RICE

Scaling Deep Learning through Optimizing Data- and Management-Plane Communications

Zhuang Wang
Department of Computer Science

Advisor: Prof. T. S. Eugene Ng

Big Data and Optical Lightpaths Driven Lab

Roadmap

- Background of distributed deep learning (DDL)
- Four research projects that optimize communications in DDL
  - Zen, OSDI ’25
  - Espresso, EuroSys ‘23
  - Cupcake, MLSys ‘23
  - Gemini, SOSP ‘23
- Implementation and evaluation
- Acknowledgement

Deep learning is everywhere

Z How to do a good PhD thesis defense?

Defending your PhD thesis can be nerve-wracking, but with thorough preparation and a positive mindset, you can make it a successful and satisfying experience. Here are some tips on how to do a good PhD thesis defense:

1. **Know Your Thesis Inside Out**: Be intimately familiar with every aspect of your thesis. Understand your research questions, methodology, findings, and conclusions. Anticipate potential questions and critiques from your committee, and be ready to provide detailed responses.
2. **Practice Your Presentation**: Practice your defense presentation multiple times in front of a mirror, friends, or family. Pay attention to your delivery, timing, and clarity. Your presentation

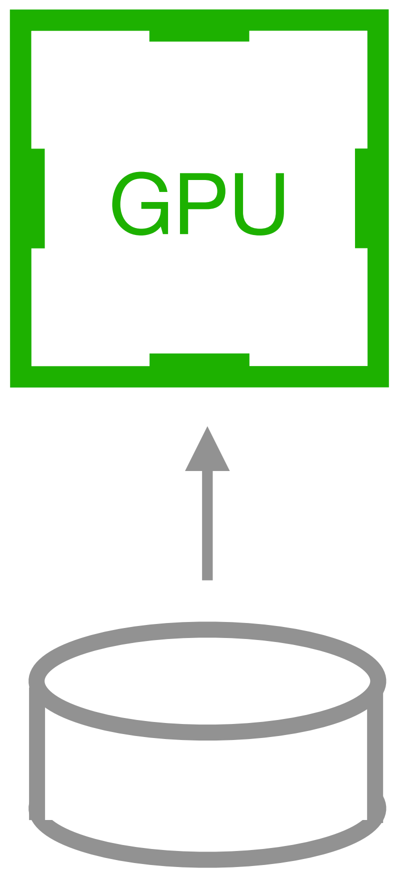
Language processing

Recommendation

Voice assistant

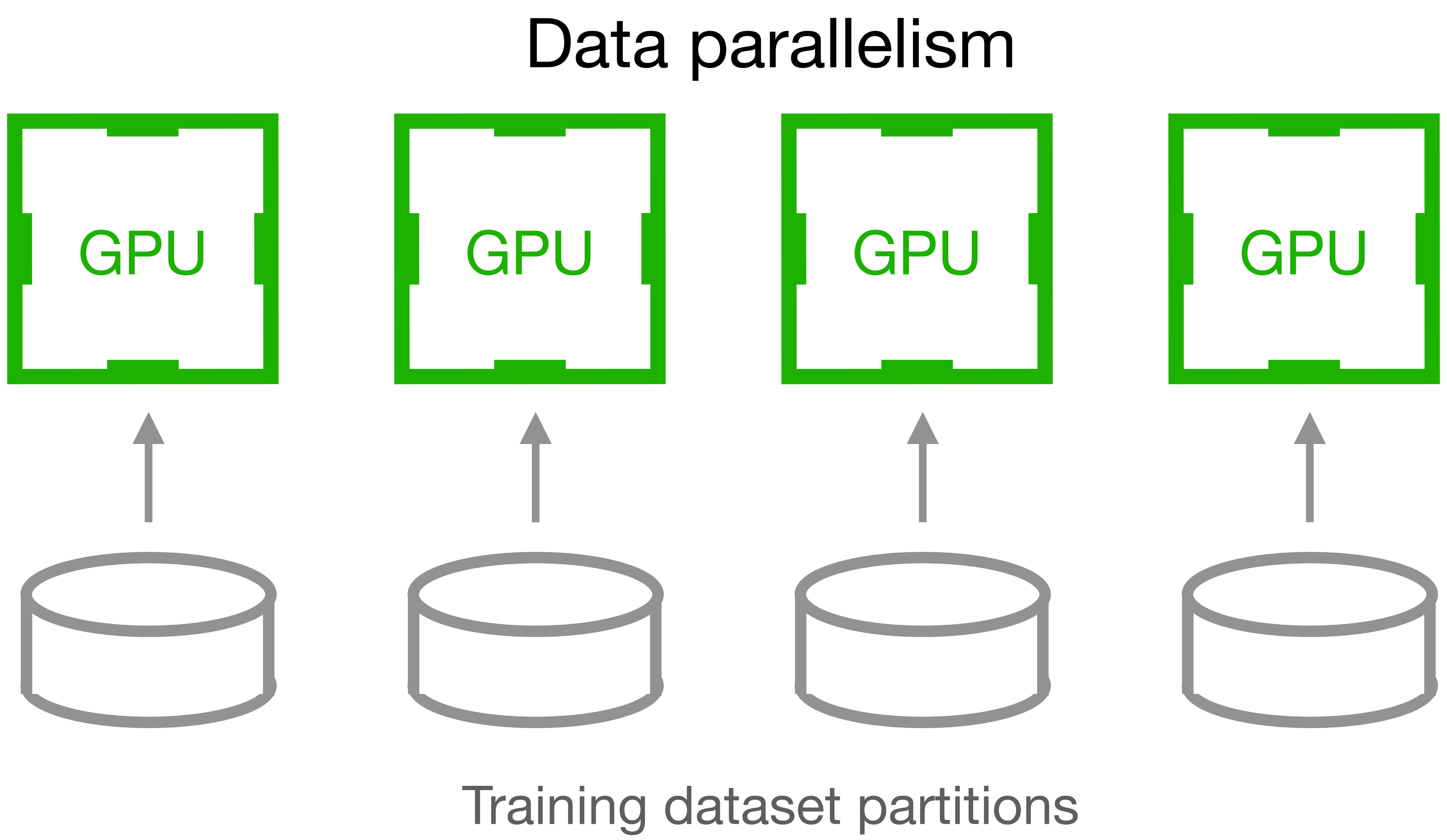
Trends in deep learning

- Data grows exponentially
  - Easy scale to terabytes



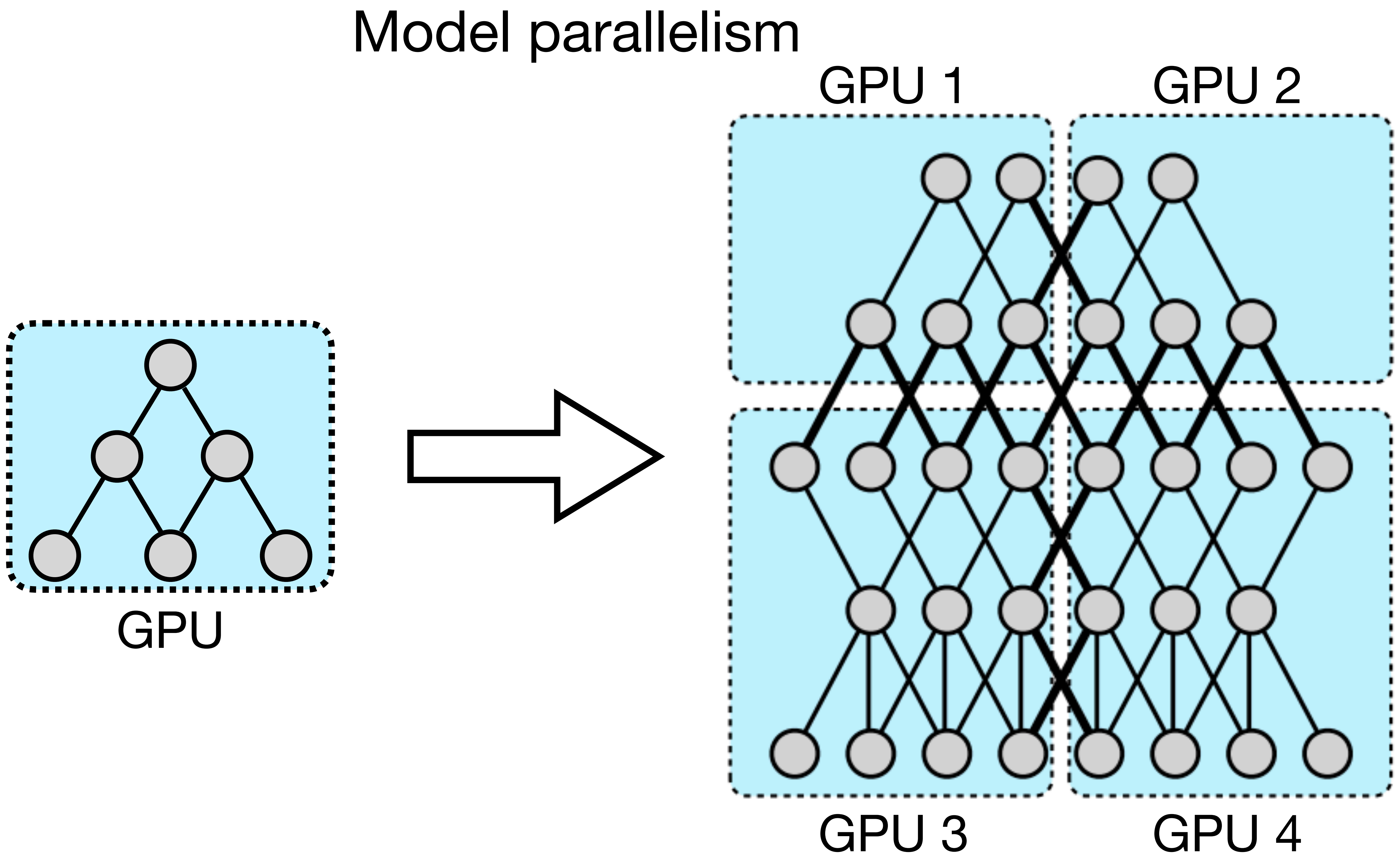
Trends in deep learning

- Data grows exponentially
  - Easy scale to terabytes



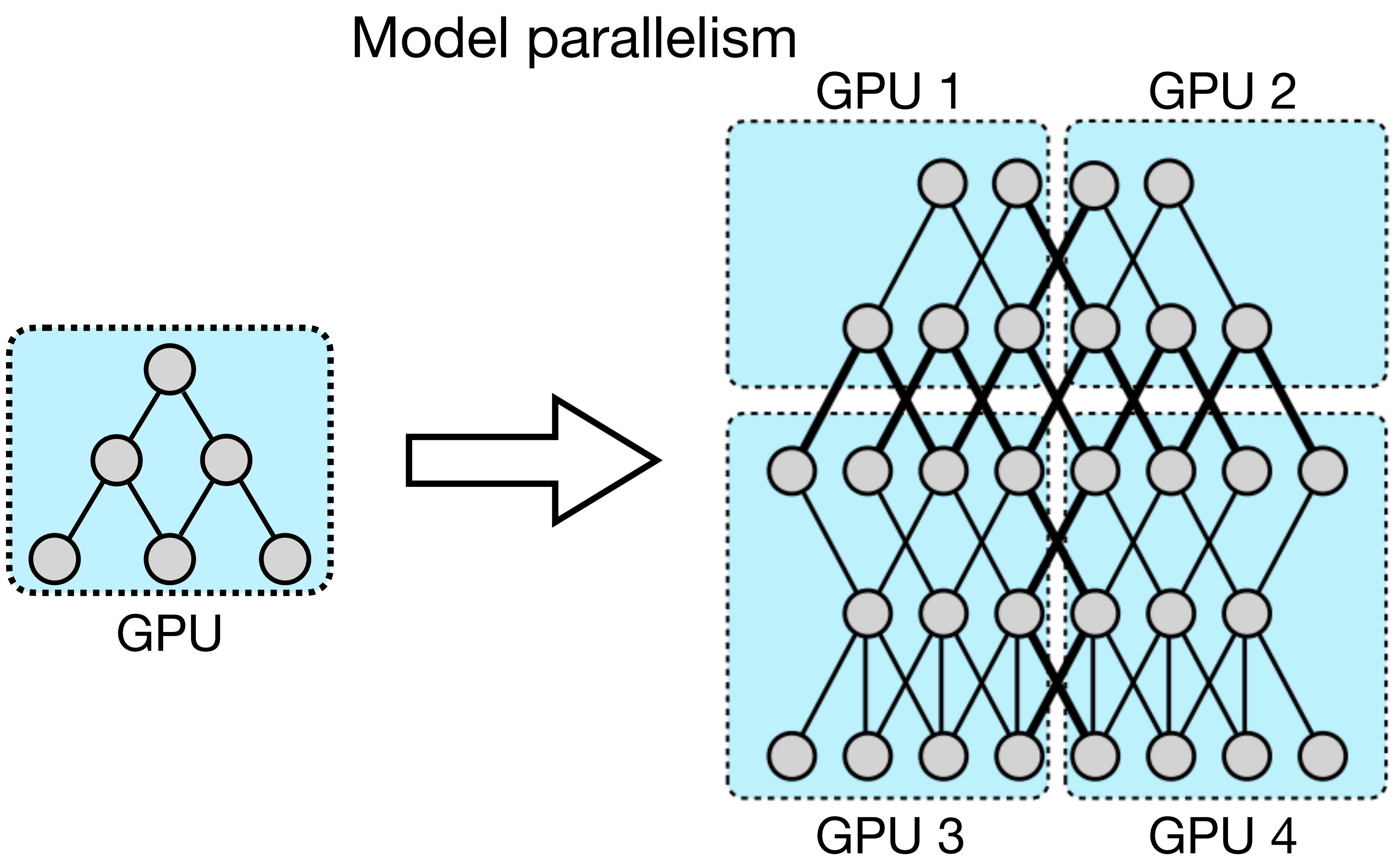
Trends in deep learning

- Data grows exponentially
  - Easy scale to terabytes
- Model size grows exponentially
  - Towards models with trillion parameters



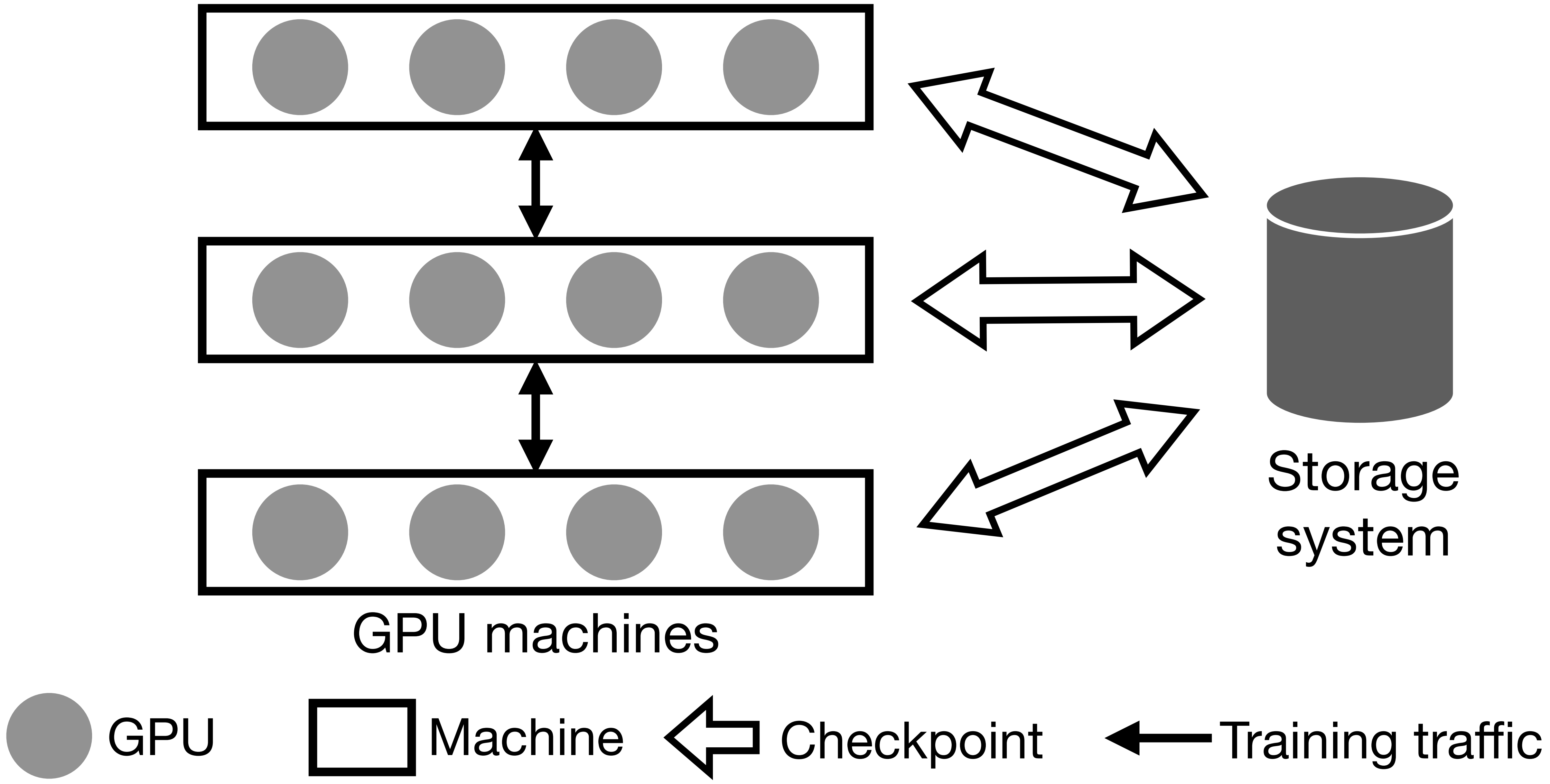
Trends in deep learning

- Data grows exponentially
  - Easy scale to terabytes
- Model size grows exponentially
  - Towards models with trillion parameters
- Distributed deep learning
  - Data parallelism
  - Model parallelism



Distributed deep learning (DDL)
Training system

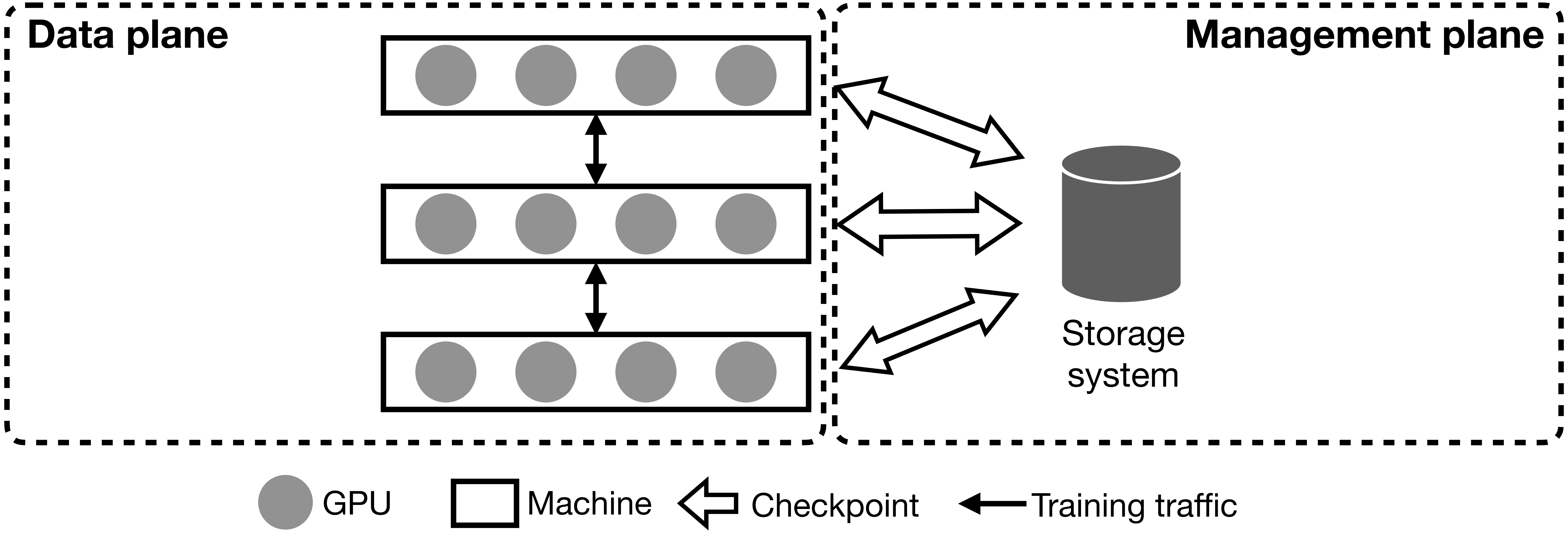
- Two components
  - GPU machines for model training
  - Storage system that stores checkpoints for fault tolerance



Distributed deep learning (DDL)

Training system

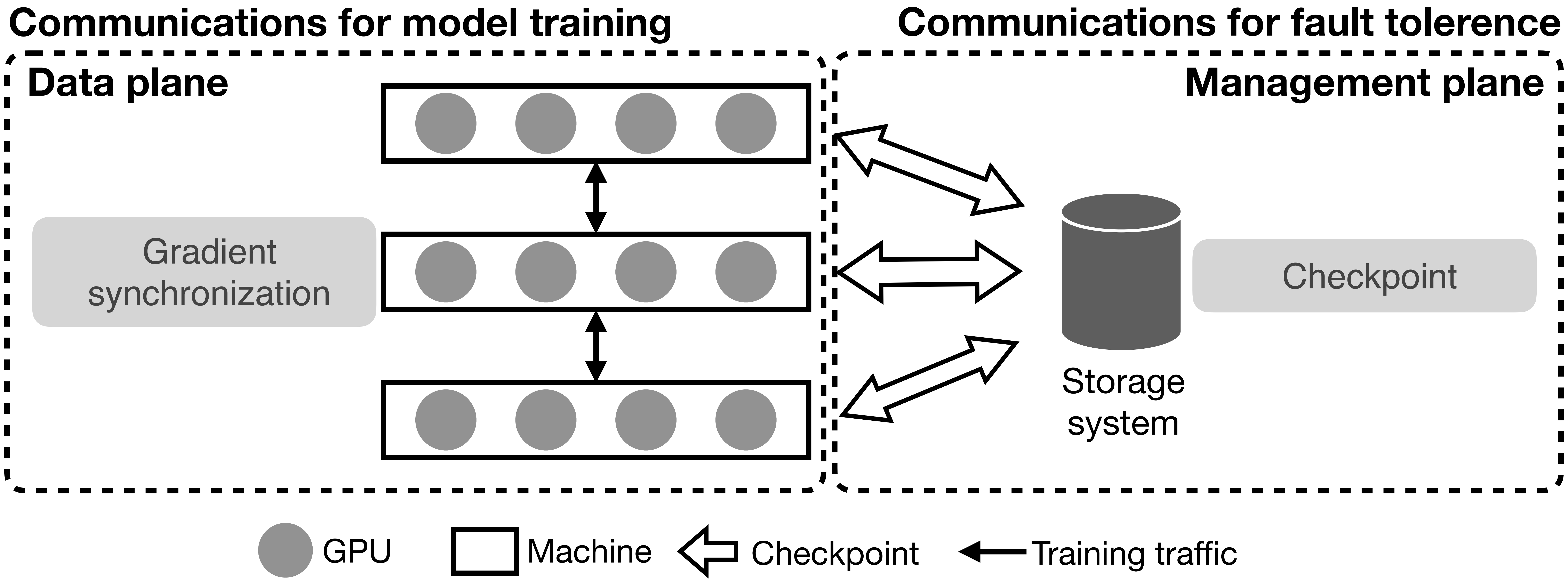
- Two components



Communications in DDL

In both planes

- Communications are bottlenecks for scalability



Goal of this thesis

- Communications are bottlenecks for scalability

Communications for model training

Communications for fault tolerence

Recognize and tackle the communication obstacles within DDL to enhance its scalability

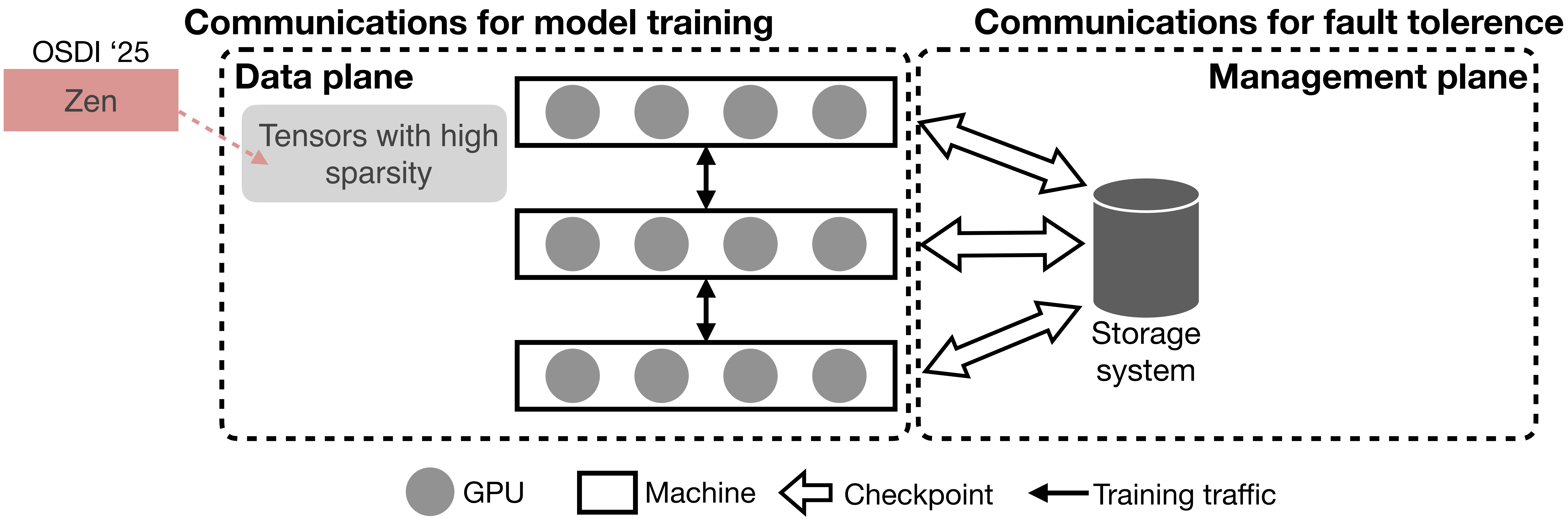
Thesis statement

This thesis demonstrates the feasibility of mitigating communication bottlenecks in distributed deep learning by utilizing **current hardware resources** within a training system, complemented by **intelligent traffic and resource scheduling algorithms**.

