Scaler

OSFF NY

Travis Liles

Ritesh Bansal

Jason Kincl

Boris Litvin

November 2026

The Program Parallelization Problem

Parallelizing workloads to the cloud is a **complex**, **time-consuming** process that requires specialized expertise.

Needs knowledge of parallel algorithms, orchestration, images, containers, networking, load balancing, etc.

Steps to Parallelize a Program

Break a program into tasks:

Data parallelization: Function level (e.g. MapReduce).

Program level parallelization: Parallelize the call-graph (DAG Structure)

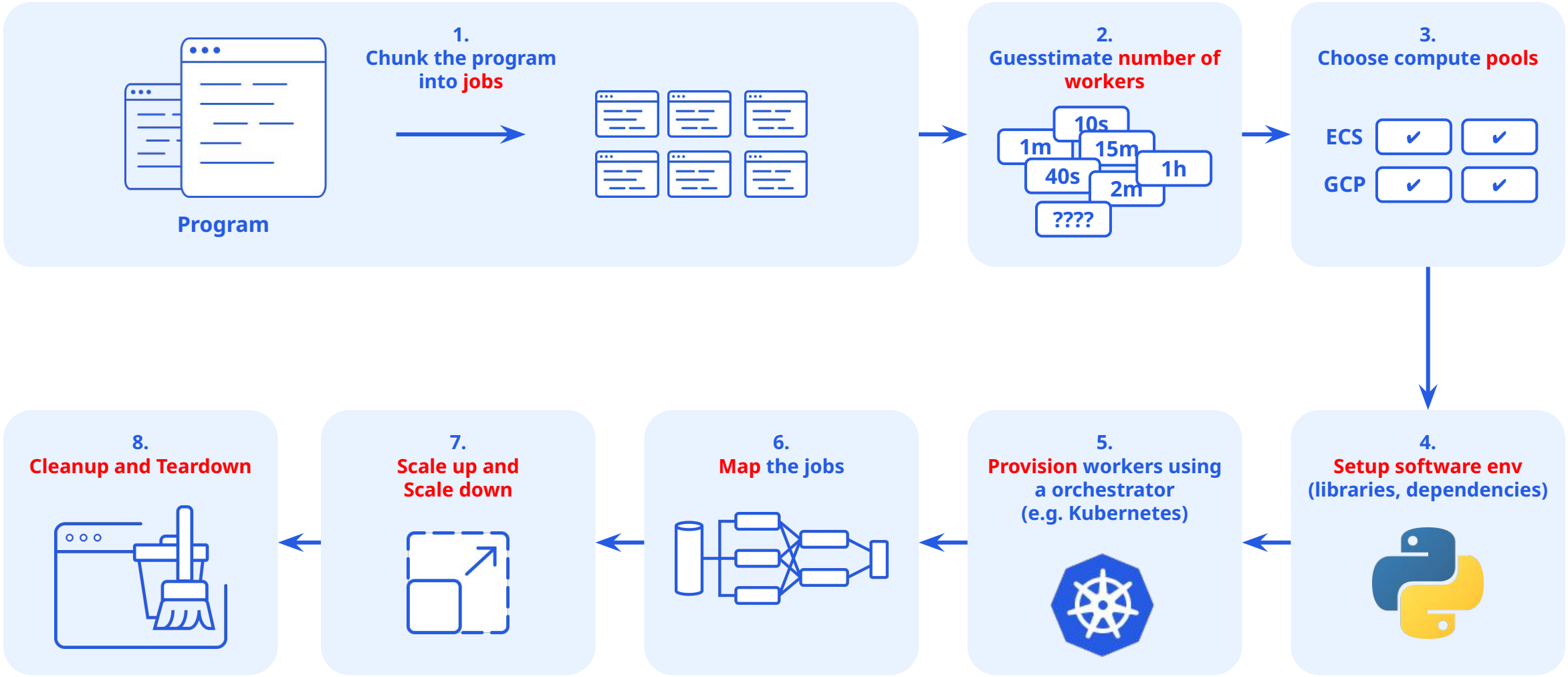


Map the tasks onto
cores/machines/instances/grids/clouds

Steps to Parallelize a Program

1. Chunk the program into **jobs**
2. Guesstimate the **workers** (cores) required to run the jobs - but APIs like AWS Batch might auto-provision.
3. Decide which compute **pools** to provision workers to run the jobs - Symphony, AWS EC2, OpenShift, etc.
4. Setup the **software** env (Libraries, Dependencies, etc.) for compute pools -
5. **Provision** workers on the pools using a **orchestrator** (e.g. EC2, Kubernetes, ..)
6. **Map** the jobs to the workers in order.
7. **Scaleup** and **Scaledown** as needed
8. Cleanup and **Teardown**

