Welcome

“The Decision-Making Side of Machine Learning: Computational, Inferential, and Economic Perspectives”

Michael I. Jordan

Twitter Hashtag: #ACMLearning

Tweet questions & comments to: @ACMeducation

Post-Talk Discourse: https://on.acm.org

Additional Info:

- Talk begins at the top of the hour and lasts 60 minutes
- On the bottom panel you’ll find a number of widgets, including Twitter and Sharing apps
- For volume control, use your master volume controls and try headphones if it’s too low
- If you are experiencing any issues, try refreshing your browser or relaunching your session
- At the end of the presentation, you will help us out if you take the experience survey
- This session is being recorded and will be archived for on-demand viewing. You’ll receive an email when it’s available.

The Decision-Making Side of Machine Learning: Computational, Inferential, and Economic Perspectives

Speaker: Michael I. Jordan

Moderator: Michael Zeller

ACM.org Highlights

For Scientists, Programmers, Designers, and Managers:

- Learning Center - https://learning.acm.org
  - View past TechTalks & Podcasts with top inventors, innovators, entrepreneurs, & award winners
  - Access to O'Reilly Learning Platform – technical books, courses, videos, tutorials & case studies
  - Access to Skillsoft Training & ScienceDirect – vendor certification prep, technical books & courses
- Ethical Responsibility – https://ethics.acm.org

By the Numbers

- 2,200,000+ content readers
- 1,800,000+ DL research citations
- $1,000,000 Turing Award prize
- 100,000+ global members
- 1160+ Fellows
- 700+ chapters globally
- 170+ yearly conferences globally
- 100+ yearly awards
- 70+ Turing Award Laureates

Popular Publications & Research Papers

- Communications of the ACM - http://cacm.acm.org
- Queue Magazine - http://queue.acm.org
- Digital Library - http://dl.acm.org

Major Conferences, Events, & Recognition

- *https://www.acm.org/conferences*
- *https://www.acm.org/chapters*
- *https://awards.acm.org*

Welcome

“The Decision-Making Side of Machine Learning: Computational, Inferential, and Economic Perspectives”

Michael I. Jordan

Twitter Hashtag: #ACMLearning

Tweet questions & comments to: @ACMeducation

Post-Talk Discourse: https://on.acm.org

Additional Info:

- Talk begins at the top of the hour and lasts 60 minutes
- On the bottom panel you’ll find a number of widgets, including Twitter and Sharing apps
- For volume control, use your master volume controls and try headphones if it’s too low
- If you are experiencing any issues, try refreshing your browser or relaunching your session
- At the end of the presentation, you will help us out if you take the experience survey
- This session is being recorded and will be archived for on-demand viewing. You’ll receive an email when it’s available.

The Decision-Making Side of Machine Learning

Computational, Inferential and Economic Perspectives

Michael Jordan

University of California, Berkeley

Outline

- Some Historical Background
- Competing Bandits in Matching Markets
- Is Q-Learning Provably Efficient?
- Anytime Control of the False-Discovery Rate
- Ray: A Distributed Platform for Emerging Decision-Focused AI Applications

Machine Learning as an Engineering Discipline

- First Generation (‘90-’00): the backend
  - e.g., fraud detection, search, supply-chain management
- Second Generation (‘00-’10): the human side
  - e.g., recommendation systems, commerce, social media
- Third Generation (‘10-now): pattern recognition
  - e.g., speech recognition, computer vision, translation

Machine Learning as an Engineering Discipline

- First Generation ('90-'00): the backend
  - e.g., fraud detection, search, supply-chain management
- Second Generation ('00-'10): the human side
  - e.g., recommendation systems, commerce, social media
- Third Generation ('10-now): pattern recognition
  - e.g., speech recognition, computer vision, translation
- Fourth Generation (emerging): markets
  - not just one agent making a decision or sequence of decisions
  - but a huge interconnected web of data, agents, decisions
  - many new challenges!

Decisions

- It's not just a matter of a threshold

Decisions

- It’s not just a matter of a threshold
- Real-world decisions with consequences
  - counterfactuals, provenance, relevance, dialog

Decisions

- It's not just a matter of a threshold
- Real-world decisions with consequences
  - counterfactuals, provenance, relevance, dialog
- Sets of decisions across a network
  - false-discovery rate (instead of precision/recall/accuracy)

Decisions

- It's not just a matter of a threshold
- Real-world decisions with consequences
  - counterfactuals, provenance, relevance, dialog
- Sets of decisions across a network
  - false-discovery rate (instead of precision/recall/accuracy)
- Sets of decisions across a network over time
  - streaming, asynchronous decisions (cf. Zrnic, Ramdas & Jordan, *Asynchronous online testing of multiple hypotheses*, arXiv, 2019)

Decisions

- It's not just a matter of a threshold
- Real-world decisions with consequences
  - counterfactuals, provenance, relevance, dialog
- Sets of decisions across a network
  - false-discovery rate (instead of precision/recall/accuracy)
- Sets of decisions across a network over time
  - streaming, asynchronous decisions (cf. Zrnic, Ramdas & Jordan, *Asynchronous online testing of multiple hypotheses*, arXiv, 2019)
- Decisions when there is scarcity and competition
  - need for an economic perspective

Markets

- Markets can be viewed as decentralized algorithms
- They accomplish complex tasks like bringing the necessary goods into a city day in and day out
- They are adaptive (accommodating change in physical or social structure), robust (working rain or shine), scalable (working in small villages and big cities), and they can have a very long lifetime
  - indeed, they can work for decades or centuries
  - if we're looking for principles for lifelong adaptation, we should be considering markets as intelligent systems!
- Of course, markets aren't perfect, which simply means that there are research opportunities

Consider Classical Recommendation Systems

- A record is kept of each customer’s purchases
- Customers are “similar” if they buy similar sets of items
- Items are “similar” are they are bought together by multiple customers

Consider Classical Recommendation Systems

- A record is kept of each customer's purchases
- Customers are "similar" if they buy similar sets of items
- Items are "similar" are they are bought together by multiple customers
- Recommendations are made on the basis of these similarities
- These systems have become a commodity

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it’s very popular)
- Is it OK to recommend the same movie to everyone?

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it's very popular)
- Is it OK to recommend the same movie to everyone?
- Is it OK to recommend the same book to everyone?
- Is it OK to recommend the same restaurant to everyone?
- Is it OK to recommend the same street to every driver?
- Is it OK to recommend the same stock purchase to everyone?

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it’s very popular)
- Is it OK to recommend the same movie to everyone?
- Is it OK to recommend the same book to everyone?

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it’s very popular)
- Is it OK to recommend the same movie to everyone?
- Is it OK to recommend the same book to everyone?
- Is it OK to recommend the same restaurant to everyone?

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it’s very popular)
- Is it OK to recommend the same movie to everyone?
- Is it OK to recommend the same book to everyone?
- Is it OK to recommend the same restaurant to everyone?
- Is it OK to recommend the same street to every driver?

Multiple Decisions with Competition

- Suppose that recommending a certain movie is a good business decision (e.g., because it's very popular)
- Is it OK to recommend the same movie to everyone?
- Is it OK to recommend the same book to everyone?
- Is it OK to recommend the same restaurant to everyone?
- Is it OK to recommend the same street to every driver?
- Is it OK to recommend the same stock purchase to everyone?

The Alternative: Create a Market

- A two-way market between consumers and producers
  - based on recommendation systems on both sides
- E.g., diners are one side of the market, and restaurants on the other side
- E.g., drivers are one side of the market, and street segments on the other side

- This isn't just classical microeconomics; the use of recommendation systems via data analysis is key

Example: Music in the Data Age

- More people are making music than ever before, placing it on sites such as SoundCloud
- More people are listening to music than ever before
- But there is no economic value being exchanged between producers and consumers
- And, not surprisingly, most people who make music cannot do it as their full-time job
  - i.e., human happiness is being left on the table

Example: Music in the Data Age

- More people are making music than ever before, placing it on sites such as SoundCloud
- More people are listening to music than ever before
- But there is no economic value being exchanged between producers and consumers
- And, not surprisingly, most people who make music cannot do it as their full-time job
  - i.e., human happiness is being left on the table

- There do exist companies who make money off of this; they stream data from SoundCloud to listeners, and they make their money … from advertising! ☹

The Alternative: Create a Market

- Use data to provide a dashboard to musicians, letting them learn where their audience is
- The musician can give shows where they have an audience
- And they can make offers to their fans

The Alternative: Create a Market

- Use data to provide a dashboard to musicians, letting them learn where their audience is
- The musician can give shows where they have an audience
- And they can make offers to their fans
- I.e., consumers and producers become linked, and value flows: a market is created
  - the company that creates this market profits simply by taking a cut from the transactions

The Alternative: Create a Market

- Use data to provide a dashboard to musicians, letting them learn where their audience is
- The musician can give shows where they have an audience
- And they can make offers to their fans
- I.e., consumers and producers become linked, and value flows: a market is created
  - the company that creates this market profits simply by taking a cut from the transactions

- In the US, the company *United Masters* is doing precisely this; see www.unitedmasters.com

Social Consequences

- By creating a market based on the data flows, new jobs are created!
- So here's a way that AI can be a job creator, and not (mostly) a job killer
- This can be done in a wide range of other domains, not just music
  - entertainment
  - information services
  - personal services
- The markets-meets-learning approach deals with other problems that a pure learning approach does not
  - e.g., recommendations when there is scarcity

Examples at the Interface of ML and Econ

- Multi-way markets in which the individual agents need to explore to learn their preferences
- Large-scale multi-way markets in which agents view other sides of the market via recommendation systems
- Inferential methods for mitigating information asymmetries
- Latent variable inference in game theory
- Data collection in strategic settings
- Information sharing, free riding

- The goal is to discover new principles to build healthy (e.g., fair) learning-based markets that are stabilized over long stretches of time

