Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints

Zhuang Wang, Zhen Jia, Shui Zheng, Zhen Zhang, Xinwei Fu,
T. S. Eugene Ng, Yida Wang

Large models

Characteristics

- Recent large language models (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

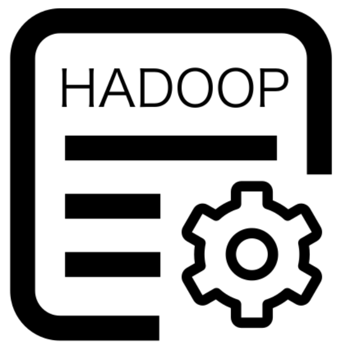
More GPUs involved

Longer training time

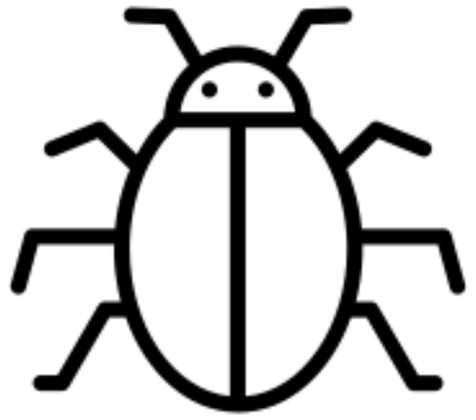
Failures are frequent

- Software failures

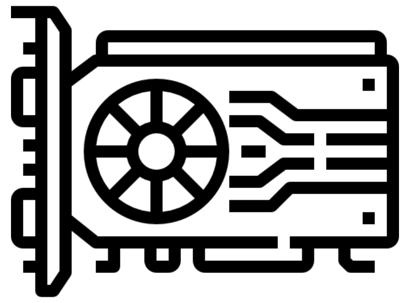
Library failures



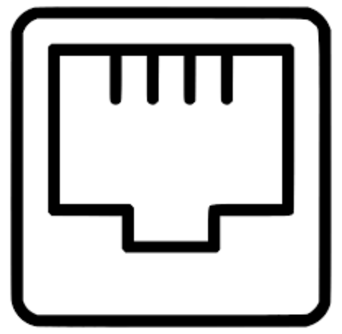
Remote storage failures



- Hardware failures



GPU failures



Link failures

Switch failures

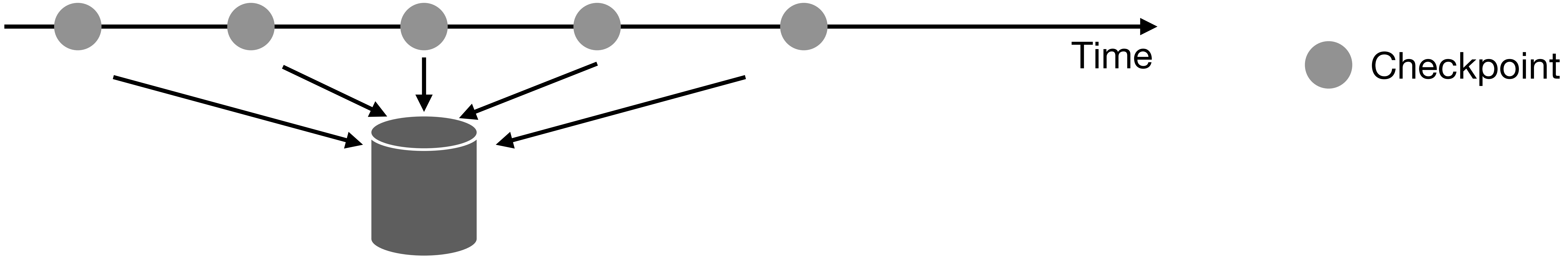
- OPT-175B: 100+ failures[1] in two months

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

Checkpoint for failure recovery

- How checkpoint works?

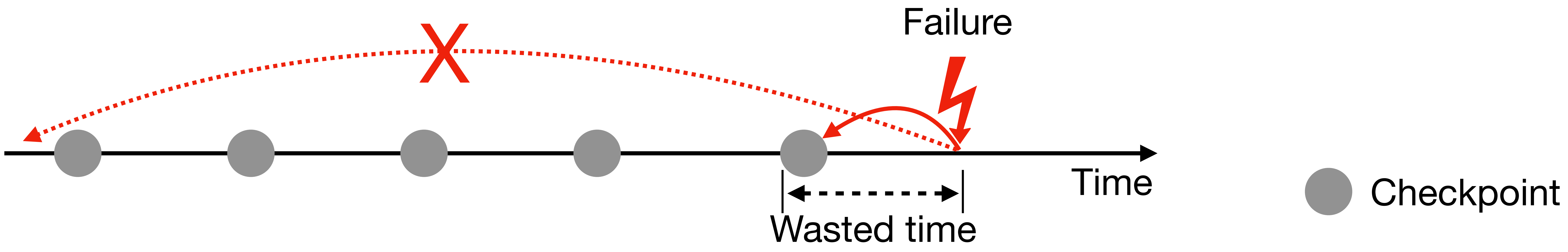
Periodically checkpoint the model states → Resume from the latest checkpoint → Redo the computation



Checkpoint for failure recovery

- How checkpoint works?

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



Desire higher checkpoint frequency

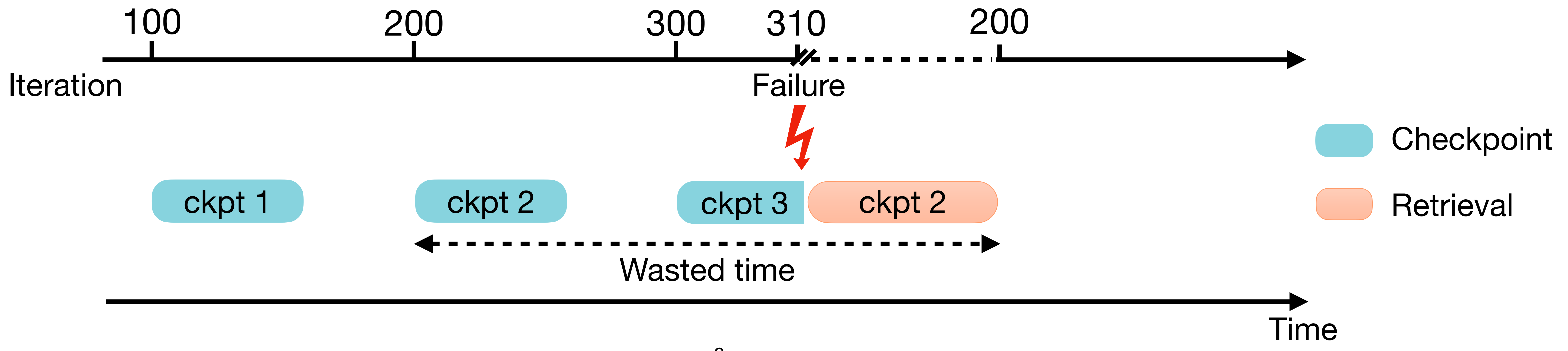
Checkpoint in LLM

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

Gemini

Checkpoint to CPU memory

- CPU memory is much larger than GPU memory

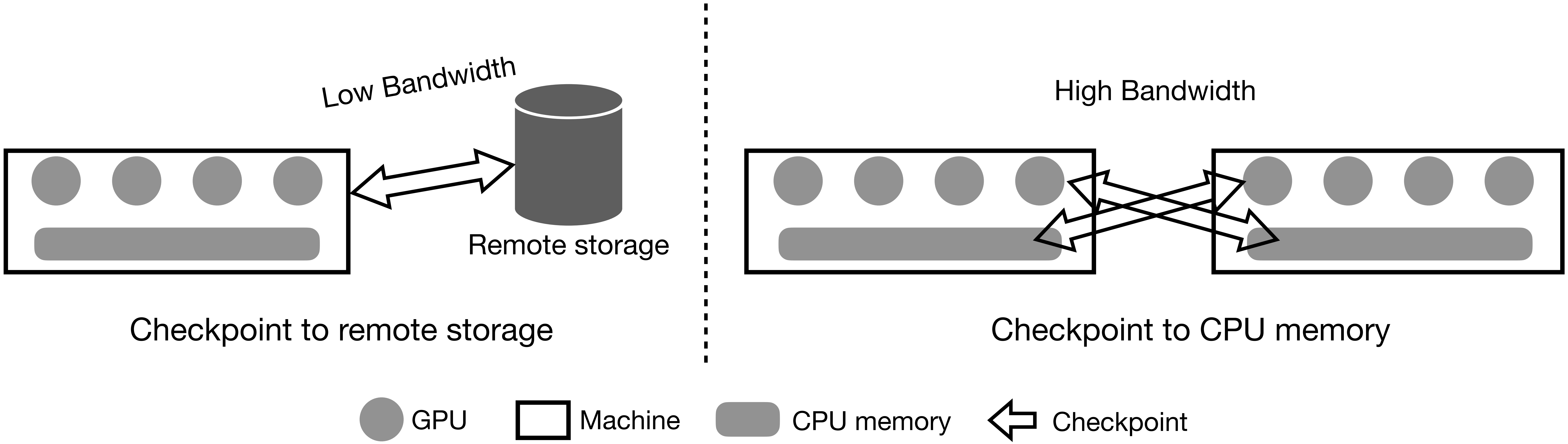
Instance type	Cloud	GPU	GPU memory	CPU memory
p3dn.24xlarge [14]	AWS	8 V100	8 × 32 GB	768 GB
p4d.24xlarge [15]	AWS	8 A100	8 × 40 GB	1152 GB
ND40rs_v2 [10]	Azure	8 V100	8 × 32 GB	672 GB
ND96asr_v4 [11]	Azure	8 A100	8 × 40 GB	900 GB
n1-8-v100 [9]	GCP	8 V100	8 × 32 GB	624 GB
a2-highgpu-8g [9]	GCP	8 A100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

