Cheetah: Accelerating database queries with switch pruning

Muhammad Tirmazi, Ran Ben Basat, Jiaqi Gao, Minlan Yu
(Harvard University)

ACM SIGMOD 2020

Cheetah at Sunset by Arturo de Frias Marques - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=35332383

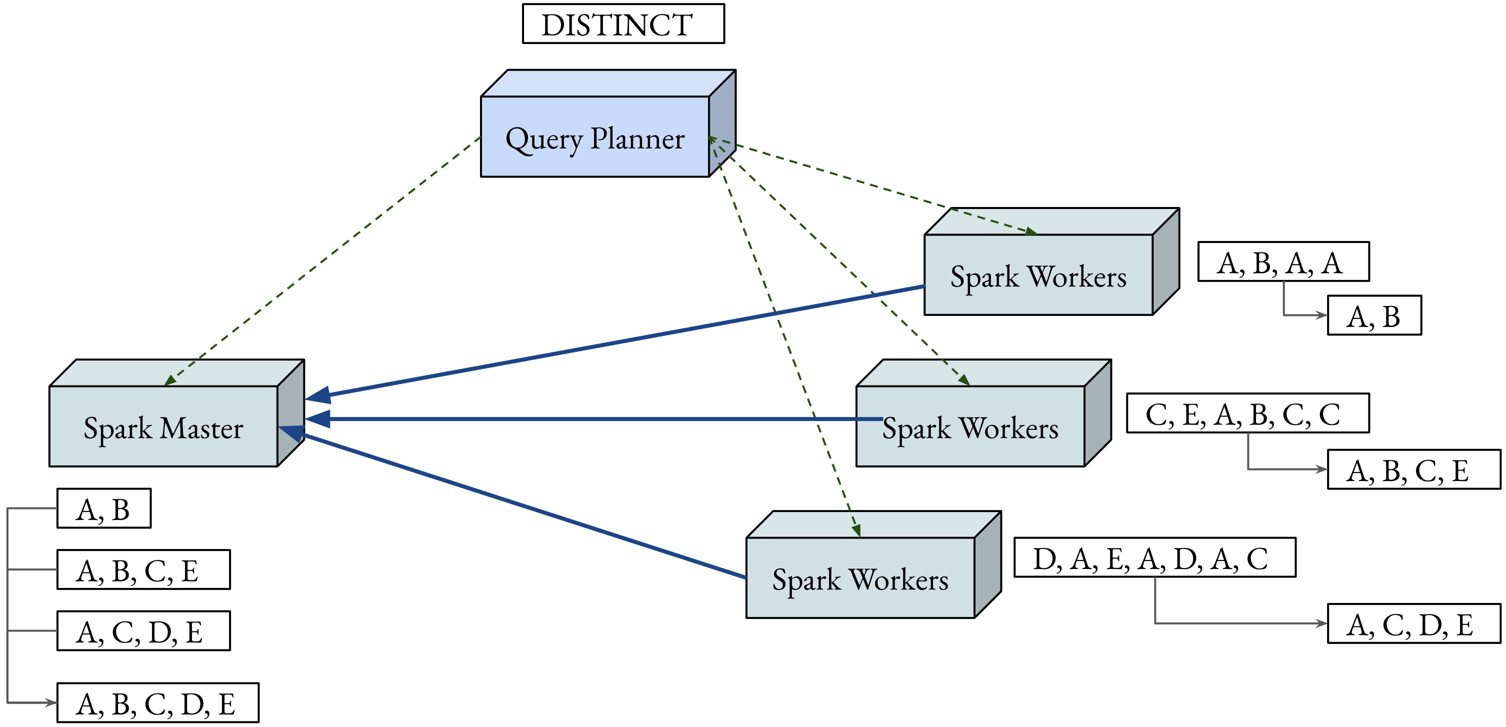
Databases

Over 8 **billion** daily queries on Alibaba cloud

Big Query

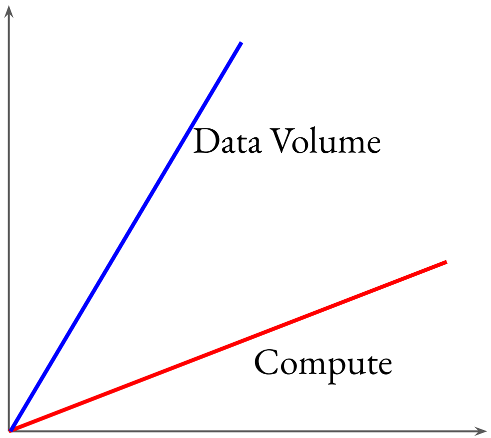
"BigQuery" (CC-BY 4.0) by Google LLC

Spark control flow



