CATWILD

Compiler **A**utotuning for **T**PU Workloads in the **Wild**

Ignacio (Nacho) Cano

*Presenting the work of many people at **Google***

Google™

Contributors

Yu Emma Wang
Mike Burrows
Ziqiang Feng
Matheus Camargo
Chao Wang
David H. Liu
Tengyu Sun
Alexander Wertheim
Arissa Wongpanich
Christof Angermueller
Hyojun Kim
Wenqi Cao
Aleksey Orekhov
Amit Sabne
Emma Sevastian
Mehrdad Khani
Karthik Srinivasa Murthy
Berkin Ilbeyi
Subhankar Shah
Ryan Lefever
Arjun Khare
Ankit Sinha
Peter Ma
Matthew Bierbaum
Jeremiah Wilke
Emily Donahue
Sami Abu-El-Haija
Nikhil Sarda
Vineetha Govindaraj
Shobha Vasudevan
Kirill Gugaev
Idan Nachman
Jie Sun
Jose Baiocchi Paredes
Samrat Ghosh
Domagoj Babic
Zongwei Zhou
Naveen Kumar
Phitchaya Mangpo
Phothilimthana

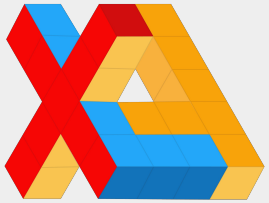
Goal

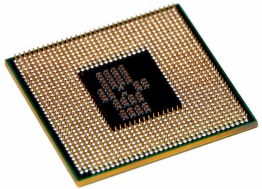
Automatic setting of **compiler knobs**, **reliably** and at **scale** for **TPU workloads** in **Google's fleet** to deliver great **out-of-the-box performance**

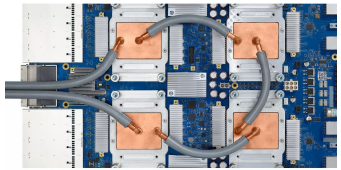
ML Compilation Stack at Google

PyTorch

...







Compiler Autotuning

- **Aids compiler** to find better optimization decisions.
- **Searches** a space of configurations of a program.
- Selects the **best configuration** according to a performance metric.

How can we use compiler autotuning in Google's fleet graphs?

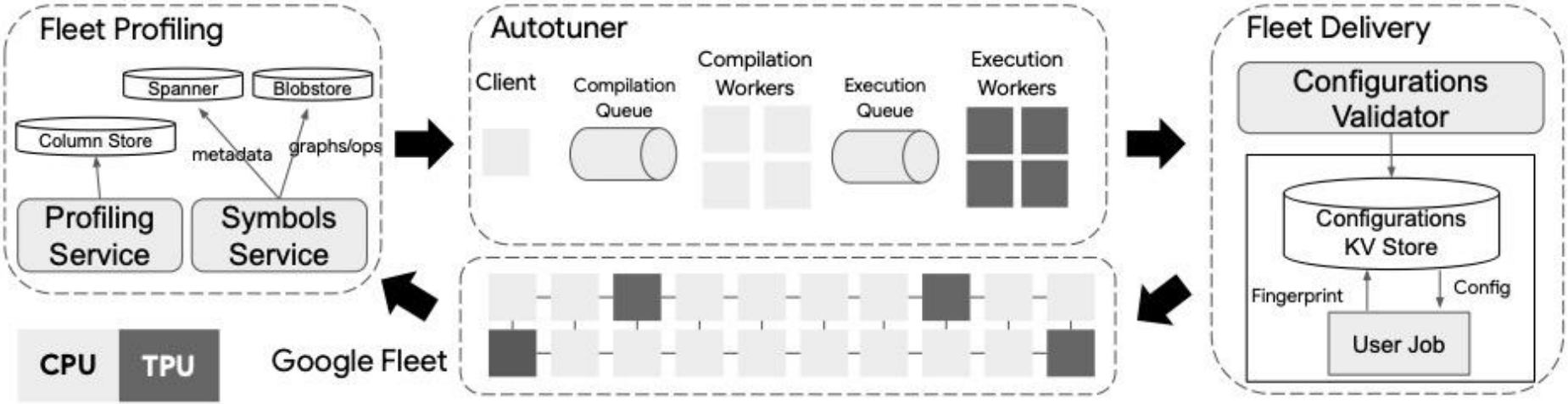
CATWILD

System that automatically optimizes ML jobs in Google's TPU fleet using compiler autotuning techniques

