Building Machine Learning Models in Spark 2

MACHINE LEARNING PACKAGES: SPARK.MLLIB VS. SPARK.ML

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Spark 1.x provided powerful support for ML in spark.mllib

Spark 2.x goes further with spark.ml

Faster execution

Ease of hyperparameter tuning

Both libraries offer powerful support

Prerequisites and Course Outline

Prerequisite Courses - Spark

Beginning Data Exploration and Analysis with Apache Spark

- Programming in Spark 1.x using Python

Getting Started with Spark 2

- Programming in Spark 2.x using Python

Prerequisite Courses - ML

How to Think About Machine Learning Algorithms

- Basic ML understanding and algorithms

Understanding Machine Learning with Python

- Basic ML in the Python language

Software and Skills

Be very comfortable programming in Python (Python 3)

Be comfortable working with Jupyter notebooks

Understand the basics of Spark and ML

Course Outline

ML libraries in Spark 1 vs Spark 2

- Basic Spark architecture
- Why 2 libraries, when to use one over the other?

Classification and regression models

- Decisions trees, random forest
- Linear regression, Lasso and Ridge regression

Clustering and dimensionality reduction

- k-means clustering, PCA

Recommendation systems

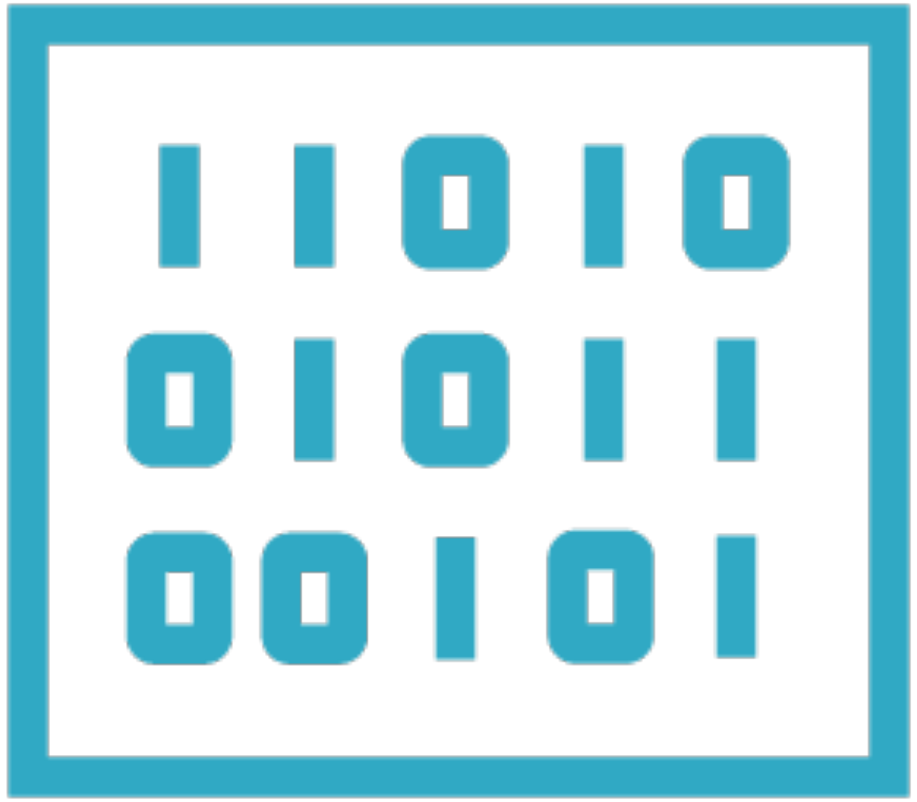
- Collaborative filtering using explicit and implicit ratings

RDDs and Spark 1.x

Why is this relevant in Spark 2?

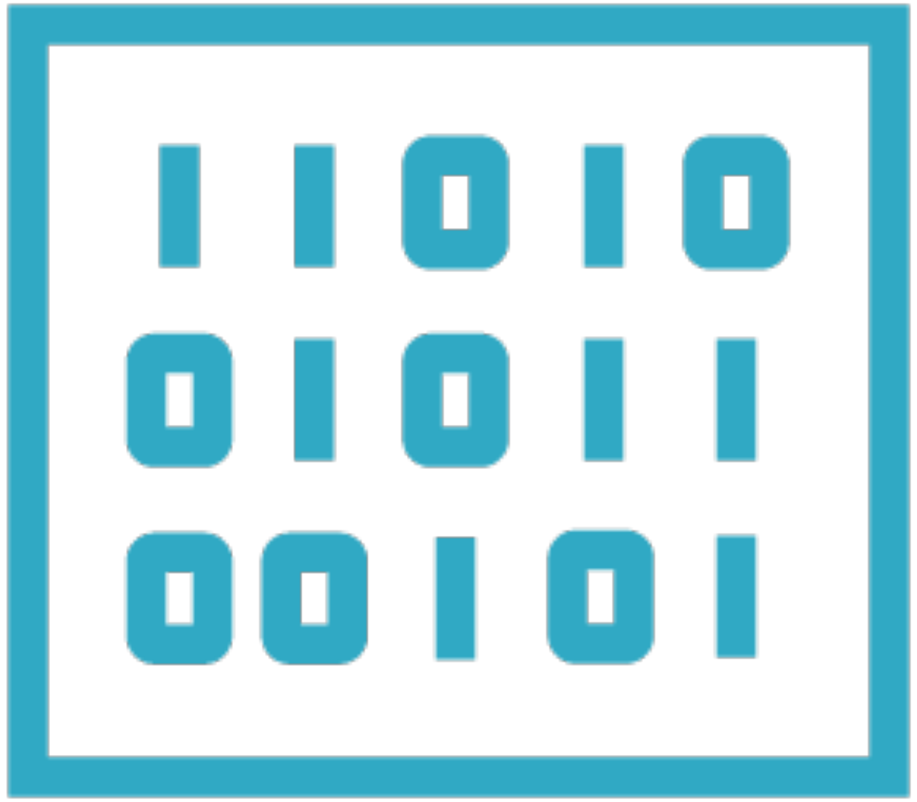
RDDs are still the **fundamental building blocks** of Spark

Resilient Distributed Datasets



All operations in Spark are performed on in-memory objects

Resilient Distributed Datasets

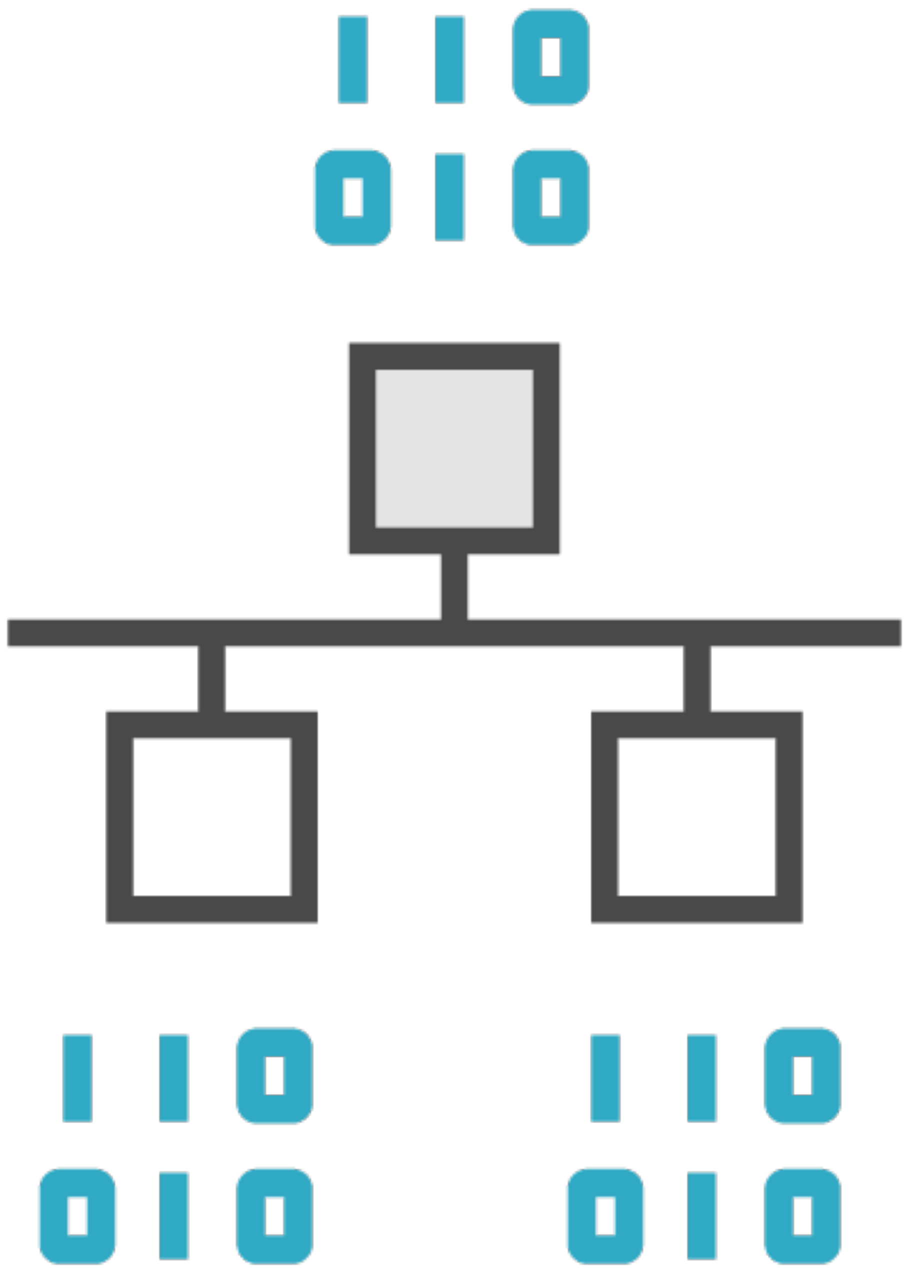


An RDD is a collection of entities - rows, records

Characteristics of RDDs

Partitioned	Immutable	Resilient
Split across data nodes in a cluster	RDDs, once created, cannot be changed	Can be reconstructed even if a node crashes

Partitioned



RDDs represent data in-memory

Partitions

1	Indigo	06:45	Bangalore
2	Jet Air	08:45	New Delhi
3	SpiceJet	09:15	Mumbai
4	Indigo	10:45	New Delhi
5	Air India	11:15	Mumbai
6	Vistara	12:00	New Delhi

Data is divided into partitions

Partitions

1	Indigo	06:45	Bangalore
2	Jet Air	08:45	New Delhi
3	SpiceJet	09:15	Mumbai
4	Indigo	10:45	New Delhi
5	Air India	11:15	Mumbai
6	Vistara	12:00	New Delhi

Data is divided into partitions

Partitions

1	Indigo	06:45	Bangalore
2	Jet Air	08:45	New Delhi
3	SpiceJet	09:15	Mumbai
4	Indigo	10:45	New Delhi
5	Air India	11:15	Mumbai
6	Vistara	12:00	New Delhi

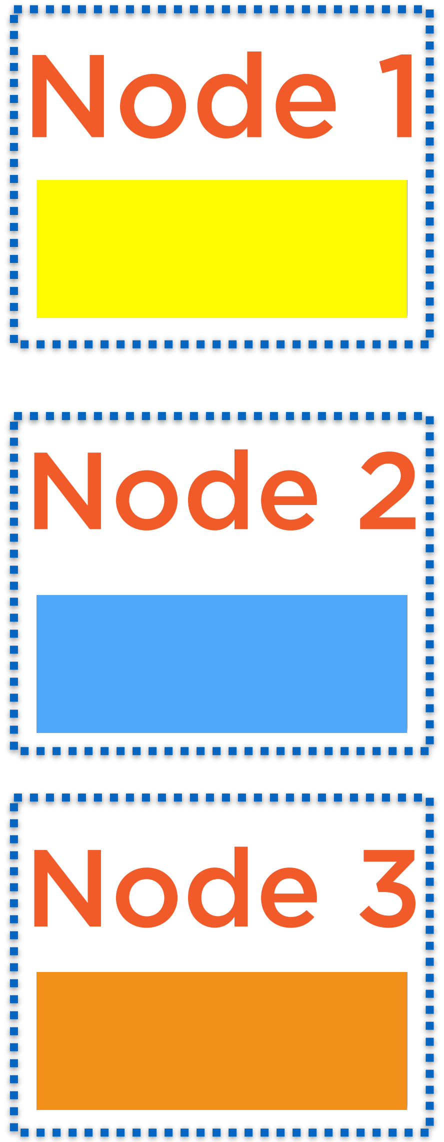
Distributed to multiple machines, called nodes

Partitions

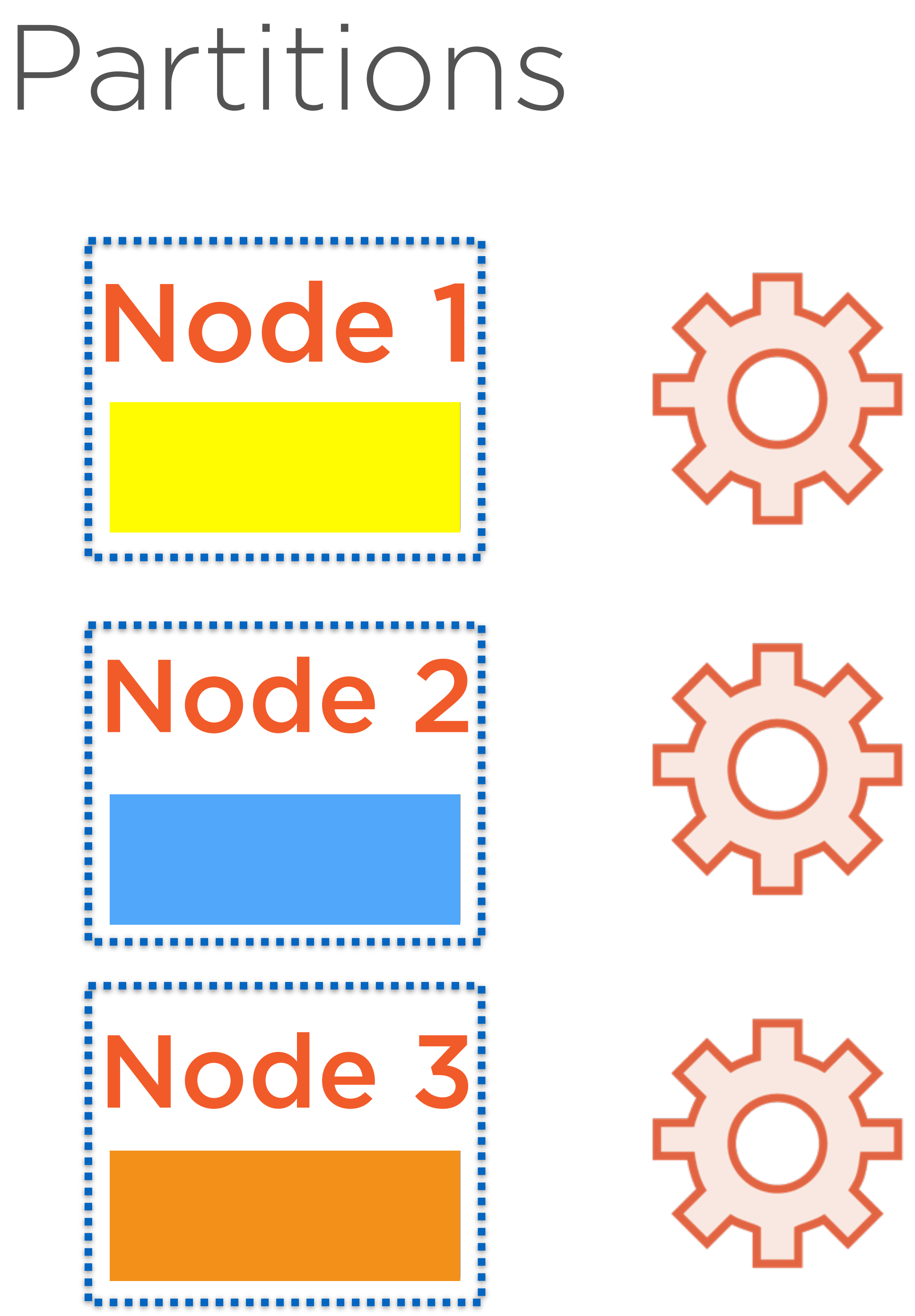
1	Indigo	06:45	Bangalore
2	Jet Air	08:45	New Delhi
3	SpiceJet	09:15	Mumbai
4	Indigo	10:45	New Delhi
5	Air India	11:15	Mumbai
6	Vistara	12:00	New Delhi