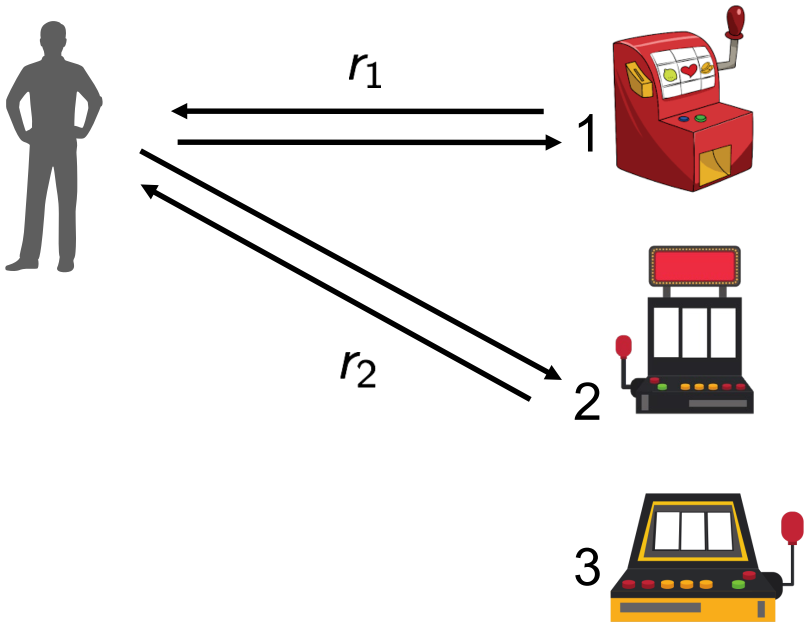
Competing Bandits in Matching Markets

Lydia Liu

Horia Mania

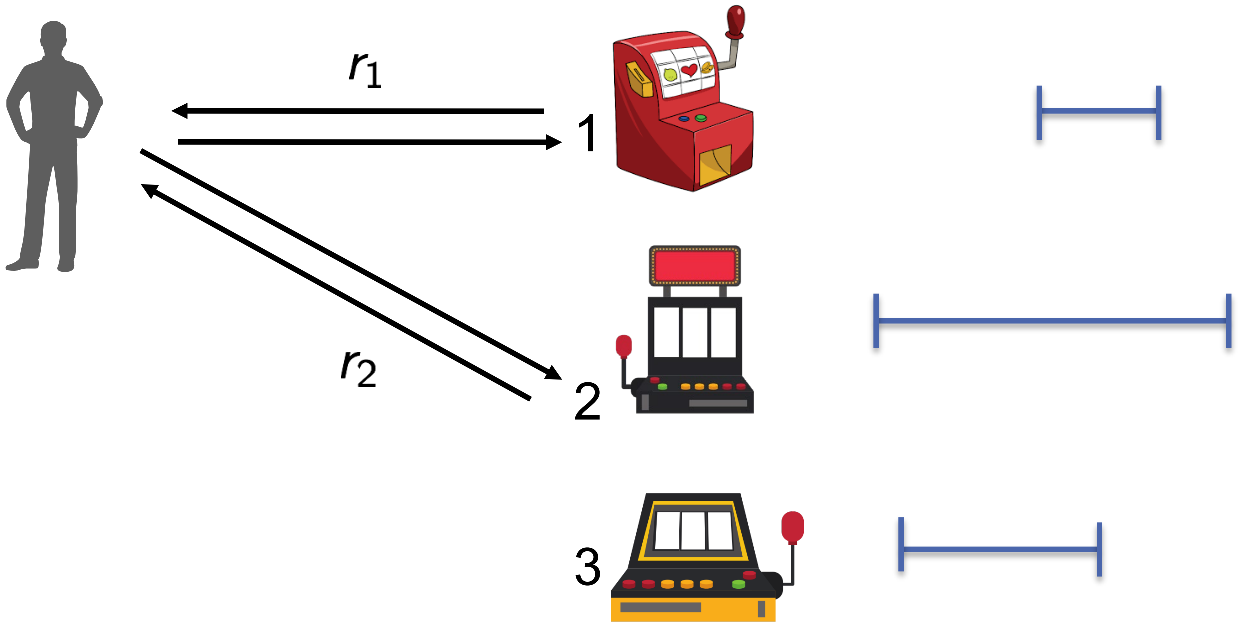
Multi-Armed Bandits

- MABs offer a natural platform to understand exploration / exploitation trade-offs



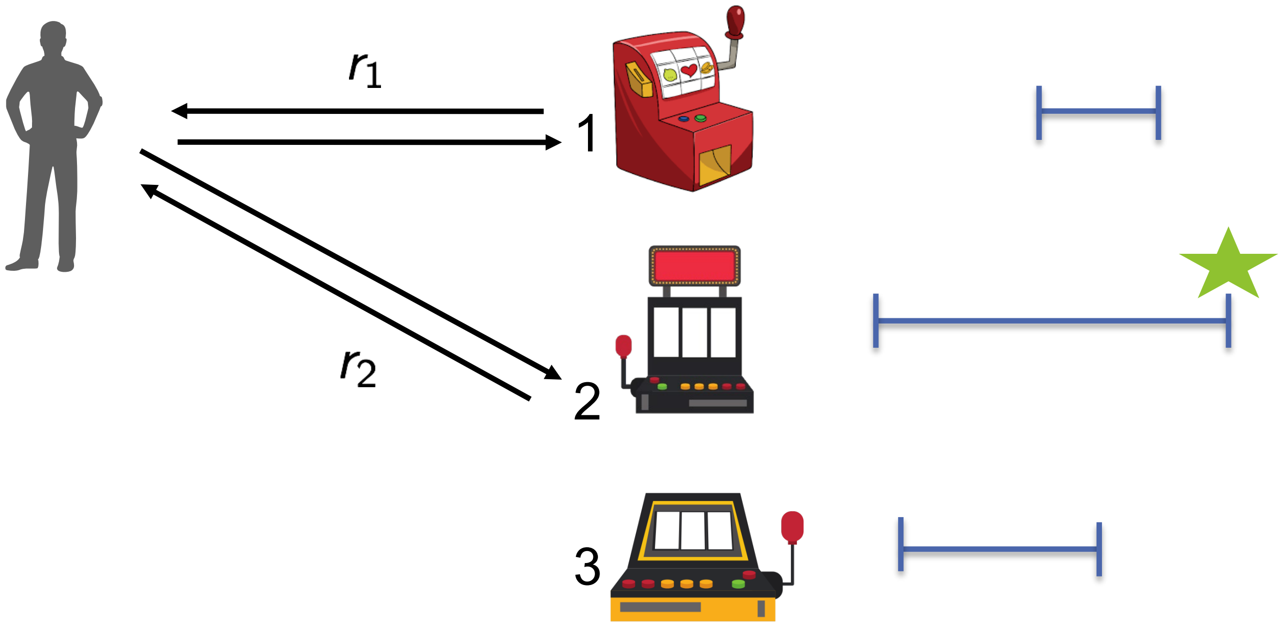
Upper Confidence Bound (UCB) Algorithm

- Maintain an upper confidence bound on reward values



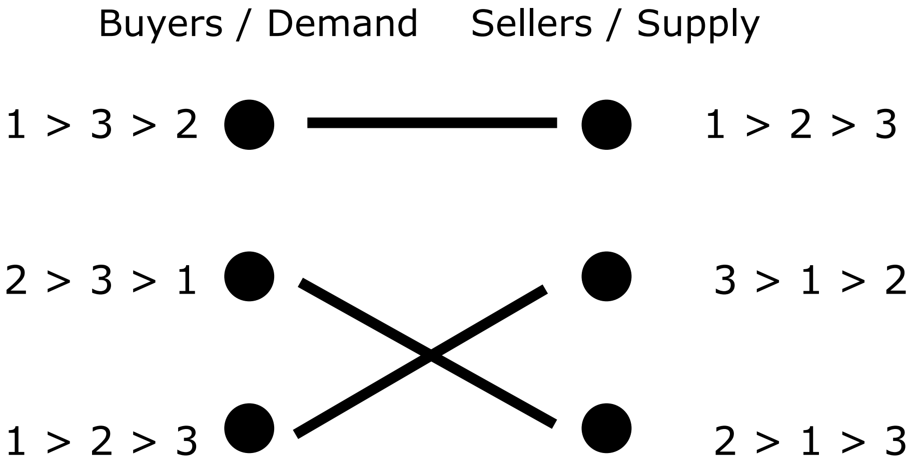
Upper Confidence Bound (UCB) Algorithm

- Maintain an upper confidence bound on reward values
- Pick the arm with the largest upper confidence bound



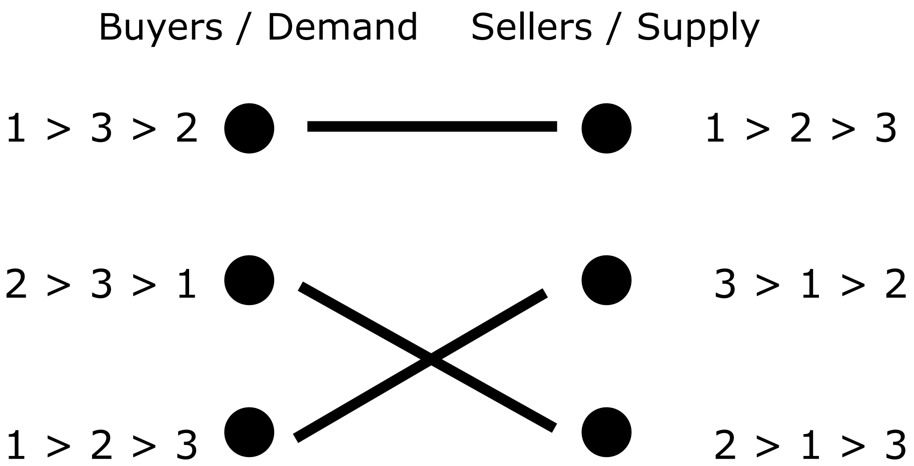
Matching Markets

Suppose we have a market in which the participants have preferences:



Matching Markets

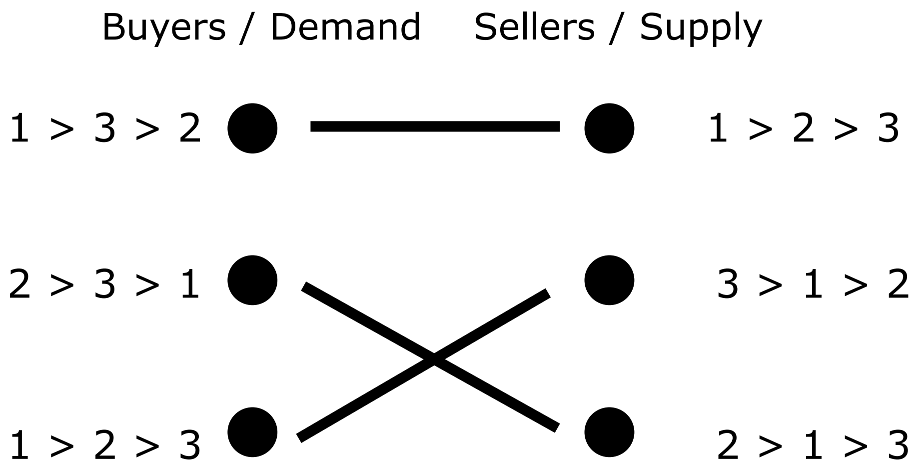
Suppose we have a market in which the participants have preferences:



Gale and Shapley introduced this problem in 1962 and proposed a celebrated algorithm that always finds a stable match

Matching Markets

Suppose we have a market in which the participants have preferences:



Gale and Shapley introduced this problem in 1962 and proposed a celebrated algorithm that always finds a stable match

In this algorithm one side of the market iteratively makes proposals to the other side

Matching Markets Meet Bandit Learning

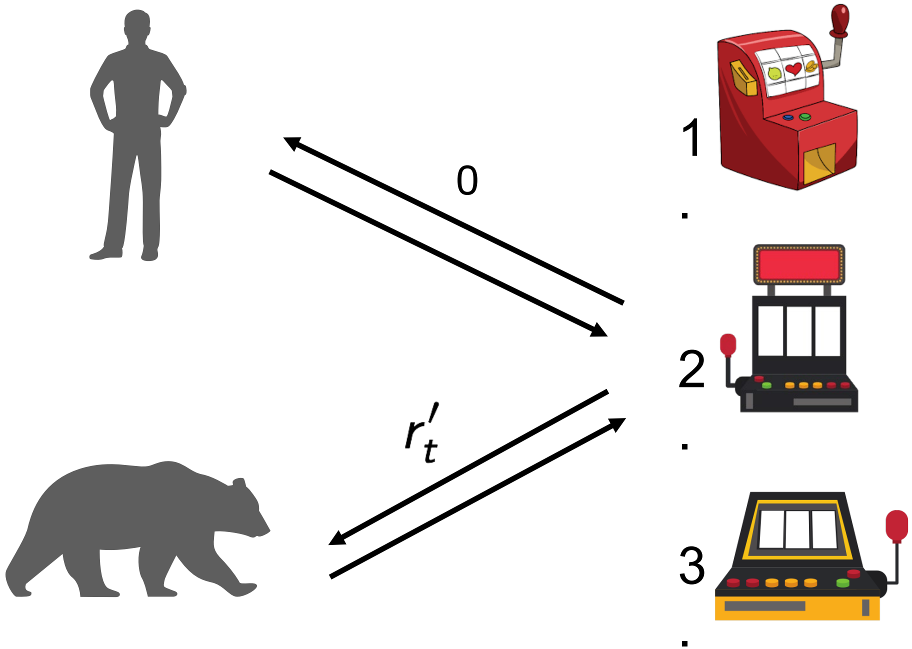
What if the participants in the market do not know their preferences a priori, but observe noisy utilities through repeated interactions?

Matching Markets Meet Bandit Learning

What if the participants in the market do not know their preferences a priori, but observe noisy utilities through repeated interactions?

Now the participants have an exploration/exploitation problem, in the context of other participants

Competing Agents



Bandit Markets

- We conceive of a bandit market: agents on one side, arms on the other side.

