

# [544] BigQuery ML and Cost Management

Tyler Caraza-Harter

# Learning Objectives

- create and use machine learning models with BigQuery
- describe the relationship between BigQuery's two billing models (capacity and on-demand)
- manage and inspect BigQuery costs

# Outline

BigQuery ML Basics

Feature Transformation

Cost Management

# Train/Test Split

BigQuery provides a `DATA_SPLIT_METHOD` config, but its a bit unusual.

Default behavior depends on dataset

- <500 rows: 100% training data
- <50K rows: 80% training data
- bigger: 10K rows for test, rest for training

Documentation: "When there is a data split, you can find the temporary split results (Training Data, Evaluation Data) on the Model Details page in the BigQuery Console and the model API `data_split_result` field. **These split tables will be saved for 48 hours.** If you will need them for longer than 48 hours, copy them out of the anonymous dataset for longer retention."

Recommendation:

- split manually using `rand()<ratio` in SQL (`rand` gives num between 0 and 1)
- disable BigQuery splitting: **`DATA_SPLIT_METHOD="NO_SPLIT"`**

# Training

Step 1: write a query to select both features and label

```
SELECT yesterday_temp, humidity, temp
FROM weather
```

features label  
(to predict)

# Training

Step 2: choose a model name and create it

```
CREATE OR REPLACE MODEL myproj.mydataset.mymodel  
OPTIONS (...)  
  
AS
```

```
SELECT yesterday_temp, humidity, temp  
FROM weather
```

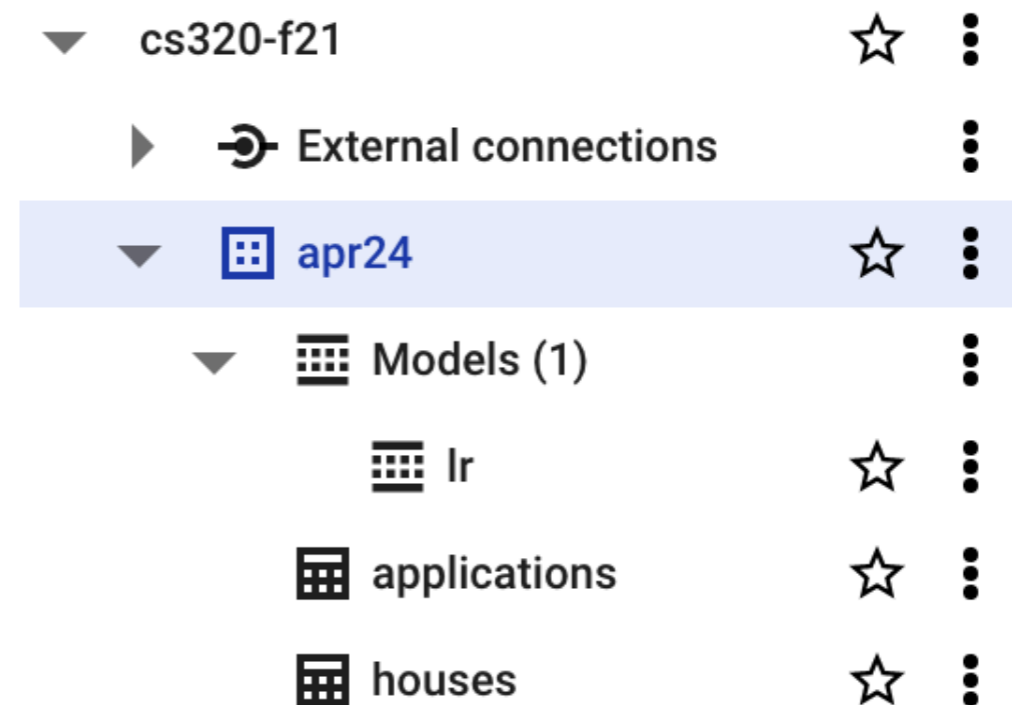
hierarchy:

projects

    datasets

        tables

        models



# Training

Step 3: choose type of model

```
CREATE OR REPLACE MODEL myproj.mydataset.mymodel  
OPTIONS (MODEL_TYPE='LINEAR_REG')
```

AS

```
SELECT yesterday_temp, humidity, temp  
FROM weather
```

**Options:** LINEAR\_REG, LOGISTIC\_REG, KMEANS, MATRIX\_FACTORIZATION,  
PCA, AUTOENCODER, AUTOML\_CLASSIFIER, AUTOML\_REGRESSOR,  
BOOSTED\_TREE\_CLASSIFIER, BOOSTED\_TREE\_REGRESSOR,  
RANDOM\_FOREST\_CLASSIFIER, RANDOM\_FOREST\_REGRESSOR,  
DNN\_CLASSIFIER, DNN\_REGRESSOR, DNN\_LINEAR\_COMBINED\_CLASSIFIER,  
DNN\_LINEAR\_COMBINED\_REGRESSOR, ARIMA\_PLUS, ARIMA\_PLUS\_XREG,  
TENSORFLOW, TENSORFLOW\_LITE, ONNX, XGBOOST


# Training

Step 4: indicate label column (others are assumed features)

```
CREATE OR REPLACE MODEL myproj.mydataset.mymodel  
OPTIONS (MODEL_TYPE='LINEAR_REG',  
         INPUT_LABEL_COLS=['temp'])
```

AS

```
SELECT yesterday_temp, humidity, temp  
FROM weather
```





# Using Trained Models

Each of these functions return a table related to a model.

*what are the coefficients used to multiply features?*

ML.WEIGHTS(MODEL ????)

*what are the predictions given the features?*

ML.PREDICT(MODEL ????, (????))

SQL query to get features



*how well do we predict (various metrics) given the features+label?*

ML.EVALUATE(MODEL ????, (????))

SQL query to get features and label



# Using Trained Models

Each of these functions return a table related to a model.

*what are the coefficients used to multiply features?*

ML.WEIGHTS(MODEL ????)

