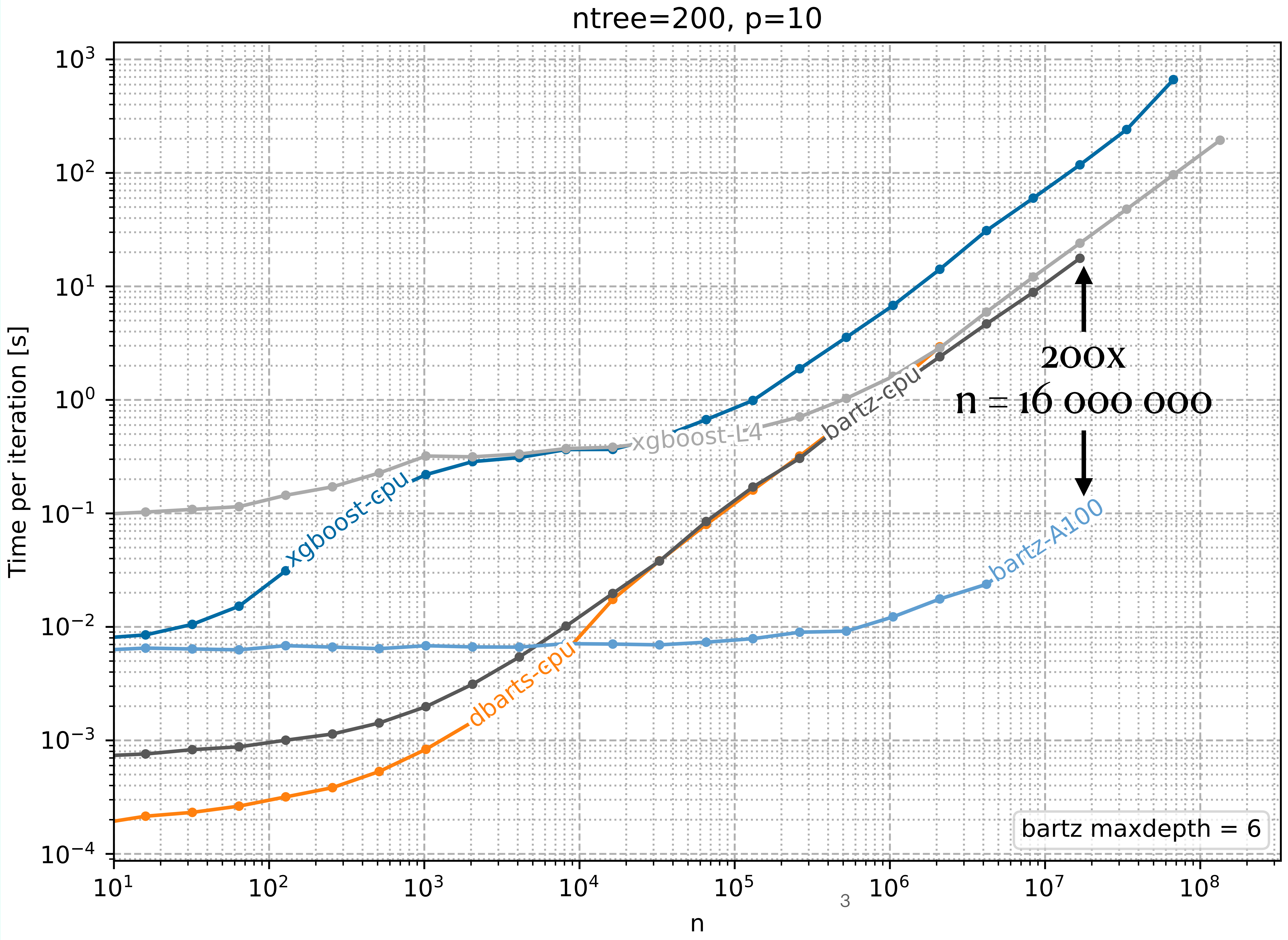
BART on GPU

up to 200x faster

Giacomo Petrillo giacomo.petrillo@unifi.it
Department of Statistics, Computer Science Applications (DISIA), University of Florence
At the BART reading group, SDS UT Austin
April 18, 2024

