addition at one time

Can be many, many, many CPU cycles for even silly simple operations!

Addition in an FPGA

Our **compiler** (we'll get to that!) is going to allocate **specific LUTs** and CLBs to this specific addition

- Storage is optimally placed by compiler as close to the CLB as possible, so memory bandwidth is huge
- Done on the given clock leading edge whether we need it or not
- **Not shared** with other CLBs
- You can do this addition many times over in **parallel**

Even for smaller clock speeds, FPGA can be much more efficient, and the parallel nature means it can have much larger overall bandwidth

Great summary from TAC-HEP

CPU advantages	FPGA Advantages
• Better with floating point numbers • Programming a CPU in normally easier than programming an FPGA (does not require to understand digital electronics) • Faster compilation • Easier code portability • Lower unit cost	• More flexible processing • More flexible input/output • Parallel processing • Better with multi-clock systems • Better with time-critical operations

More and more often, FPGAs and CPUs (or GPUs) are complementary:
They co-exist in the same system and perform different tasks

https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

Great summary from TAC-HEP

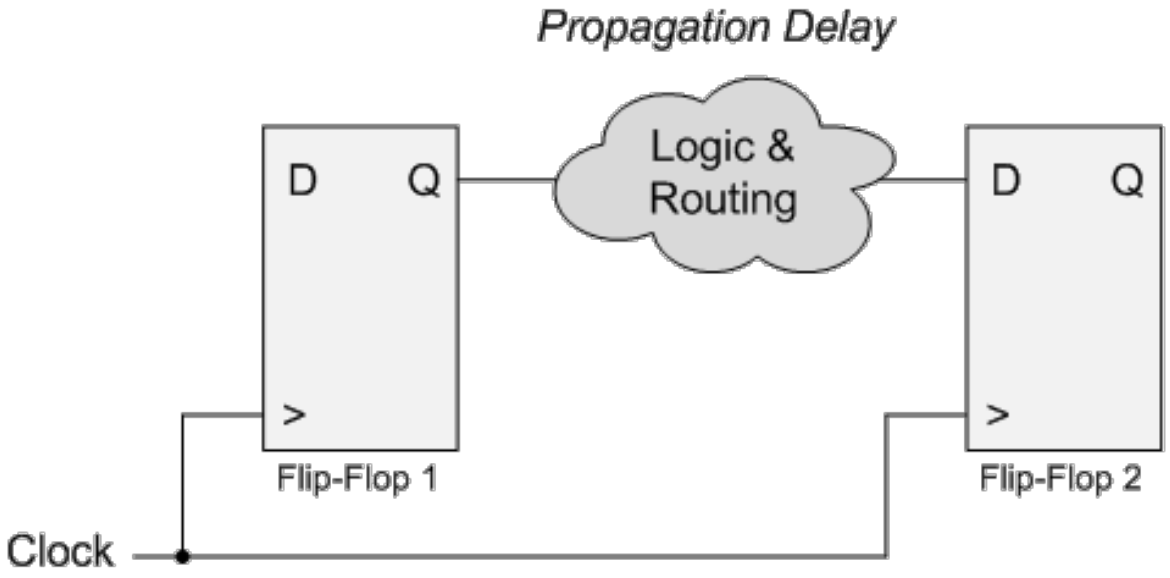
FPGA Advantages	ASIC Advantages
Faster time-to-market - no layout, masks or other manufacturing steps are needed **Lower constant/initial cost** **Simpler design cycle** - due to software that handles much of the routing, placement, and timing **More predictable project cycle** due to elimination of potential re-spins, wafer capacities, etc. **Re-programmability**: a new configuration can be uploaded	**Full custom capability (including analog) -** since device is manufactured to design specs **Lower unit costs** – For mass production **Smaller form factor** - since device is manufactured to design specs **Higher clock speeds**