*example:*

```
SELECT *  
FROM ML.WEIGHTS (MODEL mymodel)
```

**TopHat, Demos**

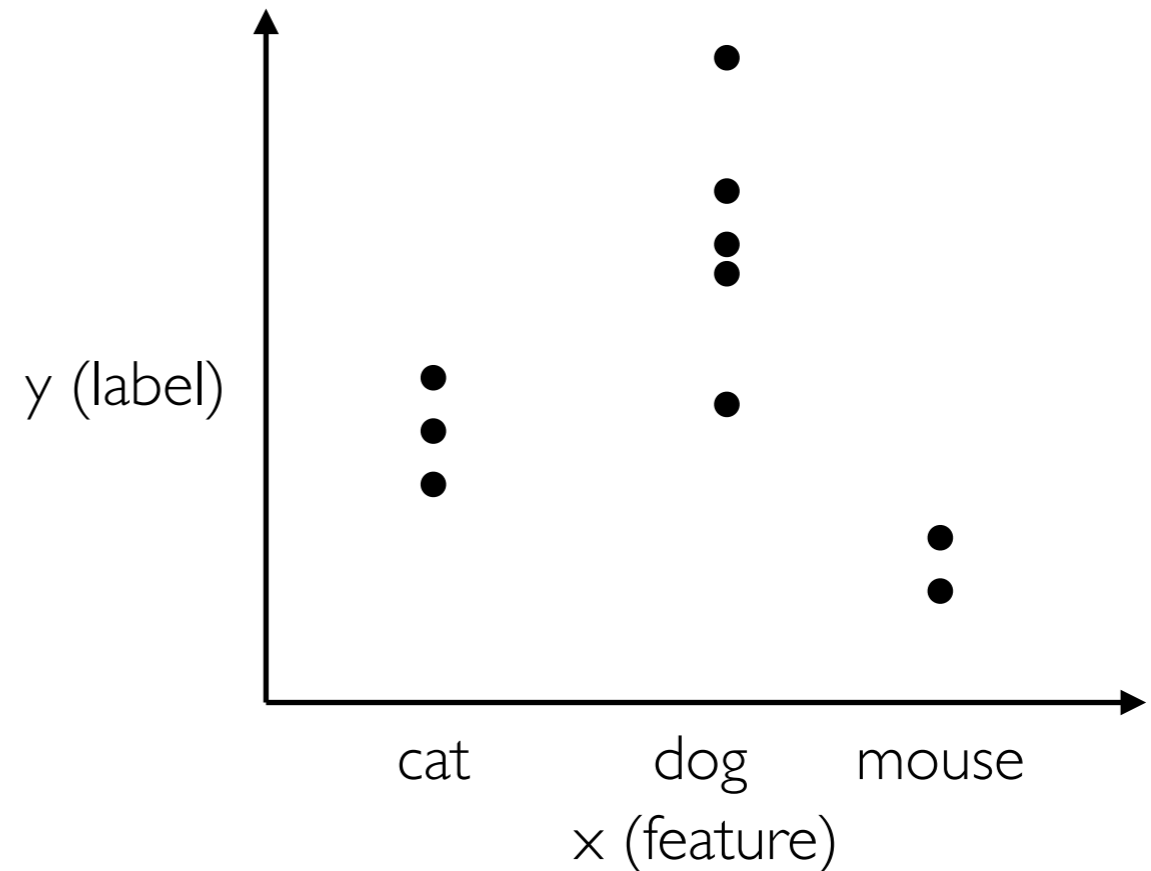
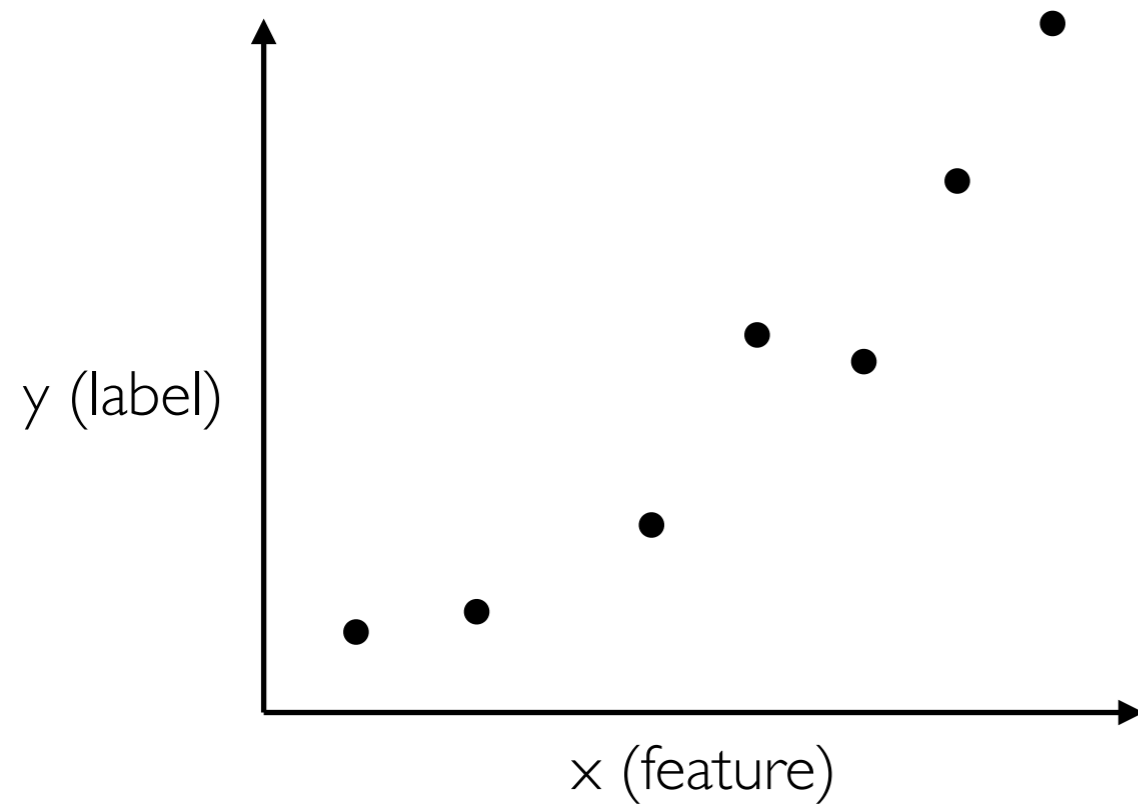
# Outline

BigQuery ML Basics

Feature Transformation

Cost Management

# Patterns and Features



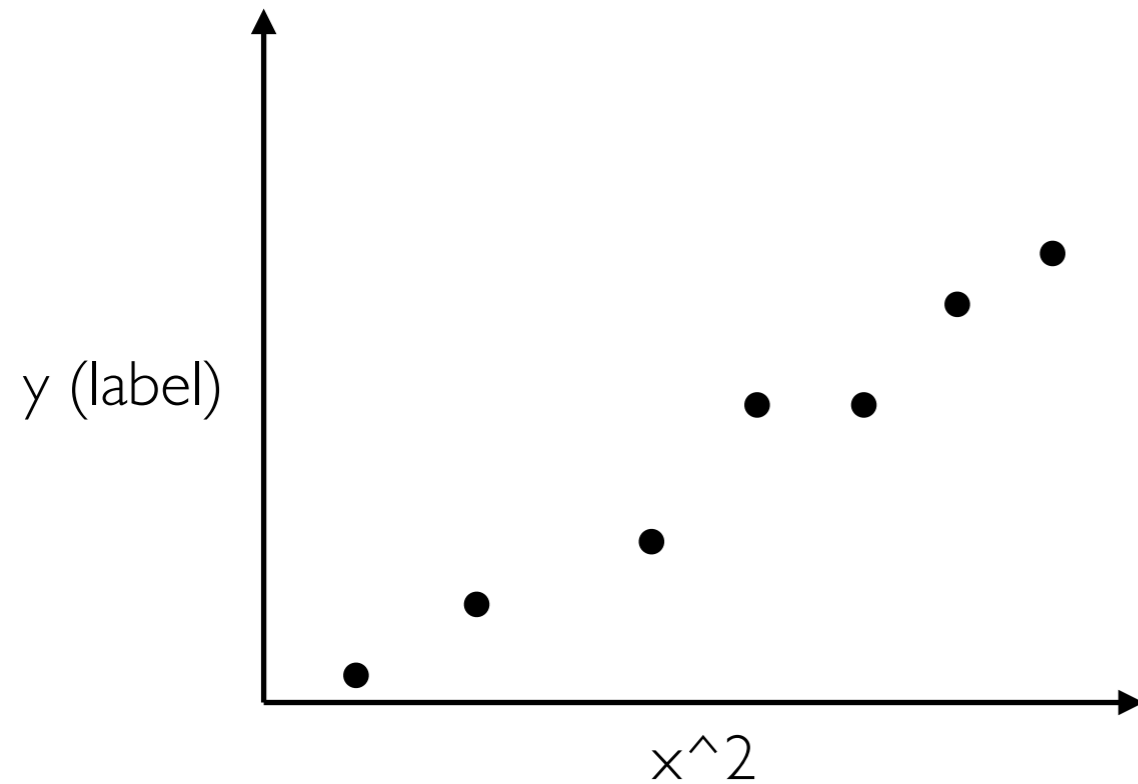
## non-linear patterns

- some models (e.g., DNNs) naturally handle this
- others (e.g., LinearRegression) do not