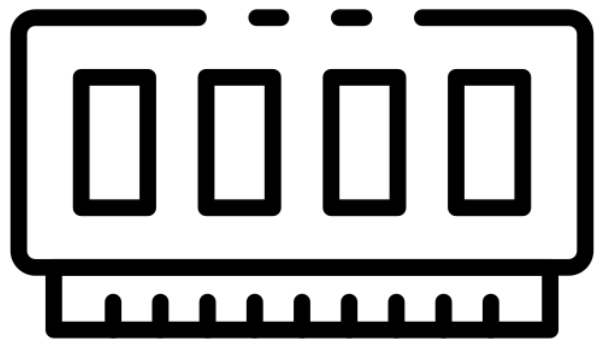


Gemini

Checkpoint to CPU memory

- CPU memory is much larger than GPU memory
- Checkpoint to CPU memory enables a much higher frequency
- CPU memory only stores checkpoints for failure recovery

Decouple the checkpoints for different purposes



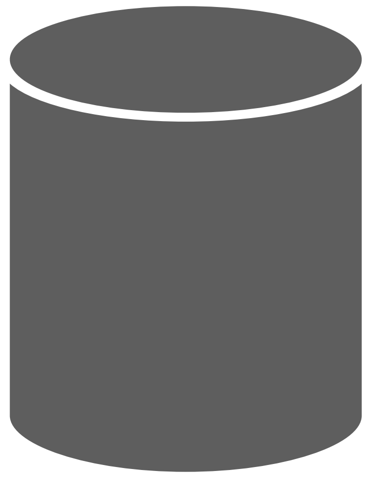
CPU memory

Failure recovery

- High-frequent checkpoints
- Only need the latest one

Debugging, accuracy evaluation

- Need checkpoint history
- Low-frequent checkpoints

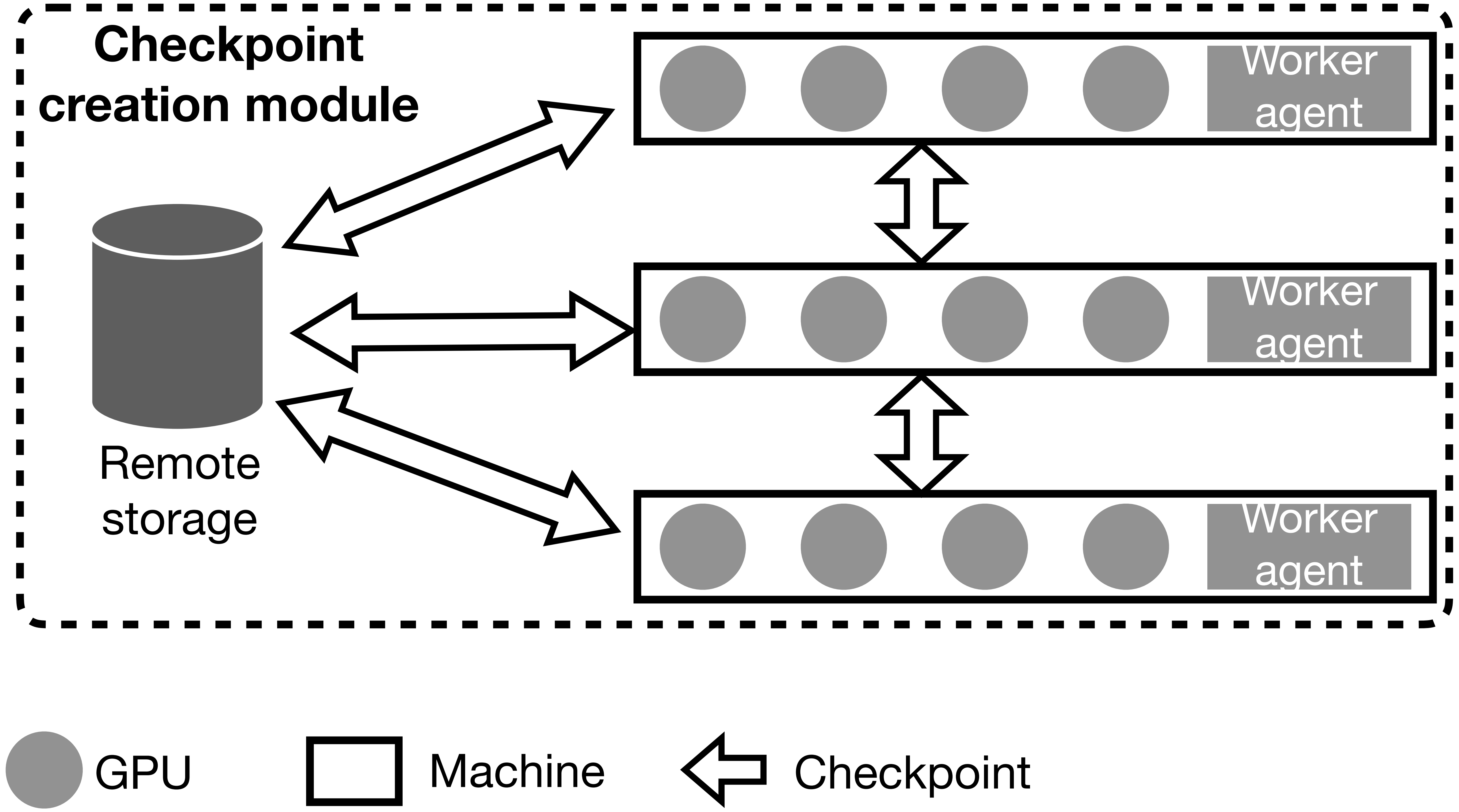


Remote storage

Gemini

Architecture

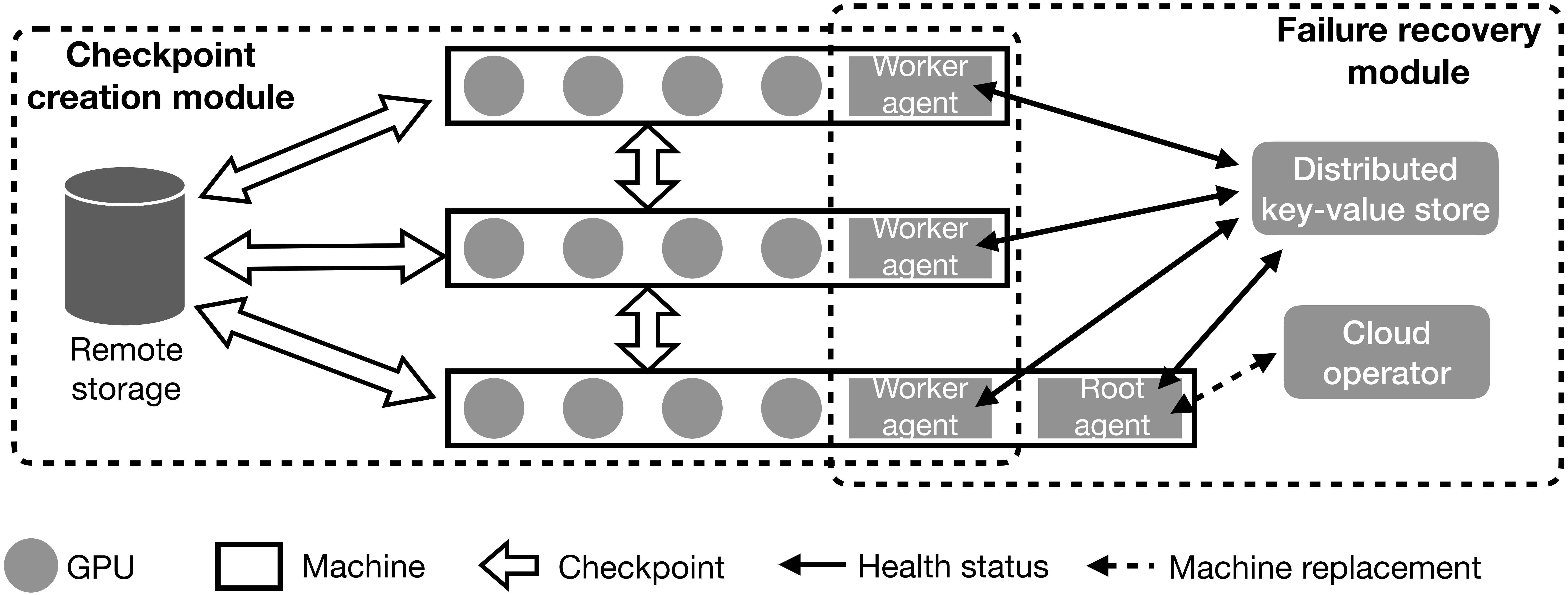
- Two modules



Gemini

Architecture