Research summary

Scaling deep learning by optimizing communications

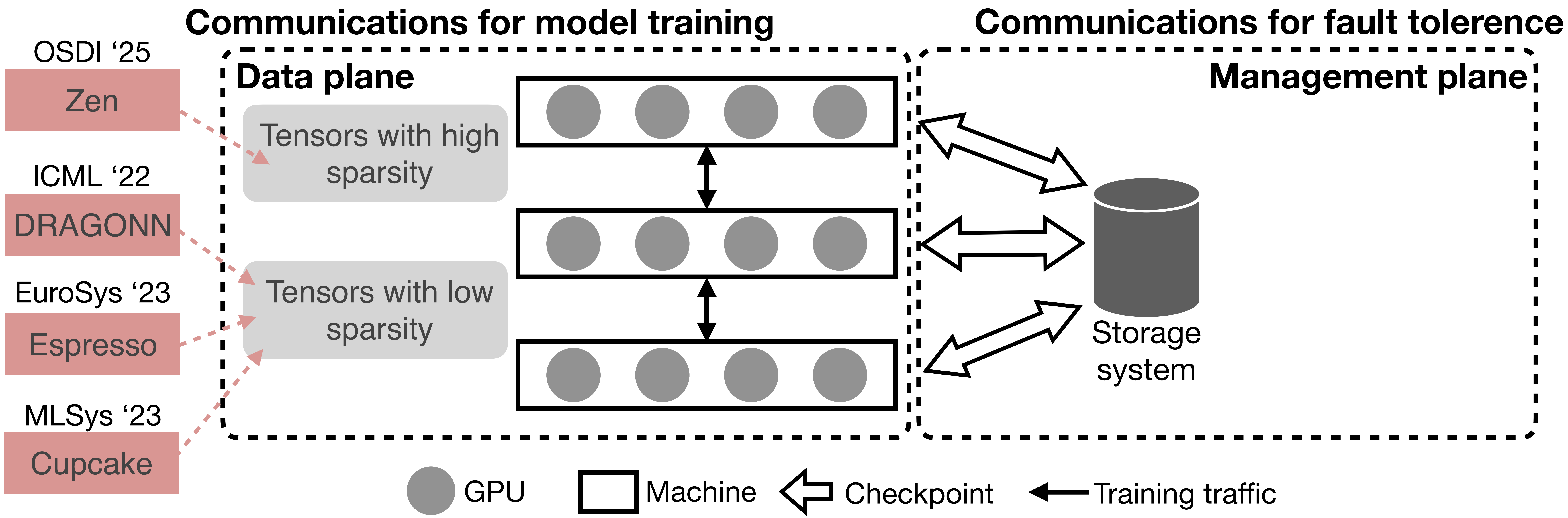
- Mitigate data-plane communication bottlenecks



Research summary

Scaling deep learning by optimizing communications

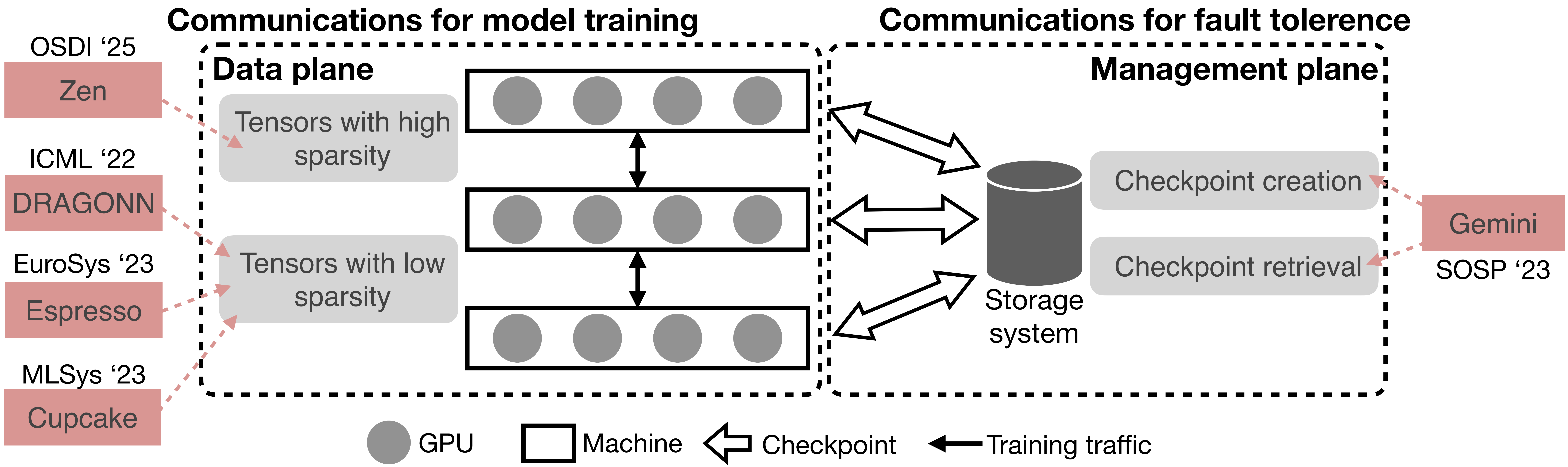
- Mitigate data-plane communication bottlenecks



Research summary

Scaling deep learning by optimizing communications

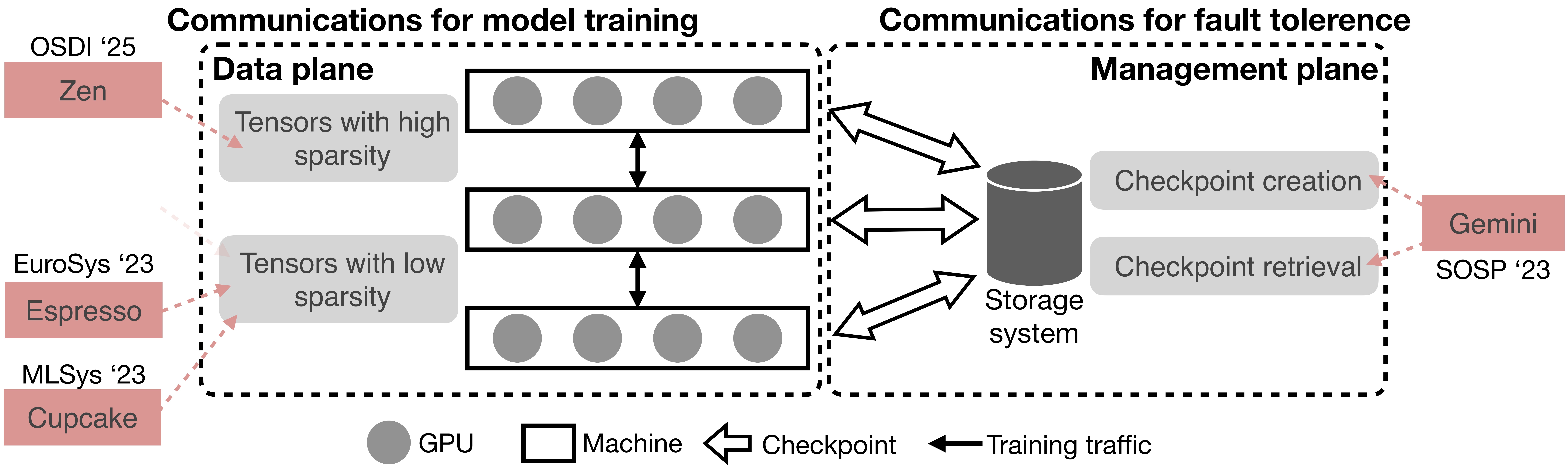
- Mitigate data- and management-plane communication bottlenecks



Research summary

Scaling deep learning by optimizing communications

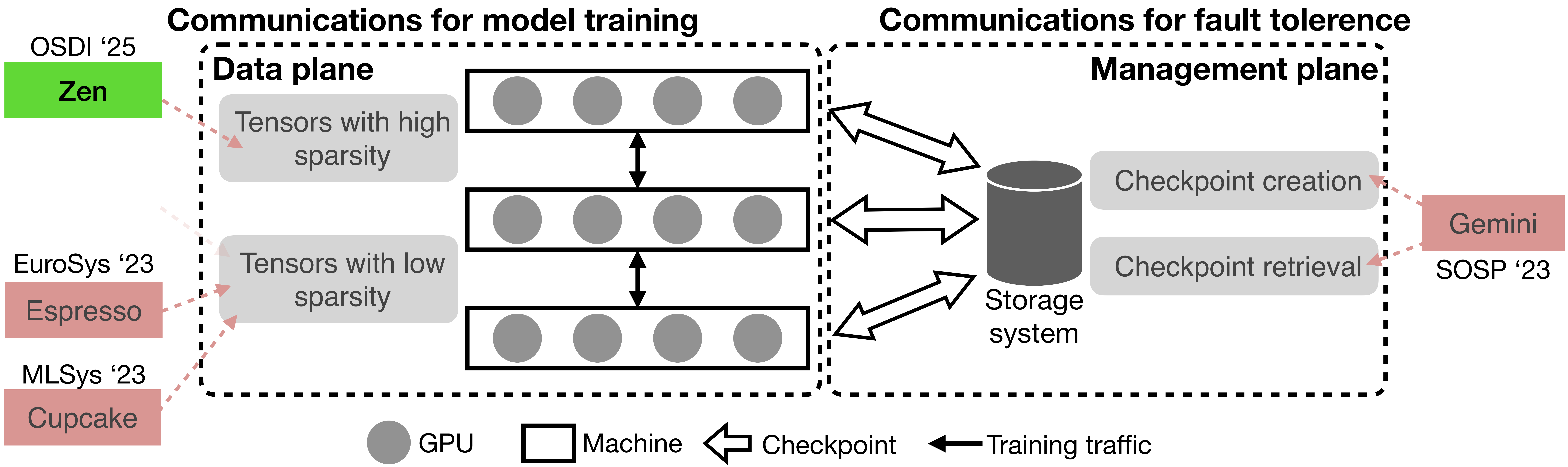
- Thesis work



The next research project

Scaling deep learning by optimizing communications

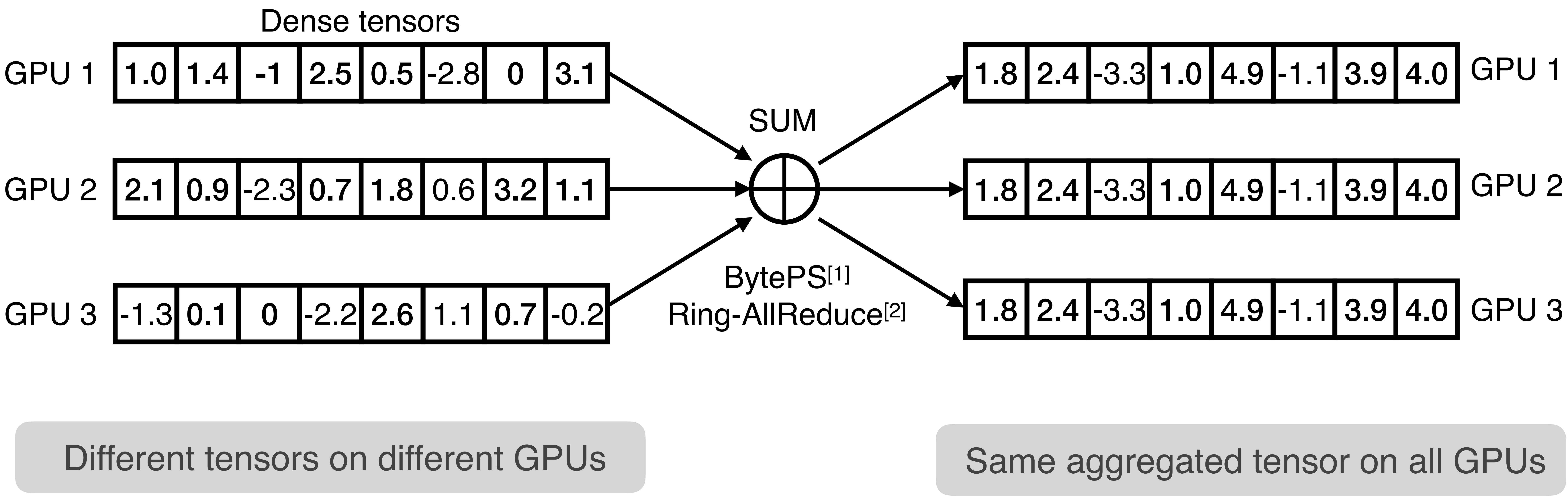
- Thesis work



Gradient synchronization

Dense tensor synchronization among GPUs

- Communication and aggregation



[1] A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters, OSDI 2020
[2] Horovod: fast and easy distributed deep learning in TensorFlow, arXiv 2018

Sparsity in gradient tensors

Non-zero gradients in tensors

- Gradients can be zero

1.0	1.4	-1	2.5	0.5	-2.8	0	3.1

Low sparsity

0	1.4	0	2.5	0	0	0	0

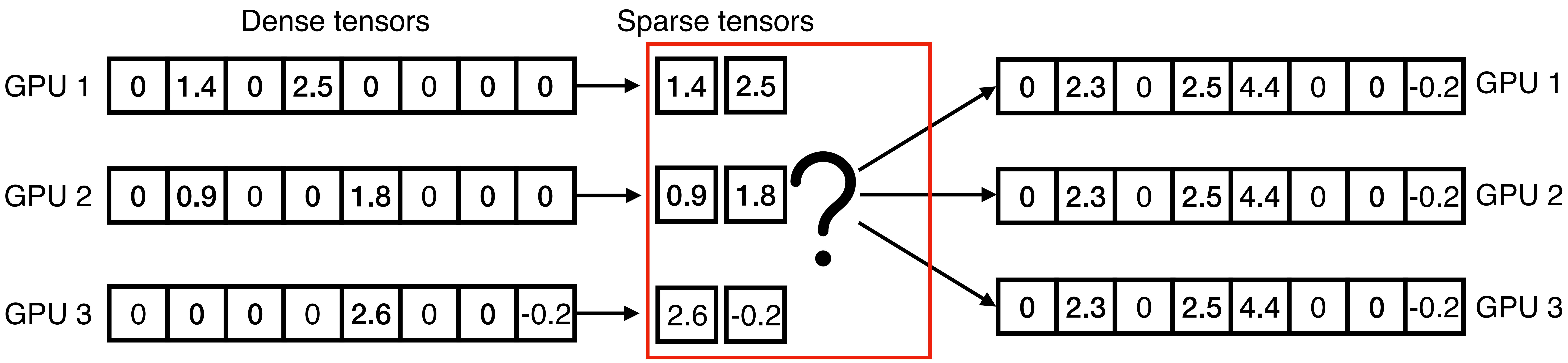
High sparsity

- Statistics from popular DNN models

Model	Task	Dataset	Batch Size	Sparisty
LSTM	Language Modeling	One Billion Word	128	98.87%
DeepFM	Click-through Rate Prediction	Criteo	1024	97.20%
NMT	Machine Translation	IWSLT 2014 De-En	64	97.53%
BERT	Question Answering	SQuAD v1.1	4	98.94%

Synchronization of sparse tensors
Opportunities

- Communicate sparse tensors, i.e., non-zero gradients
  - Greatly reduces the amount of traffic volume
  - Potentially shortens communication time



What is the optimal scheme to synchronize sparse tensors?

Contributions

Zen

- We comprehensively **analyze and understand** the fundamentals of sparsity
- We systematically **explore the design space** of schemes for the first time
  - Four dimensions to describe any scheme
- We find the **provably optimal scheme** from the design space
- We propose **a novel hierarchical hashing algorithm** that uses parallel computing on GPUs to realize the optimal scheme
- We propose **a new data format** to represent sparse tensors to minimize the overhead required for indices

Contributions

Zen

- We comprehensively analyze and understand the fundamentals of sparsity
- We systematically **explore the design space** of schemes for the first time
    - Four dimensions to describe any scheme
- We find the provably optimal scheme from the design space
- We propose **a novel hierarchical hashing algorithm** that uses parallel computing on GPUs to realize the optimal scheme
- We propose a new data format to represent sparse tensors to minimize the overhead required for indices

How to describe the design space?

Four dimensions

- Sparse tensor before synchronization

1.2	0.7	0.3	2.5
GPU 0	GPU 1	GPU 2	GPU 3

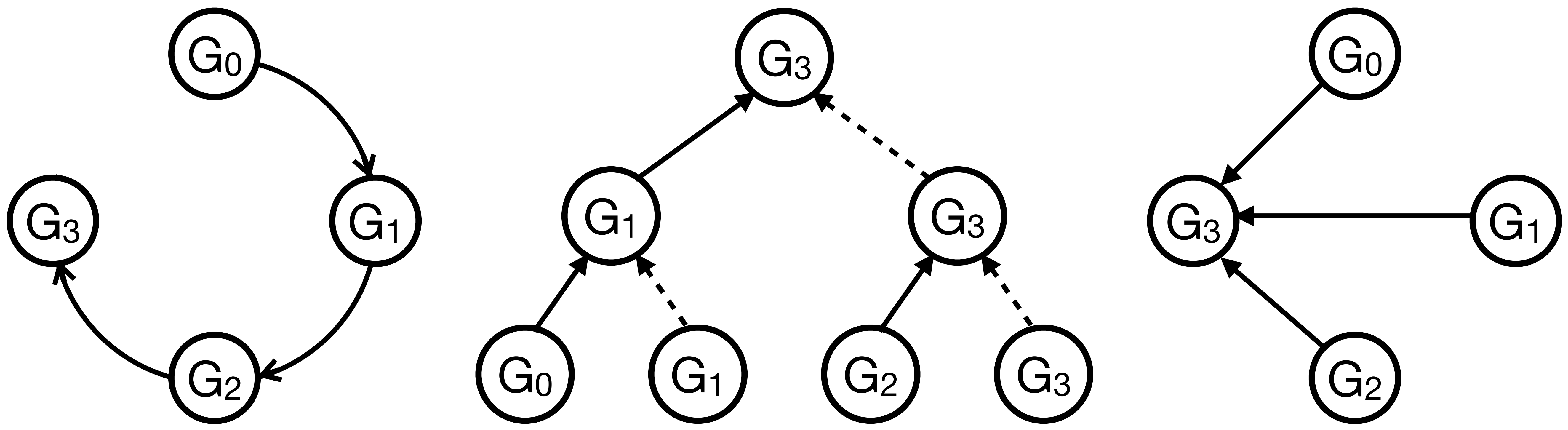
How to describe the design space?

Four dimensions

- Sparse tensor before synchronization

1.2	0.7	0.3	2.5
GPU 0	GPU 1	GPU 2	GPU 3

- Communication dimension



(a) Ring (b) Hierarchy (c) Point-to-point

Communication
- Ring
- Hierarchy
- Point-to-point

Aggregation
- Incremental
- One-shot

Partition
- Centralization
- Parallelism

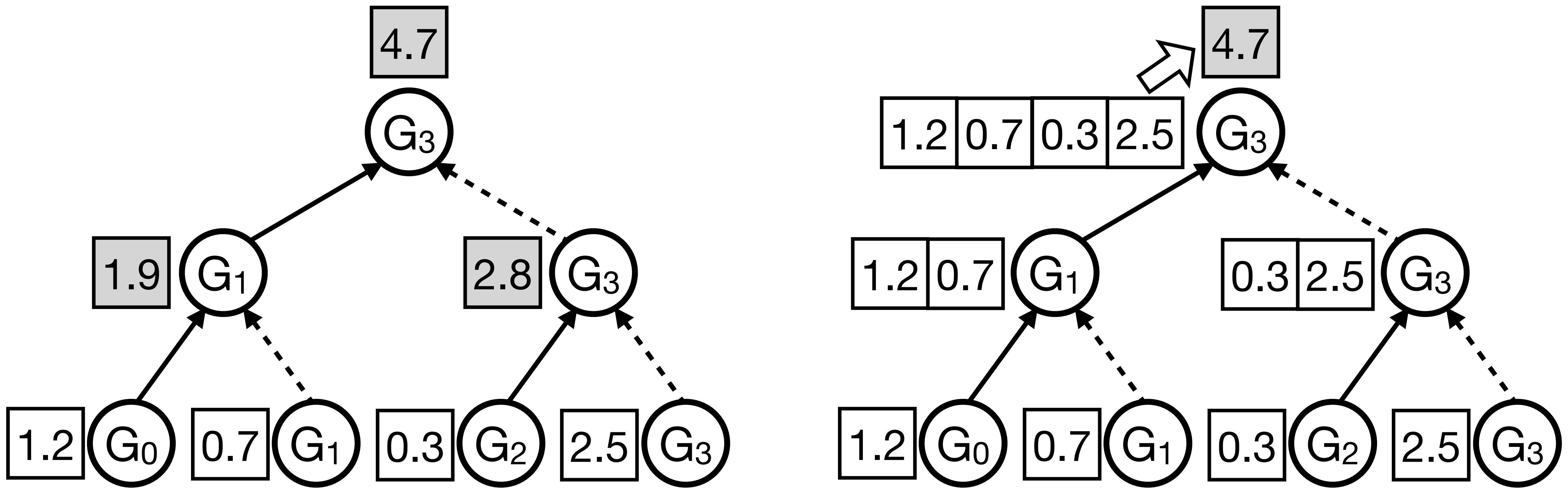
Balance
- Imbalanced
- Balanced

How to describe the design space?

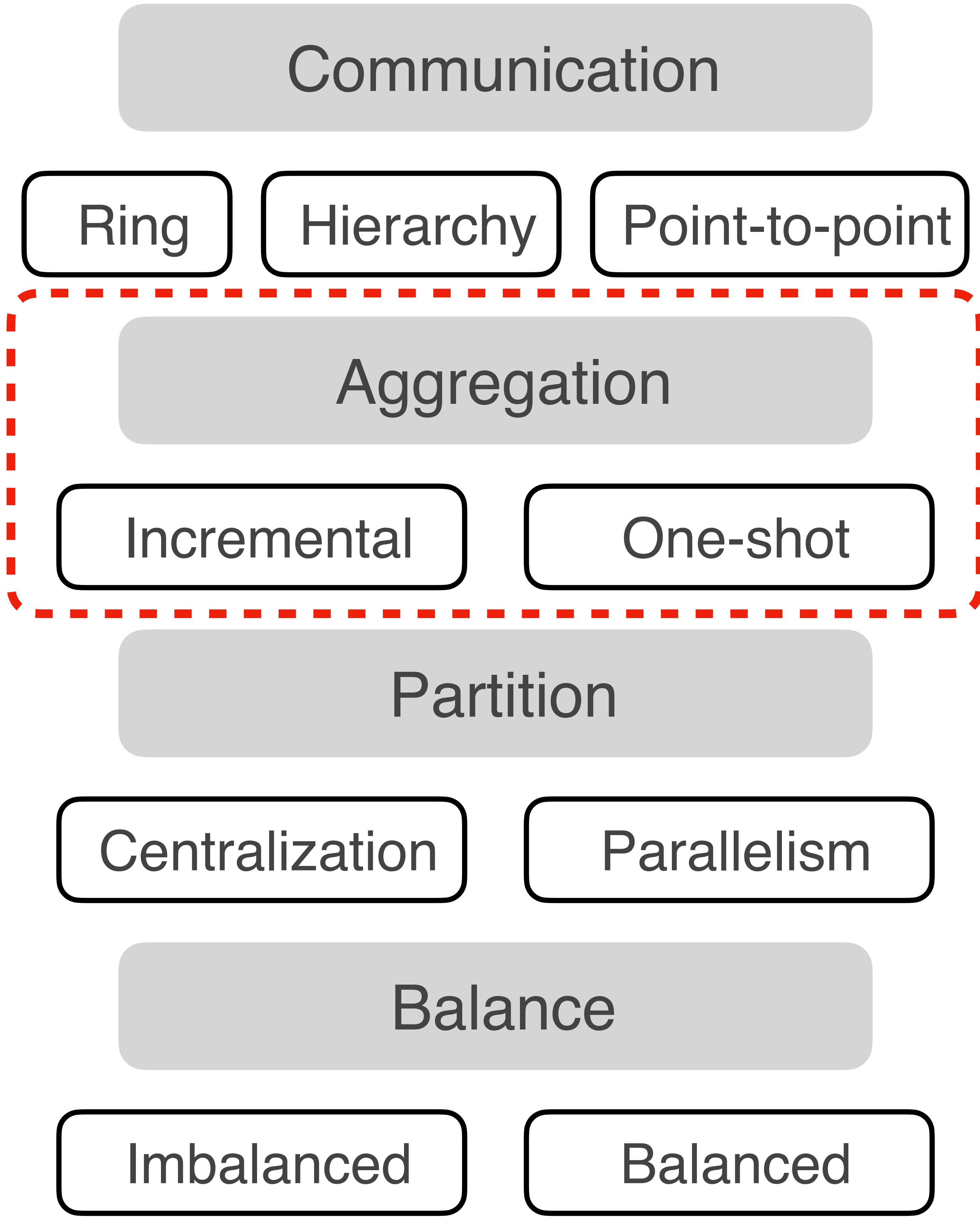
Four dimensions

- Sparse tensor before synchronization

- Aggregation dimension

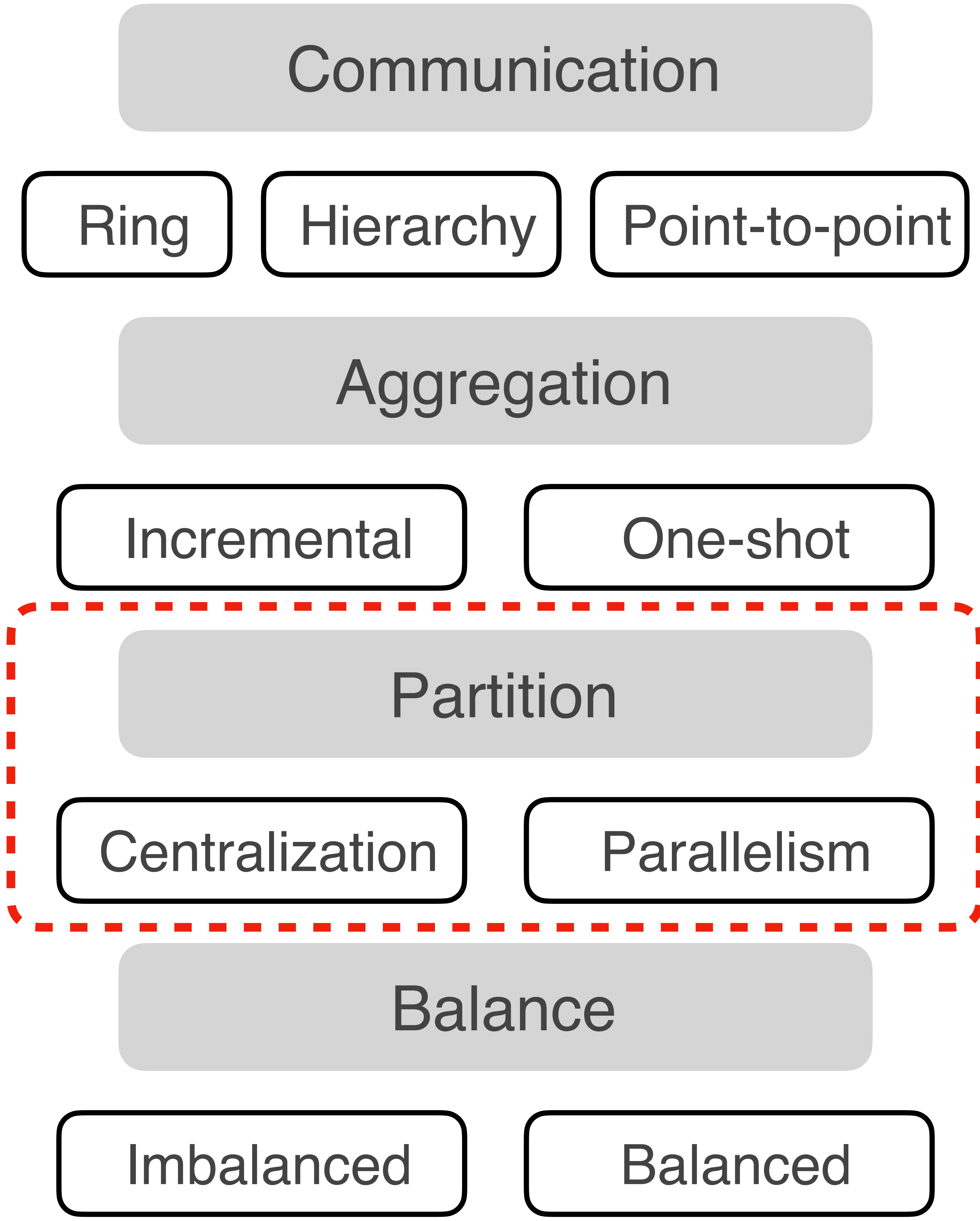


(a) Incremental (b) One-shot

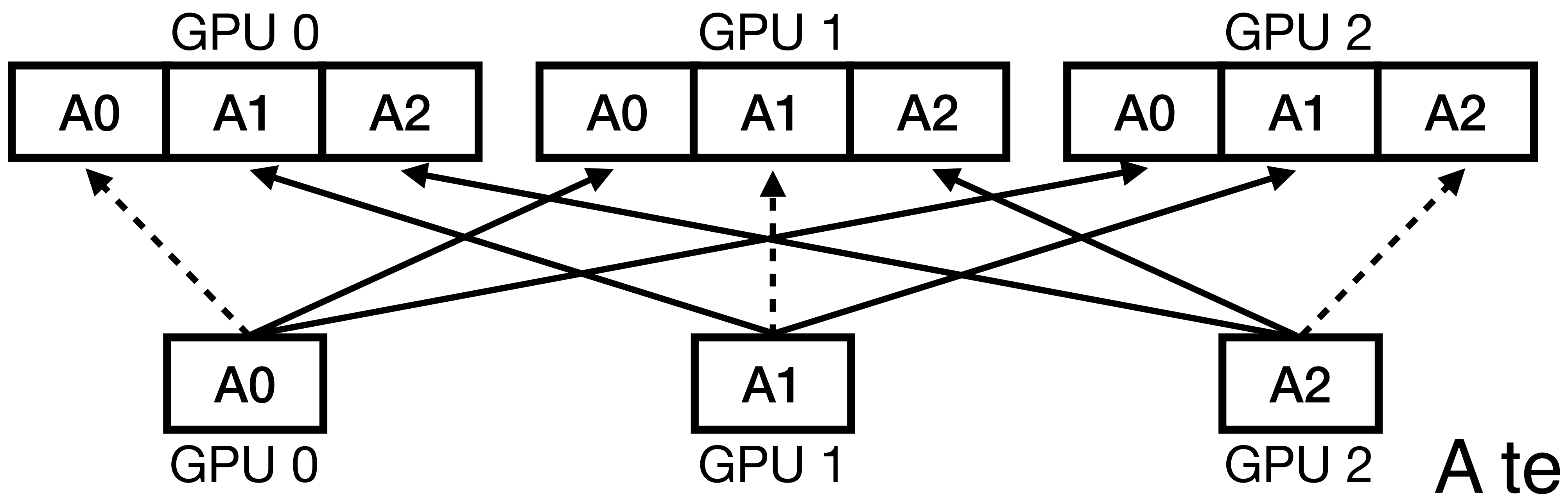


How to describe the design space?