Summary

- I implemented the original BART MCMC in JAX
- It's on PyPI: `pip install bartz`
- (JAX is a Python library for numerical computation)
- CPU: as fast as dbarts (SoTA), uses less memory
- GPU: up to 200x faster, but depends on number of trees and sample size



Time for one full MCMC step, at fixed number of trees and number of predictors, w.r.t. sample size

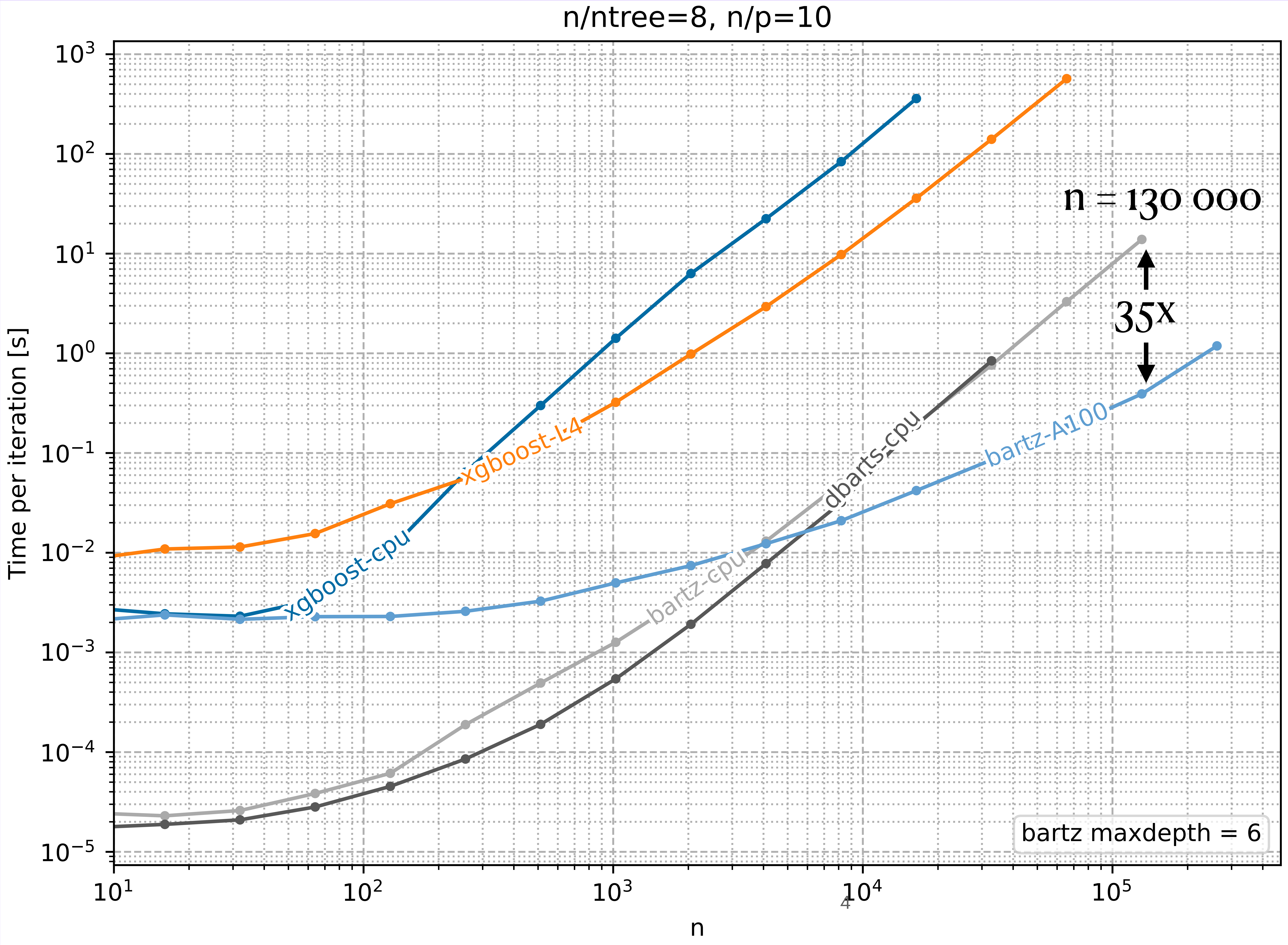
A100 = GPU you have at TACC

L4 = smaller but newer GPU

CPU = single Apple M1 core

For xgboost, I measure the time to construct all the trees

DGP is silly; bartz is branchless so it does not matter, but it may make a difference for other software