- Two modules

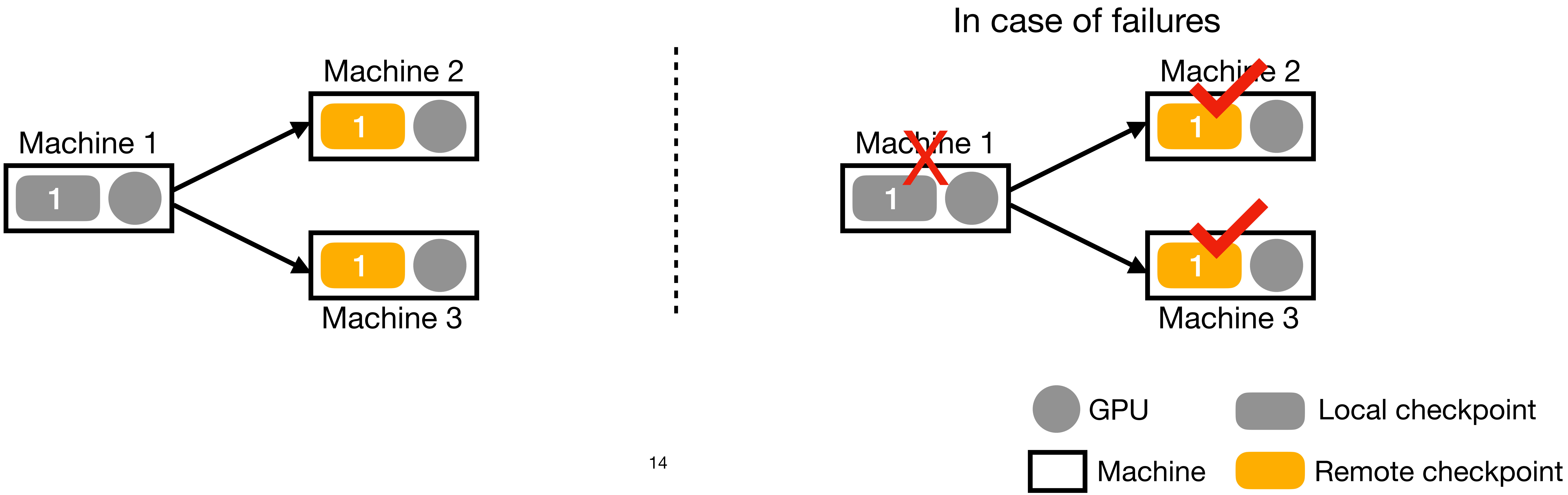


Challenge #1

- Data stored in CPU memory can get lost

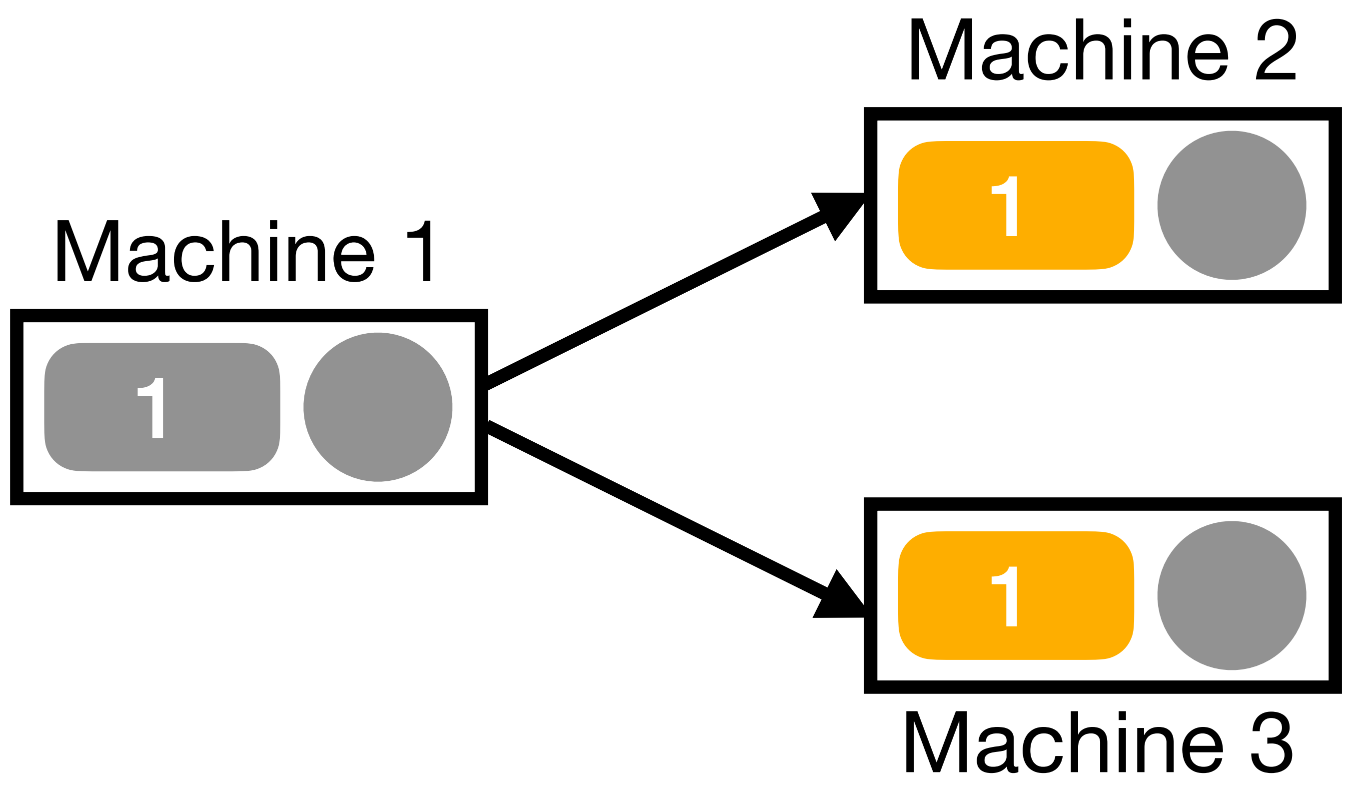
Challenge #1

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

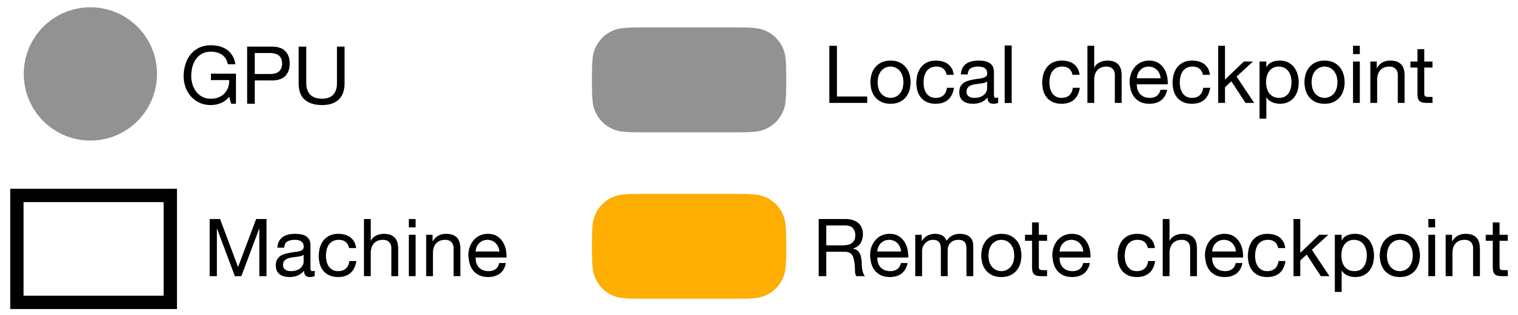


Challenge #1

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



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



Challenge #1

Optimal checkpoint placement

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

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

Solution

Group placement strategy

- Two steps

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