The problem

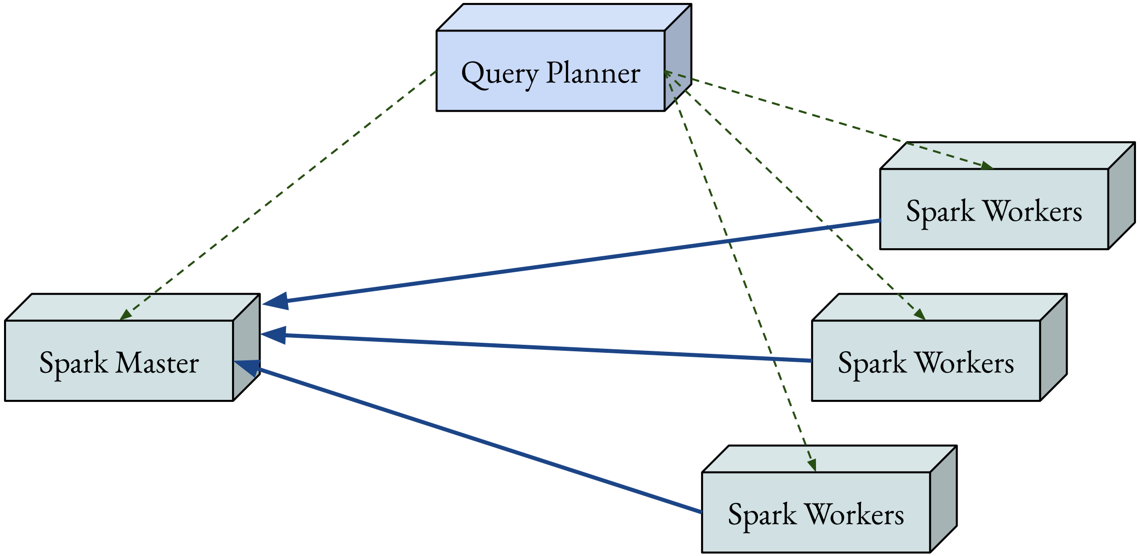
DB workload growth - fast
Generic CPUs - not fast enough



Cheetah's approach:
Offload to **programmable switches**

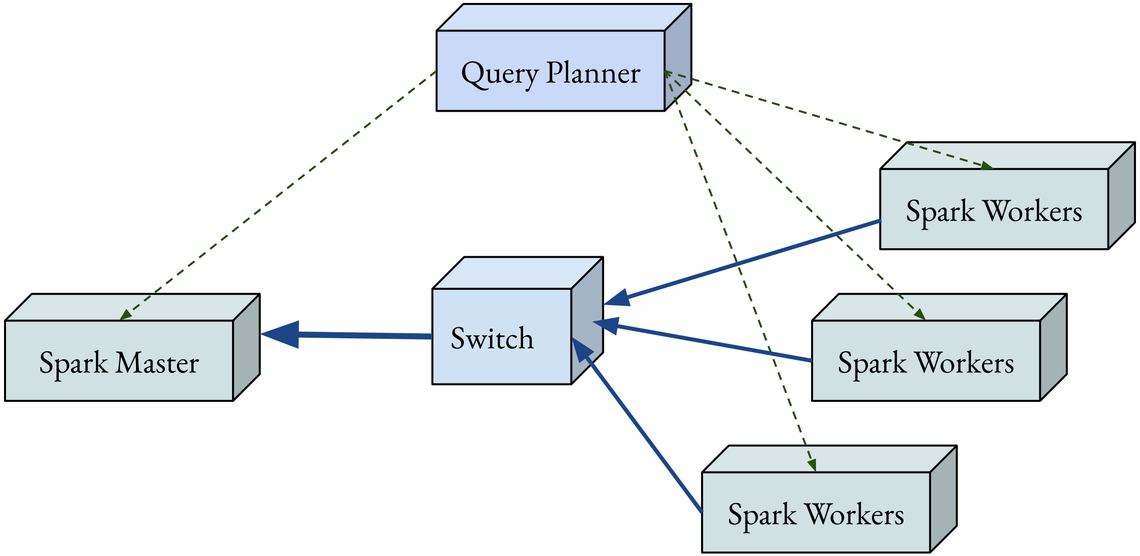
"Cheetah" (CC BY 2.0) by wwarby

Why programmable switches?



Process TBs of data - packets in microseconds

Why programmable switches?



Process TBs of data - packets in microseconds

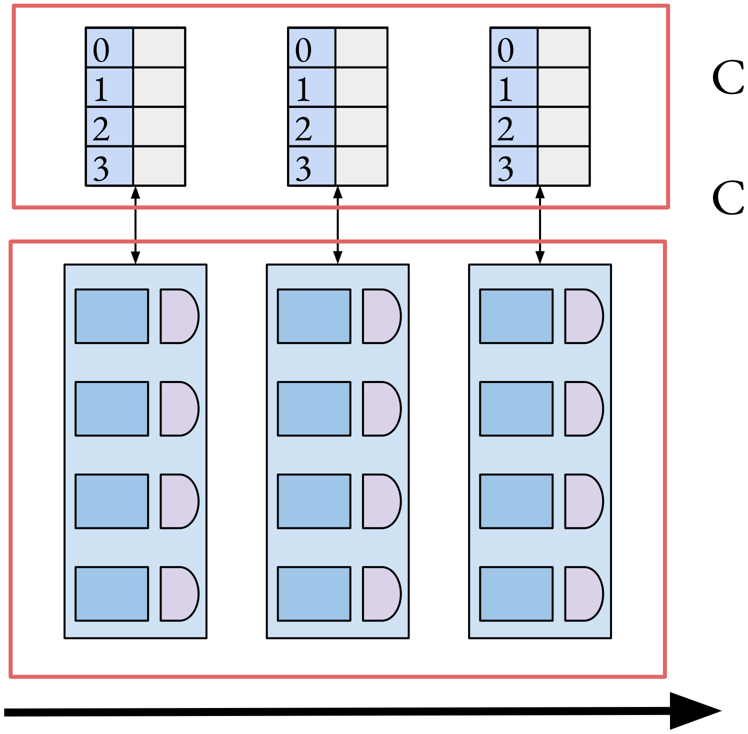
Already in the network.

