Optimizing Computation & Data Movement for Fresher Features

Sarah Wooders

PhD Student @ UC Berkeley Sky Computing (prev. RISELab)

FEATURE STORE SUMMIT 2022

Organized by HOPSWORKS

What causes features (or data) to be stale?

1. Computational Bottlenecks

   Problem: New data has not be processed to be reflected in feature values

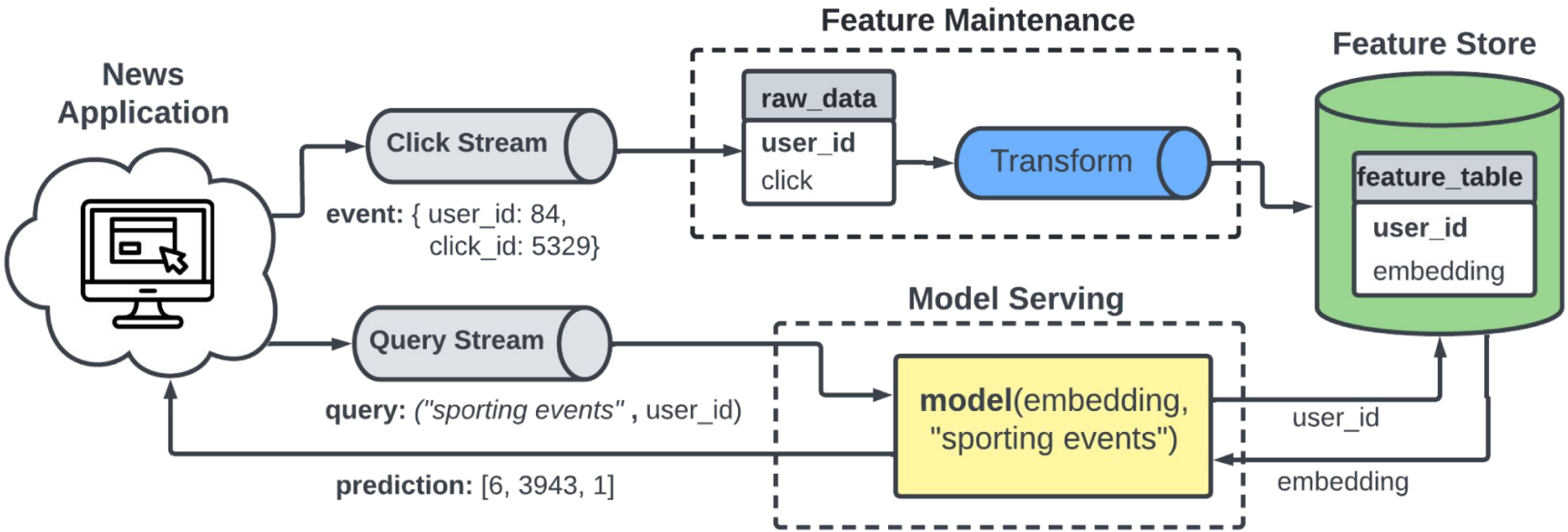
   ralf: Optimizing *computation* for fresher features

2. Data Movement Bottlenecks

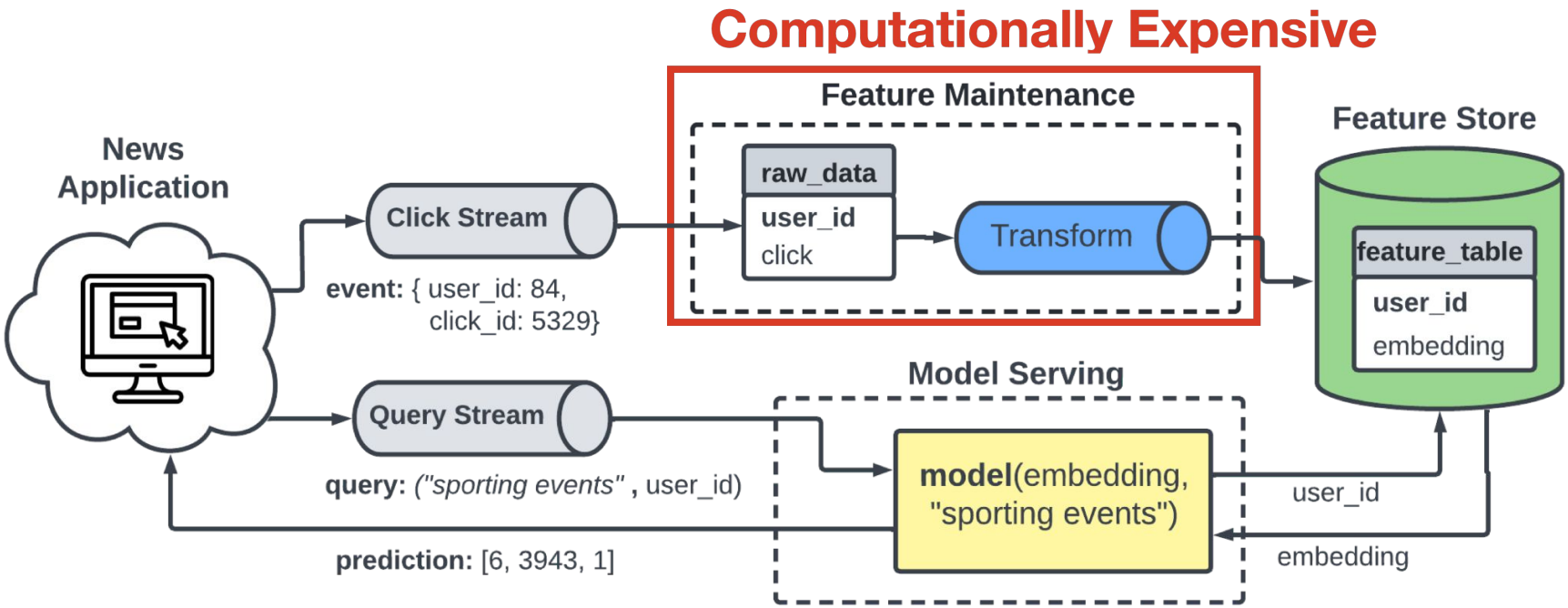
   Problem: Data is slow and expensive to replicate