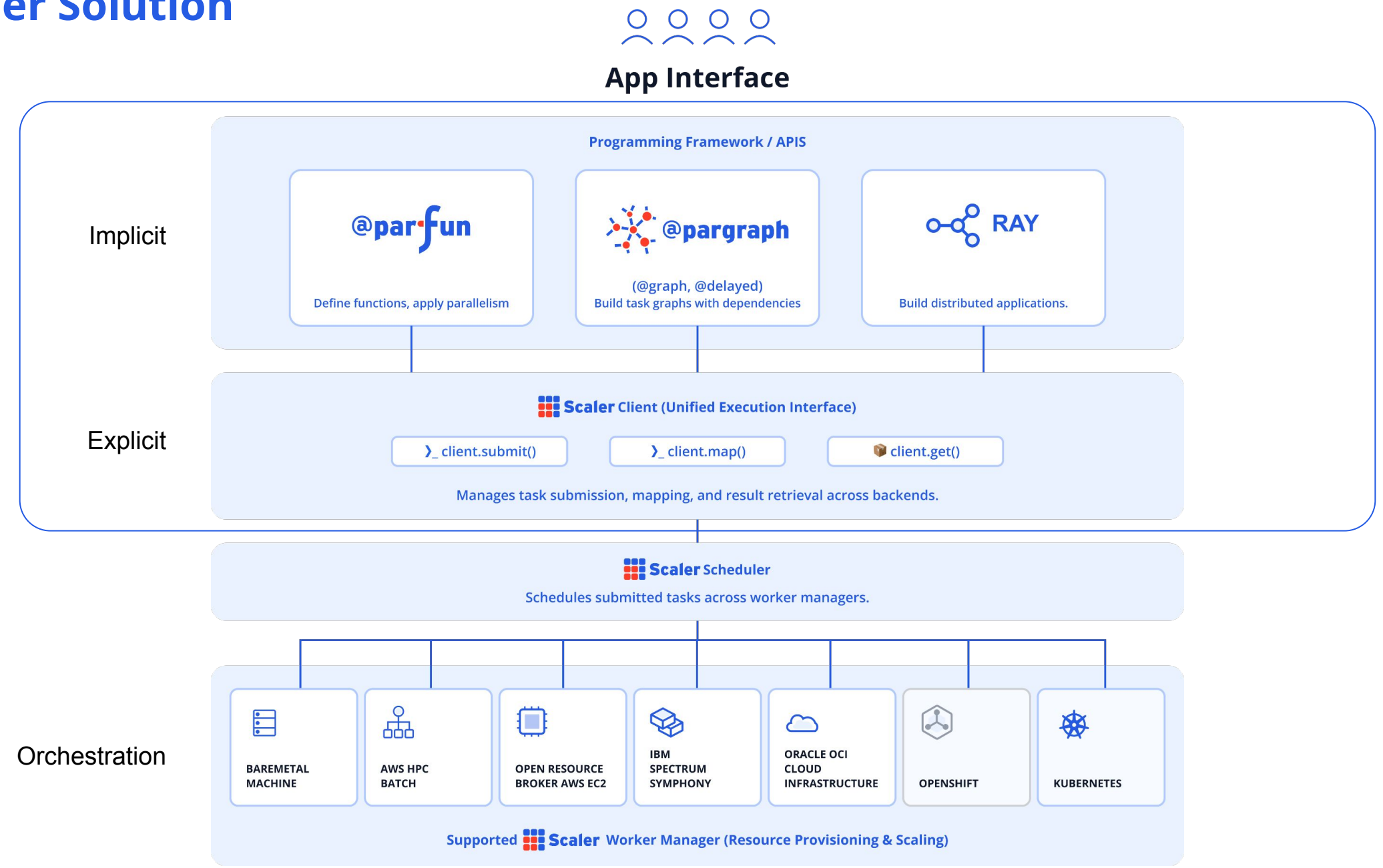
HPC: Parallelizing a program



Solution

Parallelization should be **implicit**, **automatic**, and **self-scaling**, with details like images and orchestration hidden from view.