Four dimensions

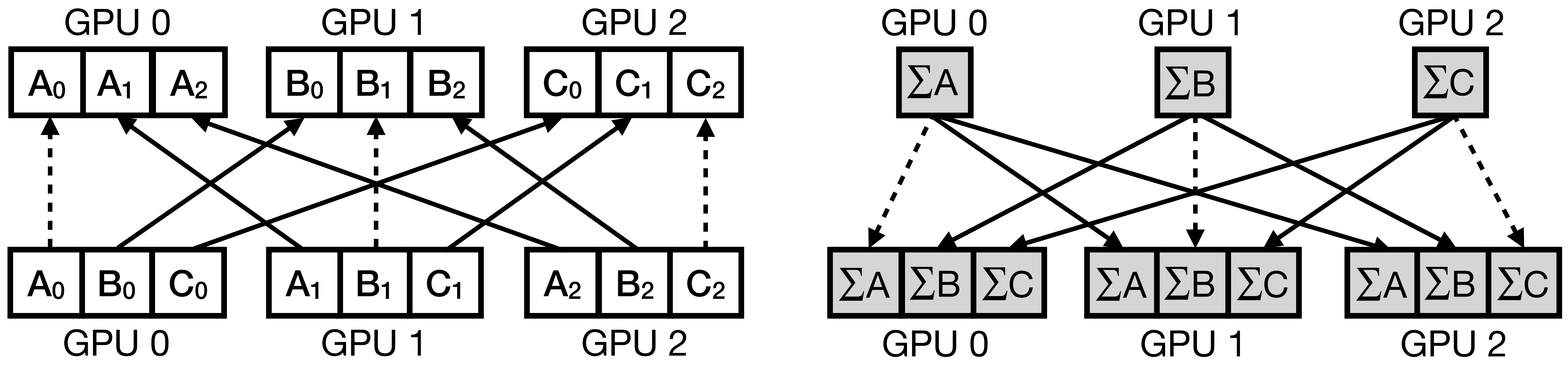


- Partition dimension



(a) Centralization

A tensor is communicated as a whole



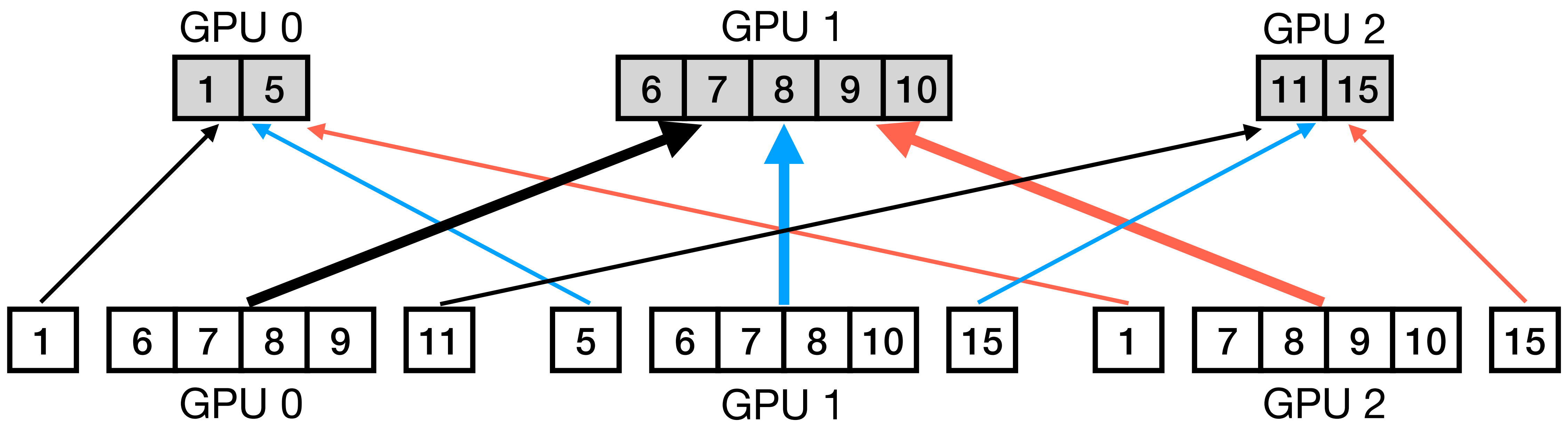
(b) Parallelism

A tensor is partitioned before communication

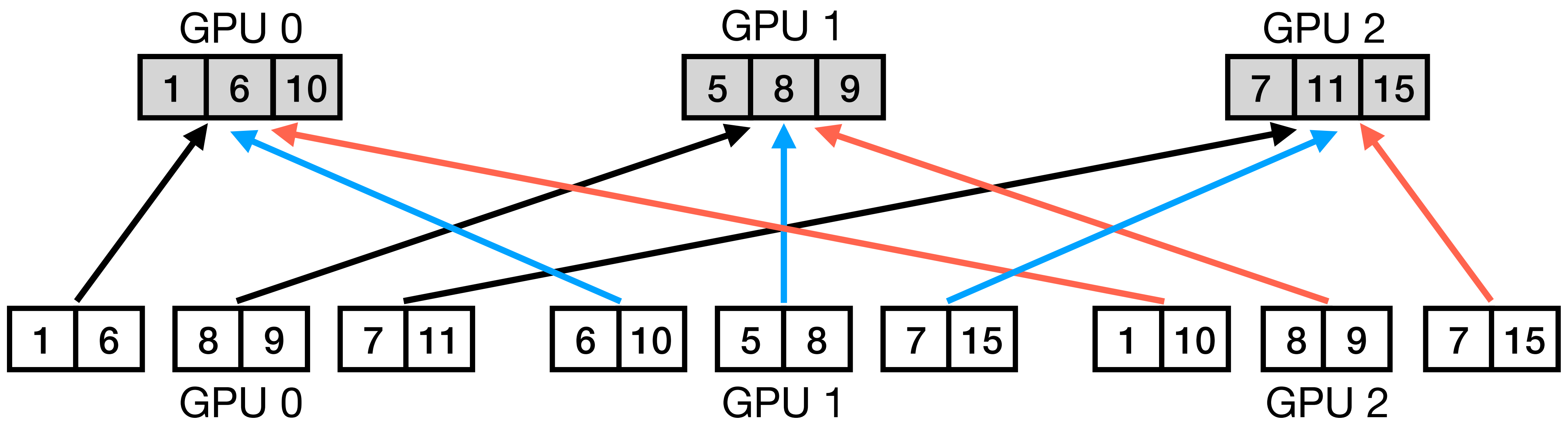
How to describe the design space?

Four dimensions

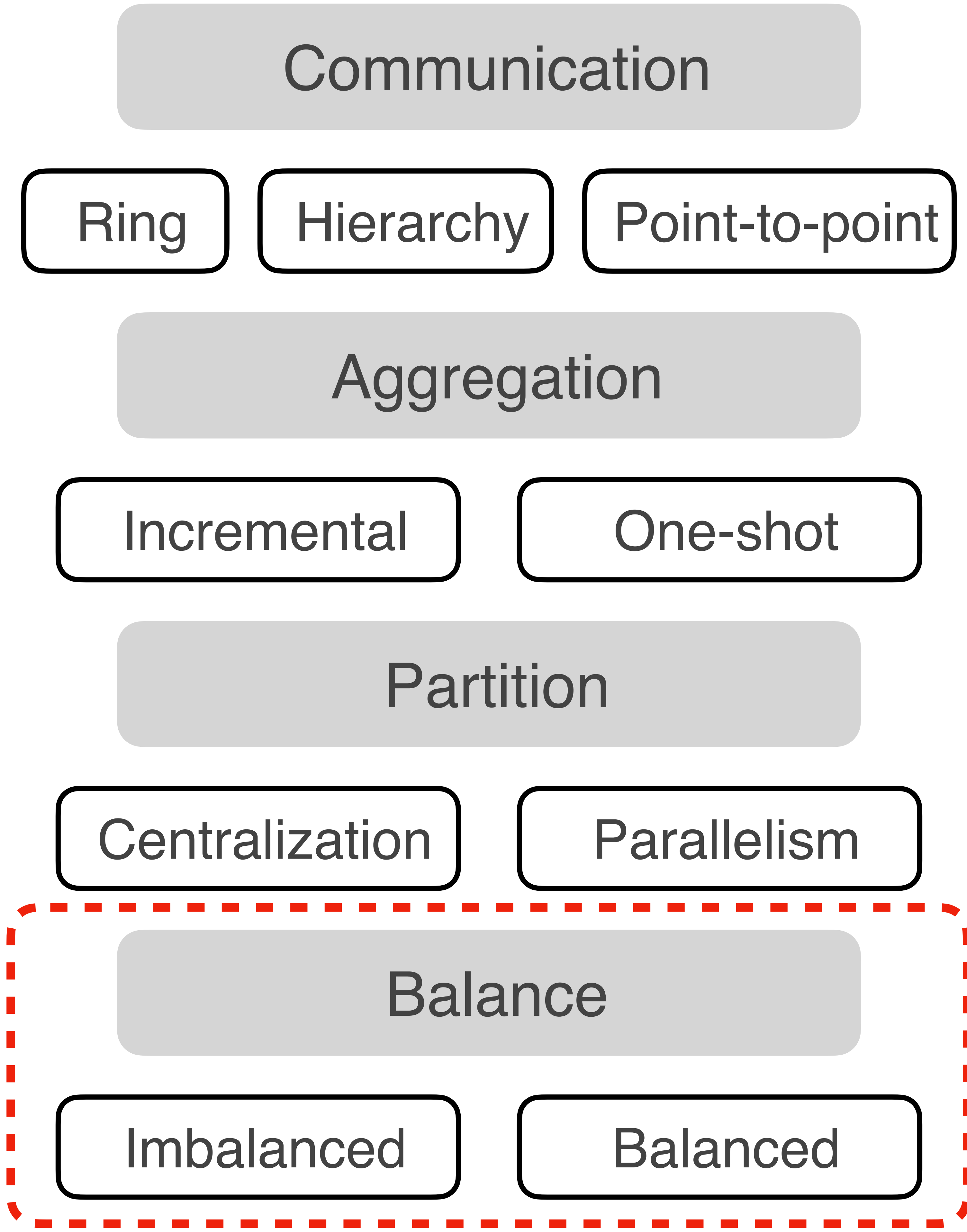
- Balance dimension



(a) Imbalanced communication



(b) Balanced communication



Expressiveness

Describe schemes for sparse tensor synchronization

- All existing schemes can be described by the four dimensions

Schemes	Communication	Aggregation	Partition	Balance
AGsparse [1]	Ring, Hierarchy, Point-to-point	One-shot	Centralization	N/A
SparCML [2]	Hierarchy	Incremental	Centralization	N/A
Sparse PS [3]	Point-to-point	One-shot	Parallelism	Imbalanced
OmniReduce [4]	Point-to-point	One-shot	Parallelism	Imbalanced

- The four dimensions can describe the whole design space

[1] Pytorch distributed: Experiences on accelerating data parallel, VLDB 2020
[2] Sparcml: High-performance sparse communication for machine learning, SC 2019
[3] Scaling distributed machine learning with the parameter server, OSDI 2014
[4] Efficient sparse collective communication and its application, SIGCOMM 2021

What is the optimal scheme?

Based on the four dimensions

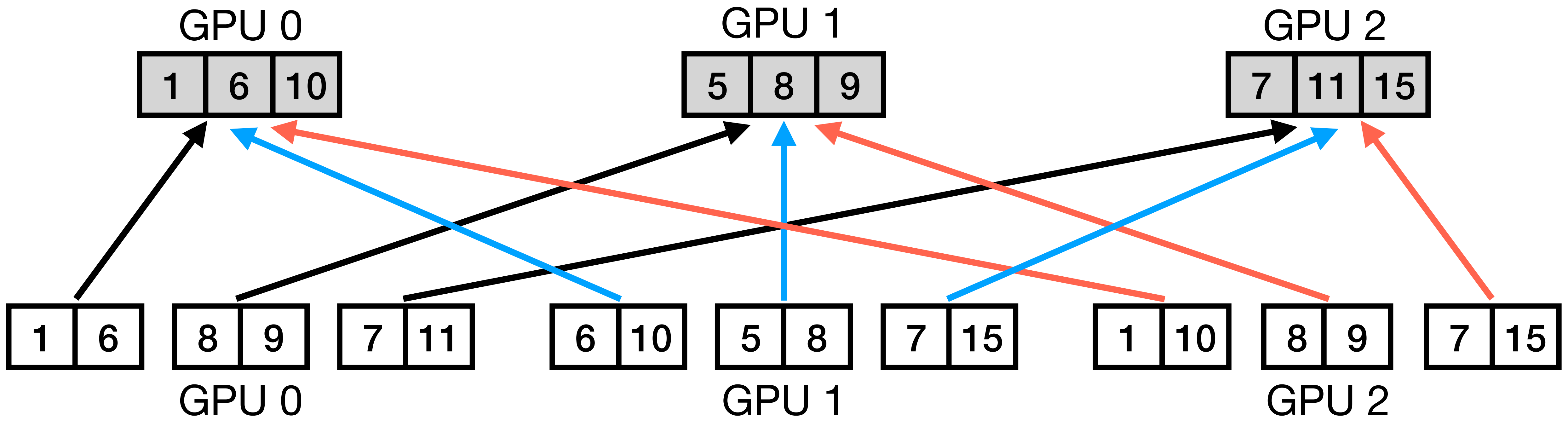
- Problem statement

What is the optimal scheme to **minimize the communication time** of sparse tensor synchronization in DDL?

Find the optimal scheme

Based on the four dimensions

- “Balanced Parallelism”



“Balanced Parallelism”

Communication

Ring | Hierarchy | Point-to-point

Aggregation

Incremental | One-shot

Partition

Centralization | Parallelism

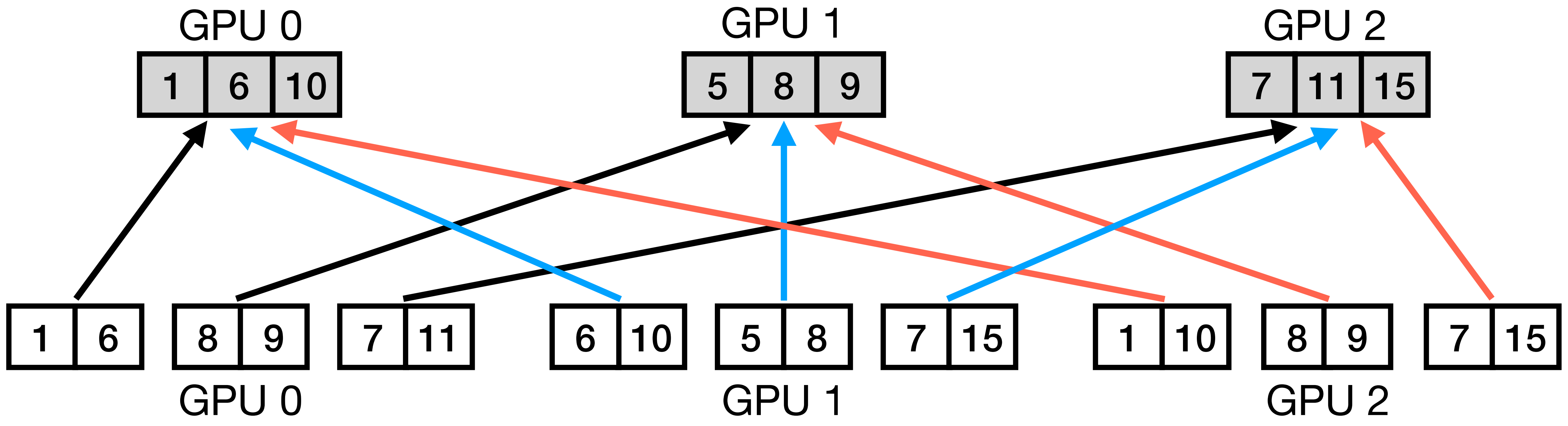
Balance

Imbalanced | Balanced

Find the optimal scheme

Based on the four dimensions

- "Balanced Parallelism"



- Sketch of proof
  - Parallelism can leverage the overlaps among sparse tensors
  -
  -

We have a formal proof in the thesis

"Balanced Parallelism"

Communication

Ring | Hierarchy | Point-to-point

Aggregation

Incremental | One-shot

Partition

Centralization | Parallelism

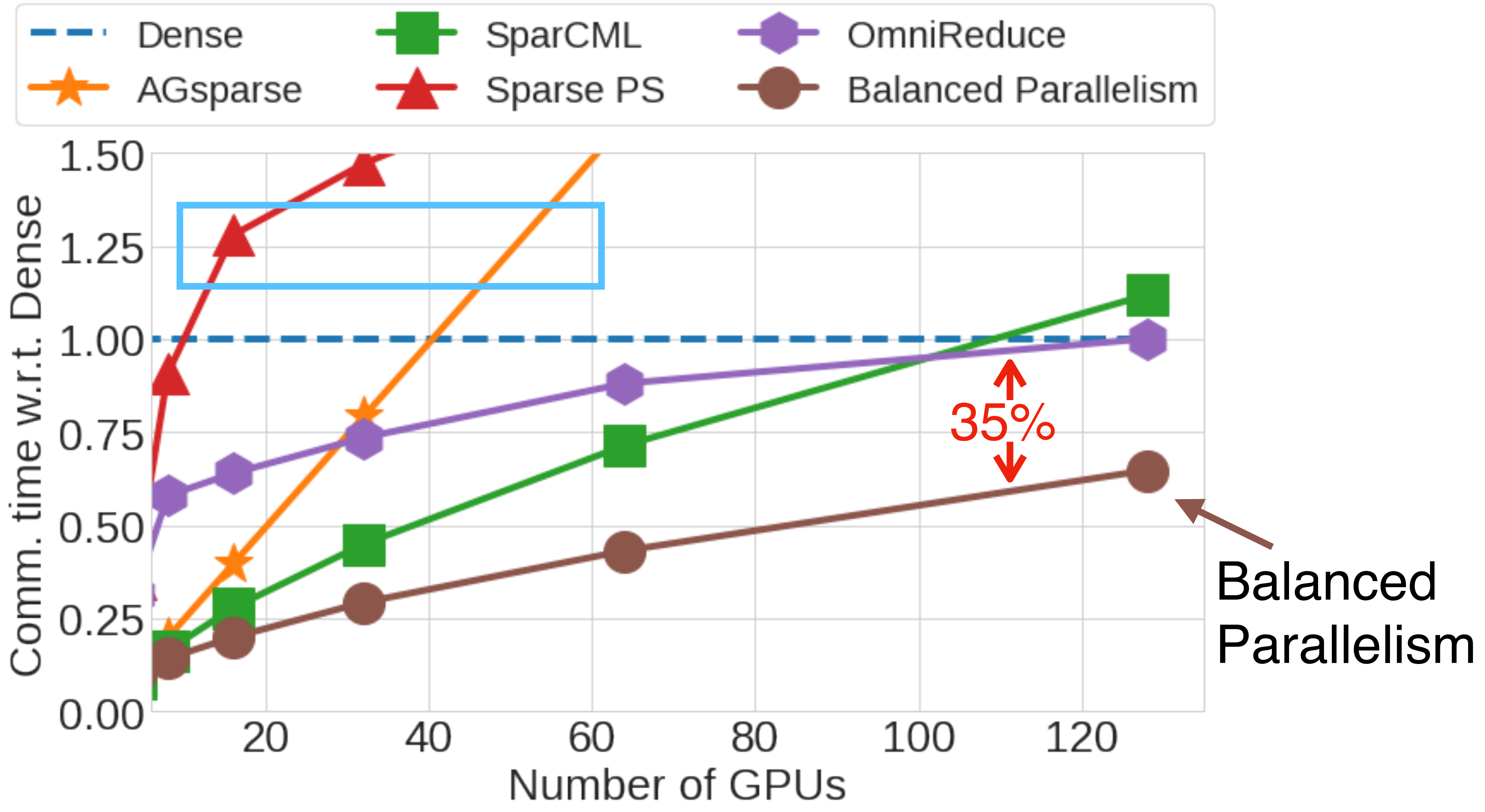
Balance

Imbalanced | Balanced

Numerical comparison

Take sparse tensors in NMT as examples

- Communication time of sparse tensor synchronization

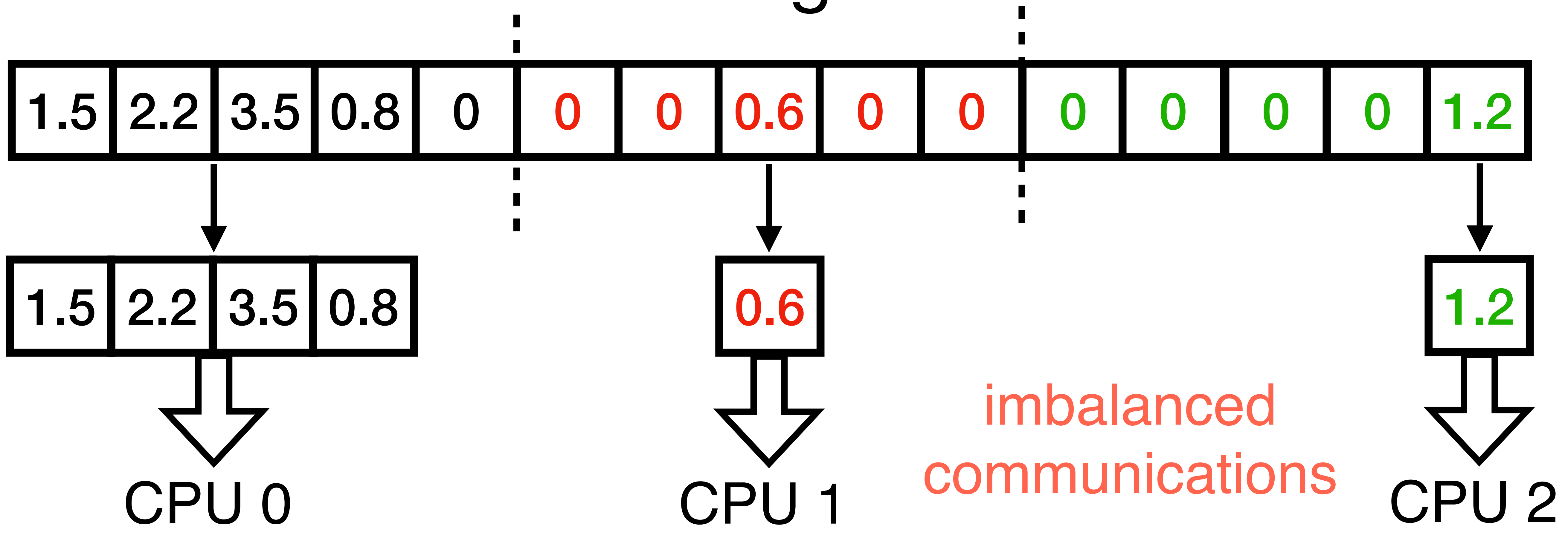


No existing realization of Balanced Parallelism

How to realize “Balanced Parallelism”?

- Skewed distribution of non-zero gradients

Sparse PS[1]
OmniReduce[2]



Communication

Point-to-point

Aggregation

One-shot

Partition

Parallelism

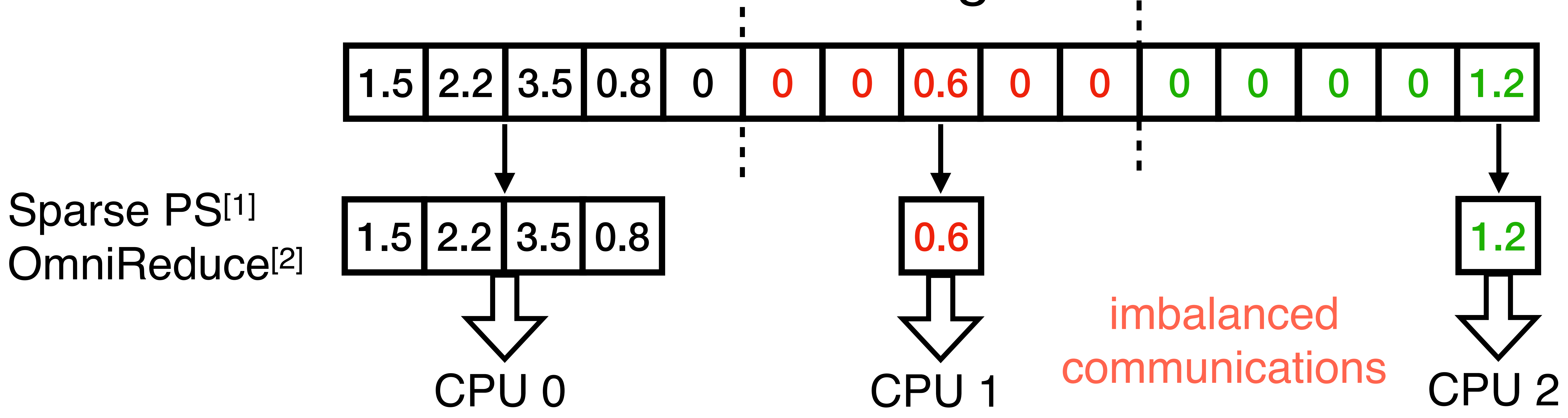
Balance

Balanced

[1] Scaling distributed machine learning with the parameter server, OSDI 2014
[2] Efficient sparse collective communication and its application, SIGCOMM 2021

How to realize "Balanced Parallelism"?

- Skewed distribution of non-zero gradients



- Challenge: how to achieve balanced communications?

Communication

Point-to-point

Aggregation

One-shot

Partition

Parallelism

Balance

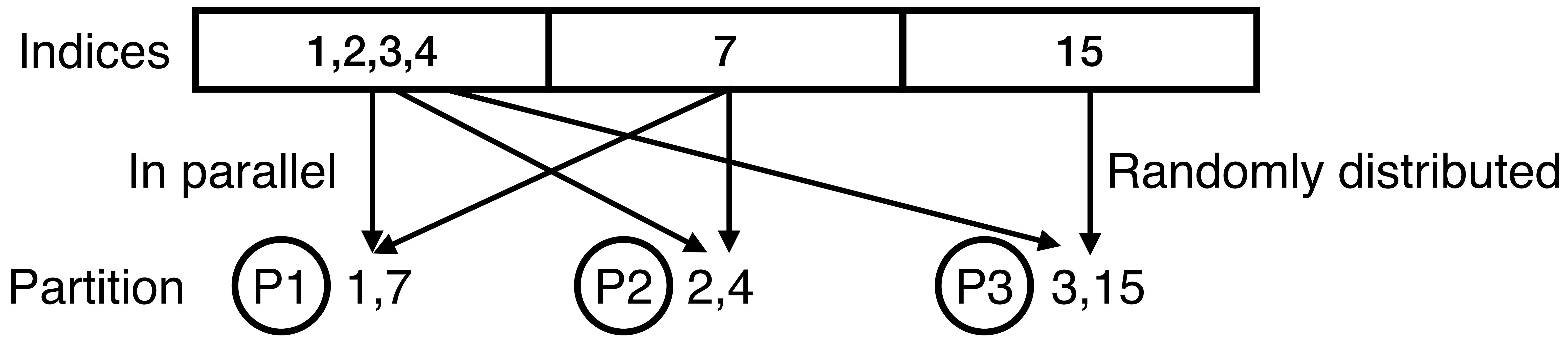
Balanced

[1] Scaling distributed machine learning with the parameter server, OSDI 2014
[2] Efficient sparse collective communication and its application, SIGCOMM 2021

Solution: A hierarchical hashing algorithm

Parallel computing on GPUs for hashing

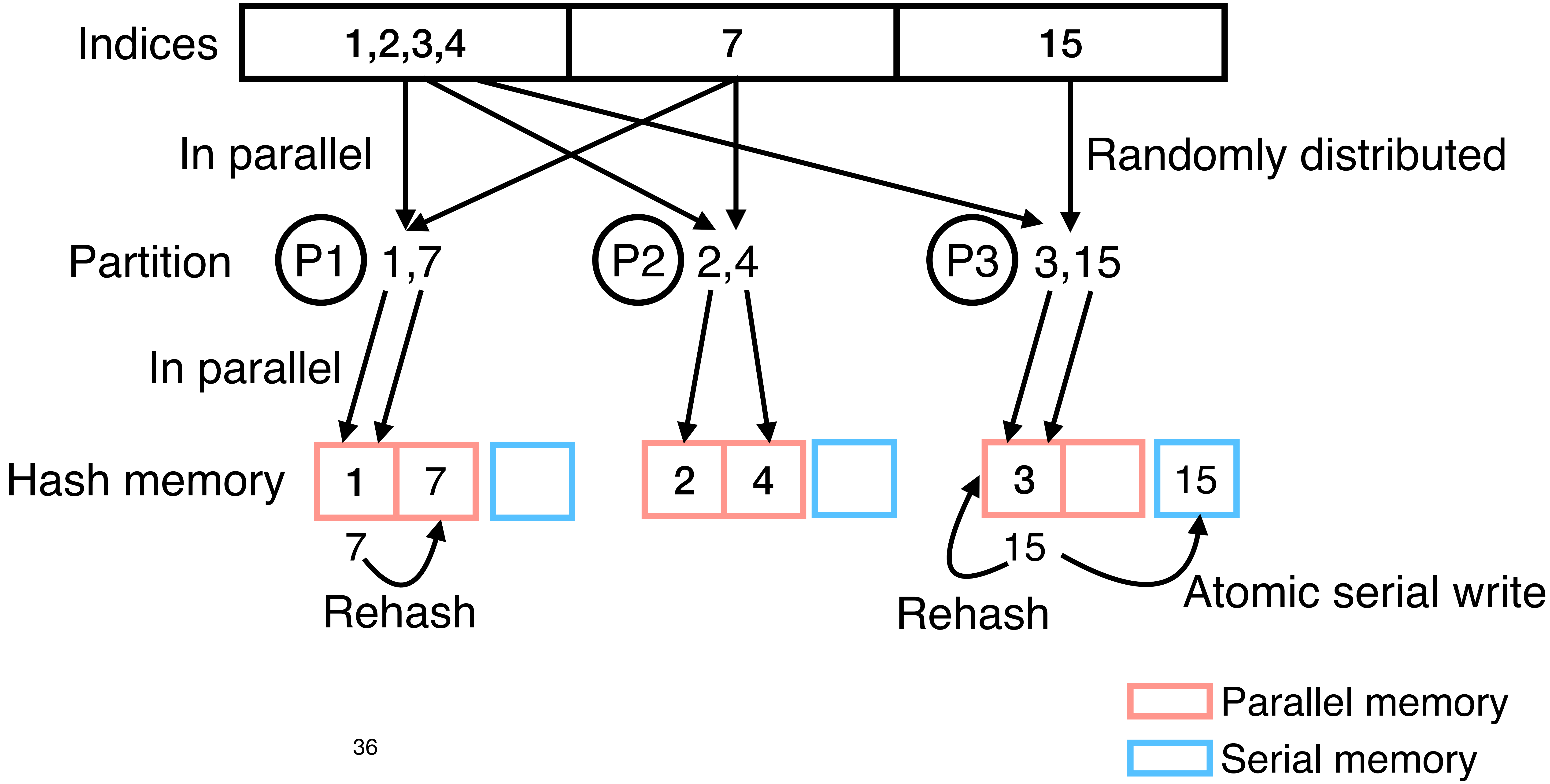
- Level-1: hash indices of non-zero gradient for partitions



