

CS162
Operating Systems and
Systems Programming
Lecture 25

Ray and vLLM

December 2nd, 2024

Prof. Ion Stoica

<http://cs162.eecs.Berkeley.edu>

Lecturers Today

Simon

Woosuk

- 4th-year PhD advised by Ion
- Research interests : Systems + AI

Some projects We've worked on in Berkeley's Labs



MemGPT



Chatbot Arena



Vicuna

Applications

Alpa

AlpaServe



Optimizers

(optimize ML workloads)



Distributed Frameworks
(dev & run distributed apps)



Apache
MESOS™



SkyPilot



Skyplane

Infra & Orchestration Layer
(allocate resources)

Last lecture



MemGPT



Chatbot Arena



Vicuna

Applications

Alpa

AlpaServe



Optimizers

(optimize ML workloads)



Distributed Frameworks
(dev & run distributed apps)



Apache
MESOS™



SkyPilot



Skyplane

Infra & Orchestration Layer
(allocate resources)

This Lecture



MemGPT



Chatbot Arena



Vicuna

Applications

Alpa

AlpaServe

LLM



Optimizers
(optimize ML workloads)



Apache
MESOS™



SkyPilot

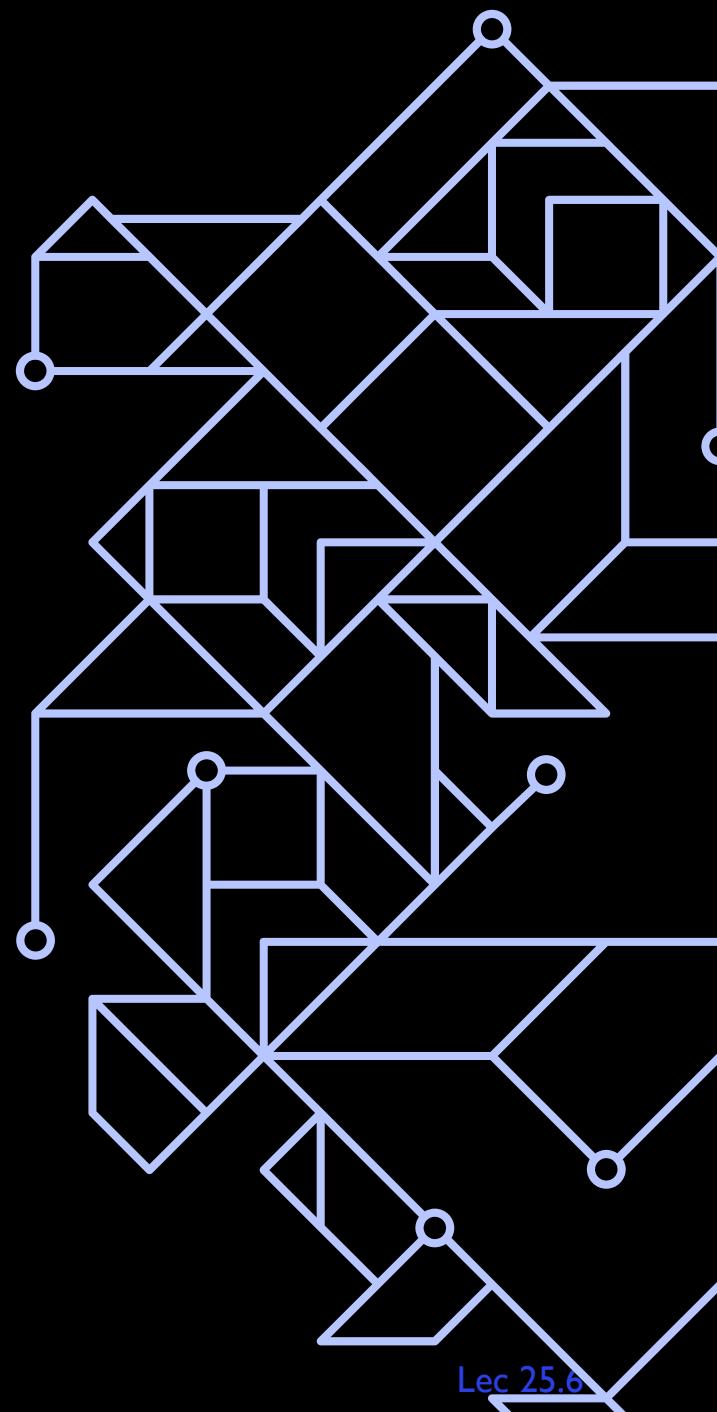


Skyplane

Distributed Frameworks
(dev & run distributed apps)

Infra & Orchestration Layer
(allocate resources)

Ray (2016 -)

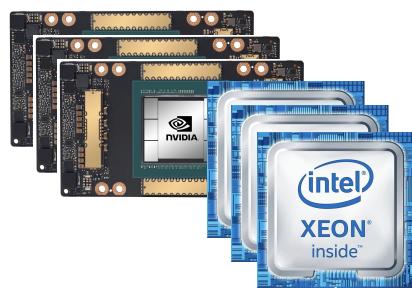


Some historical trends

2009 (Spark)



2016
(Ray)



- Big Data & Classic ML
 - Homogeneous clusters
- + DL & RL
- Heterogeneous clusters

Two ML students

They wanted to train
neural networks distributedly

Ion asked them to train on Spark

- Why reinvent the wheel?

And they did it!



**Rober
t**



**Philip
p**

SPARKNET: TRAINING DEEP NETWORKS IN SPARK

Philipp Moritz,* Robert Nishihara,* Ion Stoica, Michael I. Jordan
Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720, USA
`{pcmoritz,rkn,istoica,jordan}@eecs.berkeley.edu`

ABSTRACT

Training deep networks is a time-consuming process, with networks for object recognition often requiring multiple days to train. For this reason, leveraging the resources of a cluster to speed up training is an important area of work. However, widely-popular batch-processing computational frameworks like MapReduce and Spark were not designed to support the asynchronous and

But pretty soon some issues

Spark was based on JVM, since this is what the Big Data ecosystem was based on, e.g., Hadoop, Kafka, Flink, etc

- JVM doesn't have support for GPUs

The DNN software stack: Python and C++, e.g., Tensorflow, PyTorch, etc

- Very good support for GPUs

BSP (Bulk Synchronous Processing) model great for data, but not flexible enough for AI workloads

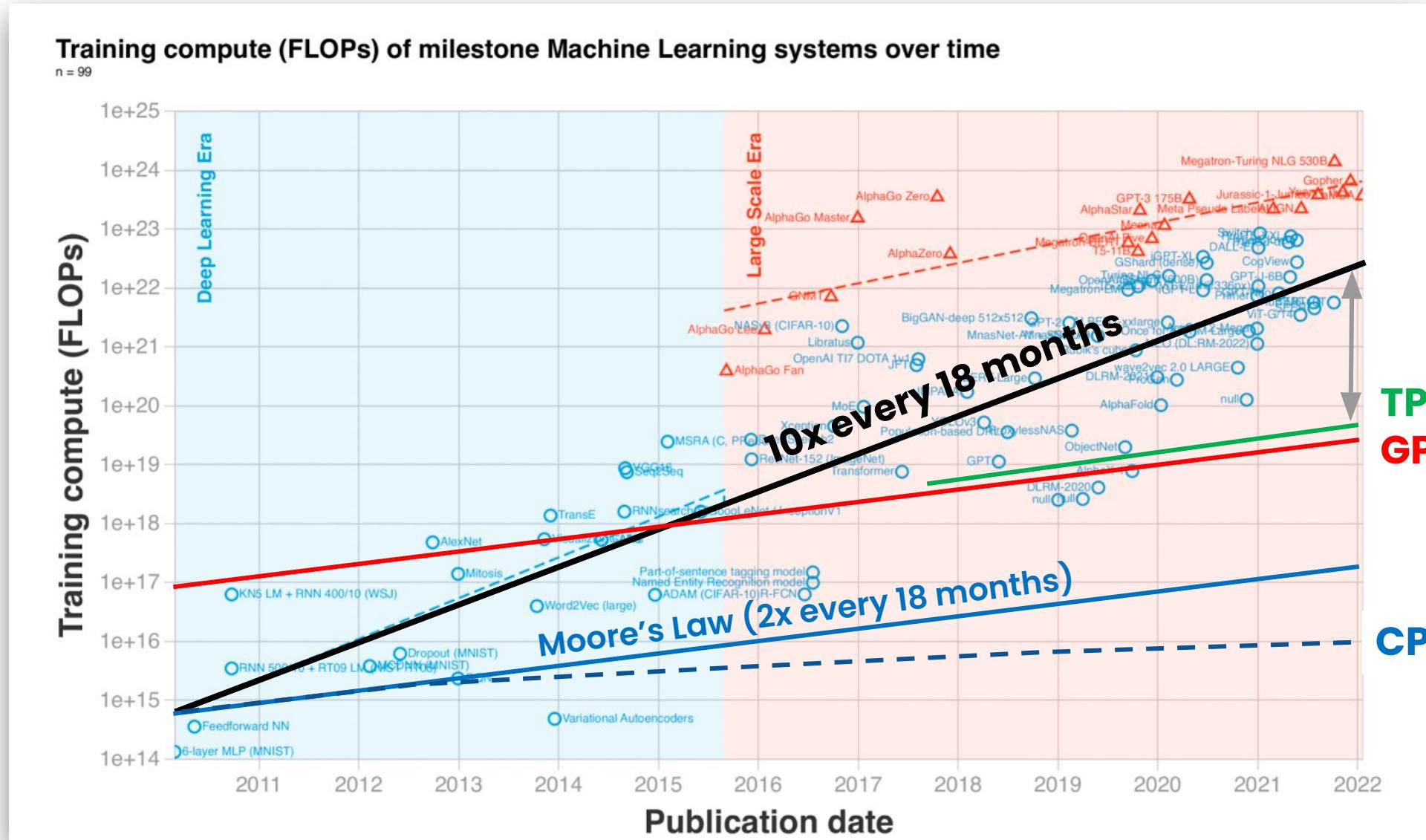
Trends when we started Ray

AI demands growing faster than single node capabilities

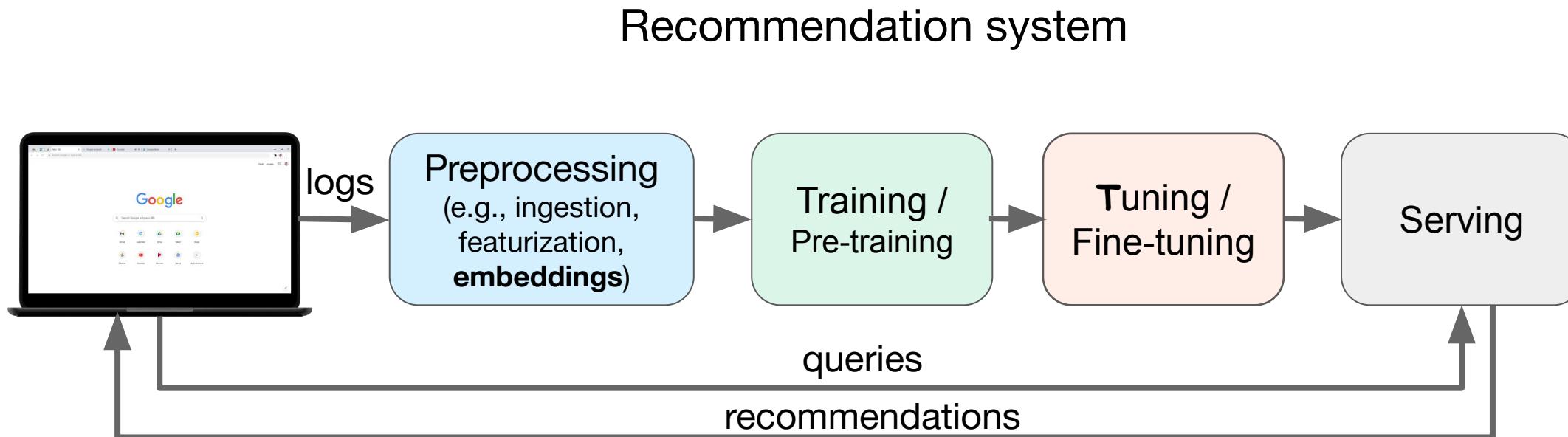
Clusters becoming heterogeneous: GPUs + CPUs

AI workloads becoming more complex

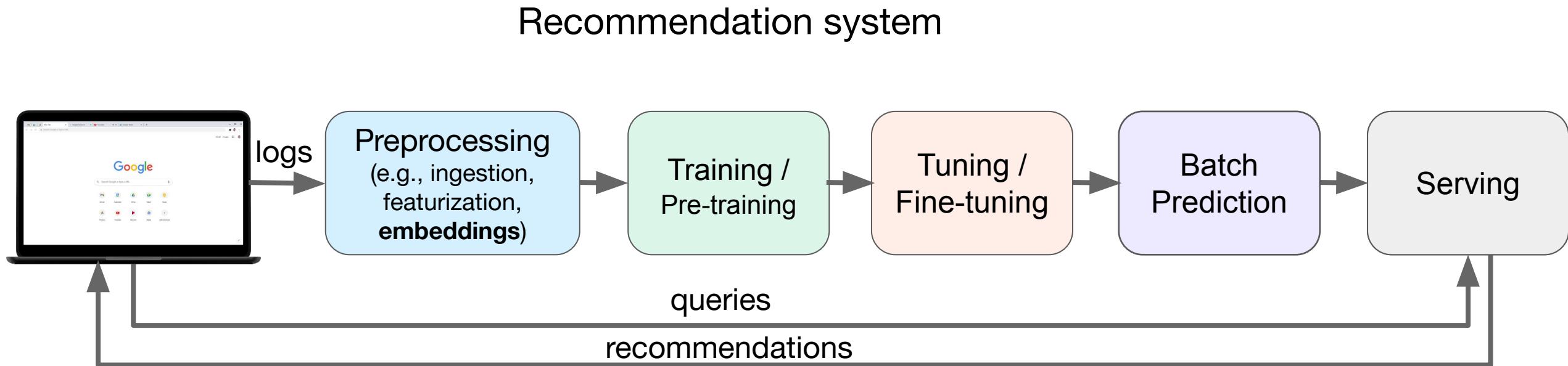
AI Demands growing faster than hardware capabilities



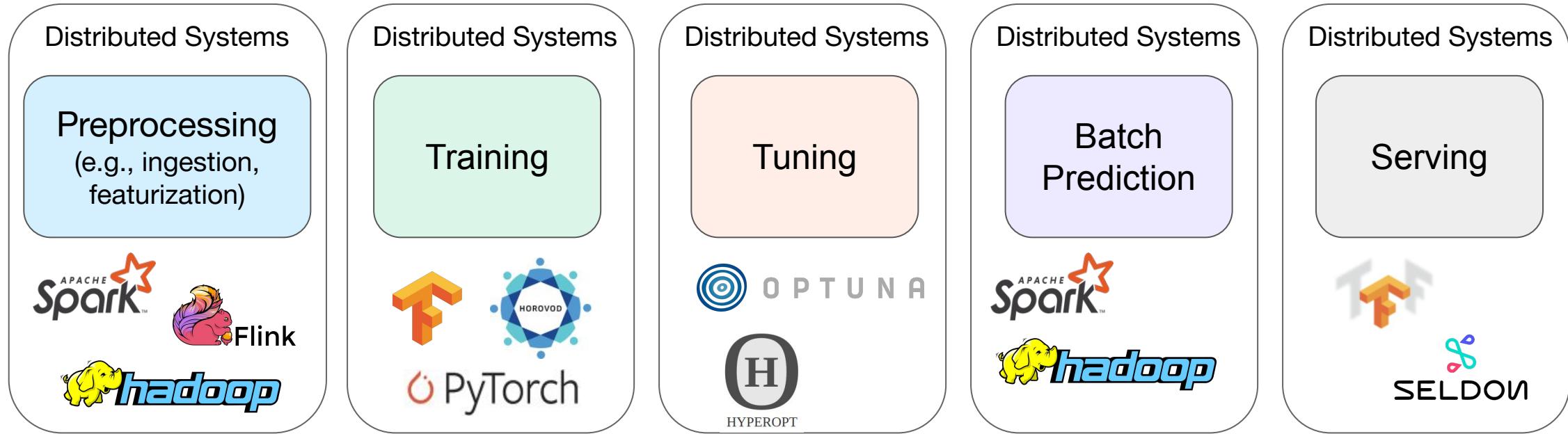
AI workloads becoming more complex



AI workloads becoming more complex



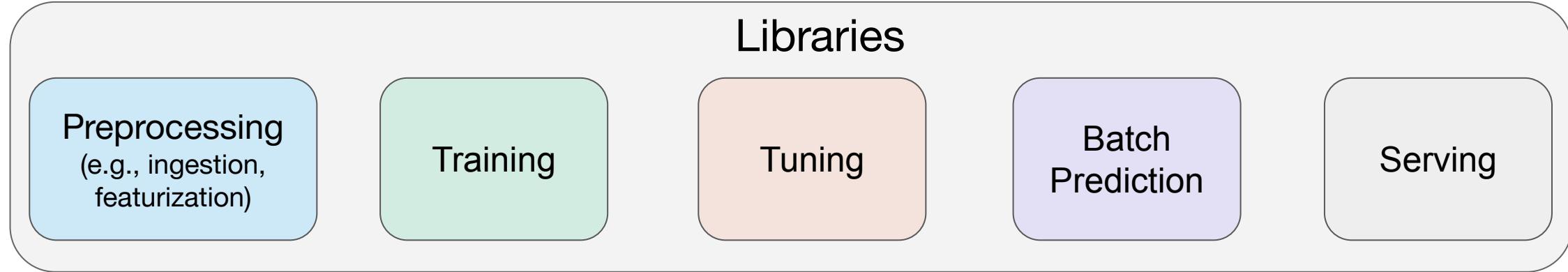
Challenge: need to scale every stage!



Need to stitch together a bunch of disparate systems

- **Hard** to develop
- **Hard** to deploy
- **Hard** to manage
- **Slow**

Ray: One system to support all these workloads



<code>ray.init()</code>	Initialize Ray context.
<code>@ray.remote</code>	Function or class decorator specifying that the function will be executed as a task or the class as an actor in a different process.
<code>.remote</code>	Postfix to every remote function, remote class declaration, or invocation of a remote class method. Remote operations are <i>asynchronous</i> .
<code>ray.put()</code>	Store object in object store, and return its ID. This ID can be used to pass object as an argument to any remote function or method call. This is a <i>synchronous</i> operation.
<code>ray.get()</code>	Return an object or list of objects from the object ID or list of object IDs. This is a <i>synchronous</i> (i.e., blocking) operation.
<code>ray.wait()</code>	From a list of object IDs returns (1) the list of IDs of the objects that are ready, and (2) the list of IDs of the objects that are not ready yet.

