Queen Mary
University of London

Global Software Technology Summit 2023
Explore New Technologies, Spark New Ideas

Scalable and Sustainable Data-Intensive Systems

Bo Zhao
Assistant Professor, Queen Mary University of London
https://zbjob.github.io/

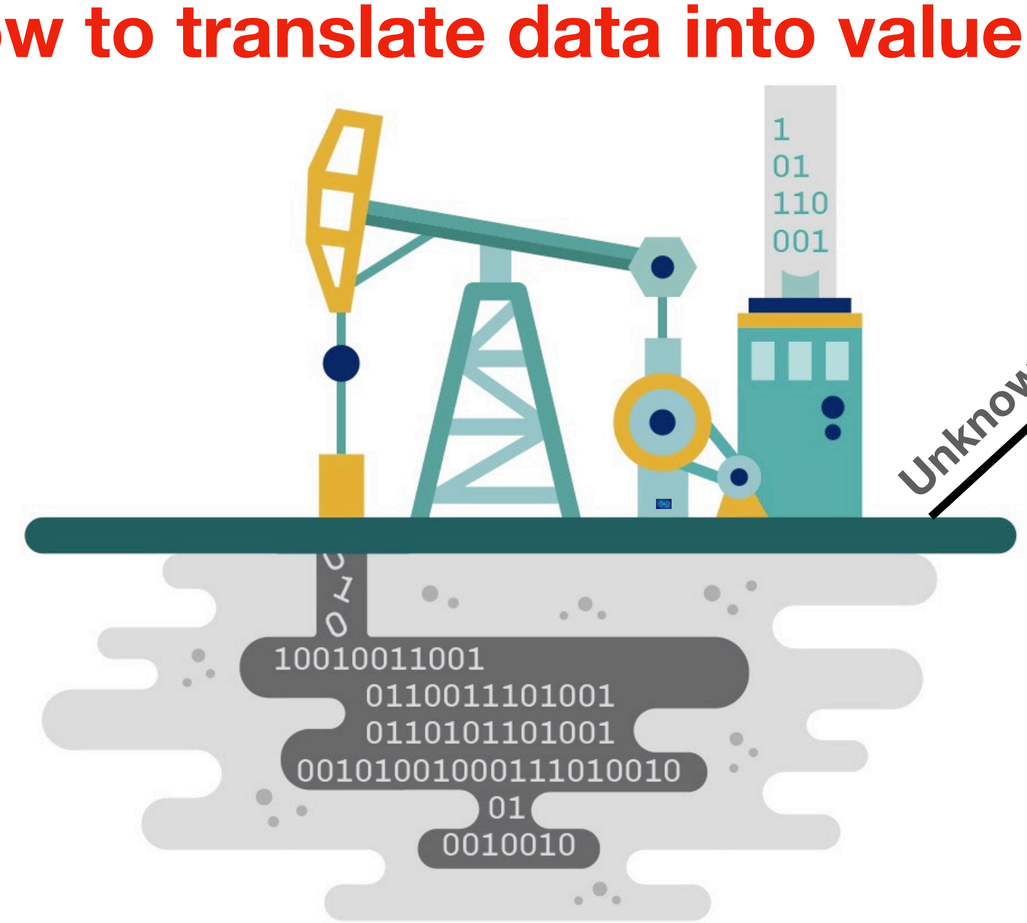
Data: the world's most valuable resource

Data: the world's most valuable resource

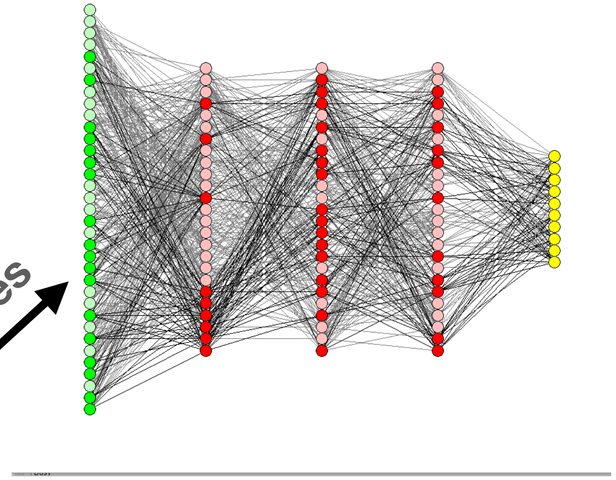
How to translate data into value ?

Data: the world's most valuable resource

How to translate data into value ?



Unknown rules

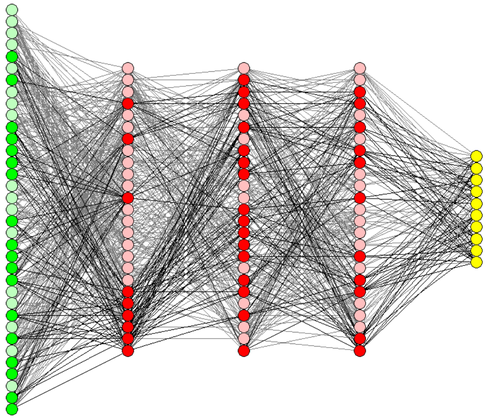


Train machine learning models for decision making

Data: the world's most valuable resource

How to translate data into value ?

Unknown rules



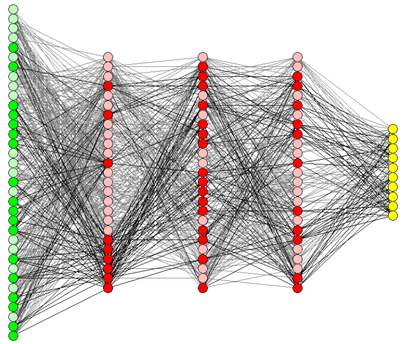
Train machine learning models for decision making

Known rules

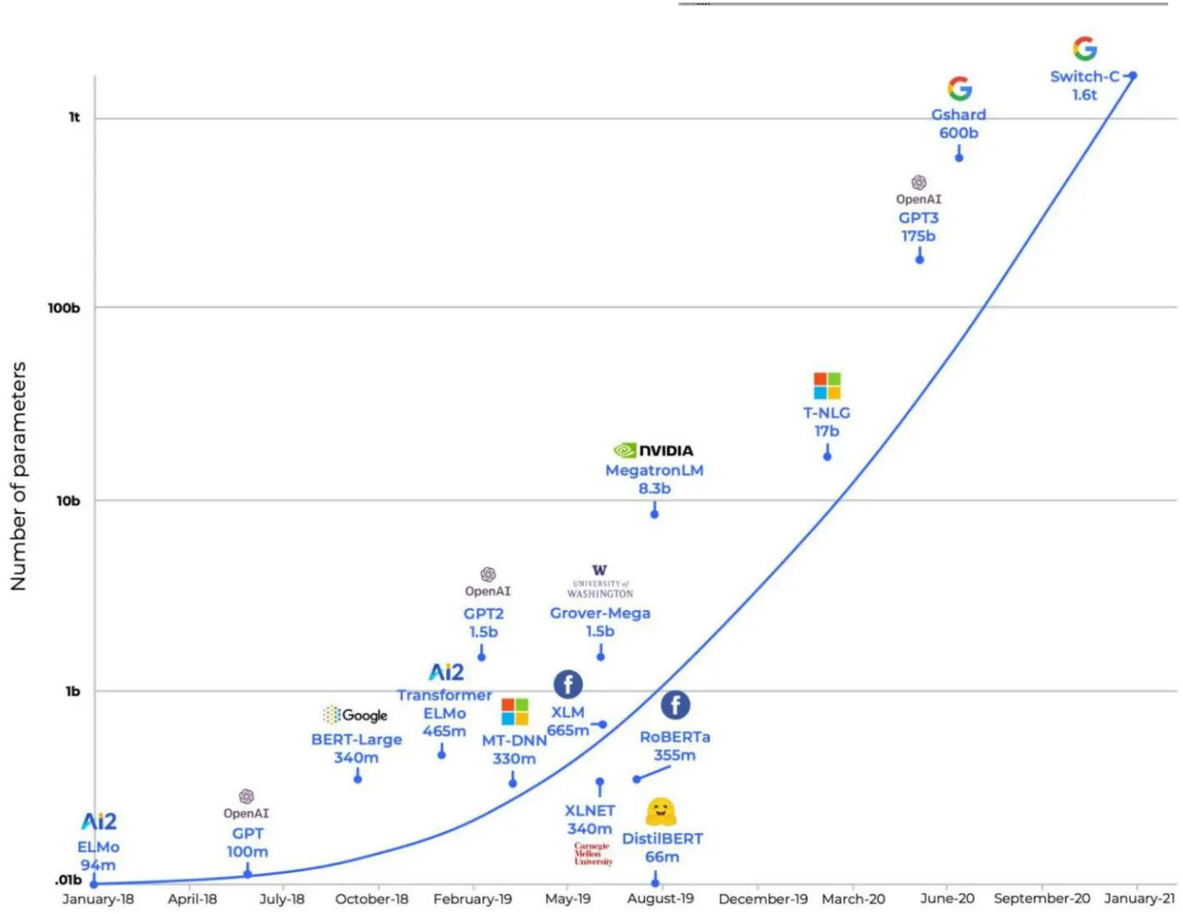
Query data and detect patterns for decision making

Translating data to value

Train machine learning models

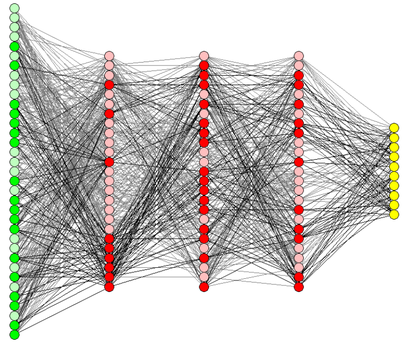


- Exponential growth of number of parameters in ML models



Translating data to value

Train machine learning models

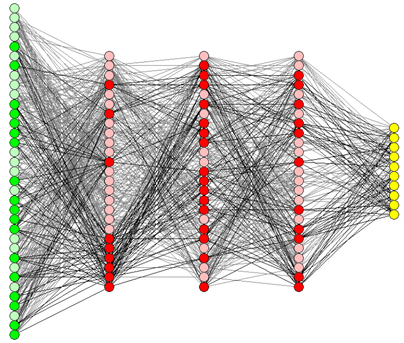


- Exponential growth of number of parameters in ML models
- Increasing computational demands for model training

- **Example 1**. DeepMind trained reinforcement learning (RL) agents to play StarCraft II game [Vinyals'19]
  - 384 TPUs and 1,800 CPUs
  - more than 44 days.
- **Example 2**. OpenAI trained RL agents to play Dota 2 games [Berner'19]
  - 1,536 GPUs and172,800 CPUs,
  - more than 10 months

Translating data to value

Train machine learning models



- Exponential growth
  parameters in ML m
- Increasing computa
  for model training

GPT-4 Technical Report

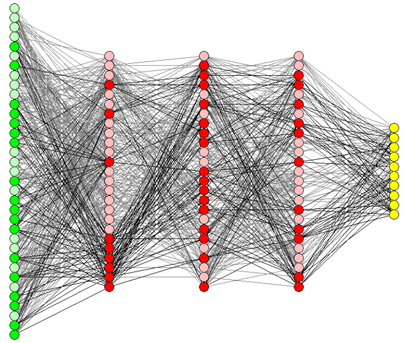
OpenAI*

Abstract

then fine-tuned using Reinforcement Learning from Human Feedback (RLHF) [40]. Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar.

Translating data to value

Train machine learning models



- Exponential growth of number of parameters in ML models
- Increasing computational demands for model training

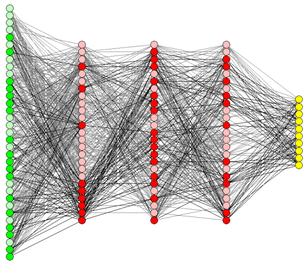
The model-centric paradigm of machine learning is shifting towards a data-centric and system-centric paradigm [Miranda'21]

- **Example 1**. DeepMind trained reinforcement learning (RL) agents to play StarCraft II game [Vinyals'19]
  - 384 TPUs and 1,800 CPUs
  - more than 44 days.
- **Example 2**. OpenAI trained RL agents to play Dota 2 games [Berner'19]
  - 1,536 GPUs and172,800 CPUs,
  - more than 10 months

Translating data to value

Efficient data-intensive systems

Data is large, fast, heterogeneous, incomplete…

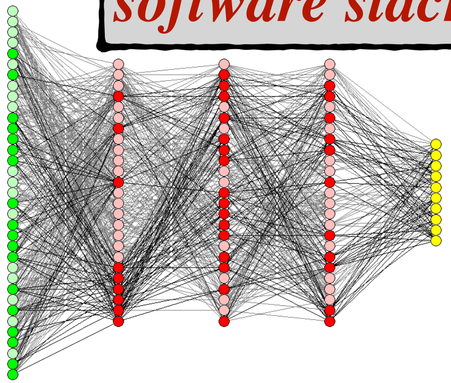


Requirements for efficient data-intensive systems

- **scalability**
- **flexibility**
- **real-time intelligence**

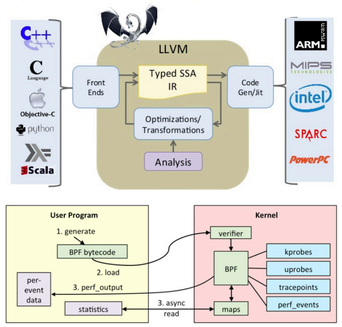
Efficient data-intensive computing systems

My research aims to answer the question "*how to co-design multiple layers of the software stack to improve scalability and performance of machine learning systems*"

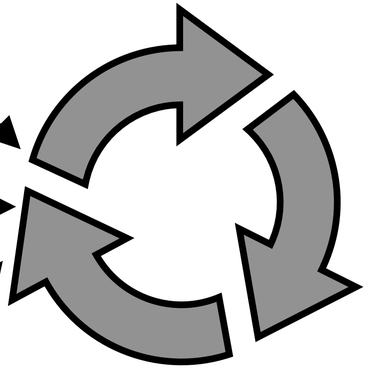


Machine learning layer

Data management layer

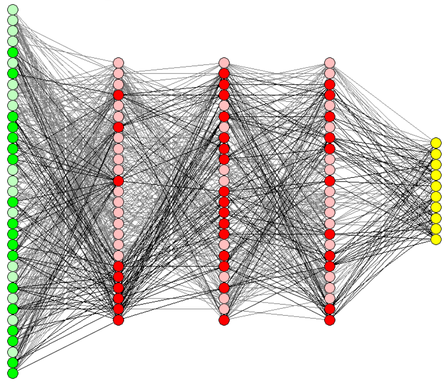


Code optimisation layer



Efficient data-intensive computing systems

My research area



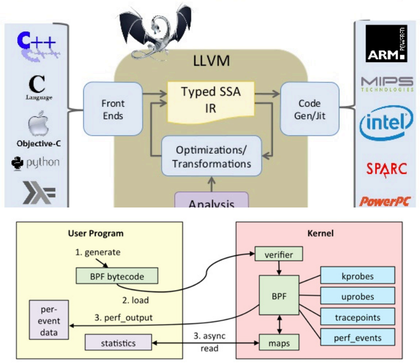
Machine learning layer

scalability and flexibility

Scalable and flexible distributed reinforcement learning systems

- Scale RL training with flexible distribution policies [**ATC'23**], Multi-dimensional parallelism for large DL models.
- Integrated into Huawei's MindSpore deep learning framework

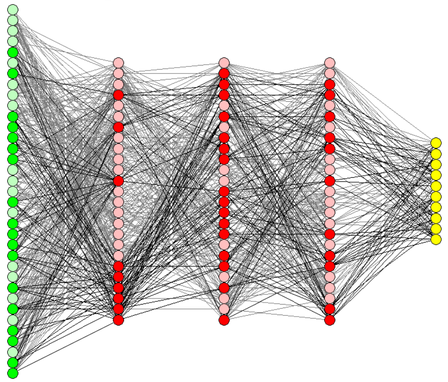
Data management layer



Code optimisation layer

Efficient data-intensive computing systems

My research area



Machine learning layer

scalability and flexibility

Scalable and flexible distributed reinforcement learning systems

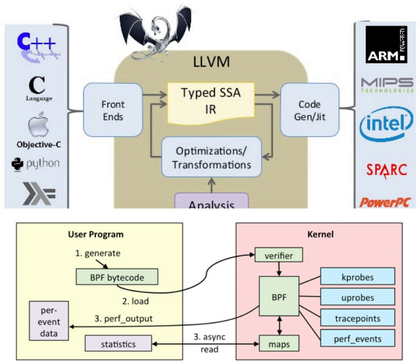
- Scale RL training with flexible distribution policies [**ATC'23**], Multi-dimensional parallelism for large DL models.
- Integrated into Huawei's MindSpore deep learning framework

Data management layer

Real-time intelligence

Efficient stateful data stream management

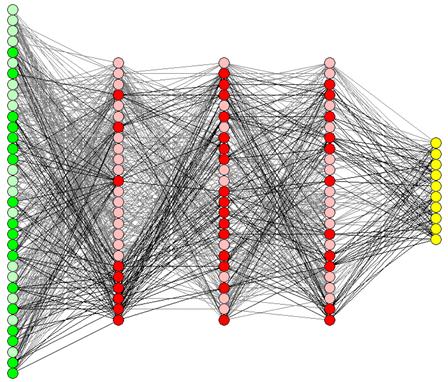
- Load shedding [**ICDE'18, ICDE'20**], Remote data integration [**SIGMOD'21**]
- Applied in real-world smart grid management [**PES-GM'19**, **J. Applied Energy'22**]



Code optimisation layer

Efficient data-intensive computing systems

My research area



Machine learning layer

scalability and flexibility

Scalable and flexible distributed reinforcement learning systems

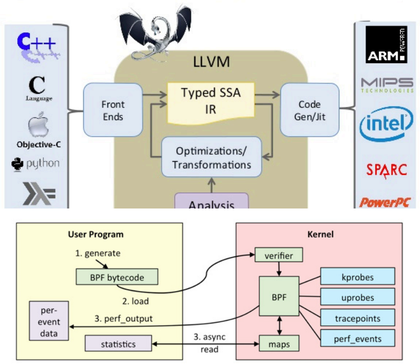
- Scale RL training with flexible distribution policies [**ATC'23**], Multi-dimensional parallelism for large DL models.
- Integrated into Huawei's MindSpore deep learning framework

Data management layer

Real-time intelligence

Efficient stateful data stream management

- Load shedding [**ICDE'18, ICDE'20**], Remote data integration [**SIGMOD'21**]
- Applied in real-world smart grid management [**PES-GM'19**, **J. Applied Energy'22**]



