Lecture 4: Data - TF Data, TF Records

Advanced Practical Data Science, MLOps

AC215

Pavlos Protopapas
Institute for Applied Computational Science, Harvard

Communication

- Please see project milestone submission instructions on Ed
- Please fill your groups here (asap), for those not in spreadsheet, we allocate you tonight.
- Exercise 1 due 09/16 2PM (submit on canvas)
- Milestone 1 due 09/17 (submit on Github private repo, see Ed)
- Quiz 2 due 09/21 2PM.
- Quiz 3 due ??

Outline

1. Recap
2. Need for building efficient pipelines
3. TensorFlow Data
4. TensorFlow Records

Outline

1. **Recap**
2. Need for building efficient pipelines
3. TensorFlow Data
4. TensorFlow Records

Motivation

The 3 components for better Deep Learning

More Data

Better Models

Faster Hardware

Motivation

The 3 components for better Deep Learning

More Data

- Storage
- Processing
- Input to Training

Better Models

- SOTA Models
- Transfer Learning
- Distillation
- Compression

Faster Hardware

- Scaling data processing
- GPU, TPU
- Multi GPU Server Training

Motivation: Data Size

Challenges:

- Medium datasets will not all fit in memory (RAM)
- Large datasets will not fit in disk (Hard drive)

Solution:

- Building data pipelines
  - Read data in batches which can fit in RAM
  - Feed data in batches to GPU
  - Read data from big data store in batches so not all data need to be present in local hard drive

Motivation: Data Size

Tools:

- Dask
- Google Cloud Storage (Big data store)
- **TensorFlow Data**
- **TensorFlow Records**

Outline

1. Recap
2. **Need for building efficient pipelines**
3. TensorFlow Data
4. TensorFlow Records

Need for building efficient pipelines

Why is training slow?

Need for building efficient pipelines

Why is training slow?

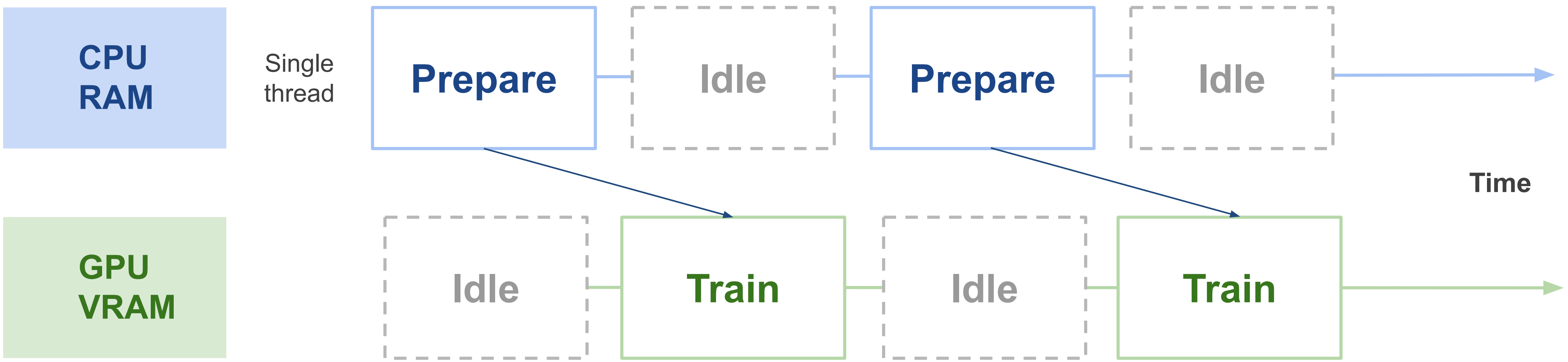
GPU Starvation

Need for building efficient pipelines

GPU Starvation:

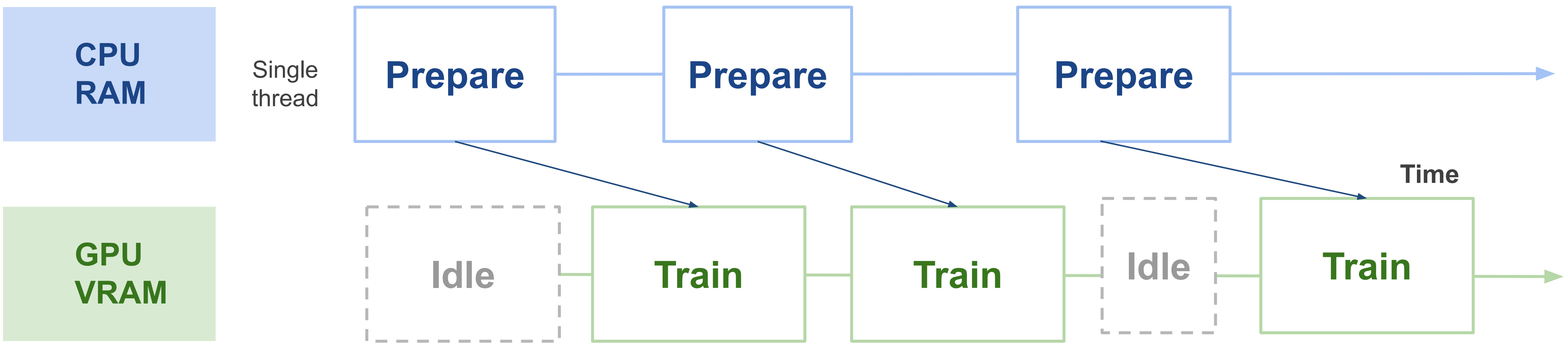
- Most times, data input to models take a long time
- Data process using CPU & RAM are the bottlenecks
- GPUs are expensive and not fully utilized

