

A Guided Tour of Ray Core

Paco Nathan @pacoid



derwen.ai



Syllabus

Intended Audience

- Python developers who want to learn how to parallelize their application code

Prerequisites

- Some prior experience developing code in Python
- Basic understanding of distributed systems



Key Takeaways

- What are the Ray core features and how to use them?
- In which contexts are the different approaches indicated?
- Profiling methods, to decide when to make trade-offs (compute cost, memory, I/O, etc.) ?

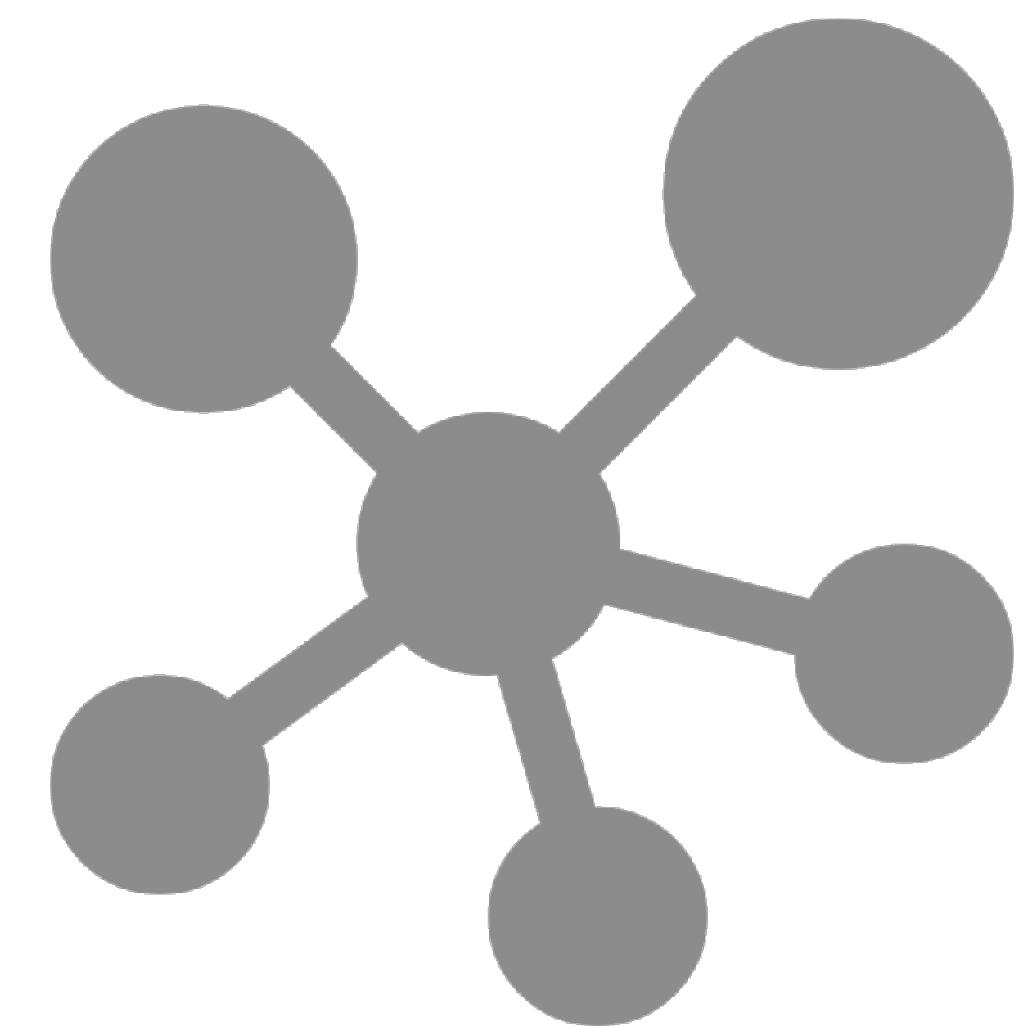
Context

Too Big To Fit, scale-up vs. scale-out

When an application becomes too big or too complex to run efficiently on a single server, there are some options:

- migrate to a larger server, and buy bigger licenses – that's called *vertical scale-up*
- distribute data+compute across multiple servers – that's called *horizontal scale-out*

The histories of **MPI**, **Hadoop**, **Spark**, **Dask**, etc., represent generations of scale-out, which imply **trade-offs** both for the risks (losing partitions, split-brain, etc.) as well as the inherent overhead costs



Too Big To Fit, scale-up vs. scale-out

When an application becomes too big or too complex to run efficiently on a single server, there are some options:

- migrate to a bigger server
that's called **scale-up**
- distribute data across multiple servers
that's called **scale-out**

Cloud Computing has arguably been an embodiment of distributed systems practices for the past 15 years, whether for scale-up or scale-out

The histories of ~~HDFS~~, **Hadoop**, **Spark**, **Dask**, etc., represent generations of scale-out, which imply **trade-offs** both for the risks (losing partitions, split-brain, etc.) as well as the inherent overhead costs



Formal definition of Cloud Computing

Professors **Ion Stoica** and **David Patterson** led EECS grad students to define cloud computing **formally** in 2009

“More than 17,000 citations to this paper...”

2019 follow-up:

“We now predict ... that *Serverless Computing* will grow to dominate the future of cloud computing in the next decade”

The screenshot shows the Ariselab website header with the logo 'ariselab UC Berkeley' and navigation links for HOME, PEOPLE, PROJECTS, PUBLICATIONS, SPONSORS, DARE, ACADEMICS, NEWS, EVENTS, RISE CAMP, BLOGS, and JENKINS. The main content area features the title 'Cloud Programming Simplified: A Berkeley View on Serverless Computing' by David Patterson and Ion Stoica, published on FEBRUARY 10, 2019. Below the title is a short bio and a paragraph explaining the impact of the paper.

Cloud Programming Simplified: A Berkeley View on Serverless Computing

ION STOICA / FEBRUARY 10, 2019 /

David Patterson and Ion Stoica

The publication of “Above the Clouds: A Berkeley View of Cloud Computing” on February 10, 2009 cleared up the considerable confusion about the new notion of “Cloud Computing.” The paper defined what Cloud Computing was, where it came from, why some were excited by it, what were its technical advantages, and what were the obstacles and research opportunities for it to become even more popular. More than 17,000 citations to this paper and an [abridged version](#) in *CACM*—with more than 1000 in the past year—document that it continues to shape the discussions and the evolution of Cloud Computing.

Evolution of cloud patterns

Initially, cloud services were simplified to make them more recognizable for IT staff accustomed to VMware

- Patterson, et al., developed industry research methodology and eventually also a **pattern language** to describe distributed systems
- **AMPLab** foresaw how cloud use cases would progress in industry over the next decade, which greatly informed **Apache Spark**, etc.

Above the Clouds: A Berkeley View of Cloud Computing



*Michael Armbrust
Armando Fox
Rean Griffith
Anthony D. Joseph
Randy H. Katz
Andrew Konwinski
Gunho Lee
David A. Patterson
Ariel Rabkin
Ion Stoica
Matei Zaharia*

Electrical Engineering and Computer Sciences
University of California at Berkeley

