# CheckFreq
## Frequent, Fine-Grained DNN Checkpointing

**Jayashree Mohan,** Amar Phanishayee, Vijay Chidambaram

Microsoft Research
TEXAS The University of Texas at Austin
vmware

# Deep Neural Networks ( DNNs )

- Widely used for a variety of tasks

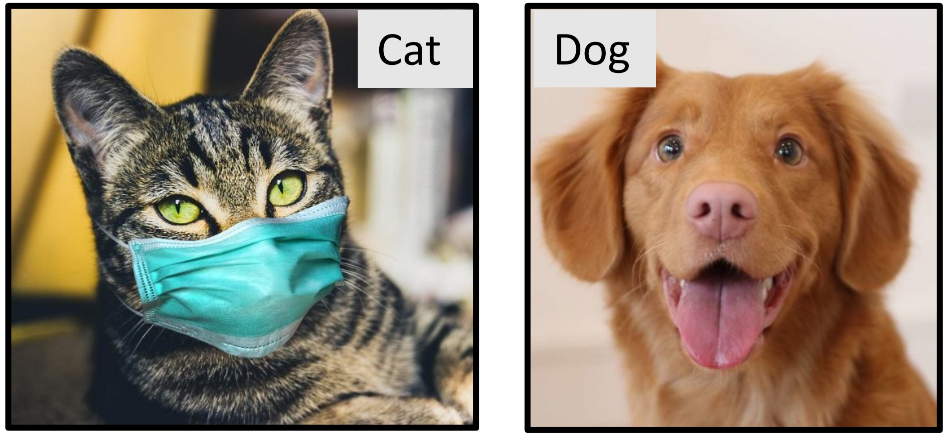
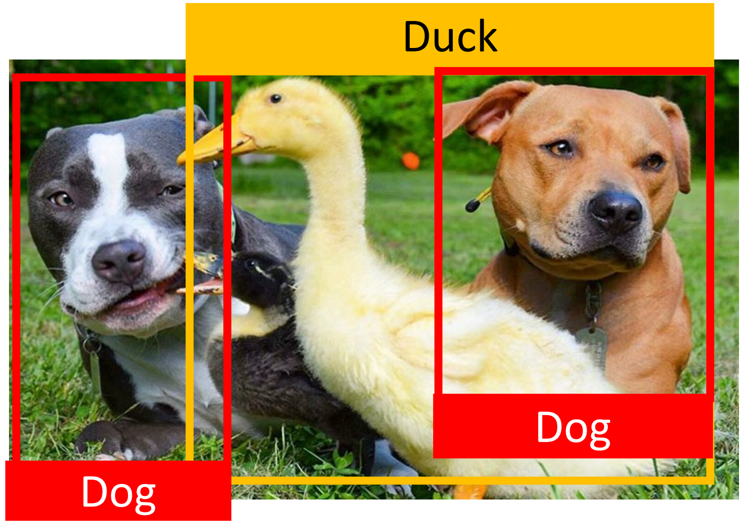
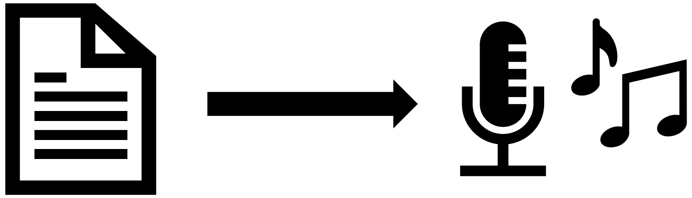