Need for building efficient pipelines



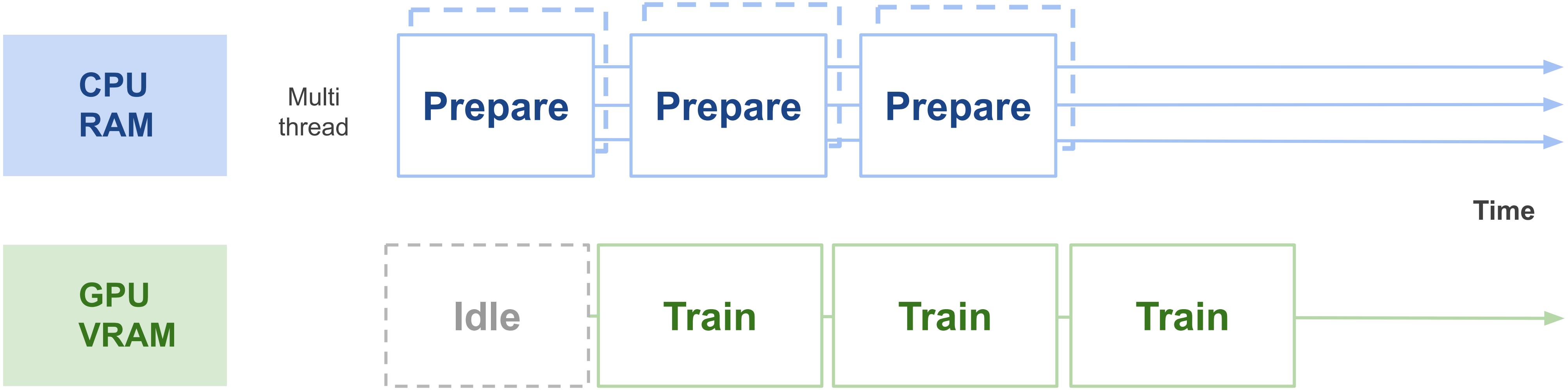
Single threaded CPU and single GPU working sequentially with **NO** prefetching

Need for building efficient pipelines



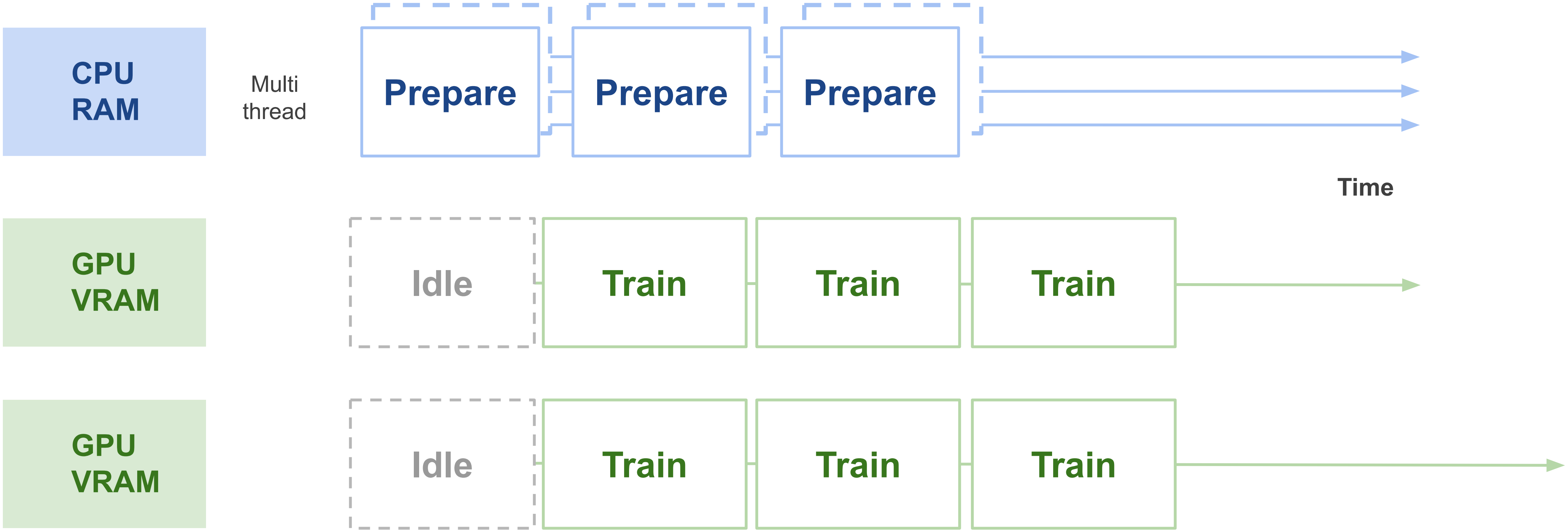
Single threaded CPU and single GPU working with prefetching

Need for building efficient pipelines



Multi threaded CPU and single GPU working with prefetching

Need for building efficient pipelines



Multi threaded CPU and multi GPU working with prefetching

Need for building efficient pipelines

To build efficient pipelines we can use:

- TensorFlow Data
- TensorFlow Records

Outline

1. Recap
2. Need for building efficient pipelines
3. **TensorFlow Data**
4. TensorFlow Records

Consuming Data in Models

Source

csv files

Images

Text

Local disk
Cloud storage

Less than **2-4 GB**

Consuming Data in Models

Source

csv files

Images

Text

Data

x	y
[0,0,....0,0]	[0]
[0,1,....1,1]	[1]
...	...
...	...
...	...
...	...
...	...
...	...
...	...
[1,1,....1,0]	[1]
[1,0,....1,0]	[0]