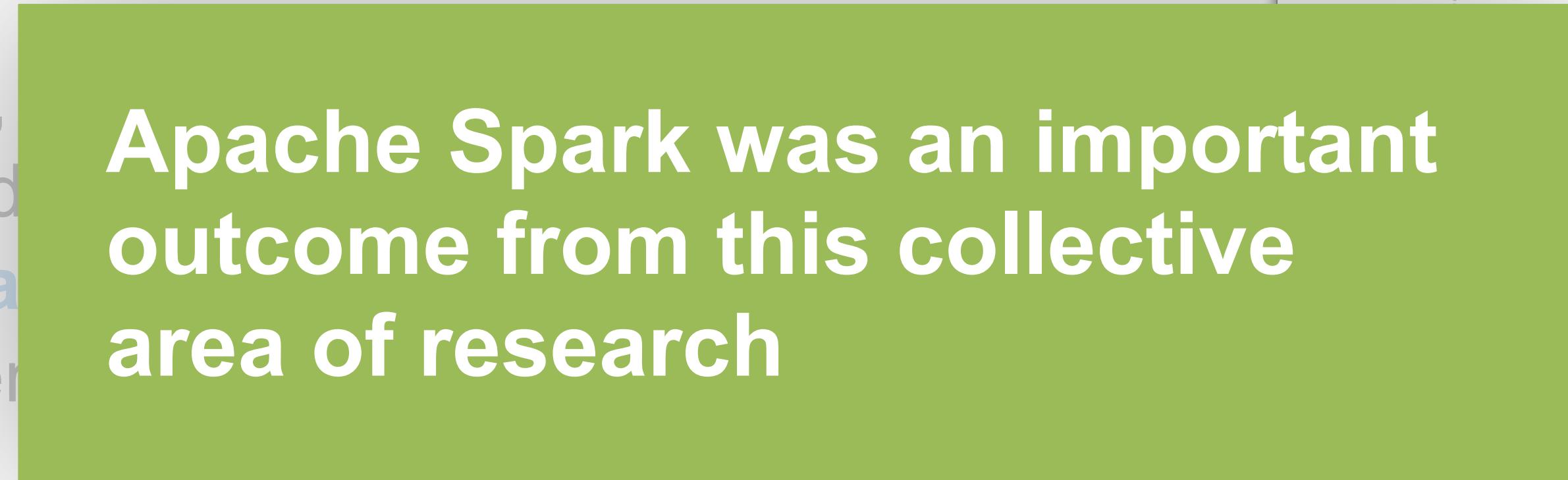
Technical Report No. UCB/EECS-2009-28
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>

February 10, 2009

Evolution of cloud patterns

Initially, cloud services were simplified to make them more recognizable for IT staff accustomed to VMware

- Patterson, et al., research method also a **pattern language** for distributed systems
- **AMPLab** foresaw how cloud use cases would progress in industry over the next decade, which greatly informed **Apache Spark**, etc.



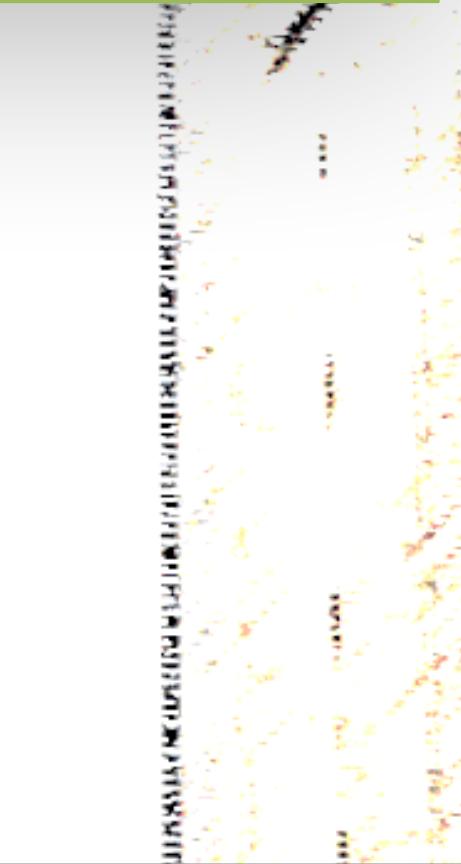
Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust
Armando Fox
Rean Griffith
Anthony D. Joseph
Randy H. Katz
Andrew Konwinski
Gunho Lee
David A. Patterson
Ariel Rabkin
Ion Stoica
Matei Zaharia

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2009-28
<http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>

February 10, 2009



2009

Evolution of cloud patterns

Eric Jonas noted >50% of **RISElab** grad students had never used Spark; also, how cloud was evolving in its second decade:

- “Decoupling of computation and storage; they scale separately and are priced independently”
- “The abstraction of executing a piece of code instead of allocating resources on which to execute that code”
- “Paying for the code execution instead of paying for resources you have allocated toward executing the code”

Cloud Programming Simplified: A Berkeley View on
Serverless Computing



*Eric Jonas
Johann Schleier-Smith
Vikram Sreekanti
Chia-Che Tsai
Anurag Khandelwal
Qifan Pu
Vaishaal Shankar
Joao Menezes Carreira
Karl Krauth
Neeraja Yadwadkar
Joseph Gonzalez
Raluca Ada Popa
Ion Stoica
David A. Patterson*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-3
<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html>

February 10, 2019

Evolution of cloud patterns

Eric Jonas noted >50% of RISElab grad students had never used Spark, plus how cloud was evolving in its second decade:

- “Decoupling of computation and storage so they scale separately and independently”
- “The abstraction of moving code instead of allocating resources on which to execute that code”
- “Paying for the code execution instead of paying for resources you have allocated toward executing the code”

Ray is an important outcome from this collective area of research

Cloud Programming Simplified: A Berkeley View on Serverless Computing

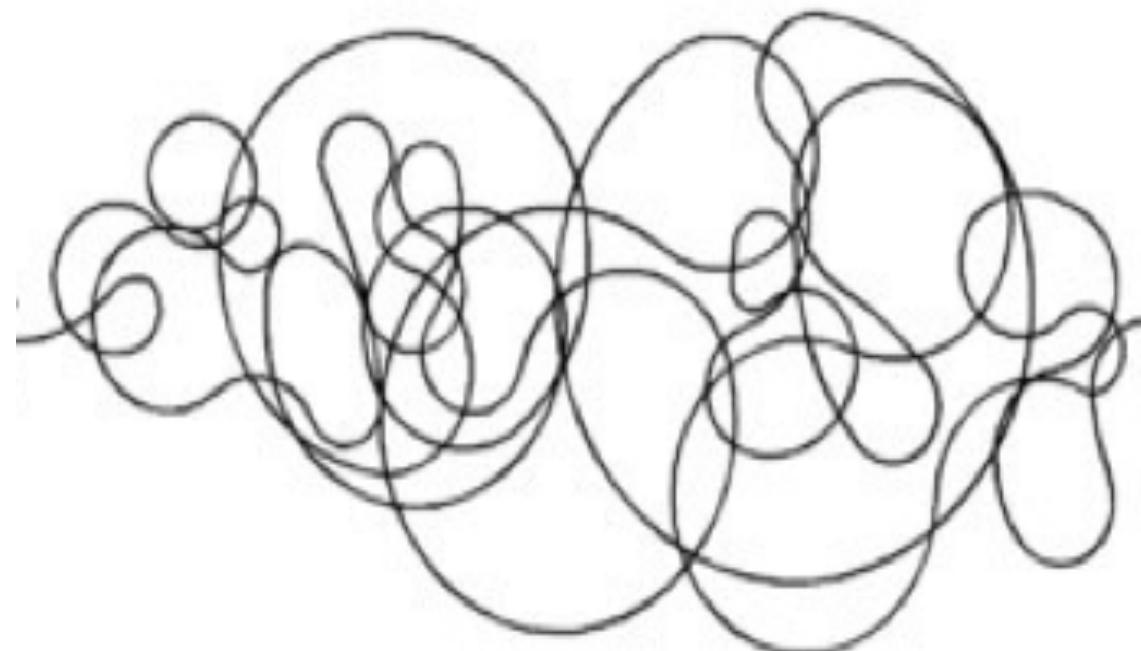
Eric Jonas
Johann Schleier-Smith
Vikram Sreekanti
Chia-Che Tsai
Anurag Khandelwal
Qifan Pu
Vaishaal Shankar
Joao Menezes Carreira
Karl Krauth
Neeraja Yadwadkar
Joseph Gonzalez
Raluca Ada Popa
Ion Stoica
David A. Patterson

Electrical Engineering and Computer Sciences
University of California at Berkeley

