The Case for Storage Optimization Decoupling in Deep Learning Frameworks

Ricardo Macedo, Cláudia Correia, Marco Dantas, Cláudia Brito, João Paulo
INESC TEC & University of Minho

Weijia Xu
Texas Advanced Computing Center (TACC)

Yusuke Tanimura, Jason Haga
National Institute of Advanced Industrial Science and Technology (AIST)

IEEE Cluster 2021
1st Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads
Virtual Meeting
September 7, 2021

INESCTEC

Universidade do Minho

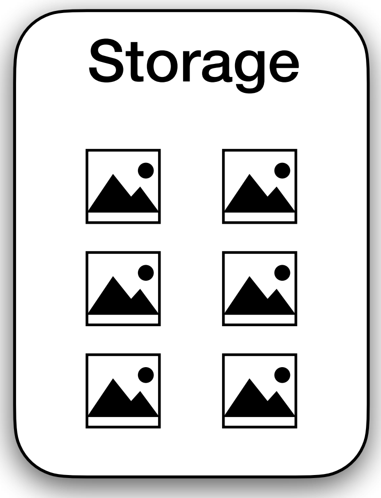
AIST

TACC

Deep Learning

- Extensive research and practical use of DL techniques
- DL models must be trained with **large** and **diverse datasets**
- DL has become prohibitively expensive
  - *Specialized hardware*
  - *Schedulers*
  - Optimizations at *compiler*, *communication*, and *GPU* layers
- **Training bottleneck** has shifted to the **storage** layer