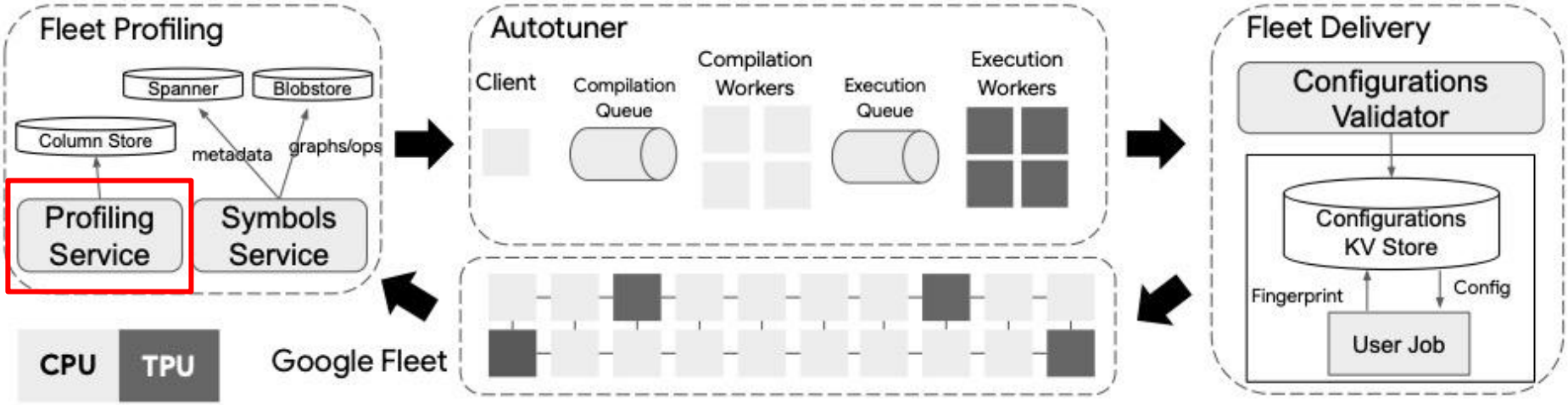
Deployment Challenges

- **Dynamicity** (20k+ experiments daily)
- **Heterogeneity** (TPU versions v4, v5p, v5e, v6e, tpu7x, ...)
- **Huge graphs** (500k+ nodes)
- **Large TPU requirements** (10k+ chip graphs)
- **Many compiler versions** (main branch @ latest version)
- ...