https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

One challenge to using an FPGA

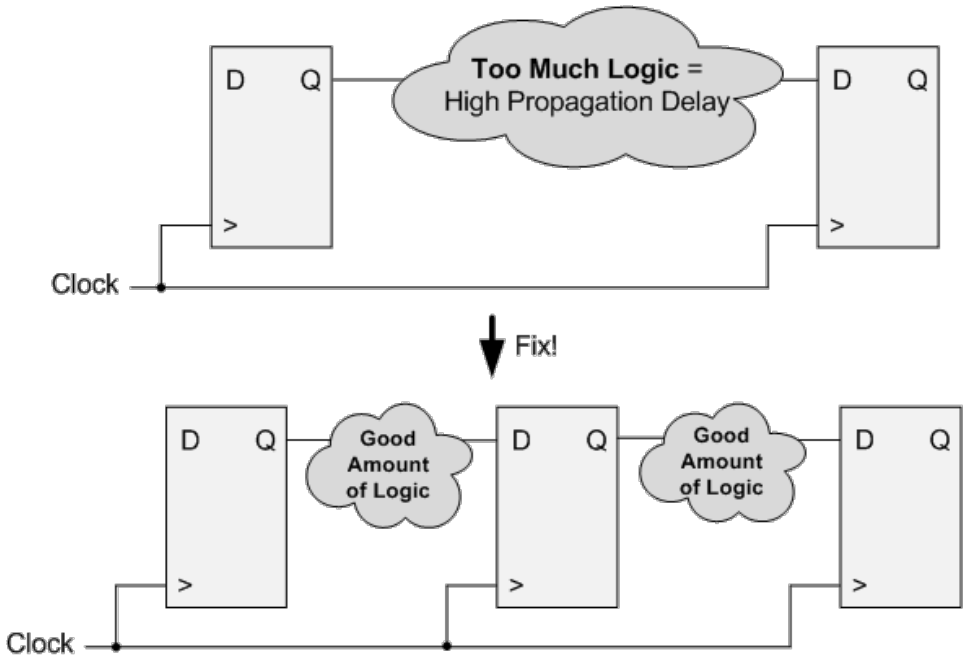
Speed of light = 3×10^8 m/s, or in more convenient units 0.3 m/ns. And since we're in the US and love imperial units, 0.3 m ~ 1 foot, so light travels ~12 inches every ns. NOT INFINITE

Propagation Delay: The time needed for some logic output to respond to a change at input. Sets the scale for the clock speed (ie how fast) your FPGA runs at!



https://nandland.com/lesson-11-what-is-propagation-delay/

Solution to propagation delay



Pipelining! Do less stuff in each clock cycle.
Now… doesn't that slow down the output?
Not necessarily! Why?

https://nandland.com/lesson-11-what-is-propagation-delay/

Pipelining FPGA vs C

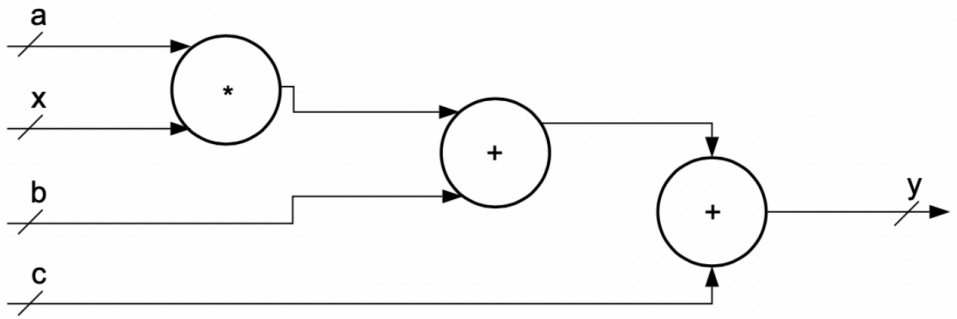
Your C code is executing your code completely sequentially. FPGA code can run pipelined: things happen simultaneously. Make sure this doesn't break anything!