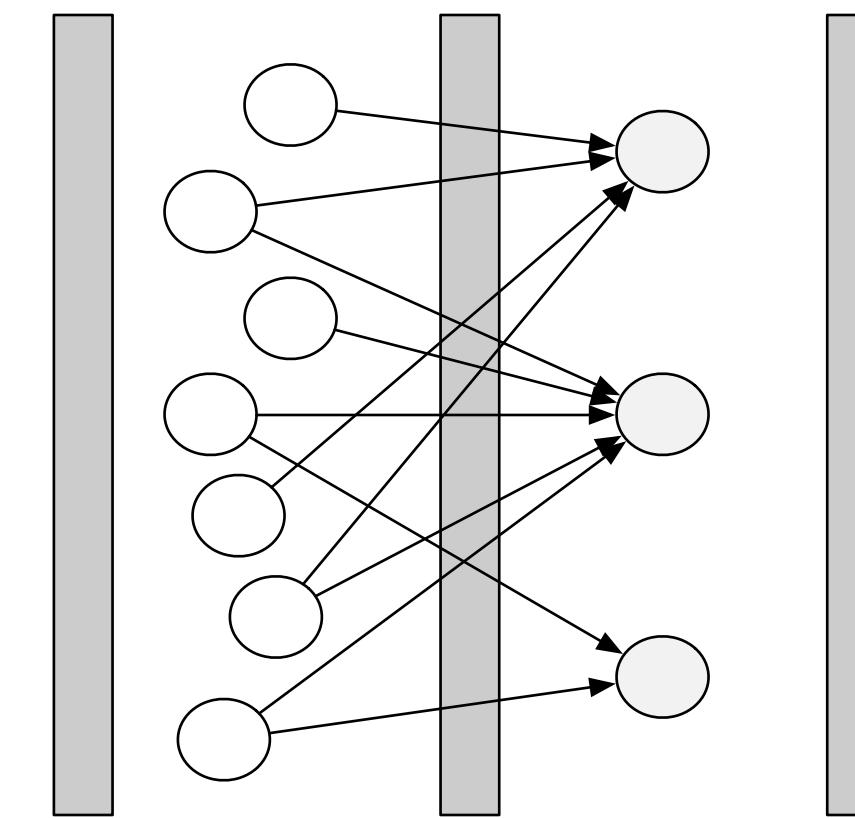
Technical Report No. UCB/EECS-2019-3
<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.html>

February 10, 2019

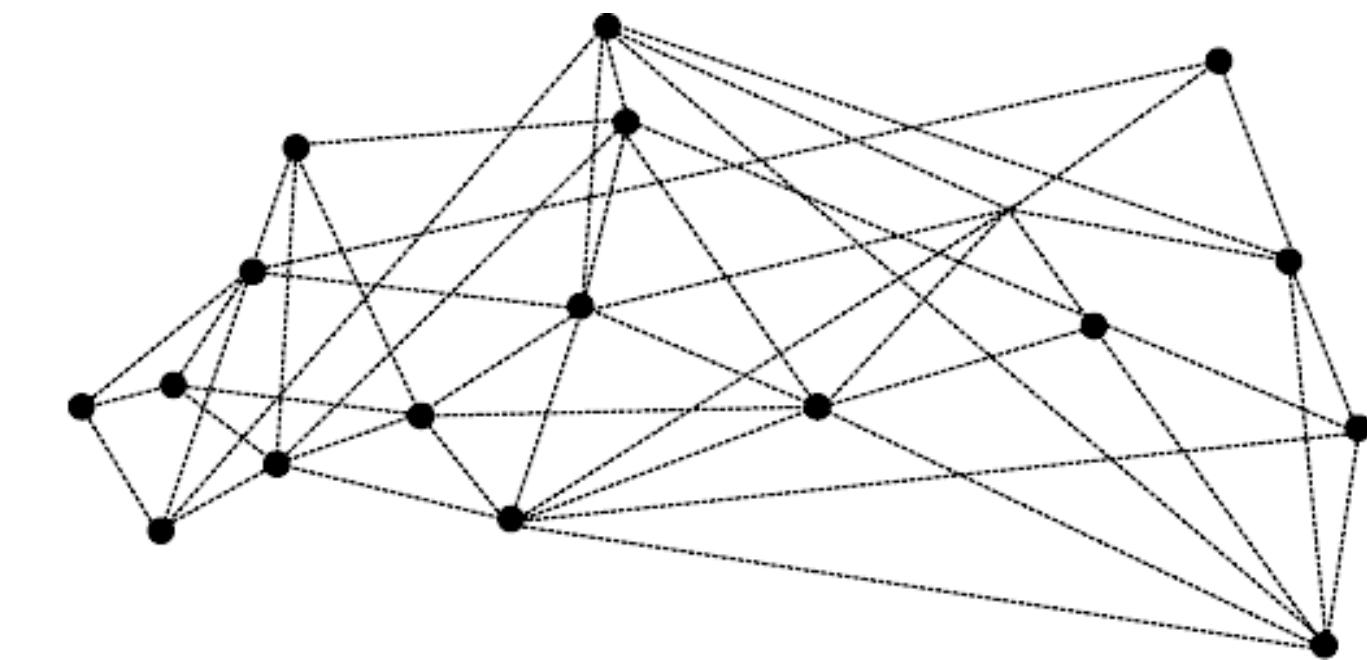
Cluster usage, by generation



1990s



2000s



current



Cluster usage, taming latency

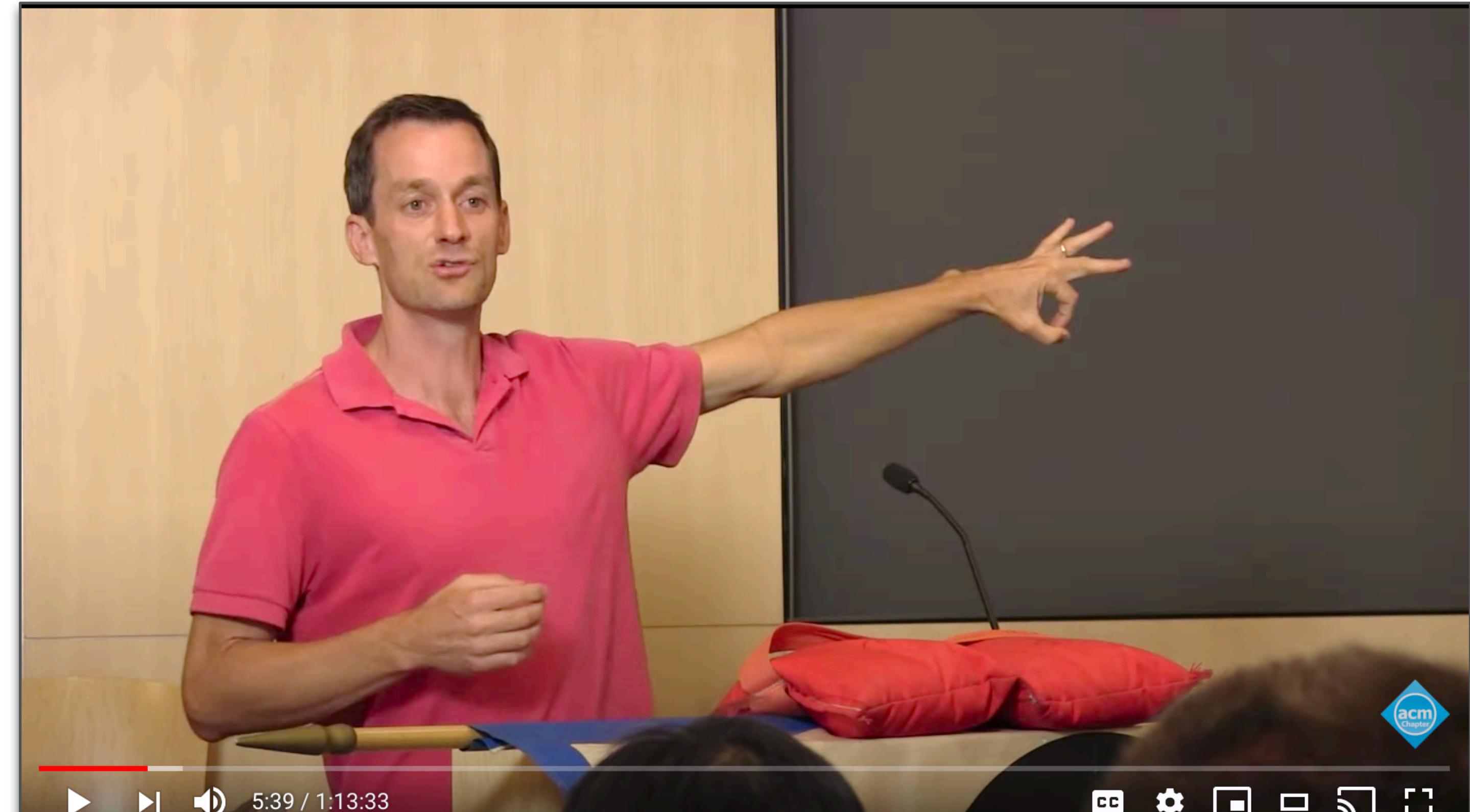
“Taming Latency”
Jeff Dean (2013)