Image Classification

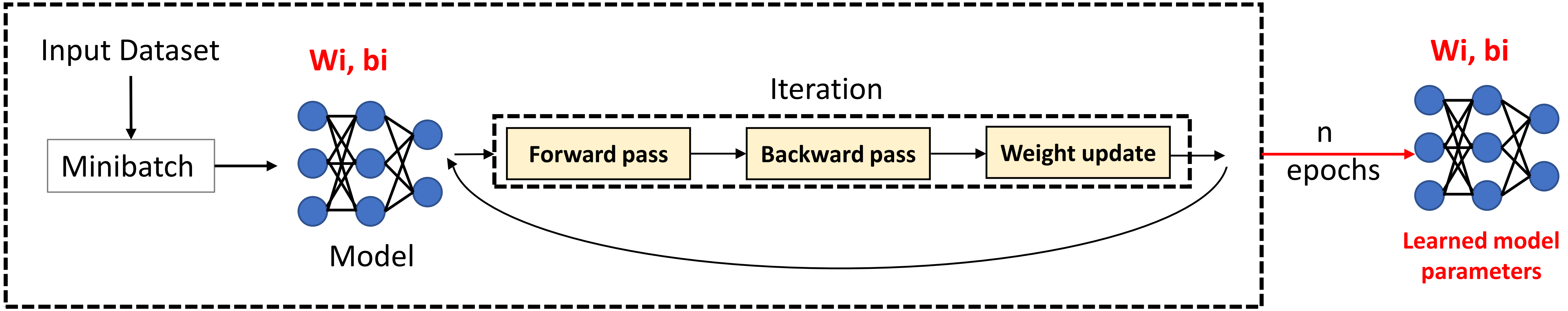


Object detection

Language Translation


Text To Speech

# DNN Training

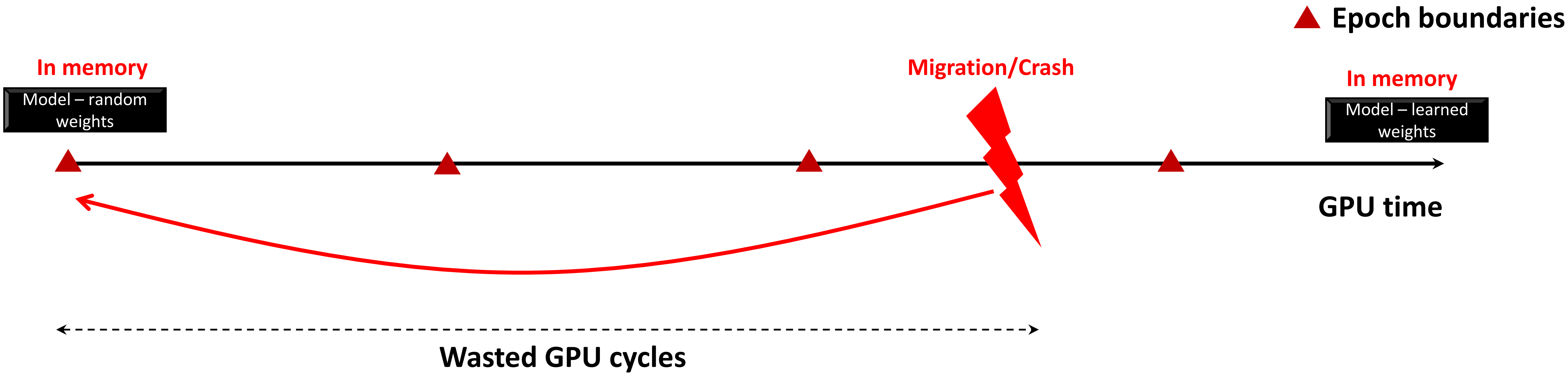



Epoch = One complete pass over the dataset

**DNN training is compute-intensive and time-consuming!**

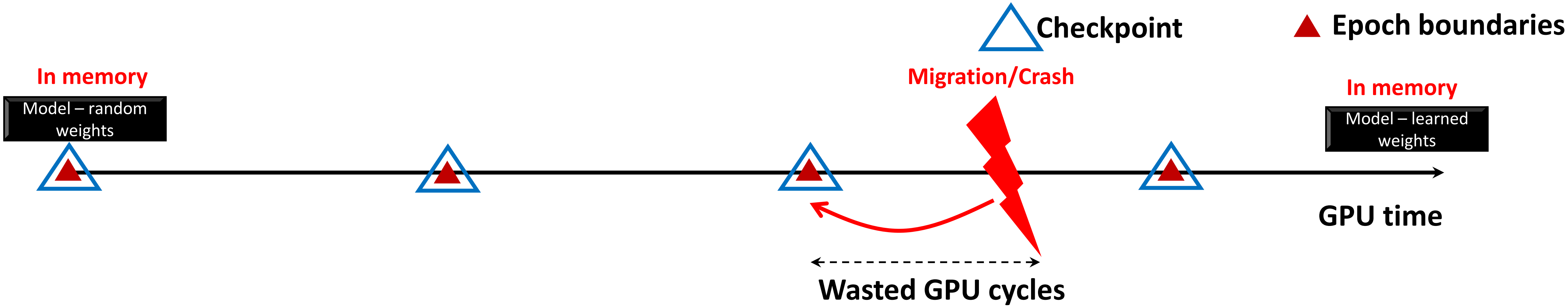
# DNN Checkpointing

Any interruption can wipe out the model parameters learned so far in memory, restarting this expensive process!



# DNN Checkpointing

- Learned model parameters are written to persistent storage every so often during training for fault-tolerance:
  - The VMs may migrate, expire, or crash (e.g., spot instances), jobs may migrate (e.g., shared GPU clusters)



# State of DNN Checkpointing Today

- Synchronous checkpoints => Large **checkpoint stalls**
- Manual checkpointing frequency => Typically performed at **epoch boundaries**
- But **epoch times are increasing** due to higher computational complexity of models and increasing dataset sizes
- **Frequent interruptions** : for e.g. preemptions in low-cost spot VMs

**Need fine-grained, iteration-level checkpointing**

# Challenges for fine-grained checkpointing

| Checkpointing frequency | Checkpoint stalls | Data invariant |
| --- | --- | --- |
| How often to checkpoint? | How to minimize the cost of a checkpoint? | How to resume correctly from a checkpoint? |

- Every epoch processes all the items in the dataset exactly once, in a random shuffled order
- Must hold when training resumes after an interruption in the middle of an epoch

# Challenges for fine-grained checkpointing

**Checkpointing frequency**

How often to checkpoint?

**Checkpoint stalls**

How to minimize the cost of a checkpoint?

**Data invariant**

How to resume correctly from a checkpoint?

Our work addresses these challenges to provide an automated, frequent checkpointing framework for DNN training

# CheckFreq

- Fine-grained, automated checkpointing framework for DNN training
- Strikes a balance between **low overhead** and **high frequency** of checkpointing => new checkpointing policy and mechanism
- Exploits the DNN computational model to perform **pipelined in-memory snapshots**, **GPU-based snapshots**, and **adaptive tuning** of checkpointing frequency
- CheckFreq reduces the recovery time for popular DNNs from **hours to seconds** during job interruptions

Source code : https://github.com/msr-fiddle/CheckFreq

# Outline

- Background and Motivation
- **CheckFreq – Design**
  - Checkpointing Mechanism
  - Checkpointing Policy
- Evaluation

# CheckFreq Design

**Mechanism**

How to perform correct, low-cost checkpointing?

2-phase DNN-aware checkpointing

Low checkpoint stalls

Resumable data iterator

Maintain data invariant

**Policy**

When to checkpoint?

Systematic online profiling

Initial checkpointing frequency

Adaptive rate tuning

Manages interference from other jobs

# CheckFreq Design

Mechanism

How to perform correct, low-cost checkpointing?

Policy

When to checkpoint?

**Recovery Guarantees**

| 2-phase DNN-aware checkpointing |
| --- |
| Low checkpoint stalls |

| Systematic online profiling |
| --- |
| Initial checkpointing frequency |

| Resumable data iterator |
| --- |
| Maintain data invariant |

| Adaptive rate tuning |
| --- |
| Manages interference from other jobs |

# Outline

- Background and Motivation
- CheckFreq – Design
  - **Checkpointing Mechanism**
  - Checkpointing Policy
- Evaluation

# 2-Phase Checkpointing

- Synchronous checkpointing introduces **checkpoint stalls** => **Runtime overhead**
- Low-cost checkpointing mechanism that is split into a **pipelined** snapshot() and persist() phase

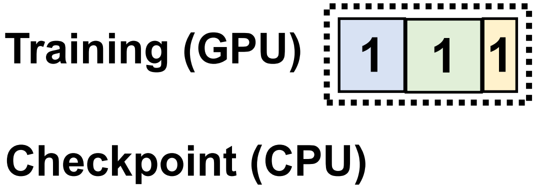
Snapshot() : Serialize and copy into a user-space buffer

Persist() : Write out the serialized contents to disk

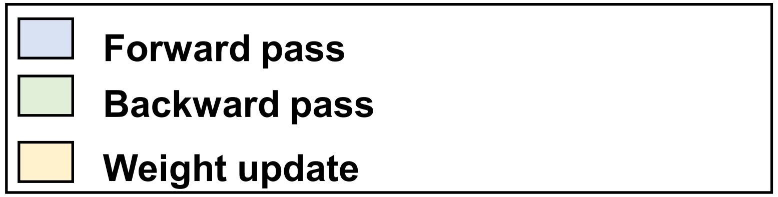
# Example

- Consider a policy that checkpoints every three iterations.