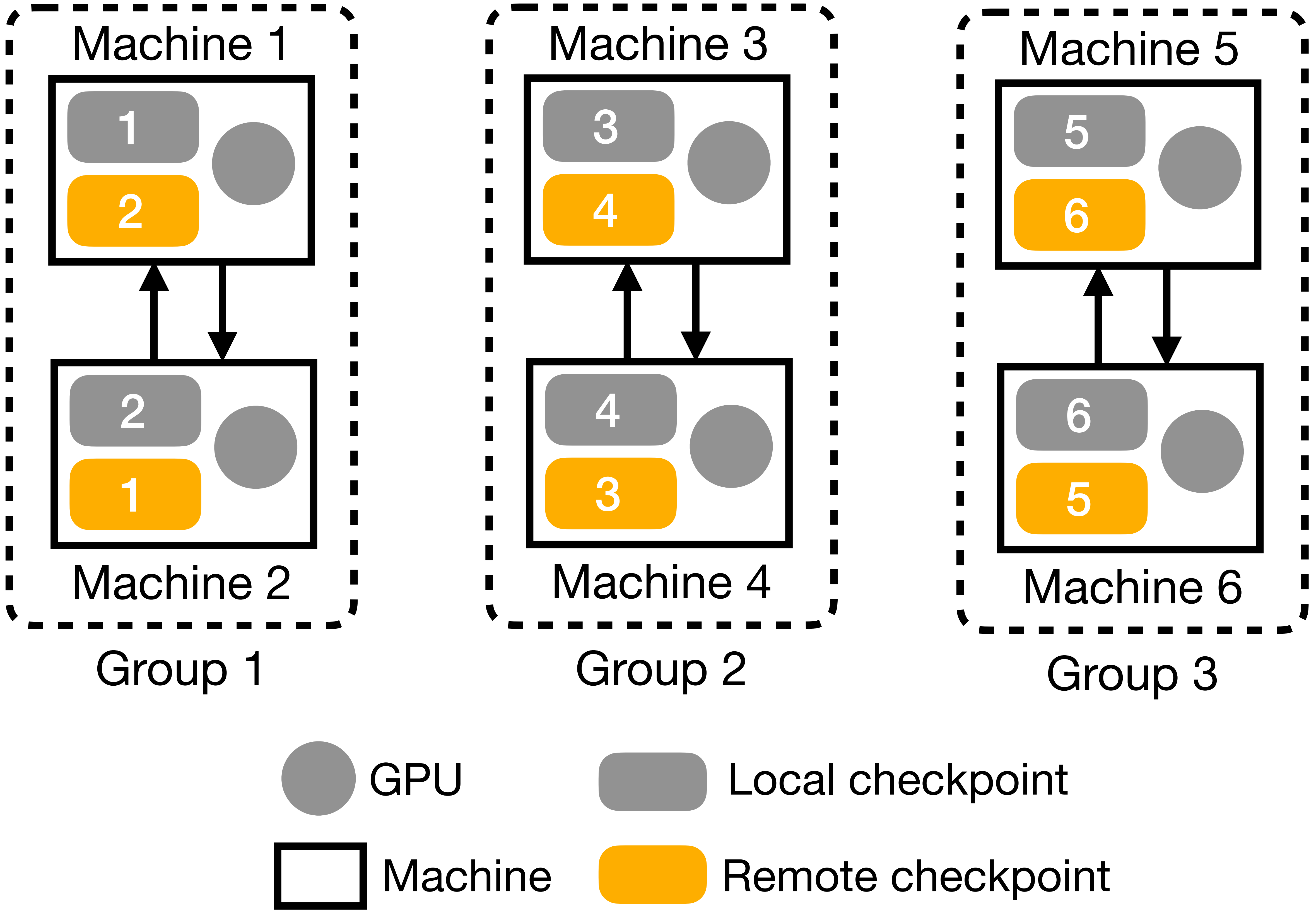
Solution

Group placement strategy

- Two steps

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

- An example with two replicas



Solution

Group placement strategy

- Two steps

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

Group placement strategy is provably optimal

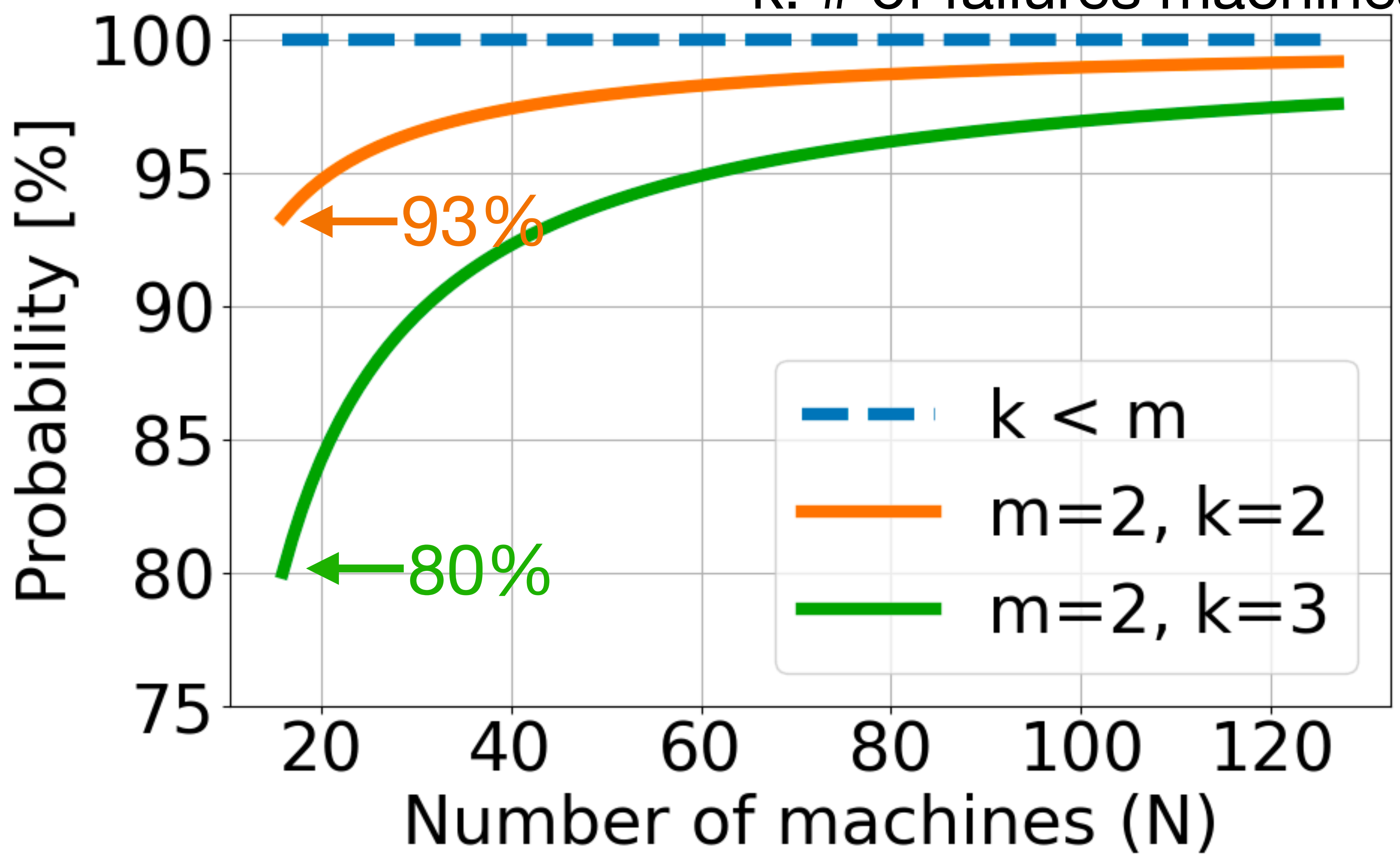
Solution

Group placement strategy

- Two steps

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

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

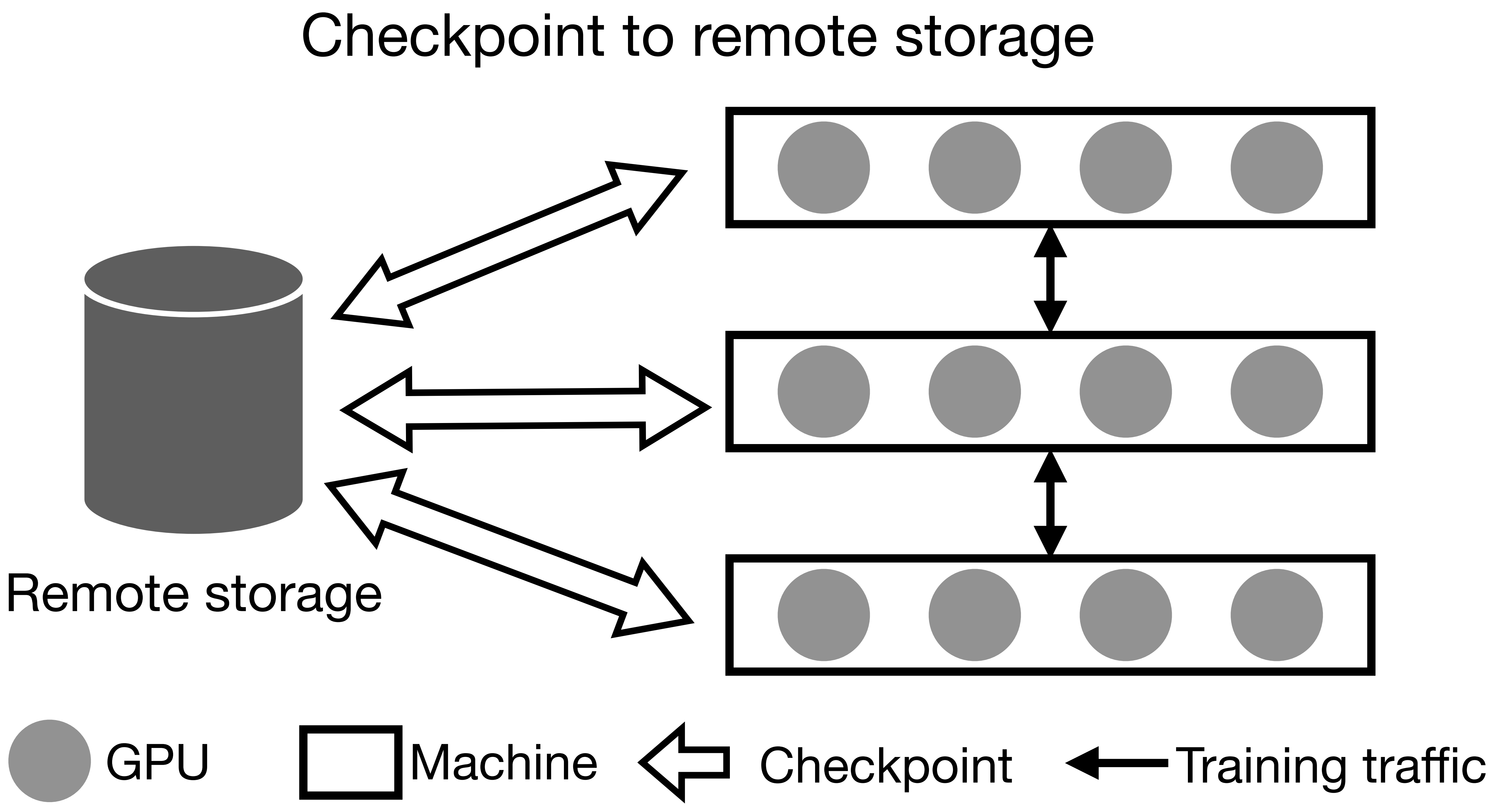


Group placement strategy is provably optimal

Two checkpoint replicas can already handle most failure cases!