youtu.be/S9twUcX1Zp0

Describing factors within their latest datacenters that drove the design of **TensorFlow**

Optimized to train networks
(graphs/tensors)

More recently:
Even Oldridge / Merlin



Taming Latency Variability and Scaling Deep Learning, by Jeff Dean, Google, 20131016

Cluster usage, taming latency

“Taming Latency”
Jeff Dean (2013)

youtu.be/S9twUcX1Zp0

Describing factors
latest datacenters
the design of Ten

Optimized to train networks
(graphs/tensors)

More recently:
Even Oldridge / Merlin

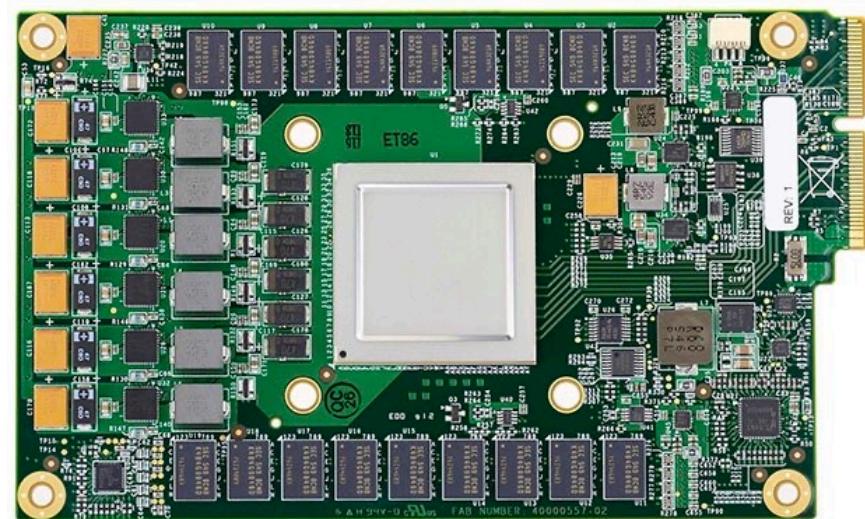
**TensorFlow, PyTorch, Horovod,
etc., come from this – as well as
informing Ray’s implementation**



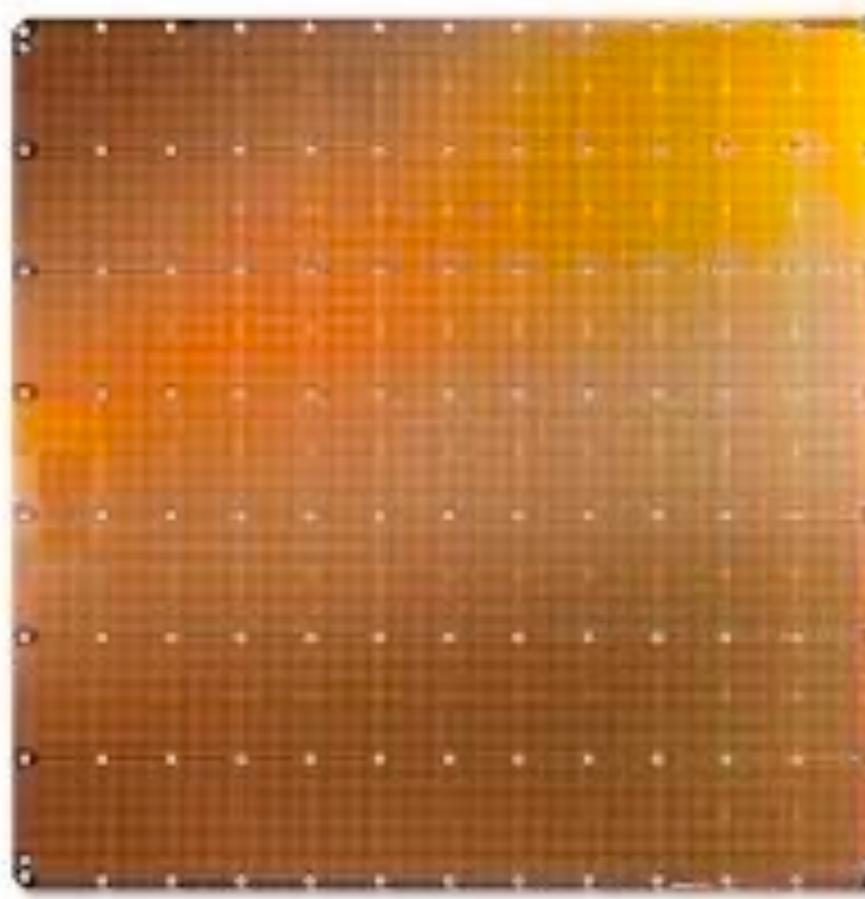
Taming Latency Variability and Scaling Deep Learning, by Jeff Dean, Google, 20131016

Hardware > Software > Process

Circa 2005: commodity hardware served Big Data needs: **log file analysis** using **task-parallel**



Circa 2009: data science workloads introduced stateful **actor pattern** through Spark, Akka, etc.



Circa 2013: deep learning placed more demands on s/w + h/w, **differentiating gradients within the context of networked data**



Circa 2017: embedded language models, **Software 2.0**, Intel falters on 10 nm features