Local disk
Cloud storage

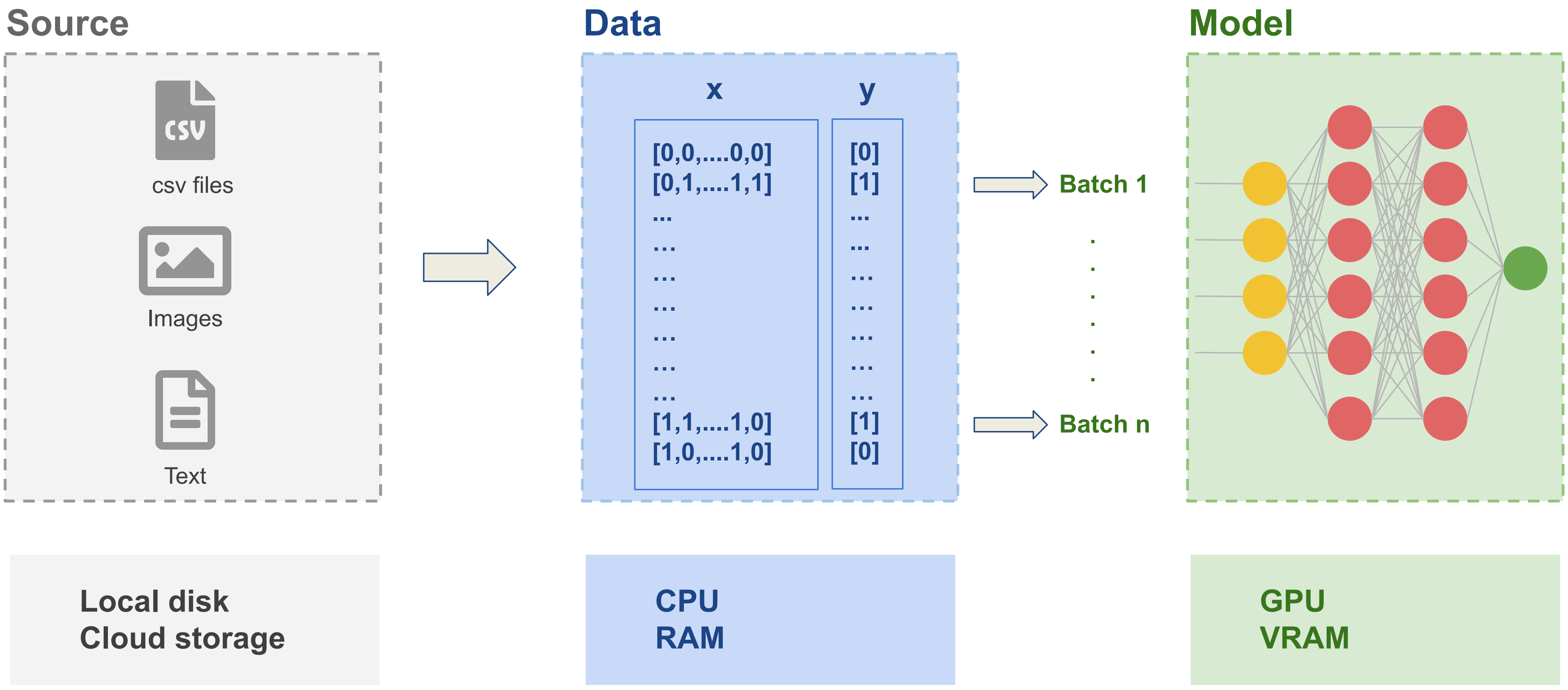
CPU
RAM

Less than **2-4 GB**

All data loaded into memory

Consuming Data in Models

CS109 scenario



Less than **2-4 GB**

All data loaded into memory

Model trained in batches of data

Consuming Data in Models

What do we do when our dataset size is greater than 5 GB?

