SMARTHEP REAL-TIME ANALYSIS FOR SCIENCE AND INDUSTRY

NexTGen EDGE ML SCHOOL

Fast Inference of Decision Forests on FPGAs with **conifer** - a tutorial

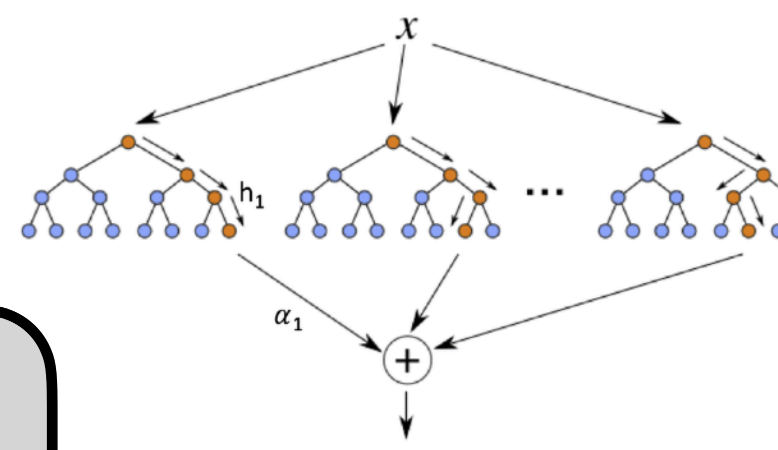
26/9/24

Sioni Summers

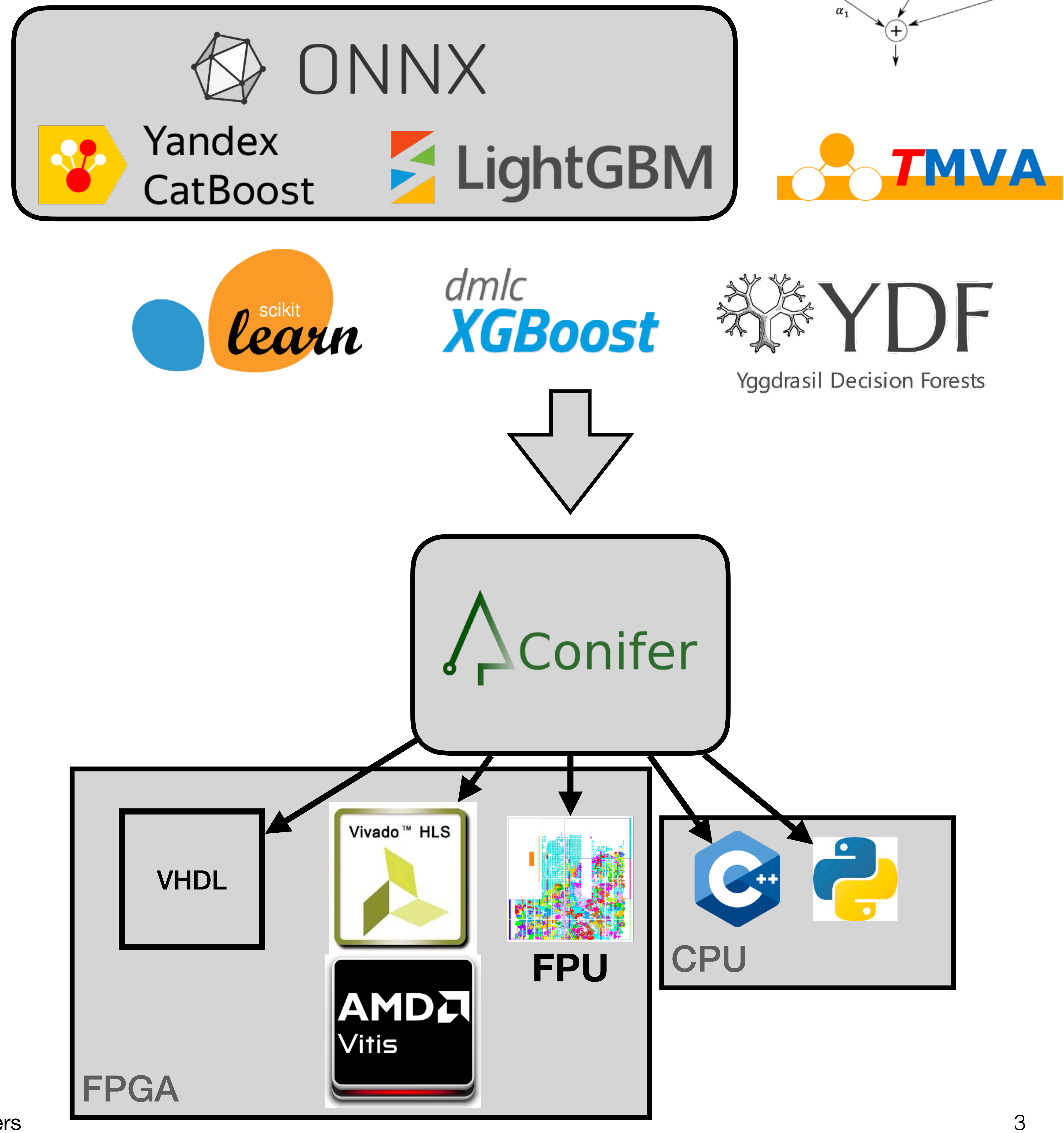
Introduction

- Today's tutorial focusses on using conifer to make fast inference:
  - Targeting low latency for custom hardware (trigger / custom flow)
  - Targeting high throughput for edge devices (accelerator flow)
- Note: there is a conifer / BDT section of the hls4ml tutorial, but this is more up to date!
  - hls4ml tutorial conifer section will be updated with Vitis HLS soon
- The notebooks will be shown as a demo only
  - They are available here: https://github.com/thesps/conifer-tutorial/tree/smarthep
- Refer to this talk at FPGA Developers Forum for a look "under the canopy"
- Refer to this tutorial for longer exercises and introduction to HLS

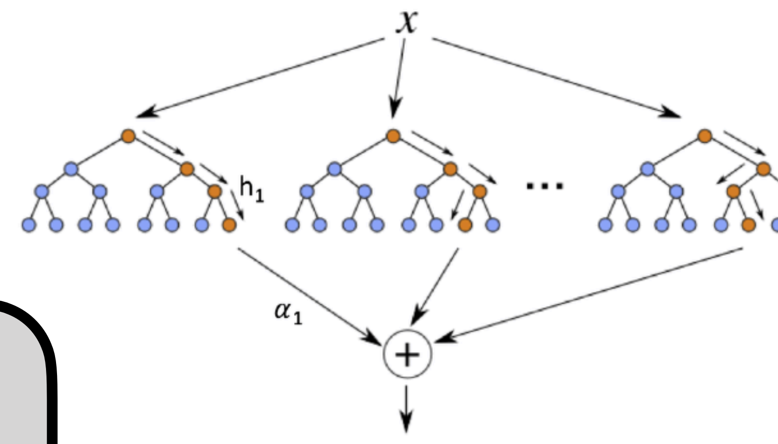
Conifer for Decision Forests



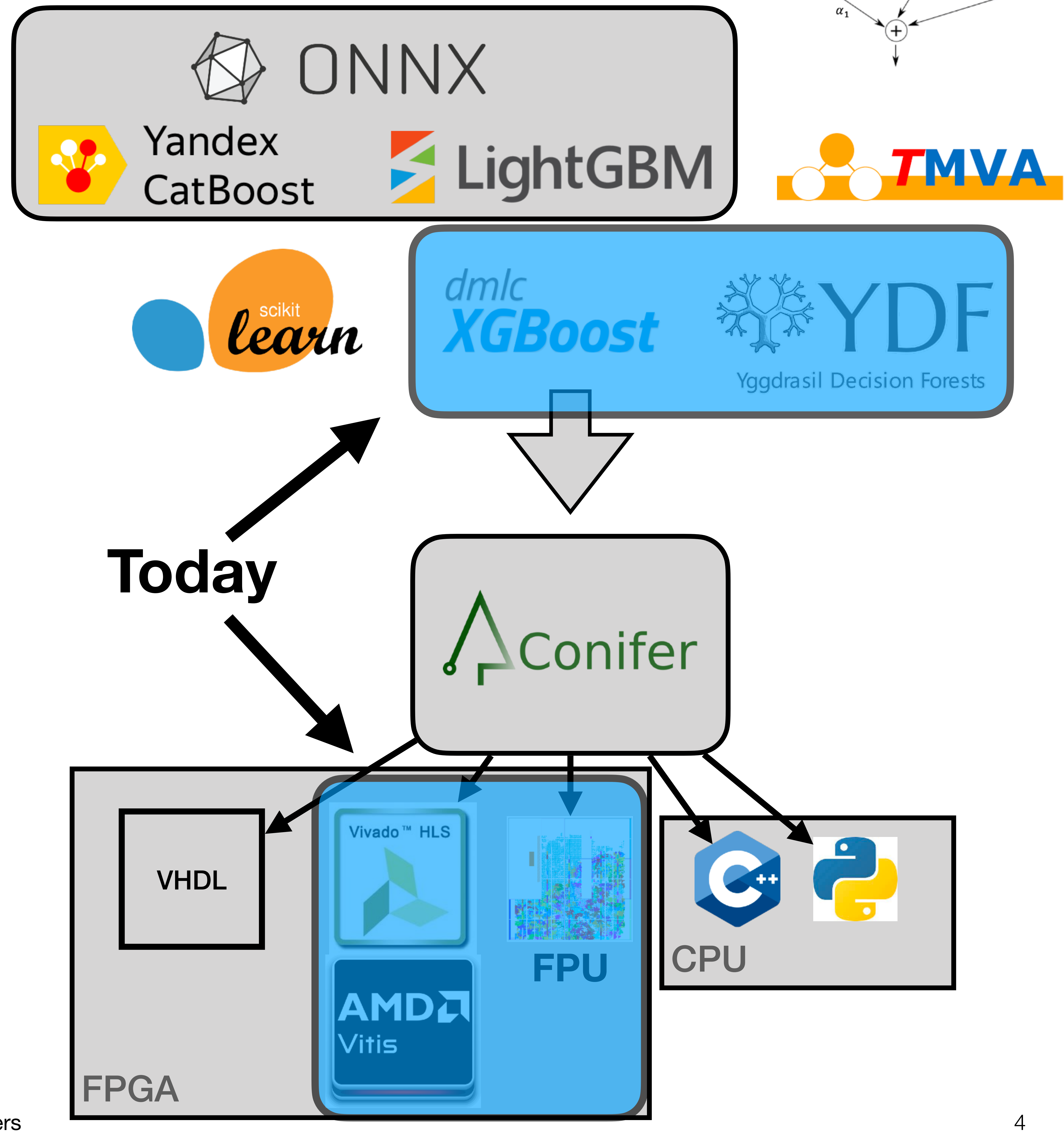
- Decision Forests are still relevant for edge / constrained ML:
  - Fast, lightweight, robust (arXiv:2207.08815, IML keynote)
- **conifer** is a tool to map DFs onto FPGA firmware
  - On Python Package Index: `pip install conifer`
- **conifer** reads from popular DF training tools and writes FPGA projects
  - Implemented with high parallelism for low latency and high throughput
  - Classification, Regression, ✨Anomaly Detection✨
  - Backends: HLS, VHDL, ✨Forest Processing Unit✨, C++, Python
- A Decision Tree *splits* on data variables until reaching a *leaf*
  - Leaves associate a *score* corresponding to prediction probability
- A Decision Forest is an ensemble of Decision Trees
  - Randomisation of each DT as a form of regularisation
  - Ensemble score is an aggregation over trees e.g. sum