Solution: A hierarchical hashing algorithm

Parallel computing on GPUs for hashing

- Level-2: rehash indices for available locations within each partition



Solution: A hierarchical hashing algorithm

Properties

- Level-2: rehash indices for available locations within each partition

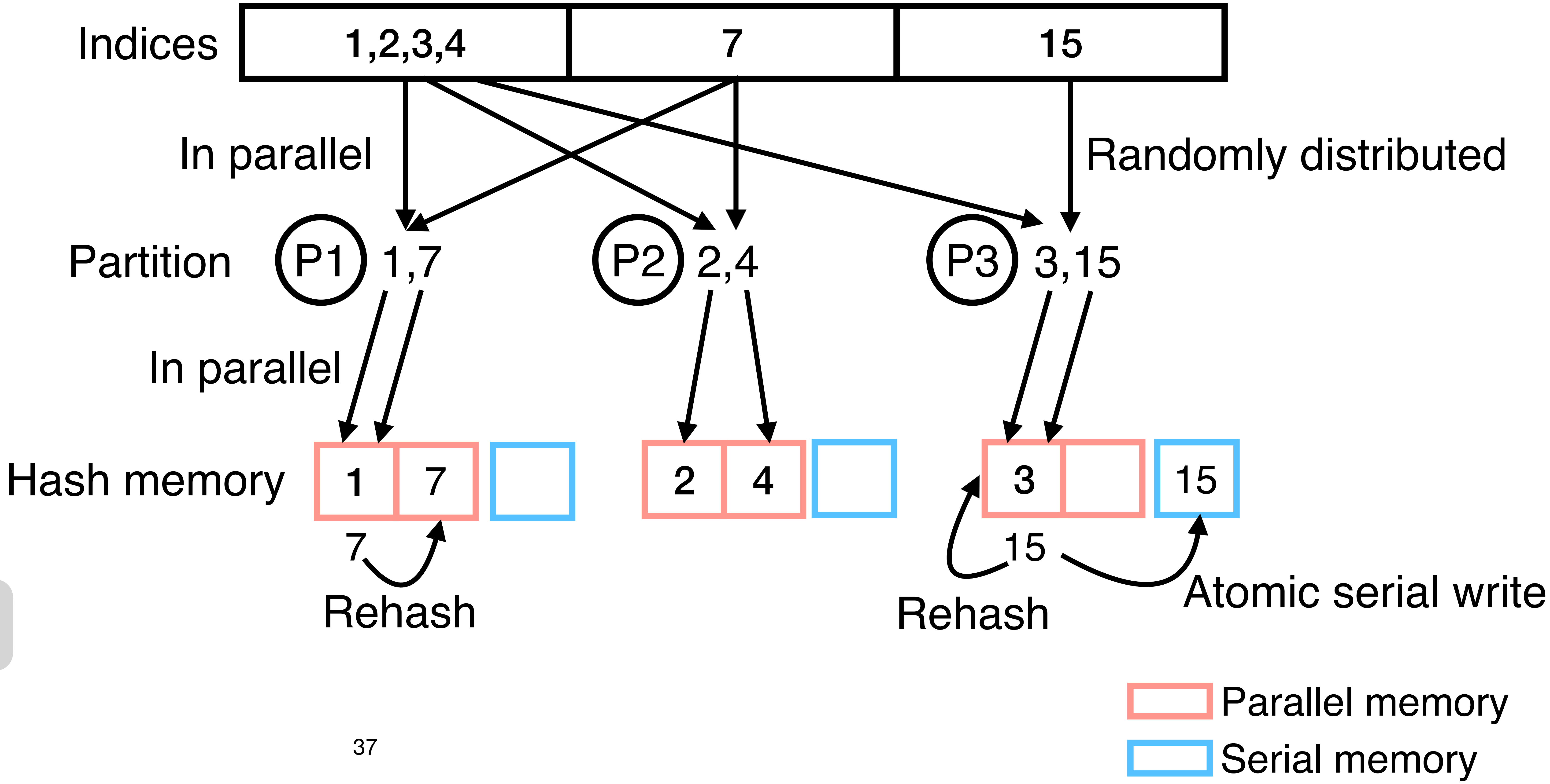
Guaranteed load balance

No information loss

Small hash memory size

Strength in parallel computing

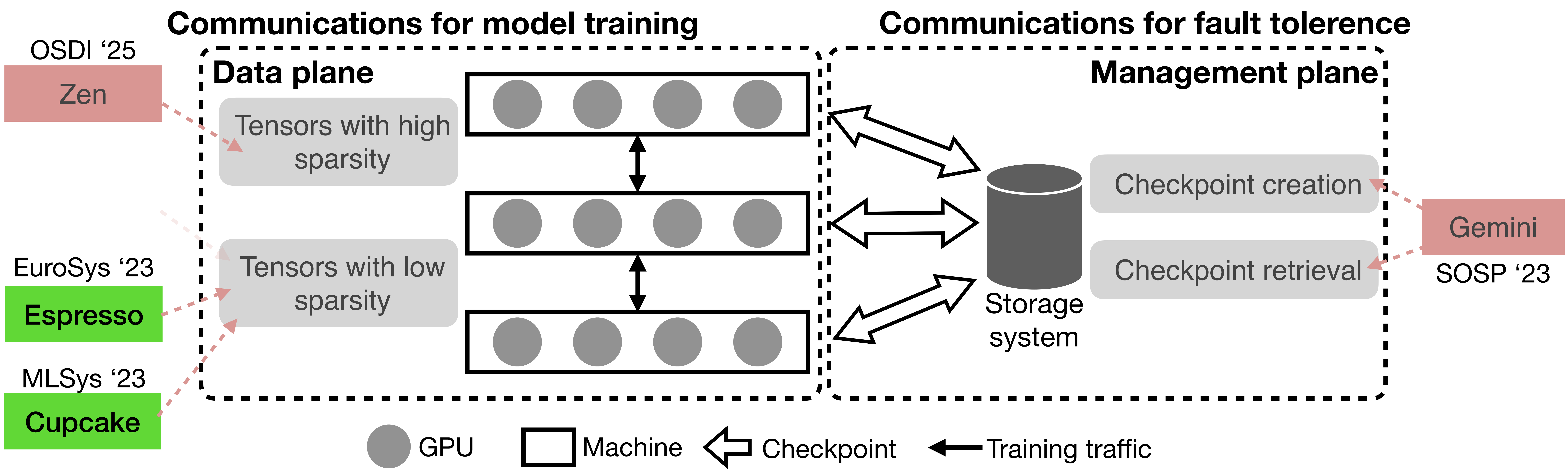
Hash consistency among workers



The next research projects

Scaling deep learning by optimizing communications

- Thesis work



Gradient compression for communications

- Some deep learning models don't have high sparsity

Model	Task	Dataset	Batch size	Sparsity
VGG16	Computer vision	ImageNet	32 images	6.4%
ResNet101	Computer vision	ImageNet	32 images	7.9%
UGATIT	Computer vision	selfie2anime	2 images	9.6%

Gradient compression for communications

- Some deep learning models don't have high sparsity

Model	Task	Dataset	Batch size	Sparsity
VGG16	Computer vision	ImageNet	32 images	6.4%
ResNet101	Computer vision	ImageNet	32 images	7.9%
UGATIT	Computer vision	selfie2anime	2 images	9.6%

- Gradient compression (GC) shrinks communication traffic volume
  - It has negligible impacts on model accuracy [1]

[1] GRACE: A compressed communication framework for distributed machine learning, ICDCS '21

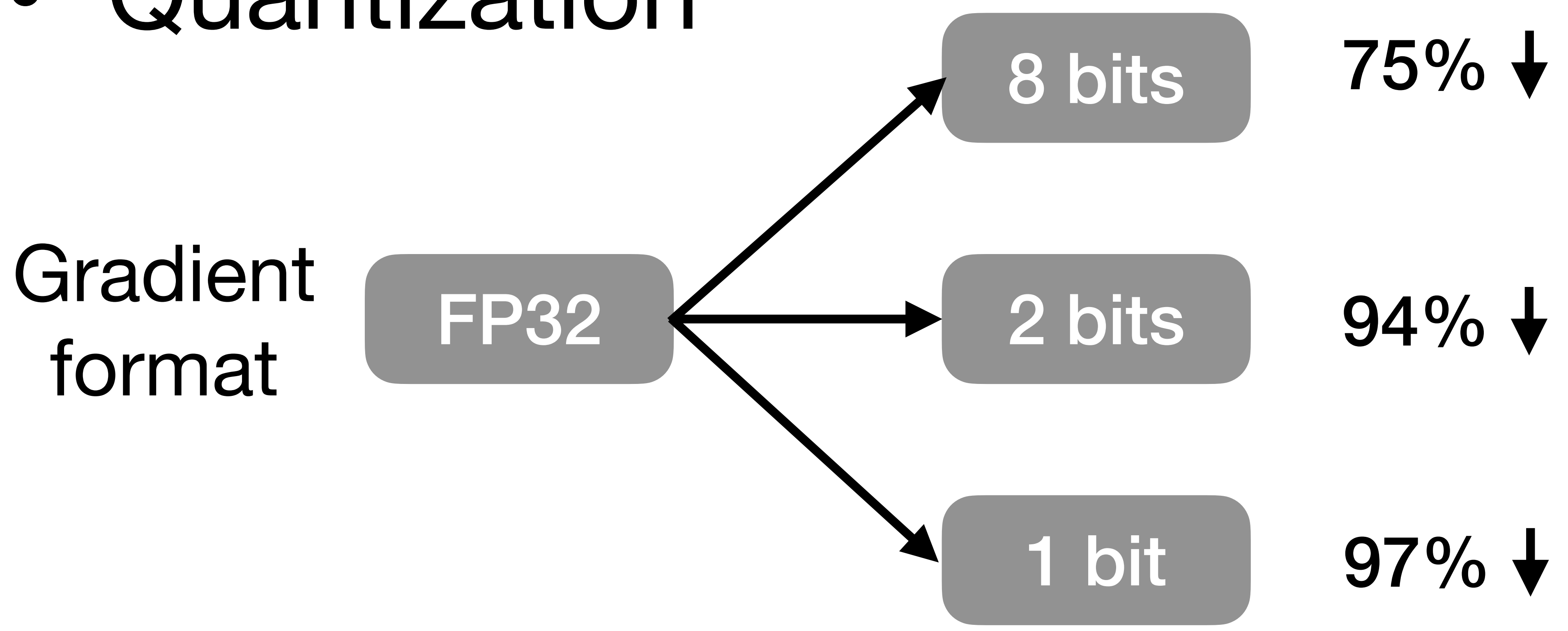
Gradient compression for communications

- Some deep learning models don’t have high sparsity

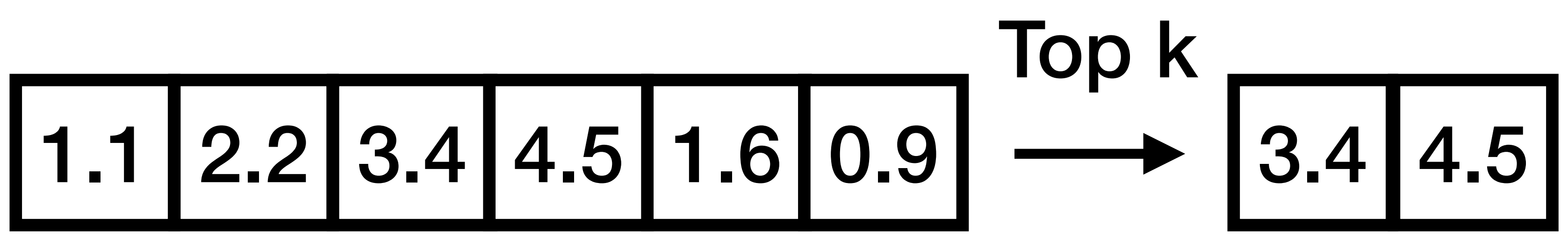
Model	Task	Dataset	Batch size	Sparsity
VGG16	Computer vision	ImageNet	32 images	6.4%
ResNet101	Computer vision	ImageNet	32 images	7.9%
UGATIT	Computer vision	selfie2anime	2 images	9.6%

- Gradient compression (GC) shrinks communication traffic volume

- Quantization

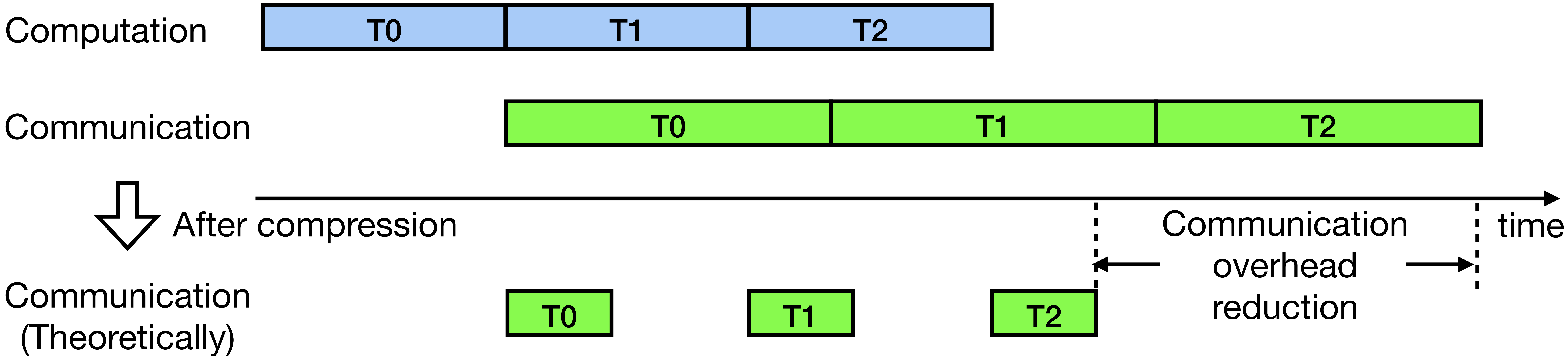


- Sparsification



Gradient compression (GC) in theory

- GC reduces communication overhead

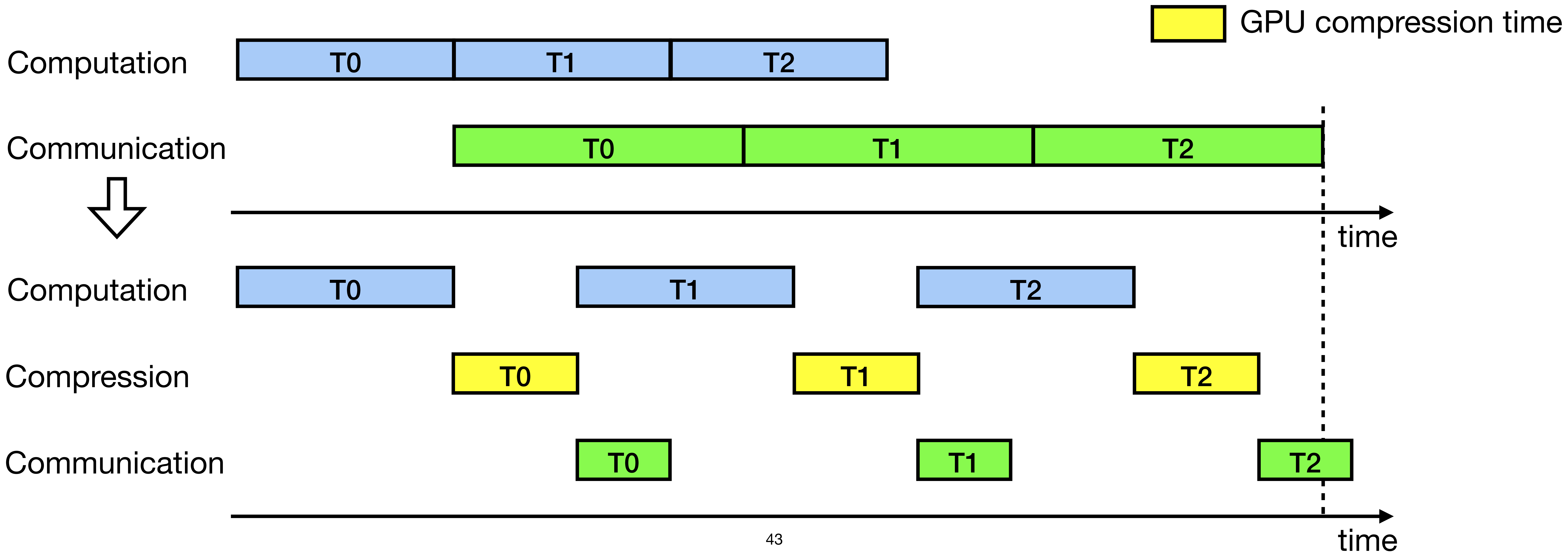


- However, GC algorithms are designed from an algorithmic perspective

Gradient compression (GC) in reality

Use GPU for compression

- GC incurs computation overhead in practice



Unleash the benefits of GC

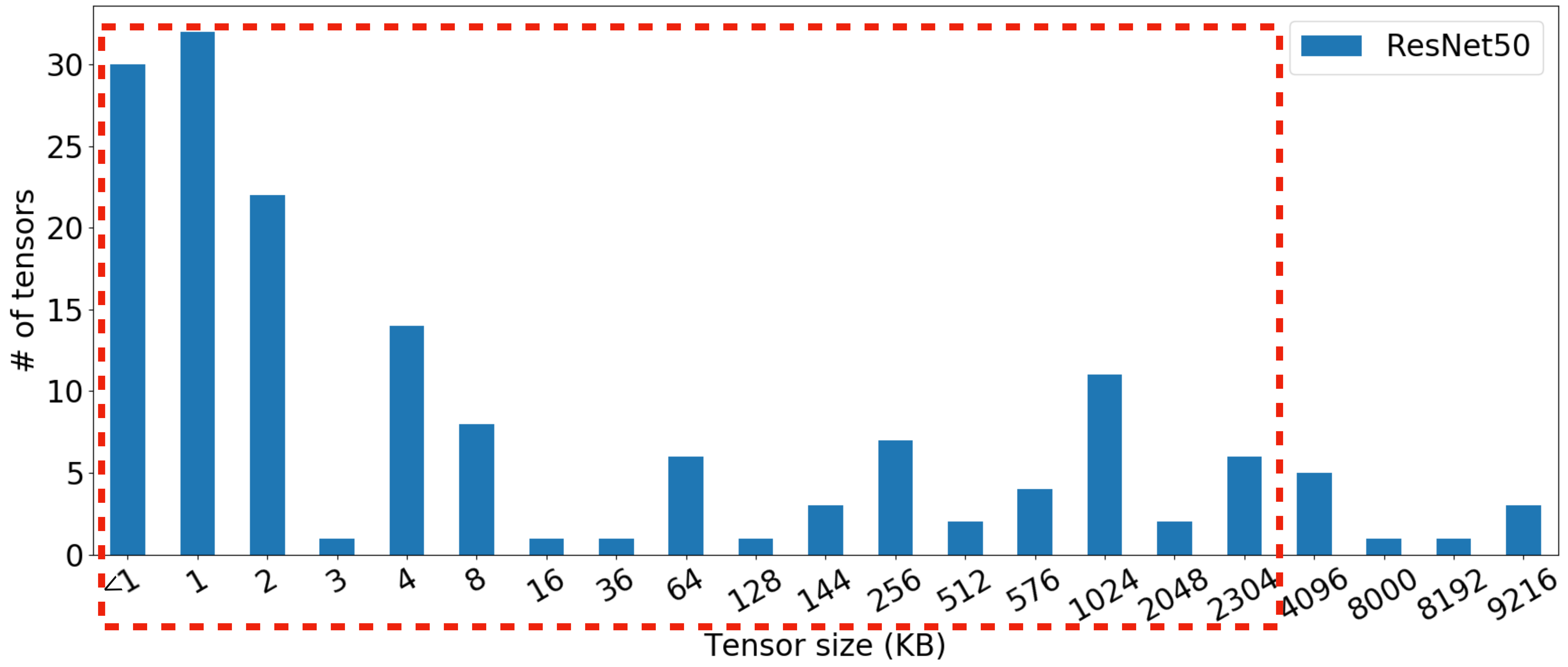
Espresso: search for optimal compression strategy

- Contributions
    - We leverage **both GPUs and CPUs** to perform gradient compression simultaneously
    - We design **a decision tree abstraction** to holistically **describe the search space** of compression strategies
        - Whether to compress each tensor?
        - the type of compute resources (e.g., CPUs or GPUs) for compression?
        - the communication schemes for compressed tensors?
    - We devise an **compression decision algorithm** that selects near-optimal strategies in seconds to optimize training throughput of DDL
- Espresso is partially deployed at ByteDance GPU cluster

Unleash the benefits of GC

Cupcake: Fuse tensors for compression

- Existing approaches compress tensor by tensor
  - Invokes compress operations for each tensor
  - fixed overheads to launch and execute kernels in CUDA, even for small tensors
- Deep learning models have many small tensors (<4MB)

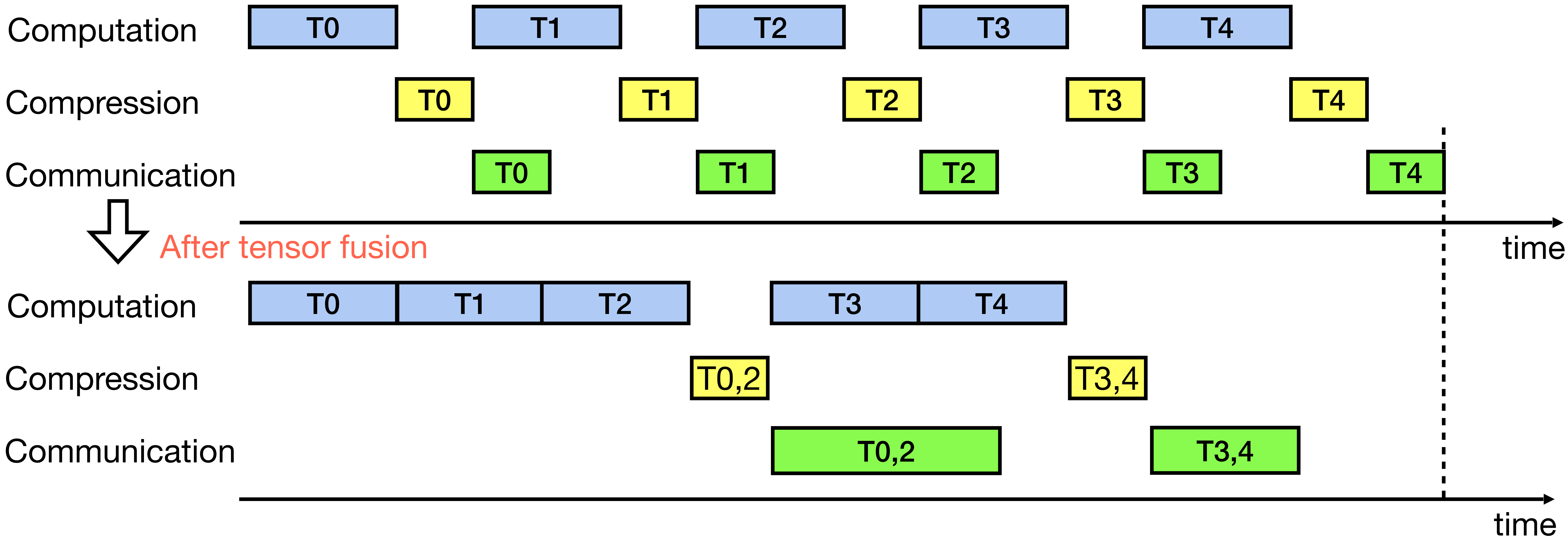


Contributions of Cupcake

- We propose a general **compression optimizer** with **a fusion fashion** for GC algorithms to accelerate the training throughput
- We design an algorithm that **finds the optimal fusion strategy** in seconds
- We build a **compression-enabled system** with this compression optimizer

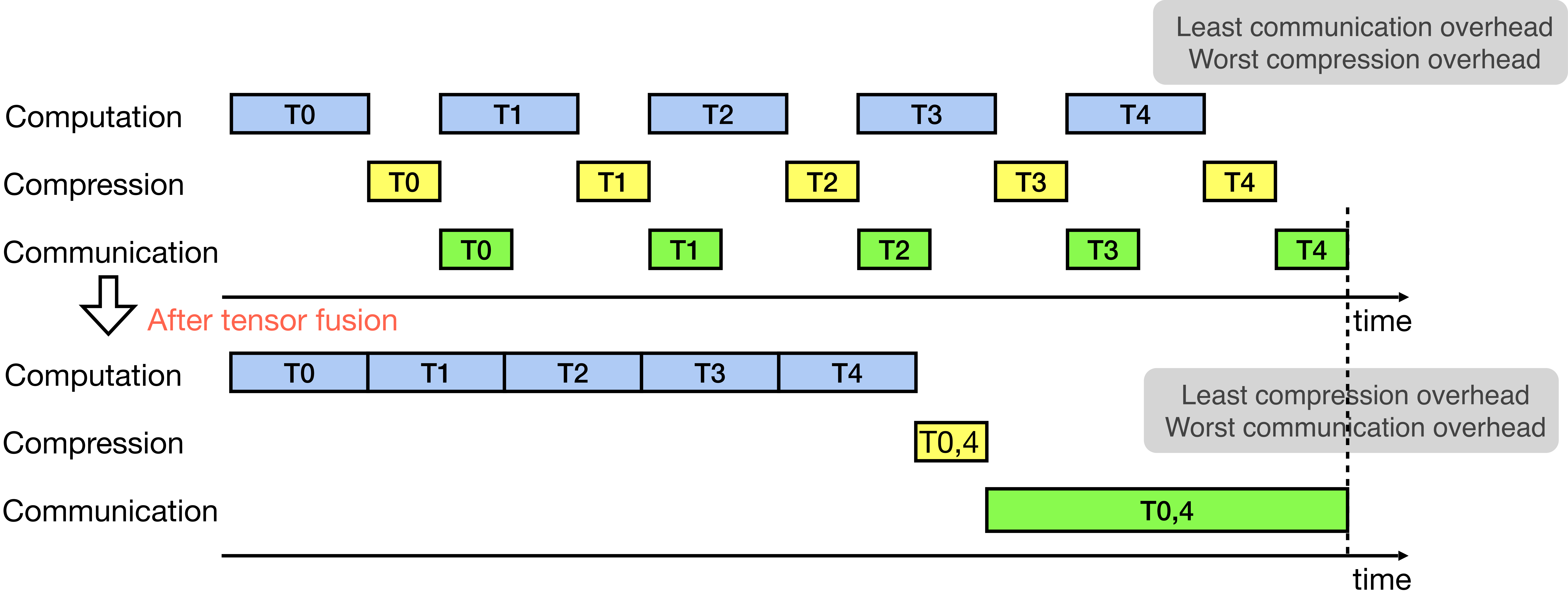
Cupcake

Fuse tensors for compression



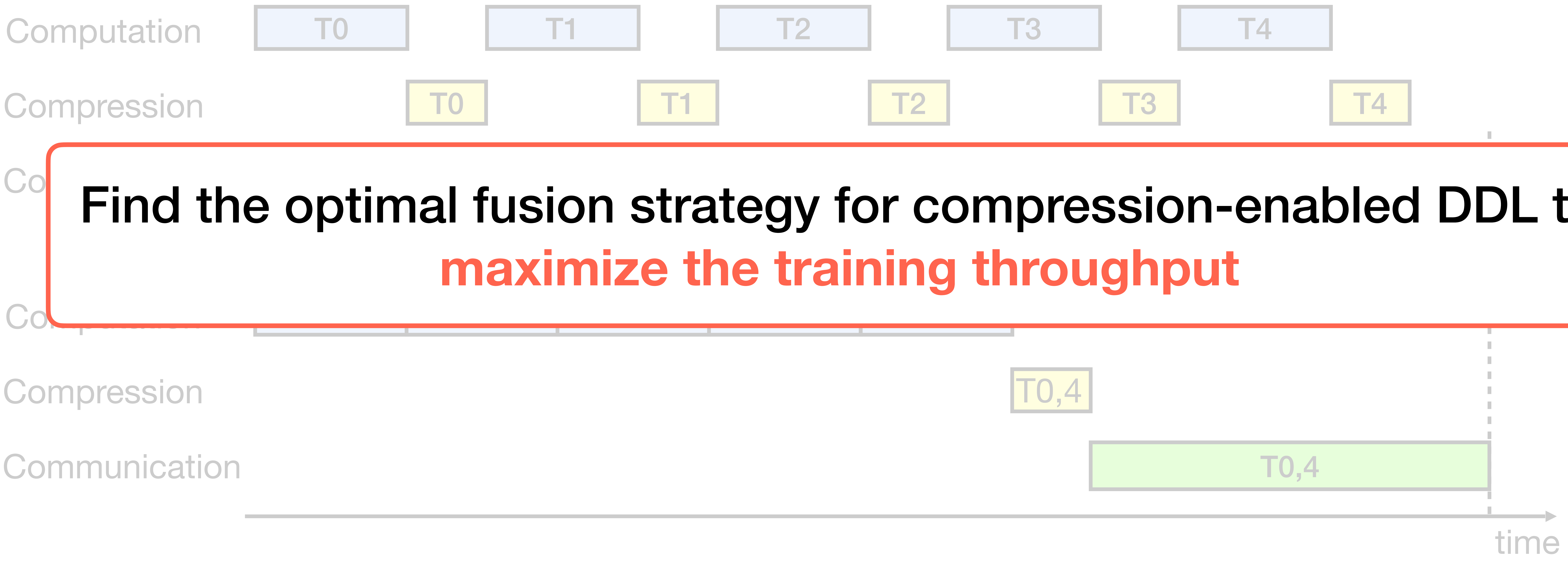
Challenge

Trade-off between compression and communication overhead



Goal

Trade-off between compression and communication overhead



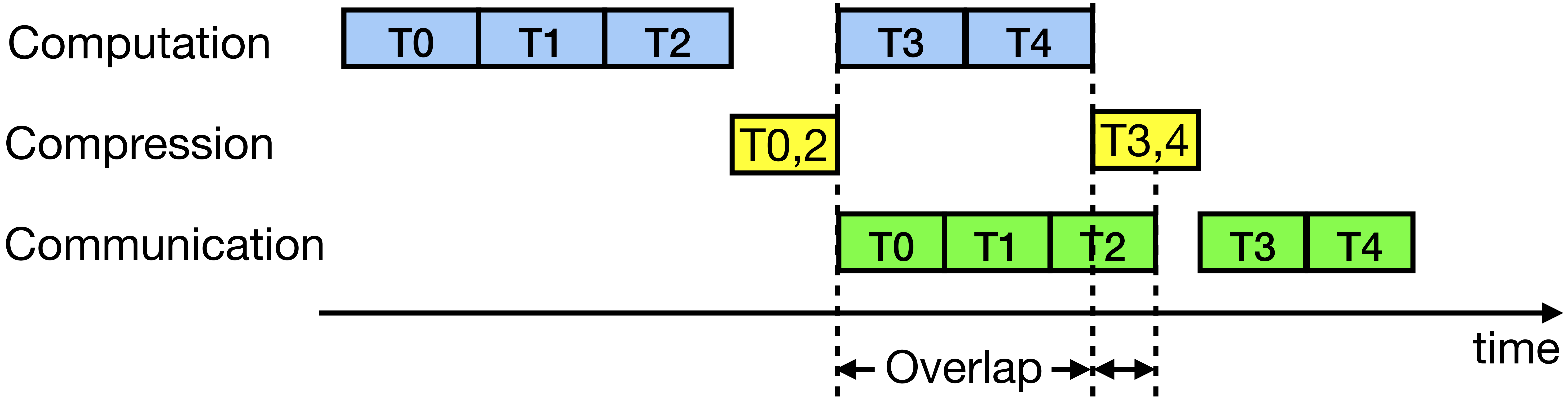
Find the optimal fusion strategy for compression-enabled DDL to **maximize the training throughput**

