Online machine learning
on the road

Max Halford — 2023/10/26 — TH Köln, Institute for Data Science, Engineering, and Analytics

https://gist.github.com/MaxHalford/f57a540333d47b21e6eaff49071e3ff1

- Head of Data @ Carbonfact
- PhD in database query optimisation
- Creator and maintainer of River
- Kaggle competitions master

The industry isn’t ready, yet

- Batch learning works for most problems
- Real-time feature engineering isn’t trivial
  - See companies in this space: Tecton, Claypot, Fennel
  - Feature Engineering for Personalised Search by Nick Parsons
- Lack of testimonials from the industry
- This is the status quo: let’s challenge it!

BigTech is leading the way

WEIBO

Microsoft

ByteDance

Google

Practical Lessons from Predicting Clicks on Ads at Facebook

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu*, Tao Xu*, Yanxin Shi*,
Antoine Atallah*, Ralf Herbrich*, Stuart Bowers, Joaquin Quiñonero Candela
Facebook
1601 Willow Road, Menlo Park, CA, United States
{panjunfeng, oujin, joaquinq, sbowers}@fb.com

Monolith: Real Time Recommendation System With Collisionless Embedding Table

Zhuoran Liu, Bytedance Inc.
Leqi Zou, Bytedance Inc.
Xuan Zou, Bytedance Inc.
Caihua Wang, Bytedance Inc.
Biao Zhang, Bytedance Inc.
Da Tang, Bytedance Inc.
Bolin Zhu*, Fudan University
Yijie Zhu, Bytedance Inc.
Peng Wu, Bytedance Inc.

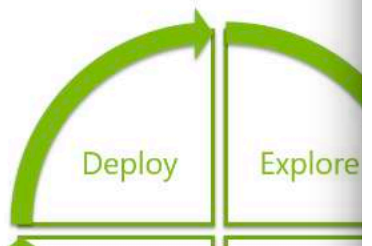
A Multiworld Testing Decision Service

Alekh Agarwal, Sarah Bird, Markus Cozowicz, Luong Hoang
Stephen Lee*, Jiaji Li*, Dan Melamed, Gal Oshri*, Osw
Siddhartha Sen, Alex Slivkins

Microsoft Research, *Microsoft

Abstract

Applications and systems are constantly faced with decisions to make, often using a *policy* to pick from a set of actions based on some contextual information. We create



Ad Click Prediction: a View from the Trenches

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young,
Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov,
Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg,
Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica
Google, Inc.
mcmahan@google.com, gholt@google.com, dsculley@google.com

The story behind River

init
MaxHalford committed on Jan 24, 2019

README.md
jacobmontiel committed on May 4, 2017

Creme

scikit-multiflow

River

Three areas of focus

- 👩‍💻 User experience
- 🎯 Accuracy
- ⚡ Throughput

👩‍💻 User experience

Why choose Python?

- Easy to learn
- Good over the internet — e.g. requests, FastAPI
- Binding compiled extensions is possible
- Fast enough for many problems
- Main language for data science and ML