Distributed to multiple machines, called nodes

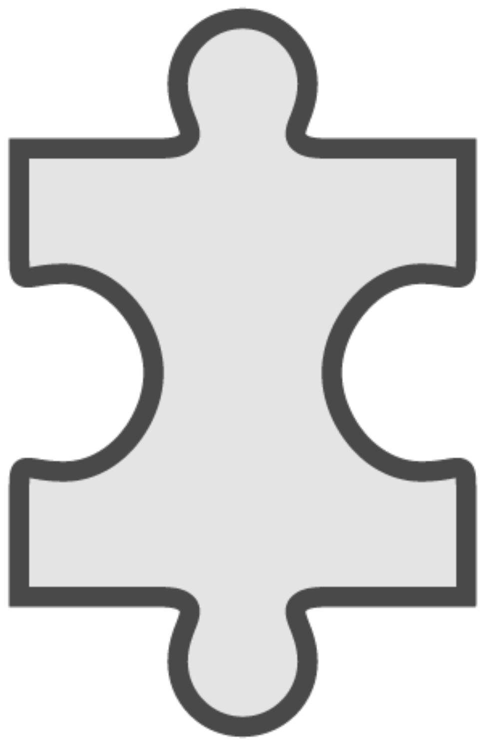
Partitions



Nodes process data in parallel



Immutable



An RDD cannot be mutated

Only two operations are permitted on an RDD

Only Two Types of Operations

Transformation

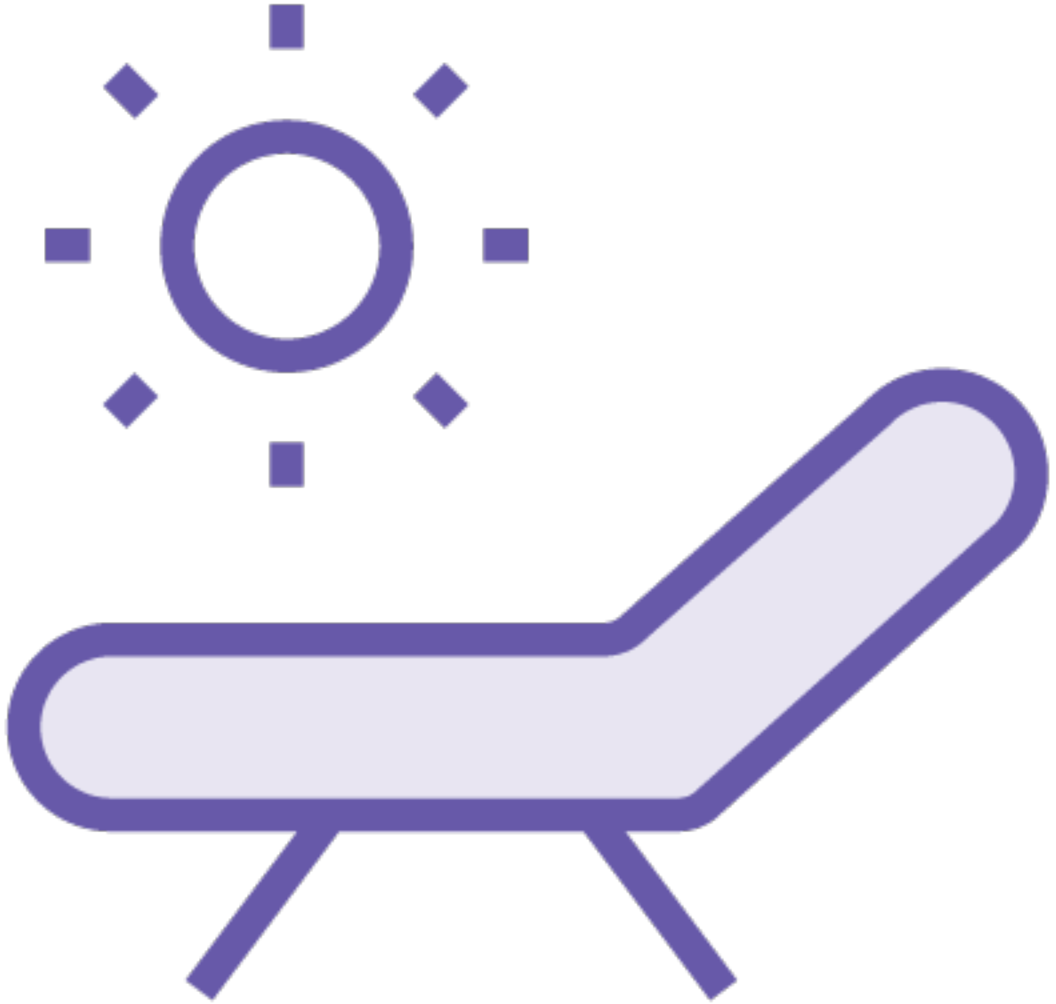
Transform into another RDD

Action

Request a result

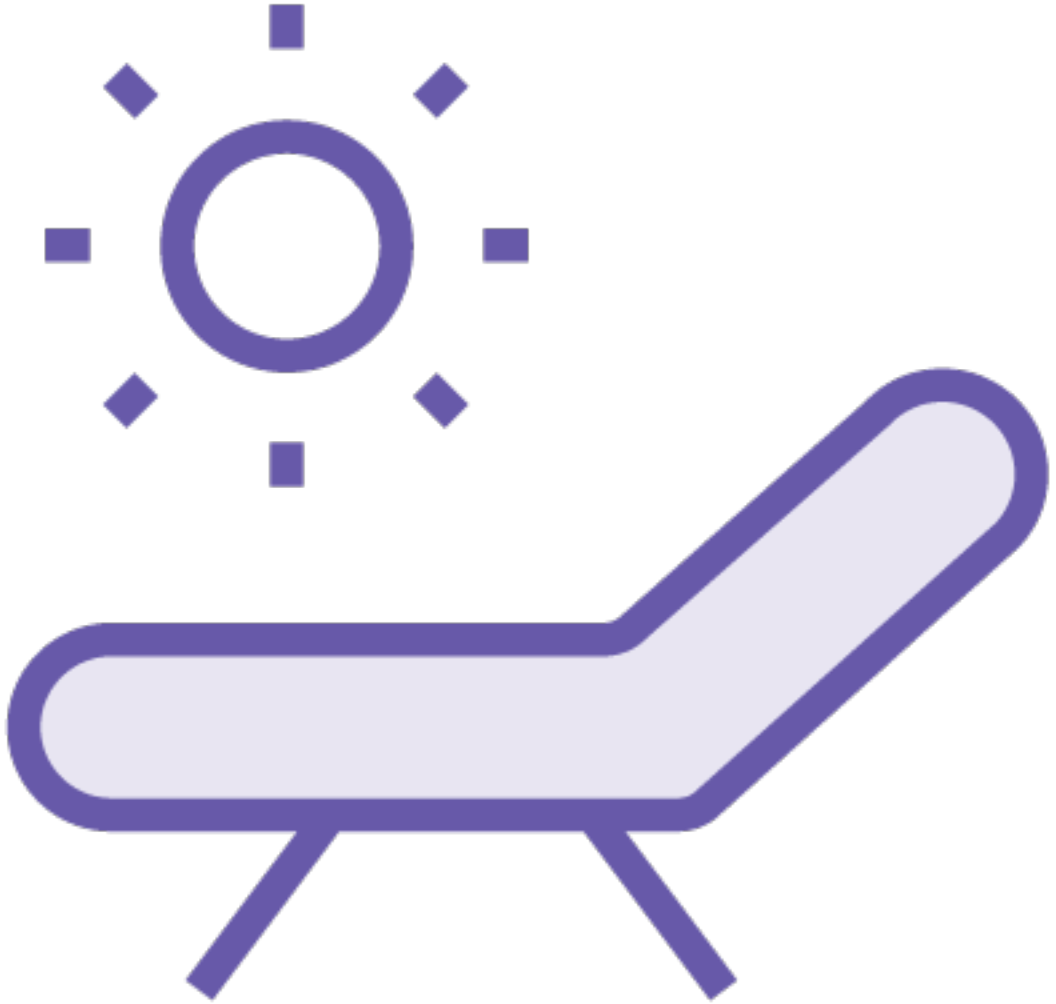
Transformations are **executed** only when a result is requested

Lazy Evaluation



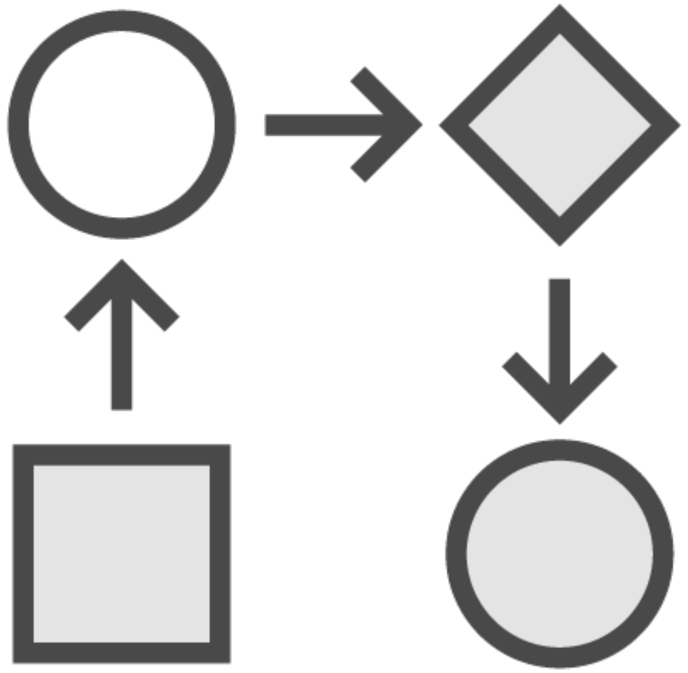
Spark keeps a record of the series of transformations requested by the user