Code optimisation layer

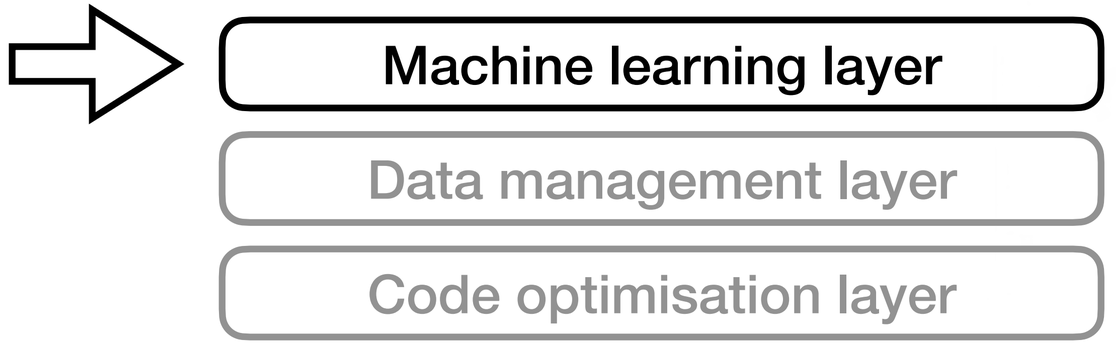
High performance code

- LLVM-based parallel code generation [**COSMIC@CGO'15**, **ICA3PP'15**]
- eBPF-based tracing for cloud-based data warehouse. Integrated into Amazon Redshift

AthenaRL: Flexible Distributed Reinforcement Learning with Dataflow Fragments

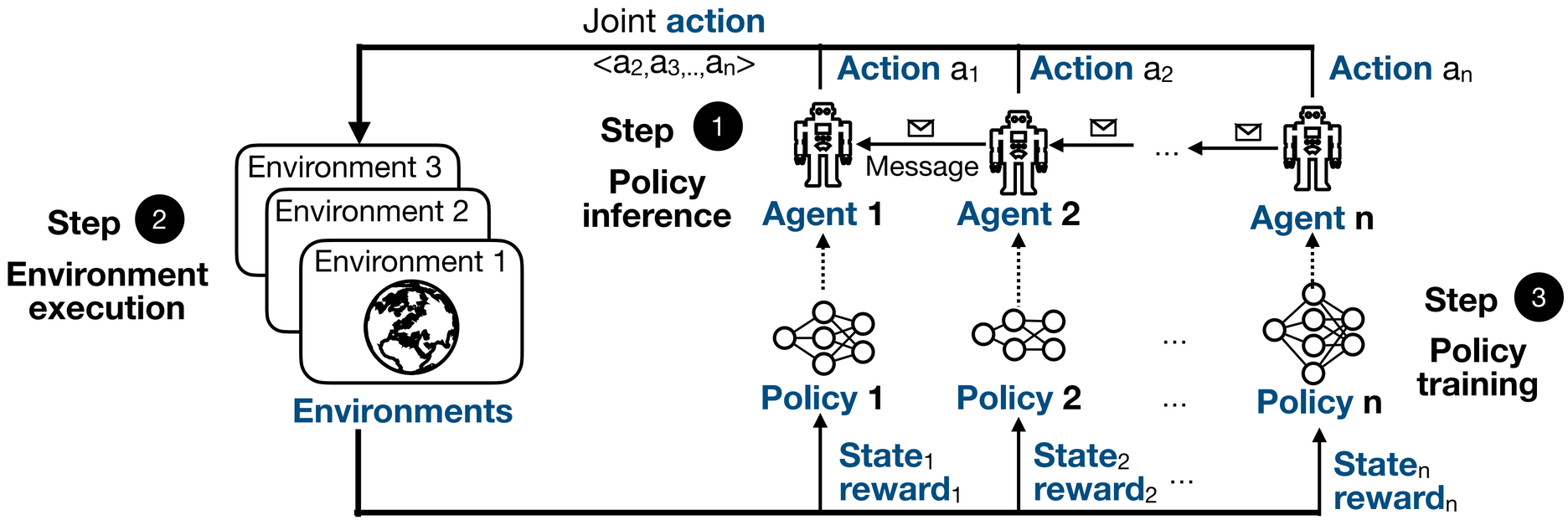
ATC'23

with Huanzhou Zhu, Peter Pietzuch, and Lei Chen

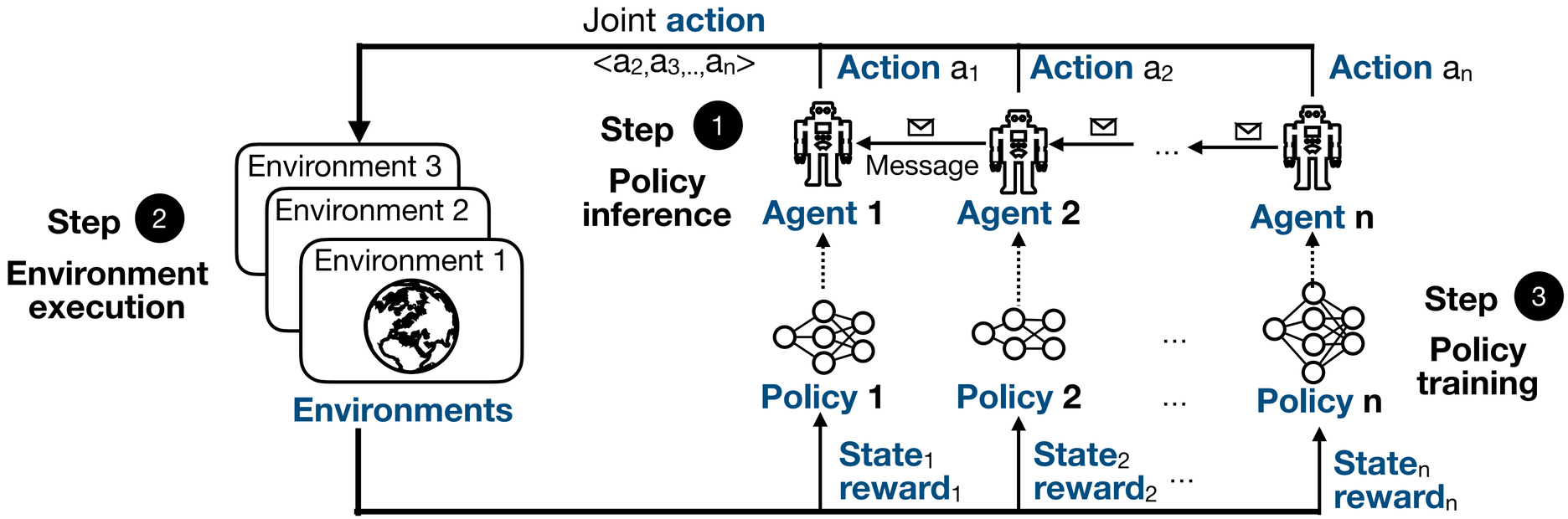


CODE & DATA https://github.com/mindspore-ai/reinforcement

Reinforcement learning training loop



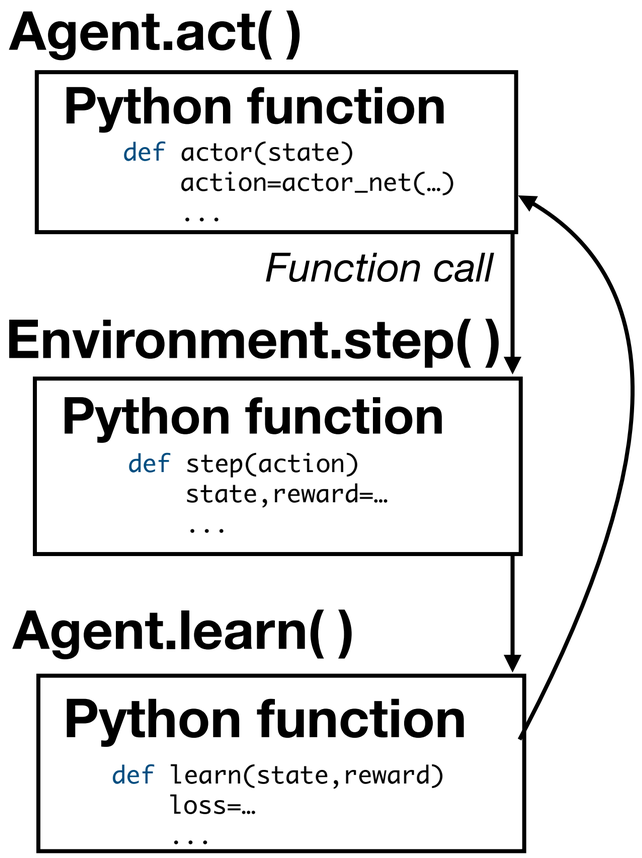
Reinforcement learning training loop



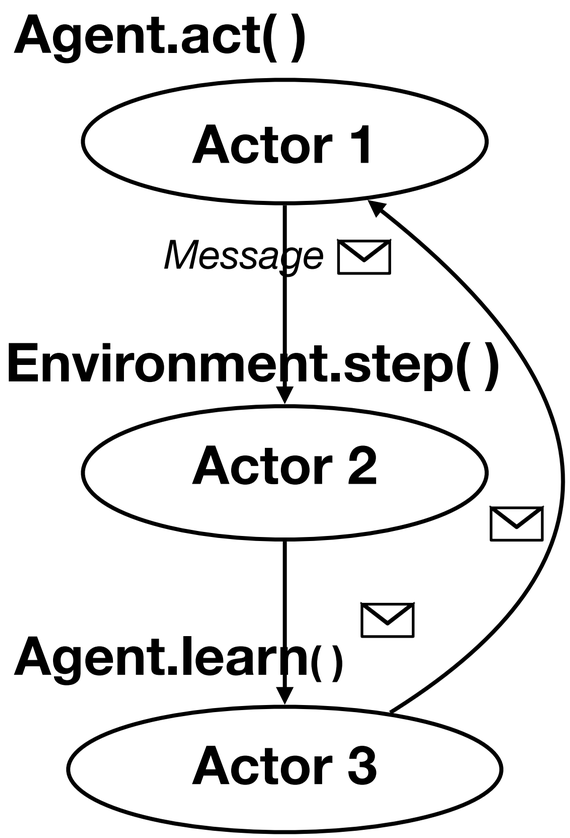
- **RL training is computationally intensive and requires:**
  - **Flexible distribution strategies**
  - **Acceleration for entire training loop**

- State-of-the-art RL frameworks only support:
  - **Fixed** distribution strategies
  - Only accelerate policy (**DNN**) computation

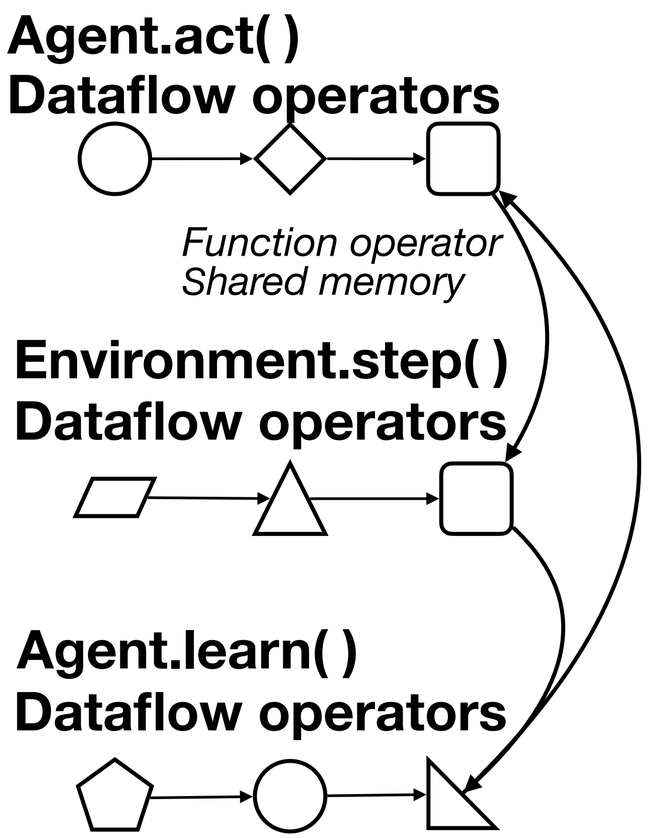
Existing types of RL system designs



Function-based



Actor-based



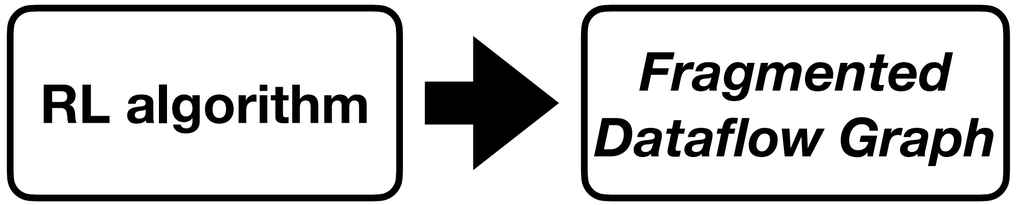
Dataflow-based

Design space of distributed RL systems

Type	System	Execution abstraction	Distribution strategies	Acceleration support	Algorithm abstraction
Function-based	SEED RL [Espeholt'20]	Python functions	environment only	DNNs only	actor/learner/env
	Acme [Hoffman'20]	Python classes			
	RLGraph [Schaarschmidt'19]		delegated to backend		agent
Actor-based	Ray [Moritz'18]	task (stateless) actor (stateful)	scheduler RPC	DNNs only	Python functions with Ray API
	RLlib [Liang'18]				
	MALib [Zhou'21]				agent/actor/learner/
Dataflow-based	Podracer [Hessel'21]	JIT-compiled by JAX	hardcoded	funcs/DNNs/envs	JAX API
	RLlib Flow [Liang'20]	predefined dataflow operators	dataflow operators/ Ray tasks	DNNs only	operator API
	WarpDrive [Lan'21]	GPU thread blocks	—	CUDA kernels	CUDA API
Fragmented dataflow	**AthenaRL**	heterogeneous fragments	any fragment	funcs/operators/ DNNs/envs	agent/actor/learner/ env

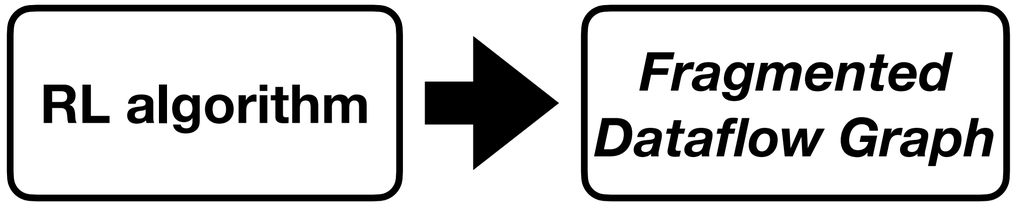
AthenaRL: Flexible distributed reinforcement learning

Decouple an RL algorithm implementation and its execution



AthenaRL: Flexible distributed re

Decouple an RL algorithm implementati



Type	API	Description
Component	`Agent, Actor, Learner, Trainer`	Abstract classes for components
	`Actor.act(...)`	Trajectory collection
	`Learner.learn(...)`	DNN policy training
	`Trainer.train(...)`	RL training loop
	`ATRL.agent_act(...)`	Invoke actor
	`ARTL.agent_learn(...)`	Invoke learner
	`ATRL.env_step(...)`	Execute environment
	`ATRL.env_reset()`	Reset environment
Interaction	`ATRL.replay_buffer_insert(...)`	Store trajectories in buffer
	`ATRL.replay_buffer_sample()`	Sample trajectories from buffer

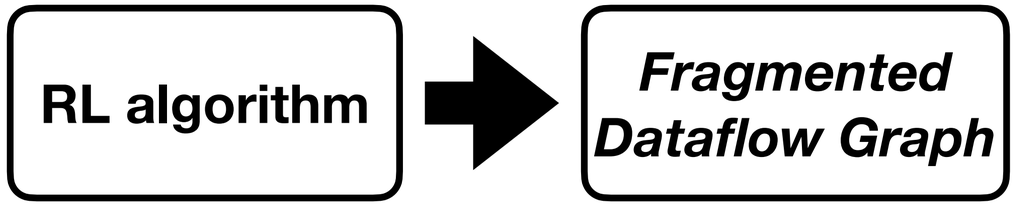
Tab. 2: AthenaRL APIs

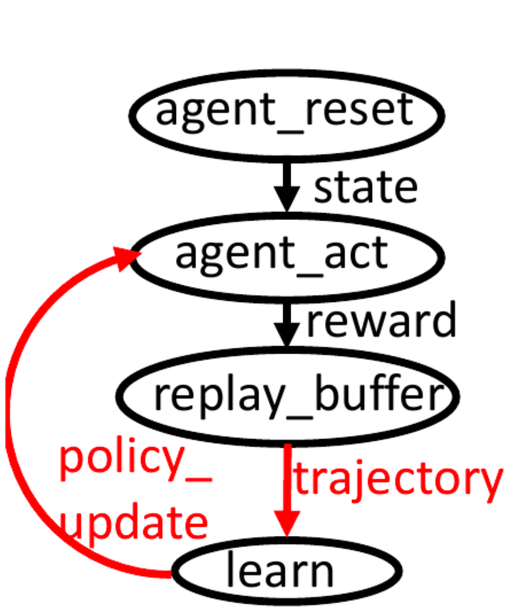
Algorithm 1: MAPPO algorithm in AthenaRL

```
1  class MAPPOAgent(Agent):
2    def act(self,state):
3      return self.actors.act(state)
4    def learn(self,sample):
5      return self.learner.learn(sample)
6  class MAPPOActor(Actor)
7    def act(state):
8      action = self.actor_net(state)
9      reward,new_state = ATRL.env_step(action)
10     ATRL.replay_buffer_insert(reward,new_state)
11     return reward,new_state
12 class MAPPOLearner(Learner):
13   def learn():
14     sample = ATRL.replay_buffer_sample()
15     action,reward,state,next_state = sample
16     last_pred = self.critic_net(next_state)
17     pred = self.critic_net(state)
18     r = discounted_reward(reward,last_pred,self.gamma)
19     adv = gae(reward,next_state,pred,last_pred,self.
           gamma)
20     for i in range(self.iter):
21       loss += self.mappo_net_train(action,state,adv,r)
22     return loss / self.iter
23 class MAPPOTrainer(Trainer):
24   def train(self,episode):
25     for i in range(episode):
26       state = ATRL.env_reset()
27       for j in range(self.duration):
28         reward,new_state = ATRL.agent_act(state)
29       loss = ATRL.agent_learn()
30 mappo_algorithm_config = {
31   'agent':{'num':4,'name':MAPPOAgent,
32            'actor':MAPPOActor,'learner':MAPPOLearner},
33   'actor':{'num':3,'name':MAPPOActor,
34            'policy':MAPPOActorNet,'env':True},
35   'learner':{'num':1,'name':MAPPOLearner,
36              'policy':[MAPPOCriticNet,MAPPONetTrain],
37              'params':{'gamma':0.9}},
38   'env':{'name':MPE,'num':32,'params':{'name':'MPE'}}}
39 mappo_deployment_config = {
40   'workers':[198.168.152.19, 198.168.152.20, [...],
41   'GPUs_per_worker':4},
42   'distribution_policy':'SingleLearnerCoarse'}
```