The FAST Compute Model

The FAST Compute Model

Actors: remote class instance (object)

Tasks: remote functions

The **FAST** Compute Model

Futures: reference to objects (possibly not created yet)

Actors: remote class instance (object)

Tasks: remote functions

The FAST Compute Model

Futures: reference to objects (possibly not created yet)

Actors: remote class instance (object)

Shared in-memory distributed object store

Tasks: remote functions

Python example

```
def f(x):  
    # compute...for 1s  
    return r
```

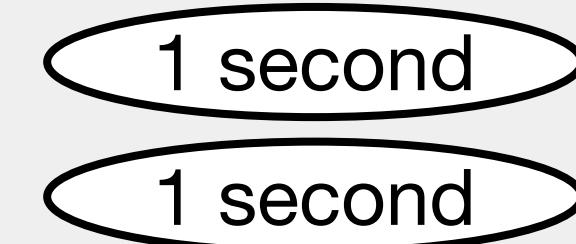
Python example

```
def f(x):  
    # compute...for 1s  
    return r  
  
x = f(a)     1 second
```

Python example

```
def f(x):  
    # compute...for 1s  
    return r
```

```
x = f(a)  
y = f(b)
```



2 seconds

Ray

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r
```

Ray: Function → Task

```
@ray.remote
```

```
def f(x):  
    # compute...for 1s  
    return r
```

```
x_ref = f.remote(a)
```

```
y_ref = f.remote(b)
```

```
ray.get([x_ref, y_ref])
```

Driver

Worker

Worker

...

Ray: Function → Task

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r  
  
x_ref = f.remote(a)  
y_ref = f.remote(b)  
ray.get([x_ref, y_ref])
```

Driver

Worker

Worker

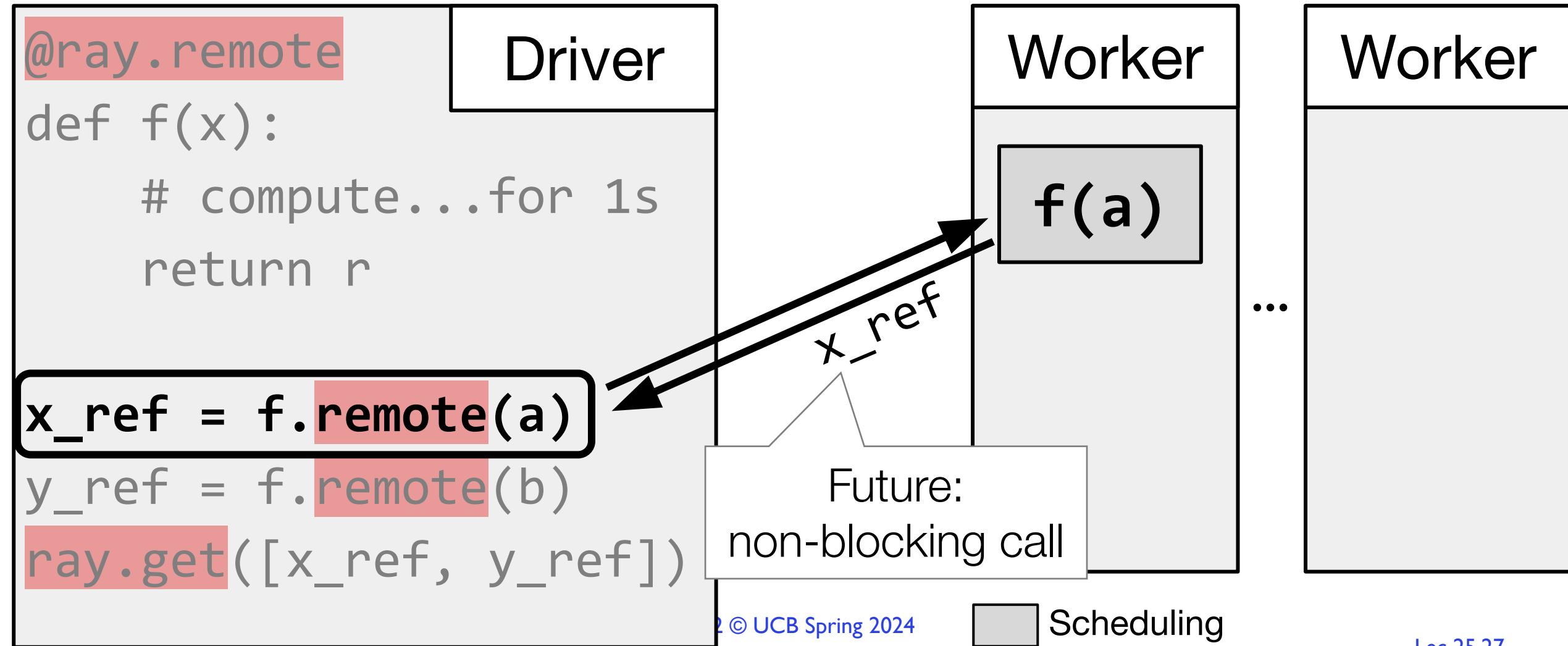
f(a)

Task

...

Scheduling

Ray: Function → Task



Ray: Function → Task

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r  
  
x_ref = f.remote(a)  
y_ref = f.remote(b)  
ray.get([x_ref, y_ref])
```

Driver

Worker

Worker

Ray: Function → Task

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r  
  
x_ref = f.remote(a)  
y_ref = f.remote(b)  
ray.get([x_ref, y_ref])
```

Driver

Worker

Worker

f(a)

f(b)

...

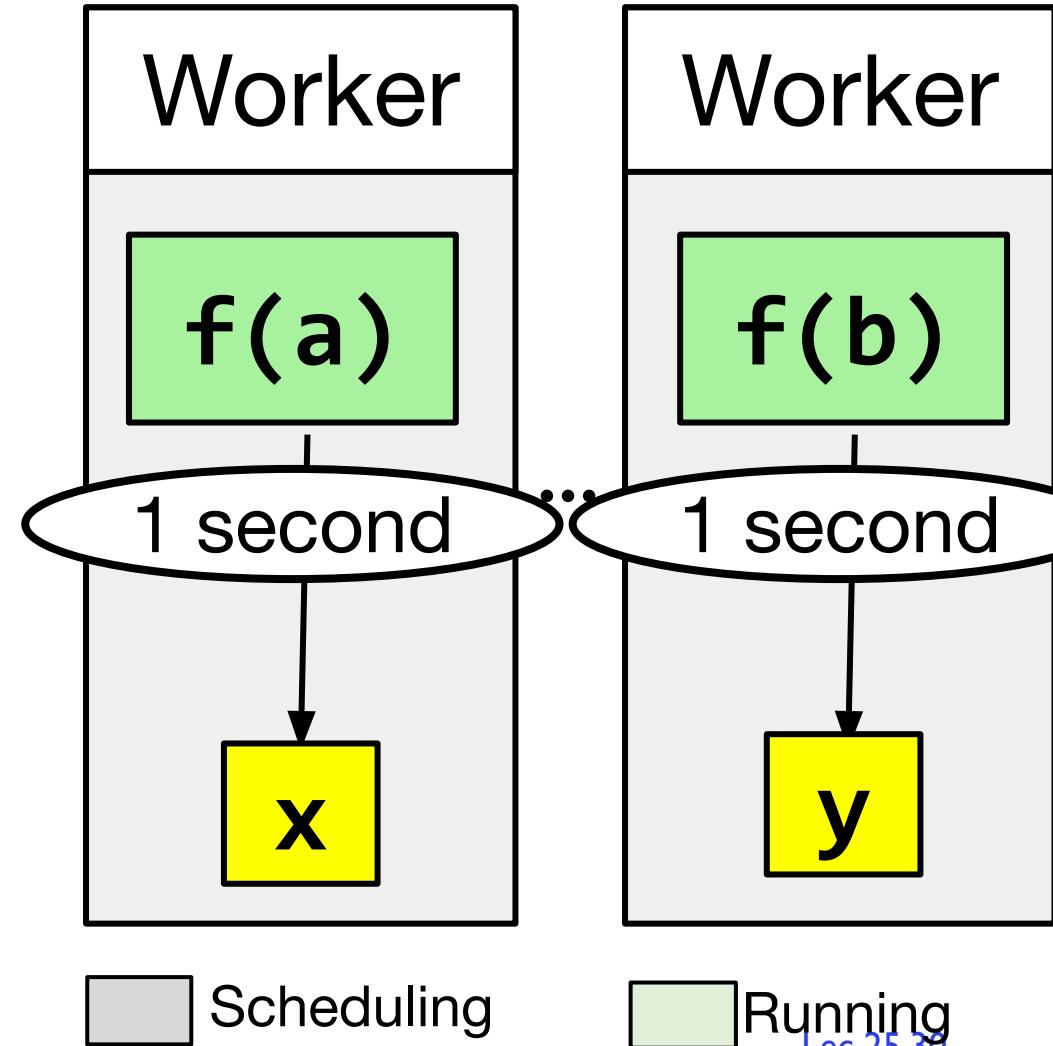
Blocking call

Scheduling

Ray: Function → Task

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r  
  
x_ref = f.remote(a)  
y_ref = f.remote(b)  
ray.get([x_ref, y_ref])
```

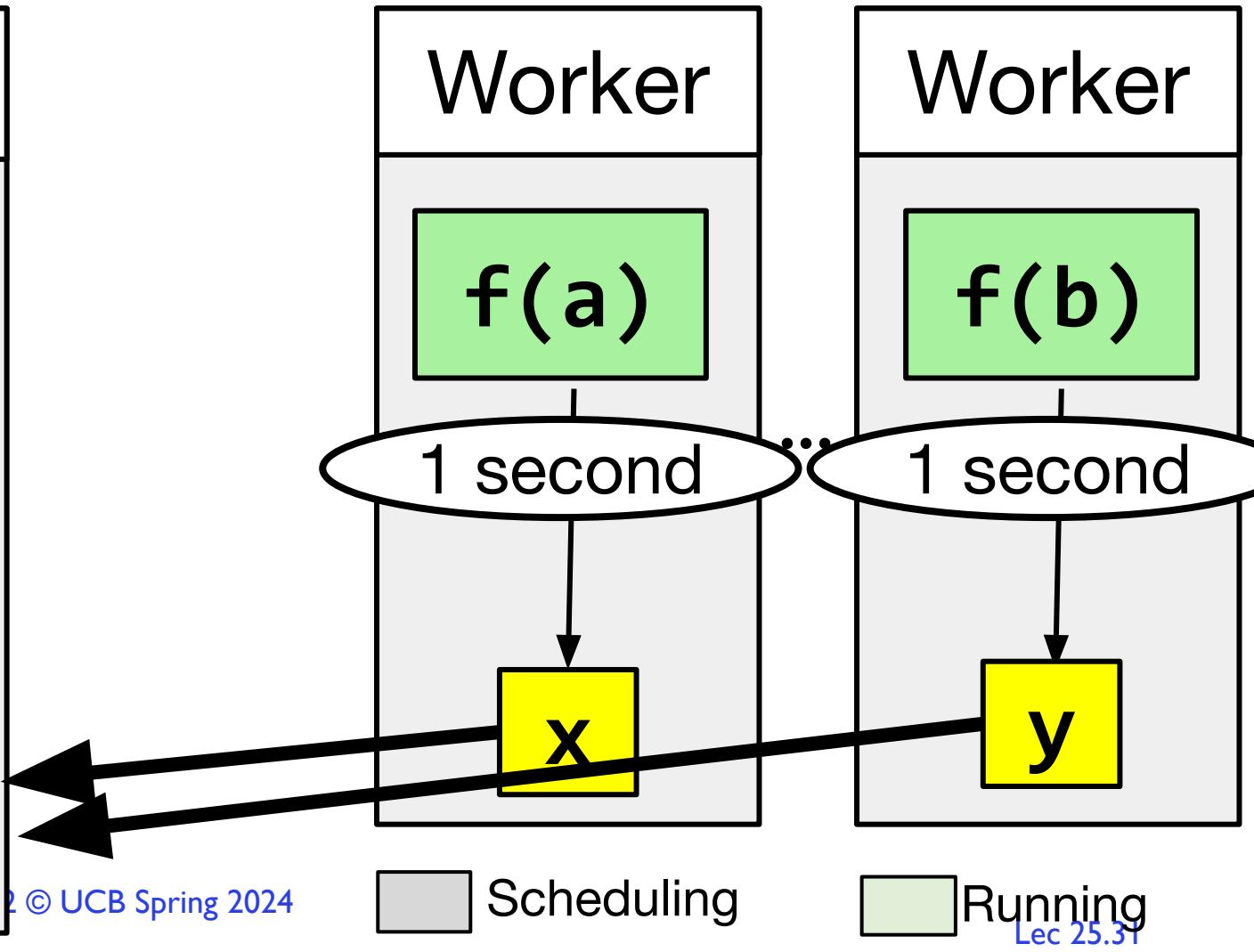
Driver



Ray: Function → Task

```
@ray.remote  
def f(x):  
    # compute...for 1s  
    return r  
  
x_ref = f.remote(a)  
y_ref = f.remote(b)  
  
ray.get([x_ref, y_ref])
```

x **y**



Python Class

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

c = Counter()
c.inc()
c.inc()
```

Python Class

Ray: Class → Actor

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

```
c = Counter()  
c.inc()  
c.inc()
```

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

```
c = Counter.remote()  
x_ref = c.inc.remote()  
y_ref = c.inc.remote()
```

Python Class

Ray: Class → Actor

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

```
c = Counter()
c.inc()
c.inc()
```

```
@ray.remote(num_cpus=2, num_gpus=1)
```

```
class Counter(object)
```

can specify resource demands;
support heterogeneous hardware

```
def inc(self).
```

```
    self.value += 1
```

```
    return self.value
```

```
c = Counter.remote()
```

```
x_ref = c.inc.remote()
```

```
y_ref = c.inc.remote()
```

Shared in-memory object store

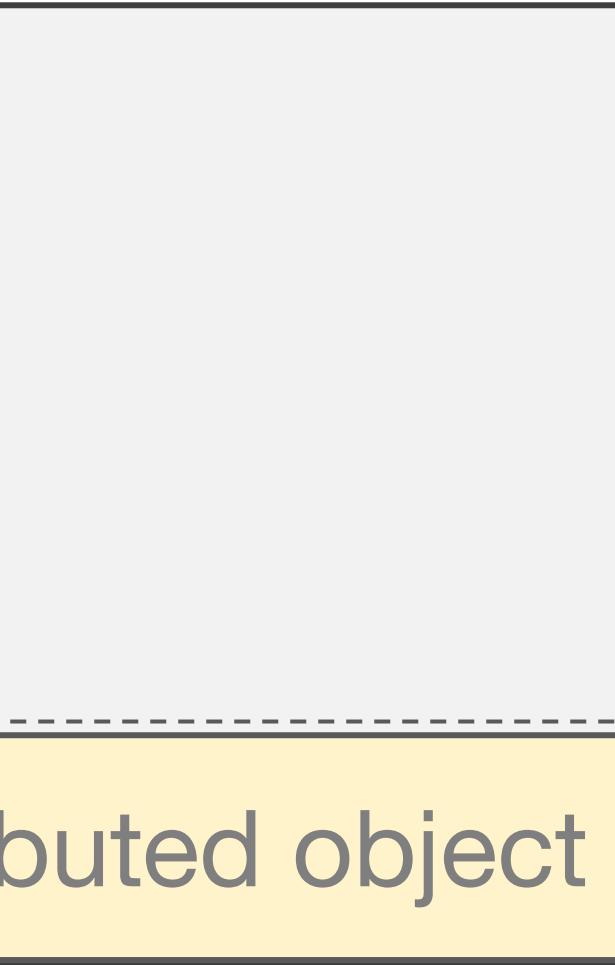
node 1

```
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

x_ref = f.remote()
y_ref = g.remote(x_ref)
```

node 2



node 3

Shared in-memory object store

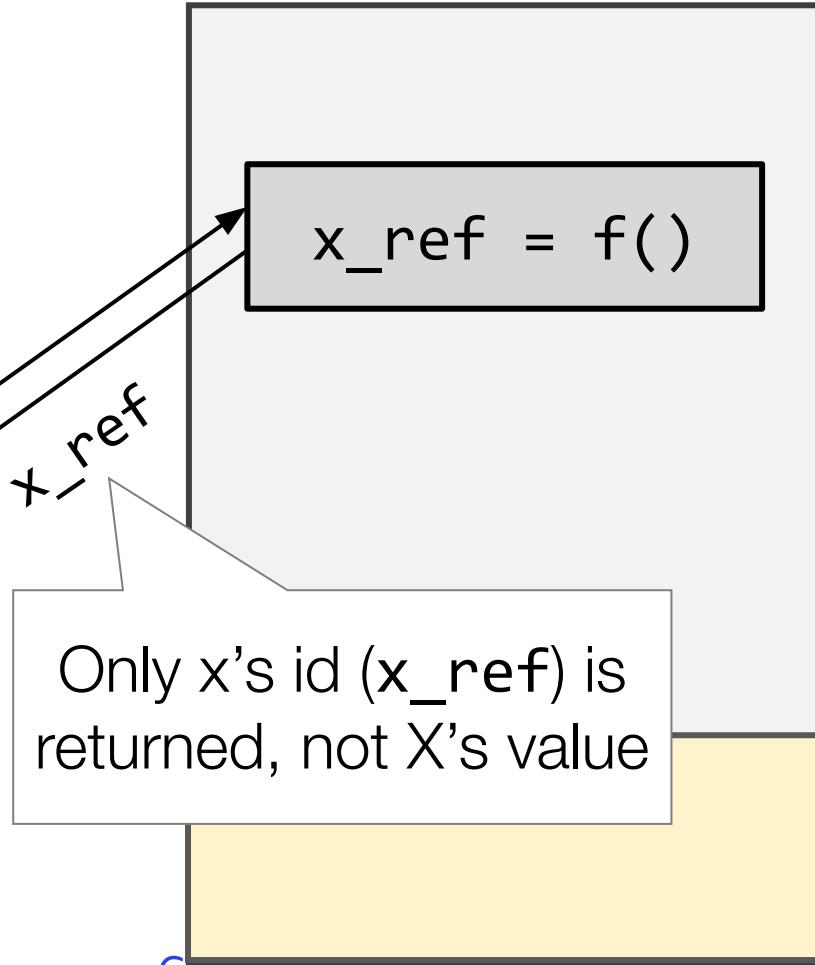
node 1

```
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

x_ref = f.remote()
y_ref = g.remote(x_ref)
```

node 2



node 3

Shared in-memory object store

node 1

```
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

x_ref = f.remote()
y_ref = g.remote(x_ref)
```

node 2

```
x_ref = f()
```

node 3

```
y_ref = g(x_ref)
```

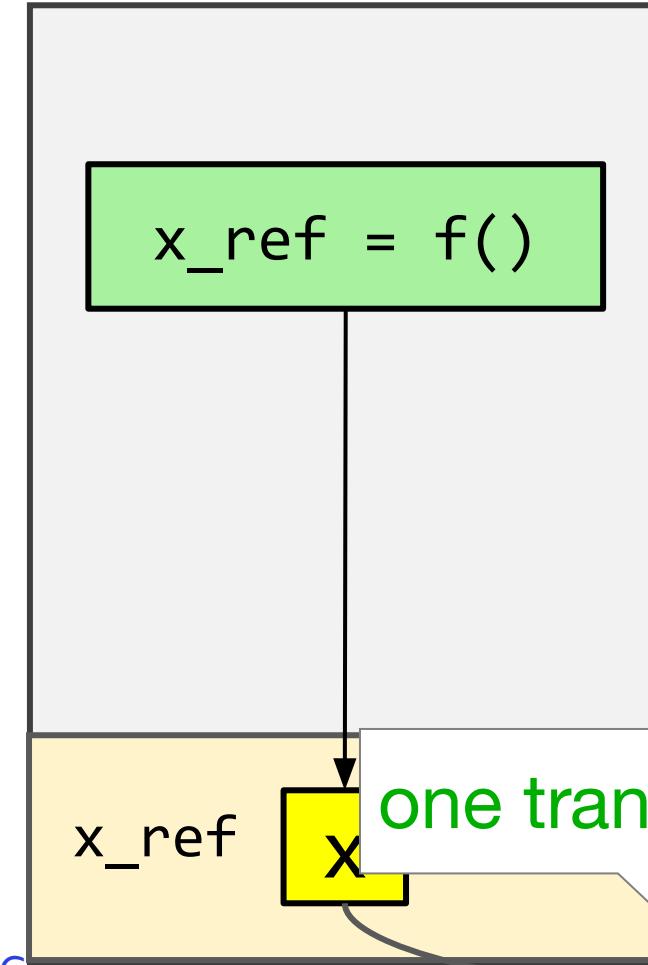
y_ref

Shared in-memory object store

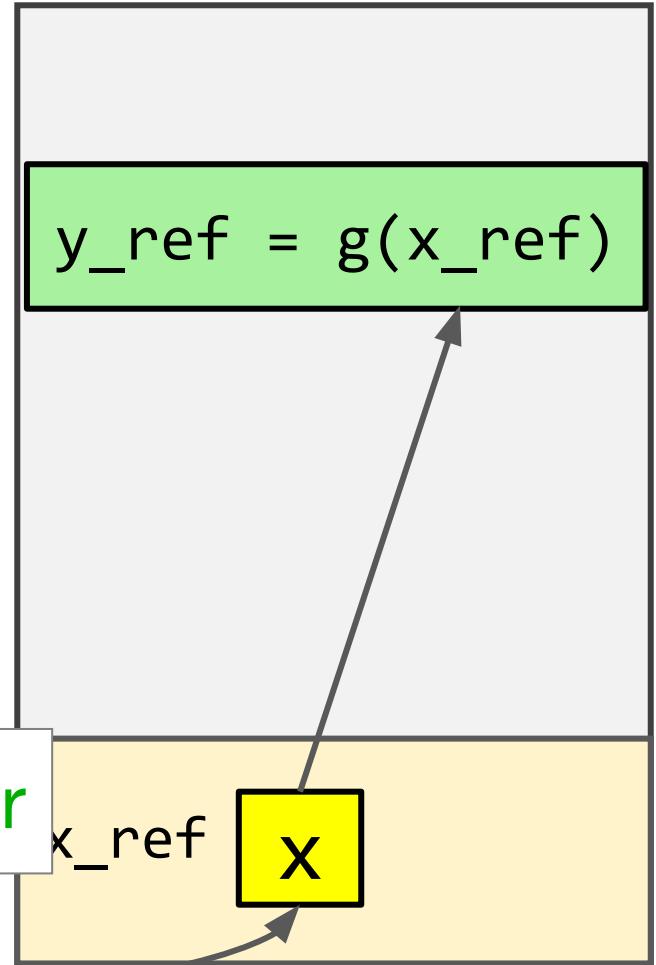
node 1

```
@ray.remote  
def f():  
    # compute...  
    return x  
  
@ray.remote  
def g(a):  
    # compute...  
    return y  
  
x_ref = f.remote()  
y_ref = g.remote(x_ref)
```

node 2



node 3



No object store: object x transferred twice!

node 1

```
@ray.remote
def f():
    # compute...
    return x

@ray.remote
def g(a):
    # compute...
    return y

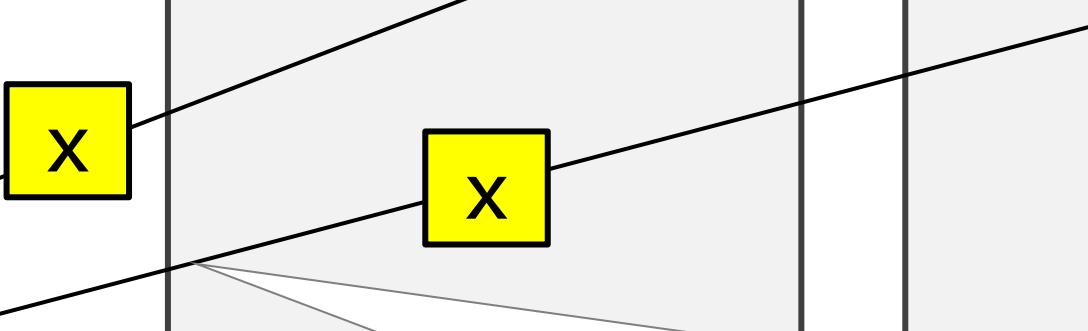
x = f.remote()
y = g.remote(x)
```

node 2

$$x = f()$$

node 3

$$y = g(x_{\text{ref}})$$



Without object store x would have been transferred **twice**:
node 2 → node 1 and node 1 → node 3!