   Skyplane: Optimizing *data movement* for faster, cheaper data replication

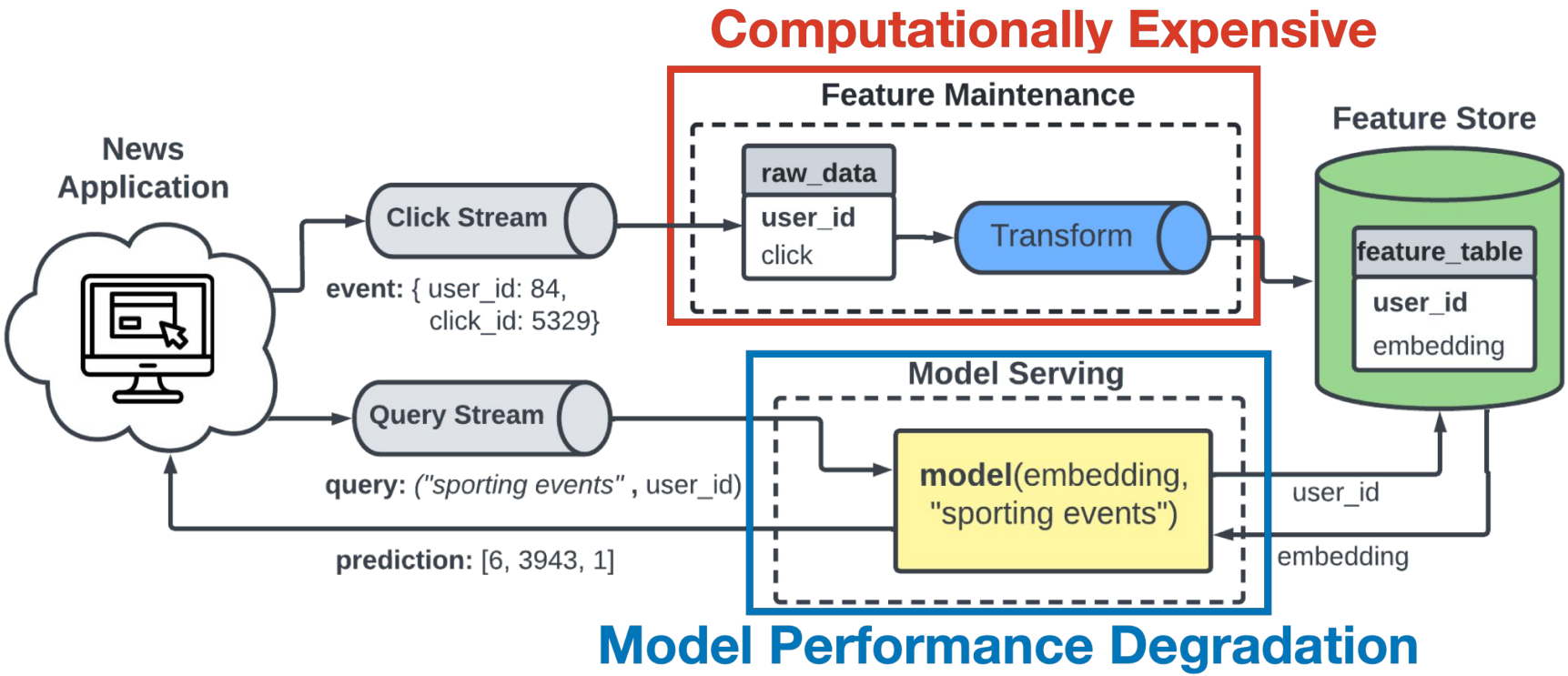
ralf: Optimizing Computation for Feature Maintenance



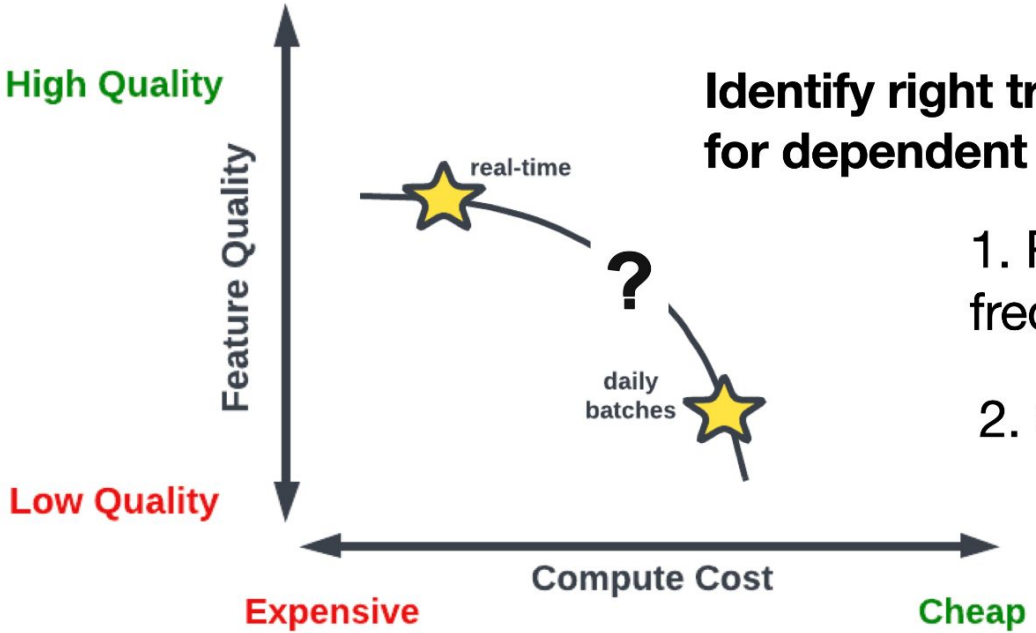
ralf: Optimizing Computation for Feature Maintenance



ralf: Optimizing Computation for Feature Maintenance



Tradeoffs for Pre-computed Features

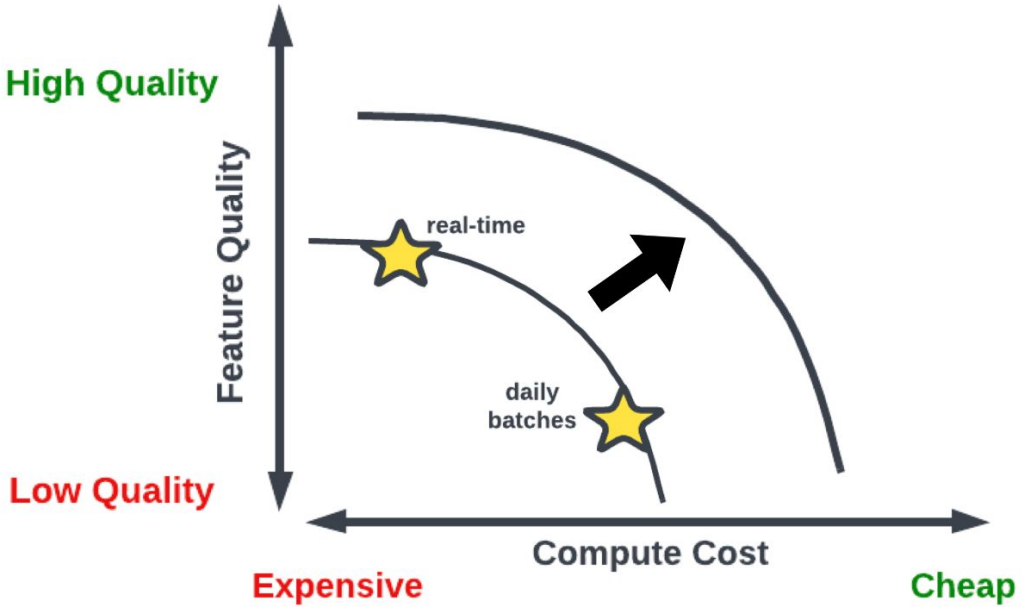


Identify right tradeoff point for dependent models

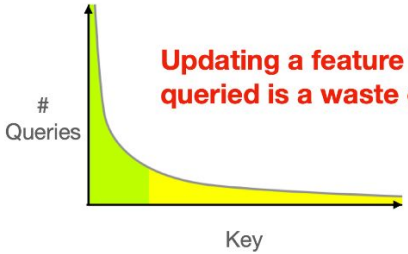
1. Resource **cost** for frequent feature updates

2. Queried feature **quality**

Opportunity for More Intelligent Scheduling



Many features keys in a feature table may never be queried.