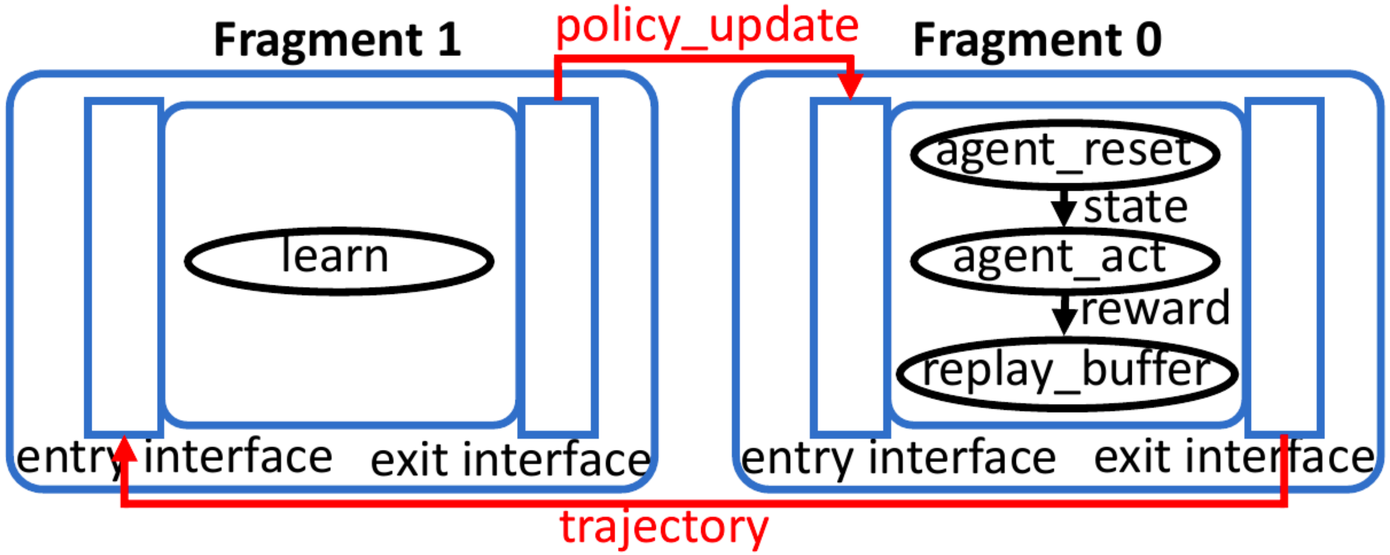
AthenaRL: Flexible distributed reinforcement learning

Decouple an RL algorithm implementation and its execution





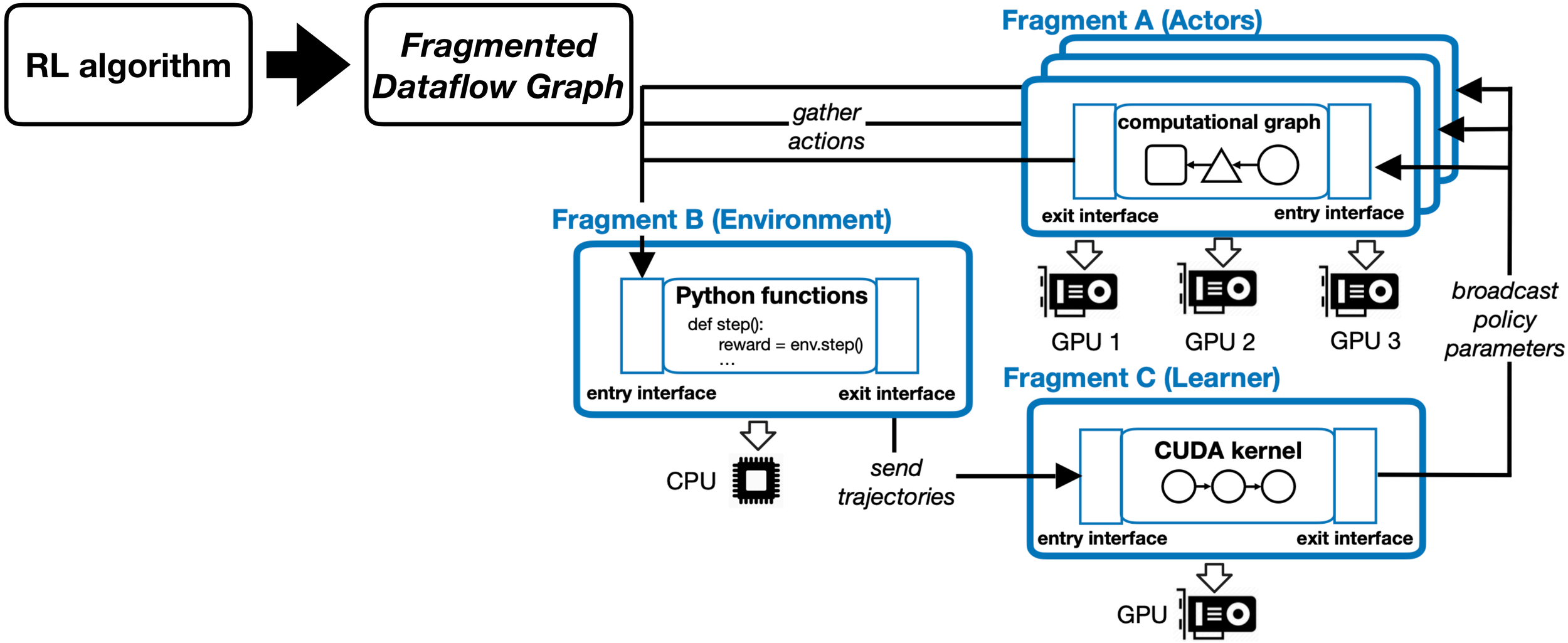
(a) Dataflow graph



(b) Two fragments

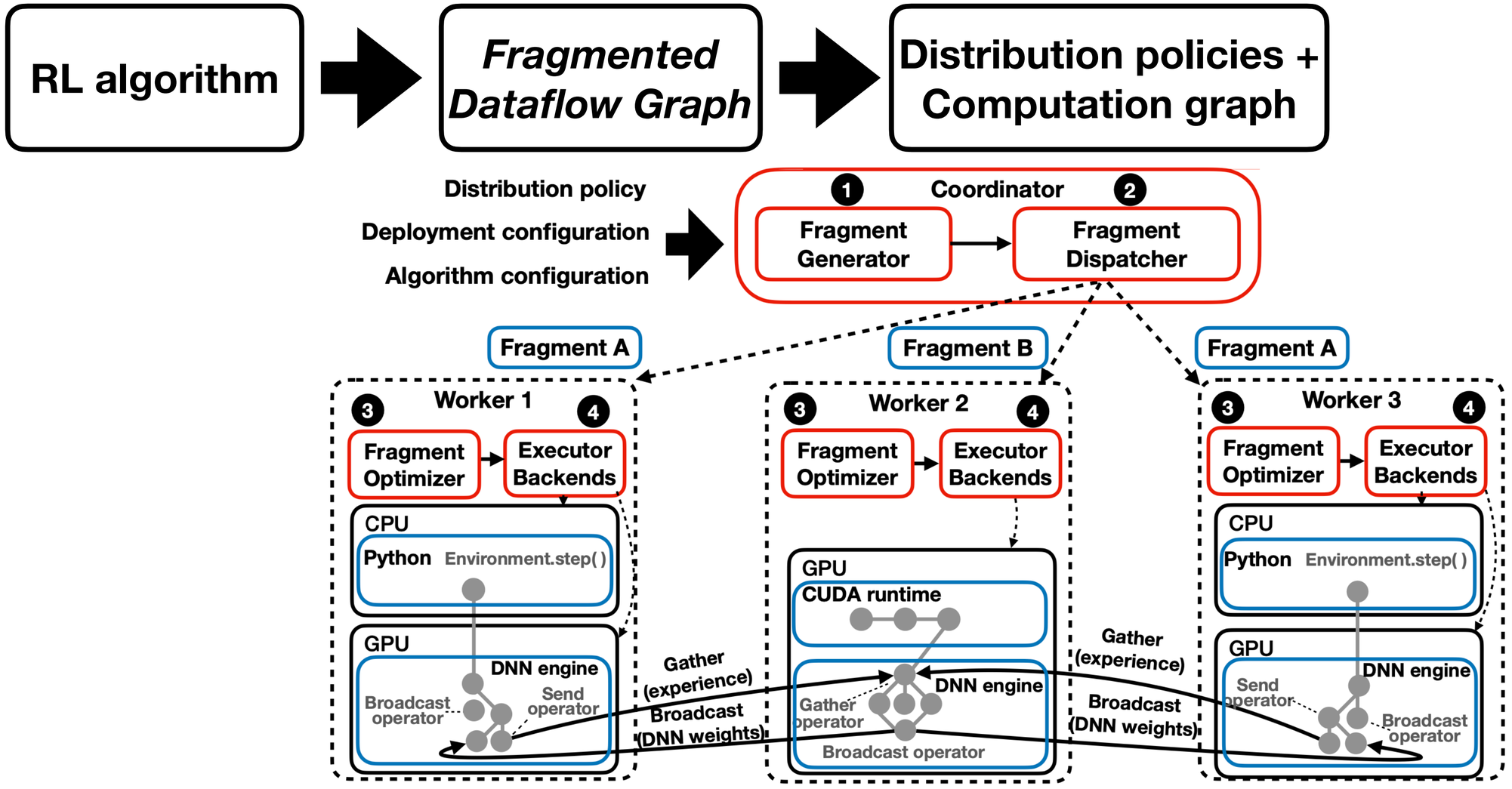
AthenaRL: Flexible distributed reinforcement learning

Decouple an RL algorithm implementation and its execution



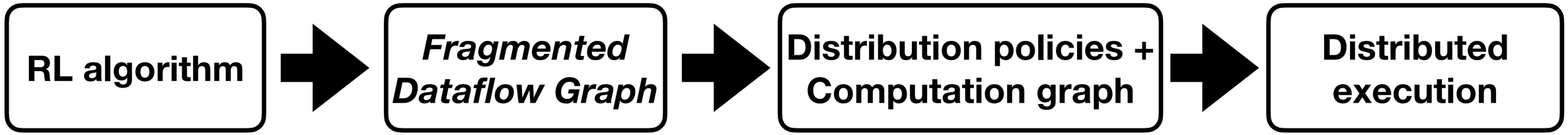
AthenaRL: Flexible distributed reinforcement learning

Decouple an RL algorithm implementation and its execution



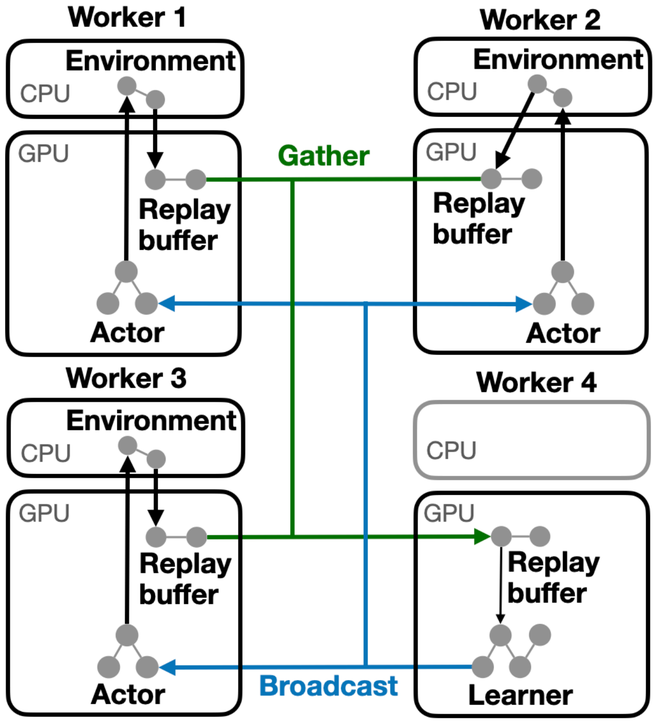
AthenaRL: Flexible distributed reinforcement learning

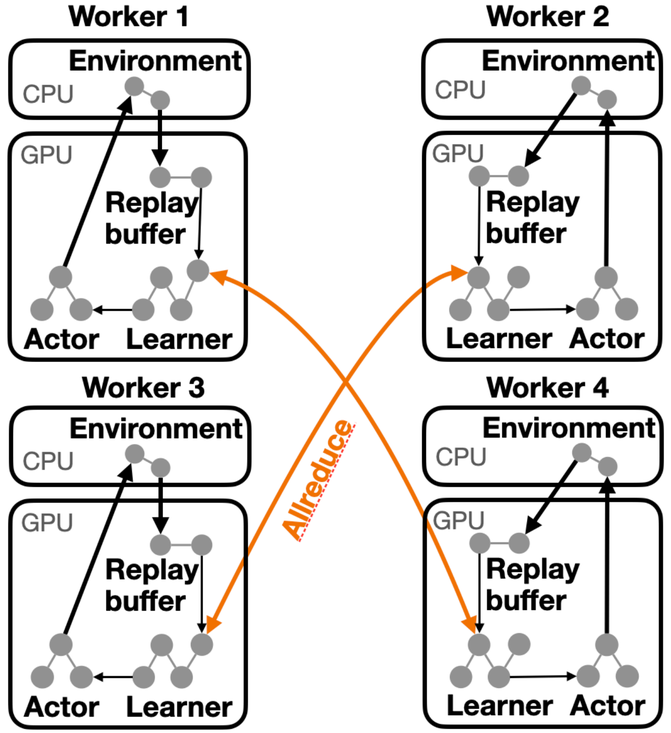
Decouple an RL algorithm implementation and its execution



Distribution police A

- Single learner
- Distribute the environment execution
- Suits complex environments



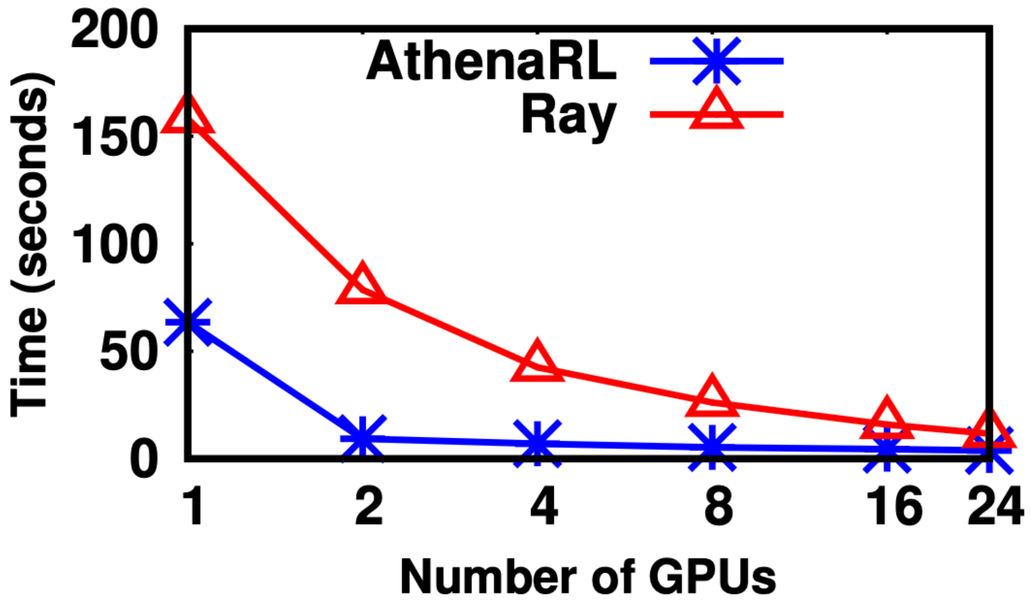


Distribution police B

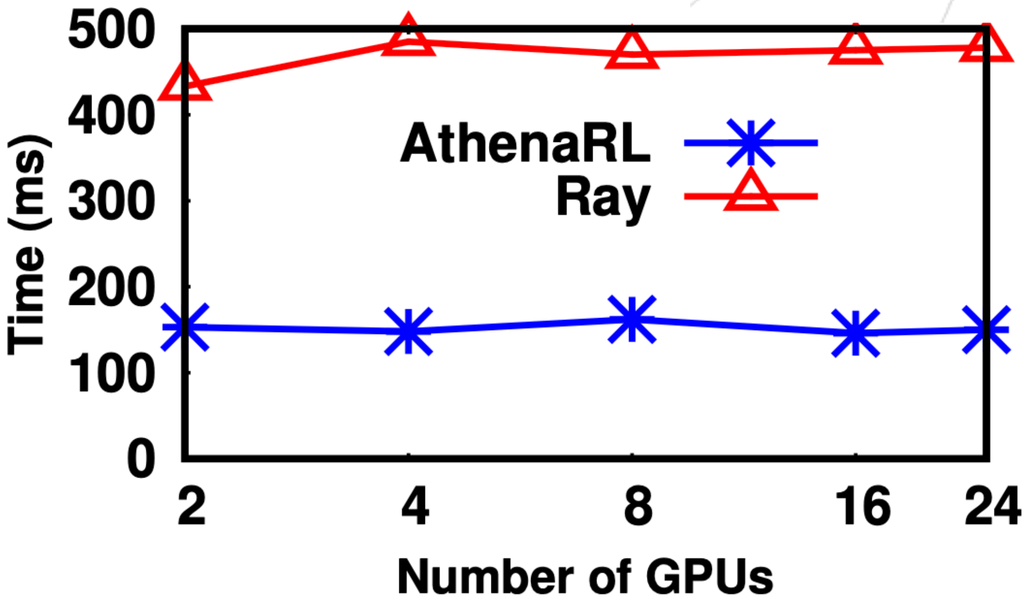
- Multiple learners
- Distribute DNN training
- Suits large models and training data

Key results

Performance comparison with Ray [Moritz'18]



(a) Episode time vs. GPUs (PPO)



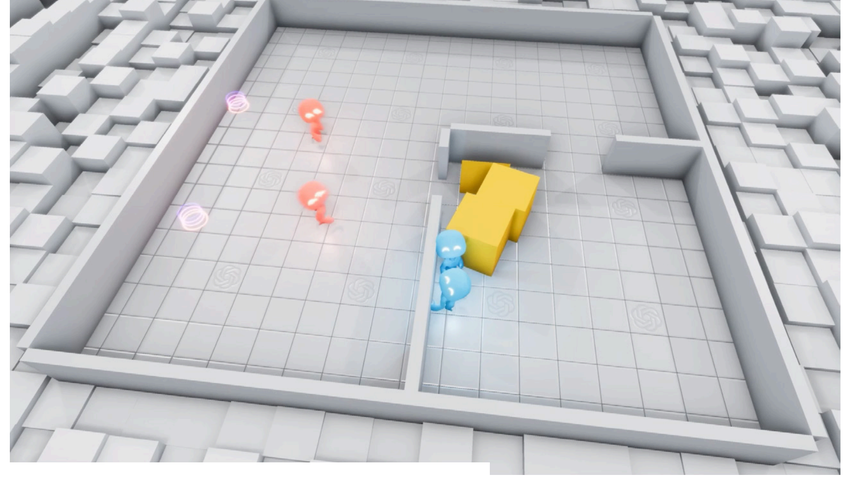
(b) Episode time vs. GPUs (A3C)