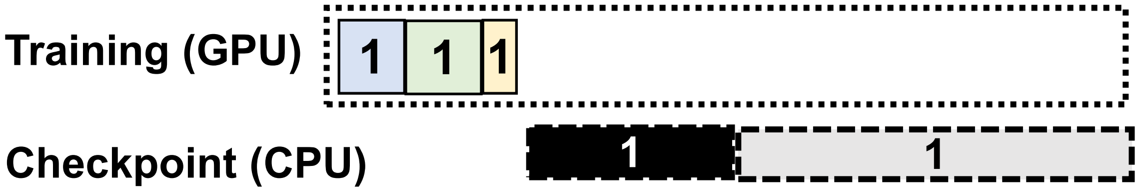
# Example



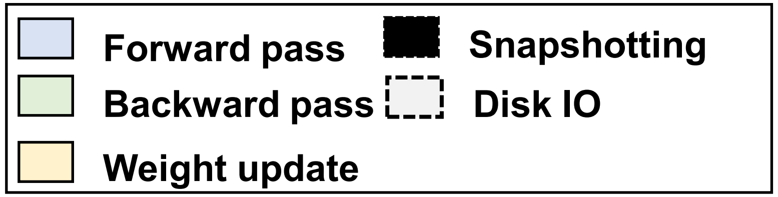

(a) Baseline : Synchronous checkpointing



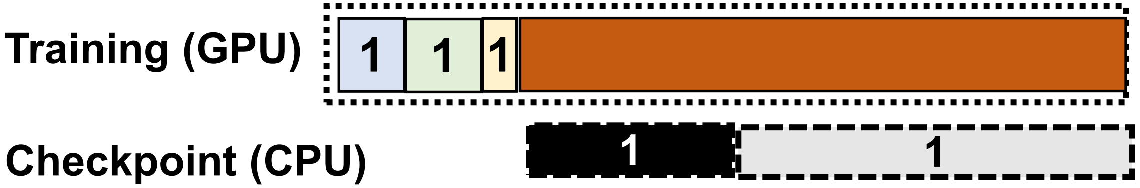
# Example



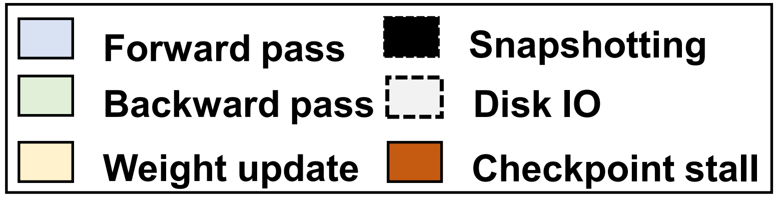

(a) Baseline : Synchronous checkpointing



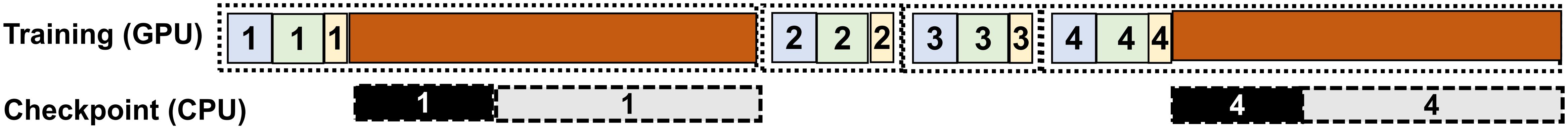
# Example



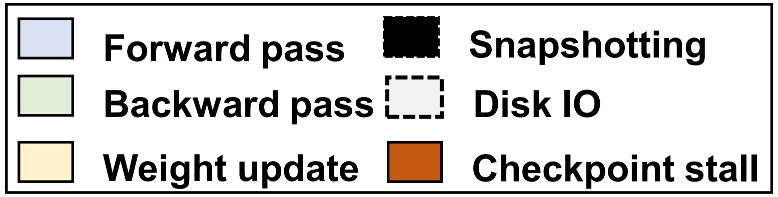

(a) Baseline : Synchronous checkpointing



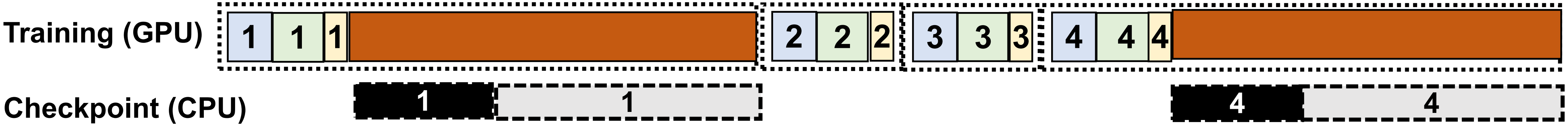
# Example



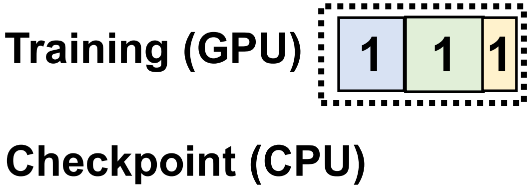

(a) Baseline : Synchronous checkpointing



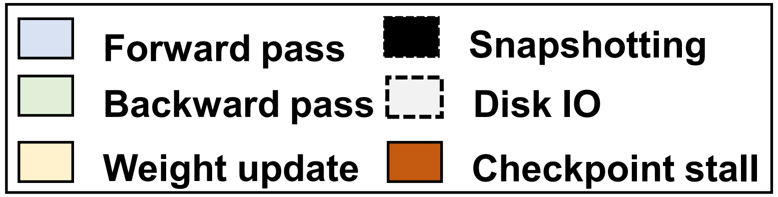
# Example



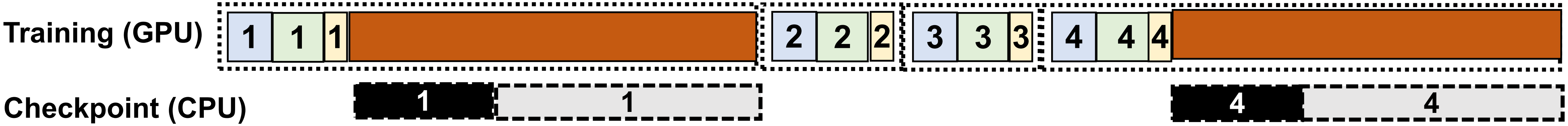

(a) Baseline : Synchronous checkpointing



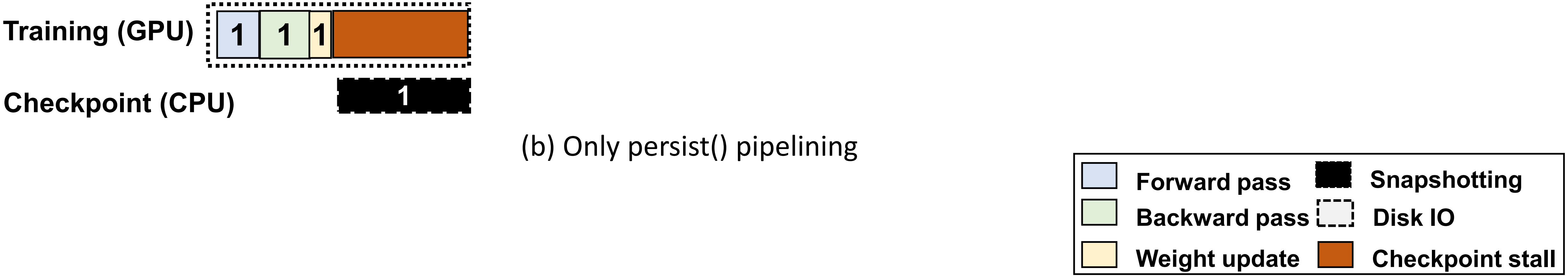

(b) Only persist() pipelining