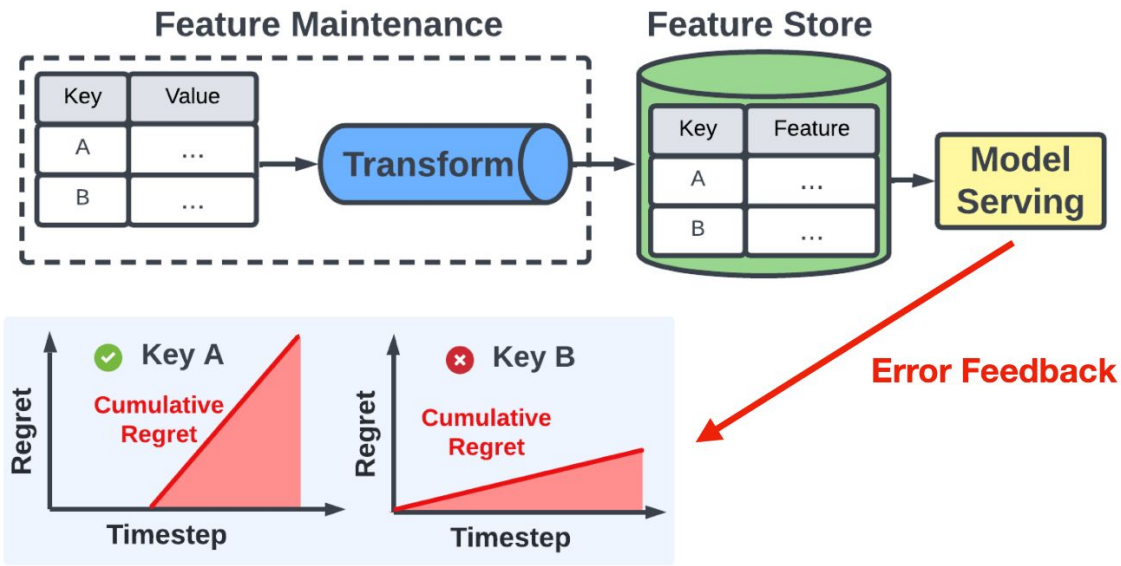
Updating a feature that is never queried is a waste of compute

Changes to features may not affect model predictions (robust to staleness)

Is staleness doesn't affect the model, there's no need to update the feature

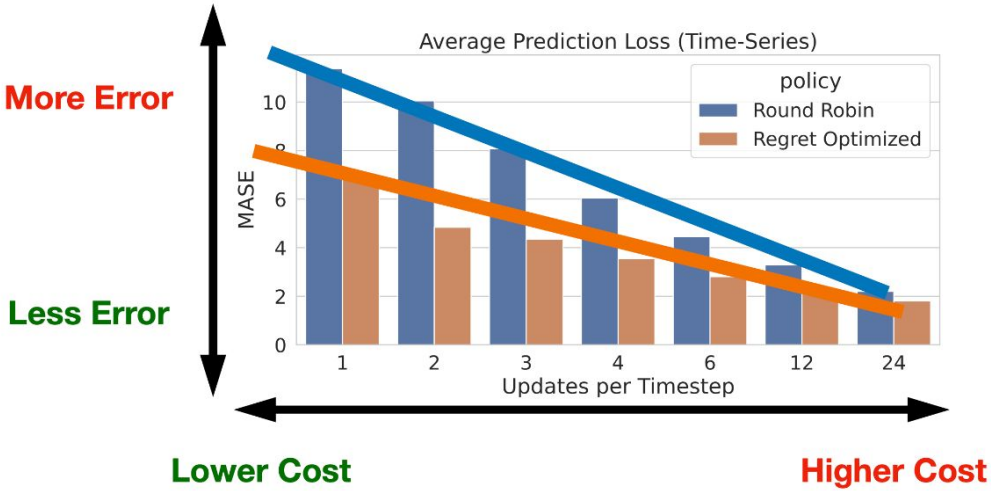
Scheduling Policy

We use the cumulative regret of a feature (since it was updated) as a measure to prioritize updating feature value.



Result: Higher Accuracy, Same Cost

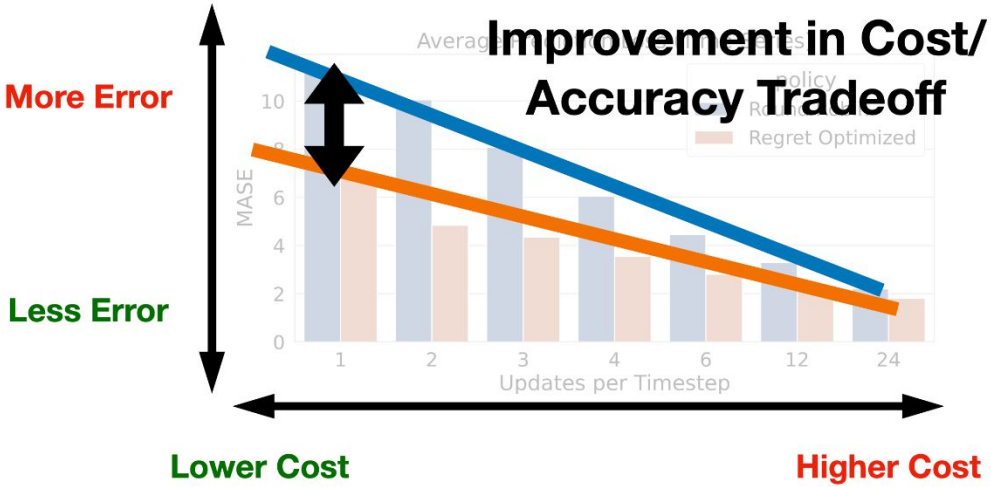
Lower prediction error some equivalent feature update budgets across different workloads (anomaly detection, recommendation)



(From current paper in-review. Please email wooders@berkeley.edu if you're interested in seeing the submission!)