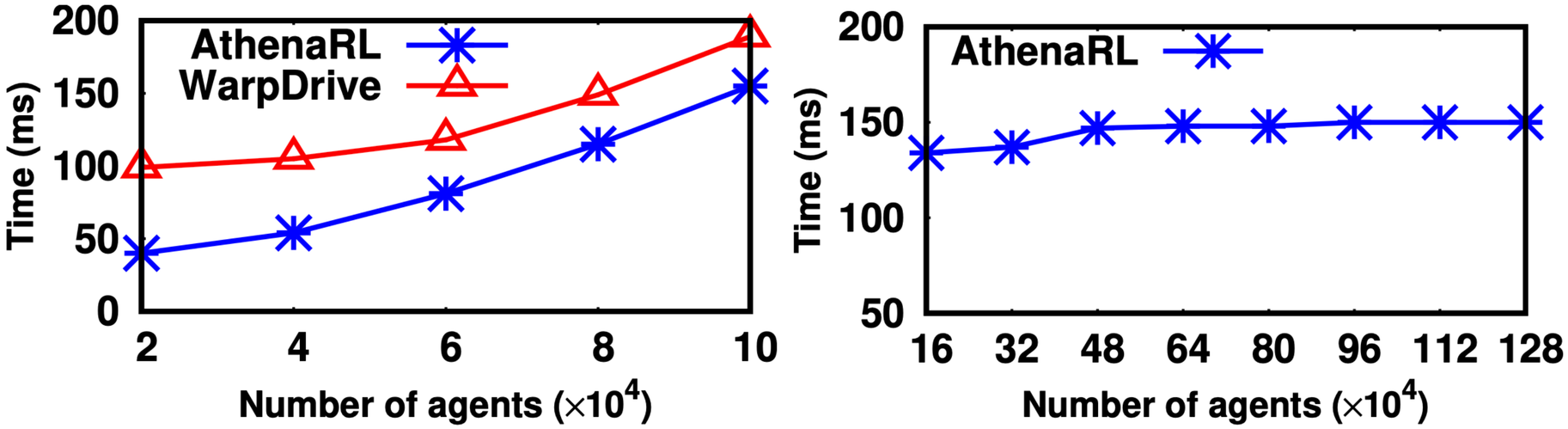
AthenaRL outperforms ray by 3×

- Compiled and optimised computational graph
- Flexible distribution policies leveraging computation and communication patterns

Fully accelerated training loop

Performance comparison with WarpDrive



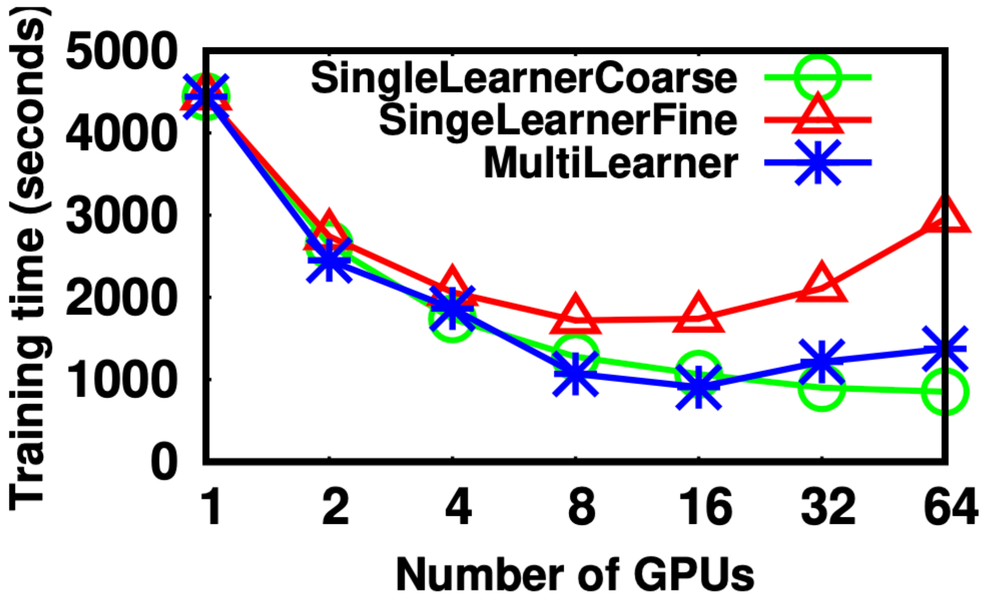


(a) Episode time vs. agents (1 GPU) (b) Episode time vs. agents (n GPUs)

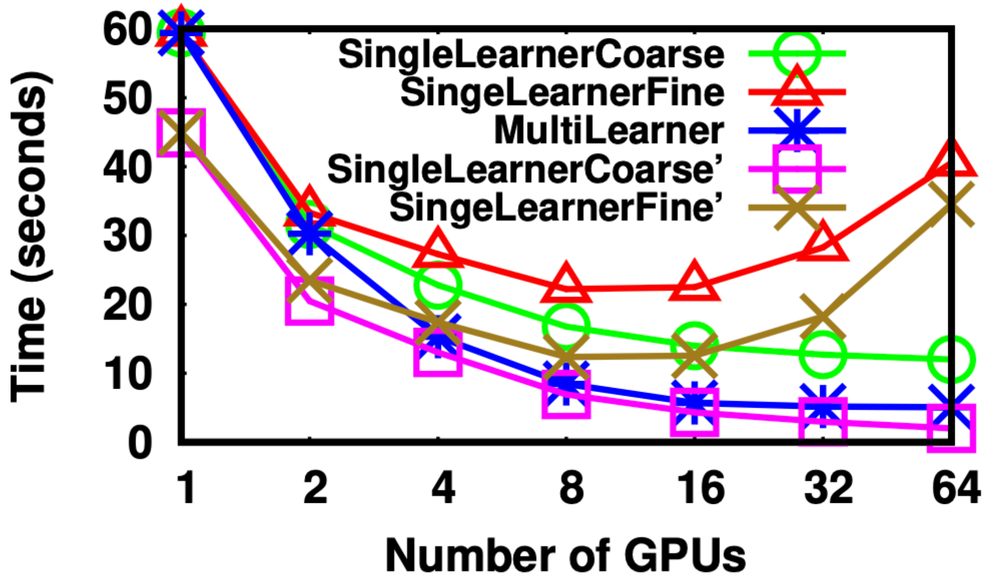
- WarpDrive: state-of-the-art system that fully accelerates the training loop on single GPU [Lan'21]
- AthenaRL achieves better single-GPU performance and scales RL training to many GPU workers

Key results

Impact of GPU counts on distribution policies



(a) Training time vs. GPUs



(b) Episode time vs. GPUs

- AthenaRL scales RL algorithms to 64 GPU workers
- Distribution policies present different scalability performance

MindSpore | Cost analysis

Resource group

Search

Overview
Activity log
Access control (IAM)
Tags
Resource visualizer
Events

Settings
Deployments
Security
Policies
Properties
Locks

Cost Management
Cost analysis
Cost alerts (preview)
Budgets
Advisor recommendations

Monitoring
Insights (preview)
Alerts
Metrics
Diagnostic settings
Logs
Advisor recommendations
Workbooks

Automation
Export template

Support + troubleshooting
New Support Request

Save | Save as | Delete view | Share | Subscribe | Refresh | Download | Cost by resource | Configure resource group | Try preview | Help

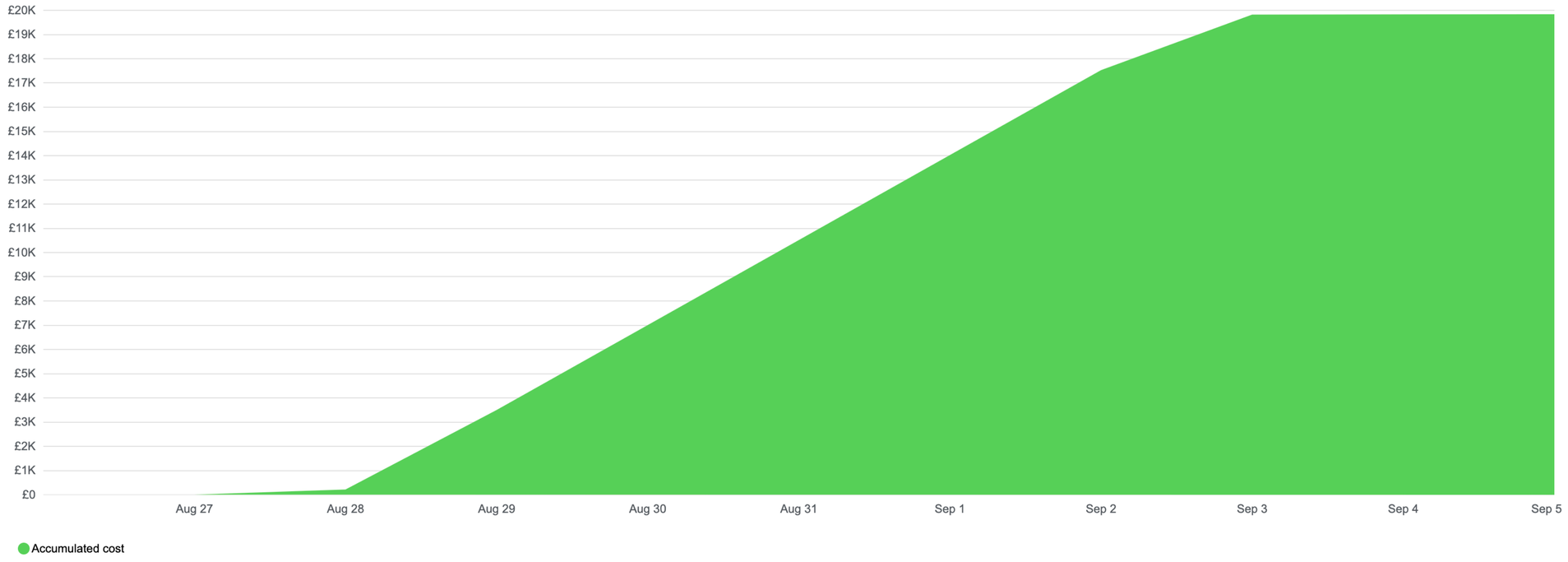
Scope : MindSpore | VIEW * AccumulatedCosts | Aug 27-Sep 5, 2022 | Add filter

ACTUAL COST (GBP ONLY) **£19,839.92**

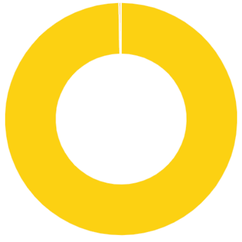
FORECAST UNAVAILABLE --

BUDGET: NONE --

Group by: None | Granularity: Accumulated | Area

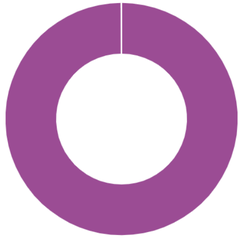


Service name



Virtual Machines £19,783.93
Storage £50.29
Container Registry £4.96
Virtual Network £0.72
Bandwidth £0.02

Location



EU West £19,833.65
US East £6.25
Intercontinental £0.02

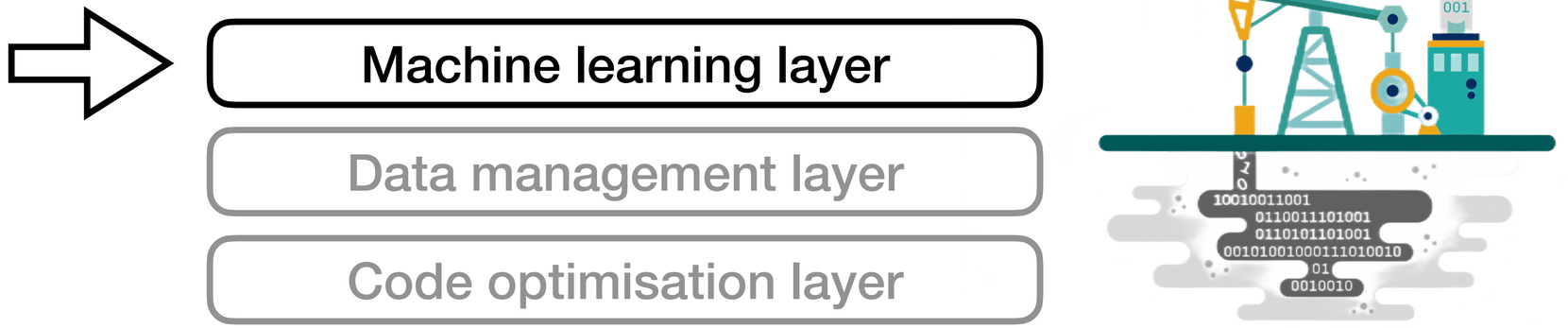
Resource

ml-base-ubuntu20-0... £1,094.63
ml-base-ubuntu20-1... £1,094.63
ml-base-ubuntu20-0... £1,094.50
ml-base-ubuntu20-1... £1,094.36
ml-base-ubuntu20-0... £1,093.43

TENPLEX: Device-Independent Deep Learning with Multi-Dimensional Parallelism

Ongoing work

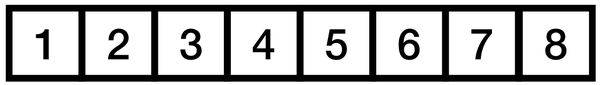
with Guo Li, Marcel Wagenländer, Luo Mai and Peter Pietzuch

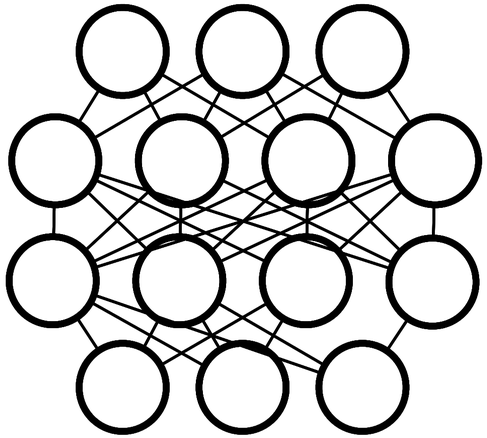


CODE & DATA to be realised soon… :)