Dictionaries are great

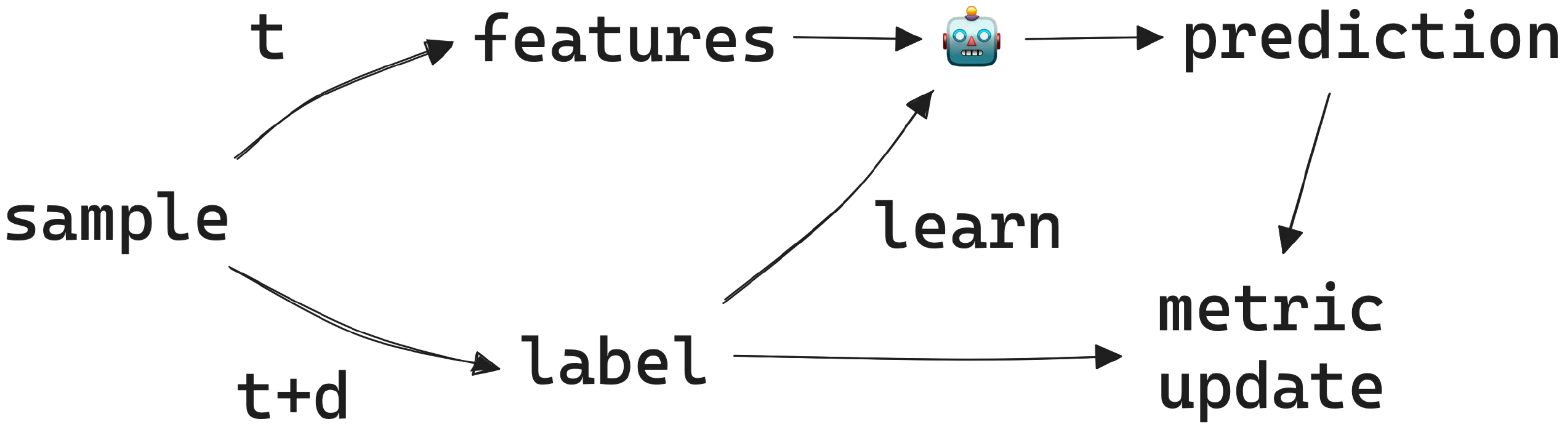
- Each feature has a name
- Naturally sparse
- Mixed types
- 1:1 equivalence with JSON
- Great support in Python

River aims to be flexible

- River caters to experimentation and production
- Inversion of control
- Users can code training loop (like PyTorch)
- High-level functions for quick experimentation

Delayed progressive validation

- Progressive validation
- Delayed progressive validation is even more realistic



Real-time feature engineering

- Behind every good model there's good features
- River is mainly for ML, not for data processing
- Real-time data manipulation is trickier

Documentation

- There is a lack of it
- Harder than writing code, believe me
- Needs to be a distributed effort
- Luckily, some people are writing books 🙏

📚

Documentation pageviews

Simple code

- River is ~48k lines of code
- There are ~3500 unit tests
- River code tries to minimise complexity
- Many modules, separation of concerns
- It works:
  - >100 unique contributors
  - Not too many bug reports