Result: Higher Accuracy, Same Cost

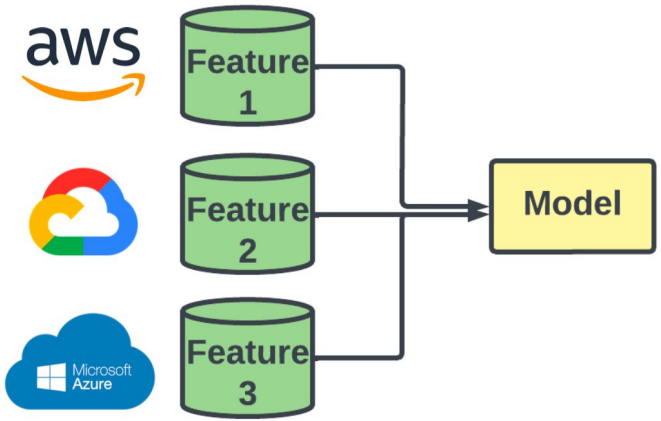
Lower prediction error some equivalent feature update budgets across different workloads (anomaly detection, recommendation)



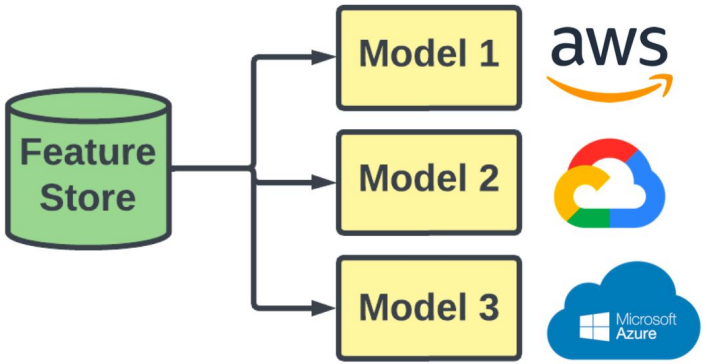
(From current paper in-review. Please email wooders@berkeley.edu if you're interested in seeing the submission!)

Skyplane: Fast, Cheap Cloud Data Replication

For many applications, data movement is the bottleneck (not compute) - e.g. for feature publishing and serving.



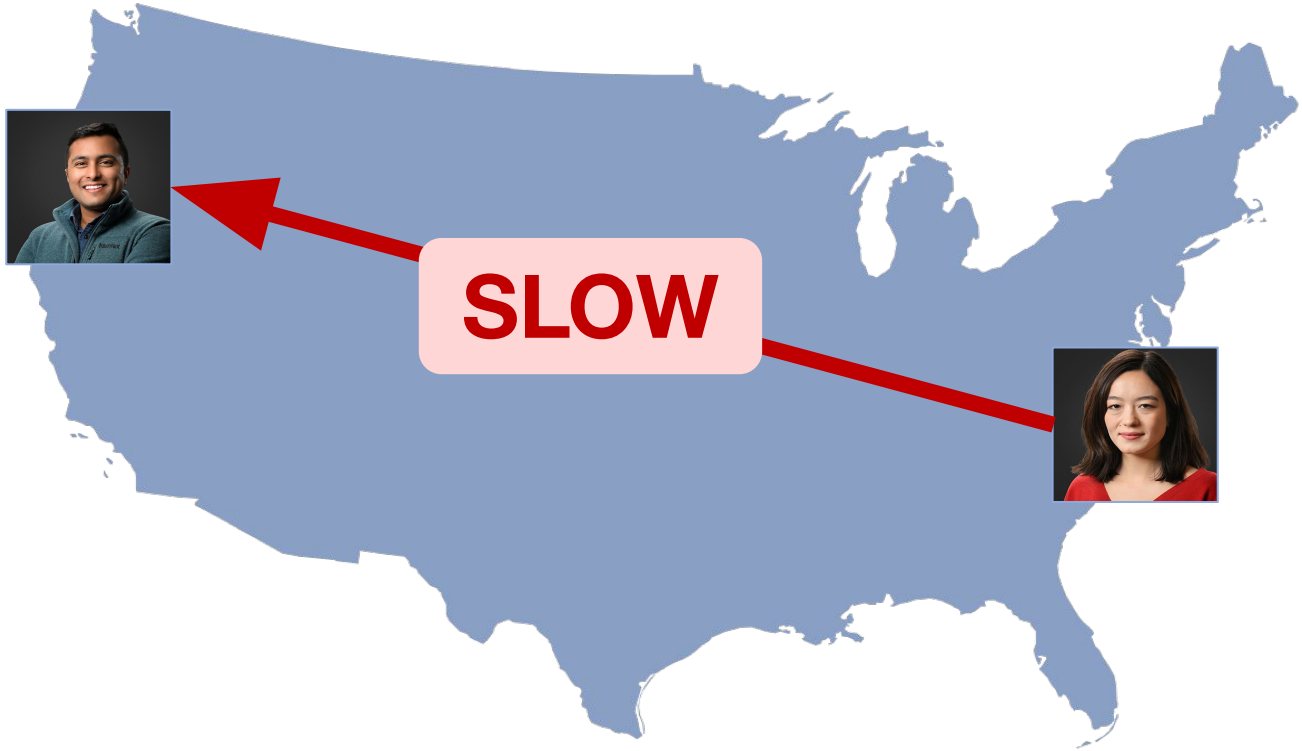
Models may need to access data located on different clouds and regions.



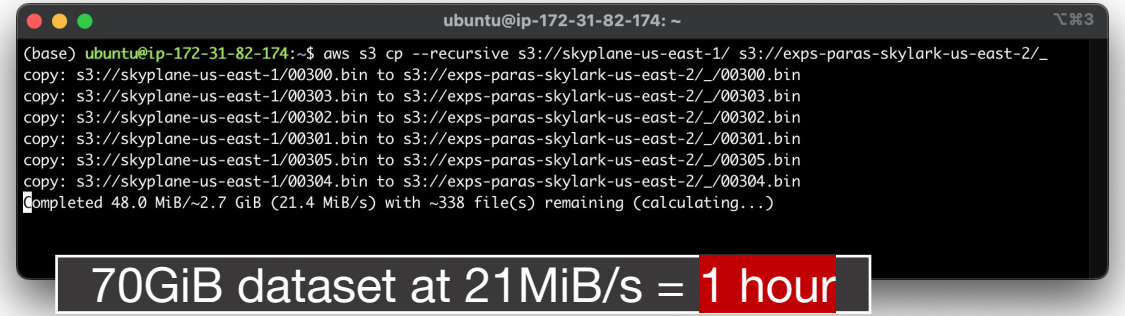
Data may need to be delivered to applications on different cloud regions or providers.