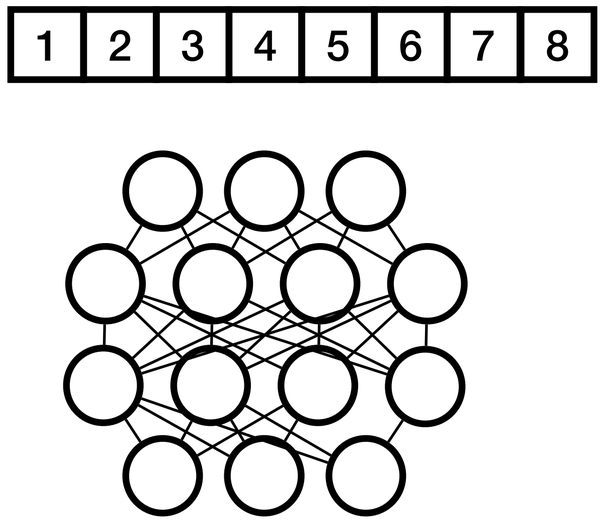
Distributed training of large DL models

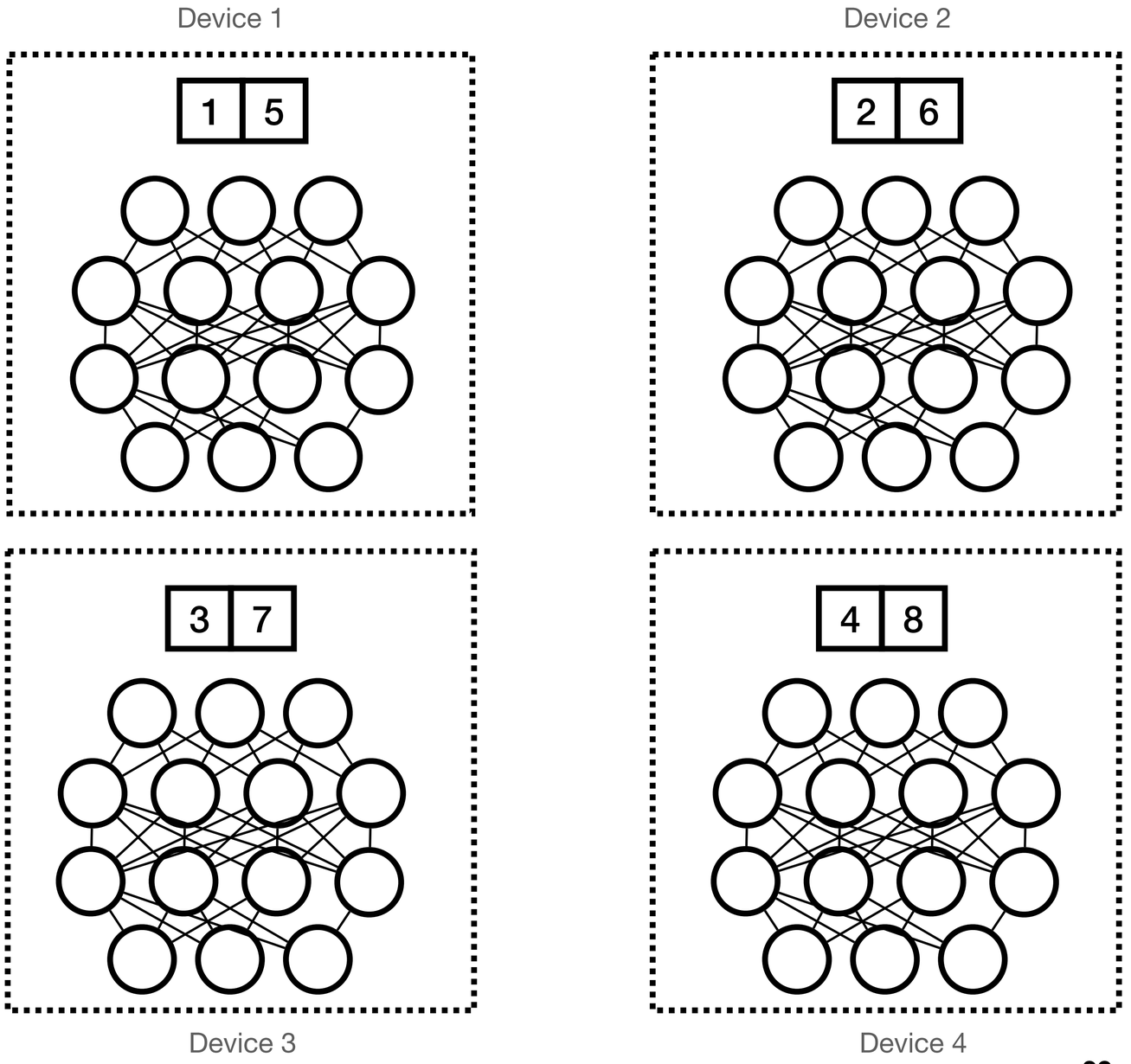
Ongoing work…



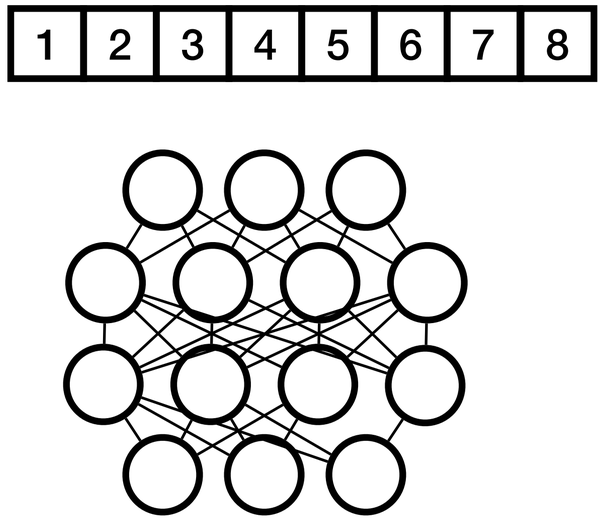


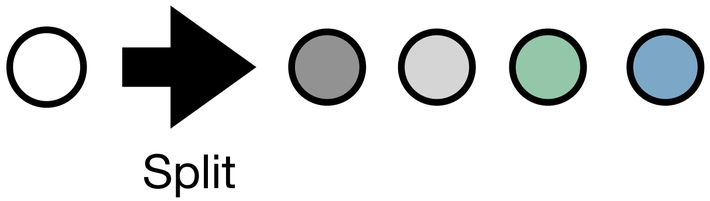
Data parallelism



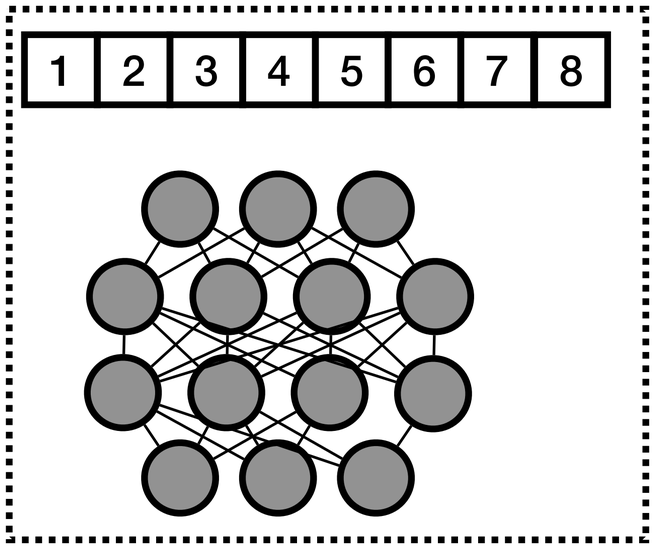


Model parallelism

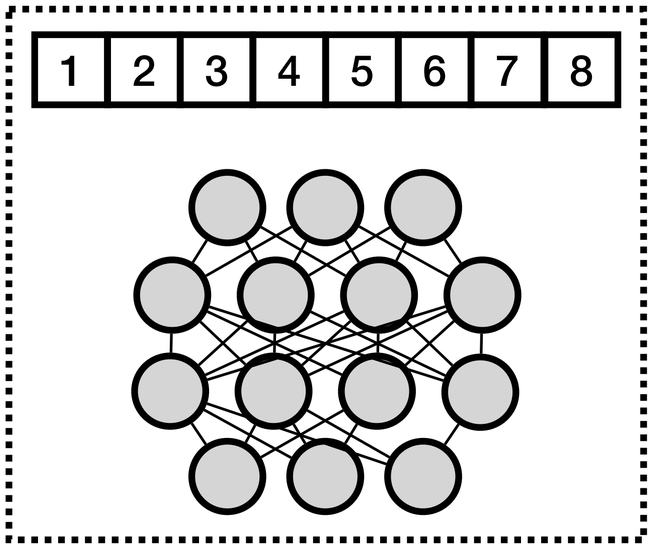


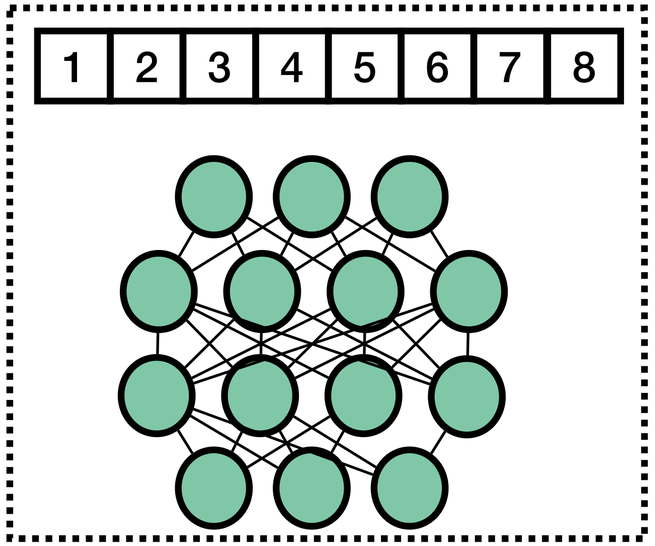


Device 1

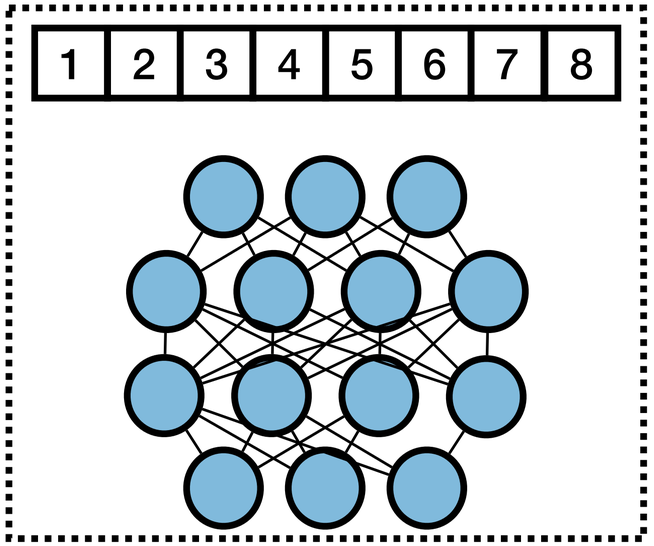


Device 2



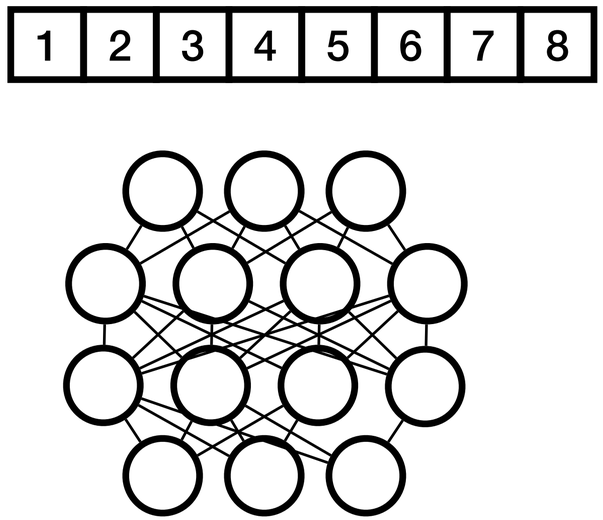


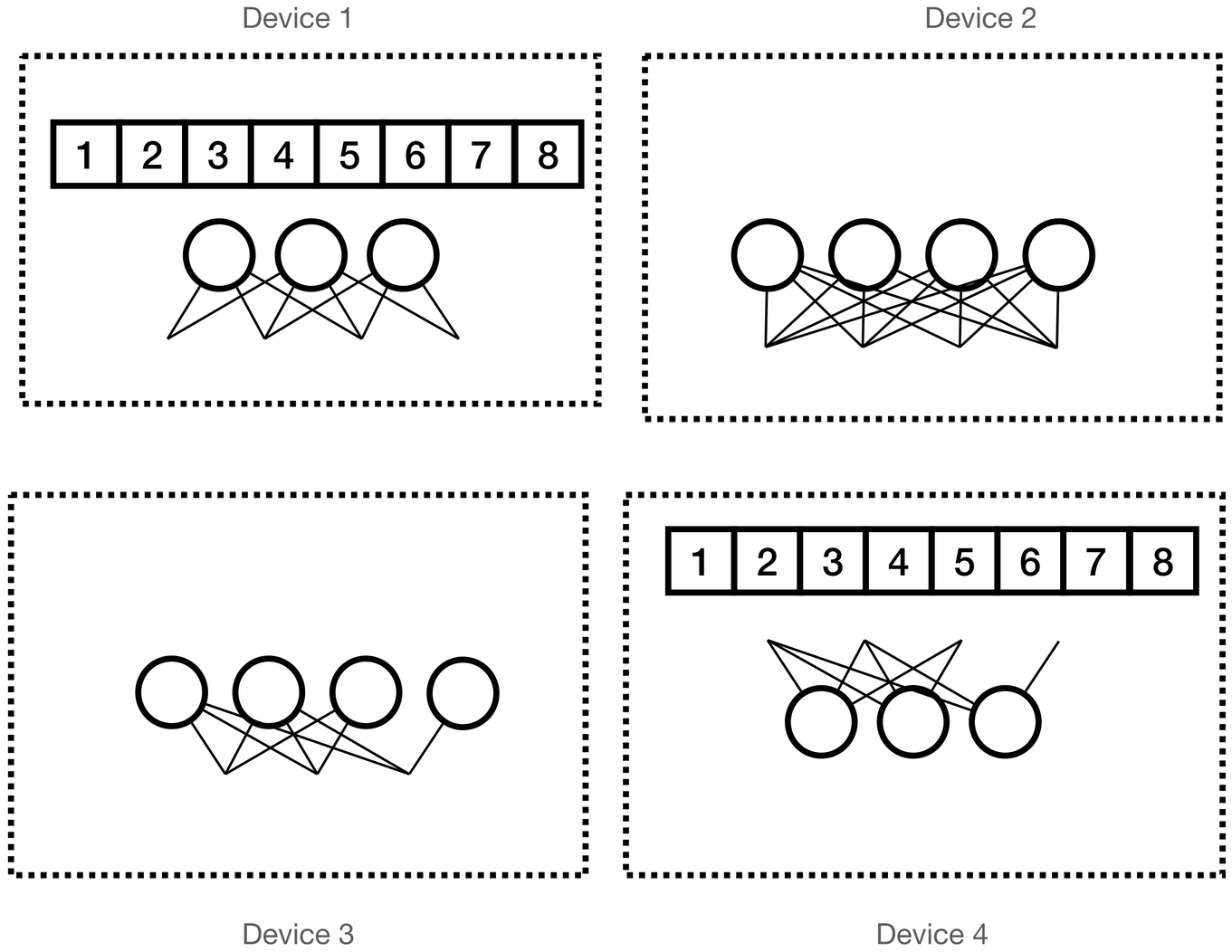
Device 3



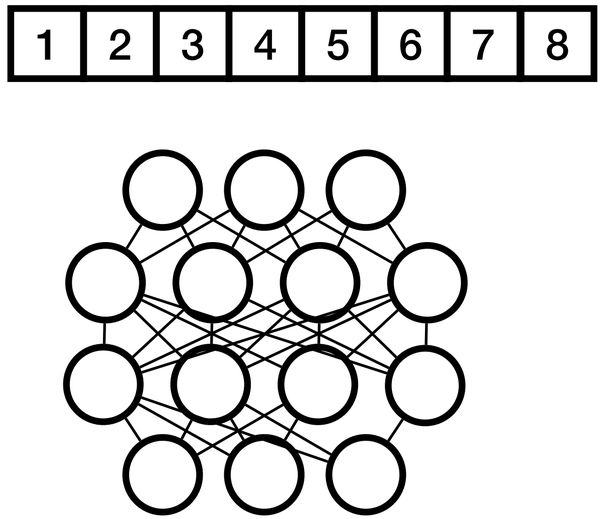
Device 4

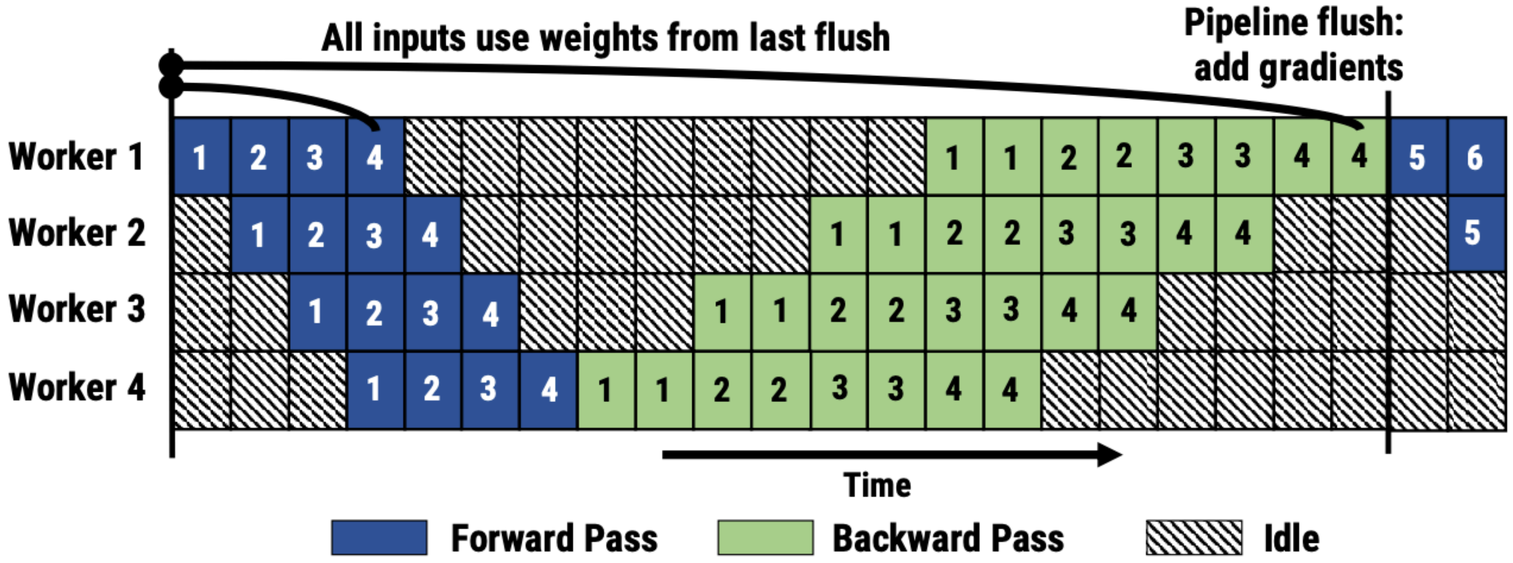
Pipeline parallelism





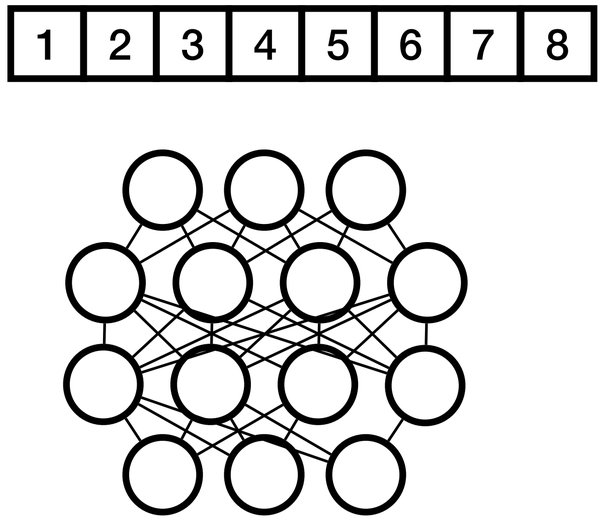
Pipeline parallelism

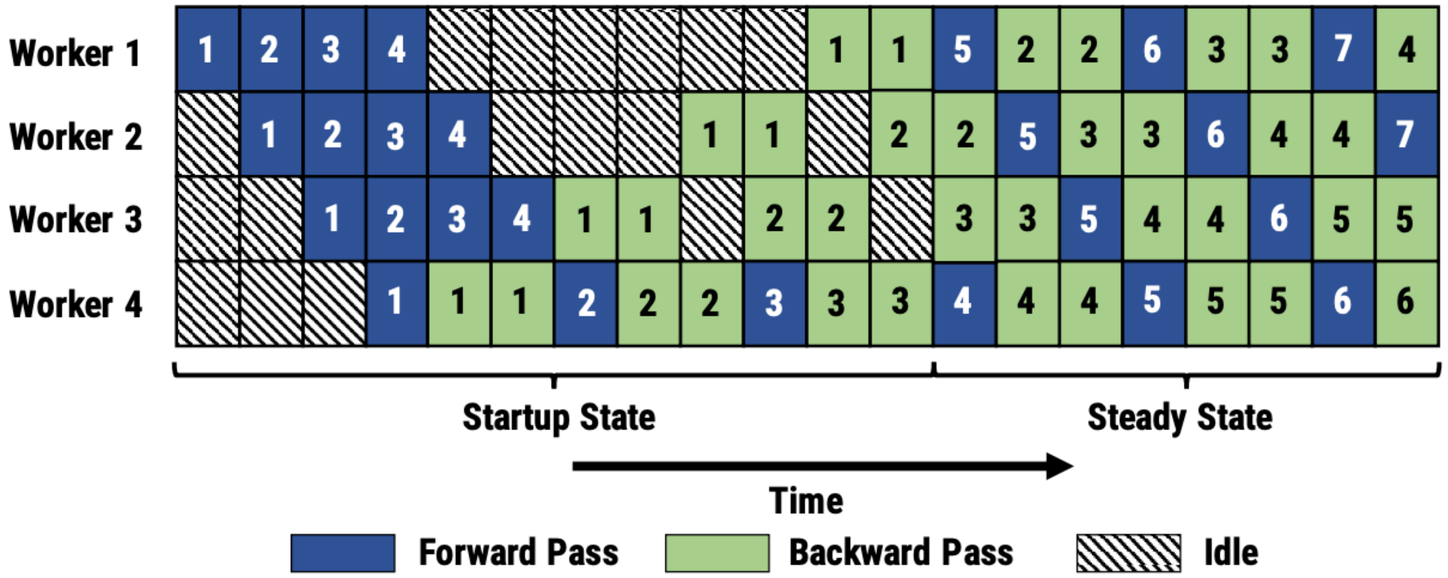




GPipe [Huang'19], inter-batch parallelism

Pipeline parallelism





PipeDream [Narayanan'19], inter-batch and intra-batch parallelism

Multi-Dimensional Parallelism (MDP)

- MDP = {data parallelism (DP), model parallelism (DP), Pipeline parallelism (PP)}