CATWILD Overview

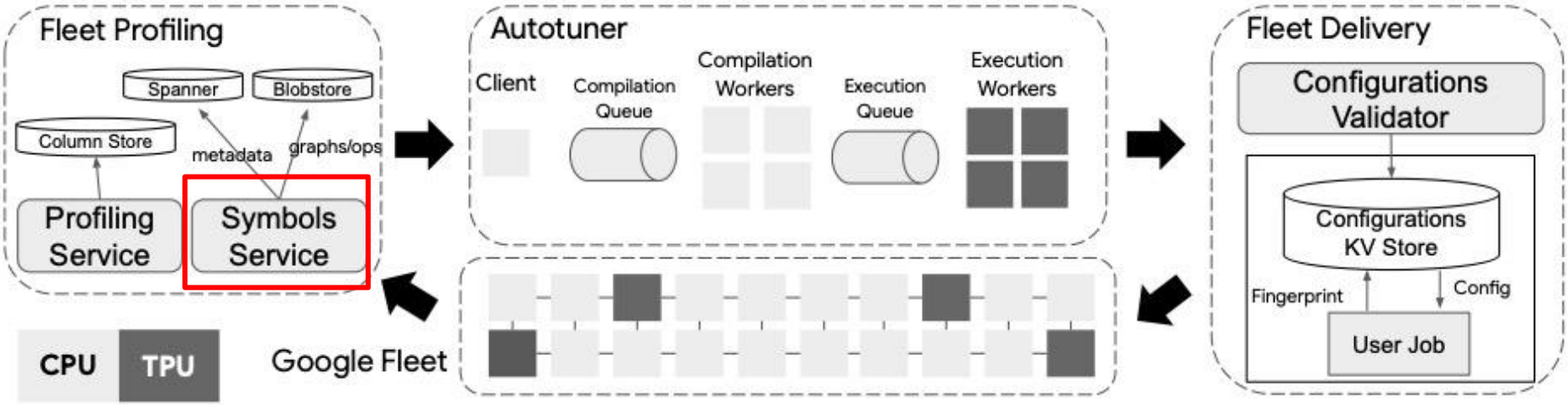


CATWILD Overview



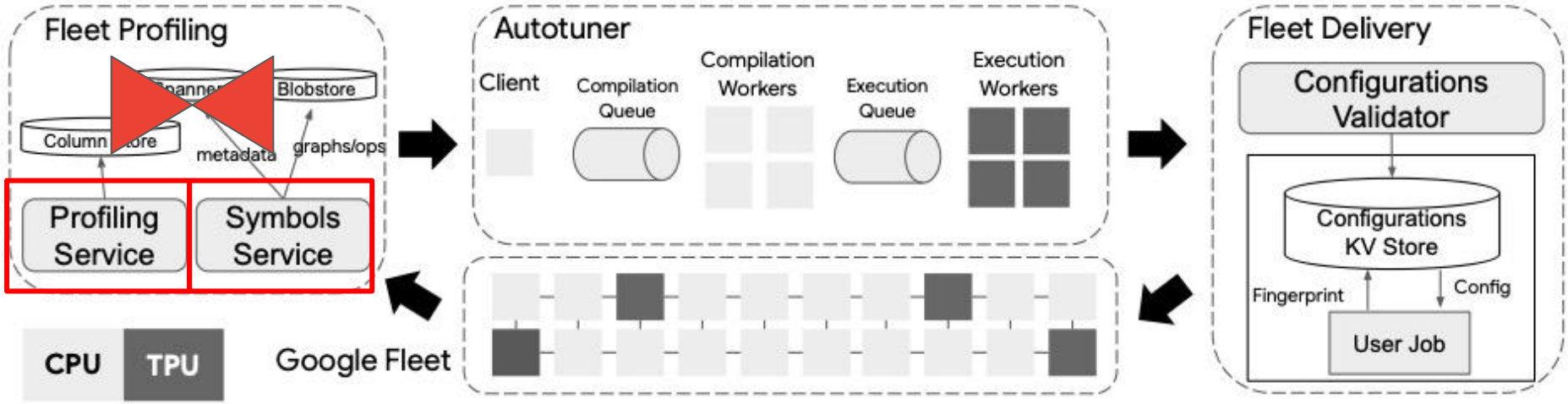
Continuous profiling of TPU jobs

CATWILD Overview



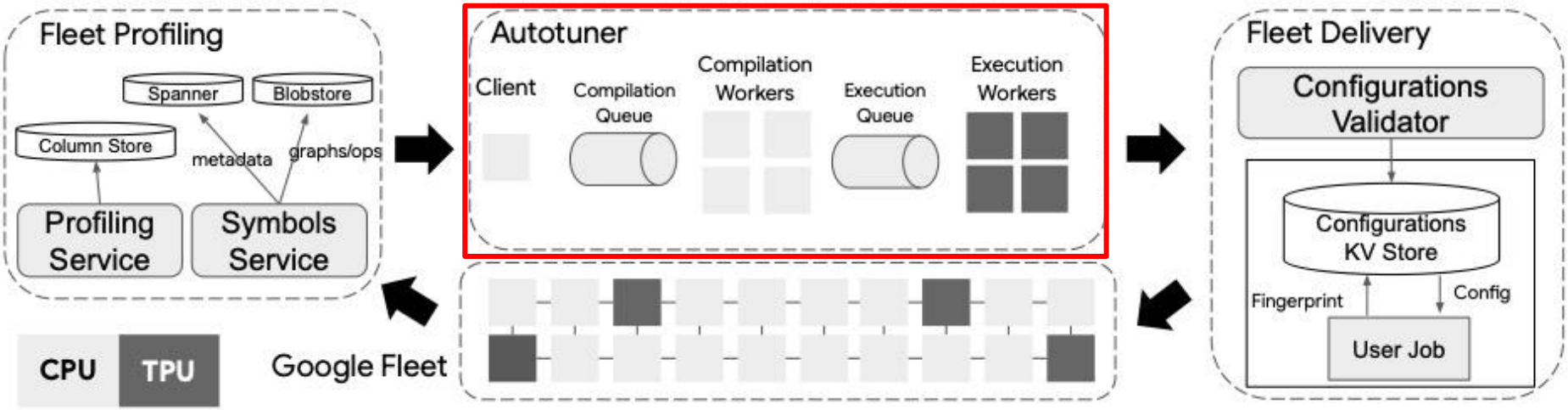
Centralized storage for graphs/ops

CATWILD Overview