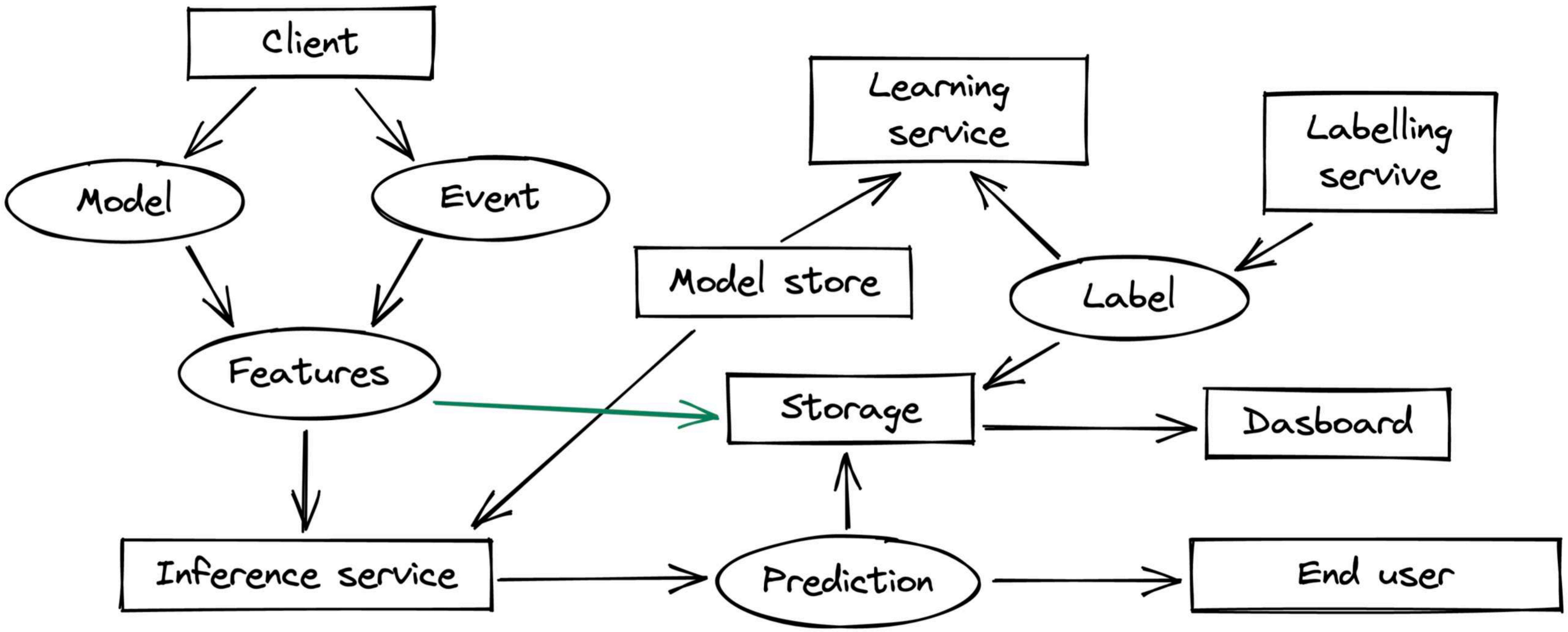
Deployment/maintenance matters

- Batch ML
  - 👍 Works well locally, easy to reason about (functional)
  - 👎 Experimentation don’t always hold in production
- Online ML
  - 👍 Immediately thinking in terms of streaming data
  - 👎 Less established patterns to draw from

Online MLOps is uncharted territory

- Many real-time requirements beyond just ML
  - Feature engineering
  - Monitoring
  - Model tuning and selection
- Companies may have some requirements in place
- Does not receive a lot of attention

We put a lot of thought into it



🎬 [See GAIA 2022 presentation]

Beaver

- Provides API endpoints to learn and predict
- Different systems can be plugged in:
   A. Task runner (Celery, Redis, ...)
   B. Message bus (Kafka, RedPanda, ...)
   C. Stream processor (Materialize, Flink, ...)

⚠️ Experimental, not meant for production

🎯 Accuracy

Is batch more accurate than online ML?

- I get this question *a lot*
- It’s possible to compare the two, but awkward
- Reproducing production conditions is paramount

Progressive validation for batch models

- Do progressive validation, without the learning step
- Pick a retraining schedule:
  A. Periodic (#samples or time-based)
  B. Triggered on performance drop
- While new model is training, keep using old model

Decision trees do well

- Reassuring: this is also true for batch
- Hoeffding trees are well established
- Mondrian forests extremely promising

Unsupervised updates during inference

- A model may have unsupervised steps
- For instance, standard scaling features
- No need for a label to do an unsupervised update
- `predict_one` not being pure confuses our users 🫨
- We removed this behaviour in River 0.19
- Can be activated via `compose.learn_during_predict`

👩🏼‍💻

Online tuning

- Not my area expertise 🤷‍♂️
- I usually advise having several models concurrently
- A meta-model does the orchestration:
    A. How to aggregate model predictions
    B. Which models are “allowed” to train
- Many approaches (expert learning, successive halving, …)
- Main issue is cost! Good models don’t need tuning

Bandits

- The current best model is the one predicting
- A subset of models are updated
- Ability to trade between exploring and exploiting

👍 Good theory and guarantees

👍 Seem to work well in practice

👎 Bandits are usually stationary

👩‍💻

There is no LightGBM equivalent

- scikit-learn is nice and all, but…
- People don’t like tuning models
- They want their model to work out of the box
- LightGBM almost always works
- All tabular Kaggle competitions use LightGBM

The same is needed for online learning!

⚡ Throughput

Throughput is a good problem to have

- Most people don’t even know about online ML
- If throughput is an issue, it means they are hooked
- Better throughput usually implies more complexity
- Not River’s main focus

Throughput can’t be ignored

- Real-time apps usually have high throughput needs
- If they didn’t, then they could just use batch learning!
- 1GB/sec seems to be a good target to reach
- 14GB/sec is the biggest workload we’ve heard of
- 🧂 Some models’ throughput is independent of data scale

Latency matters too

- One way to get throughput is to distribute computation
- Distribution involves communication (between machines)
- Communication is expensive, not good for latency
- Distribution computation is also complex (e.g. Spark, Flink)
- There are rocks to squeeze with a single machine