Multi-Dimensional Parallelism (MDP)

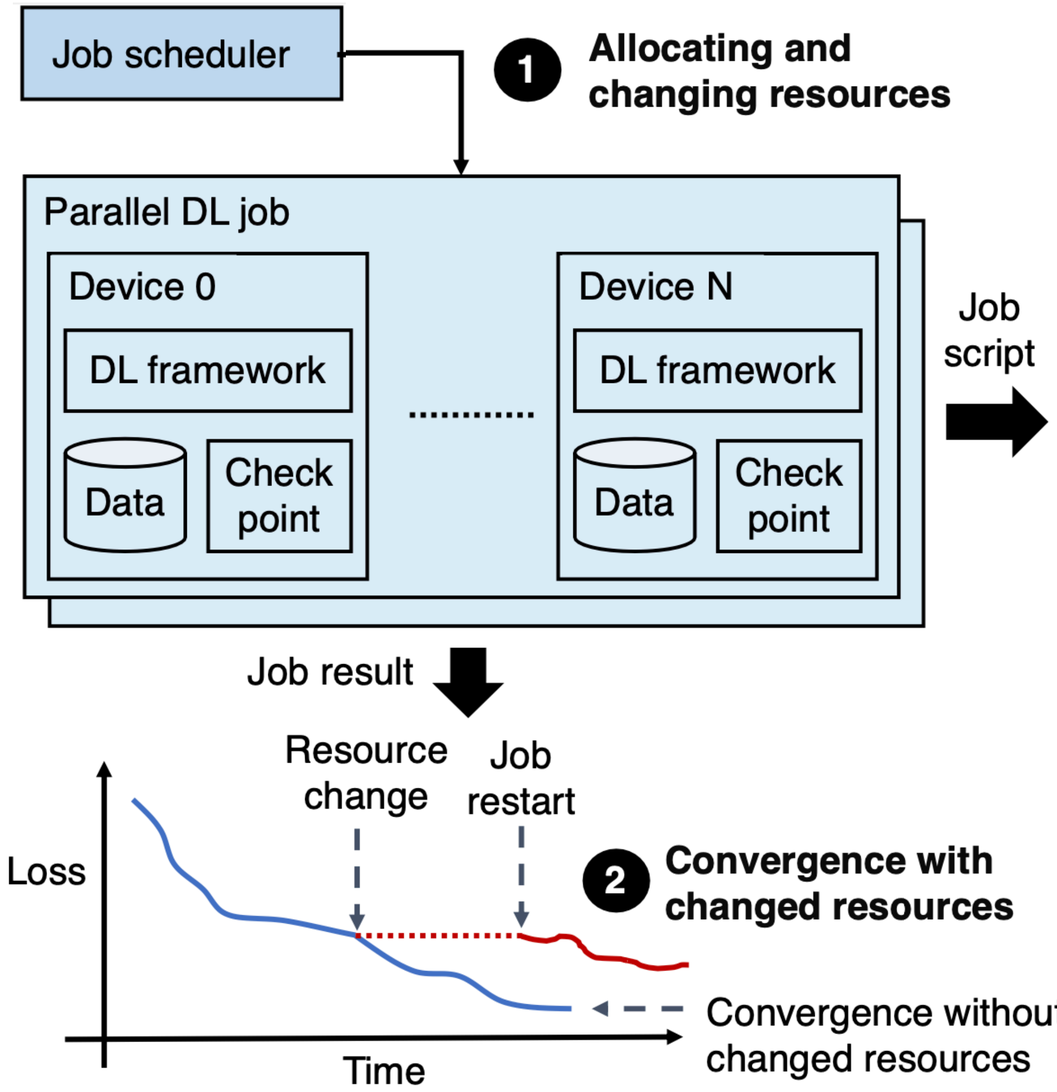
- MDP = {data parallelism (DP), model parallelism (DP), Pipeline parallelism (PP)}
- Distributed DL training may need to change resources
  - Elasticity & auto-scaling
  - Reallocating resources for fine-tuning jobs
  - Transient resources (e.g., *spot* GPU instances)
  - Multi-tenant ML jobs in cloud environment

Limitations of existing elastic DL systems

System	DP	PP	MP	Scale DP	Scale PP	Scale MP	Ensure consistency with scaling
Elastic Horovod	✓	×	×	✓	×	×	✓(user effort)
VirtualFlow	✓	×	×	✓	×	×	✓(scaling down)
Torch Distributed Elastic	✓	×	×	✓	×	×	✓(epoch boundary)
Megatron-LM	✓	✓	✓	×	×	×	×
Deepspeed	✓	✓	✓	×	×	×	×
Singularity	✓	✓	✓	✓	×	×	✓(scaling down)
Tenplex	✓	✓	✓	✓	✓	✓	✓

- Only support changes along the data parallelism dimension
- Affect dataset/data stream reading order and training convergence
- Need manual reconfiguration of multi-dimensional parallelism

Limitations of existing elastic DL systems



User intervention

```
# Sample DL job script

import dl_framework as dl

# PP: Pipeline parallel rank
# MP: Model parallel rank
m = dl.Load("/ckpt/<device>/ckpt_<PP>_<MP>.pt")
…
```

3 Changed checkpoint partitioning

```
ds = dl.Dataset(”/data”, download=True)
…
```

4 Synchronous dataset downloading

```
# DP: Data parallel rank
# DP_size: Data parallel size
s = dl.DistSampler(ds, DP_size, DP, shuffle)
…
```

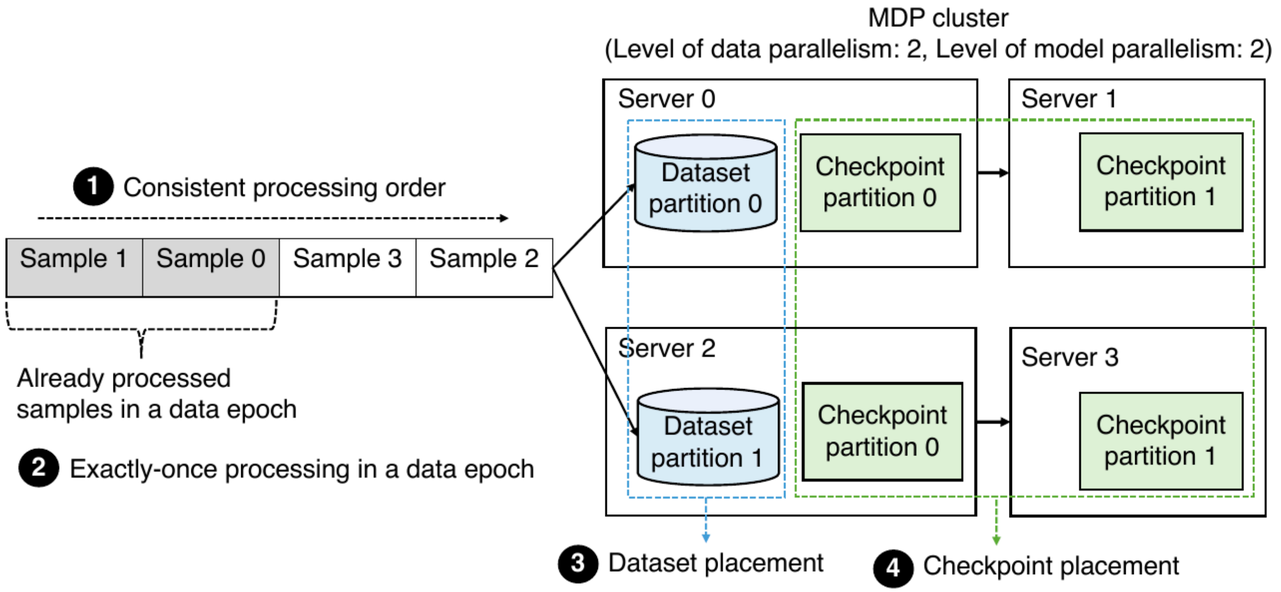
5 Changed dataset state

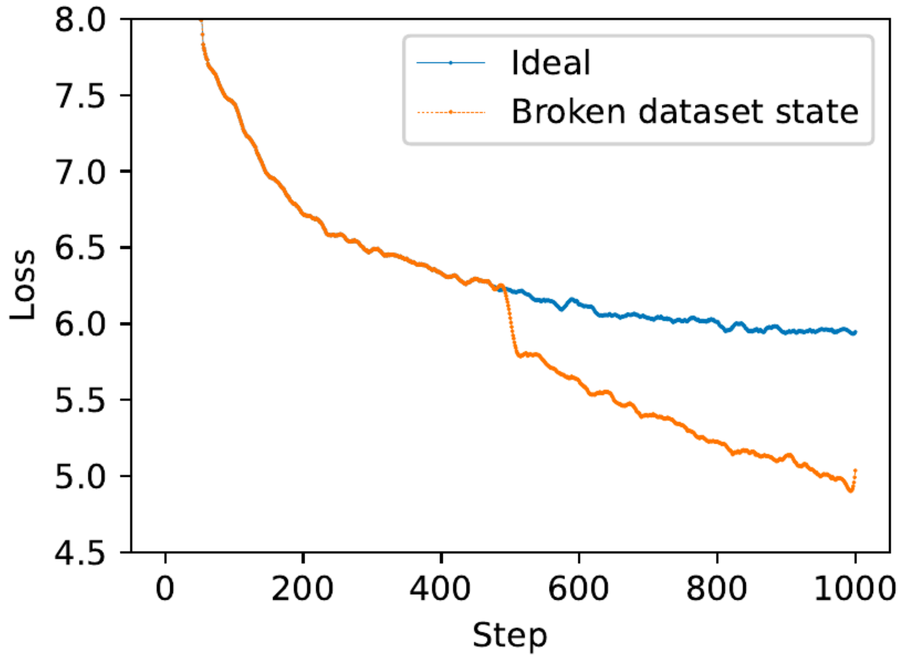
```
opt = dl.DistOptimiser(lr=lr*DP_size)
…
```

6 Changed hyper-parameter

```
for batch in s.GetNext():
   # Train the model
   m.Train(opt, batch)
…
```

Broken data reading order



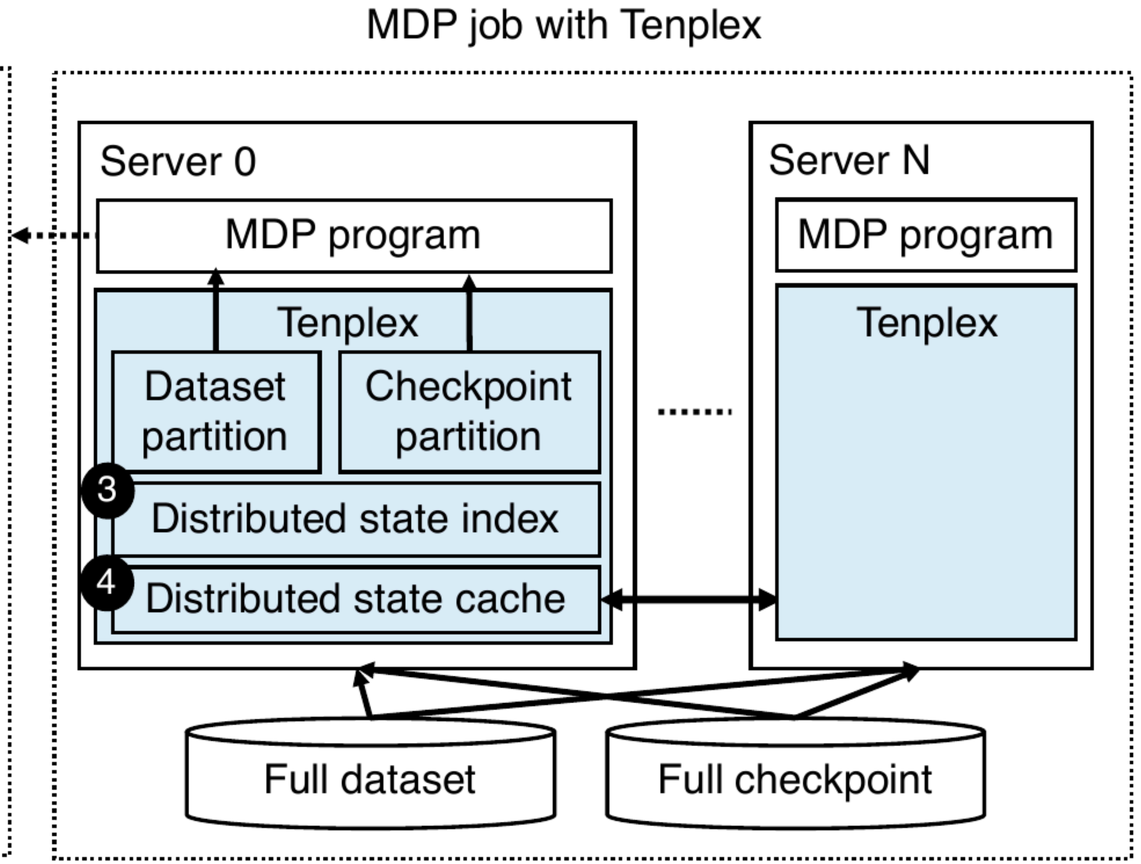


Tenplex

Transparently change resources with MDP

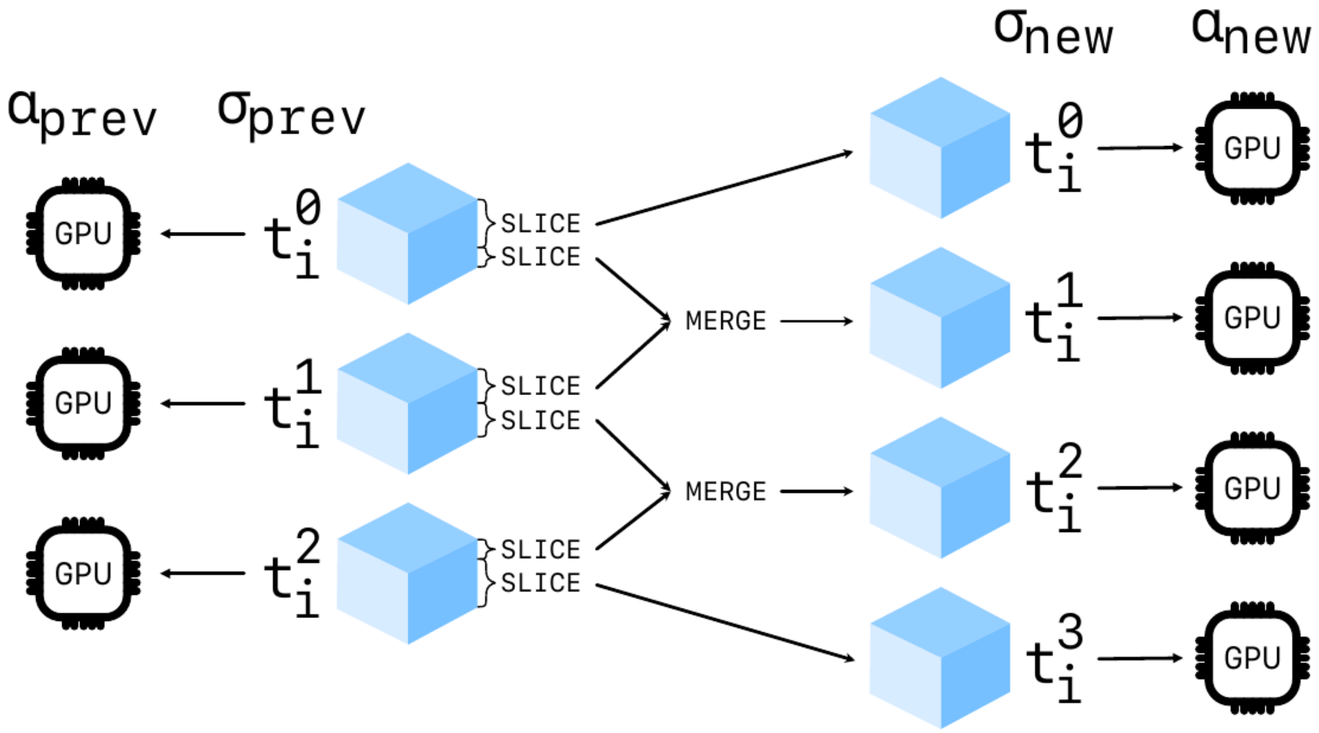
MDP program with Tenplex

```
import mdp
import tenplex as tp

# Get a checkpoint partition
c = tp.partition_checkpoint("http://checkpoint") 1
m = mdp.model(c)

# Get a dataset partition
d = tp.partition_dataset("http://dataset") 2

for b in mdp.batch(d):
  m.train(b)
```



Training state transformation

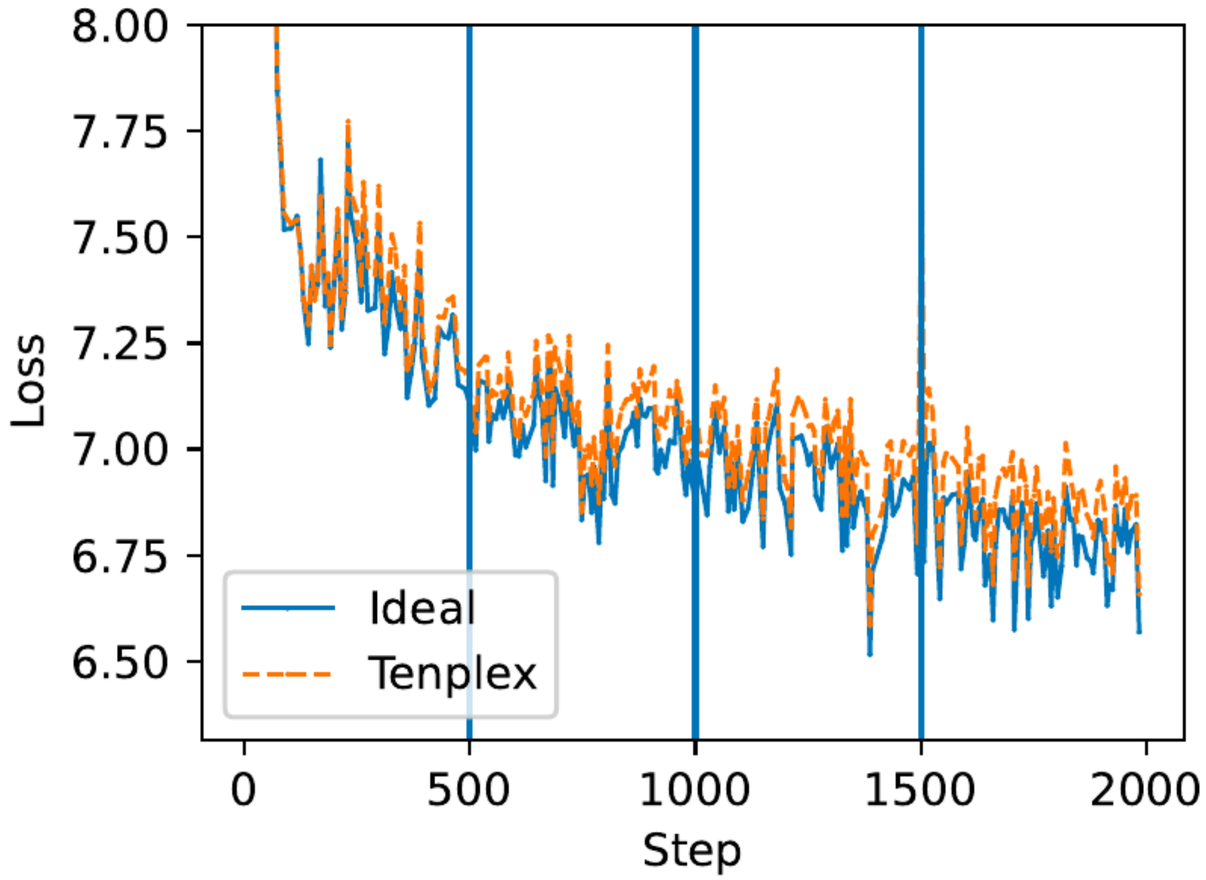
- Training state abstract
  - Tensor collections, *T* (i.e., data samples and model checkpoints)
  - Tensor slicing function, σ
  - Allocation function, α