With p and ntree $\propto n$

I reach lower n because I run out of memory

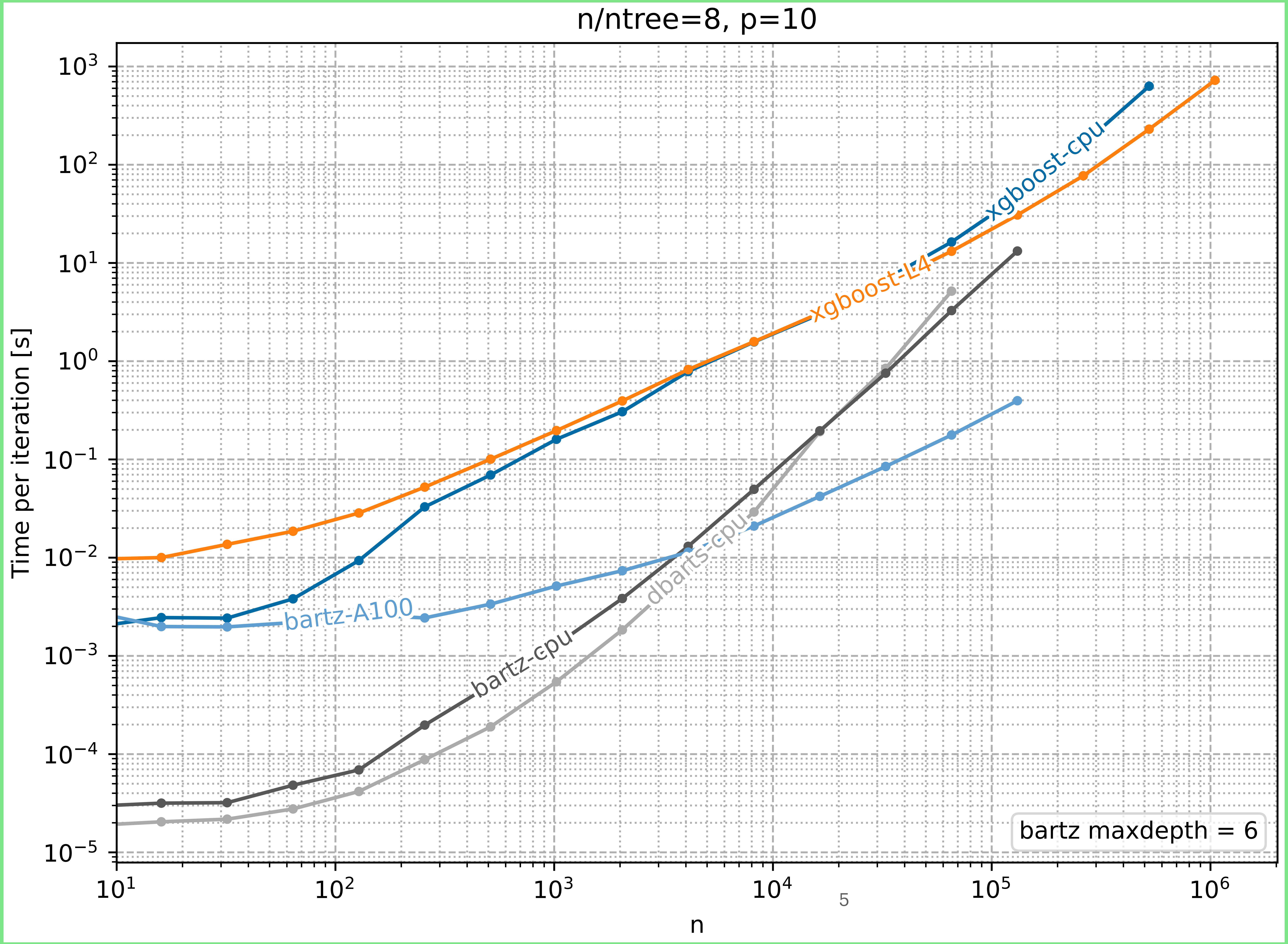
A100 = GPU you have at TACC

L4 = smaller but newer GPU

CPU = single Apple M1 core

For xgboost, I measure the time to construct all the trees

DGP is silly; bartz is branchless so it does not matter, but it may make a difference for other software