Scaler Solution



In this talk we are excited to showcase new partnerships and developments:



Worker Managers:

IBM Symphony
AWS EC2-ORB
AWS Batch
OCI (New) - Demo
K8s (New)

Policy Engine:

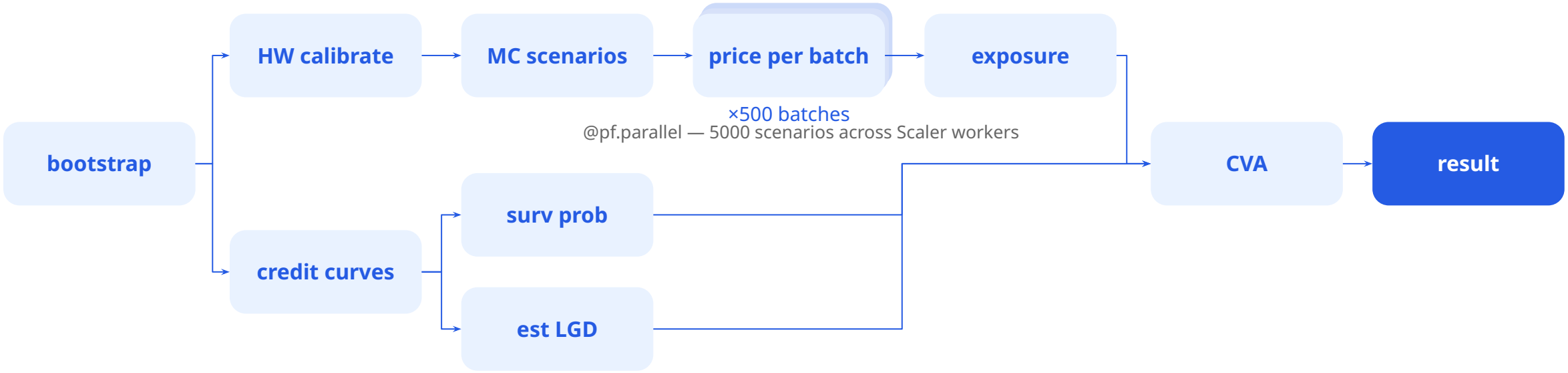
Waterfall - Seamless Routing from On-prem (local) to Symphony to AWS to OCI and K8s

Economic Policy Demo:

Cheapest region to host

Program Structure#

pargraph + parfun



HW calibrate // credit curves — pargraph resolves DAG dependencies automatically

native sequential execution



Demos

Worker Managers:

Oracle

K8s

Policy Engine:

Waterfall
On-prem (local) to Symphony to AWS to OCI and K8s

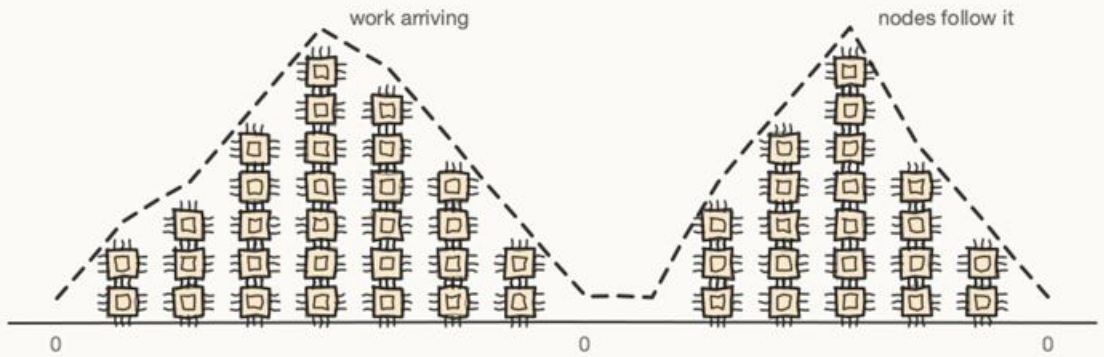
Lowest Cost:

~~OpenGRAM: An open standard for distributed resource scheduling~~

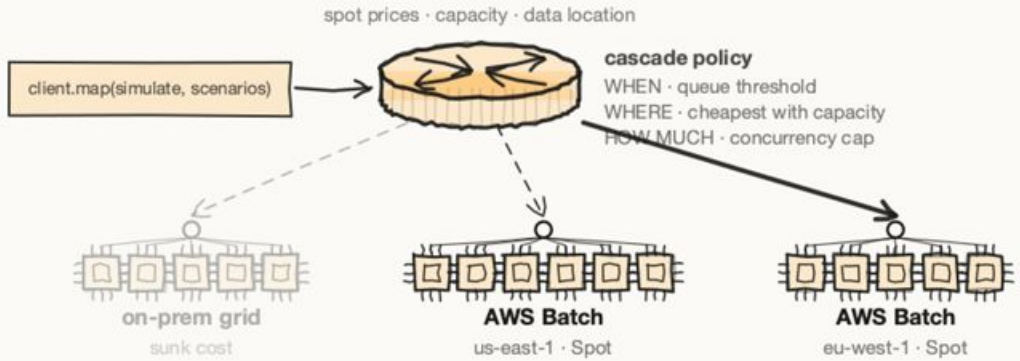
Intelligent Compute

$$\text{Minimize: } \frac{\text{Price}}{\text{Performance}} = f\ (\text{Elasticity, Utilization, Input Price, Performance})$$

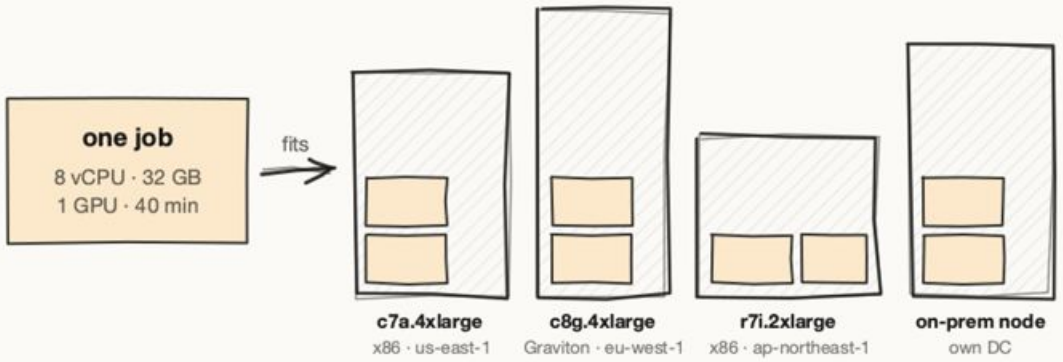
Full elasticity



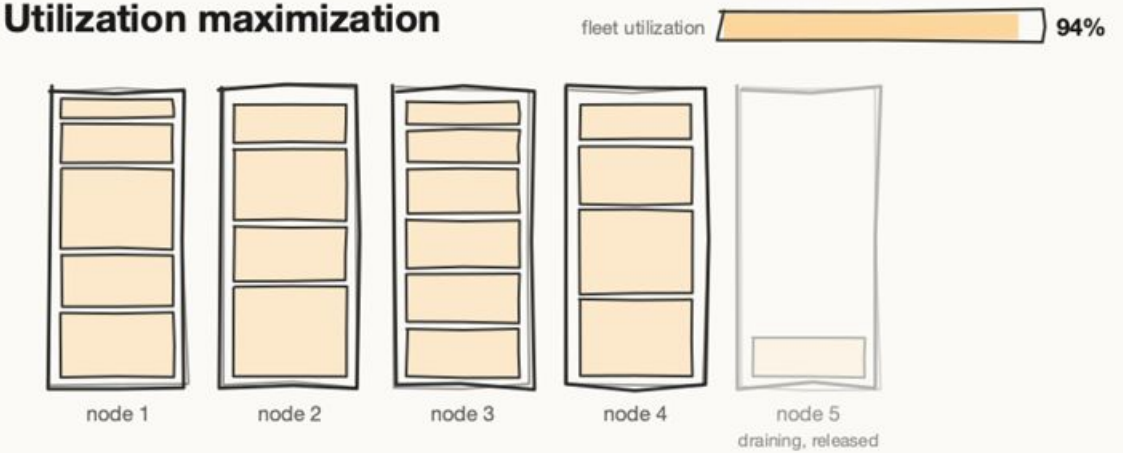
Intelligent job routing: price and availability



