

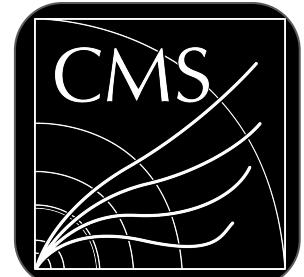
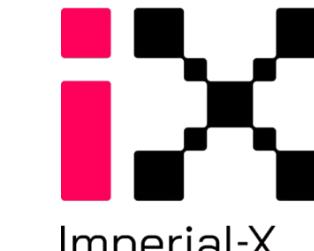
# Accelerating Machine Learning with FPGAs

AIMS 2025

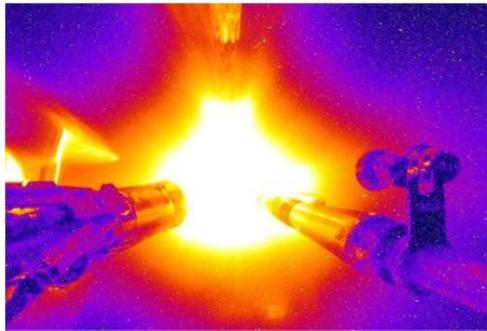
Jonathon Langford

9th May 2025

[Github link]



# Who am I?



University of Manchester  
MPhys (2013-2017)

PhD @ Imperial College  
High Energy Physics (2017-2021)

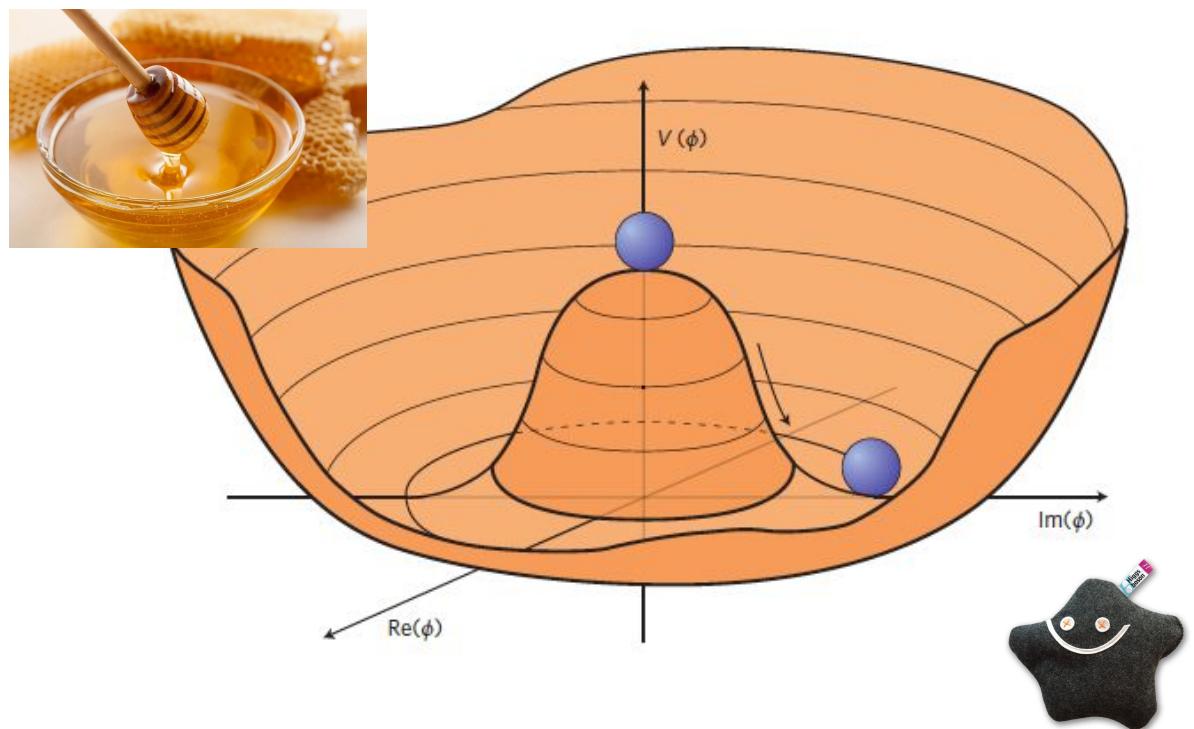
Research Associate @ Imperial College  
High Energy Physics (2021-2023)

Schmidt AI in Science Fellow  
2023-2024

Imperial College Research Fellowship  
2024-



CMS experiment  
@ Large Hadron Collider (LHC)

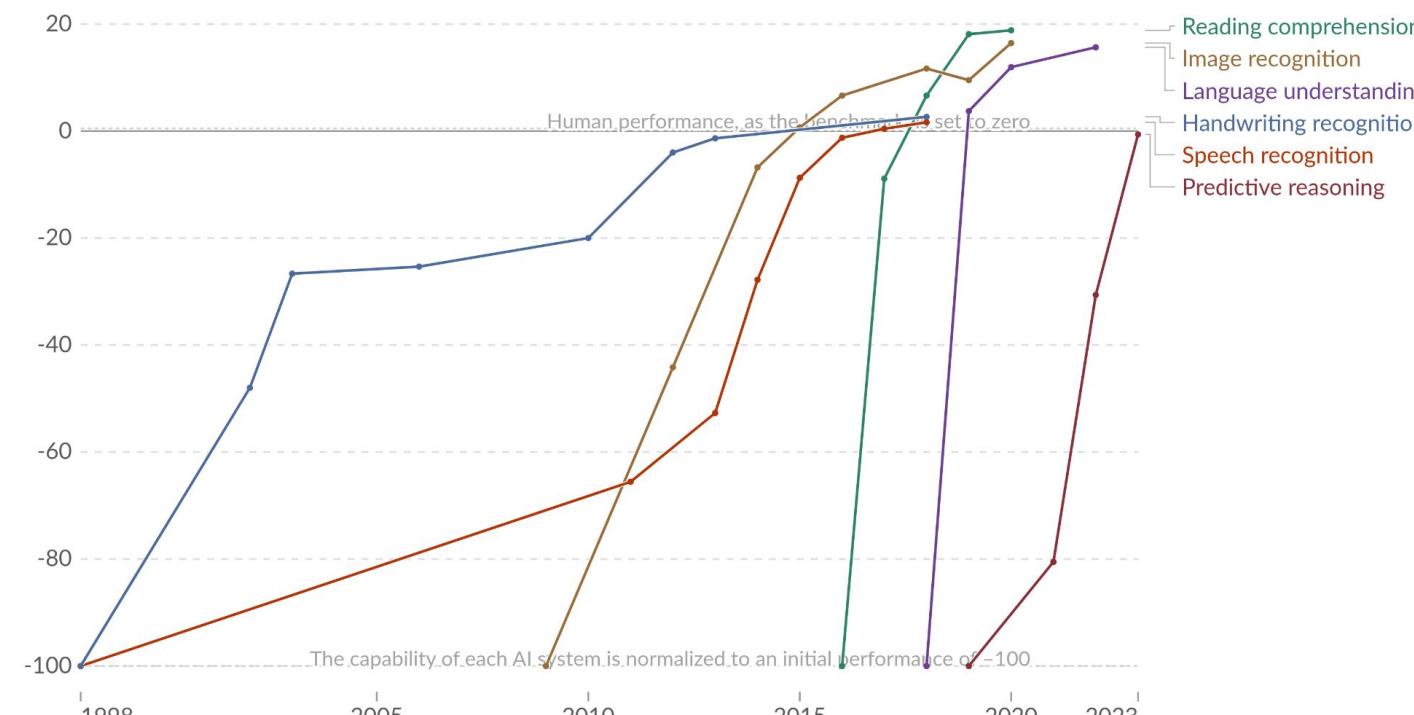


# Motivation

- State of play: we are (becoming) an AI-dependent world...

## Test scores of AI systems on various capabilities relative to human performance

Within each domain, the initial performance of the AI is set to -100. Human performance is used as a baseline, set to zero. When the AI's performance crosses the zero line, it scored more points than humans.



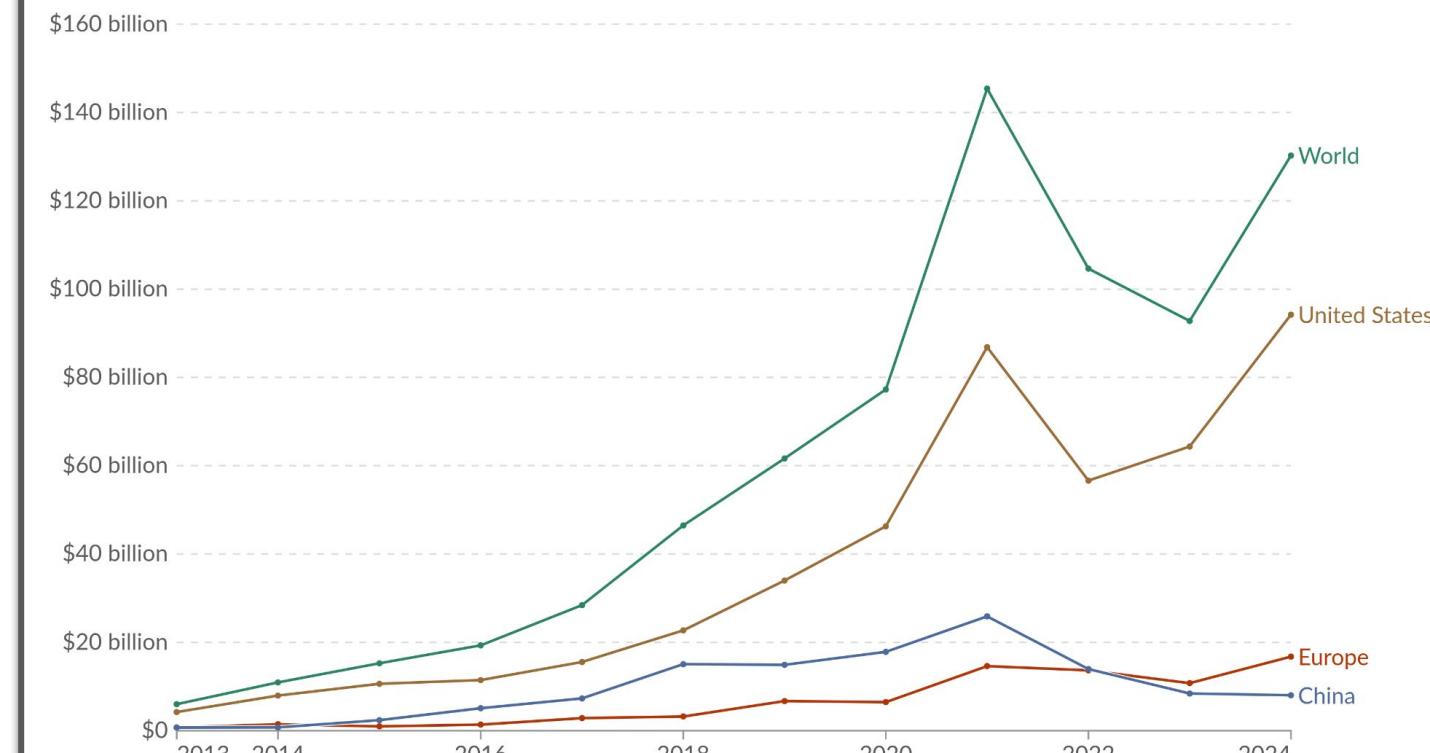
Data source: Kiela et al. (2023)

Note: For each capability, the first year always shows a baseline of -100, even if better performance was recorded later that year.

OurWorldinData.org/artificial-intelligence | CC BY

## Annual private investment in artificial intelligence

Includes companies that received more than \$1.5 million in investment. This data is expressed in US dollars, adjusted for inflation.



Data source: Quid via AI Index Report (2025); U.S. Bureau of Labor Statistics (2025)

Note: Data is expressed in constant 2021 US\$. Inflation adjustment is based on the US Consumer Price Index (CPI).

OurWorldinData.org/artificial-intelligence | CC BY

# Motivation

- State of play: we are (becoming) an AI-dependent world... with constraints
  1. Data growing exponentially
  2. Limited (compute) resources
  3. Limited time
  4. Carbon footprint

# Motivation

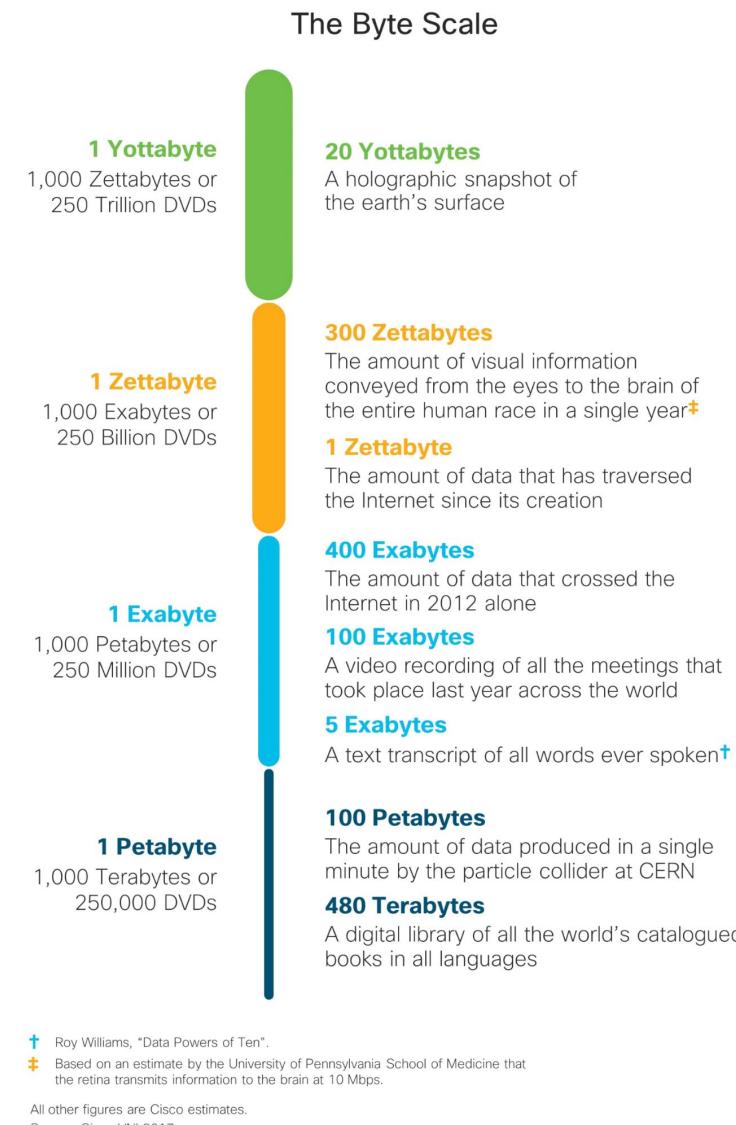
- State of play: we are (becoming) an AI-dependent world... with constraints

## 1. Data growing exponentially

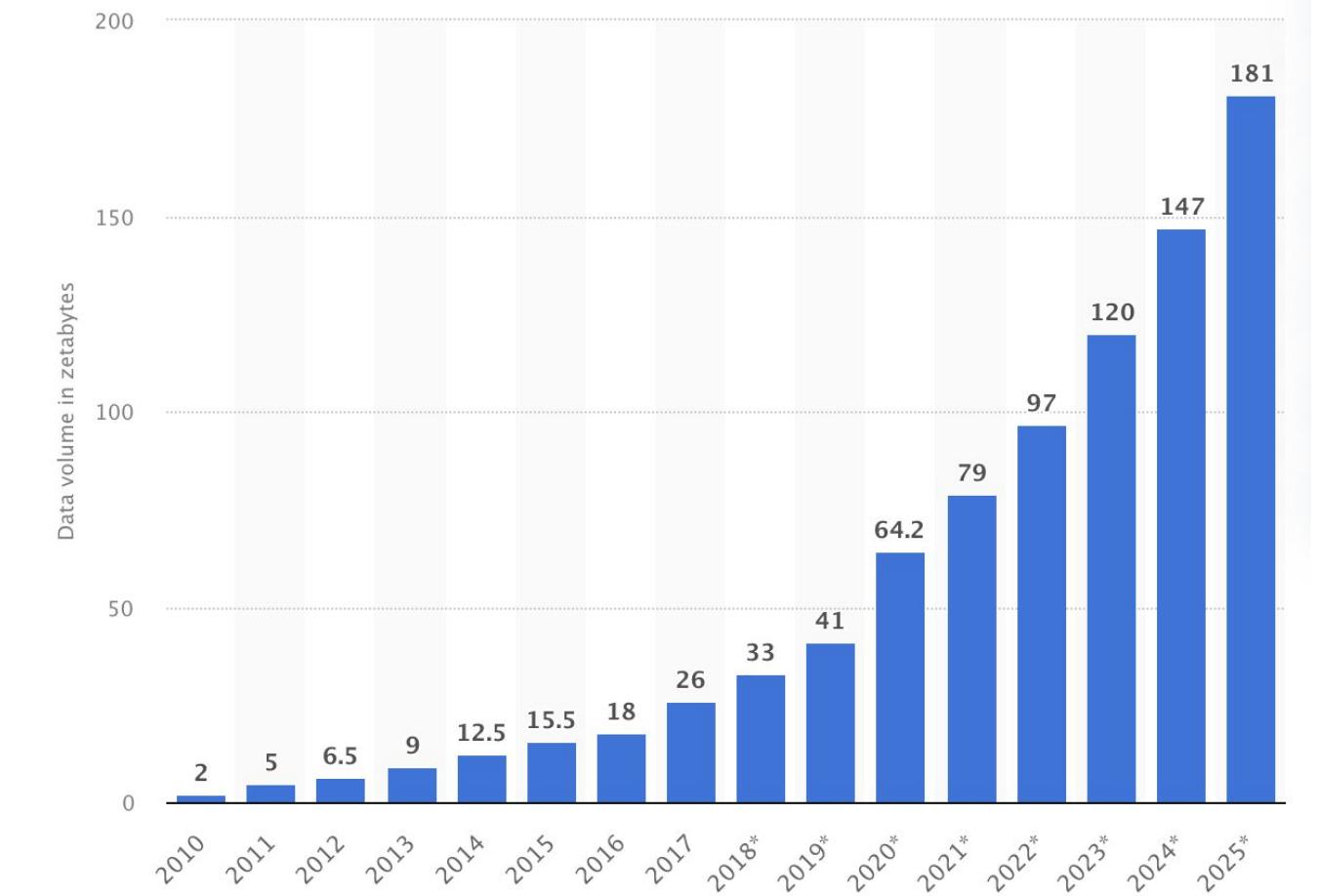
## 2. Limited (compute) resources

## 3. Limited time

## 4. Carbon footprint



Volume of data created, captured, copied and consumed worldwide



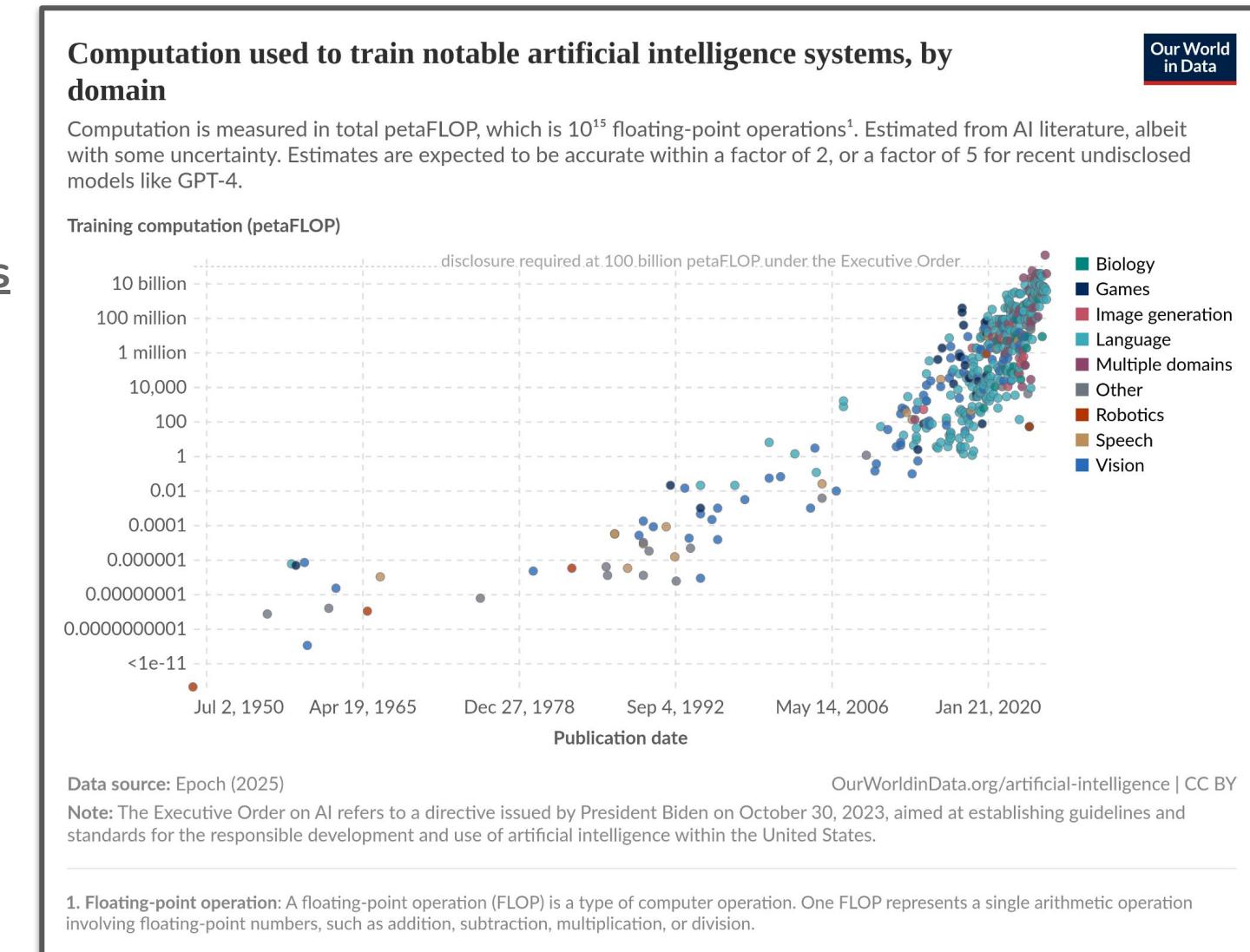
<https://www.statista.com/statistics/871513/worldwide-data-created/>

We need to process this!

# Motivation

- State of play: we are (becoming) an AI-dependent world... with constraints

1. Data growing exponentially
2. Limited (compute) resources
3. Limited time
4. Carbon footprint



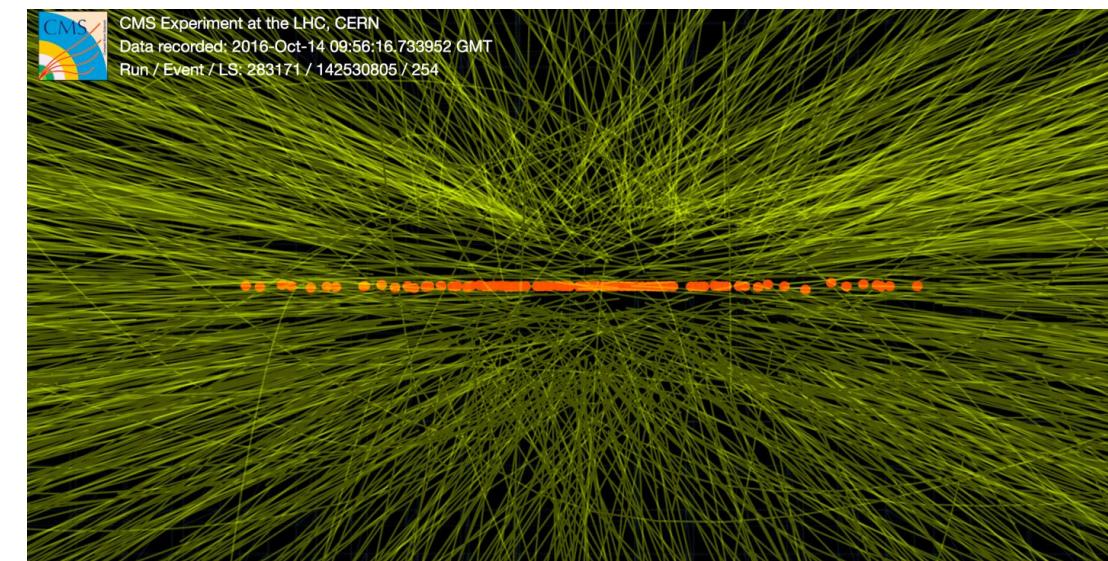
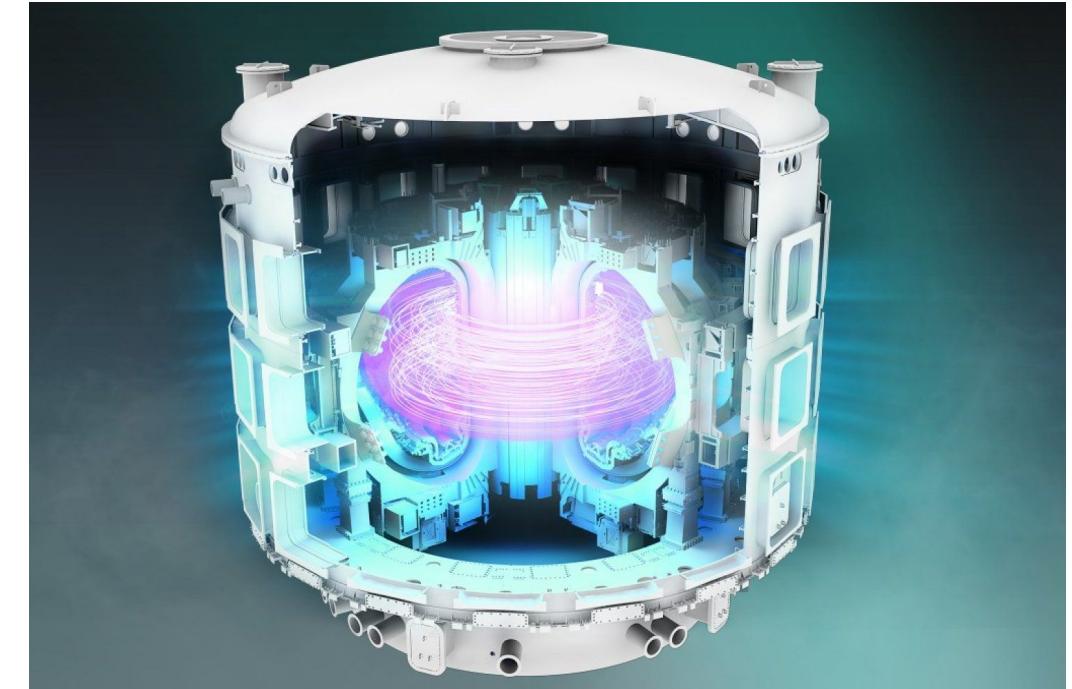
FLOP: floating point operation



# Motivation

- State of play: we are (becoming) an AI-dependent world... with constraints

1. Data growing exponentially
2. Limited (compute) resources
3. Limited time
4. Carbon footprint



# Motivation

- State of play: we are (becoming) an AI-dependent world... with constraints

1. Data growing exponentially

2. Limited (compute) resources

3. Limited time

4. Carbon footprint

## Carbon footprint comparison

Source: Strubell et al, 2019.

Training Transformer (big) w/ neural architecture search

626,155

Car, avg incl. fuel, 1 lifetime

126,000

Human life, avg, 1 year

11,023

Air travel, 1 passenger, NY-> SF

1,984

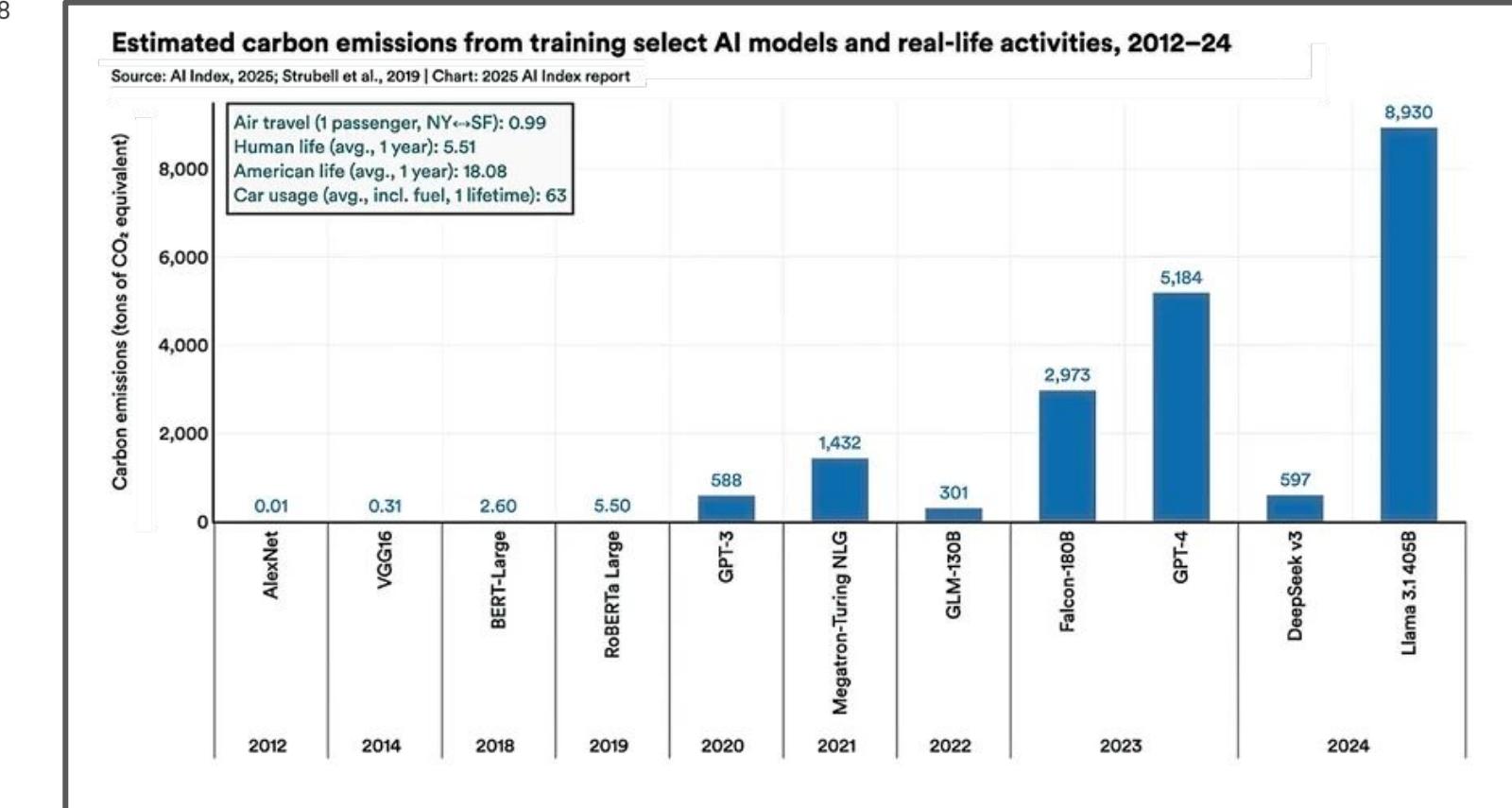
Training BERTbase on GPU

1,438

<https://arxiv.org/abs/1906.02243>

Google estimate of total energy use: 60% for inference, 40% for training

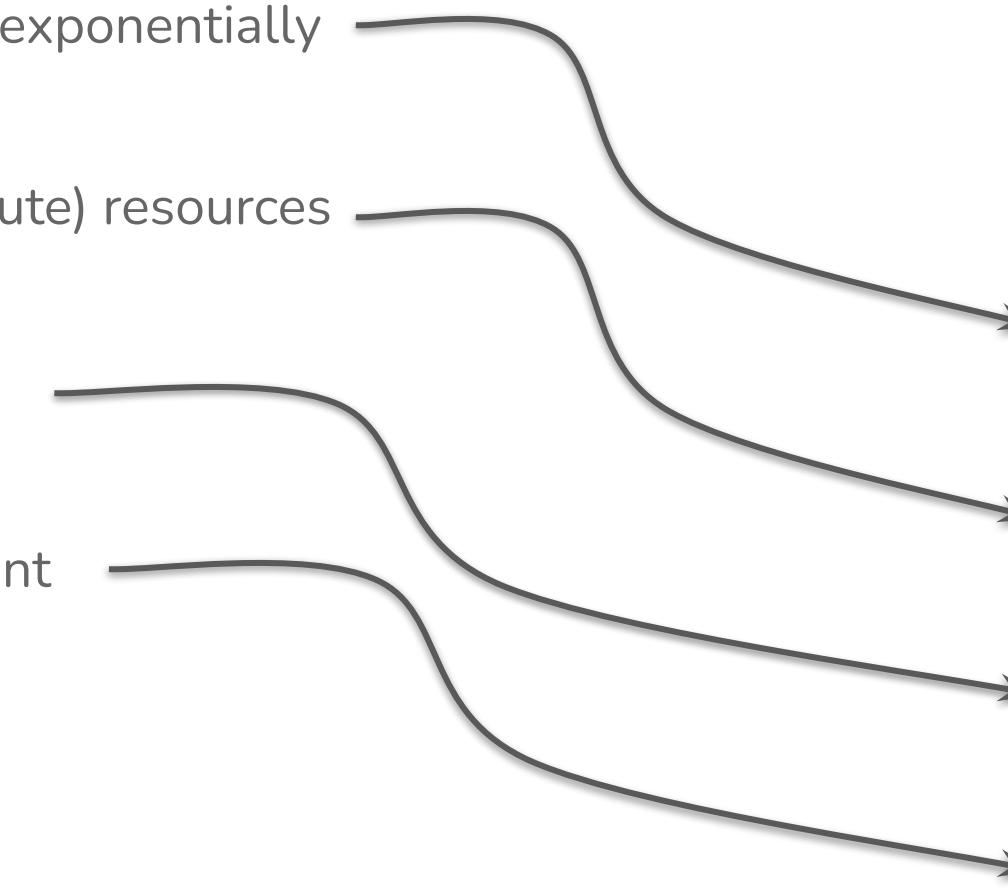
CO2 emissions (lbs)



# AI acceleration

- Increasing the speed and efficiency of AI inference on a separate acceleration device

1. Data growing exponentially
2. Limited (compute) resources
3. Limited time
4. Carbon footprint



## Aims

1. High throughput
2. Low resource-usage
3. Low latency
4. Energy efficient

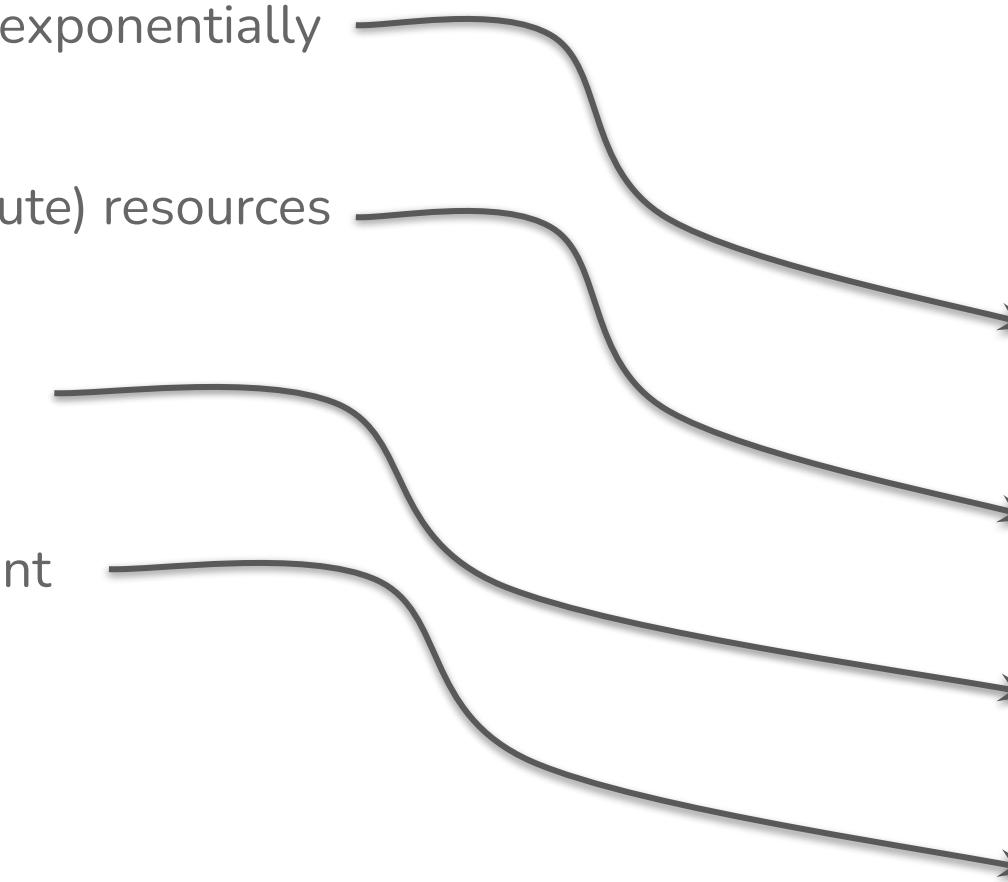
## Glossary

- *Throughput*: measure of rate that data flows through processing system
- *Latency*: time to process one item in the data

# AI acceleration

- Increasing the speed and efficiency of AI inference on a separate acceleration device

1. Data growing exponentially
2. Limited (compute) resources
3. Limited time
4. Carbon footprint



## Aims

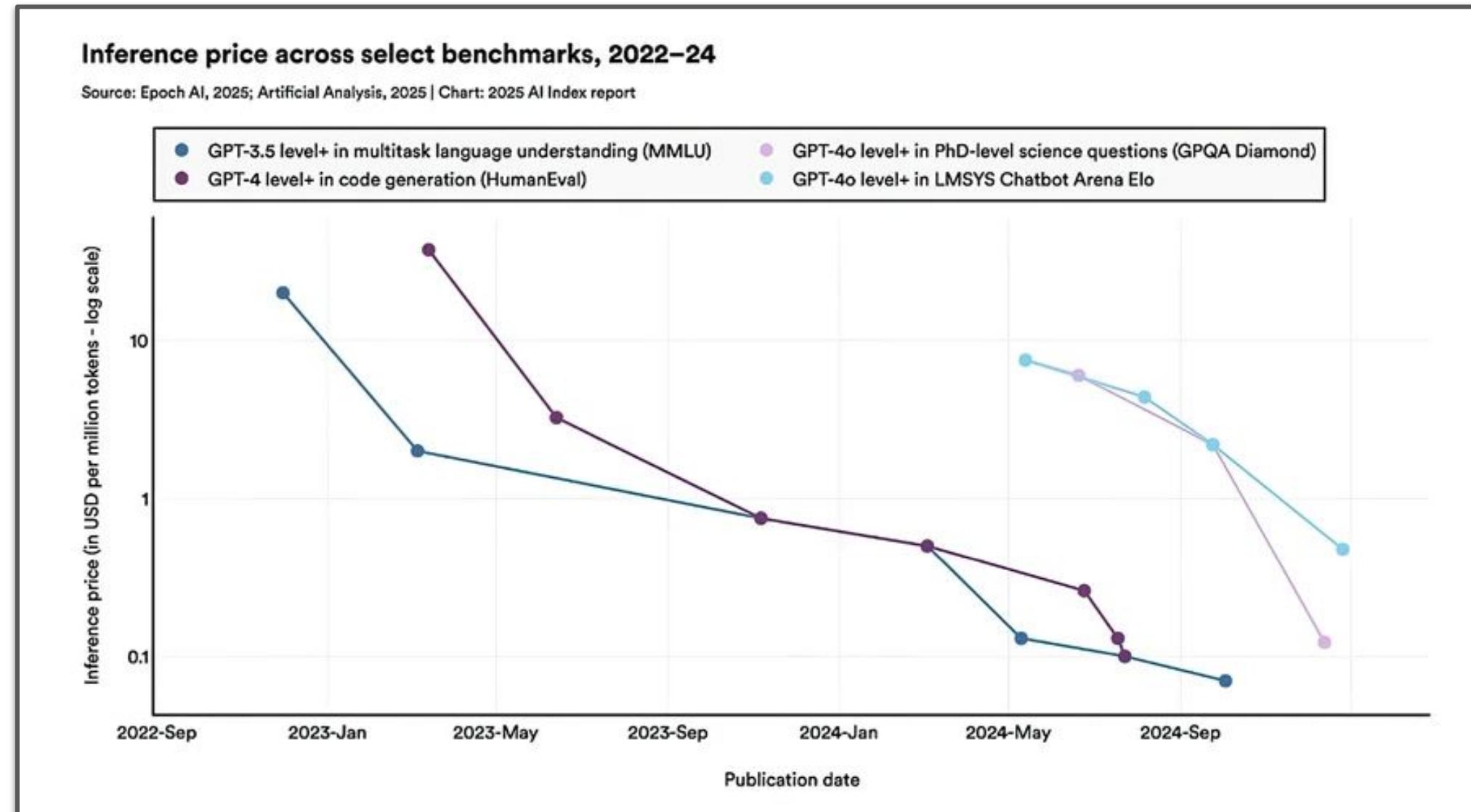
1. High throughput
2. Low resource-usage
3. Low latency
4. Energy efficient

Whilst maintaining  
performance!

## Glossary

- *Throughput*: measure of rate that data flows through processing system
- *Latency*: time to process one item in the data

# AI acceleration



- We are moving in the right direction → But plenty scope for improvement!
- Today we will learn how to “accelerate” a neural network with Field Programmable Gate Arrays (FPGAs)

# Course outline

- Motivation ✓
- Field Programmable Gate Arrays (FPGAs)

- Key properties and components

- High-level synthesis for Machine Learning



- FastML @ the Large Hadron Collider

- Collider physics primer

- CMS Level-1 Trigger Upgrade project

- Tutorial: “Online” jet-tagging @ LHC



- Converting NN to FPGA firmware

- Quantization-Aware training

- Pruning (compression)

# Course outline

- Motivation ✓
- Field Programmable Gate Arrays (FPGAs)
  - Key properties and components
  - High-level synthesis for Machine Learning 
- FastML @ the Large Hadron Collider
  - Collider physics primer
  - CMS Level-1 Trigger Upgrade project

~1 hour

- Tutorial: “Online” jet-tagging @ LHC 
  - Converting NN to FPGA firmware
  - Quantization-Aware training
  - Pruning (compression)

# Course outline

- Motivation ✓
- Field Programmable Gate Arrays (FPGAs)

- Key properties and components

- High-level synthesis for Machine Learning



- FastML @ the Large Hadron Collider

- Collider physics primer

- CMS Level-1 Trigger Upgrade project

~1 hour

- Tutorial: “Online” jet-tagging @ LHC



- Converting NN to FPGA firmware

- Quantization-Aware training

- Pruning (compression)

~3 hours

# Course outline

- Motivation ✓
- Field Programmable Gate Arrays (FPGAs)
  - Key properties and components
  - High-level synthesis for Machine Learning
- FastML @ the Large Hadron Collider
  - Collider physics primer
  - CMS Level-1 Trigger Upgrade project



~1 hour

- Tutorial: “Online” jet-tagging @ LHC The logo for Jupyter, consisting of three orange circles of increasing size forming a triangle, with the word "jupyter" written vertically next to it.
  - Converting NN to FPGA firmware
  - Quantization-Aware training
  - Pruning (compression)

~3 hours

## Course Assessment

Throughout tutorial, there are a number of **exercises** (highlighted in green). Alongside the exercises, you will see sets of questions. Some of these are **assessed** (highlighted in red), some of these are **unassessed** (highlighted in orange)

**Both are important for your understanding**

You will need to fill in your answers to the assessed questions in this Google Forms:

<https://forms.gle/2rugKQ1gvNbPTBwn9>

The assessment is out of 30 marks.

# Course outline

- Motivation ✓
- Field Programmable Gate Arrays (FPGAs)
  - Key properties and components
  - High-level synthesis for Machine Learning
- FastML @ the Large Hadron Collider
  - Collider physics primer
  - CMS Level-1 Trigger Upgrade project



~1 hour

- Tutorial: “Online” jet-tagging @ LHC
  - Converting NN to FPGA firmware
  - Quantization-Aware training
  - Pruning (compression)



~3 hours

## Course Assessment

Throughout tutorial, there are a number of exercises (highlighted in green). Alongside the exercises, you will see sets of questions. Some of these are assessed (highlighted in red), some of these are unassessed (highlighted in orange)

Both are important for your understanding

You will need to fill in your answers to the assessed questions in this Google Forms:

<https://forms.gle/2rugKQ1gvNbPTBwn9>

The assessment is out of 30 marks.