With ntree ∝ n, fixed p

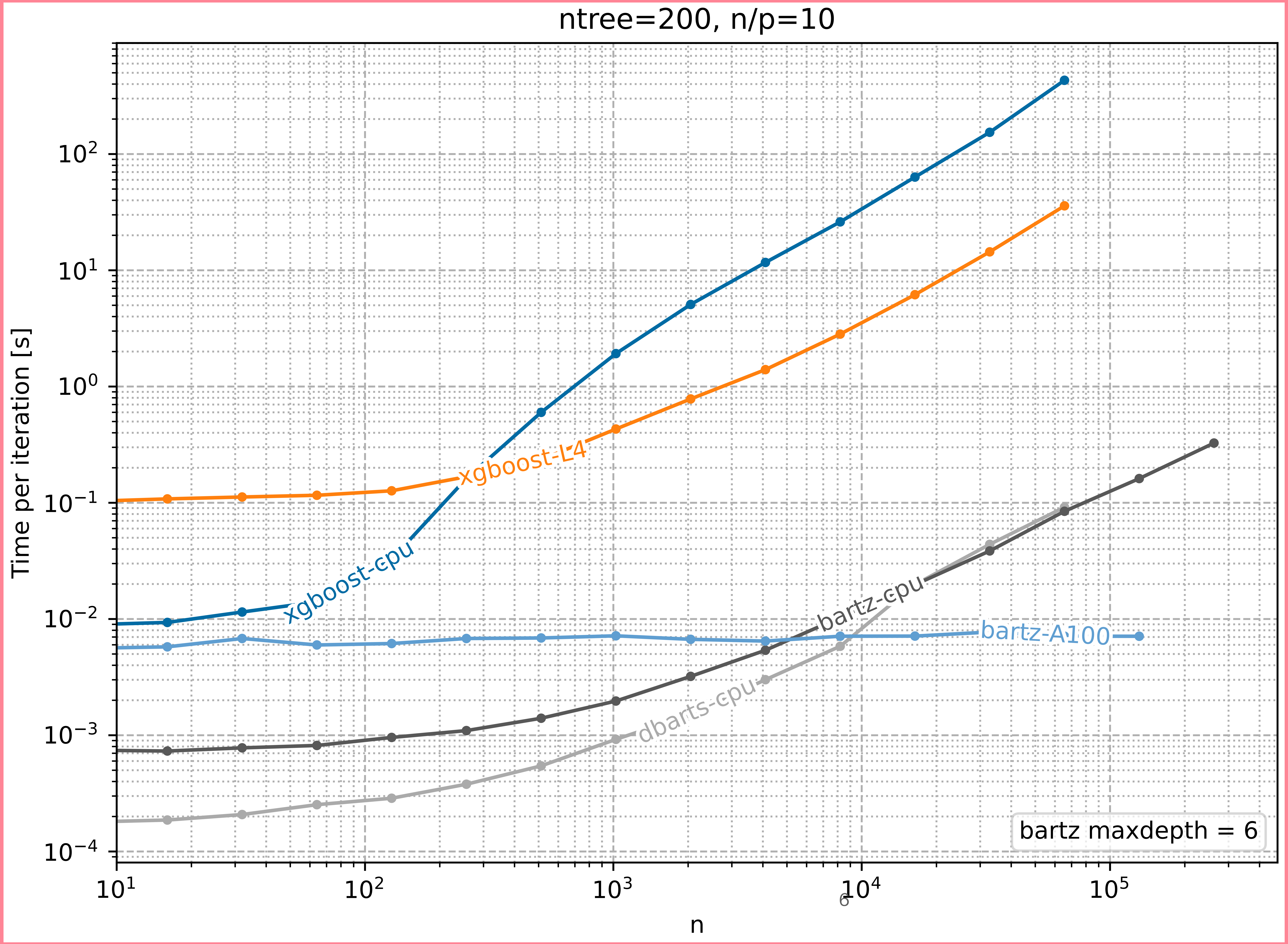
A100 = GPU you have at TACC

L4 = smaller but newer GPU

CPU = single Apple M1 core

For xgboost, I measure the time to construct all the trees

DGP is silly; bartz is branchless so it does not matter, but it may make a difference for other software



With $p \propto n$, fixed ntree

A100 = GPU you have at TACC

L4 = smaller but newer GPU

CPU = single Apple M1 core

For xgboost, I measure the time to construct all the trees

DGP is silly; bartz is branchless so it does not matter, but it may make a difference for other software

Disclaimer

- Still not checked that the result is good at high n or ntree
  - Numerical accuracy problems in my implementation because I use 32 bit floats?
    - (Probably not significant now, easy to fix anyway)
  - Is BART a good model at high n?
  - How many trees should I use? I believe $\propto n$
- I test routinely at low n against the R package BART, it's correct there