Challenge #2

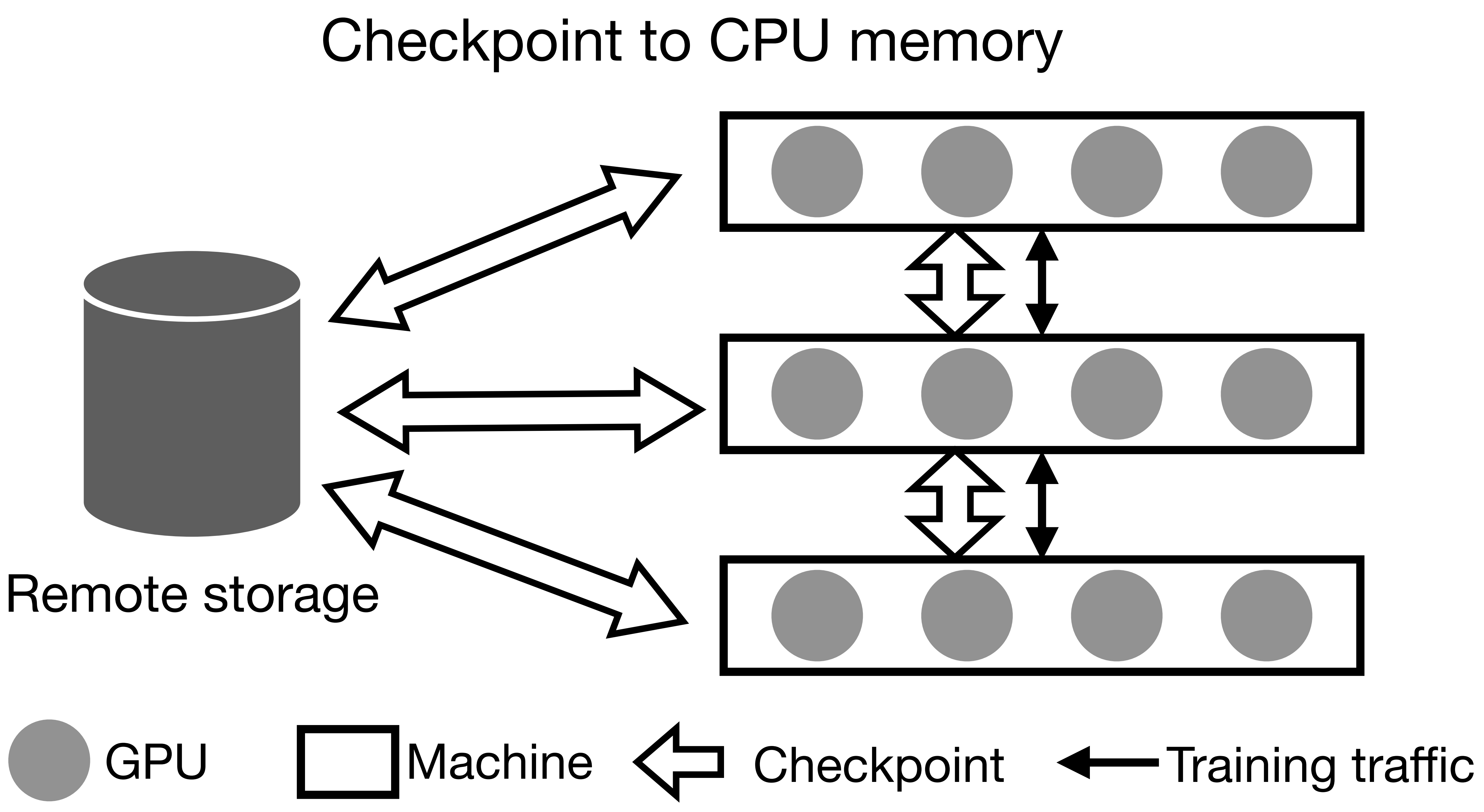
- Checkpoint traffic interferes with training traffic



Checkpoint traffic and training traffic have different networks

Challenge #2

- Checkpoint traffic interferes with training traffic



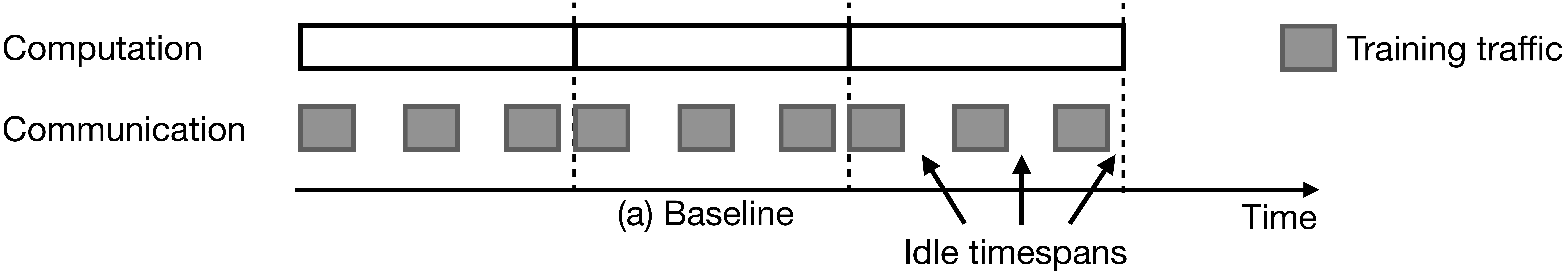
Checkpoint traffic and training traffic **shares the same network**

It can harm training throughput

Solution

Traffic interleaving

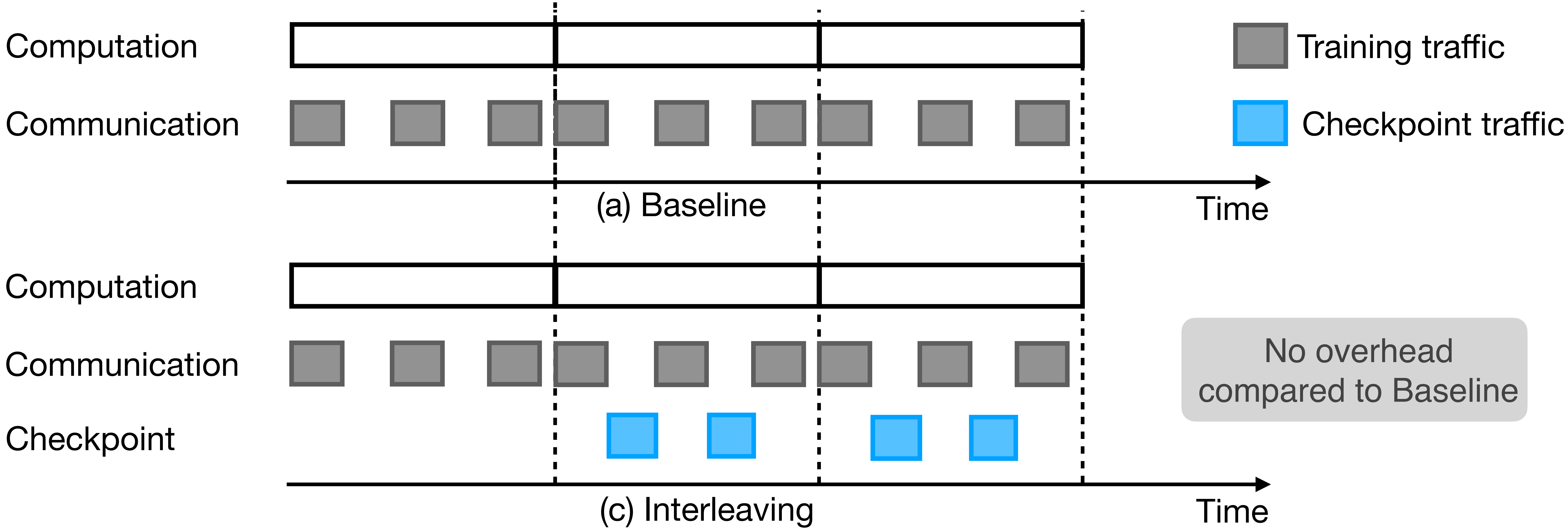
- Observation: Idle timespans in the network



Solution

Traffic interleaving

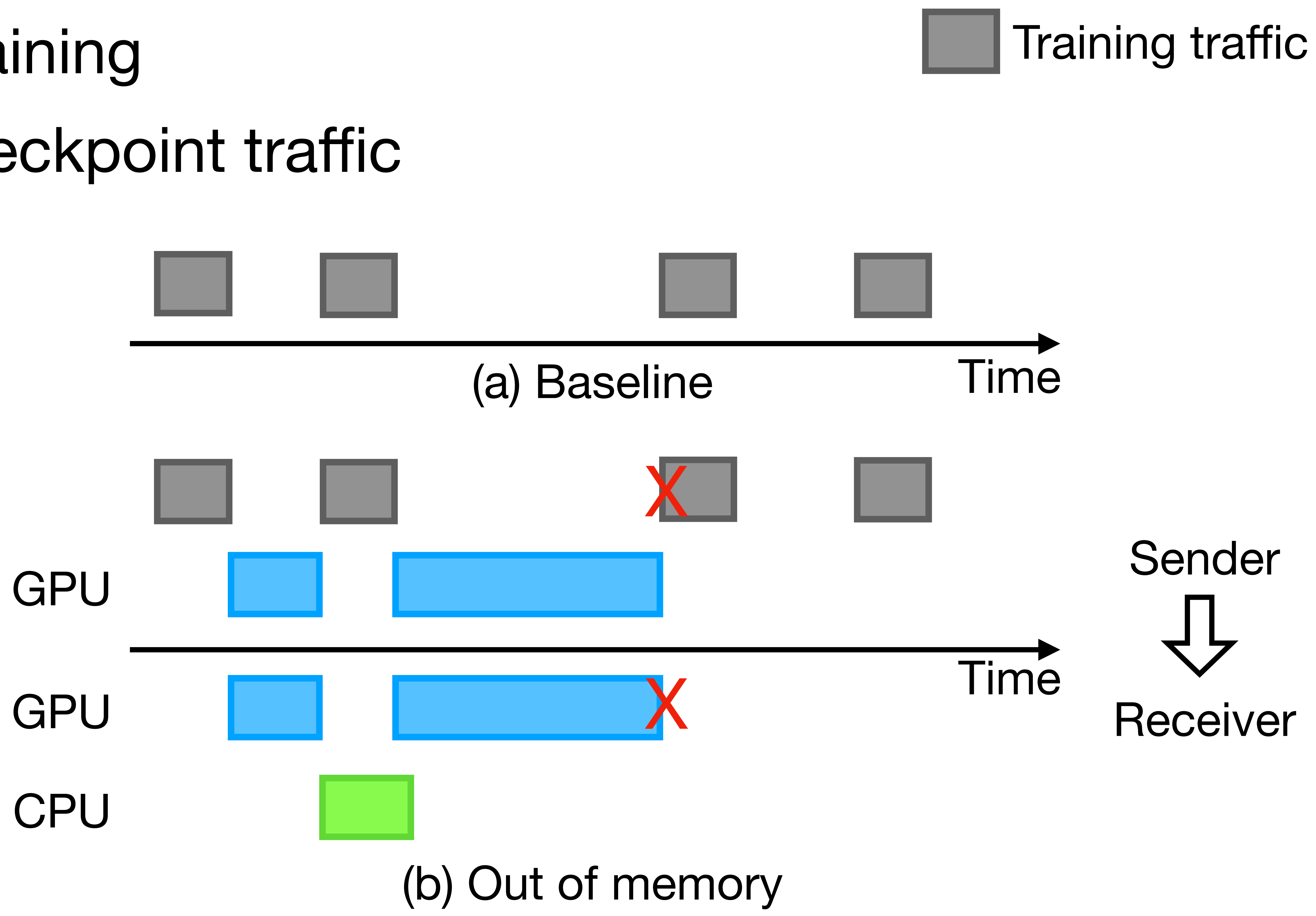
- Insert checkpoint traffic in idle timespans



Out-of-memory issue

- GPU memory is mainly used for training
- Limited spare GPU memory for checkpoint traffic

How to minimize the extra GPU memory consumption?