Process **cross-partition** data.

PISA programmable switch

Stateful memory

Match-Action pipeline



Constraint 1: Limited memory

Constraint 2: Limited stages

One way

Distinct query

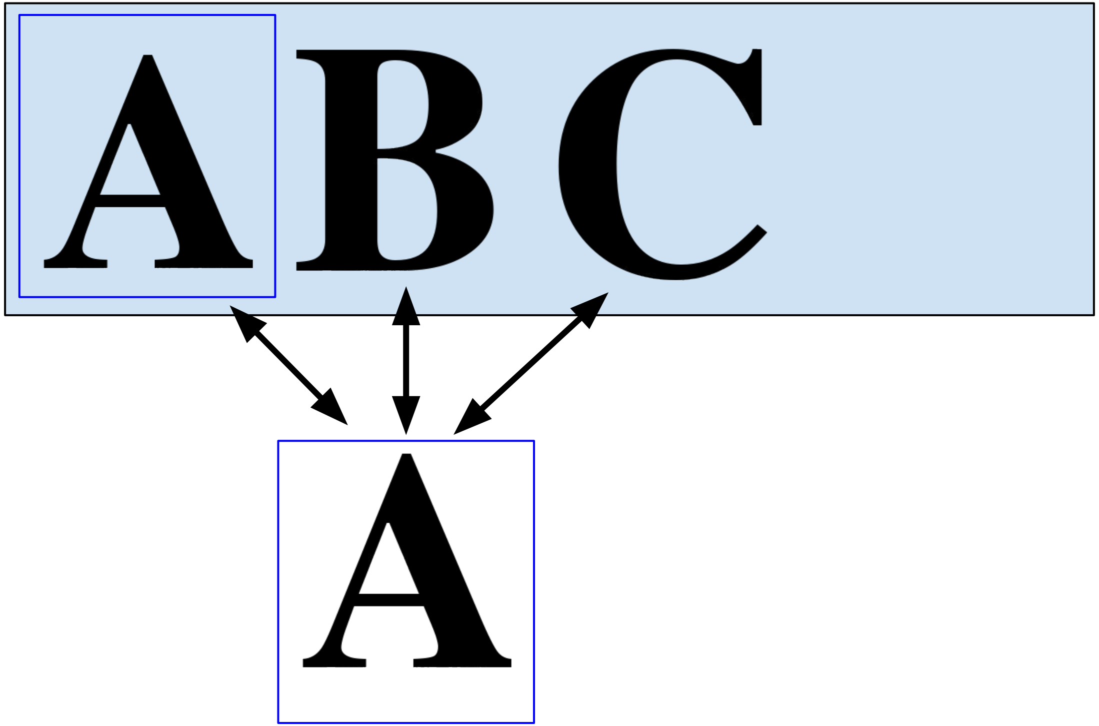
Switch

ABC

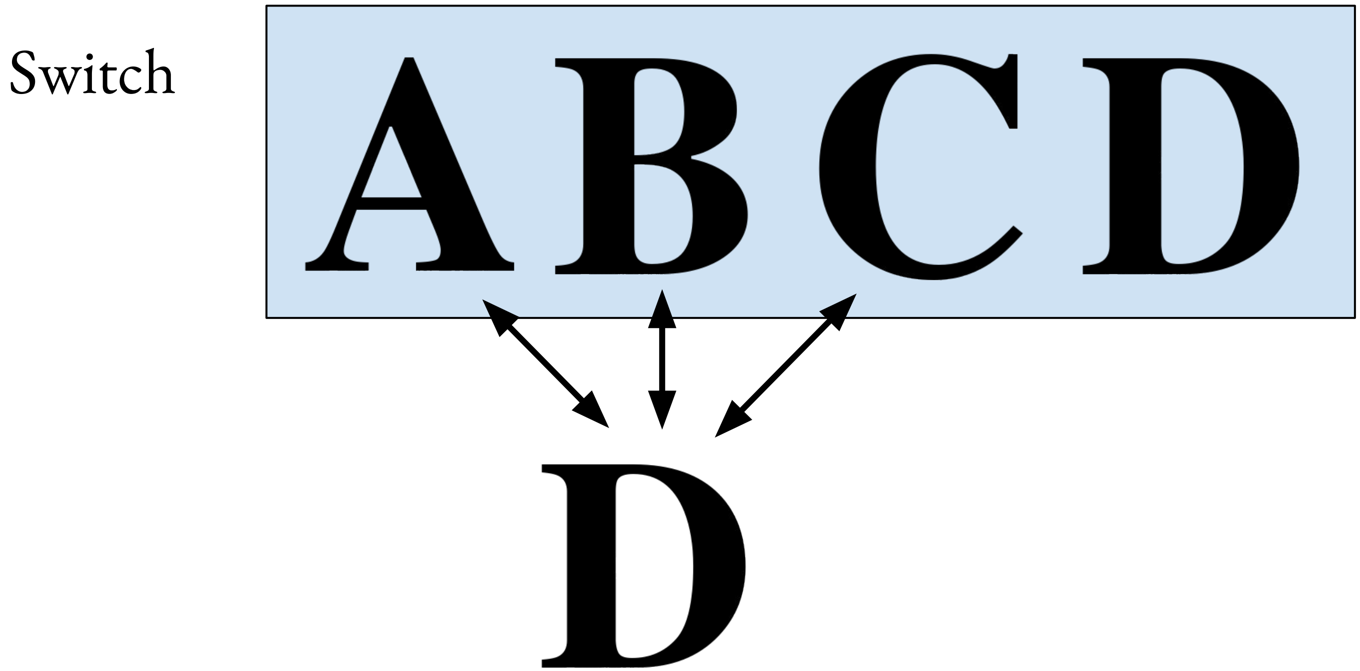
ABC

Distinct query

Switch



Append new entry



Issue: limited **stages**.