Consuming Data in Models

Source

csv files

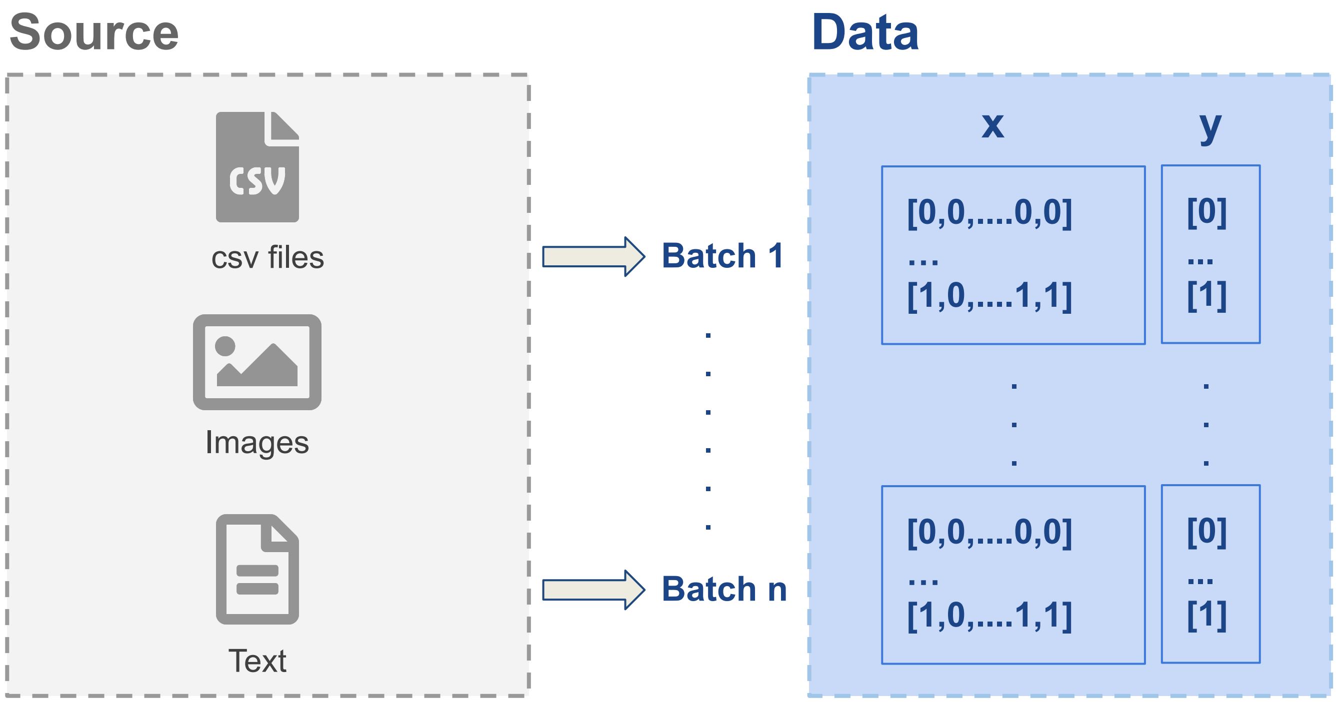
Images

Text

Local disk
Cloud storage

Greater than **5 GB**

Consuming Data in Models



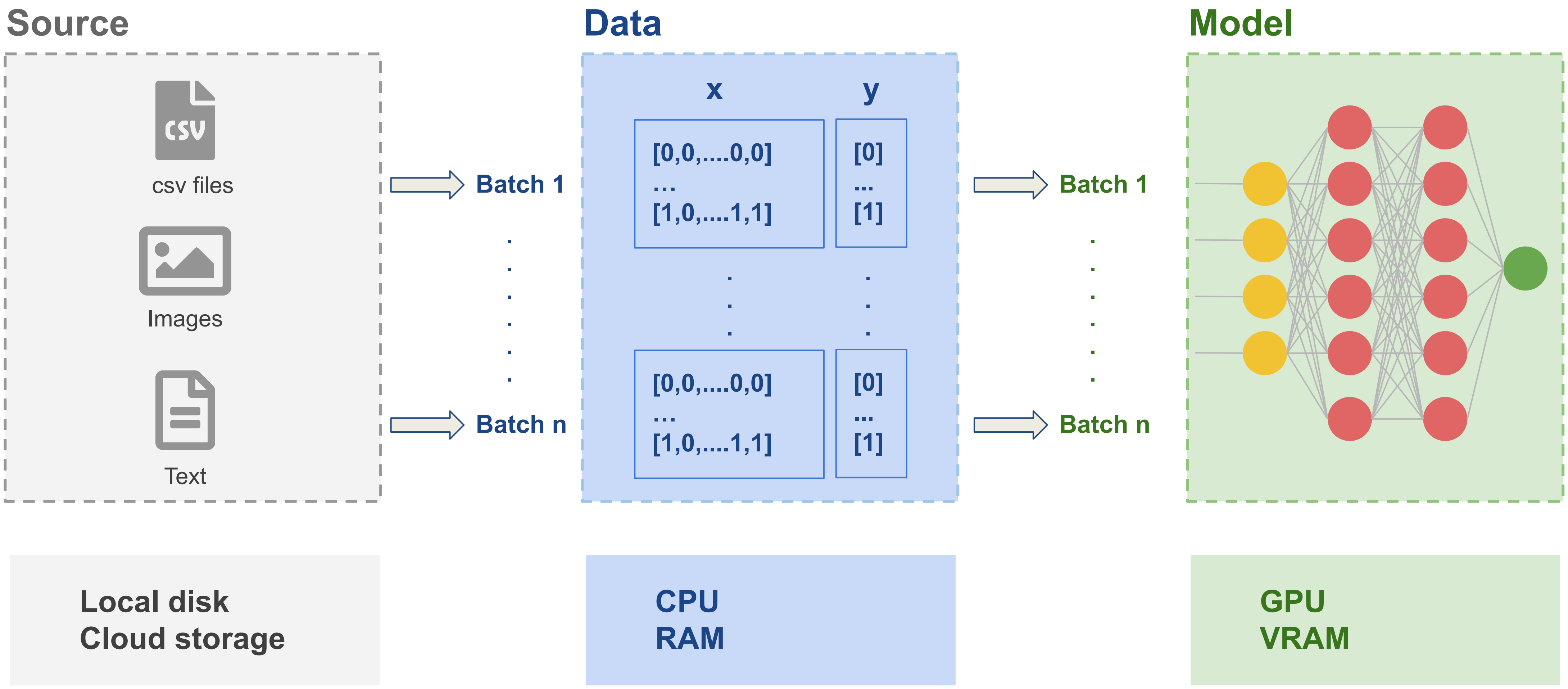
Local disk
Cloud storage

CPU
RAM

Greater than **5 GB**

Data loaded in batches

Consuming Data in Models



Greater than **5 GB**

Data loaded in batches

Model trained in batches of data

Consuming Data in Models

So we need a way to efficiently:

- Extract data from various data sources
- Transform data for pre-processing/ augmentation
- Load data ahead of training in GPU

Consuming Data in Models

So we need a way to efficiently:

- **Extract** data from various data sources
- **Transform** data for pre-processing/ augmentation
- **Load** data ahead of training in GPU

TensorFlow Data to the rescue...

TensorFlow Data

What is TensorFlow Data:

- TensorFlow makes building data pipelines easy using **tf.data**
- Build flexible and efficient data pipelines
- Read data in batches
- Parallelize data reads using CPU
- Does not load all data to memory and streams data to model
- Streams data from either local file system or distributed file systems

TensorFlow Data

What you do:

- Create a Dataset object
- Specify where to get the data
- Describe how to transform it (Extract, process, augment)

tf.data takes care of:

- Multi threading
- Queuing
- Batching
- Prefetching
- Shuffling

TensorFlow Data

All you need to do is:

```
train_ds = tf.data.Dataset.from_tensor_slices(...)
```

Create Dataset

```
# Create TF Dataset
train_data = tf.data.Dataset.from_tensor_slices((train_x, train_processed_y))
validation_data = tf.data.Dataset.from_tensor_slices((validate_x, validate_processed_y))
```

TensorFlow Data

All you need to do is:

```
train_ds = tf.data.Dataset.from_tensor_slices(...)
```

```
train_ds = train_ds.shuffle()
train_ds = train_ds.map(<pre-process-function>)
train_ds = train_ds.batch()
train_ds = train_ds.prefetch()
```

Processing logic

TensorFlow Data

All you need to do is:

```
train_ds = tf.data.Dataset.from_tensor_slices(...)
```

```
train_ds = train_ds.shuffle()
train_ds = train_ds.map(<pre-process-function>)
train_ds = train_ds.batch()
train_ds = train_ds.prefetch()
```

```
model.fit(train_ds,...)
```