Direct transfer is slow

Moving a 70GB dataset between regions (single cloud) or providers (multi-cloud)



1. Slow transfers = stale data



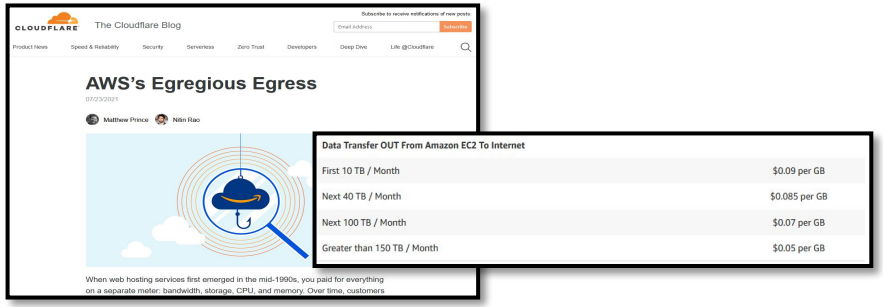
70GiB dataset at 21MiB/s = 1 hour

Cost to move 70GB dataset

=

Running **34 instances**
(m5.xlarge)

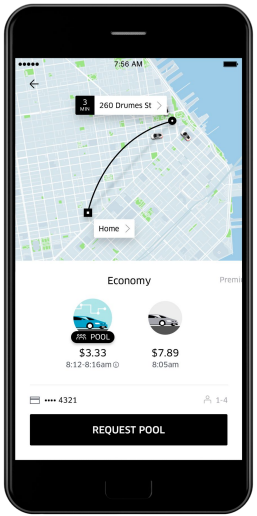
2. High egress fees = $$$

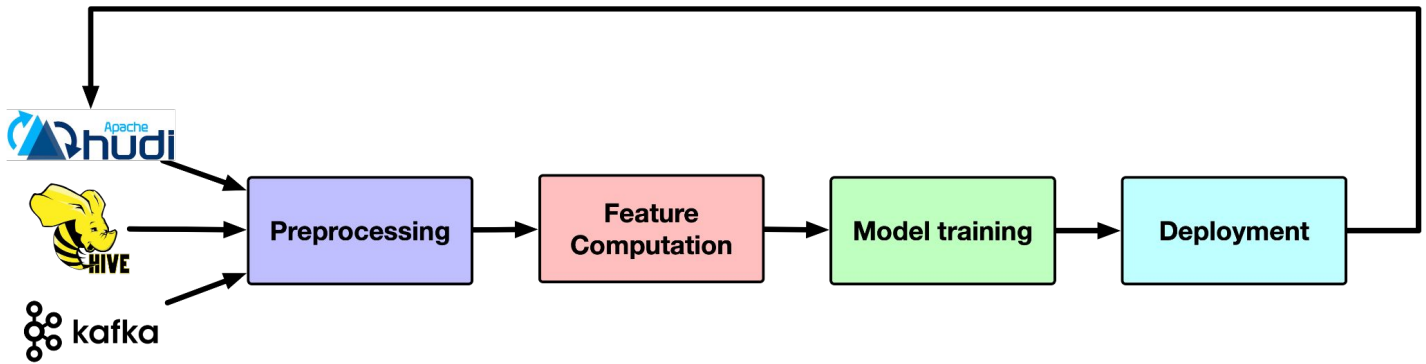


Why do slow transfers matter?

Slow transfers = worse accuracy

ETA prediction





What is Skyplane?

Problem: Managing data across regions and across clouds is **slow** and **expensive**

Skyplane is a system for fast, low-cost transfers between object stores.

```
skyplane cp {s3,gs,az}://...  {s3,gs,az}://...
```

How does it work?

1. ***Faster networking:***
   Control network route (overlay network)
   Striping over parallel VMs

2. ***Lower egress cost:***
   Compression + bandwidth tiering

Up to **113x faster** within single cloud

Up to **267x faster** between clouds

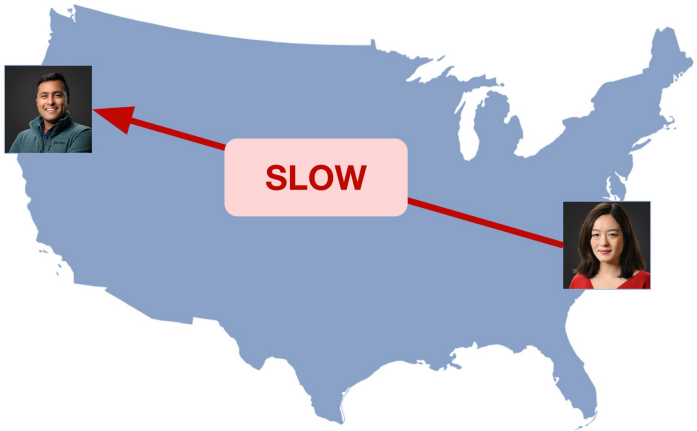
Up to **3.8x cheaper**

Open source project!

```
$ pip install skyplane
```

skyplane.org

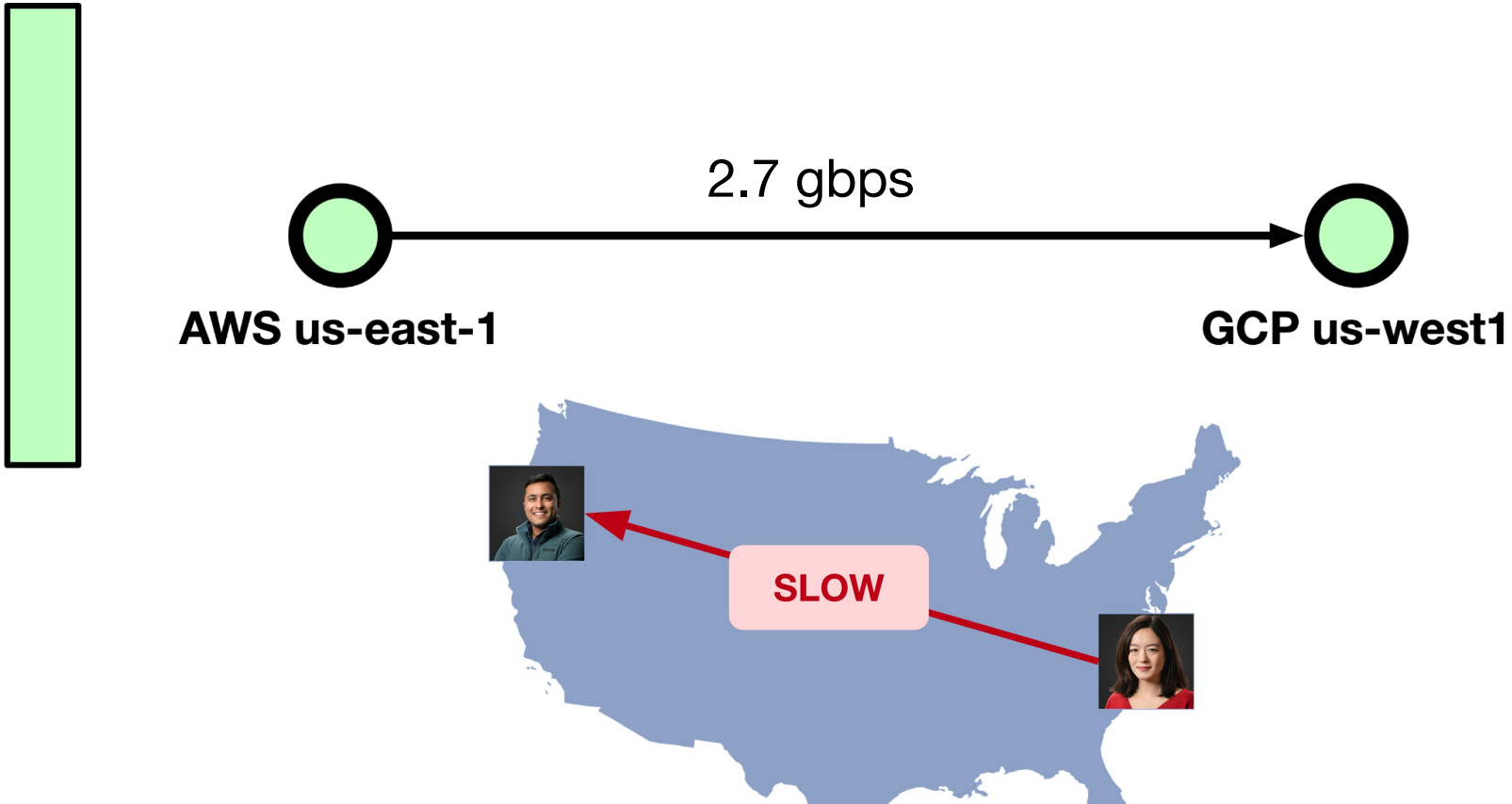
Direct internet path between clouds are often slow