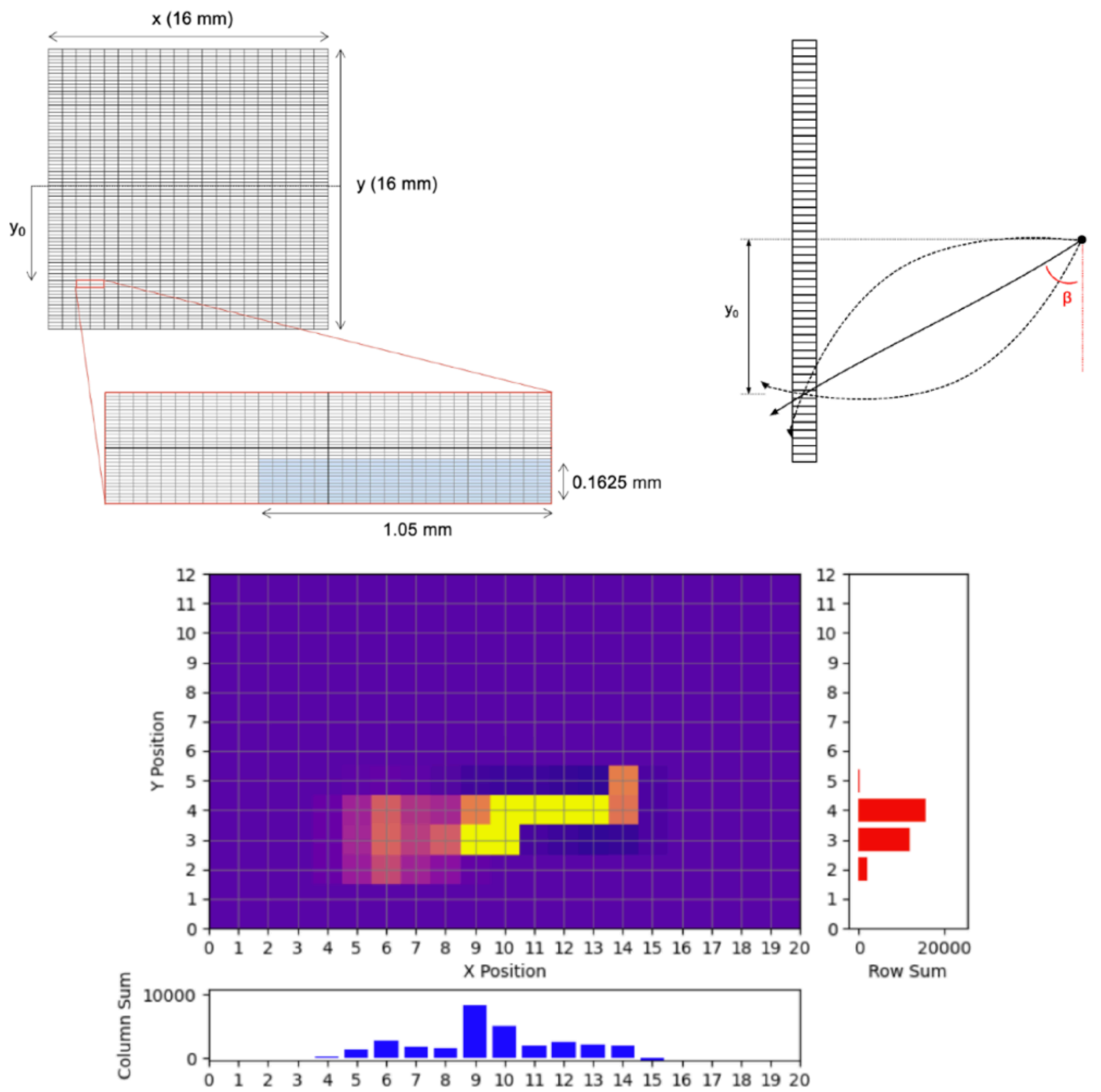
Conifer for Decision Forests

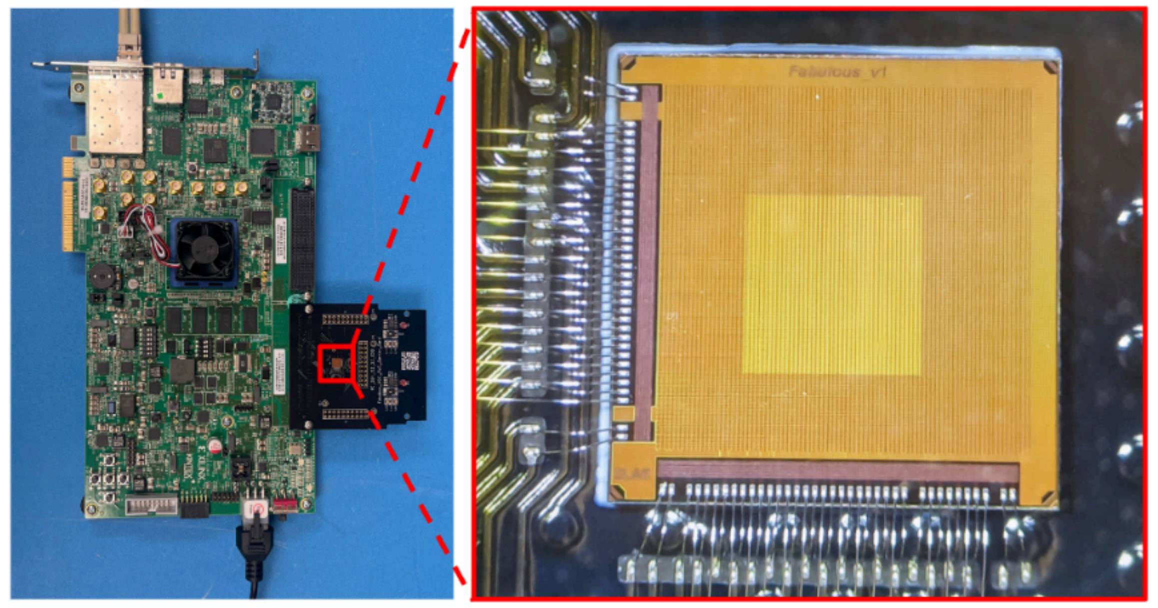


- Decision Forests are still relevant for edge / constrained ML:
  - Fast, lightweight, robust (arXiv:2207.08815, IML keynote)
- **conifer** is a tool to map DFs onto FPGA firmware
  - On Python Package Index: `pip install conifer`
- **conifer** reads from popular DF training tools and writes FPGA projects
  - Implemented with high parallelism for low latency and high throughput
  - Classification, Regression, ✨Anomaly Detection✨
  - Backends: HLS, VHDL, ✨Forest Processing Unit✨, C++, Python
- A Decision Tree *splits* on data variables until reaching a *leaf*
  - Leaves associate a *score* corresponding to prediction probability
- A Decision Forest is an ensemble of Decision Trees
  - Randomisation of each DT as a form of regularisation
  - Ensemble score is an aggregation over trees e.g. sum

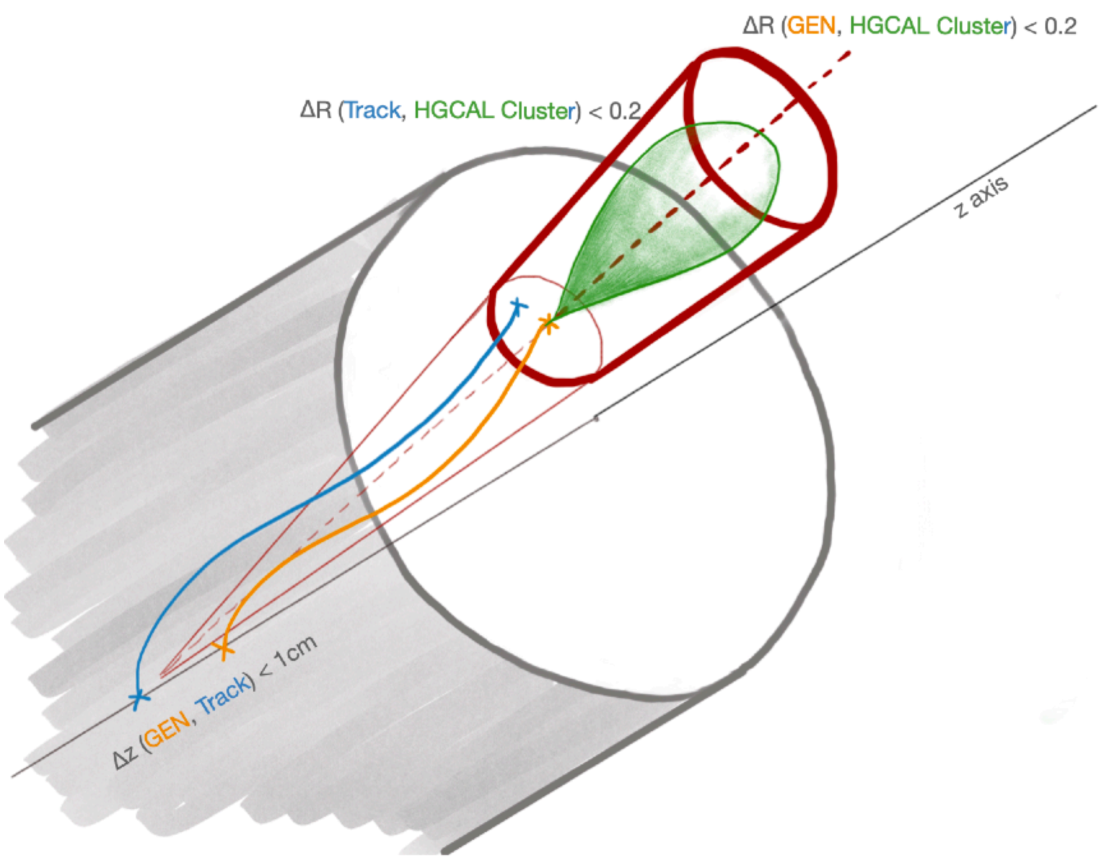


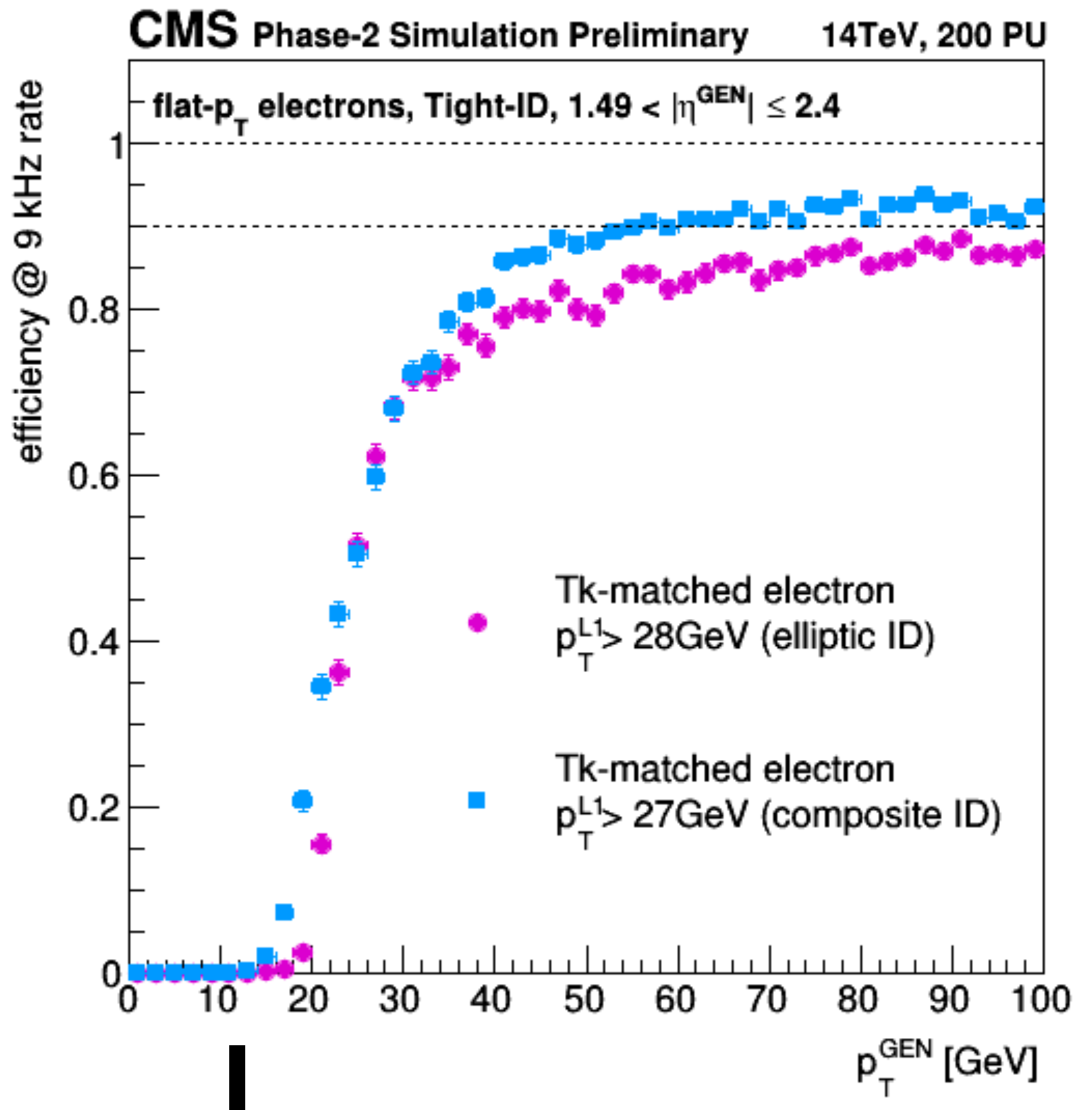
conifer applications

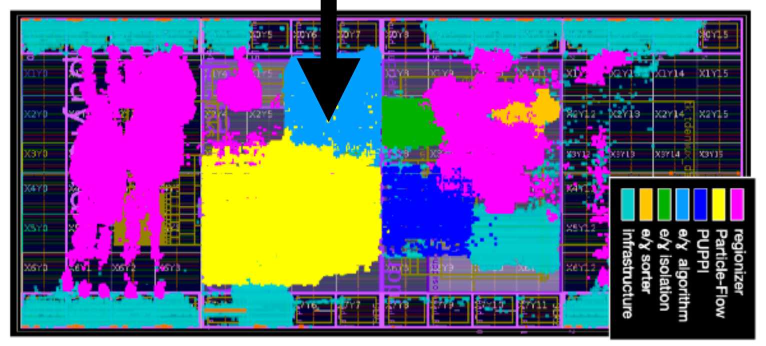




p_T filtering in an eFPGA in a tracking detector frontend (25 ns latency, 500 LUTs)