## Course competition

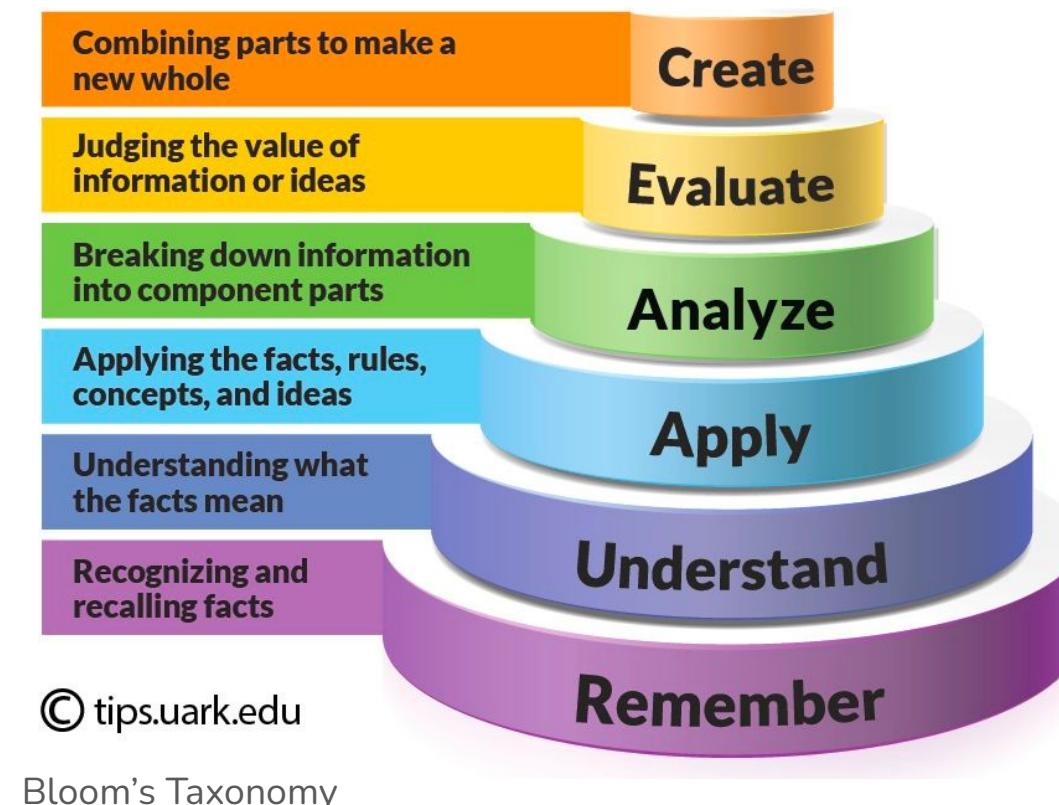
At the end of notebook, there is a competition. This is a small part of the assessment, and will be used to differentiate the top performing students.

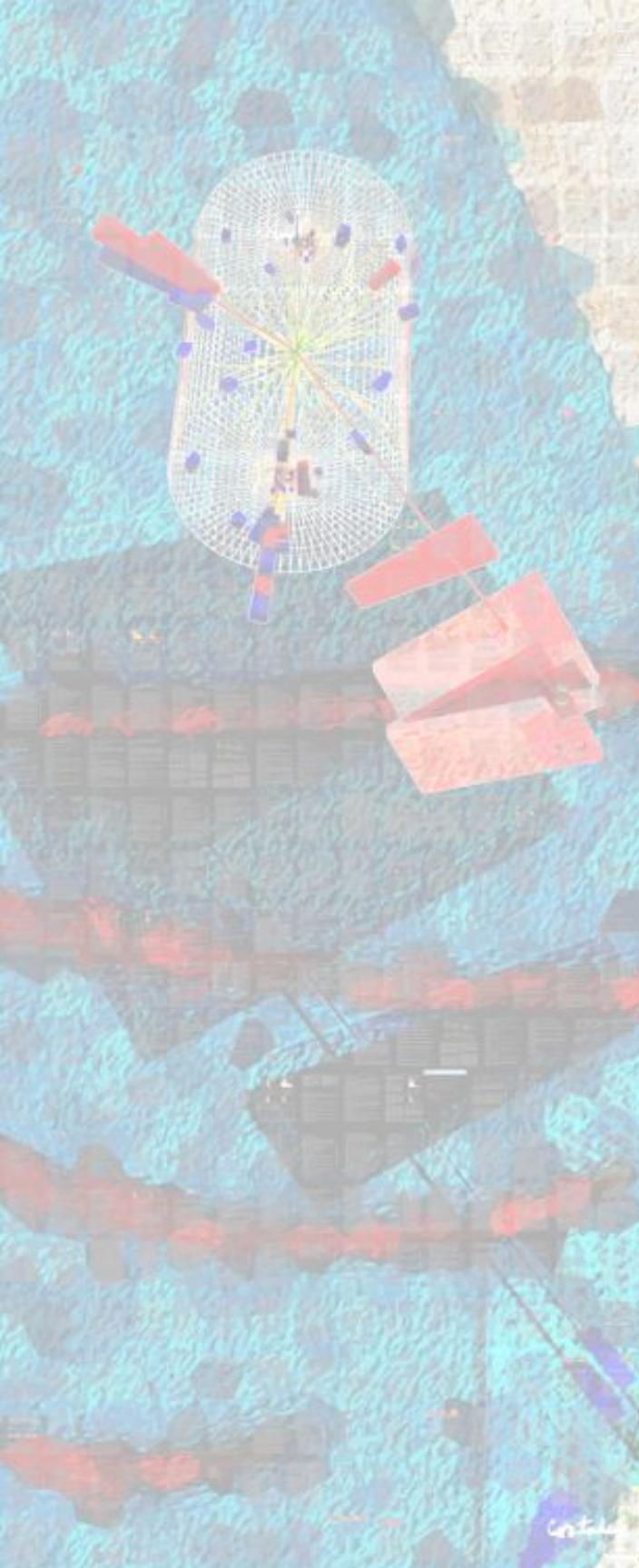


More details to follow.

# Learning outcomes (ILOs)

- **Understand** how to accelerate AI inference using FPGAs, and **apply** this to a jet classification algorithm at the LHC
- **Analyze** and **evaluate** the performance and efficiency of the FPGA algorithm, implementing techniques such as quantization-aware training and pruning (compression)
- **Create** an efficient NN design based on what you have learnt throughout the course

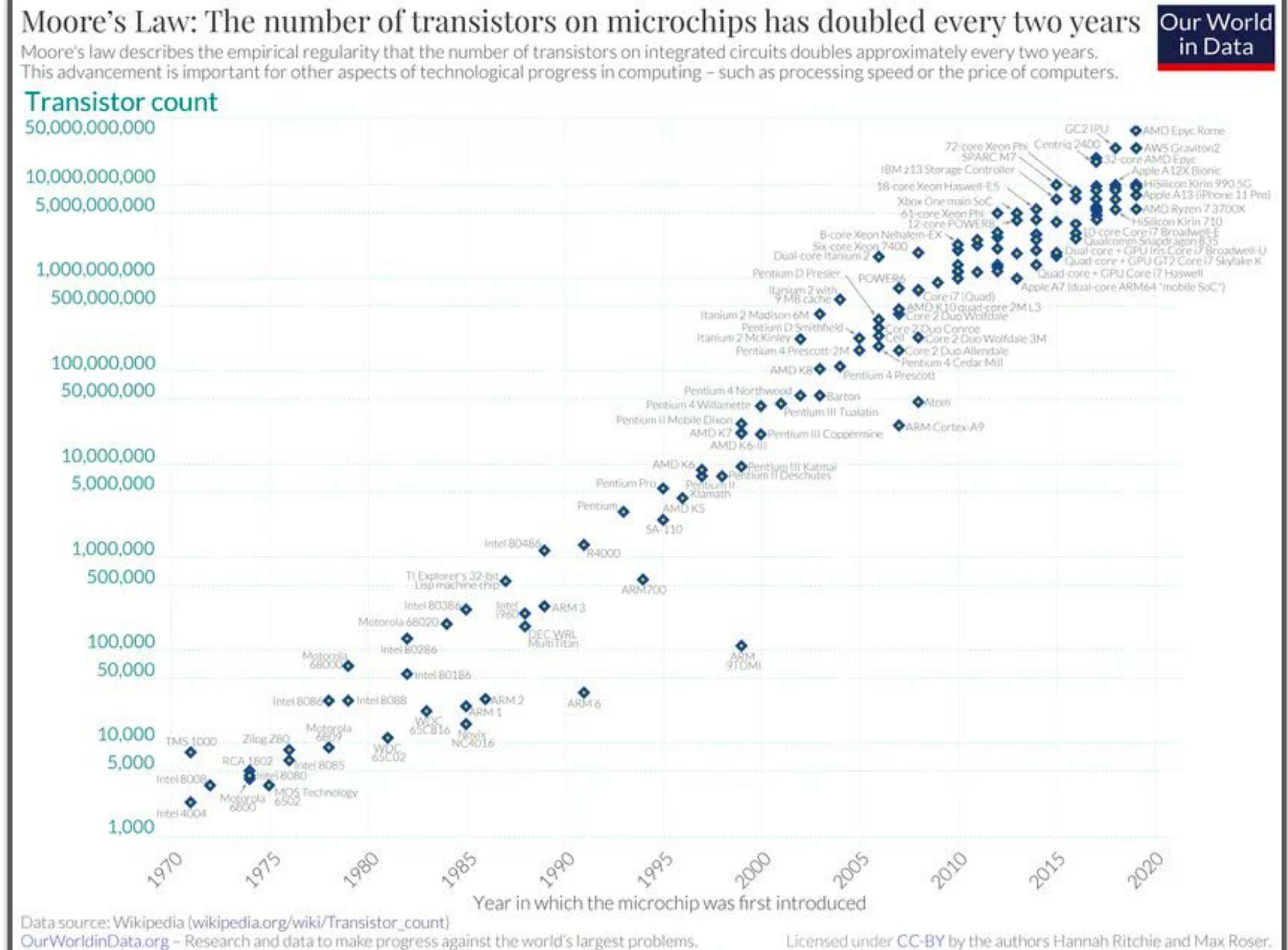




# Field programmable gate arrays (FPGAs)

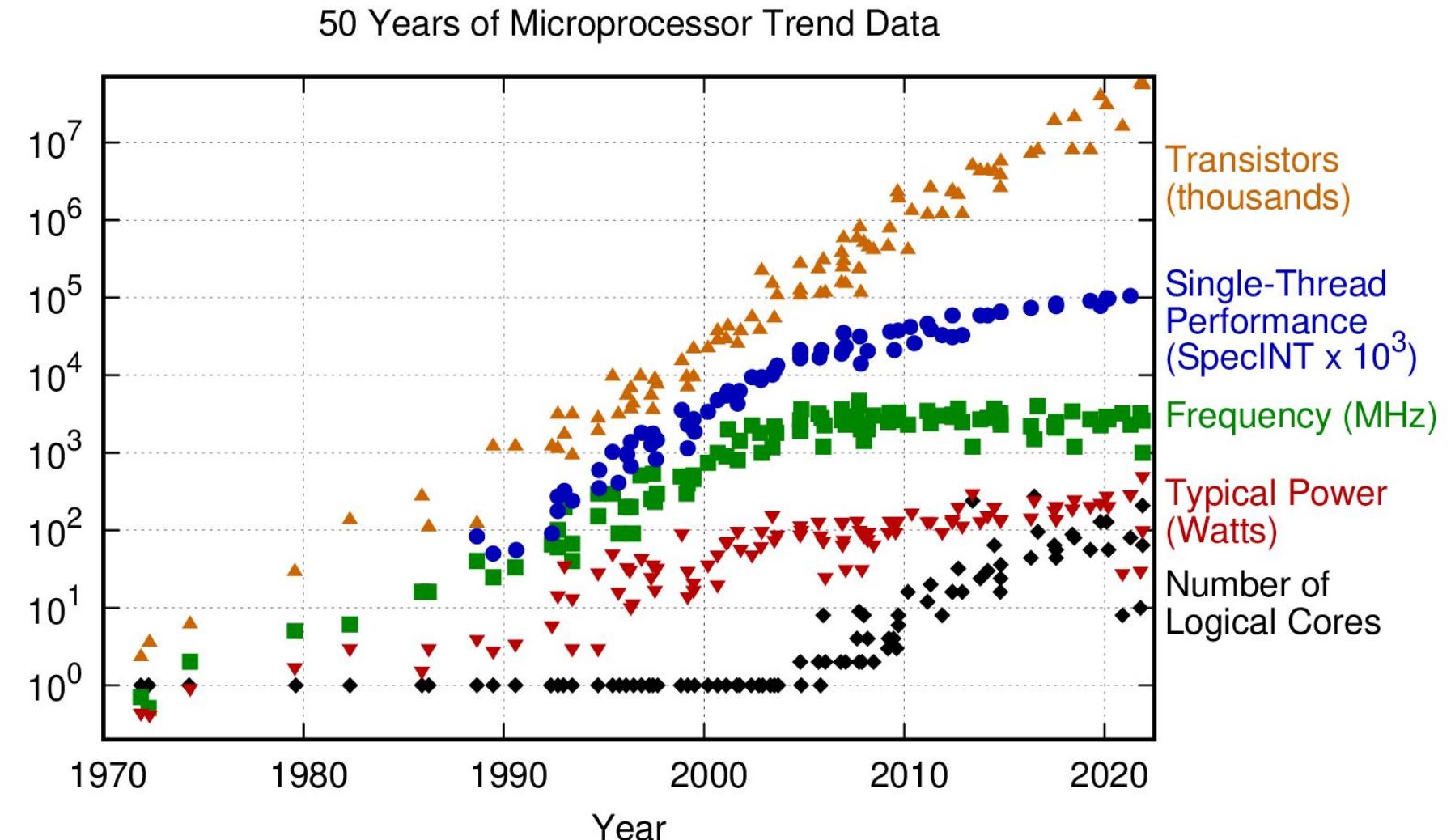
# Moore's law

- Number of transistors on chip doubles every two years
- This is not the end of the story...



# Moore's law

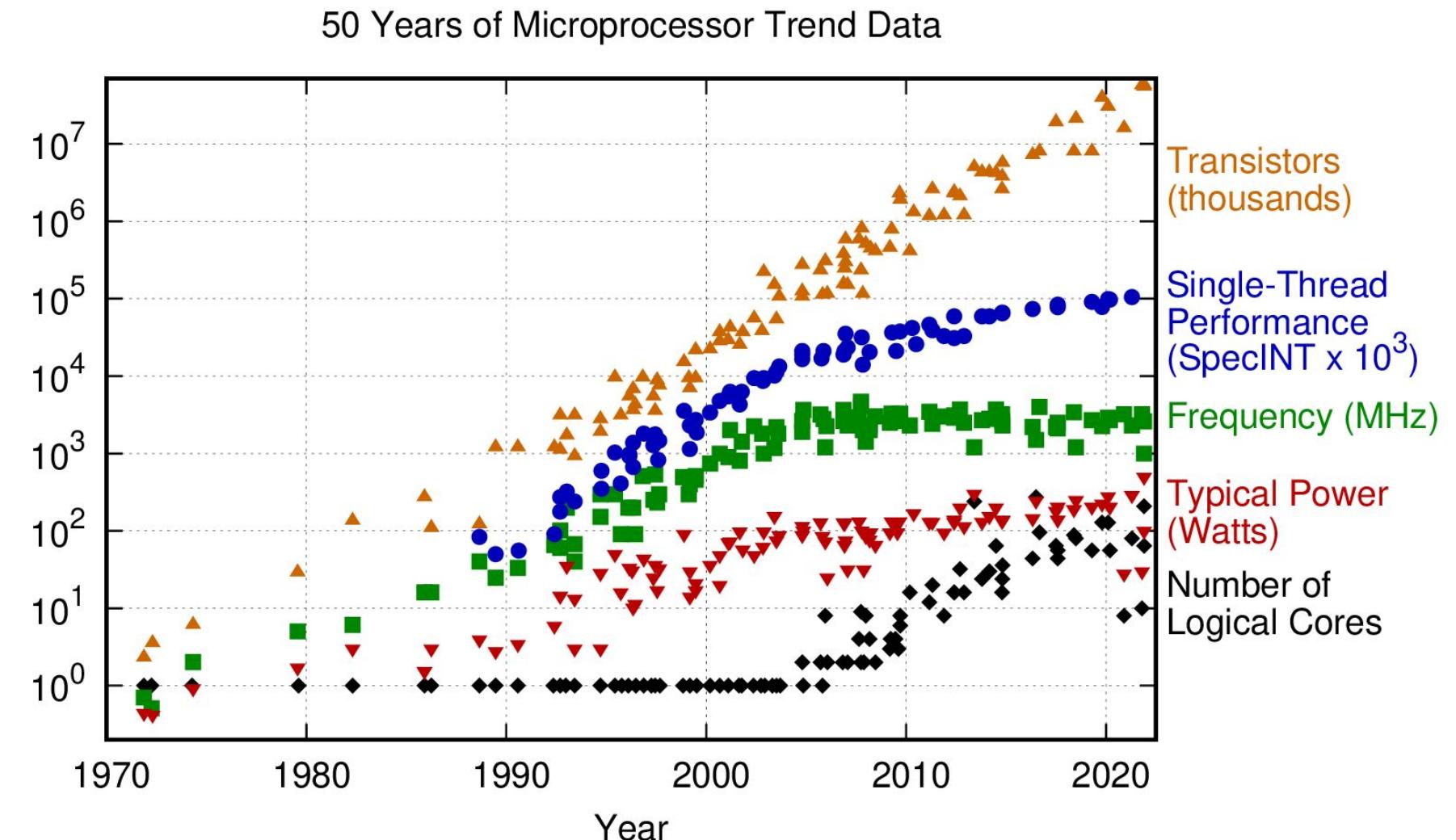
- Number of transistors on chip doubles every two years
- This is not the end of the story...
- Performance i.e. clock frequency and power began to plateau around 2005
- Turned our attention to increasing the number of cores → strong increase since 2005



<https://github.com/karlrupp/microprocessor-trend-data>

# Moore's law

- Number of transistors on chip doubles every two years
- This is not the end of the story...
- Performance i.e. clock frequency and power began to plateau around 2005
- Turned our attention to increasing the number of cores → strong increase since 2005



A different programming paradigm:

Parallelism!

# Architectures for AI

## CPUs

Central Processing Unit



- General purpose, highly versatile
- Excellent software ecosystem
- Limited parallelism for deep learning workloads
- Lower throughput
- Not power efficient at high load

# Architectures for AI

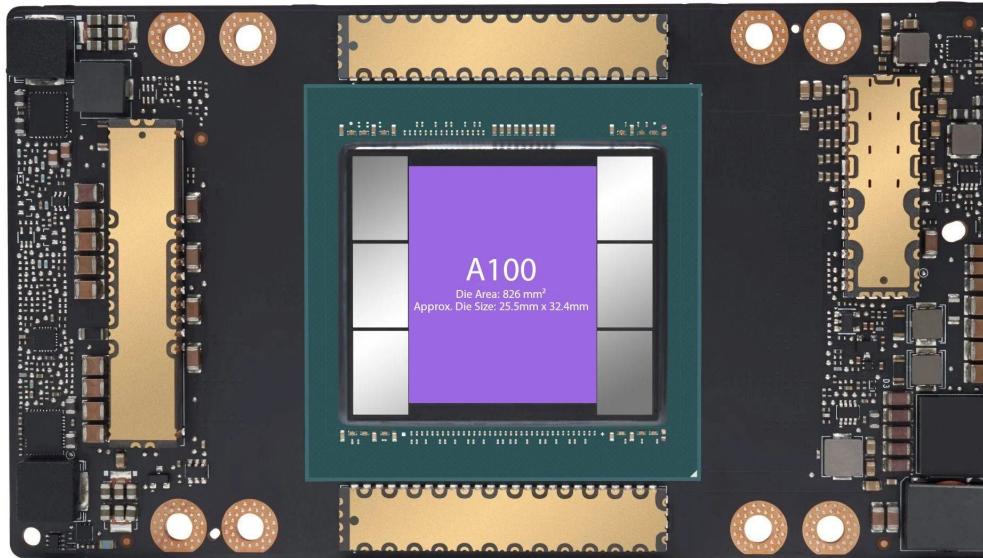
## CPUs

Central Processing Unit



## GPUs

Graphics Processing Unit



- General purpose, highly versatile
- Excellent software ecosystem
- Limited parallelism for deep learning workloads
- Lower throughput
- Not power efficient at high load
- Massive parallelism (1000s of cores)
- Optimized for matrix operations
- Mature AI libraries (cuDNN, TensorRT)
- High power consumption
- Latency can be unpredictable
- Fixed architecture

# Architectures for AI

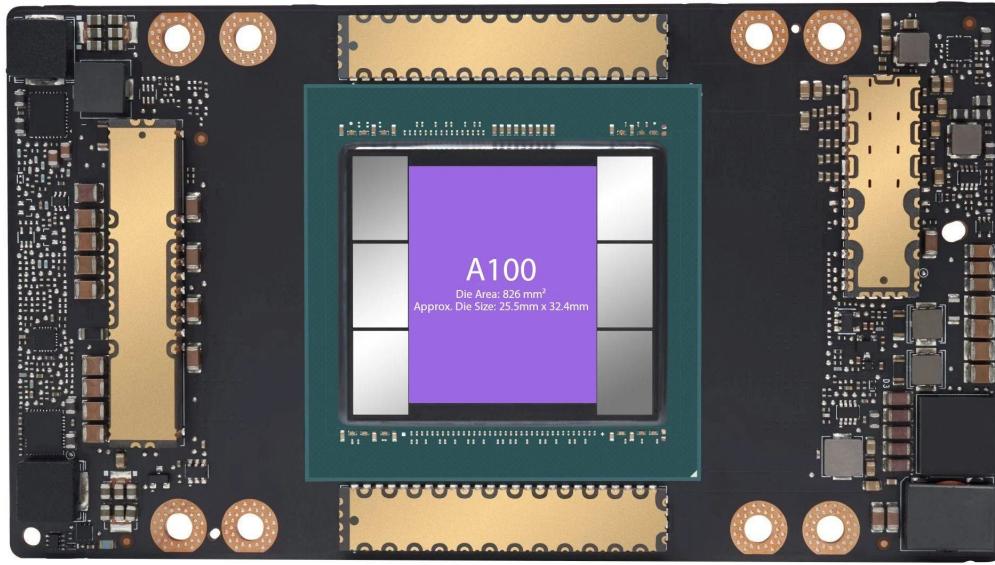
## CPUs

Central Processing Unit



## GPUs

Graphics Processing Unit



## FPGAs

Field-Programmable Gate Array



- General purpose, highly versatile
- Excellent software ecosystem
- Limited parallelism for deep learning workloads
- Lower throughput
- Not power efficient at high load

- Massive parallelism (1000s of cores)
- Optimized for matrix operations
- Mature AI libraries (cuDNN, TensorRT)
- High power consumption
- Latency can be unpredictable
- Fixed architecture

- Highly customizable (reconfigurable hardware)
- Ultra-low latency (suitable for real-time/edge AI)
- High energy efficiency
- Limited on-chip memory
- Tooling and programming complexity

# Architectures for AI

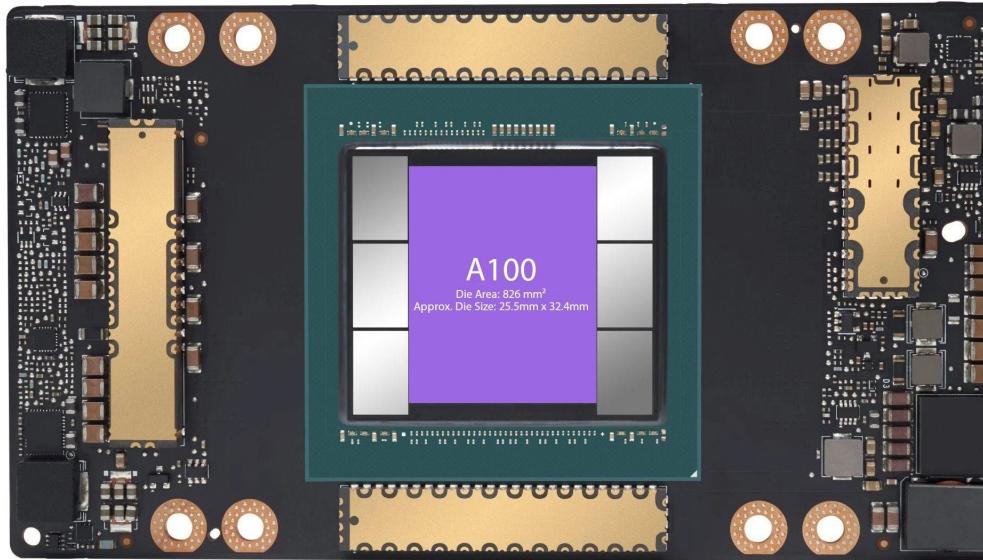
## CPUs

Central Processing Unit



## GPUs

Graphics Processing Unit



## FPGAs

Field-Programmable Gate Array

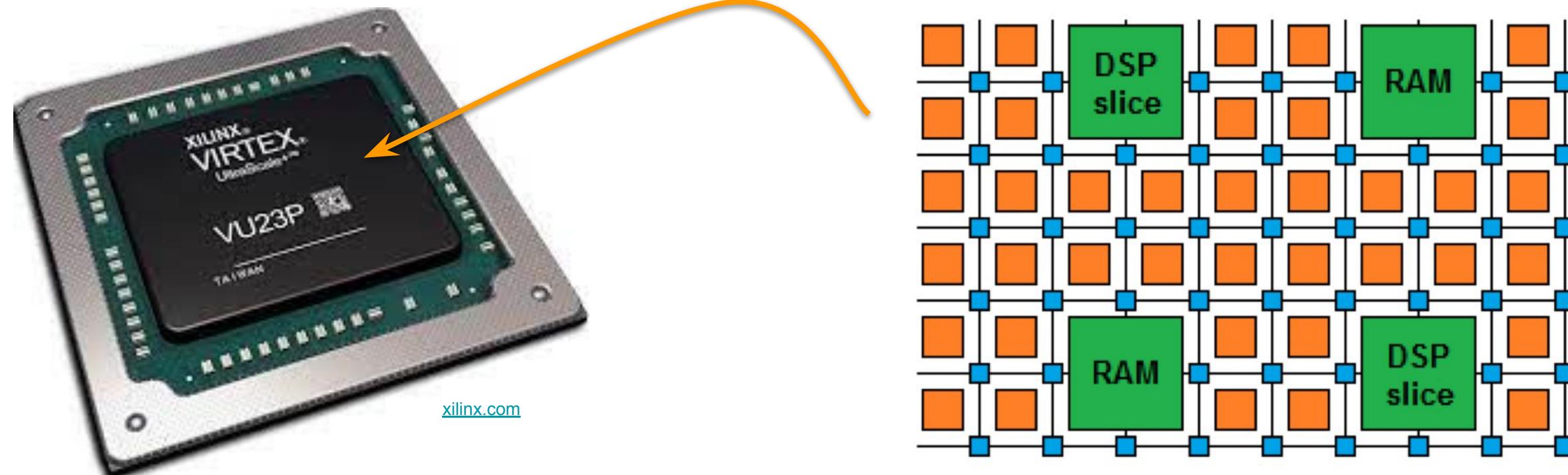


- General purpose, highly versatile
  - Excellent software ecosystem
  - Limited parallelism (thousands of cores)
  - Lower throughput
  - Not power efficient
  - Fixed architecture
- With frameworks like Intel's oneAPI and hls4ml, FPGA development for AI is becoming more accessible
- We will use both these tools in the course!

- Massive parallelism (1000s of cores)
- Optimized for specific operations
- Highly customizable (reconfigurable hardware)
- Ultra-low latency (suitable for real-time/edge AI)
- High energy efficiency
- Limited on-chip memory
- Tooling and programming complexity

# Field Programmable Gate Arrays (FPGAs)

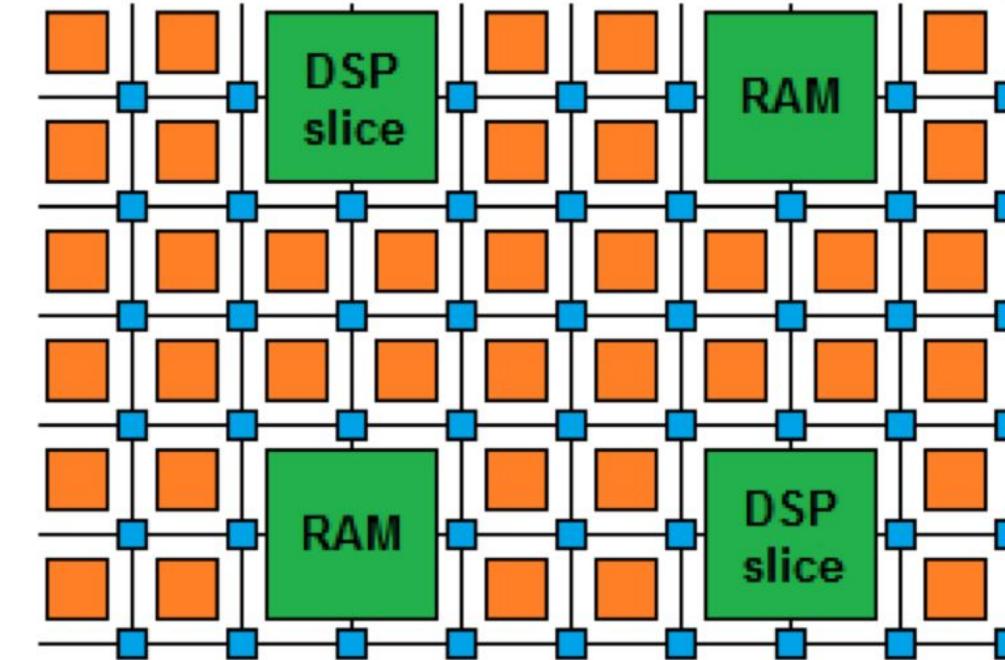
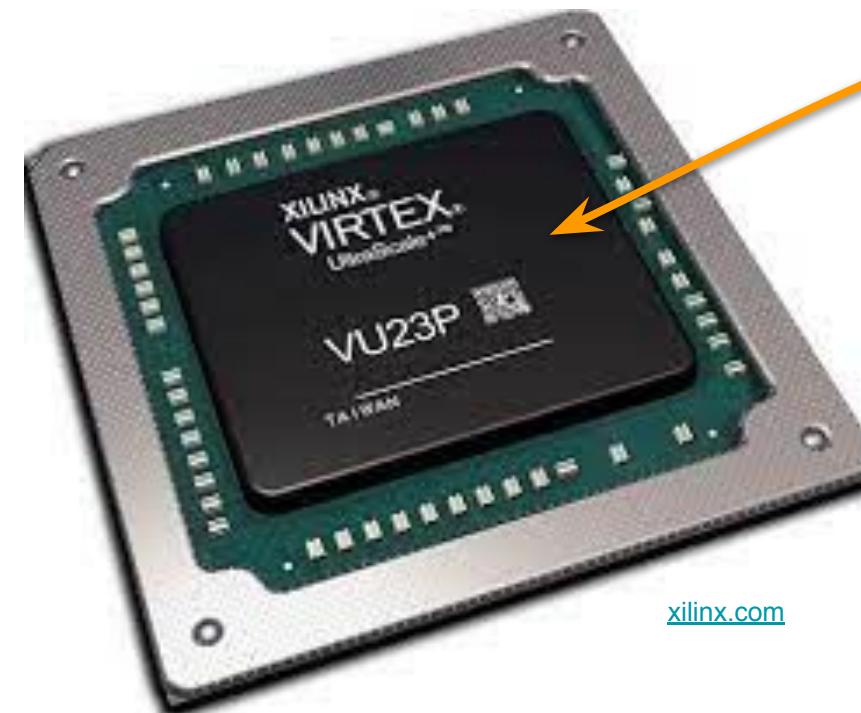
- FPGAs are reprogrammable integrated circuits → “a blank canvas”
- Contain many different building blocks (“resources”) which are connected together as you desire
  - Huge flexibility in defining circuitry (functionality), clocking etc ← (Trade off) → Steep learning curve



- Two-dimensional array of:
  1. Programmable logic blocks
  2. Memory
  3. Programmable interconnects

# Field Programmable Gate Arrays (FPGAs)

- FPGAs are reprogrammable integrated circuits → “a blank canvas”
- Contain many different building blocks (“resources”) which are connected together as you desire
  - Huge flexibility in defining circuitry (functionality), clocking etc ← (Trade off) → Steep learning curve

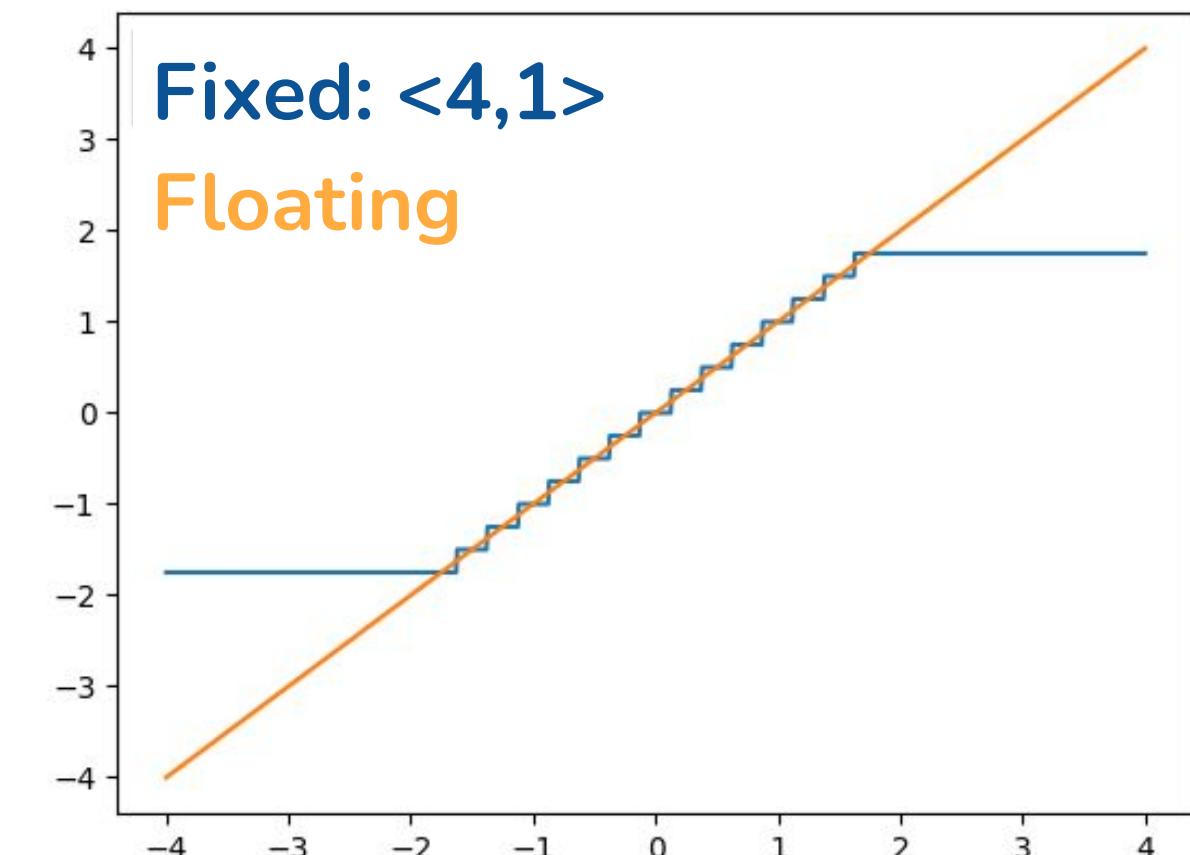
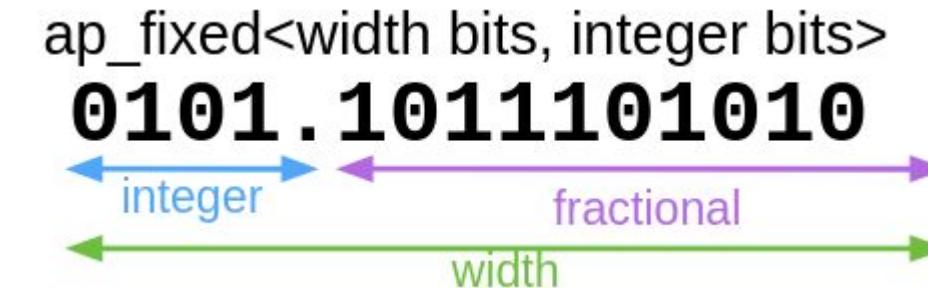


- Two-dimensional array of:
  1. Programmable logic blocks
  2. Memory
  3. Programmable interconnects

Before going through resources in more detail, there's two key properties that need to be introduced...

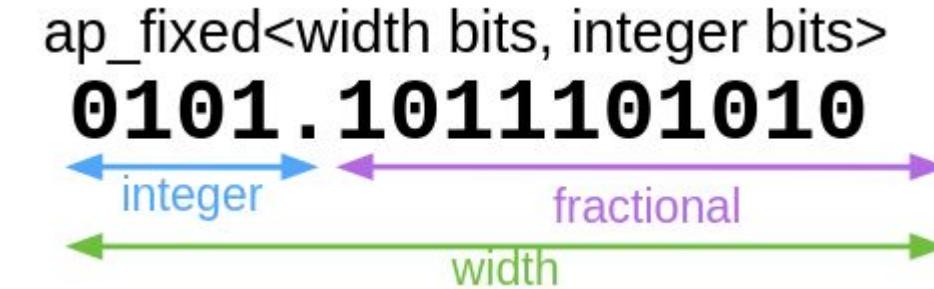
# Fixed-point precision

- Representation of a number with a fixed number of bits allocated to the integer and fractional parts
  - c.f. Floating-point represents number with mantissa+exponent → wider dynamic range



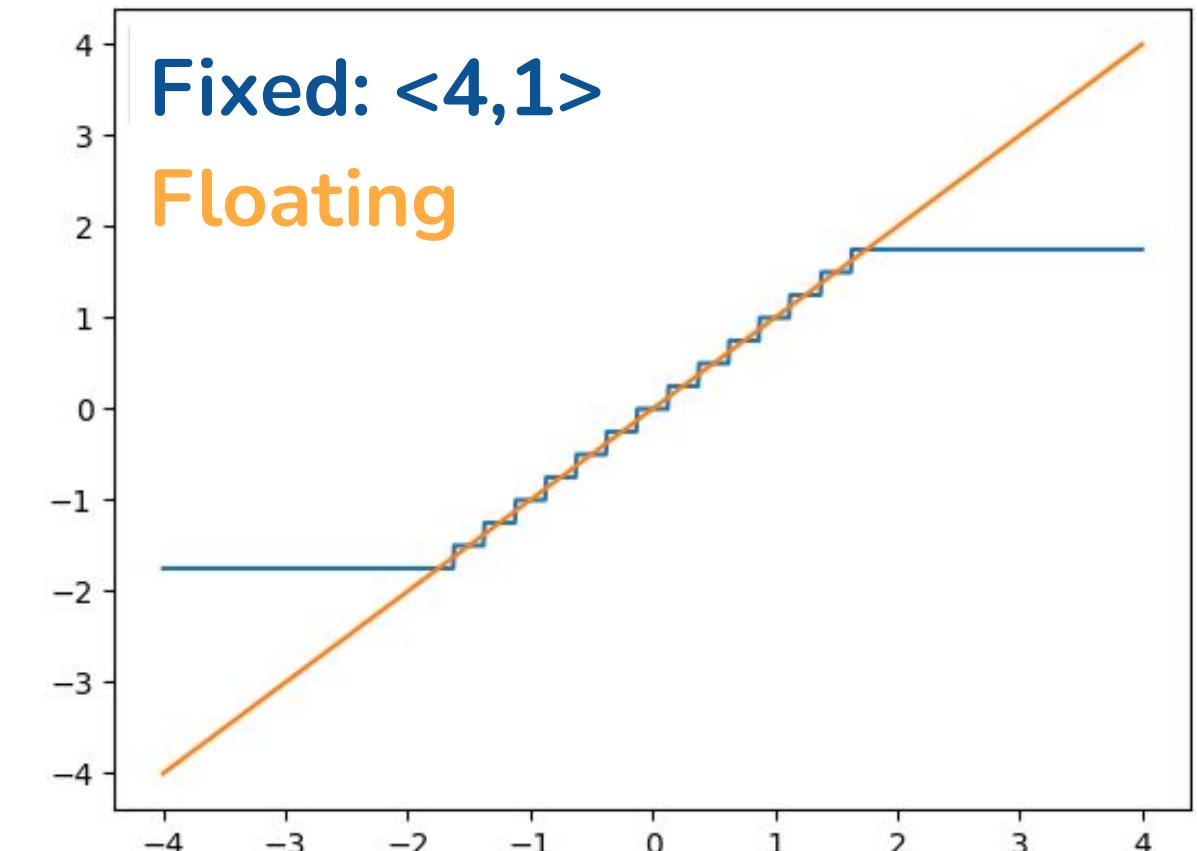
# Fixed-point precision

- Representation of a number with a fixed number of bits allocated to the integer and fractional parts
  - c.f. Floating-point represents number with mantissa+exponent → wider dynamic range



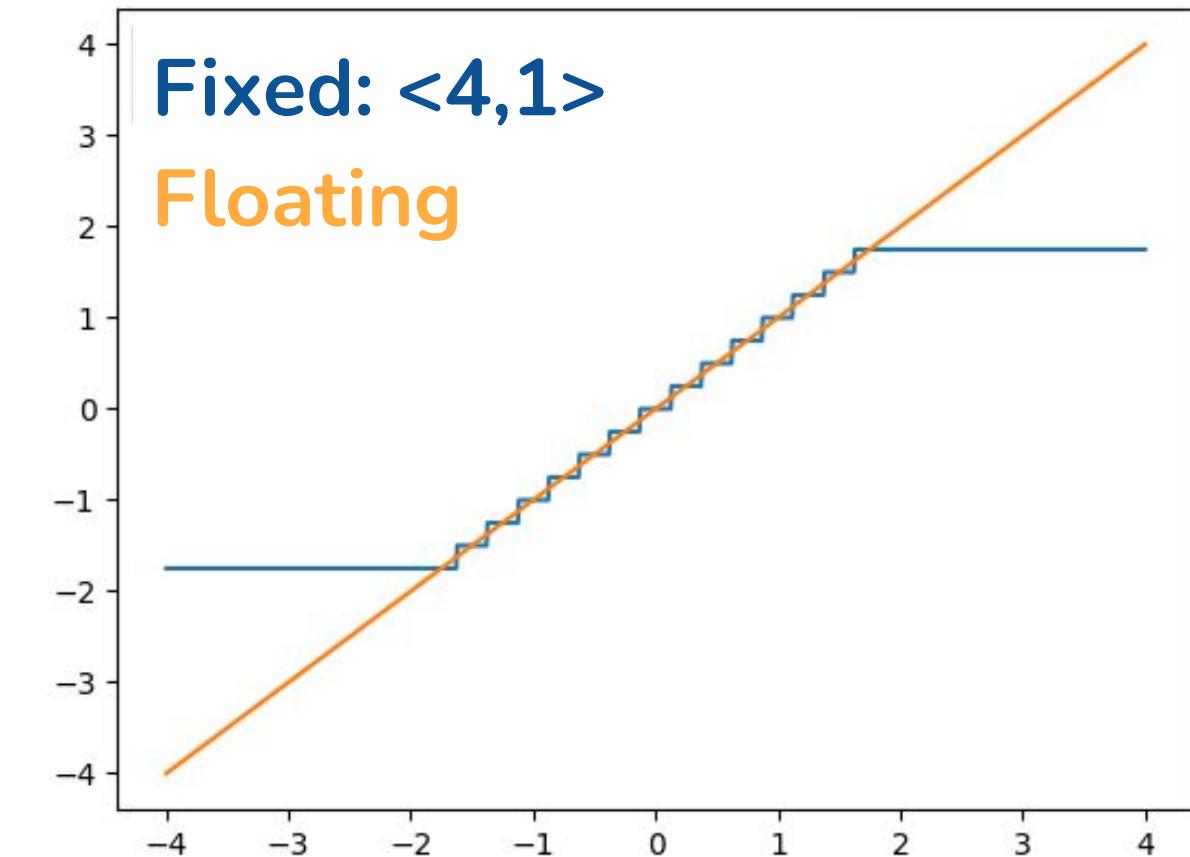
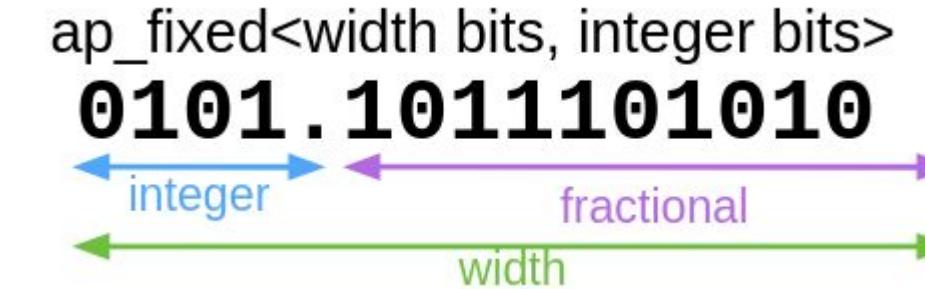
- FPGAs are customizable at the bit-level

- Avoids unnecessary precision (lower resources + power consumption)
- Faster computation
- Deterministic latency i.e. consistent timing (important for real-time systems)



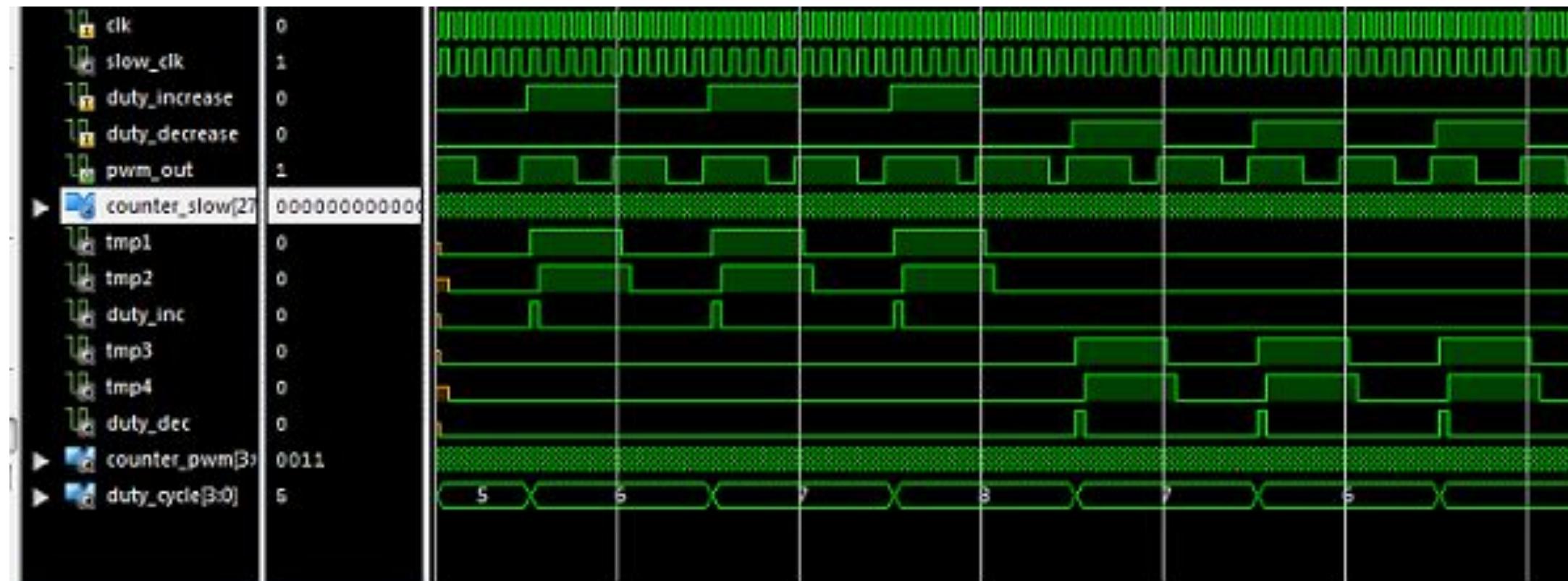
# Fixed-point precision

- Representation of a number with a fixed number of bits allocated to the integer and fractional parts
  - c.f. Floating-point represents number with mantissa+exponent → wider dynamic range
- FPGAs are customizable at the bit-level
  - Avoids unnecessary precision (lower resources + power consumption)
  - Faster computation
  - Deterministic latency i.e. consistent timing (important for real-time systems)
- We will even see how to take advantage of this property for neural networks using “Quantization-aware training”



# Fixed clock cycle

- FPGA defined by fixed-frequency clock (e.g. 200 MHz) → All logic is timed to that clock
  - i.e. operations happen at predictable, regular intervals (unlike CPUs/GPUs which operate with dynamic clock)



- Along with fixed-point precision, this means we have full control of the data flow

# FPGAs resources

- Logic cells

- Look Up Tables (LUTs)

Perform arbitrary functions on small bitwidth inputs (2-6)

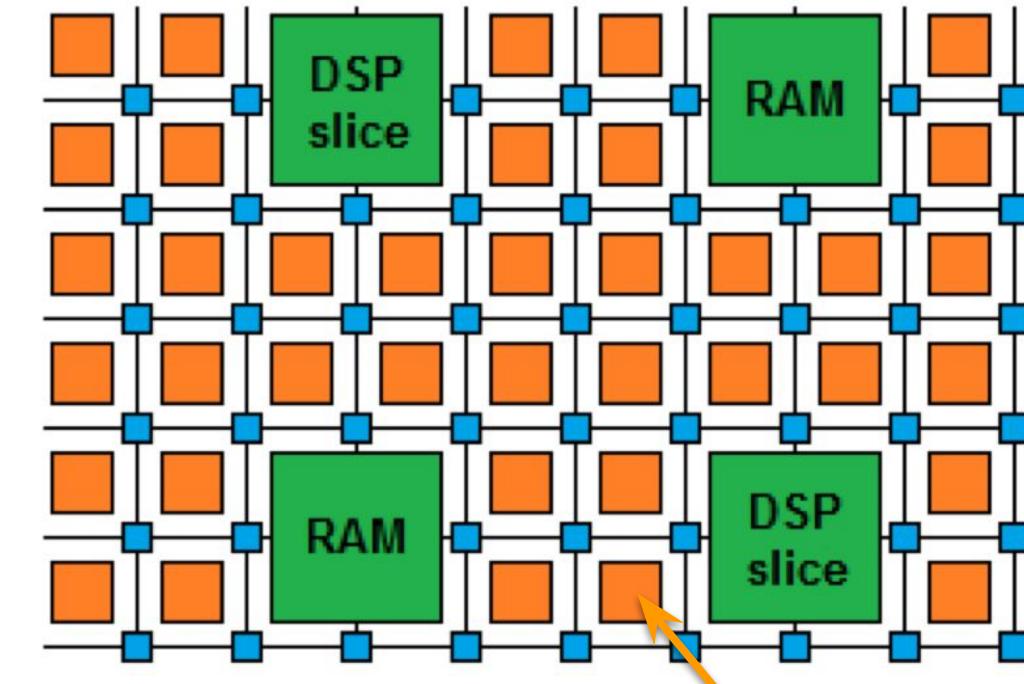
These can be used for boolean operations, arithmetic, ...