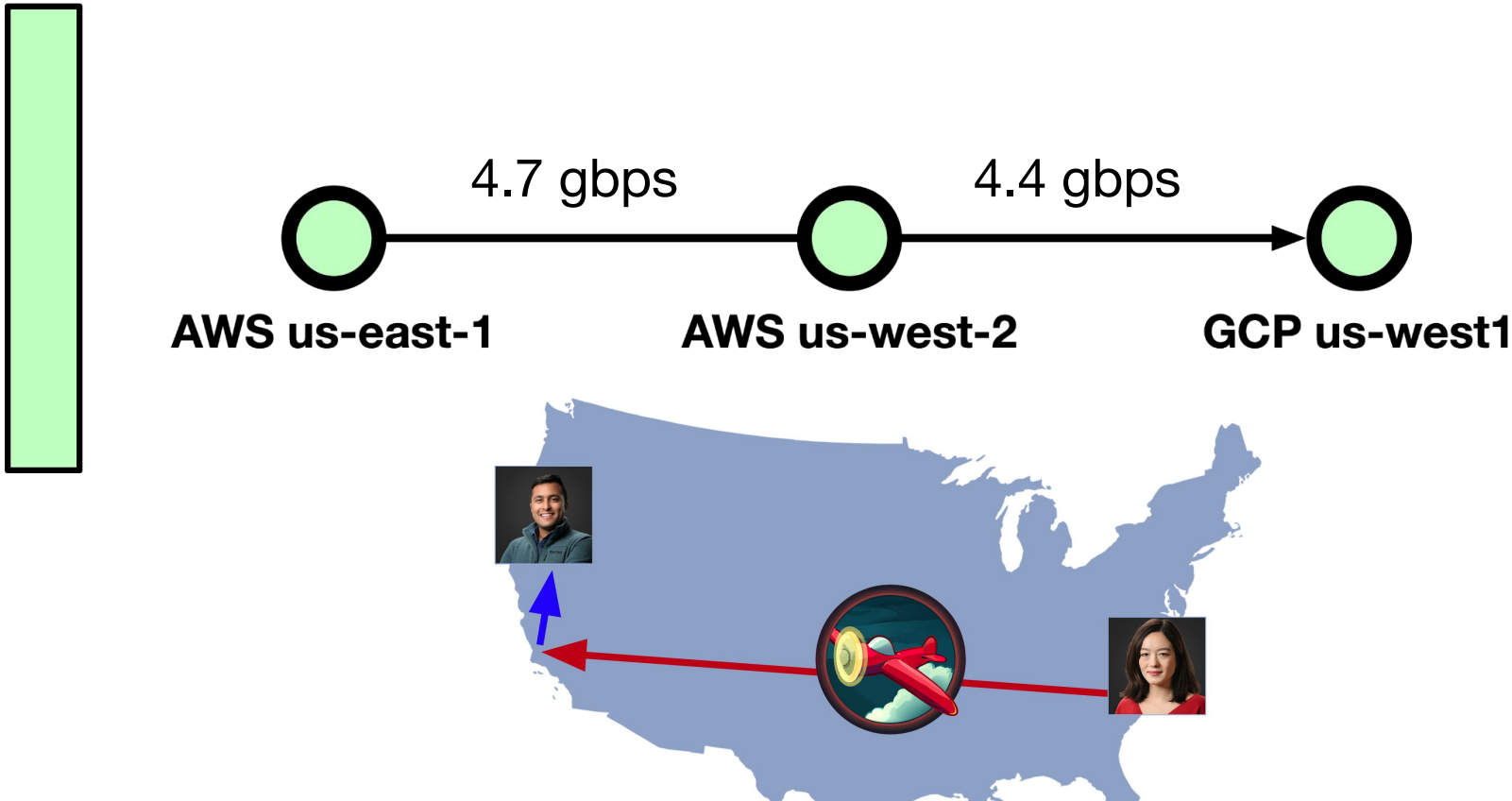
Reasons for slow transfers

1. Congestion along direct path
2. Poor peering between providers
3. Packet loss from the physical layer

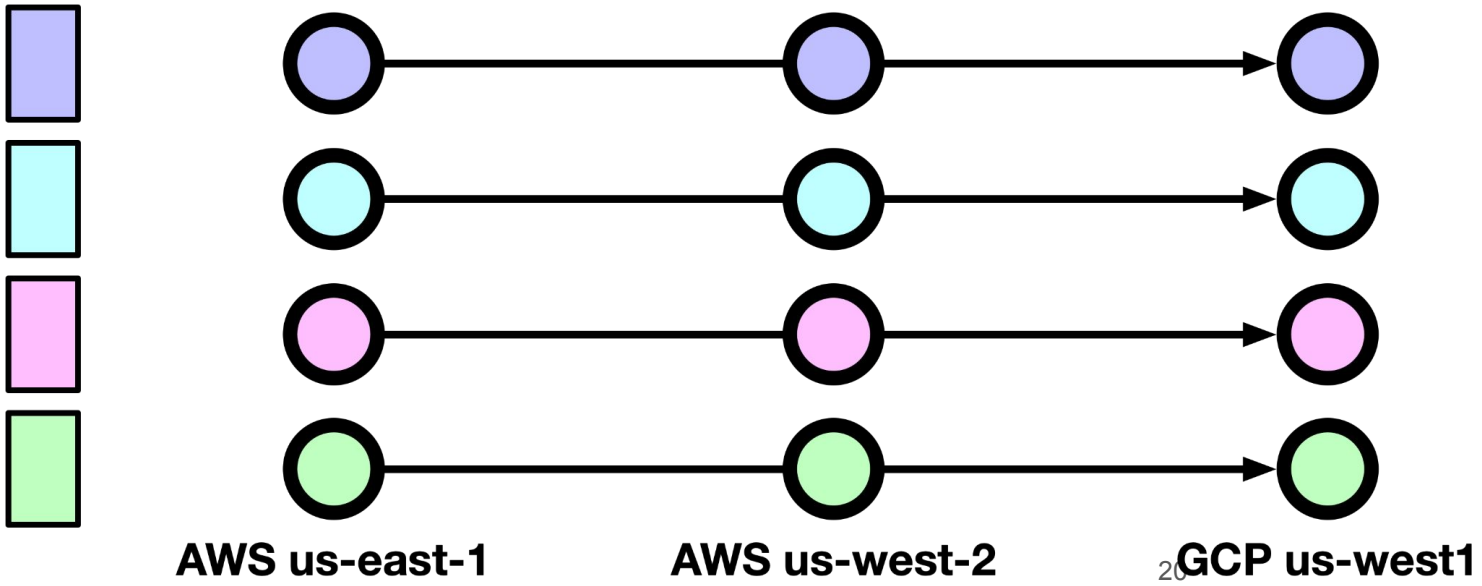
Direct internet path between clouds are often slow