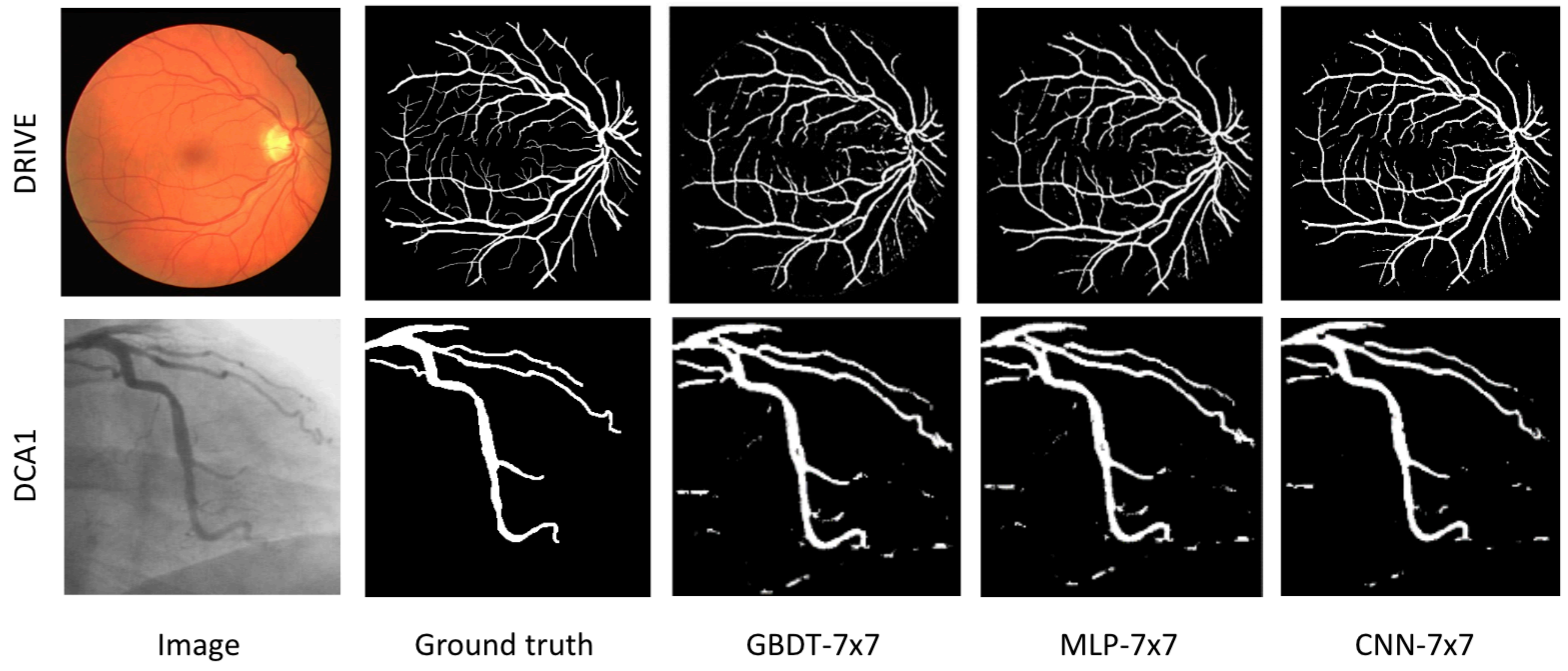
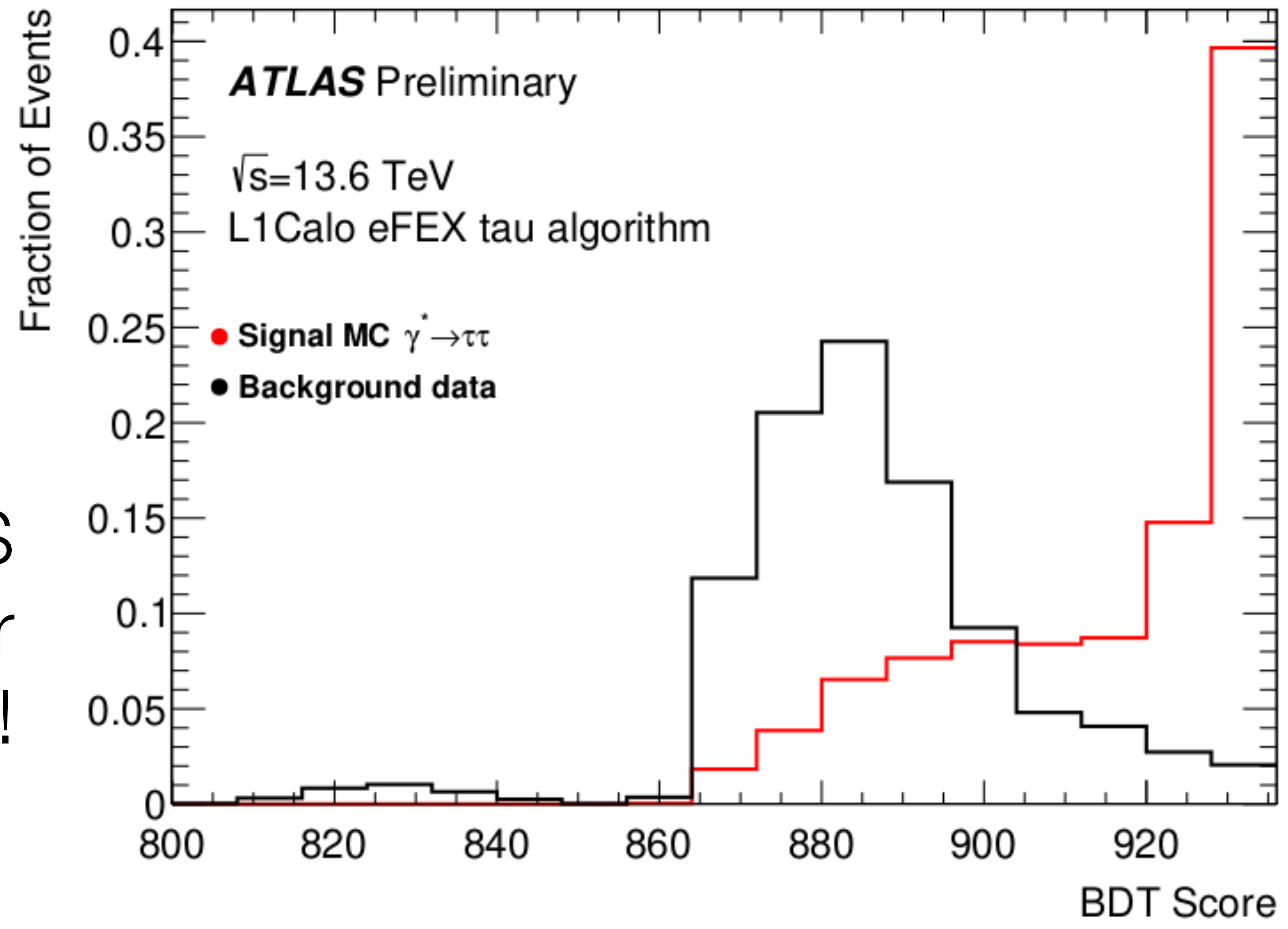


Image segmentation for blood vessels tracking in an embedded medical device (1779 FPS at 3.8 W)

Electron reconstruction in CMS Phase 2 Level 1 Trigger
(< 50 ns latency)

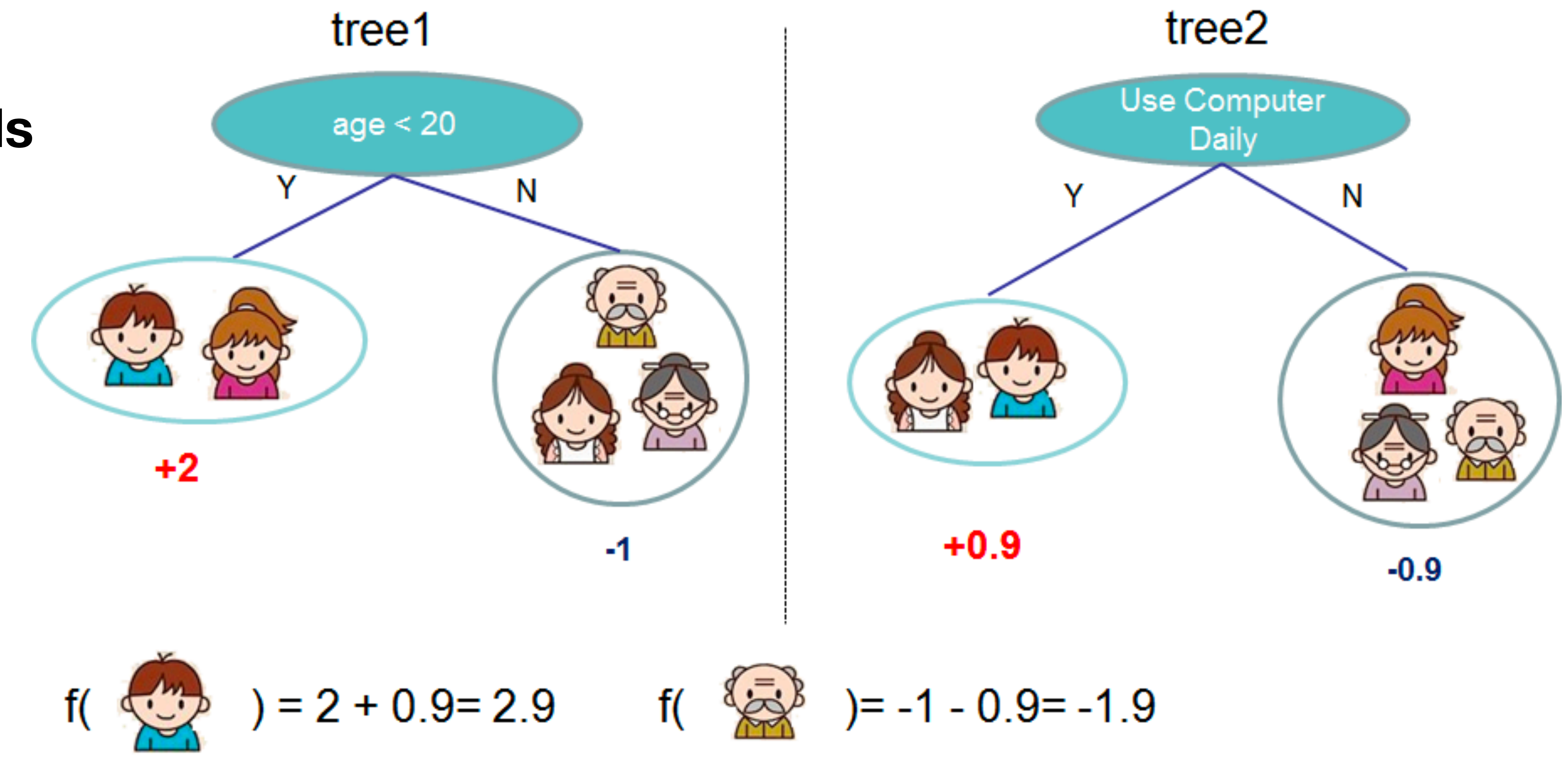
Tau reconstruction in ATLAS Run 3 calorimeter trigger
Online now!



Quick BDT Introduction

- Using XGBoost's Elements of Supervised Learning Introduction
- Train a **model** on training data to predict target variable y from features x
- A Boosted Decision Tree model is an ensemble of Decision Trees
- The splits of each Decision Tree are chosen based on the training objective function e.g. mean squared error
  - $L(\Theta) = \Sigma(y_i - \hat{y}_i)^2$ where y_i are our truth labels and $\hat{y}_i$ are the model predictions
- In an ensemble each learner (tree) is relatively weak, but the aggregation is a stronger prediction

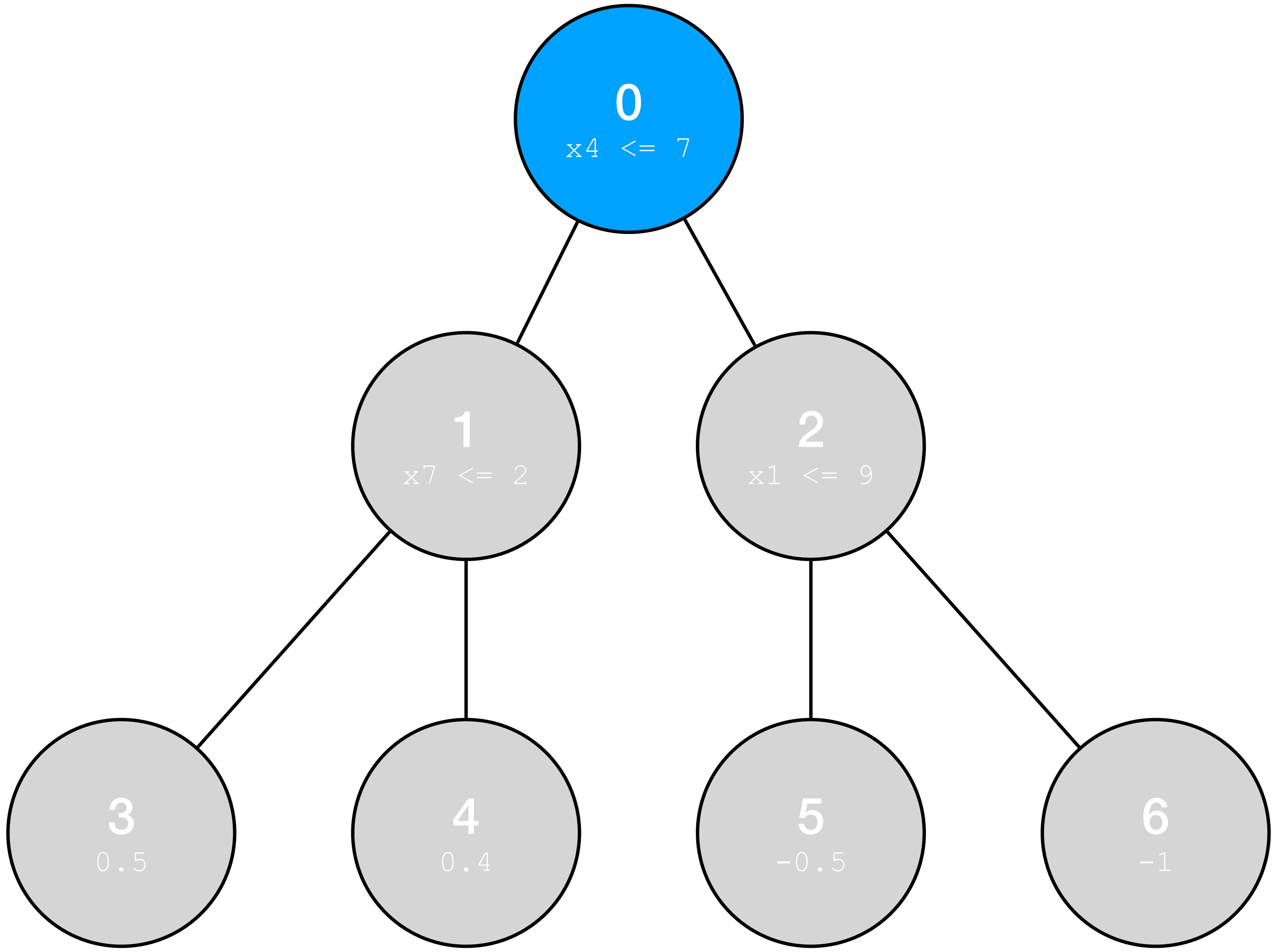
e.g. predict whether individuals will like a computer game



Decision Tree Inference

- Start at the root node - compare the selected feature with the threshold, go left or right depending on result