Train

TensorFlow Data

We can build data pipelines using a variety of operations

Extract

Read data:
- CSVs
- Images
- Text files
- Binary files
- SQL databases
- Google Big Query
- Local store
- Remote store

Transform

Prepare data:
- Clean
- Normalize
- Encode
- Embeddings
- Augment
- Shuffle
- Batch
- Repeat

Load

Pass data to Device:
- Training / Inference
- CPUs or GPUs or TPUs
- Parallelize
- Prefetch
- Cache

TensorFlow Data

So in summary **tf.data** will help you:

- Work with large amounts of data
- Read from different data formats
- Perform complex transformations
- Build efficient data pipelines to reduce training time

Outline

1. Recap
2. Need for building efficient pipelines
3. TensorFlow Data
4. **TensorFlow Records**

Consuming Data in Models

Source

csv files

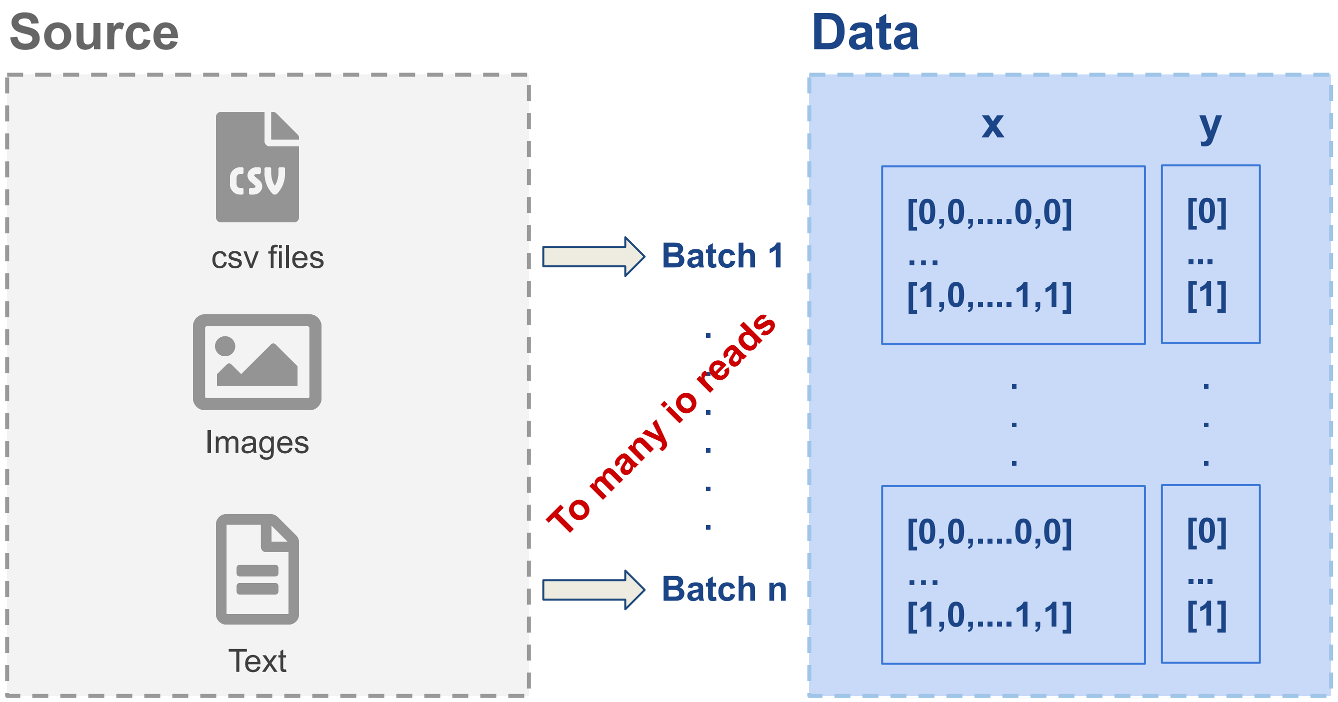
Images

Text

Local disk
Cloud storage

Greater than **30 GB**

Consuming Data in Models



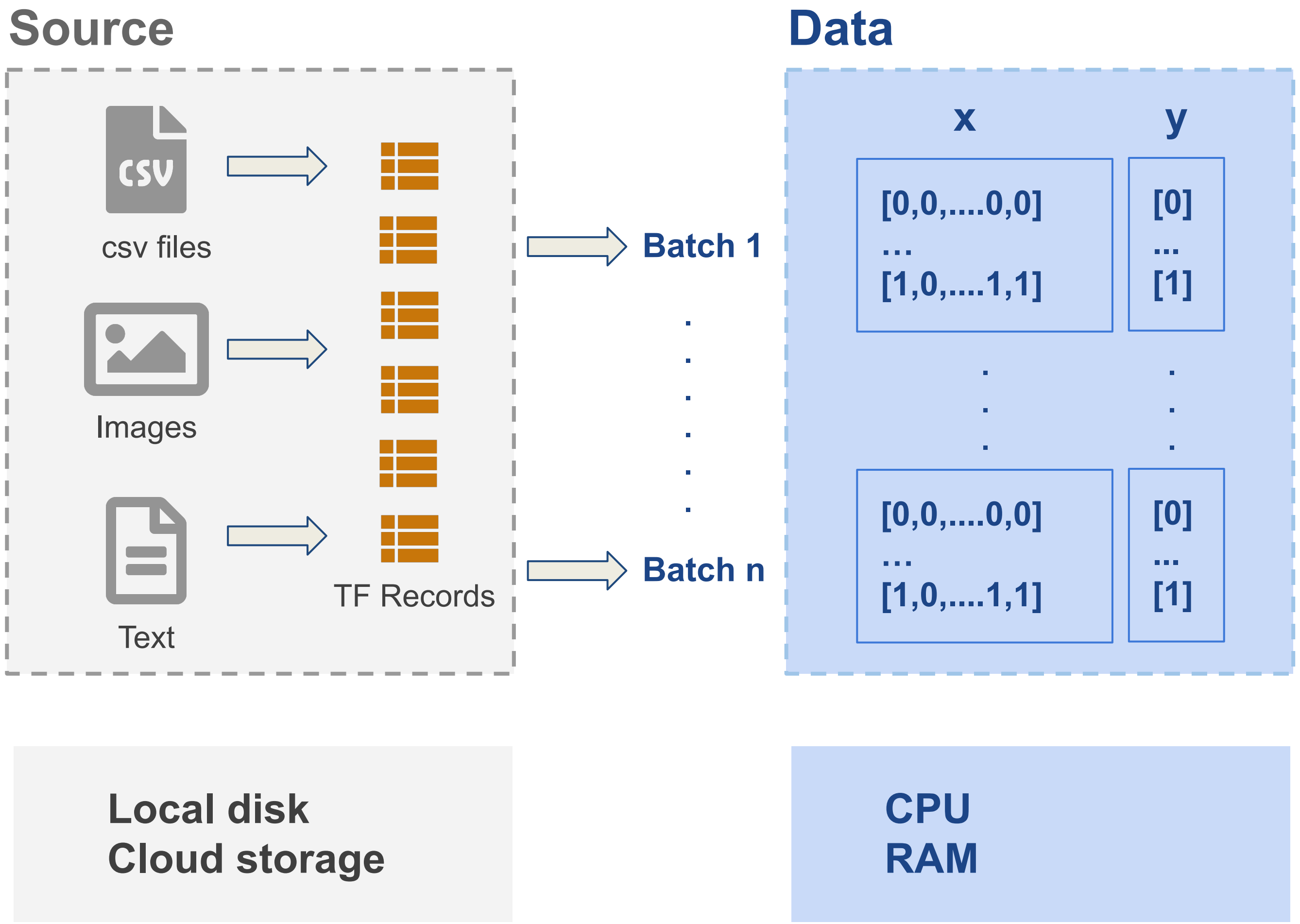
Local disk
Cloud storage

CPU
RAM

Greater than **30 GB**

Data loaded in batches

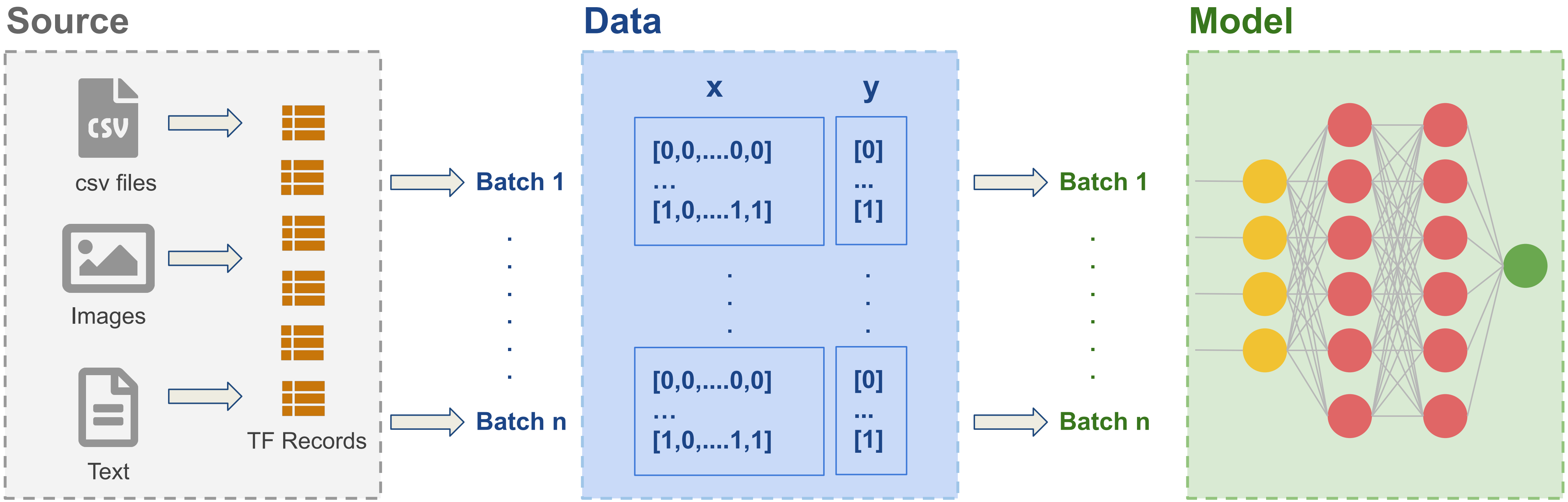
Consuming Data in Models



Greater than **30 GB**

Data loaded in batches

Consuming Data in Models



Local disk Cloud storage	CPU RAM	GPU VRAM
Greater than **30 GB**	Data loaded in batches	Model trained in batches of data

TensorFlow Records

What are TensorFlow Records:

- **TFRecord** is TensorFlow's format for large amount of data
- It is a binary format defined using protocol buffers (protobuf)*
- Data is compressed and very efficient to read
- Use **TFRecord** when reading using `tf.data` is a bottleneck to training