Agents get noisy rewards when they pull arms.

Arms have preferences over agents (these preferences can also express agents’ skill levels)

When multiple agents pull the same arm only the most preferred agent gets a reward.

Regret in Bandit Markets

Then it is natural to define the regret of agent i up to time n as:

$$R_i(n) = n\underbrace{\mu_i(m(i))}_{\text{Mean reward of stable match}} - \sum_{t=1}^{n} \mathbb{E}\underbrace{X_{i,m_t}(t)}_{\text{Reward at time t}}$$

Minimizing this regret is natural. It says that agents should expect rewards as good as their stable match in hindsight.

Regret-Minimizing Algorithm

Gale-Shapley upper confidence bounds (GS-UCB):

- Agents rank arms according to upper confidence bounds for the mean rewards.
- Agents submit rankings to a matching platform.
- The platform uses these rankings to run the Gale-Shapley algorithm to match agents and arms.
- Agents receive rewards and update upper confidence bounds.
- Repeat.

Theorem

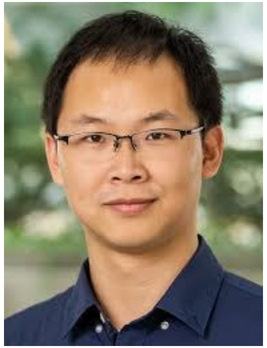
Theorem (informal): If there are N agents and K arms and GS-UCB is run, the regret of agent i satisfies

$$R_i(n) = \mathcal{O}\left(\frac{NK\log(n)}{\boxed{\Delta^2}}\right)$$

Reward gap of possibly other agents.

- In other words, if the bear decides to explore more, the human might have higher regret.
- See paper for refinements of this bound and further discussion of exploration-exploitation trade-offs in this setting.
- Finally, we note that GS-UCB is incentive compatible. No single agent has an incentive to deviate from the method.

UCB Meets Reinforcement Learning (aka, Is Q-Learning Provably Efficient?)



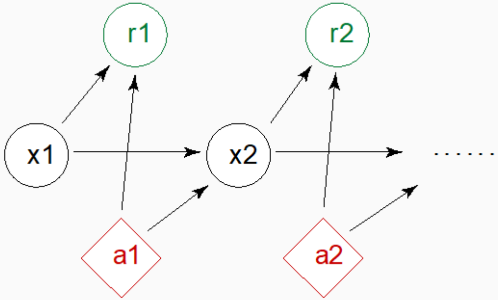
Chi Jin

Zeyuan Allen-Zhu

Sebastien Bubeck

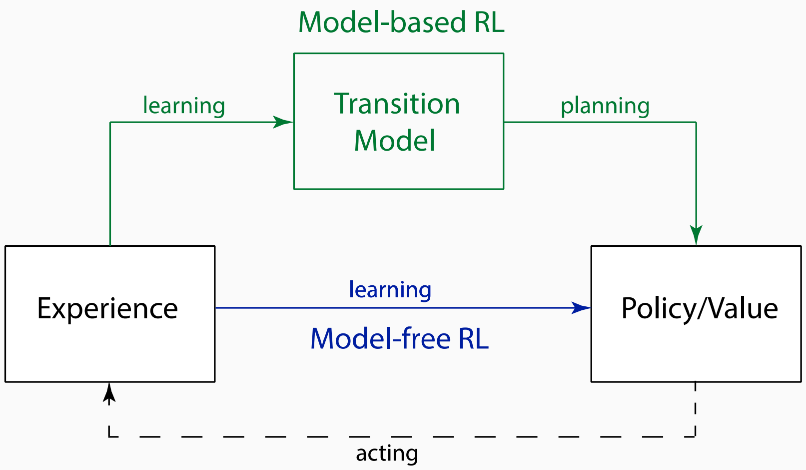
Reinforcement Learning

Maximize the cumulative rewards via interacting with an unknown environment.



Markov Decision Process MDP(S, A, P, r): state set S, action set A, transition model $\mathrm{P}(\cdot|x, a)$, and reward function $r : S \times A \to \mathbb{R}$.

Model-Based vs. Model-Free RL



Model-based algorithms: Value/policy iteration using empirical transition matrix.

Model-free algorithms: Q-learning; policy gradient methods.

Q-Learning

Q-value:

$$Q_h^{\pi}(x, a) = \mathbb{E}[\text{total reward} \mid (x_h, a_h) = (x, a)]$$

Optimal Bellman Equation:

$$Q_h^{\star}(x, a) = r_h(x, a) + \mathbb{E}_{x' \sim \mathbb{P}_h(\cdot|x,a)} \max_{a' \in \mathcal{A}} Q_{h+1}^{\star}(x', a')$$

Q-learning with ϵ-Greedy

In each step of each episode:

1. Take action $a_h \leftarrow \begin{cases} \operatorname{argmax}_{a'} Q_h(x_h, a') & \text{w.p. } 1 - \epsilon \\ \text{random action} & \text{w.p. } \epsilon \end{cases}$, and observe x_{h+1}.
2. $Q_h(x_h, a_h) \leftarrow (1 - \alpha) Q_h(x_h, a_h) + \alpha[r_h(x_h, a_h) + \max_{a' \in \mathcal{A}} Q_{h+1}(x_{h+1}, a')]$

Two flexible pieces: (1) exploration strategy; (2) learning rate α.

Q-Learning with UCB

Q-learning with UCB-Hoeffding

In each step of each episode:

1. Take action $a_h \leftarrow \mathrm{argmax}_{a'} Q_h(x_h, a')$, and observe x_{h+1}.
2. $Q_h(x_h, a_h) \leftarrow (1 - \alpha_t) Q_h(x_h, a_h) + \alpha_t [r_h(x_h, a_h) + V_{h+1}(x_{h+1}) + b_t]$
3. $V_h(x_h) \leftarrow \min\{H, \max_{a' \in \mathcal{A}} Q_h(x_h, a')\}$

- Counts: $t = N_h(x_h, a_h)$.
- UCB bonus: $b_t = \tilde{\mathcal{O}}(\sqrt{H^3/t})$.
- Learning rate: $\alpha_t = \mathcal{O}(H/t)$.

Theoretical Guarantees

Theorem (Hoeffding version)

W.h.p, the total regret of Q-learning with UCB-Hoeffding is at most $\tilde{\mathcal{O}}(\sqrt{H^4 SAT})$.

- Bernstein version has $\tilde{\mathcal{O}}(\sqrt{H^3 SAT})$ regret $\Leftrightarrow$ $\tilde{\mathcal{O}}(H^4 SA/\epsilon^2)$ samples.
- Only one $\sqrt{H}$ factor worse than best model-based regret (UCBVI, Azar et al.), but with significantly better time and space complexity.

Key components in analysis:

- design Upper Confidence Bound (UCB);
- favoring later updates.

State of the Theory

	Algorithm	Regret	Time	Space
Model-based	RLSVI	$\tilde{\mathcal{O}}(\sqrt{H^3SAT})$	$\mathcal{O}(TS^2A^2)$	$\mathcal{O}(S^2A^2H)$
	UCRL2	at least $\tilde{\mathcal{O}}(\sqrt{H^4S^2AT})$	$\Omega(TS^2A)$	$\mathcal{O}(S^2AH)$
	Agrawal & Jia, 2017	at least $\tilde{\mathcal{O}}(\sqrt{H^3S^2AT})$		
	UBEV	$\tilde{\mathcal{O}}(\sqrt{H^4SAT})$	$\tilde{\mathcal{O}}(TS^2A)$	
	UCBVI	$\tilde{\mathcal{O}}(\sqrt{H^2SAT})$		
Model-free	Q-learning (ϵ-greedy) (if 0 initialized)	$\Omega(\min\{T, A^{H/2}\})$	$\mathcal{O}(T)$	$\mathcal{O}(SAH)$
	Delayed Q-learning	$\tilde{\mathcal{O}}_{S,A,H}(T^{4/5})$		
	Q-learning (UCB-H)	$\tilde{\mathcal{O}}(\sqrt{H^4SAT})$		
	Q-learning (UCB-B)	$\tilde{\mathcal{O}}(\sqrt{H^3SAT})$		
	lower bound	$\Omega(\sqrt{H^2SAT})$	-	-

*H: # of steps per episode S: # of states A: # of actions T: total # of steps played.

*The table is presented for $T \geq \text{poly}(S, A, H)$, omitting low order terms.

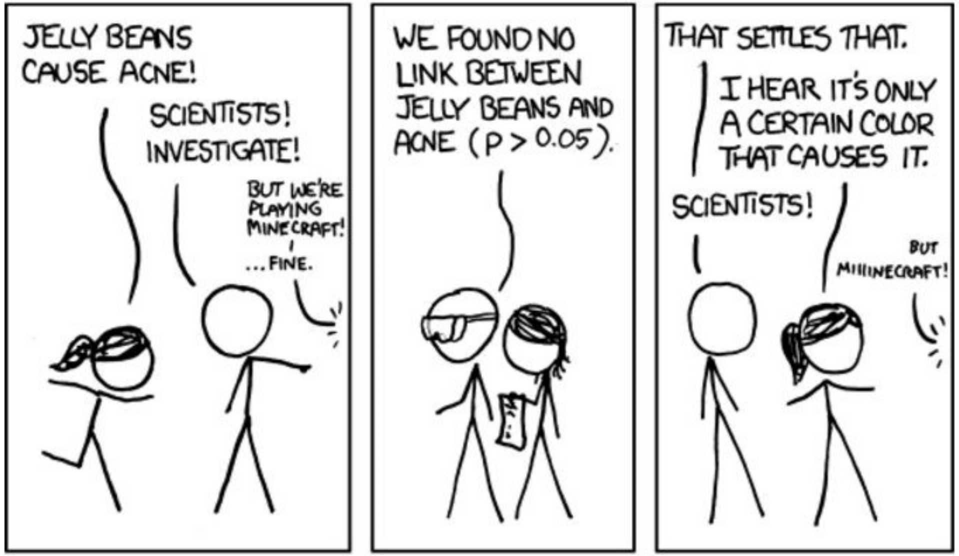
Anytime Control of the False-Discovery Rate