Deep Learning



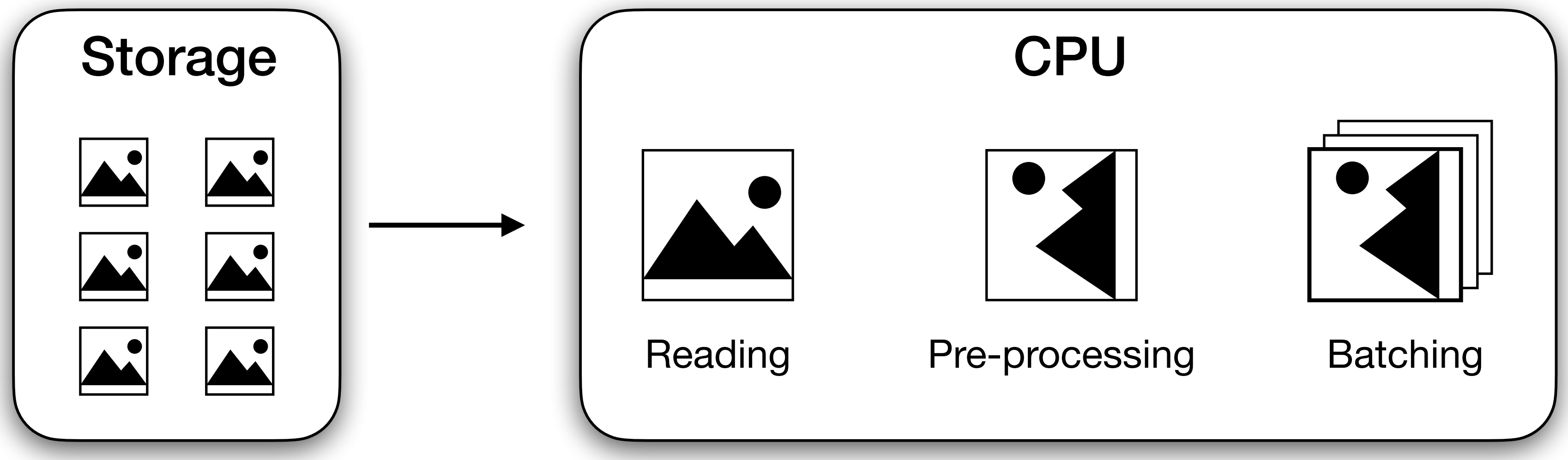
- **Data loading**

  Reading and **preparing data** to be consumed to the GPU

- **Model training**

  Adapt network's parameters to produce accurate predictions

Deep Learning



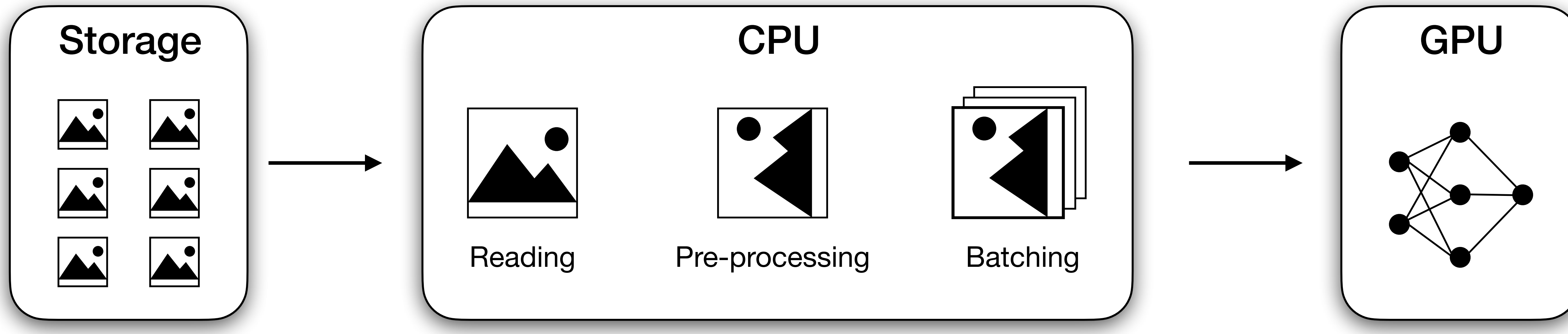
- **Data loading**

  Reading and **preparing data** to be consumed to the GPU

- **Model training**

  Adapt network's parameters to produce accurate predictions

Deep Learning



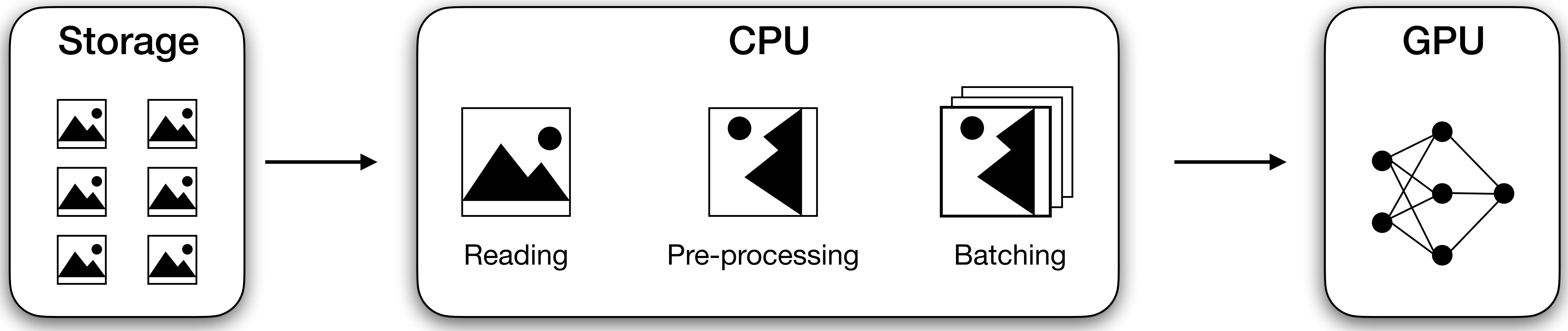
- **Data loading**

  Reading and **preparing data** to be consumed to the GPU

- **Model training**

  Adapt network's parameters to produce accurate predictions

Deep Learning



- **Data loading**

  Reading and **preparing data** to be consumed to the GPU

- **Model training**

  Adapt network's parameters to produce accurate predictions

- **Random access** pattern over **backend storage**

  Challenging to caching and data tiering storage mechanisms

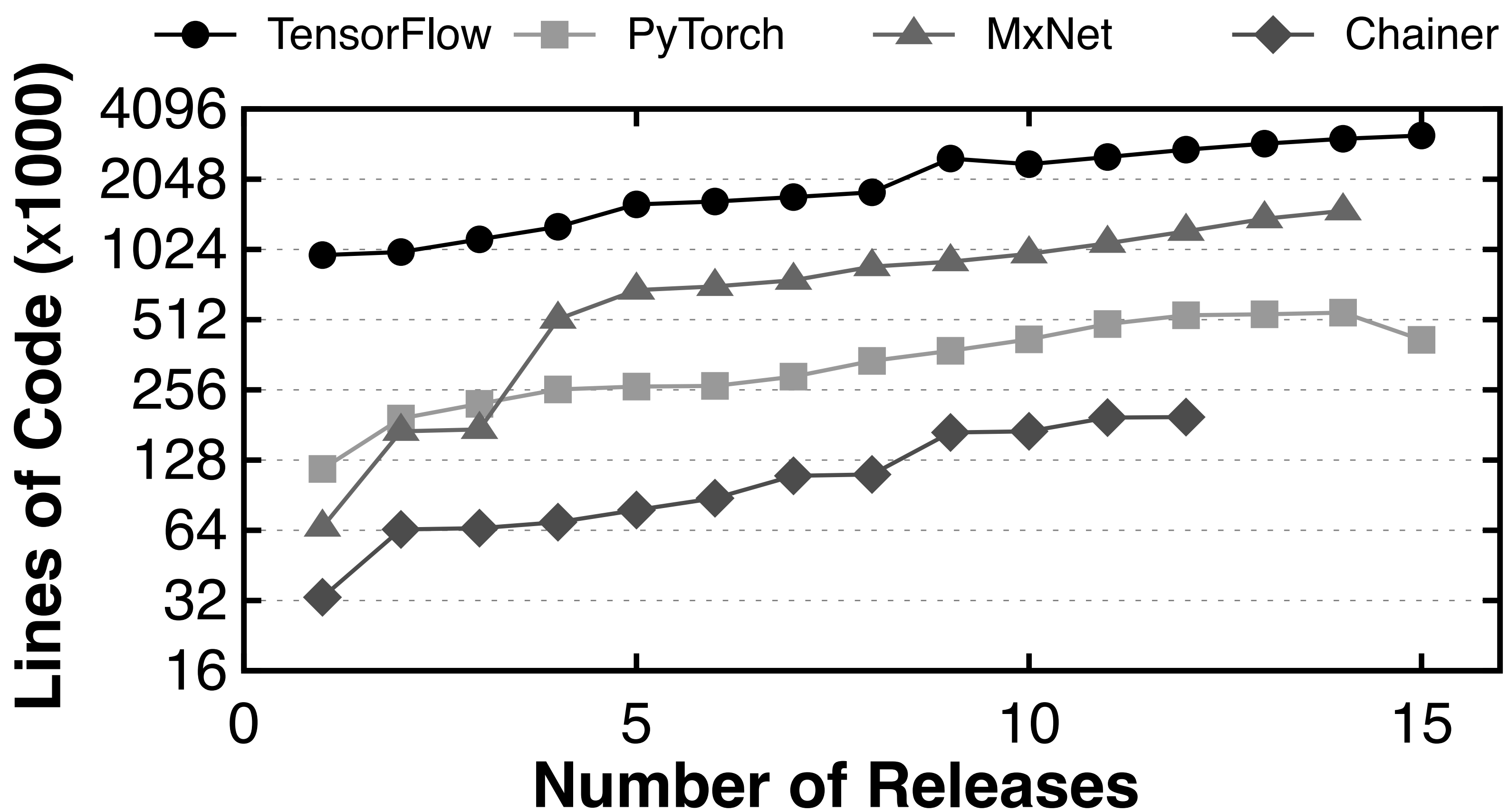
Deep Learning

- System-specific I/O optimizations over DL frameworks
  - *Caching and prefetching*
  - *Storage tiering*
  - *Data sharding*
- This approach comes with **two main drawbacks**
  - *Tightly coupled optimizations*
  - *Partial visibility*

Tightly coupled optimizations

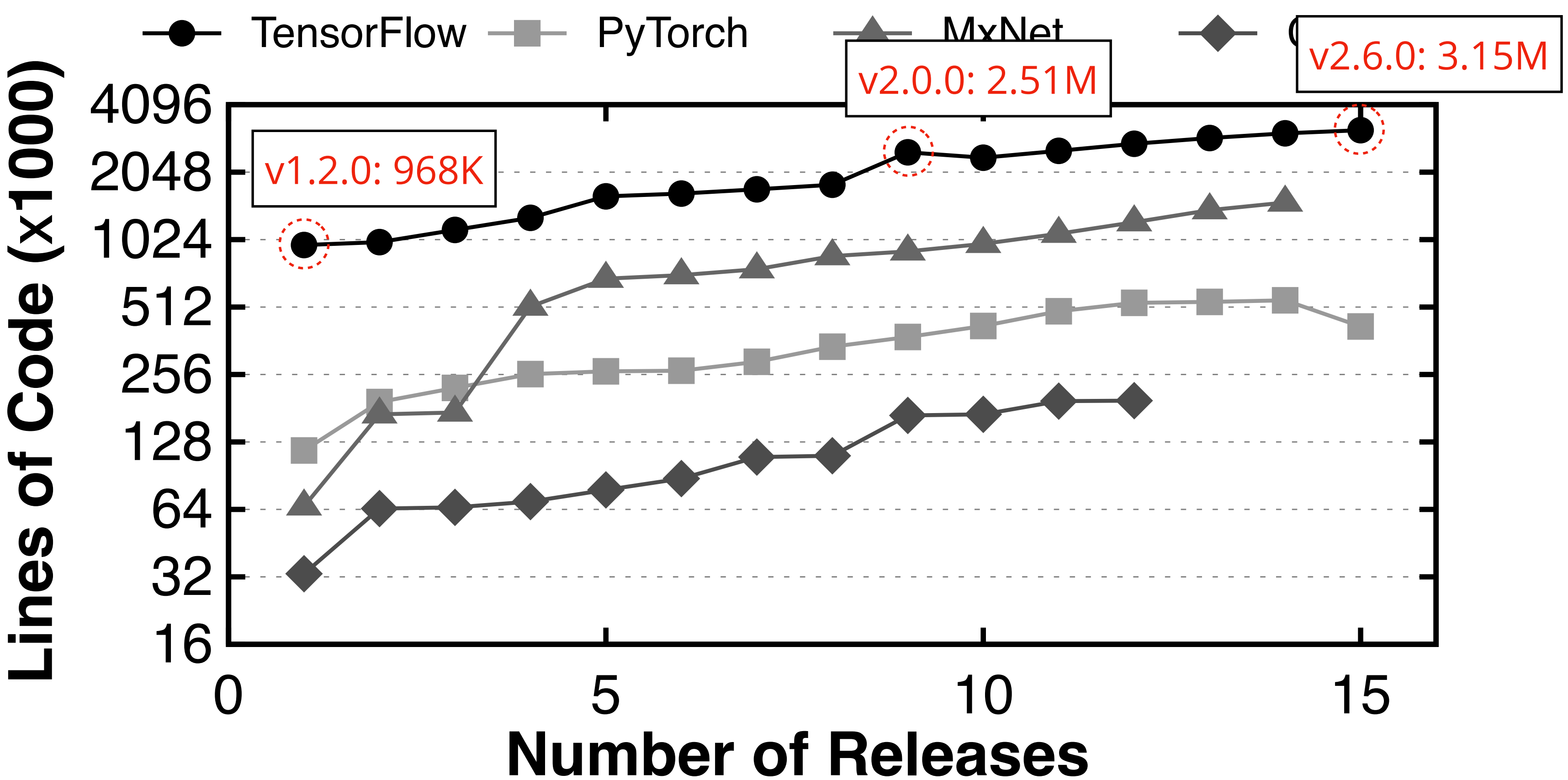
- DL **I/O optimizations** are **framework-specific**
- Require **significant system rewrite**
- Fine-tuning and extension is **complex** and **time-consuming**
- **Reduced portability** and **adoption** over other DL frameworks
    - Porting TensorFlow's **auto-tuning optimization** to **PyTorch** and **Chainer** is not trivial
    - Requires extensive system **expertise**