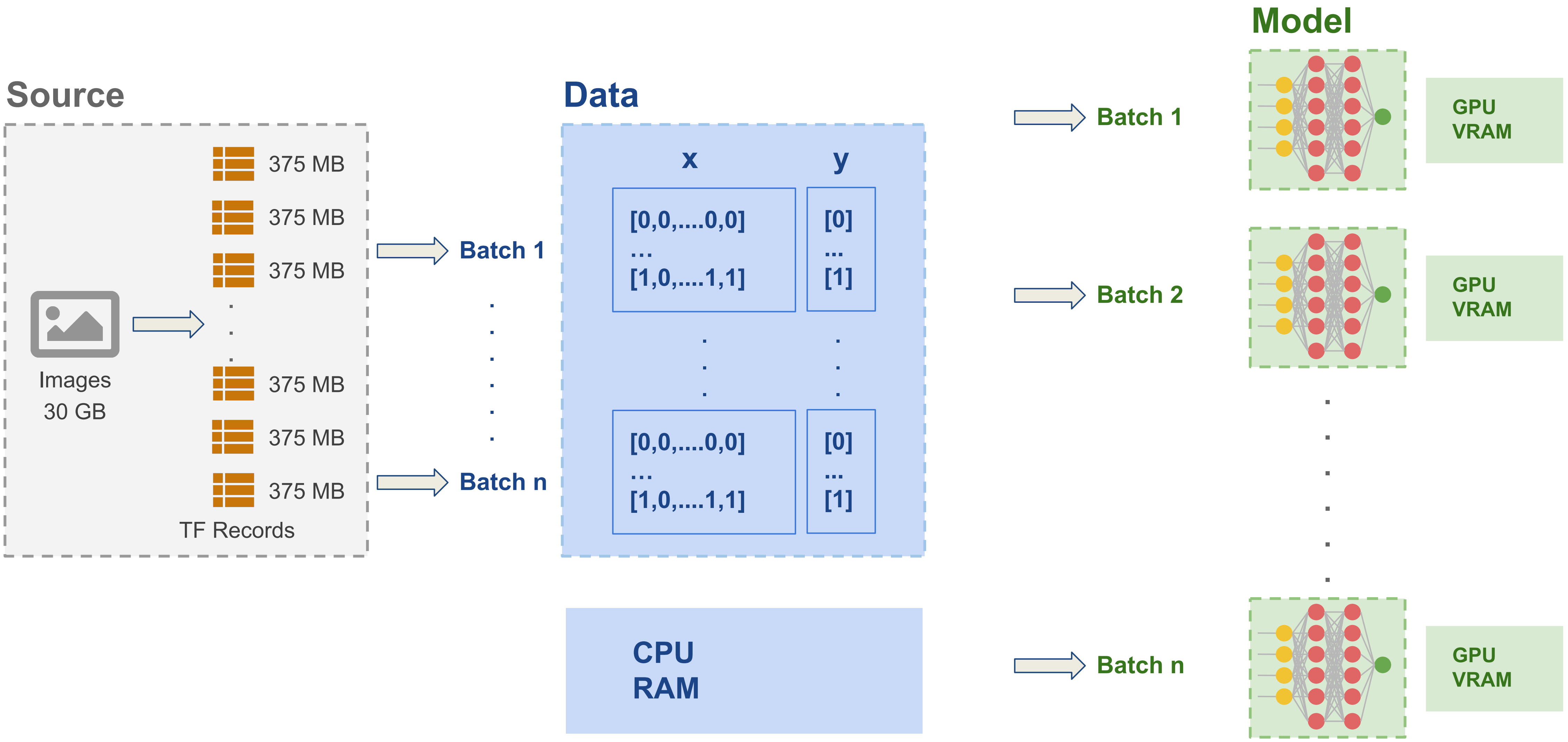
*Protocol buffers are a portable, extensible, and efficient binary format developed and open sourced by Google

TensorFlow Records

Why use TFRecords:

- **TFRecords** is used to shard your data across multiple files
- Parallelize I/O reads across one or more training servers
- Each file size should be 10 MB to ideally > 100 MB
- For example if you have 30 GB of data and 8 training servers:
  – Number of shards = 10 * 8 = 80
  – Shard size = 30,000/(10*8) = 375MB