## categorical features

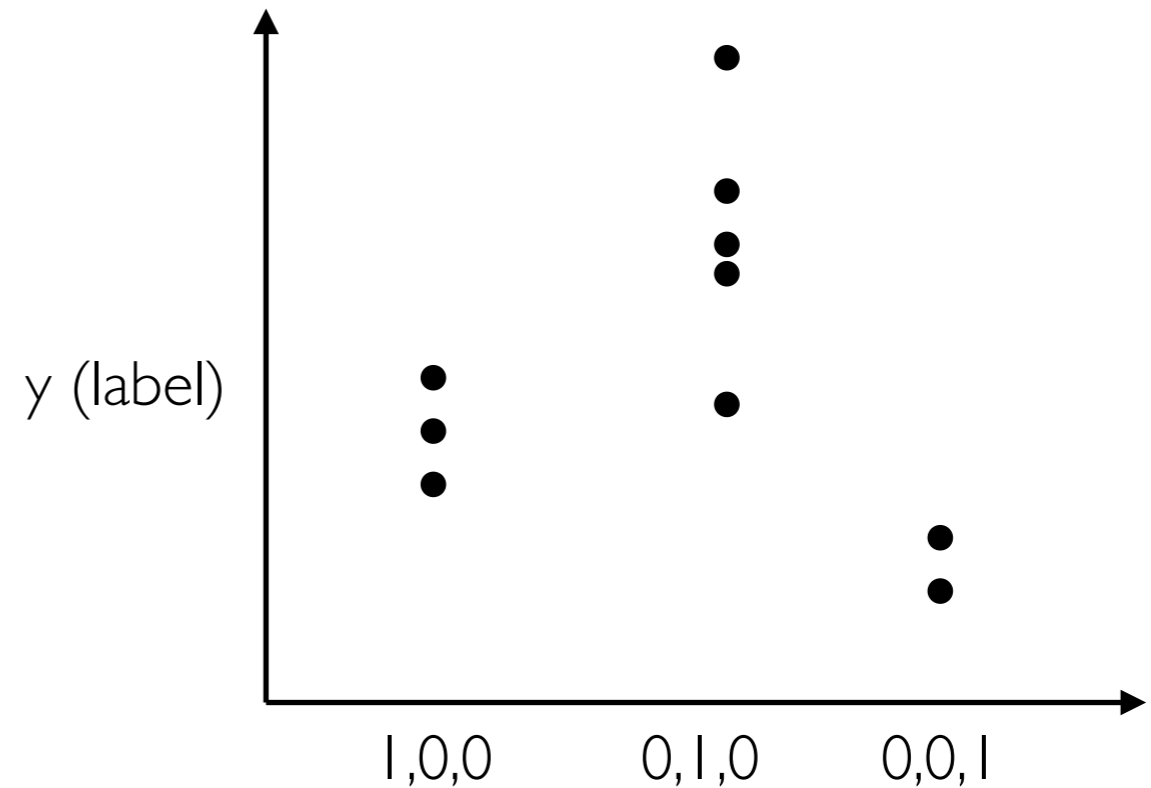
- some models (e.g., DTs) naturally handle this
- others (e.g., LinearRegression) do not

# Feature Transformation



## non-linear patterns

- can introduce new features than are computed as functions of originals (e.g.,  $x_2 = x^2$ )
- a linear model over the new features corresponds to a non-linear model over the originals



## categorical features

- encode categorical features as numbers (e.g., as matrix of zeros and ones for OneHot encoding)