Tightly coupled optimizations



- Lines of code of 15 minor releases of TensorFlow, PyTorch, MxNet, and Chainer
- Optimizations at internal DL logic, but also at scheduling, GPU, network, and **storage**
- **Porting** and **maintaining** storage optimizations between releases and DL frameworks is **extremely challenging**

Tightly coupled optimizations



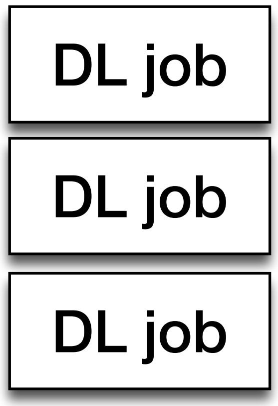
- Lines of code of 15 minor releases of TensorFlow, PyTorch, MxNet, and Chainer
- Optimizations at internal DL logic, but also at scheduling, GPU, network, and **storage**
- **Porting** and **maintaining** storage optimizations between releases and DL frameworks is **extremely challenging**

Partial visibility

- System-specific optimizations are single-purposed
- Act in isolation and are oblivious to the remainder I/O stack
  - *Conflicting optimizations*
  - *I/O contention*
  - *Performance variability*

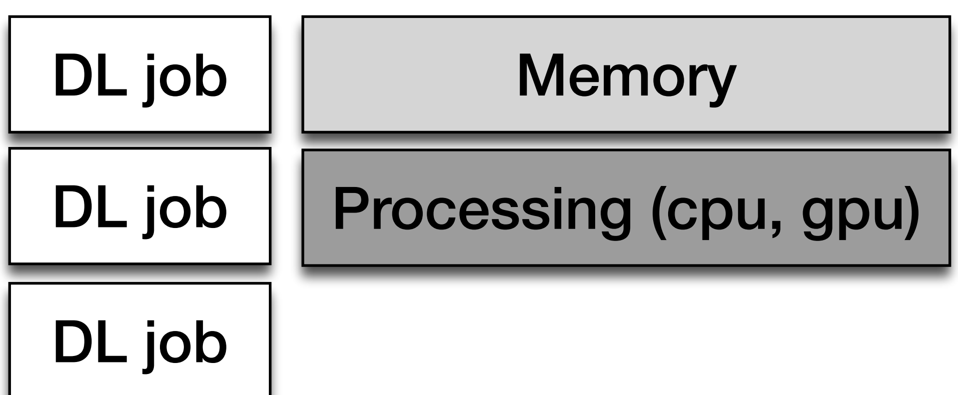
Partial visibility

- System-specific optimizations are single-purposed
- Act in isolation and are oblivious to the remainder I/O stack
  - *Conflicting optimizations*
  - *I/O contention*
  - *Performance variability*



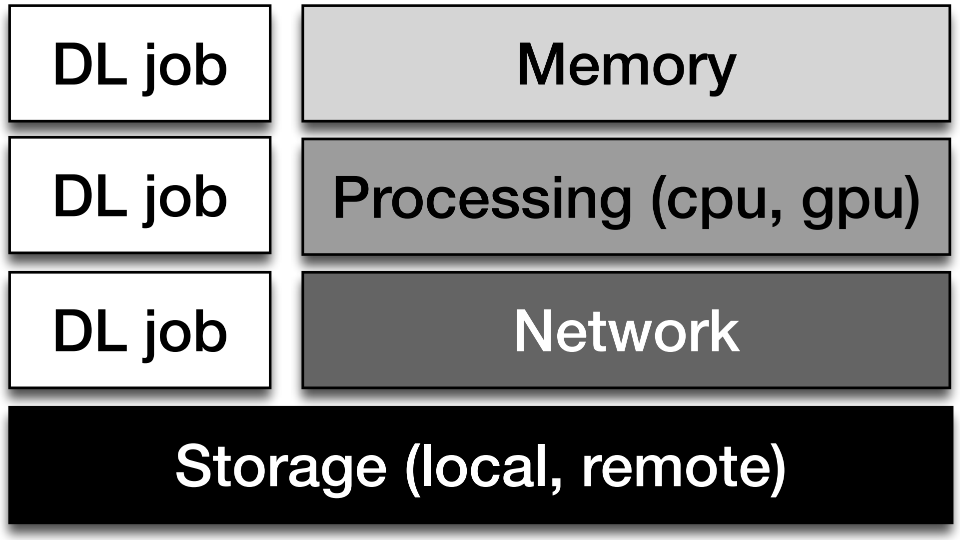
Partial visibility

- System-specific optimizations are single-purposed
- Act in isolation and are oblivious to the remainder I/O stack
  - *Conflicting optimizations*
  - *I/O contention*
  - *Performance variability*



Partial visibility

- System-specific optimizations are single-purposed
- Act in isolation and are oblivious to the remainder I/O stack
  - *Conflicting optimizations*
  - *I/O contention*
  - *Performance variability*



I/O optimizations should be **decoupled** from DL frameworks and **moved** to a **dedicated storage layer** with **system-wide visibility**

Contributions

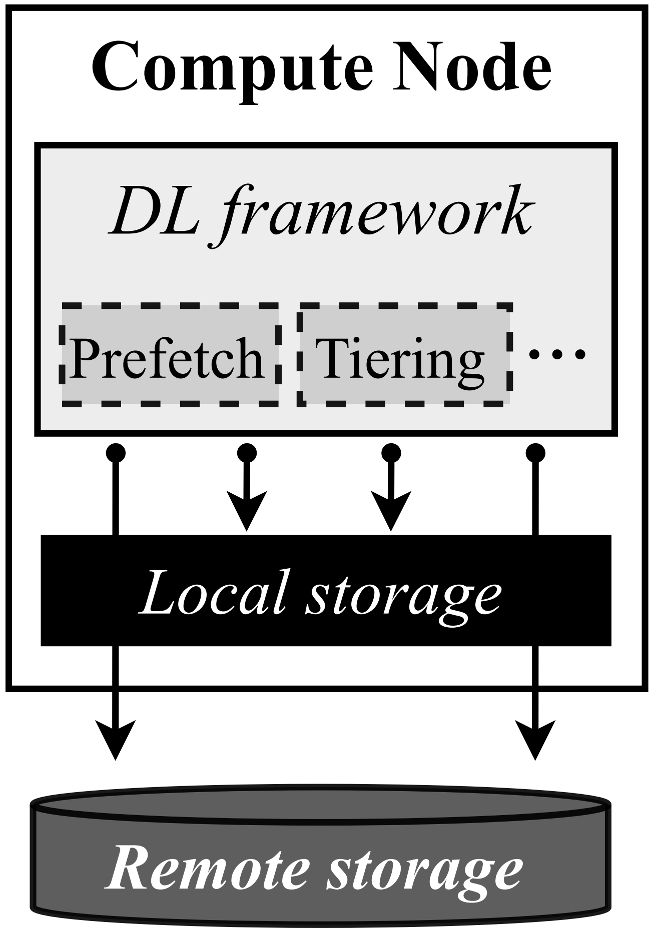
- **Redesign** DL frameworks’ **storage optimizations**
    - *Software-Defined Storage*
- **Middleware** for **accelerating training** performance
    - *PRISMA: framework-agnostic SDS-enabled middleware*
- Integration with **TensorFlow** and **PyTorch**
- Experimental evaluation
    - *Demonstration of the performance and feasibility of PRISMA*