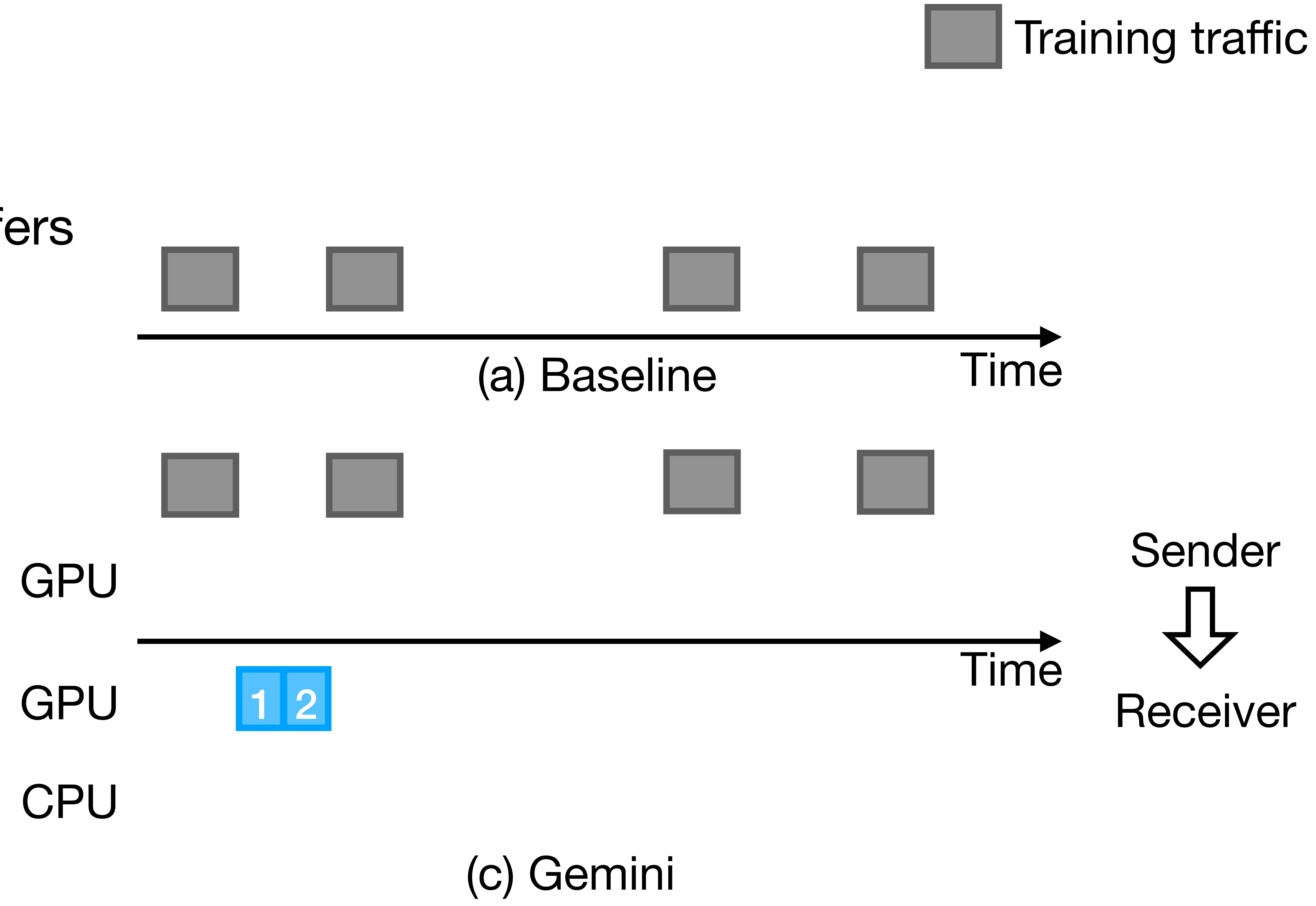
(a) Baseline

(b) Out of memory

Our design

Checkpoint partition and pipelining

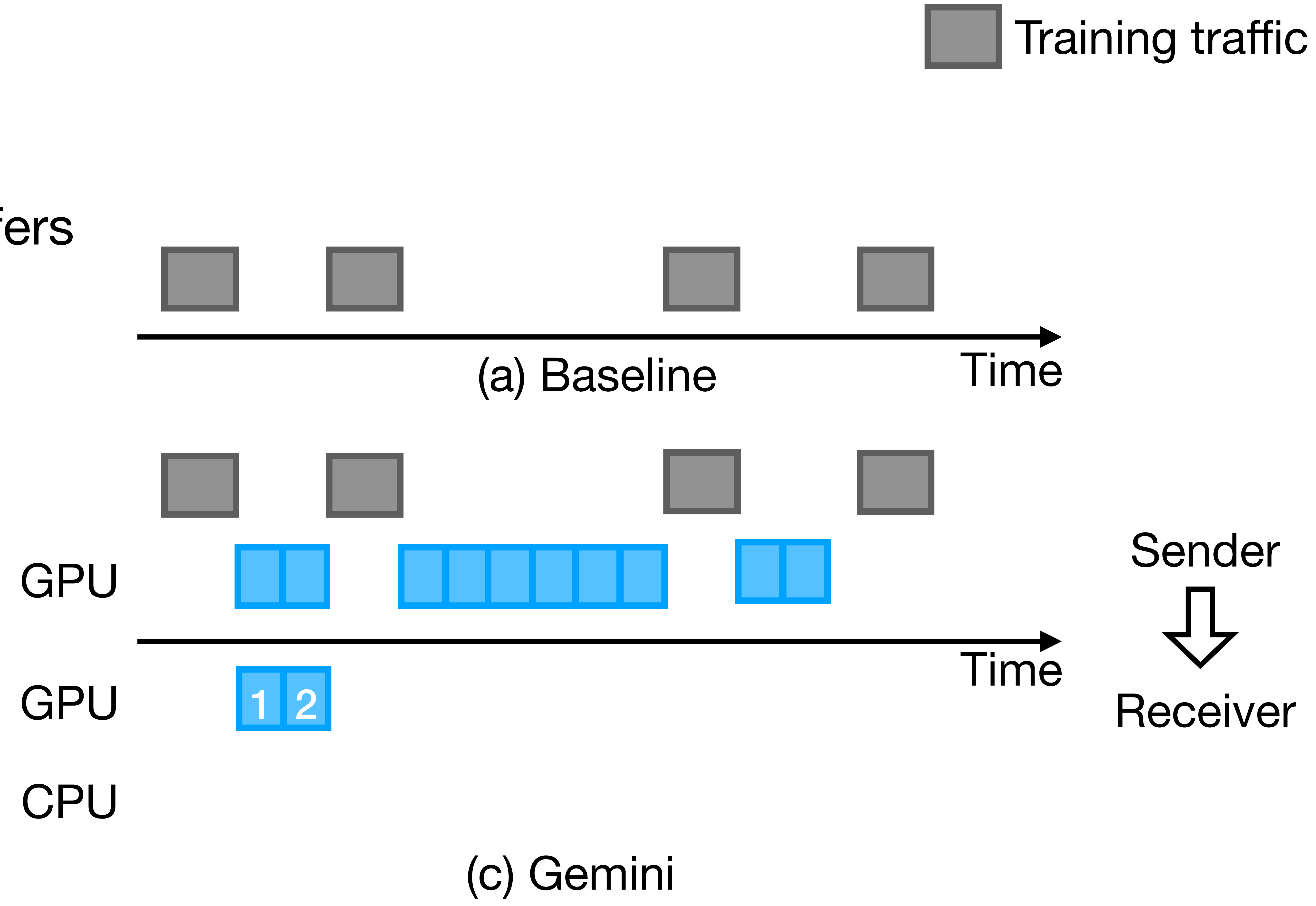
- Keys ideas
    - Reserve a GPU buffer at the receiver
    - Partition the buffer to multiple sub-buffers



Our design

Checkpoint partition and pipelining

- Keys ideas
  - Reserve a GPU buffer at the receiver
  - Partition the buffer to multiple sub-buffers
  - Pipeline checkpoint communications