Hardware > Software > Process

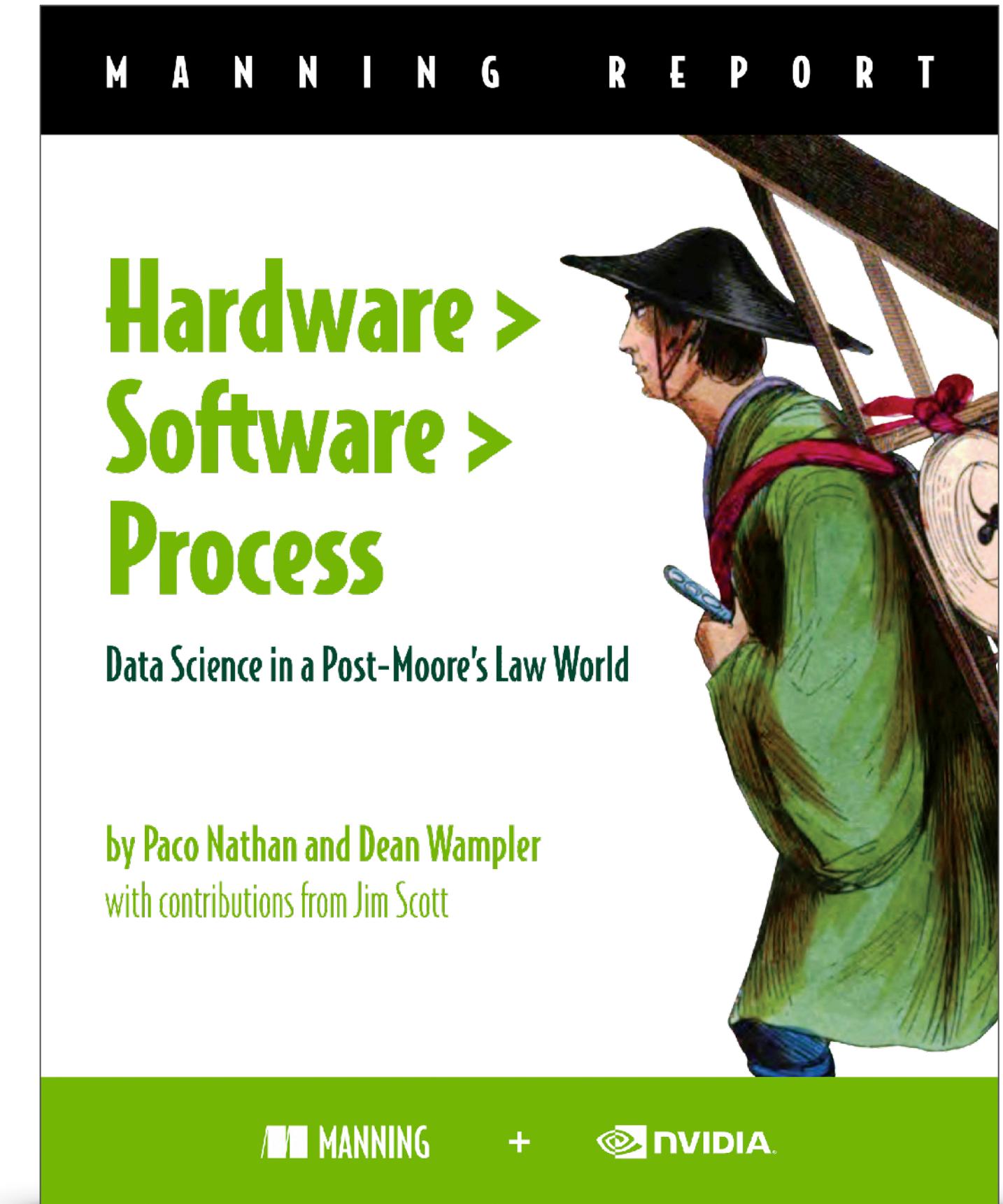
*Hardware > Software > Process:
Data Science in a Post-Moore's Law World*

Paco Nathan, Dean Wampler
Manning (2021-05-25)

free download:

nvidia.com/en-us/ai-data-science/resources/hardware-software-process-book/

- Design patterns in Python, so that hardware can optimize data workloads
- Key abstractions for distributed processing in ML workflows
- Best practices for profiling in depth, CI, etc.



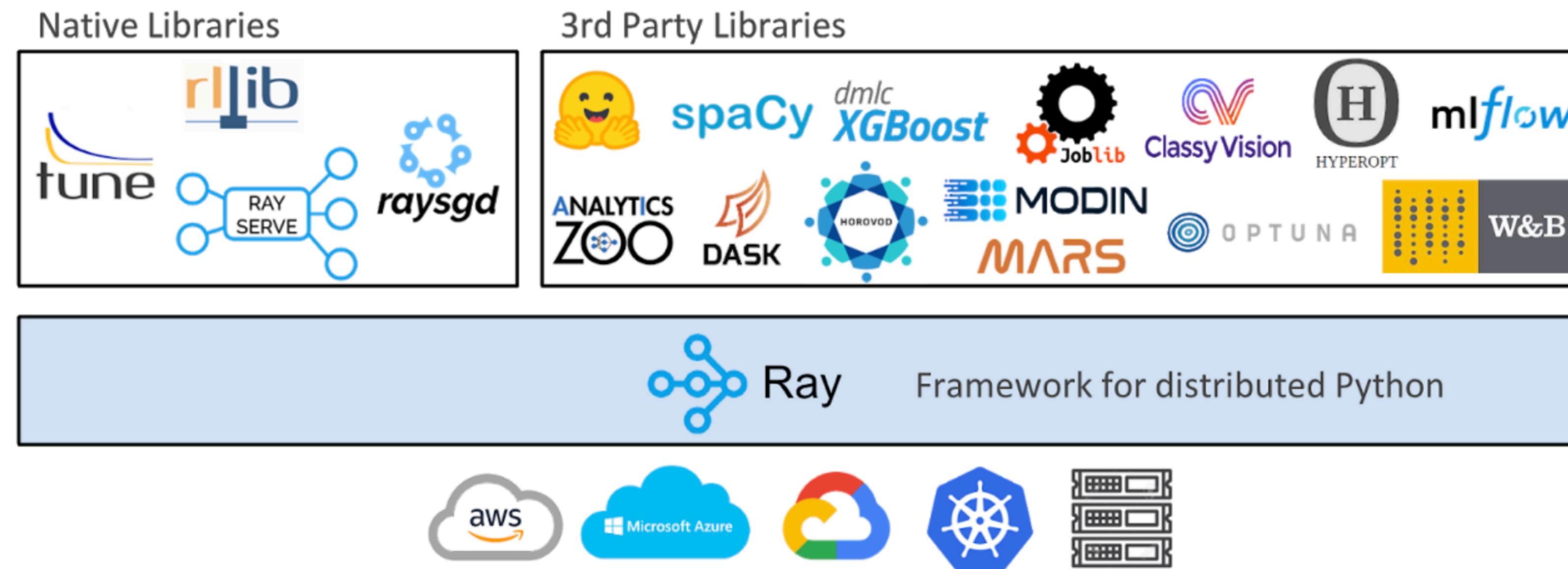
Ray core

For an excellent overview of [Ray](#), see:

**“Modern Parallel and Distributed Python:
A Quick Tutorial on Ray”**

Robert Nishihara

Towards Data Science (2019-02-10)



Infinite Laptop

While working, is your **attention** focused on your laptop, or focused somewhere in the cloud?

Why not **both**?

- Ben Lorica explores the emerging architectural pattern of an **Infinite Laptop**
- Deconstructing what RISElab means by “serverless will dominate cloud”
- One application, with sections running on different resources



Gradient Flow

Data, Machine Learning, and AI

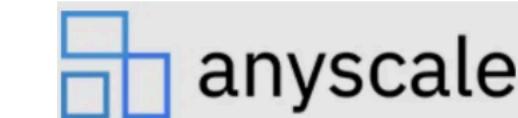
Newsletter Podcast Blog Reports Video Events



Towards an infinite laptop

The new Anyscale platform offers the ease of development on a laptop combined with the power of the cloud.

During a series of short keynotes at the [Ray Summit](#) this morning, Anyscale¹, the company formed by the creators of Ray, publicly shared their initial product offering.



Dubbed the “infinite laptop”, Anyscale’s platform allows developers to treat their laptop as an “infinite cluster”.

Developers can use their preferred development tools (e.g., IDE, notebook, text editor, etc.) on Anyscale’s platform, and seamlessly burst into a cloud platform when needed.

Pattern

Ray core

A kind of **pattern language** for distributed systems,
as a library in Python, Java, C++ (upcoming):

- **task-parallel** – stateless, data independence
- **remote objects** – key/value store
- **actor pattern** – messages among classes,
managing state
- **parallel iterators** – lazy, infinite sequences
- **multiprocessing.Pool** – drop-in replacement
- **joblib** – e.g., *scikit-learn* back-end
- Dask, Modin, Mars, etc.



Mix and match as needed, without tight coupling
to framework

Examples in notebooks...

The screenshot shows a GitHub repository page for 'README.md'. The title 'A Guided Tour of Ray Core' is displayed prominently. Below the title, there is a brief description: 'An introductory tutorial about leveraging Ray core features for distributed patterns.' A note follows: 'Note: these examples have been tested using Python 3.7+ on:' followed by a bulleted list: '• Ubuntu 18.04 LTS' and '• macOS 10.13'. A 'Getting Started' section is present with instructions: 'To get started use `git` to clone this public repository:' and a code block containing the command: `git clone https://github.com/DerwenAI/ray_tutorial.git` and `cd ray_tutorial`.