Software-Defined Storage for DL Frameworks

- I/O optimizations are **decoupled** from the DL framework
- **Control plane** holds the control logic
  - Logically centralized
  - **User-defined** policies
  - Orchestrates overall system stack
- **Data plane** implements the I/O logic
  - **Self-contained** and **extensible** building blocks
  - Tuning knobs to **adjust** upon **workload** and **policy variations**
- Implement **generally applicable** I/O optimizations with **system-wide visibility**

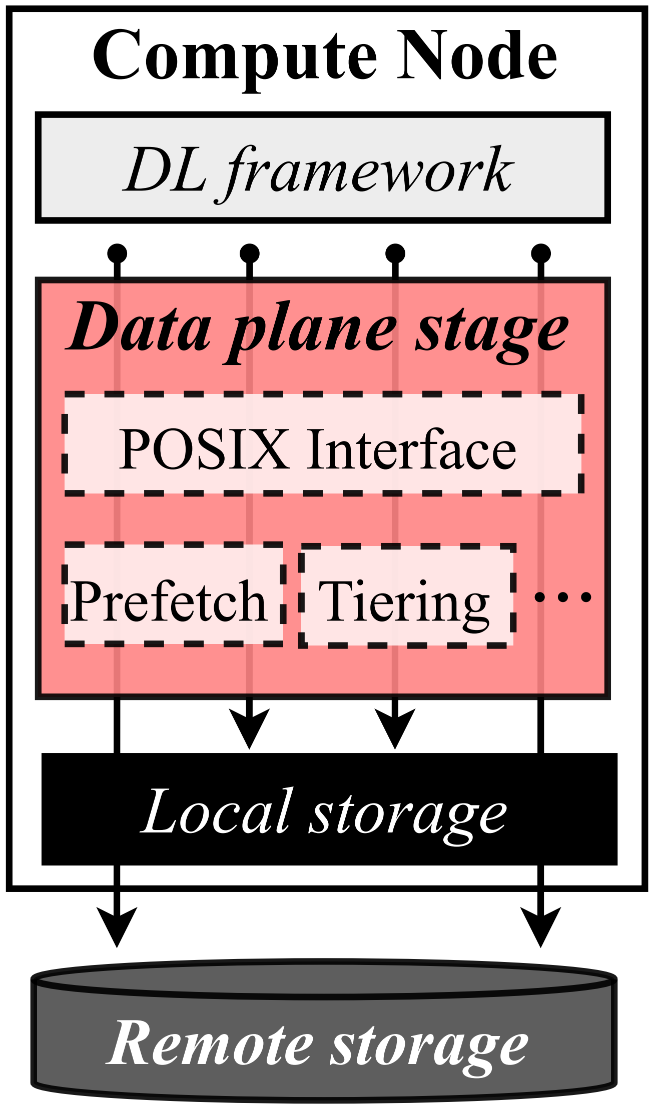
Software-Defined Storage for DL Frameworks

- I/O optimizations
  - Are implemented **internally**
  - Act in **isolation**
  - Are **oblivious** of the remainder layers of the I/O stack



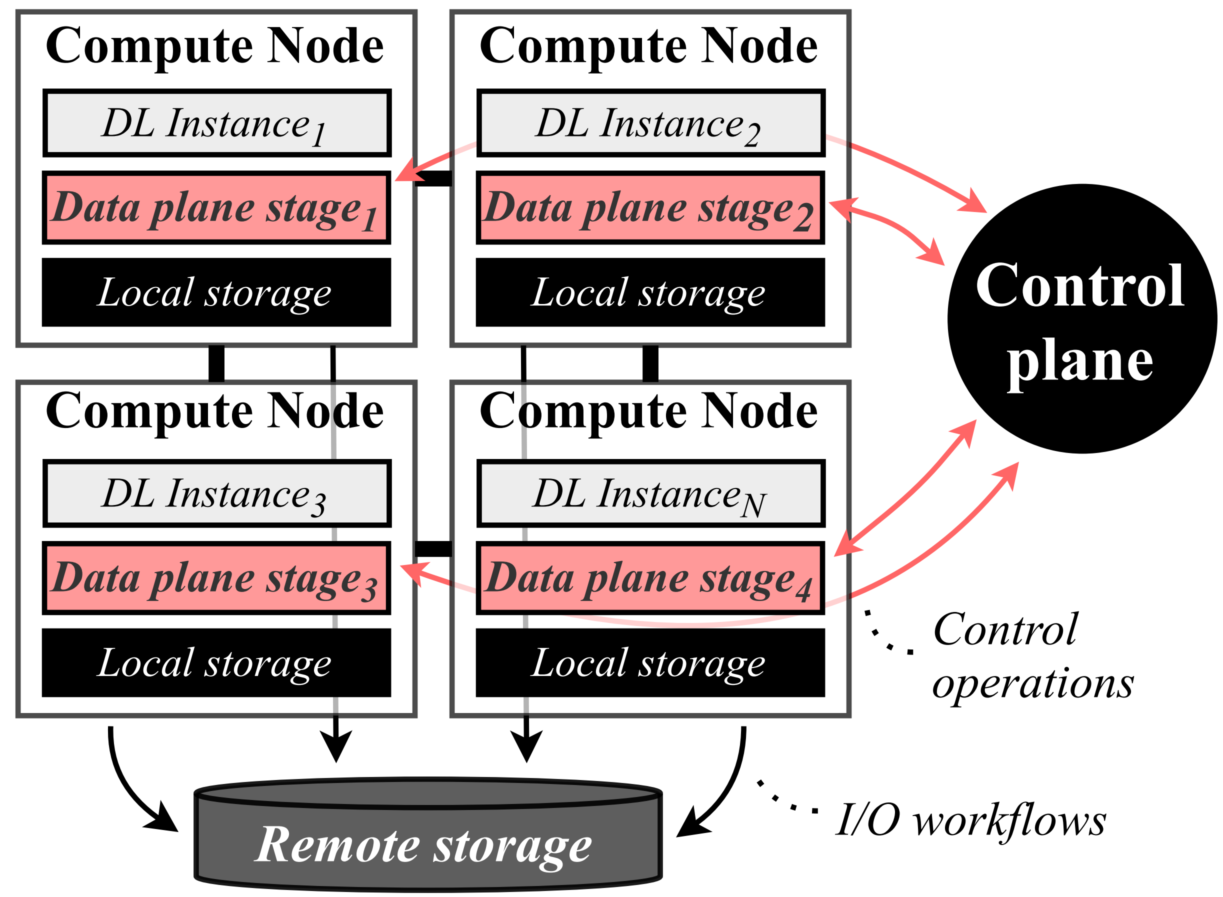
Software-Defined Storage for DL Frameworks

- **Data plane**
  - **Framework-agnostic** middleware
  - Multiple stages
  - **Optimization object** abstraction
  - **POSIX-compliant** interface
  - **Control** interface



Software-Defined Storage for DL Frameworks

- **Control plane**
  - Controls all data plane stages
  - **Centralized control** logic
  - Continuous monitoring
  - **Enforces policies** upon workload variations



PRISMA

- **SDS-enabled** storage middleware
- Implements an **auto-tuned parallel prefetching** mechanism
- Generally applicable I/O optimizations
  - **Parallel I/O** and **data prefetching**
  - Always serve data from high-speed memory
- **Auto-tuning** control algorithm
  - Finds the optimal combination of **parallel reads** and internal **buffer size**
  - Feedback control loop
  - Similar to TensorFlow's auto-tuning mechanism