Cupcake

Find the optimal fusion strategy

- Difficult to formulate the iteration time

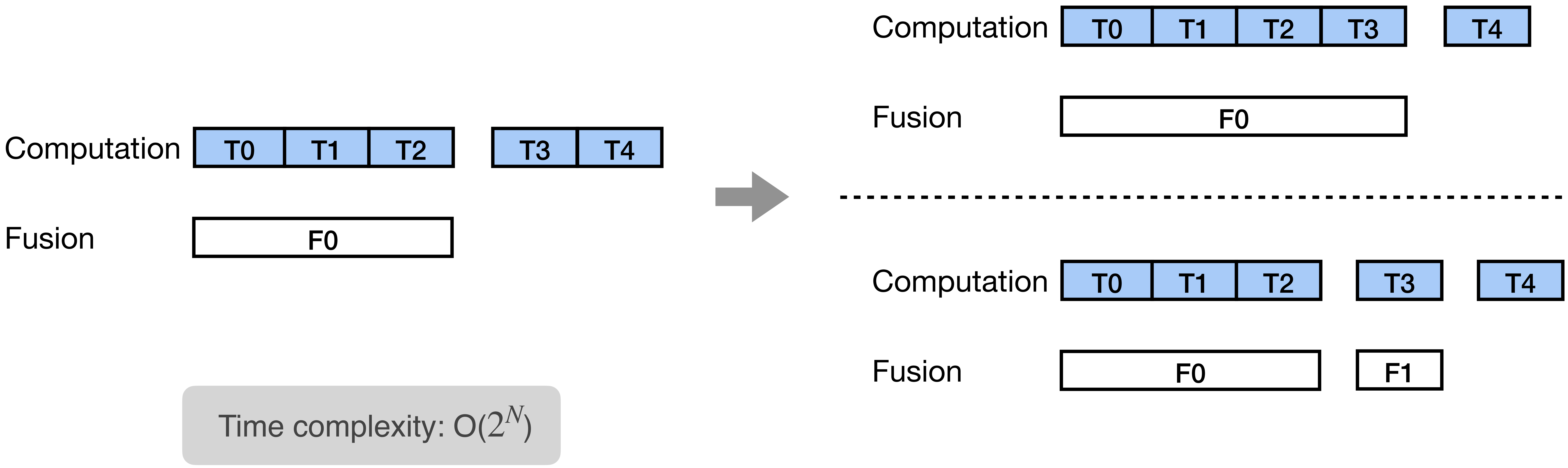
Iteration time = Computation + Communication + Compression − Overlapping



But overlapping time is determined by the intricate interactions among tensors

Search space

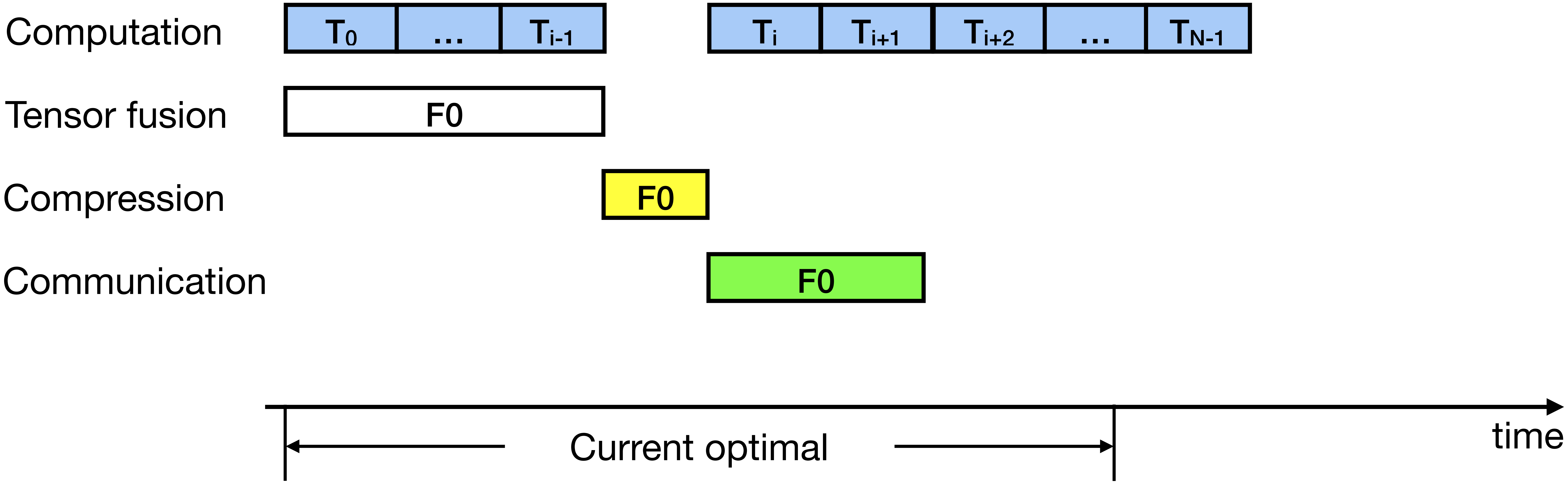
- Exponential time to find the optimal strategy with brute force



Pruning techniques #1

No need to examine all cases for the formation of F0

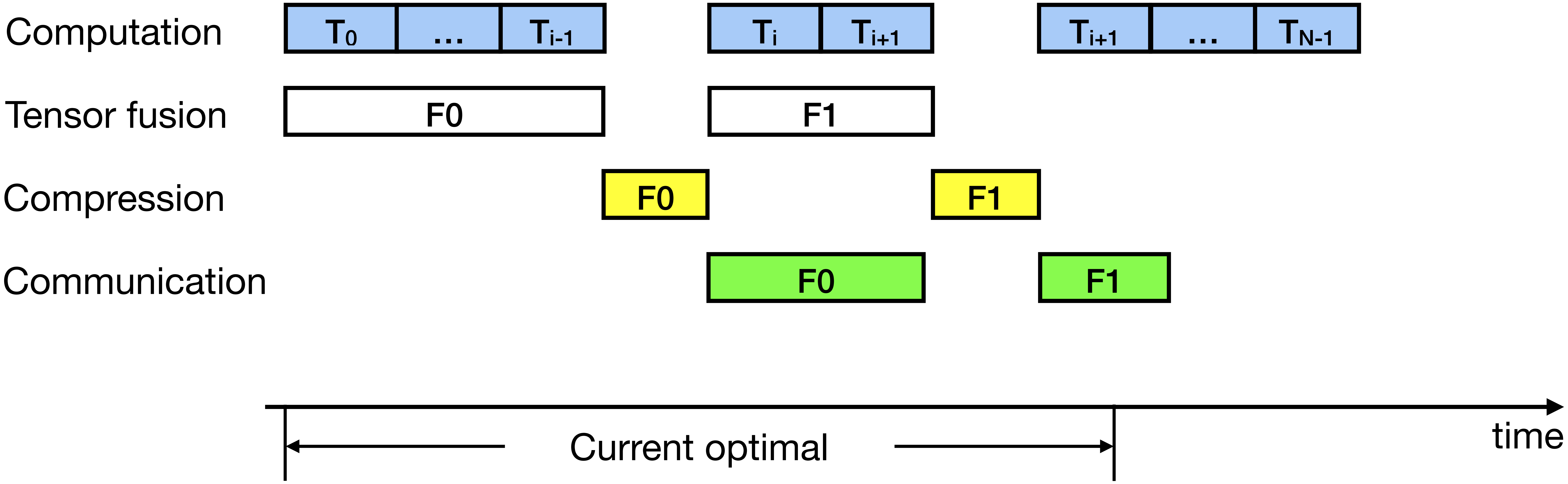
- Examine F0



Pruning techniques #1

No need to examine all cases for the formation of F0

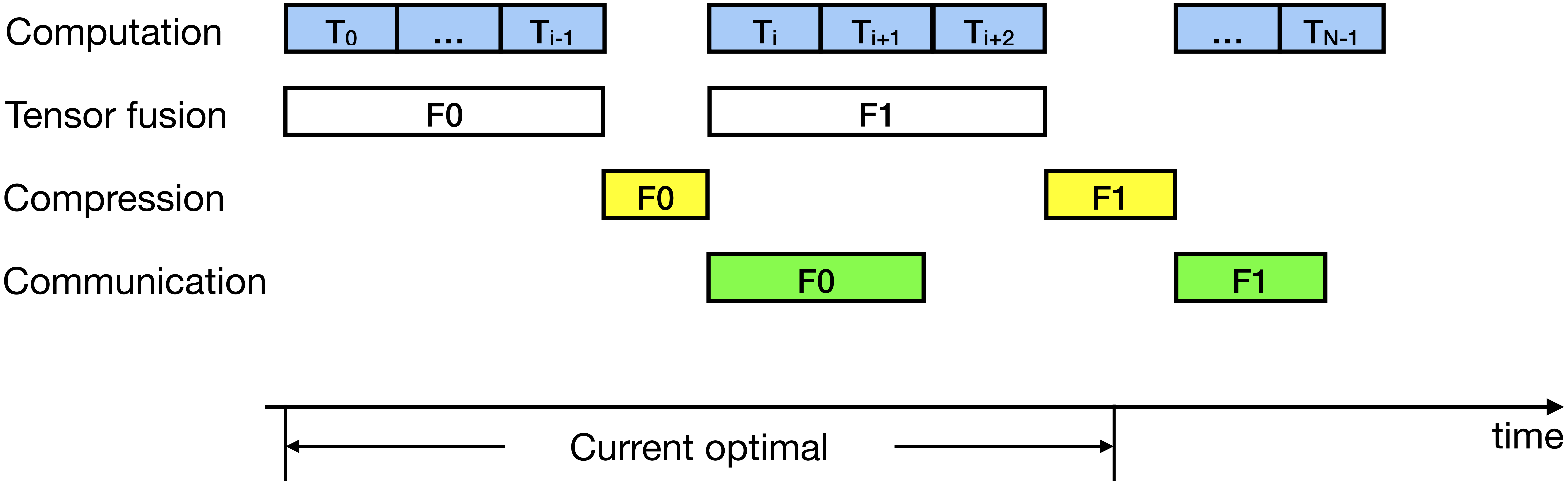
- Examine F0



Pruning techniques #1

No need to examine all cases for the formation of F0

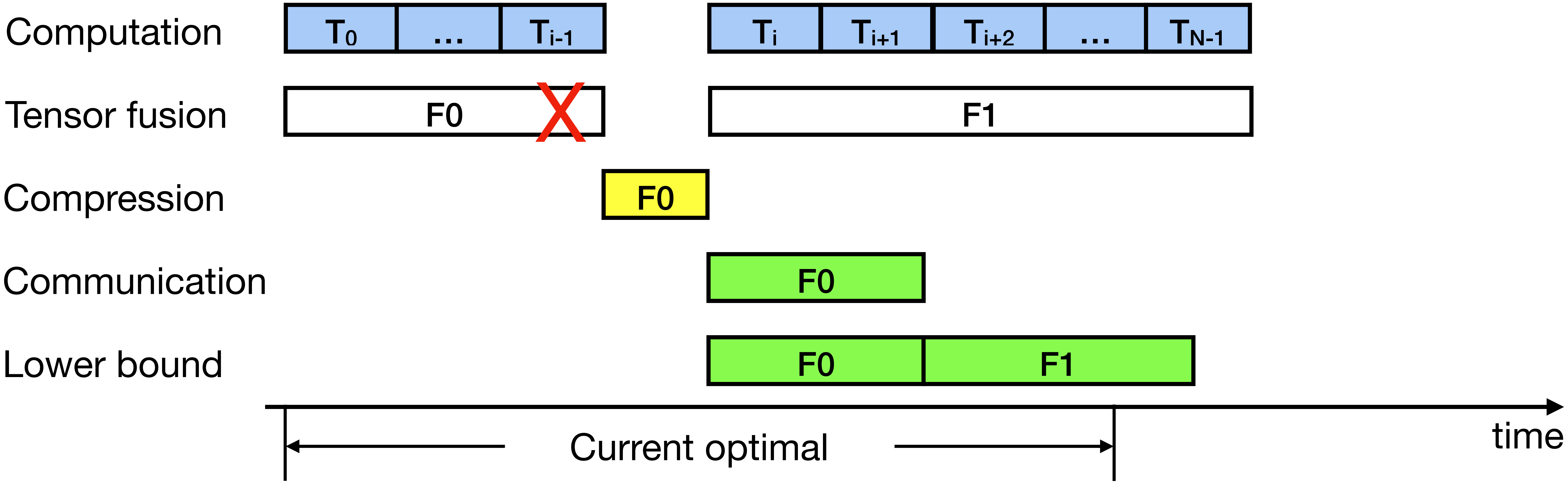
- Examine F0



Pruning techniques #1

No need to examine all cases for the formation of F0

- Examine F0

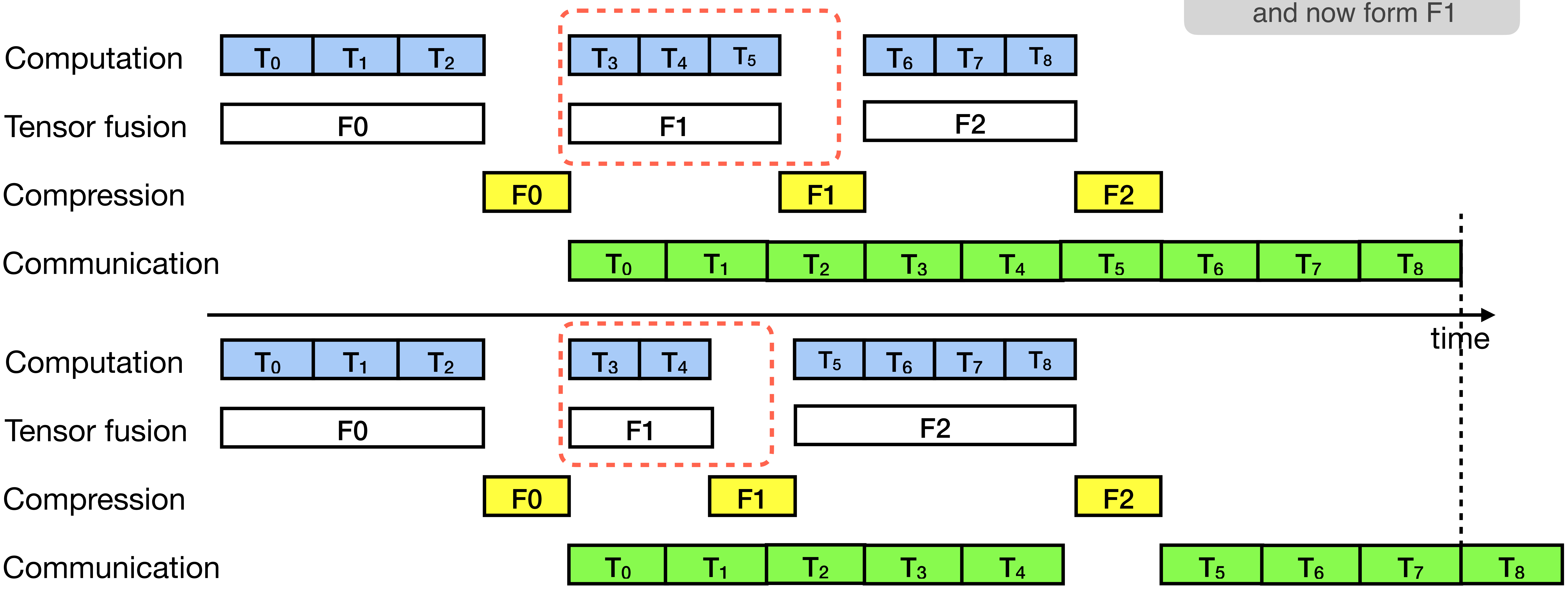


Prune strategies with such F0 that its lower bound is greater than the current optimal

Pruning techniques #2

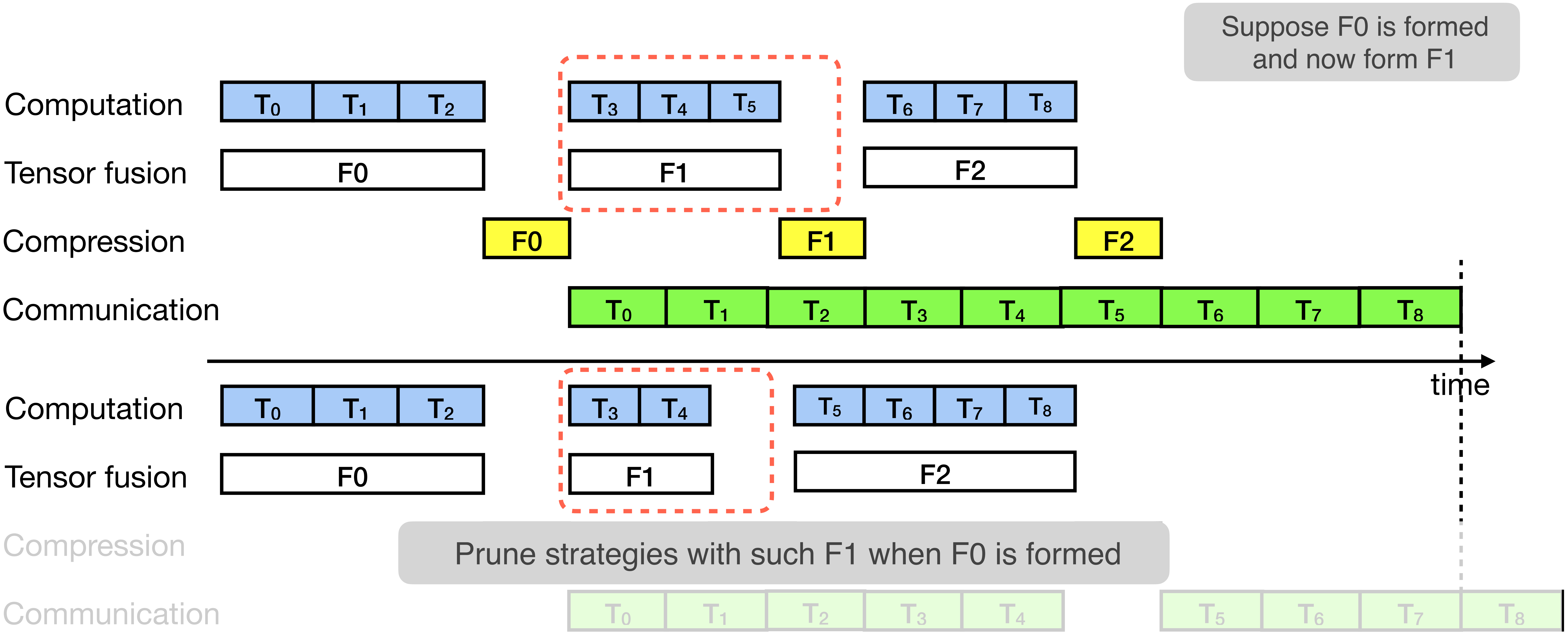
Fuse more tensors based on the communication progress

Suppose F0 is formed and now form F1



Pruning techniques #2

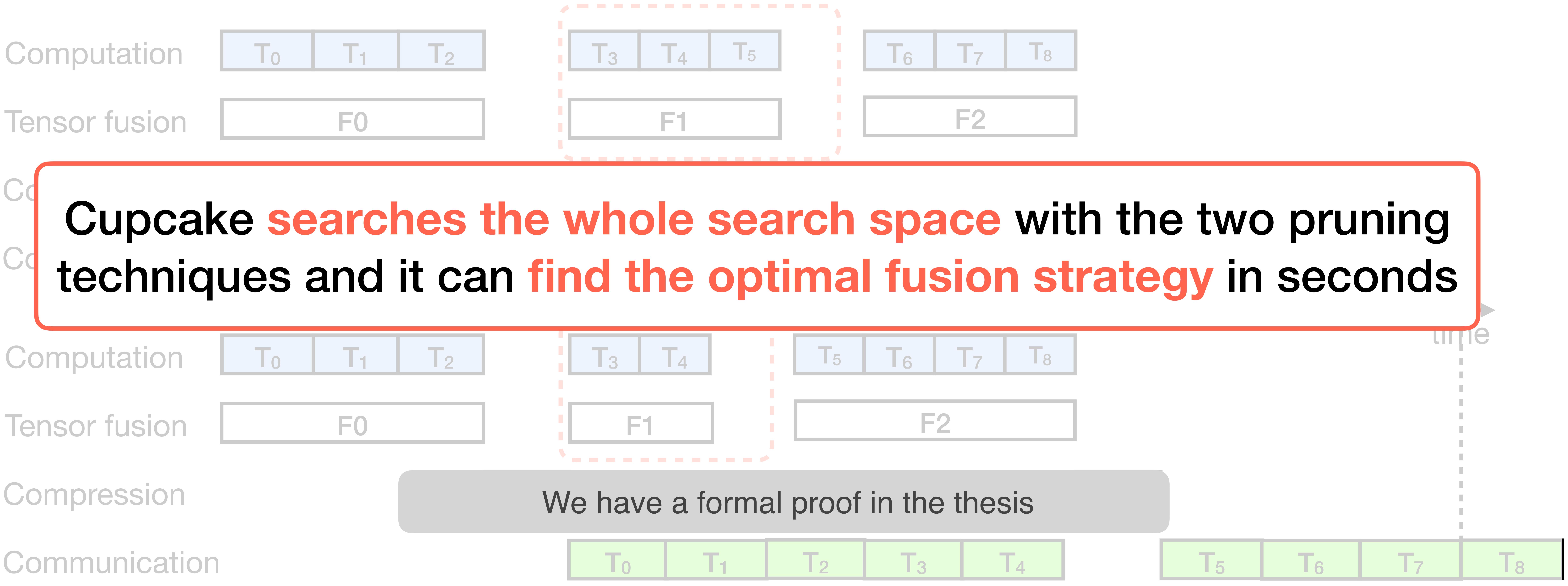
Fuse more tensors based on the communication progress



Pruning techniques #2

Fuse more tensors based on the communication progress

Cupcake **searches the whole search space** with the two pruning techniques and it can **find the optimal fusion strategy** in seconds

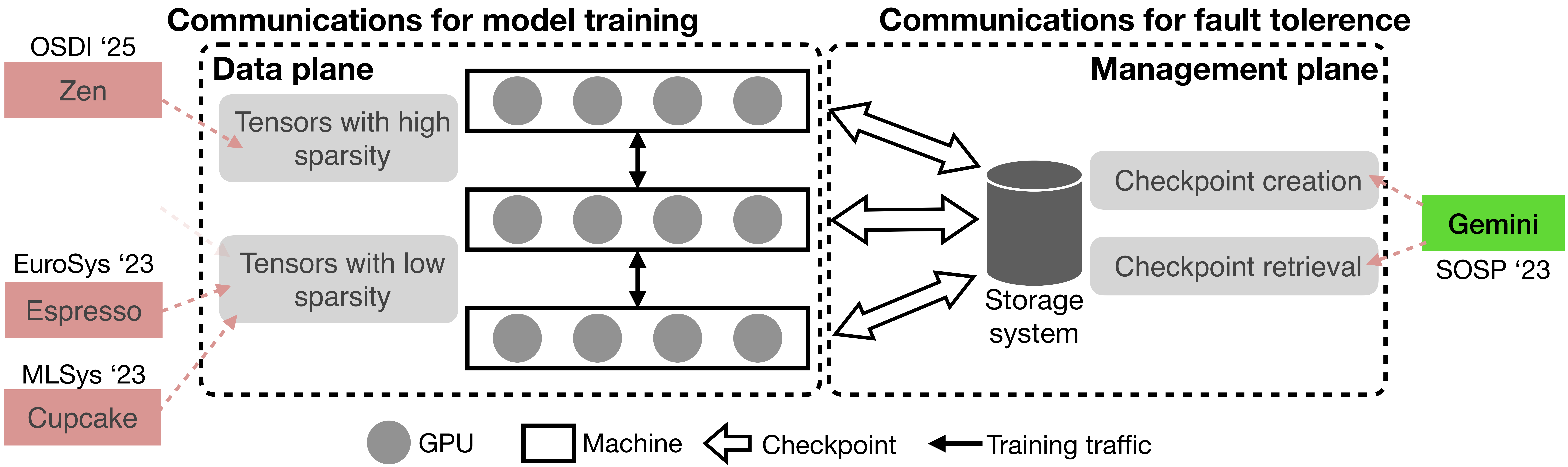


We have a formal proof in the thesis

The next research project

Scaling deep learning by optimizing communications

- Thesis work



Large Language Model (LLM)

Models towards trillion parameters

- Recent LLMs

Model	Parameters	Accelerators	Training time	Developer	Year
Turing-NLG	17.2B	256 V100	—	Microsoft	2020
GPT-3	175B	—	—	OpenAI	2020
OPT-175B	175B	992 A100	2 months	Meta	2021
Gopher	280B	4096 TPU v3	1.3 months	Google	2021
MT-NLG	530B	4480 A100	3 months	Microsoft & NVIDIA	2022
PaLM	540B	6144 TPU v4	2 months	Google	2022
GPT-4	1.76T	—	4-7 months	OpenAI	2023

Larger training models

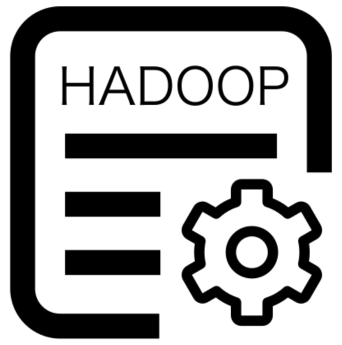
More GPUs involved

Longer training time

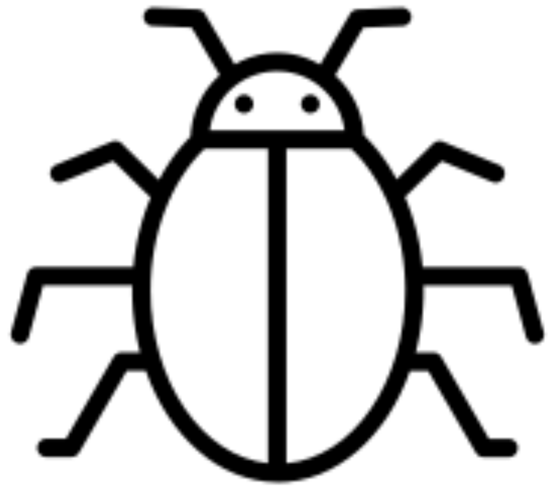
Failures are frequent

- Software failures

Library failures

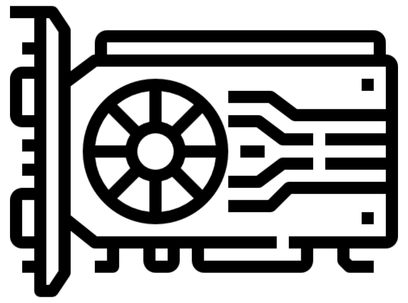


Remote storage failures

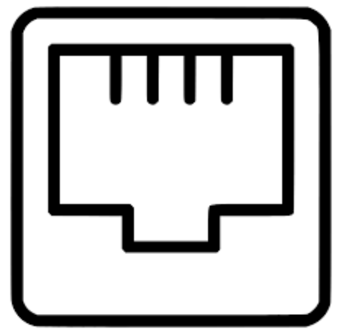


- OPT-175B: 100+ failures[1] in two months

- Hardware failures



GPU failures



Link failures

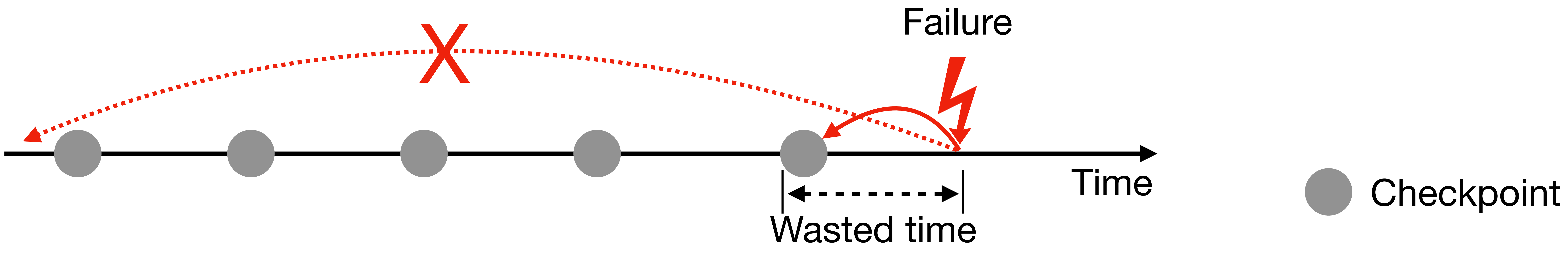
Switch failures

[1] Opt: Open pre-trained transformer language models, arXiv ’22

Checkpoint for failure recovery

- How checkpoint works?