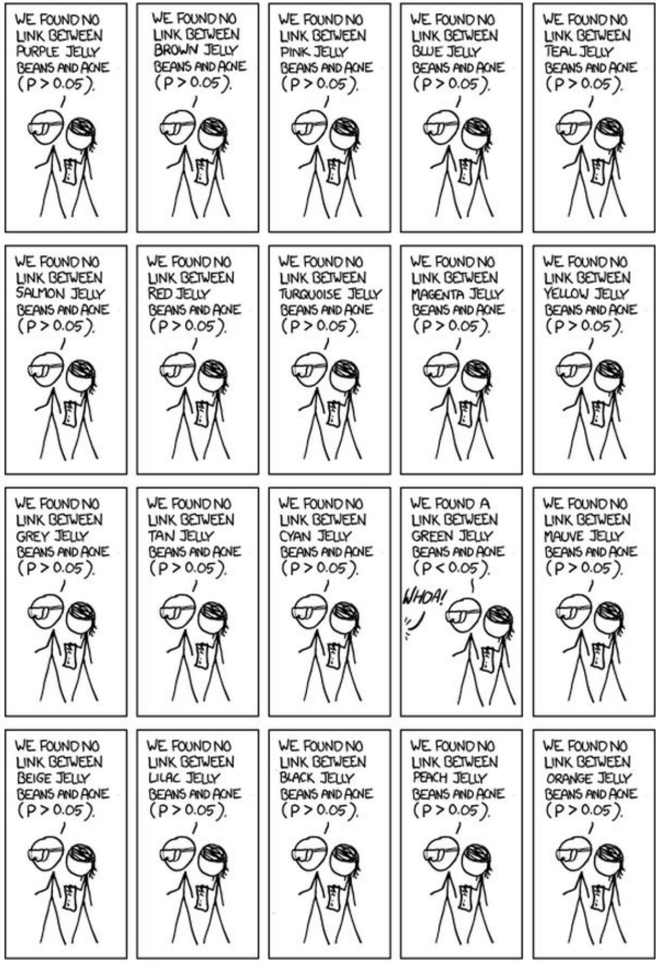
Aaditya
Ramdas

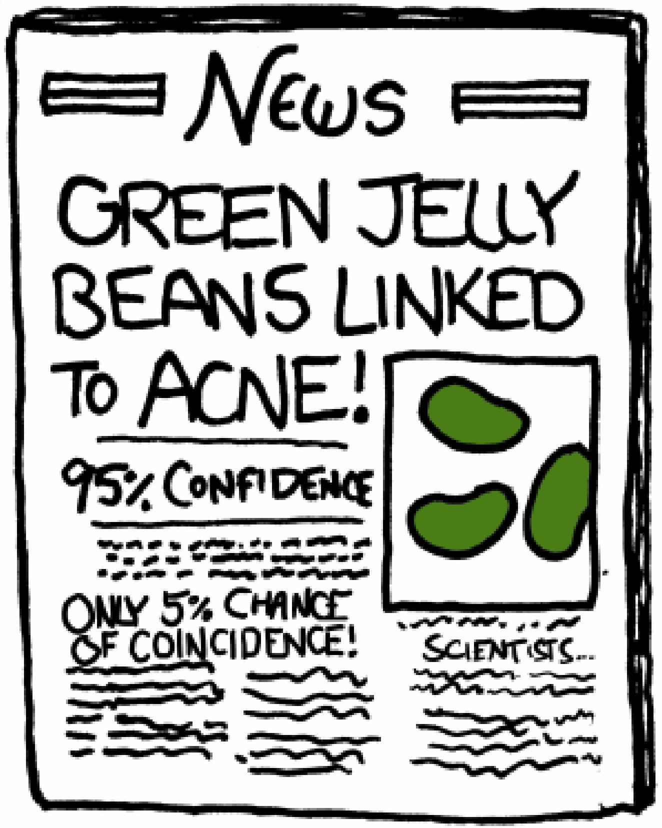
Tijana
Zrnic

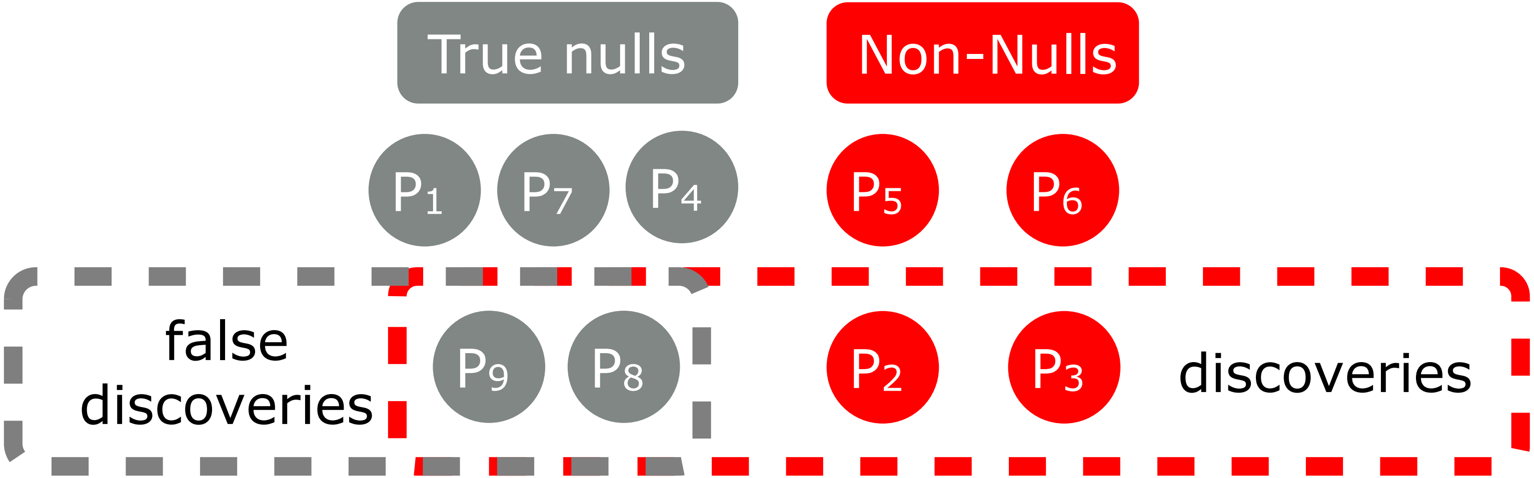
Foster-Stine ’08
Aharoni-Rosset ‘14
Javanmard-Montanari ’16

Multiple Decisions: The Statistical Problem

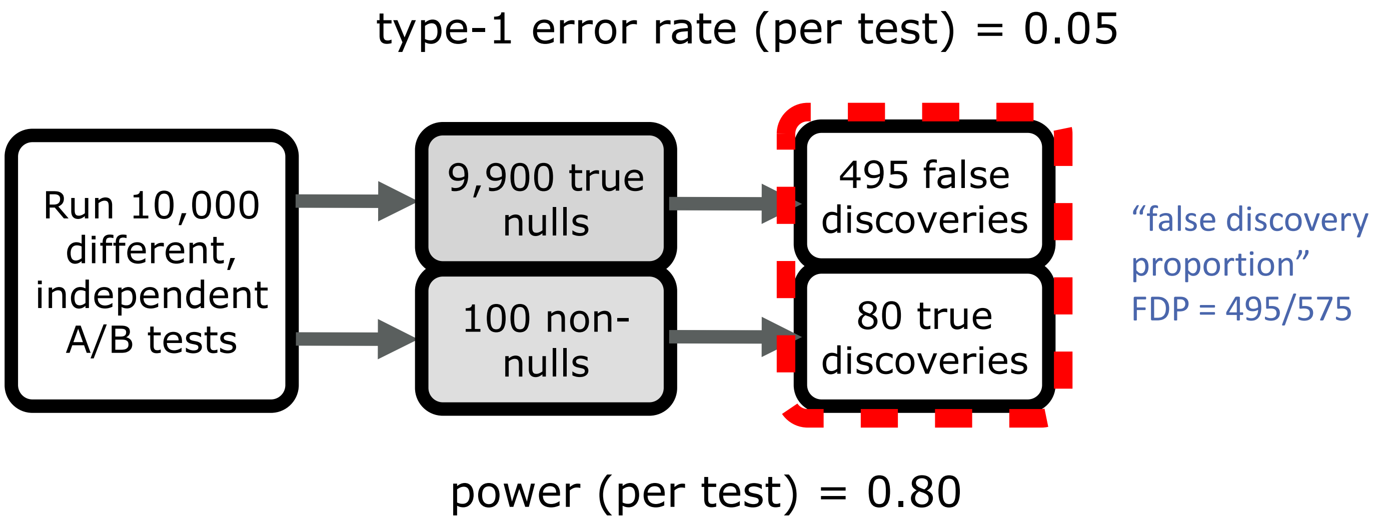



WE FOUND NO LINK BETWEEN PURPLE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BROWN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN PINK JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BLUE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TEAL JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN SALMON JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN RED JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TURQUOISE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN MAGENTA JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN YELLOW JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN GREY JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN TAN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN CYAN JELLY BEANS AND ACNE (P > 0.05).
WE FOUND A LINK BETWEEN GREEN JELLY BEANS AND ACNE (P < 0.05).
WHOA!
WE FOUND NO LINK BETWEEN MAUVE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BEIGE JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN LILAC JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN BLACK JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN PEACH JELLY BEANS AND ACNE (P > 0.05).
WE FOUND NO LINK BETWEEN ORANGE JELLY BEANS AND ACNE (P > 0.05).


News
GREEN JELLY BEANS LINKED TO ACNE!
95% CONFIDENCE
ONLY 5% CHANCE OF COINCIDENCE!
SCIENTISTS...



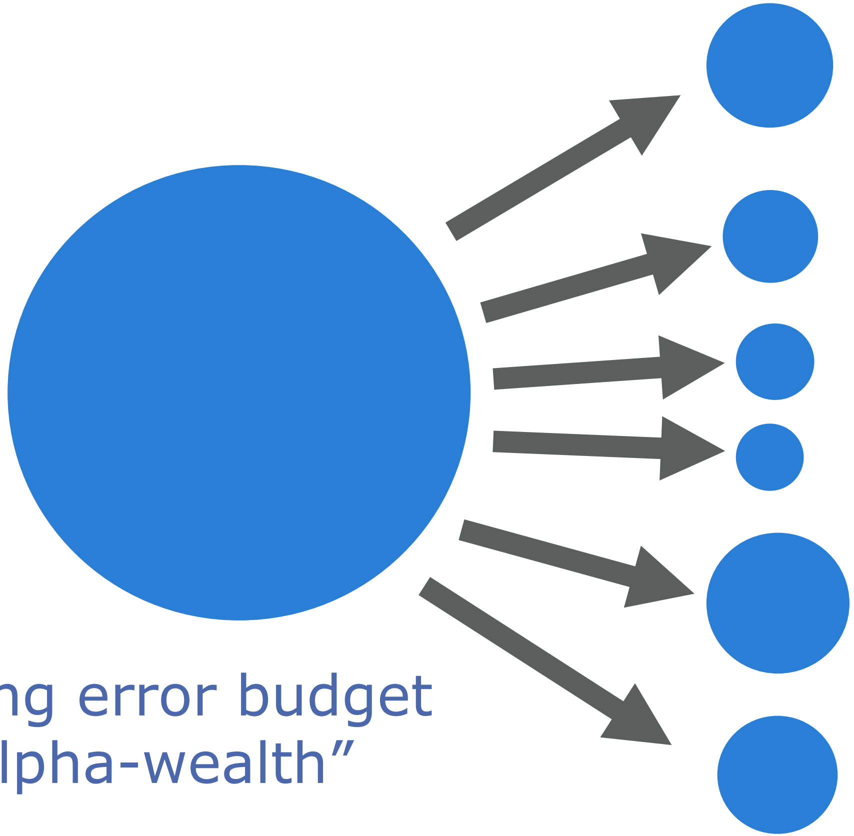
- False discovery proportion $\mathrm{FDP} = \dfrac{\#\text{ false discoveries}}{\#\text{ discoveries}}$
- Want low false discovery rate $\mathrm{FDR} = \mathbb{E}[\mathrm{FDP}]$
- Want high $\mathrm{Power} = \mathbb{E}\left[\dfrac{\#\text{ true discoveries}}{\#\text{ non-nulls}}\right]$



Summary: FDR can be larger than per-test error rate.