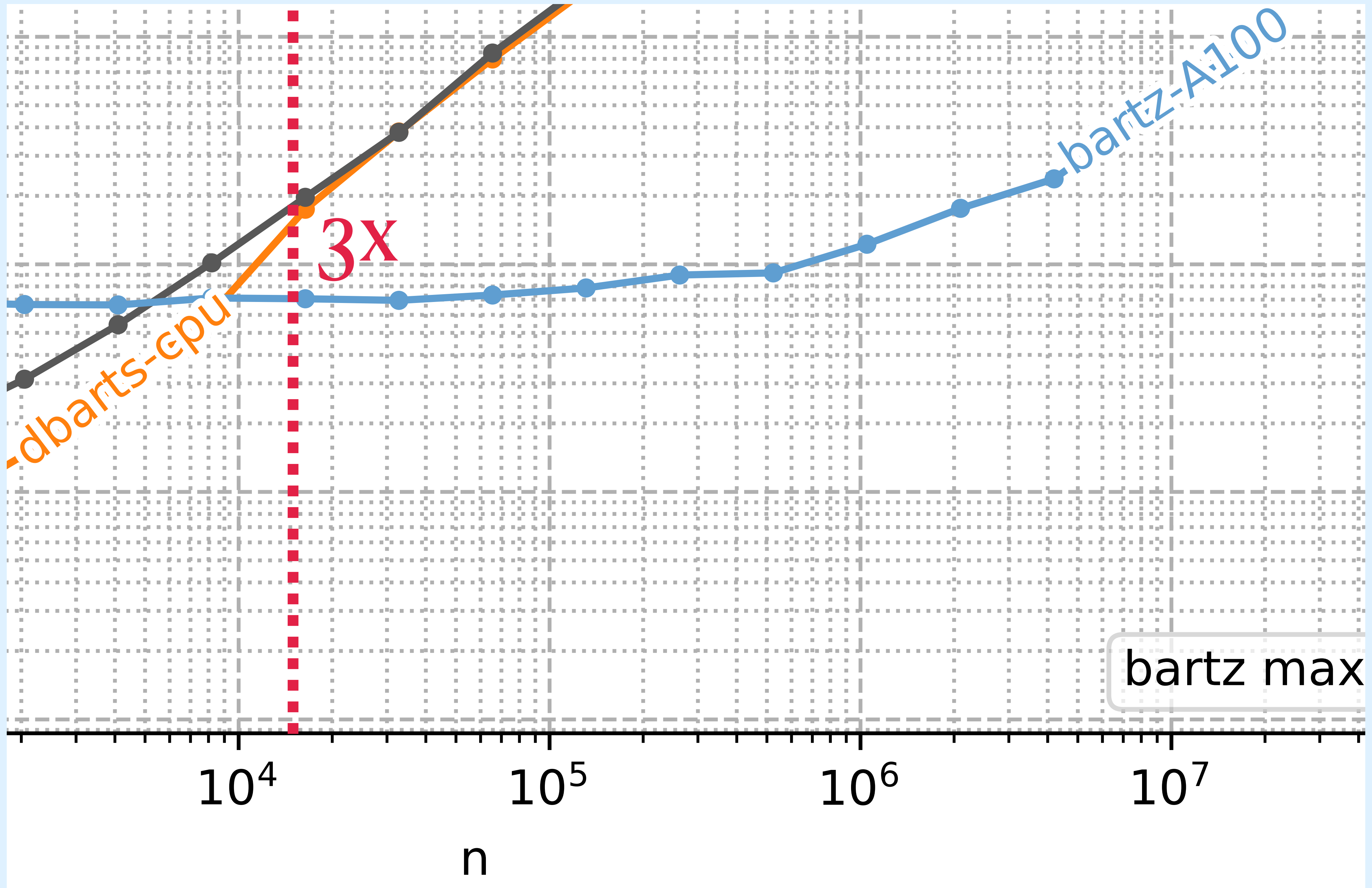
Tooling

How did I manage to make it faster?

1. I wouldn't touch R/C++ with a 10 feet pole
2. Indeed
3. Yes
4. Do I need to spell it out?

Is this enough?

- No, it is woefully inadequate!
- The problem I'd like to work on has $p = 10\,000\,000$ binary predictors
- I can fit $n_{\text{part}} < 15\,000$ on each A100 GPU (20 GB design matrix slice)