Ray Ecosystem

Contributors 1,030

Star 33.7k

Ray AI Libraries enables simple scaling of AI workloads

Data

Train

Tune

Serve

RLLib

Ray core enables generic scalable Python applications.

Custom applications



Unified computing
framework for
distributed apps

Tasks

Actors

Object Store



Ray emerging as the compute framework for Modern AI infrastructure

Ray breaks the \$1/TB barrier as the world's most cost-efficient sorting system

By Frank Sifei Luan, UC Be

The Magic of Merlin: Shopify's New Machine Learning Platform

by Isaac Vidas · Data Science & Engineering
Apr 6, 2022 · 11 minute read

Unleashing ML Innovation at Spotify with Ray



aws

Contact Us Support My Account

re:Invent Products Solutions Pricing Documentation Learn More

Blog Home Category Edition

AWS Big Data Blog

Introducing AWS Glue for Ray: Scaling your data integration workloads using Python

by Zach Mitchell, Ishan Gaur, Kinshuk Bahare, and Derek Liu | Oct 28, 2021

How Ray, a Distributed AI Framework, helps Power ChatGPT

Google Cloud Blog

Large Scale Deep Learning Training and Tuning with Ray at Uber

Xu Ning, Di Yu, Michael Mui



Han Li

Mar 17 · 9 min read · Listen



Distributed Machine Learning at Instacart

How Instacart uses distributed Machine Learning to efficiently thousands of models in production

Author



Netflix Technology Blog

Feb 13 · 11 min read · Listen



Scaling Media Machine Learning at Netflix

Foundation models trained on Ray

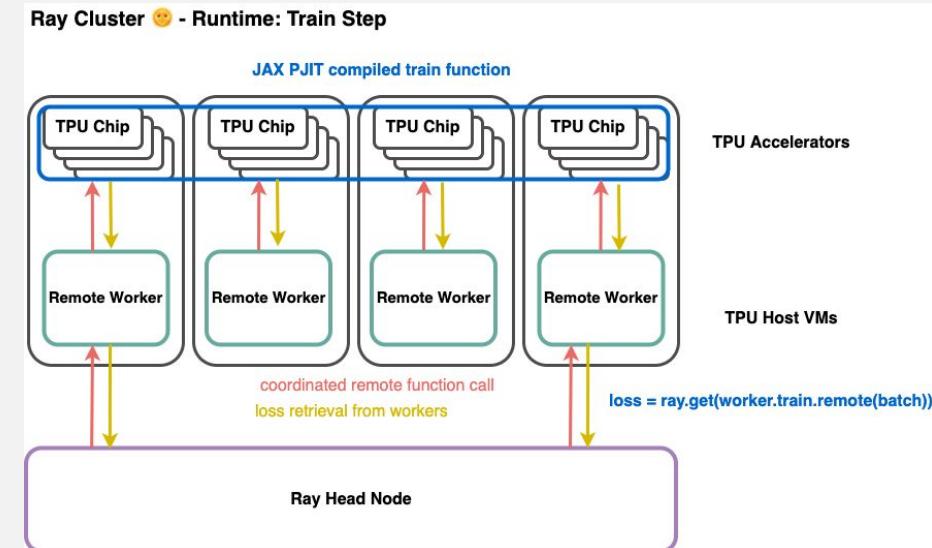


"We use Ray to train our largest models including ChatGPT"

Greg Brockman, Co-founder and President, OpenAI

co:here

Use JAX and TPU to train their LLM models

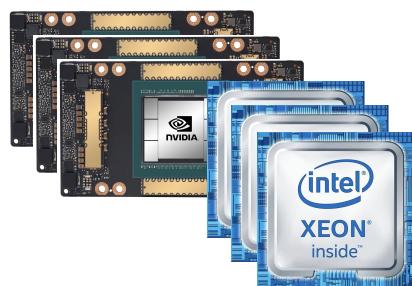


Some historical trends

2009 (Spark)



2016
(Ray)



2022



- Big Data & Classic ML
- Homogeneous clusters

- + DL & RL
- Heterogeneous clusters

- + LLMs
- Increased cluster heterogeneity;
increase vertical integrations (e.g. TPU pods)

Ray classic API: weak support for GPUs

Trend: rapid transition CPU-centric accelerator-centric world

Ray, very dynamic and flexible, but high overhead for small tasks

- Pay the cost of RPC
- Pay the cost of dynamic memory allocation
- Difficult to efficiently support p2p protocols like RDMA or NCCL
 - Require upfront resource allocation on all peers to avoid deadlock

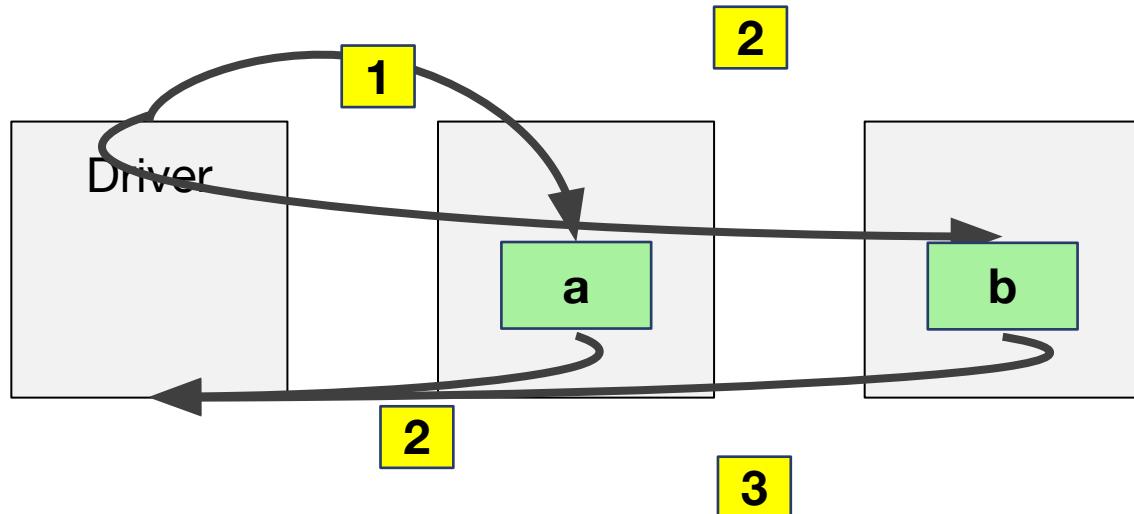
A simple example: RPC-like Execution

```
import ray

@ray.remote
class IncrementActor:
    def inc(self, value):
        return value + 1

a = IncrementActor.remote()
b = IncrementActor.remote()

res_ref = a.echo.remote(1)
res_ref = b.echo.remote(msg_ref)
print(ray.get(res_ref))
```



- Four data copies



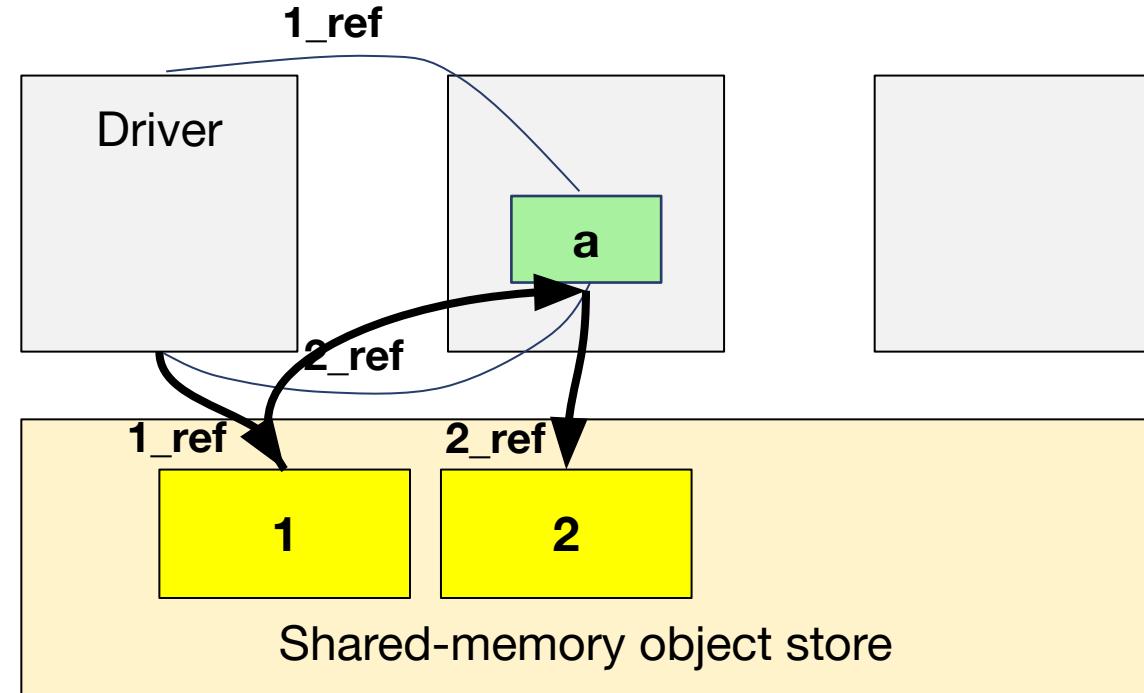
Default execution with Ray Core

```
import ray

@ray.remote
class IncrementActor:
    def inc(self, value):
        return value + 1

a = IncrementActor.remote()
b = IncrementActor.remote()

res_ref = a.echo.remote(1)
res_ref = b.echo.remote(msg_ref)
print(ray.get(res_ref))
```



→data →control

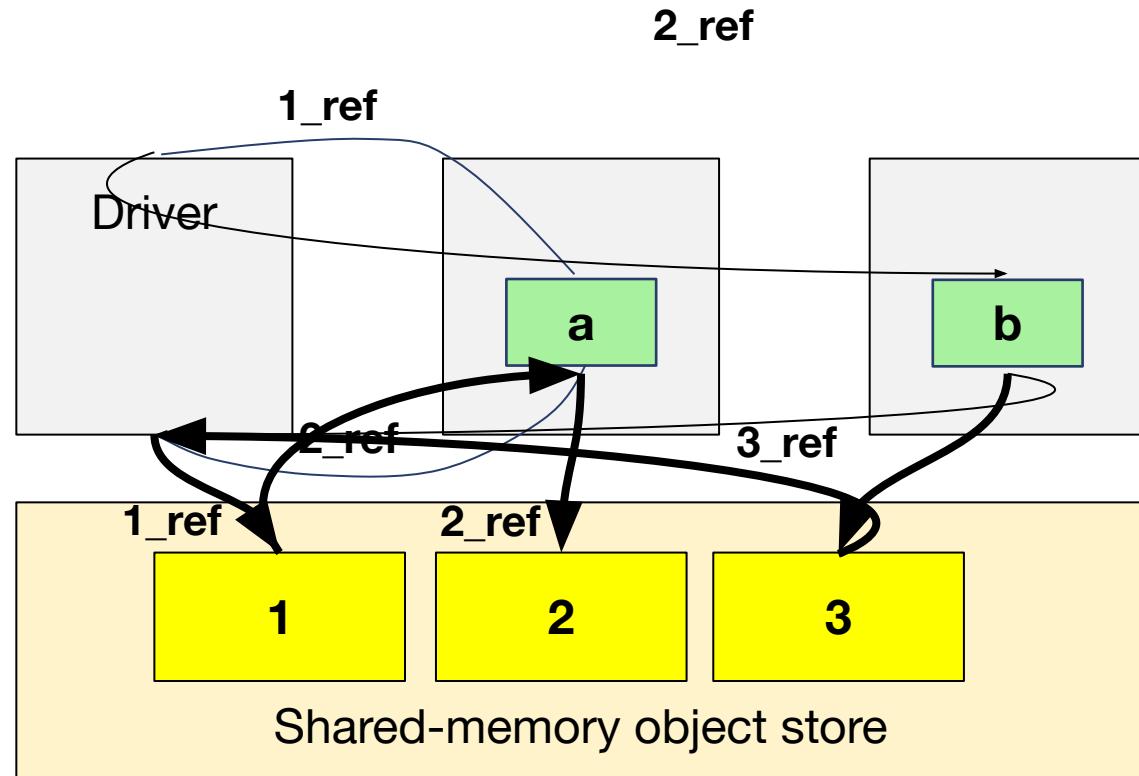
Default execution with Ray Core

```
import ray

@ray.remote
class IncrementActor:
    def inc(self, value):
        return value + 1

a = IncrementActor.remote()
b = IncrementActor.remote()

res_ref = a.echo.remote(1)
res_ref = b.echo.remote(msg_ref)
print(ray.get(res_ref))
```



- Two data copies (good for large/shared data)
- 6 RPCs & dynamic memory allocation



Compiled Graphs

Goals

- Run tasks that are 1-10ms with <1% system overhead
- GPU-GPU data transfers with <10us system overhead per op.

What are compiled graphs? Static task graphs with Ray Core-like API

- Allocate resources once, reuse them for multiple executions
- Declare p2p communication schedule before execution

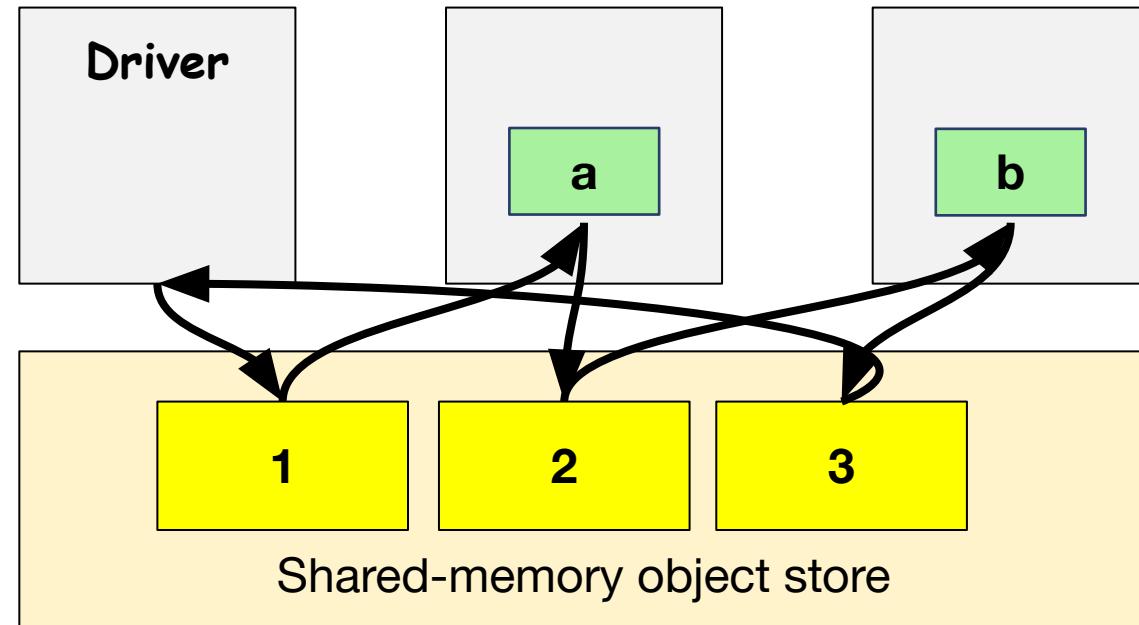
Ideal execution with Ray

```
import ray

@ray.remote
class IncrementActor:
    def inc(self, value):
        return value + 1

a = IncrementActor.remote()
b = IncrementActor.remote()

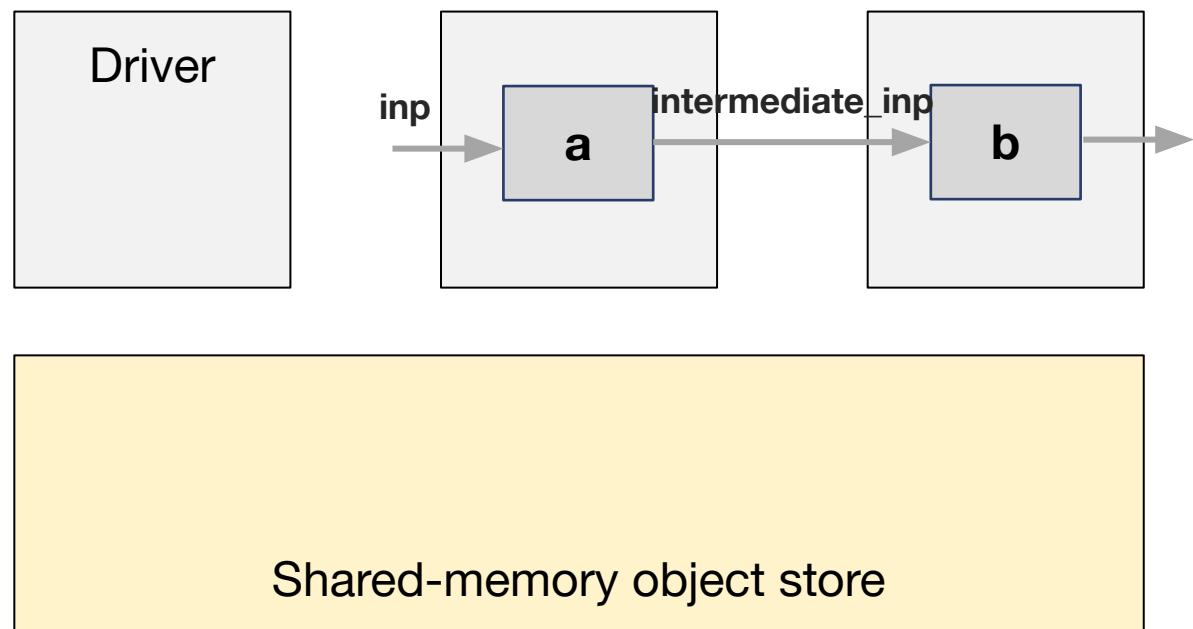
res_ref = a.echo.remote(1)
res_ref = b.echo.remote(msg_ref)
print(ray.get(res_ref))
```