- Flip-Flops (FFs)

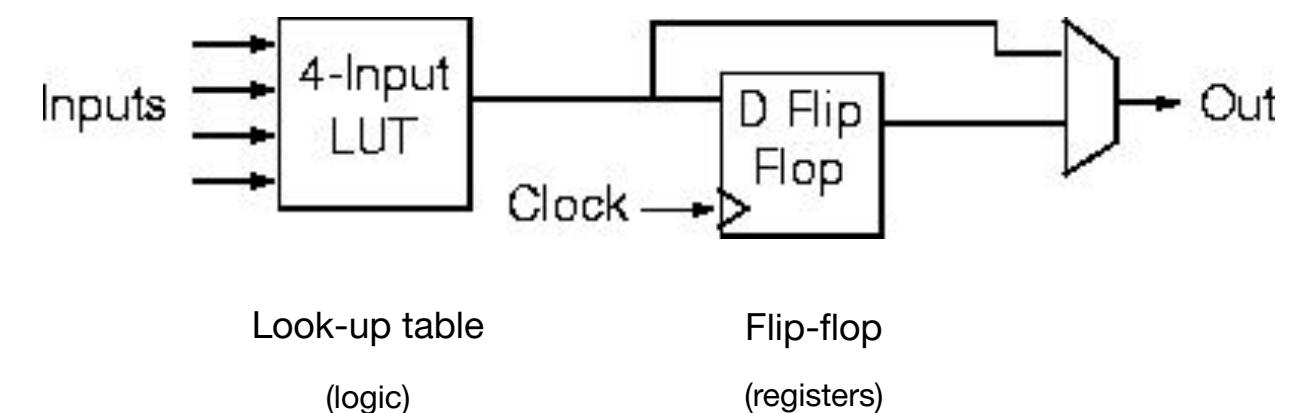
Register data in time with the clock pulse



Two-dimensional array



Logic cells



# FPGAs resources

- Logic cells

- Look Up Tables (LUTs)

Perform arbitrary functions on small bitwidth inputs (2-6)

These can be used for boolean operations, arithmetic, ...

- Flip-Flops (FFs)

Register data in time with the clock pulse

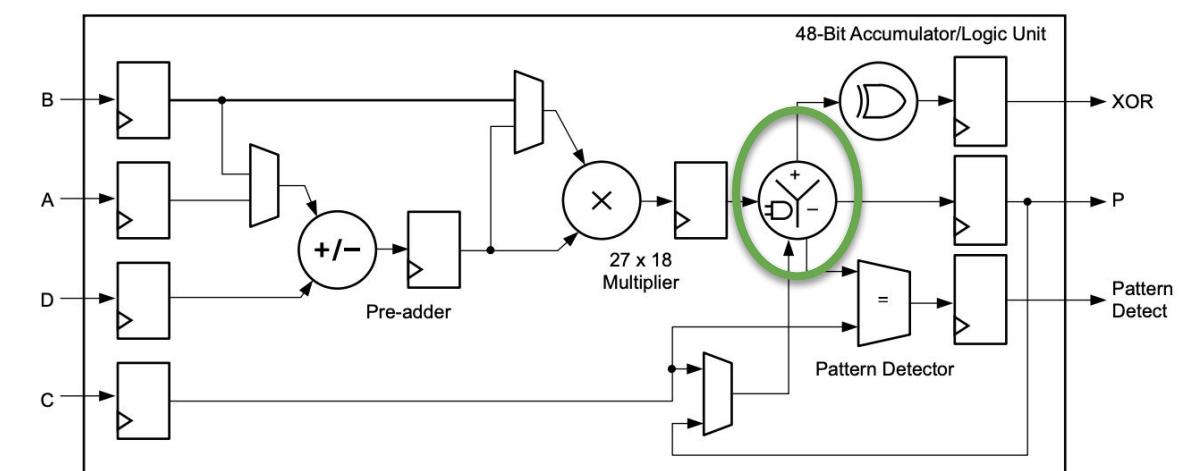
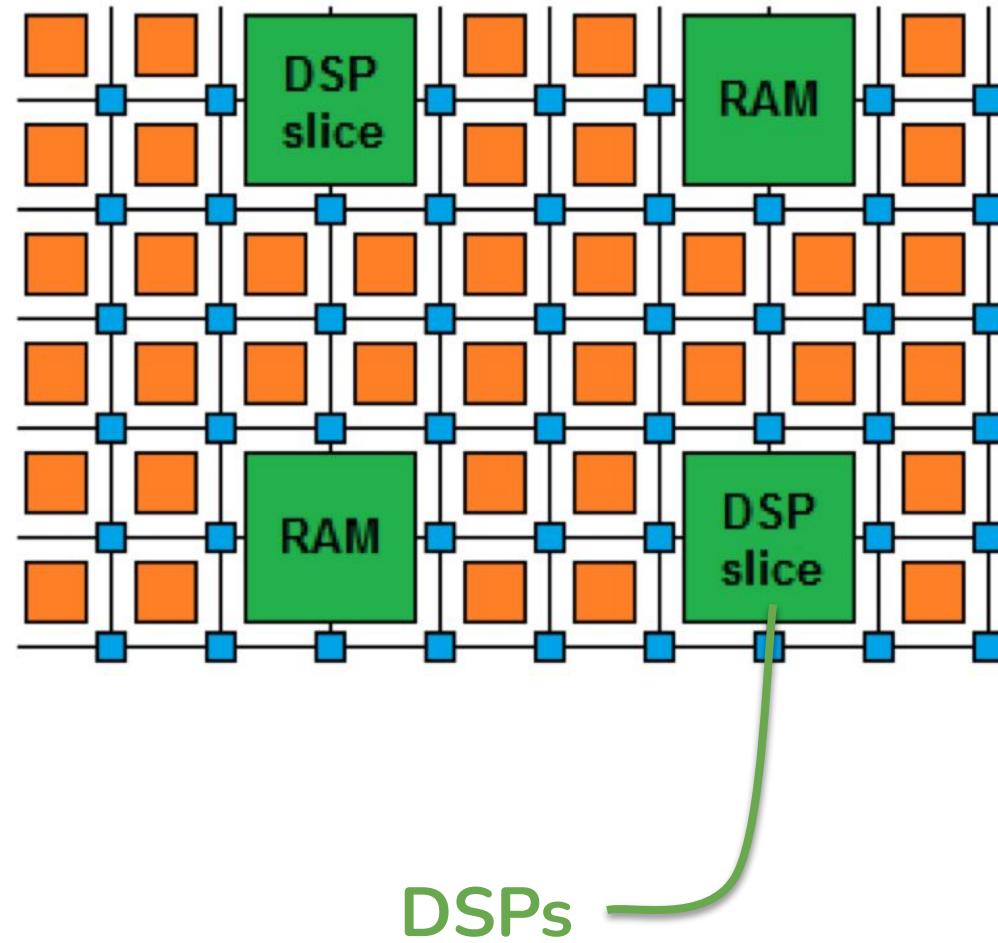
- Digital Signal Processors (DSPs)

Specialized units for multiplication/arithmetic (useful for NNs)

Faster and more efficient than LUTs for these types of operations



Two-dimensional array



# FPGAs resources

- Logic cells

- Look Up Tables (LUTs)

Perform arbitrary functions on small bitwidth inputs (2-6)

These can be used for boolean operations, arithmetic, ...

- Flip-Flops (FFs)

Register data in time with the clock pulse

- Digital Signal Processors (DSPs)

Specialized units for multiplication/arithmetic (useful for NNs)

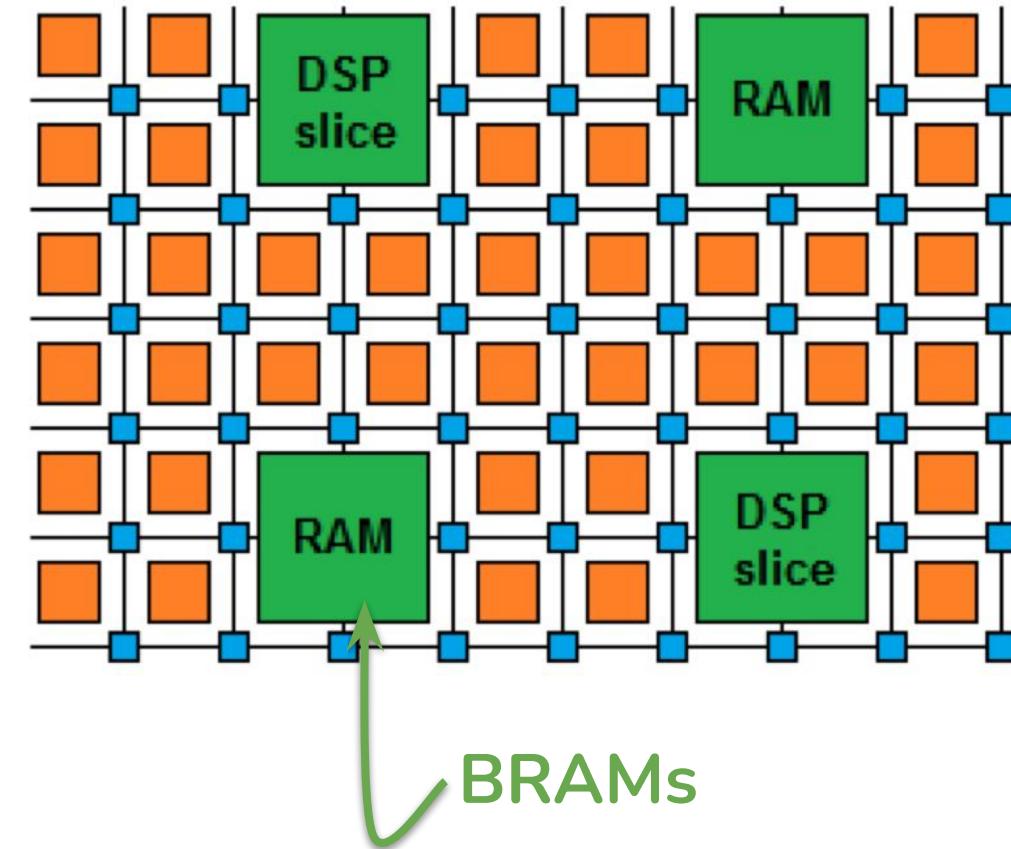
Faster and more efficient than LUTs for these types of operations

- BRAMs

Small fast memories (RAMs, ROMs, FIFOs) → More efficient than logic cells



Two-dimensional array



# FPGAs resources

- Logic cells

- Look Up Tables (LUTs)

Perform arbitrary functions on small bitwidth inputs (2-6)

These can be used for boolean operations, arithmetic, ...

- Flip-Flops (FFs)

Register data in time with the clock pulse

- Digital Signal Processors (DSPs)

Specialized units for multiplication/arithmetic (useful for NNs)

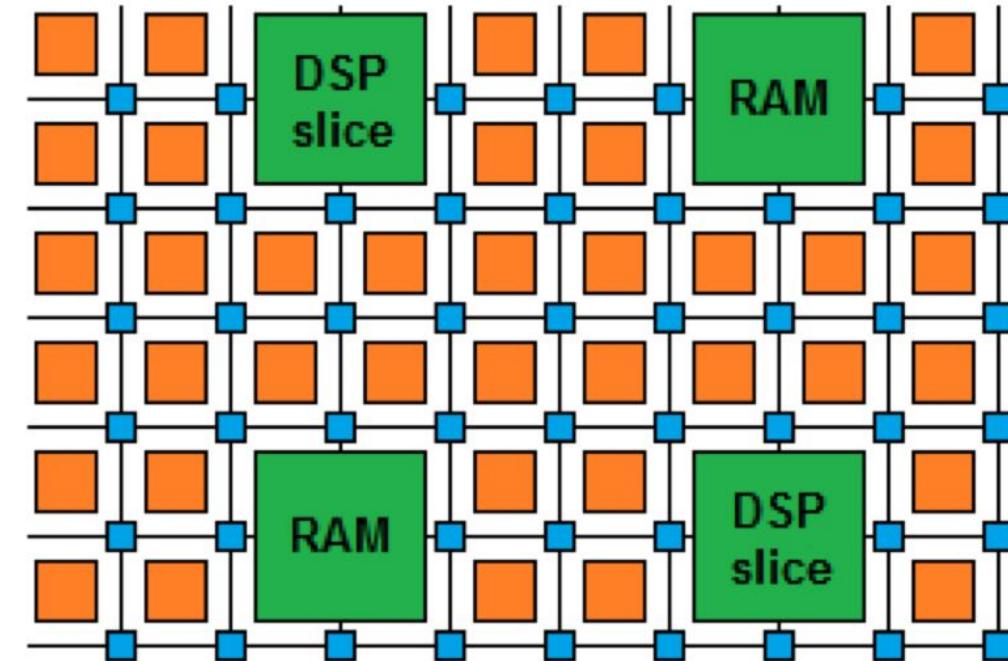
Faster and more efficient than LUTs for these types of operations

- BRAMs

Small fast memories (RAMs, ROMs, FIFOs) → More efficient than logic cells

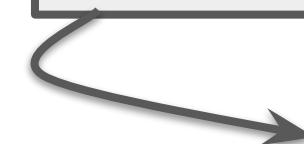


Two-dimensional array



**Xilinx Vertex Ultrascale+**

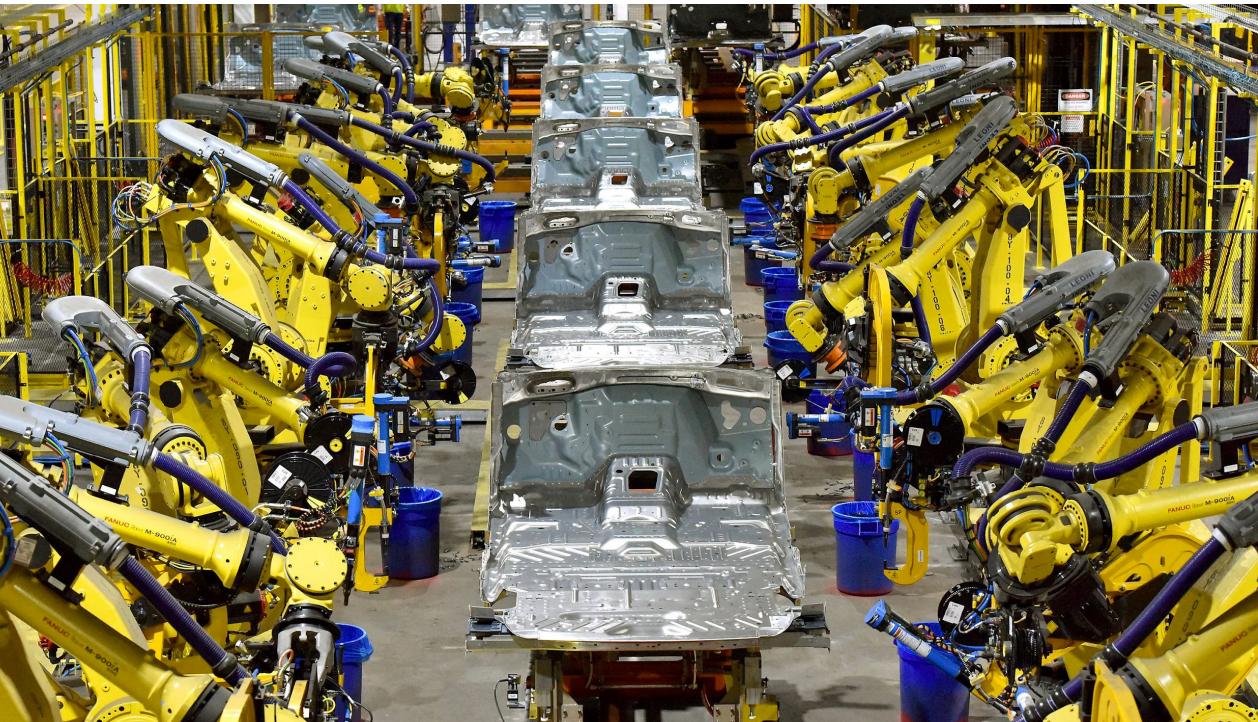
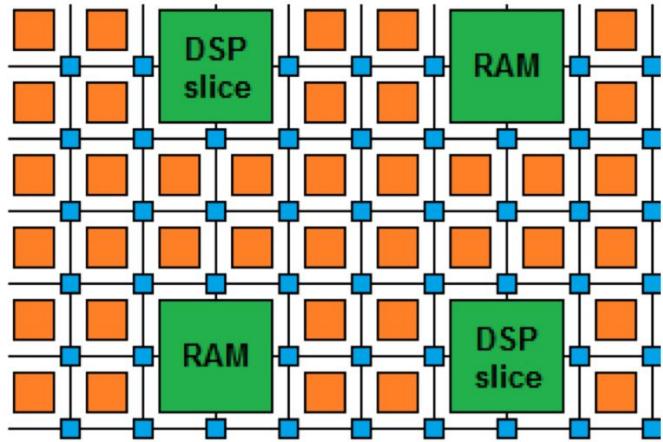
- 1.1M LUTs
- 2.3M FFs
- 6800 DSPs
- 260 Mb BRAM



**Supports highly-parallel  
algorithm implementations**

# Why are FPGAs fast?

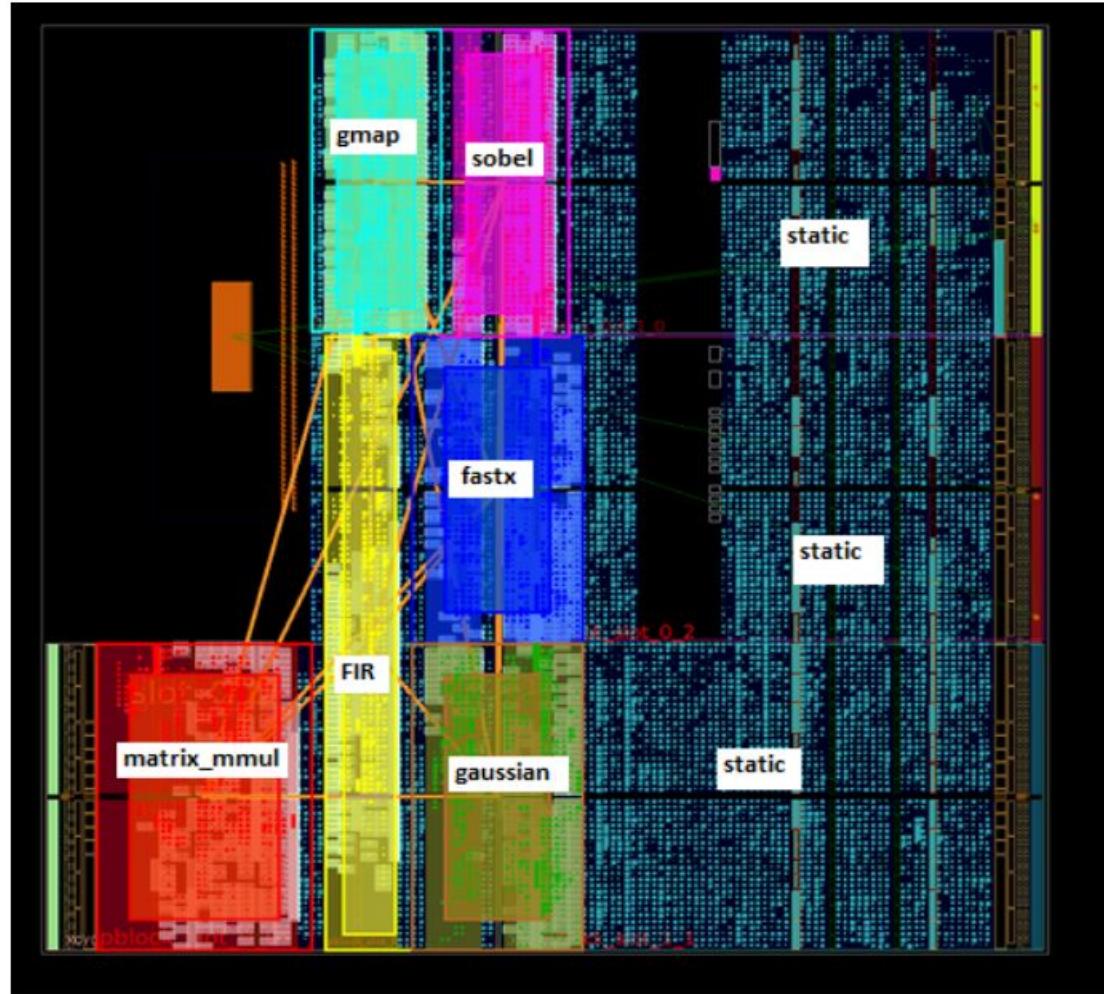
- Fine grained / **resource parallelism**
  - Use the many resources to work on different parts of the problem simultaneously
    - Allows us to achieve **low latency**
- Most problems have some sequential aspect
  - Limits how low in latency we can go
  - But we can still take advantage of it with...
- **Pipeline parallelism**
  - Complete control of the data flow (clock, precision) allows us to program the FPGA to work on different data simultaneously
    - Allows us to achieve **high throughput**



Like a production line for data

# FPGA algorithm design

Completely different to normal programming → think in terms of “building structures” and moving data through chip in 2D



<https://retis.sssup.it/~a.biondi/papers/CODES19.pdf>

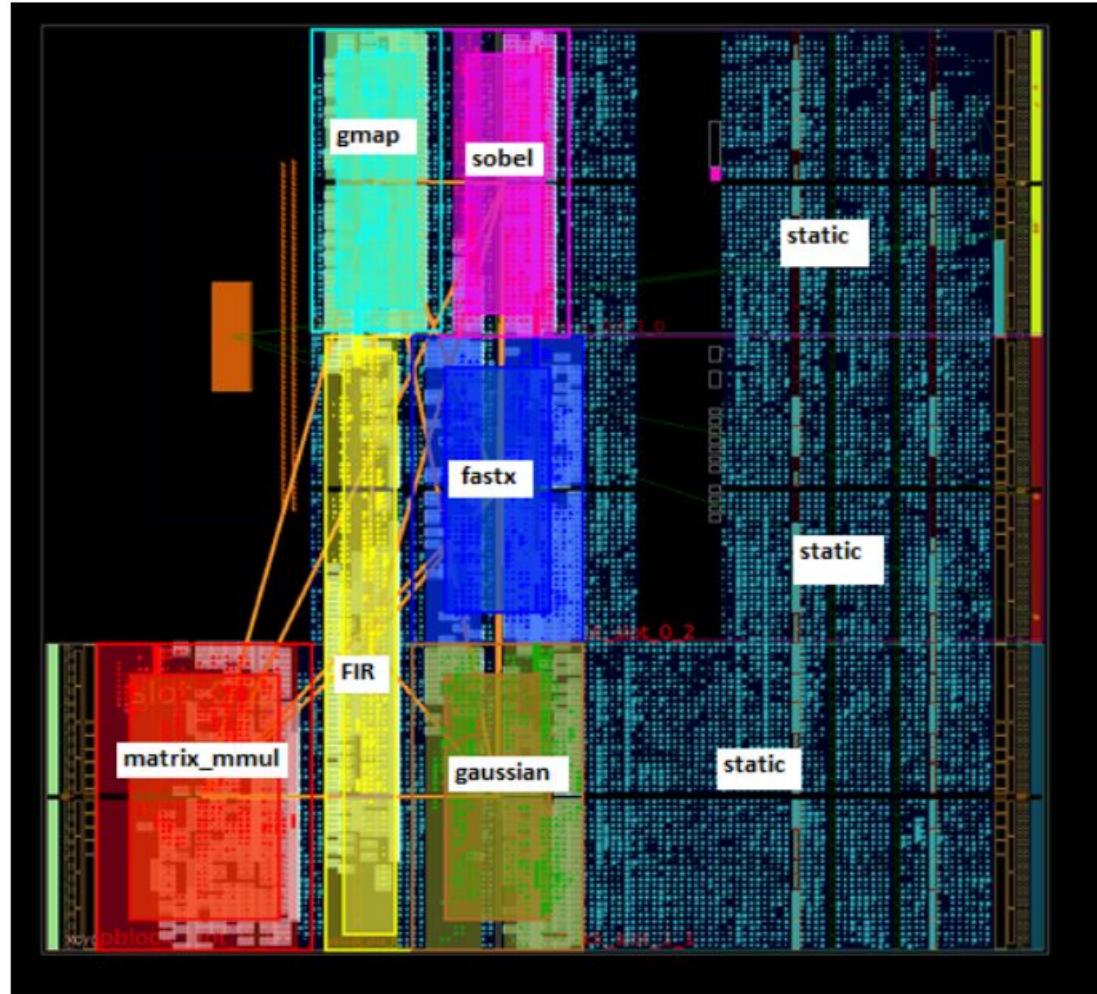
“FPGA floorplan”

# FPGA algorithm design

Completely different to normal programming → think in terms of “building structures” and moving data through chip in 2D

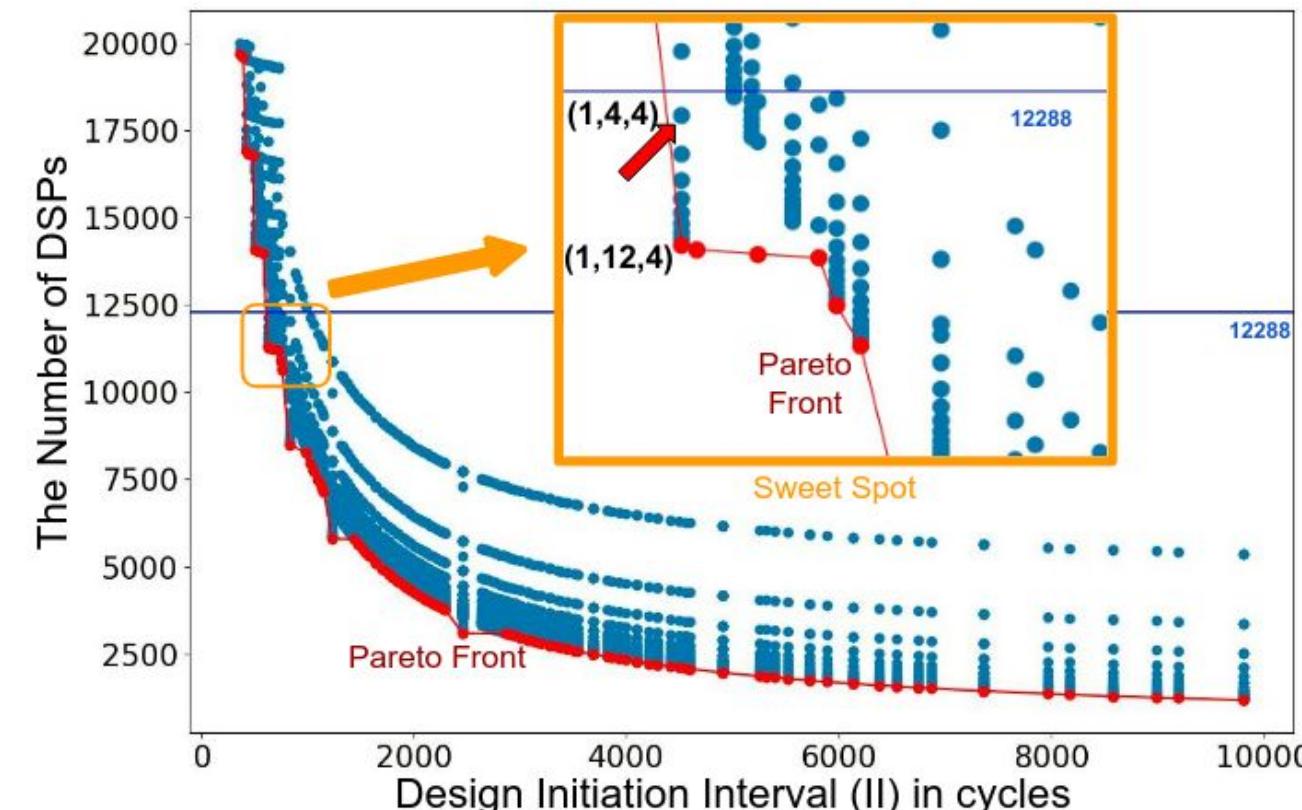
Different handles depending on design constraints

- Duplicate structures to run in parallel (reduce latency, increased resource-usage)
- Reuse structures many times for different data (longer latency, decrease resources)



<https://retis.sssup.it/~a.biondi/papers/CODES19.pdf>

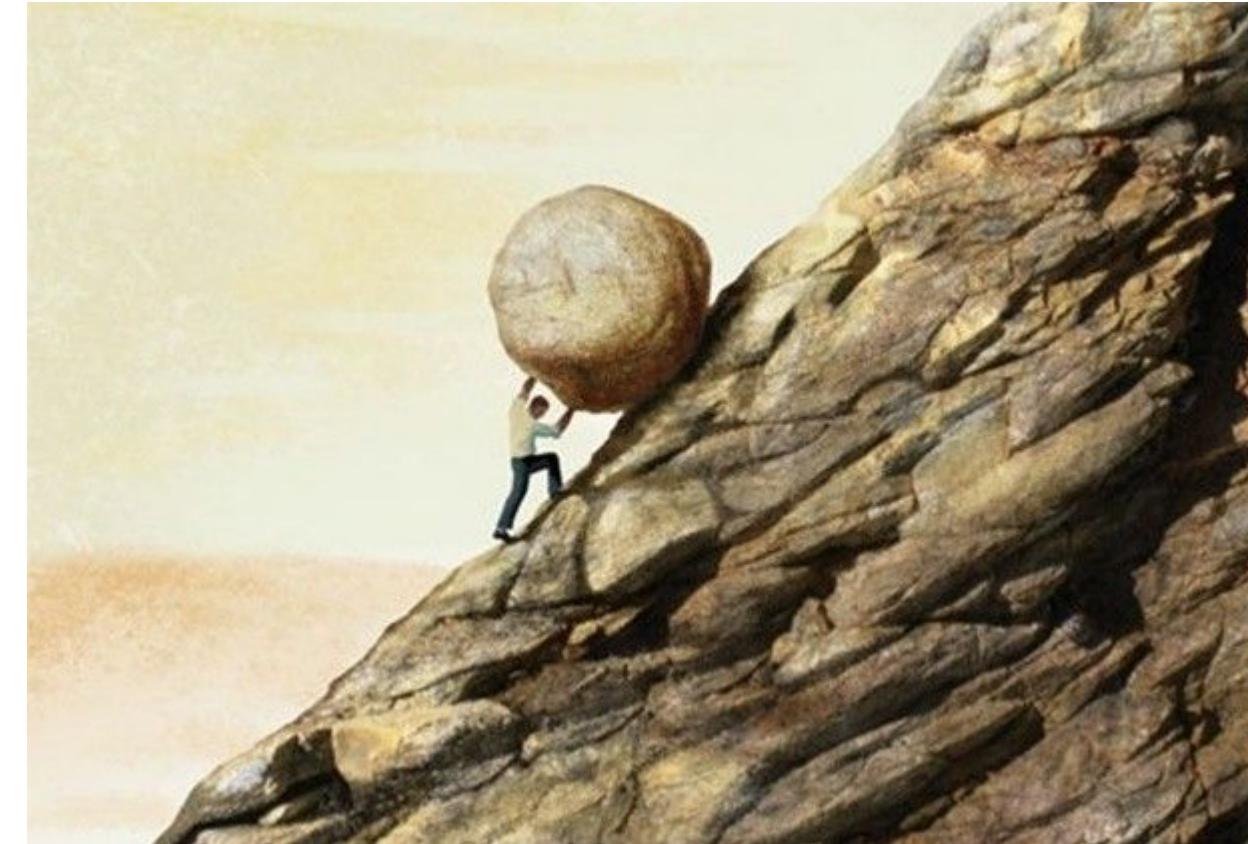
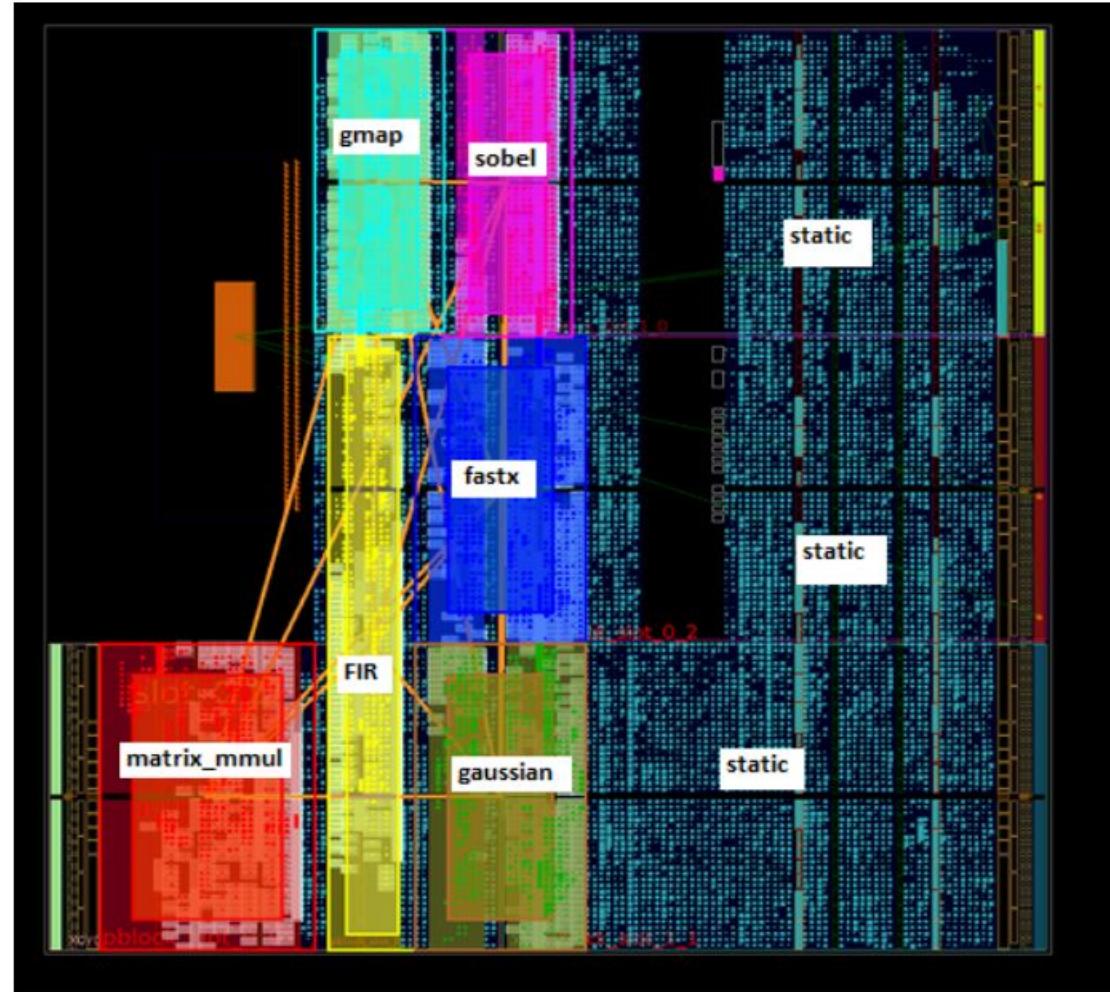
“FPGA floorplan”



Example: resource-usage vs latency for GNN on Xilinx U250 FPGAs [\[Link\]](#)

# How are FPGAs programmed?

Hardware Description Languages (HDLs) - programming languages which describe electronic circuits



“FPGA floorplan”

# How are FPGAs programmed?



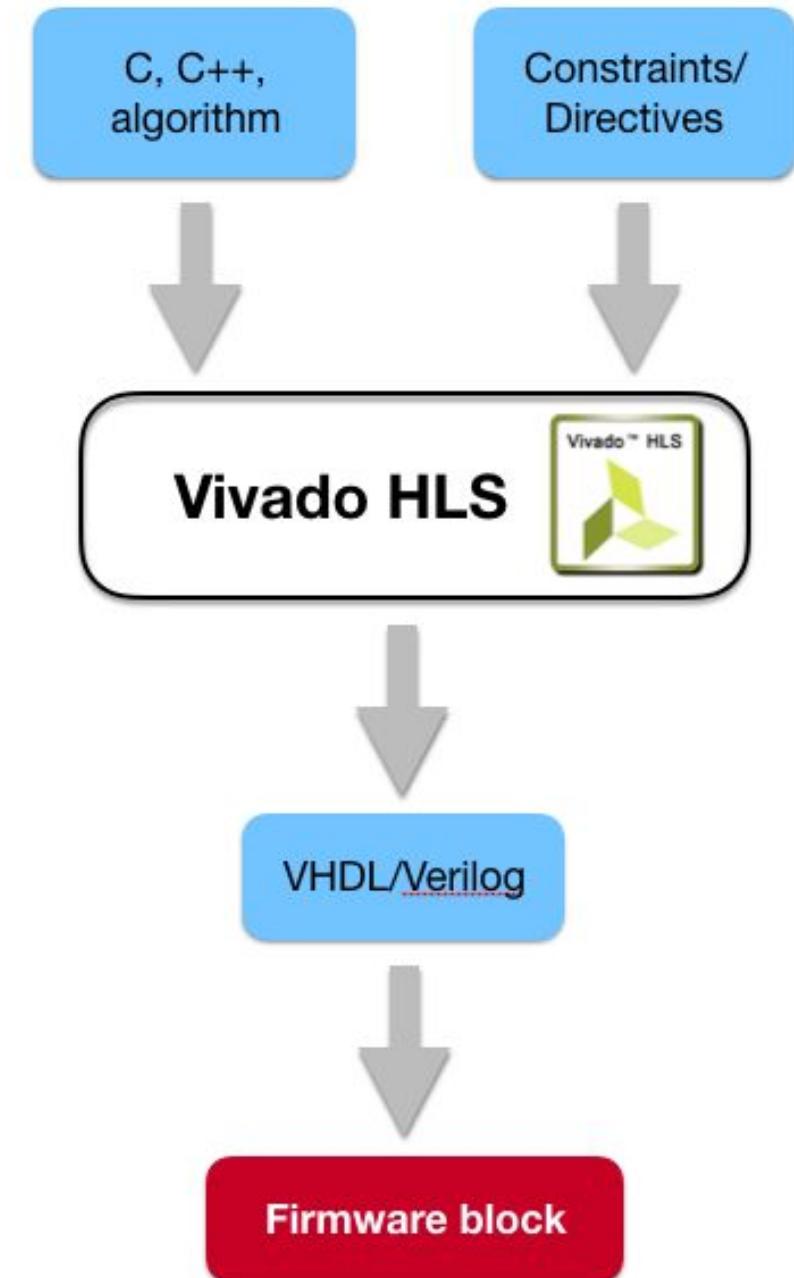
Hardware Description Languages (HDLs) - programming languages which describe electronic circuits

## High-level synthesis (HLS)

- Convert high-level code in C/C++ into (synthesizable) VHDL
- Inject constraints (resource, latency, ...) to optimise the design
- Tools like Intel HLS Compiler, Vivado HLS, ...
- **Drastic decrease in firmware development time and open to all!**