Lazy Evaluation



It groups the transformations in an efficient way when an Action is requested

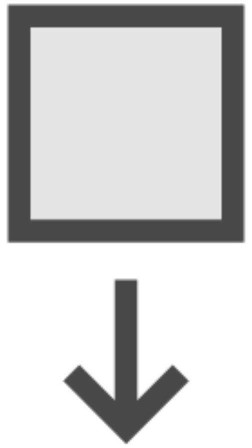
Resilient



RDDs can be reconstructed even if the node it lives on crashes

RDDs Are Resilient

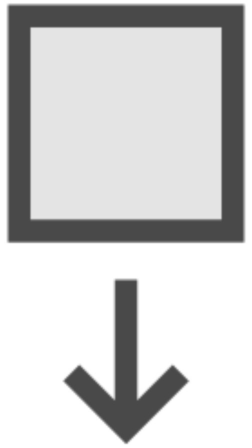
RDDs can be created in 2 ways



Reading a file

Transforming another RDD

RDDs Are Resilient

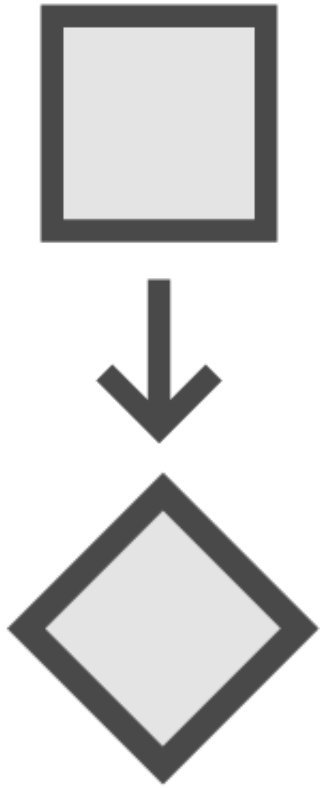


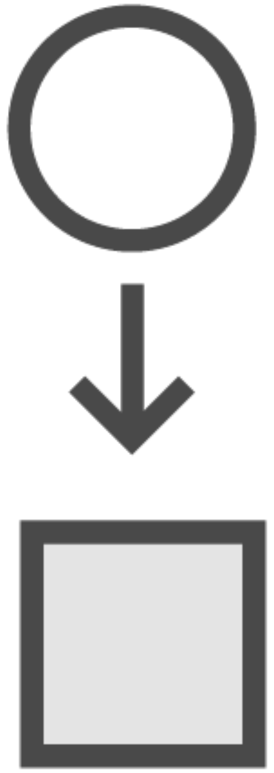
Reading a file

Transforming another RDD

Every RDD keeps track of **where** it came from

RDDs Are Resilient



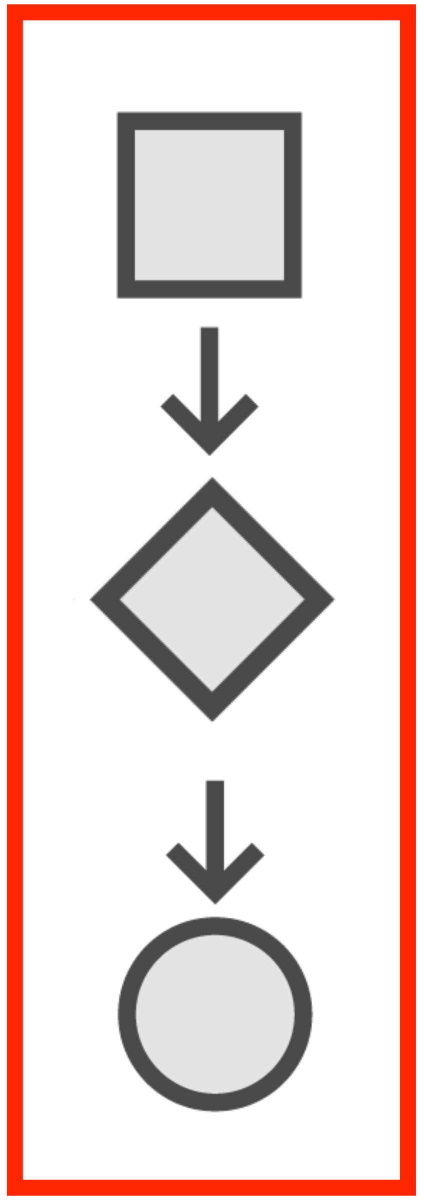


It tracks **every** transformation which led to the current RDD

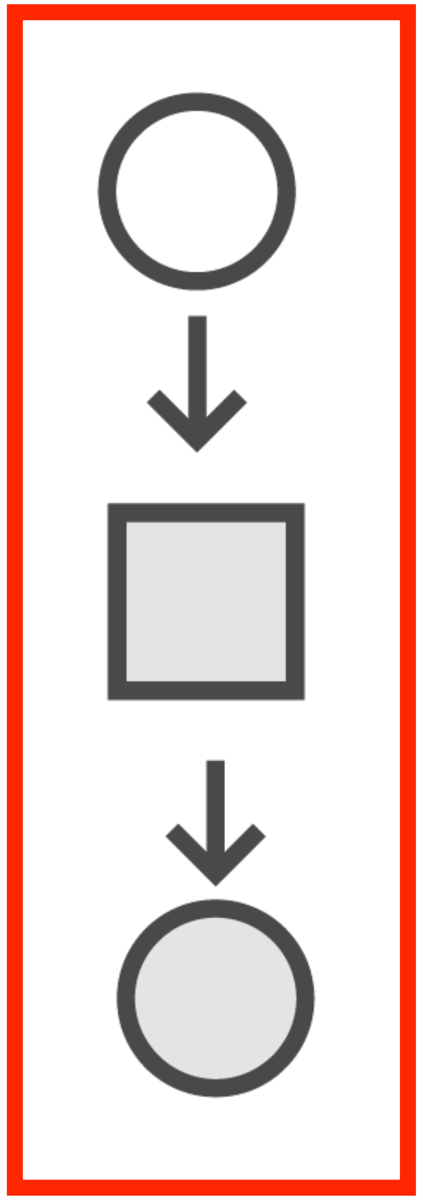
RDDs Are Resilient

However many transformations it takes

RDDs Are Resilient

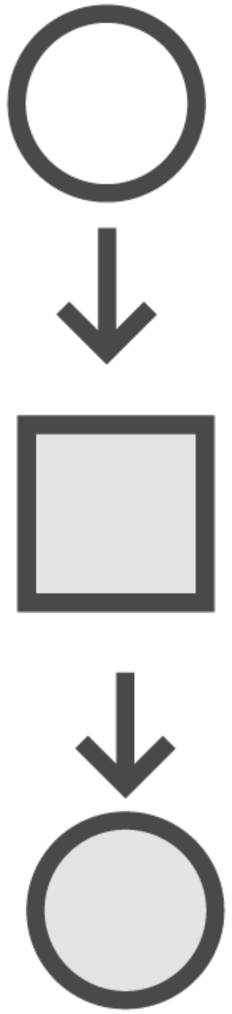


This is the
RDD's lineage



RDDs Are Resilient

None of the transformations are applied till we access the results



Characteristics of RDDs

Partitioned	Immutable	Resilient
Split across data nodes in a cluster	RDD once created cannot be changed	Can be reconstructed even if a node crashers

RDDs, DataFrames, Datasets

DataFrame: Data in Rows and Columns

DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

Each row represents
1 observation

DataFrame: Data in Rows and Columns

DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

Each column represents 1 variable (a list or vector)

From File to DataFrame

DATE	OPEN	…	PRICE
2016-12-01	772	…	779
2016-11-01	758	…	747
2006-01-01	302	…	309

File

read

DATE	OPEN	...	PRICE
2016-12-01	772	…	779
2016-11-01	758	…	747
2006-01-01	302	…	309

DataFrame

RDDs to DataFrames

RDDs	DataFrames
Primary abstraction since initial versions	Added to Spark in 1.3
Immutable and distributed	Also immutable and distributed
Conceptually similar to a collection of records	Conceptually equal to a table in an RDBMS
No concept of columns	Named columns like Pandas or R
No optimized execution	Leverage optimizers in recent versions
Available in all languages	Available in all languages

Datasets to DataFrames

Datasets	DataFrames
Scala and Java*	**Python, R, Scala, Java**
Type safe OOP interface	**No type safety at compile time**
*Datasets of the Row() object in Scala/ Java often called DataFrames	Equivalent to Dataset<Row> in Java or Dataset[Row] in Scala

Starting Spark 2.0, APIs for Datasets and DataFrames have merged

DataFrames Built on Top of RDDs

Partitioned	Immutable	Resilient
Split across data nodes in a cluster	Once created, cannot be changed	Can be reconstructed even if a node crashes

Demo

Install standalone Spark on your local machine

Set up the PySpark REPL interface

Making the Choice Between spark.ml vs. spark.mllib

Changes Starting Spark 2.0

Easier

Unifying Datasets and DataFrames, SQL support…

Faster

Optimize like a compiler, not a DBMS

Performance Improvements

Comparison of time per row, on 1 billion records on single thread

Primitive	Spark 1.6	Spark 2.0	Speedup Factor
filter	15ns	1.1ns	13.6
sum w/o group	14ns	0.9ns	15.6
sum w/ group	79ns	10.7ns	7.4
hash join	115ns	4.0ns	28.8
sort (8-bit)	620ns	5.3ns	117.0
sort (64-bit)	620ns	40ns	15.5
sort-merge-join	750ns	700ns	1.1

Source: https://databricks.com/blog/2016/07/26/introducing-apache-spark-2-0.html

Ease of Use

Unified API for DataFrames

spark.ml and ML pipelines

Advanced streaming

spark.mllib and spark.ml

spark.mllib	spark.ml
Older	Newer
RDDs	DataFrames (faster!)
For now, more functionality	Functionality catching up
ETL hard - no pipeline support	Support for ML pipelines
Hyperparameter tuning hard	Tools for hyperparameter tuning

spark.mllib and spark.ml

spark.mllib

To maintain backward compatibility with 1.x applications

To use features which are not available in the newer version

ETL is not important, do not need pipelines

spark.ml

Spark 2 is available and you want to take advantage of better performance

To use higher levels APIs and abstractions for faster development

ETL, chaining transformations significant

Both packages are currently useful - **spark.mllib** has more features

spark.ml - feature compatibility around the corner

spark.mllib will be deprecated in the future

Decision Trees

Jockey or Basketball Player?

Jockeys

Tend to be light to meet horse carrying limits

Basketball Players

Tend to be tall, strong and heavy

Jockey or Basketball Player?

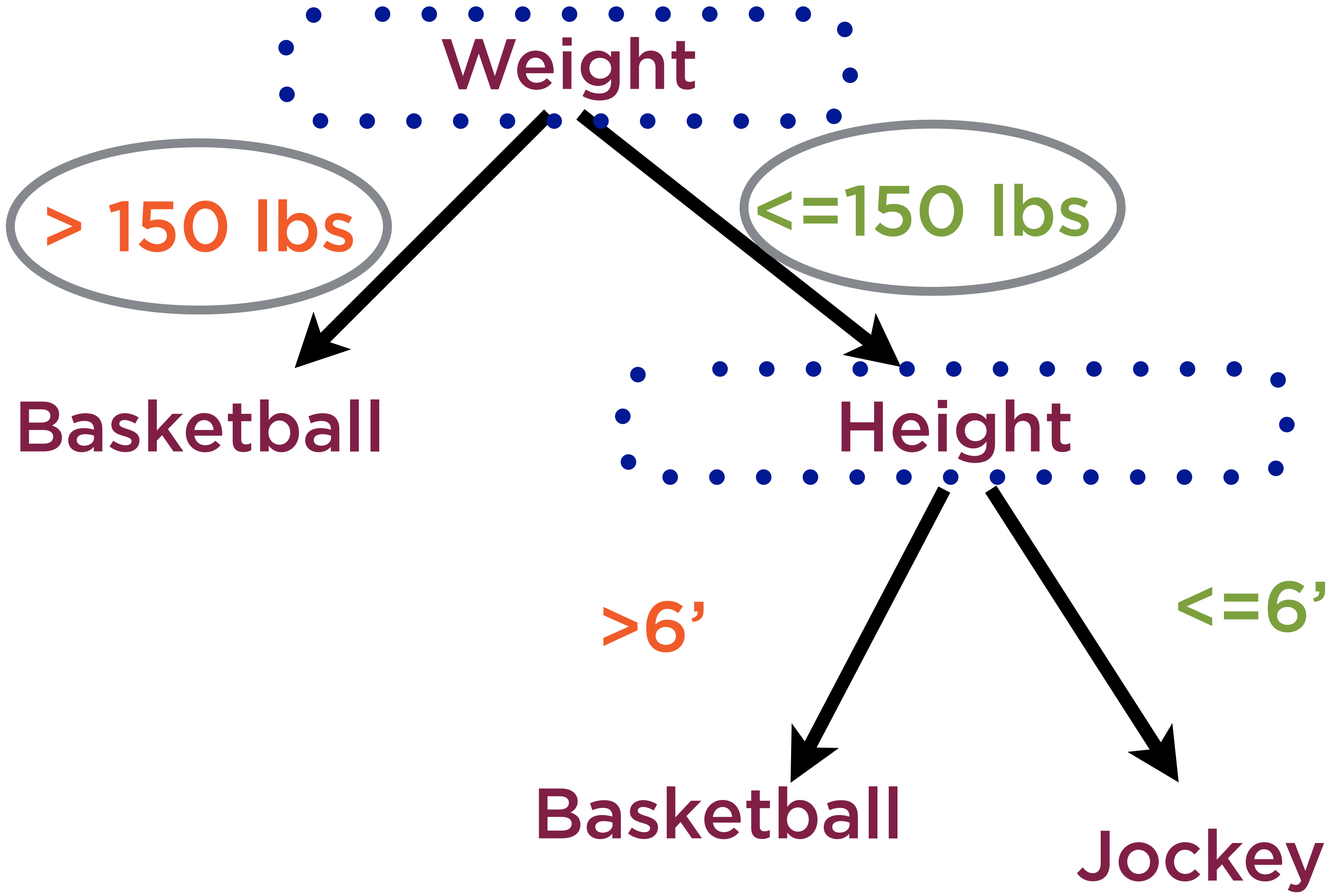
Intuitively know

- jockeys tend to be light…
- …and not very tall
- basketball players tend to be tall
- …and also quite heavy