Join profiling and symbols data → rank graphs

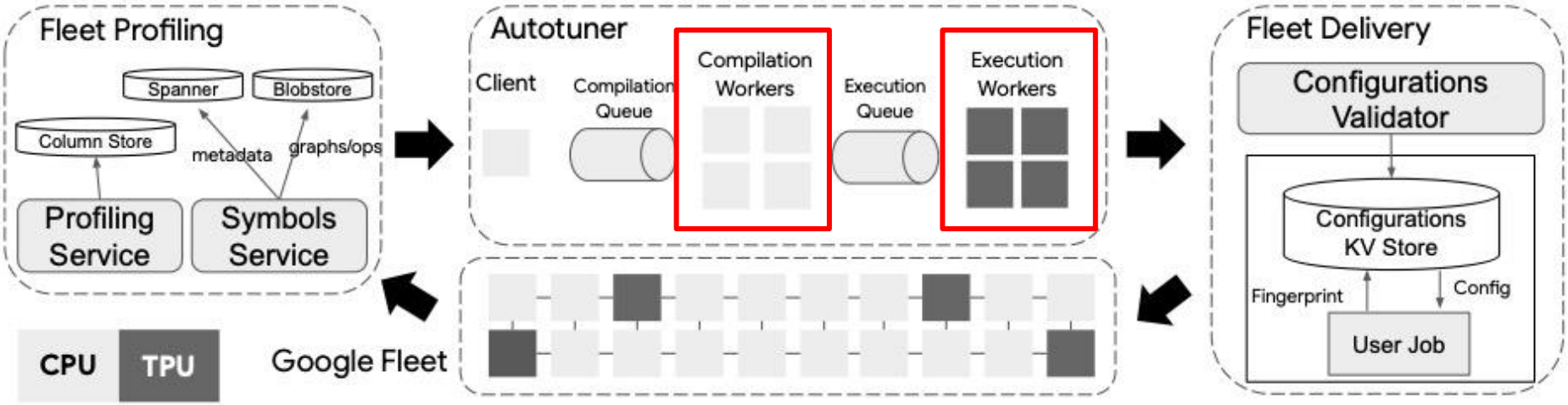
CATWILD Overview



Based on XTAT[1]: graph and op-level optimizations

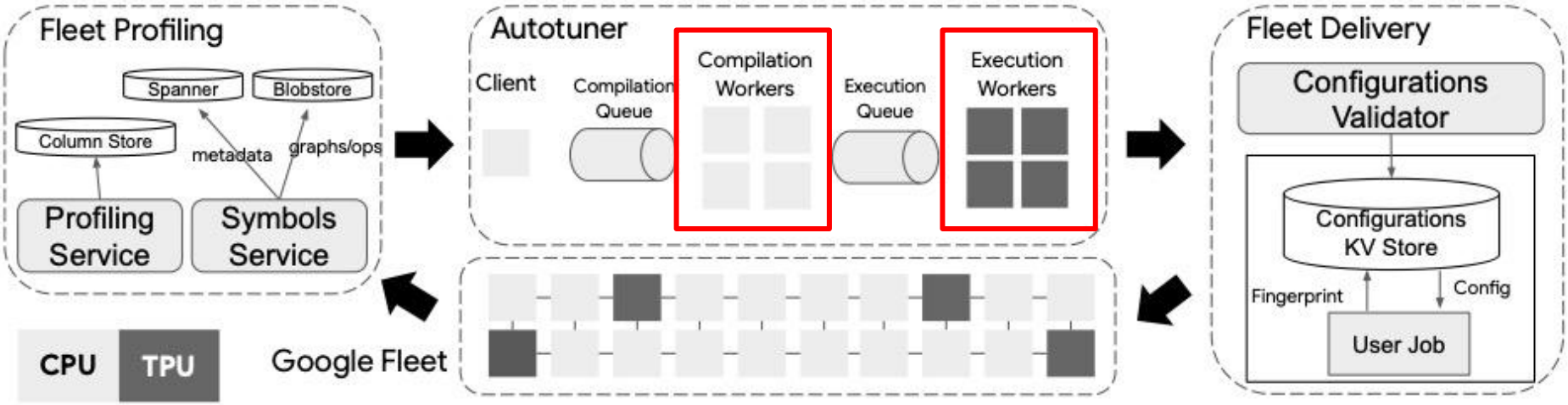
[1] *Phothilimthana et al., A Flexible Approach to Autotuning Multi-Pass Machine Learning Compilers, PACT 2021.*