Models can’t afford to be complex

- There isn’t much space for fancy tricks
- Many good online models are just linear models

A. Simon Funk’s Netflix solution

B. (F)FM — see Bytedance's Monolith paper

C. LinUCB for recsys — see Vowpal Wabbit

D. Logistic regression for CTR — see Google paper

Delegate the feature engineering

- It’s likely that feature engineering is the costliest task
- You could delegate this to a stream processing engine
- Examples: Flink, Materialize, RisingWave, ksqlDB, etc.
- Python would only be used for River
- This can increase throughput, but no guarantees for latency
- There is no free lunch: the good setup depends on your app

Python isn’t ideal

- Vectorized routines are meant for batch data
- Overhead from calling C++/Rust for each sample
- High throughput environments don’t use Python

VectorDict

- Internally, many River models use dictionaries
- Python dictionaries are not designed for linear algebra
- Python's VectorDict is a C++ dictionary implementation
- Performance gain trumps overhead from calling C++
- There isn't much more we can do 🥵

Statistics implemented in Rust

Statistics	Pure Python (s)	Rust binding (s)	x times improvement
quantile	2.359	0.148	15.955
peak to peak	0.216	0.47	4.609
EWMean	0.158	0.105	1.512
EWVar	0.426	0.104	4.075
IQR	4.541	0.169	26.846
kurtosis	1.785	0.106	16.872
skewness	1.086	0.105	10.354
Rolling Quantile	323.520	77.247	4.573
Rolling IQR	636.528	76.688	9.113