# Example

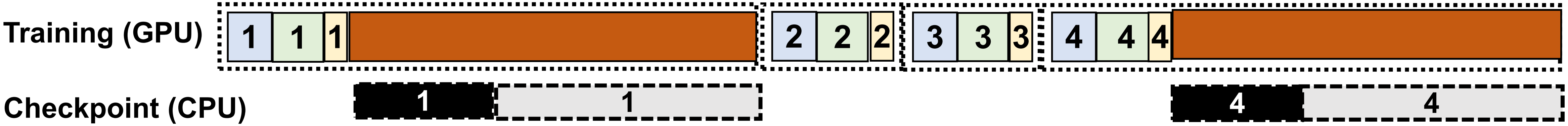



(a) Baseline : Synchronous checkpointing

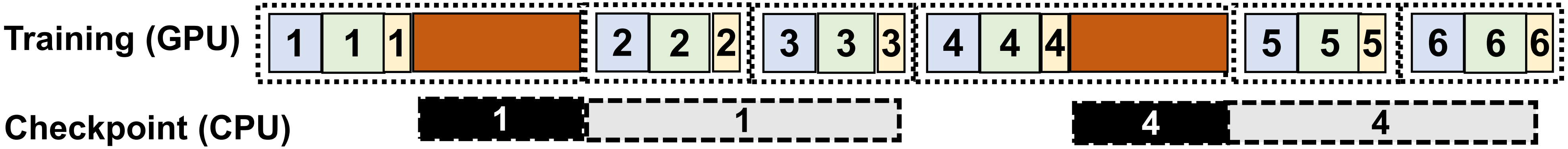



(b) Only persist() pipelining

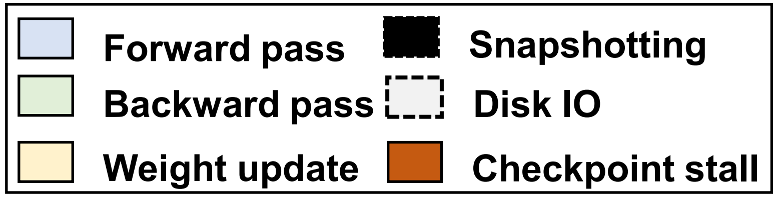
# Example



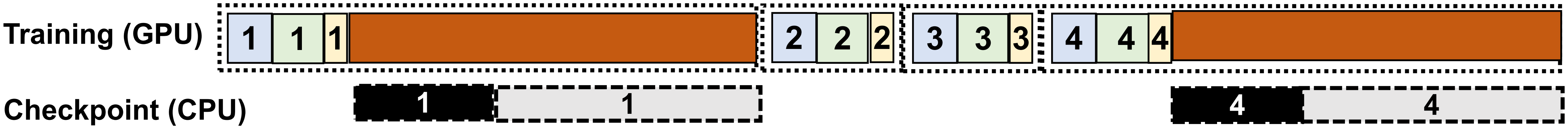

(a) Baseline : Synchronous checkpointing



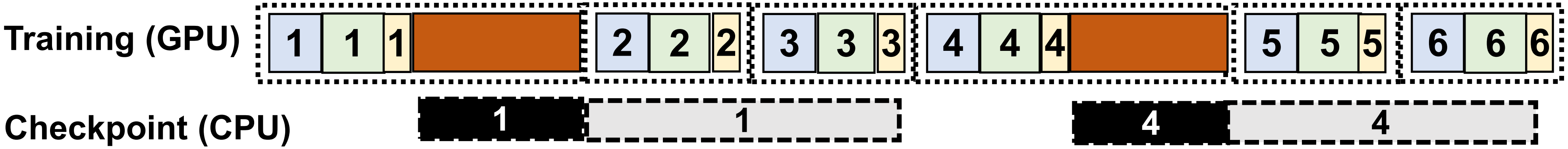

(b) Only persist() pipelining



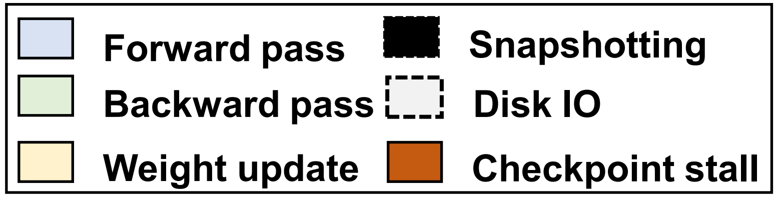
# Example

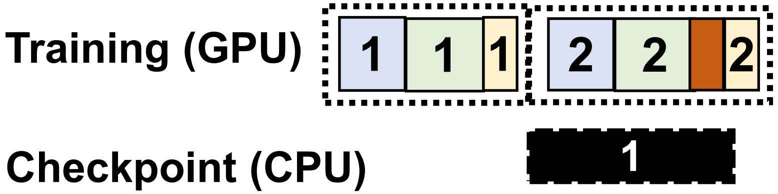



(a) Baseline : Synchronous checkpointing



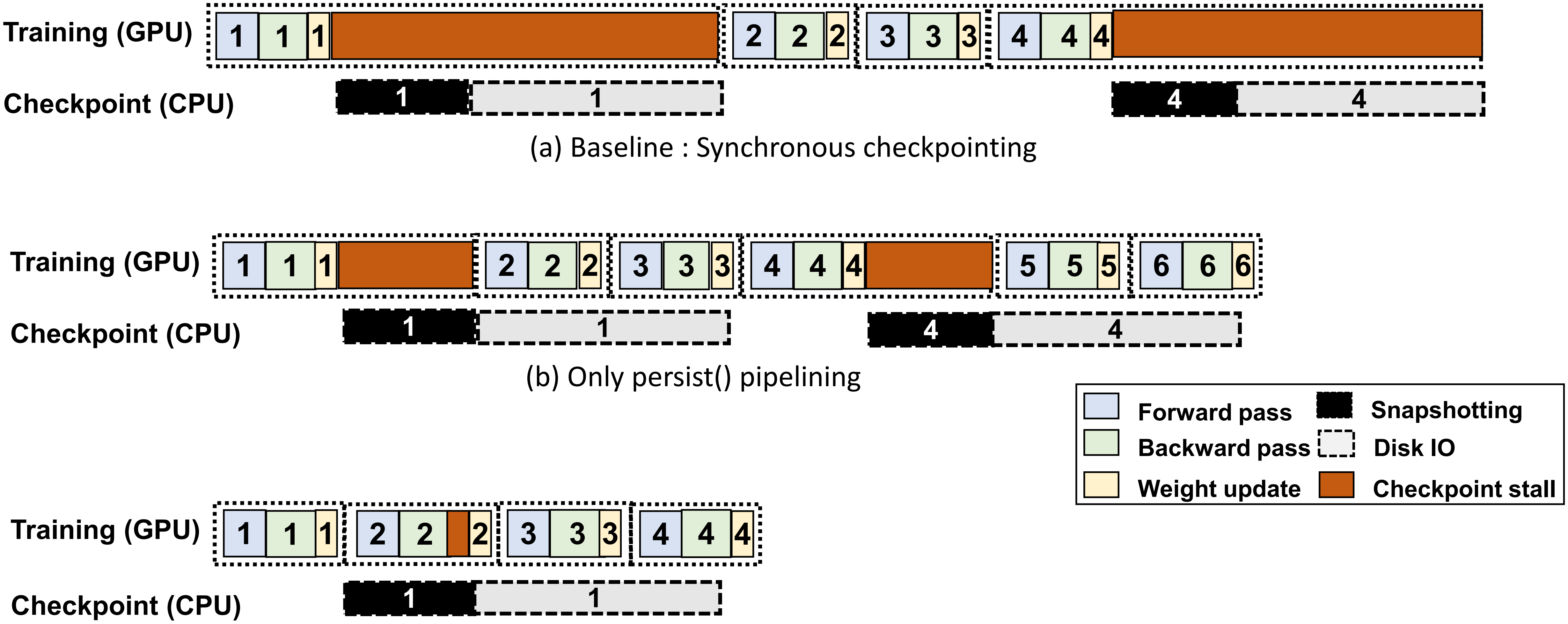

(b) Only persist() pipelining




(c) Snapshot() and persist() pipelining

# Example

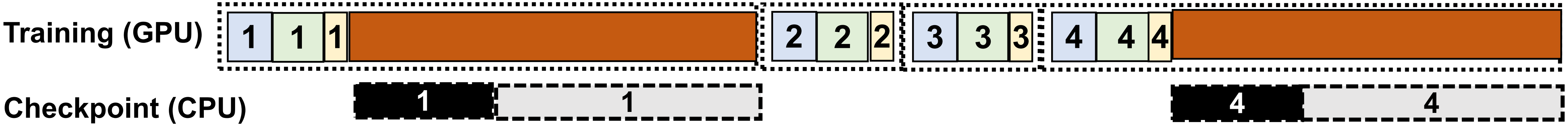



(a) Baseline : Synchronous checkpointing