CATWILD Overview



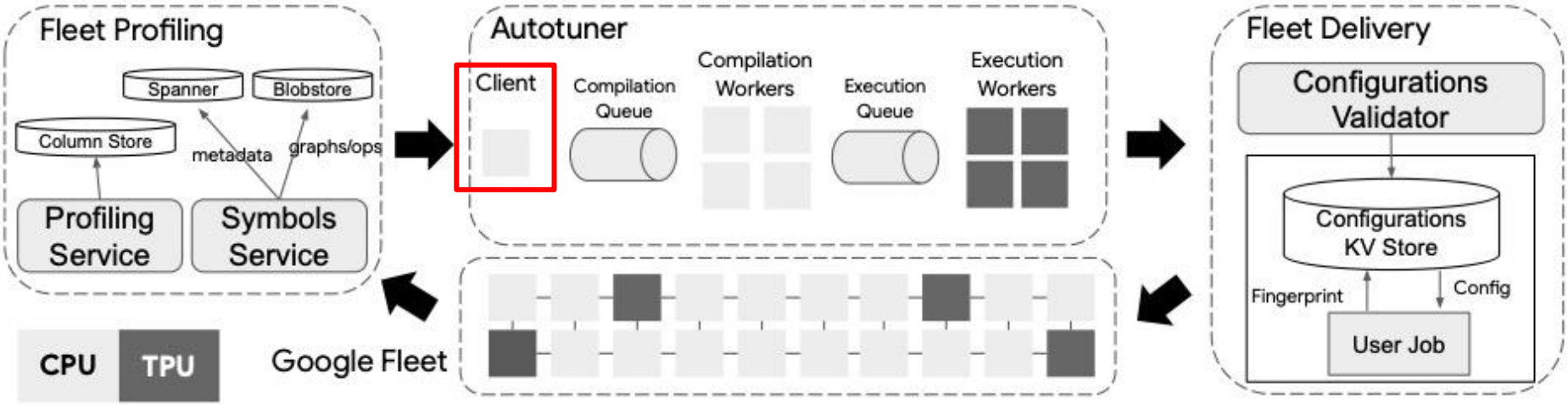
Scalable disaggregated architecture:
compilation (CPU) and execution (TPU) workers

CATWILD Overview



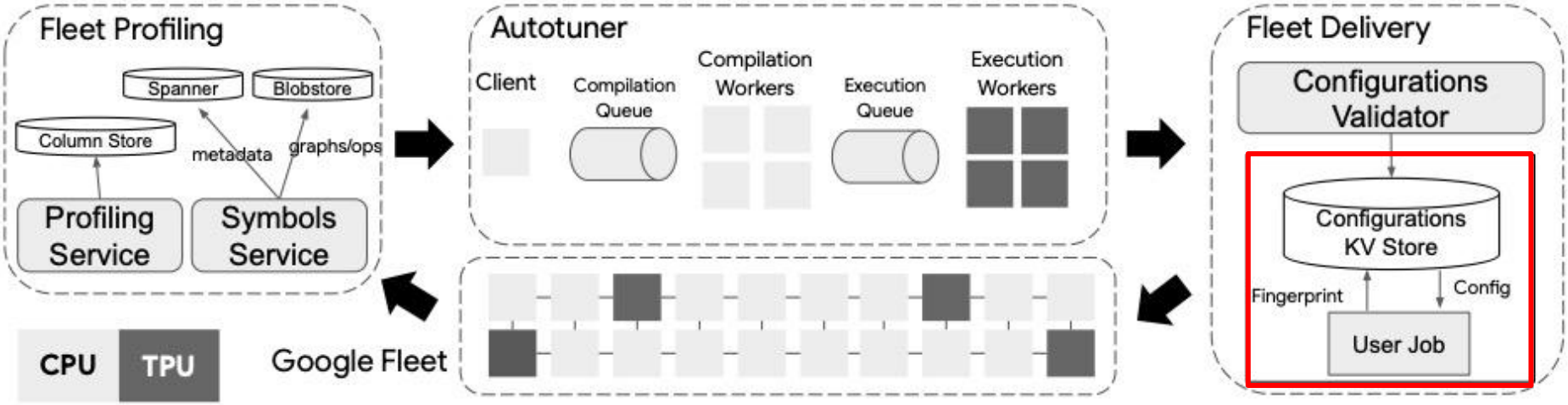
Execute modified graph on 1 TPU
Simulate communication times with ML cost models