Resource change with MDP

Preliminary results (small cluster, 20 A6000 GPUs)

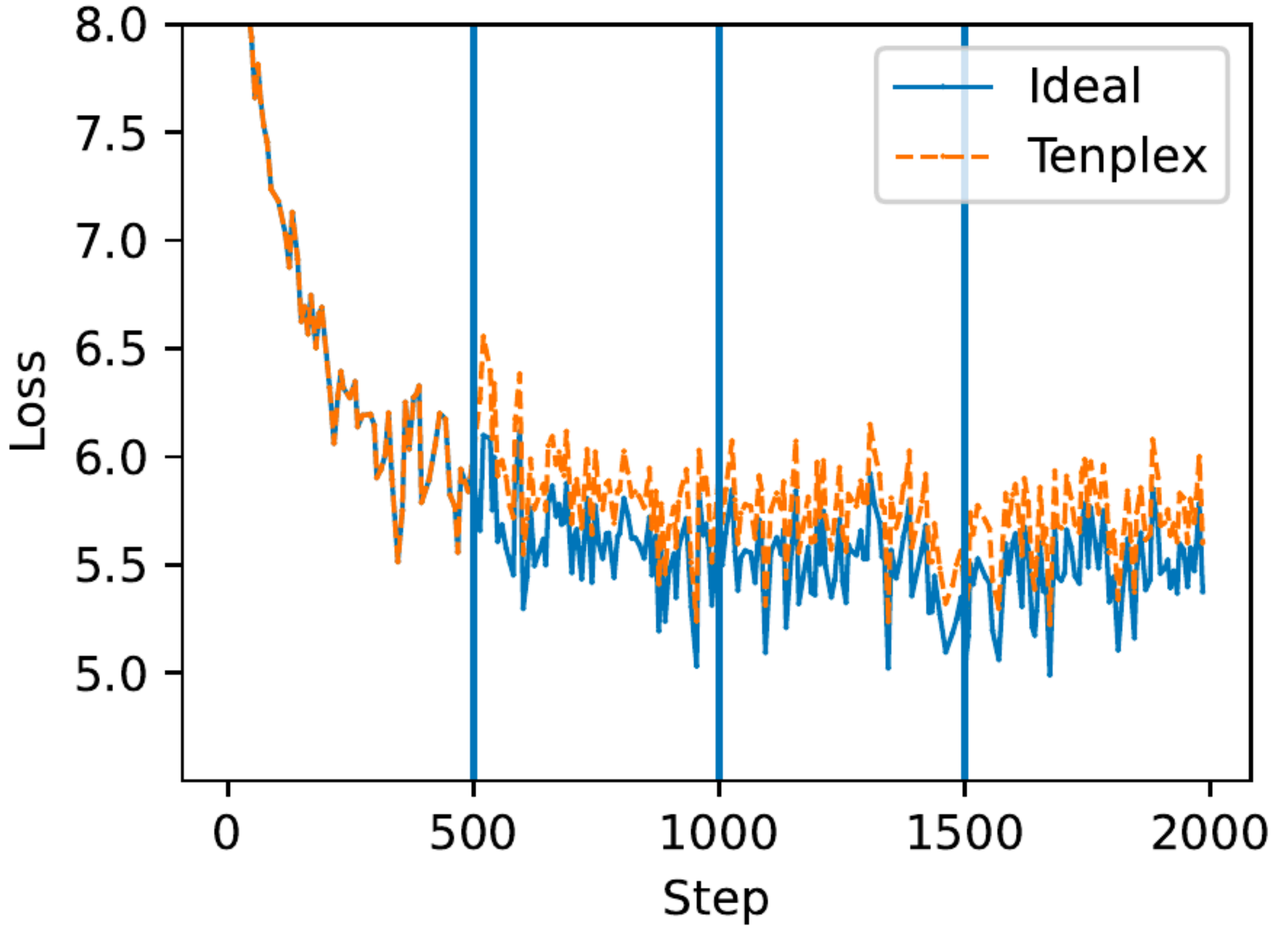
- Model: BERT-large
- Data: OpenWebText
- System: Tenplex + Megatron-LM
- Parallelsim config
  - Step 0: PP=2, MP=4, DP=2
  - Step 500: PP=2, MP=2, DP=2
  - Step 1000: PP=4, MP=2, DP=1
  - Step 1500: PP=6, MP=2, DP=1



Resource change with MDP

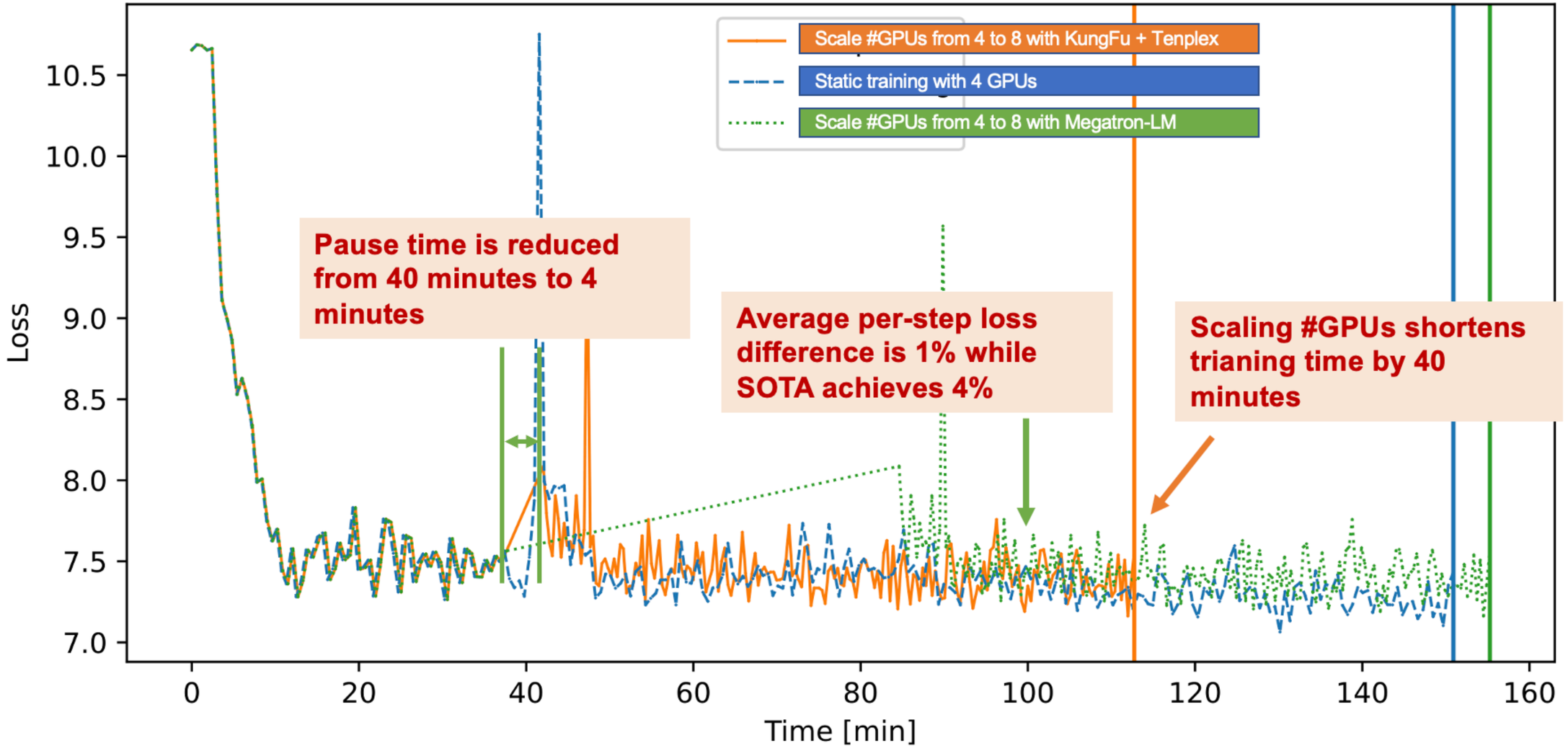
Preliminary results (small cluster, 20 A6000 GPUs)

- Model: GPT-3 6.7B
- Data: En-Wiki
- System: Tenplex + DeepSpeed
- Parallelsim config
  - Step 0: PP=3, MP=2, DP=2
  - Step 500: PP=6, MP=2, DP=1
  - Step 1000: PP=2, MP=4, DP=1
  - Step 1500: PP=4, MP=2, DP=1



End-to-end training efficiency

BERT-large (6.7B), OpenWebtext



Future Plans

Post Moore Era Data-Intensive Systems

Post-Moore era data-intensive systems

Gordon Earle Moore
January 3, 1929 – March 24, 2023

Picture source: Financial times

Post-Moore era data-intensive systems

Holistic Data Processing Pipeline

Scalable Machine Learning System

Post-Moore era data-intensive systems

Scalability

Efficient Hybrid Classic/Quantum

Holistic Data Processing Pipeline

Scalable Machine Learning System

Post-Moore era data-intensive systems

Efficient Hybrid Classic/Quantum

Scalability

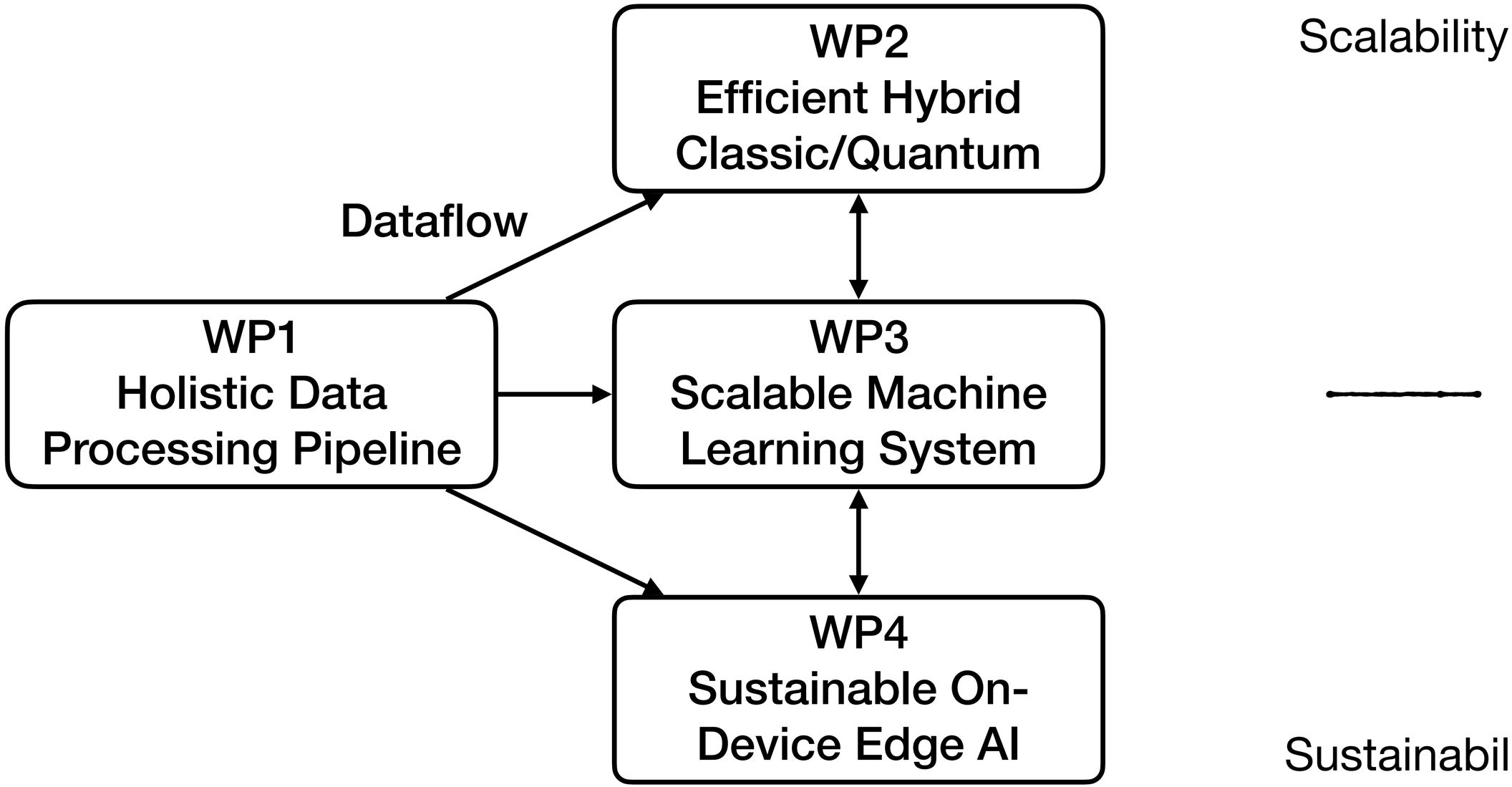
Holistic Data Processing Pipeline

Scalable Machine Learning System

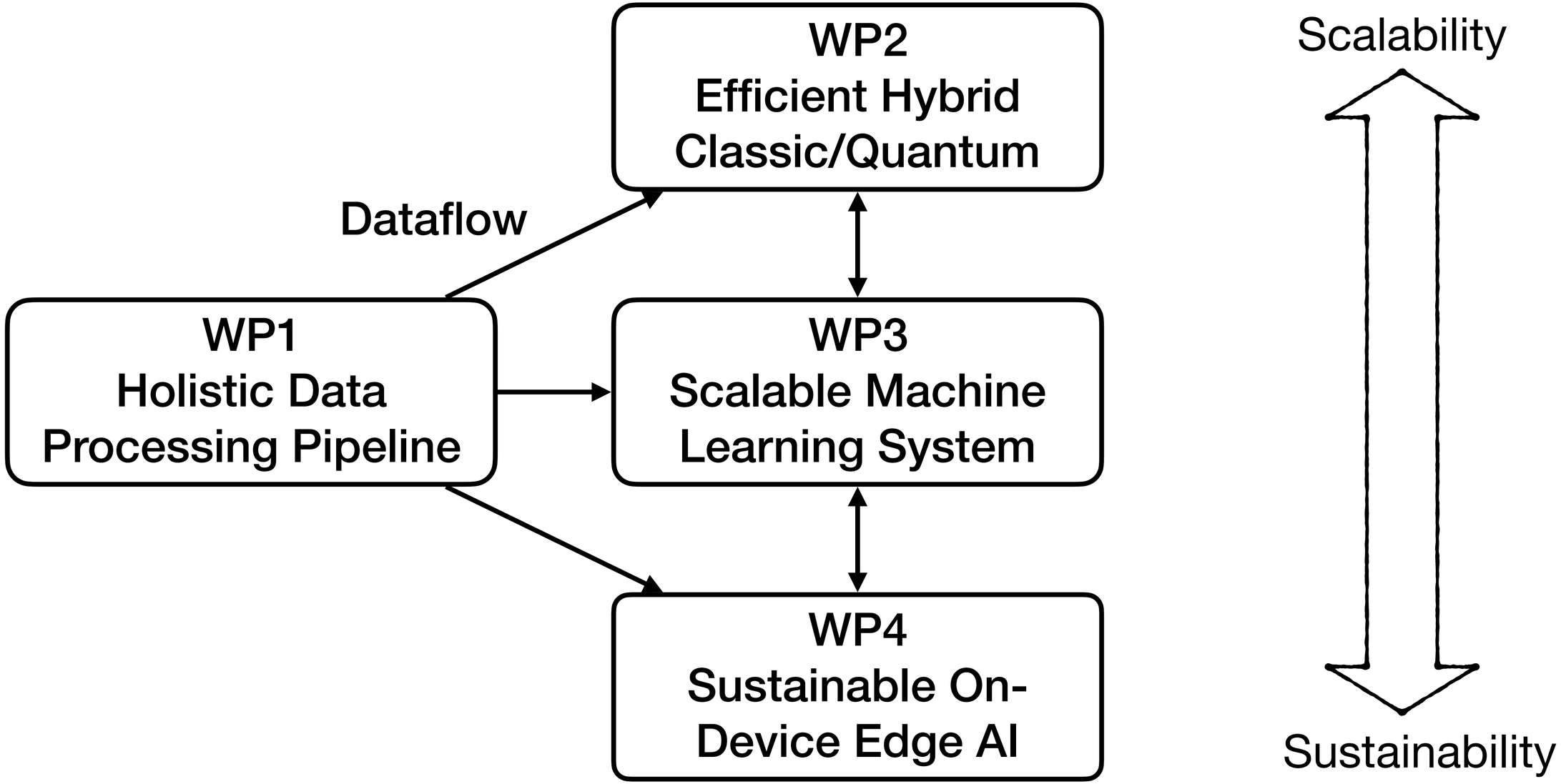
Sustainable On-Device Edge AI

Sustainability

Post-Moore era data-intensive systems



Post-Moore era data-intensive systems



Post-Moore era data-intensive systems

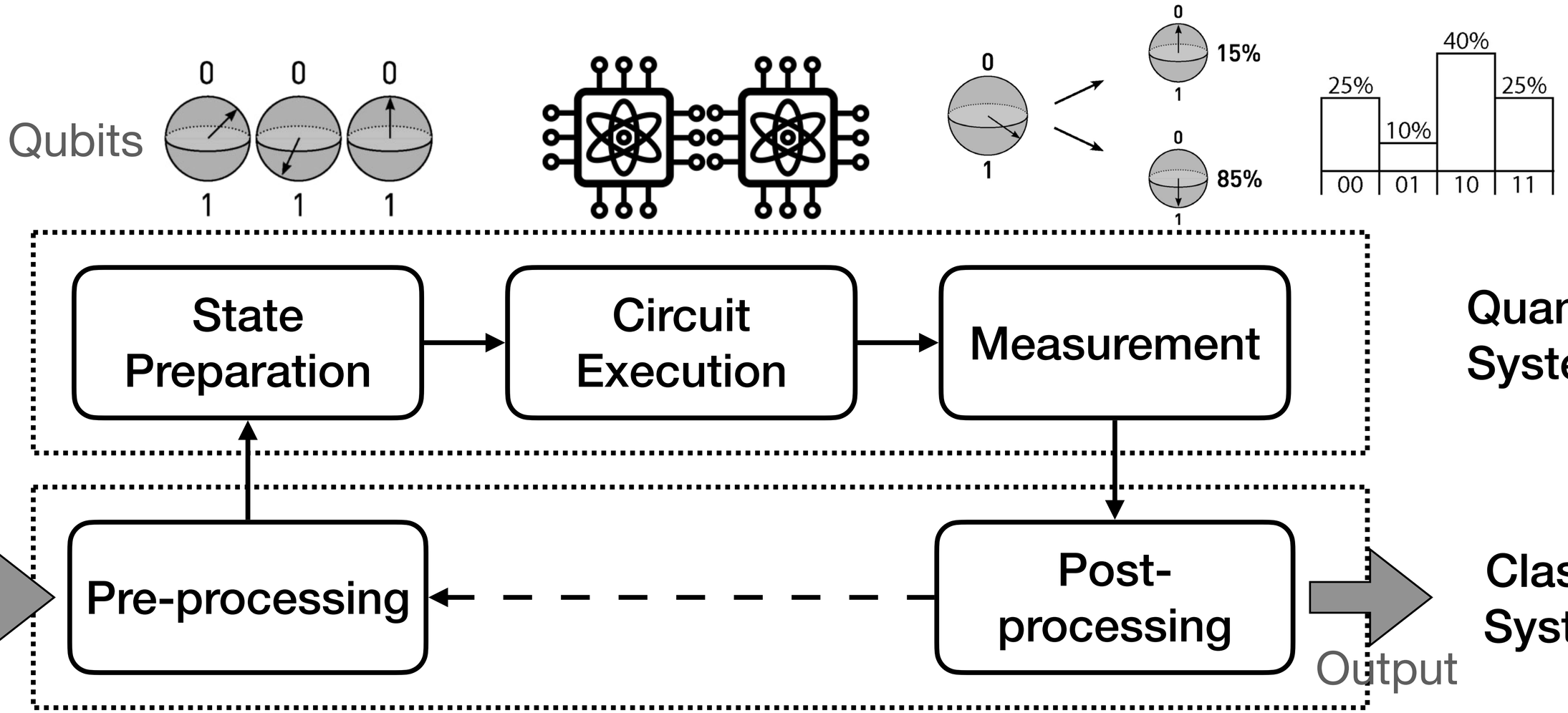
WP2: Efficient Hybrid Classic/Quantum Systems

Why hybrid systems?