Periodically checkpoint the model states → Resume from the last checkpoint → Redo the computation

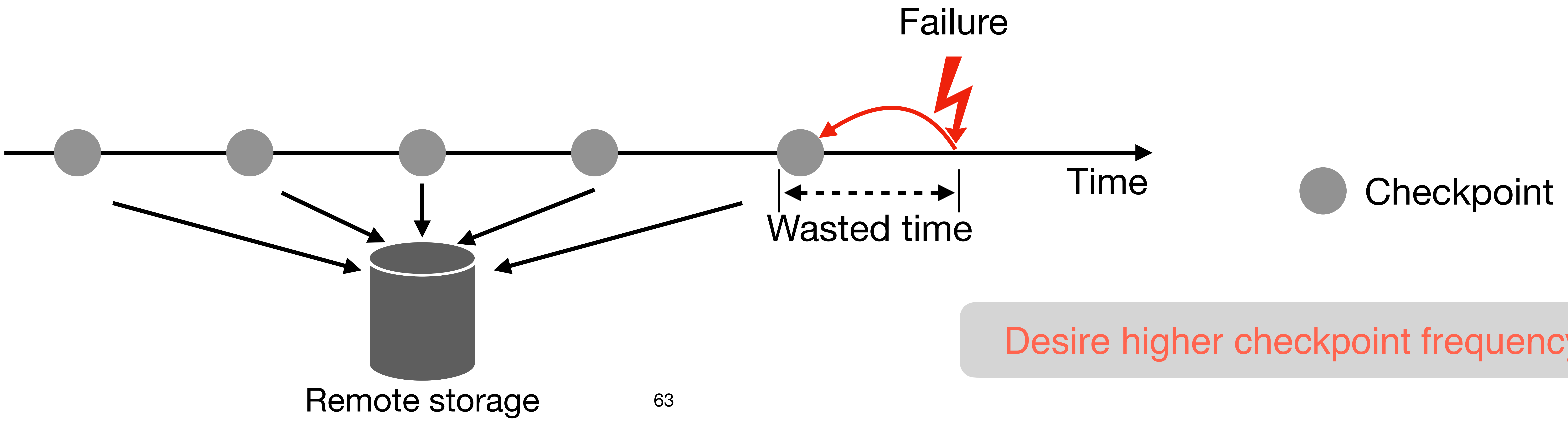


Desire higher checkpoint frequency

Checkpoint for failure recovery

- How checkpoint works?

Periodically checkpoint the model states → Resume from the last checkpoint → Redo the computation



Desire higher checkpoint frequency

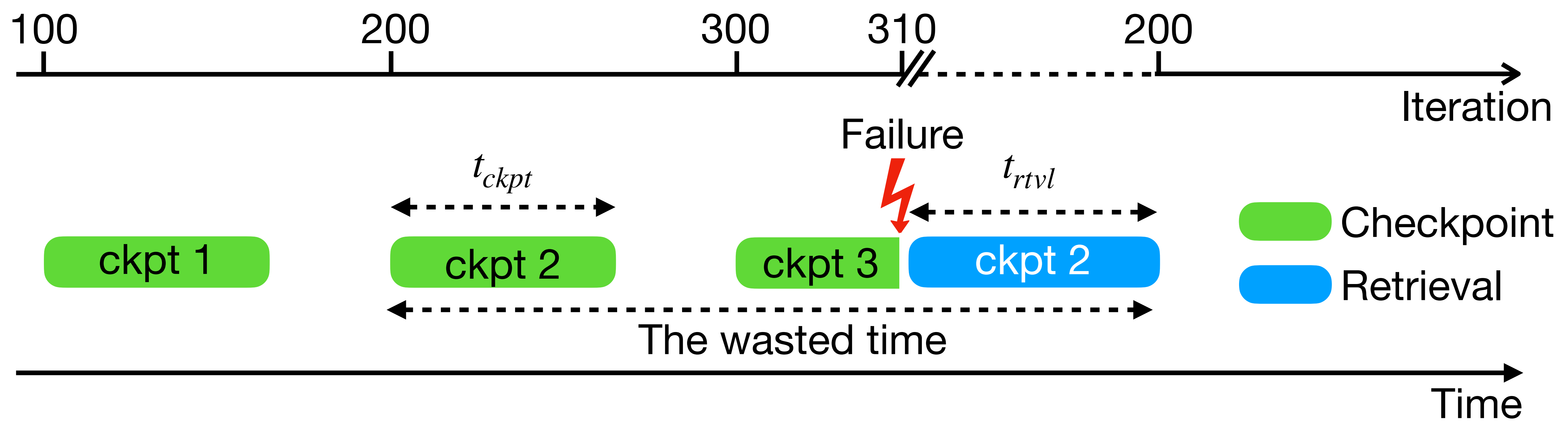
Checkpoint in LLM

Limited checkpoint frequency

- Checkpoint to remote storage takes a long time

Model	Parameters	Checkpoint size	Checkpoint time (20Gbps)
Gopher [56]	280B	3.4 TB	23 min
MT-NLG [62]	530B	6.4 TB	43 min
PaLM [23]	540B	6.5 TB	44 min

- Checkpoint frequency is limited by the checkpoint time



Checkpoint in LLM

Prohibitive failure recovery overhead

- Costly wasted time
  - Even with the highest checkpoint frequency

Model	Parameters	Checkpoint size	Checkpoint time (20Gbps)	Average wasted time
Gopher [56]	280B	3.4 TB	23 min	57 min
MT-NLG [62]	530B	6.4 TB	43 min	108 min
PaLM [23]	540B	6.5 TB	44 min	110 min

- Significant GPU resources are wasted due to failure recovery
  - Thousands of GPUs involved
  - Hundreds of failures during training

Contributions

- We propose the first system that **uses CPU memory for checkpointing** to enable fast failure recovery
    - No assumptions on the underlying parallelism strategy
- We design a **provably optimal checkpoint placement** strategy on CPU memory
- We design **a traffic scheduling algorithm** that orchestrates training and checkpoint traffic to **eliminate the interference** on training throughput
- Gemini is being deployed at AWS to provide fault tolerance to LLM training

Gemini

Checkpoint to CPU memory

- CPU memory is much larger than GPU memory

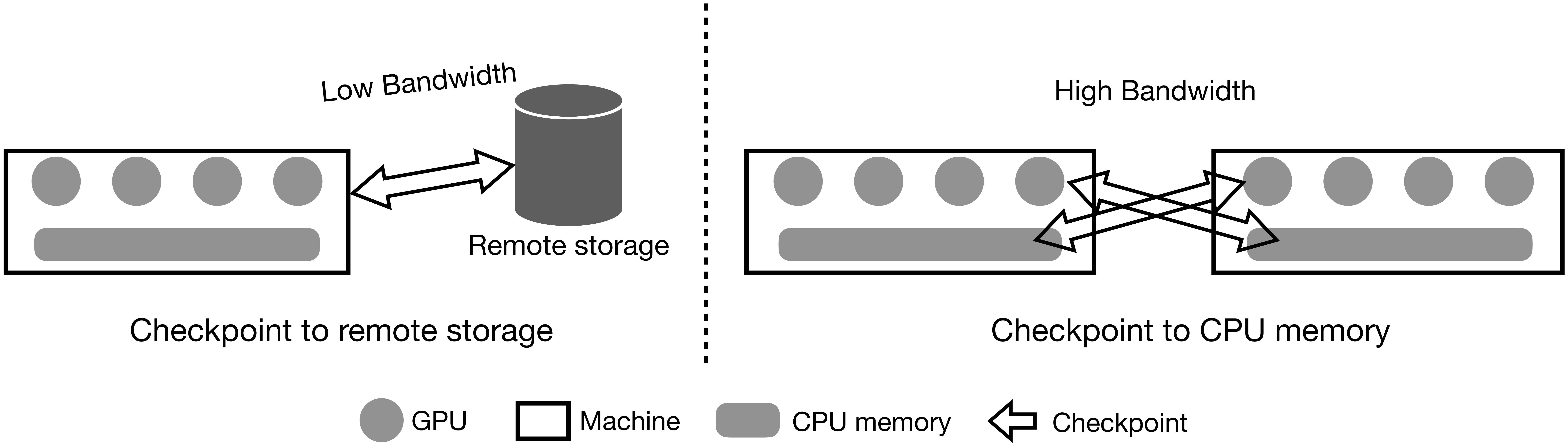
Instance type	Cloud	GPU	GPU memory	CPU memory
p3dn.24xlarge [14]	AWS	8 V100	8 × 32 GB	768 GB
p4d.24xlarge [15]	AWS	8 A100	8 × 40 GB	1152 GB
ND40rs_v2 [10]	Azure	8 V100	8 × 32 GB	672 GB
ND96asr_v4 [11]	Azure	8 A100	8 × 40 GB	900 GB
n1-8-v100 [9]	GCP	8 V100	8 × 32 GB	624 GB
a2-highgpu-8g [9]	GCP	8 A100	8 × 40 GB	640 GB
DGX A100 [12]	NVIDIA	8 A100	8 × 80 GB	2 TB

CPU memory size is sufficient to store checkpoints

Gemini

Checkpoint to CPU memory

- CPU memory is much larger than GPU memory
- Checkpoint to CPU memory enables a much higher frequency

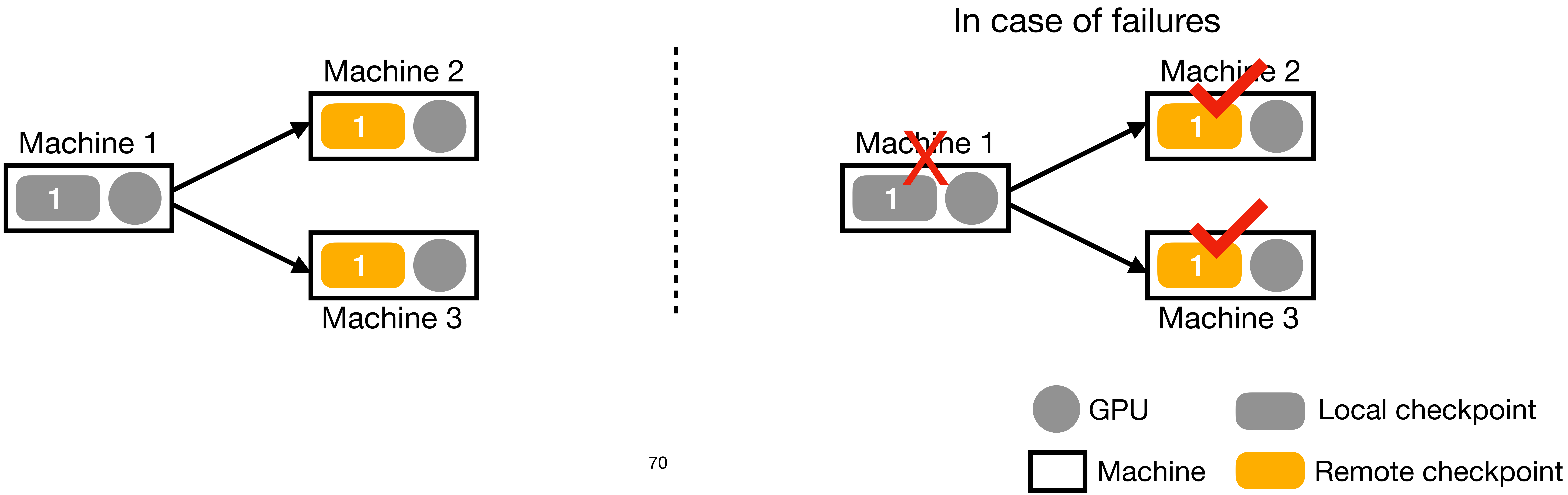


Challenge #1

- Data stored in CPU memory can get lost

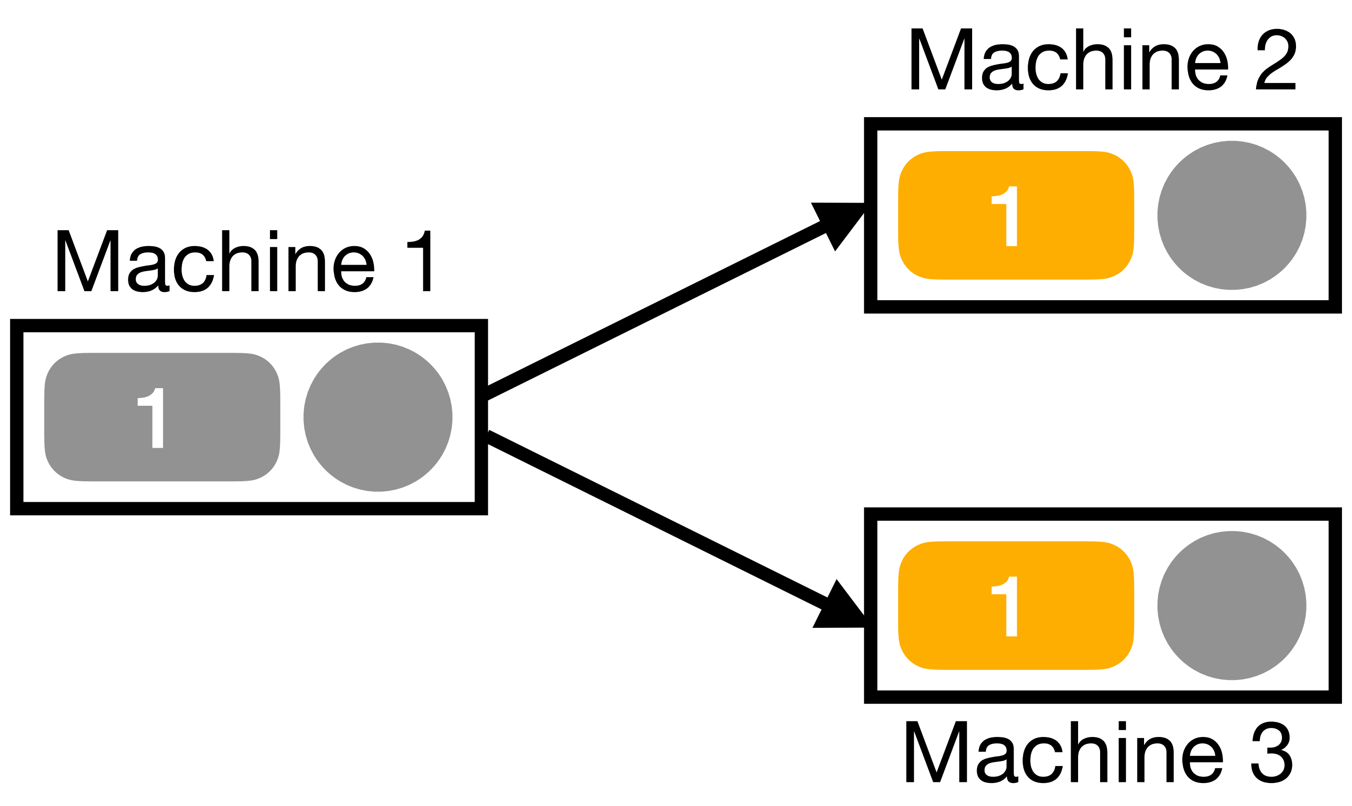
Challenge #1 and solution

- Data stored in CPU memory can get lost
- Solution: checkpoint redundancy
- Design choice: checkpoint replicas

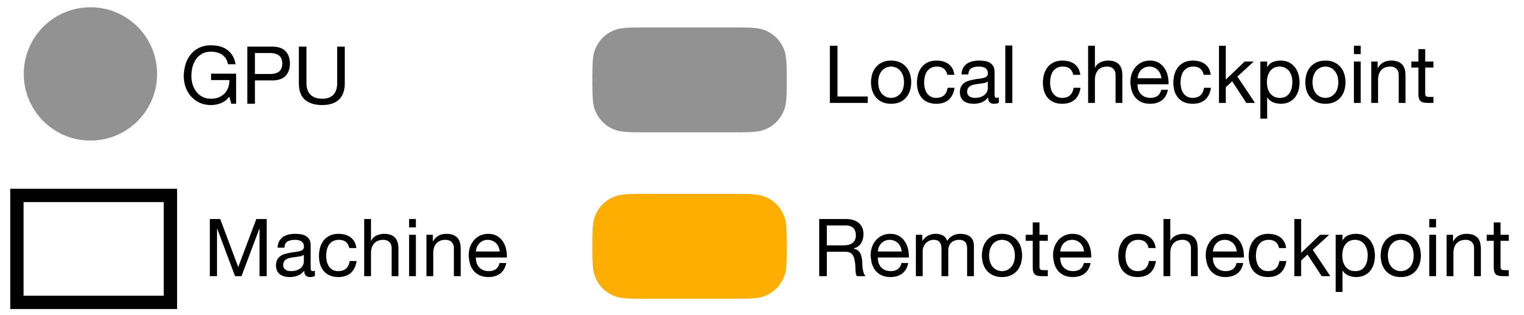


Challenge #1 and solution

- Data stored in CPU memory can get lost
- Solution: checkpoint redundancy
- Design choice: checkpoint replicas

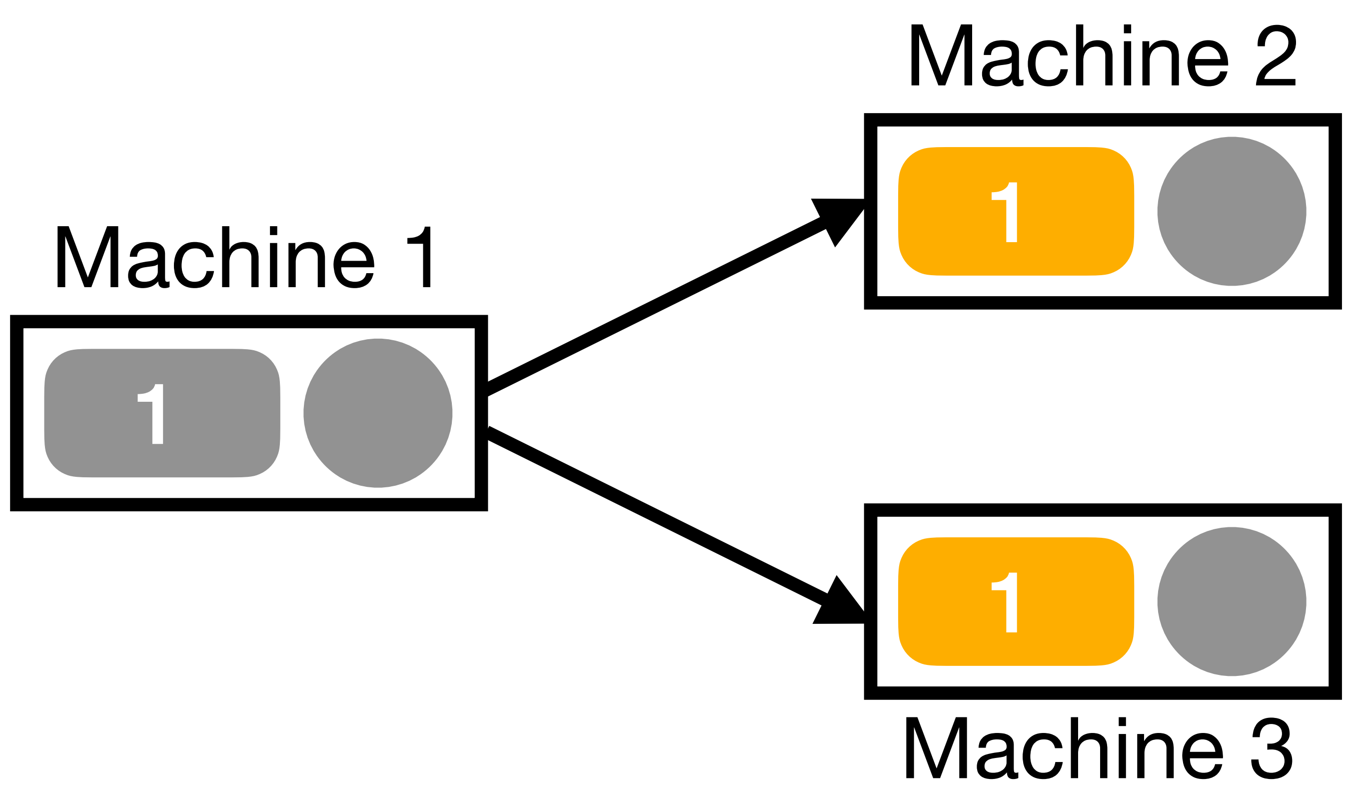


- Why not Erasure Coding?
  - Prohibitive computation cost
  - CPU memory is not a bottleneck

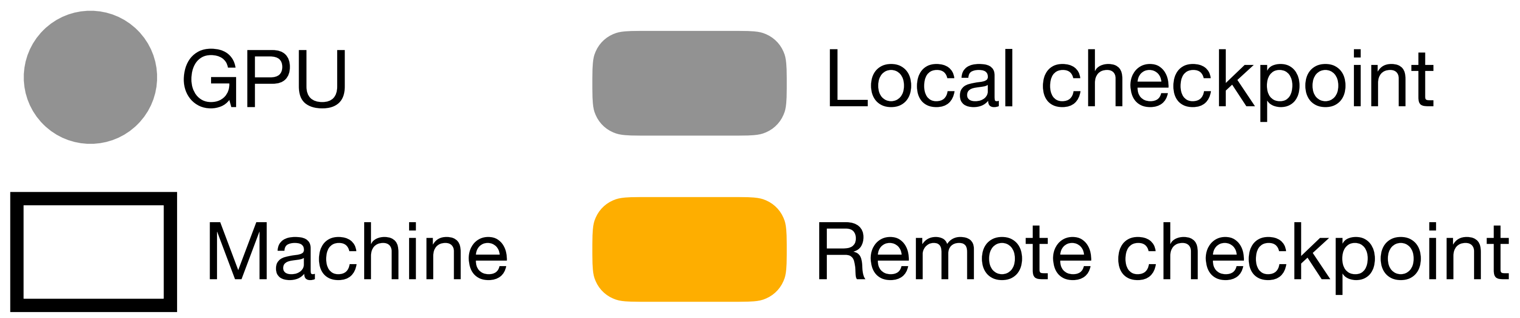


Challenge #1 and solution

- Data stored in CPU memory can get lost
- Solution: checkpoint redundancy
- Design choice: checkpoint replicas



- What is the optimal checkpoint placement?



Goal

Checkpoint replicas

- Data stored in CPU memory can get lost
- Solution: checkpoint redundancy

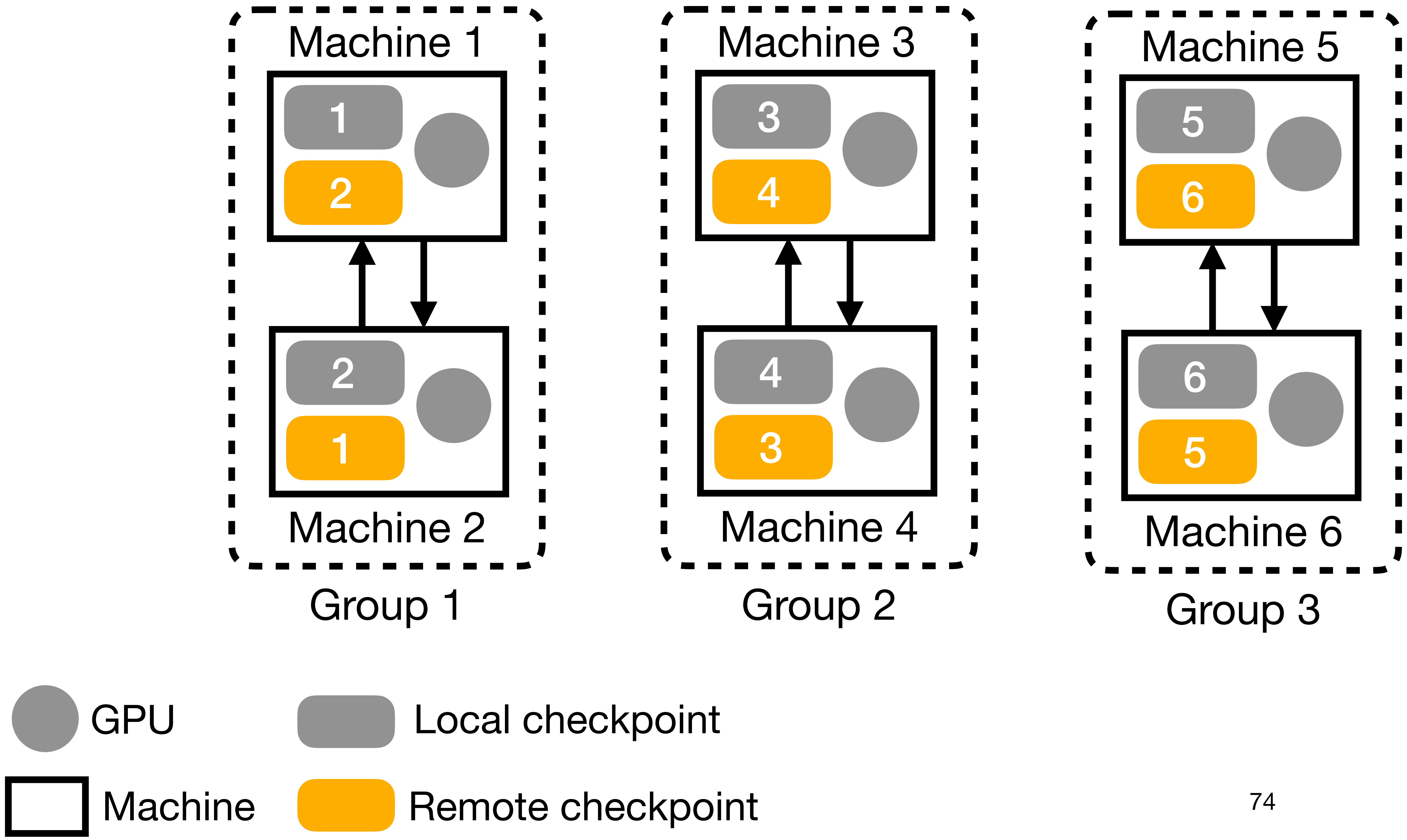
Maximize the probability of failure recovery from checkpoints stored in CPU memory

- What is the optimal checkpoint placement?

Solution

Group placement strategy

- An example with two replicas



1. Given m replicas, all machines are divided into disjoint groups and each group has m machines
2. Each machine backups a checkpoint replica for all machines within the same group