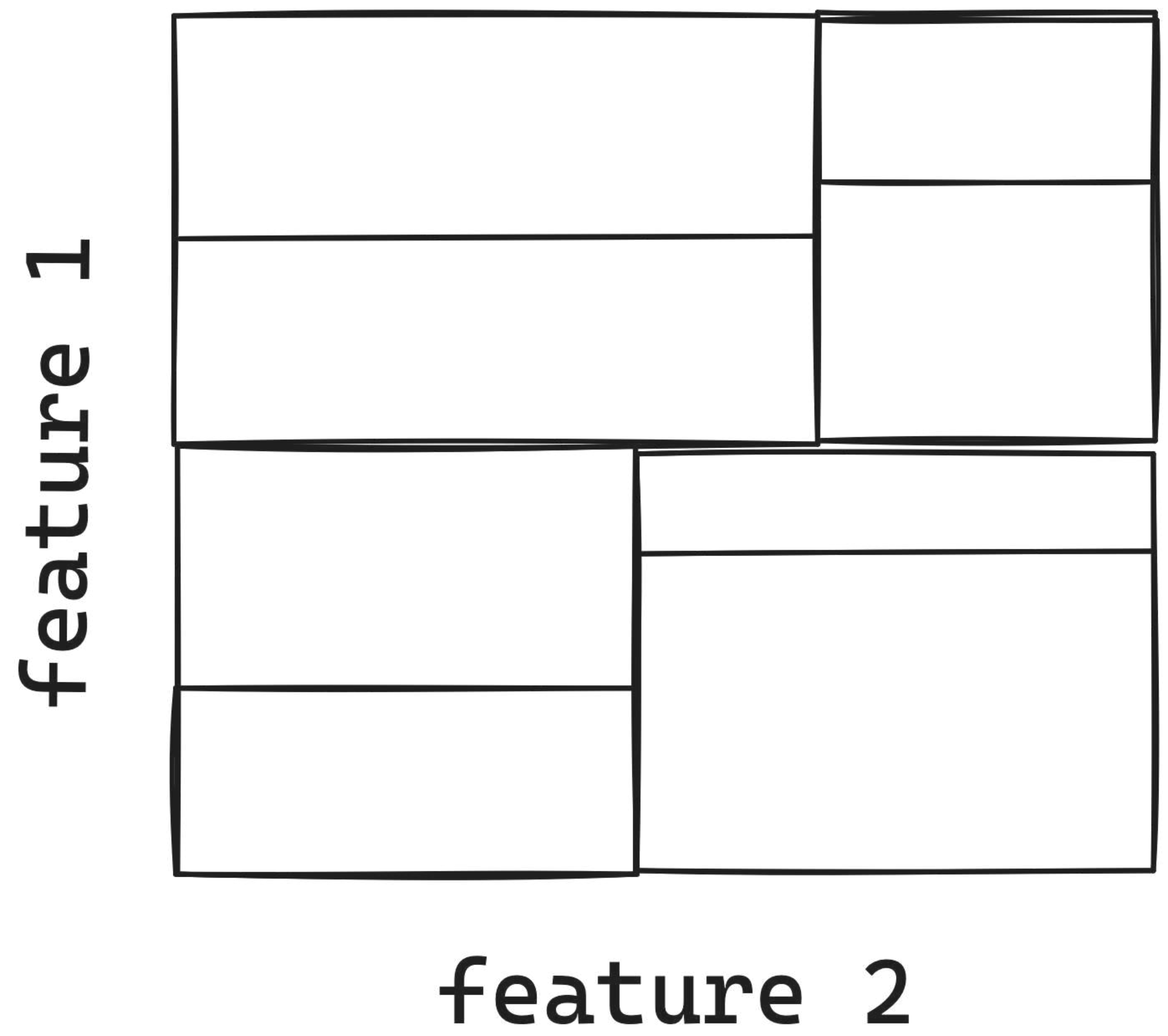
LightRiver

- Throughput objective: 1GB/sec
- Portability: available in Rust, Python, CLI
- We one or two algorithms from River for each task
- LightRiver is to River what LightGBM is to scikit-learn