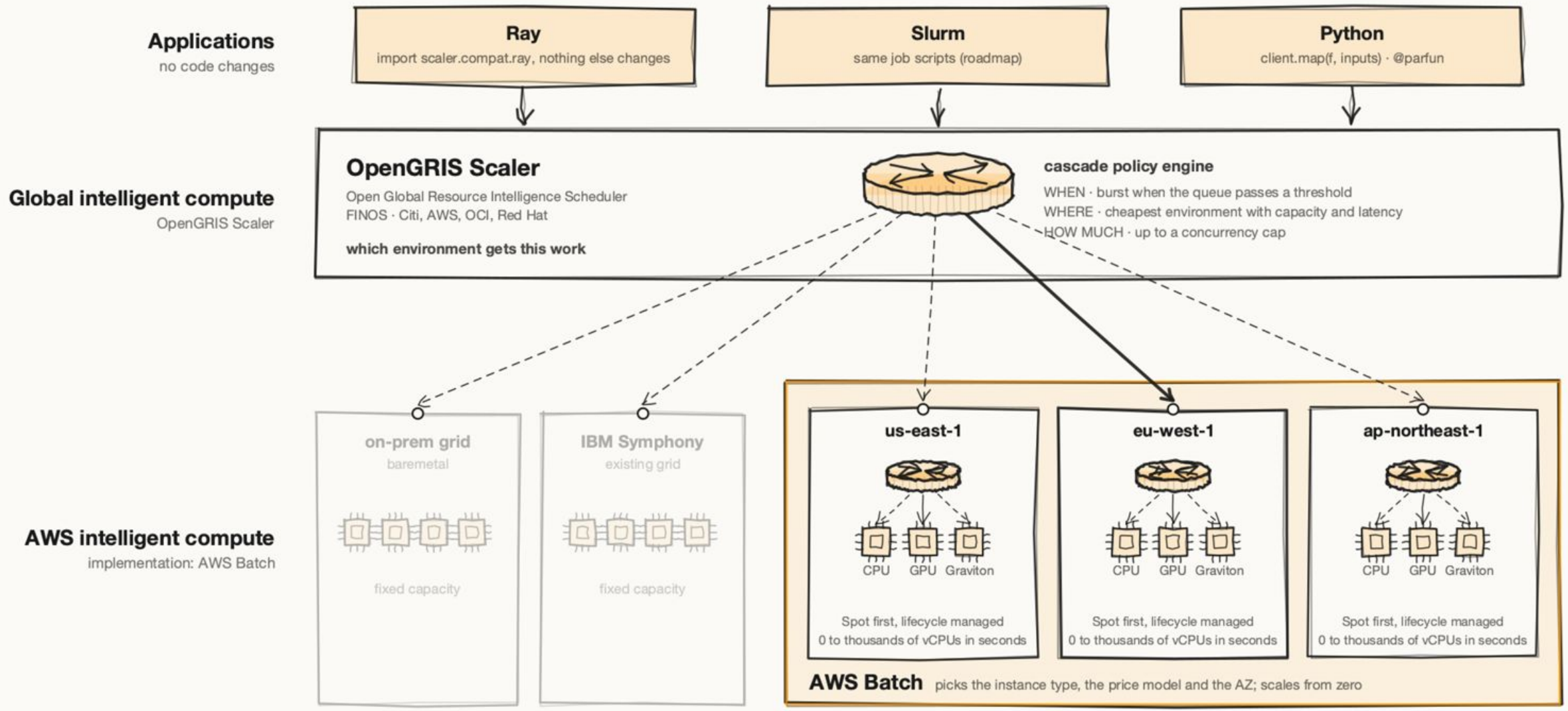
Intelligent job routing: fungible compute