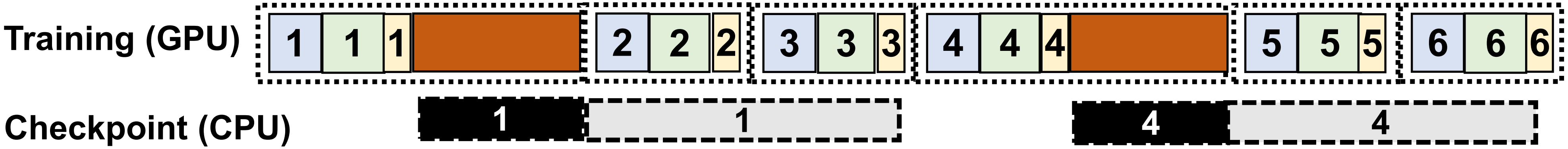
(b) Only persist() pipelining

(c) Snapshot() and persist() pipelining

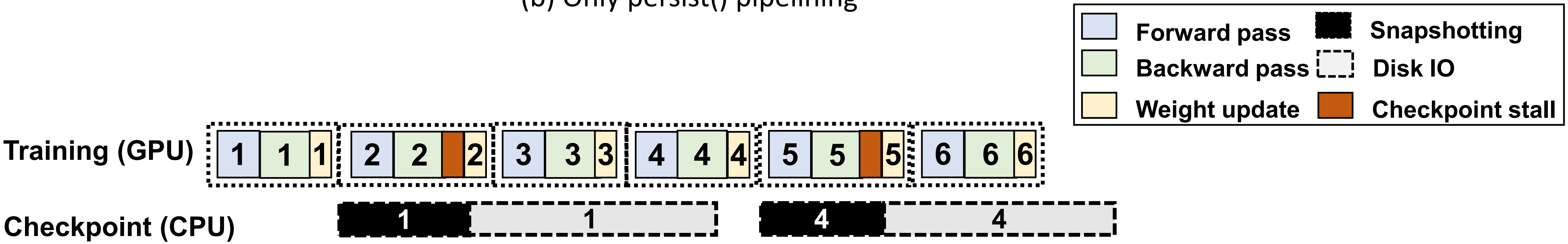
# Example




(a) Baseline : Synchronous checkpointing




(b) Only persist() pipelining




(c) Snapshot() and persist() pipelining

# GPU-optimized Snapshots

- Cost of serialization and snapshot() is upto 10x lower when done on the GPU
- To further reduce the checkpoint cost, CheckFreq snapshots on the GPU, and asynchronously writes it to CPU memory if it profiles spare memory on the GPU
- If GPU memory is fully utilized, it falls back to pipelined, CPU-side snapshots

# Outline

- Background and Motivation
- CheckFreq – Design
  - Checkpointing Mechanism
  - **Checkpointing Policy**
- Evaluation