Implementation details

Branchless

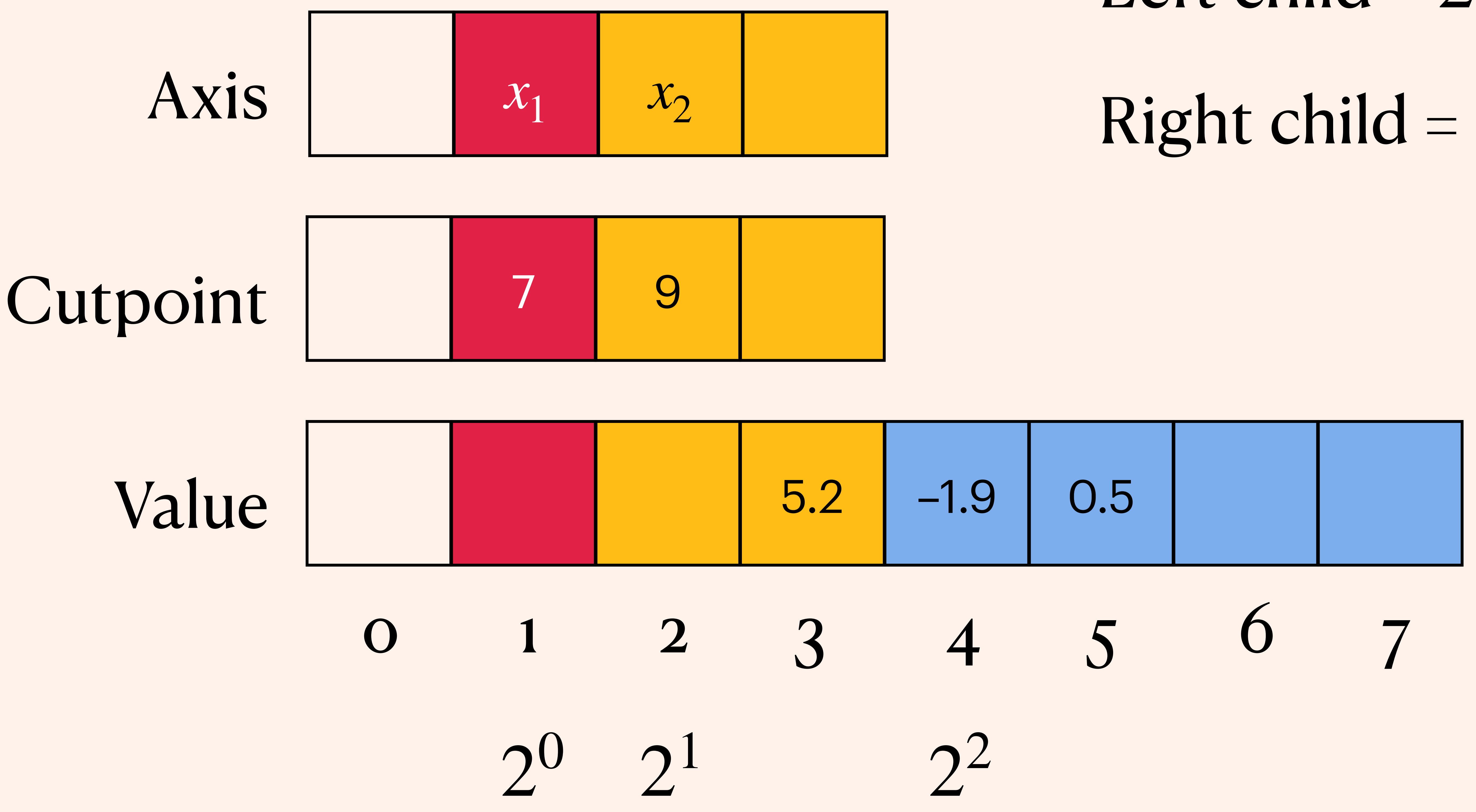
- Branchless = the algorithm always does the same sequence of operations, irrespective of the inputs
- E.g., if a leaf has depth 2, I still traverse a fixed maximum number of levels to arrive at it
- E.g., if I split a leaf in a tree, I recompute the datapoint partition for all other leaves