Integration with DL Frameworks

- **PRISMA** was integrated with **TensorFlow** and **PyTorch**
- TensorFlow
  - Replaces `POSIX.pread` with `Prisma.read`
  - Only required changing **10 LoC**
- PyTorch
  - Inter-process communication client-server with UNIX Domain Sockets
  - Only required changing **35 LoC**

Experimental Evaluation

Dataset, models, and DL frameworks

Imagenet dataset (150GiB)

LeNet, AlexNet, and ResNet-50 models

TensorFlow v2.1.0 and PyTorch v1.7.0

Methodology

10 training epochs

All 4 GPUs were used

Batch sizes: 64, 128, 256

Testbed

1x compute node at AI Bridging Cloud Infrastructure (ABCI) supercomputer

2x 20-core Intel Xeon processors

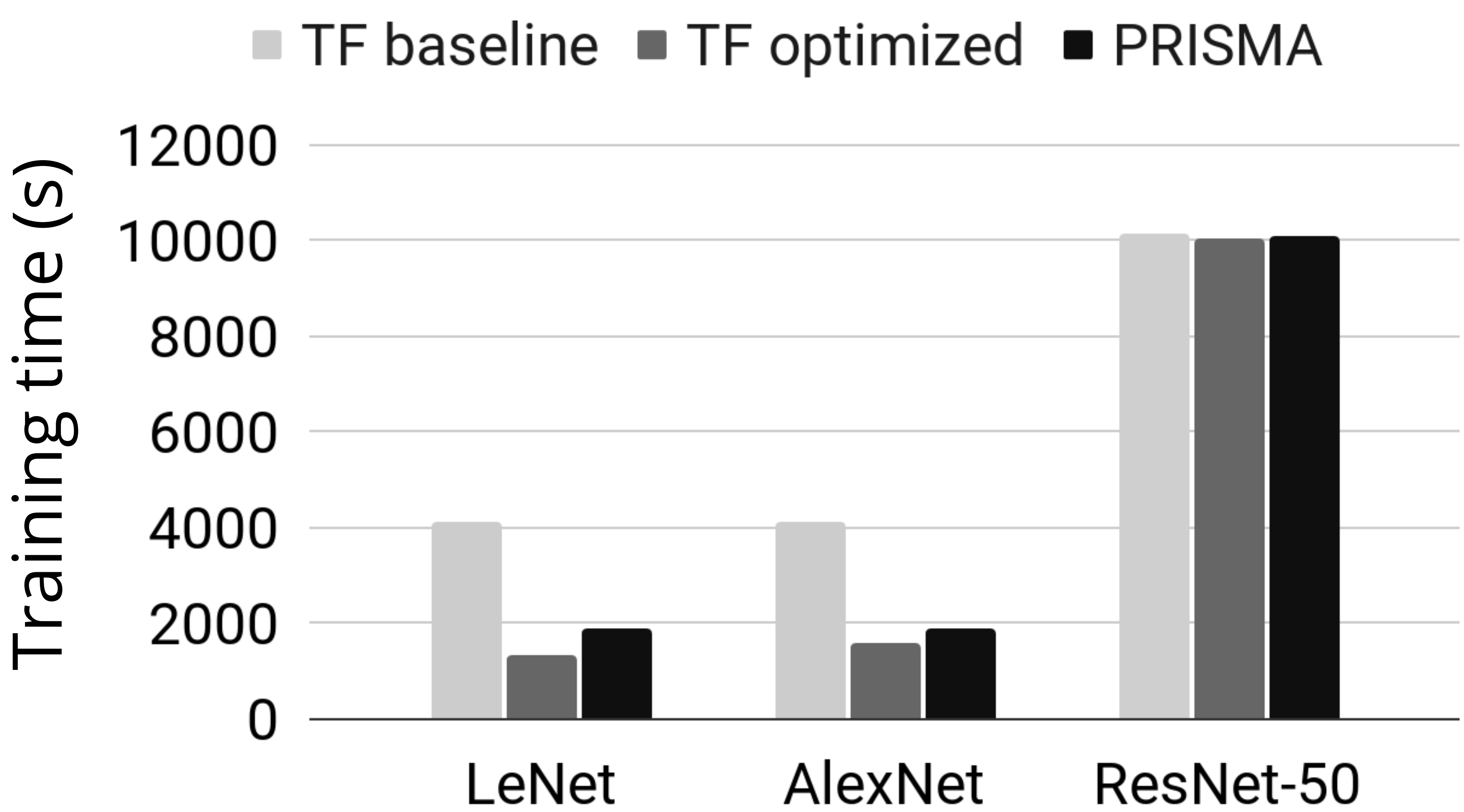
4x NVidia Tesla V100 GPUs

384GiB RAM

1.6TiB Intel SSD DC P4600

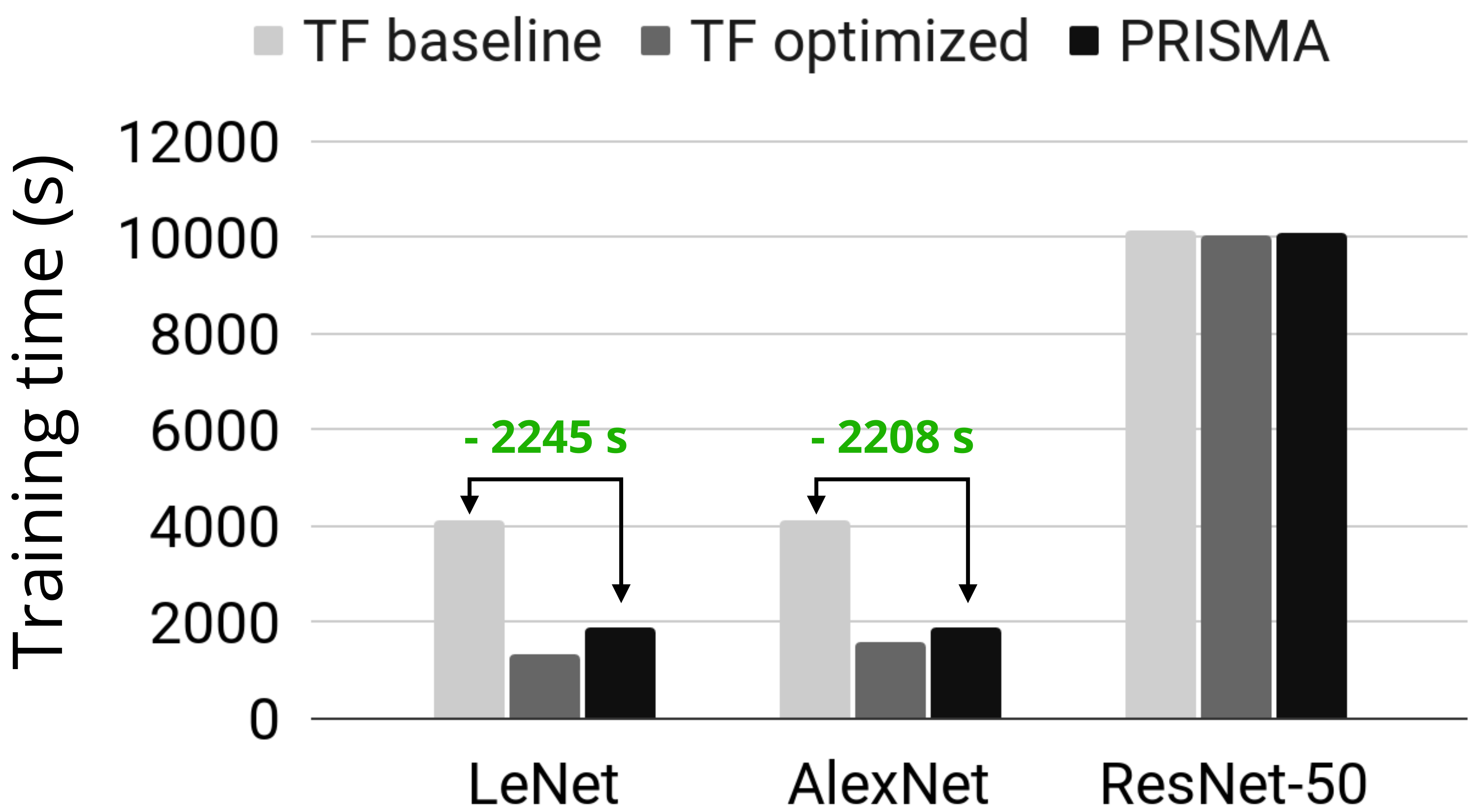
CentOS 7.5 with Linux Kernel 3.10 and XFS file system