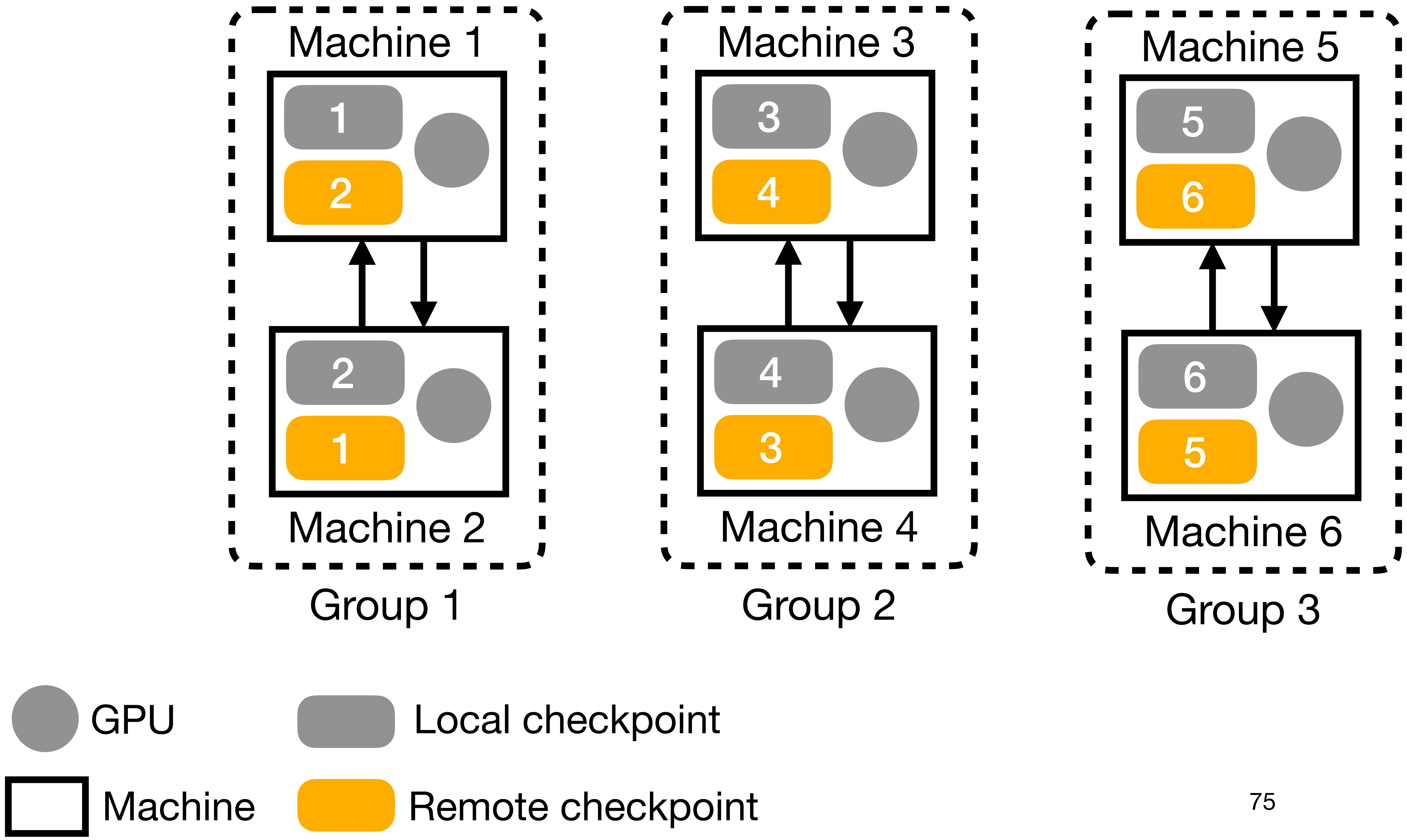
Solution

Group placement strategy

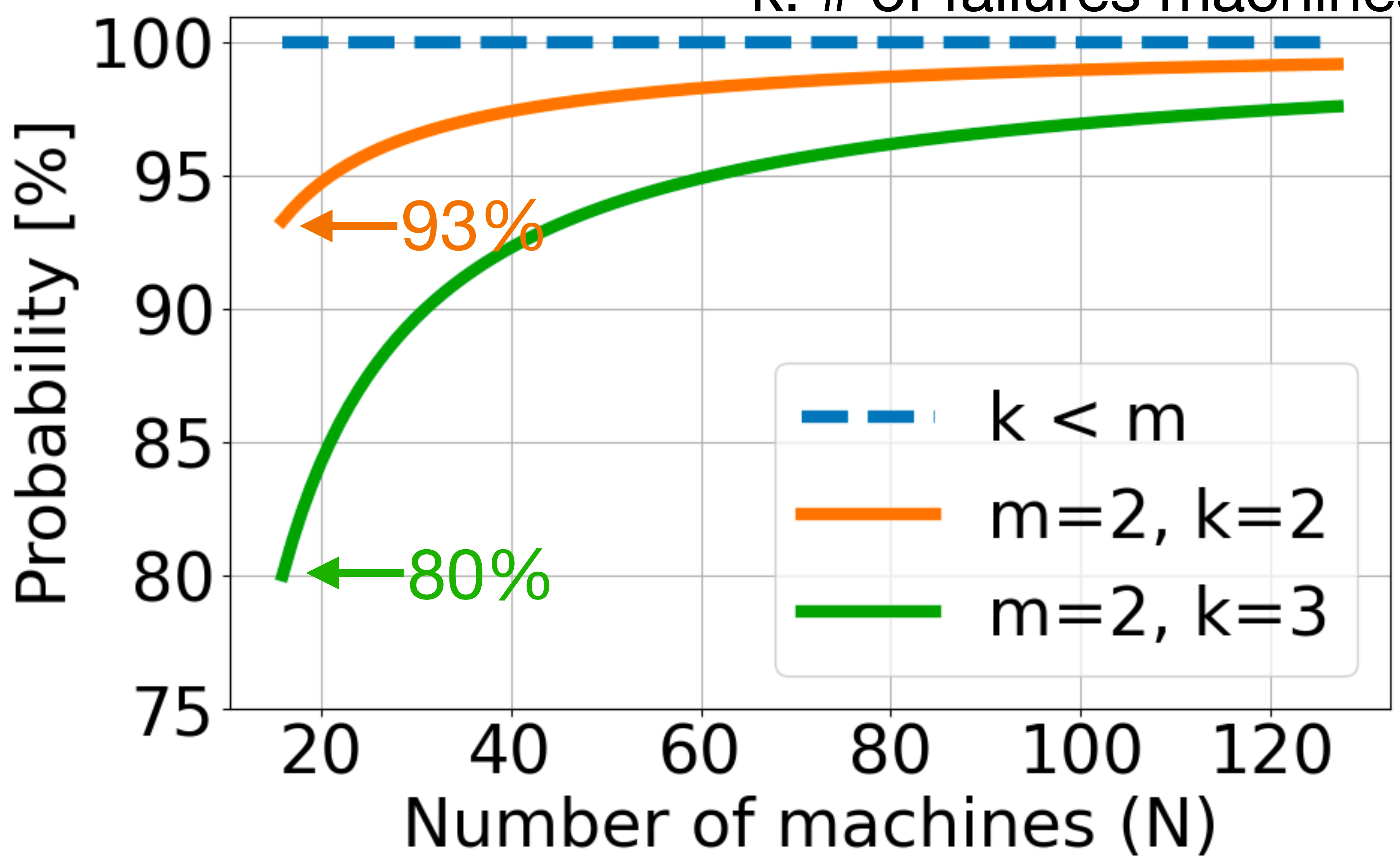
A formal proof in the thesis

Group placement strategy is provably optimal

- An example with two replicas



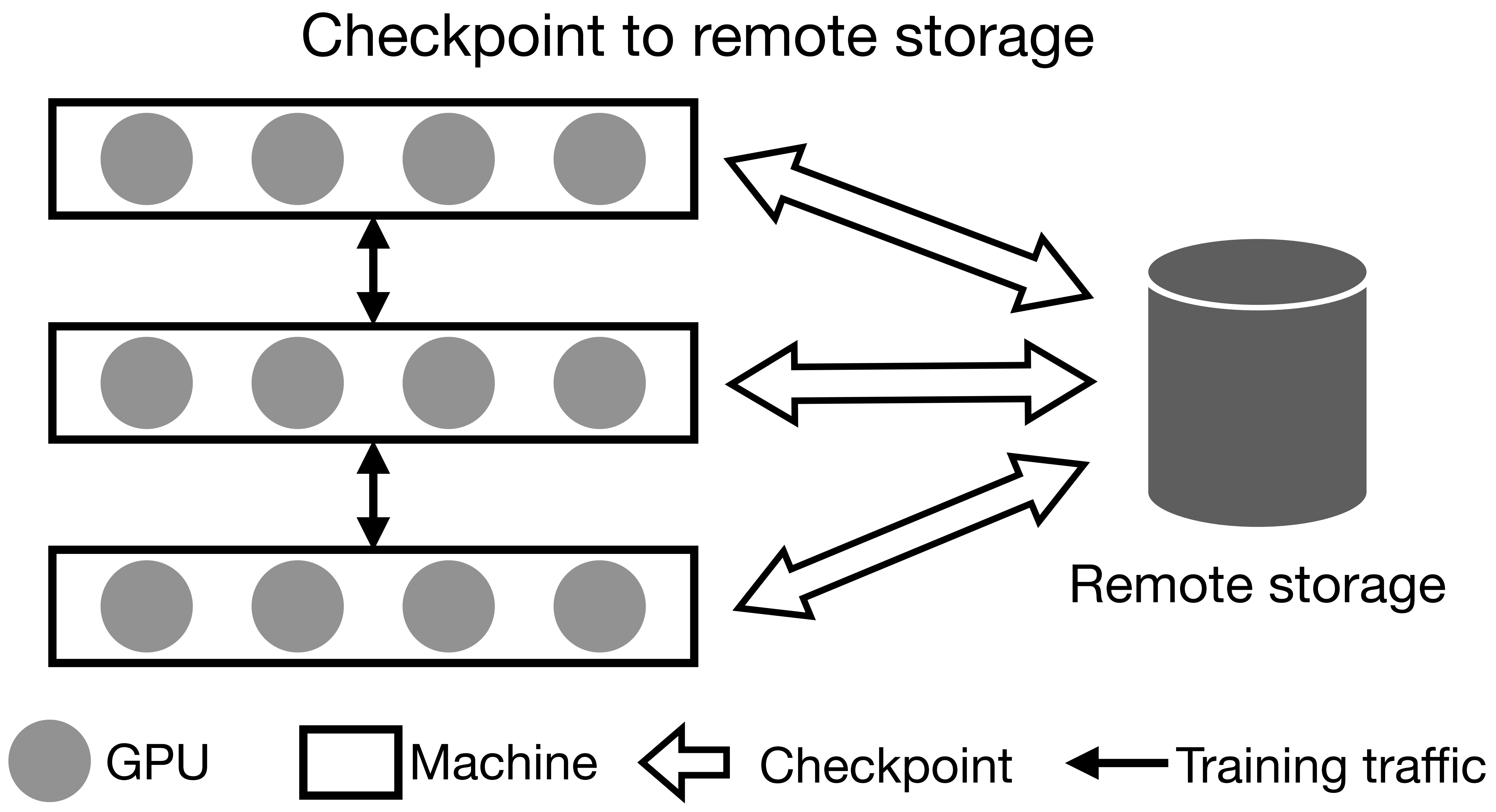
m: # of checkpoint replicas
k: # of failures machines



Two checkpoint replicas can already handle most cases!

Challenge #2

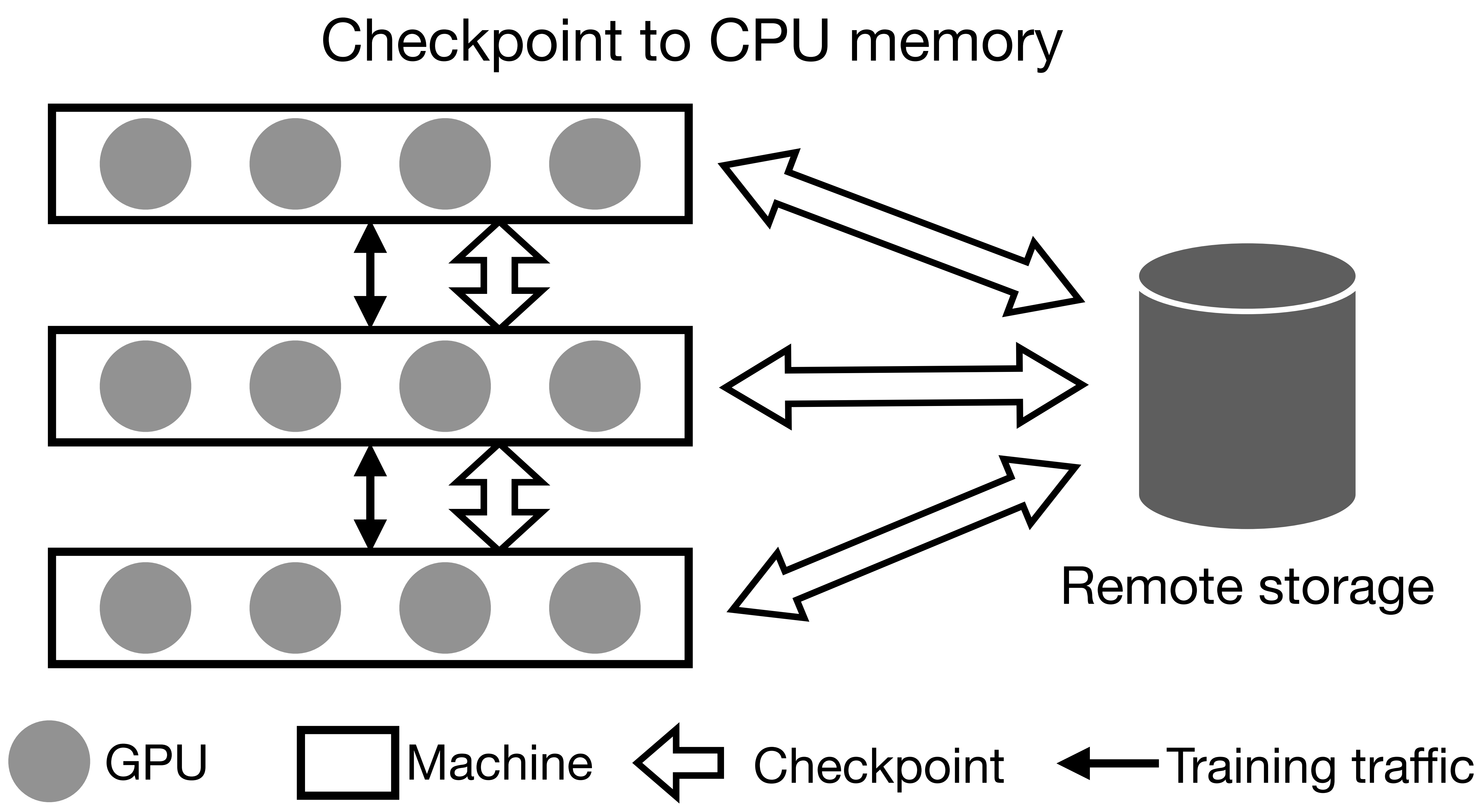
- Checkpoint traffic interferes with training traffic



Checkpoint traffic and training traffic have different networks

Challenge #2

- Checkpoint traffic interferes with training traffic

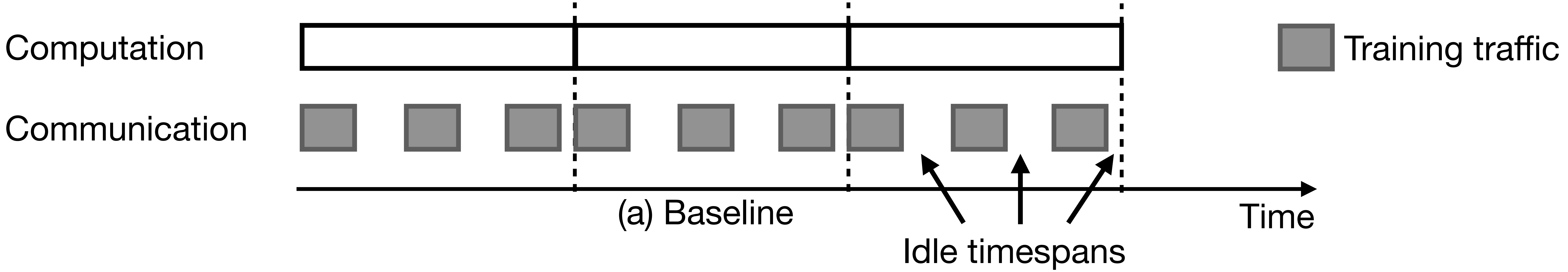


Checkpoint traffic and training traffic
shares the same network

Solution

Traffic interleaving

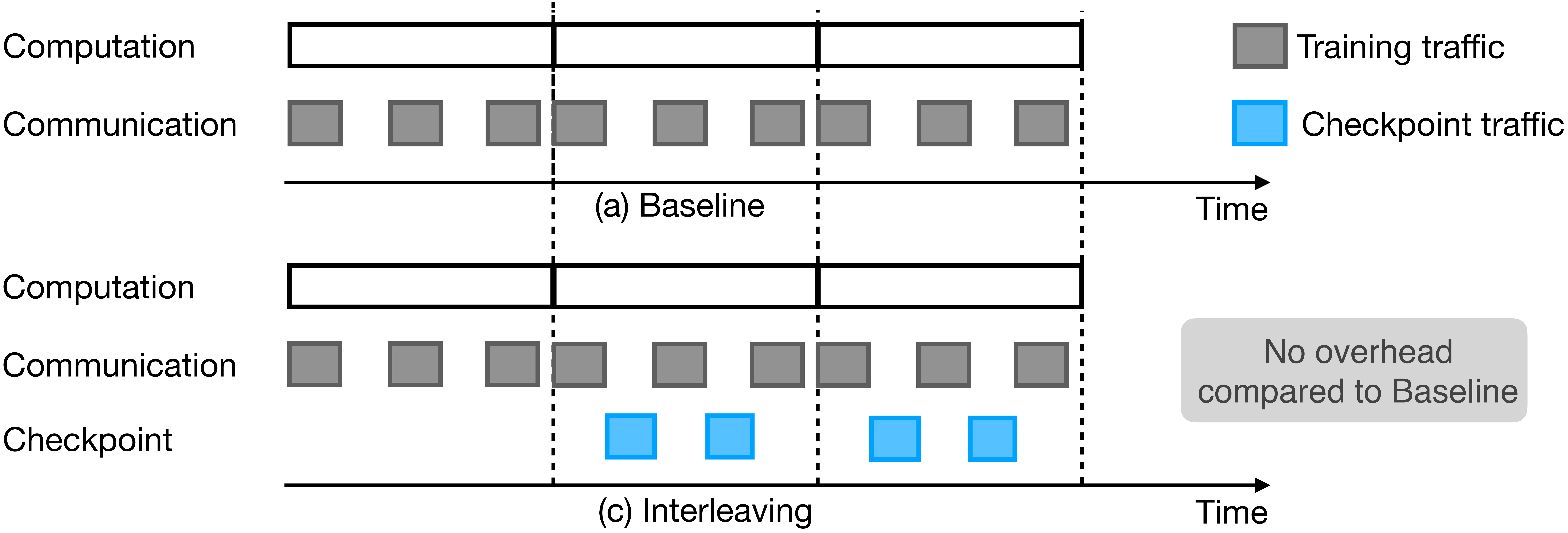
- Observation: Idle timespans in the network



Solution

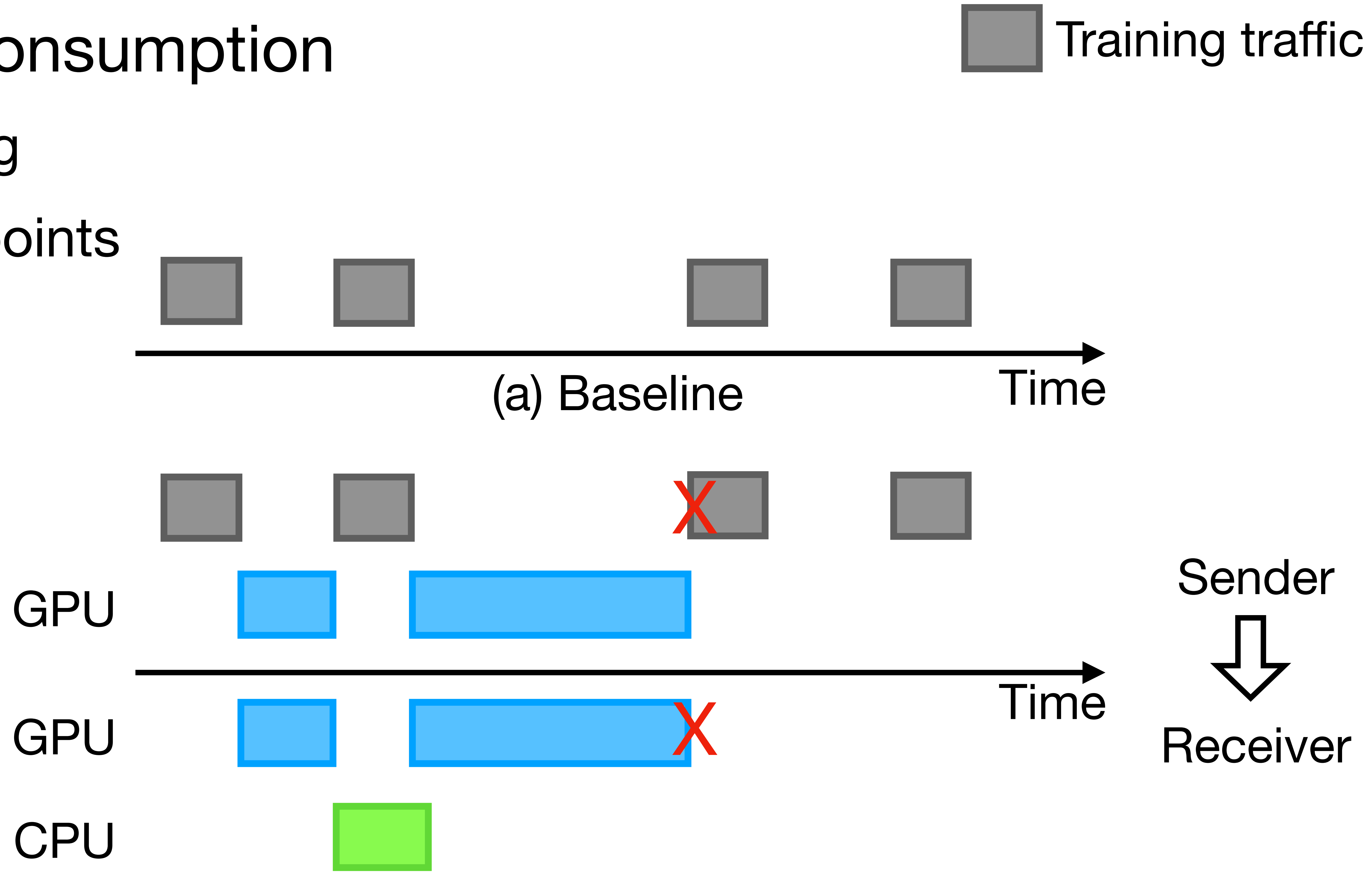
Traffic interleaving

- Insert checkpoint traffic in idle timespans



Out-of-memory issue

- Minimize the extra GPU memory consumption
  - GPU memory is mainly used for training
  - Limited spare GPU memory for checkpoints

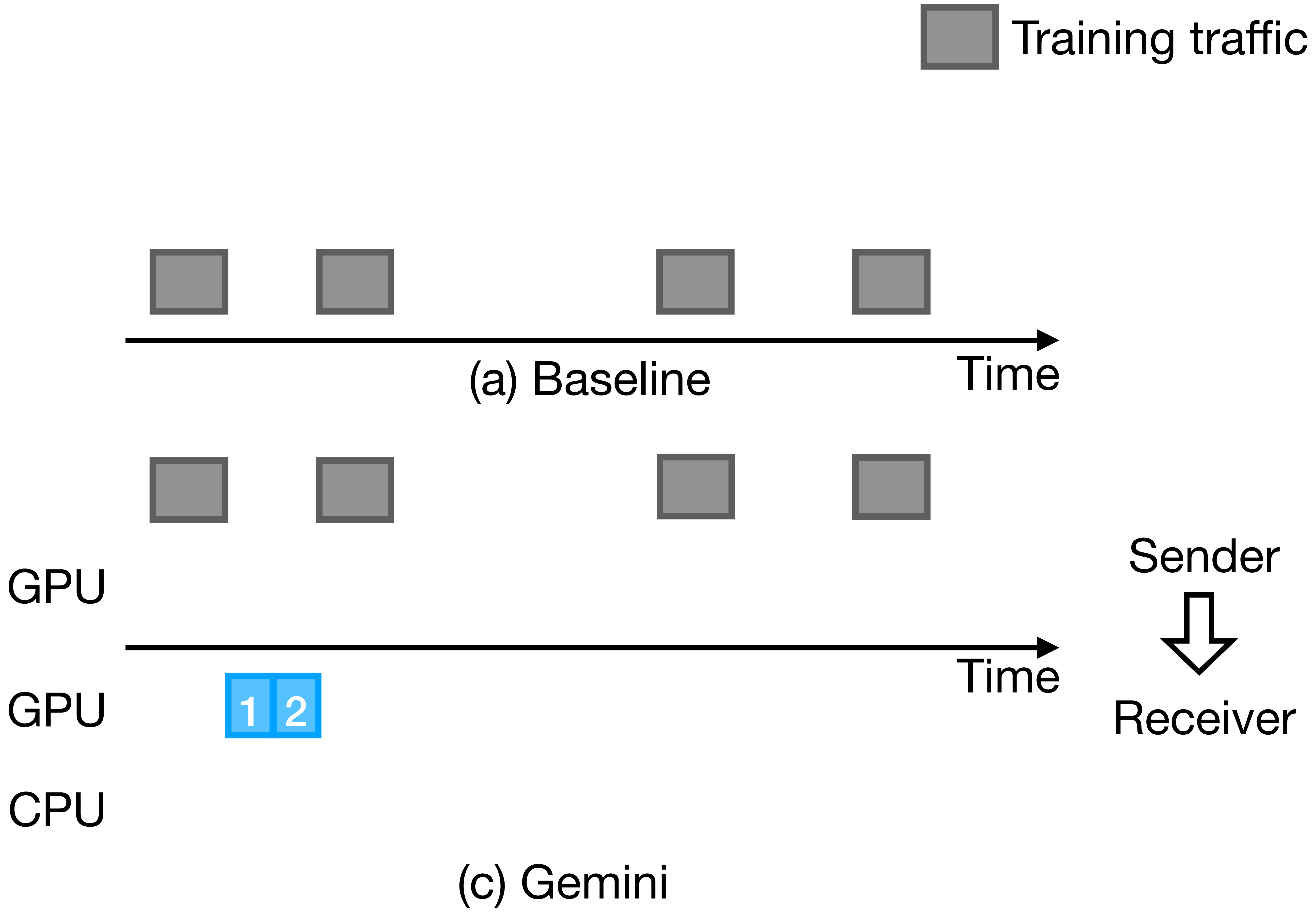


(b) Out of memory

Our design

Address out-of-memory issue

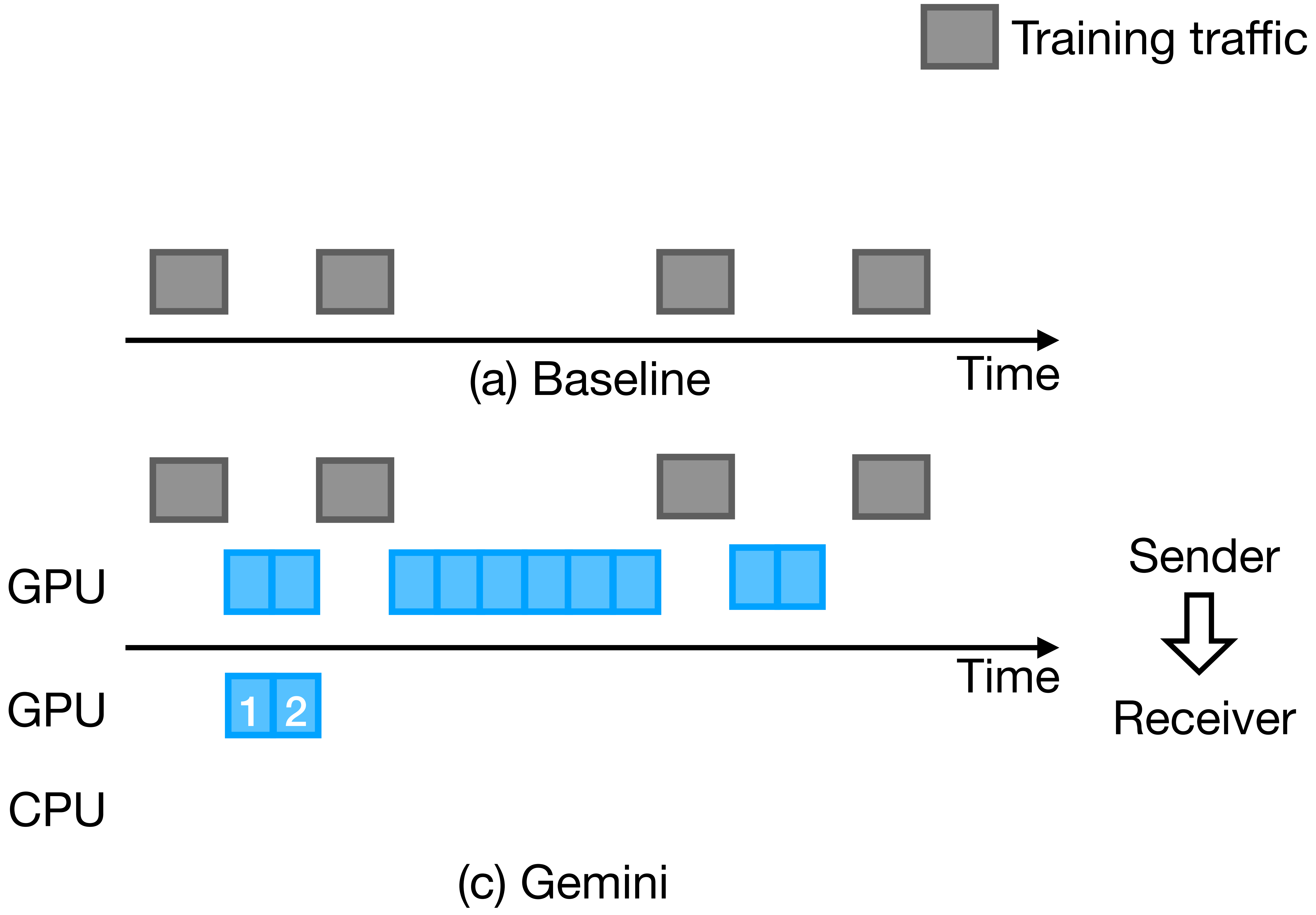
- Checkpoint partition and pipelining
  - Reserve a GPU buffer at the receiver
  - Partition the buffer to multiple parts



Our design

Address out-of-memory issue

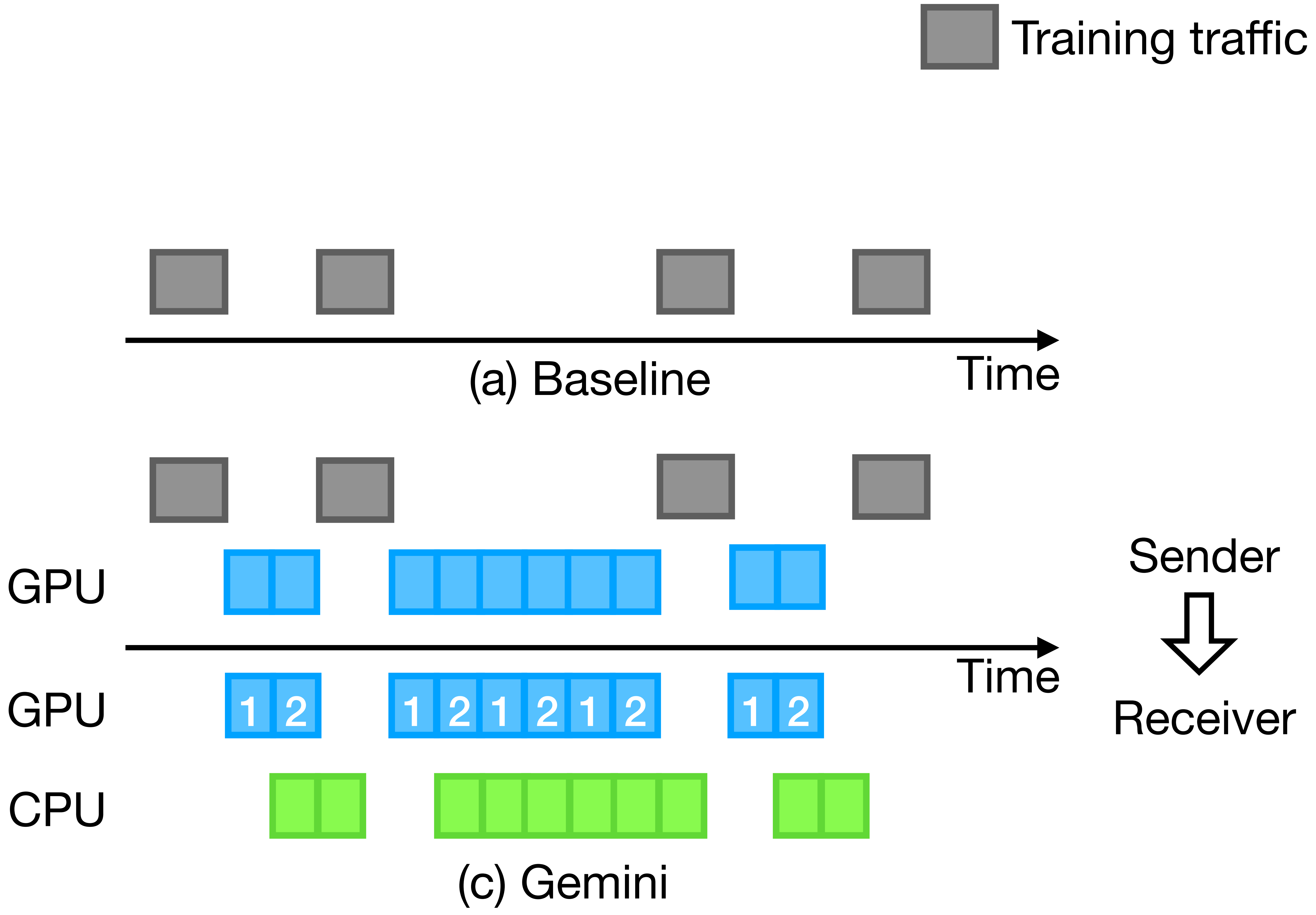
- Checkpoint partition and pipelining
  - Reserve a GPU buffer at the receiver
  - Partition the buffer to multiple parts
  - Pipeline checkpoint communications