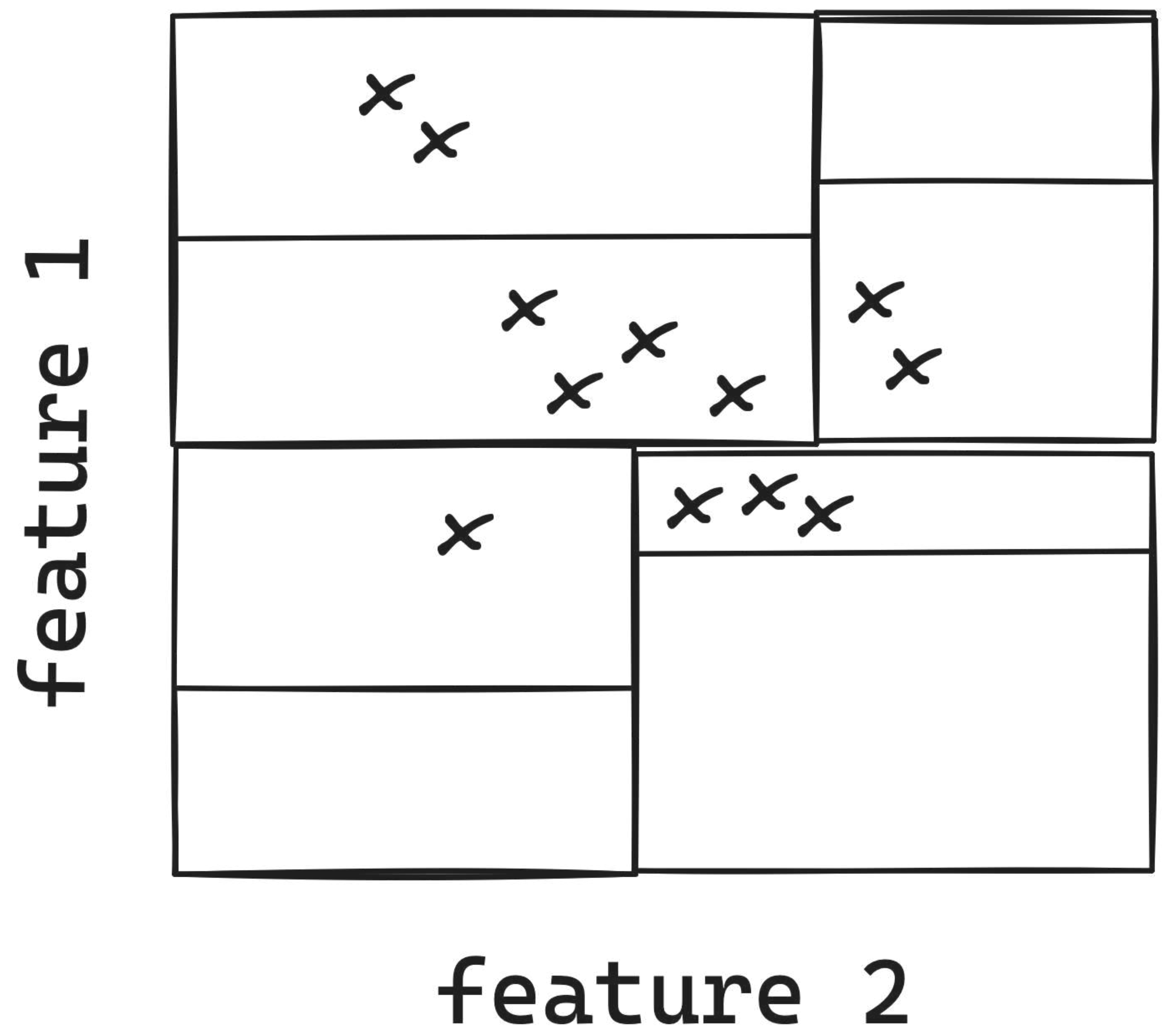
LightRiver algorithms

- Anomaly detection: half-space trees
- Regression: Mondrian forests
- Classification: Mondrian forests
- Recsys: TreeUCB, a research topic!