Experimental Evaluation: TensorFlow



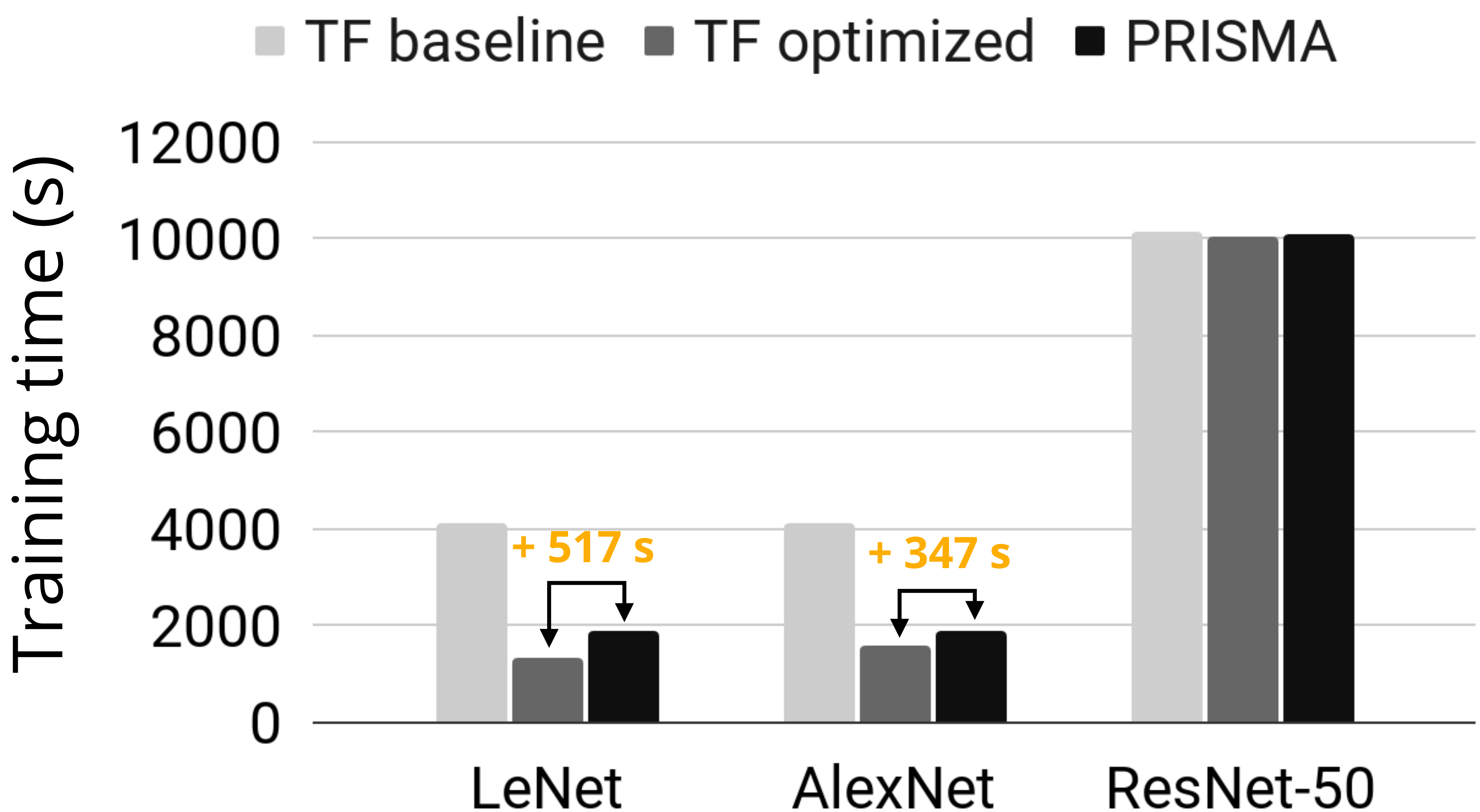
- PRISMA **improves** overall training time in **I/O-bound models**
- PRISMA **does not optimize** the I/O of **validation** files (11% of the dataset)
- PRISMA uses **4 I/O threads**, while TF optimized uses **30**

Experimental Evaluation: TensorFlow



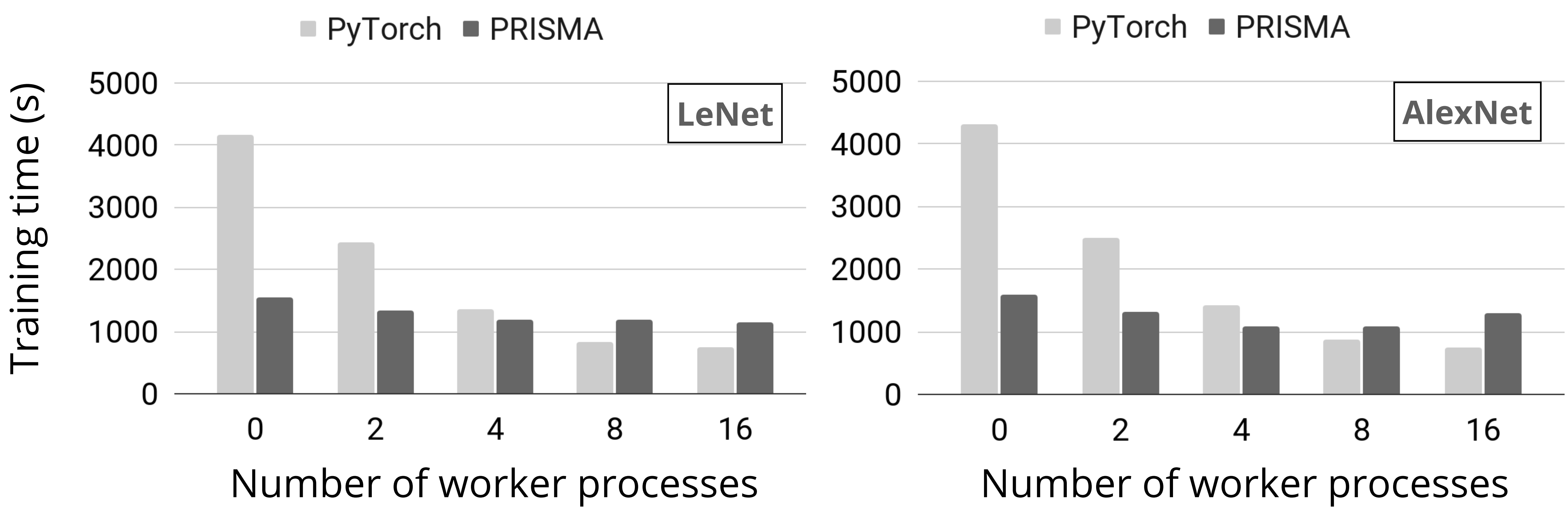
- PRISMA **improves** overall training time in **I/O-bound models**
- PRISMA **does not optimize** the I/O of **validation** files (11% of the dataset)
- PRISMA uses **4 I/O threads**, while TF optimized uses **30**