(even if hypotheses, tests, data are independent)

Online FDR control : high-level picture



Error budget
for first test

Error budget for
second test

Tests use wealth

Discoveries
earn wealth

Error budget
is data-dependent

Infinite process

Remaining error budget
or "alpha-wealth"

Ray: A Distributed Platform for Emerging Decision-Focused AI Applications

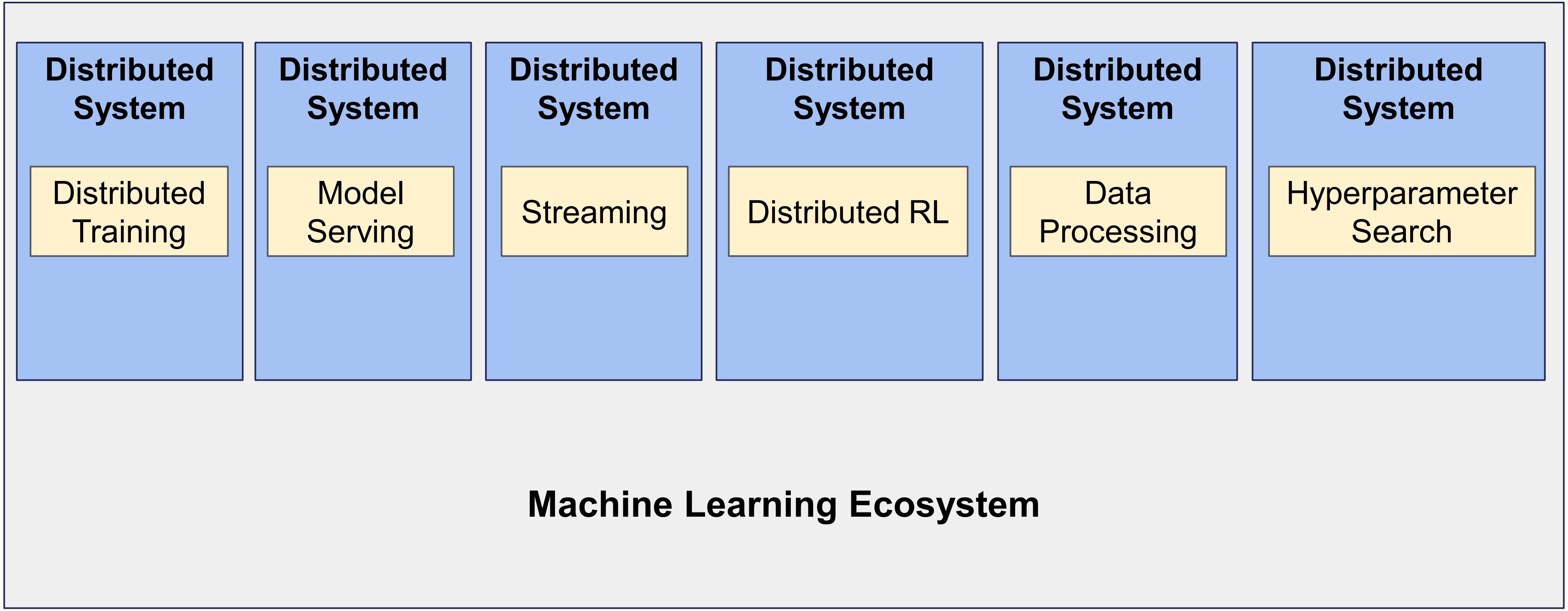
with *P Moritz, R Nishihara, S Wang, A Tumanov, R Liaw, E Liang, and I Stoica*

What is Ray?

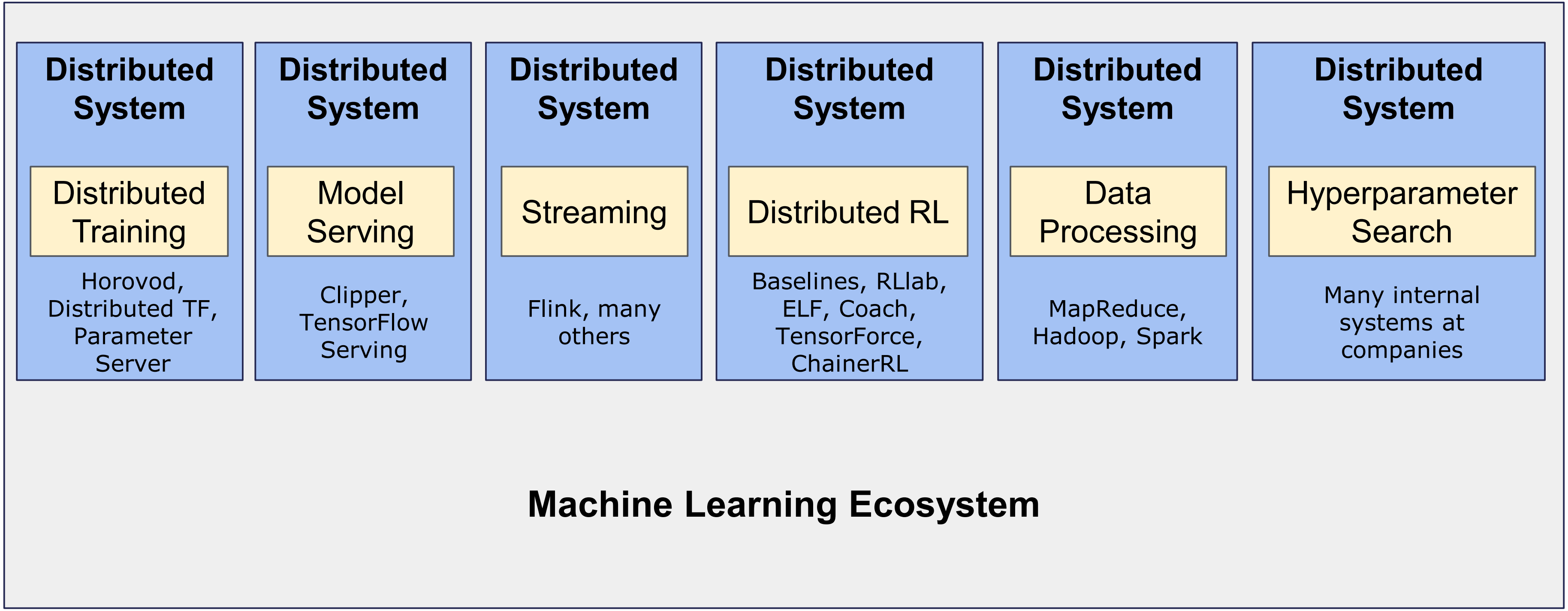
- Distributed Training
- Model Serving
- Streaming
- Distributed RL
- Data Processing
- Hyperparameter Search

Machine Learning Ecosystem

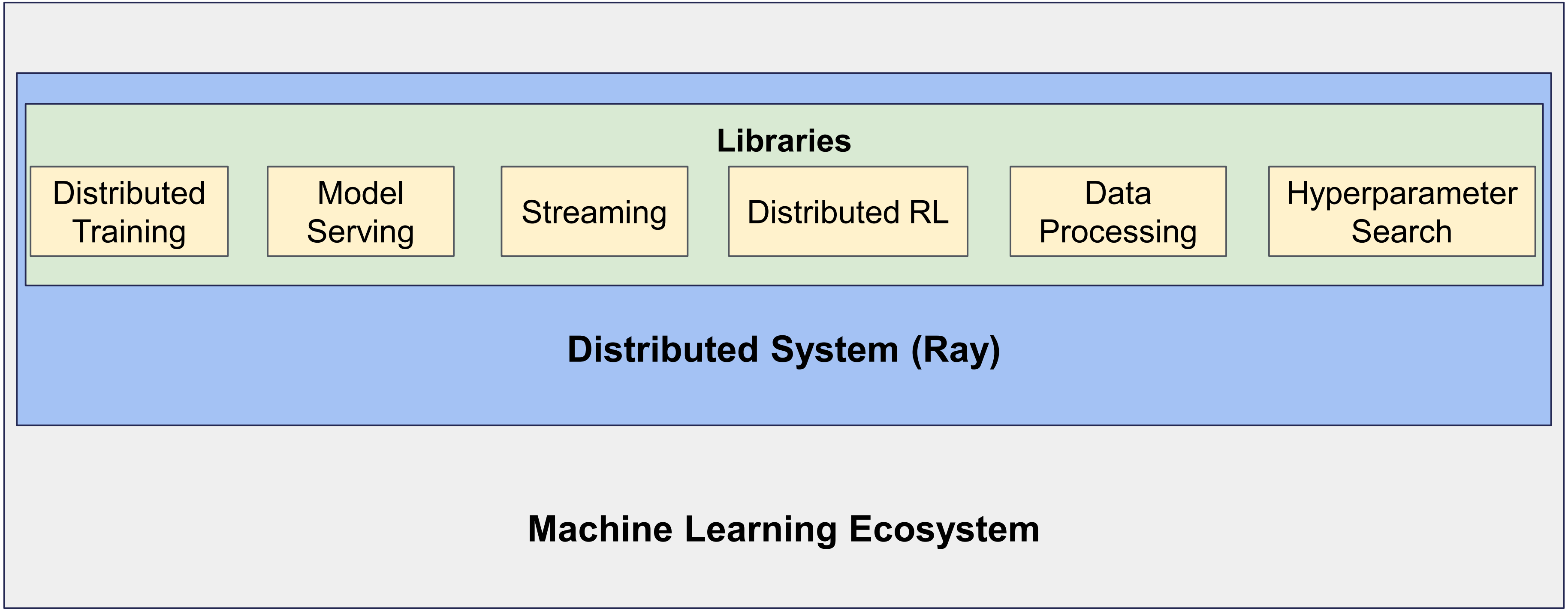
What is Ray?


Distributed System
Distributed Training
Distributed System
Model Serving
Distributed System
Streaming
Distributed System
Distributed RL
Distributed System
Data Processing
Distributed System
Hyperparameter Search
Machine Learning Ecosystem

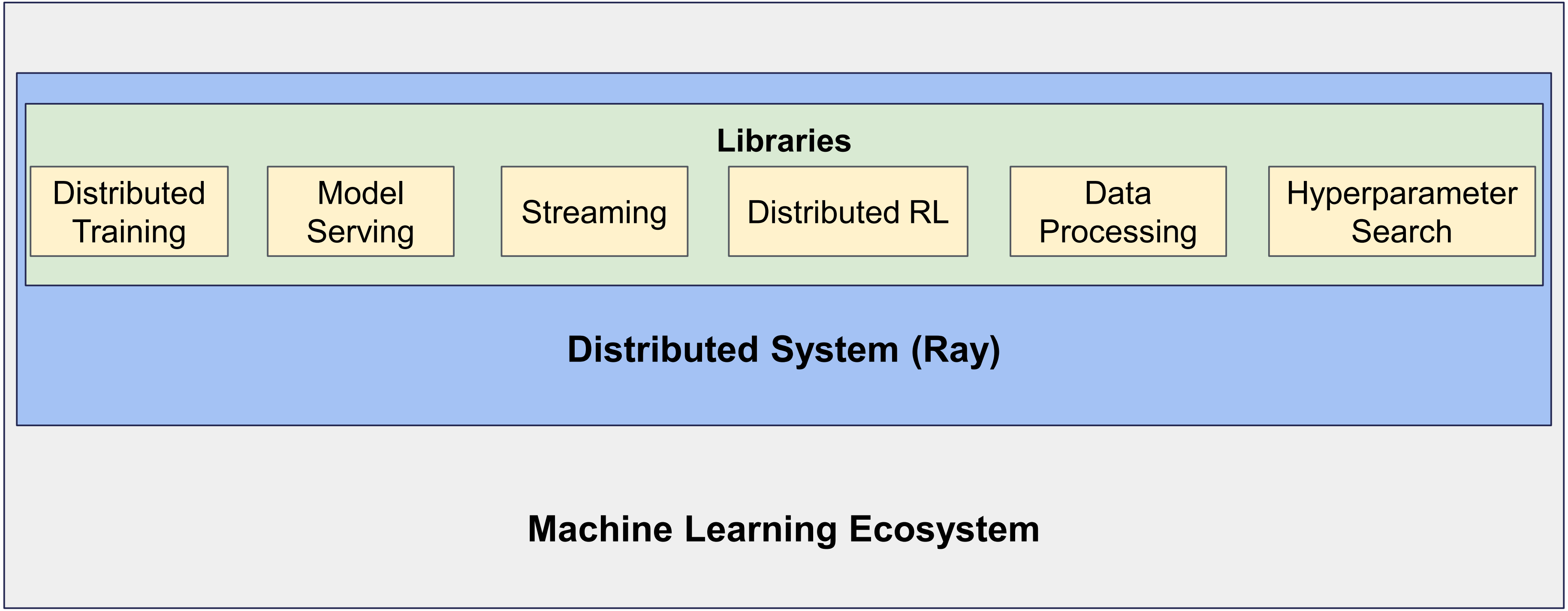
What is Ray?


Distributed System
Distributed Training
Horovod, Distributed TF, Parameter Server
Distributed System
Model Serving
Clipper, TensorFlow Serving
Distributed System
Streaming
Flink, many others
Distributed System
Distributed RL
Baselines, RLlab, ELF, Coach, TensorForce, ChainerRL
Distributed System
Data Processing
MapReduce, Hadoop, Spark
Distributed System
Hyperparameter Search
Many internal systems at companies
Machine Learning Ecosystem

What is Ray?