**Demos**

# Outline

BigQuery ML Basics

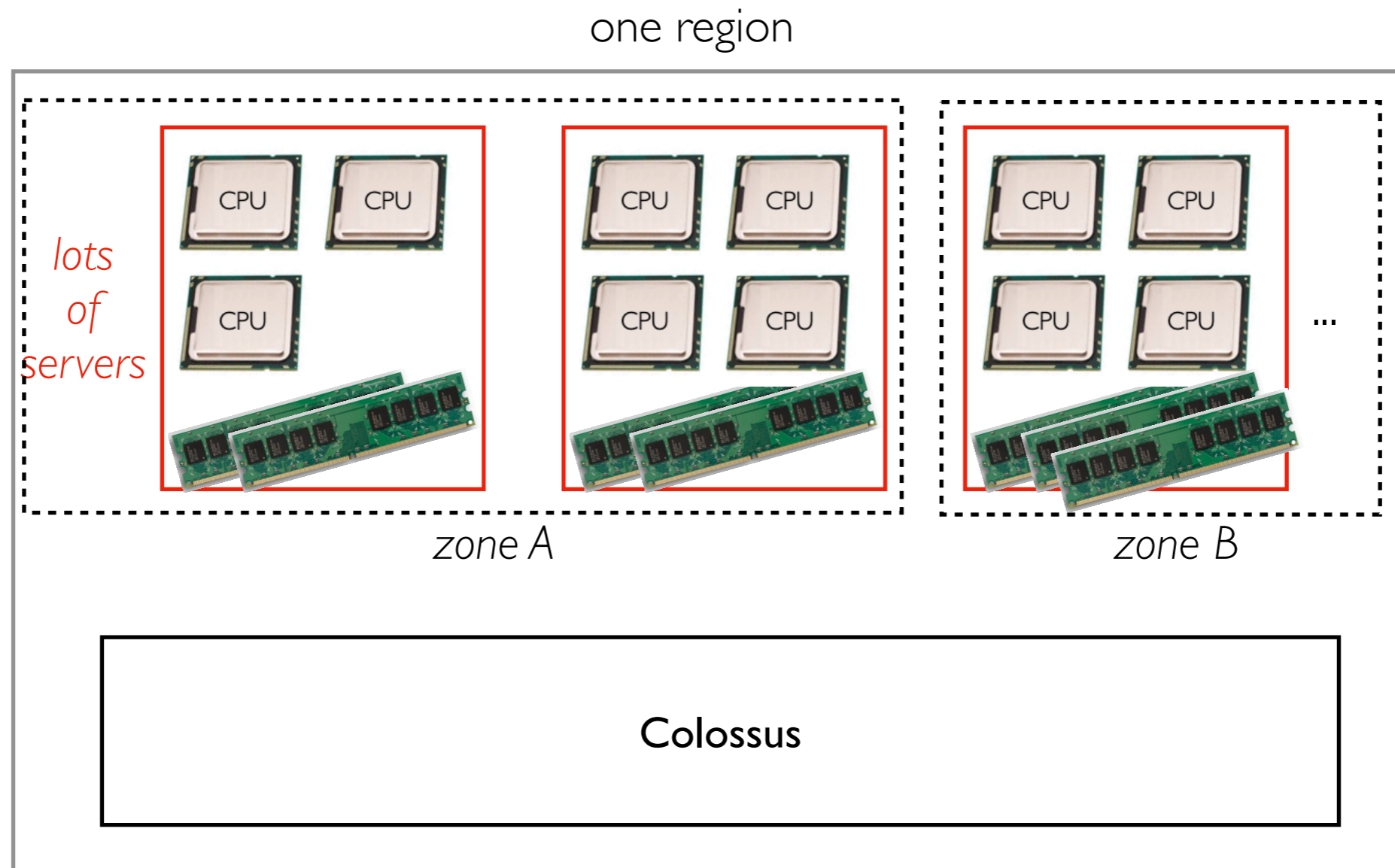
Feature Transformation

Cost Management



# Resources

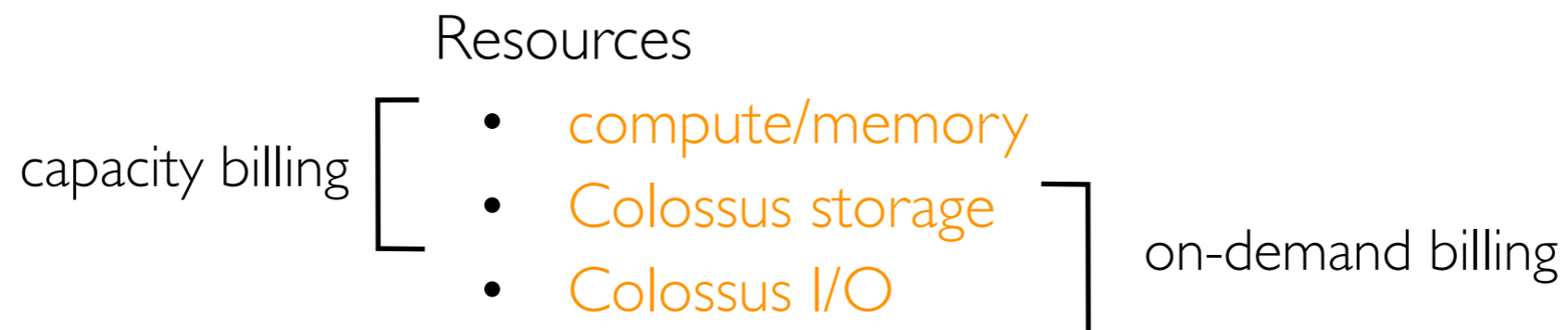
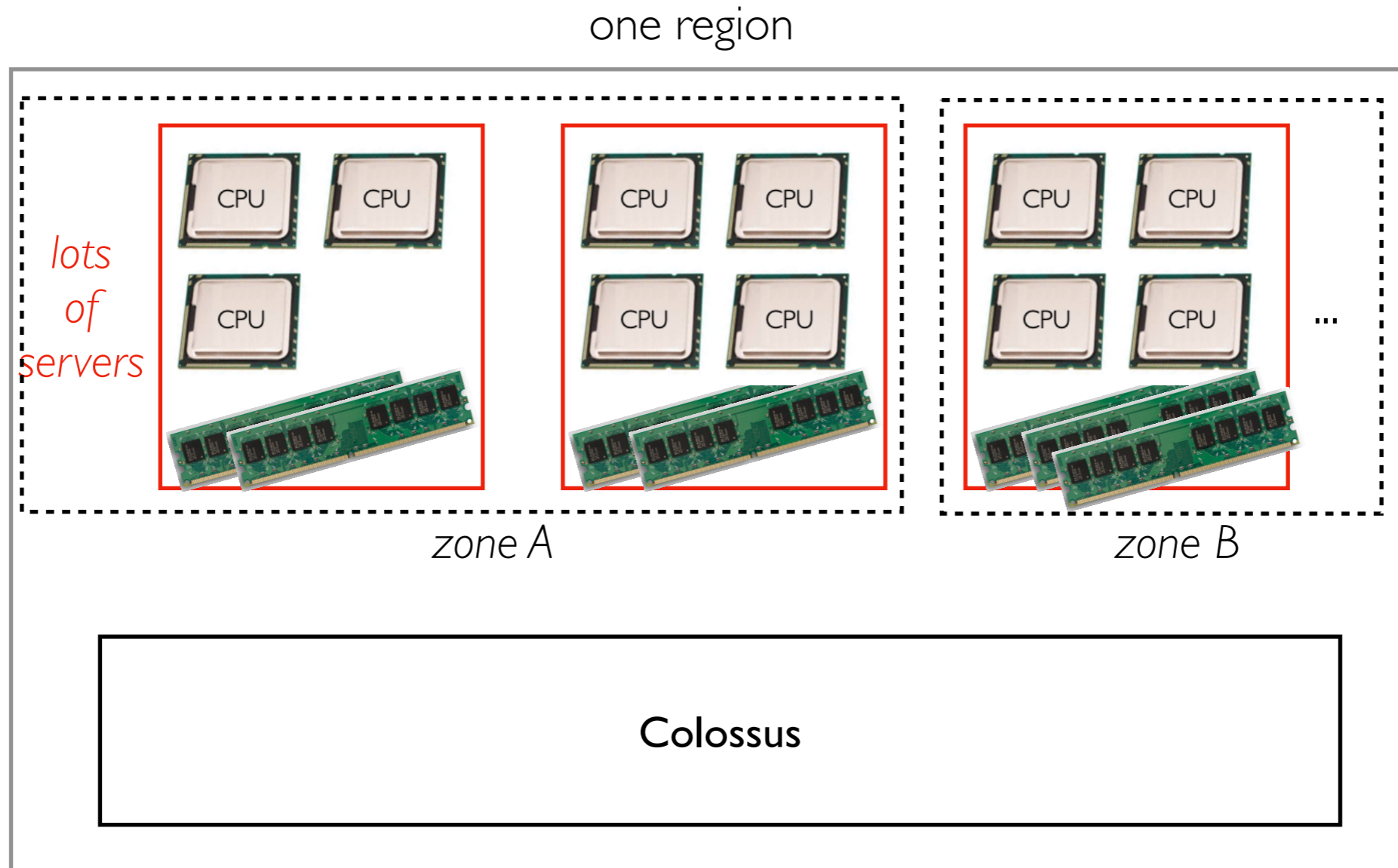
other regions...



- **Query engine:** Dremel running on many servers with lots of CPU+RAM
- **Storage engine:** Capacitor files in Colossus file system  
(not clear if Dremel+Colossus servers are co-located on same machines)

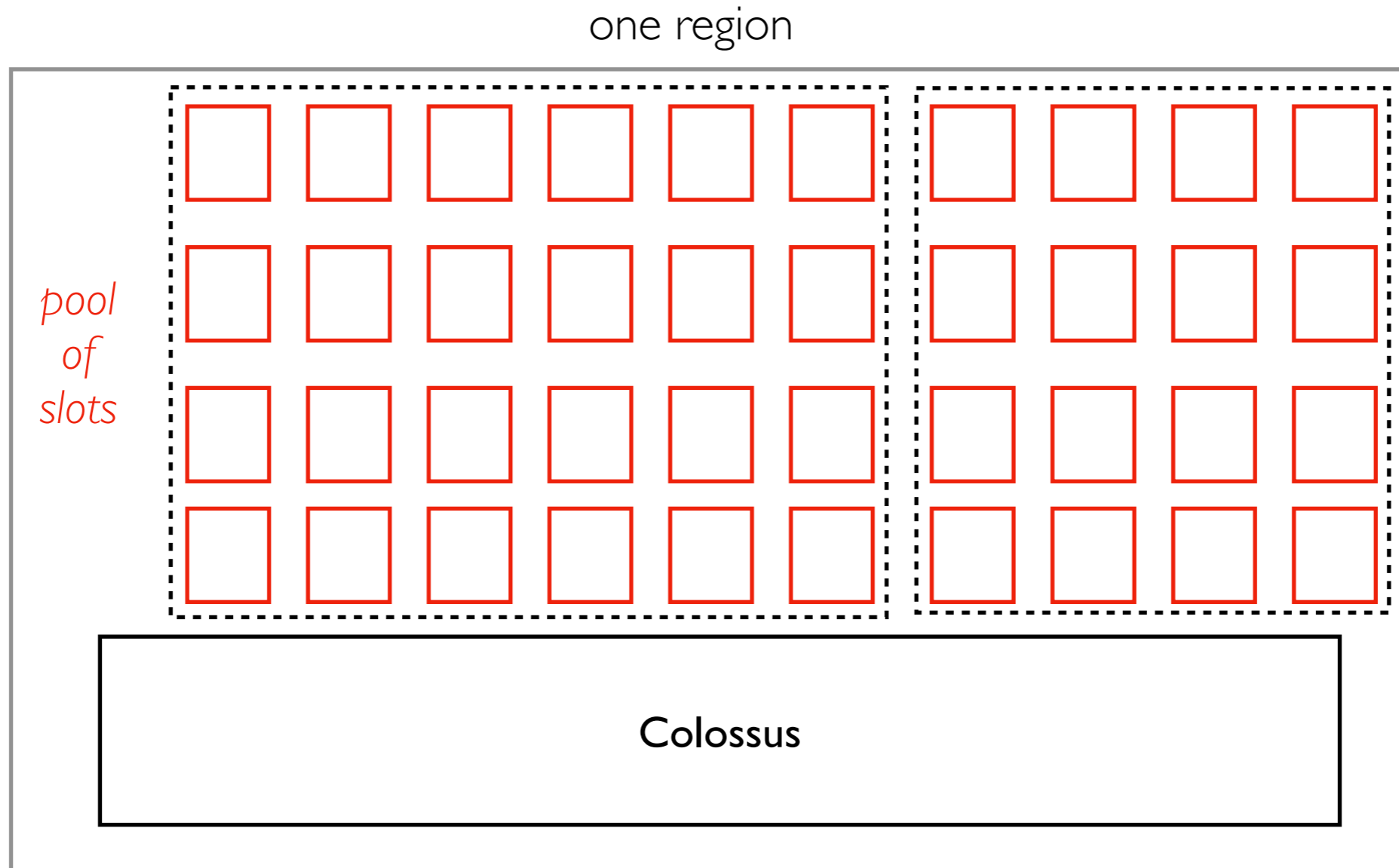
# Resources

other regions...