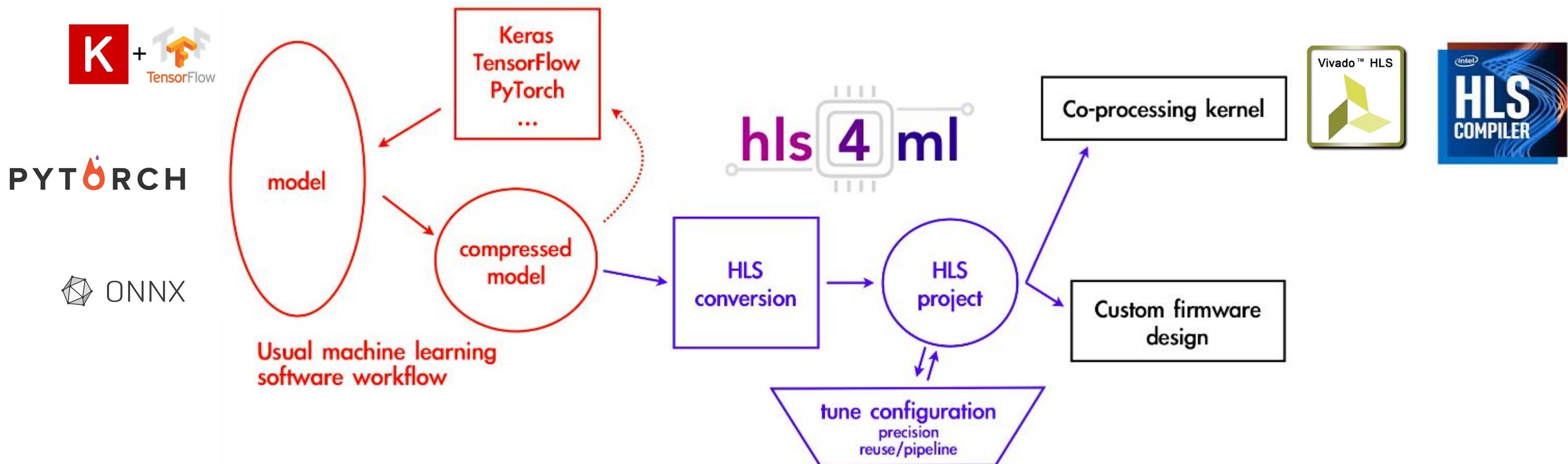
After this we can synthesize and map design onto actual FPGA fabric

- Of course this includes simulation → testing → optimising (in feedback loop) before eventual deployment
- Beauty in “reprogrammable” nature



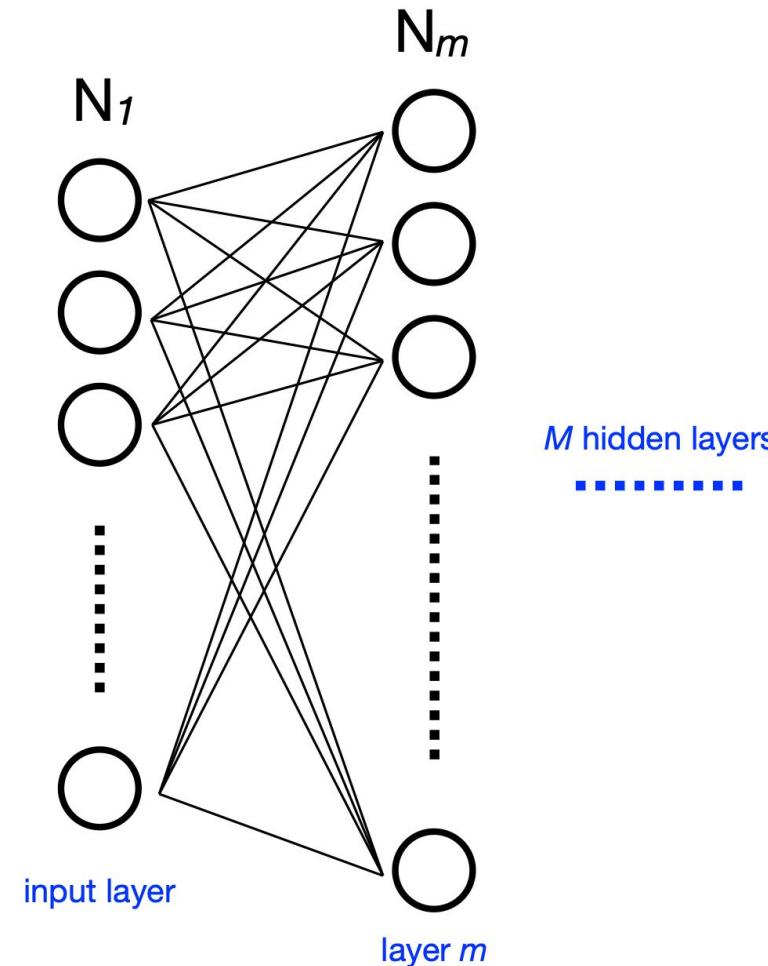
# High-level synthesis for ML (hls4ml)

Goal: provide an efficient and fast translation of ML models from open-source packages (Keras, PyTorch, ...) to HLS code than can be synthesized to run on an FPGA

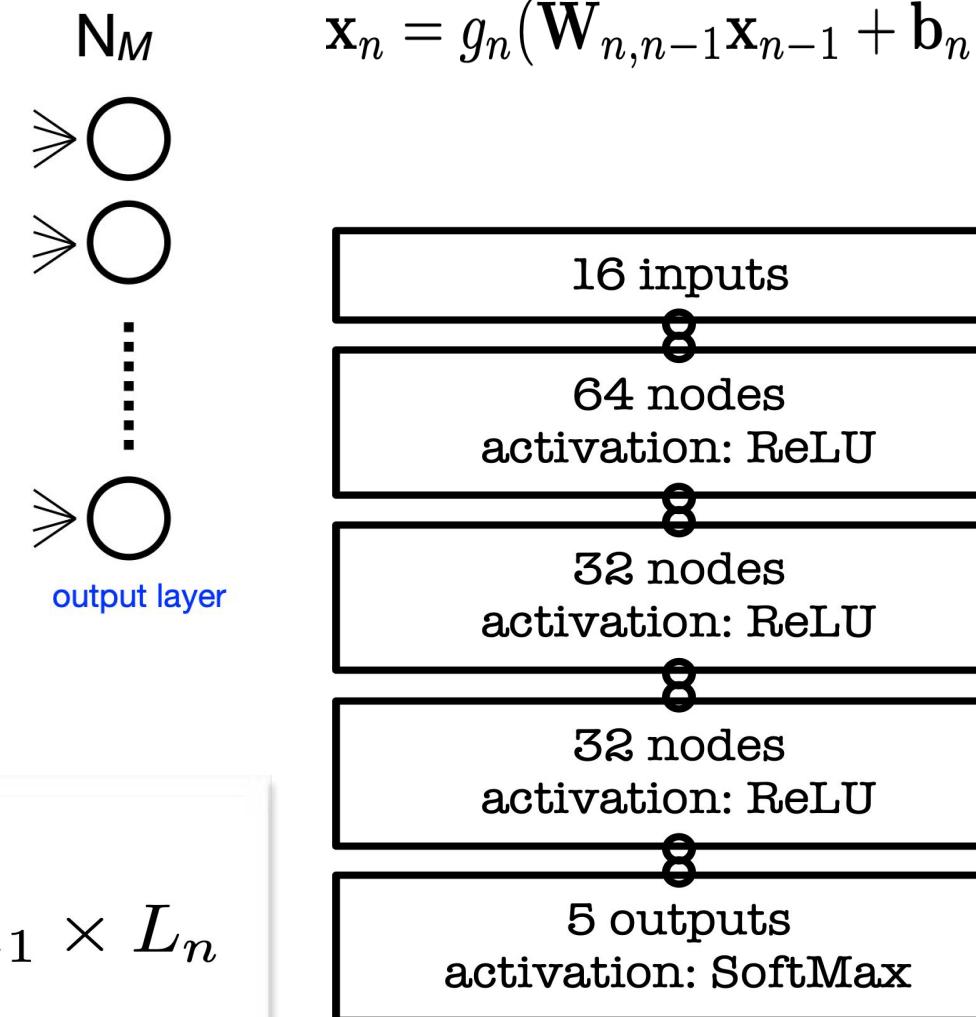


The `hls4ml` package enables fast prototyping of a machine learning algorithm implementation in FPGAs, greatly reducing the time to results and giving the user intuition for how to best design a machine learning algorithm for their application while balancing performance, resource utilization and latency requirements.

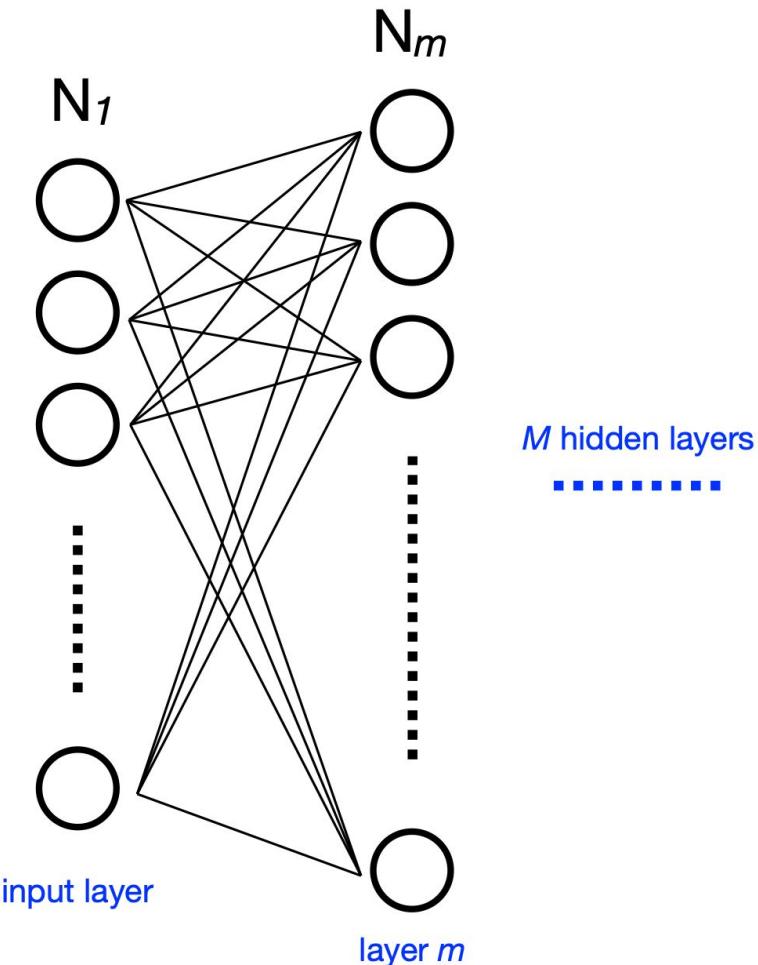
# FPGA resources for inference



$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

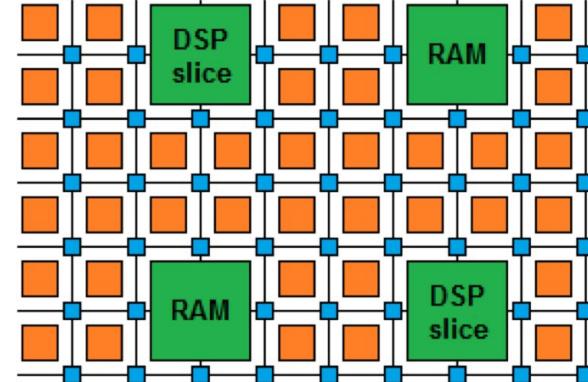
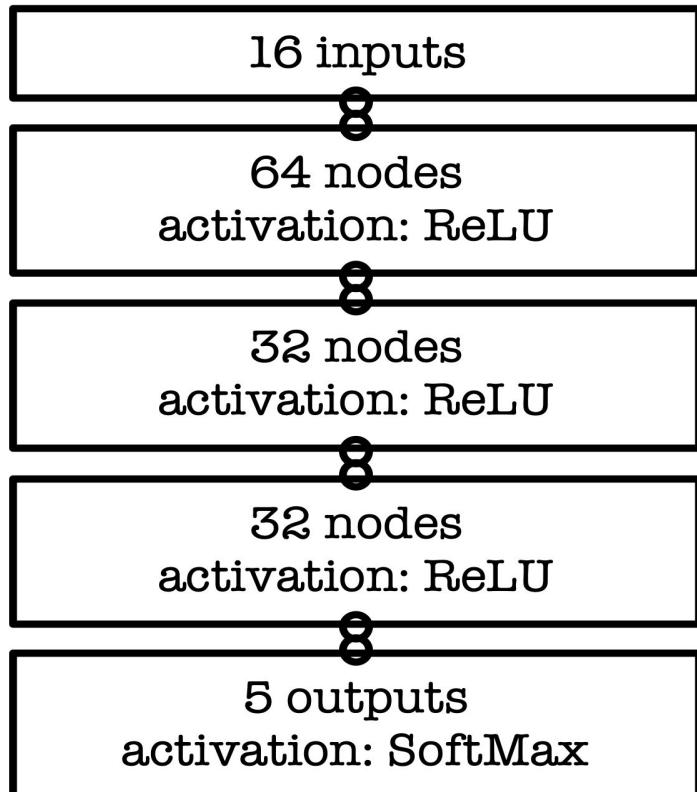
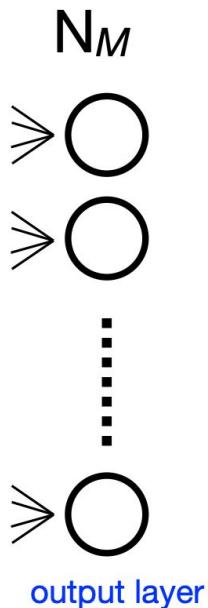


# FPGA resources for inference



$$N_{\text{multiplications}} = \sum_{n=2}^N L_{n-1} \times L_n$$

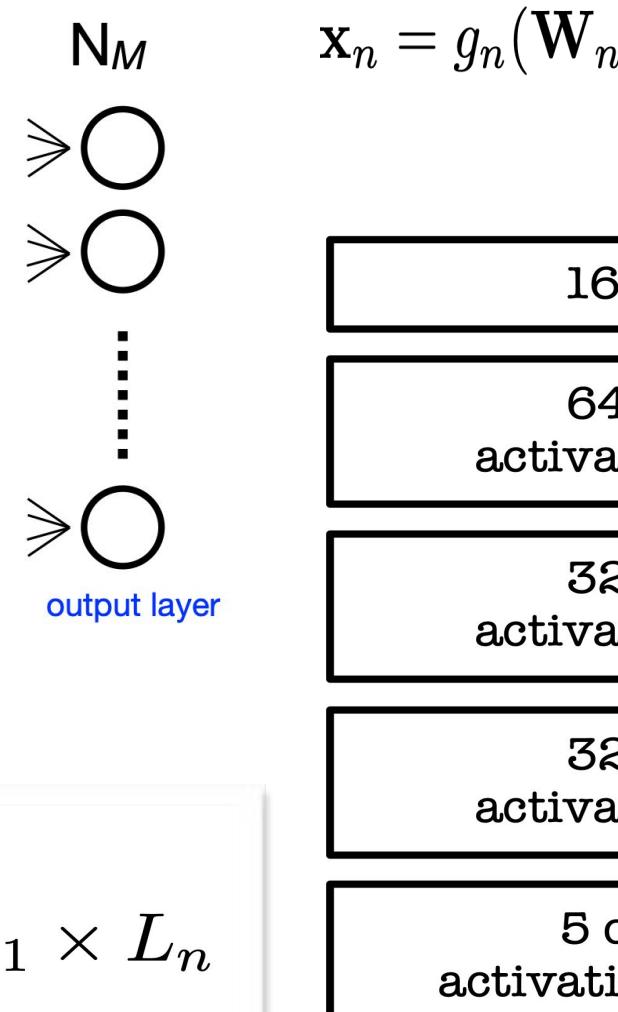
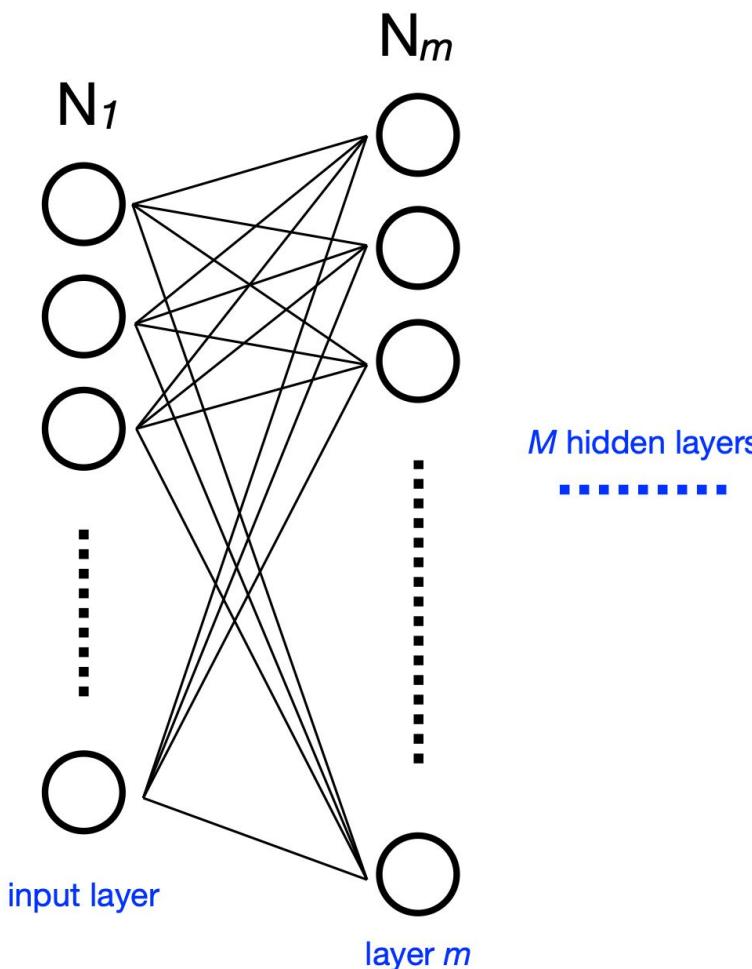
$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$



Neural Network Inference Operations → FPGA Resource Mapping		
NN Operation	Maps To FPGA Resource(s)	Notes
Matrix Multiplication (e.g., GEMM)	DSP slices, LUTs, BRAMs	DSPs perform multiply-accumulate (MAC); large arrays built from parallel DSPs
Convolution	DSP slices, LUTs, pipelining logic, shift registers	Often implemented as a sliding window MAC unit; can share structure with GEMM
Element-wise operations (ReLU, add, etc.)	LUTs, FFs	Simple combinational logic or pipelined logic (e.g., max(0, x))
Batch Normalization	LUTs, DSPs	Treated as a scale + shift (multiplication + addition)
Pooling (max/avg)	LUTs, FFs, comparators, adders	Max pooling uses comparators; avg pooling uses adders/dividers
Activation Functions (e.g., Sigmoid, Tanh)	LUTs, Look-Up Tables in BRAM, or Piecewise Approximation	Often approximated with LUTs or linear segments to reduce resource use
Quantization / Dequantization	LUTs, DSPs, bit manipulators	Typically fixed-point conversion and scaling
Memory Access / Buffering	BRAMs, UltraRAM, external DRAM	For activations, weights, intermediate results



# FPGA resources for inference

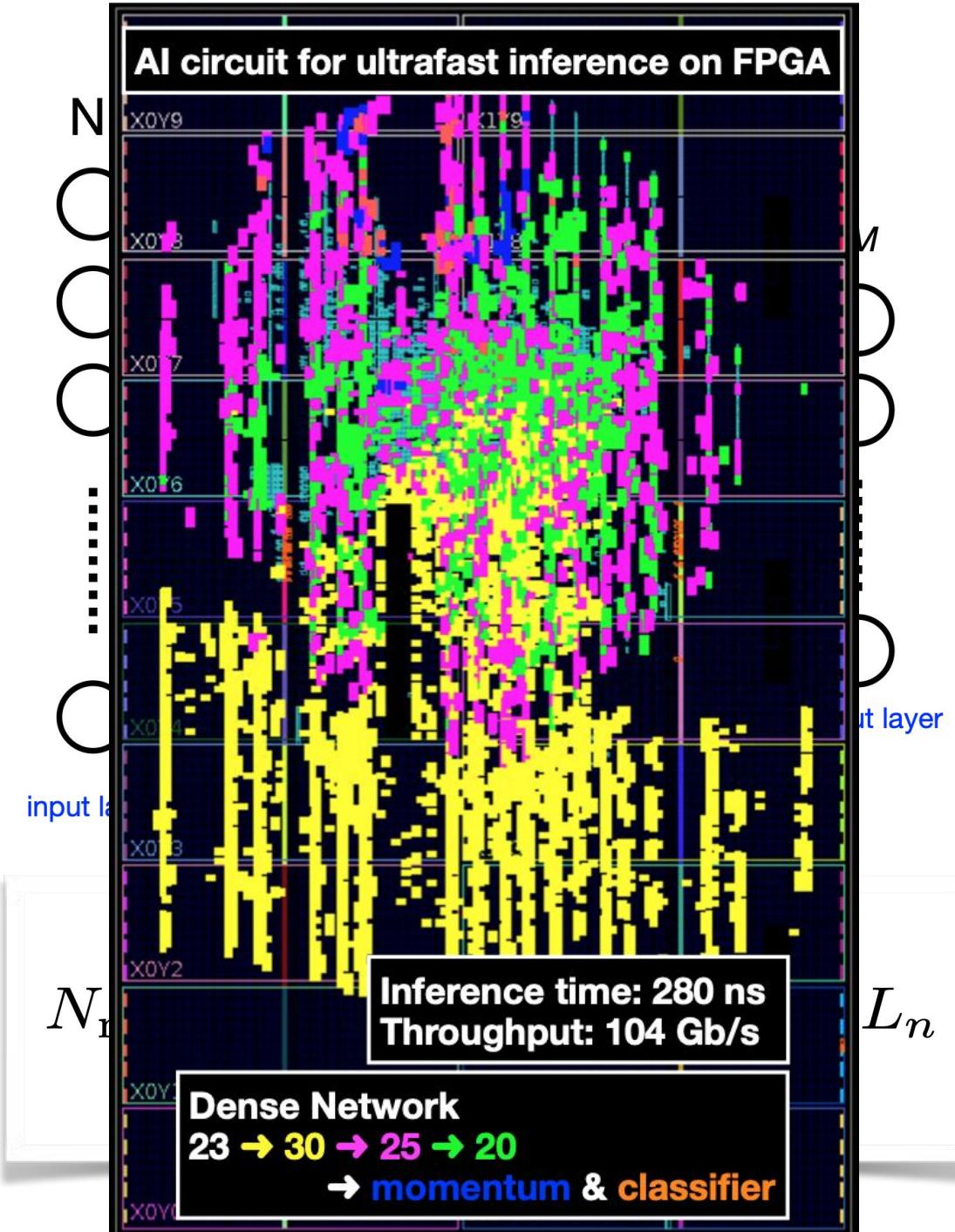


NN Operation	Maps To FPGA Resource(s)	Notes
Matrix Multiplication (e.g., GEMM)	DSP slices, LUTs, BRAMs	DSPs perform multiply-accumulate (MAC); large arrays built from parallel DSPs
Convolution	DSP slices, LUTs, pipelining logic, shift registers	Often implemented as a sliding window MAC unit; can share structure with GEMM
Element-wise operations (ReLU, add, etc.)	LUTs, FFs	Simple combinational logic or pipelined logic (e.g., $\max(0, x)$ )
Batch Normalization	LUTs, DSPs	Treated as a scale + shift (multiplication + addition)
Pooling (max/avg)	LUTs, FFs, comparators, adders	Max pooling uses comparators; avg pooling uses adders/dividers
Activation Functions (e.g., Sigmoid, Tanh)	LUTs, Look-Up Tables in BRAM, or Piecewise Approximation	Often approximated with LUTs or linear segments to reduce resource use
Quantization / Dequantization	LUTs, DSPs, bit manipulators	Typically fixed-point conversion and scaling
Memory Access / Buffering	BRAMs, UltraRAM, external DRAM	For activations, weights, intermediate results

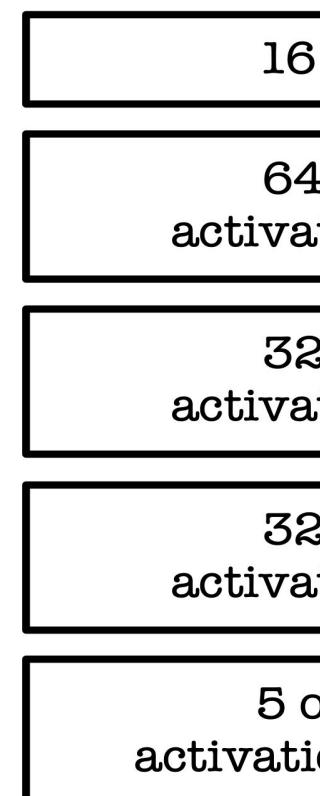


# FPGA resources for inference

## “FPGA floorplan”



$$\mathbf{x}_n = g_n(\mathbf{W}_n,$$



NN Operation	Maps To FPGA Resource(s)	Notes
Matrix Multiplication (e.g., GEMM)	DSP slices, LUTs, BRAMs	DSPs perform multiply-accumulate (MAC); large arrays built from parallel DSPs
Convolution	DSP slices, LUTs, pipelining logic, shift registers	Often implemented as a sliding window MAC unit; can share structure with GEMM
Element-wise operations (ReLU, add, etc.)	LUTs, FFs	Simple combinational logic or pipelined logic (e.g., $\max(0, x)$ )
Batch Normalization	LUTs, DSPs	Treated as a scale + shift (multiplication + addition)
Pooling (max/avg)	LUTs, FFs, comparators, adders	Max pooling uses comparators; avg pooling uses adders/dividers
Activation Functions (e.g., Sigmoid, Tanh)	LUTs, Look-Up Tables in BRAM, or Piecewise Approximation	Often approximated with LUTs or linear segments to reduce resource use
Quantization / Dequantization	LUTs, DSPs, bit manipulators	Typically fixed-point conversion and scaling
Memory Access / Buffering	BRAMs, UltraRAM, external DRAM	For activations, weights, intermediate results

How many resources? Does the model fit in the latency requirements of our problem?



# Efficient NN design for FPGAs

- FPGAs provide huge flexibility → performance depends on how well you take advantage of this

Any application of AI inference on FPGAs is defined by a number constraints:

- Input bandwidth (rate at which data can be transferred to/from/within the FPGA)
- FPGA resources
- Latency



There are a number of techniques we use to tune AI inference to meet bandwidth, resource and latency constraints:

- Quantization: reduces the precision of calculations (inputs, weights, biases)
- Compression: reduce number of synapses or neurons via pruning
- Parallelization: tune how much to parallelize to make inference faster/slower vs FPGA resources

# Efficient NN design for FPGAs

- FPGAs provide huge flexibility → performance depends on how well you take advantage of this

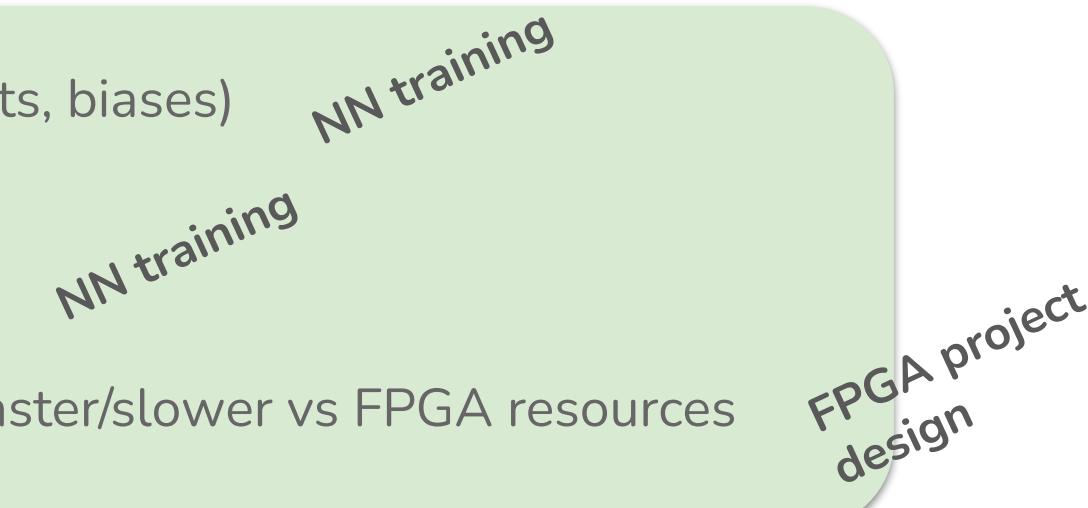
Any application of AI inference on FPGAs is defined by a number constraints:

- Input bandwidth (rate at which data can be transferred to/from/within the FPGA)
- FPGA resources
- Latency



There are a number of techniques we use to tune AI inference to meet bandwidth, resource and latency constraints:

- Quantization: reduces the precision of calculations (inputs, weights, biases)
- Compression: reduce number of synapses or neurons via pruning
- Parallelization: tune how much to parallelize to make inference faster/slower vs FPGA resources



# Efficient NN design for FPGAs

- FPGAs provide huge flexibility → performance depends on how well you take advantage of this

Any application of AI inference on FPGAs is defined by a number constraints:

- Input bandwidth (rate at which data can be transferred to/from/within the FPGA)
- FPGA resources
- Latency



There are a number of techniques we use to tune AI inference to meet bandwidth, resource and latency constraints:

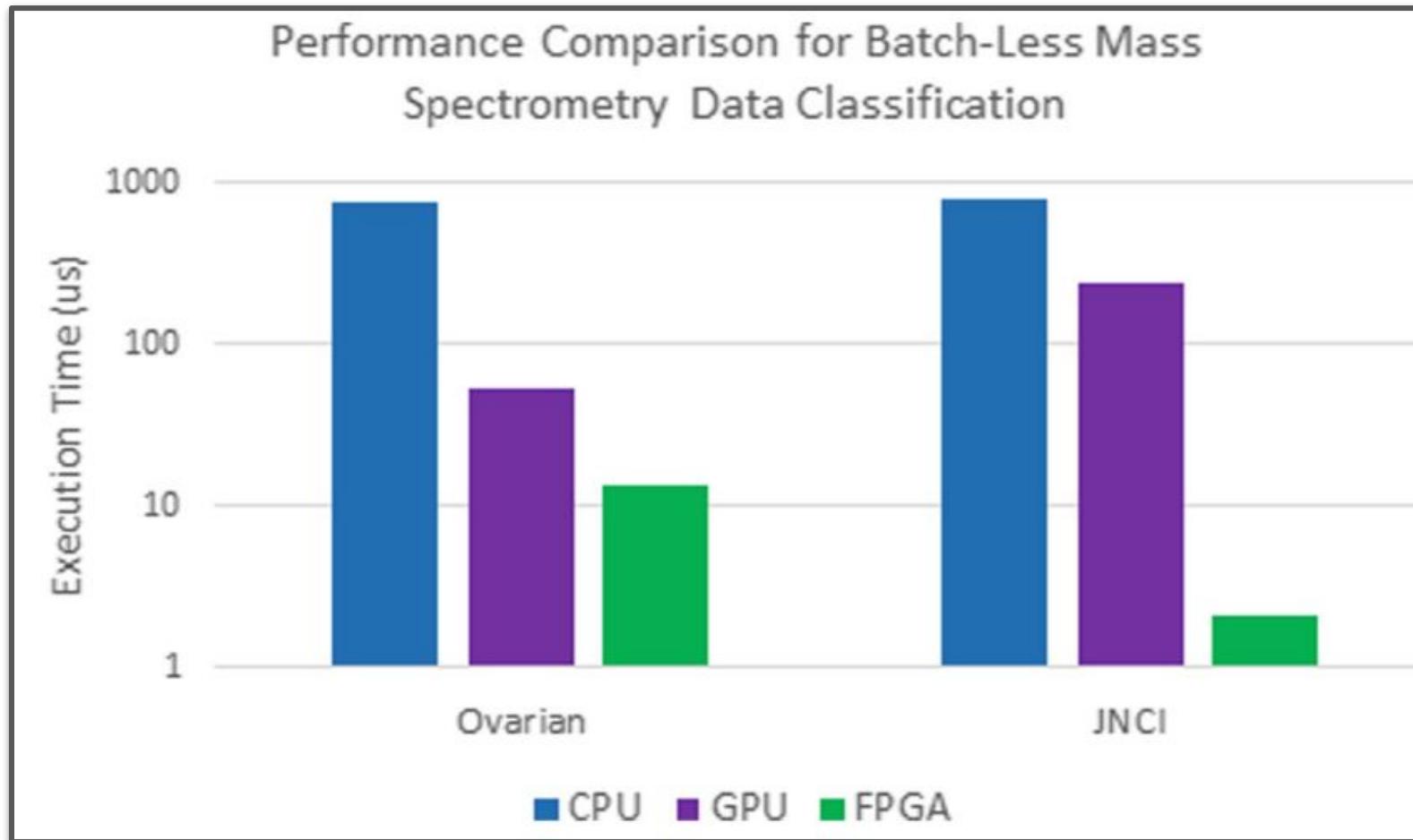
- Quantization: reduces the precision of calculations (inputs, weights, biases)
- Compression: reduce number of synapses or neurons via pruning
- Parallelization: tune how much to parallelize to make inference faster/slower vs FPGA resources

NN training  
NN training

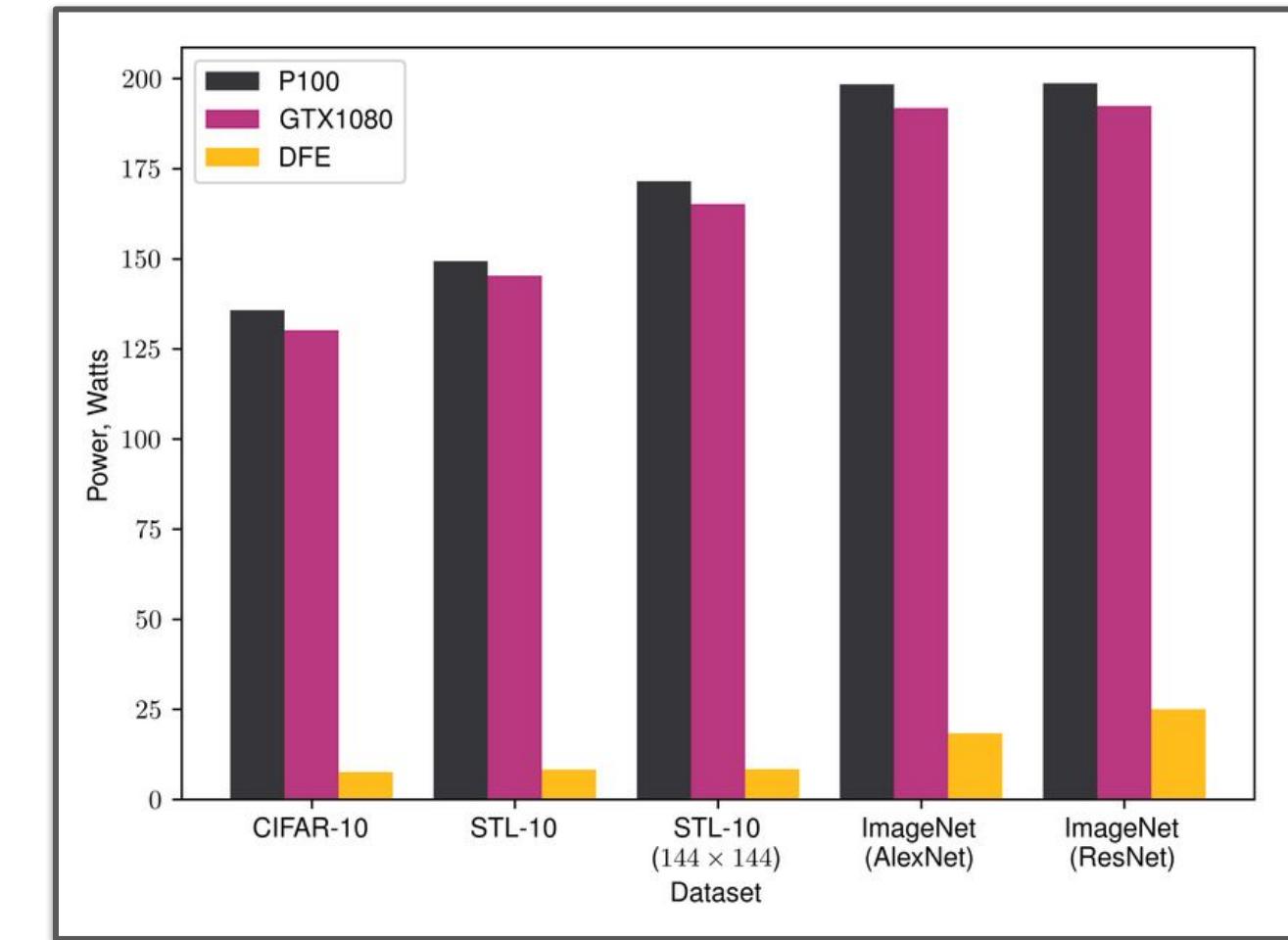
Cover these  
techniques in  
today's tutorial

FPGA project  
design

# FPGA performance

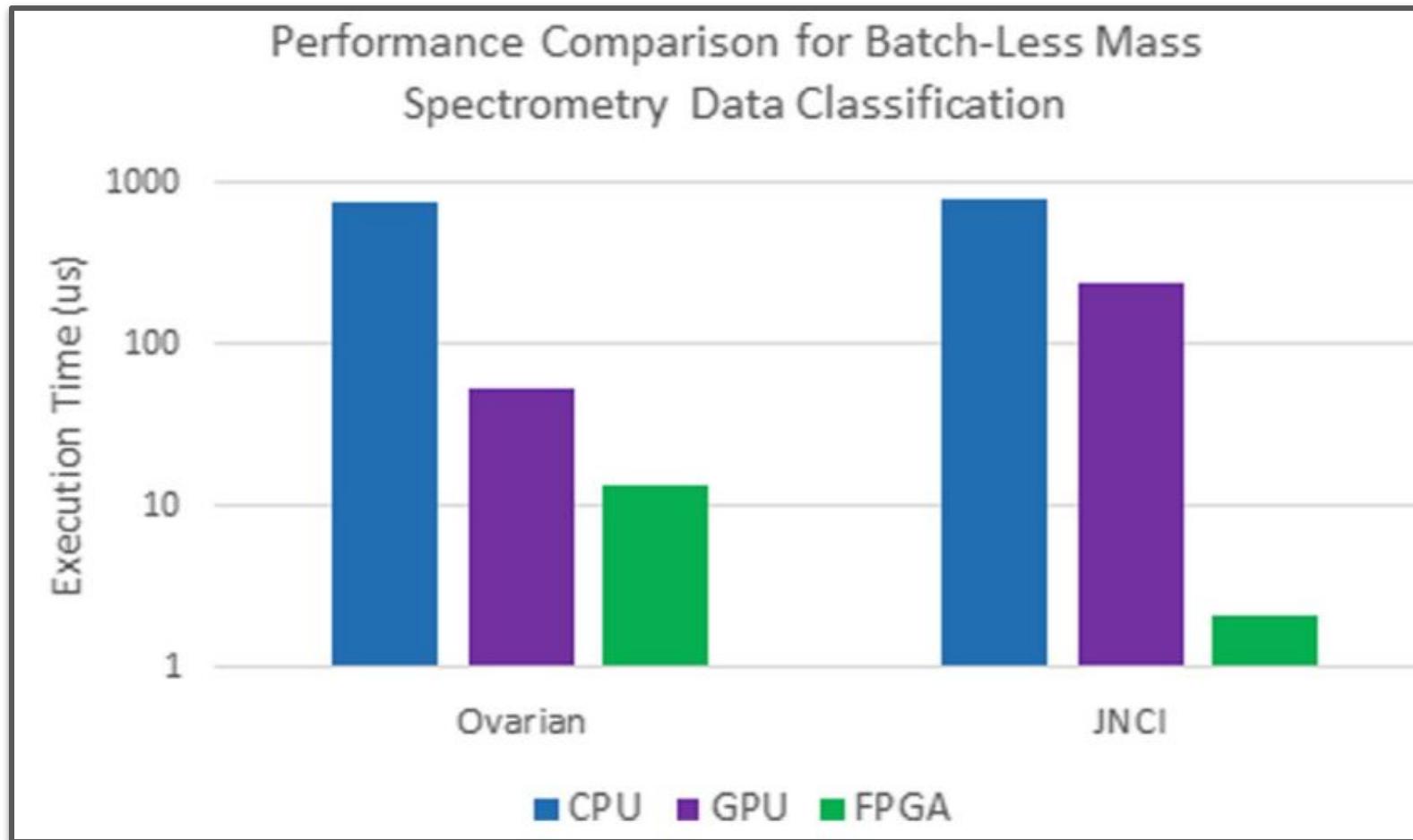


[Real-time data analysis for medical diagnosis using FPGA-accelerated NNs](#)

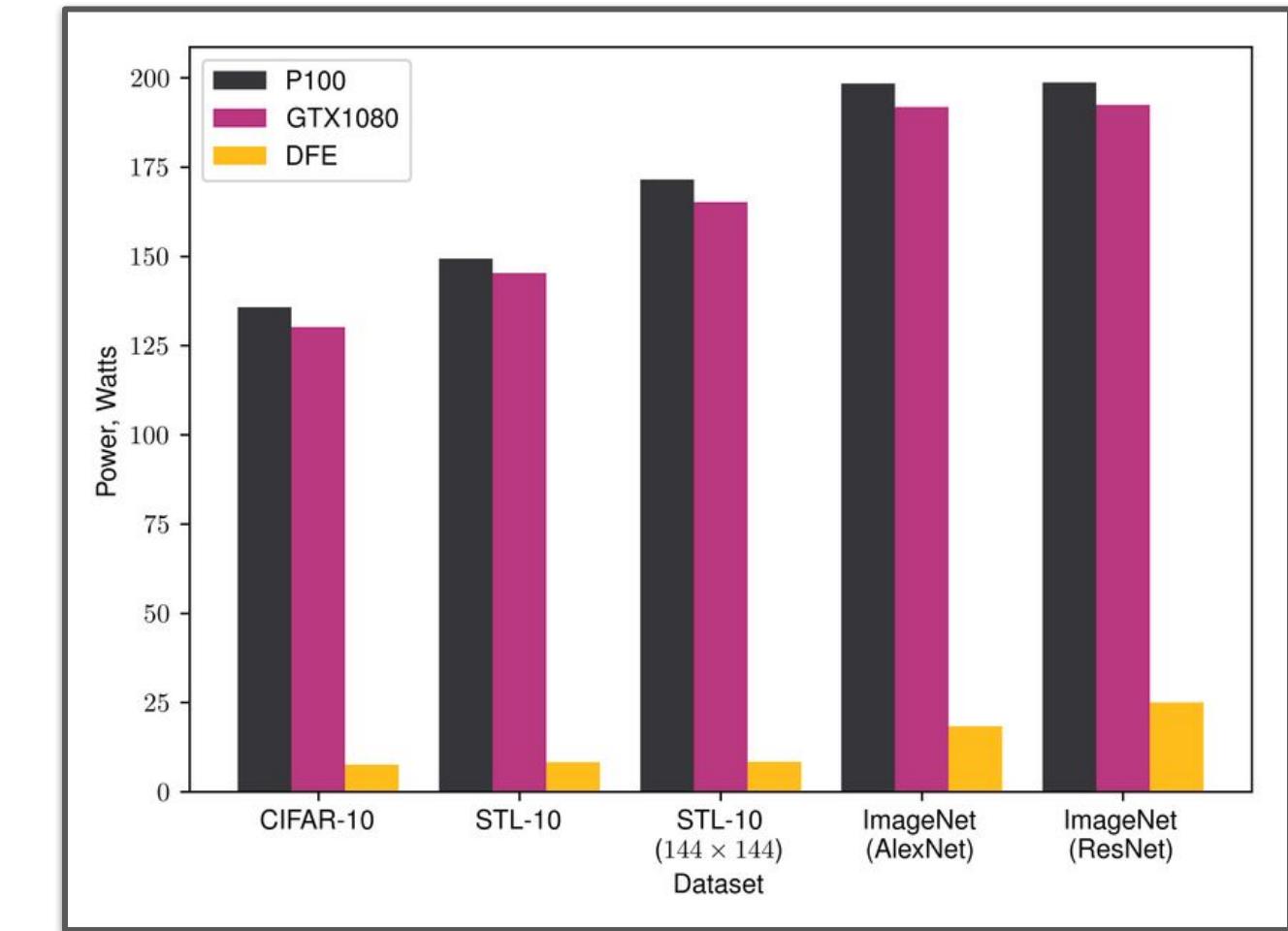


[Large-scale Quantized NNs on FPGAs](#)

# FPGA performance



[Real-time data analysis for medical diagnosis using FPGA-accelerated NNs](#)