Switch

A B C D

The **pruning** abstraction

Dataset

A, B, A, A, C, D, D, E, A, B, C, C, D, A, E, A, D, A, C

The **pruning** abstraction

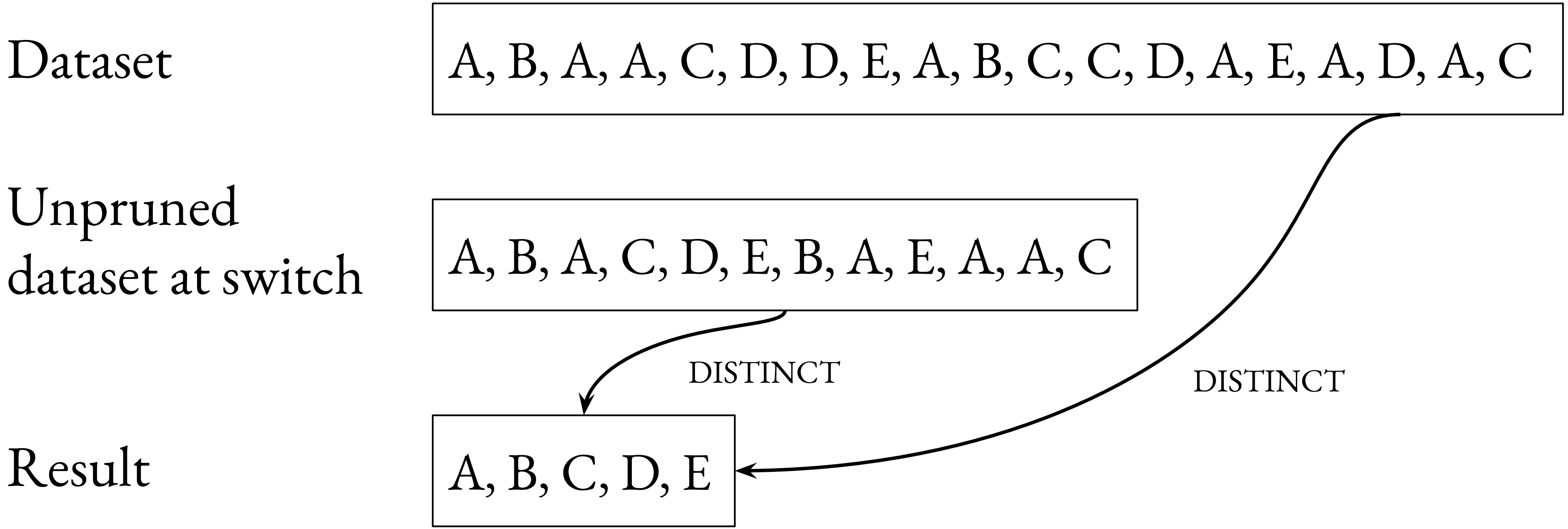
Dataset

A, B, ~~A~~, A, C, D, ~~D~~, E, ~~A~~, B, ~~C, C, D~~, A, E, A, ~~D~~, A, C

Unpruned dataset at switch

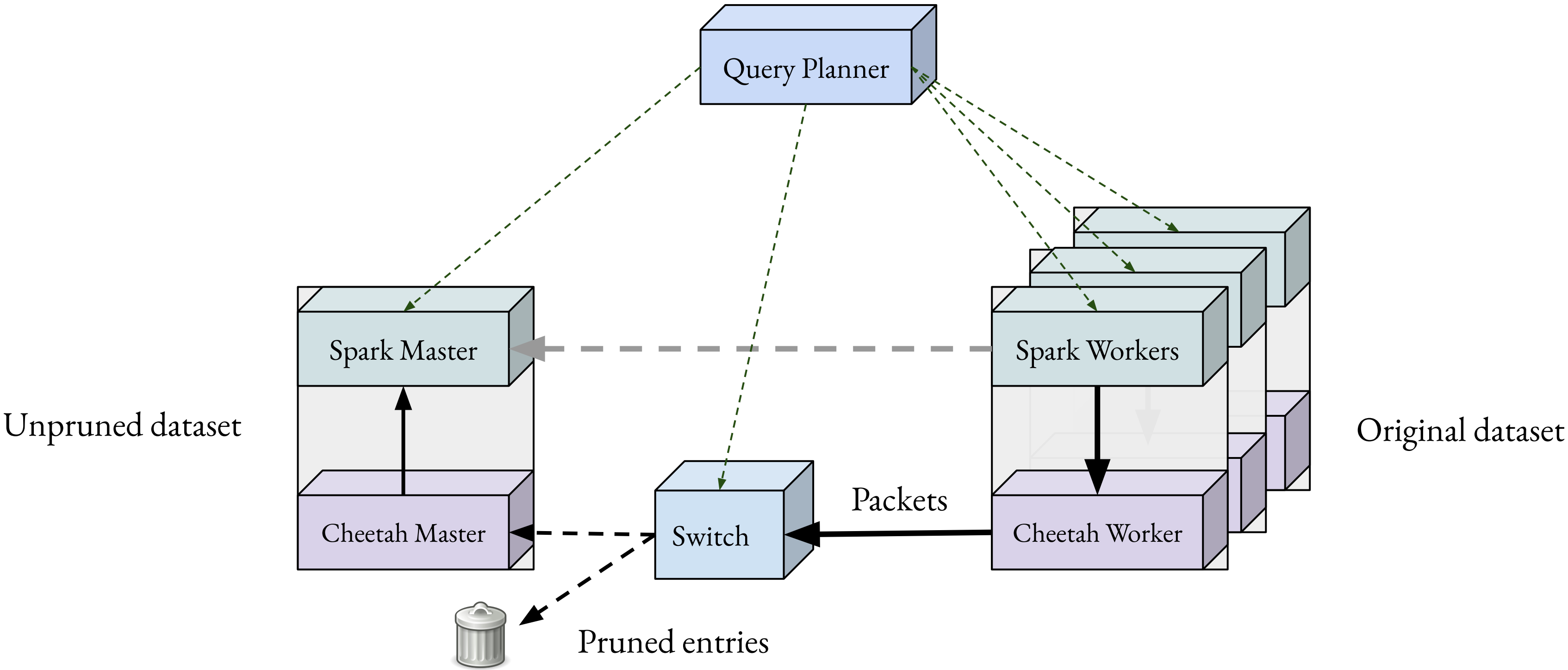
A, B, A, C, D, E, B, A, E, A, A, C

The **pruning** abstraction