- Get rid of control RPCs for each invocation
- Avoid dynamic memory allocation

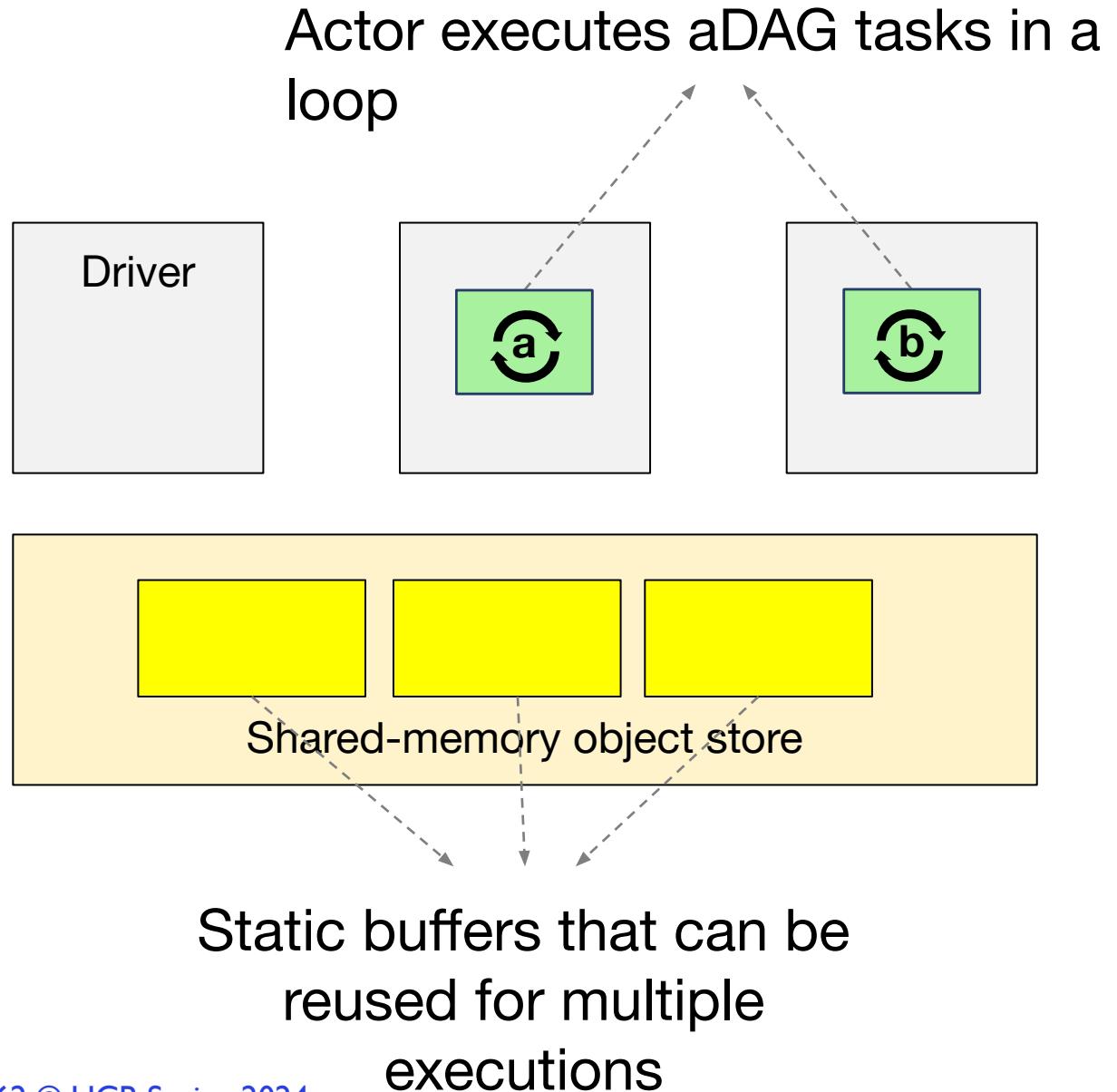
Ray compiled graphs: Compilation

```
a = IncrementActor.remote()  
b = IncrementActor.remote()
```



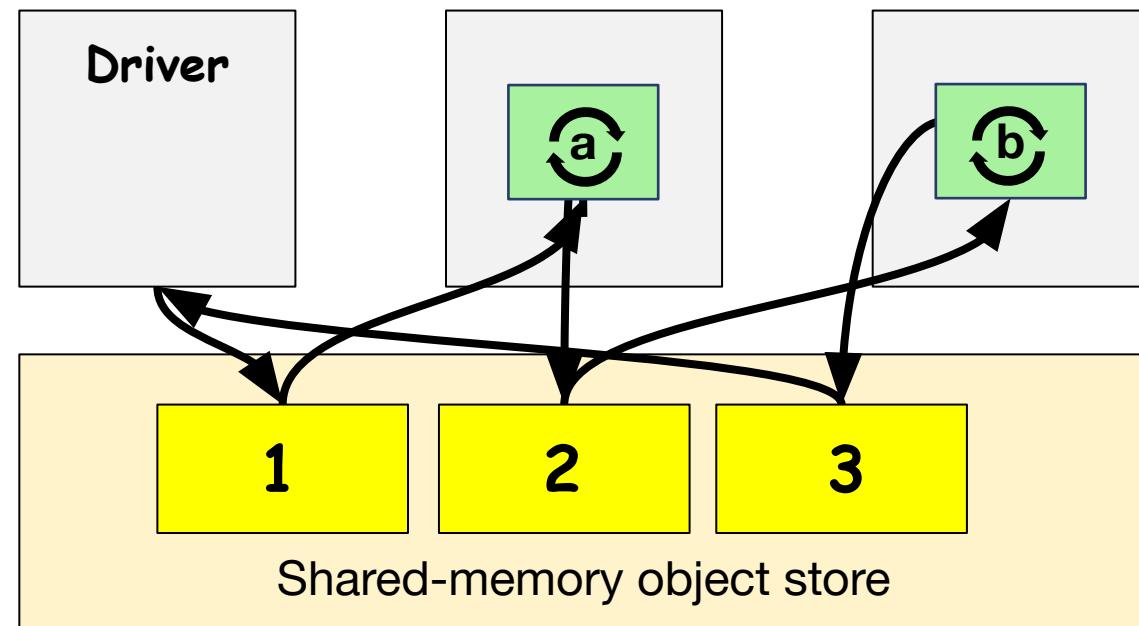
Ray compiled graphs: Compilation

```
a = IncrementActor.remote()  
b = IncrementActor.remote()  
  
# Using DAG for lazy execution.  
with ray.dag.InputNode() as inp:  
    # Bind the actor task to an  
    # input placeholder.  
    intermediate_inp = a.inc.bind(inp)  
    dag = b.inc.bind(intermediate_inp)
```



Ray compiled graphs: Compilation

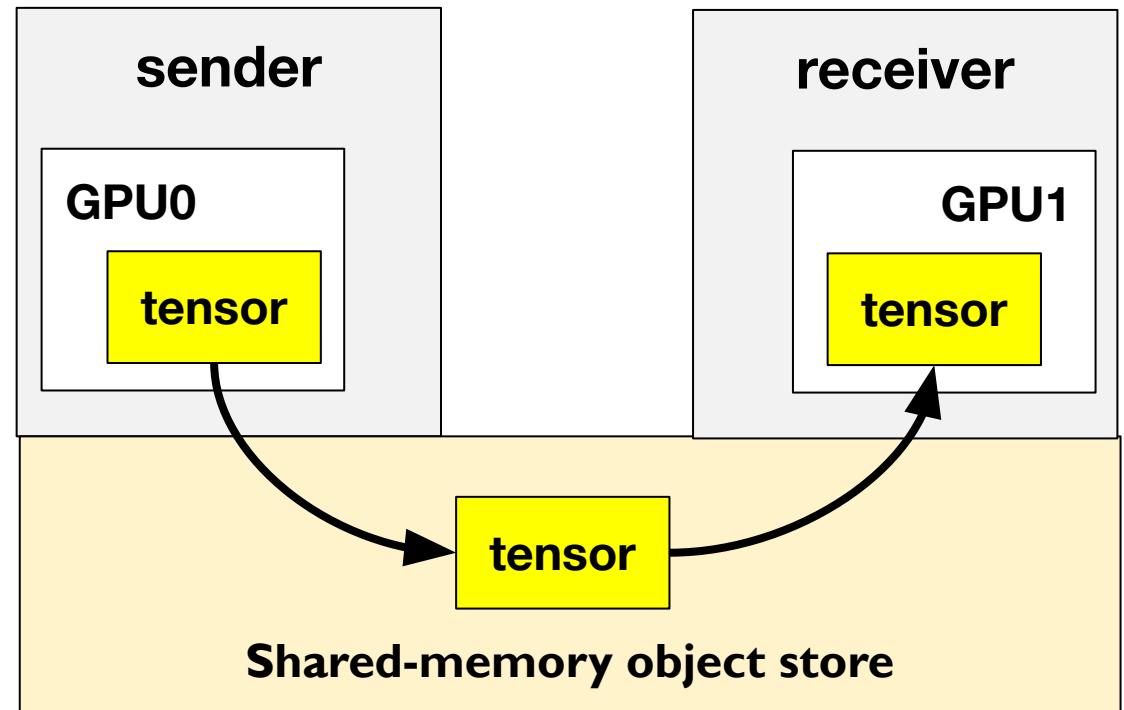
```
adag = dag.experimental_compile()
```



GPU-GPU transfers: default execution with Ray

```
@ray.remote(num_gpus=1)
class GPUSender:
    def send(self, shape):
        return torch.zeros(shape,
                           device="cuda")

@ray.remote(num_gpus=1)
class GPUReceiver:
    def recv(self, tensor: torch.Tensor):
        assert tensor.device.type == "cuda"
        return tensor.shape
```



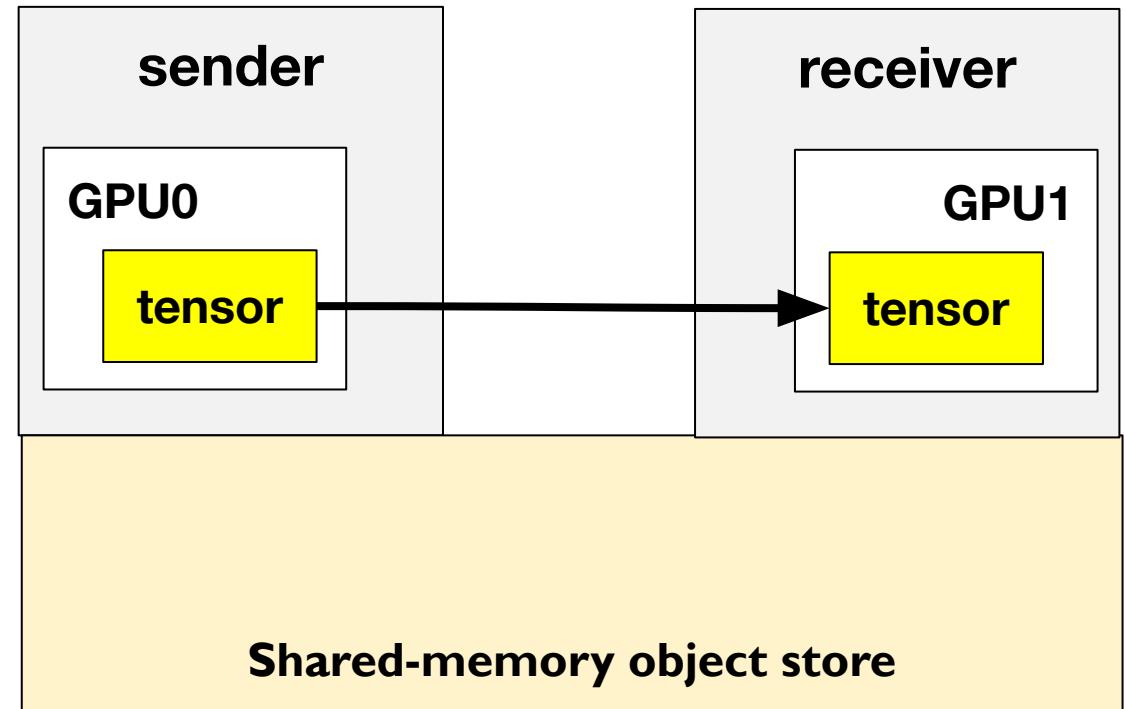
- **Must copy through host memory**
- **Naive implementation also requires multiple other copies**

GPU-GPU transfers with Ray compiled graphs

```
from ray.experimental.channel.torch_tensor_type
import TorchTensorType

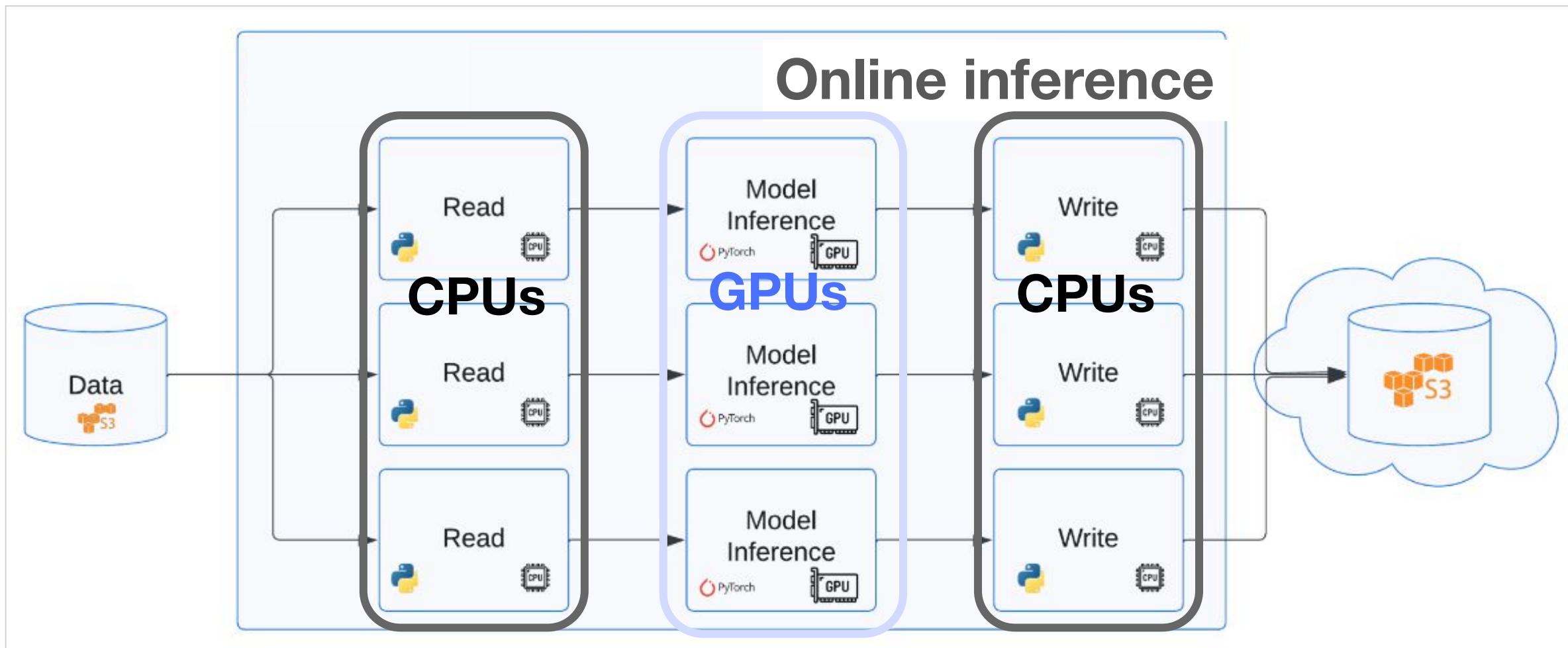
with ray.dag.InputNode() as inp:
    dag = sender.send.bind(inp)
    dag = dag.with_type_hint(TorchTensorType(
        transport="nccl",
    ))
    dag = receiver.recv.bind(dag)

adag = dag.experimental_compile()
adag.execute((100, ))
```



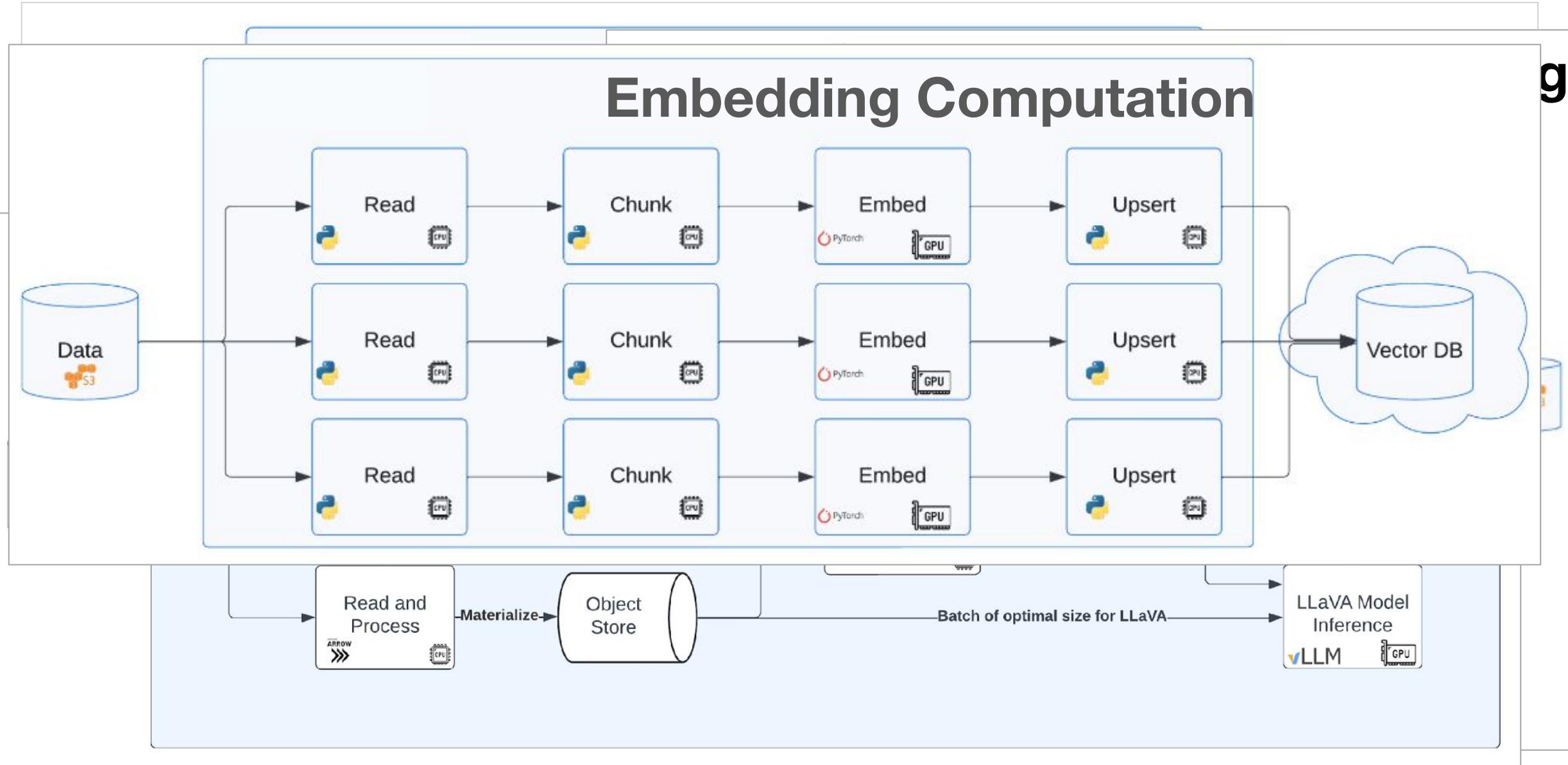
- **Schedule determined at compile time**
- **Future: collective ops, overlapping with compute**

Application examples

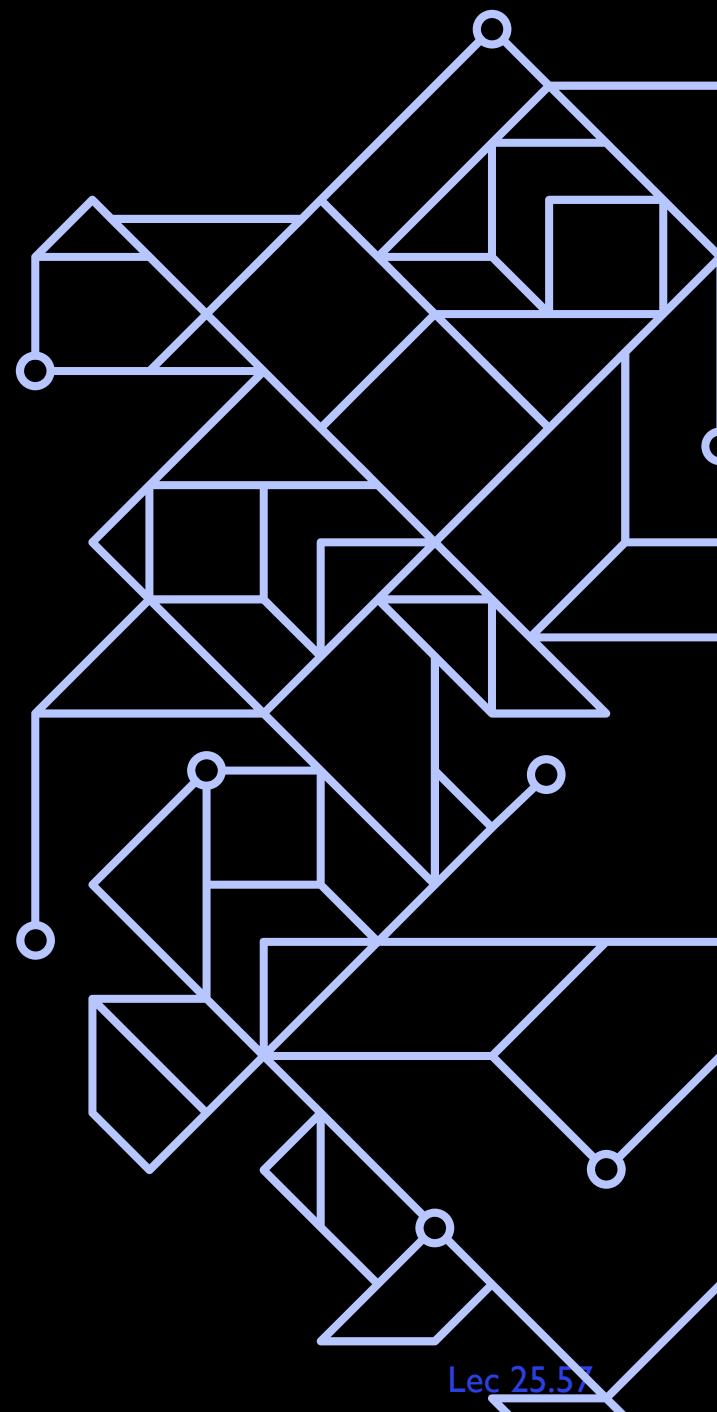


Exploiting CPU/GPU heterogeneity can reduce cost by 10x !