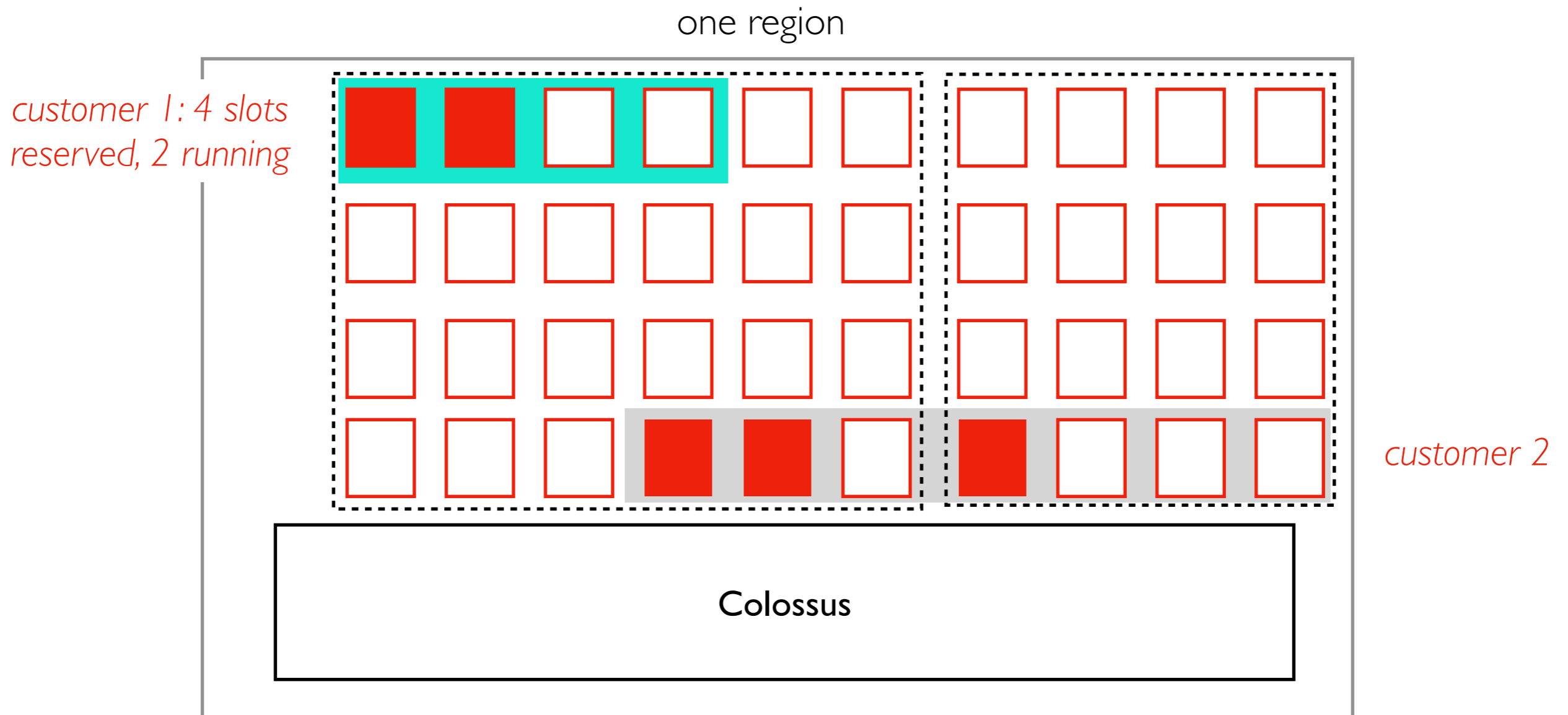
# BigQuery Slots

other regions...



- the compute and memory resources of the servers are broken down into a pool of "slots"
- a slot has approximately  $\frac{1}{2}$  cores and 1 GB of RAM
- if newer servers get added with faster CPUs or different core/memory ratios, the exact resources can change a bit