Libraries
Distributed Training
Model Serving
Streaming
Distributed RL
Data Processing
Hyperparameter Search
Distributed System (Ray)
Machine Learning Ecosystem

What is Ray?



Libraries
Distributed Training
Model Serving
Streaming
Distributed RL
Data Processing
Hyperparameter Search
Distributed System (Ray)
Machine Learning Ecosystem

Programming Languages

What do modern languages such as Python provide?

- Functions
- Objects

Programming Languages

What do modern languages such as Python provide?

- Functions
- Objects

What is the focus of distributed frameworks such as Hadoop and Spark?

- Functions

Ray as a Language for Distributed Computing

What does Ray provide?

- Distributed functions (“tasks”)
- Distributed objects (“actors”)

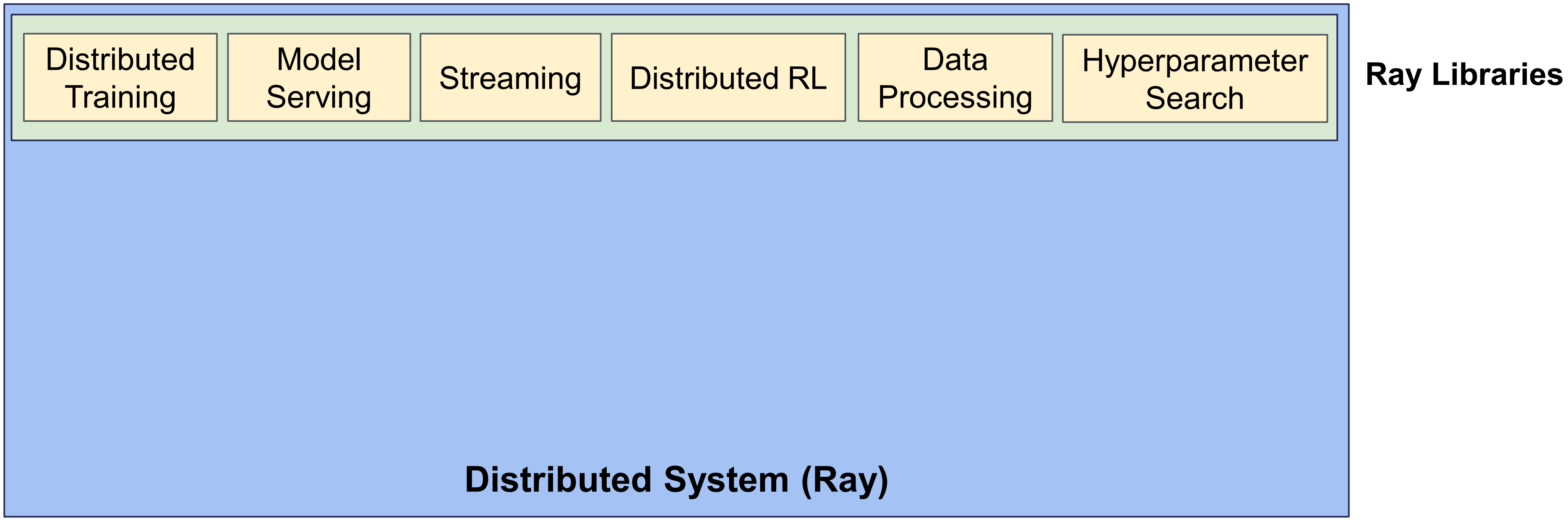
Ray as a Language for Distributed Computing

What does Ray provide?

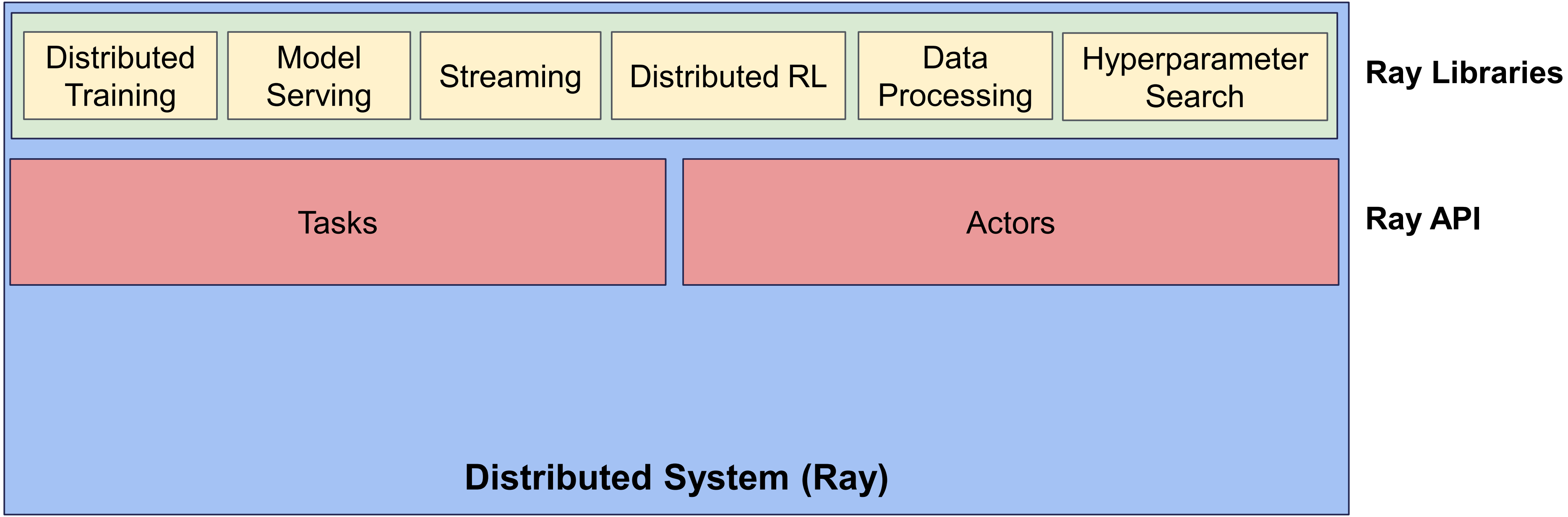
- Distributed functions (“tasks”)
- Distributed objects (“actors”)

The distributed implementation is provided by the Ray system, with the user not needing to know anything about the details of the implementation

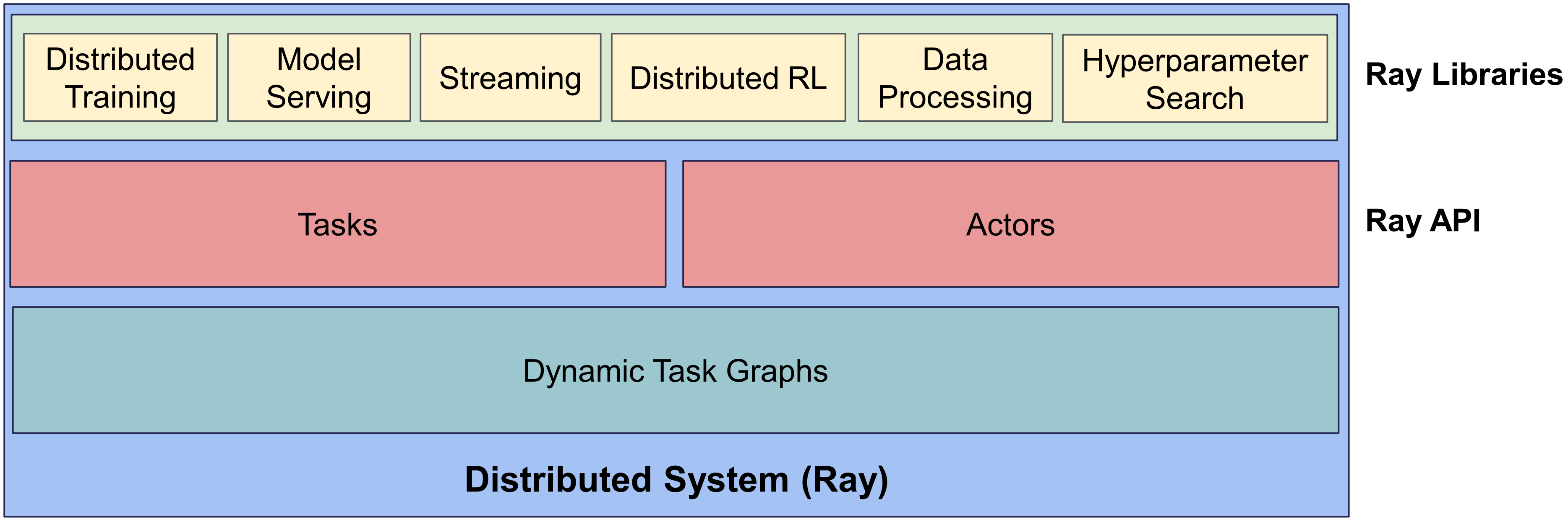
What is Ray?


Distributed Training
Model Serving
Streaming
Distributed RL
Data Processing
Hyperparameter Search
Ray Libraries
Distributed System (Ray)

What is Ray?


Distributed Training
Model Serving
Streaming
Distributed RL
Data Processing
Hyperparameter Search
Ray Libraries
Tasks
Actors
Ray API
Distributed System (Ray)

What is Ray?


Distributed Training
Model Serving
Streaming
Distributed RL
Data Processing
Hyperparameter Search
Ray Libraries
Tasks
Actors
Ray API
Dynamic Task Graphs
Distributed System (Ray)

The Ray API

```
def zeros(shape):
    return np.zeros(shape)


def dot(a, b):
  return np.dot(a, b)
```

The Ray API

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)
```

The Ray API

Tasks

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
result = ray.get(id3)
```

The Ray API

Tasks

```
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
result = ray.get(id3)
```

```
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

The Ray API

Tasks

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
result = ray.get(id3)
```

Actors

```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value
```

The Ray API

Tasks

```python
@ray.remote
def zeros(shape):
    return np.zeros(shape)

@ray.remote
def dot(a, b):
  return np.dot(a, b)

id1 = zeros.remote([5, 5])
id2 = zeros.remote([5, 5])
id3 = dot.remote(id1, id2)
result = ray.get(id3)
```

Actors

```python
@ray.remote(num_gpus=1)
class Counter(object):
    def __init__(self):
        self.value = 0
    def inc(self):
        self.value += 1
        return self.value

c = Counter.remote()
id4 = c.inc.remote()
id5 = c.inc.remote()
result = ray.get([id4, id5])
```