# Checkpointing policy

- Determines when to initiate a checkpoint
- Checkpoints every k iterations, such that
  - the cost of one checkpoint can be amortized over k iterations
  - Runtime overhead introduced due to checkpointing is within a small user-given percentage of the actual compute time (say 5%)

# Systematic Online Profiling

- CheckFreq’s data iterator automatically profiles several iteration-level and checkpoint-specific metrics

- Iteration time
- Time for weight update
- Time for GPU snapshot()
- Time for CPU snapshot()
- Available disk throughput
- Checkpoint size
- Peak GPU memory util
- Total GPU memory

Algorithmically determines the checkpointing frequency such that:

- Overhead due to checkpoint stalls is within the user-given limit

# Outline

- Background and Motivation
- CheckFreq – Design
  - Checkpointing Mechanism
  - Checkpointing Policy
- **Evaluation**

# Experimental Setup

- Checkfreq is integrated with PyTorch
  - Uses the state-of-the-art NVIDIA DALI data loading library to support resumability

- Experiments are performed on two different servers from an internal GPU cluster at Microsoft
  1. Conf-Volta :  Server with eight V100 GPUs (32GiB), with a SSD
  2. Conf-Pascal : Server with eight 1080Ti GPUs (11GiB), with a HDD

# Models and Experiments

- We evaluate CheckFreq on 7 different DNNs :
  - ResNet18, ResNet50, ResNext101, DenseNet121, VGG16, InceptionV3 on Imagenet-1k
  - Bert-Large pretraining on Wikipedia & BookCorpus dataset

- Experiments to evaluate:

Accuracy implications of data invariant

Checkpoint stalls

Recovery Time

Breakdown of benefits due to pipelining

Adaptive frequency tuning

End-to-end training with interruptions

# Models and Experiments

- We evaluate CheckFreq on 7 different DNNs :
  - ResNet18, ResNet50, ResNext101, DenseNet121, VGG16, InceptionV3 on Imagenet-1k
  - Bert-Large pretraining on Wikipedia & BookCorpus dataset

- Experiments to evaluate:

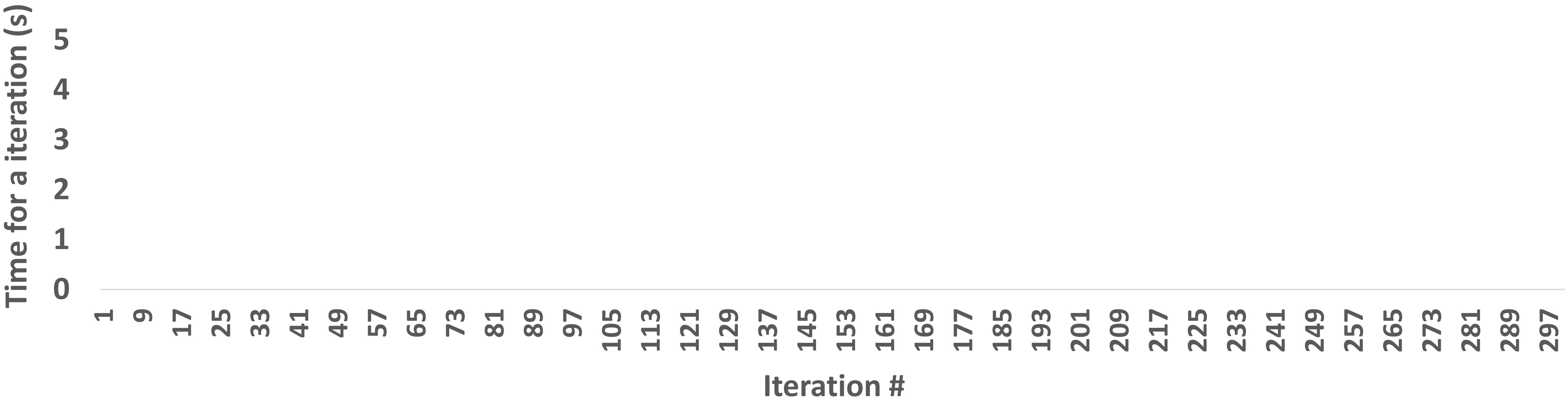
| | | |
|---|---|---|
| Accuracy implications of data invariant | **Checkpoint stalls** | **Recovery Time** |
| **Breakdown of benefits due to pipelining** | Adaptive frequency tuning | End-to-end training with interruptions |

# CheckFreq reduces checkpoint stalls

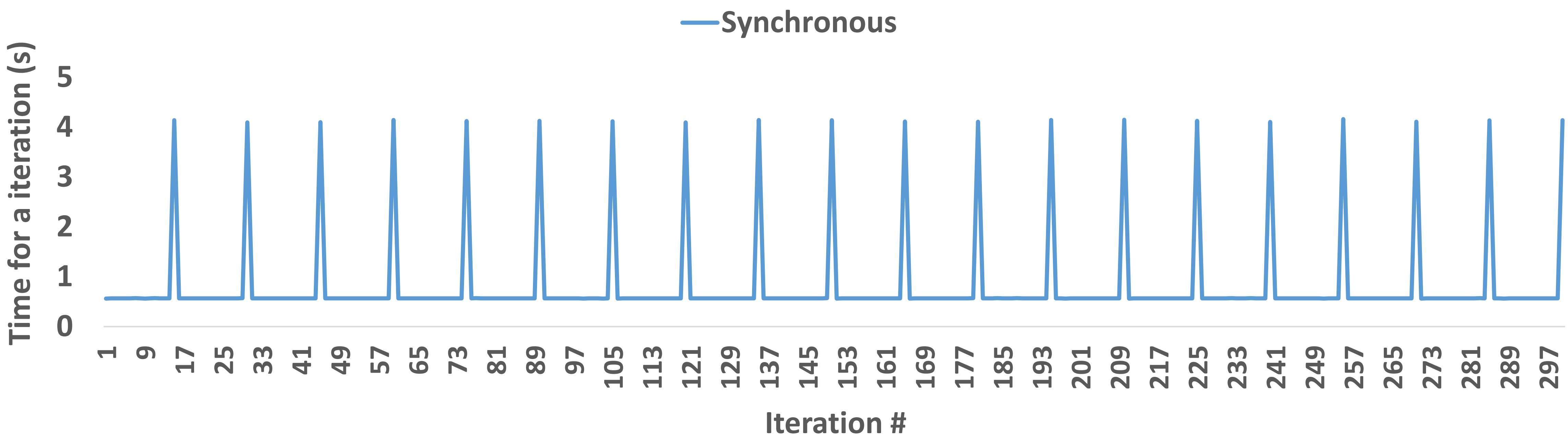
- Train VGG16 for 300 iterations on Conf-Volta
- Checkpointing mechanisms :
    - Synchronous
    - Persist() pipelining only
    - CheckFreq - Persist() and snapshot() pipelining