CATWILD Overview



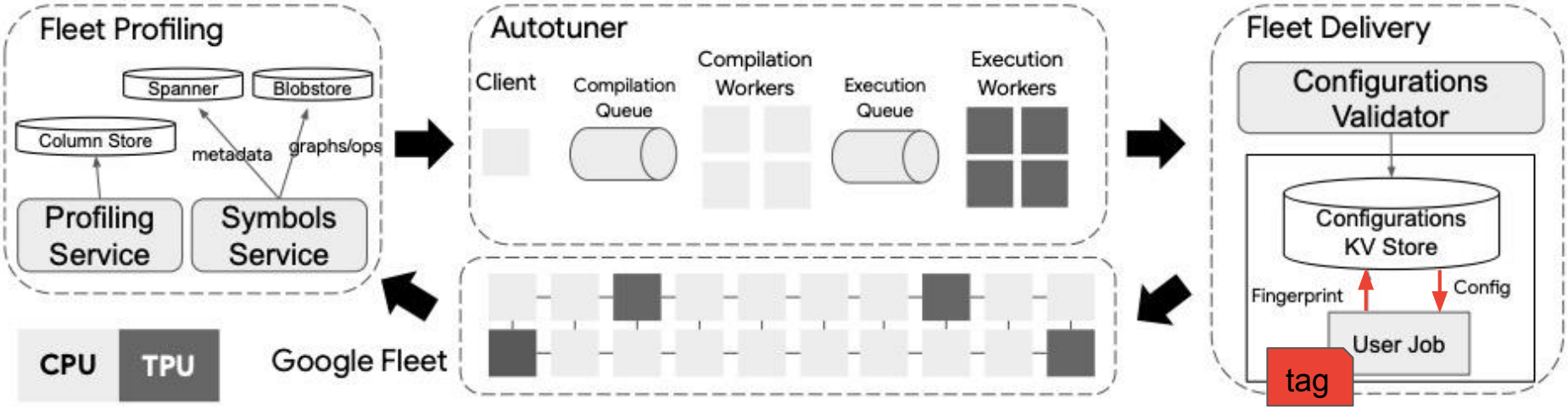
Configurable: optimizations, search strategies, search spaces, TPU versions, ...

CATWILD Overview



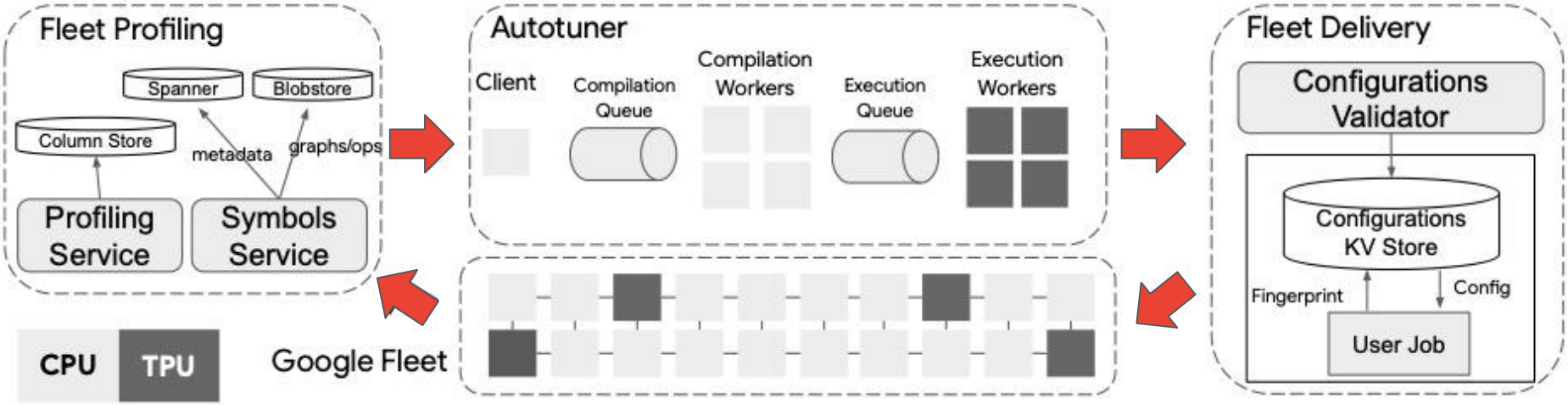
KV Store checked in and embedded in user binaries → reproducibility