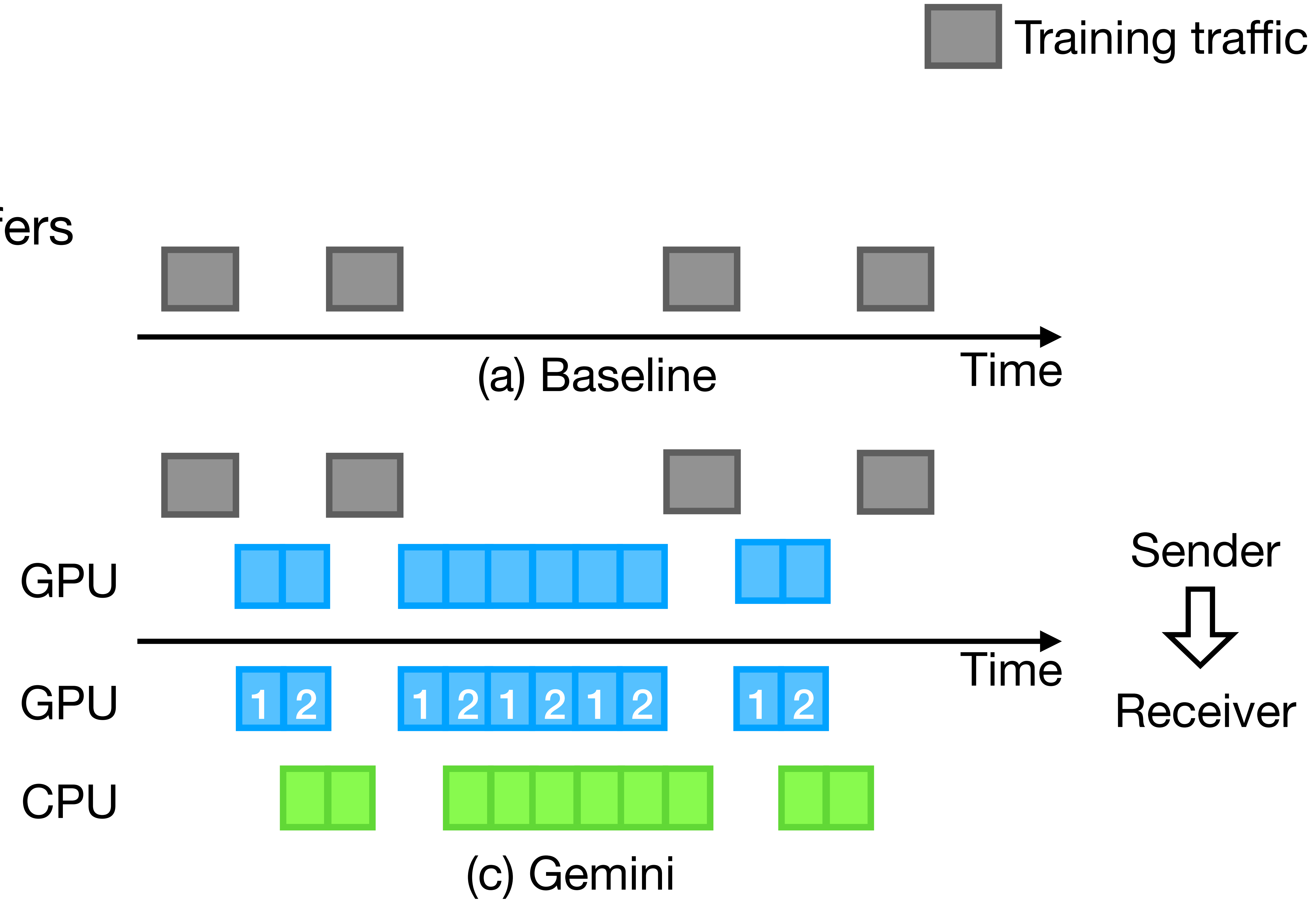
Our design

Checkpoint partition and pipelining

- Keys ideas
  - Reserve a GPU buffer at the receiver
  - Partition the buffer to multiple sub-buffers
  - Pipeline checkpoint communications

The GPU sub-buffers are reused

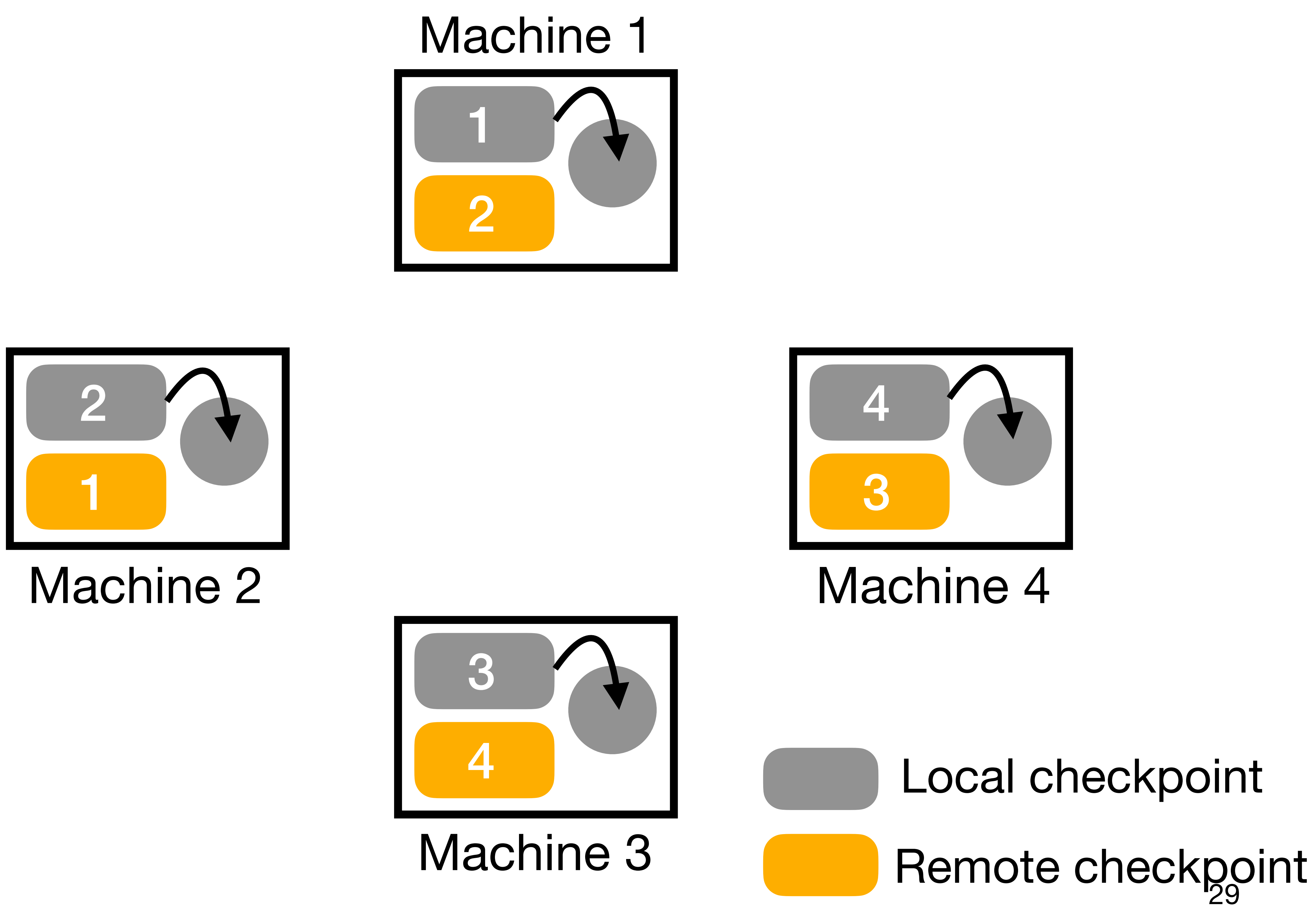
A small GPU buffer, e.g., 128MB, is sufficient