Experimental Evaluation: TensorFlow



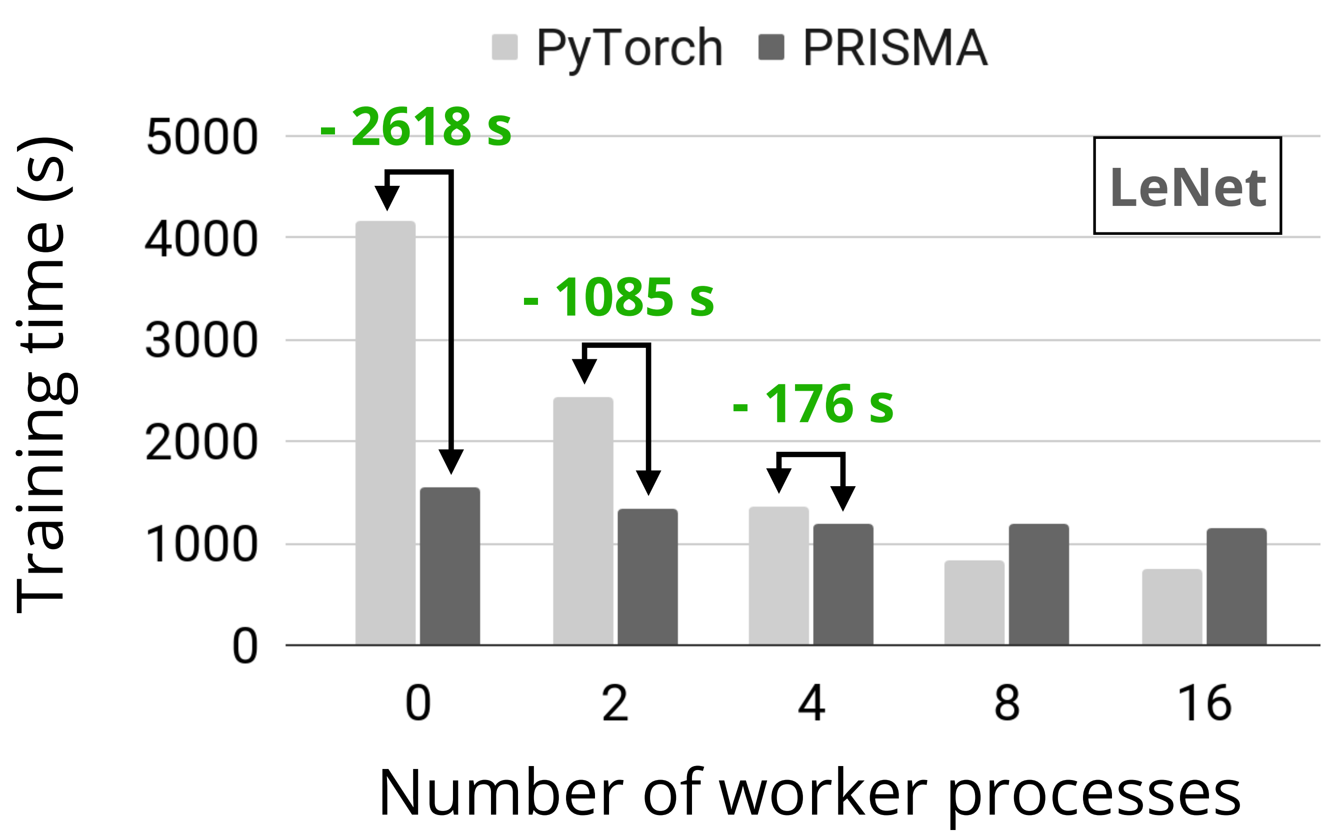
- PRISMA **improves** overall training time in **I/O-bound models**
- PRISMA **does not optimize** the I/O of **validation** files (11% of the dataset)
- PRISMA uses **4 I/O threads**, while TF optimized uses **30**

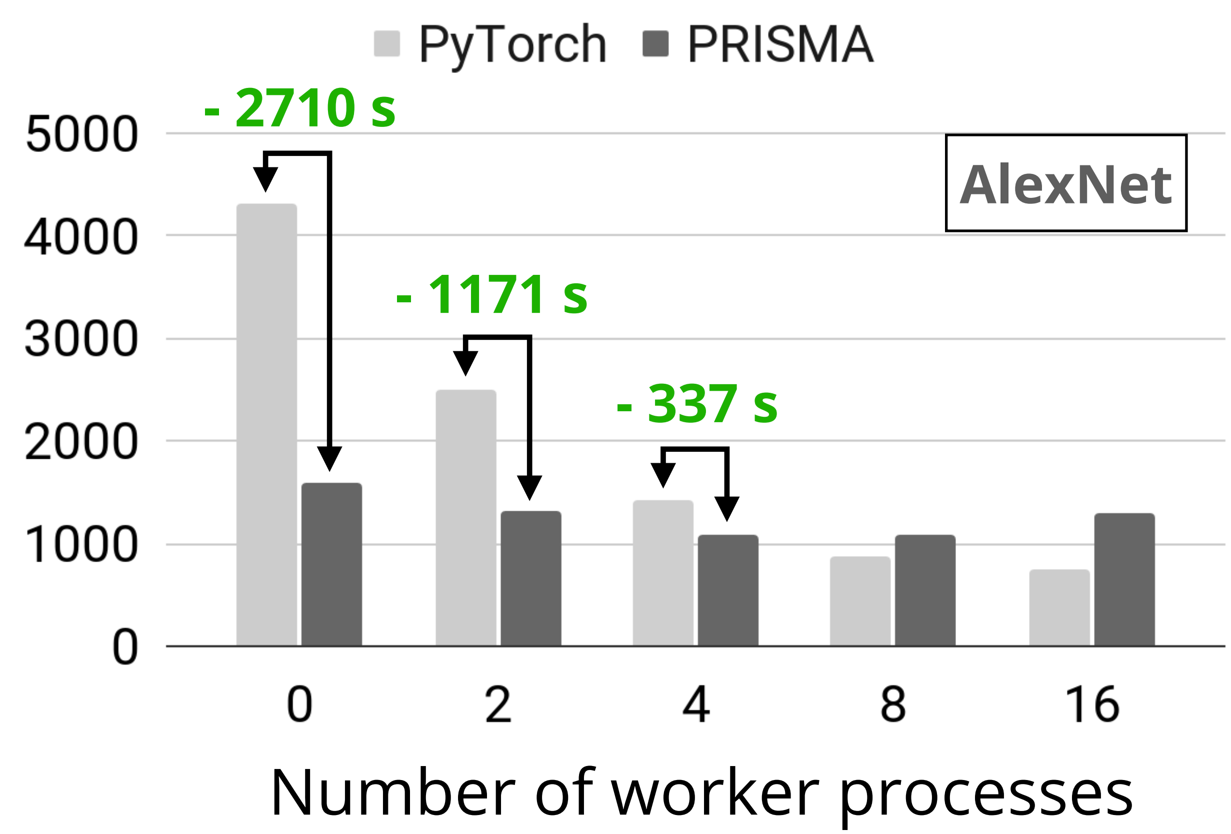
Experimental Evaluation: PyTorch



- PRISMA **outperforms PyTorch** for a lower number of workers
- PRISMA enables the **auto-tuning** mechanism over PyTorch
- PRISMA **concurrency control mechanisms** add small overhead