Fit knowledge into rules

Each rule involves a threshold

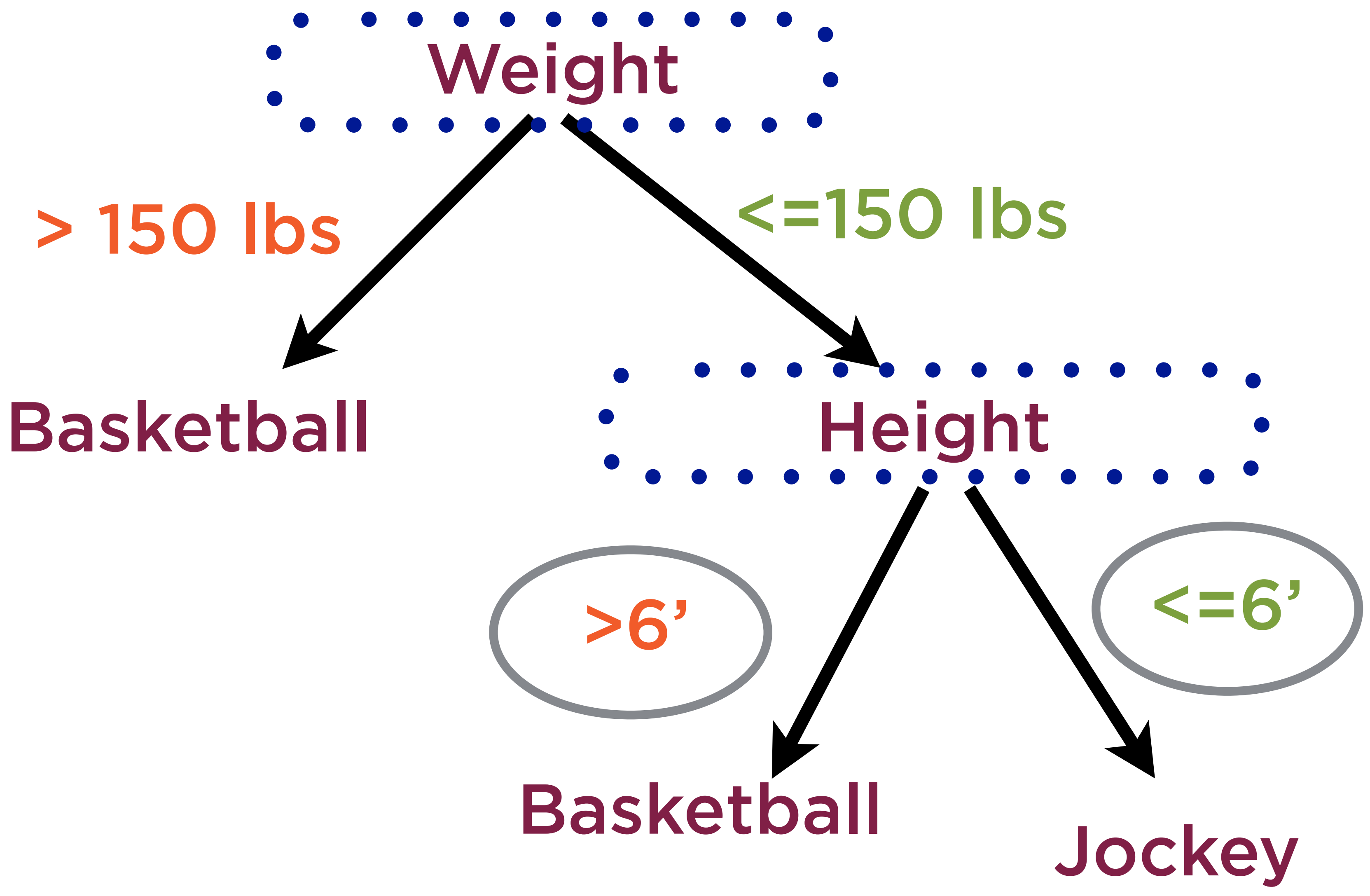
Decision Tree



Fit knowledge into rules

Each rule involves a threshold

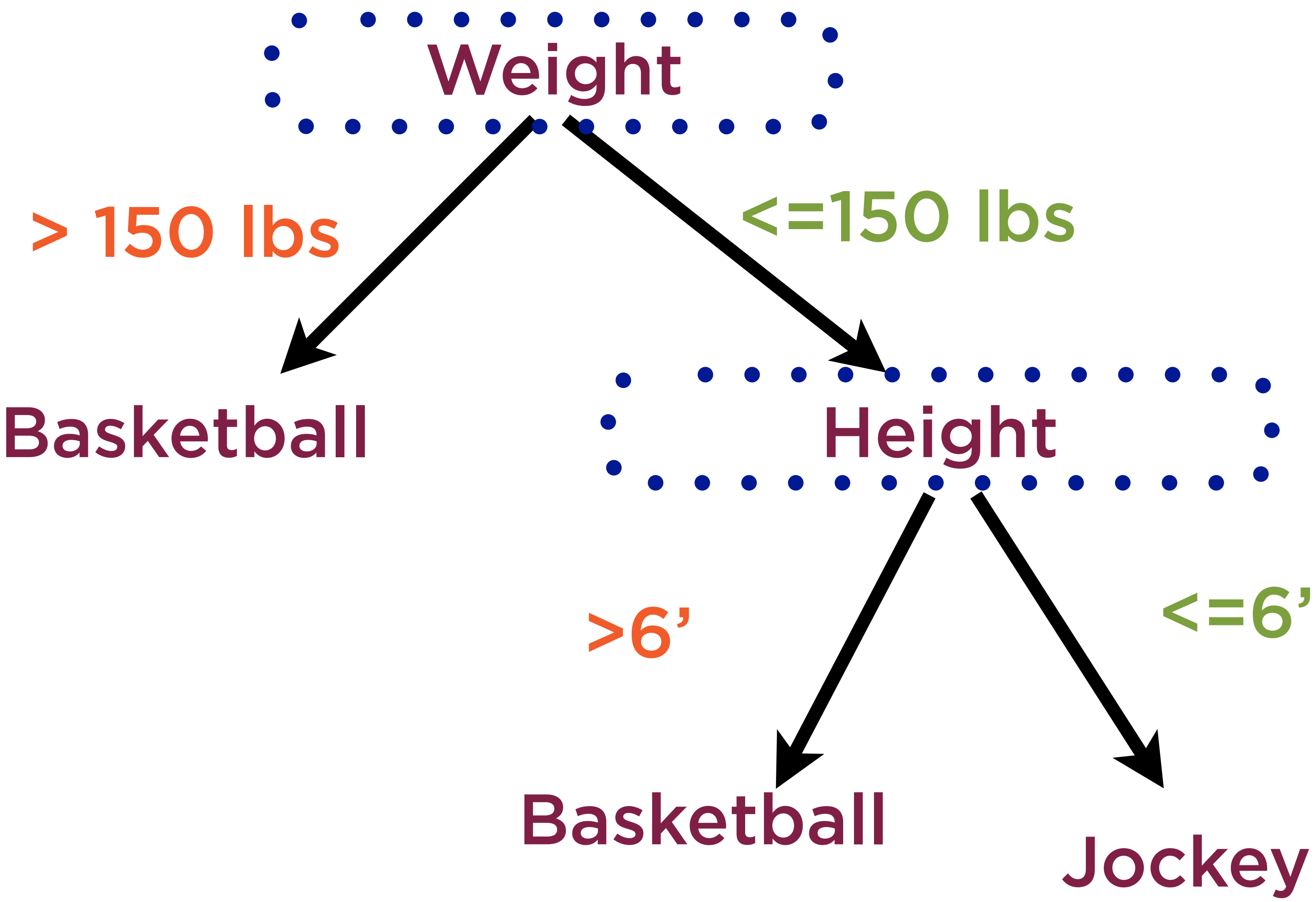
Decision Tree



Order of decision variables matters

Rules and order found using ML

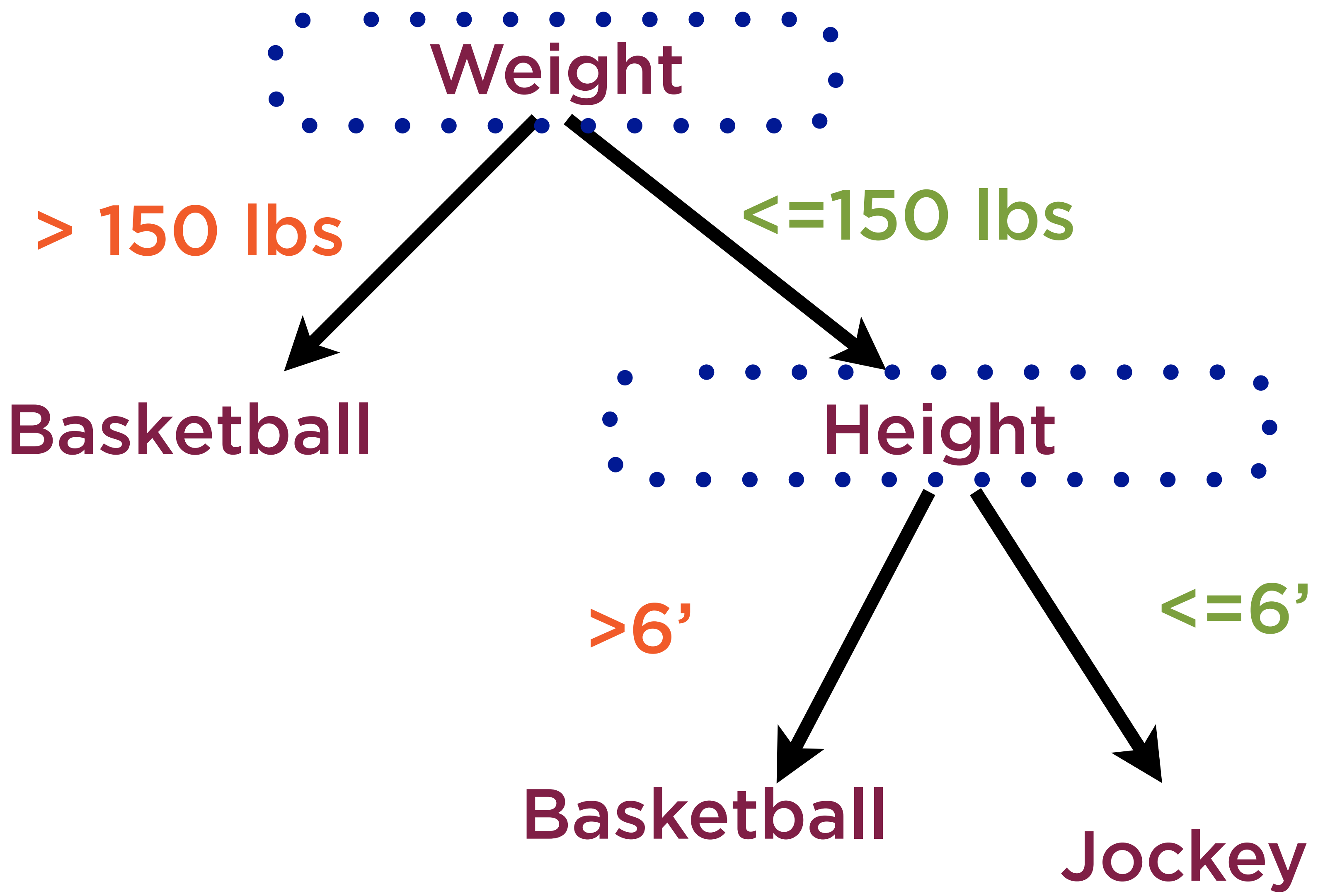
Decision Tree



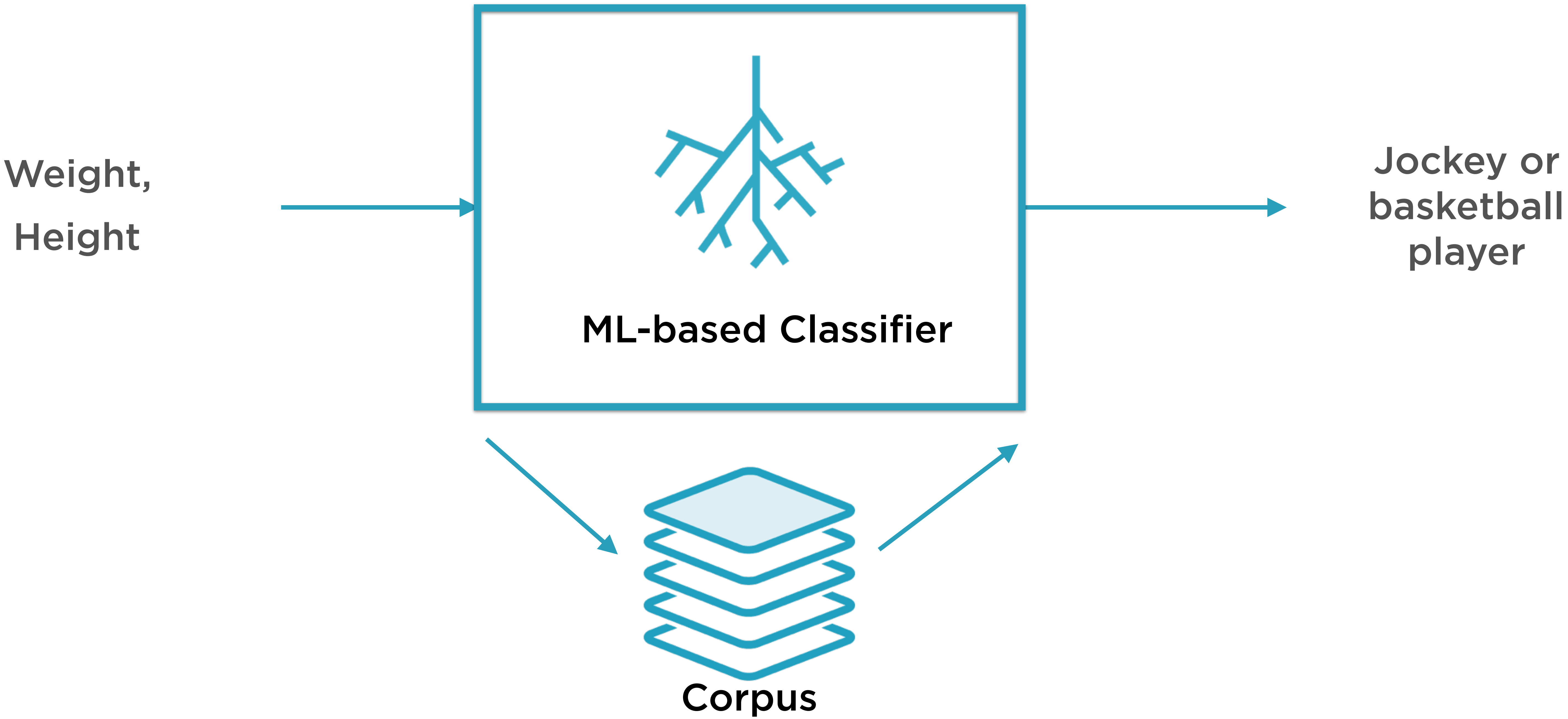
“CART”