TAC-HEP 2023

Pipelining

https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

C implementation

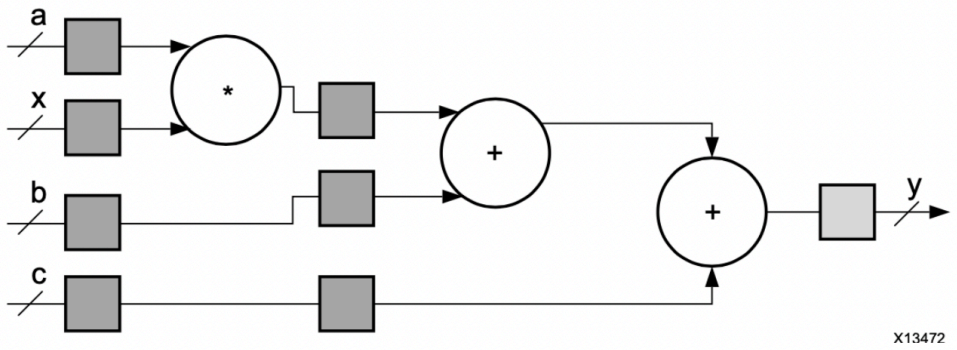


$$y = (a \times x) + b + c$$

Pipeline transformation

Fig. 5

Pipelined implementation



Your C code is executing your code completely sequentially. FPGA code can run pipelined: things happen simultaneously. Make sure this doesn't break anything!

Pipelining

- Boxes: registers implemented by FF blocks
- Each box column counted as single clock cycle
- Result in 3 clock cycles.
- Addition of registers, leads to separated compute sections for each block
  - Multiplier & two adders can run in parallel and reduce latency

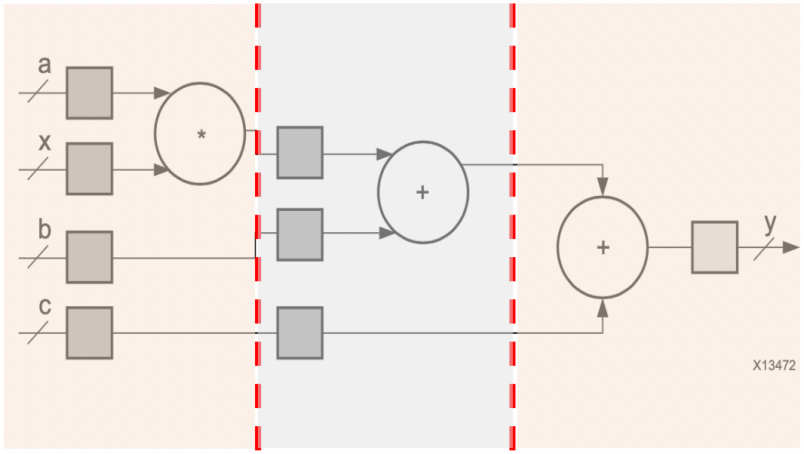


Fig. 6

https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

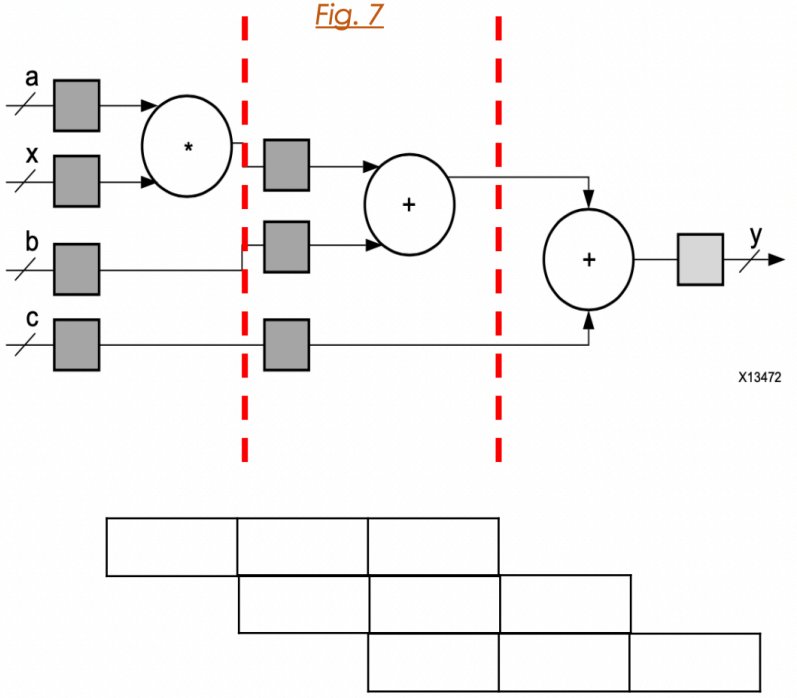
Even if first initial computation is not faster, if we compute this for many inputs, things are improved!