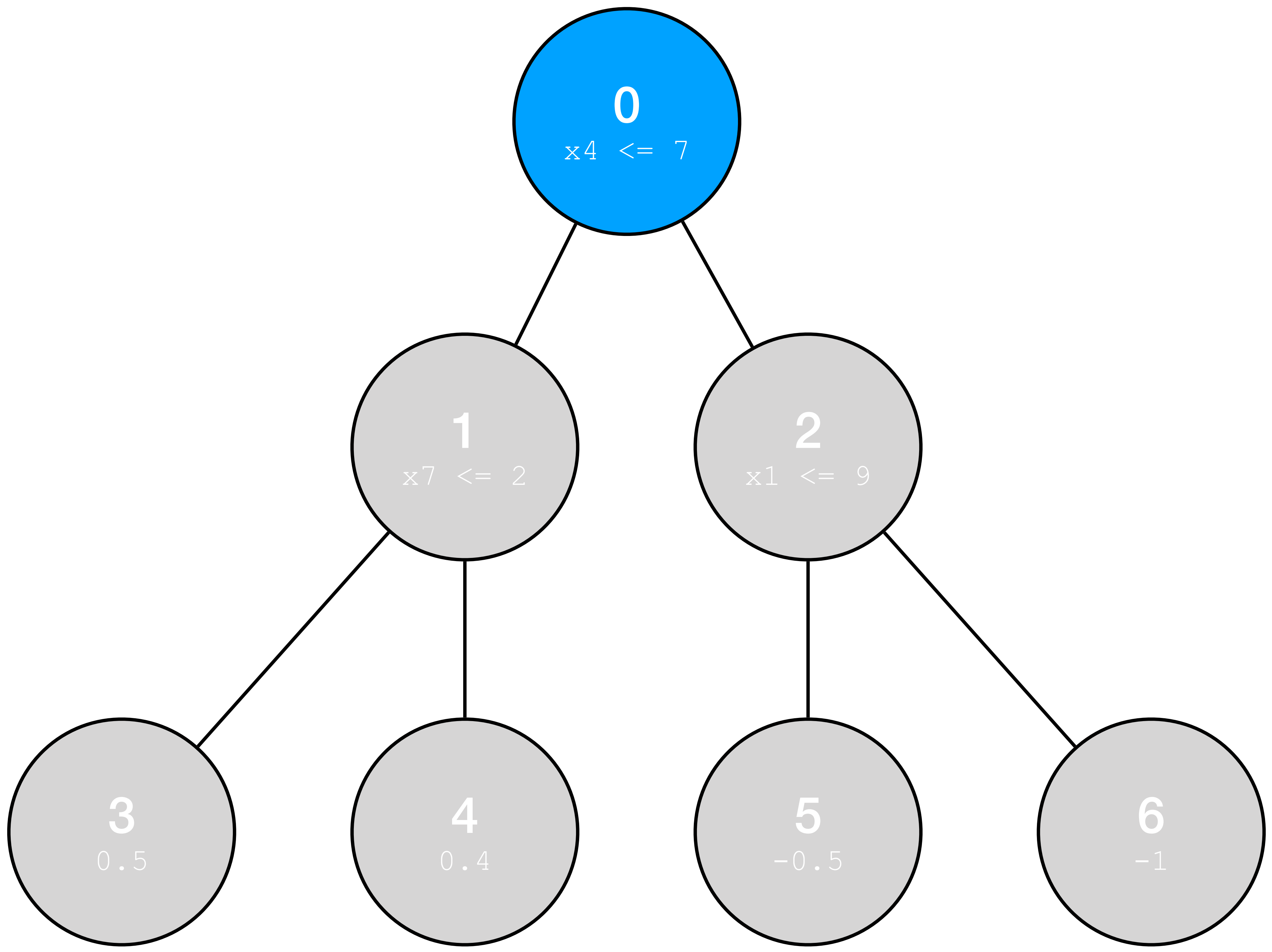
$$x = [\ x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7\ ]$$



Decision Tree Inference

- Start at the root node - compare the selected feature with the threshold, go left or right depending on result

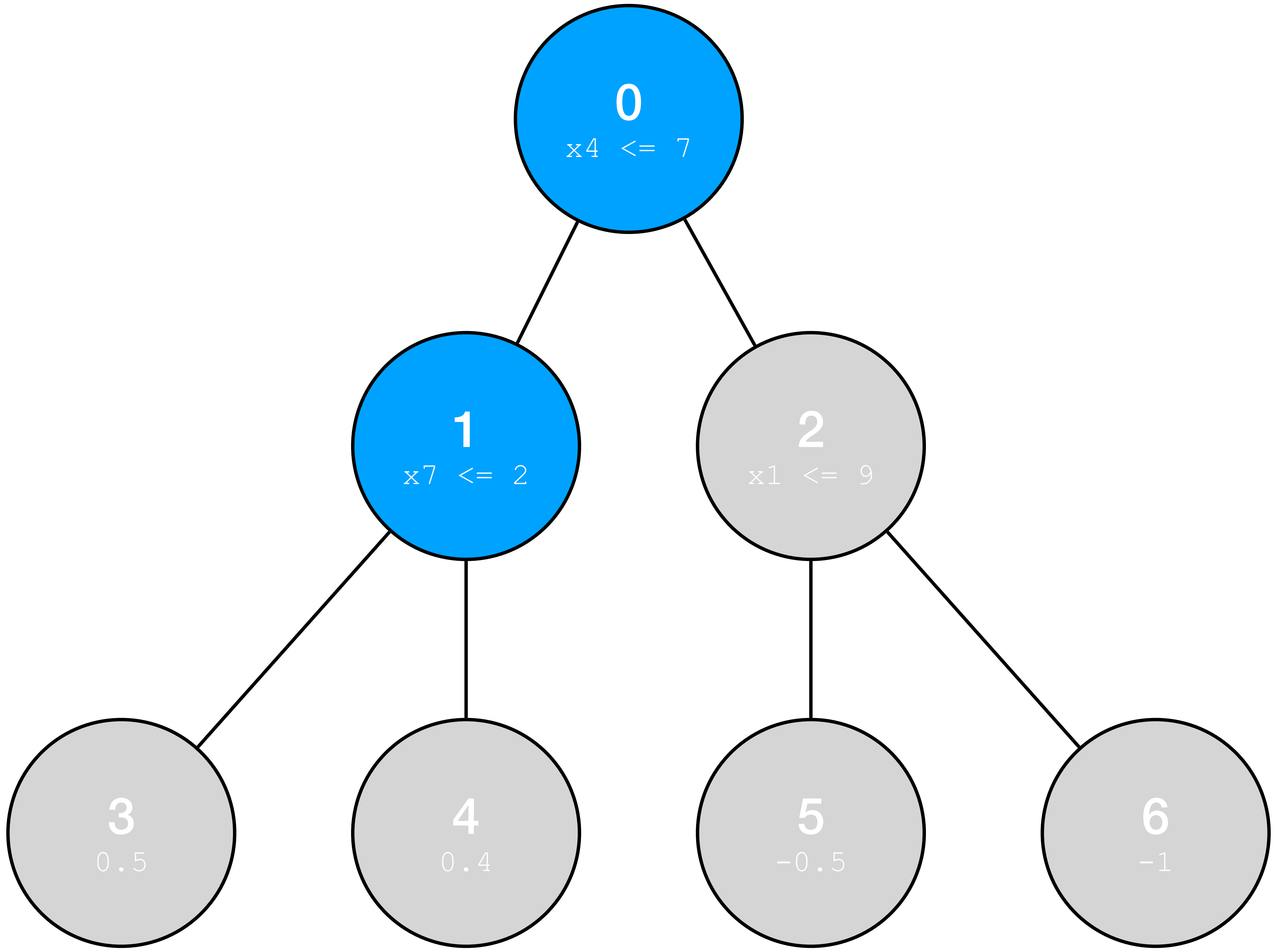
`x = [ -, 12, -, -, 3, -, -, 5 ]`



Decision Tree Inference

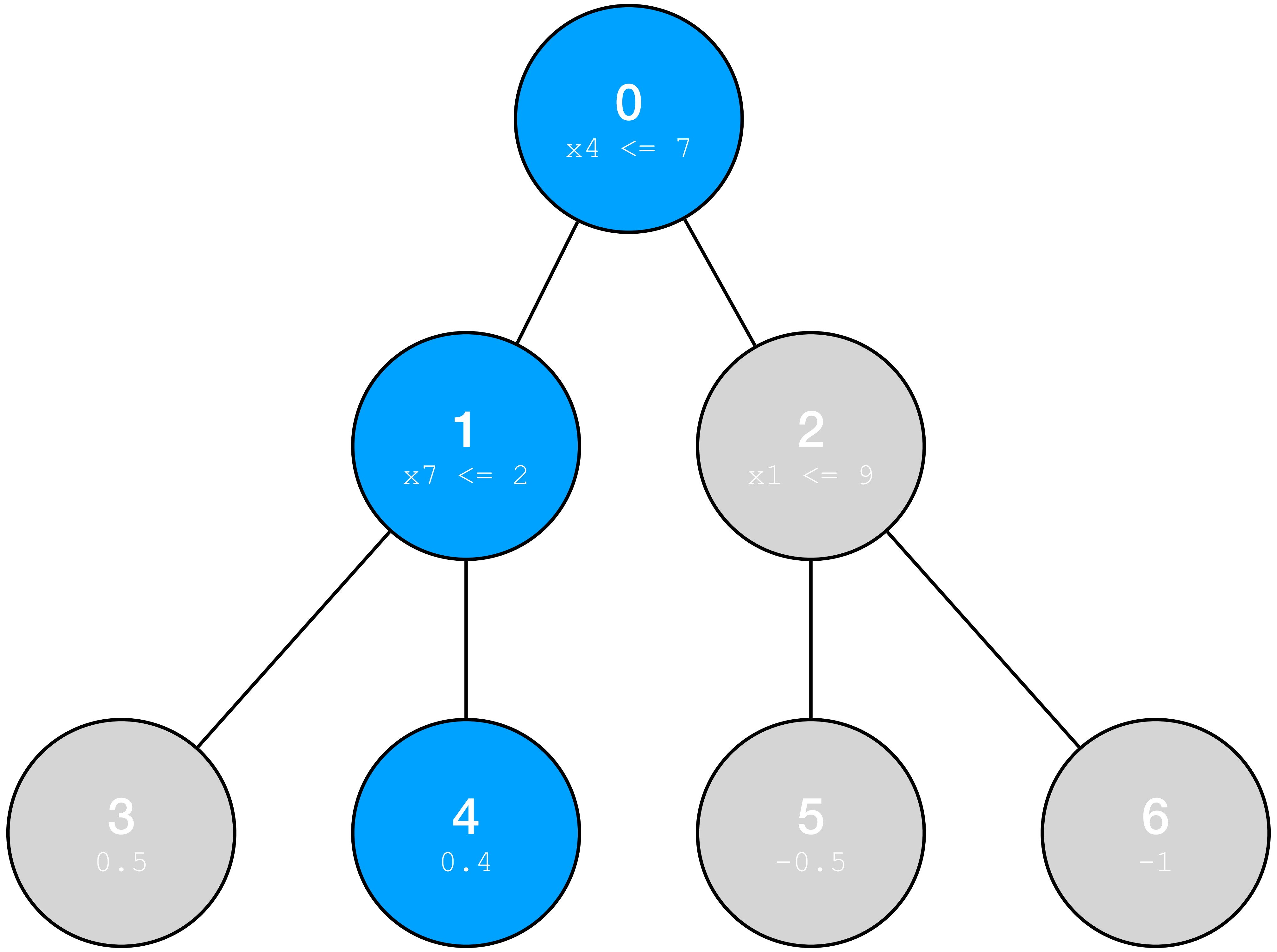
- Start at the root node - compare the selected feature with the threshold, go left or right depending on result
- Continue until reaching leaf - compare the selected feature with the threshold, go left or right depending on result

```
x = [ -, 12, -, -, 3, -, -, 5 ]
```



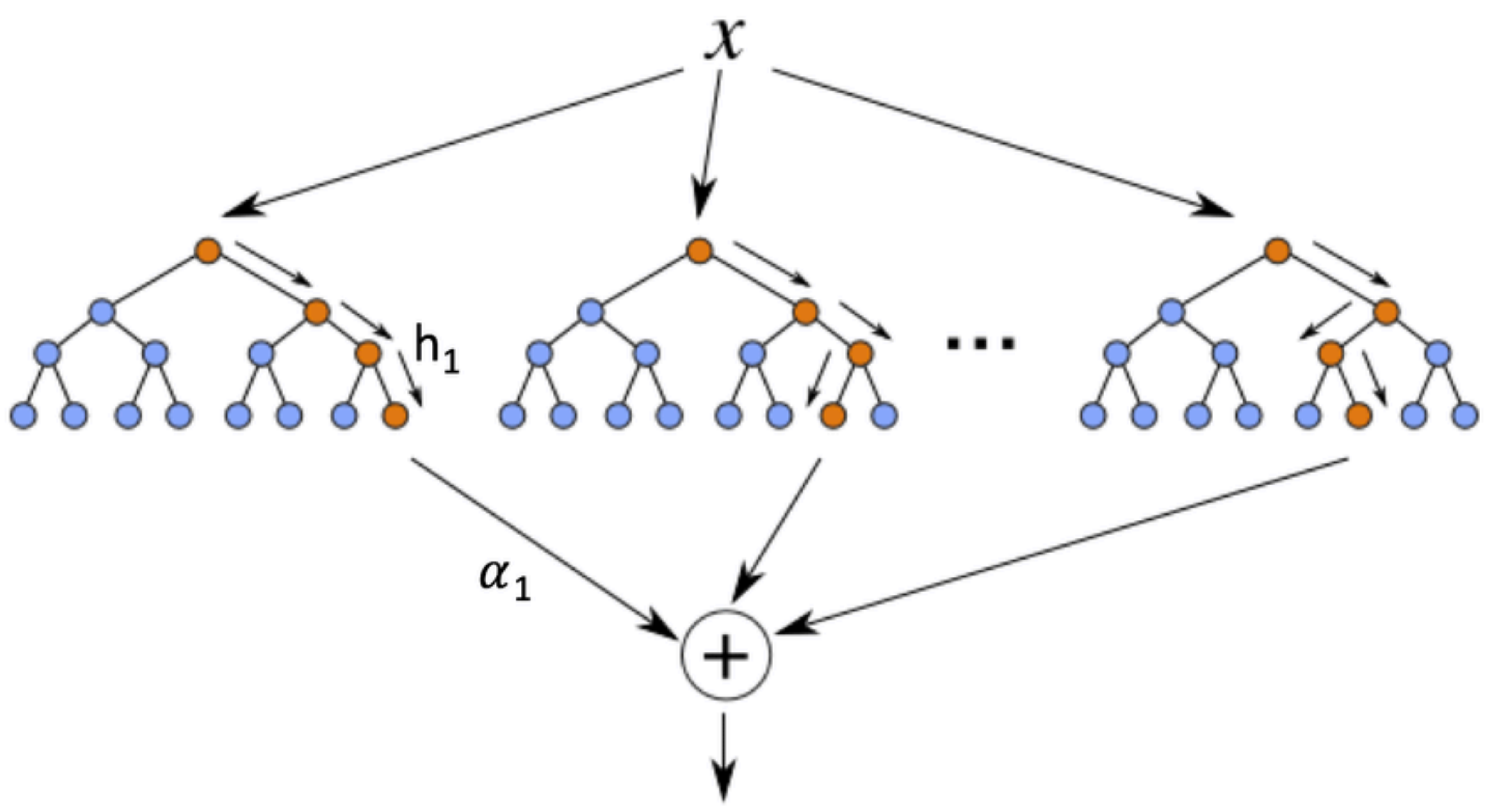
Decision Tree Inference

- Start at the root node - compare the selected feature with the threshold, go left or right depending on result
- Continue until reaching leaf - compare the selected feature with the threshold, go left or right depending on result
- The value of the terminal leaf is the tree prediction