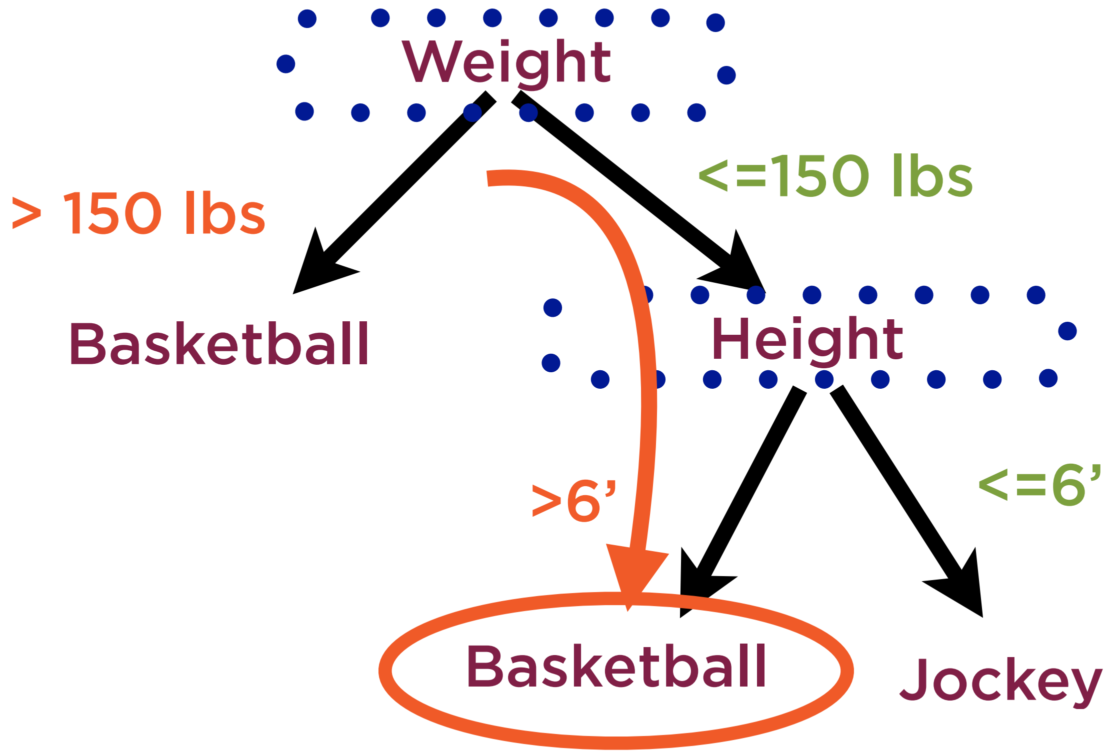
Classification And Regression Tree

Decision Tree



Decision Trees for Classification



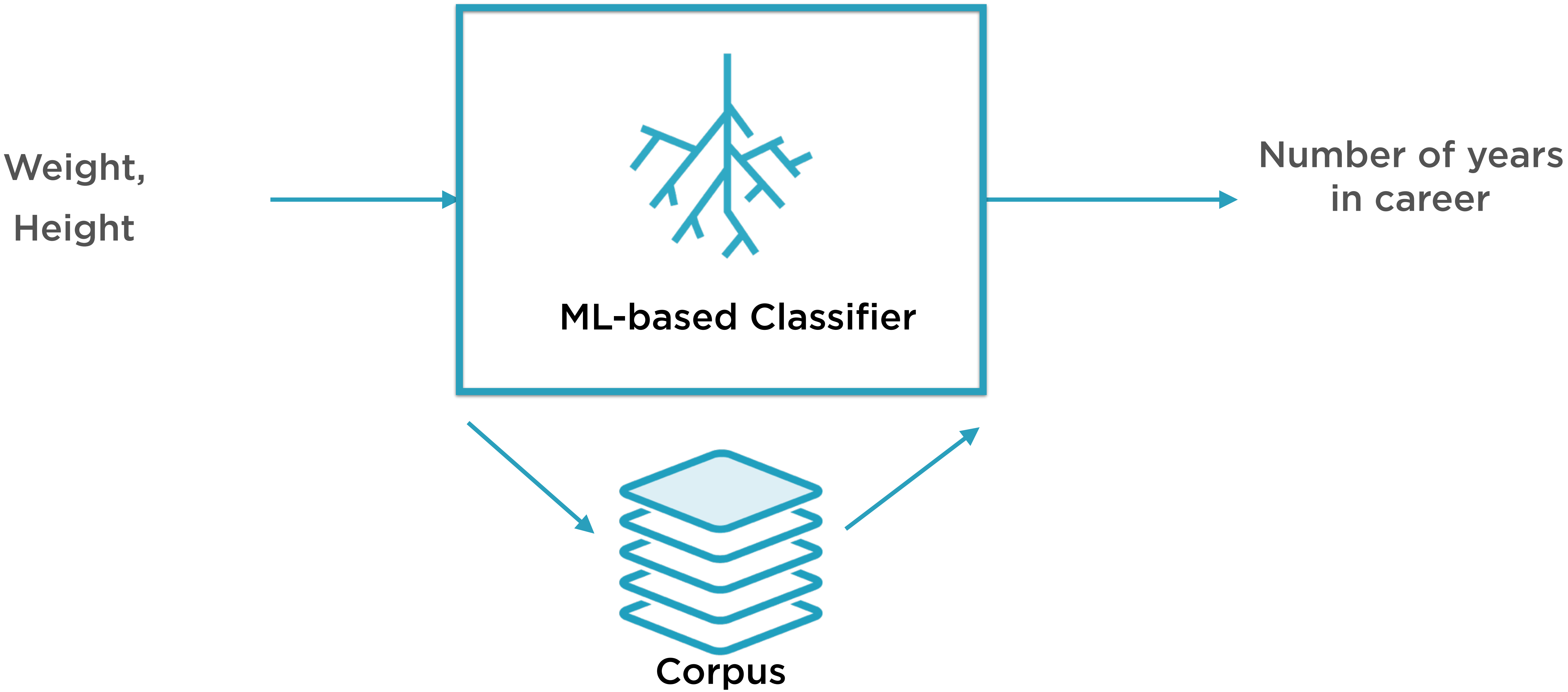


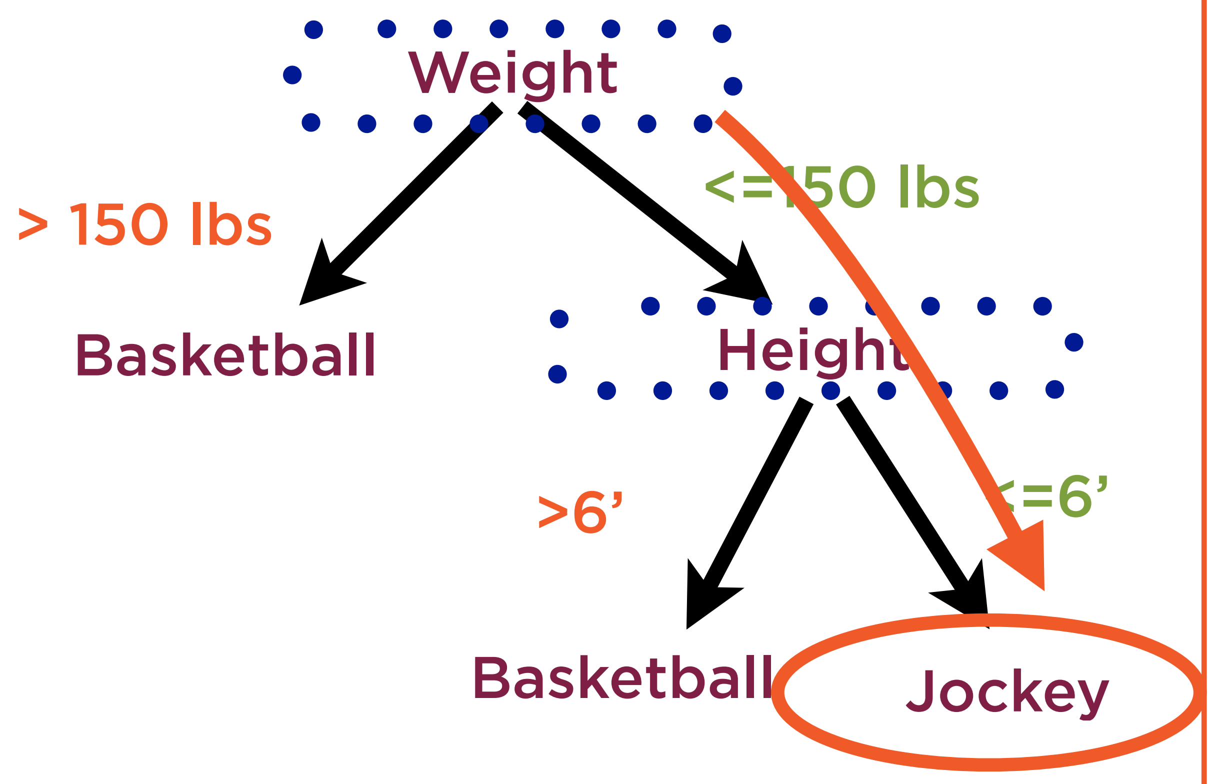
Decision Trees for Classification

To solve

- Traverse tree to find right node
- Return most frequent label of all training data points in that node

Decision Trees for Regression





Decision Trees for Regression

To solve

- **Traverse tree to find right node**
- **Return average number of years of all training data points in that node**

Muggsy Bogues

Shortest player ever in the NBA

5’3” and 135 lbs

Our tree would classify him as Jockey

No threshold is perfect!

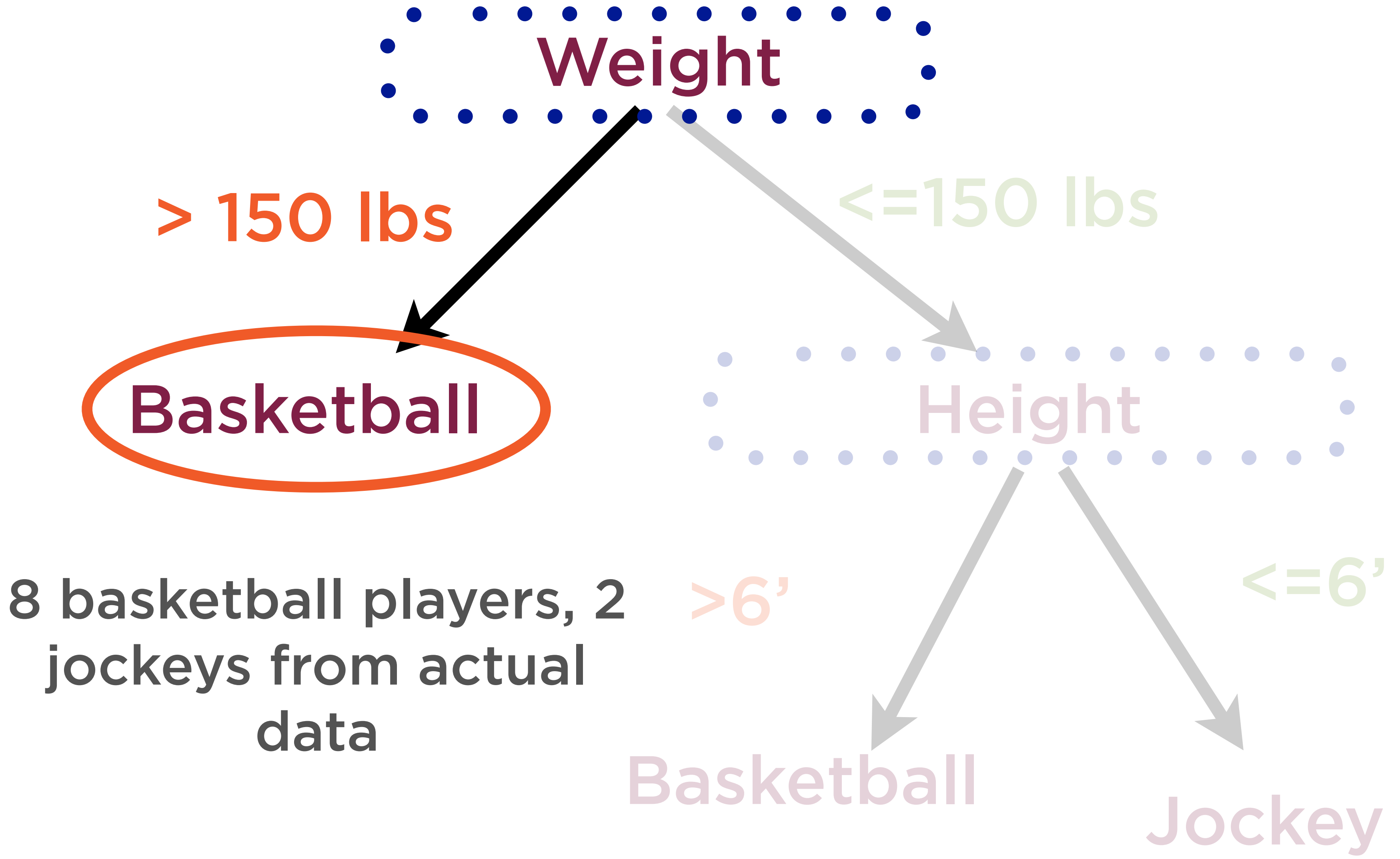
Tree Construction

CART optimizes tree construction

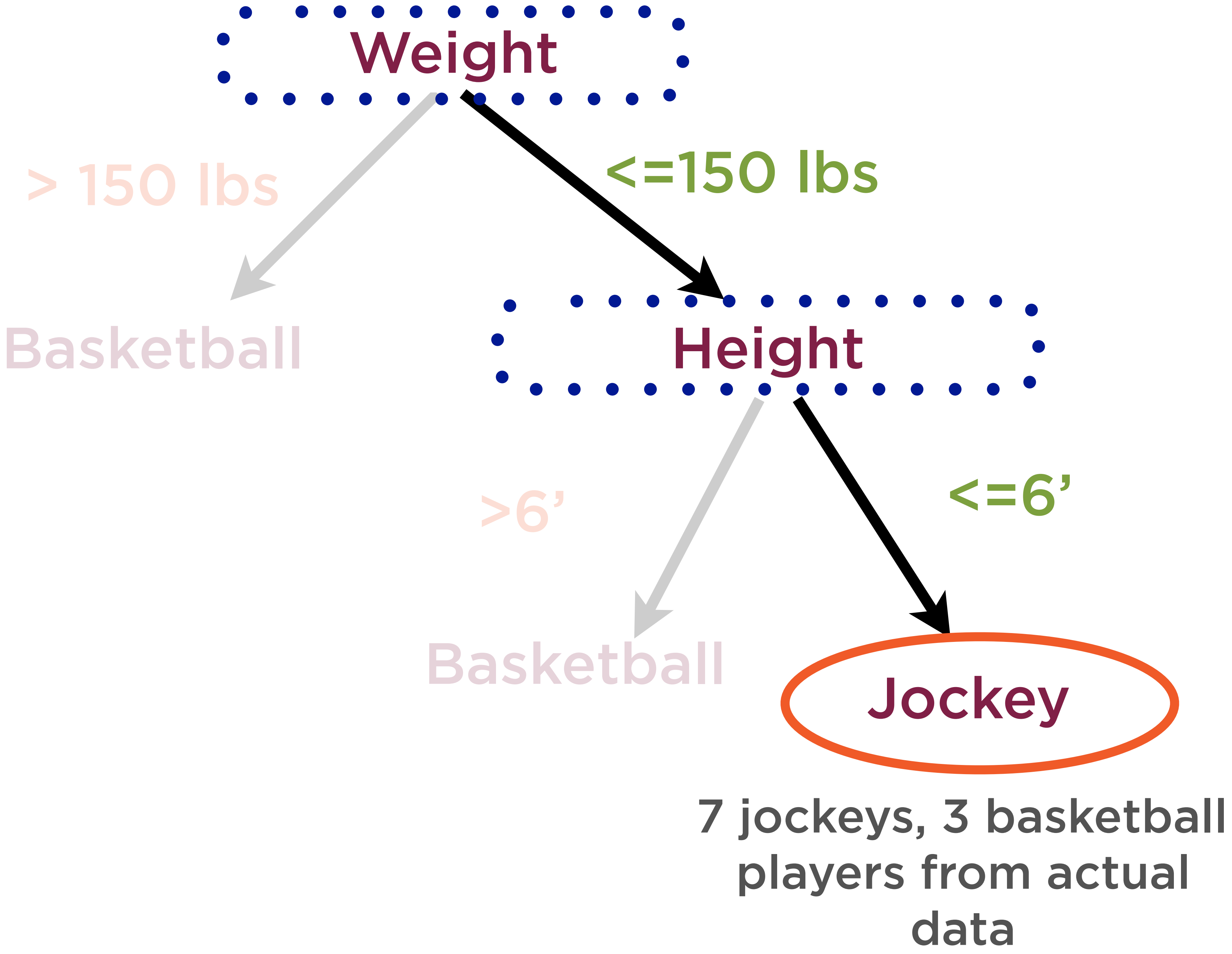
Minimizes “impurity” of each node

Impurity ~ misclassified data points

Impurity


Weight
> 150 lbs
<=150 lbs
Basketball
Height
8 basketball players, 2 jockeys from actual data
>6'
<=6'
Basketball
Jockey

Impurity

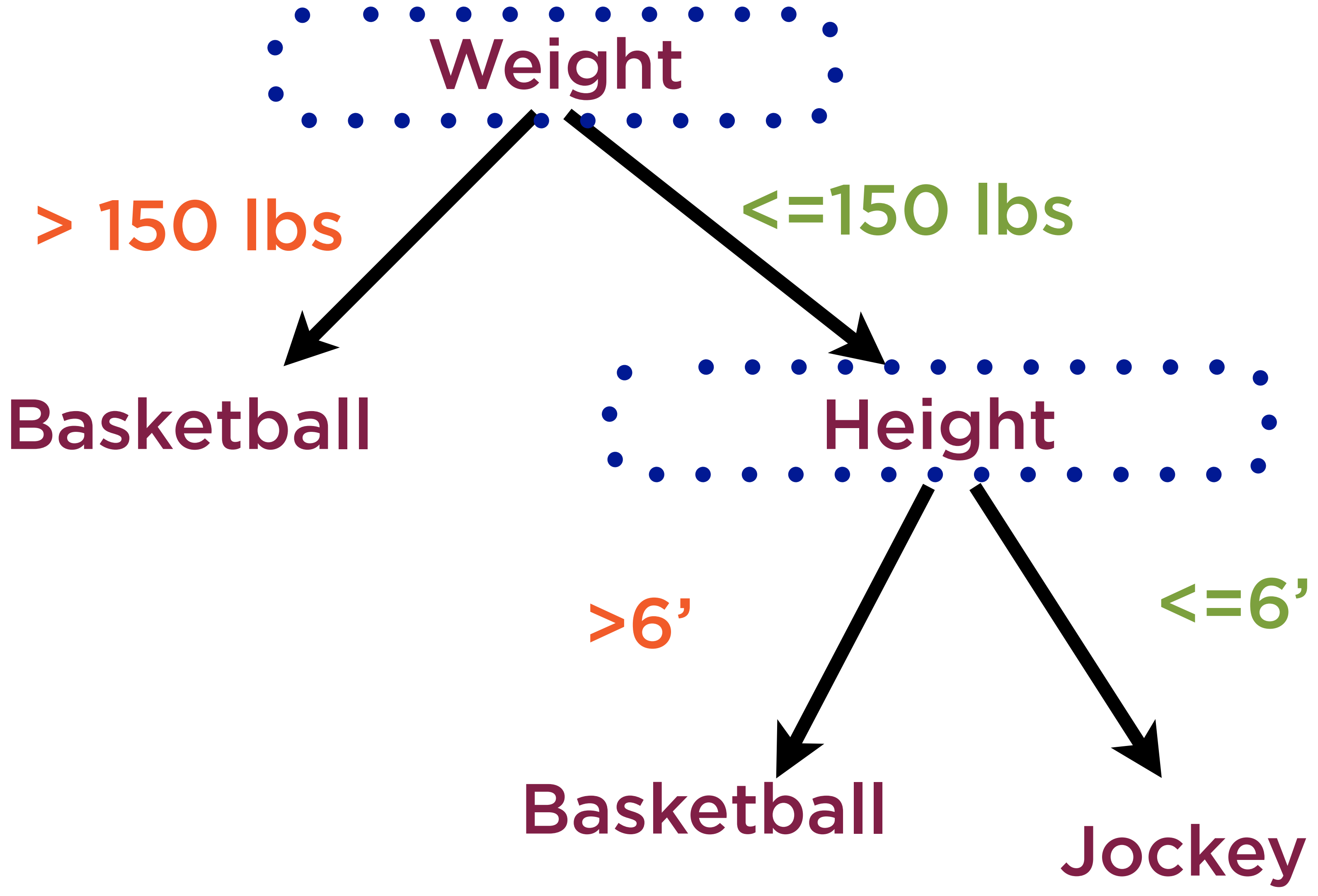

Weight
> 150 lbs
<=150 lbs
Basketball
Height
>6’
<=6’
Basketball
Jockey
7 jockeys, 3 basketball players from actual data