Single-Threaded Hyperparameter Search

```
from collections import defaultdict
import numpy as np

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data


def train_cnn_and_compute_accuracy(params, steps, train_images,
train_labels,
                                   validation_images, validation_labels,
                                   weights=None):
    # Extract the hyperparameters from the params dictionary.
    learning_rate = params["learning_rate"]
    batch_size = params["batch_size"]
    keep = 1 - params["dropout"]
    stddev = params["stddev"]
    # Create the network and related variables.
    with tf.Graph().as_default():
        # Create the input placeholders for the network.
        x = tf.placeholder(tf.float32, shape=[None, 784])
        y = tf.placeholder(tf.float32, shape=[None, 10])
        keep_prob = tf.placeholder(tf.float32)
        # Create the network.
        train_step, accuracy, loss = cnn_setup(x, y, keep_prob, learning_rate,
                                               stddev)
        # Do the training and evaluation.
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            if weights is not None:
                variables.set_weights(weights)
            # Do some steps of training.
            for i in range(1, steps + 1):
                image_batch = get_batch(train_images, i, batch_size)
                label_batch = get_batch(train_labels, i, batch_size)
                sess.run(train_step, feed_dict={x: image_batch, y: label_batch,
                                                keep_prob: keep})
            totalacc = accuracy.eval(feed_dict={x: validation_images,
                                                y: validation_labels,
                                                keep_prob: 1.0})
            new_weights = variables.get_weights()
    return float(totalacc), new_weights
```

```
mnist = input_data.read_data_sets("MNIST_data", one_hot=True)
train_images = mnist.train.images
train_labels = mnist.train.labels
validation_images = mnist.validation.images
validation_labels = mnist.validation.labels
accuracies_by_num_steps = defaultdict(lambda: [])

# Define a method to determine if an experiment looks promising or not.
def is_promising(experiment_info):
    accuracies = experiment_info["accuracies"]
    total_num_steps = experiment_info["total_num_steps"]
    comparable_accuracies = accuracies_by_num_steps[total_num_steps]
    if len(comparable_accuracies) == 0:
        if len(accuracies) == 1:
            return True
        else:
            return (np.mean(accuracies[:len(accuracies) // 2]) <
                    np.mean(accuracies[len(accuracies) // 2:]))
    return np.mean(accuracy > np.array(comparable_accuracies)) > 0.5

experiment_info = {}
remaining_vals = []

# Keep track of the best hyperparameters and the best accuracy.
best_hyperparameters = None
best_accuracy = 0

# A function for generating random hyperparameters.
def generate_hyperparameters():
    return {"learning_rate": 10 ** np.random.uniform(-5, 5),
            "batch_size": np.random.randint(1, 100),
            "dropout": np.random.uniform(0, 1),
            "stddev": 10 ** np.random.uniform(-5, 5)}

for _ in range(5):
    hyperparameters = generate_hyperparameters()
    experiment_val = train_cnn_and_compute_accuracy (
        hyperparameters, steps, train_images, train_labels,
        validation_images, validation_labels)
    experiment_info[experiment_val] = {"hyperparameters": hyperparameters,
                                       "total_num_steps": steps,
                                       "accuracies": []}
    remaining_vals.append(experiment_val)
```

```
for _ in range(10):
    ready_vals, remaining_vals = remaining_vals[0], remaining_vals[1:]
    experiment_val = ready_vals[0]
    accuracy, weights = experiment_val)
    previous_info = experiment_info[experiment_val]
    previous_info["accuracies"].append(accuracy)

    if accuracy > best_accuracy:
        best_hyperparameters = previous_info["hyperparameters"]
        best_accuracy = accuracy

    if is_promising(previous_info):
        # If the experiment does not look promising, start a new
        # experiment.
        print("Ending the experiment with hyperparameters {}."
              .format(previous_info["hyperparameters"]))

        new_hyperparameters = previous_info["hyperparameters"]
        new_info = {"hyperparameters": new_hyperparameters,
                    "total_num_steps": (previous_info["total_num_steps"] +
                                        steps),
                    "accuracies": previous_info["accuracies"][:]}
        starting_weights = weights
    else:
        new_hyperparameters = generate_hyperparameters()
        new_info = {"hyperparameters": new_hyperparameters,
                    "total_num_steps": steps, "accuracies": []}
        starting_weights = None

    new_experiment_val = train_cnn_and_compute_accuracy.(
        new_hyperparameters, steps, train_images, train_labels,
        validation_images, validation_labels, weights=starting_weights)
    experiment_info[new_experiment_val] = new_info
    remaining_vals.append(new_experiment_val)


accuracies_by_num_steps[previous_info["total_num_steps"]].append(ac
curacy)
```

Distributed With Ray

```
from collections import defaultdict
import numpy as np
import ray
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

@ray.remote
def train_cnn_and_compute_accuracy(params, steps, train_images,
train_labels,
                                 validation_images, validation_labels,
                                 weights=None):
    # Extract the hyperparameters from the params dictionary.
    learning_rate = params["learning_rate"]
    batch_size = params["batch_size"]
    keep = 1 - params["dropout"]
    stddev = params["stddev"]
    # Create the network and related variables.
    with tf.Graph().as_default():
        # Create the input placeholders for the network.
        x = tf.placeholder(tf.float32, shape=[None, 784])
        y = tf.placeholder(tf.float32, shape=[None, 10])
        keep_prob = tf.placeholder(tf.float32)
        # Create the network.
        train_step, accuracy, loss = cnn_setup(x, y, keep_prob, learning_rate,
                                               stddev)
        # Do the training and evaluation.
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            if weights is not None:
                variables.set_weights(weights)
            # Do some steps of training.
            for i in range(1, steps + 1):
                image_batch = get_batch(train_images, i, batch_size)
                label_batch = get_batch(train_labels, i, batch_size)
                sess.run(train_step, feed_dict={x: image_batch, y: label_batch,
                                                keep_prob: keep})
            totalacc = accuracy.eval(feed_dict={x: validation_images,
                                                y: validation_labels,
                                                keep_prob: 1.0})
            new_weights = variables.get_weights()
    return float(totalacc), new_weights

ray.init()
```

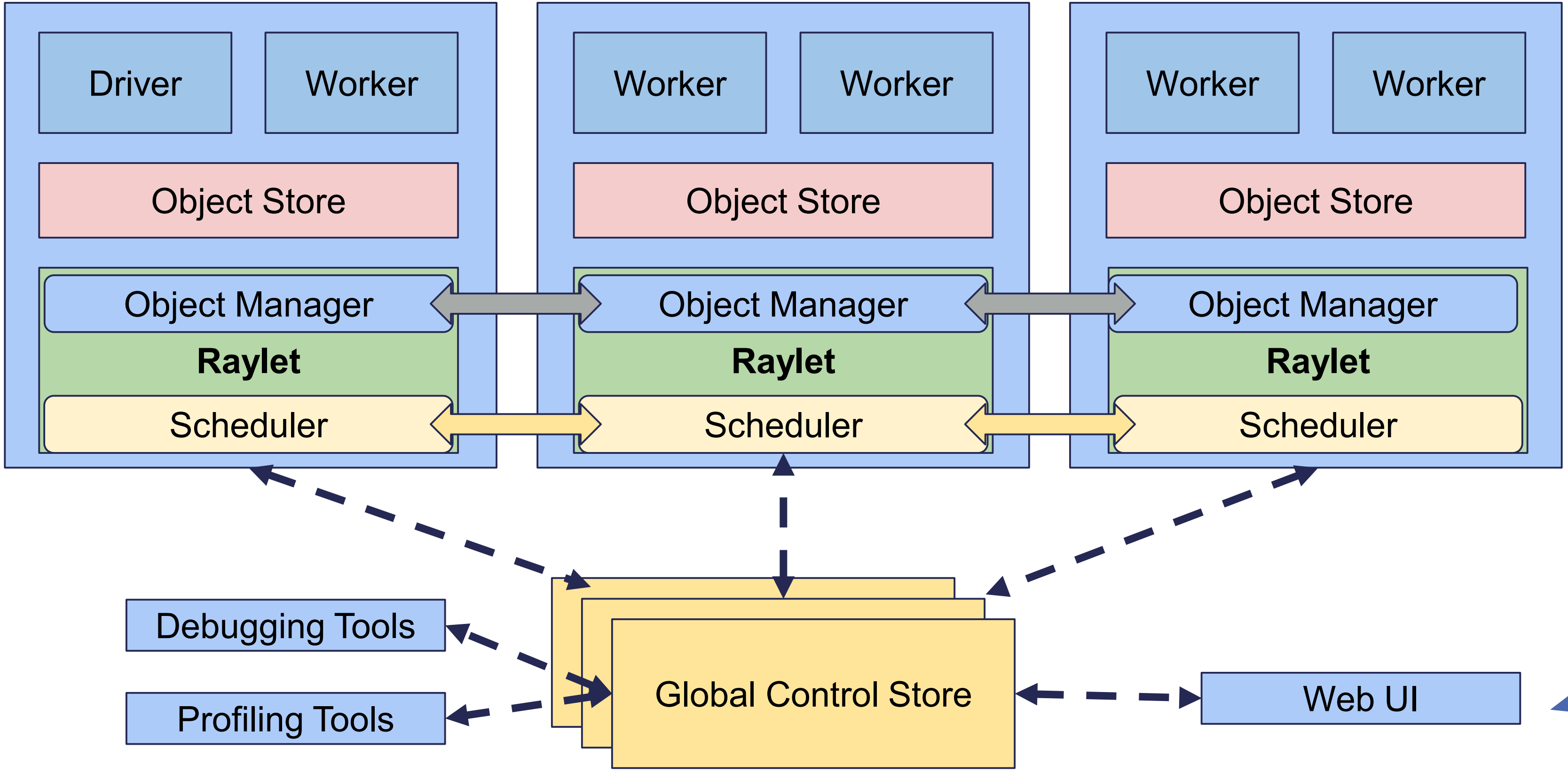
```
mnist = input_data.read_data_sets("MNIST_data", one_hot=True)
train_images = mnist.train.images
train_labels = mnist.train.labels
validation_images = mnist.validation.images
validation_labels = mnist.validation.labels
accuracies_by_num_steps = defaultdict(lambda: [])

# Define a method to determine if an experiment looks promising or not.
def is_promising(experiment_info):
    accuracies = experiment_info["accuracies"]
    total_num_steps = experiment_info["total_num_steps"]
    comparable_accuracies = accuracies_by_num_steps[total_num_steps]
    if len(comparable_accuracies) == 0:
        if len(accuracies) == 1:
            return True
        else:
            return (np.mean(accuracies[:len(accuracies) // 2]) <
                    np.mean(accuracies[len(accuracies) // 2:]))
    return np.mean(accuracy > np.array(comparable_accuracies)) > 0.5

experiment_info = {}
remaining_vals = []

# Keep track of the best hyperparameters and the best accuracy.
best_hyperparameters = None
best_accuracy = 0

# A function for generating random hyperparameters.
def generate_hyperparameters():
    return {"learning_rate": 10 ** np.random.uniform(-5, 5),
            "batch_size": np.random.randint(1, 100),
            "dropout": np.random.uniform(0, 1),
            "stddev": 10 ** np.random.uniform(-5, 5)}

for _ in range(5):
    hyperparameters = generate_hyperparameters()
    experiment_val = train_cnn_and_compute_accuracy.remote(
        hyperparameters, steps, train_images, train_labels,
        validation_images, validation_labels)
    experiment_info[experiment_val] = {"hyperparameters": hyperparameters,
                                       "total_num_steps": steps,
                                       "accuracies": []}
    remaining_vals.append(experiment_val)
```

```
for _ in range(10):
    ready_vals, remaining_vals = ray.wait(remaining_vals)
    experiment_val = ready_vals[0]
    accuracy, weights = ray.get(experiment_val)
    previous_info = experiment_info[experiment_val]
    previous_info["accuracies"].append(accuracy)

    if accuracy > best_accuracy:
        best_hyperparameters = previous_info["hyperparameters"]
        best_accuracy = accuracy

    if is_promising(previous_info):
        # If the experiment does not look promising, start a new
        # experiment.
        print("Ending the experiment with hyperparameters {}."
              .format(previous_info["hyperparameters"]))

        new_hyperparameters = previous_info["hyperparameters"]
        new_info = {"hyperparameters": new_hyperparameters,
                    "total_num_steps": (previous_info["total_num_steps"] +
                                        steps),
                    "accuracies": previous_info["accuracies"][:]}
        starting_weights = weights
    else:
        new_hyperparameters = generate_hyperparameters()
        new_info = {"hyperparameters": new_hyperparameters,
                    "total_num_steps": steps, "accuracies": []}
        starting_weights = None

    new_experiment_val = train_cnn_and_compute_accuracy.remote(
        new_hyperparameters, steps, train_images, train_labels,
        validation_images, validation_labels, weights=starting_weights)
    experiment_info[new_experiment_val] = new_info
    remaining_vals.append(new_experiment_val)


accuracies_by_num_steps[previous_info["total_num_steps"]].append(ac
curacy)
```