Overlay routing allows circumventing slow links

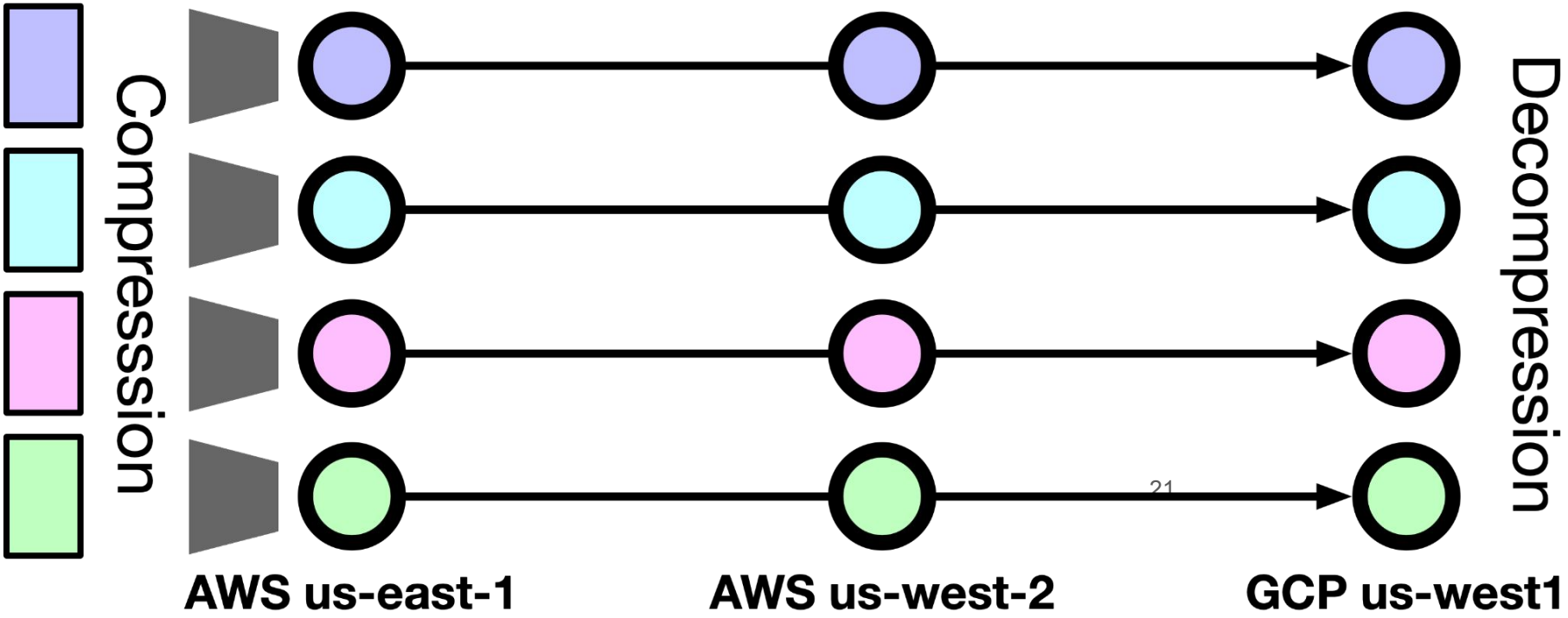


Striping transfers large files across parallel VMs

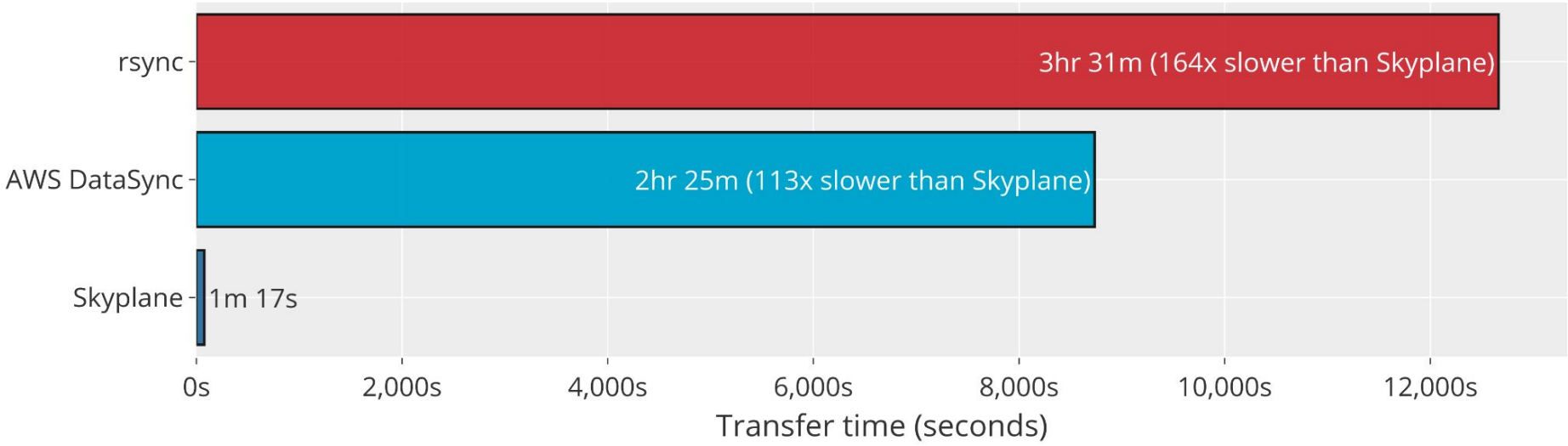


2.7 gbps => 17.6 gbps

Low cost: Compression to reduce egress cost



All together: transfer datasets in minutes with Skyplane

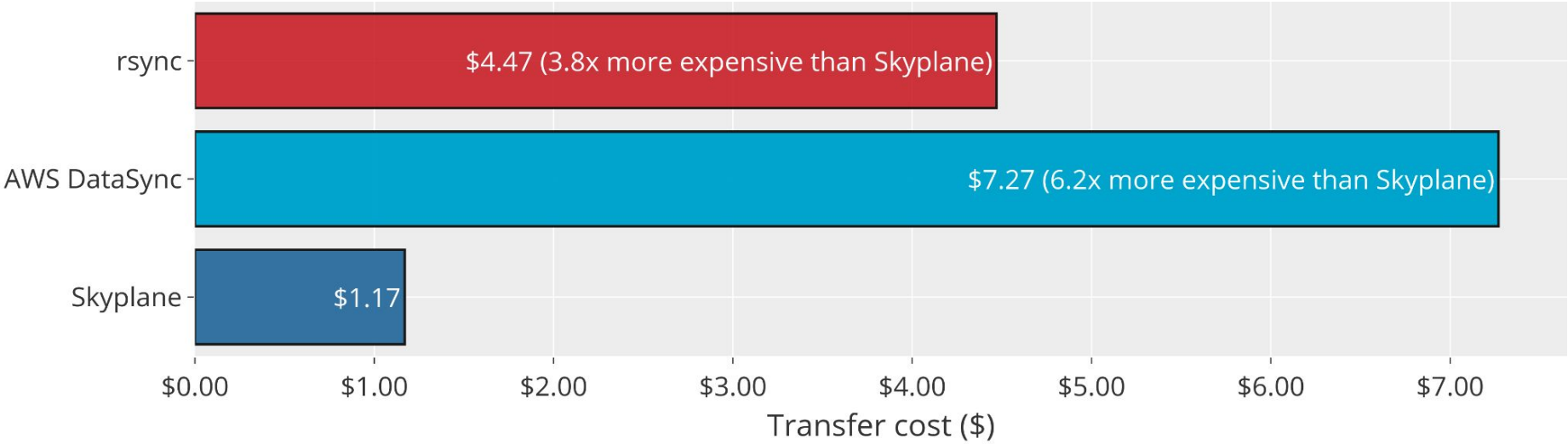


Workload: 1x 64GB file
Source: AWS ap-southeast-2
Destination: AWS eu-west-3
# of VMs: 8

2.5 hours => 1.5 mins

Full benchmarks online at https://skyplane.org/en/latest/benchmark.html

Low cost: up to 3.8x cheaper for compressible data



Workload: 228GB Wikipedia dump
Source: AWS us-east-1
Destination: AWS us-west-2
# of VMs: 8

$4.47 => $1.17

All techniques explained in our NSDI 2023 paper

Overlay routing

Longer indirect paths are worthwhile for slow links

of VMs per region

Access throughput beyond NIC, AWS and GCP throttle egress

of parallel TCP connections

Unlike internet, fairness is a provider-level concern due to egress fees

Network tier selection

Hot potato routing up to 40% cheaper than cold potato

Skyplane: Network Overlays for Navigating the Cost–Performance Tradeoff for Inter-Cloud Bulk Transfers

Paras Jain, Sam Kumar, Sarah Wooders, Shishir Patil, Joseph Gonzalez, and Ion Stoica
University of California, Berkeley

Abstract

Cloud applications are increasingly distributing data across multiple regions and cloud providers due to privacy regulations, availability of specialized hardware, and to prevent vendor lock-in. Unfortunately, wide-area bulk data transfers are often slow, bottlenecking applications. We demonstrate that it is possible to significantly improve inter-cloud bulk transfer throughput by adapting network overlays to the cloud setting—that is, by routing data through indirect paths at the application layer. However, directly applying network overlays in this setting results in unacceptable increases in cloud egress prices. We present Skyplane, a system for bulk data transfer between cloud object stores that uses network overlays to optimally navigate the trade-off between price and performance. Skyplane's planner uses mixed-integer linear programming to determine the optimal overlay path and resource allocation for data transfer, subject to user-provided constraints on price or performance. Skyplane's data plane saturates the available bandwidth found by the planner with parallel reads/writes and integrates well with object stores, aligning with the stores' internal layouts. Together, these techniques allow Skyplane to significantly improve object transfer throughput at very low cost overheads.

1 Introduction