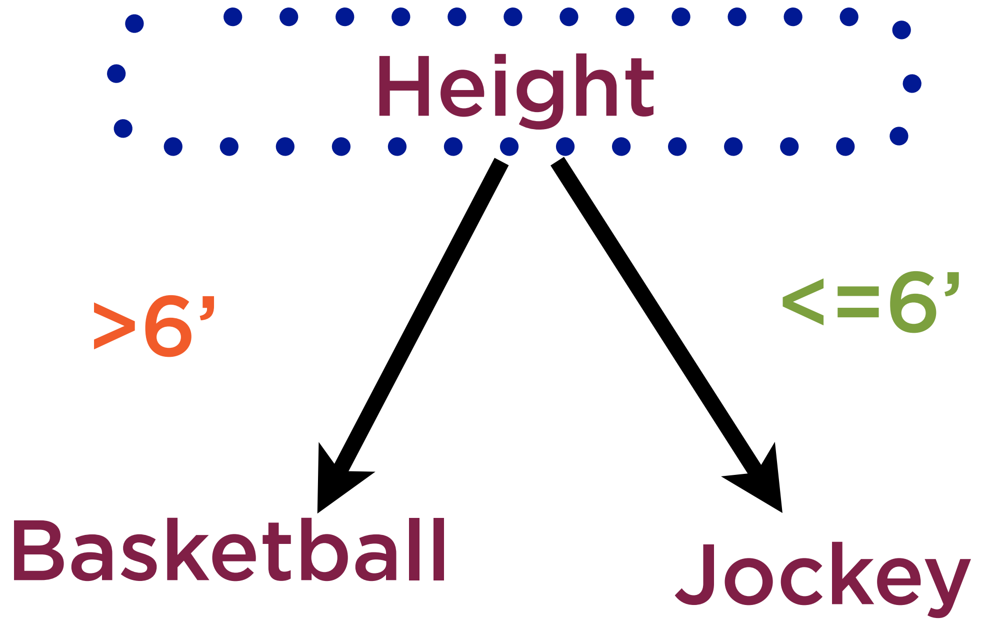
Tree Construction

Two ways to measure impurity

- Gini impurity
- Entropy

Yield similar trees



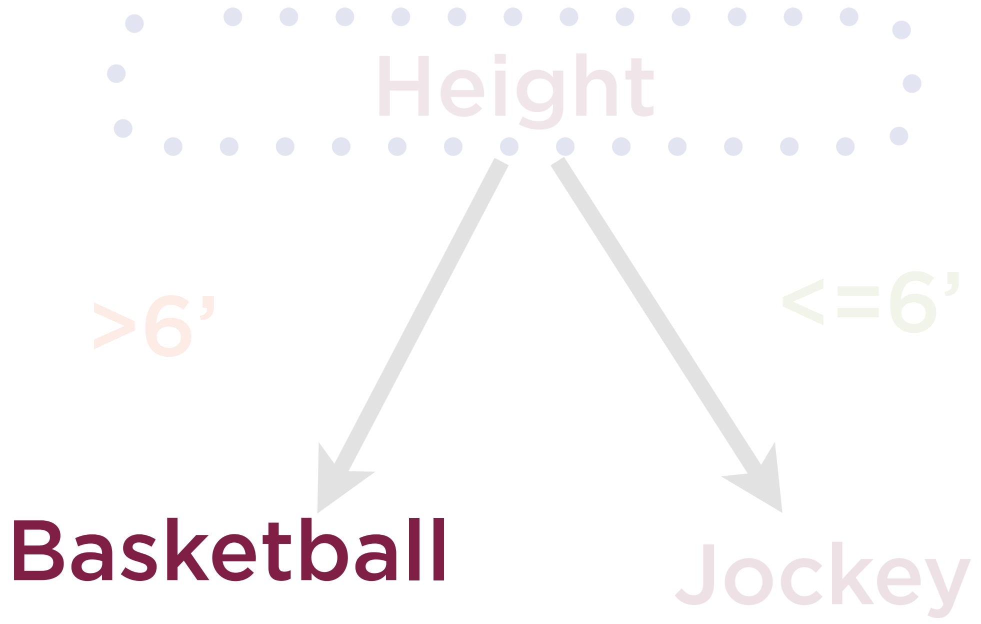


Gini Impurity

CART seeks to minimize Gini impurity at each node

Gini impurity is found from rule violations in training data

Gini Impurity



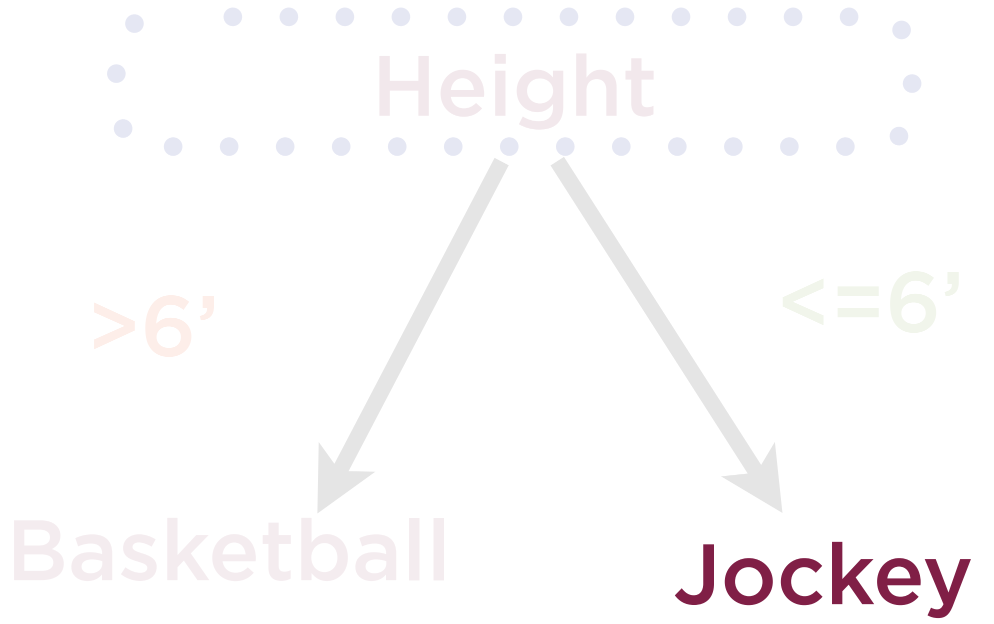
$G_i = 0.095$

In training data:

100 samples with height > 6’

- 95 basketball players
- 5 jockeys

$G_i = 1 - (95\%)^2 - (5\%)^2 = 0.095$



$G_i = 0.0$

Completely pure

Gini Impurity

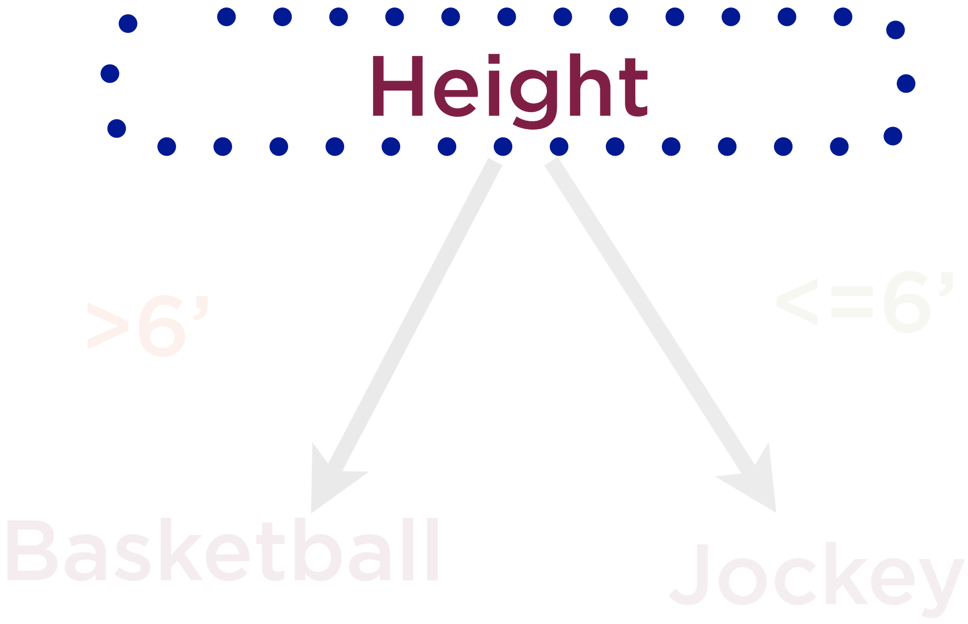
In training data:

100 samples with height <= 6’

- 0 basketball players
- 100 jockeys

$G_i = 1 - (0\%)^2 - (100\%)^2 = 0$

$G_i = 0.49875$



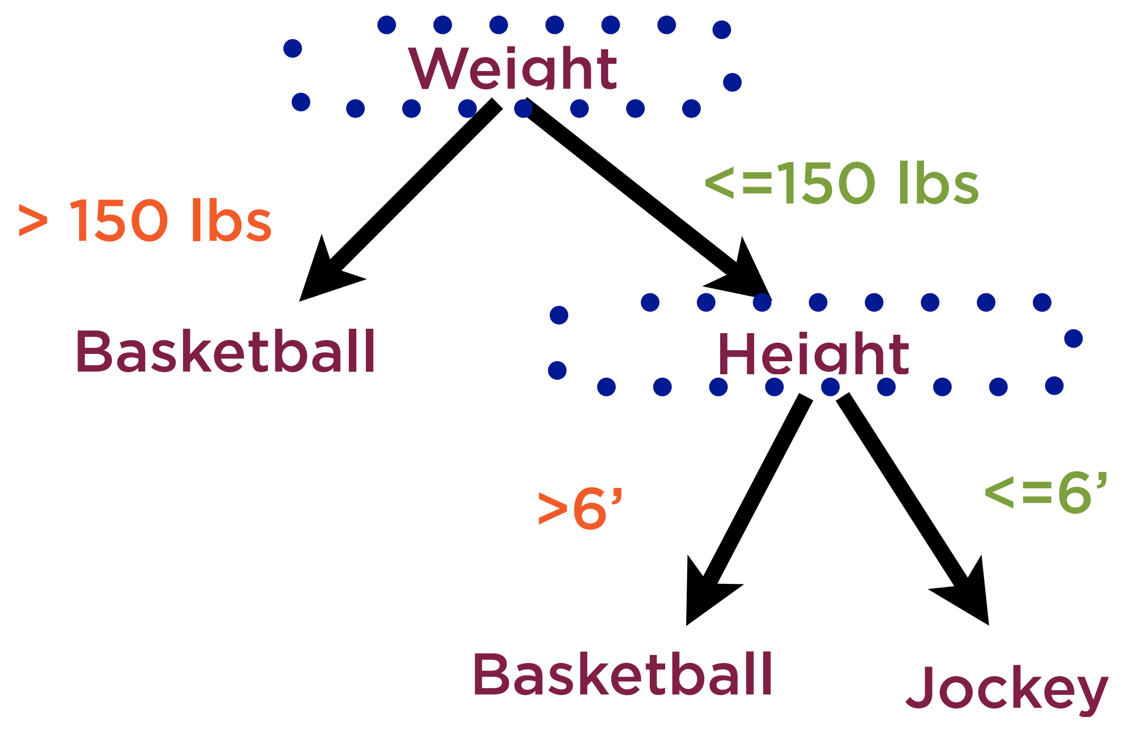
Gini Impurity

200 samples (sum of the leaf nodes)