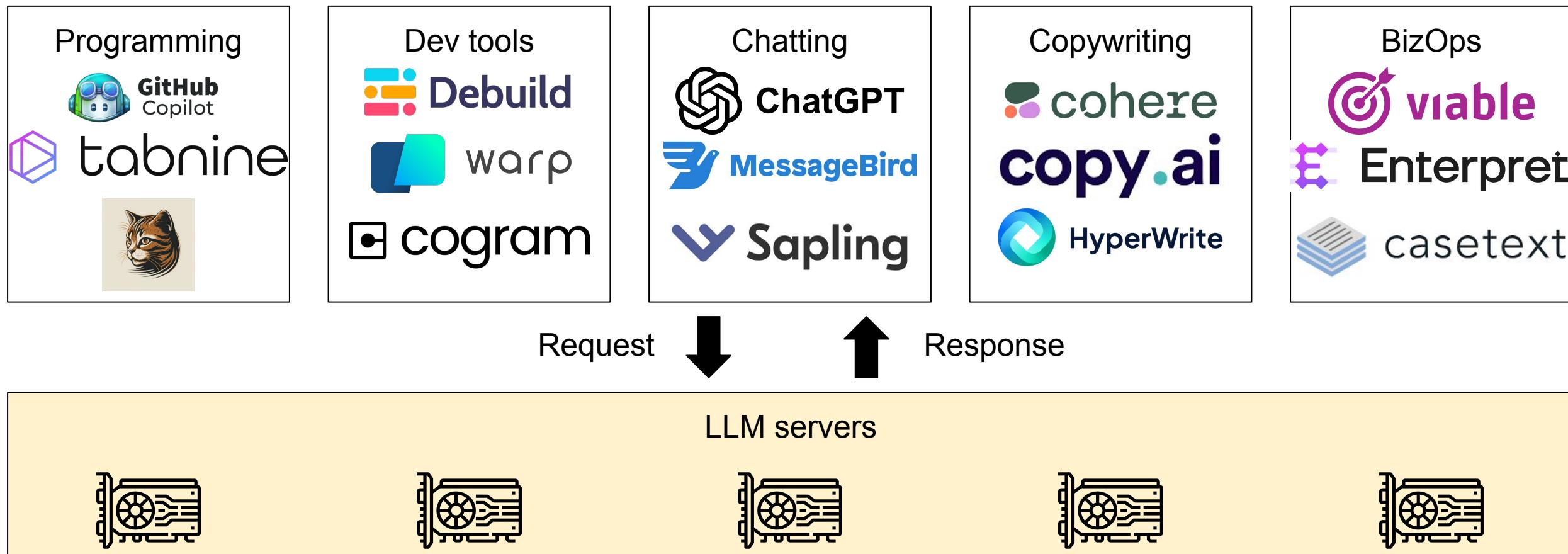
Application examples



vLLM (2023-)



Growing number of LLM-powered services



Problem: Serving LLMs very expensive

LLMs run on high-end GPUs such as NVIDIA A100 and H100

Each GPU can only serve a handful of requests/sec

- For LLaMA-13B, one A100 can process just a few requests/sec

Huge # of GPUs are required for production-scale LLM services



Paige Bailey 
@DynamicWebPaige

Inference on LLMs is slow (frustratingly high latency), expensive (multiple GPUs or TPUs), & engineering intensive (requires specialized skills to do it well)

...



r/LocalLLaMA • 8 days ago
by Financial_Stranger52

Join

...

Is local LLM cheaper than ChatGPT API?

ChatGPT api only costs 0.002 dollar for 1k token. I found that LLMs like llama output only 10-20 tokens per second, which is very slow. And such machines costs over 1 dollar per hour. It seems that using api is much cheaper. Based on these observations, it seems that utilizing the ChatGPT API might be a more affordable option.

May 2022 – OPT-175B Serving Demo

The screenshot shows the Alpa website interface. At the top, it says "Large Model for Everyone". Below that, a sub-headline reads "Alpa is a system for training and serving gigantic machine learning models. Alpa makes training and serving large models like GPT-3 simple, affordable, accessible to everyone." There are two buttons: "Star" with a count of 2,990 and "Try Live Generation (OPT-175B)". Another button says "Host Your Own Service (OPT, BLOOM, CodeGen)". A prominent section below is titled "Free, Unlimited OPT-175B Text Generation". It includes a warning: "Warning: This model might generate something offensive. No safety measures are in place as a free service." Below the warning are several buttons for different applications: "Fact", "Chatbot", "Airport Code", "Translation", "Cryptocurrency", "Code", and "Math". A large input field is labeled "Type the prompts here".

Our lab created a free inference service for OPT-175B, the largest open LLM at the time.

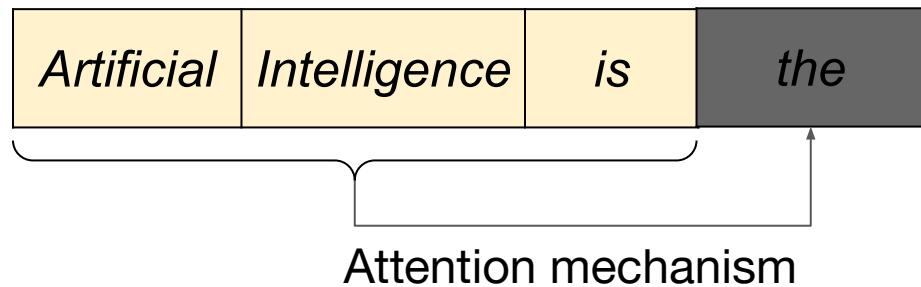
- 16 A100 GPUs per model instance

However, users experienced long queueing time, because of **low serving throughput**.

LLM query processing

Given a prompt:

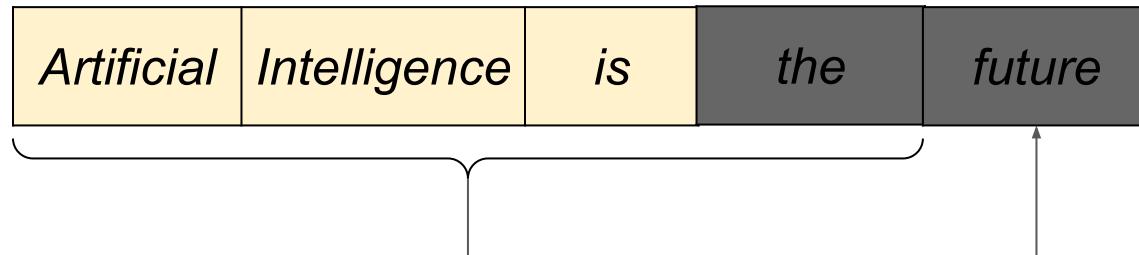
- Generate words (tokens) sequentially
- Each token is based on all previous tokens



LLM query processing

Given a prompt:

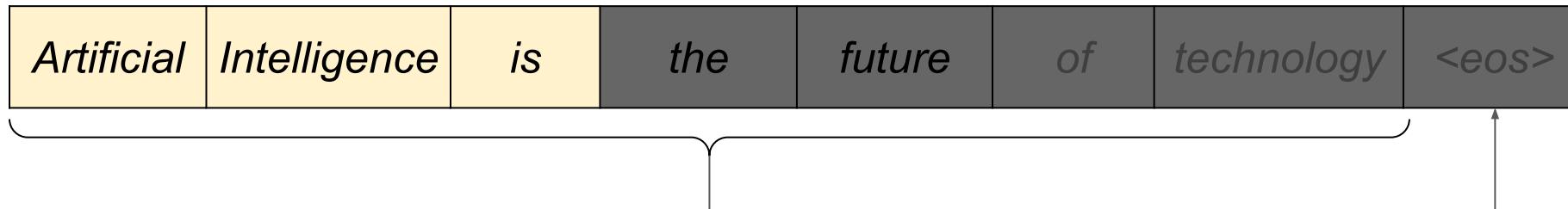
- Generate words (tokens) sequentially
- Each token is based on all previous tokens



LLM query processing

Given a prompt:

- Generate words (tokens) sequentially
- Each token is based on all previous tokens



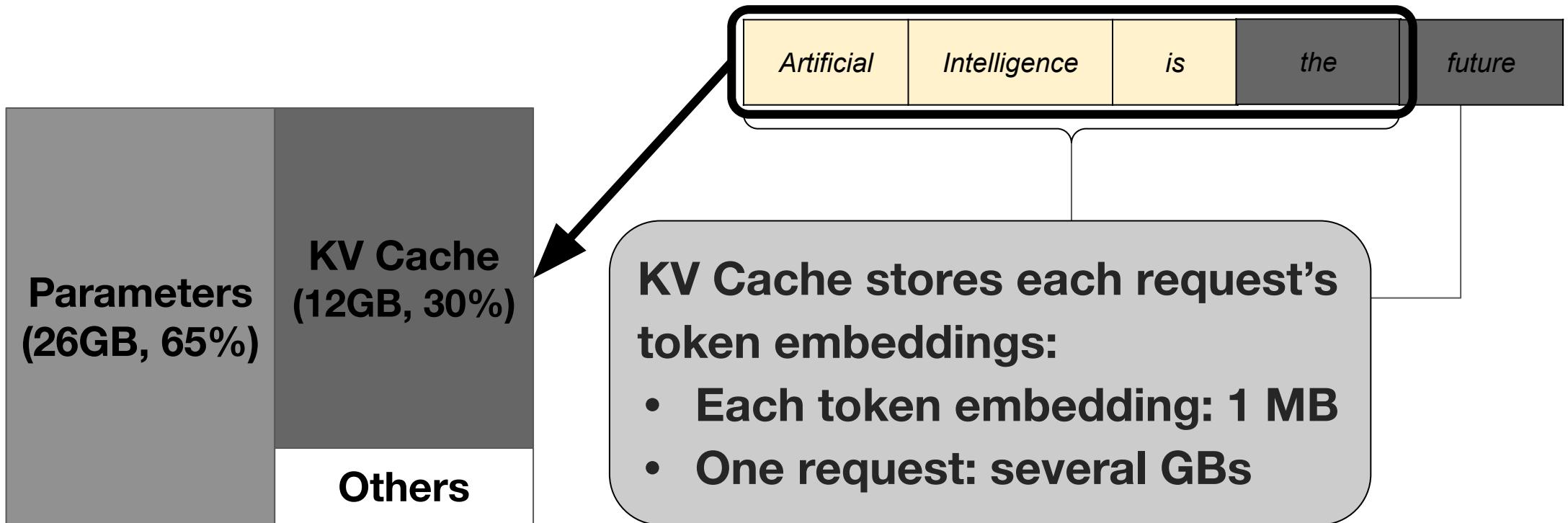
Challenge: low throughput high cost

Generating one token at a time doesn't use GPU parallelism

Solution: process multiple requests at the same time (batch)

Challenge: run out of memory!

Where does the memory go?



13B LLM on A100-40GB

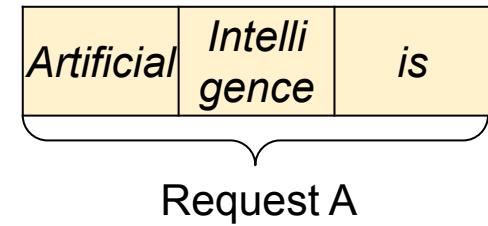
Memory management in previous systems

Stores a sequence's token states in contiguous memory space

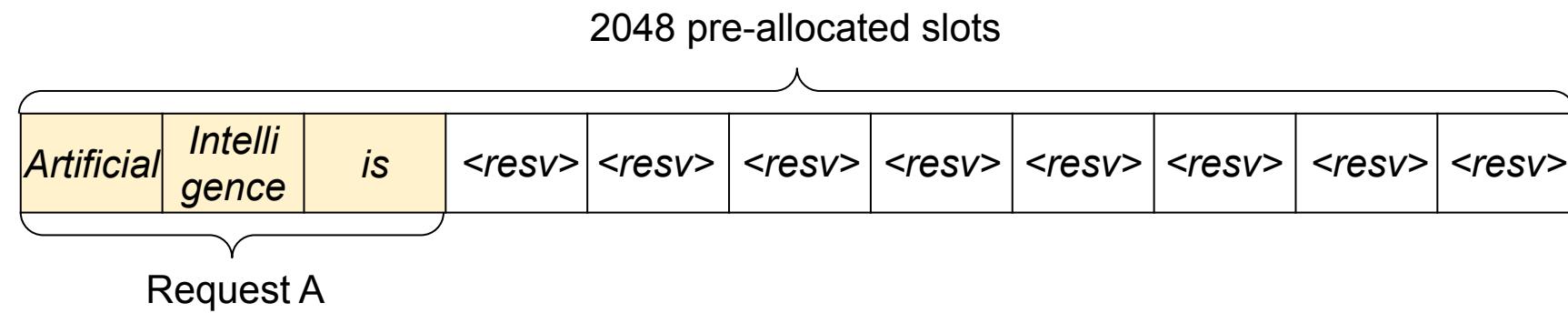
- Convention in traditional DNN workloads

Pre-allocate (reserve) the space for future token states, to handle unknown output lengths

Memory waste in KV Cache

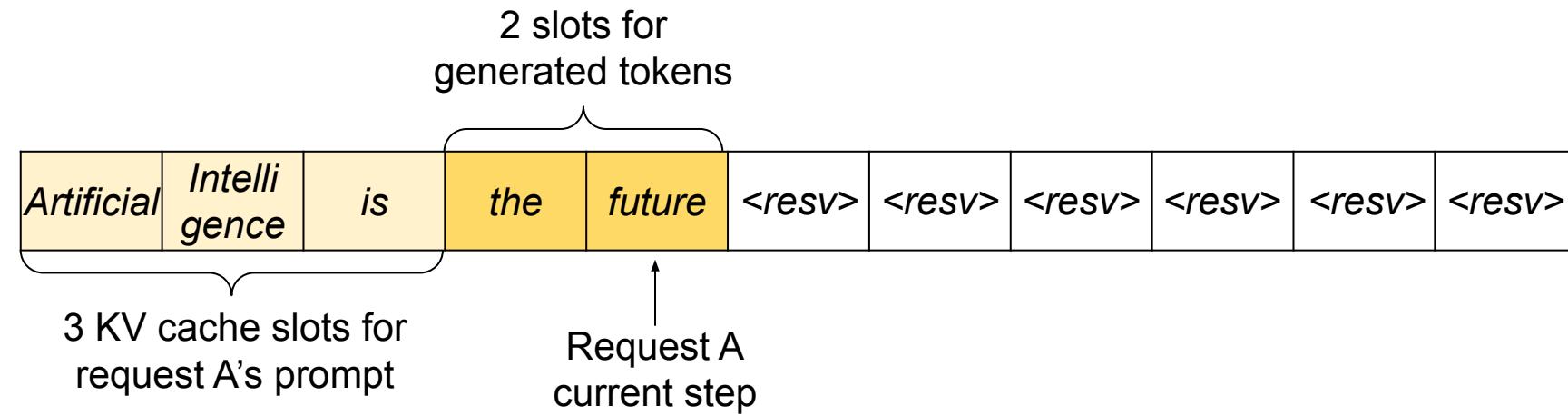


Memory waste in KV Cache



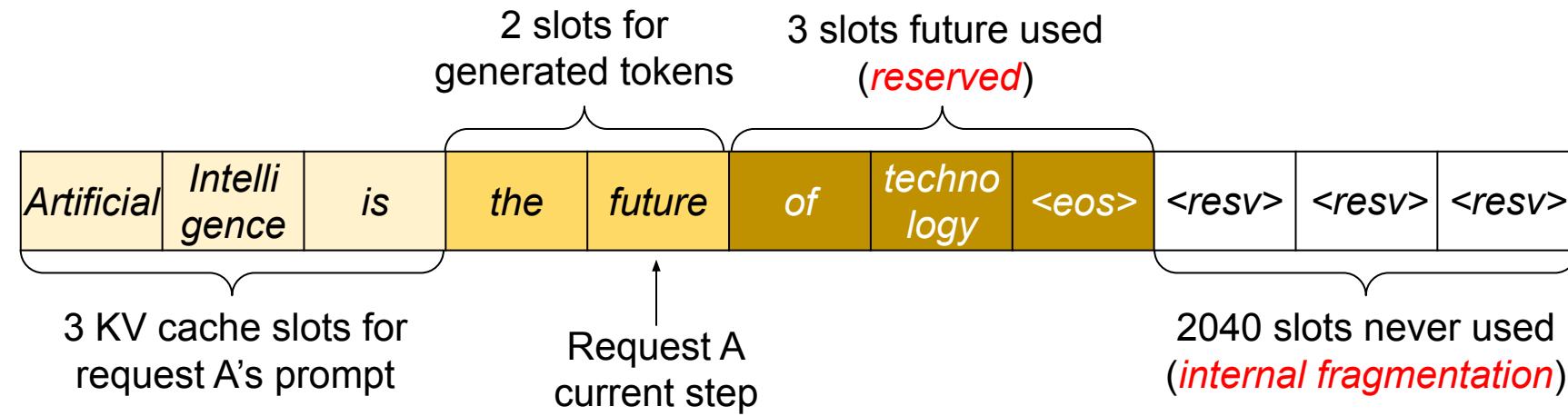
Internal fragmentation: over-allocated due to the unknown output length