Why branchless?

- Parallelize automatically on GPU
- Even on CPU, it's good:
  - Doesn't disrupt the pipeline
    - (Pipeline = the CPU starts the next instruction before finishing the current one, this is broken if the next instruction depends on the result of the previous)
  - Vectorization
  - Predictable memory access
    - (Getting things from RAM is the slowest operation)

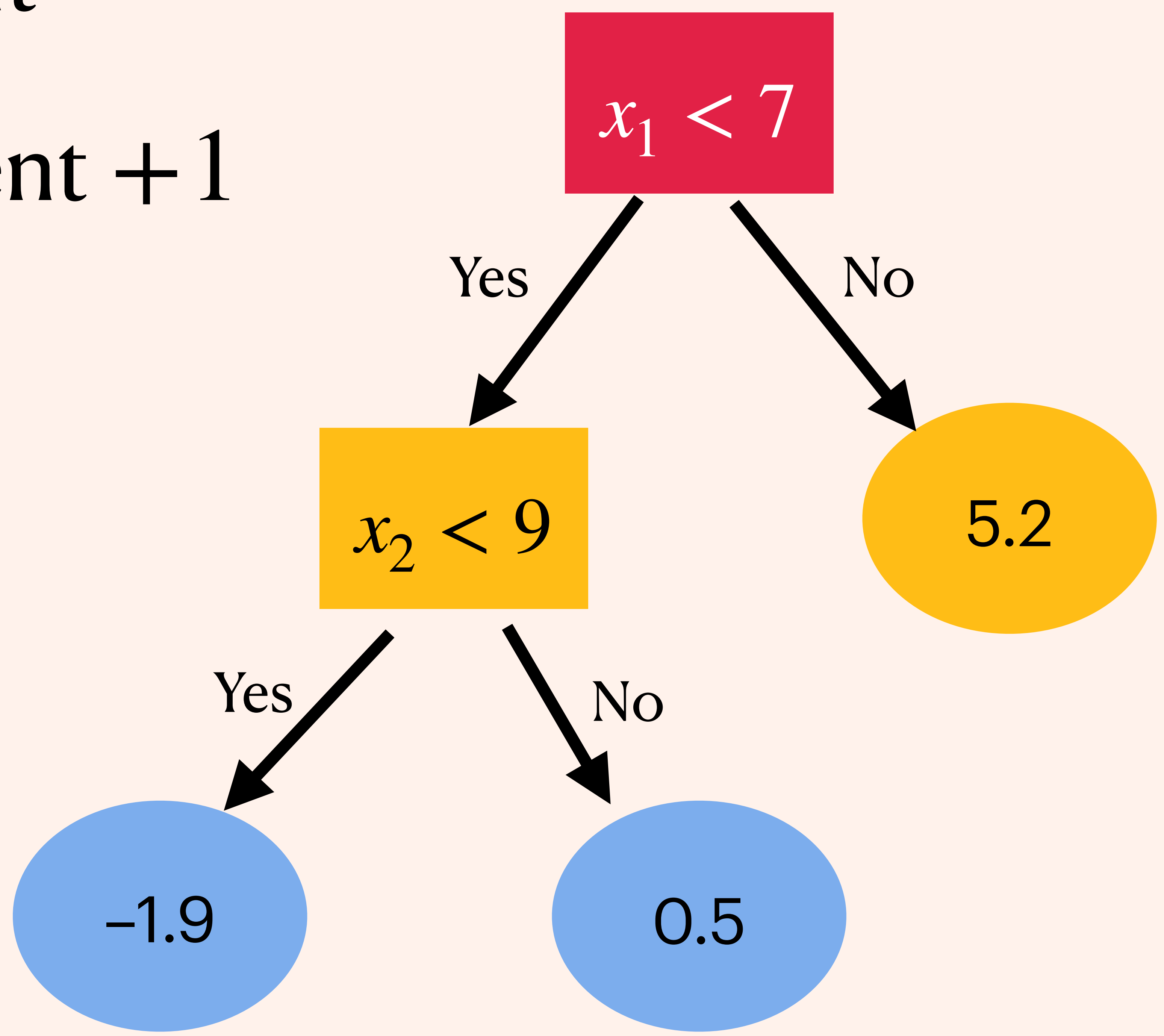
Tree representation

I represent trees as heaps



$$\text{Left child} = 2 \times \text{parent}$$

$$\text{Right child} = 2 \times \text{parent} + 1$$



Tree traverse

(sorting datapoints into leaves)

- Big ntree $\times\, n$ matrix of indices
- M_{ti} = index of leaf containing point i in tree t
- If max tree depth ≤ 8, requires one byte per element
- At $n = 100\,000$, ntree $= 10\,000$, it's 1 GB

datapoint

tree

3	3	3	3	2	3	3	3	3
3	3	2	3	3	2	3	3	3
2	3	2	3	3	3	3	3	3
2	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	1
4	4	4	4	4	5	4	4	4

Tree sampling step outline

Parallel part

- For all trees at once:
  - Propose a grow or prune move (grow = make two new leaves, prune = remove two leaves)
  - Where grow, update the leaf indices to represent the grow move
  - Count the number of points per leaf
  - Compute the posterior variance
  - Sample centered leaf values
  - Compute most of the Metropolis ratio terms

Tree sampling step outline

Sequential part

- One tree at a time:
  - Sum the residuals in each leaf (**SLOOOOOW)**
  - Subtract the old leaf values from the sum of residuals
  - Finish MH ratio calculation
  - Accept/reject move
  - Add posterior mean to new leaves
  - Add new leaves to residuals

Bottleneck

Slowest part of the algorithm

- Summing residuals
  - I can't really parallelize it across trees
  - It doesn't parallelize enough within a single tree if n is not high (see slide 3)
  - Makes the running time $O(\text{ntree})$ at smallish n (see slide 4)
  - This operation is called *indexed reduce*
    - It's *memory-bound*: I do a simple operation on many elements, so the bottleneck is fetching the elements, not the operation
    - This means float16 does a 2x respect to float32, not 10x

Ideas to parallelize across trees?

Links

- https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units#Tesla
- https://github.com/Gattocrucco/bartz (has documentation)
- If you want to use this and need a feature, open an issue: https://github.com/Gattocrucco/bartz/issues
- Example on Colab, if you don't have a local GPU: https://colab.research.google.com/drive/1BHl_NnhoVY-cUvCe5Topub4mgnOkGGO5?usp=sharing