- 95 basketball players
- 105 jockeys

$$G_i = 1 - (95/200)^2 - (105/200)^2$$

$$= 0.49875$$



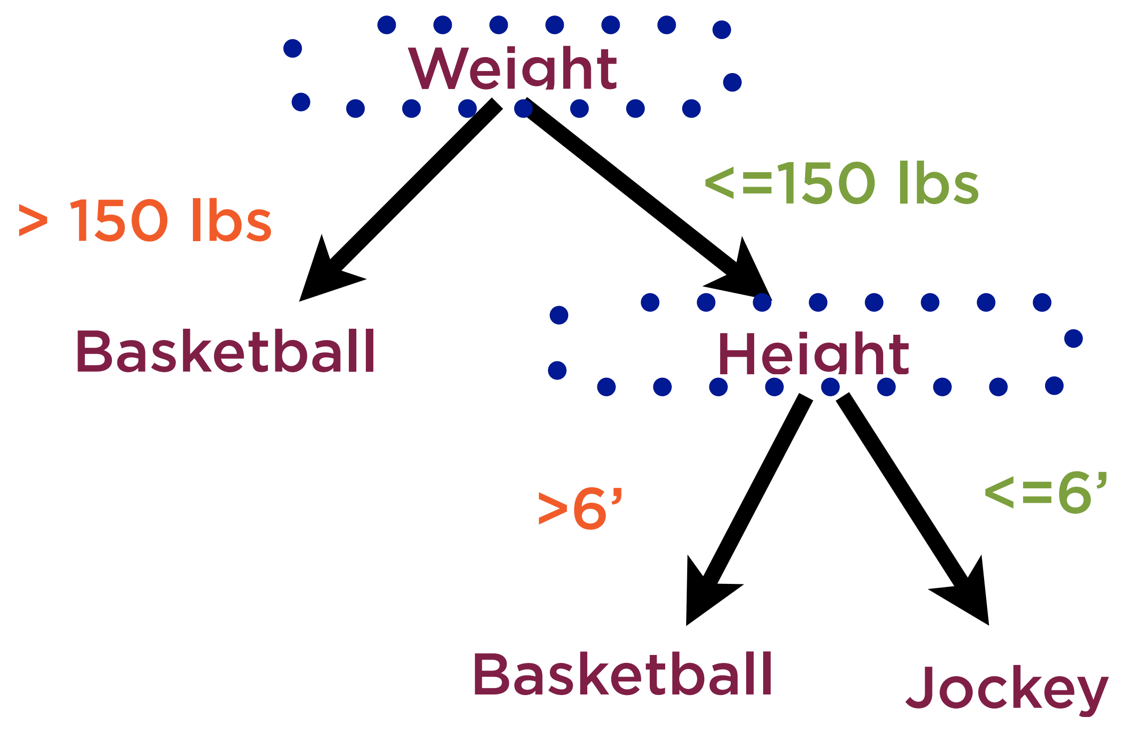
Advantages of Decision Trees

"White Box" ML ~ leverage experts

Non-parametric

- Little hyperparameter tuning
- Little data prep

Drawbacks of Decision Trees



Prone to overfitting

- Common risk with non-parametric

Unstable

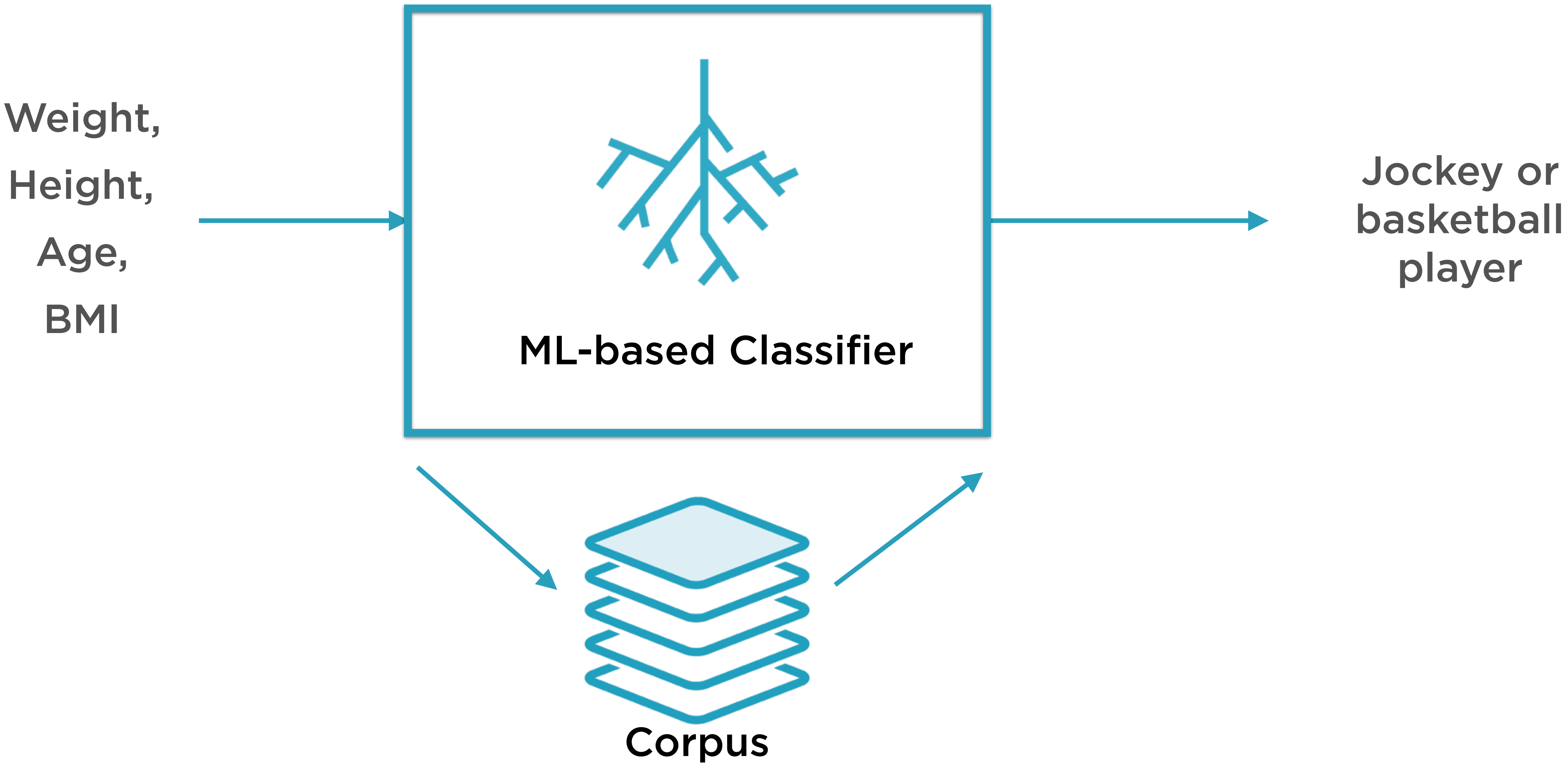
- Small changes in data cause big changes in model

Demo

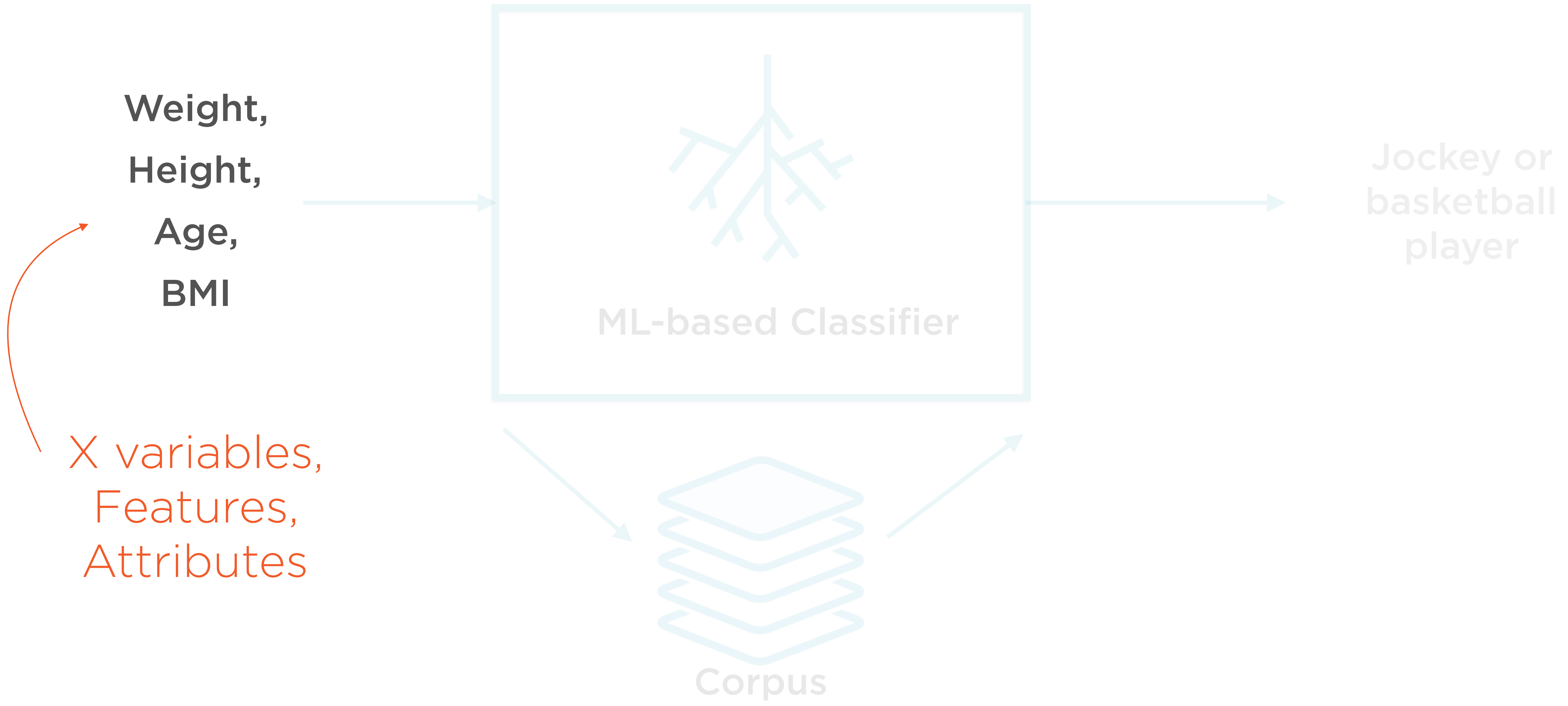
Implement classification using decision trees in spark.mllib