# Billing Model 1: Capacity Pricing (compute based)

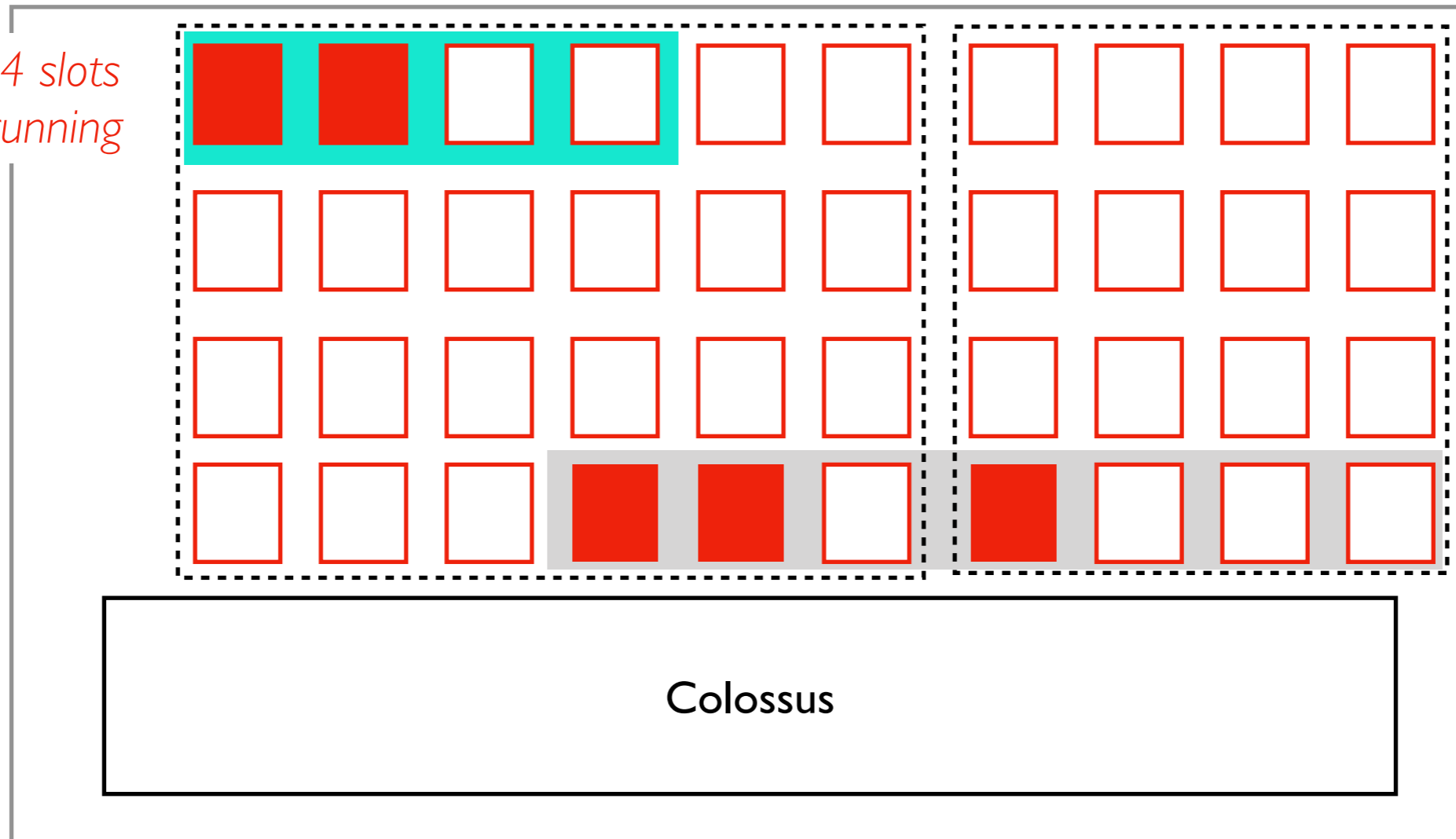


- customers can pay a fixed rate for slot capacity (about \$0.96 for 1 slot day)
- whether or not they use the slot does not affect the cost
- reservations aren't fixed to one location (execution will ideally happen near the data).
- *slightly more expensive than the e2-medium instances we used this semester, which have 2x compute and 4x memory resources (but not free Colossus I/O). But VMs are IaaS and BigQuery is PaaS.*

# Billing Model 1: Capacity Pricing (compute based)

one region

*customer 1: 4 slots reserved, 2 running*



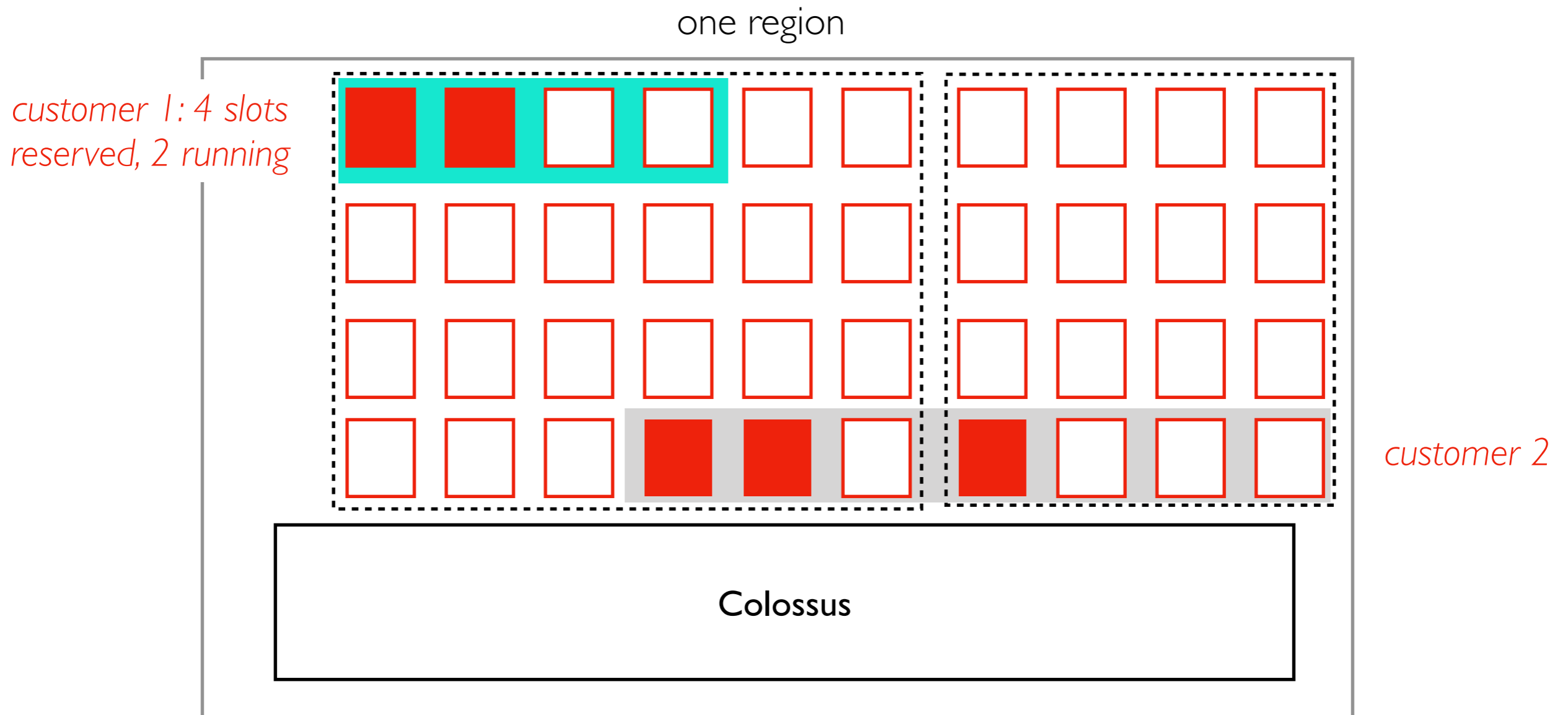
*customer 2*

Excess capacity cases:

- not reserved
- reserved, but not currently used

Billing Model 2 (On-Demand) draws from this excess...

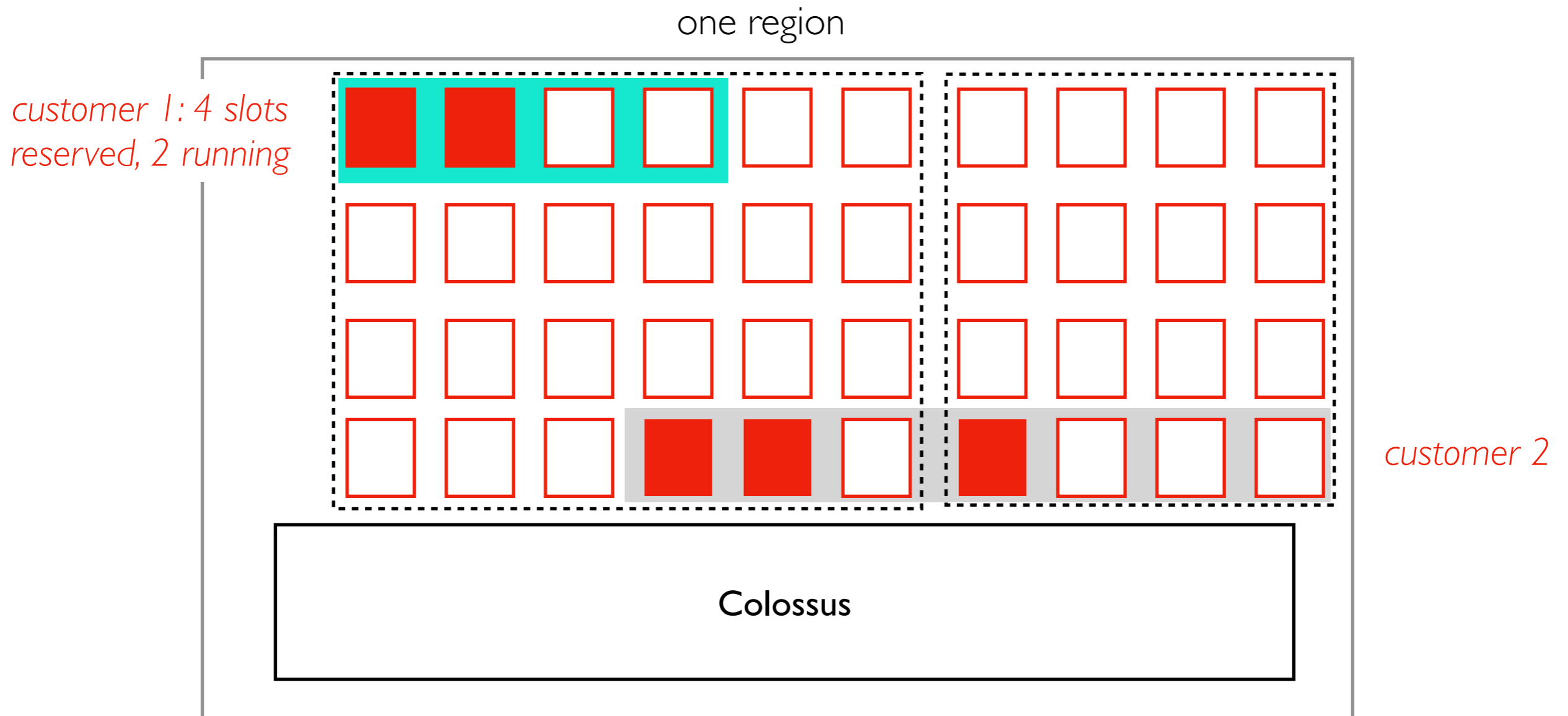
# Billing Model 2: On-Demand Pricing (I/O based)