Utilization maximization



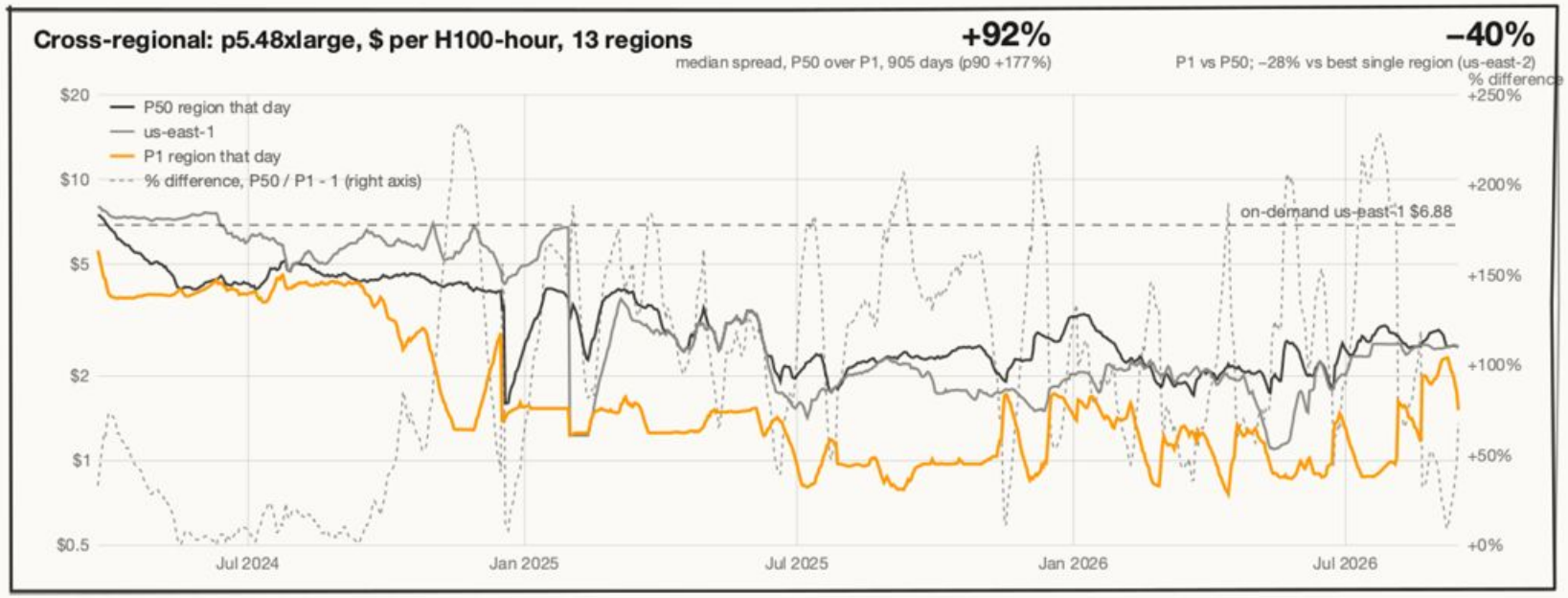
Intelligent Compute: OpenGRIS and AWS Batch