Data in the CSV as well as the LIBSVM format

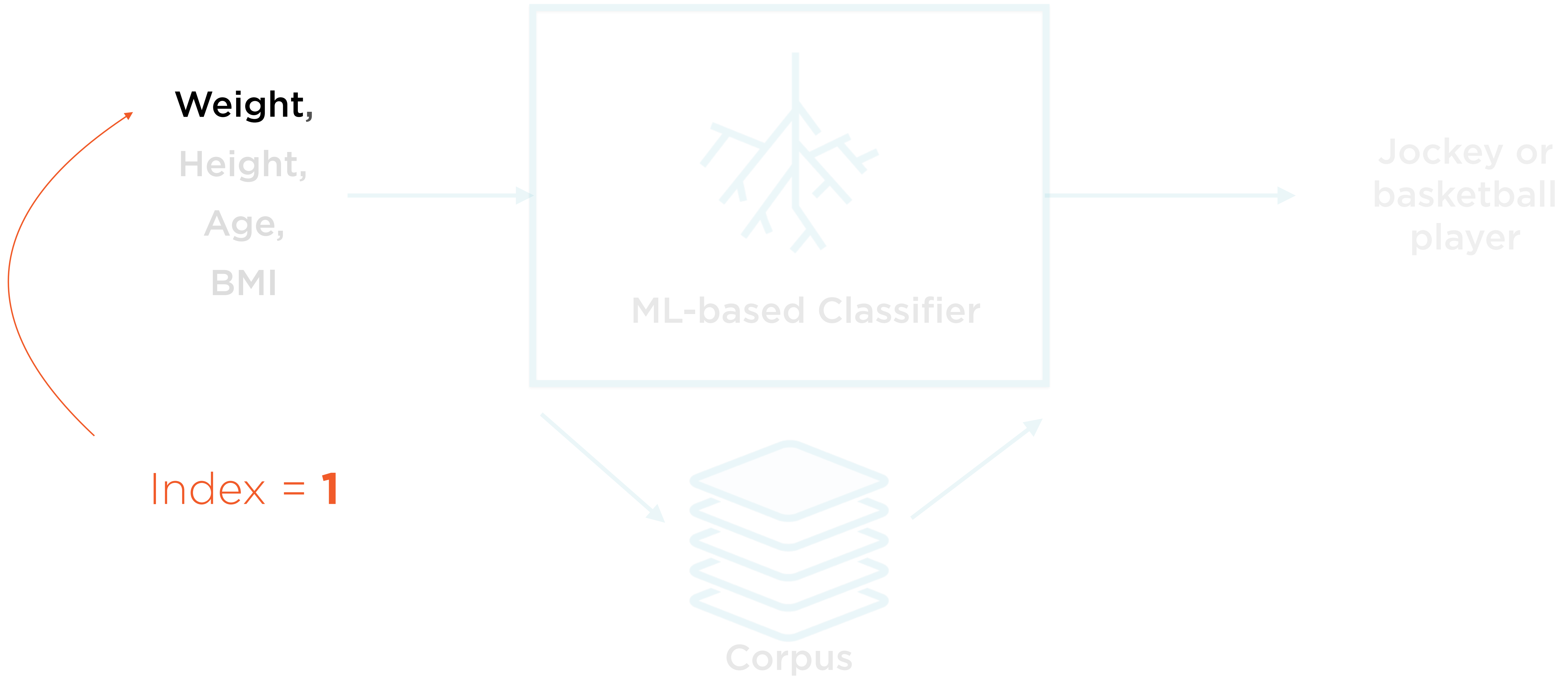
LIBSVM Data Format



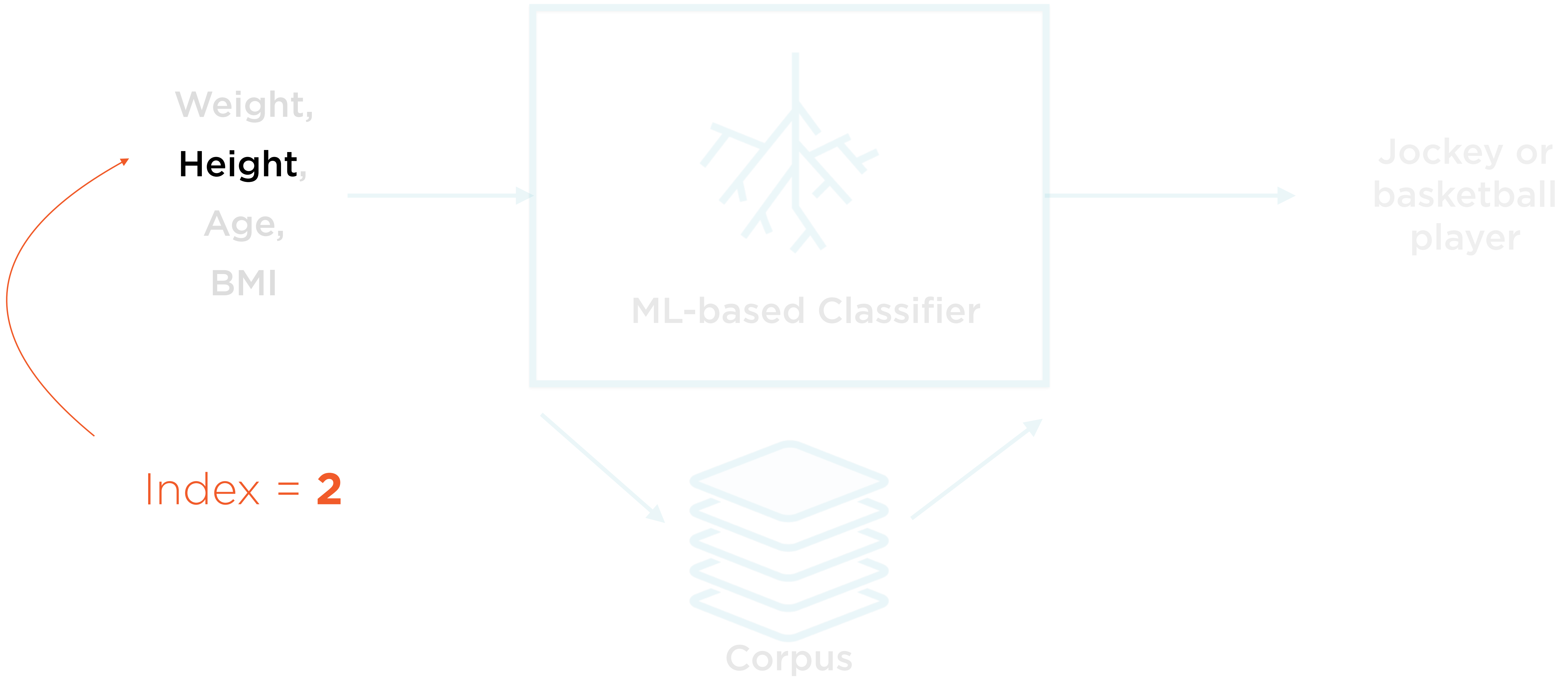
LIBSVM Data Format



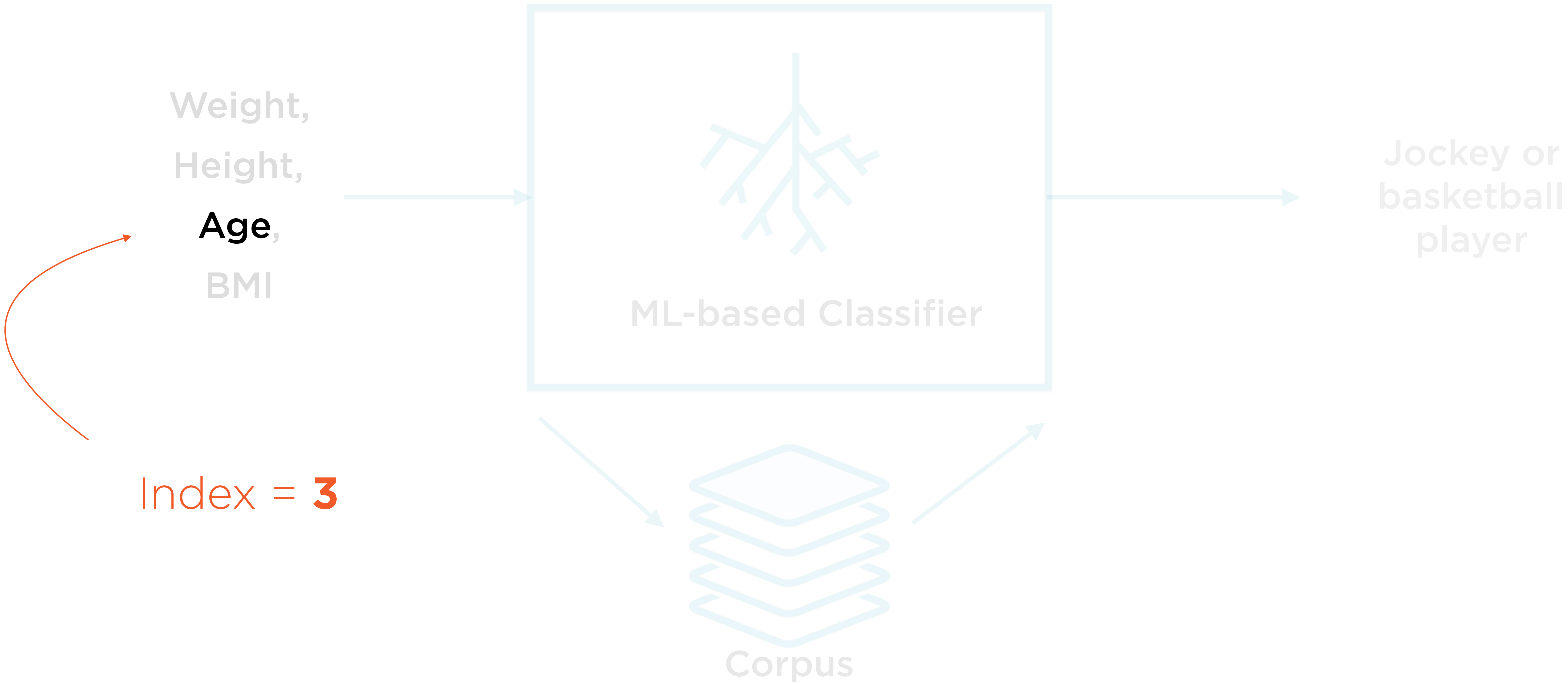
LIBSVM Data Format



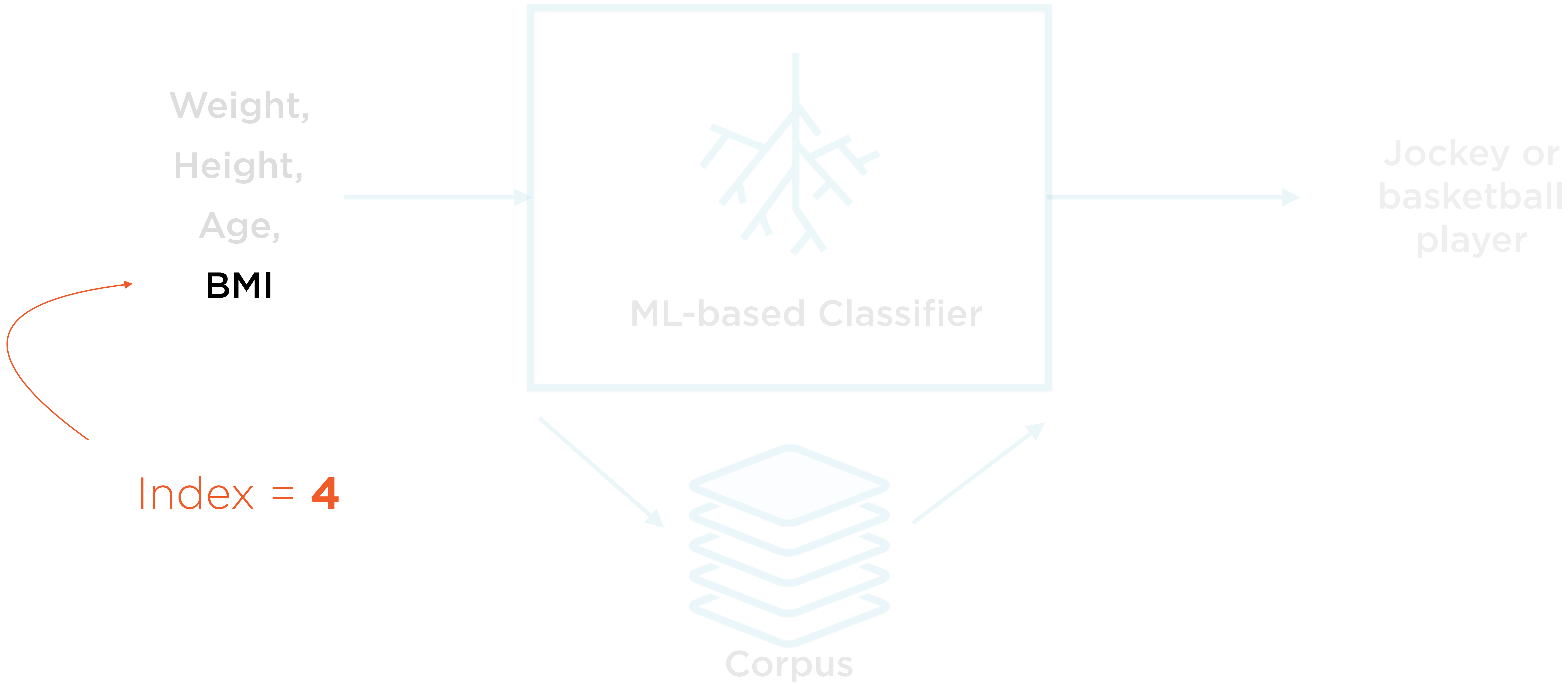
LIBSVM Data Format



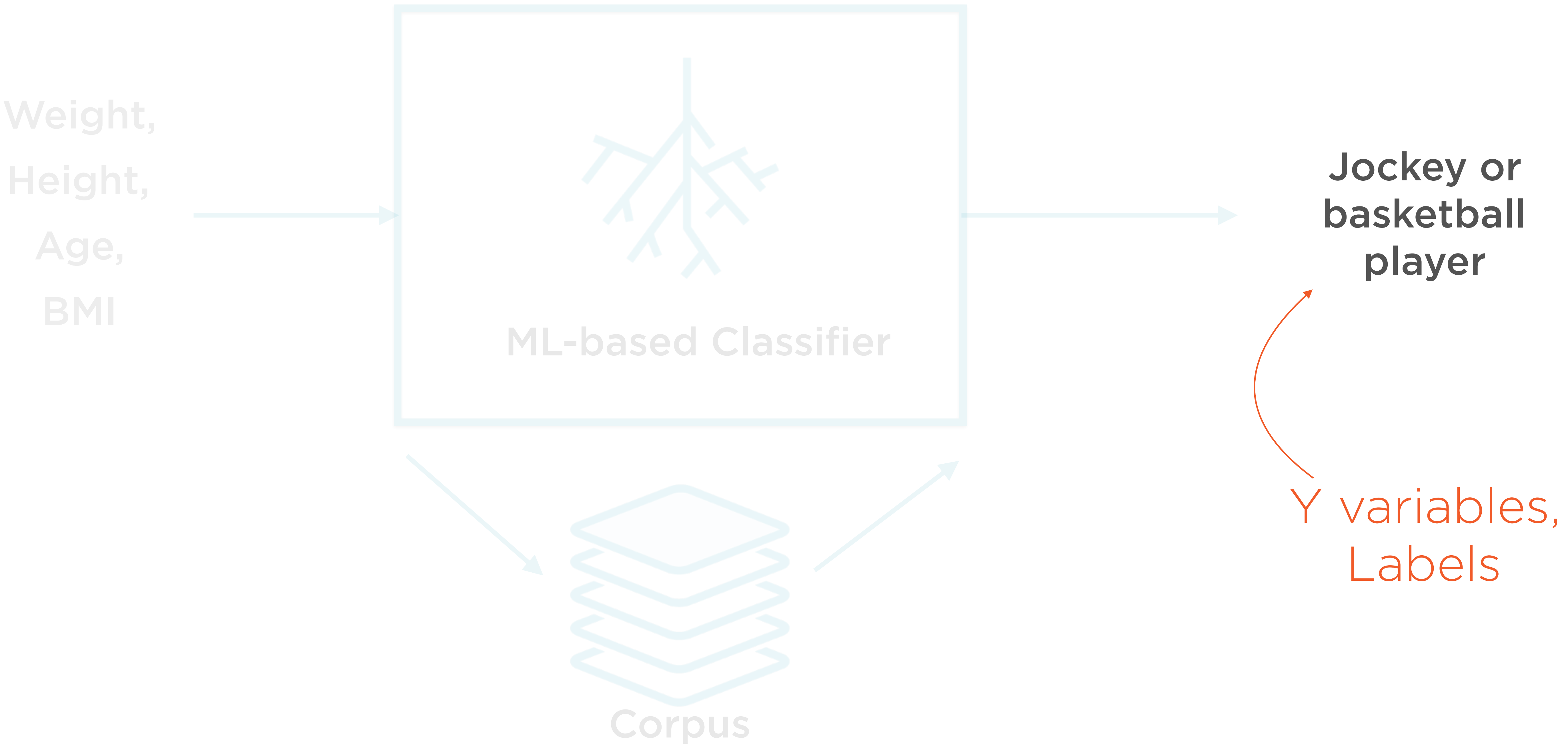
LIBSVM Data Format



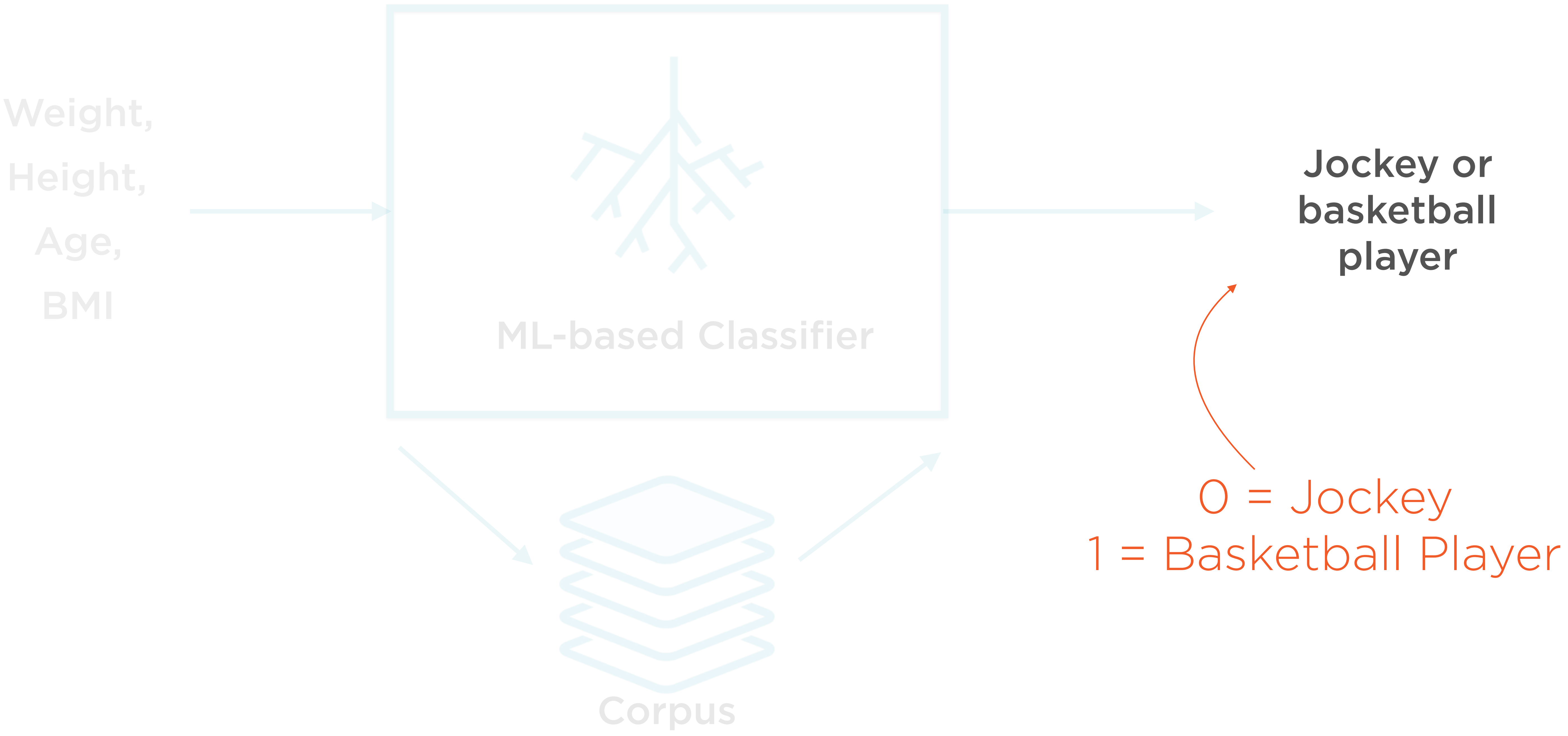
LIBSVM Data Format



LIBSVM Data Format



LIBSVM Data Format



LIBSVM Data Format

<label> <index1>:<value1> <index2>:<value2>...

0 1:230 2:188 3:32 4:29.4

1 line per instance

Each line ends with '\n'

Sparse - missing attributes can be omitted

LIBSVM Data Format

<label> <index1>:<value1> <index2>:<value2>...

0 1:230 2:188 3:32 4:29.4

0 = Jockey

LIBSVM Data Format

<label> **<index1>:**<value1> <index2>:<value2>...

0 **1:**230 2:188 3:32 4:29.4

Index = 1 for attribute **Weight**

LIBSVM Data Format

<label> **<index1>:<value1>** <index2>:<value2>...

0 **1:230** 2:188 3:32 4:29.4

Weight in lbs = 230

LIBSVM Data Format

<label> <index1>:<value1> **<index2>:**<value2>...

0 1:230 **2:**188 3:32 4:29.4

Index = 2 for attribute **Height**

LIBSVM Data Format

<label> <index1>:<value1> **<index2>:<value2>**...

0 1:230 **2:188** 3:32 4:29.4

Height in cm = 188

LIBSVM Data Format

<label> <index1>:<value1> <index2>:<value2>...

0 1:230 2:188 **3:32** 4:29.4

Index = 3 for attribute **Age**
Age in years = 32

LIBSVM Data Format

<label> <index1>:<value1> <index2>:<value2>...

0 1:230 2:188 3:32 **4:29.4**

Index = 4 for attribute **BMI**
BMI ratio = 29.4

LIBSVM Data Format

<label> <index1>:<value1> <index2>:<value2>...

0 1:230 2:188 3:32 4:29.4

1 line per instance

Each line ends with '\n'

Sparse - missing attributes take value 0

Missing Attributes

<label> <index1>:<value1> <index2>:<value2>...

0 1:145 2:158 3:39

Value for index 4 is missing

No worries - can calculate from height and weight

Sparse - missing attributes can be omitted

Summary

Spark 1.x provided powerful support for ML in spark.mllib

Spark 2.0 goes further with spark.ml

Faster execution

Ease of hyperparameter tuning

ETL support with ML pipelines

spark.mllib currently has more features but will be deprecated in the future