Pricing:

- pay for Colossus I/O after free tier (about \$6.25/TB)
- slots (compute/memory) are free
- use whatever is left over from capacity-based usage (up to 2000 slots!)
- **preemptible**: a task running in a slot can be interrupted (if a reservation is suddenly needed or new on-demand jobs start -- want to share capacity between these fairly)

# Billing Model 2: On-Demand Pricing (I/O based)



Pricing:

- pay
  - slots
  - use
  - preemptible: a task running in a slot can be interrupted (if a reservation is suddenly needed or new on-demand jobs start -- want to share capacity between these fairly)
- BigQuery tasks are atomic and idempotent so we have exactly-once semantics. Don't want interrupted and restarted tasks to cause duplicate results.** (2000 slots!)

# Comparison

## Capacity Billing

- very predictable costs
- very predictable performance (other customers don't affect you)
- discounts if commit to buying lots of cores for long time (e.g., a year)
- pay when using nothing
- can't use lots of resources for a short while


## On-Demand Billing

- pay-as-you-go: use nothing, pay nothing
- if resources are available, you can use 1000 cores at once -- very fast!
- how to make sure you don't accidentally spend more than intended?



# Estimating/Capping On-Demand Costs

Filter Metric : bigquery.googleapis.com/quota/query/usage  Enter property name or value

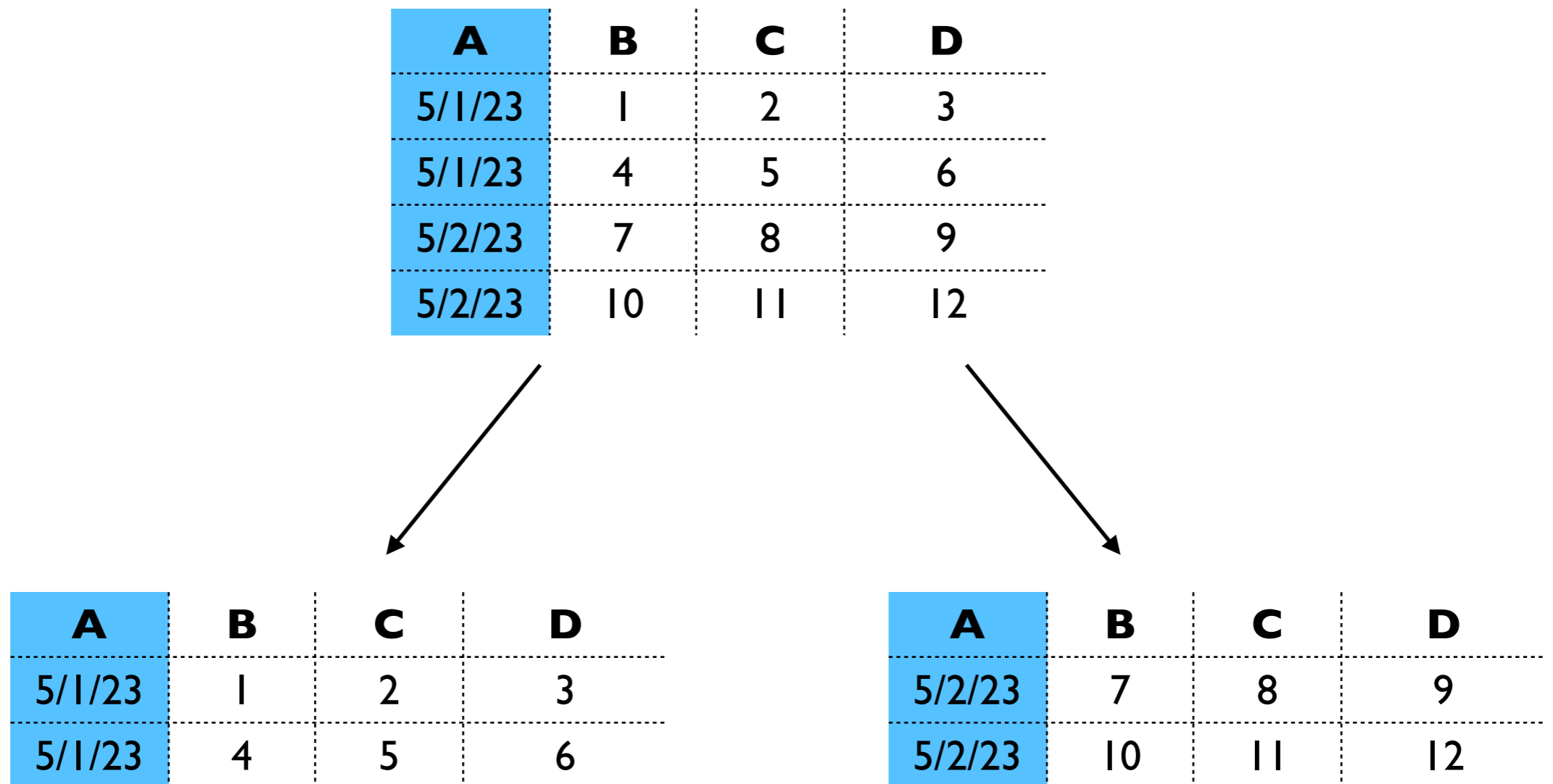
| <input type="checkbox"/> | Quota                        | Dimensions (e.g. location) | Limit                   | Current usage percentage ↓   | Current usage |
|--------------------------|------------------------------|----------------------------|-------------------------|--|---------------|
| <input type="checkbox"/> | Query usage per day          |                            | 1,048,576 MiB (1 TiB) ⓘ |  0% | 0 MiB         |
| <input type="checkbox"/> | Query usage per day per user |                            | Unlimited               | – ⓘ  |               |

## Options:

- **Limit per day:**  
<https://console.cloud.google.com/iam-admin/quotas>
- **Estimate before run:**  
`job_config=bigquery.QueryJobConfig(dry_run=True)`
- **Set max per query:**  
`bigquery.QueryJobConfig(maximum_bytes_billed=200*1024**2)`
- **See most expensive queries:**  
`cs320-f21.region-us.INFORMATION_SCHEMA.JOBS_BY_PROJECT`