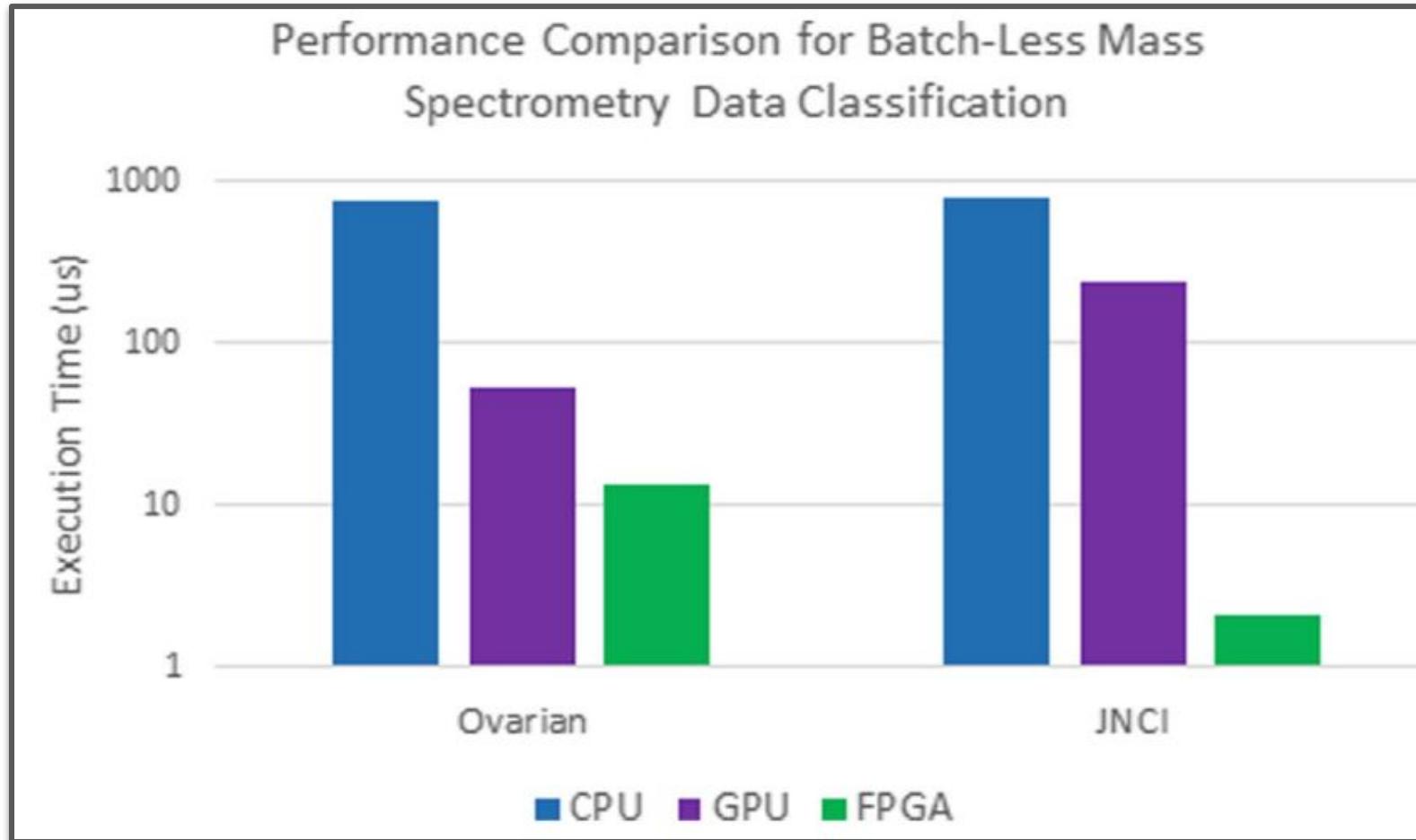


[Large-scale Quantized NNs on FPGAs](#)

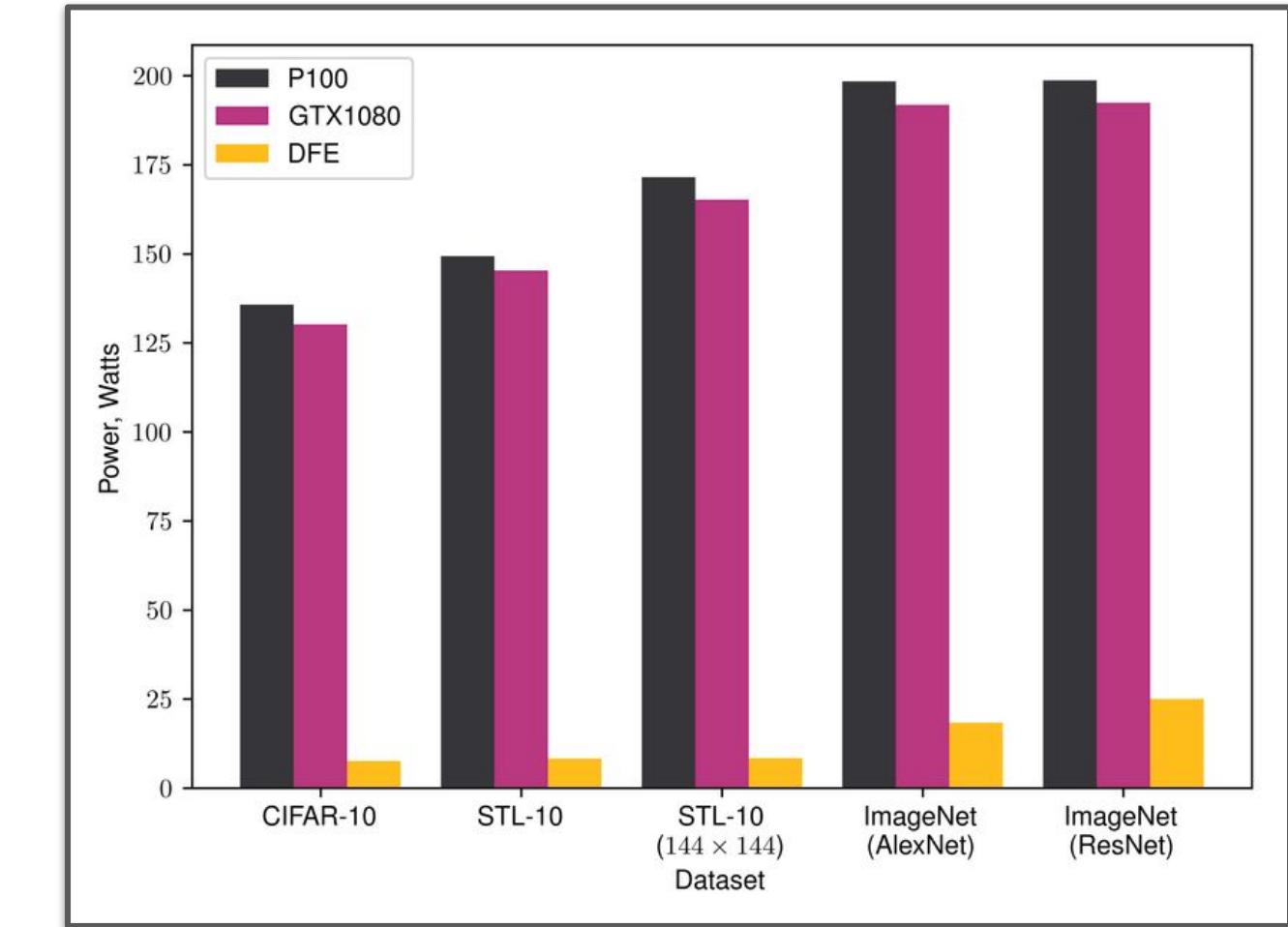
Key takeaways:

- Low latency: excellent for real-time applications e.g. autonomous cars
- Power efficient: much lower than GPUs for same task (especially with custom precision)
- Customizability: tailor models for optimal resource usage

# FPGA performance



[Real-time data analysis for medical diagnosis using FPGA-accelerated NNs](#)



[Large-scale Quantized NNs on FPGAs](#)

Key takeaways:

- Low latency: excellent for real-time applications e.g. autonomous cars
- Power efficient: much lower than GPUs for same task (especially with custom precision)
- Customizability: tailor models for optimal resource usage



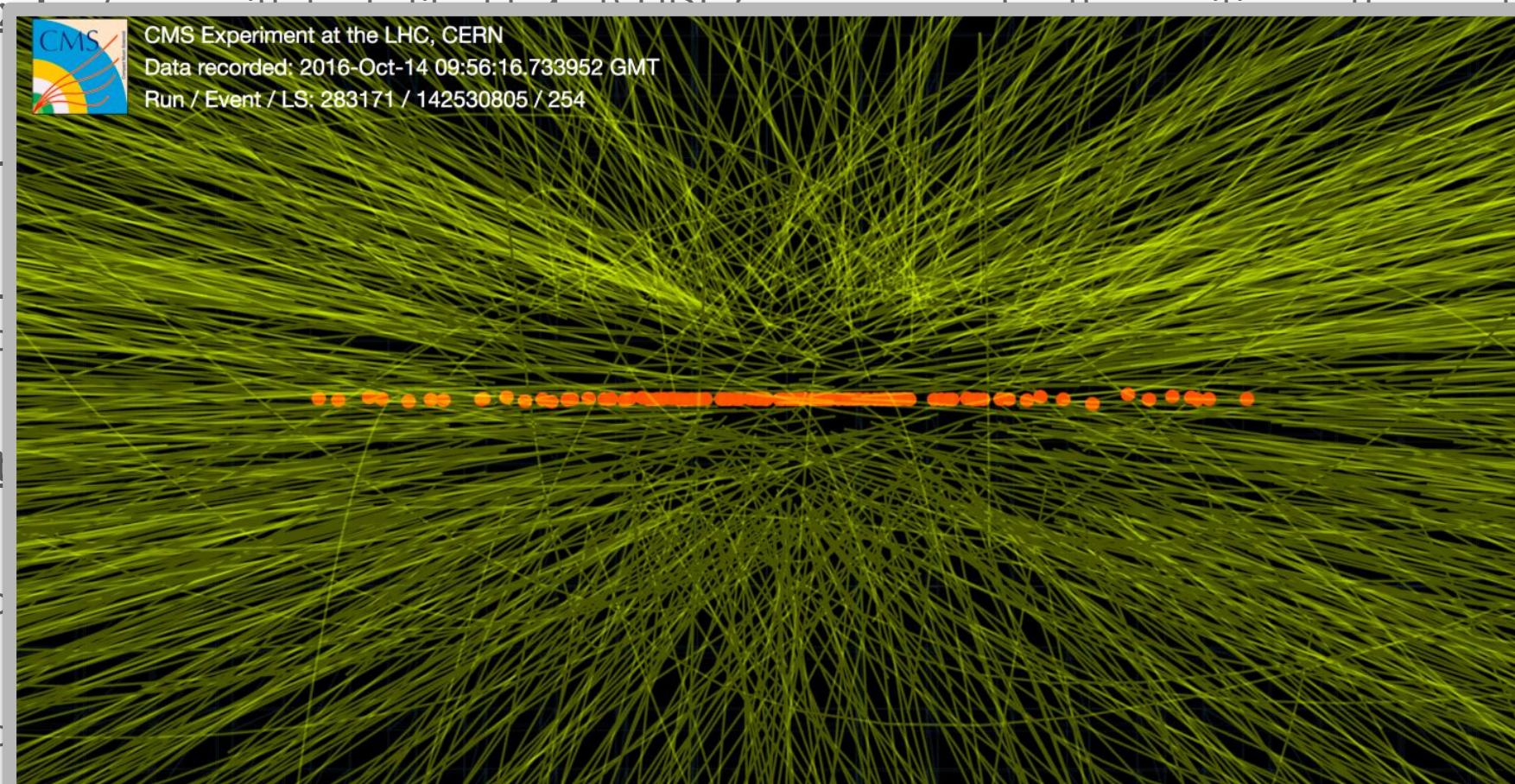
# Why FPGAs aren't used everywhere?

1. Programming complexity: (even with tools like hls4ml) HDL far more complex than writing python code, requires hardware expertise
2. Limited on-chip resources: large models (e.g. GPT) may not fit, or require aggressive (performance-degrading) quantization/pruning
3. Lower throughput for large batches: FPGAs excel at streaming small(ish), real-time inputs. For e.g. processing millions of GPT queries per-second then GPUs scale better (higher raw compute and memory bandwidth)
4. Longer development time: prototyping/optimizing for FPGAs takes longer than software-based solutions
5. Ecosystem: CPU/GPU-based ecosystems have much more mature libraries and community support
6. Cost: High-end FPGA boards are expensive, O(\$10k+), and often require licensed toolchains e.g. Xilinx Vivado

Nevertheless, FPGAs are extremely powerful and are driving innovation in many applications...

# Why FPGAs aren't used everywhere?

1. Programming complexity: Requires hardware expertise
2. Limited on-chip resources: Need off-chip memory, floating-point rounding) quantization/pruning
3. Lower throughput for浮点数: slower per-second than GPUs so processing millions of GPT queries
4. Longer development time: Many months
5. Ecosystem: CPU/GPU-based
6. Cost: High-end FPGA boards



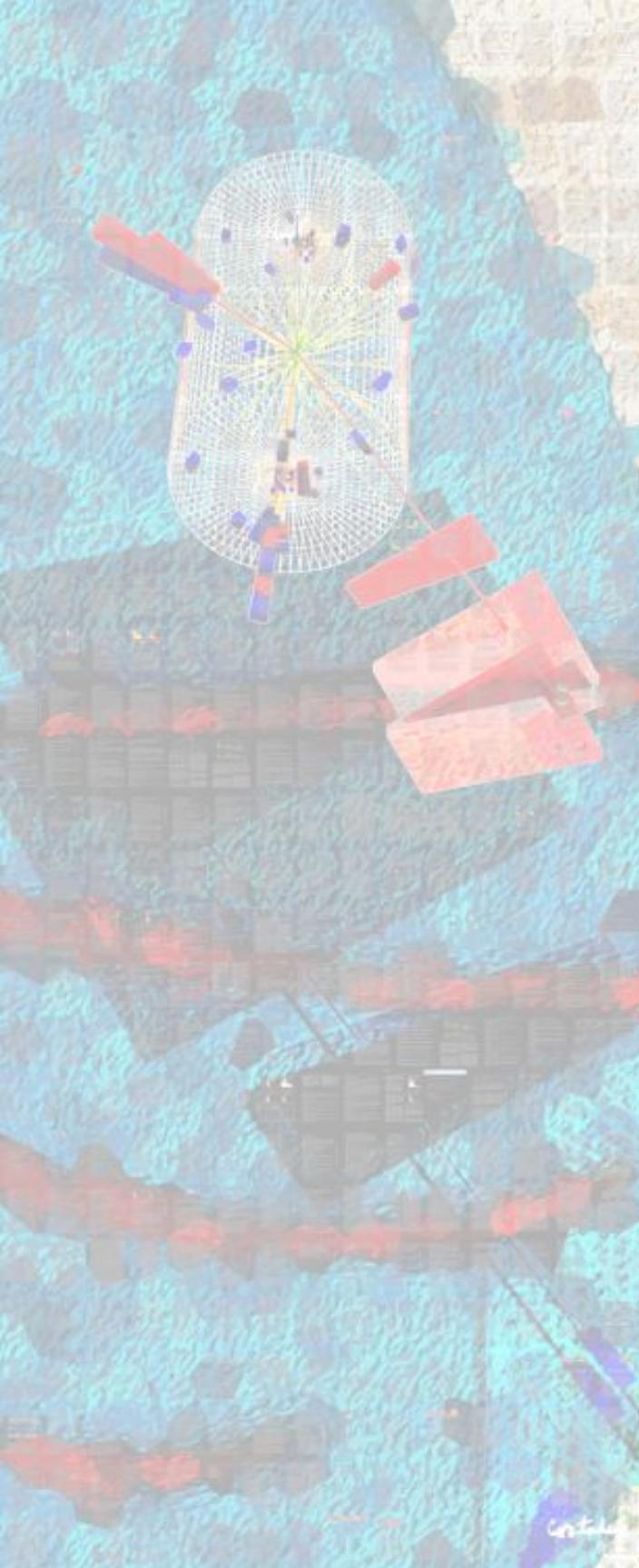
Nevertheless, FPGAs are extremely powerful and are driving innovation in many applications...

e.g. Data-taking at the Large Hadron Collider

# FPGA jargon glossary

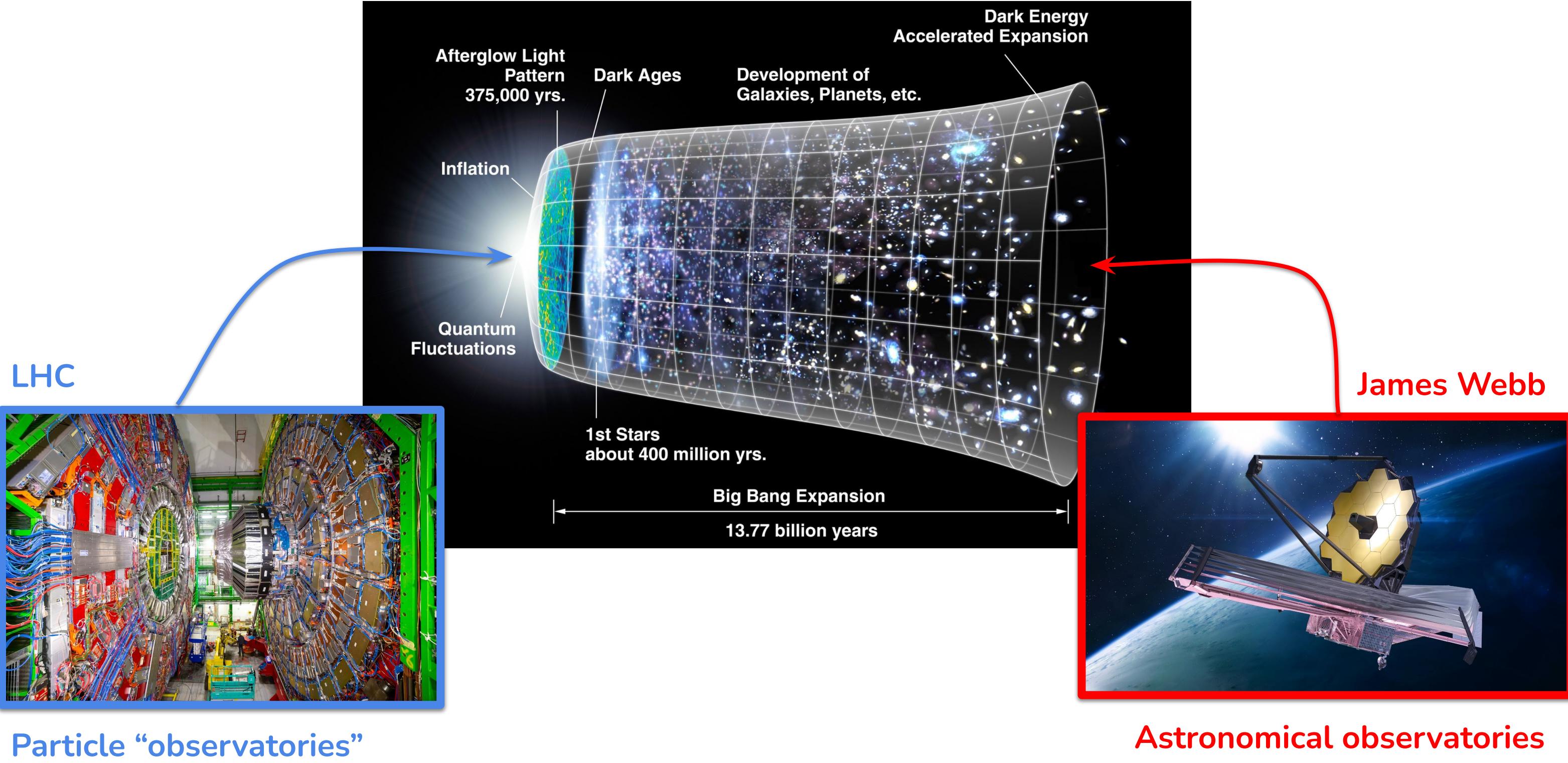
- LUT - Look Up Table (aka logic): generic functions on small bitwidth inputs. Combine many to build the algorithm
- FF - Flip Flops: control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- DSP - Digital Signal Processor: performs multiplication and other arithmetic in the FPGA
- BRAM - Block RAM: hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- HDL - Hardware Description Language: low-level language for describing circuits
- HLS - High-level Synthesis - compiler for C, C++ into FPGA IP cores
- RTL - Register Transfer Level: the very low-level description of the function and connection of the logic gates
- Latency: Time between starting processing and receiving results for one item in the data (measured in clock cycles or seconds)
- Throughput: Measure of rate that data flows through processing system
- Bandwidth: Rate at which data can be transferred into, out-of or within the FPGA
- II - Initiation Interval: time from accepting first input to accepting next input





## **“FastML” @ Large Hadron Collider**

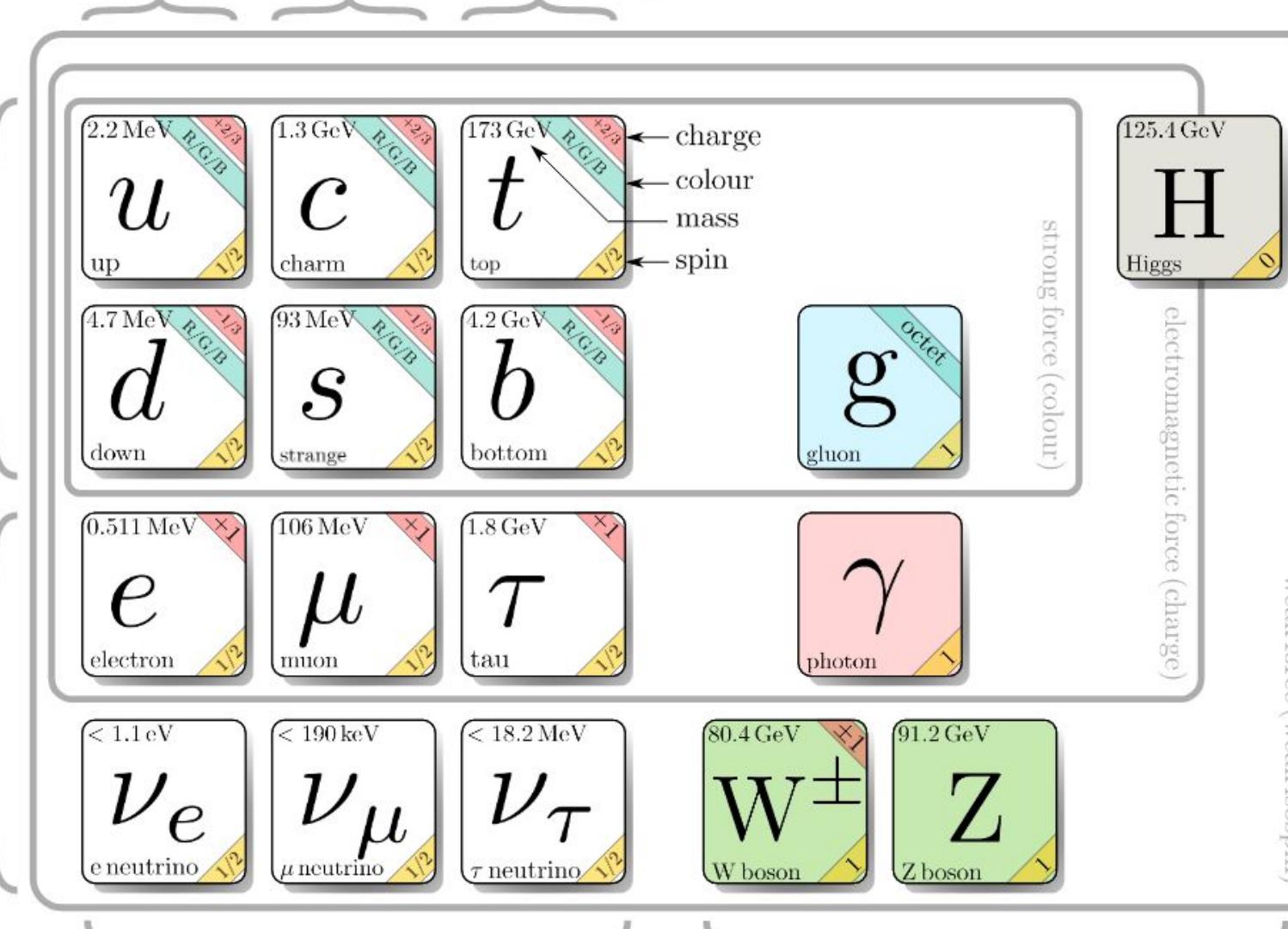
# Collider physics: a primer/recap



# What we know?

Quarks

1<sup>st</sup>      2<sup>nd</sup>      3<sup>rd</sup> generation

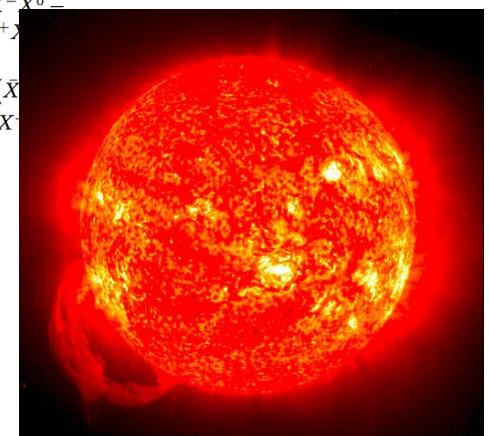
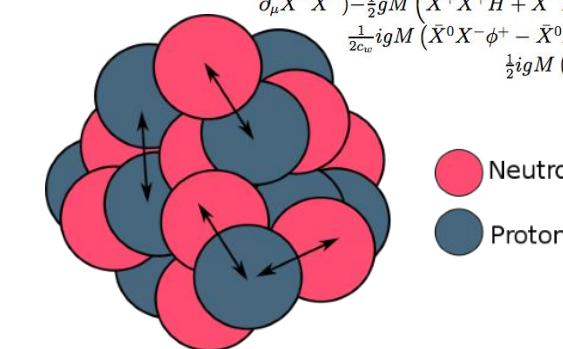


Fermions

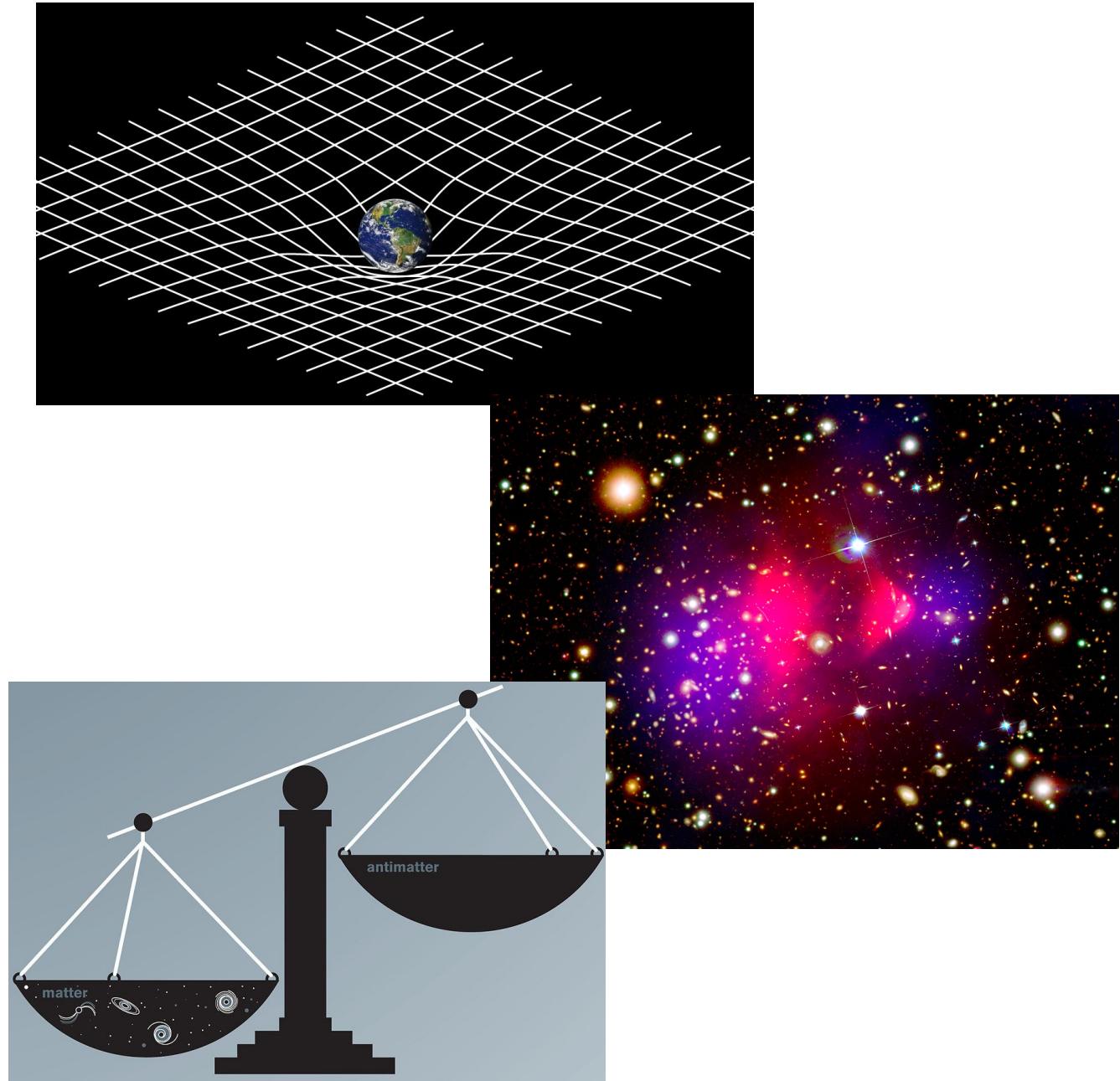
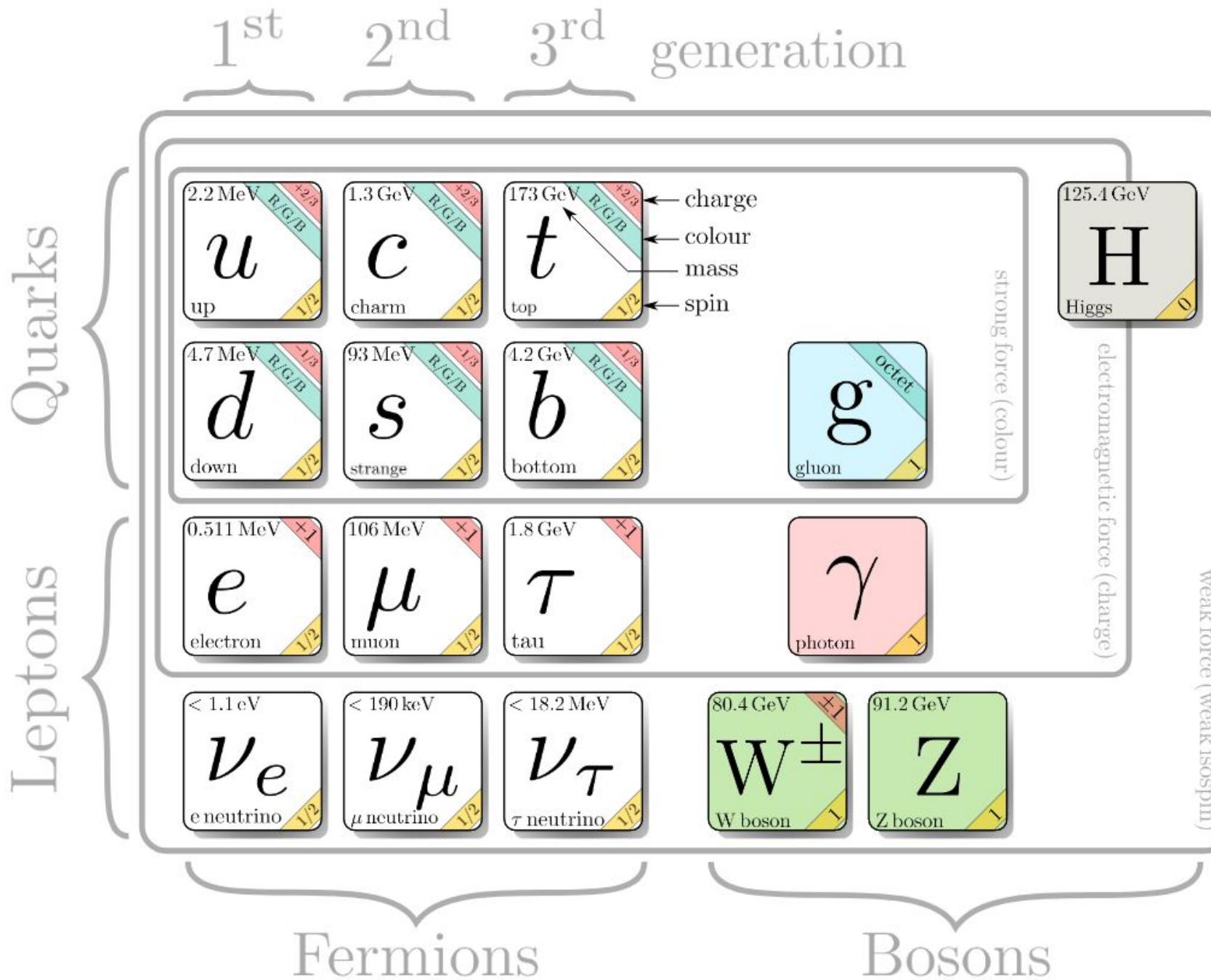
Bosons



$$\begin{aligned}
 \mathcal{L}_{SM} = & -\frac{1}{2}\partial_\nu g_\mu^a \partial_\mu g_\nu^a - g_s f^{abc} \partial_\mu g_\nu^a g_\mu^b g_\nu^c - \frac{1}{2}g_w^2 f^{abc} f^{ade} g_\mu^b g_\nu^c g_\mu^d g_\nu^e - \partial_\nu W_\mu^+ \partial_\mu W_\nu^- - \\
 & M^2 W_\mu^+ W_\nu^- - \frac{1}{2}\partial_\nu Z_\mu^0 \partial_\mu Z_\nu^0 - \frac{1}{2c_w^2} M^2 Z_\mu^0 Z_\nu^0 - \frac{1}{2}\partial_\mu A_\nu \partial_\nu A_\mu - ig c_w (\partial_\nu Z_\mu^0 (W_\mu^+ W_\nu^- - \\
 & W_\nu^+ W_\mu^-) - Z_\mu^0 (W_\mu^+ \partial_\nu W_\mu^- - W_\mu^- \partial_\nu W_\mu^+) + Z_\mu^0 (W_\mu^+ \partial_\nu W_\mu^- - W_\nu^- \partial_\nu W_\mu^+)) - \\
 & ig s_w (\partial_\nu A_\mu (W_\mu^+ W_\nu^- - W_\mu^- W_\nu^+) - A_\nu (W_\mu^+ \partial_\nu W_\mu^- - W_\mu^- \partial_\nu W_\mu^+) + A_\mu (W_\mu^+ \partial_\nu W_\mu^- - \\
 & W_\nu^- \partial_\nu W_\mu^+)) - \frac{1}{2}g^2 W_\mu^+ W_\nu^- W_\mu^+ W_\nu^+ + \frac{1}{2}g^2 W_\mu^+ W_\nu^- W_\mu^+ W_\nu^- + g^2 c_w^2 (Z_\mu^0 W_\mu^+ Z_\nu^0 W_\nu^- - \\
 & Z_\nu^0 Z_\mu^0 W_\nu^+ W_\mu^-) + g^2 s_w^2 (A_\mu W_\mu^+ A_\nu W_\nu^- - A_\mu A_\nu W_\mu^+ W_\nu^-) + g^2 s_w c_w (A_\mu Z_\nu^0 (W_\mu^+ W_\nu^- - \\
 & W_\nu^+ W_\mu^-) - 2A_\mu Z_\mu^0 W_\nu^- W_\mu^-) - \frac{1}{2}\partial_\mu H \partial_\mu H - 2M^2 \alpha_h H^2 - \partial_\mu \phi^+ \partial_\mu \phi^- - \frac{1}{2}\partial_\mu \phi^0 \partial_\mu \phi^0 - \\
 & \partial_h \left( \frac{2M^2}{g^2} + \frac{2M}{g} H + \frac{1}{2}(H^2 + \phi^0 \phi^0 + 2\phi^+ \phi^-) \right) + \frac{2M^4}{g^2} \alpha_h - \\
 & go_h M (H^3 + H \phi^0 \phi^0 + 2H \phi^+ \phi^-) - \\
 & \frac{1}{8}g^2 \alpha_h (H^4 + (\phi^0)^4 + 4(\phi^+ \phi^-)^2 + 4(\phi^0)^2 \phi^+ \phi^- + 4H^2 \phi^+ \phi^- + 2(\phi^0)^2 H^2) - \\
 & g M W_\mu^+ W_\mu^- H - \frac{1}{2}g \frac{M}{c_w^2} Z_\mu^0 Z_\nu^0 H - \\
 & \frac{1}{2}ig (W_\mu^+ (\phi^0 \partial_\mu \phi^- - \phi^- \partial_\mu \phi^0) - W_\mu^- (\phi^0 \partial_\mu \phi^+ - \phi^+ \partial_\mu \phi^0)) + \\
 & \frac{1}{2}g (W_\mu^+ (H \partial_\mu \phi^- - \phi^- \partial_\mu H) + W_\mu^- (H \partial_\mu \phi^+ - \phi^+ \partial_\mu H)) + \frac{1}{2}g \frac{1}{c_w} (Z_\mu^0 (H \partial_\mu \phi^0 - \phi^0 \partial_\mu H) + \\
 & M (\frac{1}{c_w} Z_\mu^0 \partial_\mu \phi^0 + W_\mu^+ \partial_\mu \phi^+ + W_\mu^- \partial_\mu \phi^-) - ig \frac{s_w^2}{c_w} M Z_\mu^0 (W_\mu^+ \phi^- - W_\mu^- \phi^+) + ig s_w M A_\mu (W_\mu^+ \phi^- - \\
 & W_\mu^- \phi^+) - ig \frac{1-2c_w}{2c_w} Z_\mu^0 (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) + ig s_w A_\mu (\phi^+ \partial_\mu \phi^- - \phi^- \partial_\mu \phi^+) - \\
 & \frac{1}{4}g^2 W_\mu^+ W_\mu^- (H^2 + (\phi^0)^2 + 2\phi^+ \phi^-) - \frac{1}{8}g^2 \frac{1}{c_w^2} Z_\mu^0 Z_\nu^0 (H^2 + (\phi^0)^2 + 2(2s_w^2 - 1)^2 \phi^+ \phi^-) - \\
 & \frac{1}{2}g^2 \frac{s_w^2}{c_w} Z_\mu^0 \phi^0 (W_\mu^+ \phi^- + W_\mu^- \phi^+) - \frac{1}{2}ig \frac{s_w^2}{c_w} Z_\mu^0 H (W_\mu^+ \phi^- - W_\mu^- \phi^+) + \frac{1}{2}g^2 s_w A_\mu \phi^0 (W_\mu^+ \phi^- + \\
 & W_\mu^- \phi^+) + \frac{1}{2}ig^2 s_w A_\mu H (W_\mu^+ \phi^- - W_\mu^- \phi^+) - g^2 \frac{s_w}{c_w} (2c_w^2 - 1) Z_\mu^0 A_\mu \phi^+ \phi^- - \\
 & g^2 s_w^2 A_\mu A_\mu \phi^+ \phi^- + \frac{1}{2}ig s_w \lambda_{ij}^a (q_i^\sigma \gamma^\mu q_j^\sigma) g_a^a - \bar{\nu}^\lambda (\gamma \partial + m_\nu^\lambda) e^\lambda - \bar{\nu}^\lambda (\gamma \partial + m_\nu^\lambda) \nu^\lambda - \bar{u}_j^\lambda (\gamma \partial + \\
 & m_u^\lambda) u_j^\lambda - \bar{d}_j^\lambda (\gamma \partial + m_d^\lambda) d_j^\lambda + ig s_w A_\mu (-(\bar{e}^\lambda \gamma^\mu e^\lambda) + \frac{2}{3}(\bar{u}_j^\lambda \gamma^\mu u_j^\lambda) - \frac{1}{3}(\bar{d}_j^\lambda \gamma^\mu d_j^\lambda)) + \\
 & \frac{ig}{4c_w} Z_\mu^0 \{(\bar{\nu}^\lambda \gamma^\mu (1 + \gamma^5) \nu^\lambda) + (\bar{e}^\lambda \gamma^\mu (4s_w^2 - 1 - \gamma^5) e^\lambda)\} + (\bar{d}_j^\lambda \gamma^\mu (\frac{4}{3}s_w^2 - 1 - \gamma^5) d_j^\lambda) + \\
 & (\bar{u}_j^\lambda \gamma^\mu (1 - \frac{8}{3}s_w^2 + \gamma^5) u_j^\lambda) + \frac{ig}{2\sqrt{2}} W_\mu^+ \{(\bar{\nu}^\lambda \gamma^\mu (1 + \gamma^5) U^{lep} \nu^\lambda) + (\bar{u}_j^\lambda \gamma^\mu (1 + \gamma^5) C_{\lambda \kappa} d_j^\kappa)\} + \\
 & \frac{ig}{2\sqrt{2}} W_\mu^- \left( (\bar{e}^\kappa U^{lep} \kappa \lambda \gamma^\mu (1 + \gamma^5) \nu^\lambda) + (\bar{d}_j^\kappa C_{\lambda \kappa}^\dagger \gamma^\mu (1 + \gamma^5) u_j^\lambda) \right) + \\
 & \frac{ig}{2M\sqrt{2}} \phi^+ (-m_e^\kappa (\bar{\nu}^\lambda U^{lep} \lambda \kappa (1 - \gamma^5) e^\kappa) + m_\nu^\kappa (\bar{\nu}^\lambda U^{lep} \lambda \kappa (1 + \gamma^5) e^\kappa)) + \\
 & \frac{ig}{2M\sqrt{2}} \phi^- (m_e^\lambda (\bar{e}^\lambda U^{lep} \lambda \kappa (1 + \gamma^5) \nu^\kappa) - m_\nu^\lambda (\bar{e}^\lambda U^{lep} \lambda \kappa (1 - \gamma^5) \nu^\kappa)) - \frac{g m_\lambda^\lambda}{2M} H (\bar{\nu}^\lambda \nu^\lambda) - \\
 & \frac{g m_\lambda^R}{2M} H (\bar{e}^\lambda e^\lambda) + \frac{ig m_\lambda^R}{2M} \phi^0 (\bar{\nu}^\lambda \gamma^5 \nu^\lambda) - \frac{ig m_\lambda^R}{2M} \phi^0 (\bar{e}^\lambda \gamma^5 e^\lambda) - \frac{1}{4} \bar{\nu}_\lambda M_{\lambda \kappa}^R (1 - \gamma_5) \bar{\nu}_\kappa - \\
 & \frac{1}{4} \bar{\nu}_\lambda M_{\lambda \kappa}^R (1 - \gamma_5) \hat{\nu}_\kappa + \frac{ig}{2M\sqrt{2}} \phi^+ (-m_d^\kappa (\bar{u}_j^\lambda C_{\lambda \kappa} (1 - \gamma^5) d_j^\kappa) + m_u^\lambda (\bar{u}_j^\lambda C_{\lambda \kappa} (1 + \gamma^5) d_j^\kappa) + \\
 & \frac{ig}{2M\sqrt{2}} \phi^- (m_d^\lambda (\bar{d}_j^\lambda C_{\lambda \kappa}^\dagger (1 + \gamma^5) u_j^\kappa) - m_u^\kappa (\bar{d}_j^\kappa C_{\lambda \kappa}^\dagger (1 - \gamma^5) u_j^\kappa)) - \frac{g m_\lambda^\lambda}{2M} H (\bar{u}_j^\lambda u_j^\kappa) - \\
 & \frac{g m_\lambda^R}{2M} H (\bar{d}_j^\lambda d_j^\lambda) + \frac{ig m_\lambda^R}{2M} \phi^0 (\bar{u}_j^\lambda \gamma^5 u_j^\lambda) - \frac{ig m_\lambda^R}{2M} \phi^0 (\bar{d}_j^\lambda \gamma^5 d_j^\lambda) + \bar{G}^a \partial^2 G^a + g_s f^{abc} \partial_\mu \bar{G}^a G^b g_\mu^c + \\
 & \bar{X}^+ (\partial^2 - M^2) X^+ + X^- (\partial^2 - M^2) X^- + \bar{X}^0 (\partial^2 - \frac{M^2}{c_w^2}) X^0 + \bar{Y} \partial^2 Y + ig c_w W_\mu^+ (\partial_\mu \bar{X}^0 X^- - \\
 & \partial_\mu \bar{X}^+ X^-) + ig s_w W_\mu^+ (\partial_\mu \bar{X}^- Y - \partial_\mu \bar{Y} X^+) + ig c_w W_\mu^- (\partial_\mu \bar{X}^+ Y - \partial_\mu \bar{Y} X^-) + ig s_w A_\mu (\partial_\mu \bar{X}^+ X^- - \\
 & \partial_\mu \bar{X}^- X^+) + ig s_w A_\mu (\partial_\mu \bar{X}^- Y - \partial_\mu \bar{Y} X^+) - ig M (\bar{X}^+ X^+ H + \bar{X}^- X^- H + \frac{1}{c_w^2} \bar{X}^0 X^0 H) + \frac{1-2c_w^2}{2c_w} ig M (\bar{X}^+ X^+ H + \bar{X}^- X^- H) - \\
 & \frac{1}{2c_w} ig M (\bar{X}^0 X^- \phi^+ - \bar{X}^0 X^+ \phi^-) + ig M s_w (\bar{X}^0 X^- \phi^+ - \bar{X}^0 X^+) - \frac{1}{2}ig M (\bar{X}^+ X^+ \phi^0 - \bar{X}^- X^- \phi^0).
 \end{aligned}$$