Query on **unpruned** dataset = Query on original dataset

Integrating pruning with Spark



Solution 1: Bloom filter

New key hashed to bits in **switch memory**

Same key arrives **again** - pruned by switch

Issue: false **positives** break **pruning guarantee**!

Solution 2: cache

Switch

A B C D

E

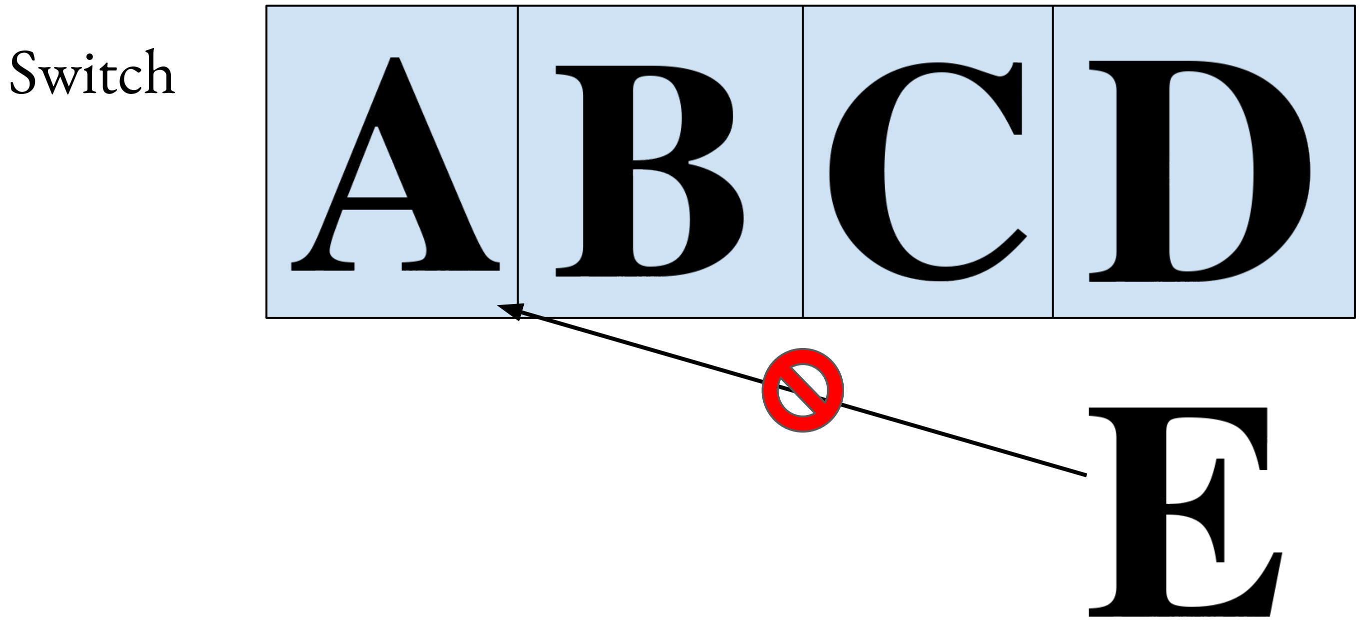
False negatives: **not** an issue!