Memory waste in KV Cache



Internal fragmentation: over-allocated due to the unknown output length

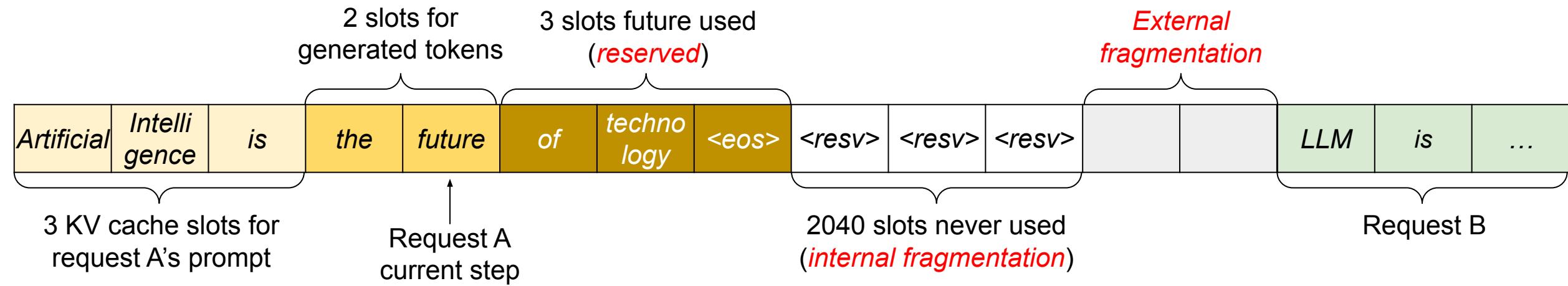
Memory waste in KV Cache



Internal fragmentation: over-allocated due to the unknown output length

Reservation: not used at the current step but used in the future (this fragmentation happens even if output length is known)

Memory waste in KV Cache



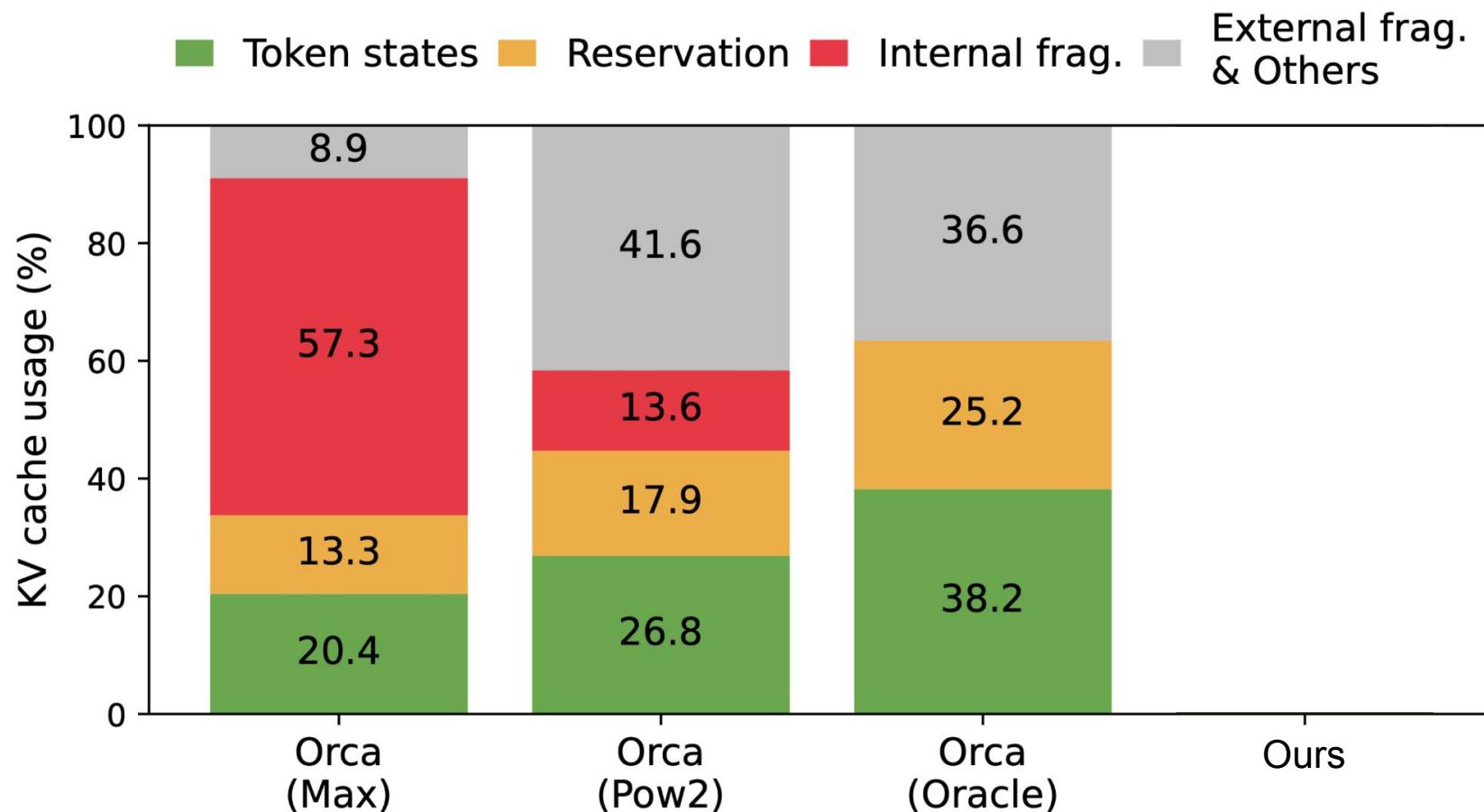
Internal fragmentation: over-allocated due to the unknown output length

Reservation: not used at the current step but used in the future (this fragmentation happens even if output length is known)

External fragmentation: due to different allocation lengths

Significant memory waste

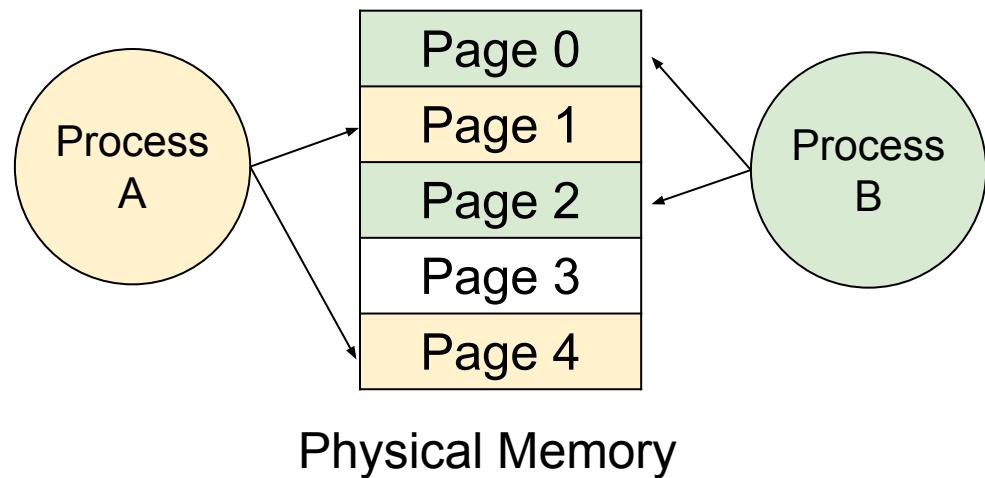
Only 20–40% of KV cache is utilized to store token states



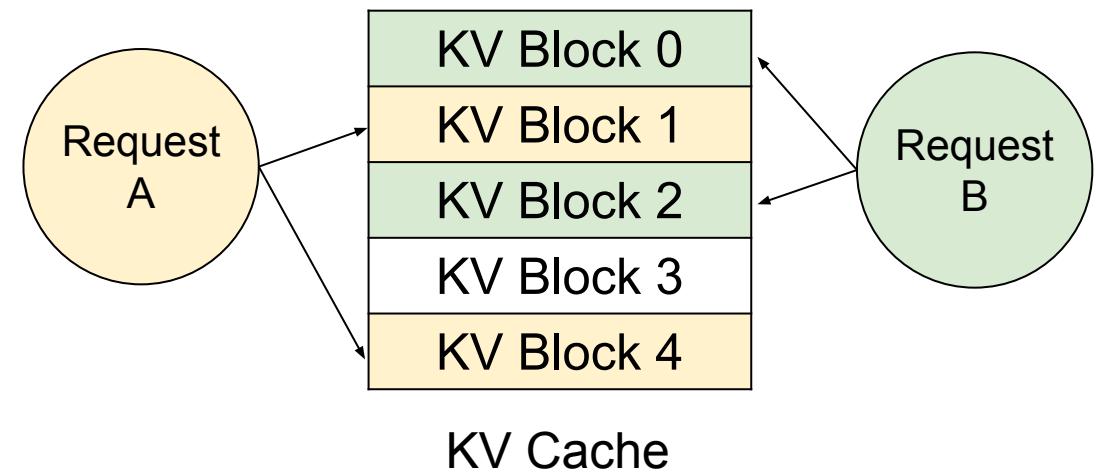
vLLM: Efficient memory management for LLM inference