TAC-HEP 2023

Pipelining

- Both sections of the datapath run in parallel
  - Essentially computing the ***y*** and ***y'*** in parallel
  - ***y'*** result of the next execution
  - First computation of y: pipeline fill time = 3 CLK
  - After this initial computation, a new value of y is available at the output on every clock cycle, because the computation pipeline contains overlapped data sets for the current and subsequent y computations

Fig. 7



https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

Latency is not the full picture since we continuously get output after the first cycle

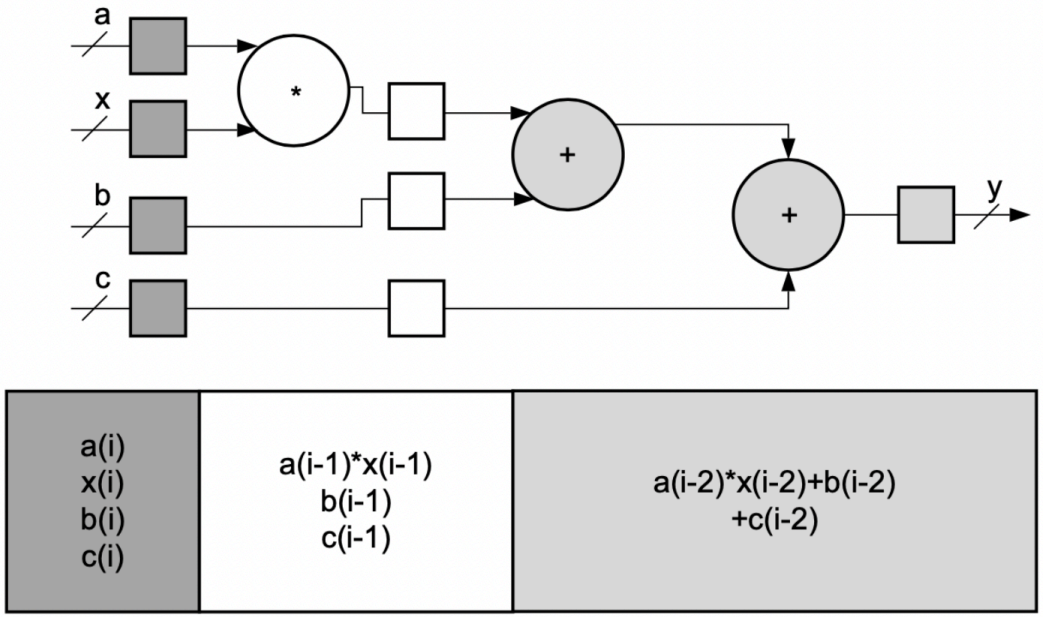
TAC-HEP 2023

Pipelining

- Raw data: dark gray,
- Semi-computed data: white
- Final data: light gray

All exist simultaneously & each stage result is captured in its own set of registers

Although the latency for such computation is in multiple cycles, there is new result with every cycle



https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf Fig. 8

Scheduling is critical! Because each operation can happen simultaneously with the clock edge, need to ensure that all inputs are appropriate and ready

TAC-HEP 2023

Scheduling

E.g.: Scheduling phases for a simple code

```
int foo(char x, char a, char b, char c) {
 char y;
 y = x*a+b+c;
 return y;
}
```