Switch

E	**B**	**C**	**D**

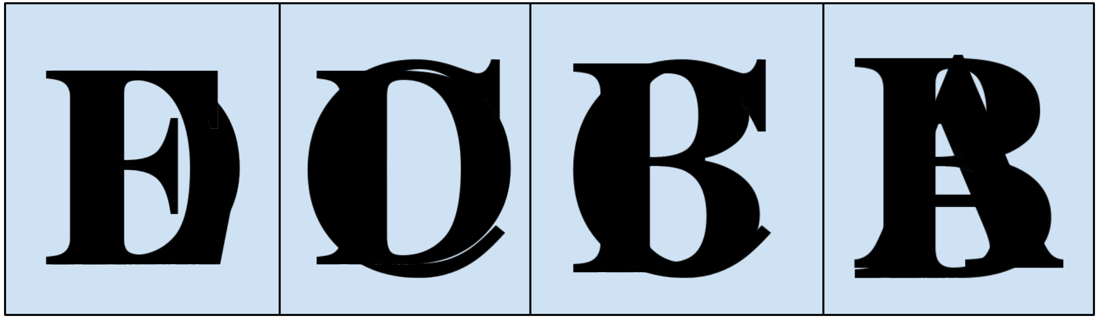
E

Issue: partitioned memory



Rolling replacement insertion

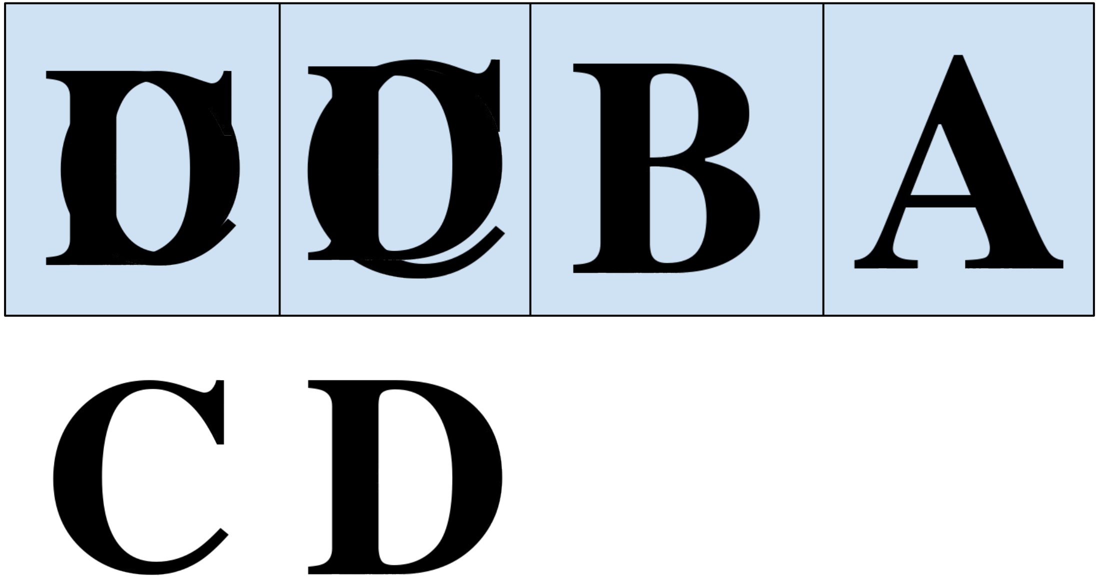
Switch



E D C B

Rolling replacement insertion

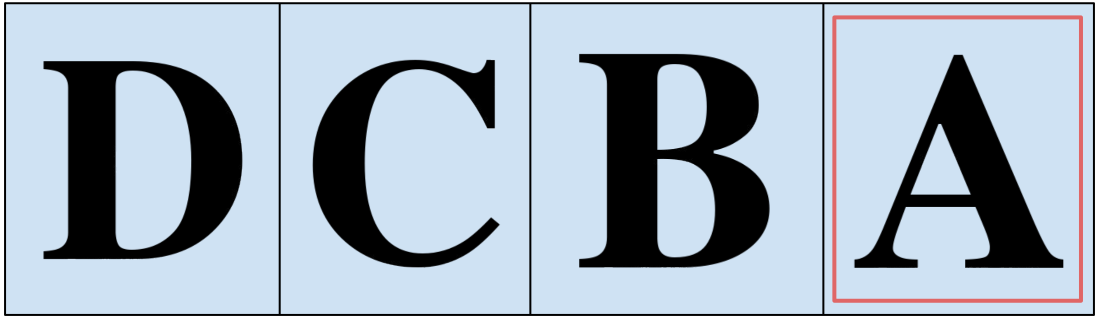
Switch



Rolling replacement insertion

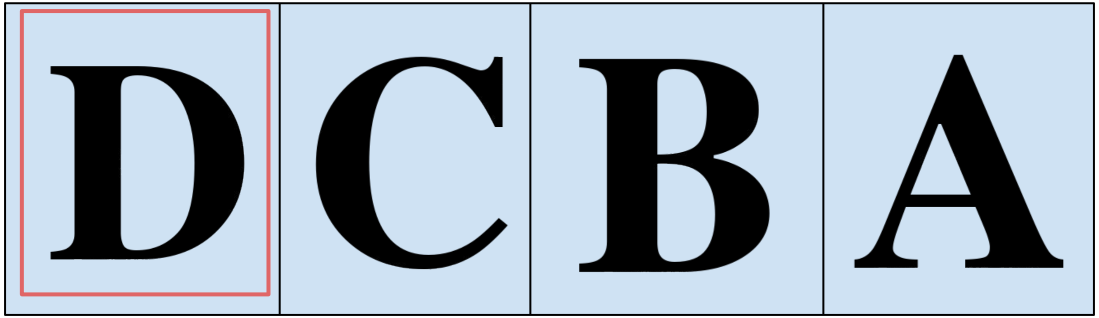
Least Recent

Switch



Issue: pruning rate

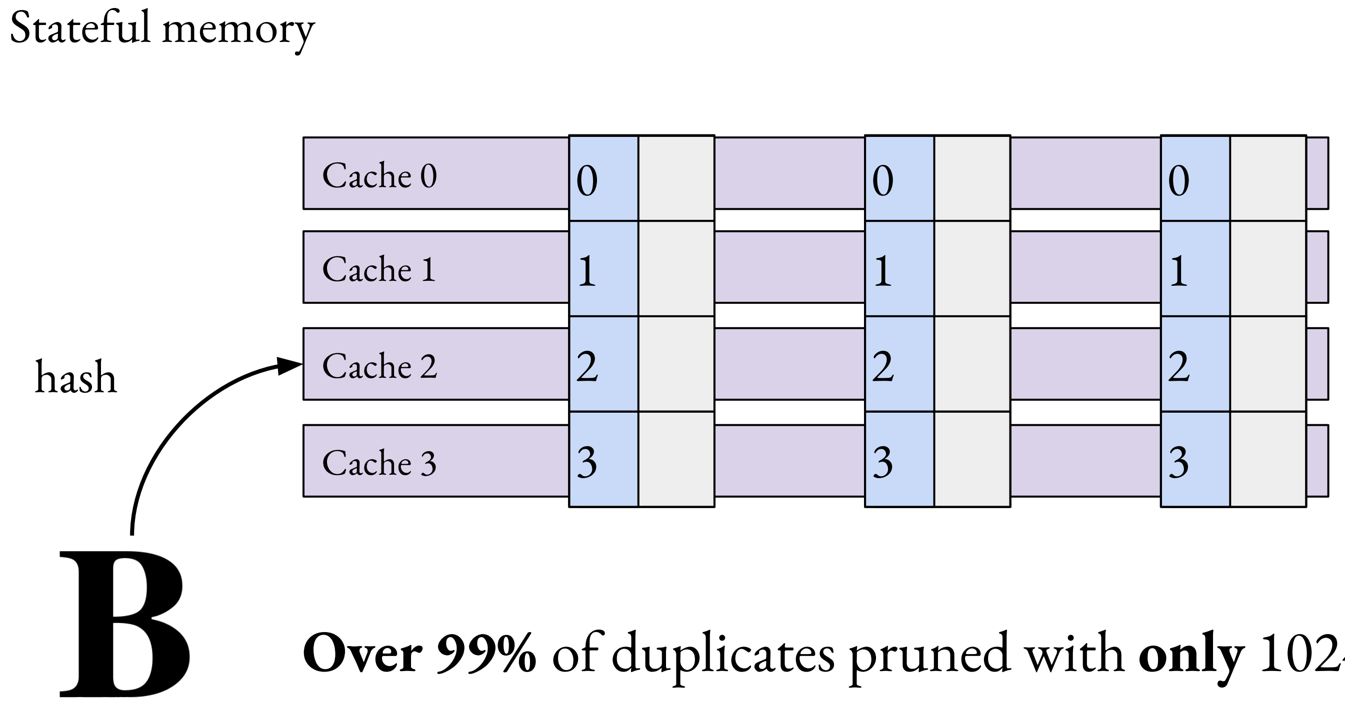
Switch



1 value per stage

Less than 50% of duplicates pruned for all tested workloads

Solution: multi-row cache



Over 99% of duplicates pruned with **only** 1024 rows

Queries supported