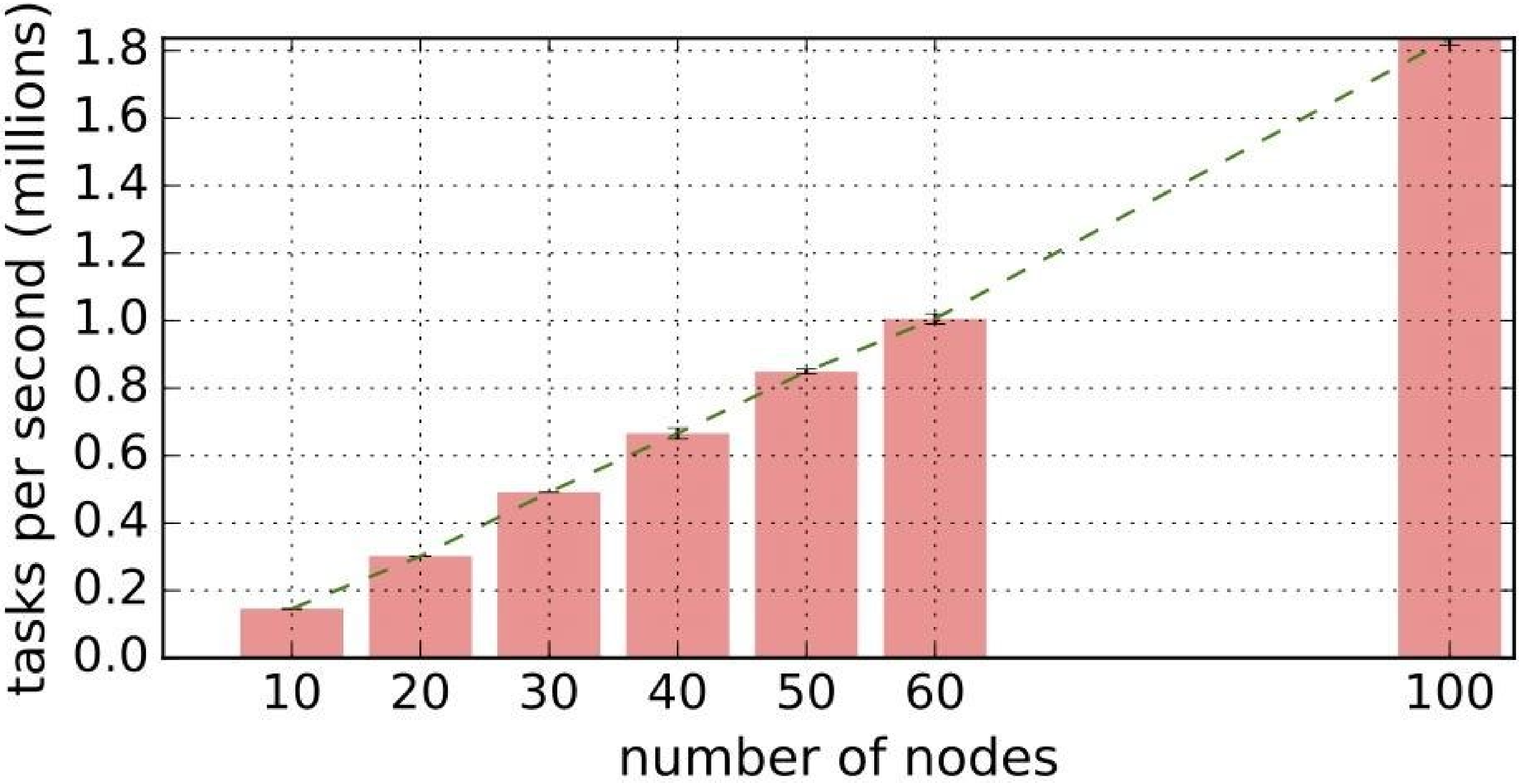
Broad Range of Scalable Algorithms

- High-throughput architectures
  - Distributed Prioritized Experience Replay (Ape-X)
  - Importance Weighted Actor-Learner Architecture (IMPALA)
  - Asynchronous Proximal Policy Optimization (APPO)
- Gradient-based
  - Soft Actor-Critic (SAC)
  - Advantage Actor-Critic (A2C, A3C)
  - Deep Deterministic Policy Gradients (DDPG, TD3)
  - Deep Q Networks (DQN, Rainbow, Parametric DQN)
  - Policy Gradients
  - Proximal Policy Optimization (PPO)
- Derivative-free
  - Augmented Random Search (ARS)
  - Evolution Strategies
- Multi-agent specific
  - QMIX Monotonic Value Factorisation (QMIX, VDN, IQN)
- Offline
  - Advantage Re-Weighted Imitation Learning (MARWIL)

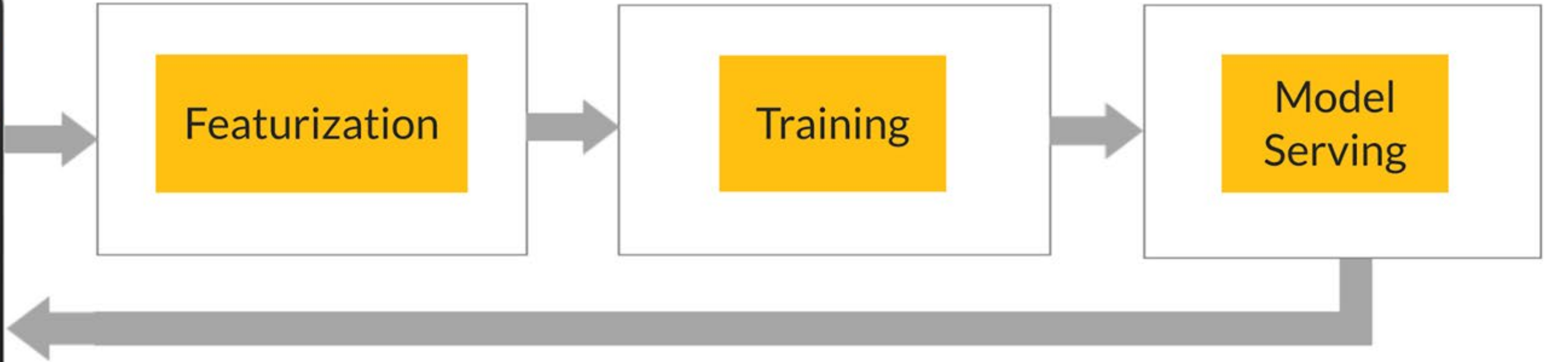
Ray Architecture



Performance



Example: Online Learning



Model updated every **1 day**

+ 5% CTR

Model updated every **1 hour** (using state-of-the-art solution)

?

Example: Online Learning

Model updated every **1 day**

+ 5% CTR

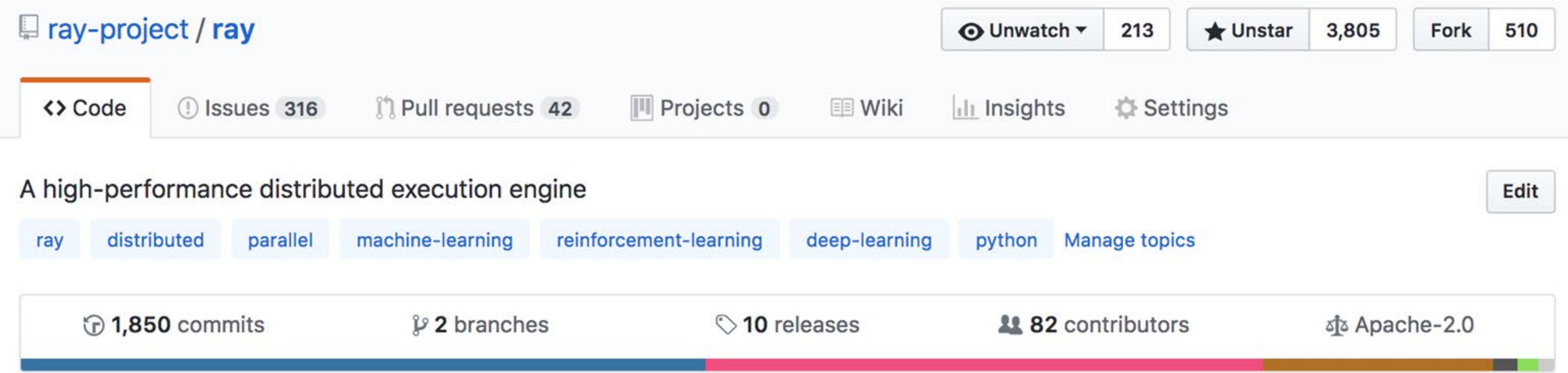
Model updated every **1 hour** (using state-of-the-art solution)

+ 1% CTR

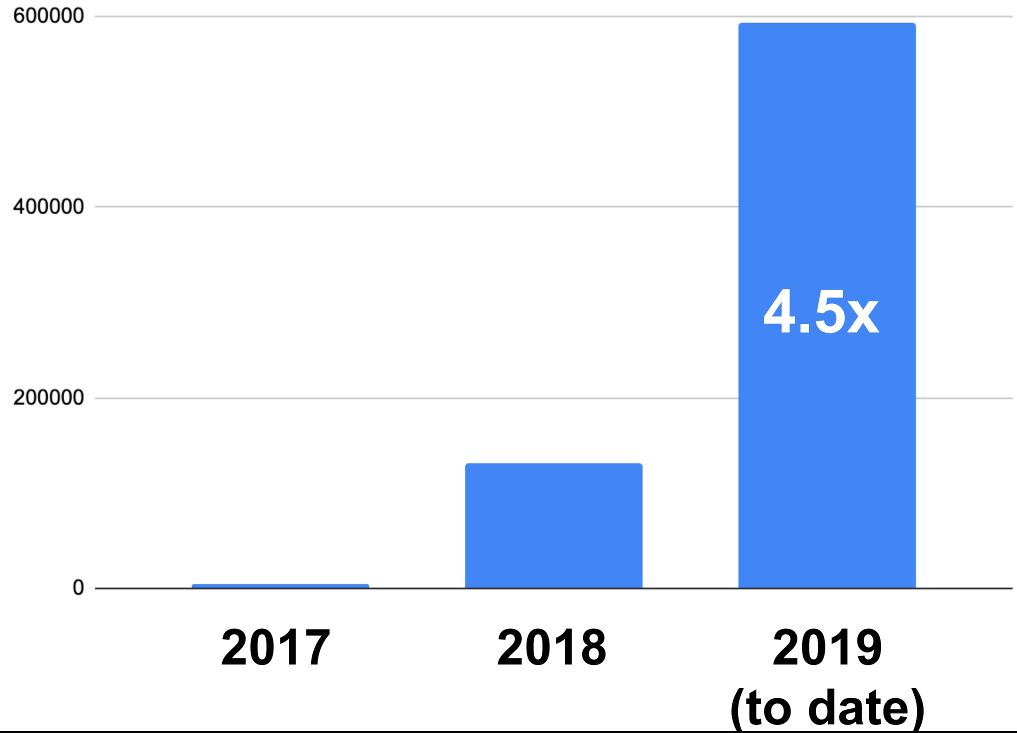
Model updated every **5 min** using Ray

Community

Ray is Open Source!



Ray Downloads



A Growing Number of Industry Use Cases

蚂蚁金服
ANT FINANCIAL

intel

OpenAI

ERICSSON

Berkeley
UNIVERSITY OF CALIFORNIA

Alibaba.com

BONSAI AI

PRIMER

Morgan Stanley

Conclusions

- Ray is a system for distributed Python
  - includes libraries targeting AI applications
- Open source at **github.com/ray-project/ray**
- Install with **pip install ray**

- *Reference*: Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Jordan, M. I., & Stoica, I. (2018). A distributed framework for emerging AI applications. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI).

riselab UC Berkeley

Parting Comments

- The current era of machine learning has focused on pattern recognition
  - platforms such as TensorFlow and PyTorch have arisen to help turn pattern recognition into a commodity
- The decision-making side of machine learning will be a focus in the future
  - individual high-stake decisions
  - explanations for decisions, and dialog about decisions
  - sequences of decisions
  - multiple simultaneous decisions
  - decisions in the context of multiple decision-makers
  - market mechanisms

The Learning Continues…

TechTalk Discourse: https://on.acm.org

TechTalk Inquiries: learning@acm.org

TechTalk Archives: https://learning.acm.org/techtalks

Learning Center: https://learning.acm.org

Professional Ethics: https://ethics.acm.org

Queue Magazine: https://queue.acm.org