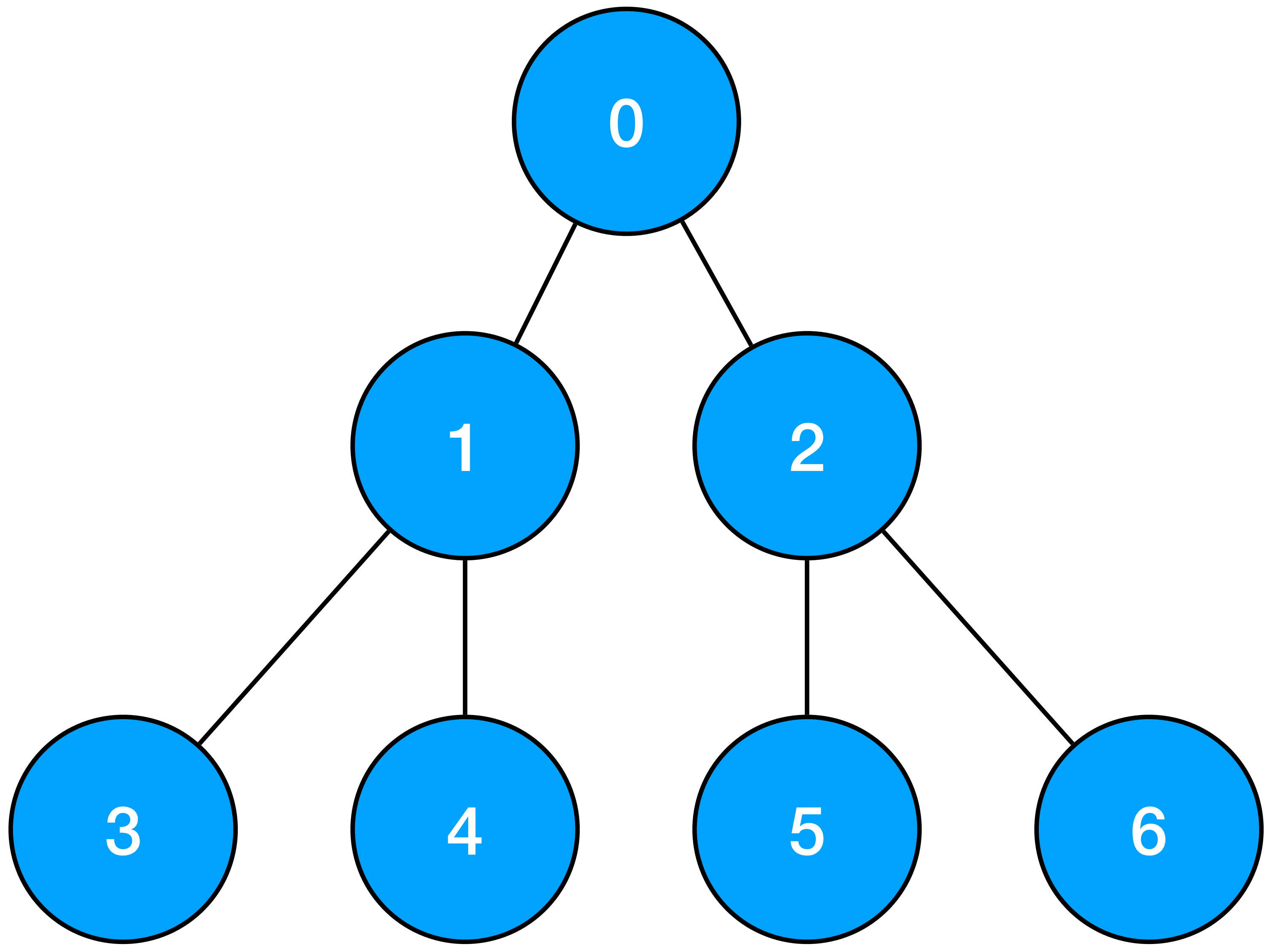
Decision Forest Inference

- Repeat the same procedure for every tree in the ensemble, sum up the tree scores for the BDT prediction
- Apply the inverse of the training loss function to obtain class probabilities



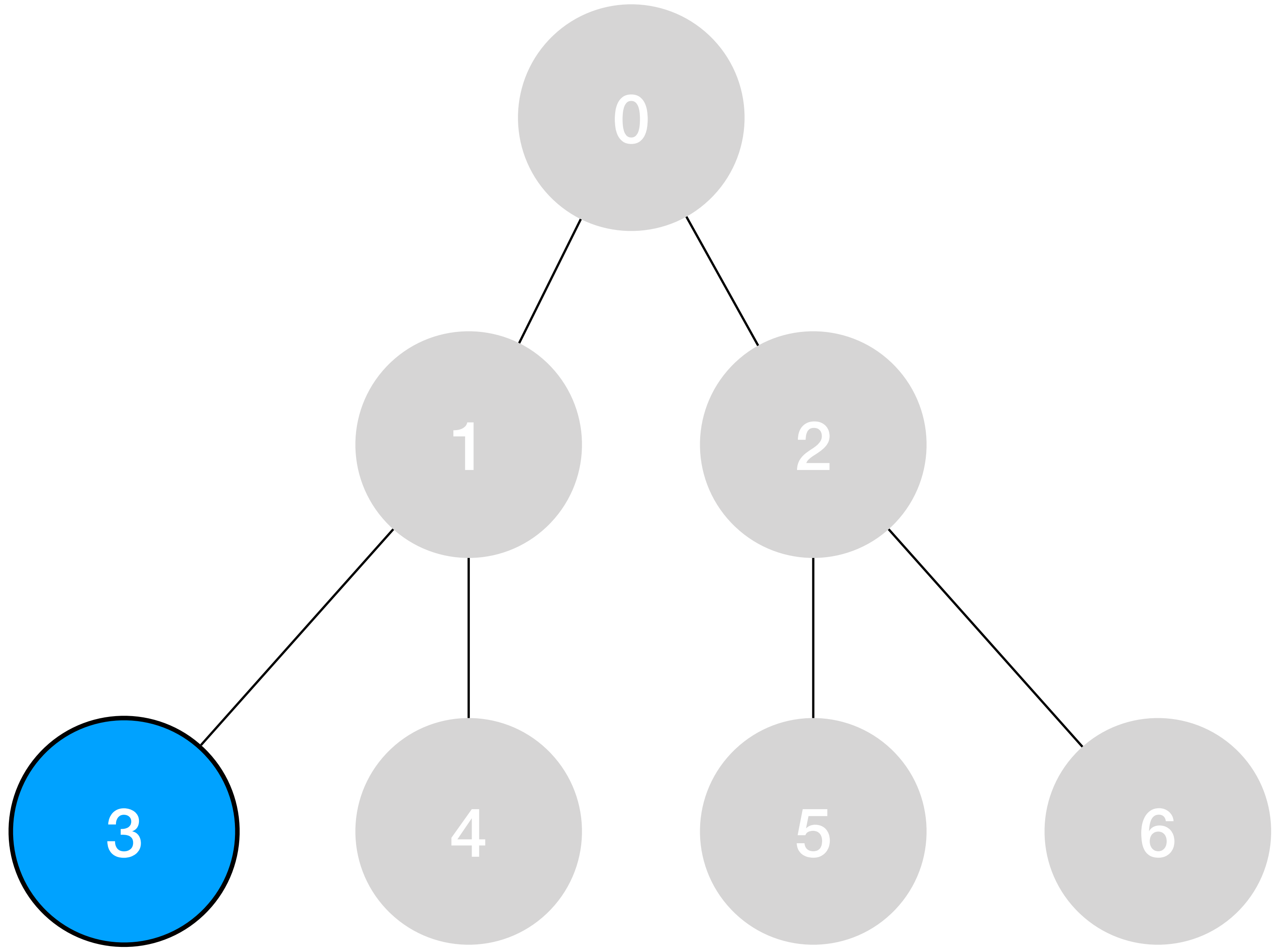
Conifer Implementation

- For a tree: find which leaf is reached given a data sample x
- ‘Invert’ the problem: for each node ask “does the decision path reach this node?” starting at the leaves



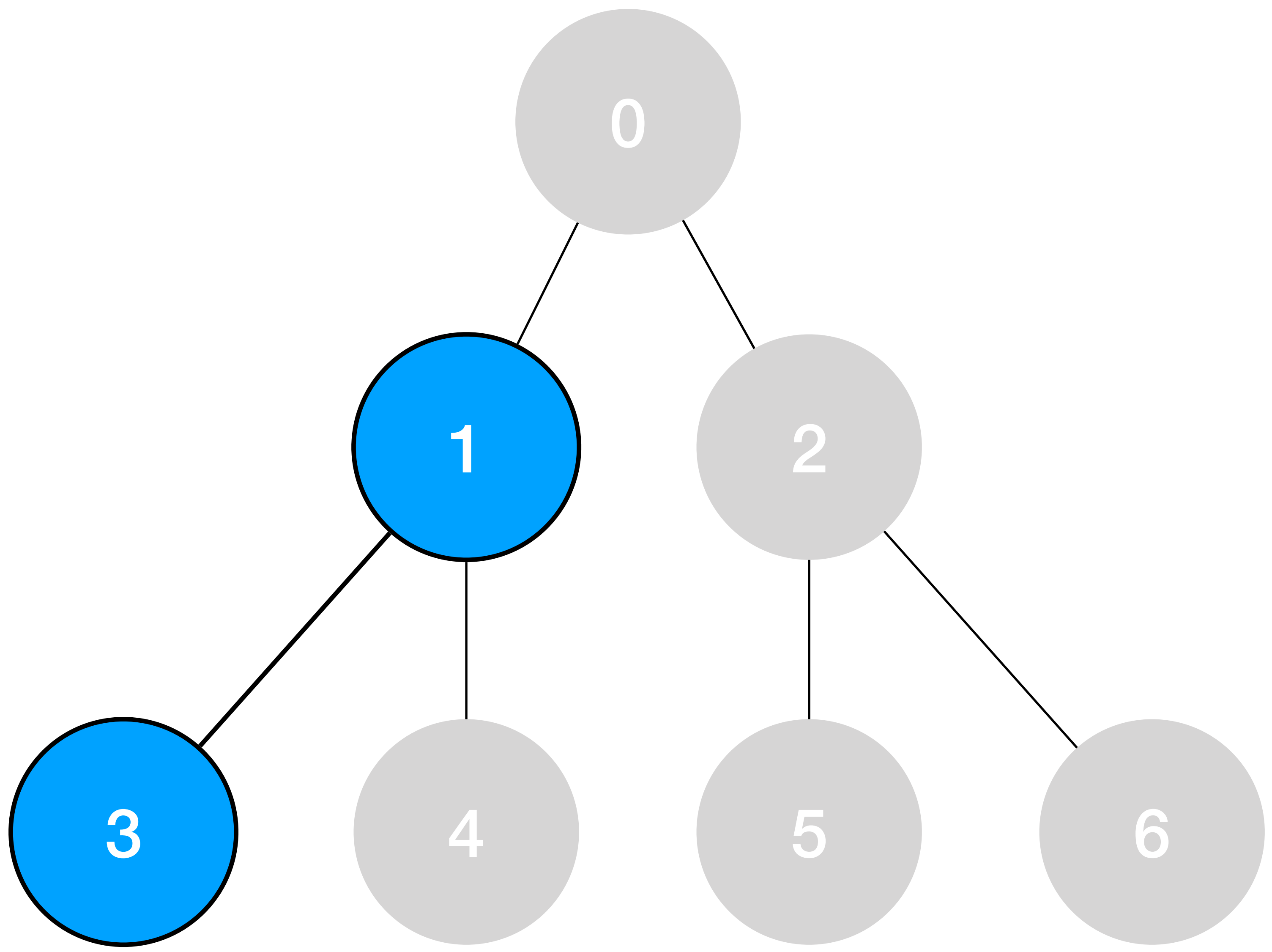
Conifer Implementation

- For a tree: find which leaf is reached given a data sample x
- ‘Invert’ the problem: for each node ask “does the decision path reach this node?” starting at the leaves
- For leaf node ‘3’:
  - The decision path reaches ‘3’ if: the decision path reached ‘1’ AND the comparison at ‘1’ goes ‘left’