- Checkpointing frequency : 15 iterations

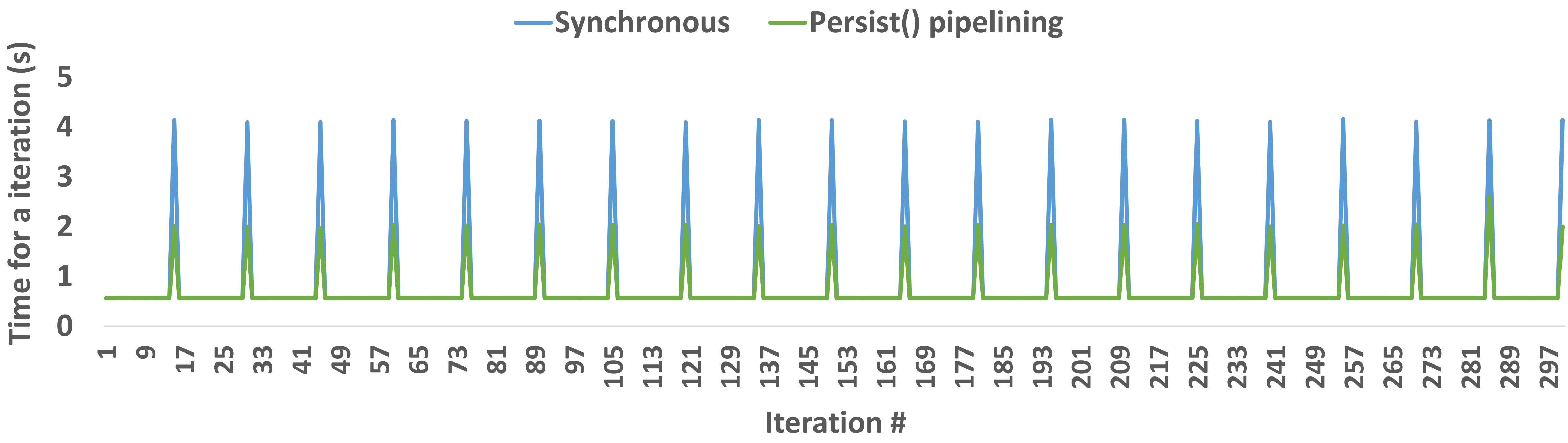
# CheckFreq reduces checkpoint stalls



# CheckFreq reduces checkpoint stalls

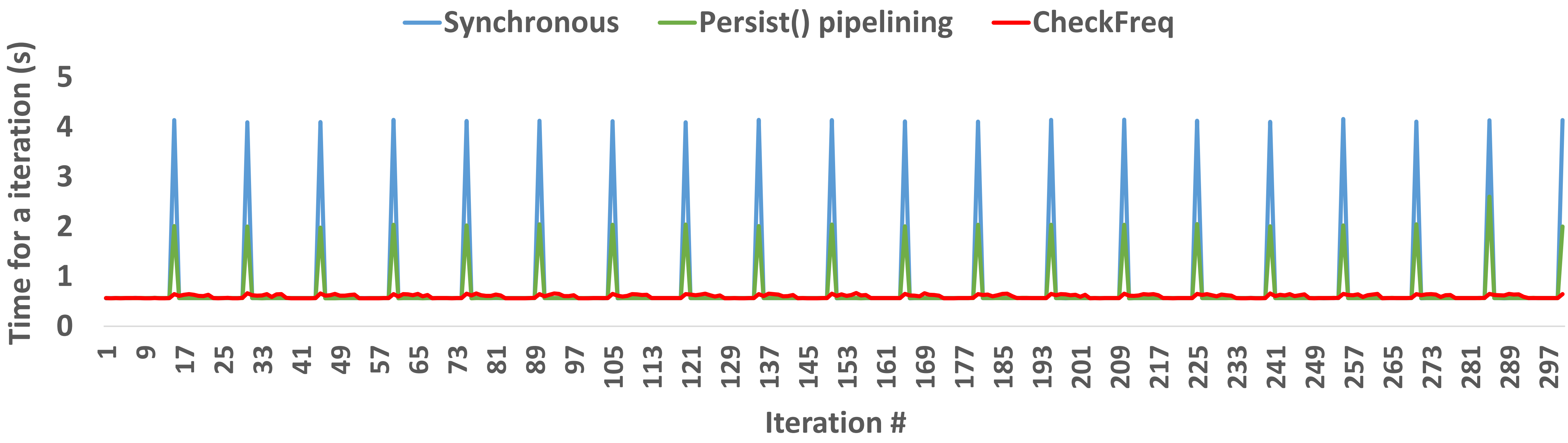


# CheckFreq reduces checkpoint stalls



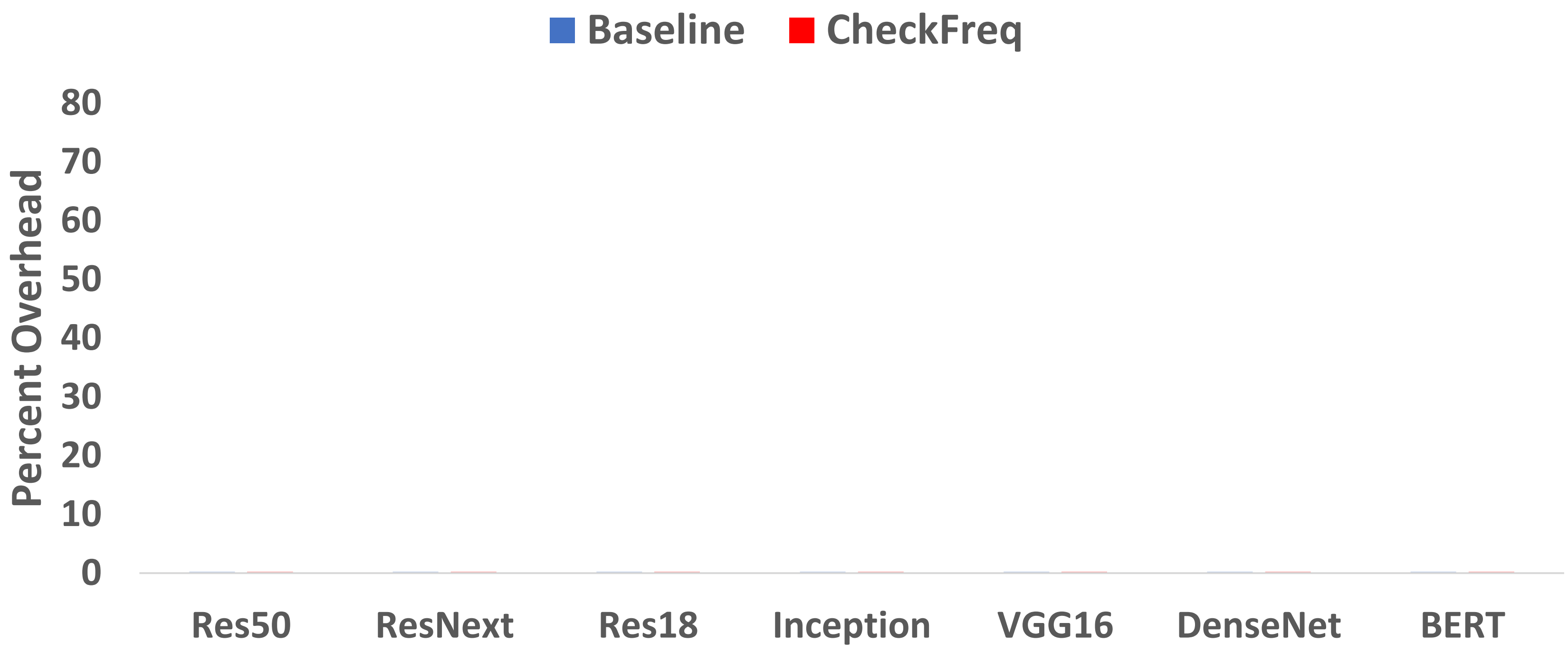

- Performing asynchronous IO reduces checkpoint cost by 2x but still results in significant stalls

# CheckFreq reduces checkpoint stalls

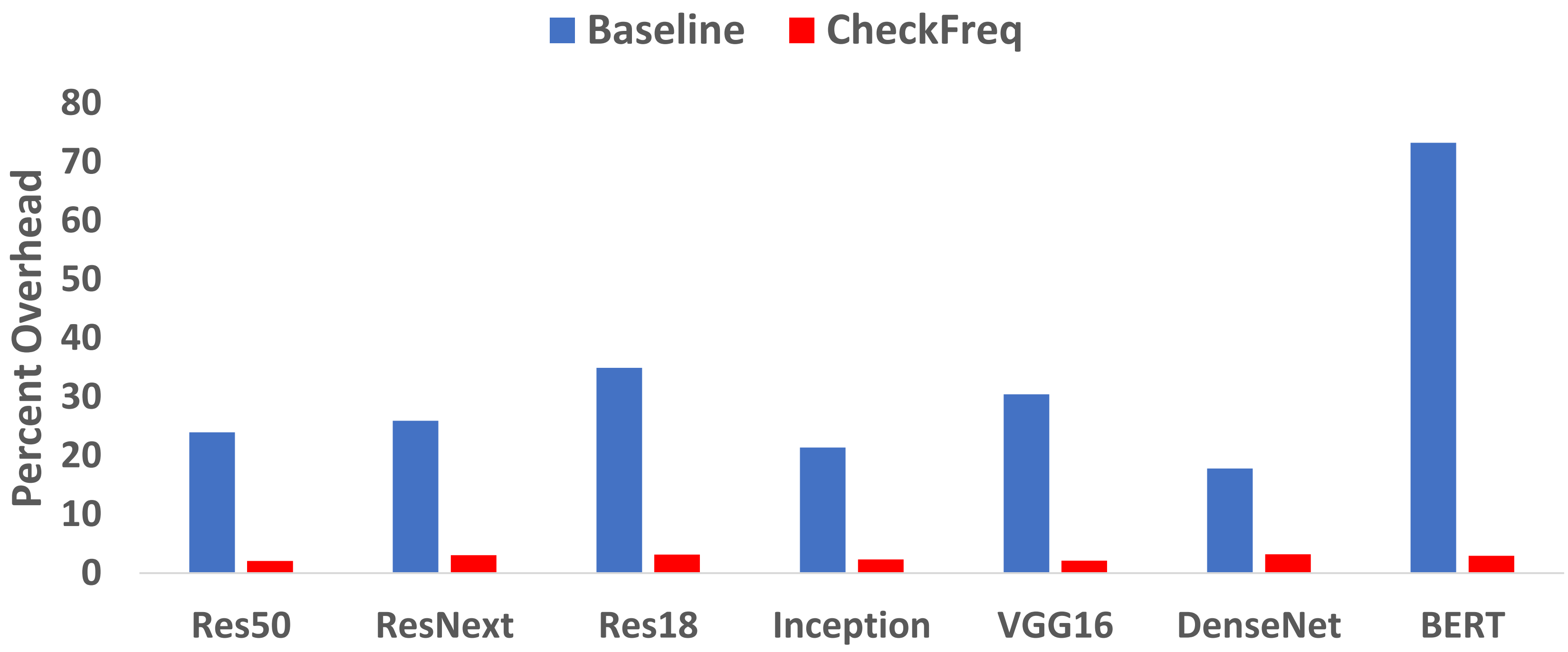



- CheckFreq further reduces stalls by carefully pipelining checkpointing with compute

# Overall Training Overhead



# Overall Training Overhead




- When the baseline checkpointing mechanism is performed at a frequency chosen by CheckFreq, it introduces 20 – 70% overhead in training time

# CheckFreq lowers recovery time

| Model | Epoch-based (s) | CheckFreq (s) |
| --- | --- | --- |
| Res18 | | |
| Res50 | | |
| VGG16 | | |
| ResNext | | |
| DenseNet | | |
| Inception | | |
| BERT | | |

- Recovery time : Time spent by the model to recover to the same state as it was before interruption

# CheckFreq lowers recovery time

| Model | Epoch-based (s) | CheckFreq (s) |
|---|---|---|
| Res18 | 840 | 5 |
| Res50 | 2100 | 24 |
| VGG16 | 5700 | 25 |
| ResNext | 7080 | 32 |
| DenseNet | 2340 | 7 |
| Inception | 3000 | 27 |
| BERT | 4920 | 85 |

- Recovery time : Time spent by the model to recover to the same state as it was before interruption
- CheckFreq reduces recovery time during an interruption from hours to seconds

# Conclusion

- CheckFreq provides an automatic, fine-grained checkpointing framework for DNN training
- CheckFreq allows frequent checkpointing while incurring a low cost
- When the job is interrupted, CheckFreq reduces recovery time for popular DNNs from hours to seconds

# Thank you!

**Source code :** https://github.com/msr-fiddle/CheckFreq

**Contact** : jaya@cs.utexas.edu