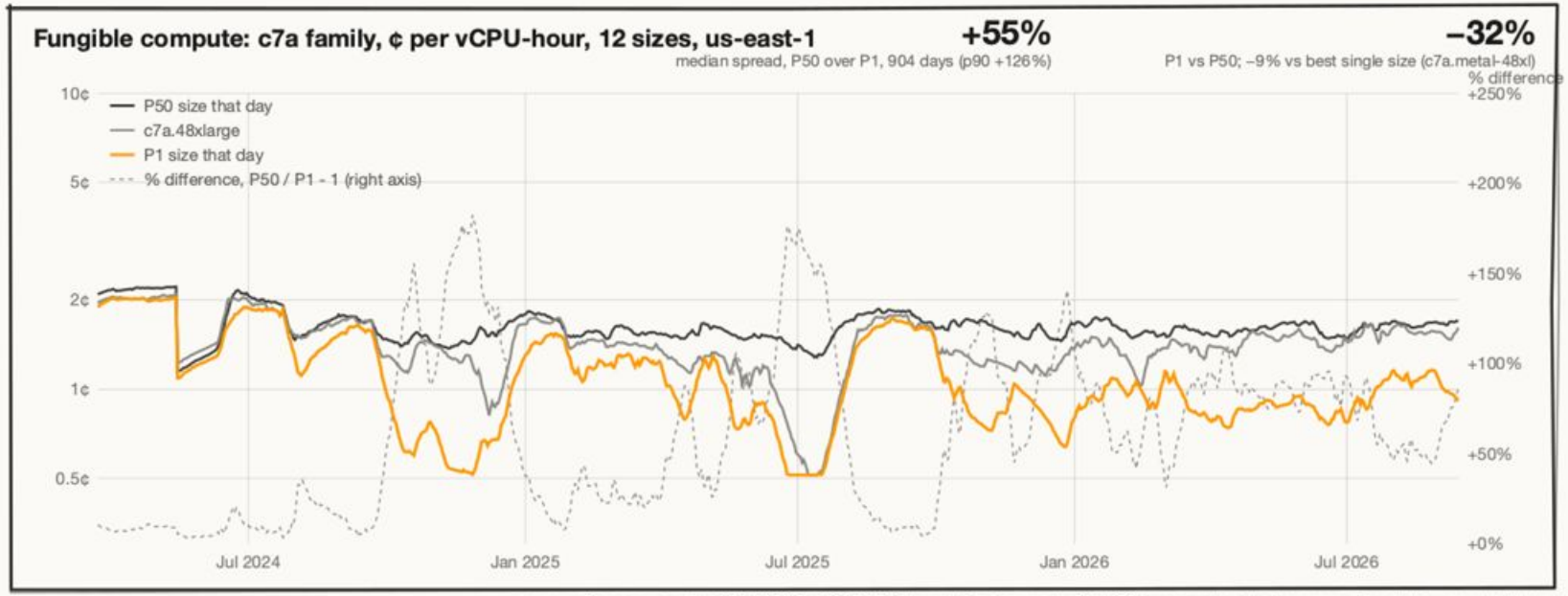
6 lines of config to add a region · 60-90% Spot savings · 0 to thousands of vCPUs in seconds

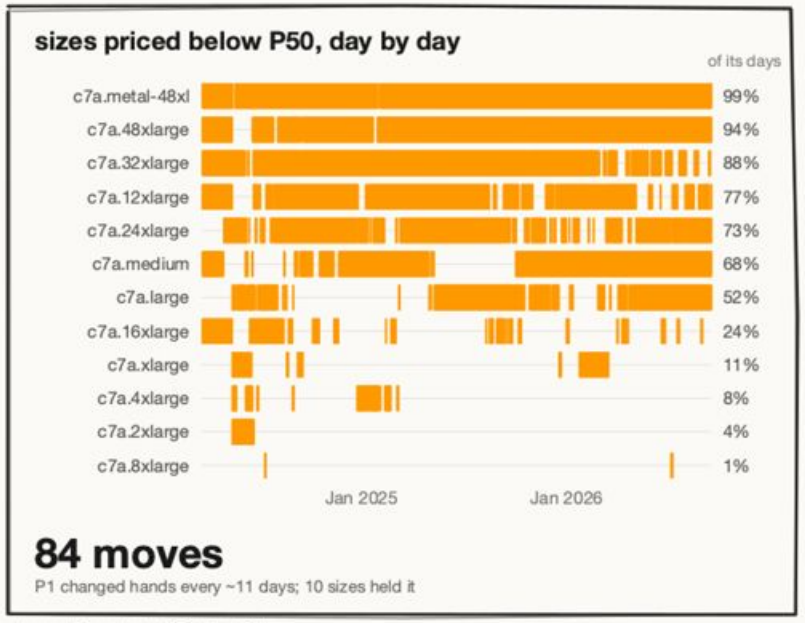
the researcher's call is identical wherever the work lands

Spot Price Spread: across regions, across sizes









source: ec2_spot_price_hist, 2024-03-23 to 2026-09-02; lowest AZ price per day; P1 / P50 across regions (top) or across sizes within us-east-1 (bottom); on-demand from the AWS Pricing API

https://github.com/aws-samples/quant-research