Inspired by **virtual memory** and **paging**

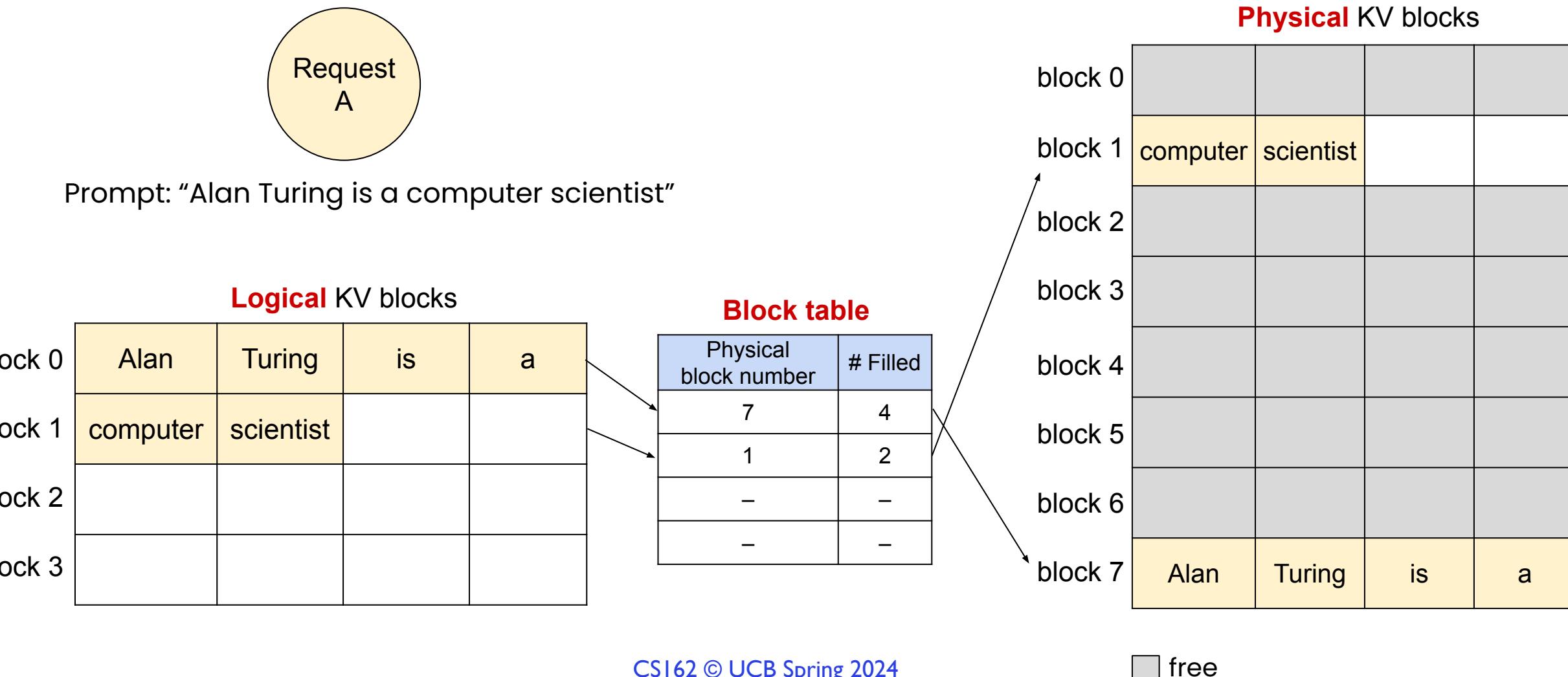
Memory management in OS



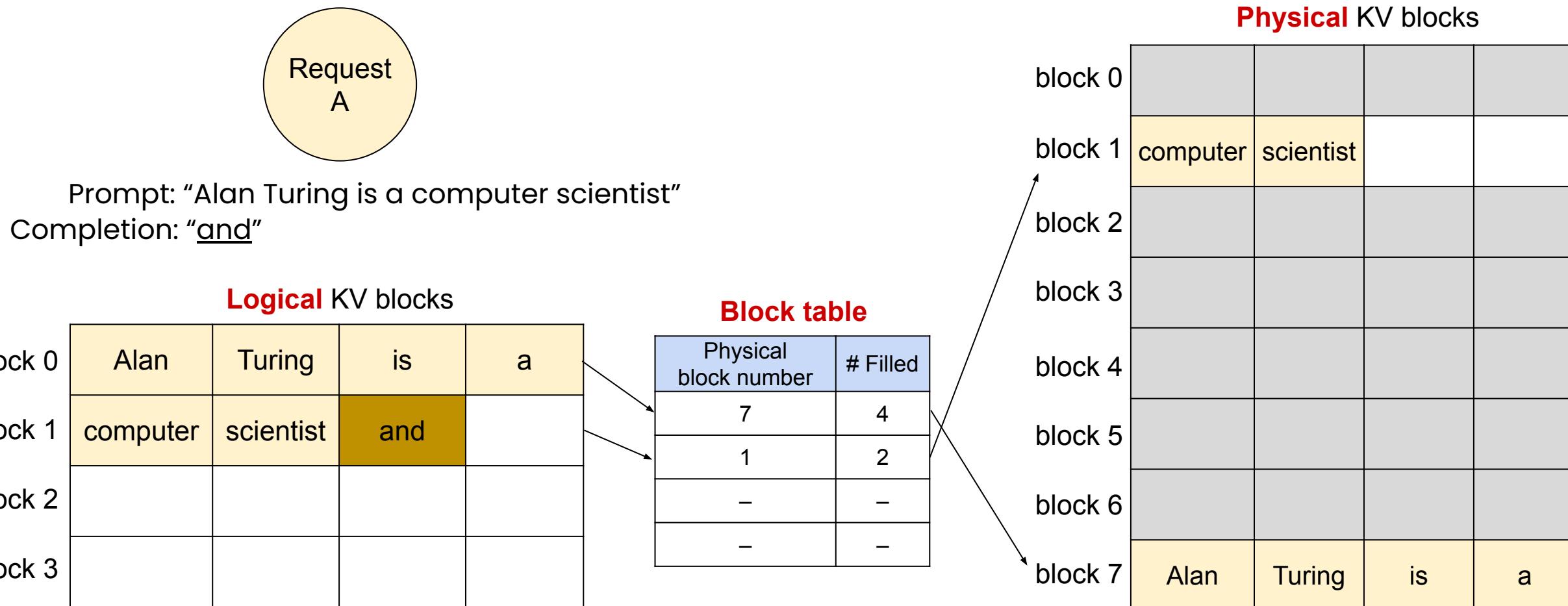
Memory management in vLLM



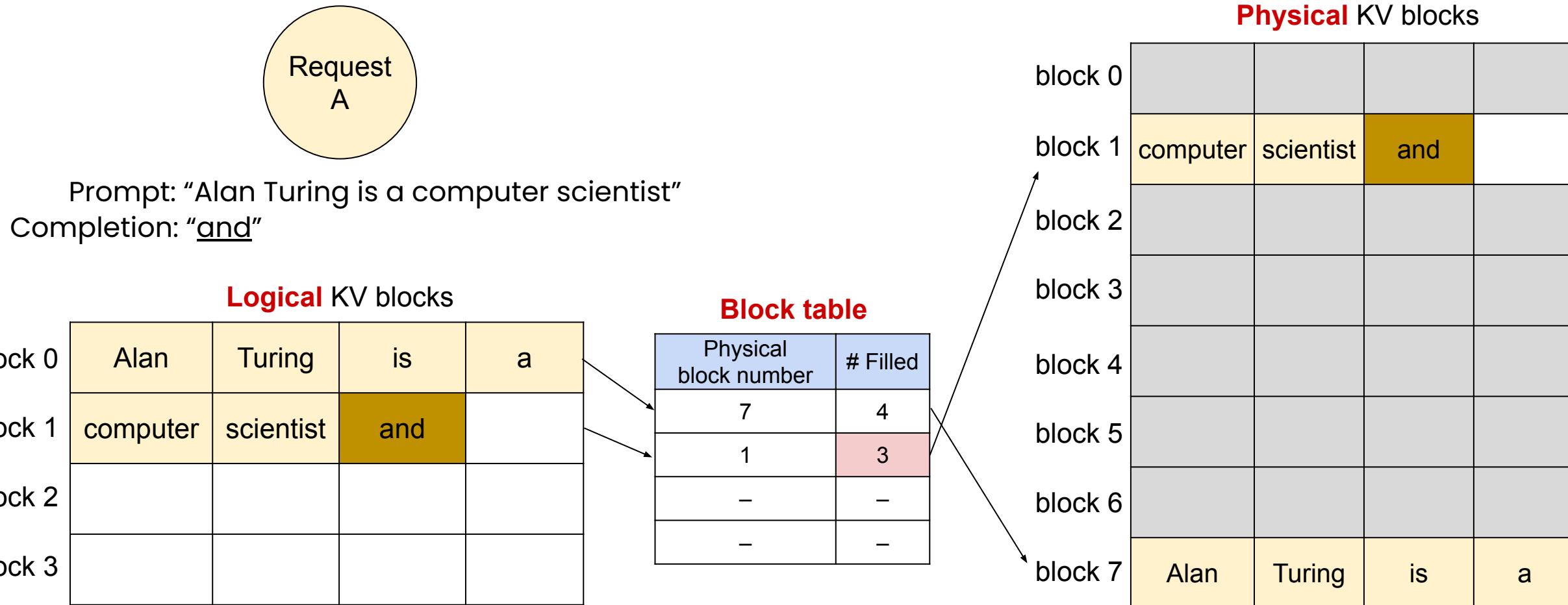
PagedAttention



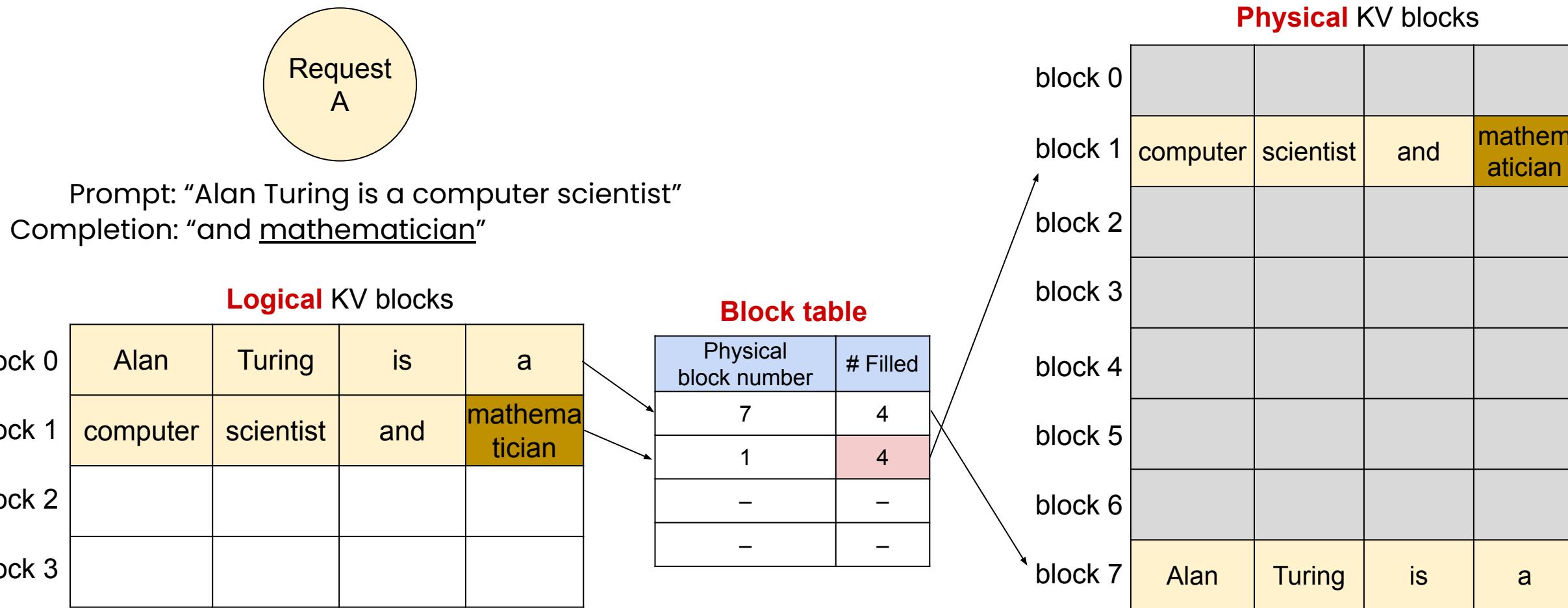
PagedAttention



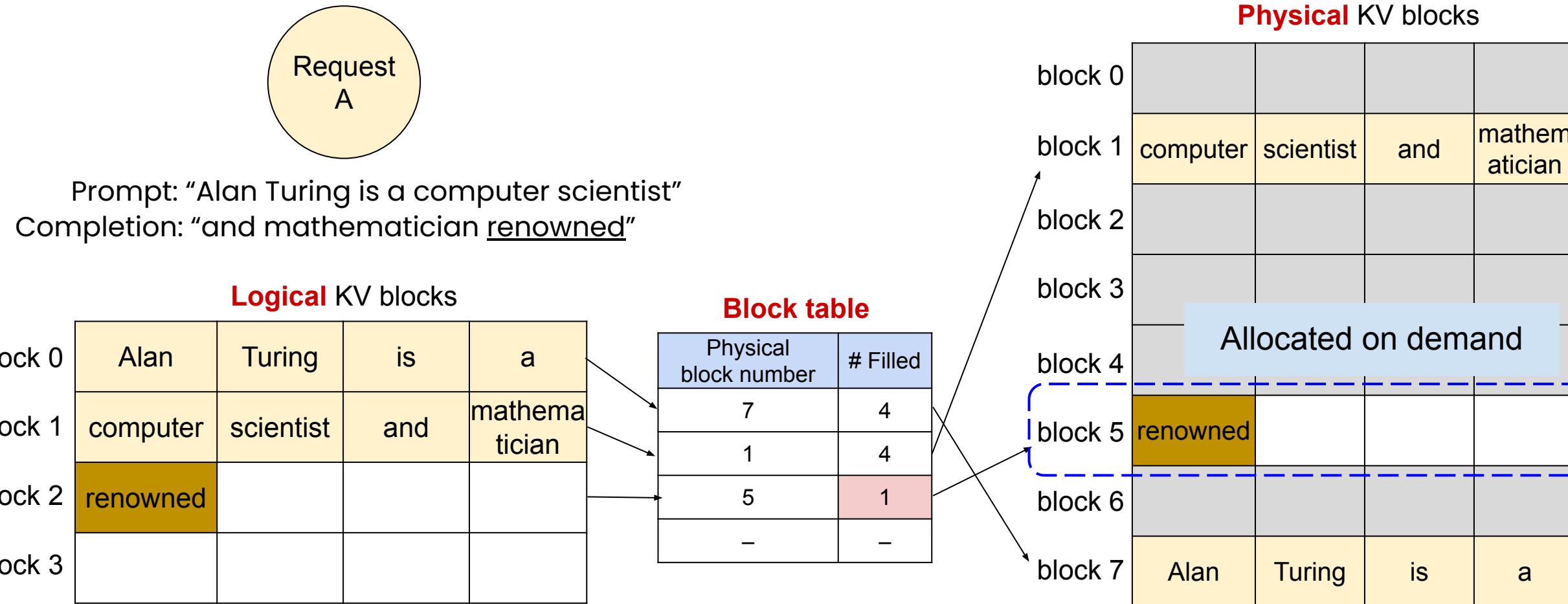
PagedAttention



PagedAttention



PagedAttention

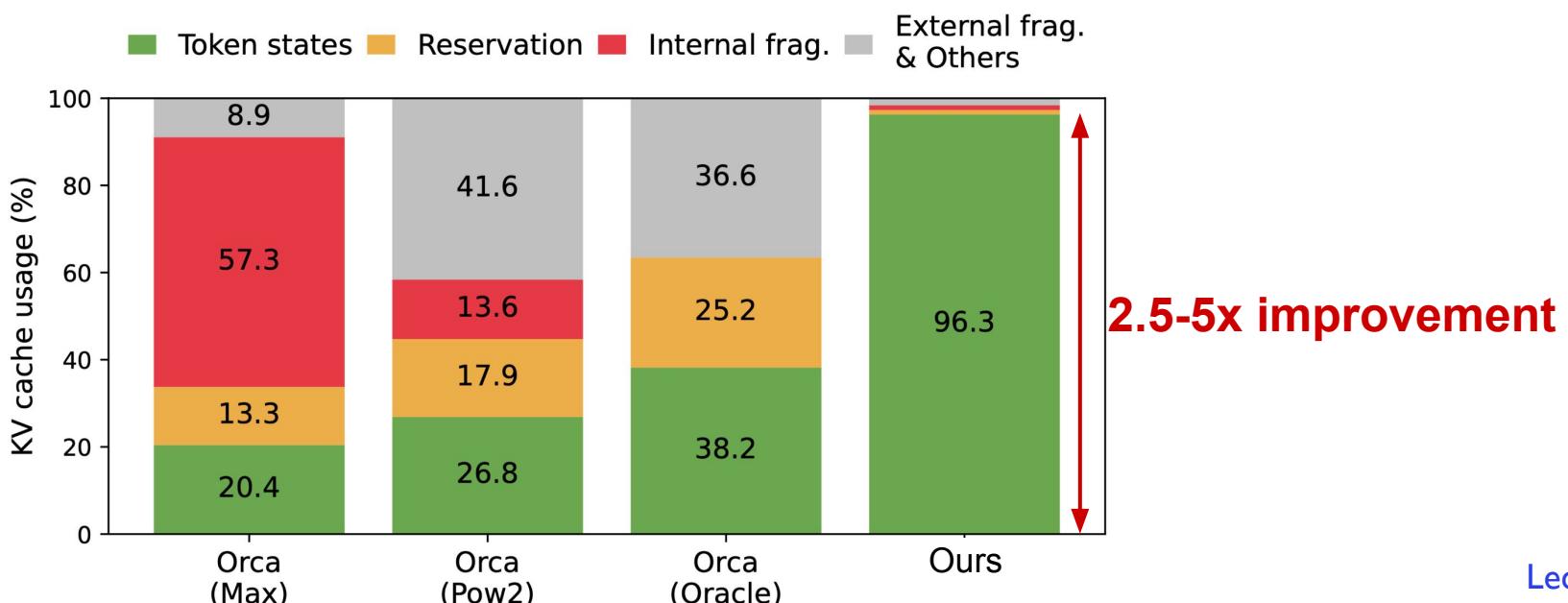


Memory efficiency of PagedAttention

- Minimal internal fragmentation
 - Only happens at the last block of a sequence
 - $\# \text{ wasted tokens} / \text{seq} < \text{block size}$
 - Sequence: $O(100) - O(1000)$ tokens
 - Block size: 16 or 32 tokens
- No external fragmentation

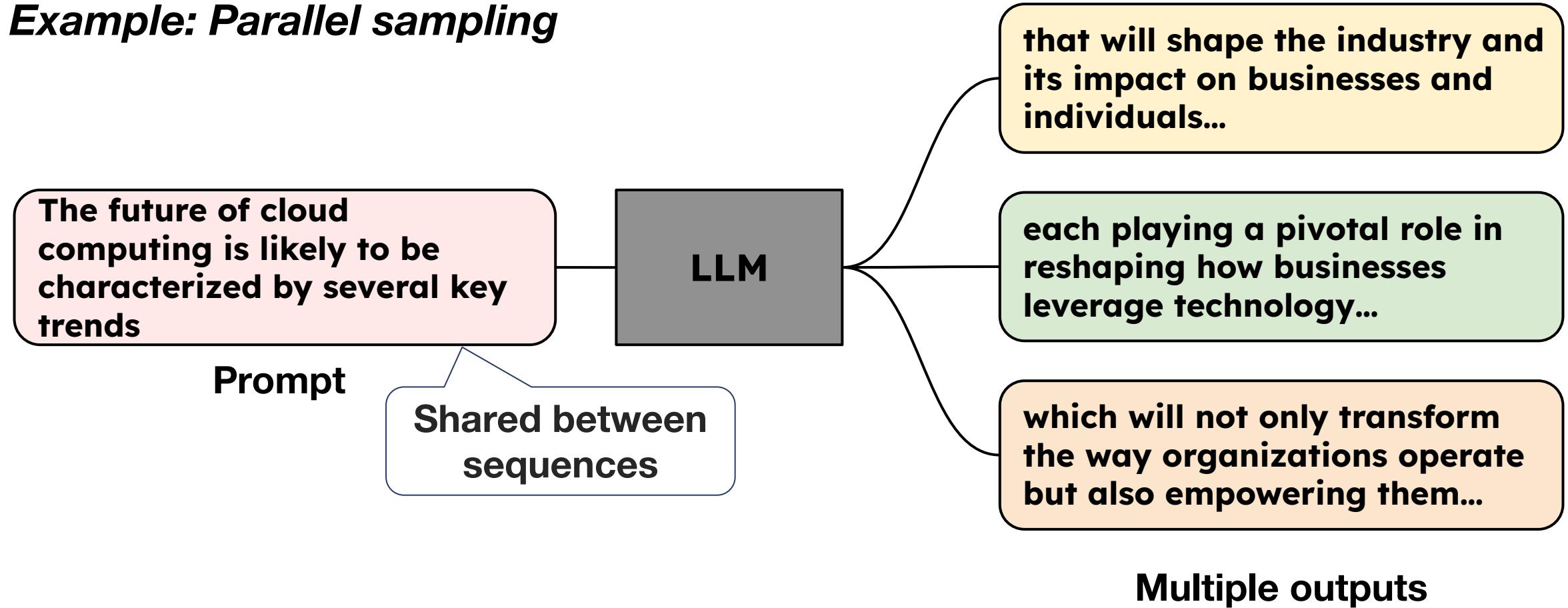
Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Internal fragmentation

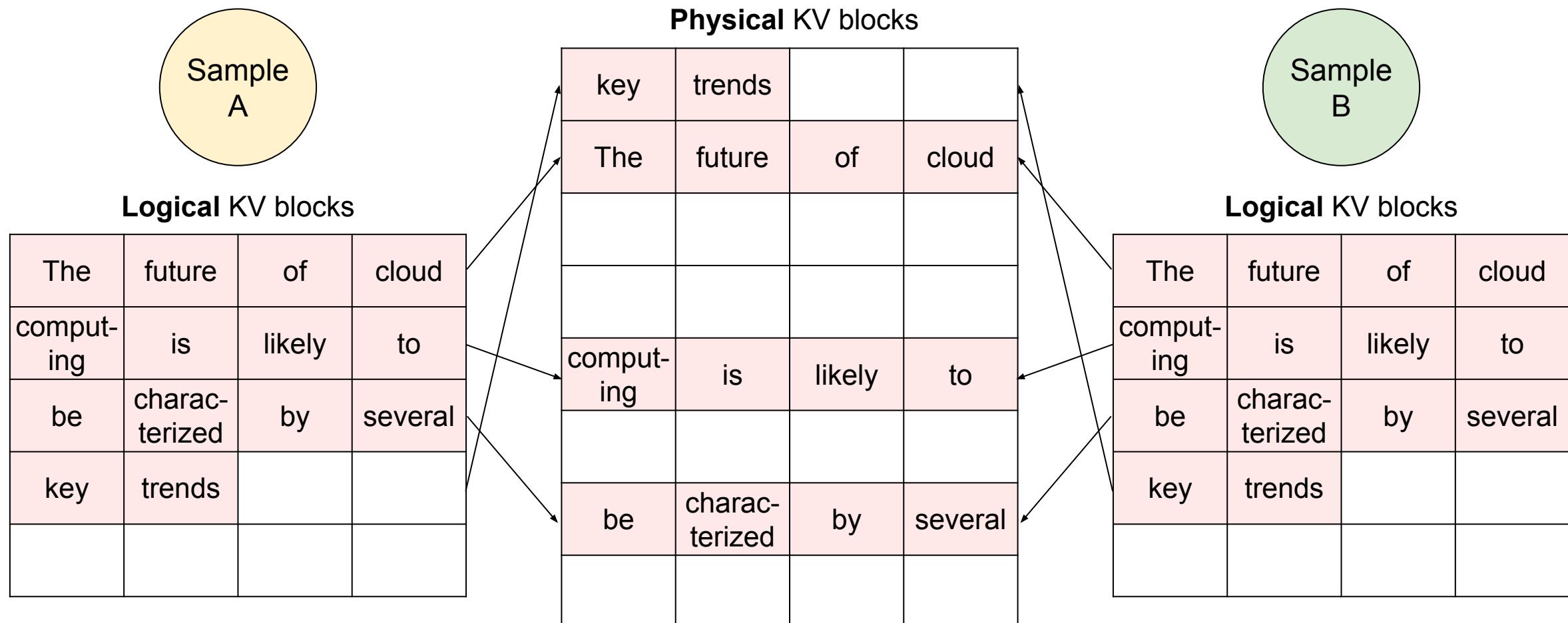


Paging enables sharing

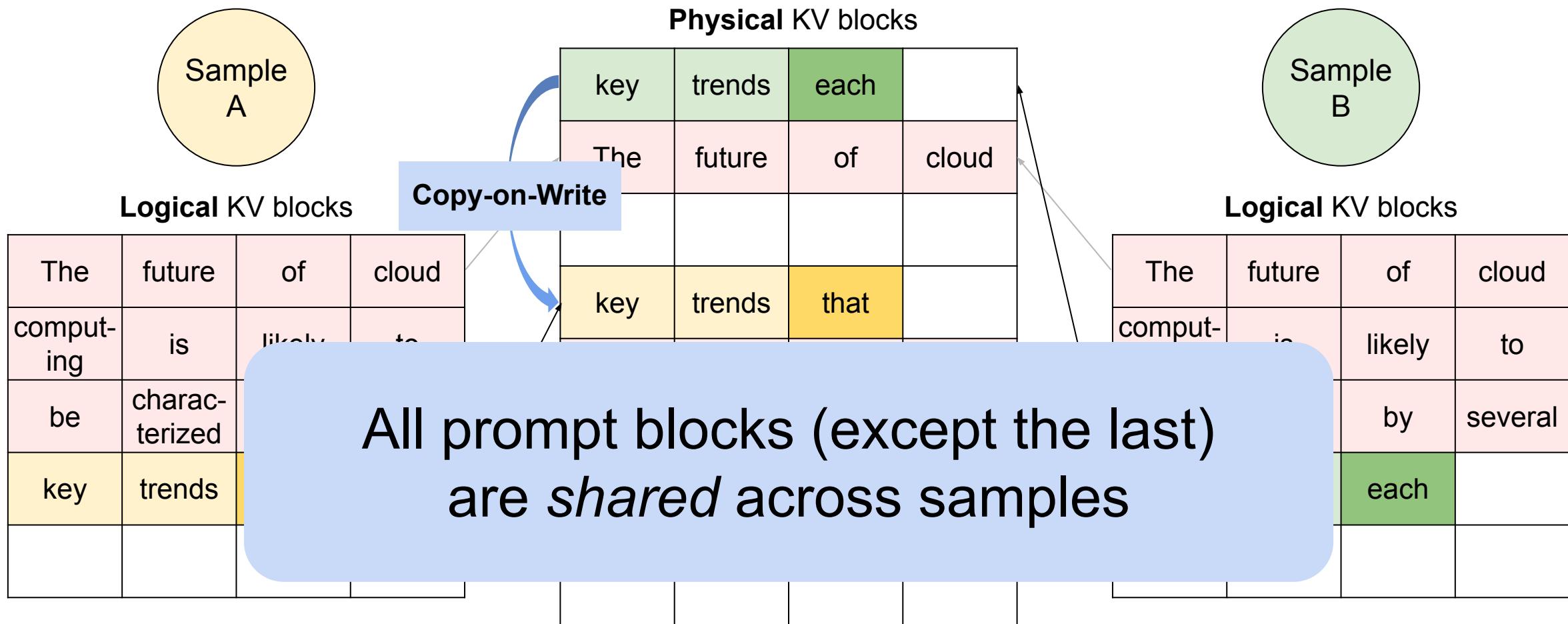
Example: Parallel sampling



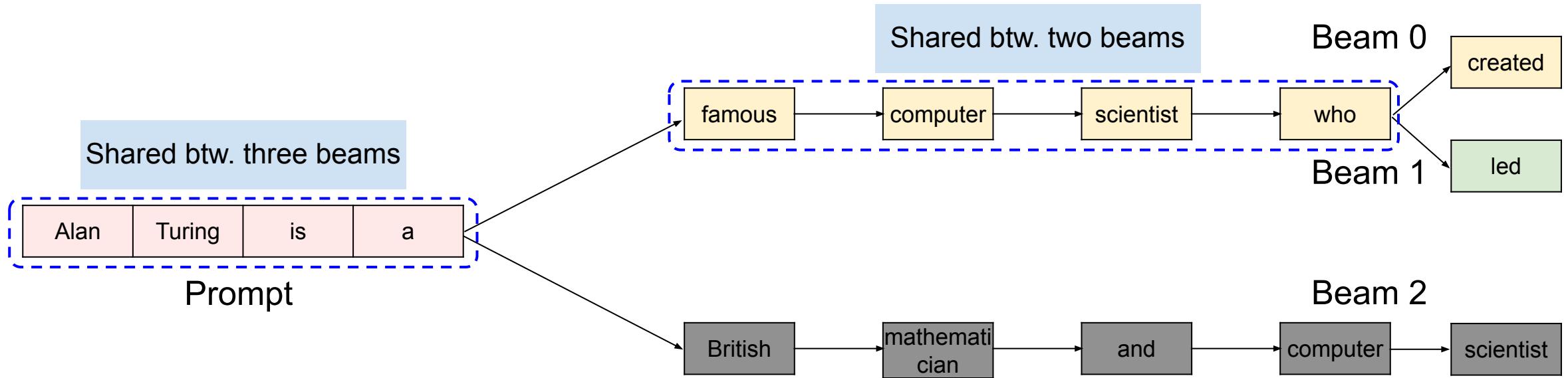
Sharing KV blocks



Sharing KV blocks

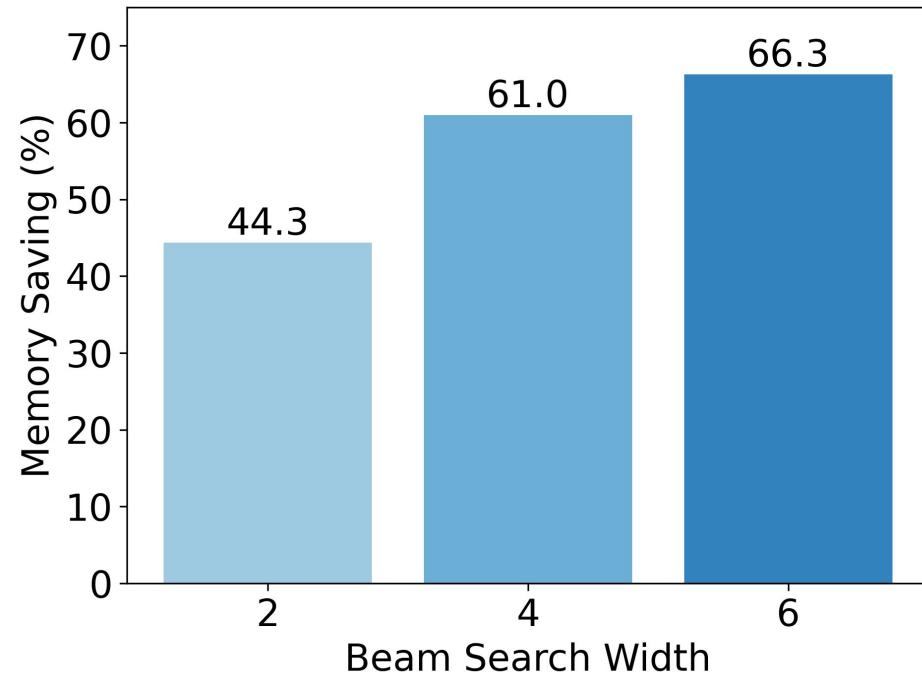
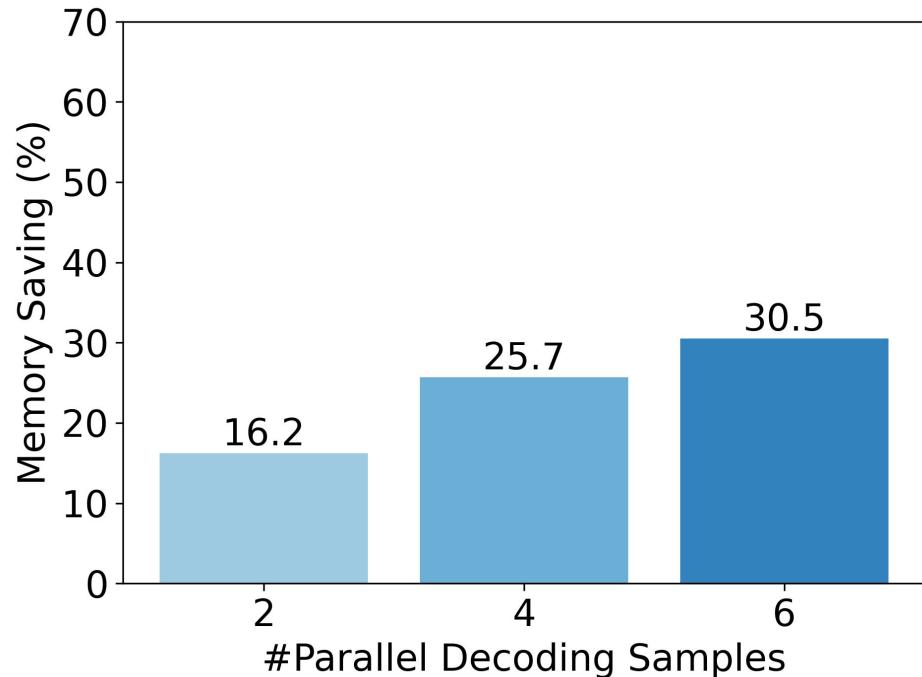