CATWILD Overview



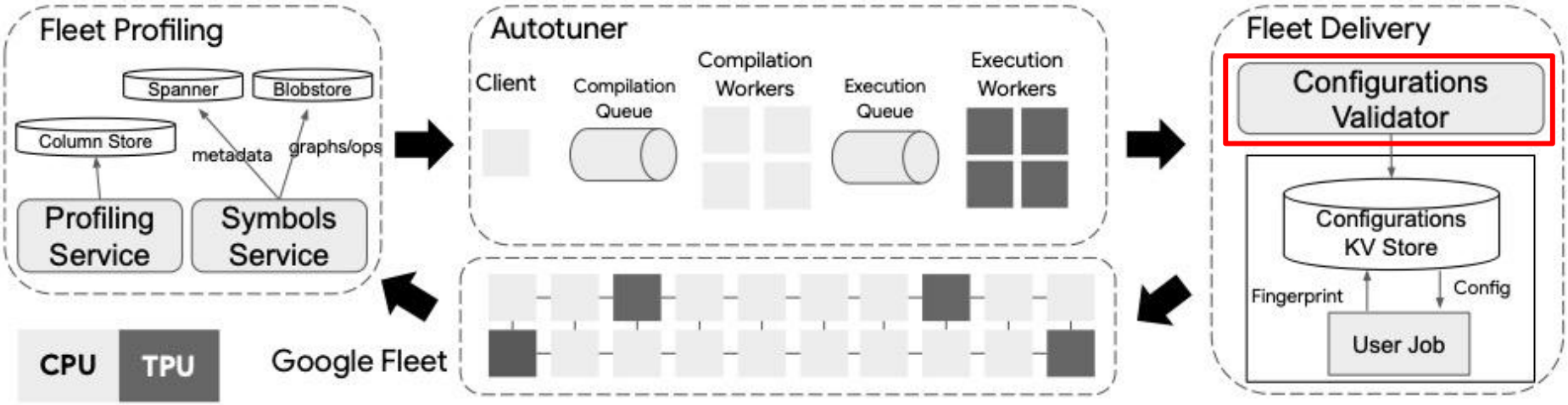
KV Store lookup upon compilation
Add metadata to monitor adoption

CATWILD Overview



Feedback-directed optimization loop at fleet scale!

CATWILD Overview



Eventual-consistent background process to keep configs up-to-date

Deployment Challenges

- Dynamicity → Continuous Loop
- Heterogeneity → Configurable
- Huge Graphs → Offline Autotuning
- Large TPU requirements → Hybrid Simulator
- Many compiler versions → Background Checker
- ...

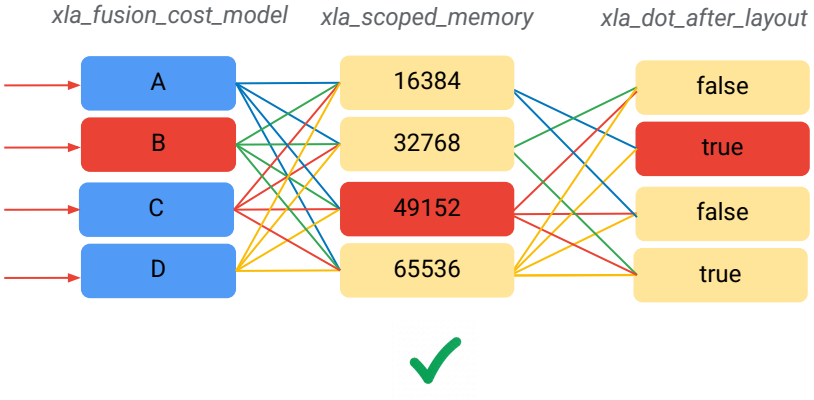
Impact

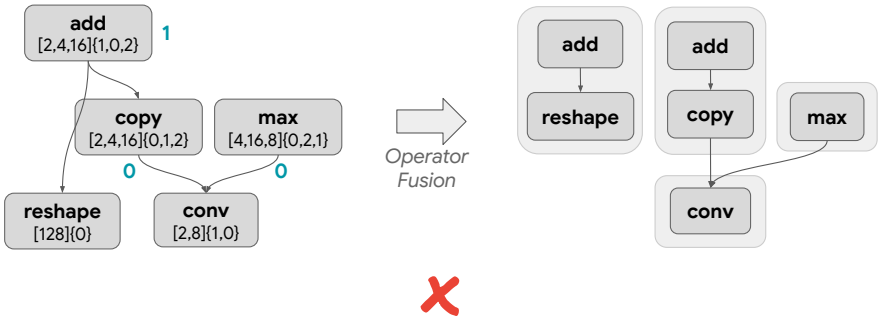
- **5 years in production!**
- **Tuning ~70% of top training workloads**
- **Currently deployed two tuners**
  - Graph-level **flag** tuner
  - Op-level **tile** tuner
- **Average speedups ~= 5% - 15%**
- **~100k autotuned configurations used daily**
- **Savings / Cost = 8x - 30x**

More detailed results in the paper

Coarse vs. Fine-Grained Decisions

- Tuning high-level heuristics (**flags**) yielded **better** fleet outcomes and **less complexity** than fine-grained op-level decisions





“Meta-tuner” ⟶ multiple optimizations

Streamline Research to Production!