README.md

A Guided Tour of Ray Core

An introductory tutorial about leveraging Ray core features for distributed patterns.

Note: these examples have been tested using Python 3.7+ on:

- Ubuntu 18.04 LTS
- macOS 10.13

Getting Started

To get started use `git` to clone this public repository:

```
git clone https://github.com/DerwenAI/ray_tutorial.git  
cd ray_tutorial
```

https://github.com/DerwenAI/ray_tutorial

Background

Ray makes use of *closures* and *decorators* in Python:

- **"Closures and Decorators in Python"**
- **PEP 318**
- **asyncio**

See also:

- **patterns.eecs.berkeley.edu**
- **"Ray Design Patterns"** (WIP)

Pattern: task-parallel

Remote Functions:

- a `@ray.remote` decorator on a function
- properties: *data independence, stateless*
- patterns: **Task Parallelism, Task Graph**

reference:

Patterns for Parallel Programming

Timothy G. Mattson, Beverly A. Sanders, Berna L. Massingill
Addison-Wesley (2004)

Pattern: task-parallel

However, by adding the `@ray.remote` decorator, a regular Python function becomes a Ray remote function:

```
@ray.remote  
def my_function():  
    return 1
```

To invoke this remote function, use the `remote` method. This will immediately return an object ref (a *future* in Python) and then create a task that will be executed on a worker process.

```
obj_ref = my_function.remote()  
obj_ref
```

```
ObjectRef(df5a1a828c9685d3fffffffff0100000001000000)
```

The result can be retrieved with `ray.get`

```
ray.get(obj_ref)
```

Pattern: task-parallel

Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_01_remo_func.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_01_remo_func.ipynb)

Others:

[colab.research.google.com/github/ray-project/tutorial/
blob/master/exercises/colab01-03.ipynb](https://colab.research.google.com/github/ray-project/tutorial/blob/master/exercises/colab01-03.ipynb)

[github.com/ansyscale/academy/blob/master/ray-crash-
course/01-Ray-Tasks.ipynb](https://github.com/ansyscale/academy/blob/master/ray-crash-course/01-Ray-Tasks.ipynb)

Pattern: distributed object store

Remote Objects:

- distributed shared-memory object store
- “object lives somewhere on the cluster”
- usage similar to key/value stores, e.g., **Redis**
- object spilling

reference:

https://en.wikipedia.org/wiki/Shared_memory

Pattern: distributed object store

To start, we'll put an object into the Ray object store...

```
y = 1  
obj_ref = ray.put(y)
```

Then get the value of this object reference

```
ray.get(obj_ref)
```

1

You can also access the values of multiple object references in parallel:

```
ray.get([ray.put(i) for i in range(3)])
```

[0, 1, 2]

Pattern: distributed object store

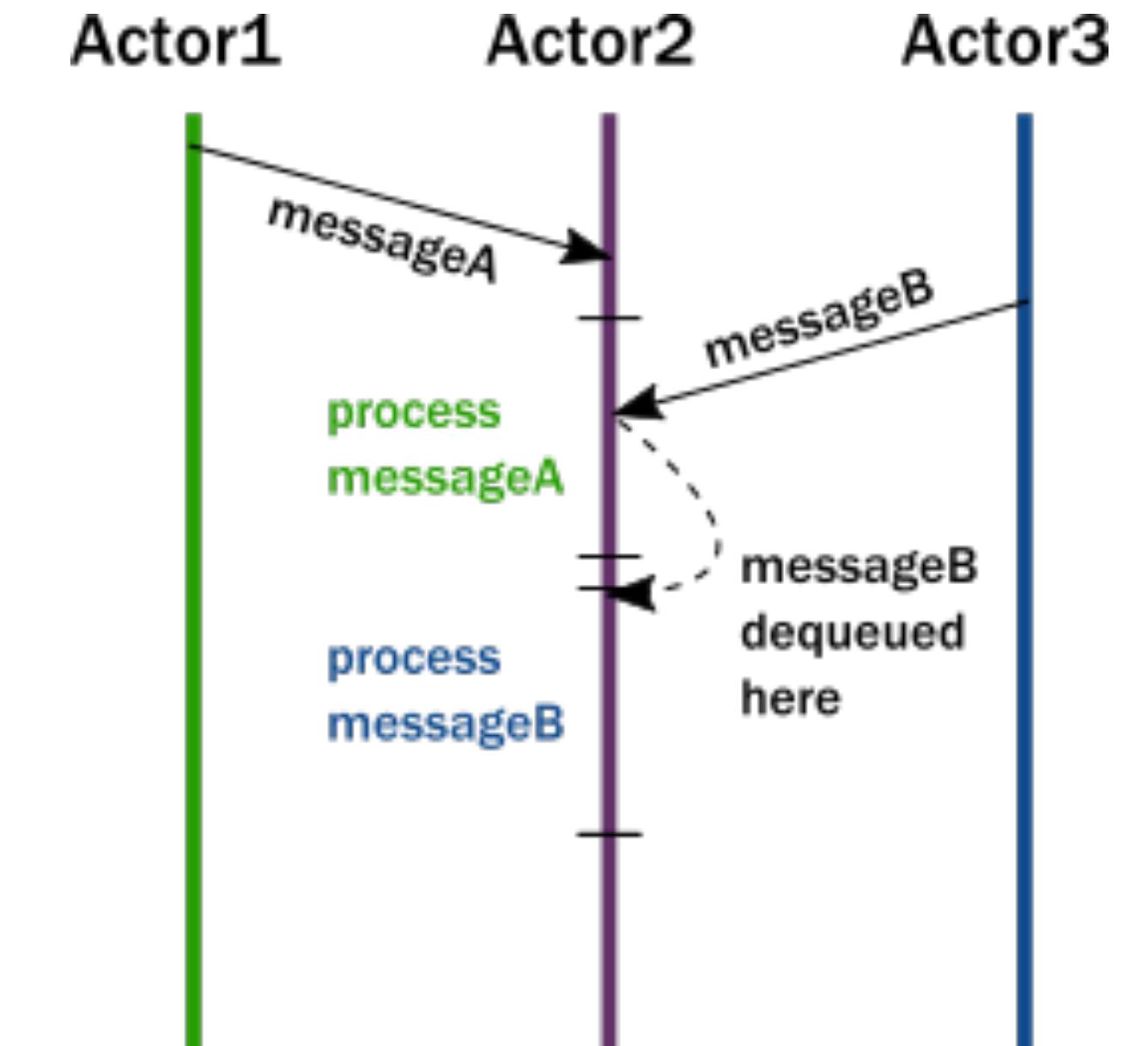
Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_02_remo_objs.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_02_remo_objs.ipynb)

Pattern: actors

Remote Classes:

- a `@ray.remote` decorator on a class
- properties: *stateful, message-passing semantics*
- pattern: **Actors**
- “actor lives somewhere on the cluster”



reference:

“A Universal Modular Actor Formalism for Artificial Intelligence”

Carl Hewitt, Peter Bishop, Richard Steiger

IJCAI (1973)

open: <https://www.ijcai.org/Proceedings/73/Papers/027B.pdf>

Pattern: actors

To start, we'll define a class and use the decorator:

```
@ray.remote
class Counter(object):
    def __init__(self):
        self.value = 0

    def increment(self):
        self.value += 1
        return self.value
```

Now use this class `Counter` to create an actor:

```
counter = Counter.remote()
```

Then call the actor:

```
obj_ref = counter.increment.remote()
ray.get(obj_ref)
```

Pattern: actors

Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_03_remo_meth.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_03_remo_meth.ipynb)

Others:

[colab.research.google.com/github/ray-project/tutorial/
blob/master/exercises/colab04-05.ipynb](https://colab.research.google.com/github/ray-project/tutorial/blob/master/exercises/colab04-05.ipynb)