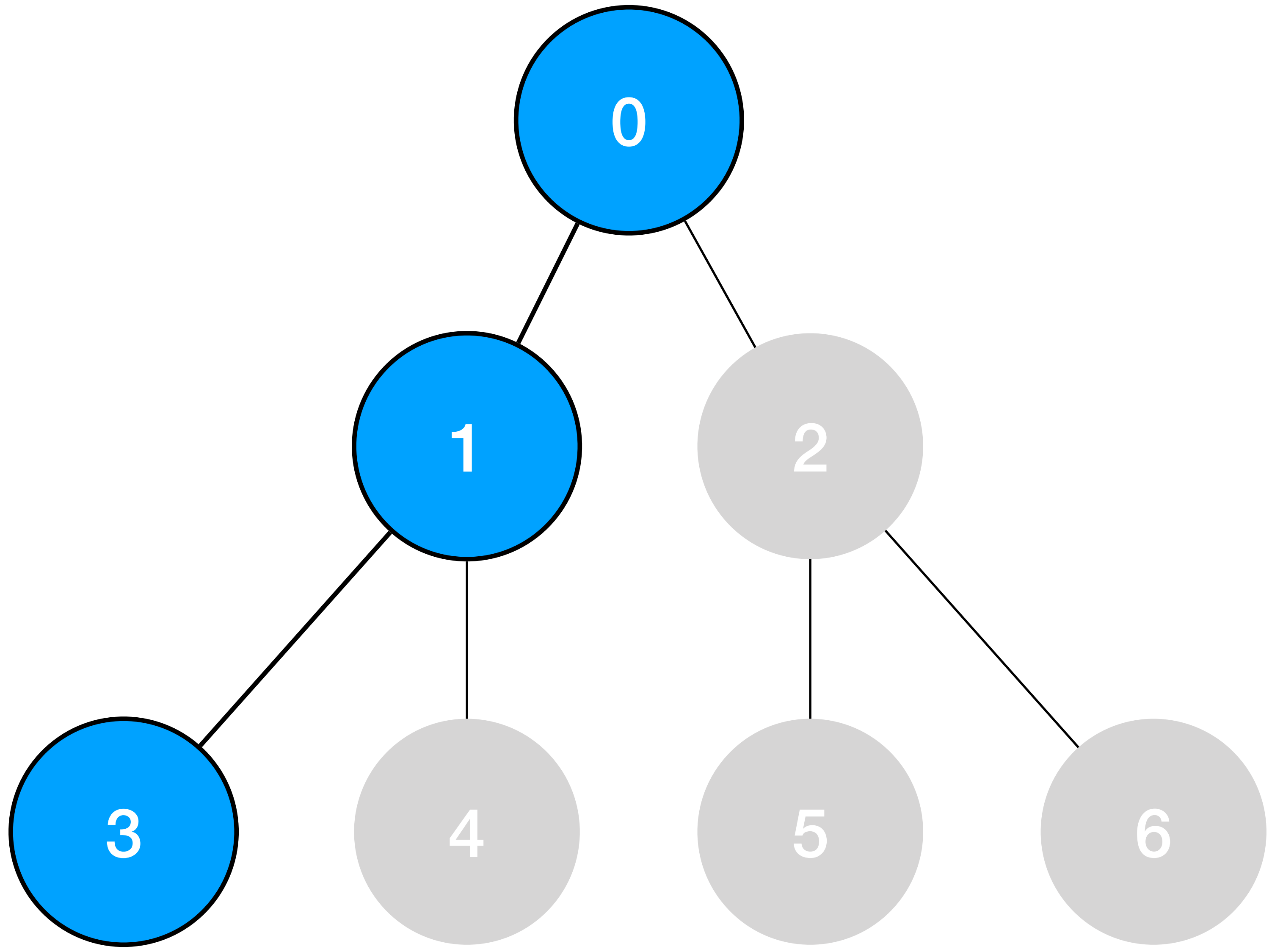
Conifer Implementation

- For a tree: find which leaf is reached given a data sample x
- ‘Invert’ the problem: for each node ask “does the decision path reach this node?” starting at the leaves
- For leaf node ‘3’:
  - The decision path reaches ‘3’ if: the decision path reached ‘1’ AND the comparison at ‘1’ goes ‘left’
- For node ‘1’:
  - The decision path reaches ‘1’ if: the decision path reached ‘0’ AND the comparison at ‘0’ goes ‘left’



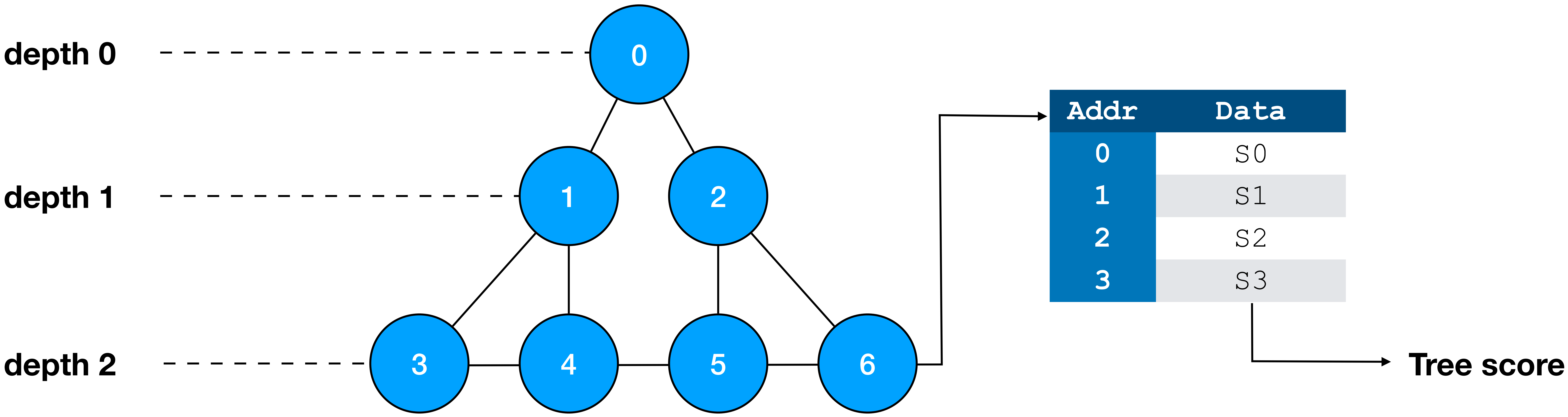
Conifer Implementation

- For a tree: find which leaf is reached given a data sample x
- ‘Invert’ the problem: for each node ask “does the decision path reach this node?” starting at the leaves
- For leaf node ‘3’:
  - The decision path reaches ‘3’ if: the decision path reached ‘1’ AND the comparison at ‘1’ goes ‘left’
- For node ‘1’:
  - The decision path reaches ‘1’ if: the decision path reached ‘0’ AND the comparison at ‘0’ goes ‘left’
- For node ‘0’:
  - The decision path always passes through the root node



Conifer Implementation

- For a tree: find which leaf is reached given a data sample x
- ‘Invert’ the problem: for each node ask “does the decision path reach this node?” starting at the leaves
- We can **parallelise** this over paths by brute force: evaluate all nodes at the same depth simultaneously
- We can **pipeline** this over different data: each node can do a comparison on new data with II=1
- For each leaf node we have a boolean: TRUE if the decision path reaches leaf, otherwise FALSE
- Concatenate the boolean for each leaf node → select the value corresponding to the leaf



Part 1: basics

Part 1: basics

- These notebooks are at https://github.com/thesps/conifer-tutorial/tree/smarthep
  - Training a BDT with XGBoost
  - Converting it to conifer with Xilinx HLS backend and fixed point representation
  - Emulation on CPU
  - Synthesis to FPGA for standalone IP (to be integrated into a custom design)
  - Synthesis to FPGA for pynq-z2 card
- My local setup:
  - Desktop PC for building FPGA firmware (good CPU and much RAM)
    - conifer master branch at 5ac32ec (conifer-1.6.dev10+g5ac32ec) - ahead of 1.5 with profiling and anomaly detection
    - Vitis HLS and Vivado 2024.1
  - pynq-z2 board
    - Base pynq image additionally with conifer 1.5 installed

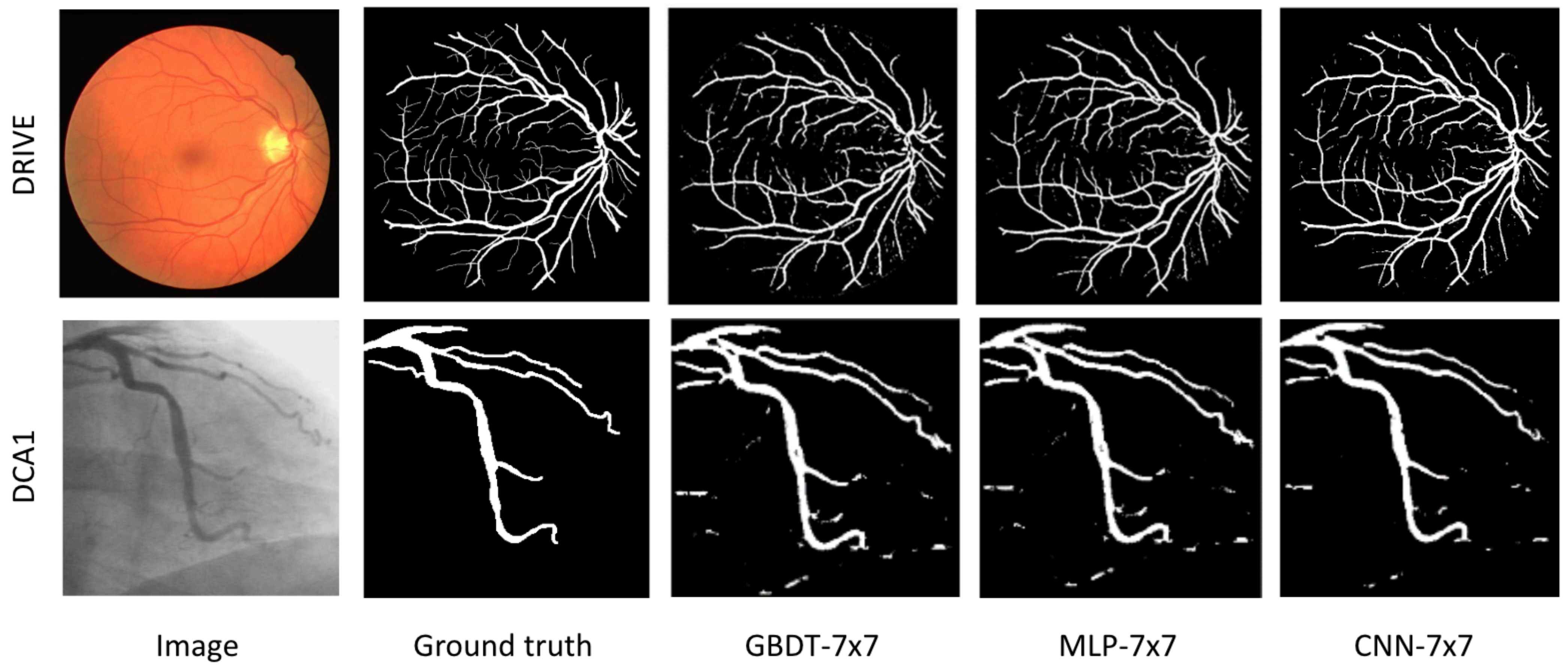
Part 2: Deployment

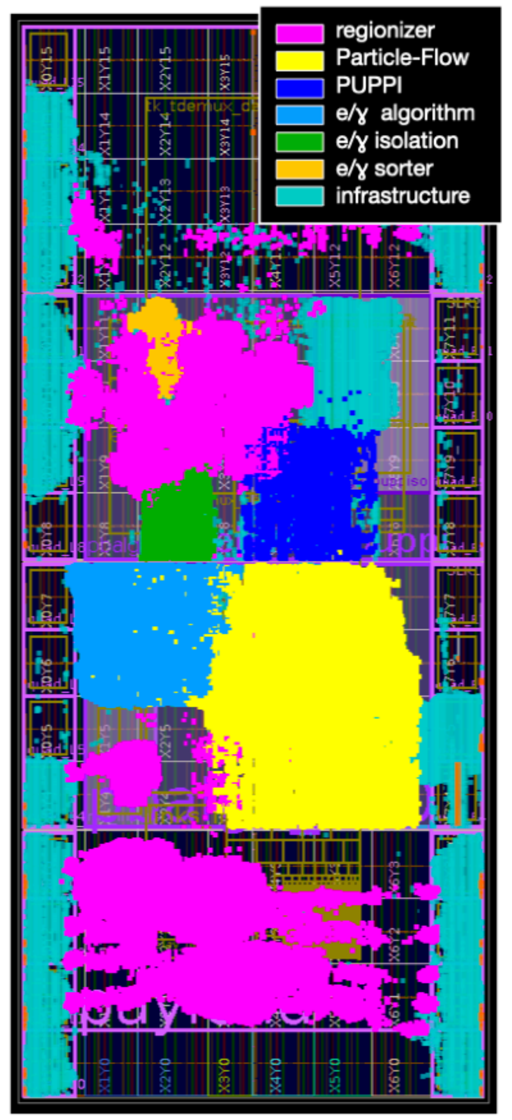
conifer deployment options

- There are five main ways to deploy conifer models to production:

1. Synthesize the HLS backend code ➝ produce RTL ➝ integrate it into some full design with RTL or Block Design
2. Call the HLS function from some other HLS, synthesize that ➝ integrate it into some bigger design
3. Use the VHDL backend ➝ integrate it into some bigger design
4. Synthesize the HLS backend code with a “board config” for a supported board ➝ build bitfile ➝ run with conifer runtime
5. Download or build a Forest Processing Unit bitfile ➝ run with conifer runtime

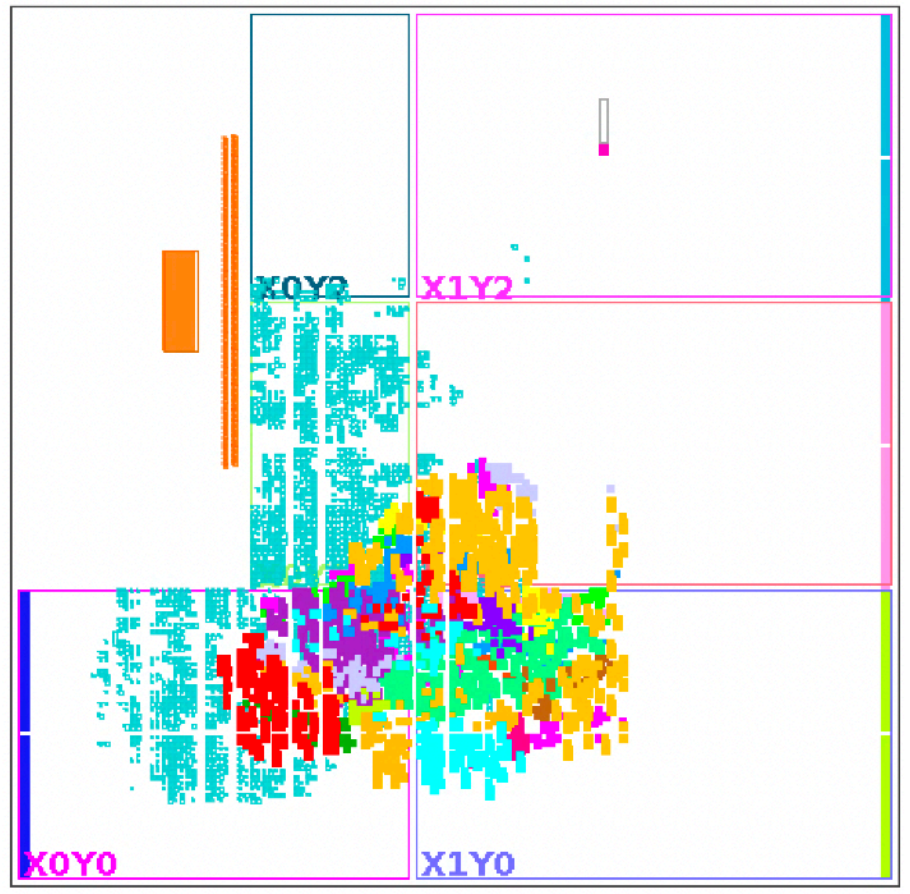
Uses 1.