Leverage CATWILD to improve core XLA

- Analysis of **tuning results** to **improve** compiler **heuristics / cost models**

before

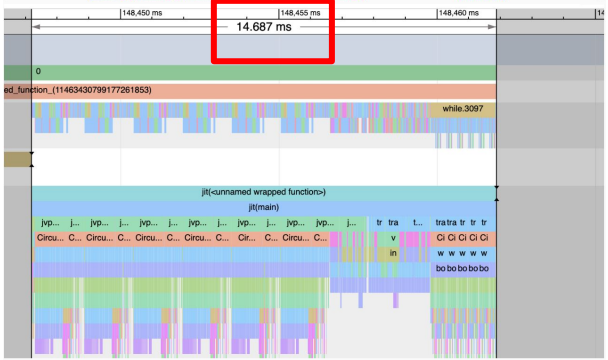
after

Avoid a particular fusion if no high HBM pressure

before

Better prefetching

after



Great out-of-the-box performance!

Conclusions

CATWILD: system that automatically optimizes ML jobs in Google's TPU fleet using compiler autotuning techniques

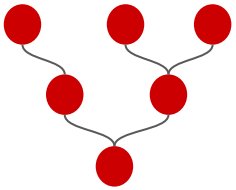
- Fleet profiling and collection of TPU graphs/ops
- Disaggregated autotuner architecture off the user jobs critical path
- Safe delivery of tuned configurations back into the fleet
- Continuous monitoring and refresh of tuned configs

Thanks!

Paper link

Backup Slides

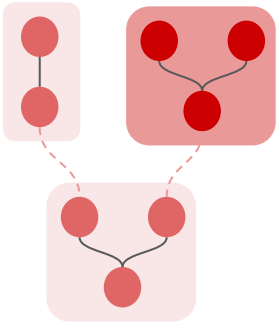
XTAT: XLA TPU Autotuner



tensor computation graph

Graph-Level Optimizations

Algebraic simplification,
Dot conversion,
Layout assignment,
Rematerialization,
Operator scheduling,
Memory assignment,
Phase ordering,
Parameters (flags)



kernels / subgraphs / ops

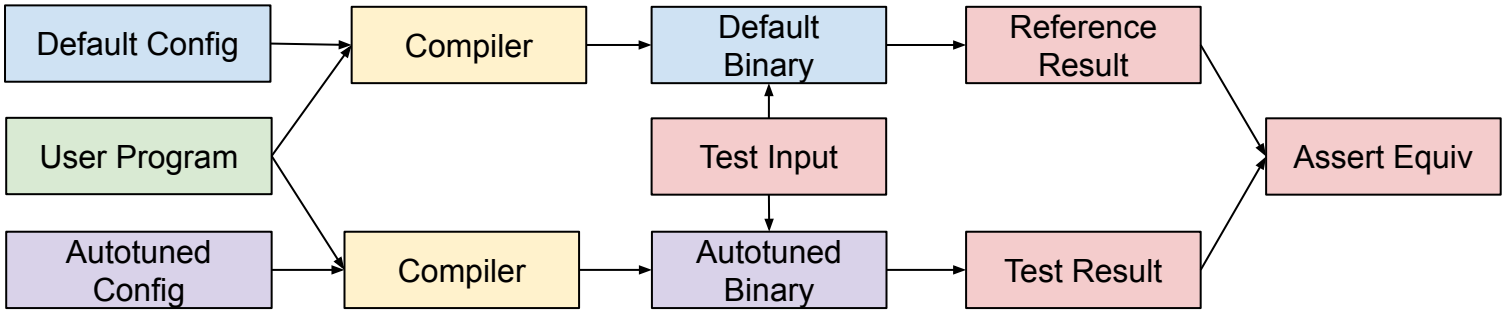
Op-Level HW Lowering

Loop Tiling,
Loop ordering/unrolling,
Heuristics parameters (flags),
Overlapping data-transfer & compute

Ref: Phothilimthana et al., A Flexible Approach to Autotuning Multi-Pass Machine Learning Compilers, PACT 2021.

CATWILD for XLA Correctness

- **Surface** latent compiler **bugs**



Fleetwide fuzzing system
Testing daily the latest compiler