[github.com/ansyscale/academy/blob/master/ray-crash-
course/02-Ray-Actors.ipynb](https://github.com/ansyscale/academy/blob/master/ray-crash-course/02-Ray-Actors.ipynb)

Pattern: distributed multiprocessing pool

Distributed multiprocessing.Pool:

- make Python programs distributed, using actors
- easy to scale existing applications that use `multiprocessing.Pool`

```
from ray.util.multiprocessing import Pool

def f(index):
    return index

pool = Pool()
for result in pool.map(f, range(100)):
    print(result)
```

Pattern: distributed multiprocessing pool

Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_04_mult_pool.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_04_mult_pool.ipynb)

Others:

[github.com/anyscale/academy/blob/master/ray-crash-
course/04-Ray-Multiprocessing.ipynb](https://github.com/anyscale/academy/blob/master/ray-crash-course/04-Ray-Multiprocessing.ipynb)

Pattern: joblib

Distributed scikit-learn:

- drop-in replacement to parallelize **JobLib** (scikit-learn backend)

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC

digits = load_digits()
param_space = {
    'C': np.logspace(-6, 6, 30),
    'gamma': np.logspace(-8, 8, 30),
    'tol': np.logspace(-4, -1, 30),
    'class_weight': [None, 'balanced'],
}

model = SVC(kernel='rbf')
search = RandomizedSearchCV(model, param_space, cv=5, n_iter=300, verbose=10)

import joblib
from ray.util.joblib import register_ray

register_ray()
with joblib.parallel_backend('ray'):
    search.fit(digits.data, digits.target)
```



Pattern: joblib

Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_05_job_lib.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_05_job_lib.ipynb)

Others:

[github.com/anyscale/academy/blob/master/ray-crash-
course/04-Ray-Multiprocessing.ipynb](https://github.com/anyscale/academy/blob/master/ray-crash-course/04-Ray-Multiprocessing.ipynb)

Pattern: sharded, lazy iterators

Parallel Iterators:

- API for simple data ingest and processing
- fully serializable
- can operate over infinite sequences of items
- transformations based on method chaining
- passed to remote tasks and actors to represent data shards

Pattern: sharded, lazy iterators

```
import ray
import numpy as np

ray.init()

@ray.remote
def train(data_shard):
    for batch in data_shard:
        print("train on", batch) # perform model update with batch

it = (
    ray.util.iter.from_range(1000000, num_shards=4, repeat=True)
    .batch(1024)
    .for_each(np.array)
)