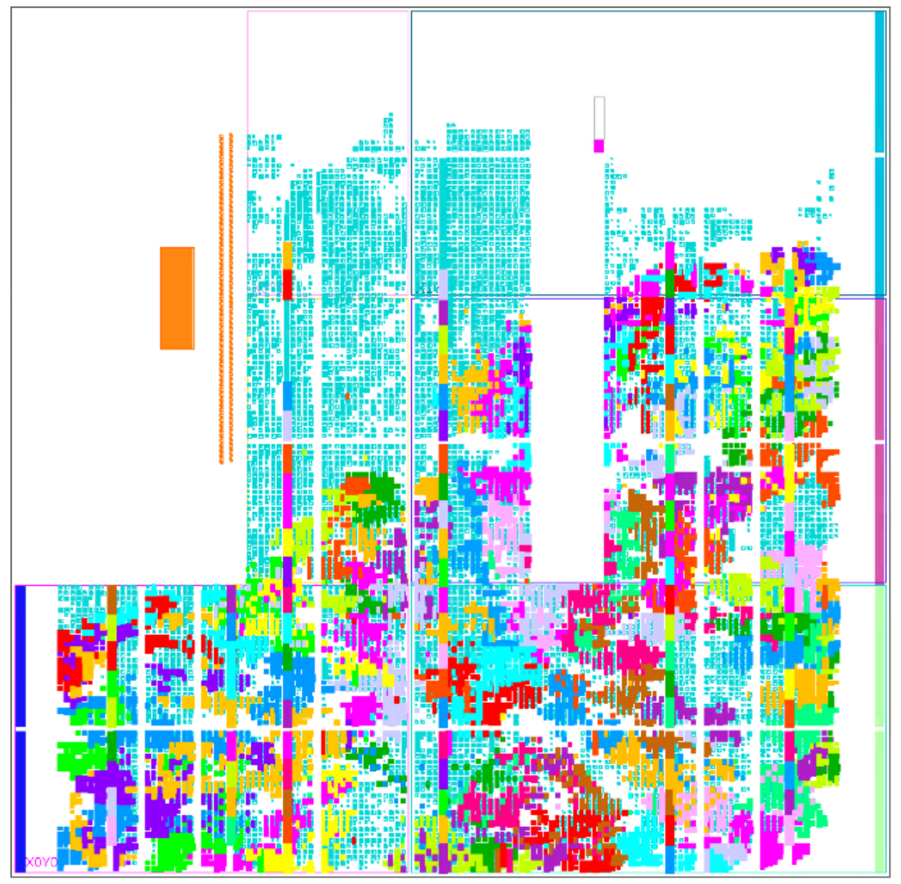


Uses 2.

4. HLS on pynq-z2

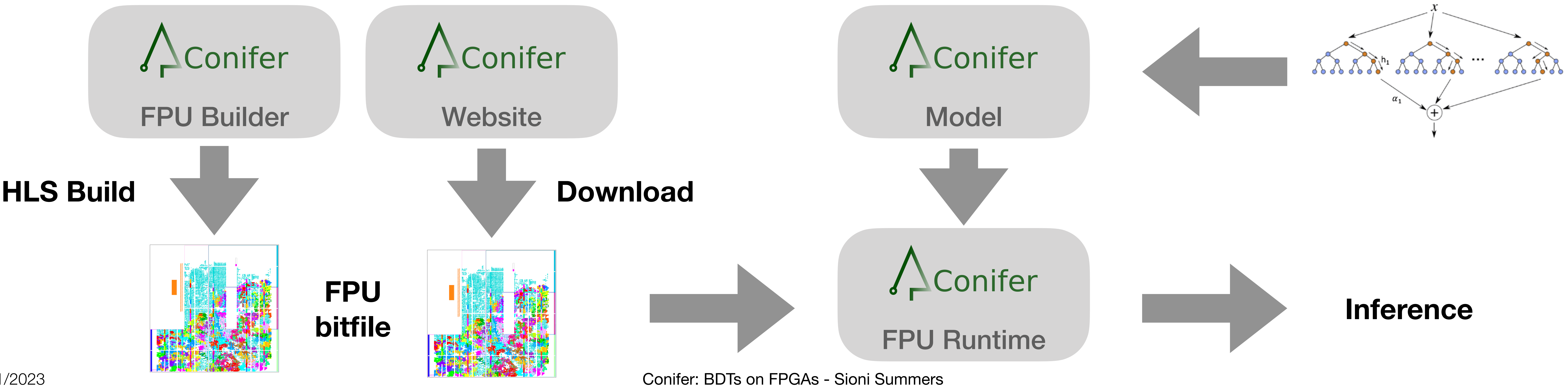


5. FPU on pynq-z2



Forest Processing Unit

- So far we looked at ‘static’ BDT evaluation
  - One trained model → one HLS function → one IP → one bitfile
  - So if the model changes at all, we need to redo everything → takes hours!
- In next section we will look at a more dynamic & reconfigurable implementation called “Forest Processing Unit” (FPU)
- Since one bitfile supports inference of many models, we can make the bitfiles for common hardware in advance
  - Check the downloads section of the conifer website: https://ssummers.web.cern.ch/conifer/downloads/
  - There are binaries for Zynq-based boards like pynq-z2, ultra96v2, Kria, and also Alveo boards like U200

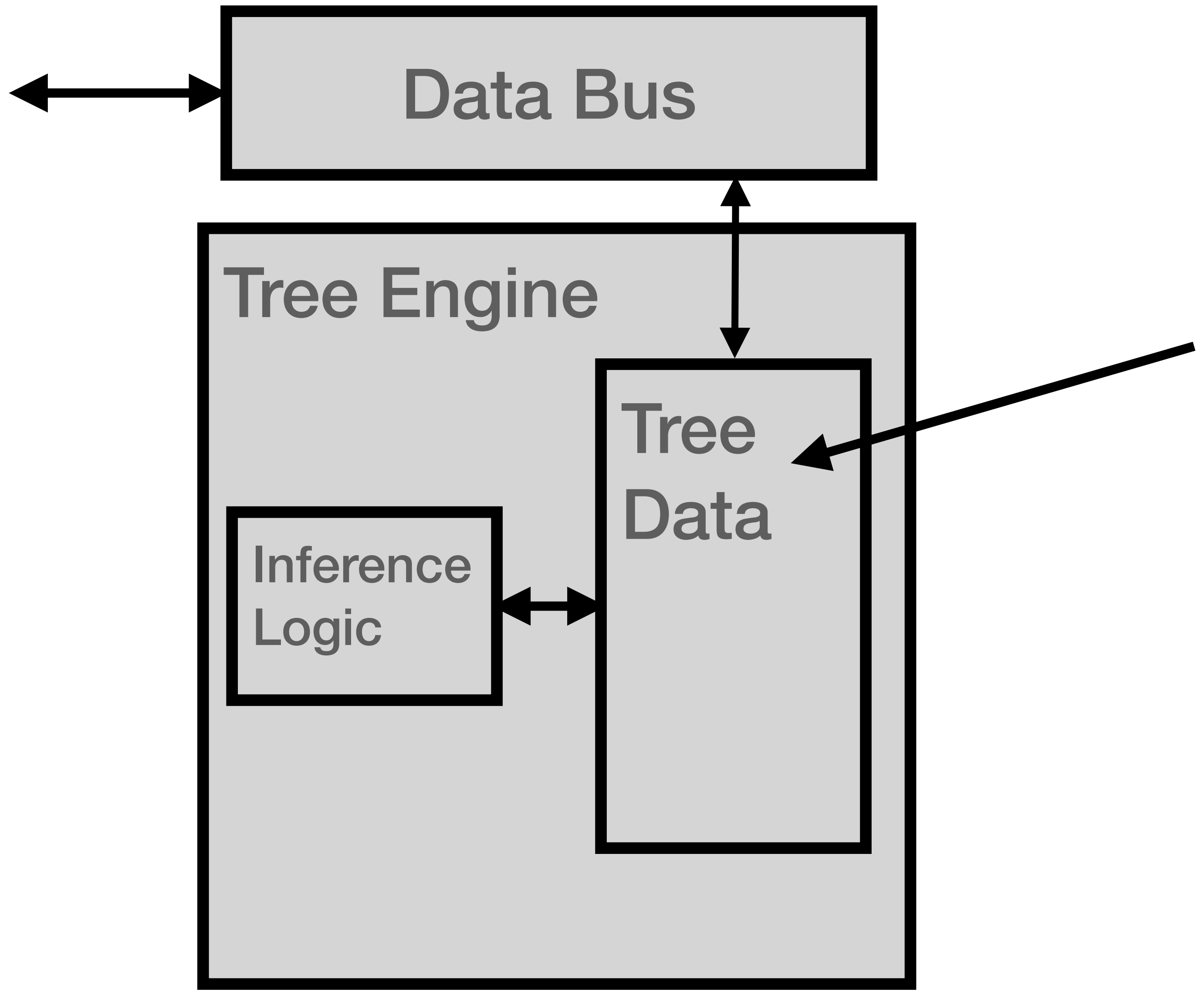


FPU Design

- We would like a base design that can perform inference of ~any BDT model afterwards (within some limits)
- And we would like to take advantage of the FPGA to get good performance (fast inference)
- **Idea 1**: represent the BDT as data, operate inference on that data, and load new data for a new model
- **Idea 2**: parallelise over trees by having independent ‘Tree Engines’, aggregate their output for the model

FPU Design

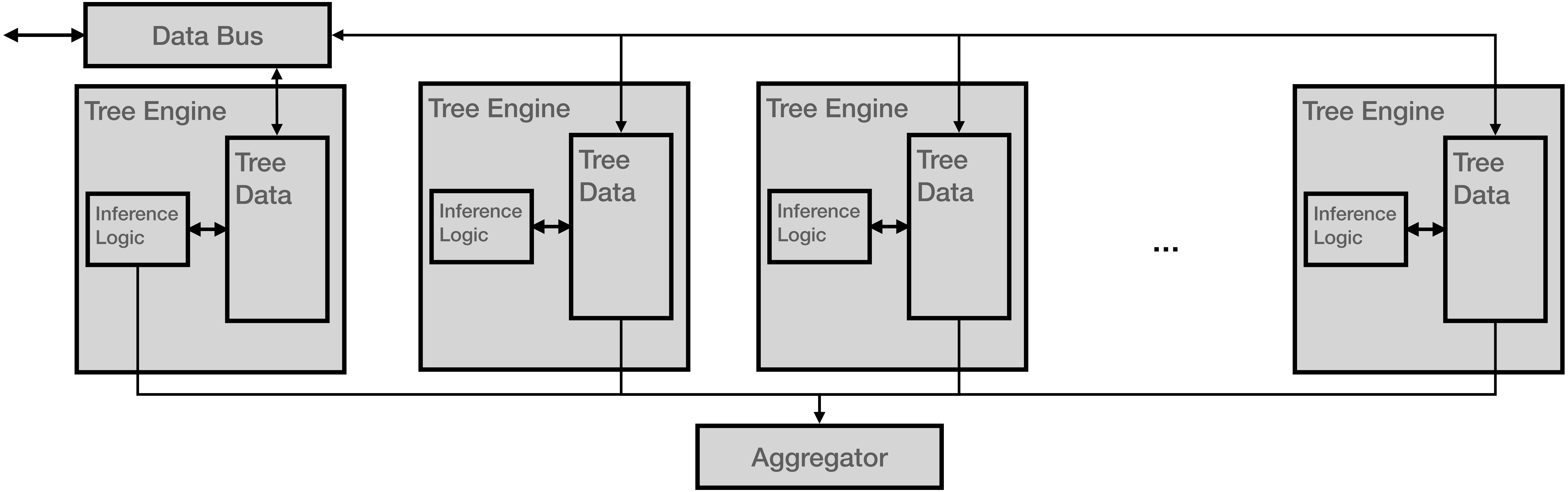
- **Idea 1**: represent the BDT as data, operate inference on that data, and load new data for a new model over a bus
- Map Decision Trees onto memory
    - Target FPGA Block RAMS: many independent small memories
- Store one node at each address, child indices are pointers to other addresses
- Logic starts inference at the root node and iterates until reaching a leaf




```
Tree:
      index          : [ 0,  1,  2,   3,   4,    5,  6]
      children_left  : [ 1,  3,  5,  -2,  -2,   -2, -2]
      children_right : [ 2,  4,  6,  -2,  -2,   -2, -2]
      parent         : [ -1, 0,  0,   1,   1,    2,  2]
      feature        : [ 4,  7,  1,  -2,  -2,   -2, -2]
      threshold      : [ 7,  2,  9,  -2,  -2,   -2, -2]
      value          : [-1, -1, -1, 0.5, 0.4, -0.5, -1]
```