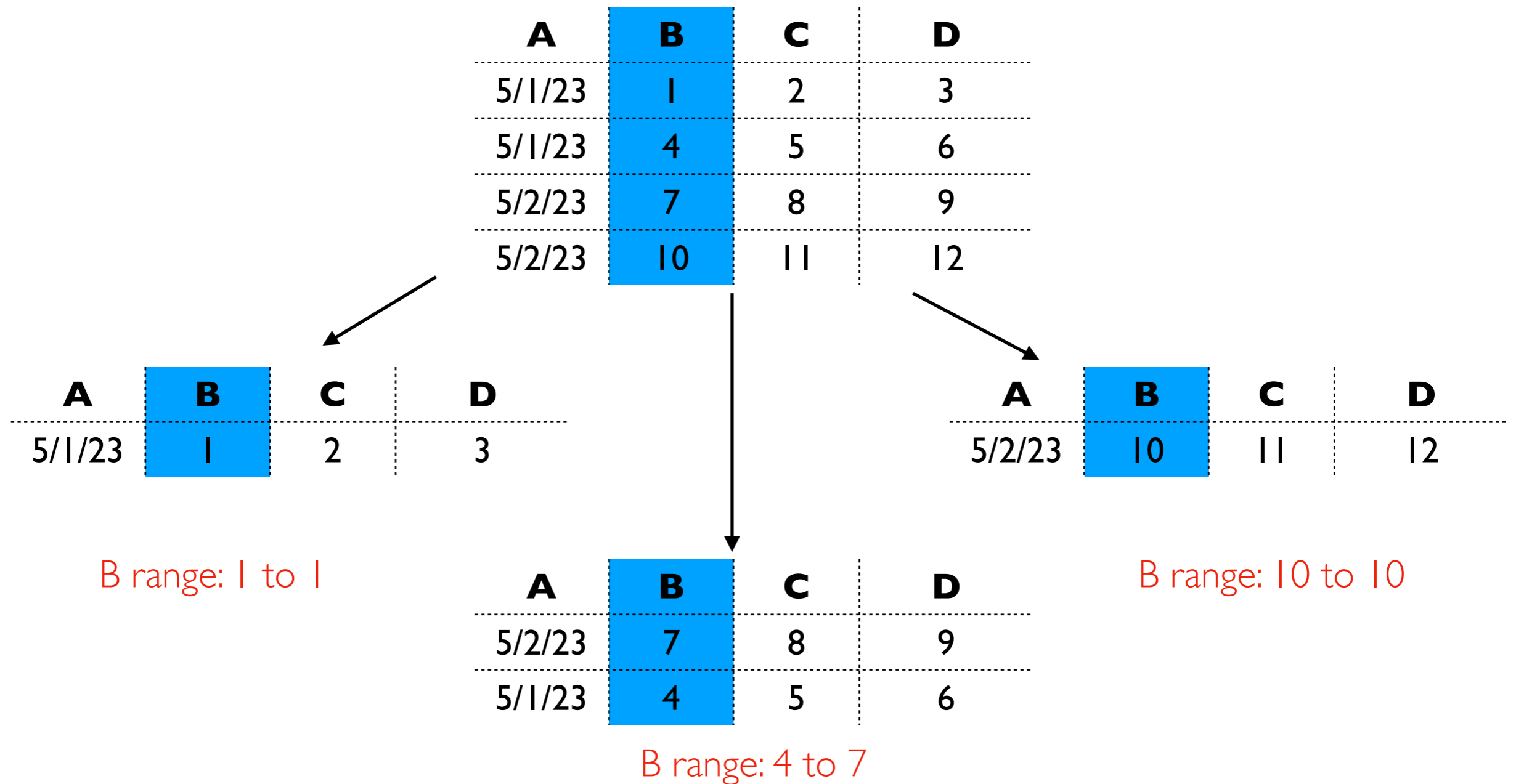
Demos

# Partitioning



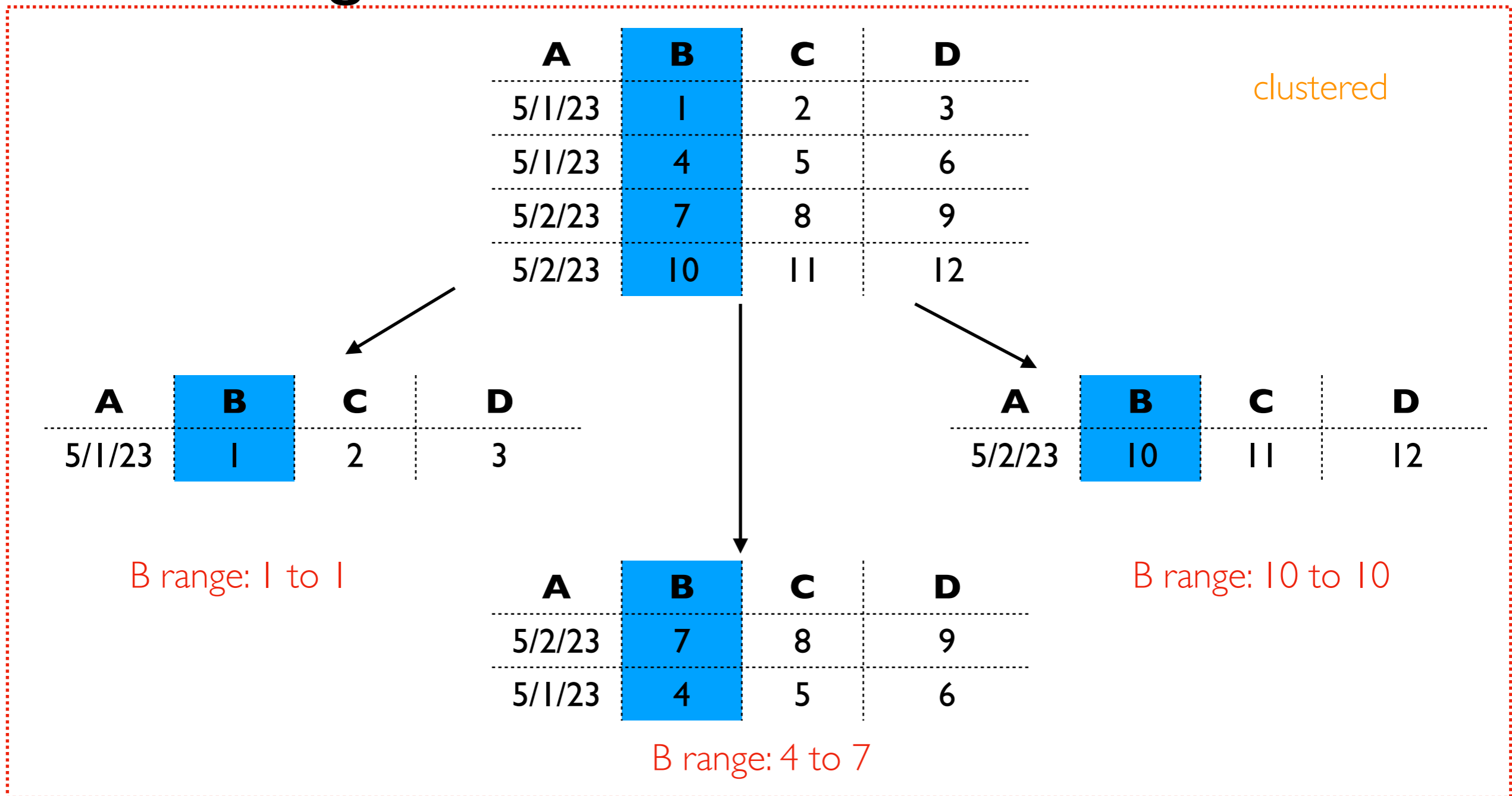
- each unique value in a partition column corresponds to a partition (basically a mini table)
- WHERE filters can limit which mini tables need to be read (saving I/O cost)
- limited options for types (e.g., ints, dates)
- only works well when substantial data per partition

# Clustering



- semi sorted: sub files are non overlapping on cluster key, but no order within file
- all types, combinations of columns possible
- some queries will be cheaper because they can look at subset of files

# Clustering



- some min ratio of data is clustered
- don't want few new rows to force total reorg

| A      | B  | C | D |
|--------|----|---|---|
| 5/2/23 | 5  | 1 | 2 |
| 5/1/23 | 12 | 3 | 4 |

unclustered

Demos