More complex sharing: beam search



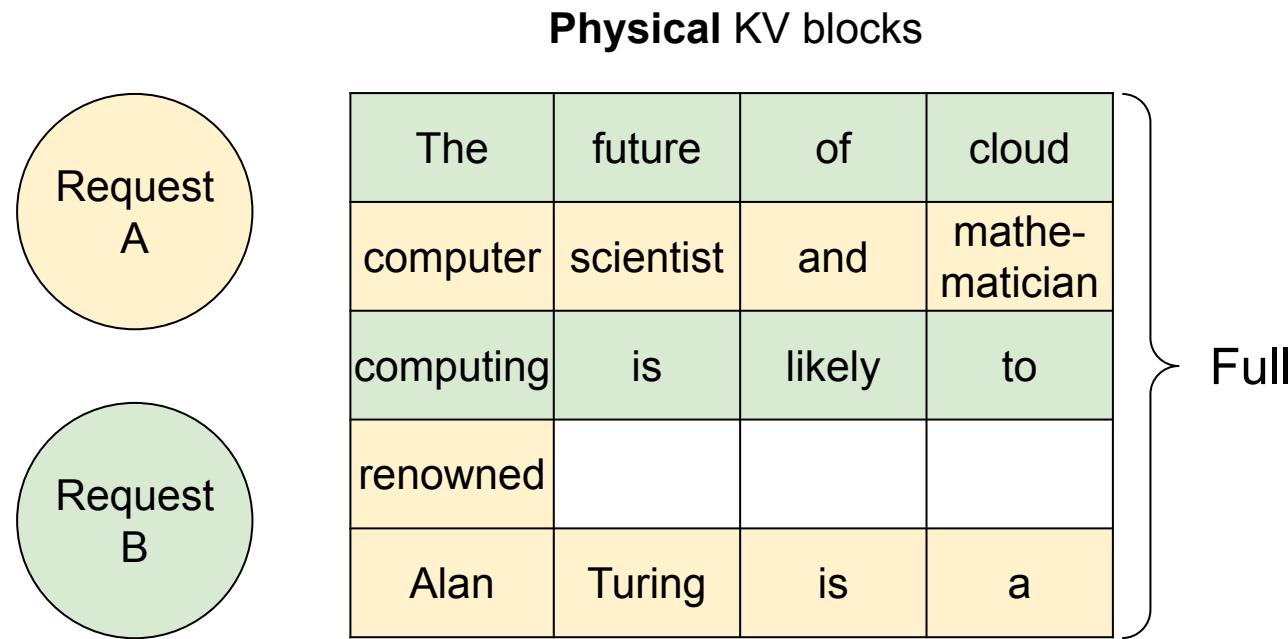
- Similar to process tree (fork & kill)
- Efficiently supported by paged attention and copy-on-write mechanism

Memory saving via sharing

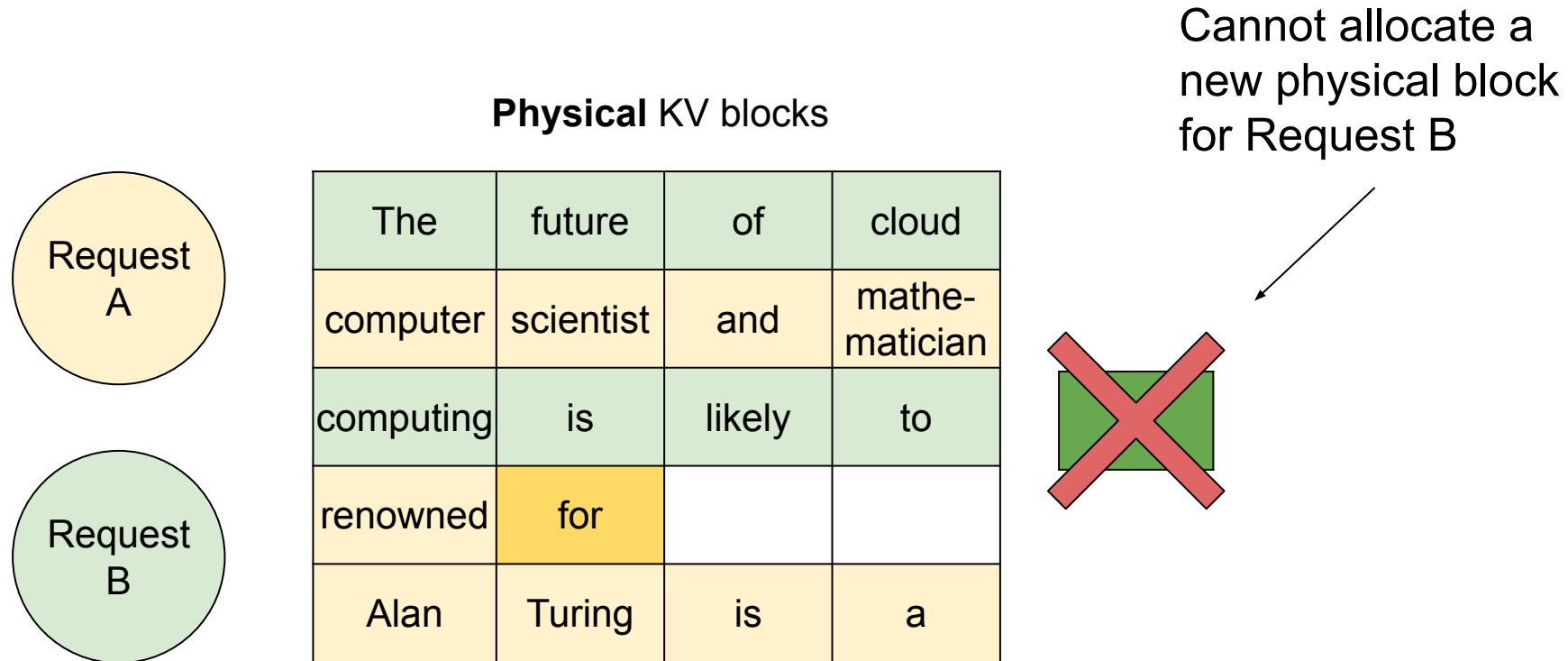


Percentage = (#blocks saved by sharing) / (#total blocks without sharing)
OPT-13B on 1x A100-40G with ShareGPT dataset

Out of KV Cache Memory



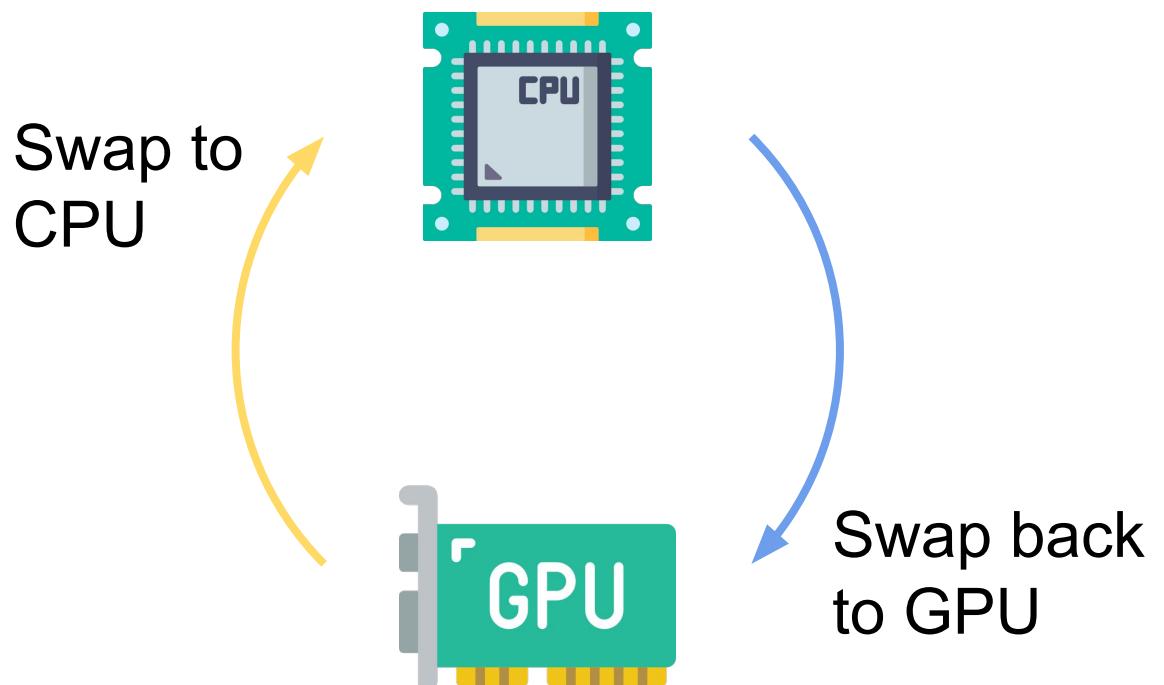
Out of KV Cache Memory



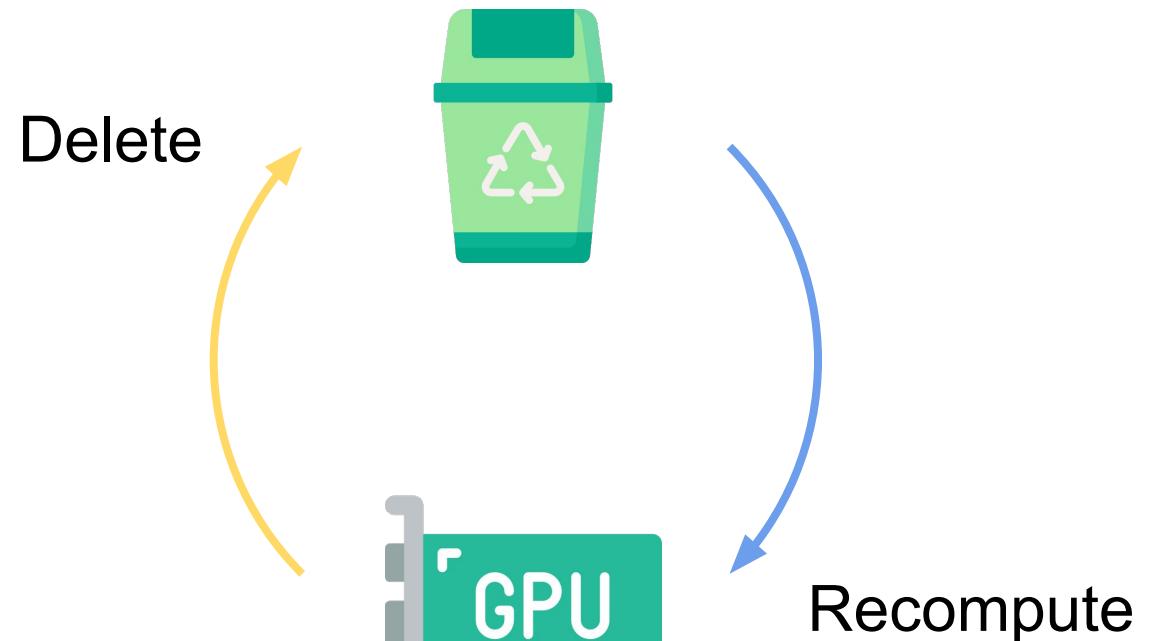
Request Preemption & Recovery

Goal: Free some requests' KV cache to let others run first.

Option 1: Swapping



Option 2: Recomputation



PagedAttention vs operating system virtual memory

Analogies

OS pages \leftrightarrow KV blocks

- Reduce memory fragmentation

Shared pages across processes

\leftrightarrow Shared KV blocks across samples

- Reduce memory waste via sharing

Differences

Eviction

- Request-level vs page-level eviction

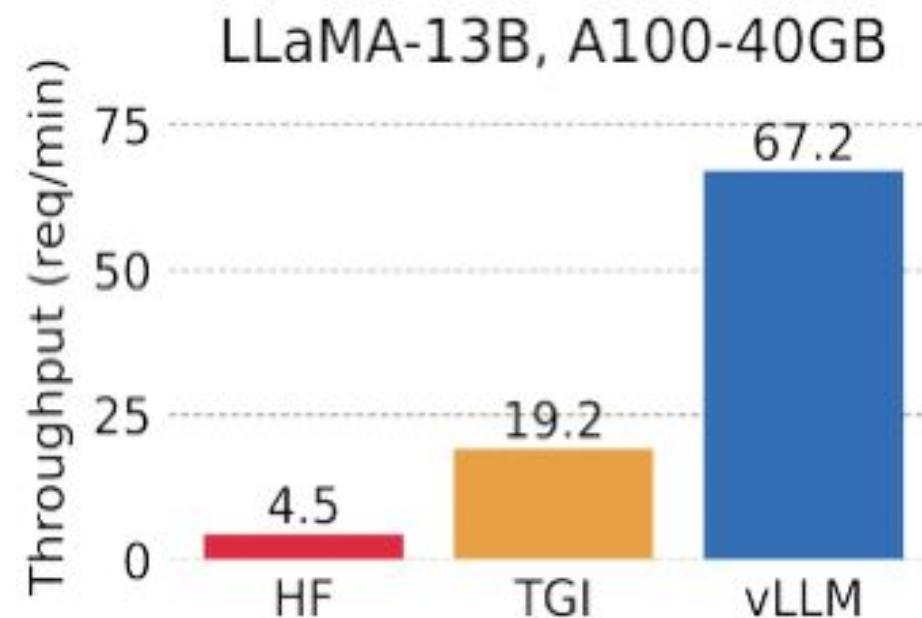
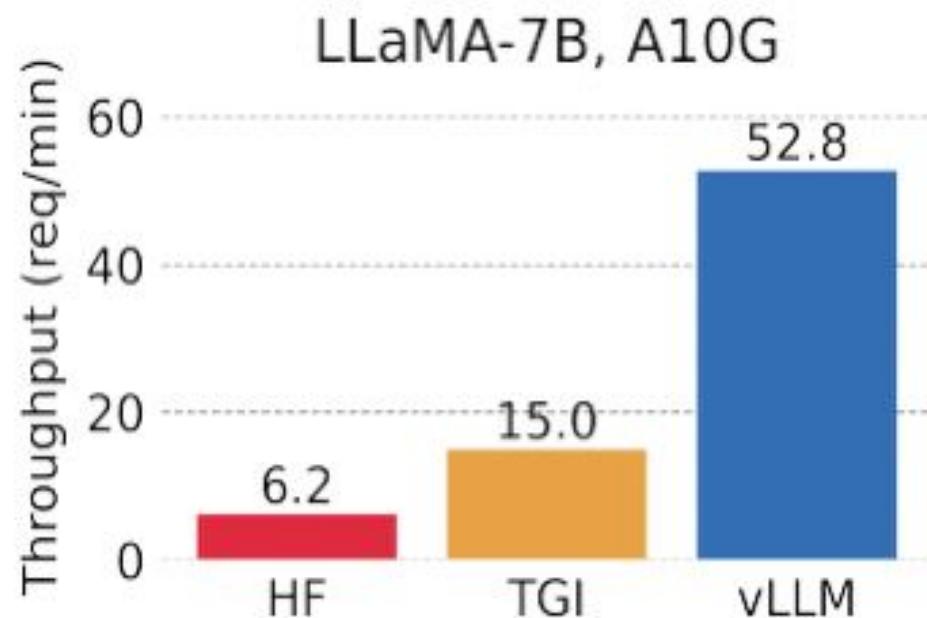
Loading blocks/pages back to memory

- Recompute blocks vs saving them to CPU RAM and bringing them back

Throughput benefit of vLLM

Up to **24x** higher throughput than HuggingFace (HF)

Up to **3.5x** higher throughput than Text Generation Inference (TGI)



Serving throughput when each request asks for 3 output completions.

vLLM Adoption

Contributors 714

Star 31.2k

Open-Source Projects



Im-sys/FastChat



OPENCHAT

Ai2 allenai/open-instruct

SkyPilot



WizardLM

...

Companies



anyscale



Uber



databricks



Google Cloud



Meta



Modal



Replicate

scale



Lepton AI

...

Accelerator Support



AMD

intel

aws Inferentia

Google TPU

PagedAttention is becoming an industry standard

developers to integrate new AI building blocks into new and innovative products.

Fast, Affordable LLM Inference with Fireworks

Efficient Inference

Efficient inference of LLMs is an active area of research, but we are industry veterans from the PyTorch team specializing in performance optimization. We use model optimizations including [multi/group query attention](#) optimizations, sharding, quantization, kernel optimizations, CUDA graphs, and custom cross-GPU communication primitives. At the service level, we employ continuous batching, [paged attention](#), prefill disaggregation, and pipelining to maximize throughput and reduce latency. We carefully tune deployment parameters including the level of parallelism and hardware choice for each model.

Fireworks.ai

- Serve the most popular Large Language Models with a simple launcher
- Tensor Parallelism for faster inference on multiple GPUs
- Token streaming using Server-Sent Events (SSE)
- [Continuous batching of incoming requests](#) for increased total throughput
- Optimized transformers code for inference using [flash-attention](#) and [Paged Attention](#) on the most popular architectures
- Quantization with [bitsandbytes](#) and [GPT-Q](#)
- [Safetensors](#) weight loading
- Watermarking with [A Watermark for Large Language Models](#)
- Logits warper (temperature scaling, top-p, top-k, repetition penalty, more details see [transformers.LogitsProcessor](#))
- Stop sequences
- Log probabilities
- Production ready (distributed tracing with Open Telemetry, Prometheus metrics)

HuggingFace TGI
CS162 © UCB Spring 2024

requiring deep knowledge of C++ or NVIDIA CUDA. TensorRT-LLM improves ease of use and extensibility through an open-source modular Python API for defining, optimizing, and executing new architectures and enhancements as LLMs evolve, and can be customized easily.

For example, MosaicML has added specific features that it needs on top of TensorRT-LLM seamlessly and integrated them into their inference serving. Naveen Rao, vice president of engineering at Databricks notes that "it has been an absolute breeze."

"TensorRT-LLM is easy to use, feature-packed with streaming of tokens, in-flight batching, [paged-attention](#), quantization, and more, and is efficient," Rao said. "It delivers state-of-the-art performance for LLM serving using NVIDIA GPUs and allows us to pass on the cost savings to our customers."

Performance comparison

Summarizing articles is just one of the many applications of

NVIDIA TRT-LLM

Summary

The open-source is the future of AI/LLM stack

Two projects covered today (among many):

- . Ray: scale and productize AI workloads
- . vLLM: high throughput LLM inference engine

Just scratching the surface: many more challenges at the intersection between systems & AI/LLMs