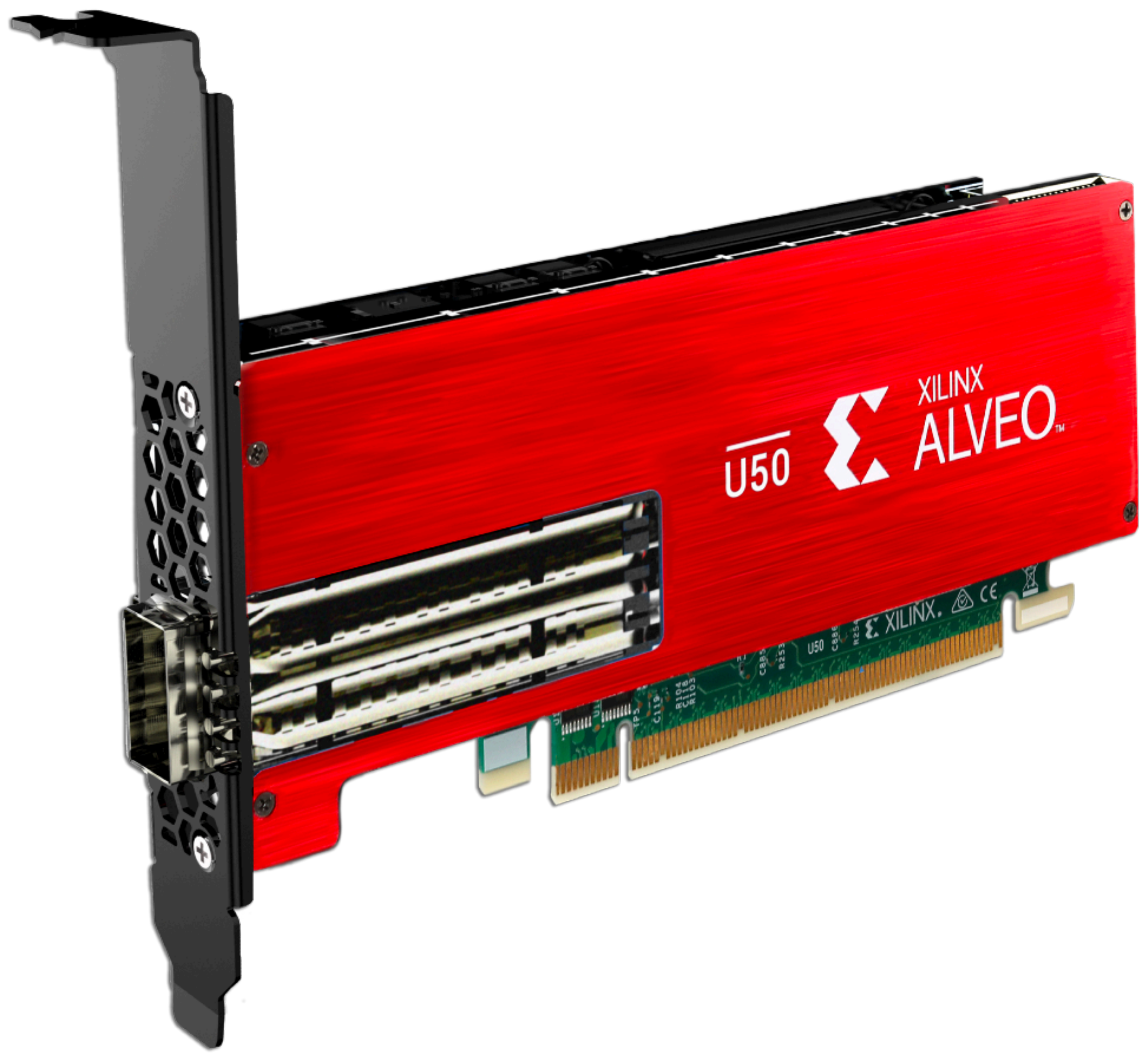
FPU Design

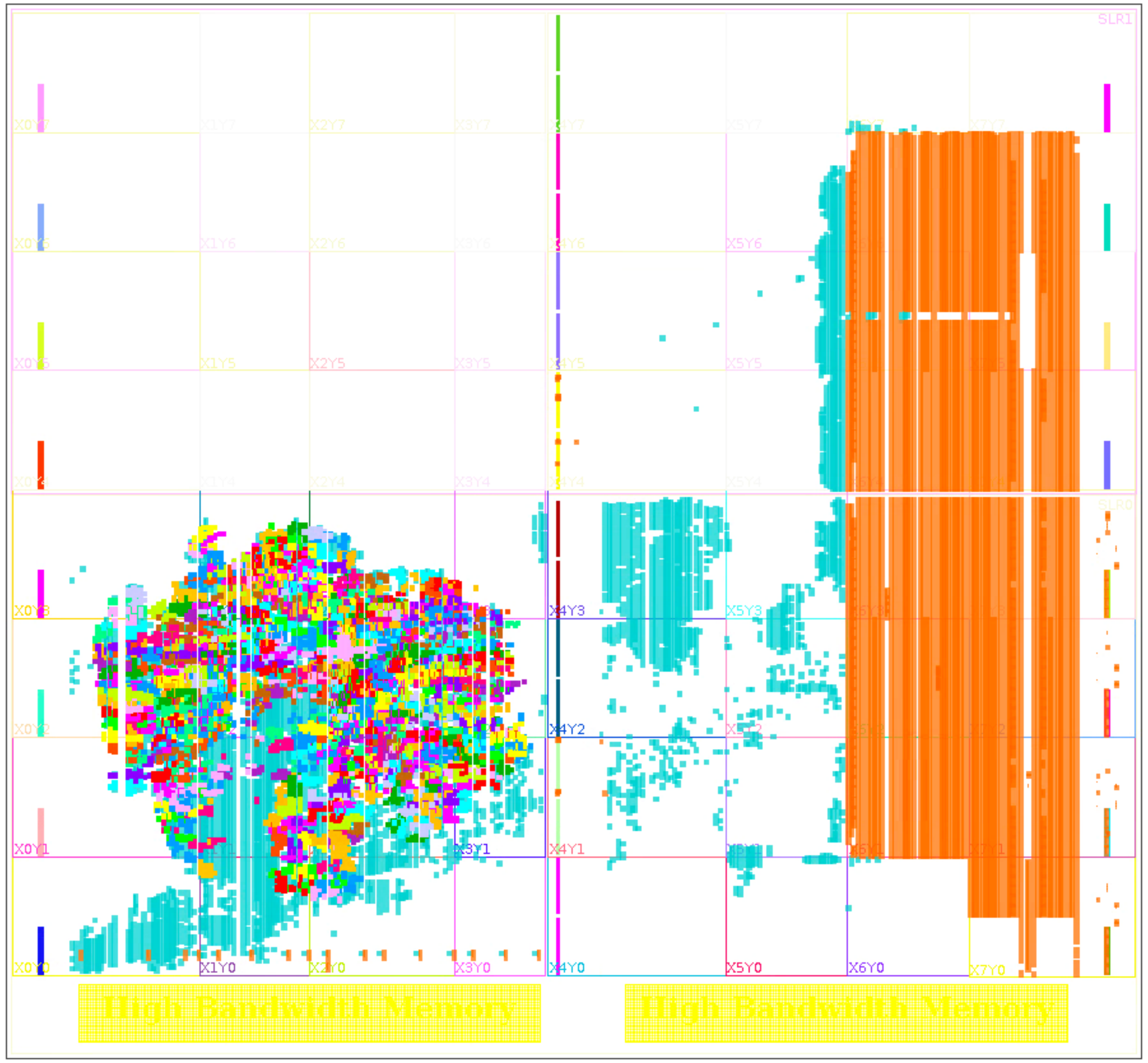
- **Idea 2**: parallelise over trees by having independent 'Tree Engines', aggregate their output for the model
- Put as many Tree Engines as will fit in the FPGA
- Number of Tree Engines will constrain the model size that fits



FPU Floorplan

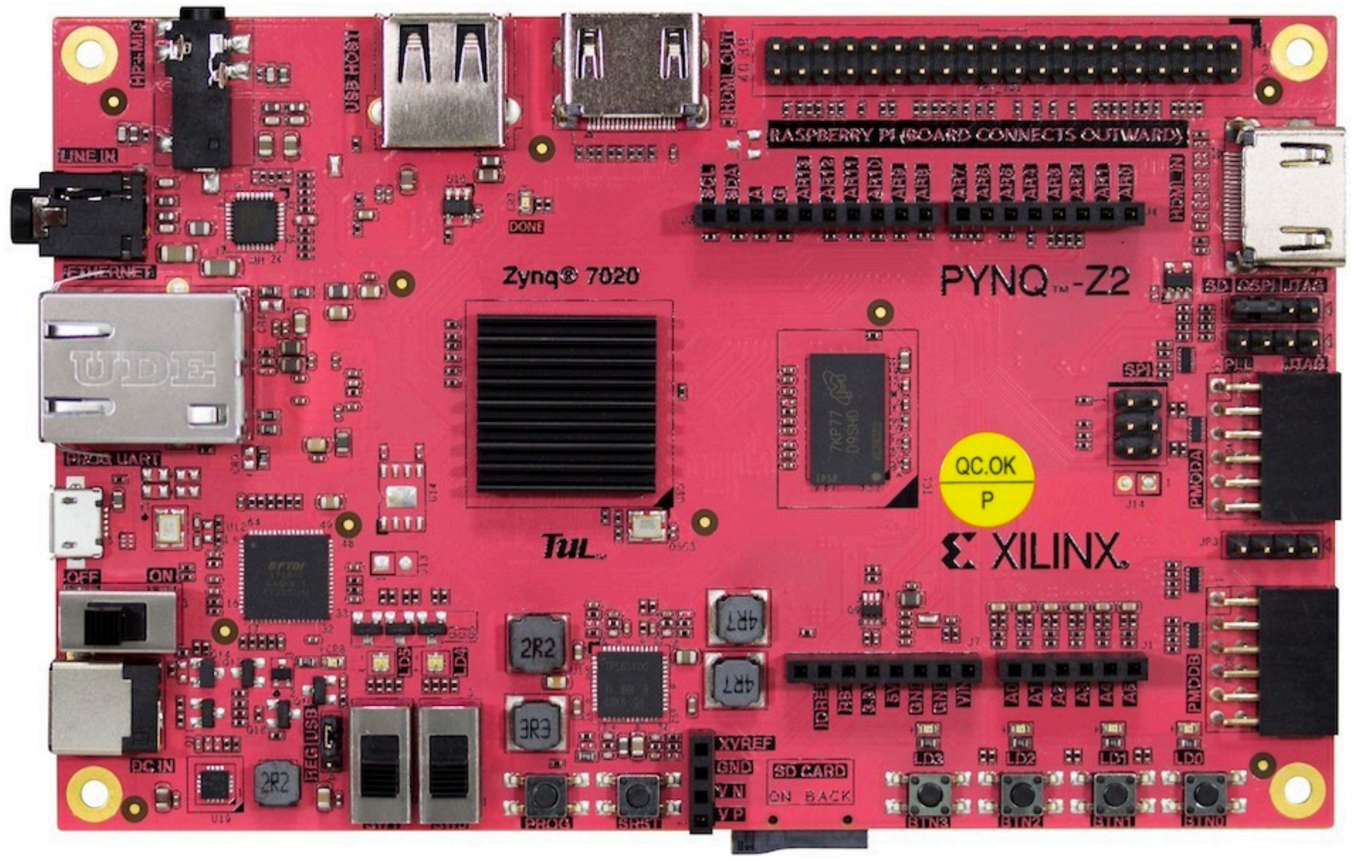
- FPU with 200 Tree Engines in Alveo U50
  - Each TE is highlighted in colour (with a repeating cycle)
- BRAMs for nodes are in columns
- Logic near BRAMs is TE inference logic

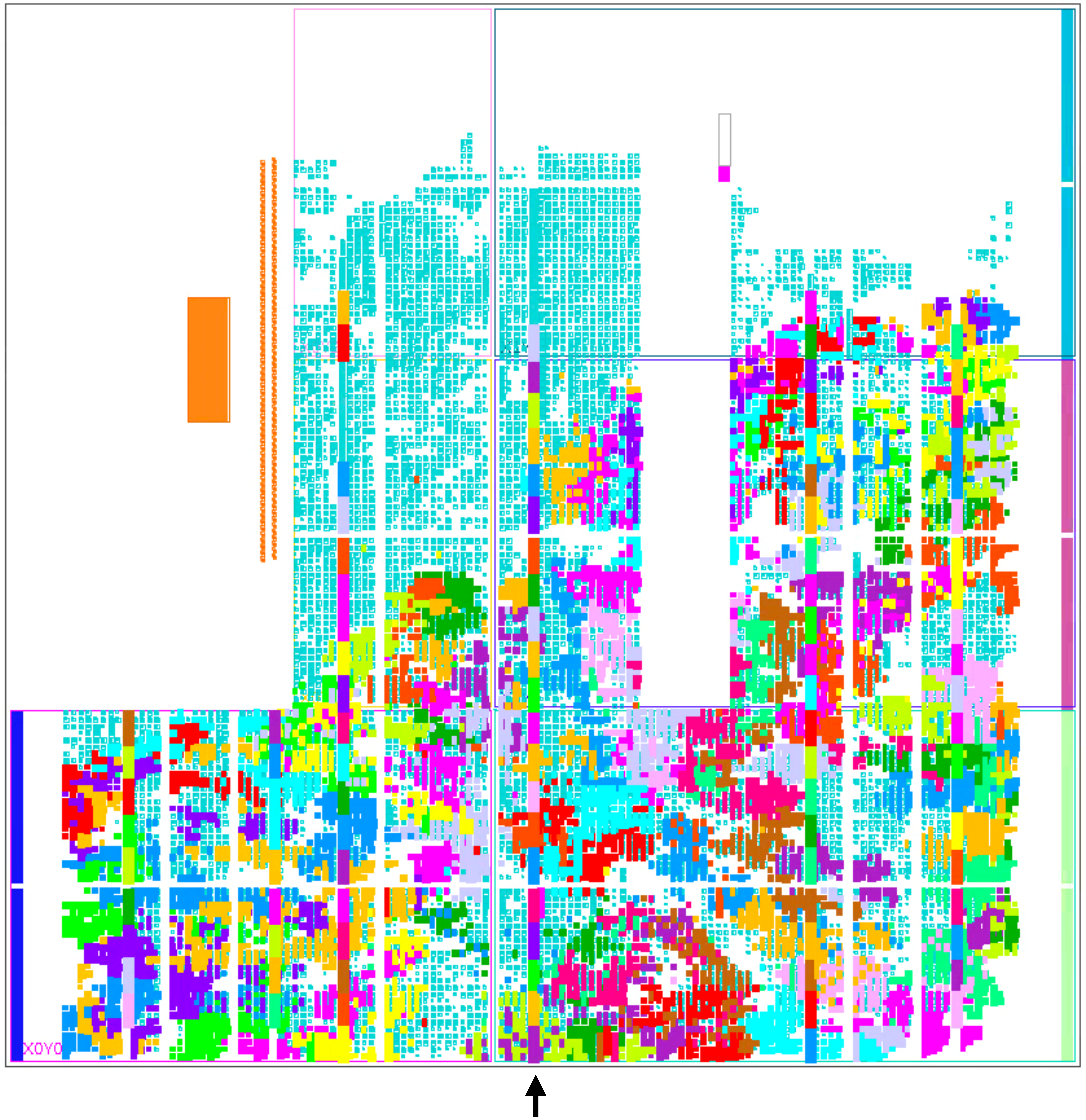




FPU Floorplan

- FPU with 100 Tree Engines in pynq-z2
  - Each TE is highlighted in colour (with a repeating cycle)
- BRAMs for nodes are in columns
- Logic near BRAMs is TE inference logic





BRAM column

Part 3: Anomaly Detection

Anomaly Detection

- conifer recently added support for the popular Decision Forest anomaly detection algorithm called "Isolation Forest" with the Yggdrasil package (`ydf`)
  - It's not yet release, but is in the master branch and will be in conifer 1.6
- Anomaly Score of a data point is related to the average depth that it takes to segment it
- In this demo we train an Isolation Forest with `ydf` and deploy to FPGA with conifer
- Liu et al., Isolation Forest