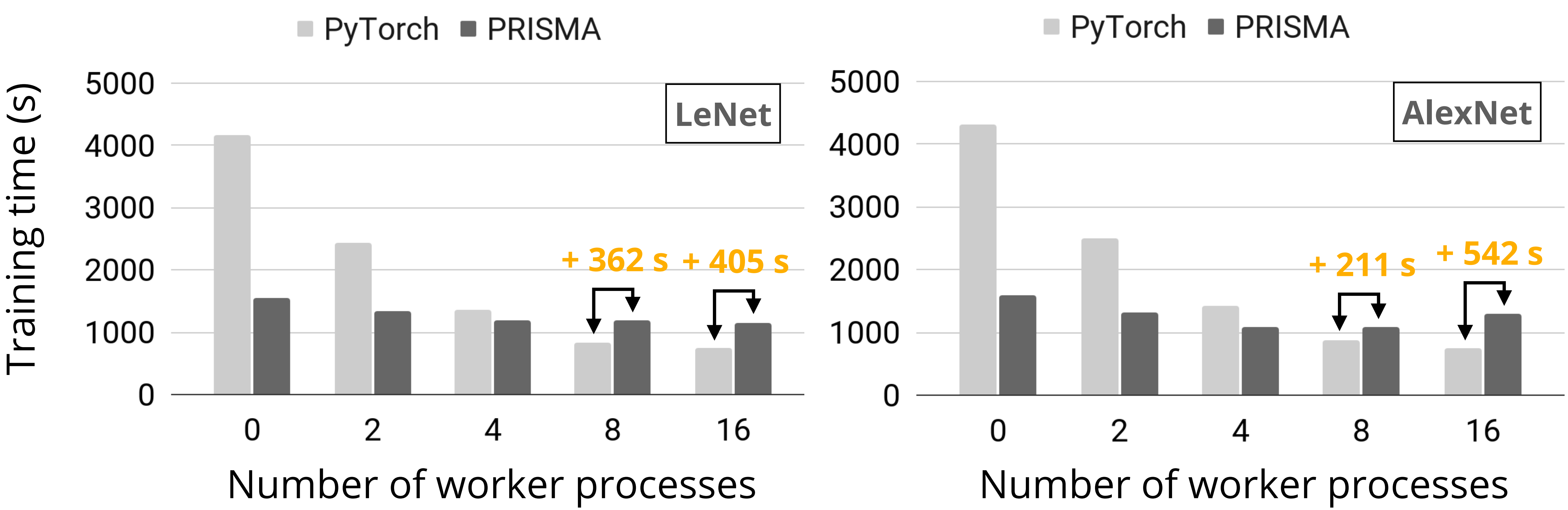
Experimental Evaluation: PyTorch





- PRISMA **outperforms PyTorch** for a lower number of workers
- PRISMA enables the **auto-tuning** mechanism over PyTorch
- PRISMA **concurrency control mechanisms** add small overhead

Experimental Evaluation: PyTorch



- PRISMA **outperforms PyTorch** for a lower number of workers
- PRISMA enables the **auto-tuning** mechanism over PyTorch
- PRISMA **concurrency control mechanisms** add small overhead

Summary

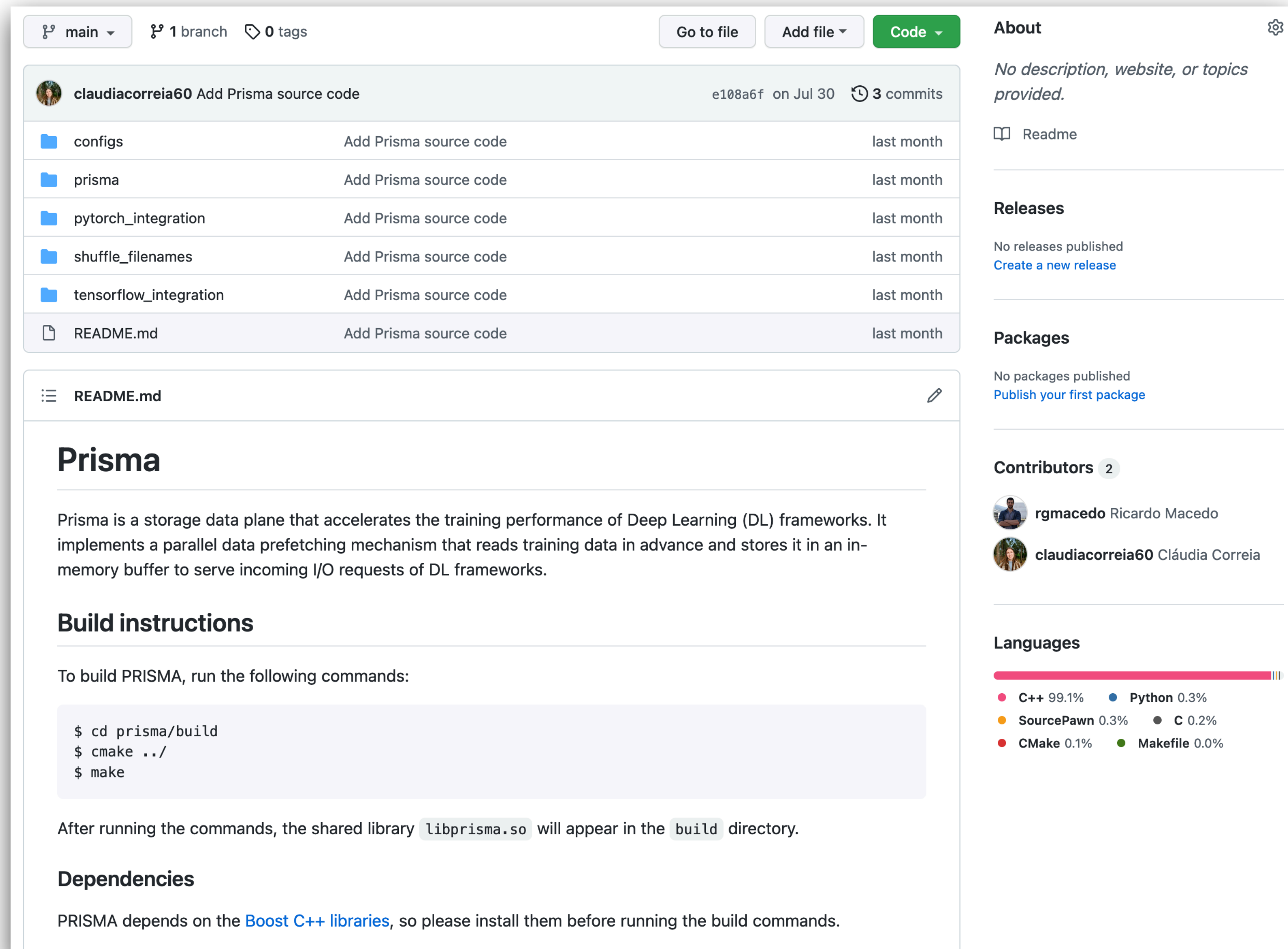
- **Decoupling** I/O optimizations from DL frameworks is **feasible**
- **SDS** architecture for **accelerating DL training** performance
- PRISMA storage middleware
- **Generally applicable** of I/O mechanisms
- **Outperforms** baseline **TensorFlow**
- **Optimizes PyTorch** for a low number of workers

Future Directions

- Implement other I/O optimizations
- Distributed training setting
- Access coordination to shared datasets
- Control plane scalability and dependability

PRISMA is open source!

https://github.com/dsrhaslab/prisma



The Case for Storage Optimization Decoupling in Deep Learning Frameworks

Ricardo Macedo, Cláudia Correia, Marco Dantas, Cláudia Brito, João Paulo
INESC TEC & University of Minho

Weijia Xu
Texas Advanced Computing Center (TACC)

Yusuke Tanimura, Jason Haga
National Institute of Advanced Industrial Science and Technology (AIST)

IEEE Cluster 2021
1st Workshop on Re-envisioning Extreme-Scale I/O for Emerging Hybrid HPC Workloads
Virtual Meeting
September 7, 2021

INESCTEC Universidade do Minho AIST TACC