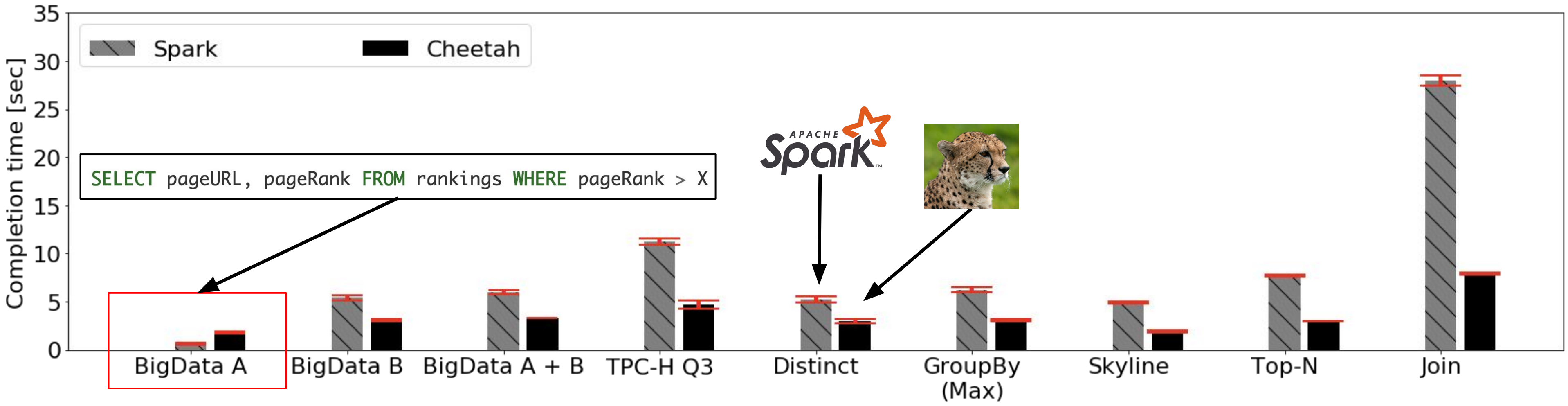
Store helpful pruning points

Query	Distinct	Join	Group-By	Top-N	Having	Filtering	Skyline
Row Partitioning	✓		✓	✓	✓		
Caches	✓				✓		
Bloom filters		✓					
Sketches			✓		✓		
Thresholds				✓			
Partial query offloading					✓	✓	
Projection							✓

Experimental setup

- Programmable switch: Intel's Barefoot Tofino
- Default Spark 2x deployment - 5 workers
- Big Data Uservisits: 6.4 Million Entries ~ 1.3 M per partition
- 10 Gbps network bandwidth. 2 cores / 4 GB per worker.

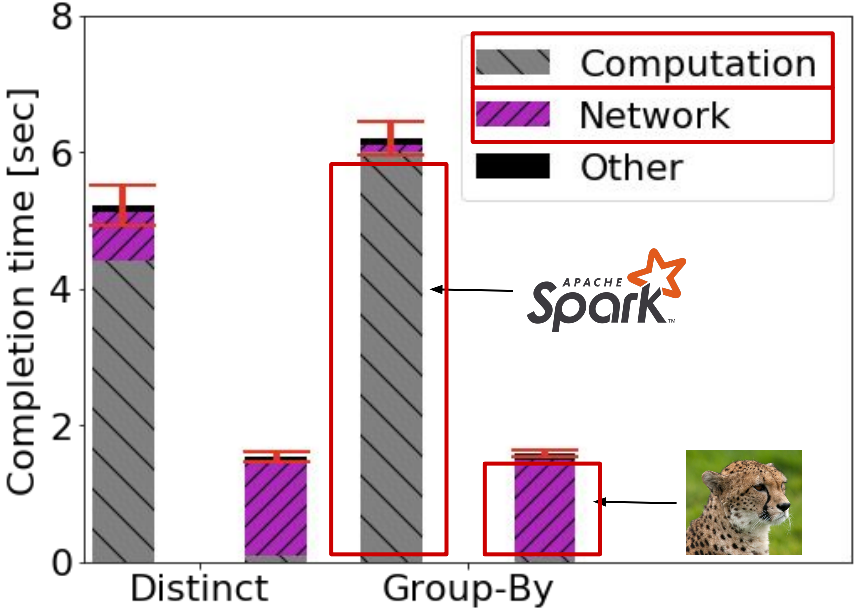
Pruning optimizes **compute bound** queries.



40 % to 75 % **faster** completion times

"Cheetah" (CC BY 2.0) by wwarby

The Network-Compute Tradeoff



Also in the paper

Pruning algorithms for **Join**, **Group-By**, **Having,** Skyline, Top-K, and Filtering along with workload-independent pruning rate guarantees.

Reliability protocol that supports pruning.

Support for **compound queries** and **multiple switches**.

Takeaway

Questions?

The **network** should play an **active role** in **query processing**.

Switches have **constraints**, but can optimize queries with the right abstraction: **pruning**.

`github.com/harvard-cns/cheetah-release`

"Cheetah" (CC BY 2.0) by wwarby