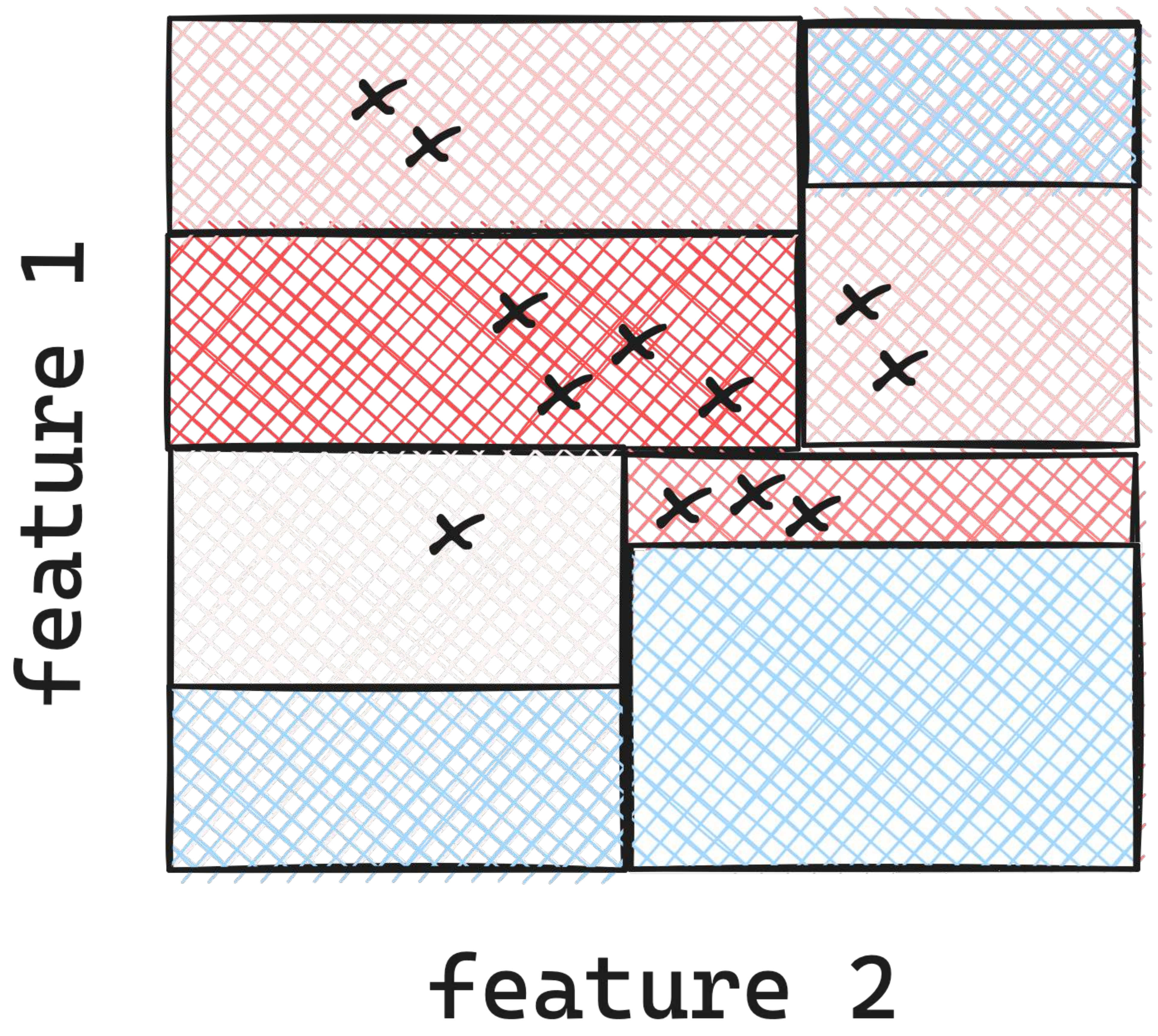
Mondrian cuts



Mondrian cuts

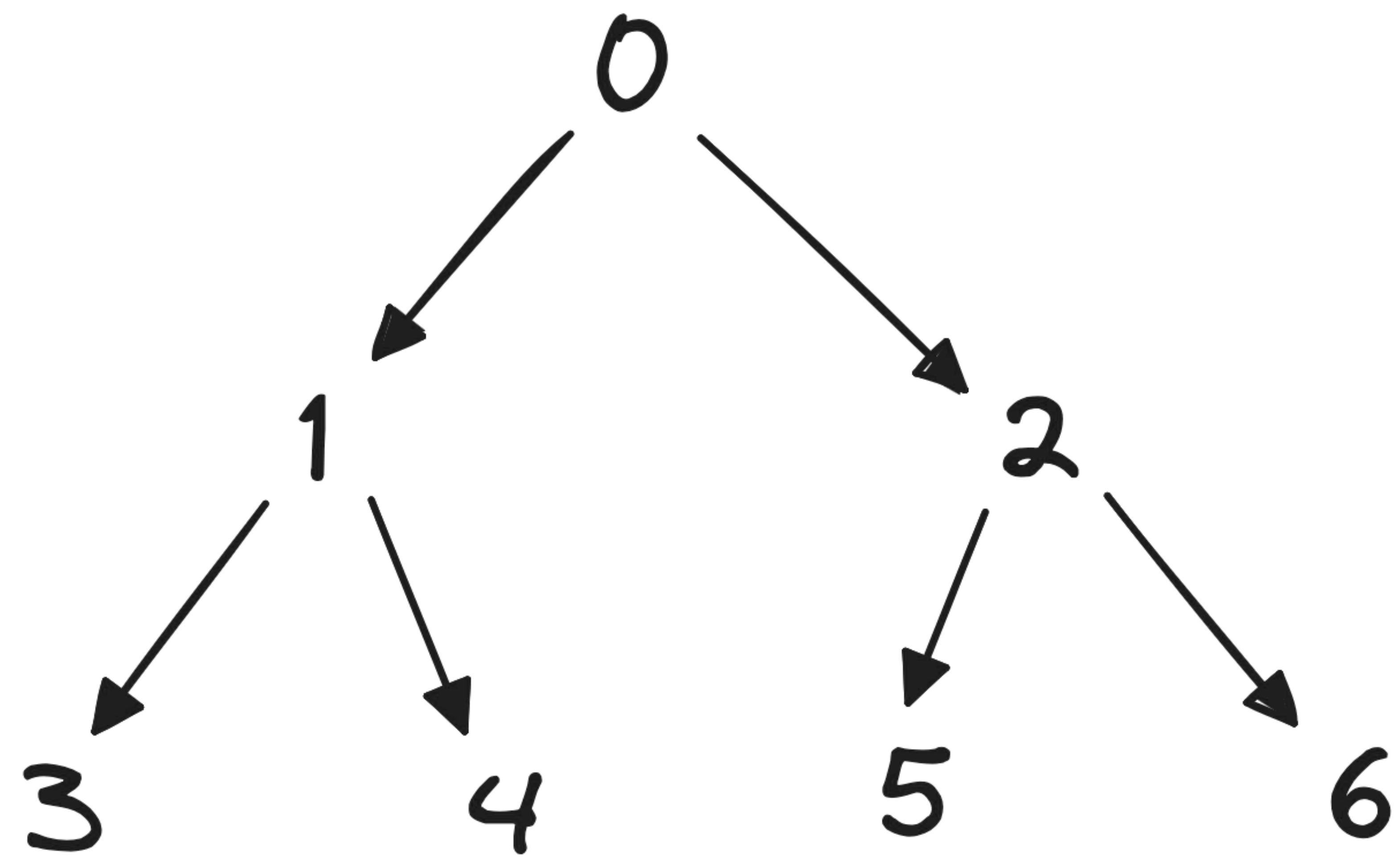


Mondrian cuts

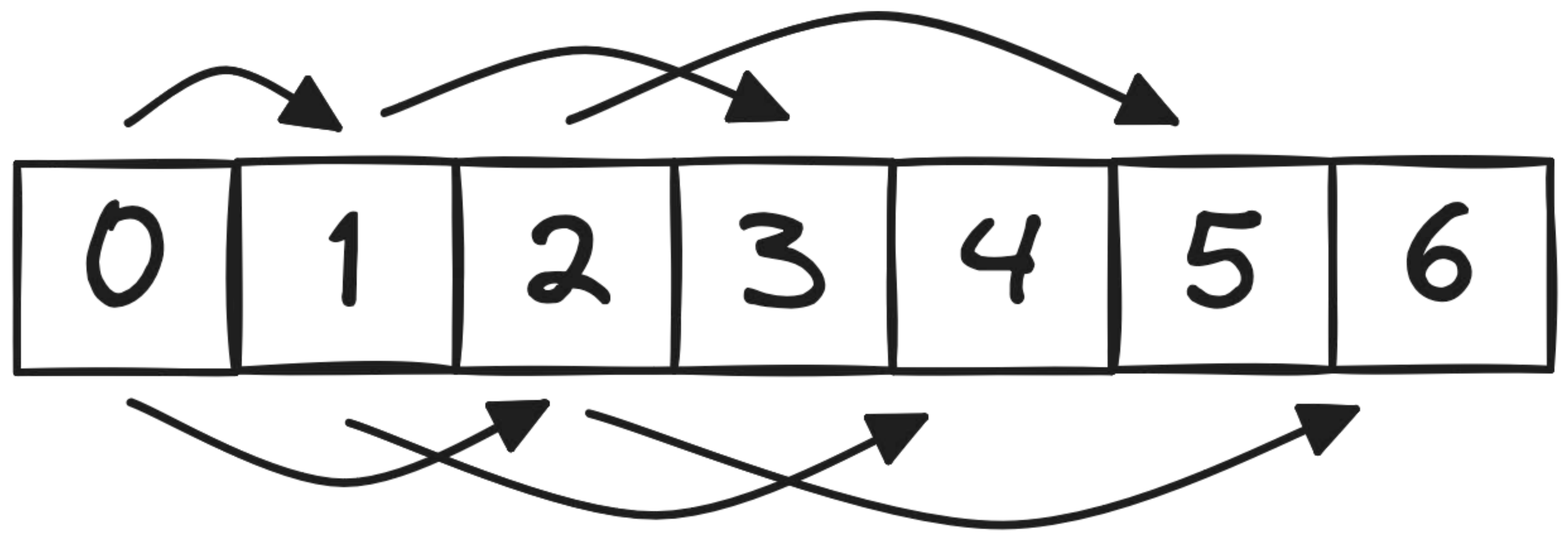


Memory layout matters

Pointer based



Array based



left = 2 x node + 1

right = 2 x node + 2

More information courtesy of Andrew Tulloch

Mondrian tree advantages

- Tree size is known beforehand: good for array layout
- Speed is not a function of #features
- Ability to trade between speed and accuracy
    A. Number of trees
    B. Tree depth
- We hope this bet pays off!

Thank You!