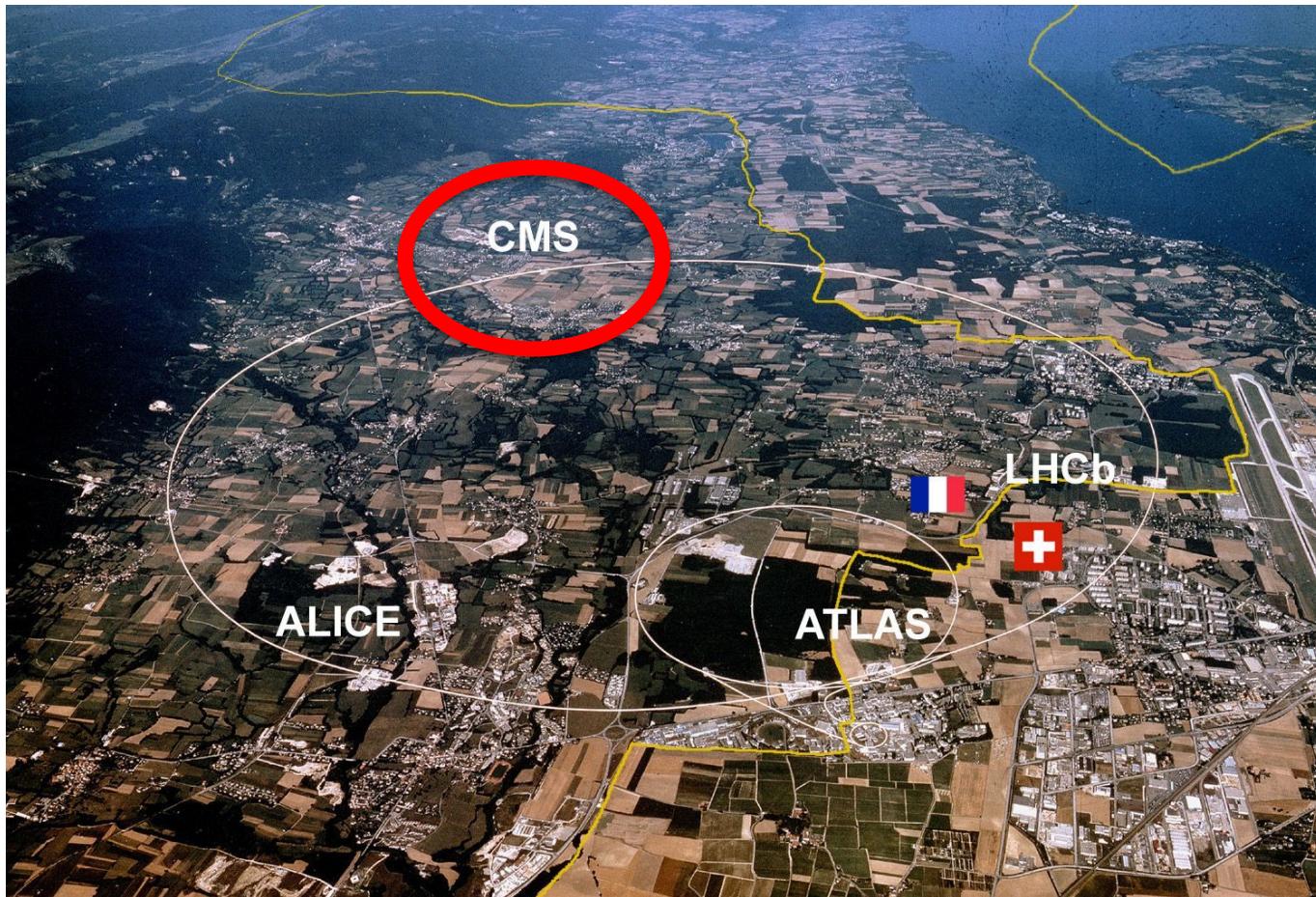


# What we don't know?



Searching for new fundamental physics  
at LHC to explain shortcomings

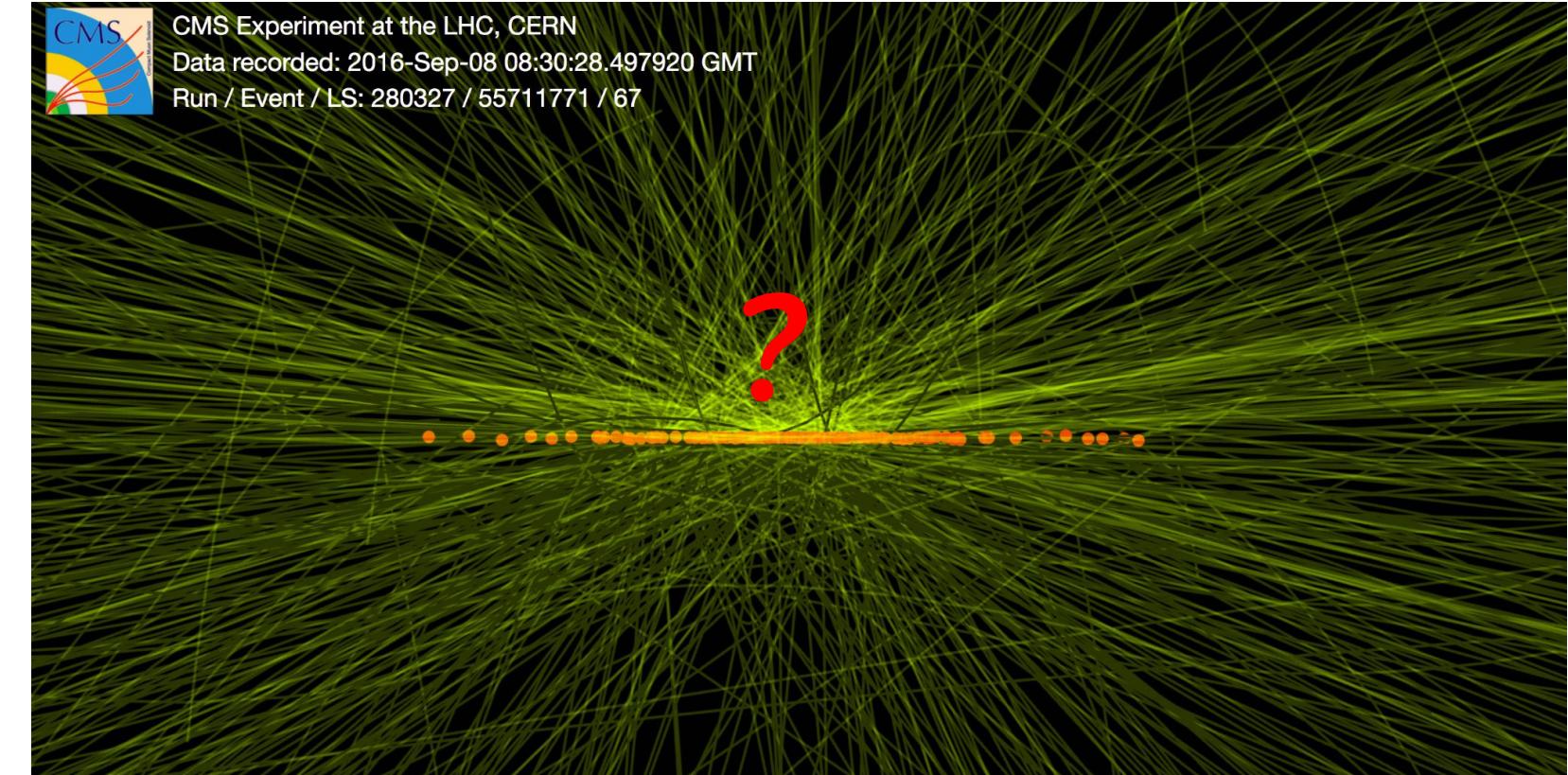
# LHC collisions



Large = 27km ring of superconducting magnets, 100m underground French-Swiss border

Hadron = accelerate protons (hadrons) to almost the speed of light

Collider = two proton beams smashed together  
**40 million times a second** at four points around the ring



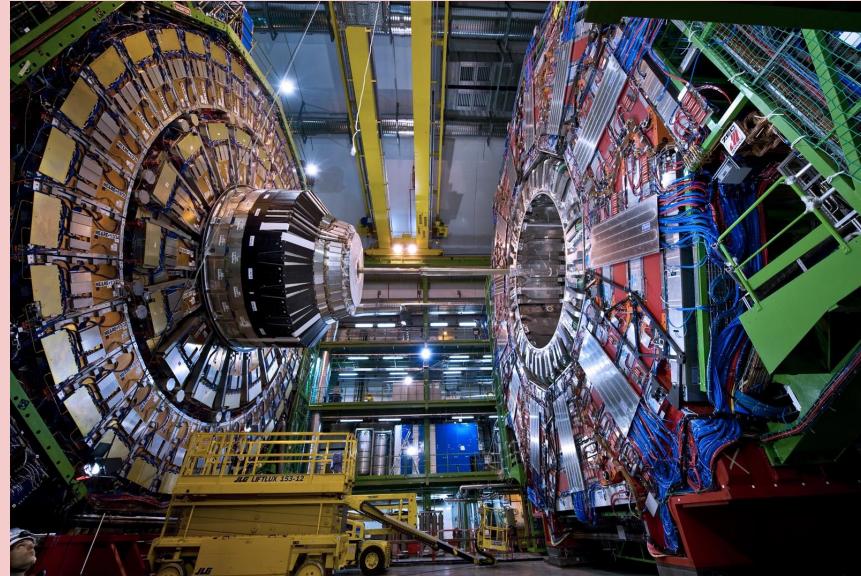
At interaction points collision fragments are photographed with massive imaging detectors

Proton collisions are messy → requires state-of-the-art detector technology

# CMS detector

- One of two general purpose detectors @ LHC (CMS + ATLAS) → Giant camera operating at 40M frames per second

## Compact Muon Solenoid



**Length:** 28m

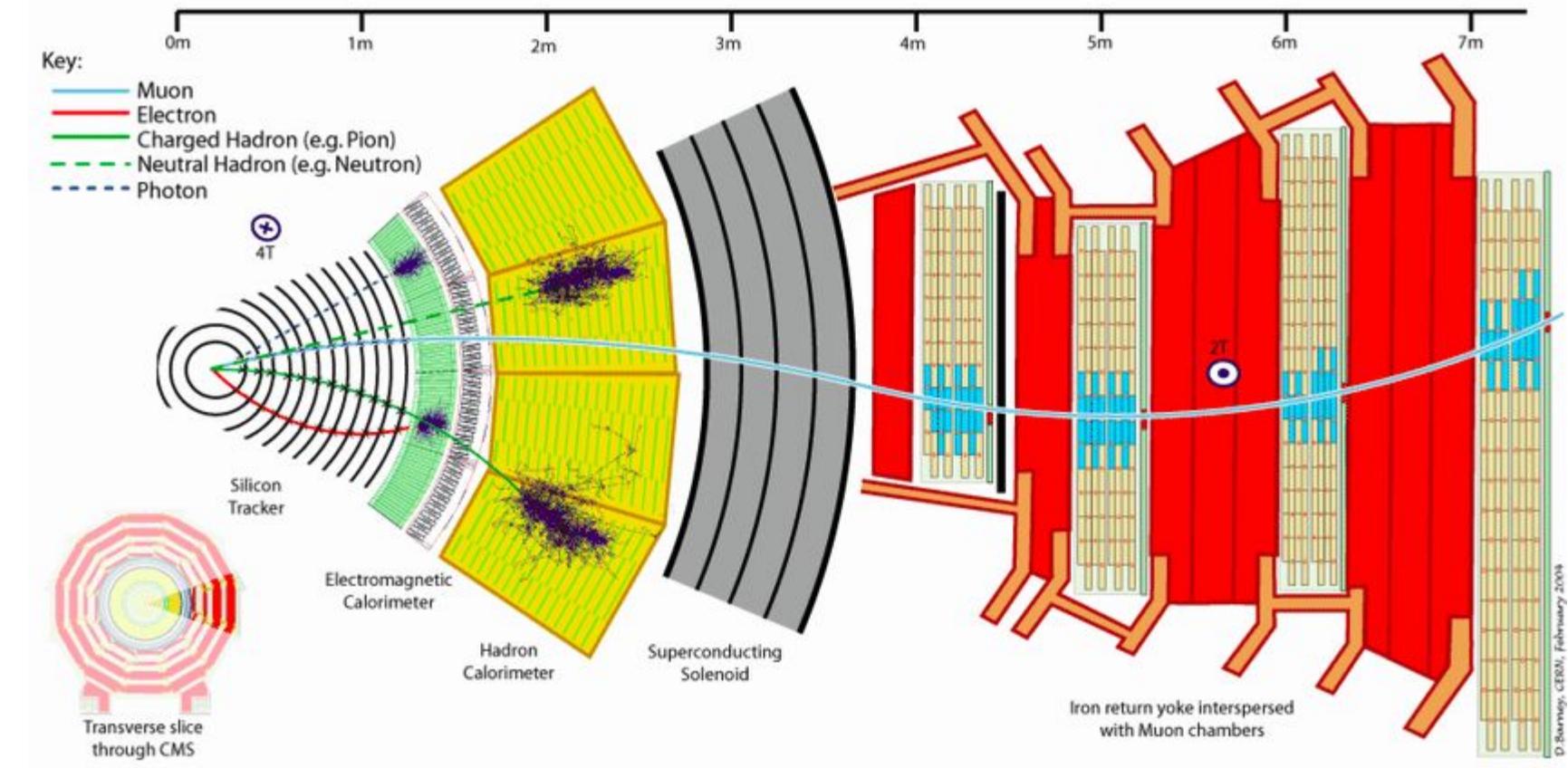
**Diameter:** 15m

**Weight:** 14,000 tonnes

**Superconducting solenoid:** 3.8T

**Readout channels:** O(100) million

**Radiation environment:** Extreme!!!



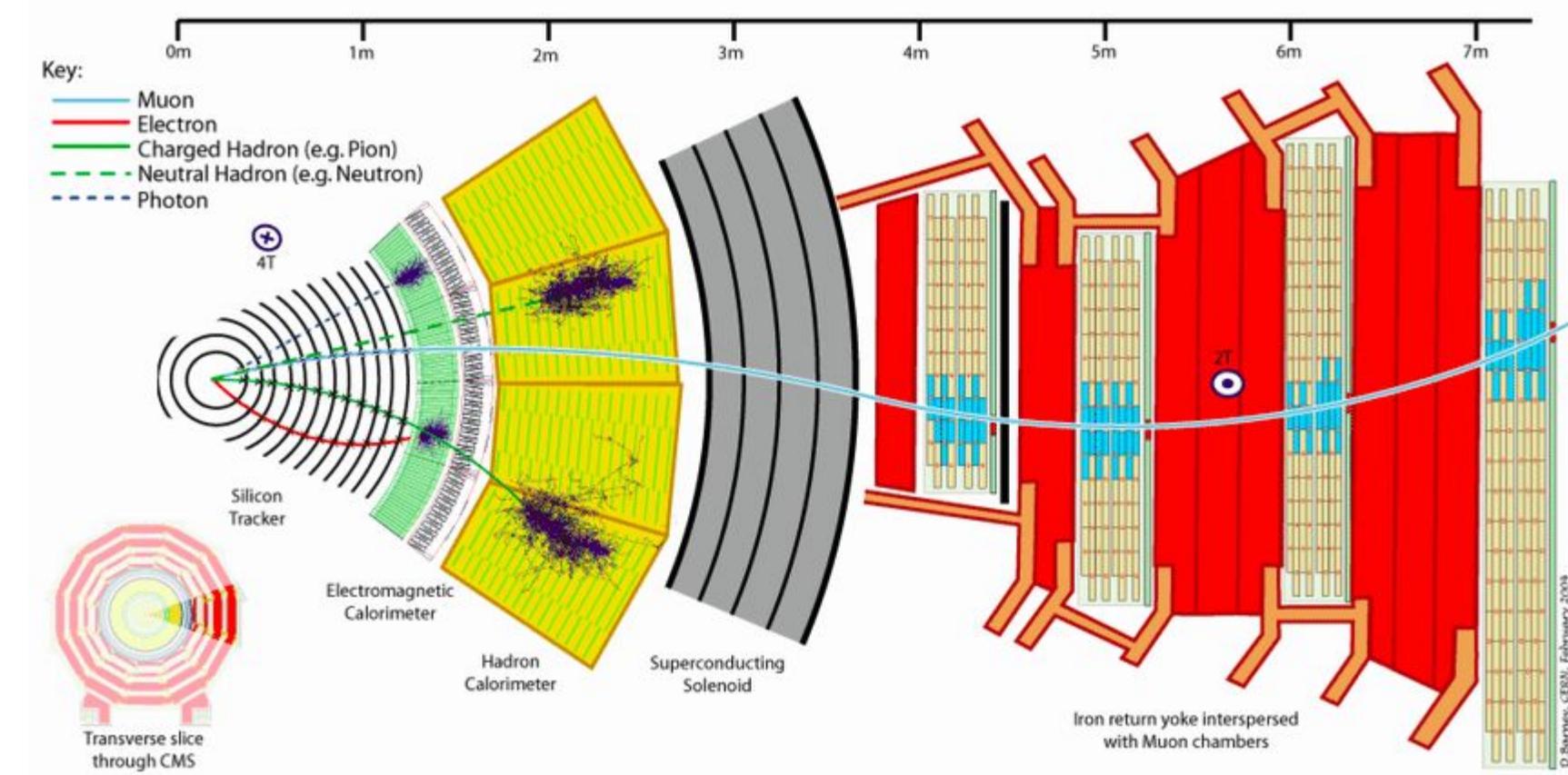
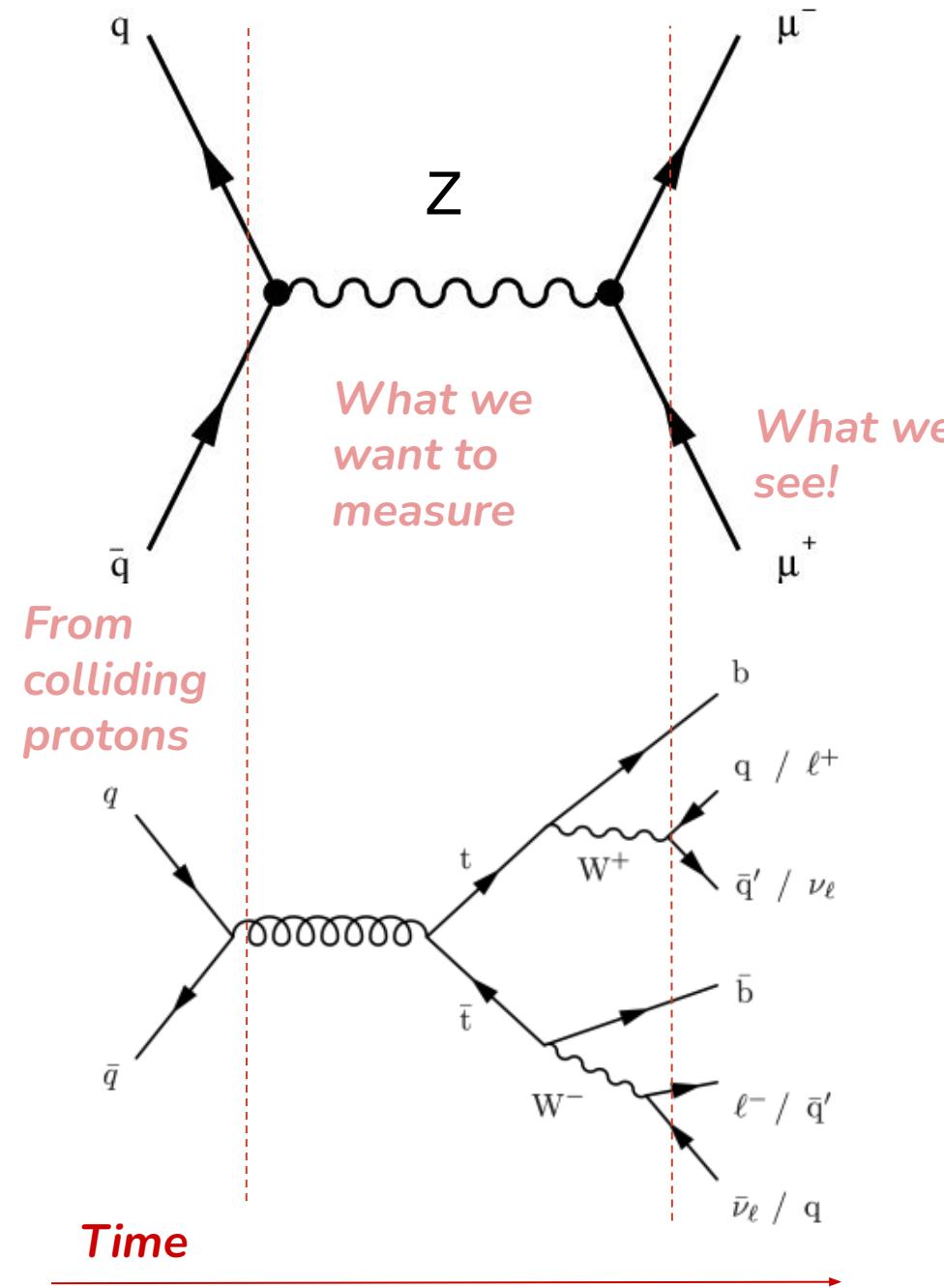
Interesting physics (e.g. Higgs boson) is unstable → infer its presence from decay products

Layers of “sub-detectors” to measure different final state particles

Essentially a pattern recognition problem

# CMS detector

- One of two general purpose detectors @ LHC (CMS + ATLAS) → Giant camera operating at 40M frames per second



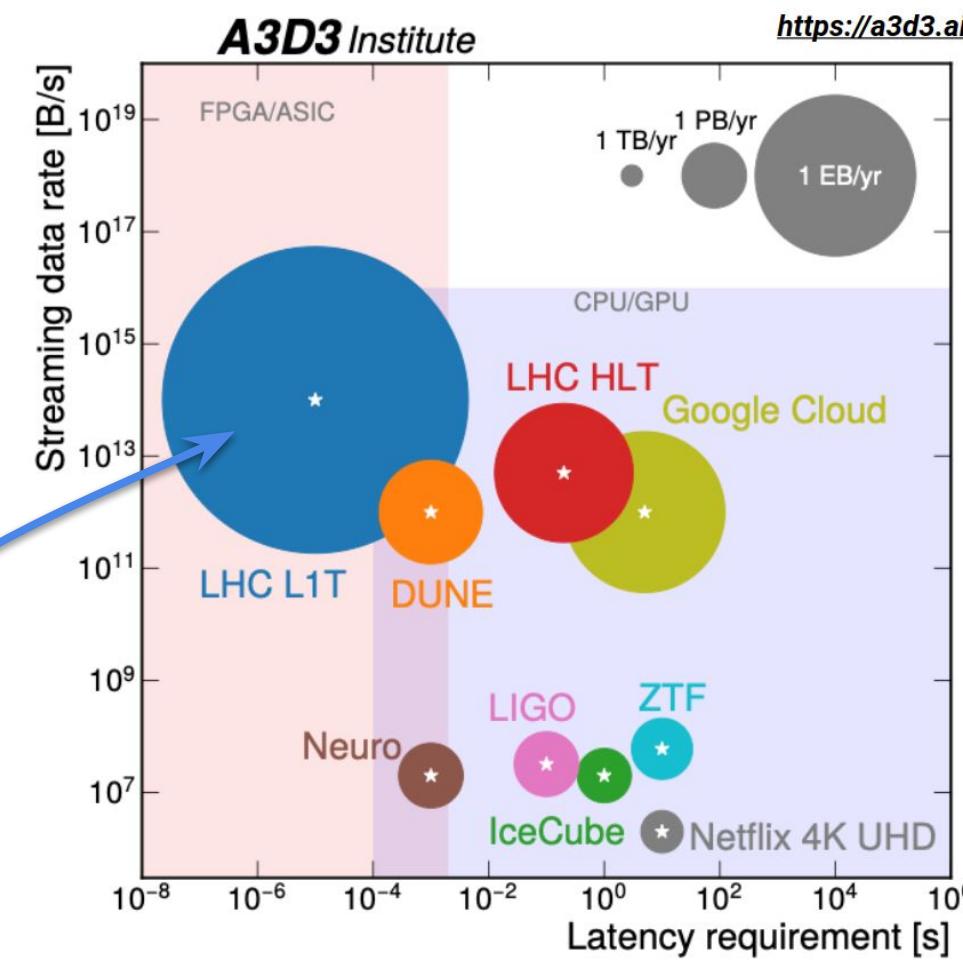
Interesting physics (e.g. Higgs boson) is unstable → infer its presence from decay products

Layers of “sub-detectors” to measure different final state particles

Essentially a pattern recognition problem

# Needle in the haystack

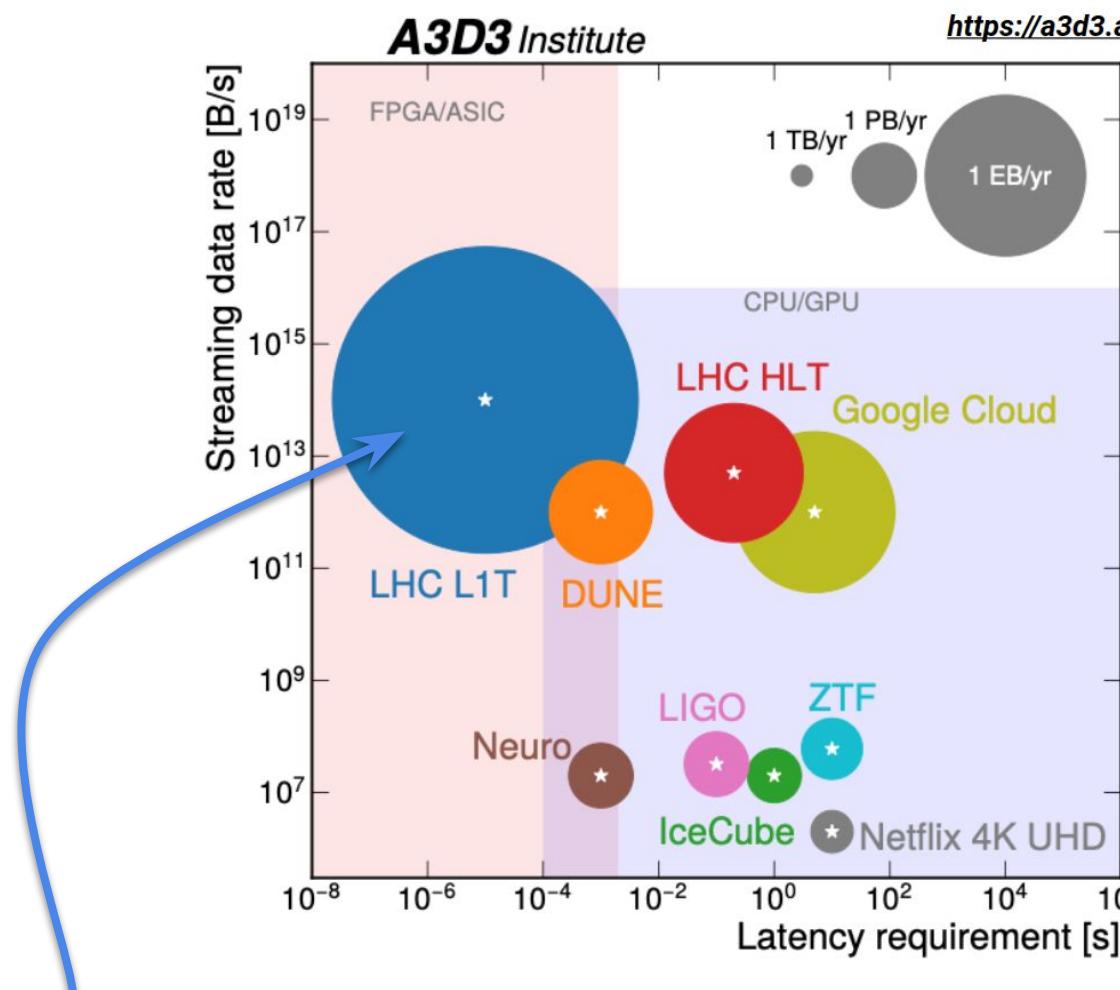
- 40 MHz collision rate x O(100 million) readout channels =  
~500Tb data produced by CMS per-second
- We cannot write all of the data to disk!



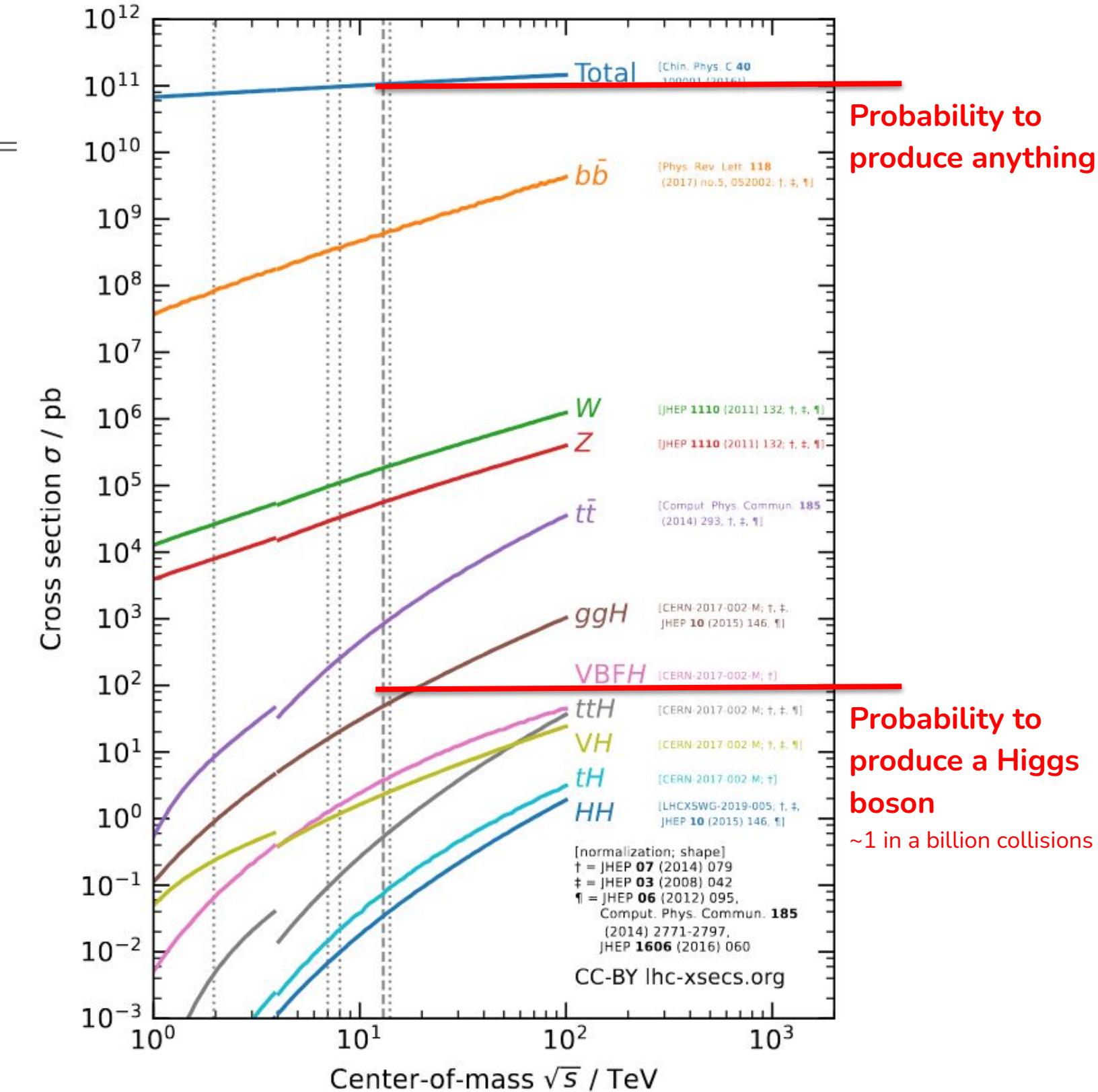
Cutting-edge of data science

# Needle in the haystack

- 40 MHz collision rate x O(100 million) readout channels =  
~500Tb data produced by CMS per-second
- We cannot write all of the data to disk!



Cutting-edge of data science

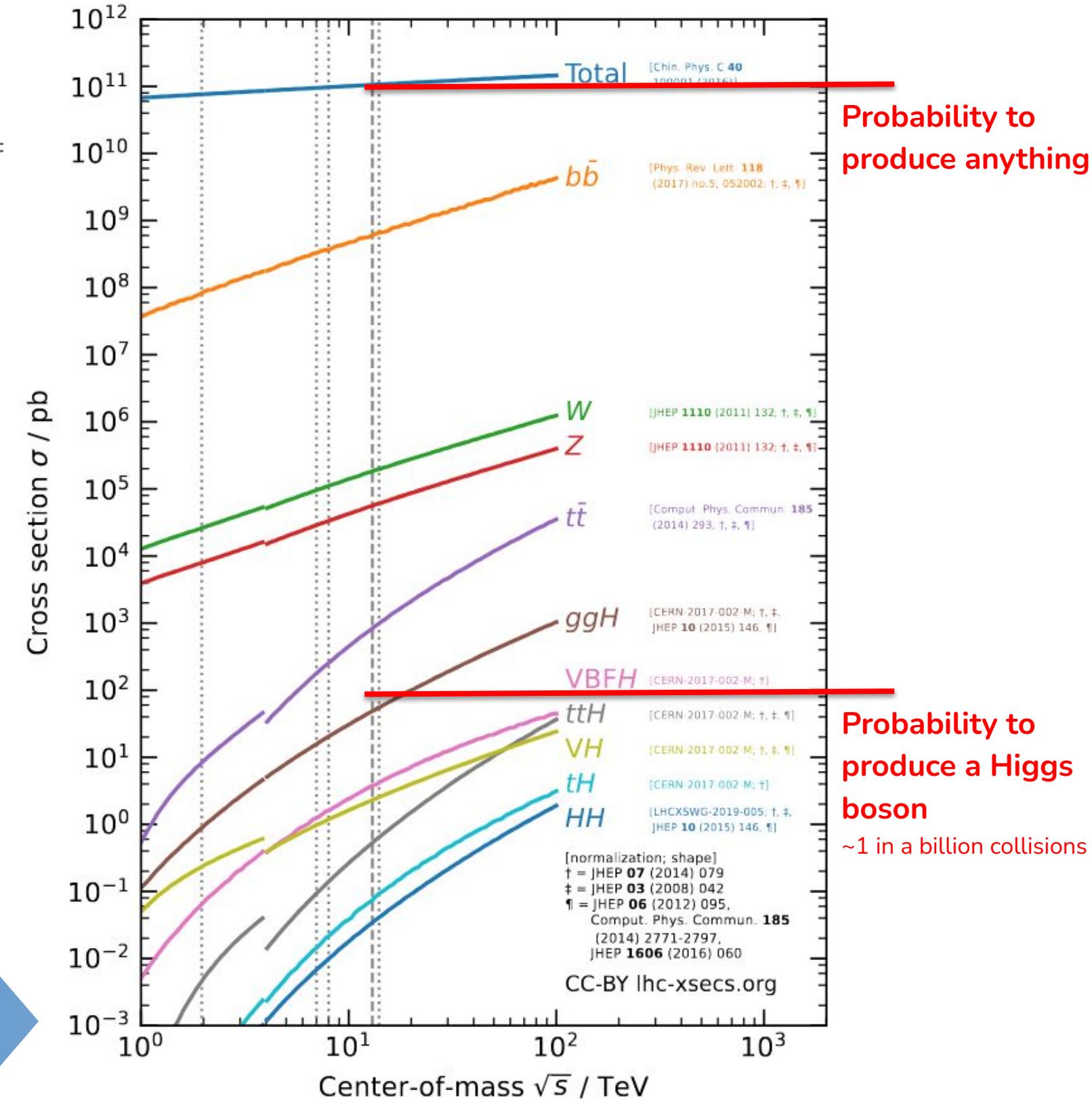
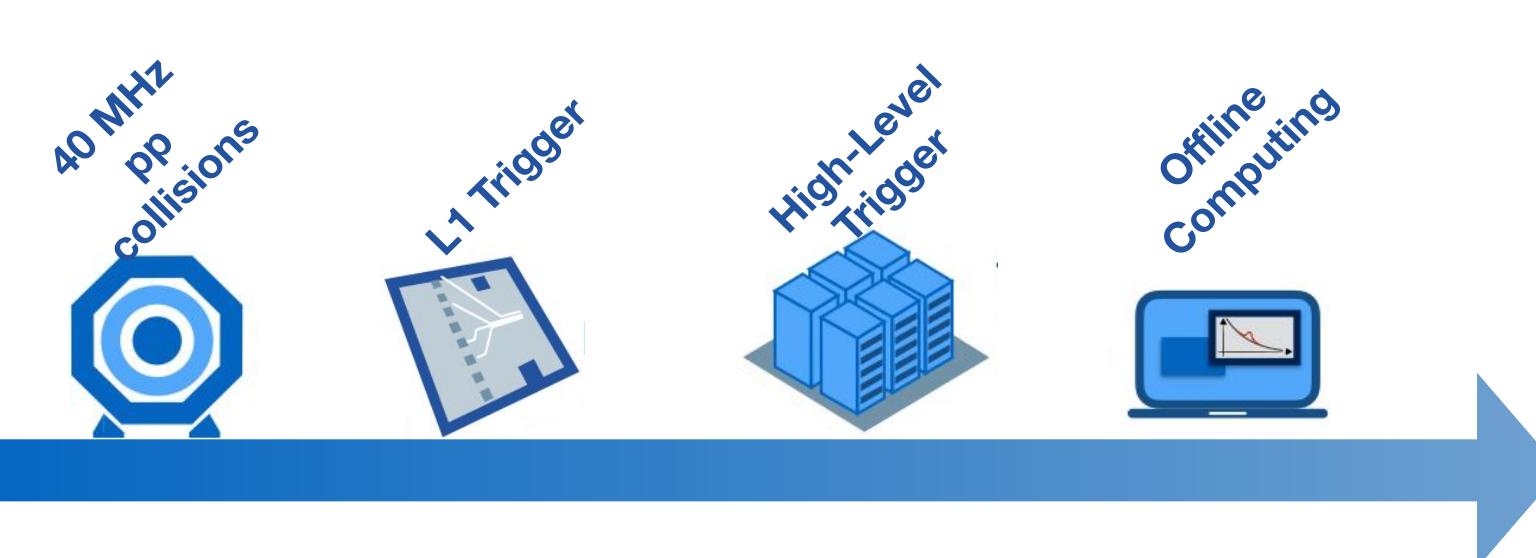


Probability to produce anything

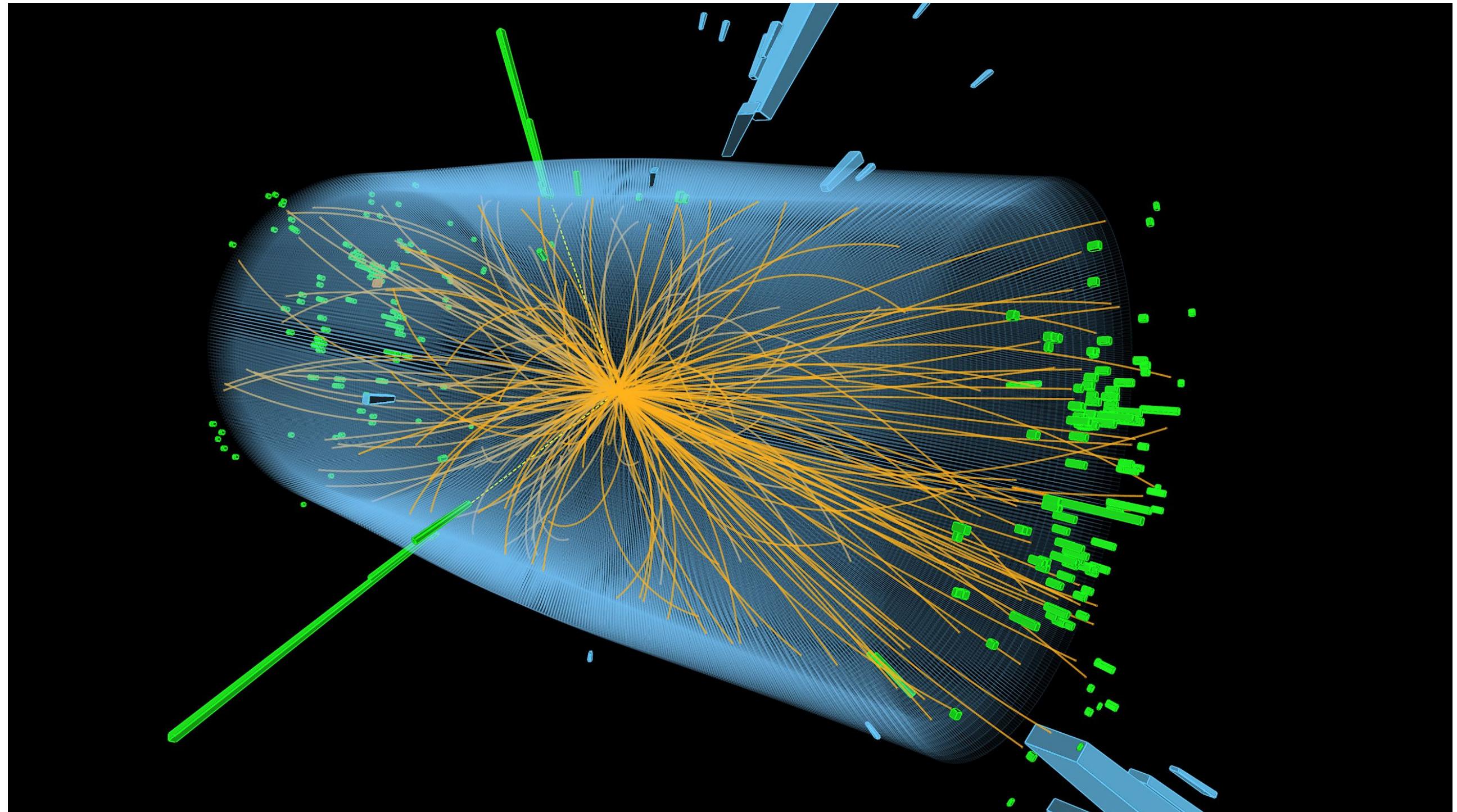
Probability to produce a Higgs boson  
~1 in a billion collisions

# Needle in the haystack

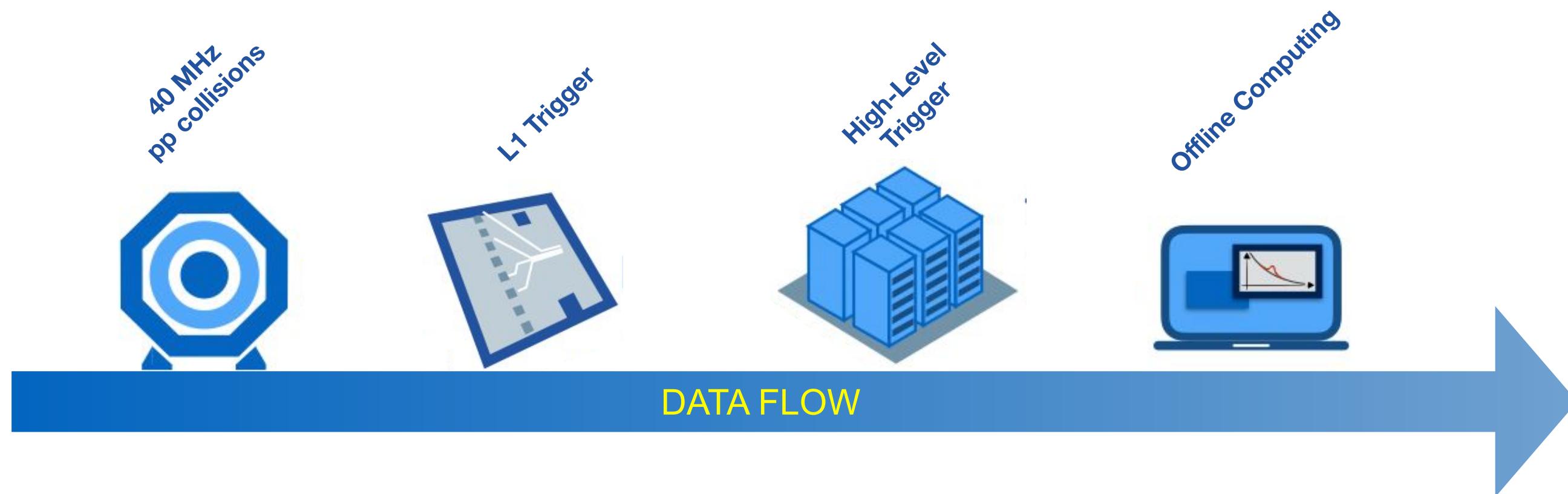
- 40 MHz collision rate  $\times$  O(100 million) readout channels =  
~500Tb data produced by CMS per-second
- We cannot write all of the data to disk!
- Online filtering: **CMS level-1 Trigger**
  - Identify interesting collisions in the ultra-low latency (~ $\mu$ s), high-bandwidth environment
  - Decide which data to save for offline analysis