Many applications replicate data across multiple regions and datacenters, requiring fast bulk data transfers across the wide area. These applications include transferring data from one datacenter to others for data analysis (e.g. ETL [6], Geo-Distributed Analytics [38]), and transferring the results of analysis to other datacenters for production (e.g. moving search indices [25]). There has been extensive prior work in optimizing the transfer time of bulk data transfers between datacenters, with application requirements on transfer time often modeled as deadlines [25, 26, 28, 46]. All mainstream cloud providers offer services for bulk transfers (e.g. AWS' DataSync, Azure AzCopy, and GCP Cloud Transfer Service).

Increasingly, applications are built not only on multiple regions within a cloud provider (multi-region), but also across multiple cloud providers (multi-cloud), due to variations in pricing, features, and availability of cloud services. In a recent survey [19], more than 86% of 727 respondents had adopted a multi-cloud strategy across diverse workloads. Thus, support for fast, *cross-cloud* bulk transfers is increasingly important.

Bulk transfer performance is governed by two factors: transfer time (latency) and cost. Thus, we ask: *How can we improve the transfer of data objects to support latency-sensitive workloads, while minimizing the additional cost of achieving faster data transfers?* We study this question in the context of designing and implementing Skyplane, an inter-cloud object transfer system which we plan to open-source.

A seemingly natural approach is to optimize the routing protocols used internally by cloud providers to support higher-throughput inter-cloud data transfers. Unfortunately, this is not feasible for two reasons. (1) Re-architecting the IP-layer routing protocol to optimize for high-throughput bulk transfer could be suboptimal for other applications that are sensitive to network latency. (2) Cloud providers lack a strong incentive to optimize efficient data transfer to other clouds. Indeed, AWS DataSync [4], AzCopy [15], GCP Data Transfer [23], AWS Snowball [44] and Azure Data Box Disk [8], all support data transfer *into*, but not out of, their respective clouds. However, the network path between clouds is ultimately split among multiple cloud providers, and the best results can only be achieved with the source cloud's cooperation as well.

Skyplane's key observation is that we can instead identify *overlay paths*—paths that send data via intermediate cloud regions—that are faster than the direct paths. For example, consider transferring data from Azure's `Central Canada` region to GCP's `asia-northeast1`. Using direct TCP connections would yield 6.2 Gbps. Instead, Skyplane can first transfer data from `Central Canada` to Azure's `US West 2` at 25.9 Gbps, and then transfer it from `US West 2` to `asia-northeast1` at 12.4 Gbps, for a 2.0× speedup (Fig. 1). Crucially, this can be implemented on top of the cloud providers' services, without their explicit buy-in.

1

Thank you!

Contact: wooders@berkeley.edu

ralf: https://github.com/feature-store/ralf

Skyplane:

```
$ pip install skyplane
$ skyplane init
$ skyplane cp -r s3://… gcs://…
```

skyplane.org