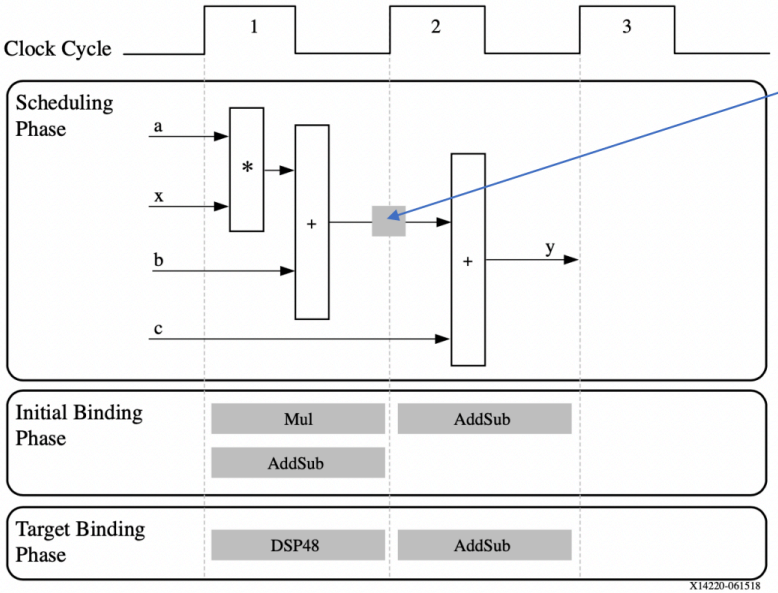


Fig. 3

Internal register storing a variable

First cycle: reads x, a, and b data ports
Second cycle: reads data port c & generates output y

First cycle: Multiplication and the first addition
Second cycle: Second addition and output generation

https://tac-hep.org/assets/pdf/uw-gpu-fpga/2023-03-22-FPGA-HLS-Lecture-2.pdf

So what is programming the FPGA?

Register Transfer Level (RTL) is the way to describe a circuit. It is the lowest level description used before synthesizing the circuit (converting it to the actual gates and connections)

Hardware Description Language (HDL) is a low-level programming language that is converted into RTL. One common example of HDL is **VHDL (VHSIC HDL),** where, *because we love acronyms*, **VHSIC=Very High Speed Integrated Circuit Program**. Speaking from experience as someone who wrote VHDL code in grad school, **you don't want to write VHDL**

So **VHDL = Very High Speed Integrated Circuit Program Hardware Description Language** ***(phew)***

So what is programming the FPGA?

Register Transfer Level (RTL) is the way to describe a circuit. It is the lowest level description used before synthesizing the circuit (converting it to the actual gates and connections)