# Higgs to two photons



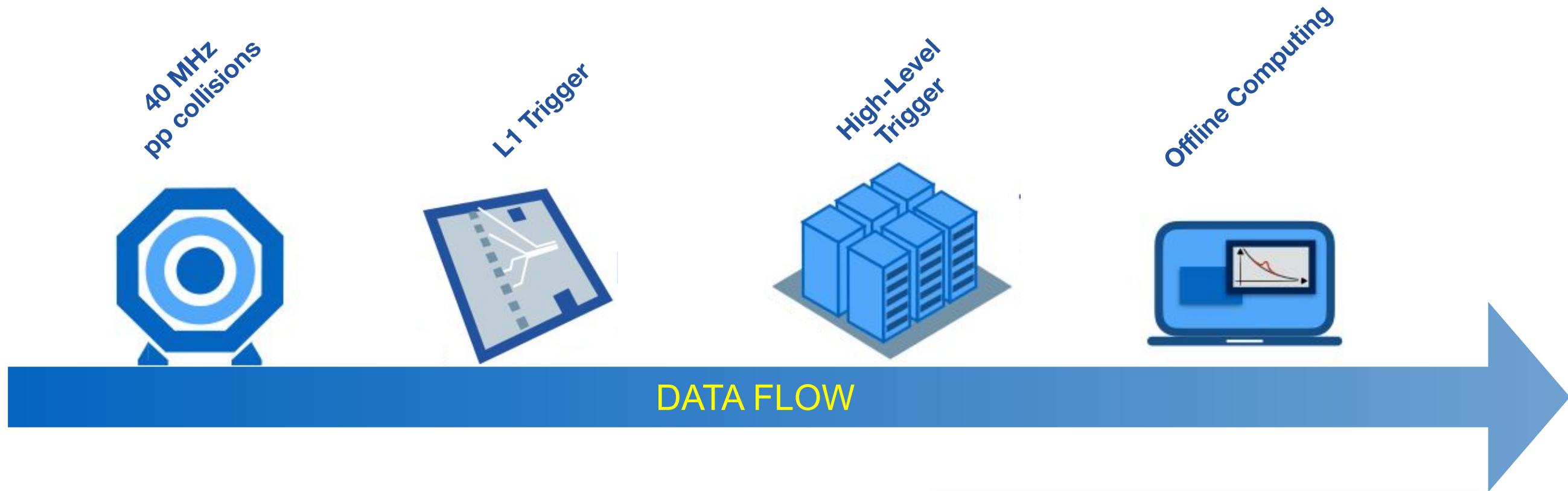
# LHC Experiment Data Flow



## L1 trigger:

- 40 MHz in / 100 KHz out
- Process 100s TB/s
- Trigger decision to be made in  $\approx 10 \mu\text{s}$
- Coarse local reconstruction
- FPGAs / Hardware implemented

# LHC Experiment Data Flow



## L1 trigger:

- 40 MHz in / 100 KHz out
- Process 100s TB/s
- Trigger decision to be made in  $\approx 10 \mu\text{s}$
- Coarse local reconstruction
- FPGAs / Hardware implemented

## Key warning:

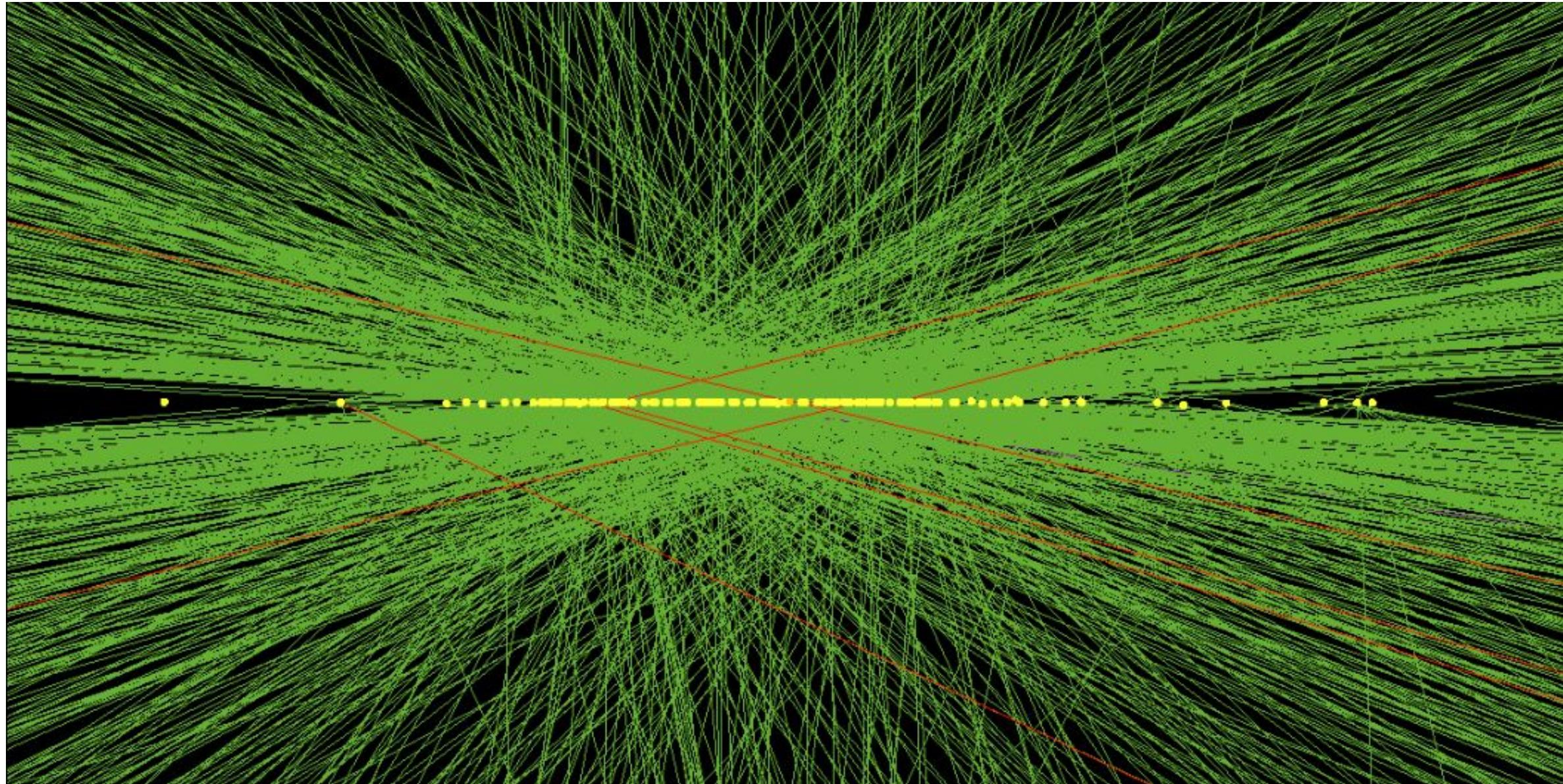
Trigger discards data forever!

We need trigger algorithms to be:

- Very fast = get more data through
- Very accurate = select the right data

# Triggering @ HL-LHC

In 2029 we move to High-Luminosity LHC operations → Increase data rate by a factor of ~5



- Today's (traditional) trigger algorithms will fail @ HL-LHC

**FAIL**



# CMS Level-1 Trigger upgrade

- System of ~1000 FPGAs to decide which collisions to keep within  $\sim 12\mu\text{s}$



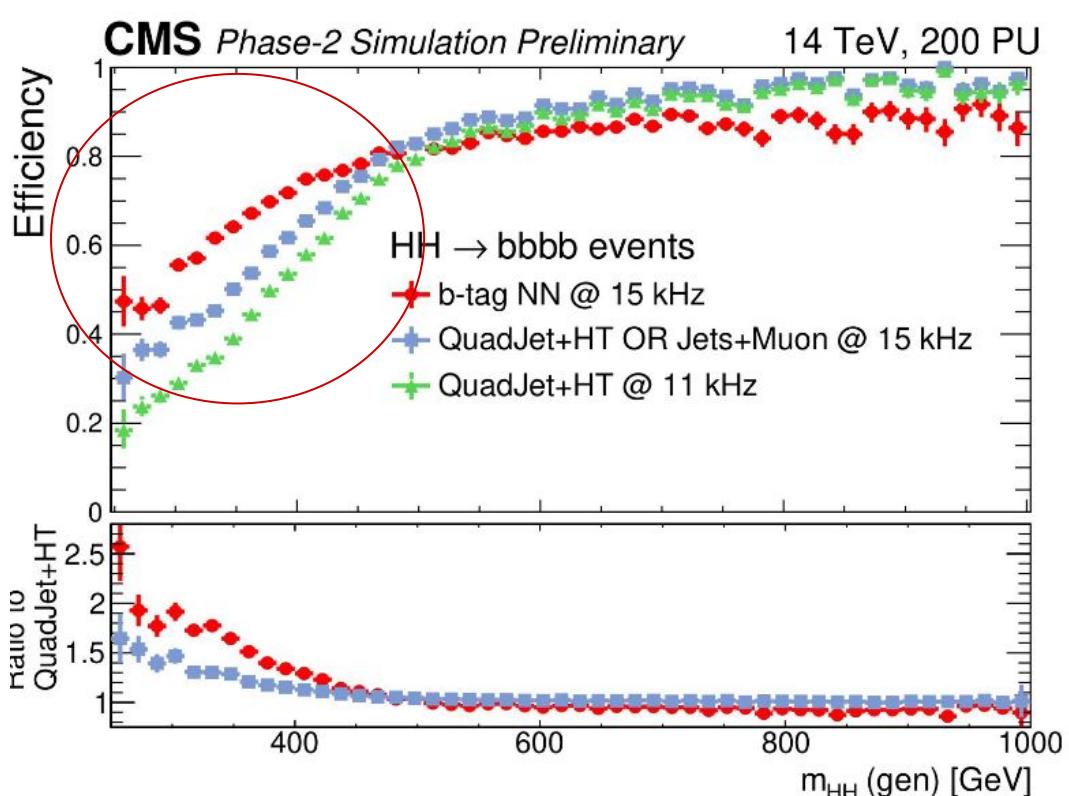
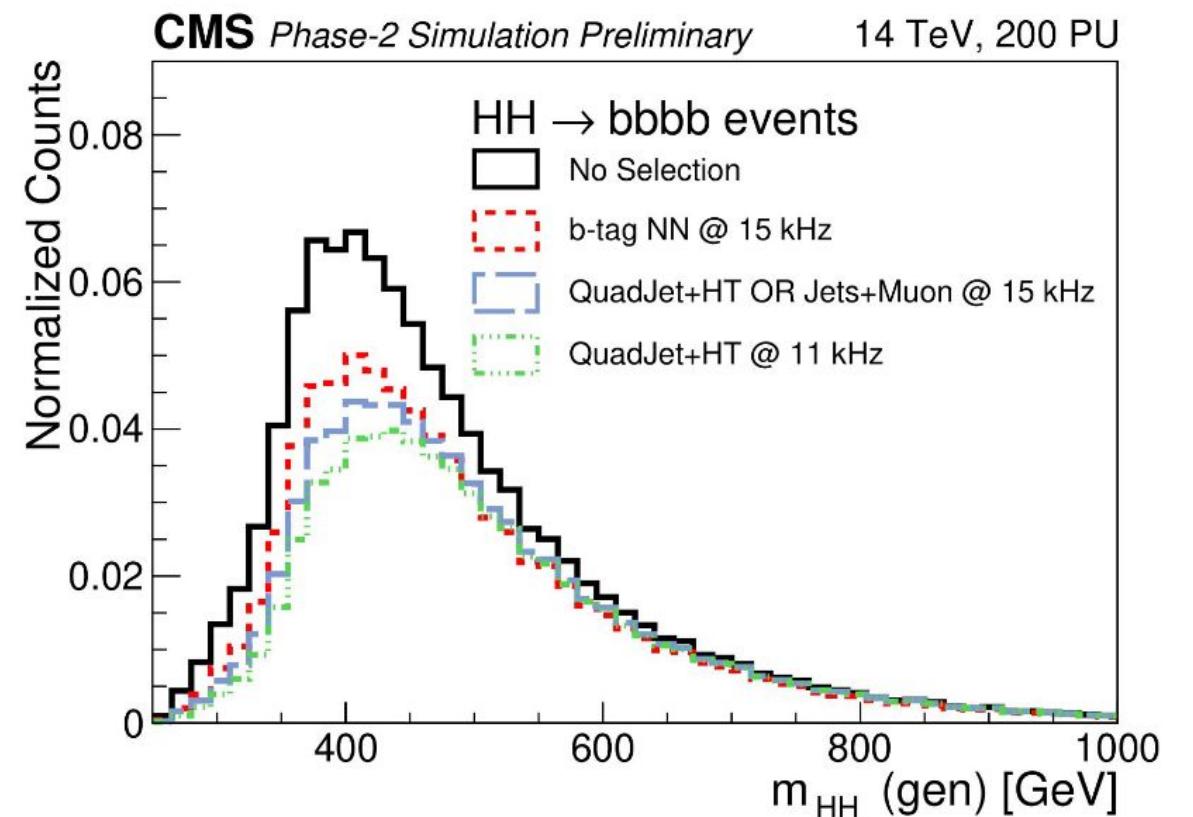
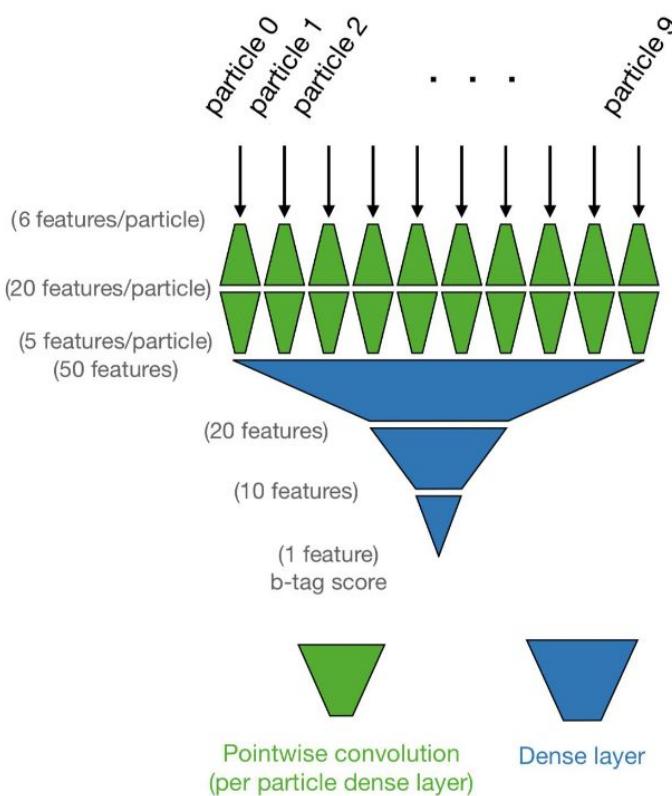
# CMS Level-1 Trigger upgrade

- System of ~1000 FPGAs to decide which collisions to keep within  $\sim 12\mu\text{s}$
- Facilitates **AI inference** in ultra-low latency, high-bandwidth environment
  - Sophisticated “real-time” (online) selection of interesting physics in harsher HL-LHC collisions
  - AI is necessary to bring data rate to manageable level whilst maintaining high efficiency



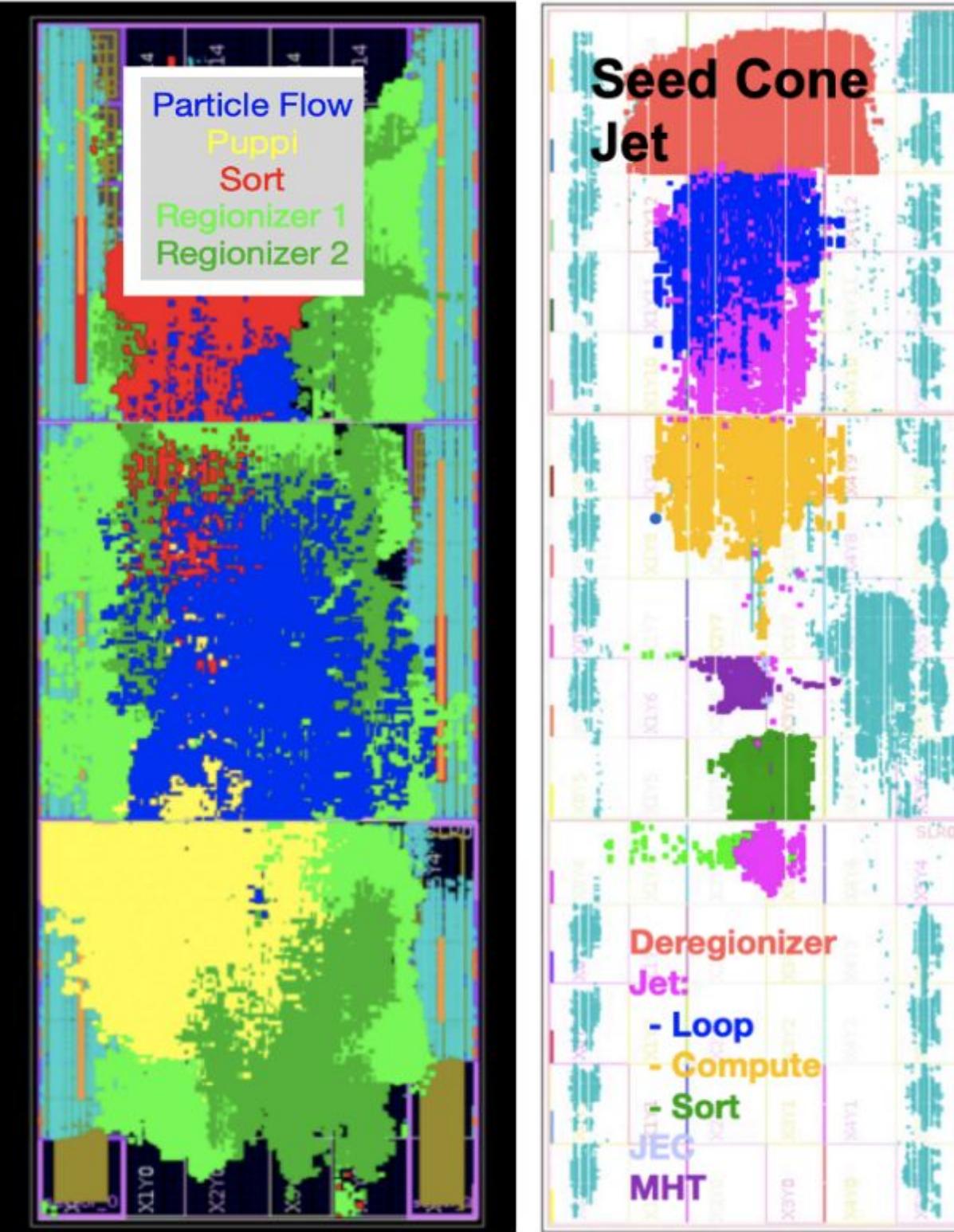
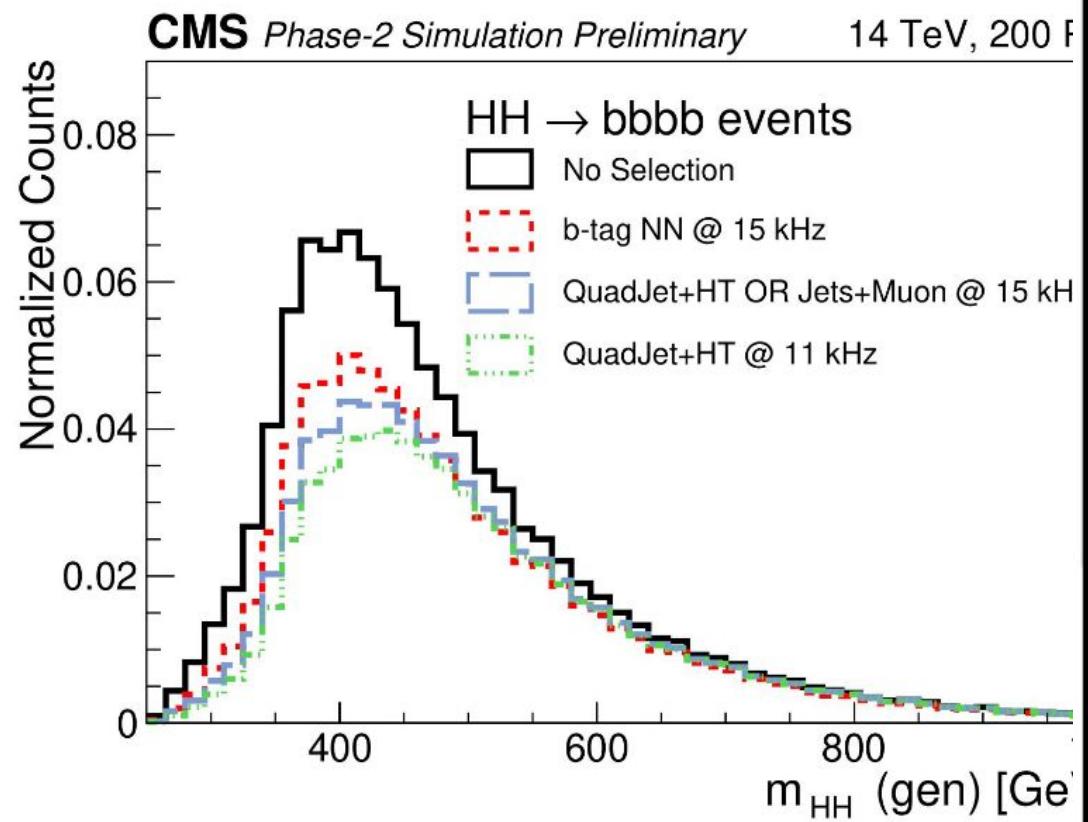
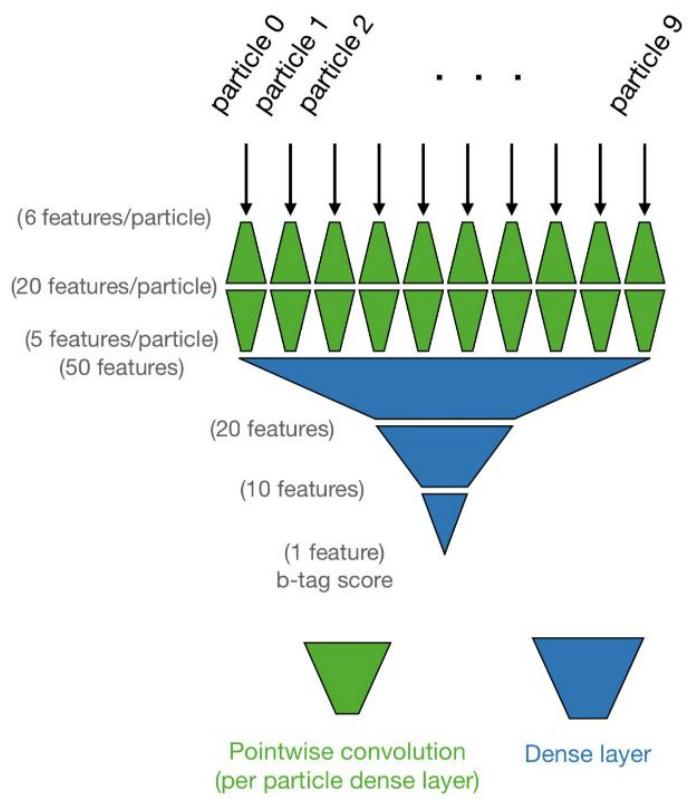
# CMS Level-1 Trigger upgrade

- System of  $\sim 1000$  FPGAs to decide which collisions to keep within  $\sim 12\mu\text{s}$
- Facilitates **AI inference** in ultra-low latency, high-bandwidth environment
  - Sophisticated “real-time” (online) selection of interesting physics in harsher HL-LHC collisions
  - AI is necessary to bring data rate to manageable level whilst maintaining high efficiency

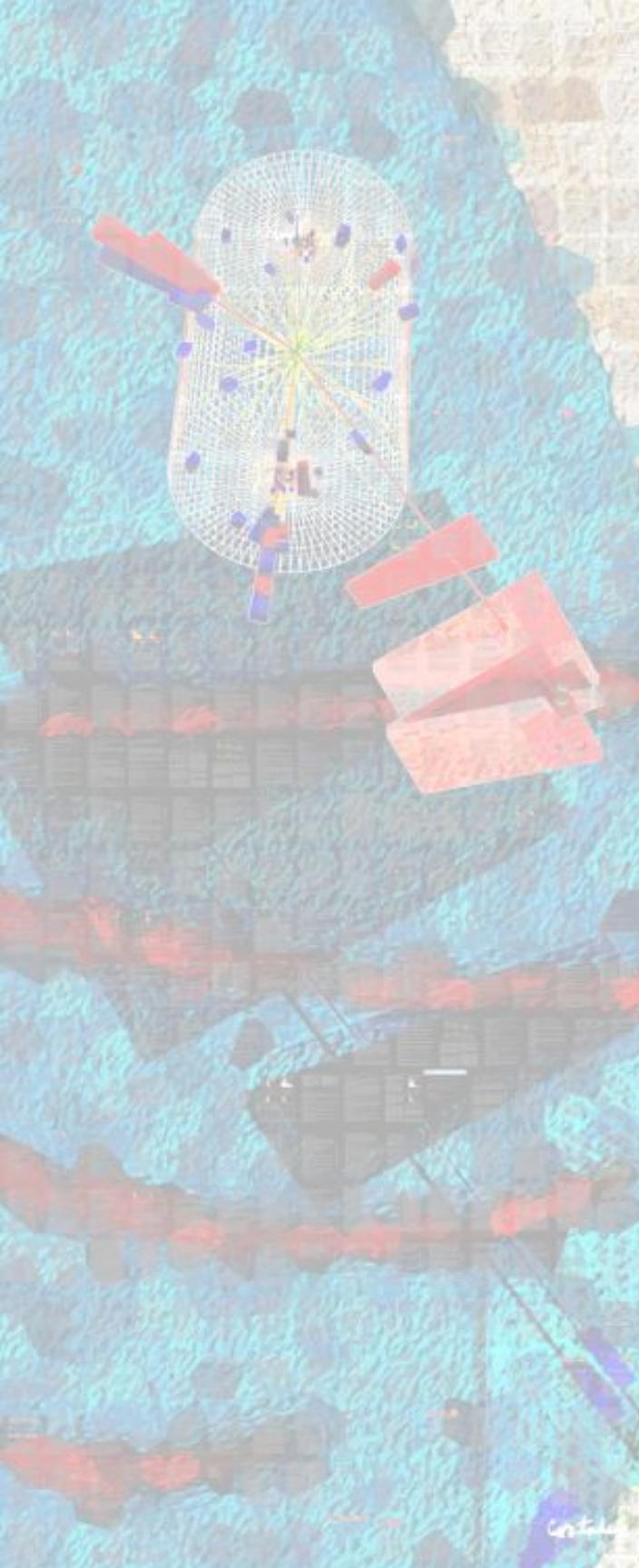


# CMS Level-1 Trigger upgrade

- System of  $\sim 1000$  FPGAs to decide which collisions to keep within  $\sim 12\mu\text{s}$
- Facilitates **AI inference** in ultra-low latency, high-bandwidth environment
  - Sophisticated “real-time” (online) selection of interesting physics in harsher HL
  - AI is necessary to bring data rate to manageable level whilst maintaining high



- We need to ensure the ML algorithms are designed efficiently → fit in latency, resource-usage and bandwidth constraints



## Tutorial

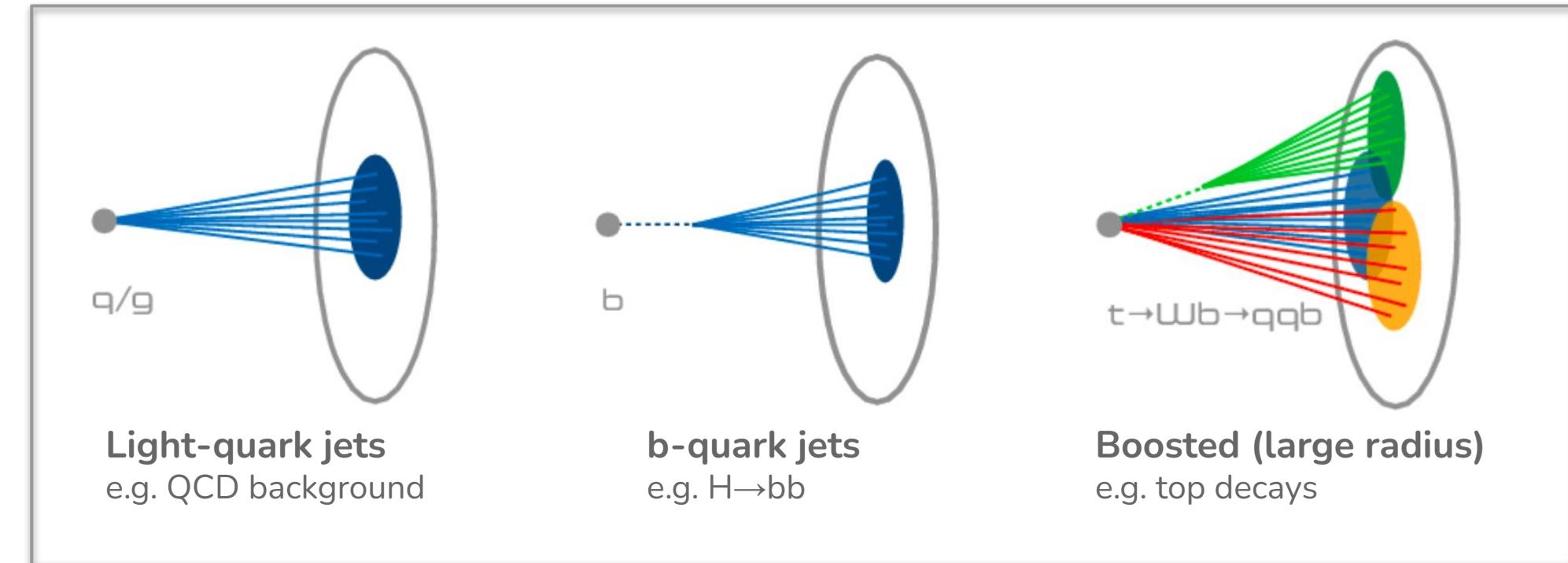
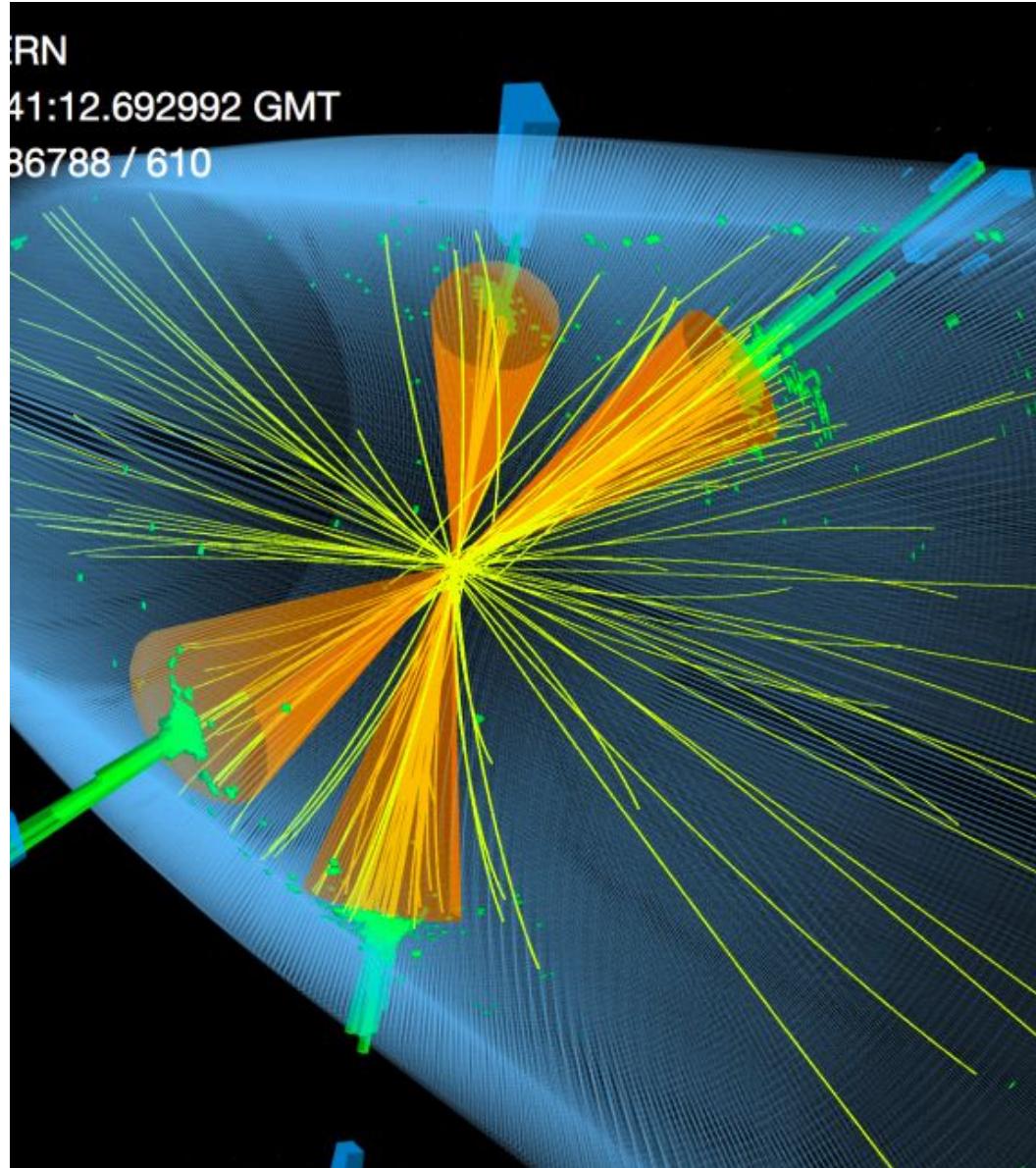
“Online” jet tagging @ LHC

(Parts 1, 2 and 3)

[\[Github link\]](#)

# Jet tagging

- Jet - collimated spray of particles ejected from primary interaction point
- Jets come in different flavours (classes) with different substructure → “Jet-tagging” = jet classification

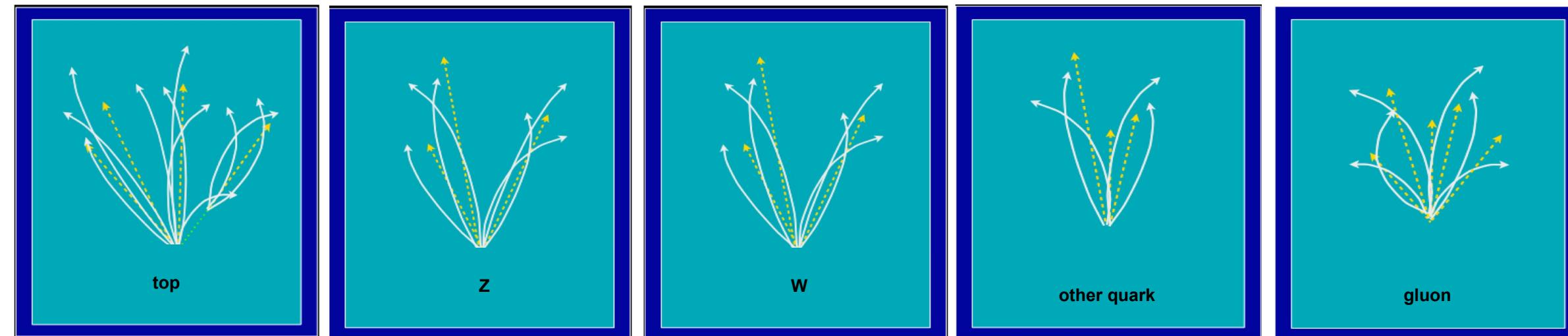


- Jet constituent particles produce patterns of “hits” as they traverse detector
  - A pattern recognition problem
  - Has become a huge frontier in ML over last years (see [ML4Jets](#))

# Tutorial problem

Goal: study a jet multi-classification task to be implemented on an FPGA

- Such an algorithm could be used in the CMS Level-1 Trigger to identify collisions of interest



$t \rightarrow bW \rightarrow bqq$

**3-prong jet**

$Z \rightarrow qq$

**2-prong jet**

$W \rightarrow qq'$

**2-prong jet**

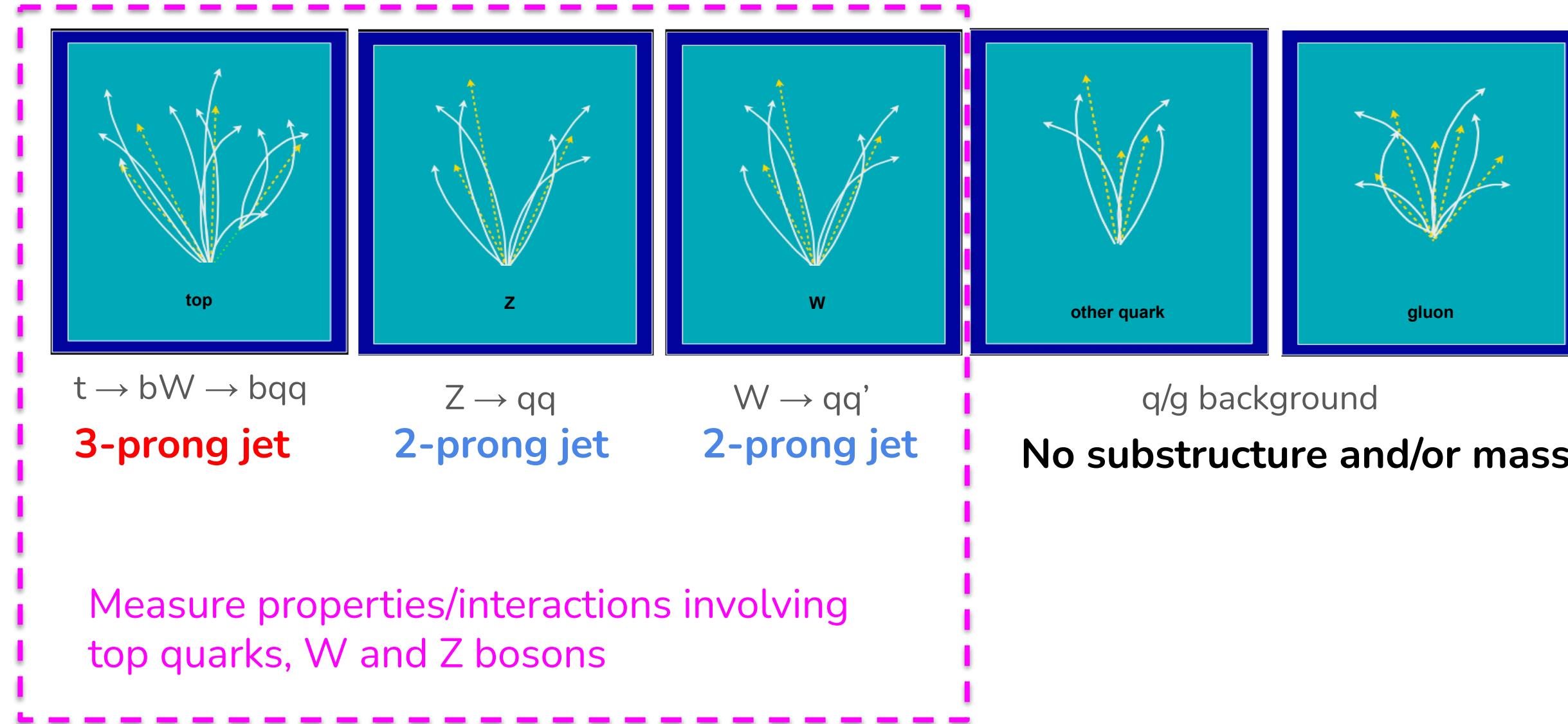
q/g background

**No substructure and/or mass**

# Tutorial problem

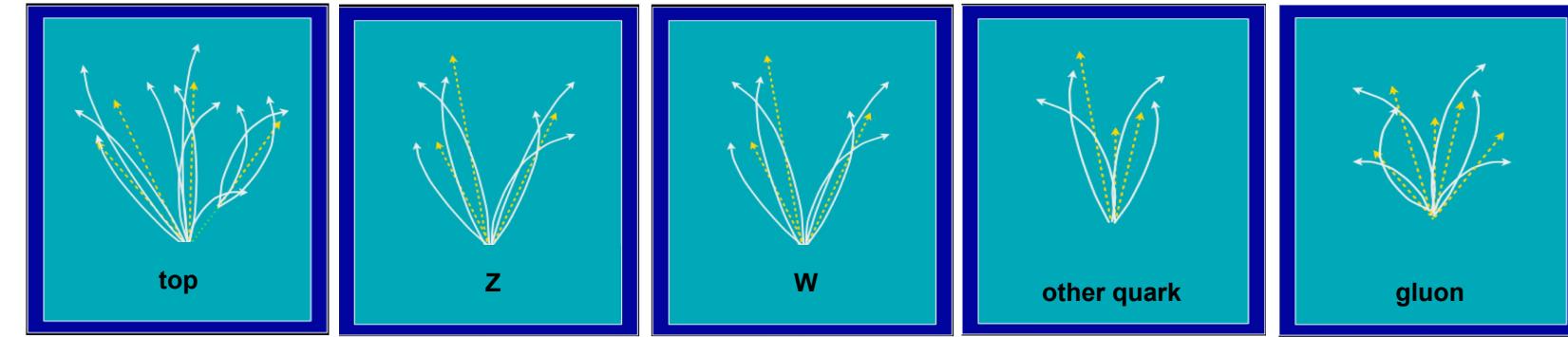
Goal: study a jet multi-classification task to be implemented on an FPGA

- Such an algorithm could be used in the CMS Level-1 Trigger to identify **collisions of interest**



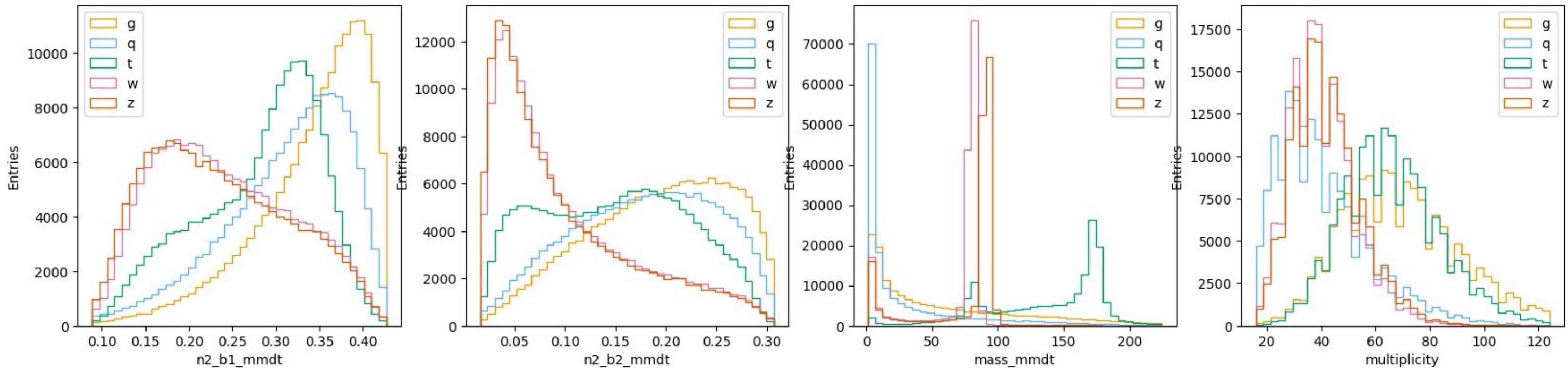
# Understanding the dataset

[More data set details](#)



Data set: ~1M jets (~200k of each class)

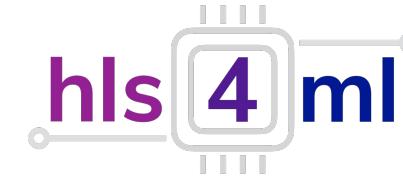
- 16 “input features” known to have high-discrimination power between the classes (four shown as example)