- Quantum computing shows the potential to offer a significant computational advantage over classic systems (*i.e.* Von-Neumann's architectures)
- Limited capabilities of current noisy intermediate-scale quantum (NISQ) devices require quantum computers to interoperate with classic systems (*i.e.* error correction)
- Quantum computing can only solve limited problems

Post-Moore era data-intensive systems

WP2: Efficient Hybrid Classic/Quantum Systems



Place the right data at right place and right time

Avoid classic and quantum systems waiting for each other

GPU instances

Instance	Core(s)	RAM	Temporary storage	GPU	Pay as you go	1 year savings plan	3 year savings plan	Spot
NC24ads A100 v4	24	220 GiB	1,123 GiB	1X A100	$3.7020/hour	$3.0864/hour ~16% savings	$2.2949/hour ~38% savings	$1.4982/hour ~59% savings

quantum instances

SKU Quota per month	SOLVERS	PERFORMANCE	PRICE (GLOBAL) Price per compute hour, billed per second
Performance at scale 1,000 hours of compute (default) 50,000 hour of compute (max)	Simulated annealing Simulated annealing (Parameter-free) Parallel tempering Parallel tempering (Parameter-free) Quantum Monte Carlo Tabu search	Up to 100 concurrent jobs	0-1 hour - Free 1-100 hours - $90/hour 100-500 hours - $80/hour 500-1,000 hours - $70/hour > 1,000 hours - $50/hour

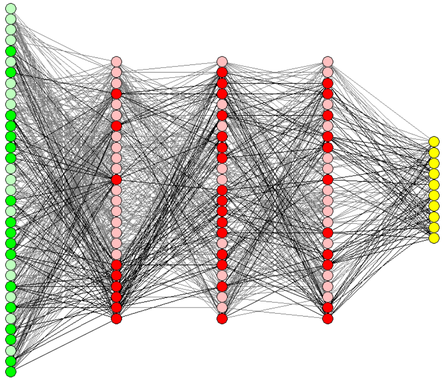
Microsoft Azure cloud

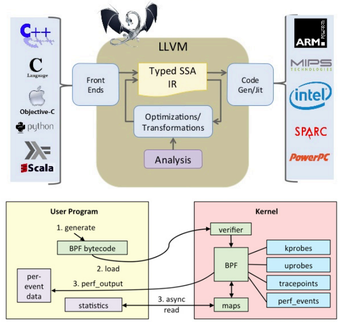
Post-Moore era data-intensive systems

WP2: Efficient Hybrid Classic/Quantum Systems

- Challenges
  - Learning-based quantum code synthesis (problem mapping)
  - Quantum intermediate representation
  - Scalable dataflow optimisation for classic/quantum interaction loop
  - Efficient data pre-processing for quantum state

Acknowledgement

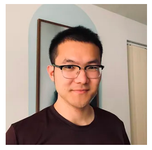




Machine learning layer

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

PEKING UNIVERSITY 1898



Data management layer

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

NUS National University of Singapore

Griffith UNIVERSITY Queensland, Australia

IN OMNIBUS VERITAS UNIVERSITÄT MANNHEIM

Code optimisation layer

RICE UNIVERSITY

Georgia Institute of Technology

TECHNISCHE UNIVERSITÄT DARMSTADT

Imperial College London

HUAWEI [M]S MindSpore

HUMBOLDT-UNIVERSITÄT ZU BERLIN

aws

amazon REDSHIFT

XI'AN JIAOTONG UNIVERSITY 1896

RWTH AACHEN UNIVERSITY

intel

CnC

Conclusion

- Extracting value from data needs holistic end-to-end approaches
- co-design multiple layers of the software stack to improve scalability and performance

Scalable machine learning layer

Efficient data management layer

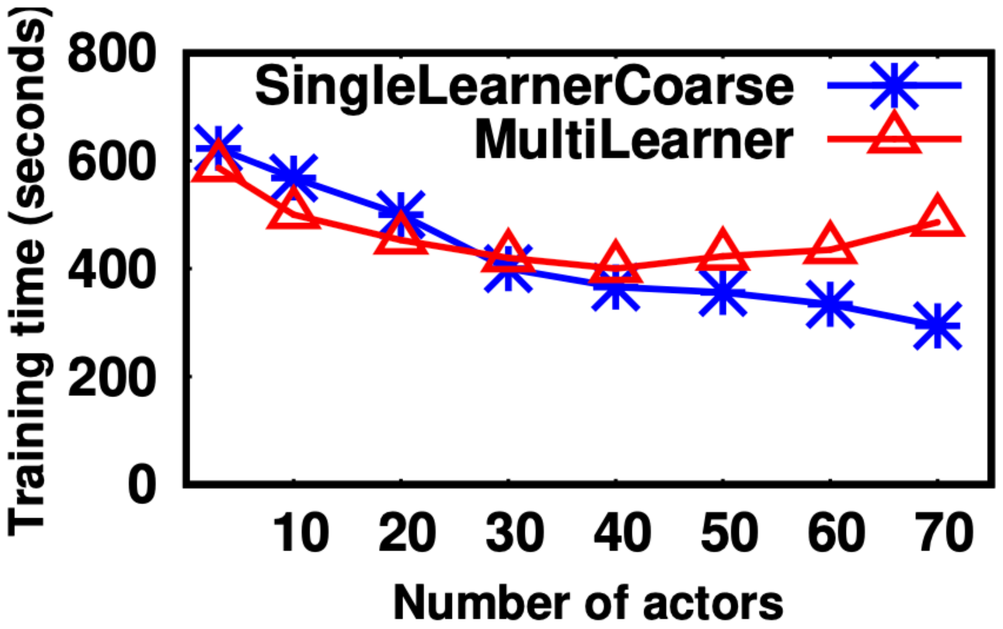
Code optimisation layer

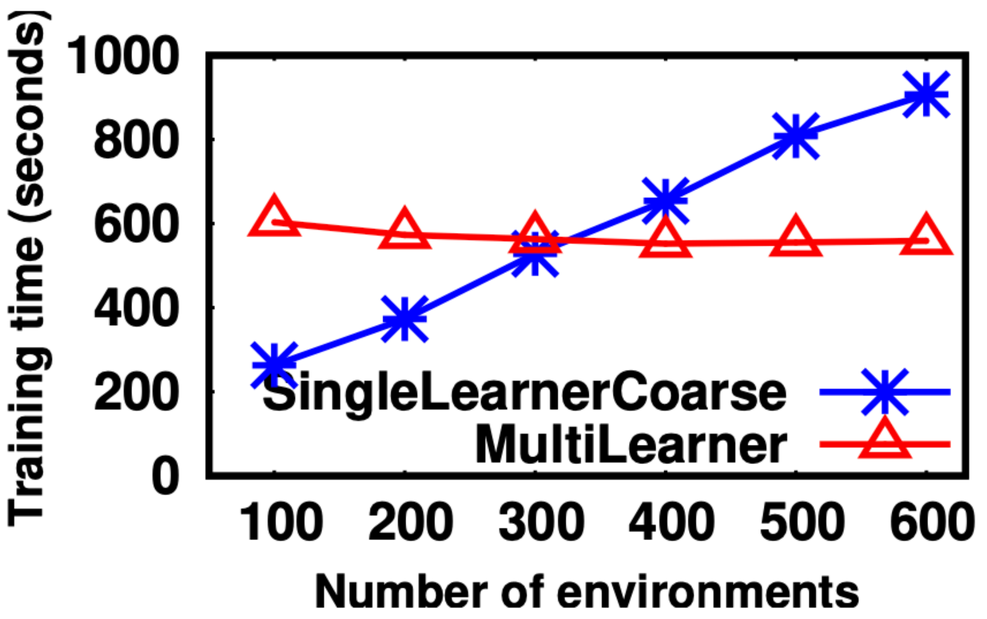
→ Efficient data-intensive computing systems

Backup slides

Key results

Impact of parameters (#actors, #env) on distribution policies



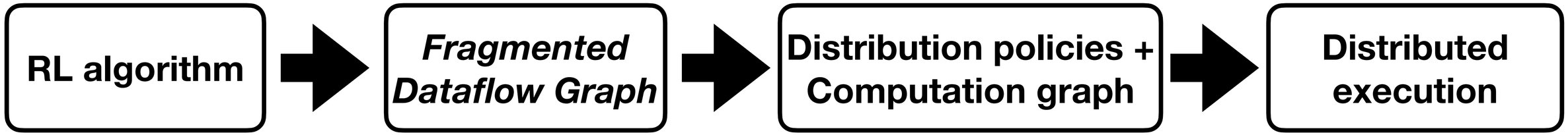


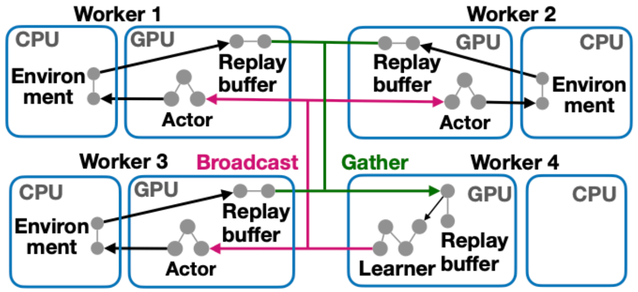
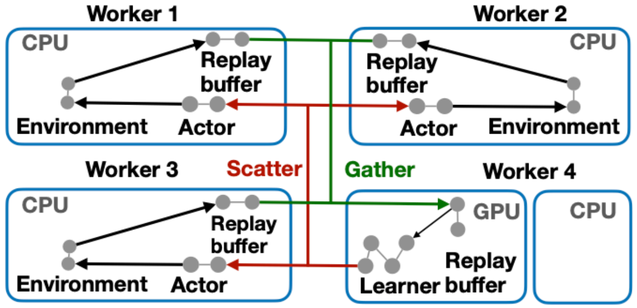
- Hyper-parameters impact the performance of distribution policies
- AthenaRL achieves the best performance in different scenarios without changing the algorithm implementation

AthenaRL Distribution Policies

AthenaRL: Flexible distributed reinforcement learning

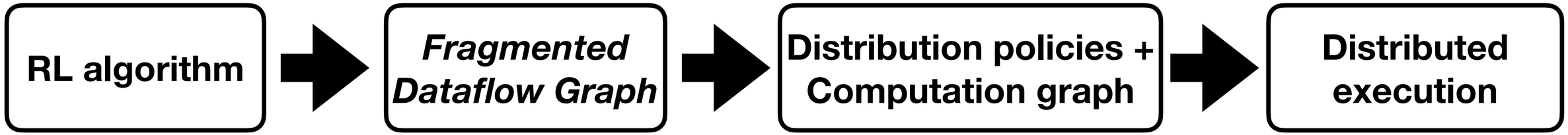
Decouple an RL algorithm implementation and its execution

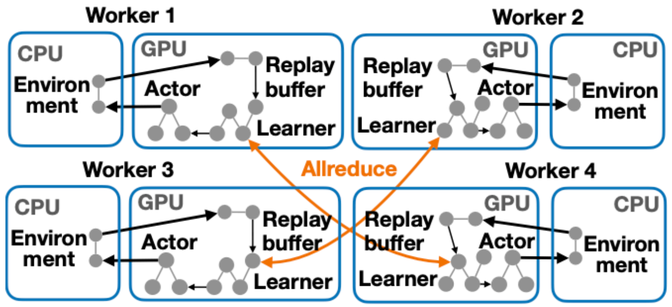
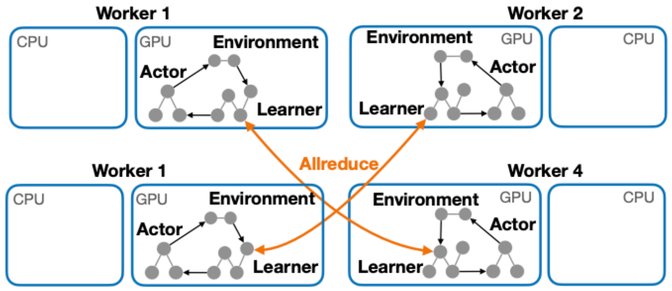


Distribution policy	Deployment	Description
[DP-SingleLearnerCoarse] replicate: (*actor*, *env*) split: (*learner*) e.g., Acme [18], Sebulba [16]		DP-SingleLearnerCoarse replicates the actor and environment fragments: W1–W3 co-locate 1 GPU fragment with an actor for DNN policy inference and 1 CPU fragment for the environment execution. A single GPU fragment with a learner performs policy training (W4), gathering batched training data, training the policy and broadcasting updates.
[DP-SingeLearnerFine] replicate: fused *actor/env* split: *learner* e.g., SEED RL [8]		DP-SingleLearnerFine fuses the actor and environment into 1 fragment (W1–W3) but handles policy inference at the learner (W4), i.e., actors do not contain DNNs. W4 executes policy inference and training in 1 GPU fragment; W1–3 only have CPU fragments. W4 scatters actions to W1–W3 and gathers data for policy training.

AthenaRL: Flexible distributed reinforcement learning

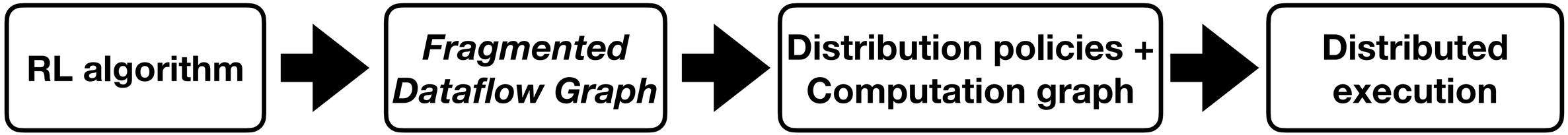
Decouple an RL algorithm implementation and its execution

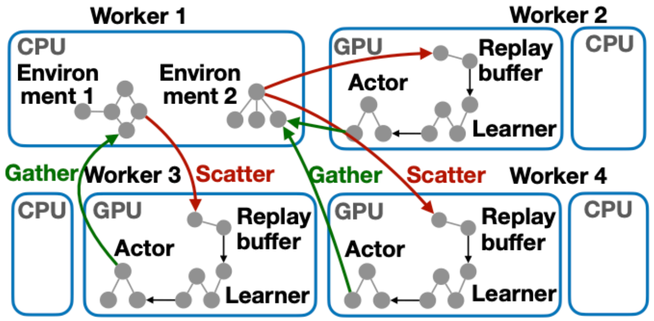
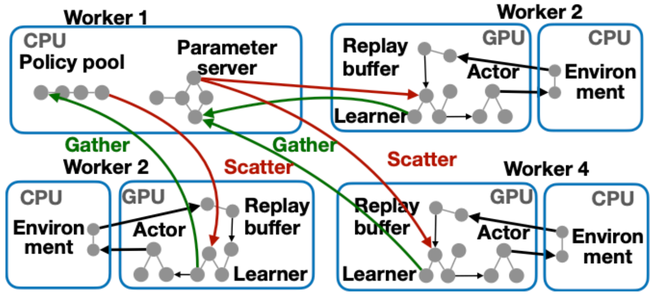


[DP-MultiLearner] replicate: fused *actor/learner*, *env*		DP-MultiLearner performs data-parallel training with multiple learners, supporting fully decentralised MARL training [5, 43, 59, 62]. DP-MultiLearner co-locates 2 fragments: a GPU fragment that fuses the actor and learner, accelerating policy inference, training and replay buffer management, and a CPU fragment for environment execution.
[DP-GPUOnly] replicate: fused *actor/learner/env*		DP-GPUOnly fuses the training loop into 1 GPU fragment. To enable communication among GPU fragments, DP-GPUOnly uses `Allreduce` operators compiled into the computational graph with NCCL2 [39]. DP-GPUOnly is a distributed implementation of the single-node systems (e.g., WarpDrive [23]).

AthenaRL: Flexible distributed reinforcement learning

Decouple an RL algorithm implementation and its execution



[DP-Environments] replicate: fused *actor/learner* split: *env* e.g., MALib [61]		DP-Environments has a dedicated worker for environment execution. W1 has CPU fragments to execute environment instances on multiple CPU cores; W2–W4 fuse the actor and learner to accelerate policy inference and training. W1 gathers the inferred actions and scatters the states and rewards.
[DP-Central] replicate: fused *actor/learner*, *env* split: *param server/policy pool*		DP-Central supports a central *policy pool* [61] or *parameter server* [24] on a separate worker (W1). W2–W4 co-locate GPU fragments for policy inference and training and CPU fragments for environment execution.