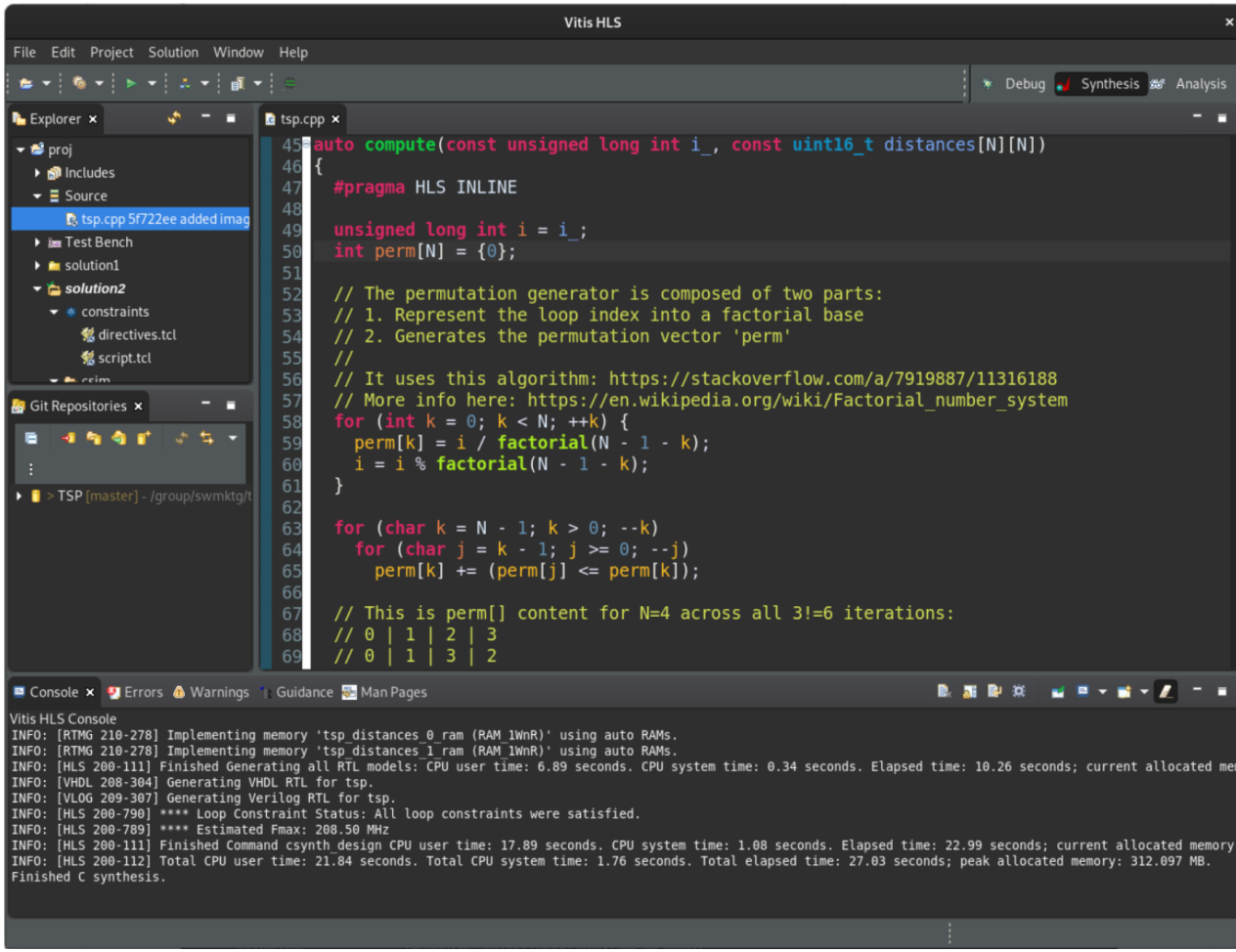
Hardware Description Language (HDL) is a low-level programming language that is converted into RTL. One common example of HDL is **VHDL (VHSIC HDL),** where, *because we love acronyms*, **VHSIC=Very High Speed Integrated Circuit Program**. Speaking from experience as someone who wrote VHDL code in grad school, **you don't want to write VHDL**

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;    -- for the unsigned type

entity COUNTER is
  generic (
    WIDTH : in natural := 32);
  port (
    RST   : in std_logic;
    CLK   : in std_logic;
    LOAD  : in std_logic;
    DATA  : in std_logic_vector(WIDTH-1 downto 0);
    Q     : out std_logic_vector(WIDTH-1 downto 0));
end entity COUNTER;

architecture RTL of COUNTER is

begin

  process(all) is
  begin
    if RST then
      Q <= (others => '0');
    elsif rising_edge(CLK) then
      if LOAD='1' then
        Q <= DATA;
      else
        Q <= std_logic_vector(unsigned(Q) + 1);
      end if;
    end if;
  end process;

end architecture RTL;
```

Wikipedia: VHDL Simple Counter

So what is the replacement for HDL coding?

HLS (High Level Synthesis) is a **C-like** programming language (with extra directives) that is converted by tools such as **Vitis** into either HDL or RTL. You can program in C, which allow for easier-to-read code and simpler syntax



https://xilinx.github.io/Vitis-Tutorials/2021-2/build/html/docs/Hardware_Acceleration/Desig
salesperson/project.html

So what is the replacement for HDL coding?

Goal of this workshop - get you to understand how to **use HLS to write efficient and optimized firmware!**

Conclusion

Because of pipelining and the custom circuitry designed for a given problem (as opposed to CPUs, which are generic all the way until usage), FPGAs are typically much more efficient than CPUs in terms of data output AND power usage. That is their true motivation.