Task: train neural network to discriminate between the different classes and evaluate performance (**Parts 1 & 2**)

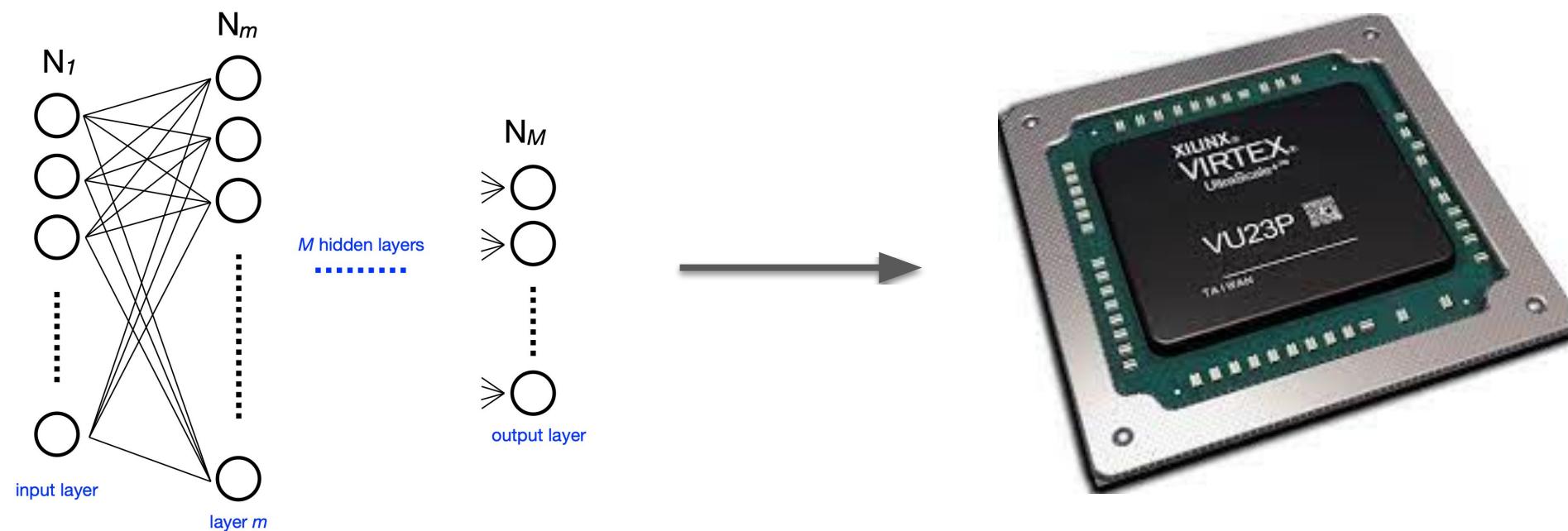
# Online jet-tagging

**Part 3:** convert the trained model to FPGA firmware using



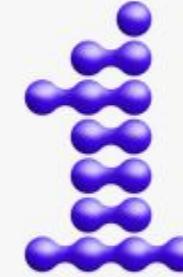
with Intel OneAPI backend (see next slide)

- Understand configuration for conversion and how this can be adapted to meet design constraints
- Evaluate accuracy of FPGA neural network
- Emulate the FPGA build and estimate the resource usage (LUTs, FFs, DSPs, BRAMs)
- Study how changing the bit-width during inference affects the performance and resource-usage



N.B. In this tutorial we will not estimate the algorithm latency → this requires a full simulation of the data flow  
Instead we will focus on resource-usage and how this can be optimised with more efficient NN designs

A Vision of Developer Freedom for the Future of Accelerated Compute



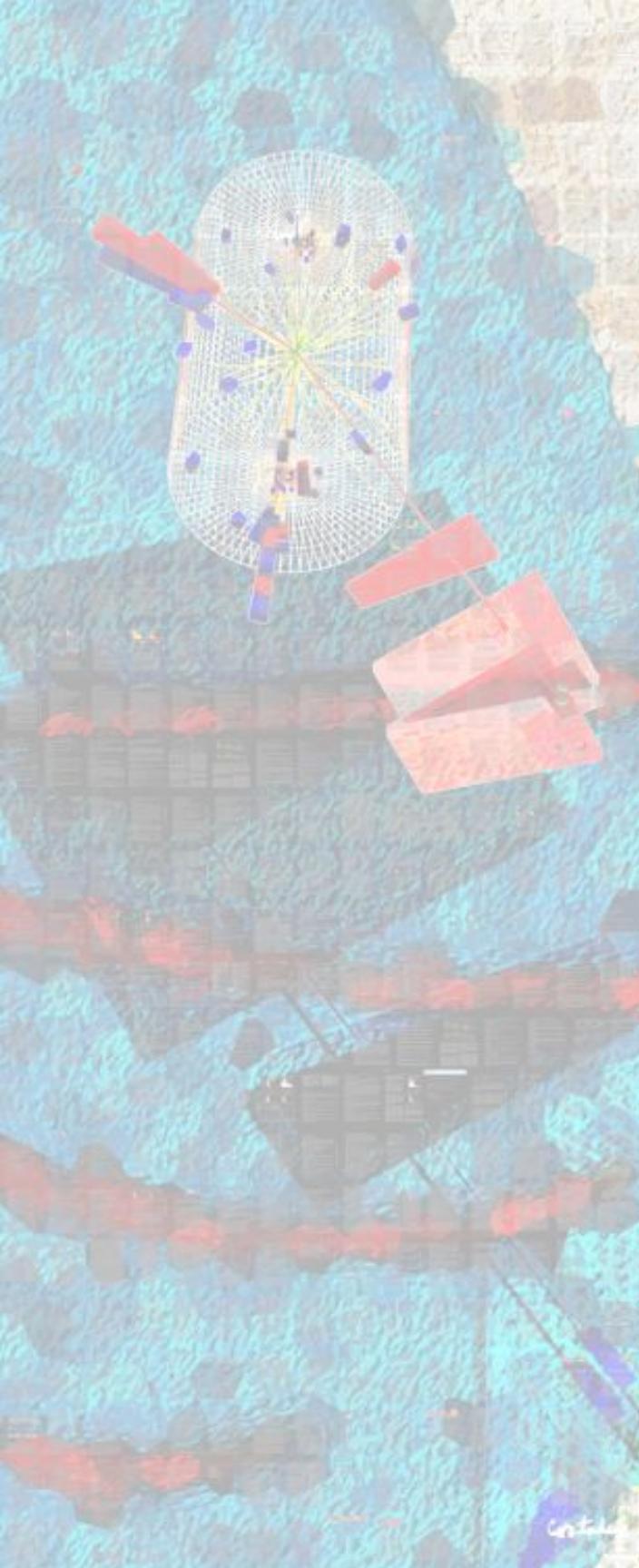
**oneAPI**

oneAPI provides a comprehensive set of libraries, open source repositories, SYCL-based C++ language extensions, and optimized reference implementations to accelerate the following goals:

- › Define a common, unified, and open multiarchitecture and multivendor software platform.
- › Ensure functional code portability and performance portability across hardware vendors and accelerator technologies.
- › Enable an extensive set of specifications and library APIs to cover programming domain needs across industries and compute as well as AI use cases.
- › Meet the needs of modern software applications that merge high-end computational needs and AI.
- › Provide a developer community and open forum to drive a unified API for a unified industry-wide multiarchitecture software development platform.
- › Encourage ecosystem collaboration on the oneAPI specification and compatible oneAPI implementations.

- Unified programming model → enables hardware acceleration using standard C++ (with python)
- We are going to use OneAPI to estimate the resource usage of our NN on a Agilex7 FPGA
- Dependencies have been “containerized” in Docker image (30Gb local installation)
- Installed on Google Virtual Machines (VMs) → We will run the notebooks from these machines
- Please follow the instructions carefully!

[\[Github link\]](#)



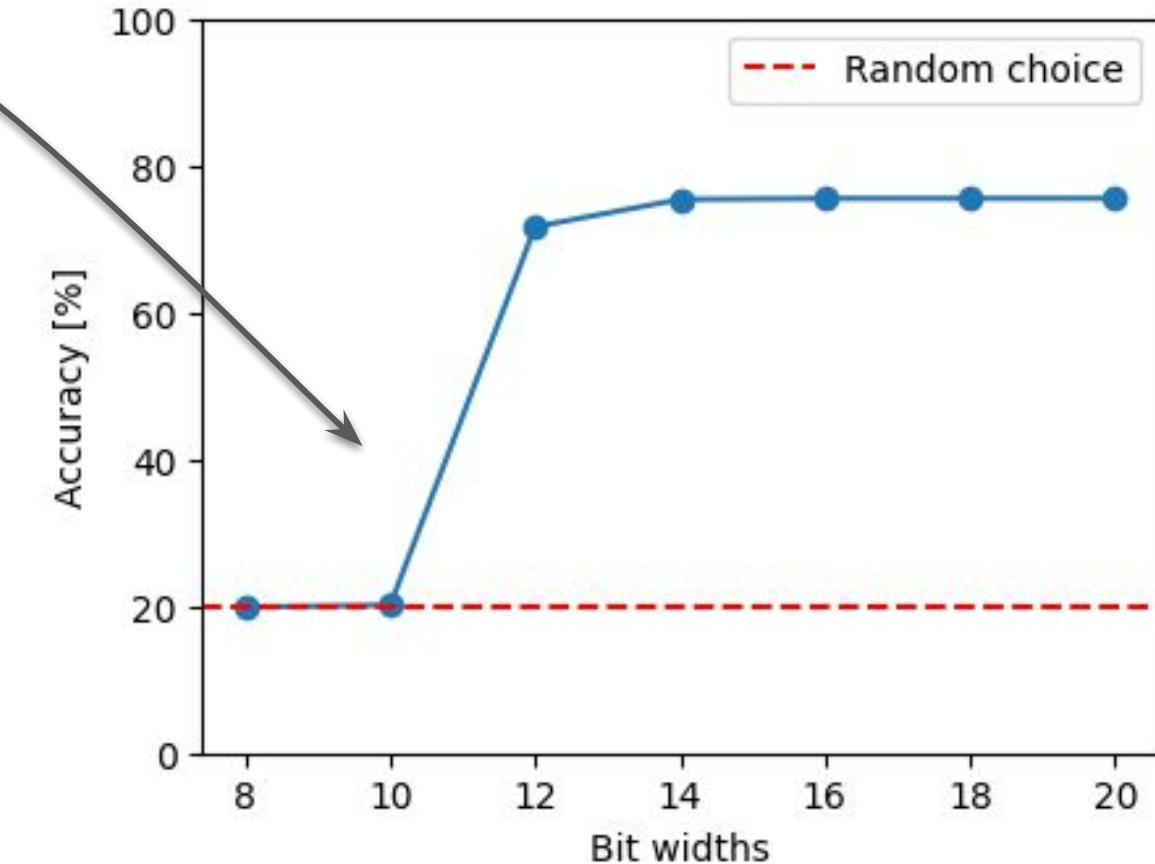
# Tutorial

## Quantization-aware training

(Part 4)

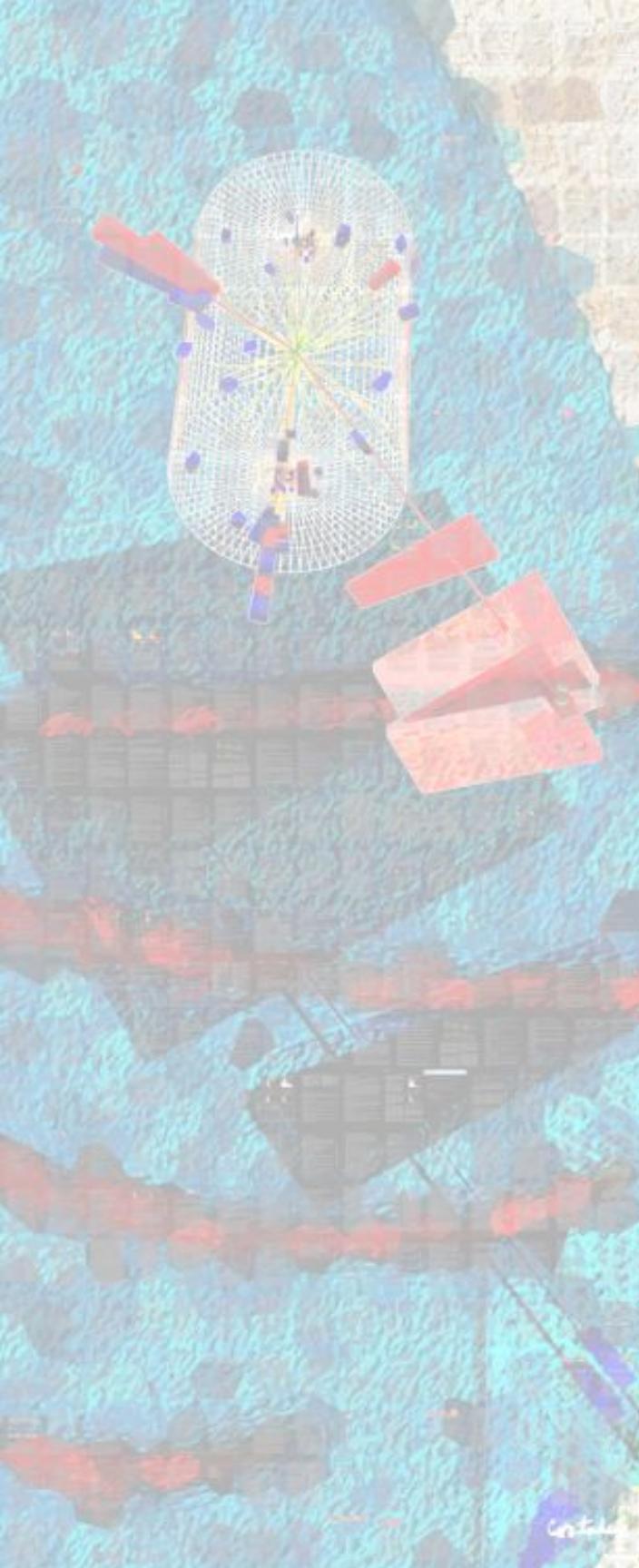
# Quantization-aware training

- In Part 3 we saw how the classification performance drops when using too few bits during inference
- QKeras: introduce knowledge of bit-widths during training
  - “Quantization-aware training”
  - Emulates fixed-point precision during forward-pass
  - Easy-to use “drop-in” replacements for Keras layers e.g.
    - Dense → QDense
    - Conv2D → QConv2D
    - Use quantizers to specify how many bits to use where



- Can achieve very good performance with very few bits and drastically reduce the resource usage
- **Part 4:** implement QA training, evaluate performance/resource-usage and compare to results of Part 3
  - Study performance/resource-usage for different bit-widths in training
- N.B. hls4ml allows different data types everywhere e.g. in different layers (see tutorial) → Remember this for the “competition”





# Tutorial

## *Pruning*

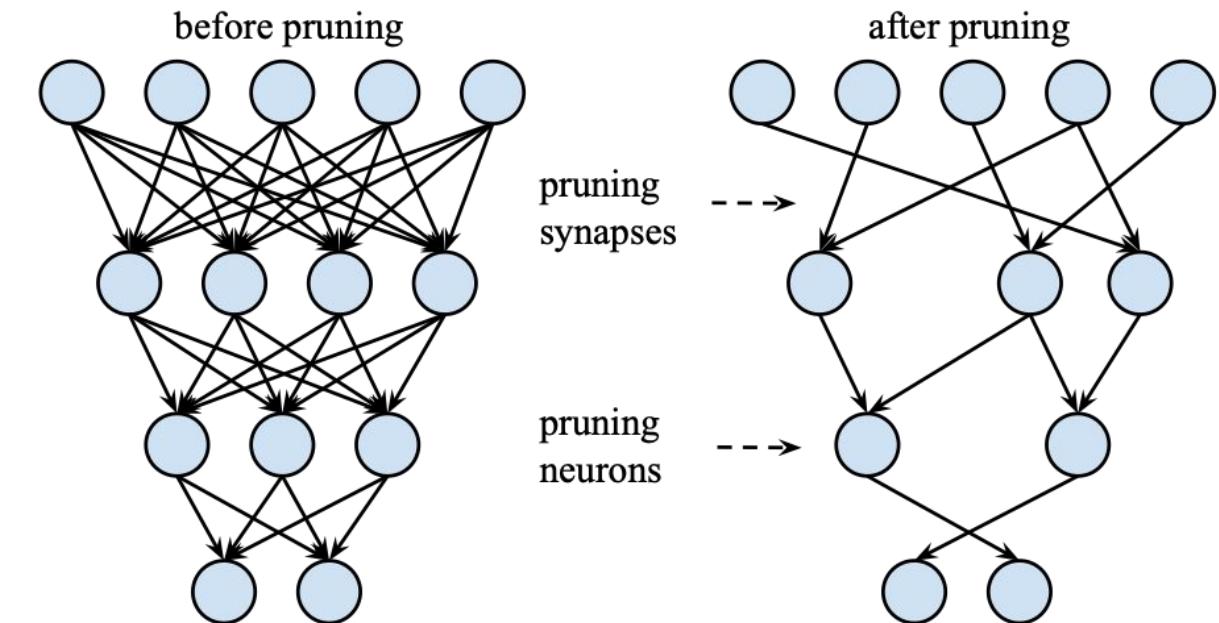
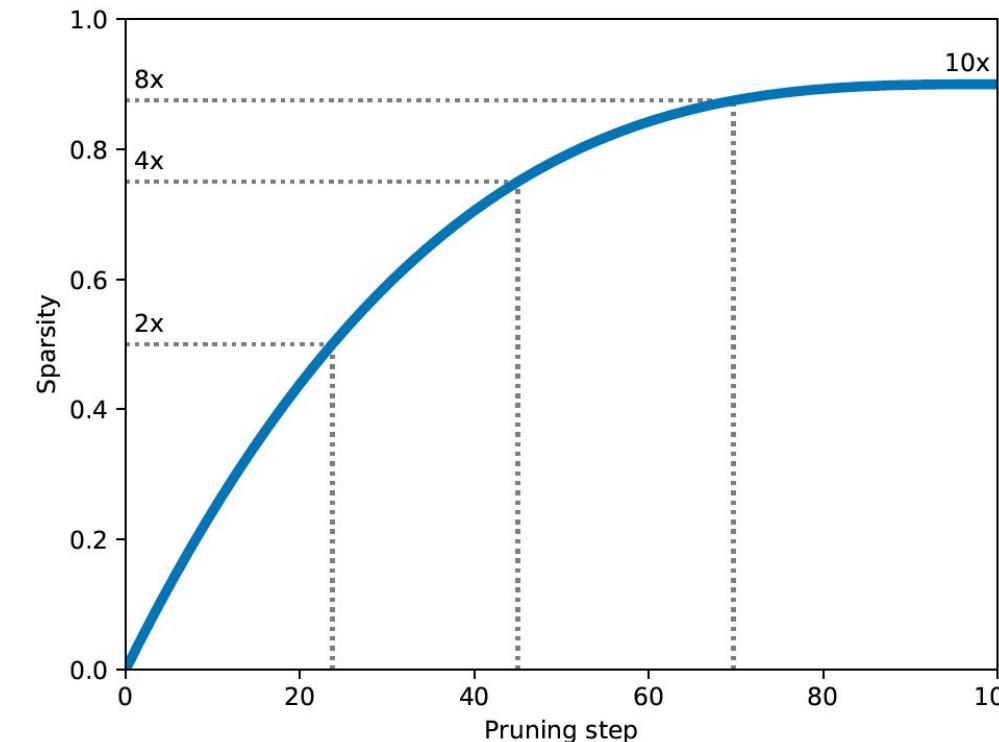
*(Part 5)*

# Network compression

- Neural network compression is a widespread tool to reduce size, energy consumption, and overtraining for deep NNs
- Several techniques have been studied:
  - **Parameter pruning**: selective removal of weights based on a particular ranking [[arxiv.1510.00149](#), [arxiv.1712.01312](#)]
  - **Low-rank factorization**: using matrix/tensor decomposition to estimate informative parameters [[arxiv.1405.3866](#)]
  - **Compact convolutional filters**: special structural convolution filters to save parameters [[arxiv.1602.07576](#)]
  - **Knowledge distillation**: training a compact network with distilled knowledge from larger network [[doi:10.1145/1150402.1150464](#)]

# Network compression

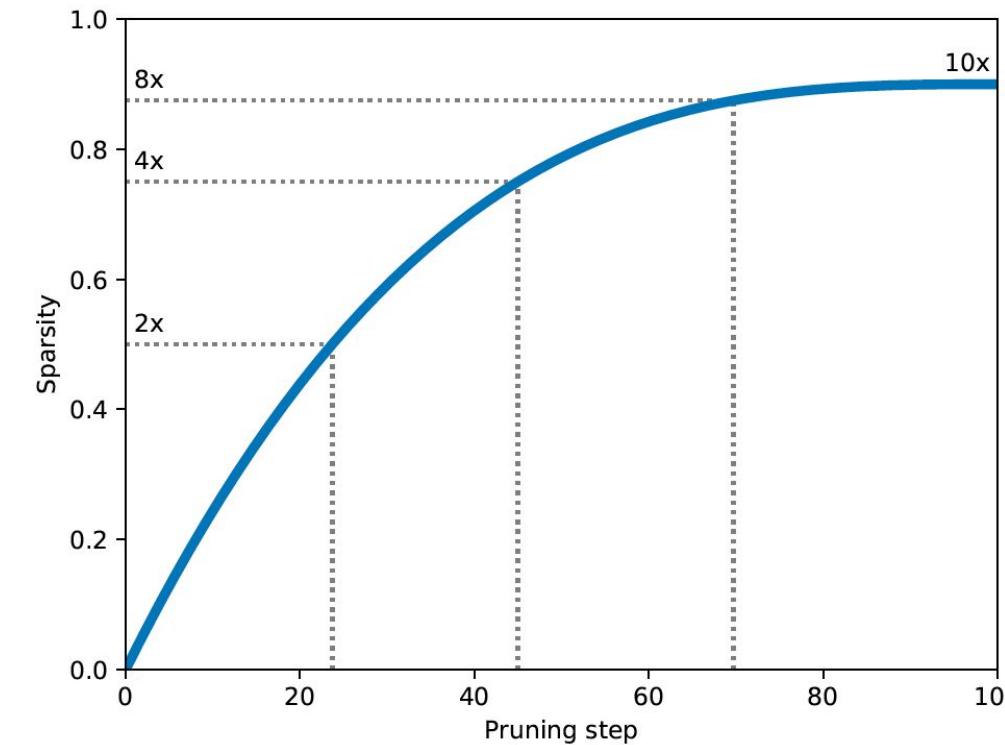
- Neural network compression is a widespread tool to reduce size, energy consumption, and overtraining for deep NNs
- Several techniques have been studied:
  - **Parameter pruning:** selective removal of weights based on a particular ranking [[arxiv.1510.00149](https://arxiv.org/abs/1510.00149), [arxiv.1712.01312](https://arxiv.org/abs/1712.01312)]
- In this tutorial we will use [tensorflow's model sparsity toolkit](#) (but you can use other methods)



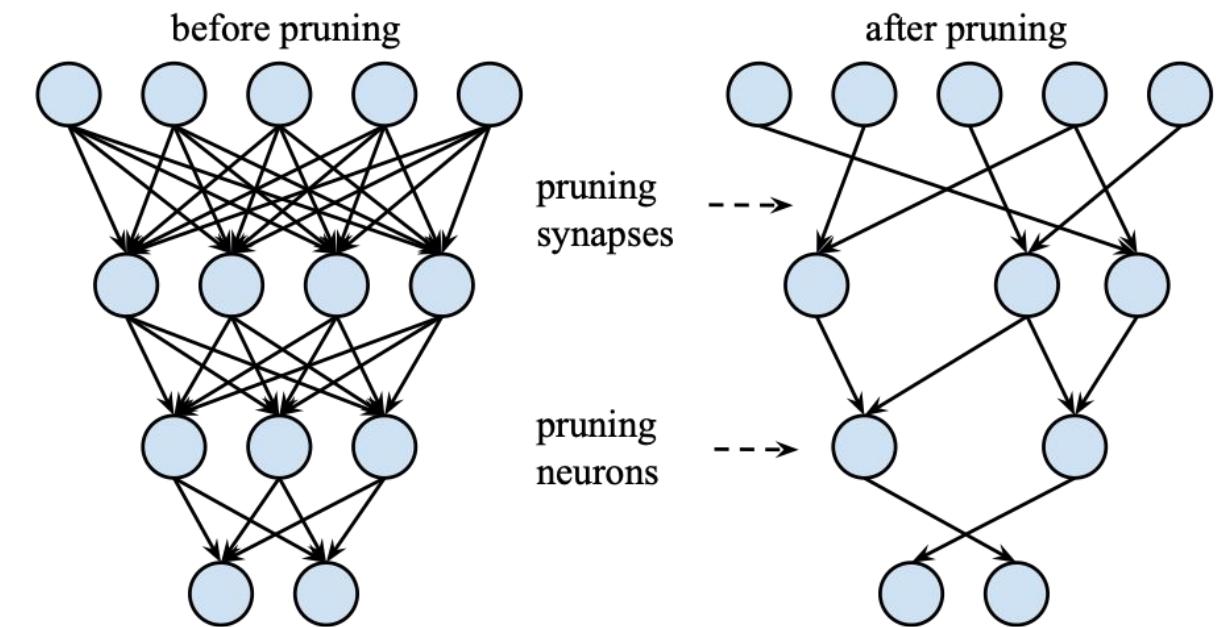
Iteratively remove low magnitude weights (starting at zero sparsity), smoothly increasing until target sparsity is reached during training

# Network compression

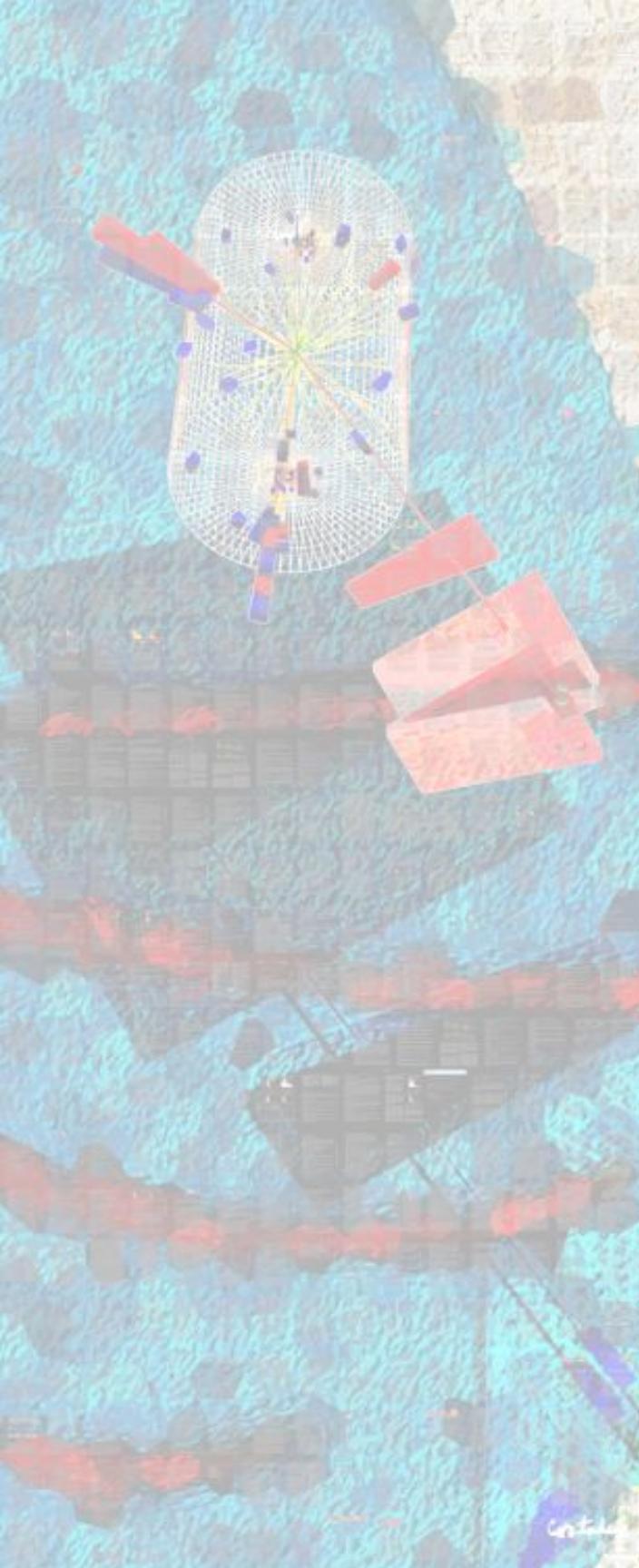
- Neural network compression is a widespread tool to reduce size, energy consumption, and overtraining for deep NNs
- Several techniques have been studied:
  - **Parameter pruning:** selective removal of weights based on a particular ranking [[arxiv.1510.00149](https://arxiv.org/abs/1510.00149), [arxiv.1712.01312](https://arxiv.org/abs/1712.01312)]
- In this tutorial we will use [tensorflow's model sparsity toolkit](#) (but you can use other methods)



Iteratively remove low magnitude weights (starting at zero sparsity), smoothly increasing until target sparsity is reached during training



**Part 5:** evaluate impact of pruning on NN performance and resource usage for different sparsities



## Competition

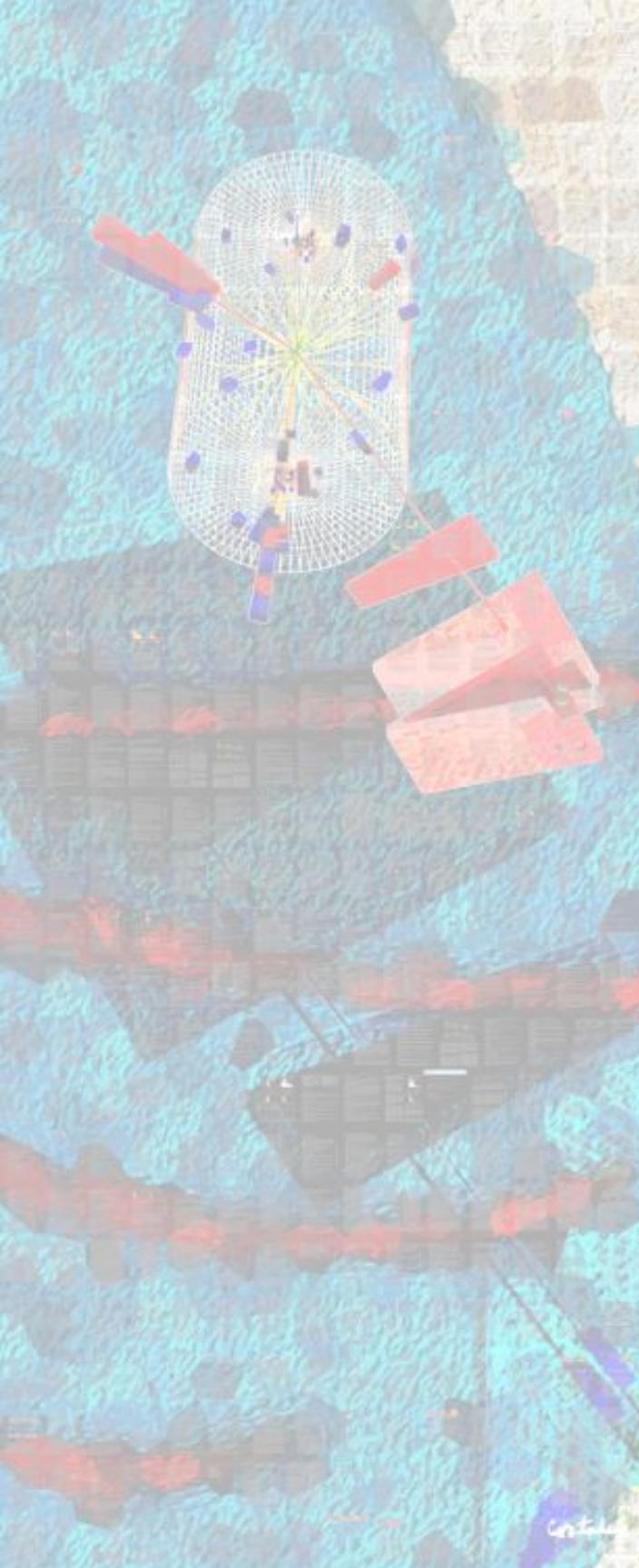


# Course competition

Task: how low can you go in resource usage whilst maintaining a global classification accuracy of greater than 74%

- Metric: sum of LUT [%] and FF [%] usage
- Submission (via Google Forms):
  - Written summary (bullet points) of approach
  - Resource usage and accuracy outputs (print statements)
  - Code
  - Graphs (if possible)
- Tip: use a combination of quantization and compression
- **This is part of the assessment but it's an extension (used to distinguish top performing students)**





## Closing remarks

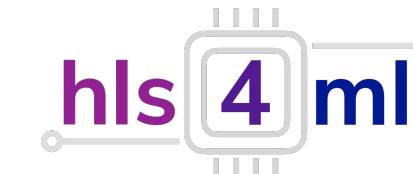
# Course summary

- **Understand** how to accelerate AI inference using FPGAs, and **apply** this to a jet classification algorithm at the LHC
- **Analyze** and **evaluate** the performance and efficiency of the FPGA algorithm, implementing techniques such as quantization-aware training and pruning (compression)
- **Create** an efficient NN design based on what you have learnt throughout the course

# Course summary

- **Understand** how to accelerate AI inference using FPGAs, and **apply** this to a jet classification algorithm at the LHC
- **Analyze** and **evaluate** the performance and efficiency of the FPGA algorithm, implementing techniques such as quantization-aware training and pruning (compression)
- **Create** an efficient NN design based on what you have learnt throughout the course

Hopefully you have enjoyed the hands-on experience with



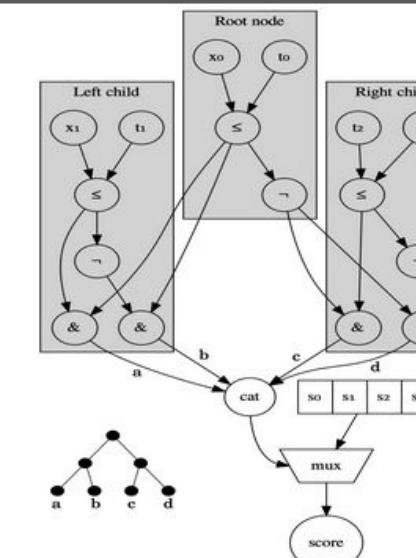
- The Google VMs will be closed down after the assignment deadline due to running costs
- If you are interested in continuing this kind of work, I have added Docker installation instructions to the [Github](#) (30Gb)
- You may also want to try some other features of Intel OneAPI like latency estimation
- And even accelerate your own project

AI acceleration is an active and fast moving field...

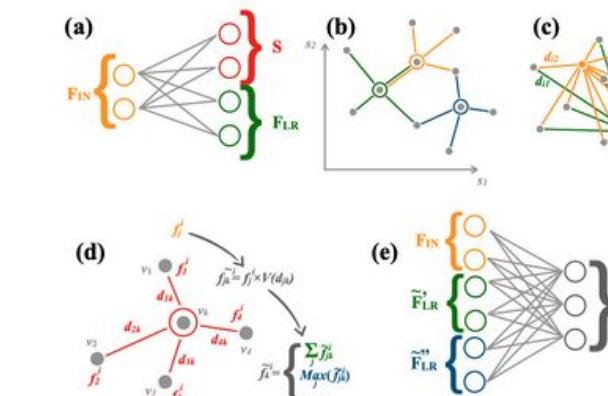
# Developments in & FastML community

-  is a very active community, with many developments driven by constraints of LHC triggers
- FastML international conference: focus is accelerated AI inference for fundamental science

- Binary & Ternary neural networks: [\[2020 Mach. Learn.: Sci. Technol\]](#)
  - Compressed weights for low resource inference
- Boosted Decision Trees: [\[JINST 15 P05026 \(2020\)\]](#)
  - Low latency for Decision Tree ensembles
- GarNet / GravNet: [\[arXiv: 2008.03601\]](#)
  - Distance weighted graph neural networks suitable for sparse/irregular point-cloud data
- Quantization aware training QKeras + support in hls4ml: [\[arXiv: 2006.10159\]](#)
- Convolutional neural networks  
[Mach. Learn.: Sci. Technol. 2 045015 \(2021\)](#)



The diagram illustrates a boosted decision tree structure. It starts with a 'Root node' containing a comparison operator ( $\leq$ ). This node has two children: 'Left child' and 'Right child'. Each child node contains a feature ( $x_1$ ,  $x_2$ ) and a threshold ( $t_1$ ,  $t_2$ ). The outputs of these nodes are combined via logical operators ( $\neg$ ,  $\wedge$ ,  $\vee$ ) to produce a final score. Below the tree, a small binary tree structure is shown with labels a, b, c, d corresponding to the nodes in the main tree.

The diagram shows the GarNet/GravNet architecture. (a) A point cloud is processed by a feature extraction network  $F_{IN}$  to produce features  $s$ . (b) These features are used to calculate distances  $d_{ij}$  between points. (c) The network  $F_{LR}$  processes these distances to produce learned representations. (d) A detailed view of a node processing multiple inputs  $v_i, v_j, v_k$  with weights  $f_i^j, f_i^k$  and distance  $d_{ik}$  to produce a weighted sum. (e) The final output  $F_{OUT}$  is produced by another feature extraction network  $\tilde{F}_{LR}$ .



FastML 24 (Purdue)

# The bigger picture

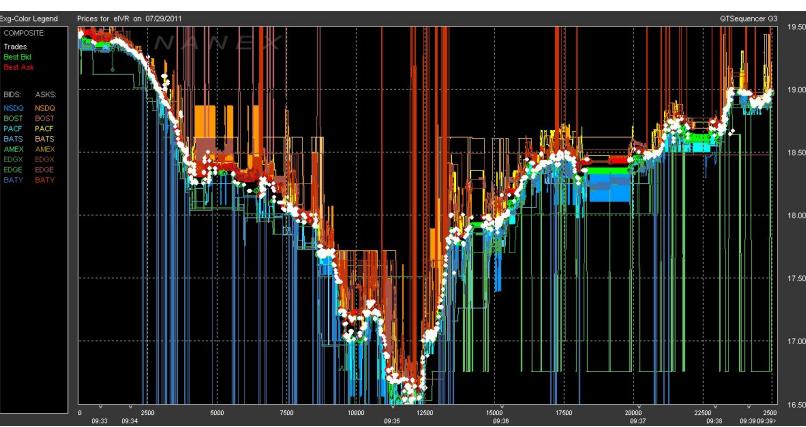
Many use-cases outside of particle physics...



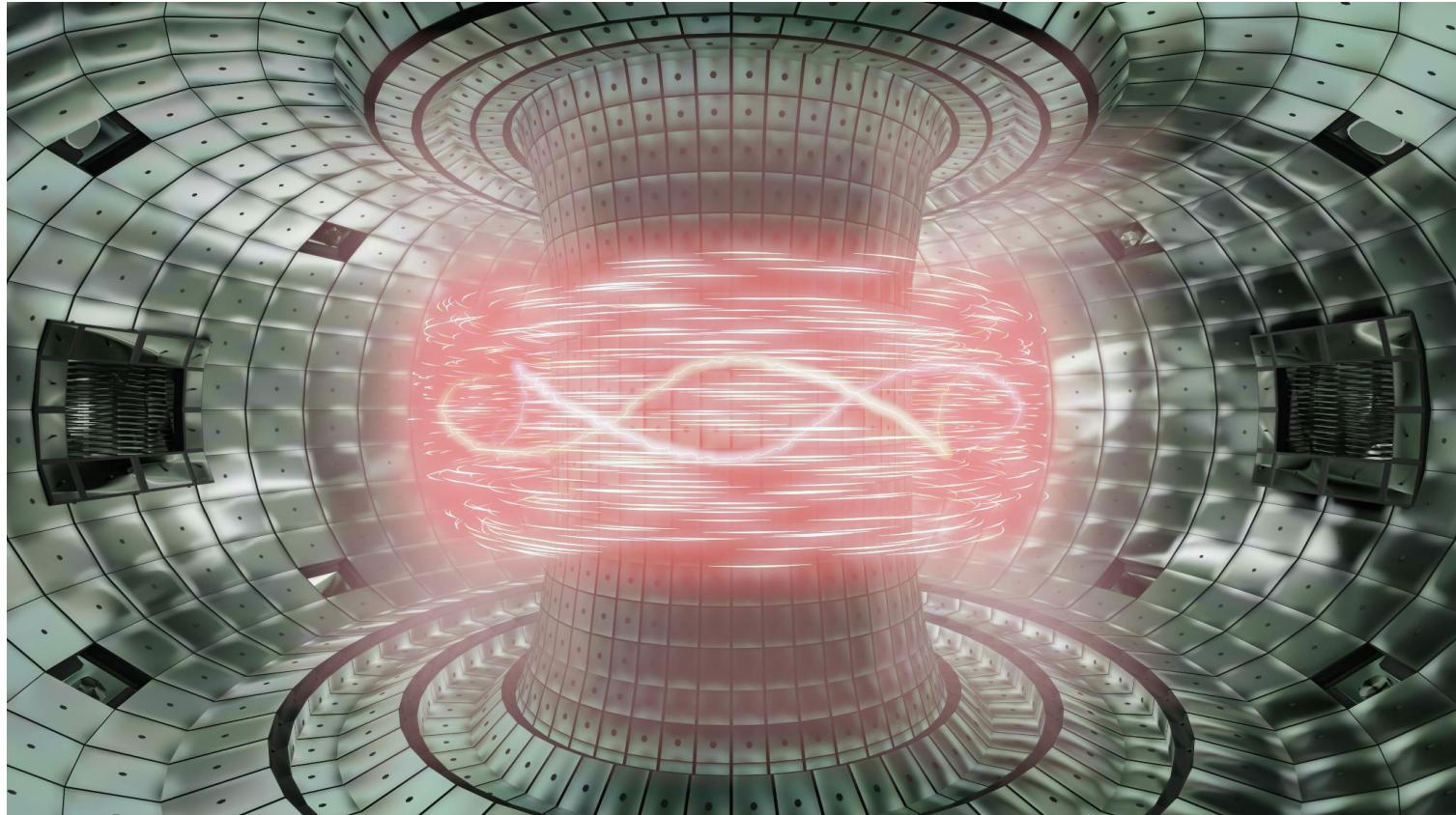
[FPGAs in medical imaging](#)



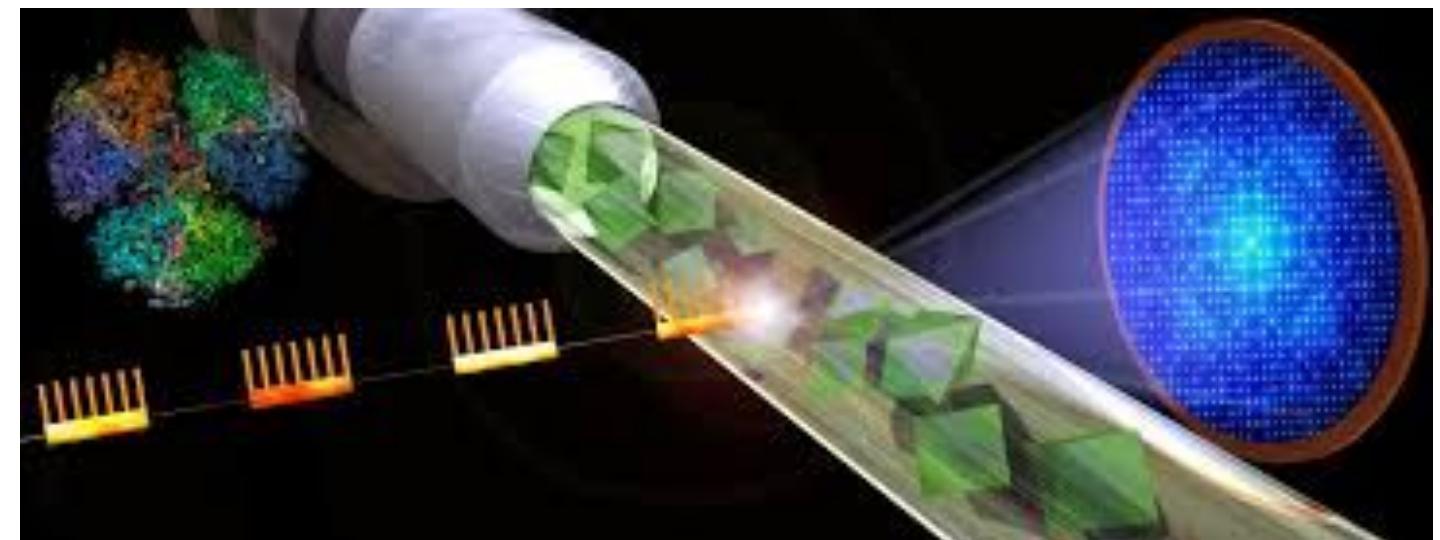
[Autonomous vehicles](#)



[High frequency trading](#)



[Nuclear fusion control](#)



[X-ray crystallography @ XFEL](#)

# Course feedback