work = [train.remote(shard) for shard in it.shards()]
ray.get(work)
```

Pattern: sharded, lazy iterators

Tutorial:

[github.com/DerwenAI/ray_tutorial/blob/main/
ex_06_para_iter.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/ex_06_para_iter.ipynb)

Others:

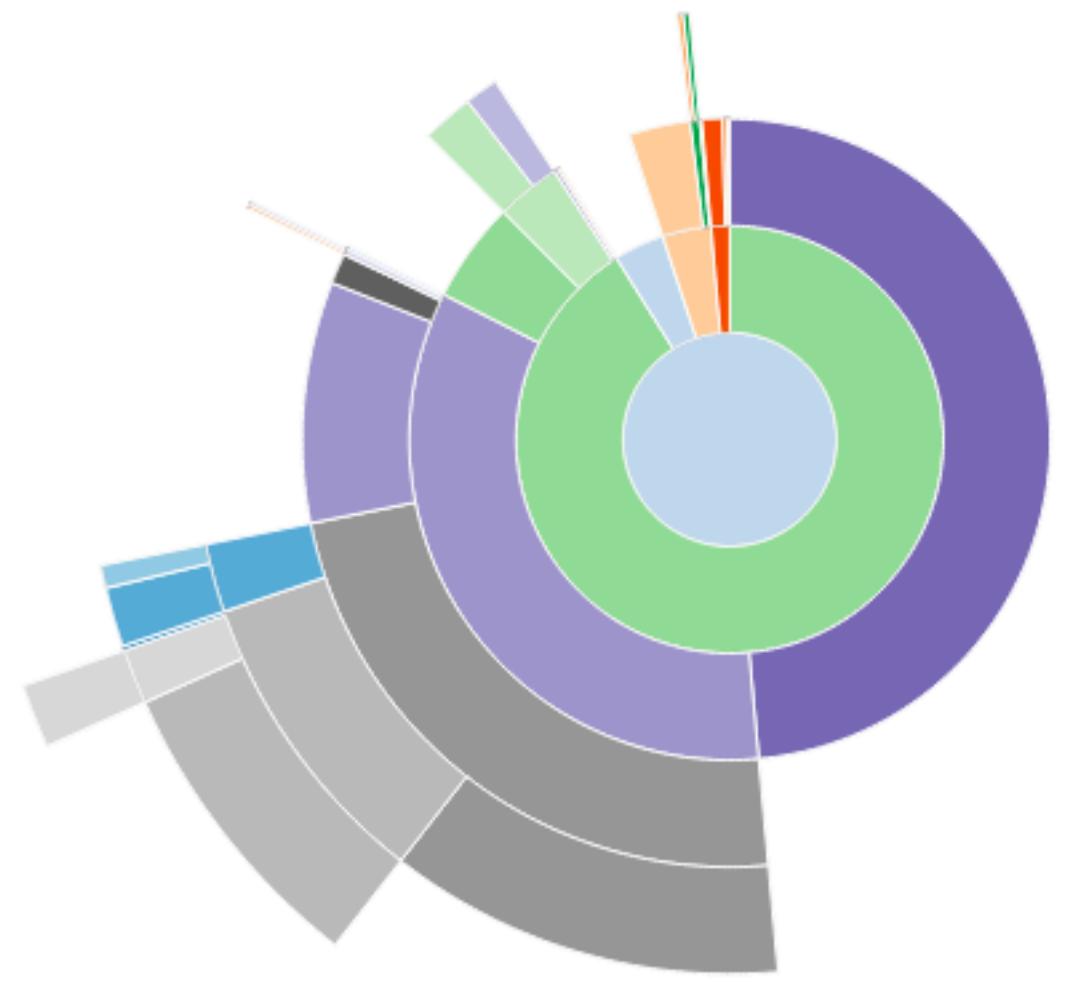
[github.com/anyscale/academy/blob/master/ray-crash-
course/05-Ray-Parallel-Iterators.ipynb](https://github.com/anyscale/academy/blob/master/ray-crash-course/05-Ray-Parallel-Iterators.ipynb)

Measure

Performance Analysis

Have a strategy for profiling in depth and how to use the feedback obtained from it:

1. Start with coarse grained analysis in Python.
2. Drill-down to finer grained profiling tools, troubleshooting bottlenecks as they get identified.
3. Iterate with “inspect, visualize, analyze, take action” to guide the performance analysis.



Performance Analysis

Py package	purpose	usage
watermark	Jupyter magic extension prints timestamps, library version numbers, hardware info	Keeping track of the details for each configuration you're profiling
Fil	Tracing peak memory usage	Determining which section of code caused the high-water mark
objgraph	Tracing and visualizing the object graph	Finding out which objects are referencing which other objects
tracemalloc	Tracing memory blocks allocated by Python	Computing the differences between snapshots to detect memory leaks
SnakeViz	Browser-based graphical viewer for cProfile output	Using icicle charts and sunburst charts to visualize compute-bound functions
cProfile	Built-in profiler (in C, for less overhead) for deterministic profiling of the call stack	Capturing the full statistics of the run time for a Python application
Pyinstrument	Statistical profiler of the call stack	Estimating compute times for particular sections of code (less distorted by overhead)

Performance Analysis

Tutorial:

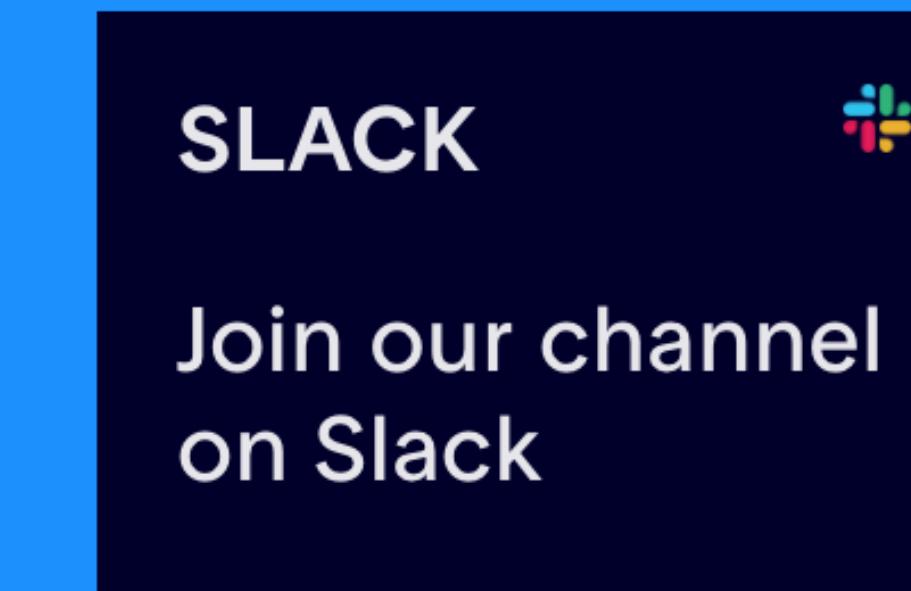
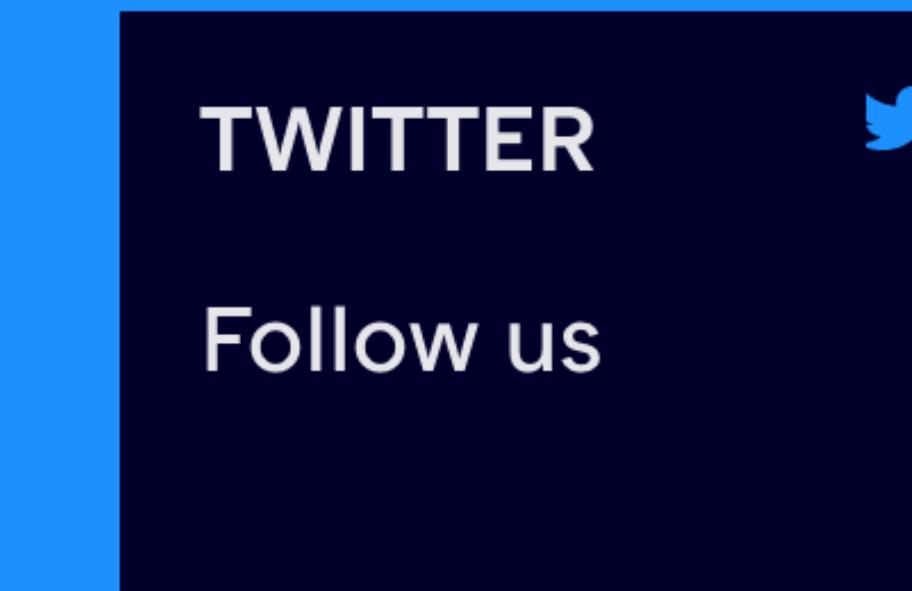
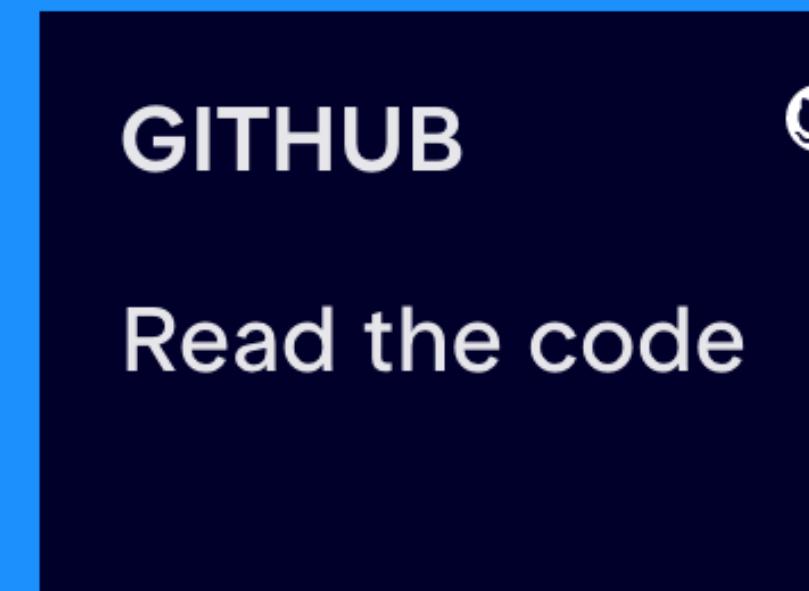
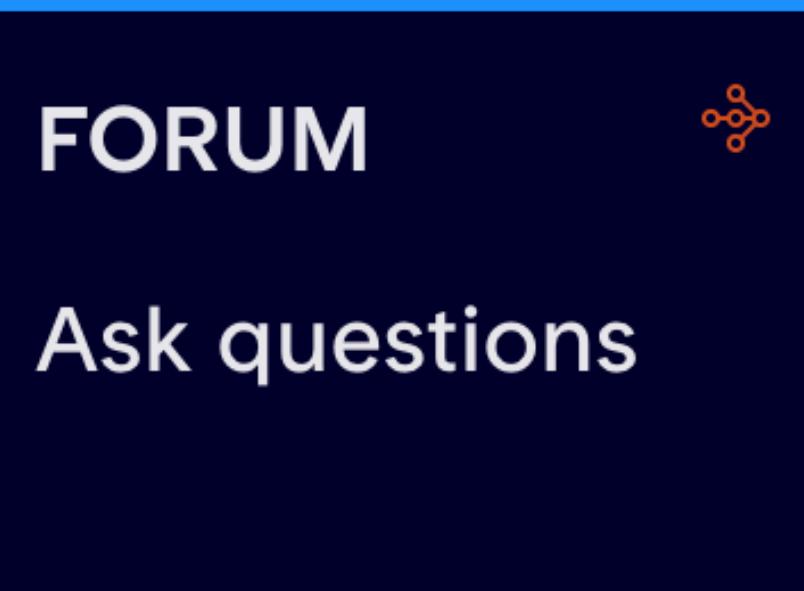
[github.com/DerwenAI/ray_tutorial/blob/main/
pi.ipynb](https://github.com/DerwenAI/ray_tutorial/blob/main/pi.ipynb)

Others:

- docs.ray.io/en/master/auto_examples/testing-tips.html
- jakevdp.github.io/PythonDataScienceHandbook/01.07-timing-and-profiling.html
- towardsdatascience.com/speed-up-jupyter-notebooks-20716cbe2025
- scoutapm.com/blog/identifying-bottlenecks-and-optimizing-performance-in-a-python-codebase
- code.tutsplus.com/tutorials/understand-how-much-memory-your-python-objects-use--cms-25609

Resources

Get involved with the Ray community



- <https://discuss.ray.io/>
- <https://github.com/ray-project/ray>
- <https://twitter.com/raydistributed>
- <https://tinyurl.com/rayslack>

A tour through “Ray Design Patterns”

The screenshot shows a Google Document interface with the title "Ray Design Patterns".

Left Sidebar (Table of Contents):

- Ray Design Patterns
 - ★ This is a community maint...
- Basic Patterns
 - Pattern: Tree of Actors
 - Notes
 - Code example
 - Pattern: Tree of Tasks
 - Example use case
 - Code example
 - Pattern: Map and Reduce
 - Example use case
 - Pattern: Using ray.wait to limit t...
 - Code example
- Basic Antipatterns
 - Antipattern: Accessing Global V...
 - Code example

publications, interviews, conference summaries...

[@pacoid](https://derwen.ai/paco)

Thank you!



derwen.ai