TensorFlow Records+**Data**

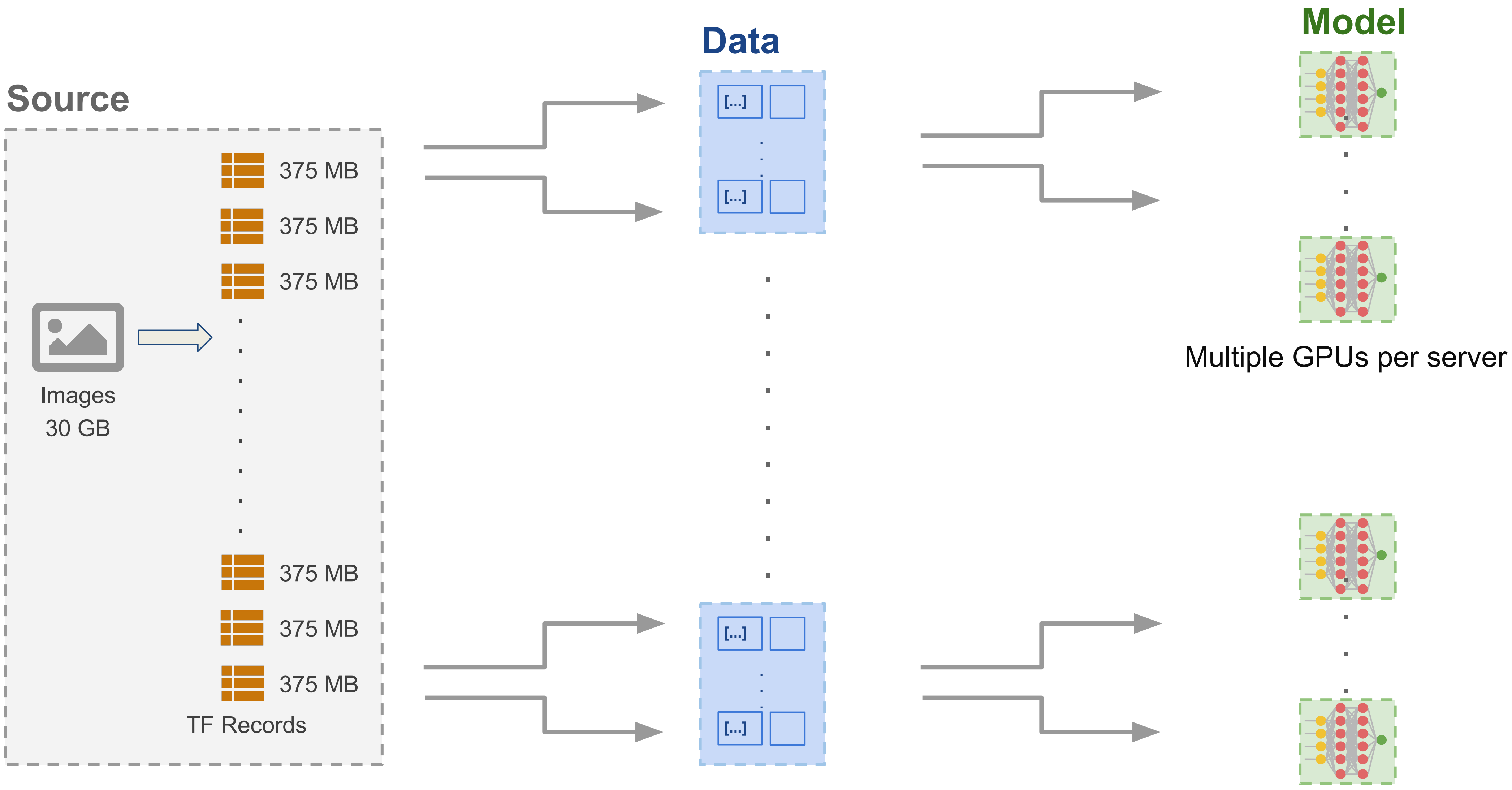


Data stored as TFRecords

1 Training Server

Model trained on multiple GPUs

TensorFlow Records+Data



Data stored as TFRecords

n Training Servers

TensorFlow Records

Creating TFRecords:

- **TFRecords** consists of files that are composed of a series of **tf.Example** messages
- **tf.Example** is a {"key": value} mapping where key is the feature name, and value is its binary representation
- For example to store image dataset, serialize all image attributes along with the label in a TFRecord file

```
{
      "image":   image.bytes(),
      "height":  image.height,
      "width":   image.width,
      "channel": image.channel,
      "label":   label
}
```

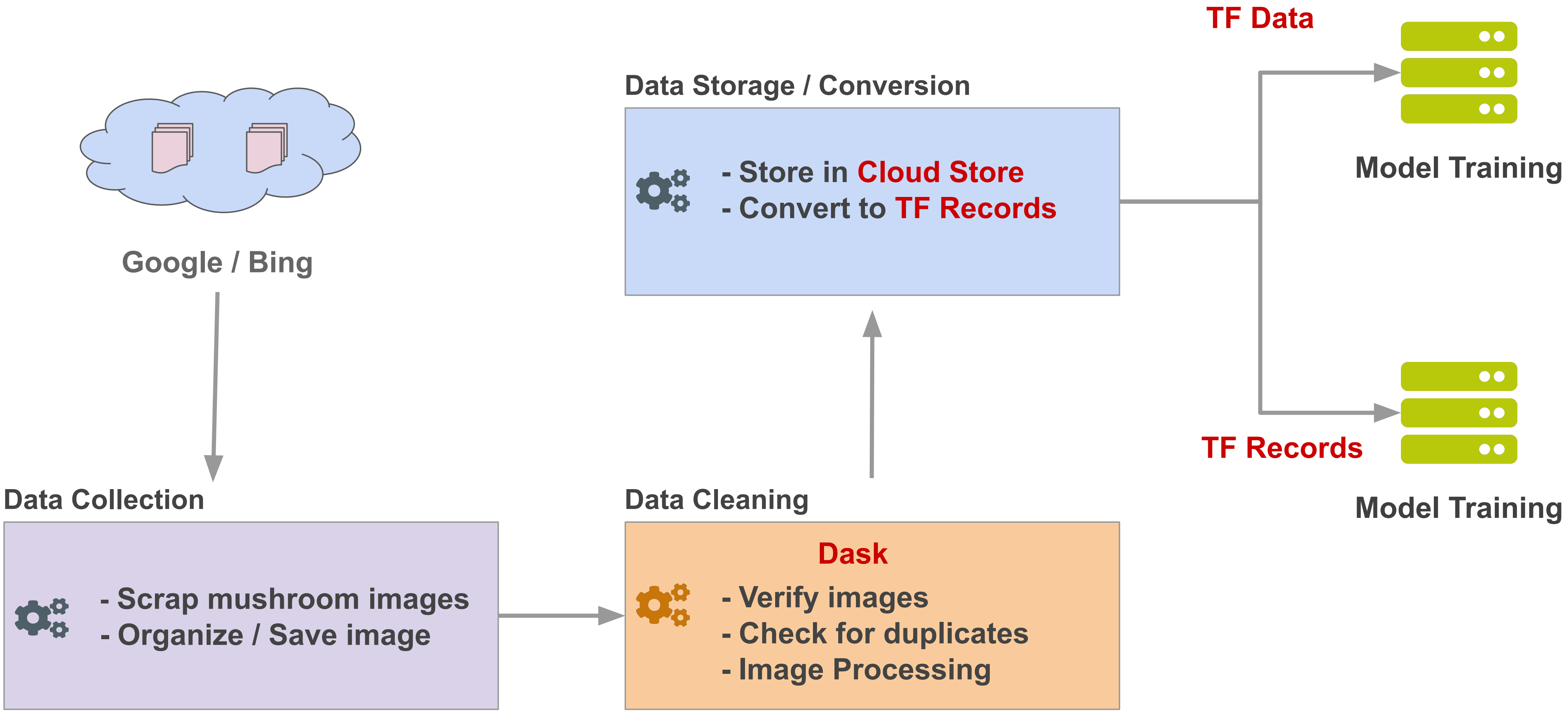
TensorFlow Records

Creating TFRecords:

```
{
      "image":   image.bytes(),
      "height":  image.height,
      "width":   image.width,
      "channel": image.channel,
      "label":   label
}
```

The image is stored as binary and not using any compression format. So files can be read linearly as a sequence of bytes without a need to decode images. This saves time to read but using more disk space.

Putting it all together



THANK YOU