Our design

Address out-of-memory issue

- Checkpoint partition and pipelining
  - Reserve a GPU buffer at the receiver
  - Partition the buffer to multiple parts
  - Pipeline checkpoint communications



The GPU buffers are reused

A small GPU buffer, e.g., 128MB, is sufficient

Implementation and evaluations

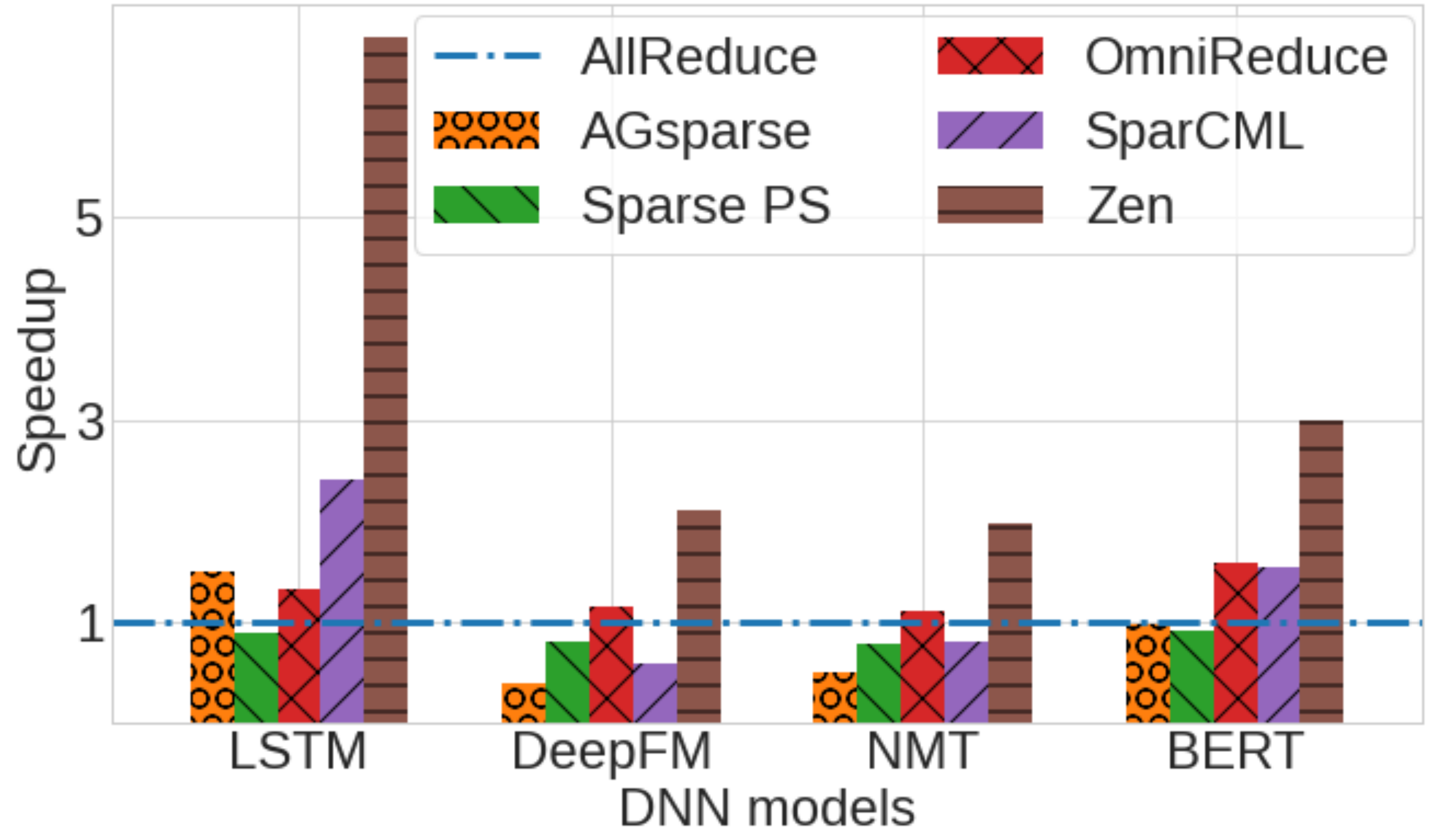
- Zen
  - Built upon Horovod and PyTorch
  - Hierarchical hashing algorithm is implemented in CUDA C (~500 LoC)
- Espresso
  - A compression module in BytePS[1]
  - Partially deployed at ByteDance GPU clusters
- Cupcake, open source[2]
- GEMINI
  - Built upon DeepSpeed
  - Deploying at AWS to support fault tolerance in LLM training

[1] Espresso: https://github.com/bytedance/byteps/tree/Espresso
[2] Cupcake: https://github.com/zhuangwang93/Cupcake

Zen

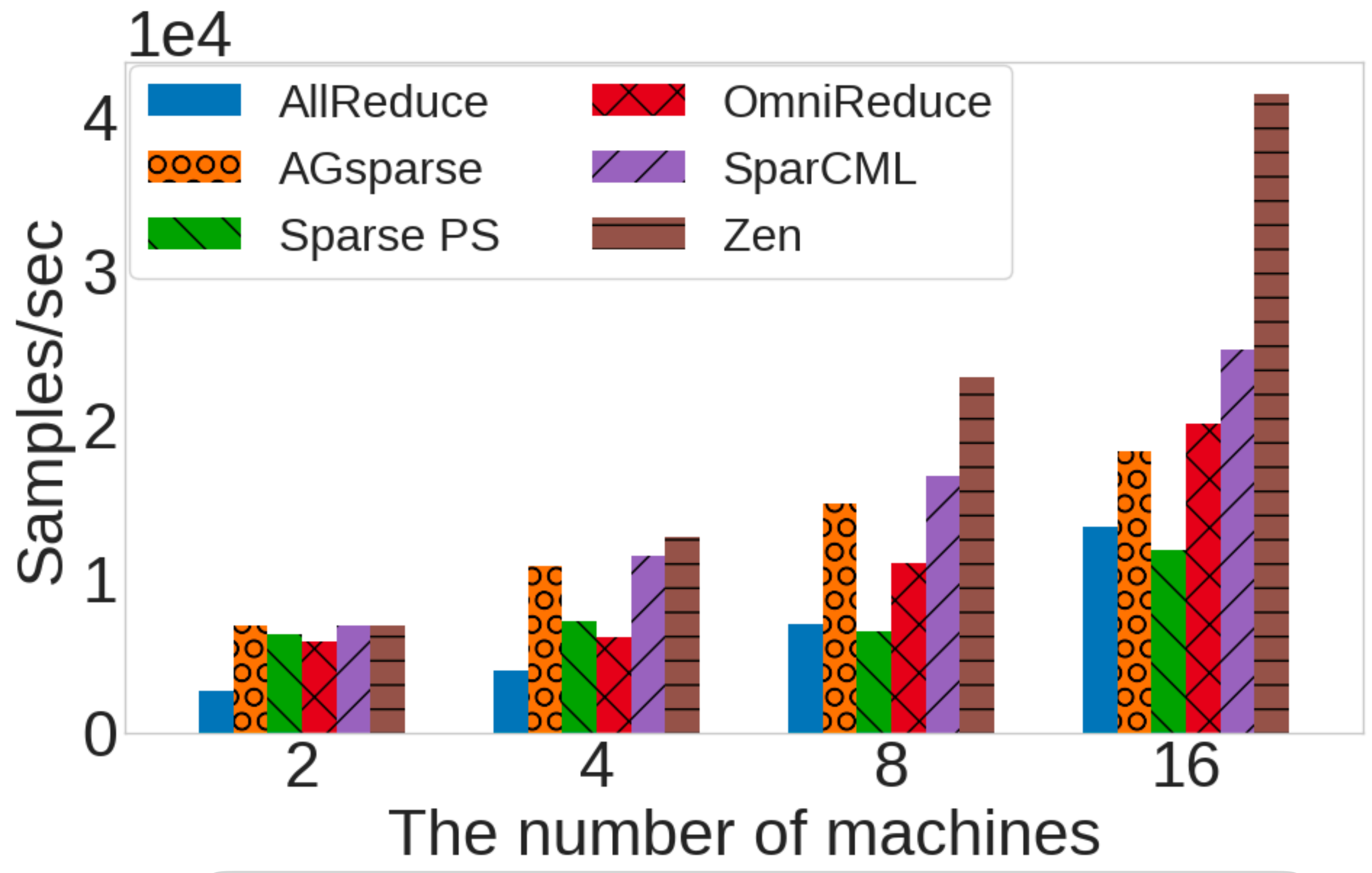
128 V100 GPUs with 25Gbps network

- Communication improvement
  - Speedups are normalized to AllReduce



6.8x communication speedup

- End-to-end efficiency improvement
  - Training throughput

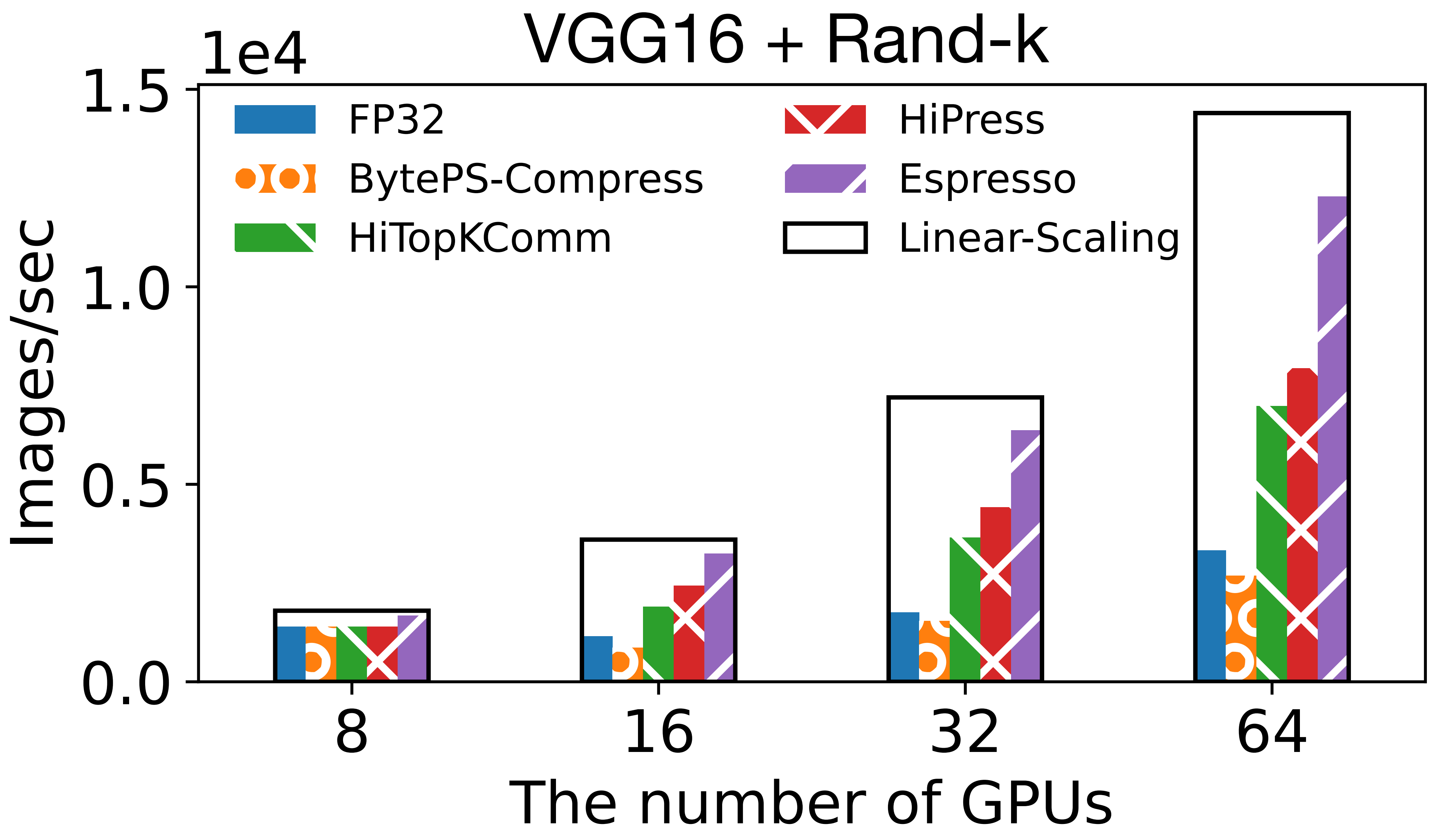


LSTM, 1.7x end-to-end speedup

Espresso and Cupcake

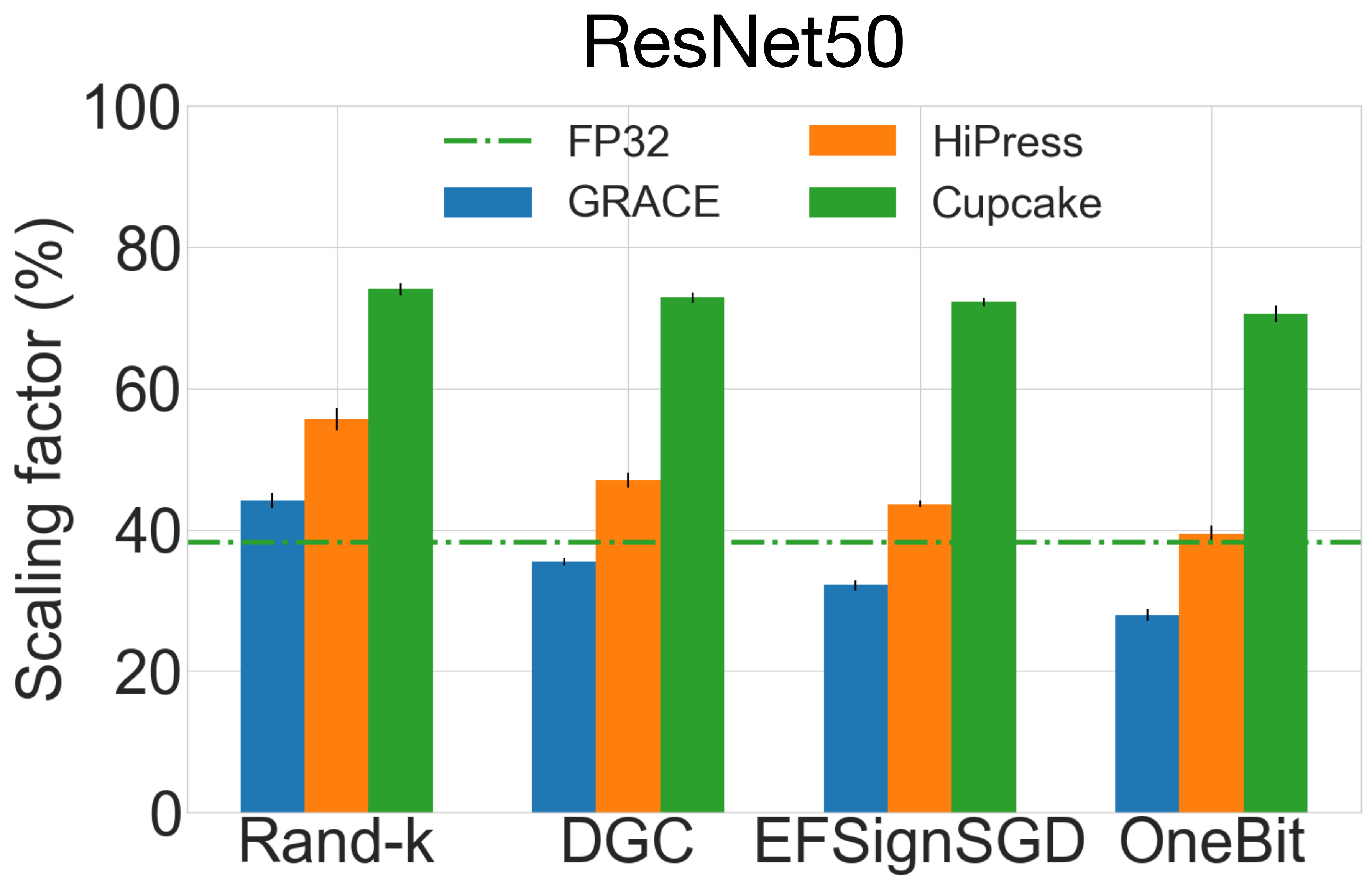
64 V100 GPUs with 25Gbps network

- Espresso
  - Speedups are normalized to AllReduce



1.8x end-to-end speedup

- Cupcake
  - Compared to layer-wise approaches

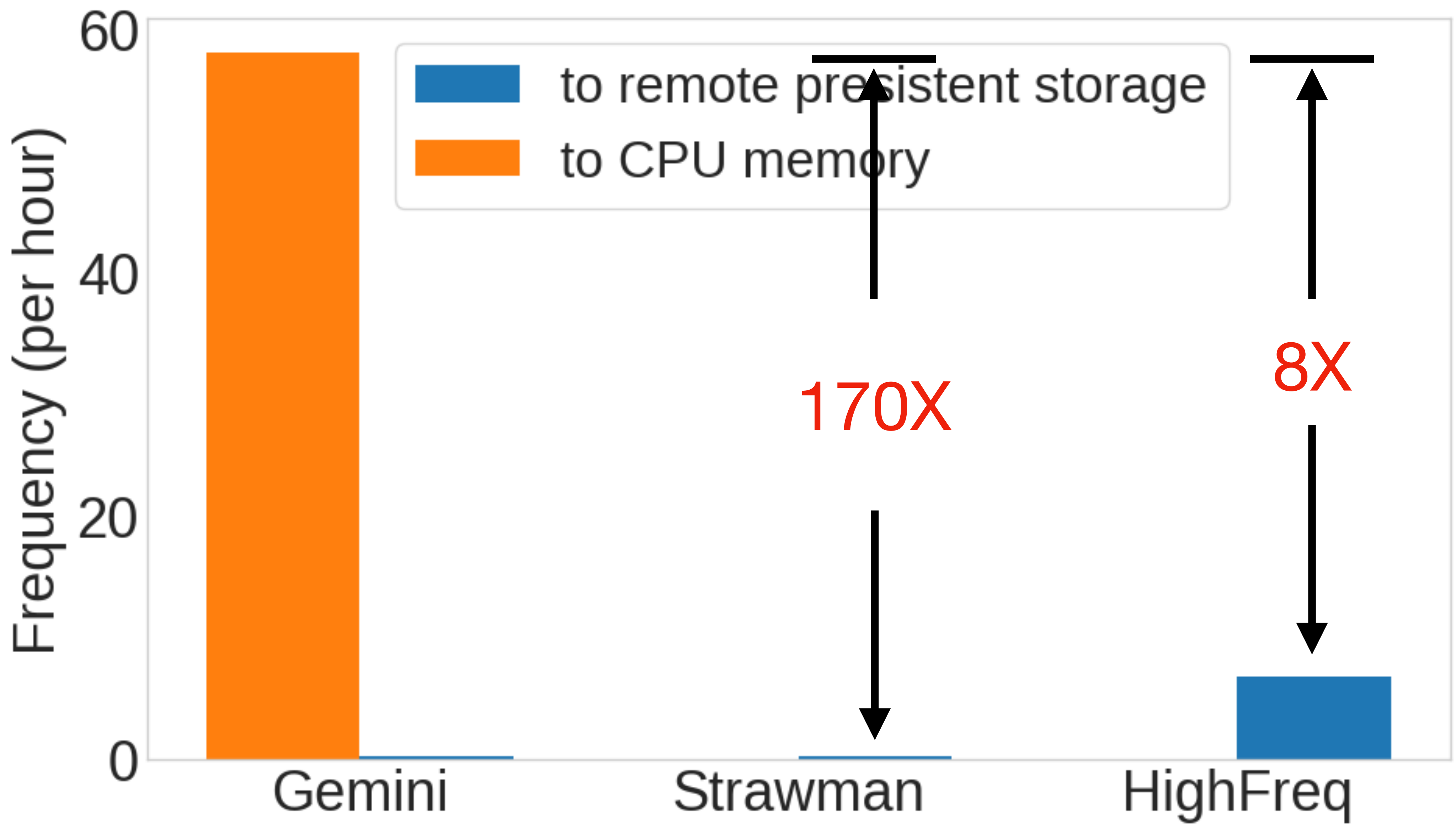


1.7x end-to-end speedup

GEMINI

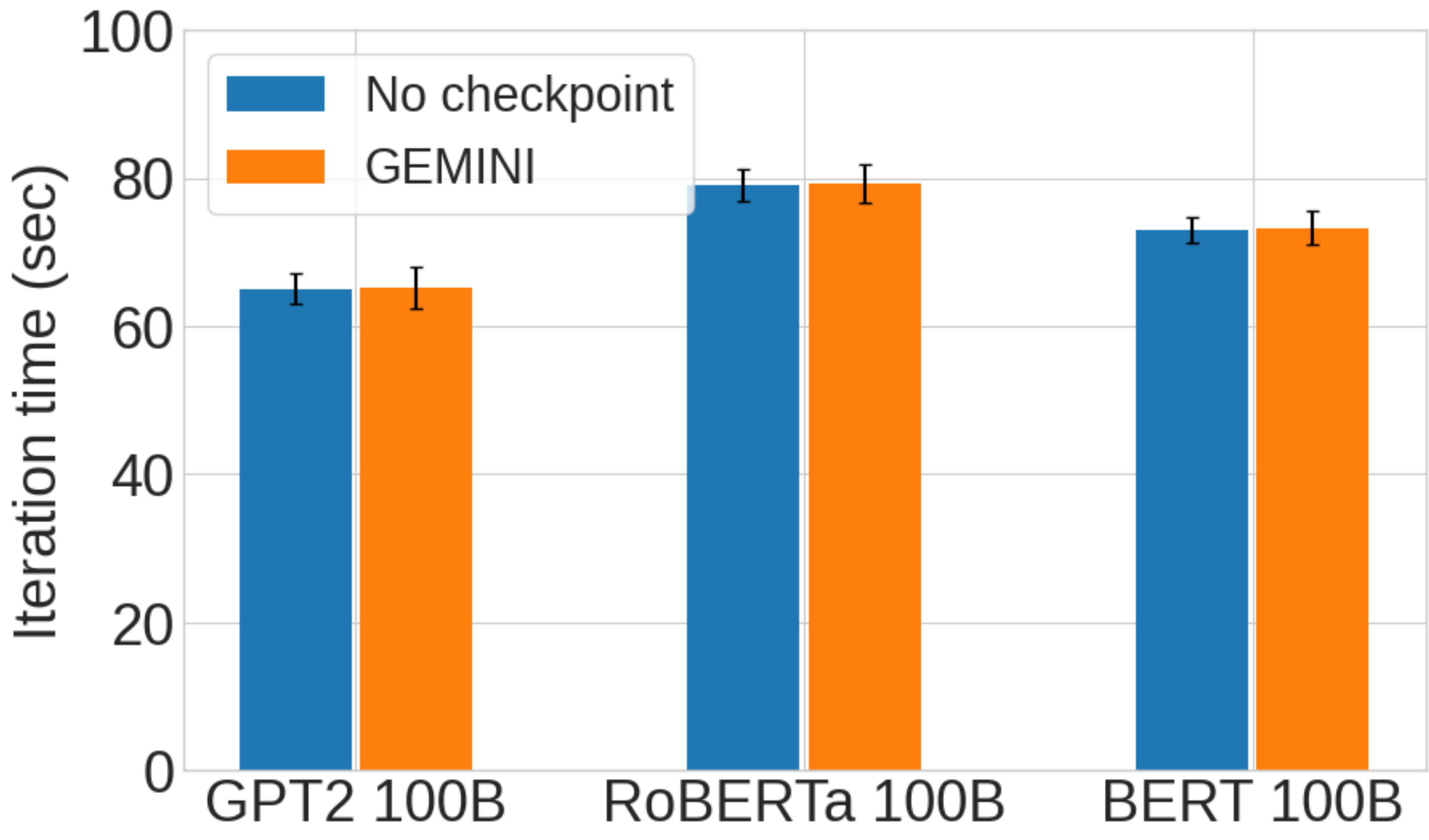
128 A100 GPUs, 100 billion parameters

- Checkpoint frequency



Checkpoint model states every iteration

- Training throughput

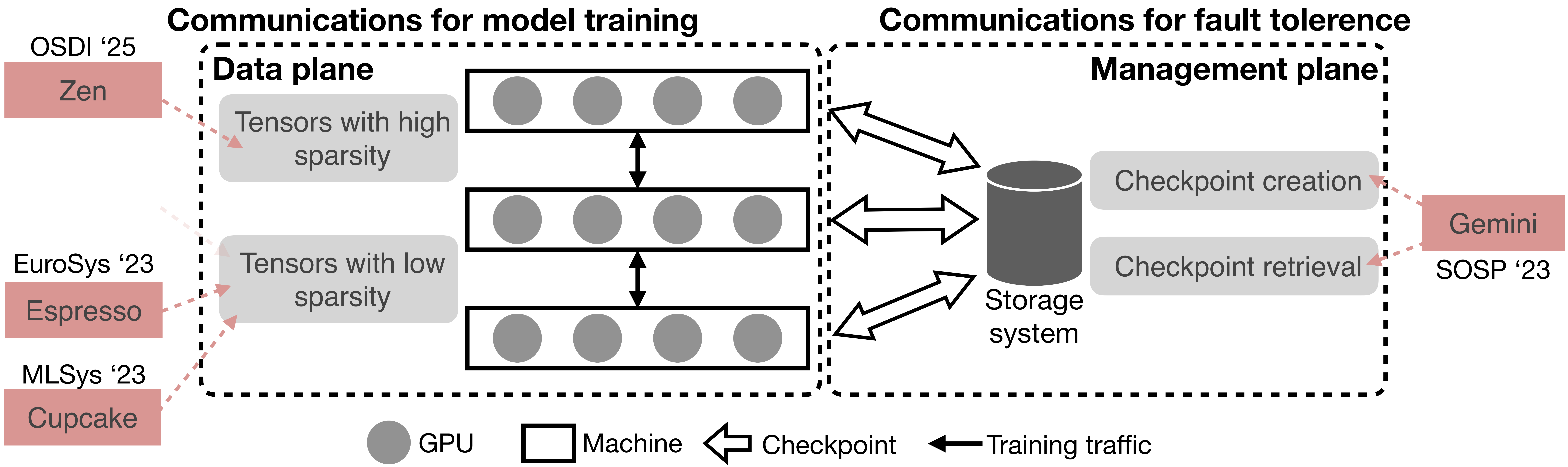


Negligible overhead on iteration time

Research summary

Scaling deep learning by optimizing communications

- Thesis work



Acknowledgement

Prof. Eugene

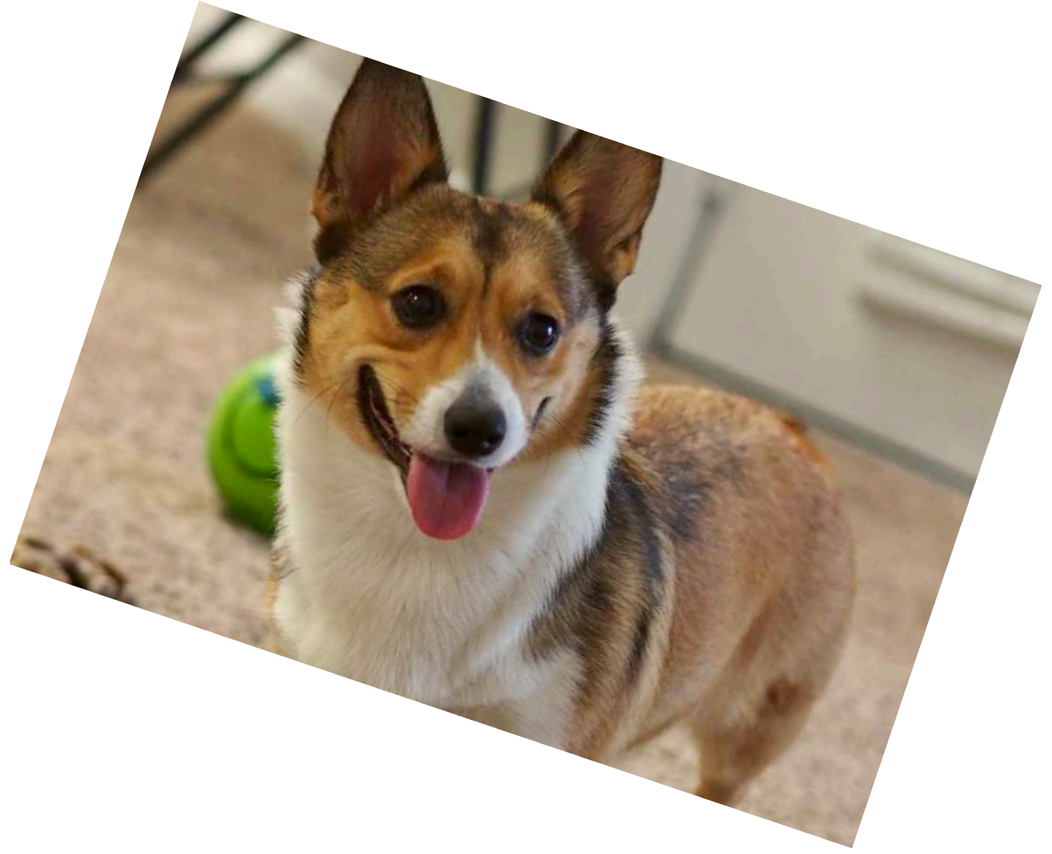
Prof. Edward

Prof. Anshumali

Prof. Santiago

Acknowledgement

Acknowledgement



Research summary

Scaling deep learning by optimizing communications

- Thesis work