Resume training from failures

Software failures

- Checkpoints are available at local



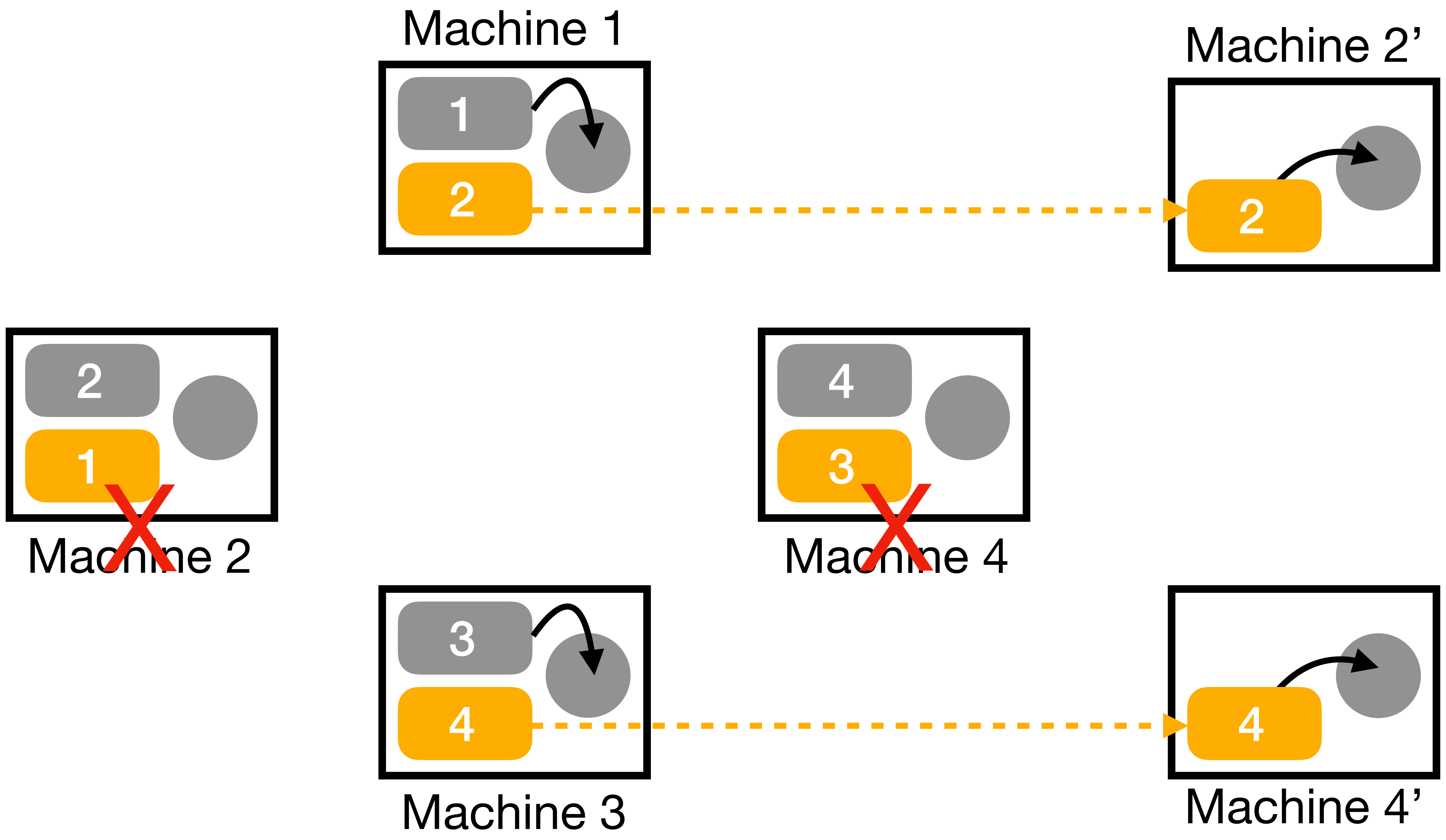
Negligible retrieval time

Just few iterations are lost

Resume training from failures

Hardware failures

- Checkpoints are still available at other machines

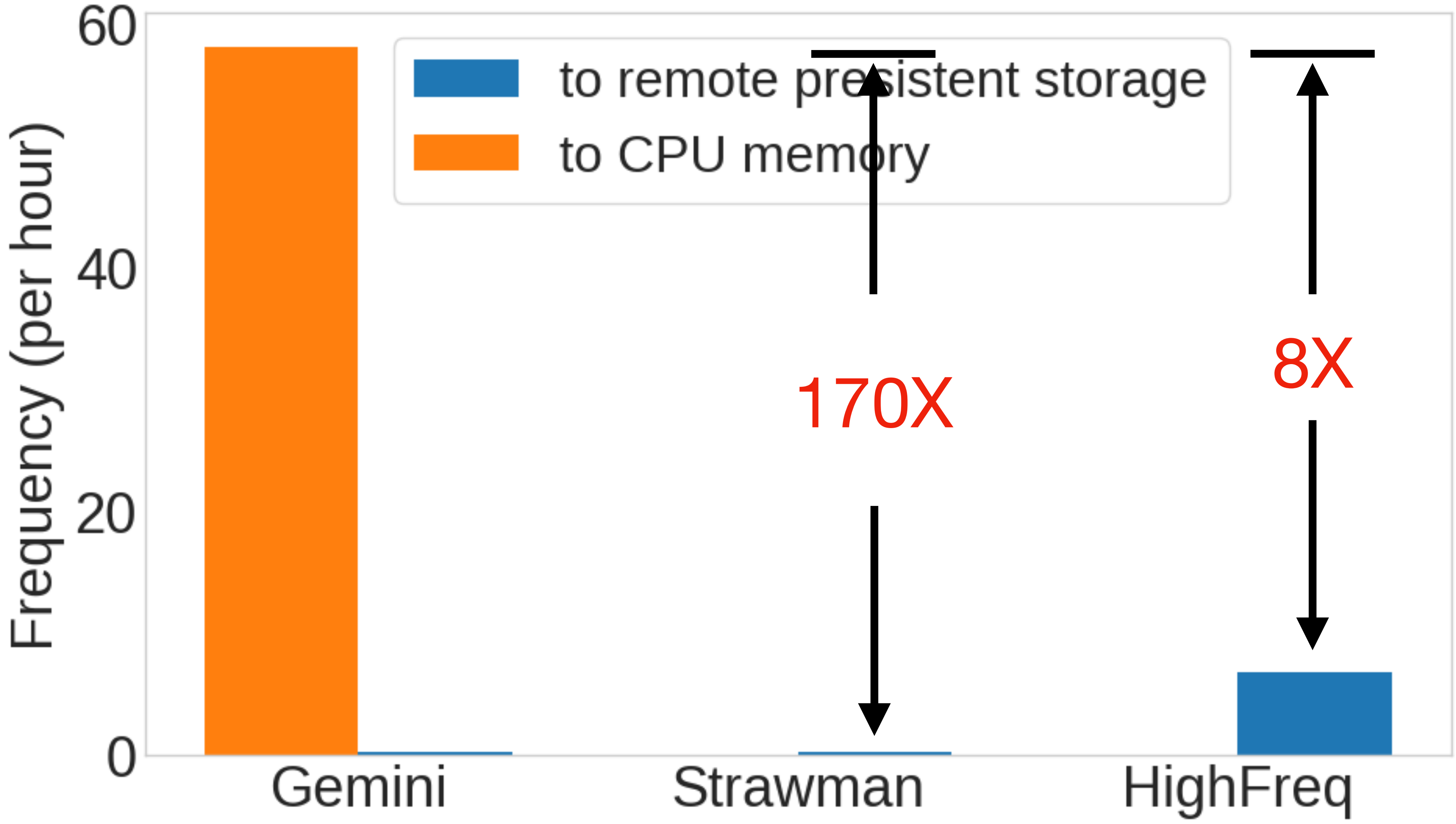


Evaluation

- Settings
  - Framework: DeepSpeed, ZeRO-3
  - 16 p4d instances (128 A100 GPUs), 400Gbps network bandwidth
  - The aggregated bandwidth of remote storage: 20Gbps
  - The size of LLM: 100 billion parameters
  - Reserved GPU buffer size: 128MB

Checkpoint frequency

- Baselines
  - Strawman: every 3 hours (BLOOM's frequency [1])
  - HighFreq: saturate the remote storage bandwidth capacity

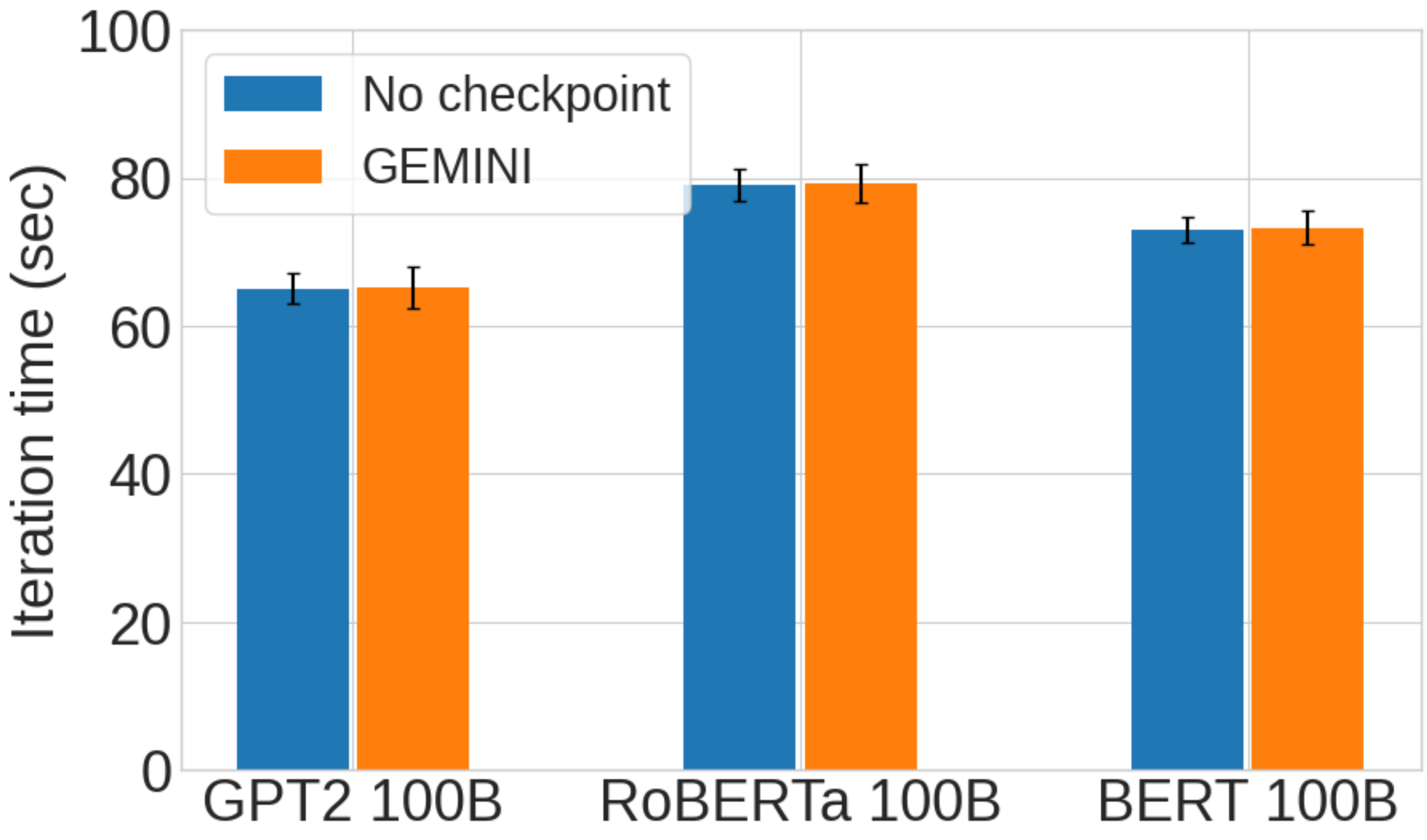


Gemini: Checkpoint model states every iteration

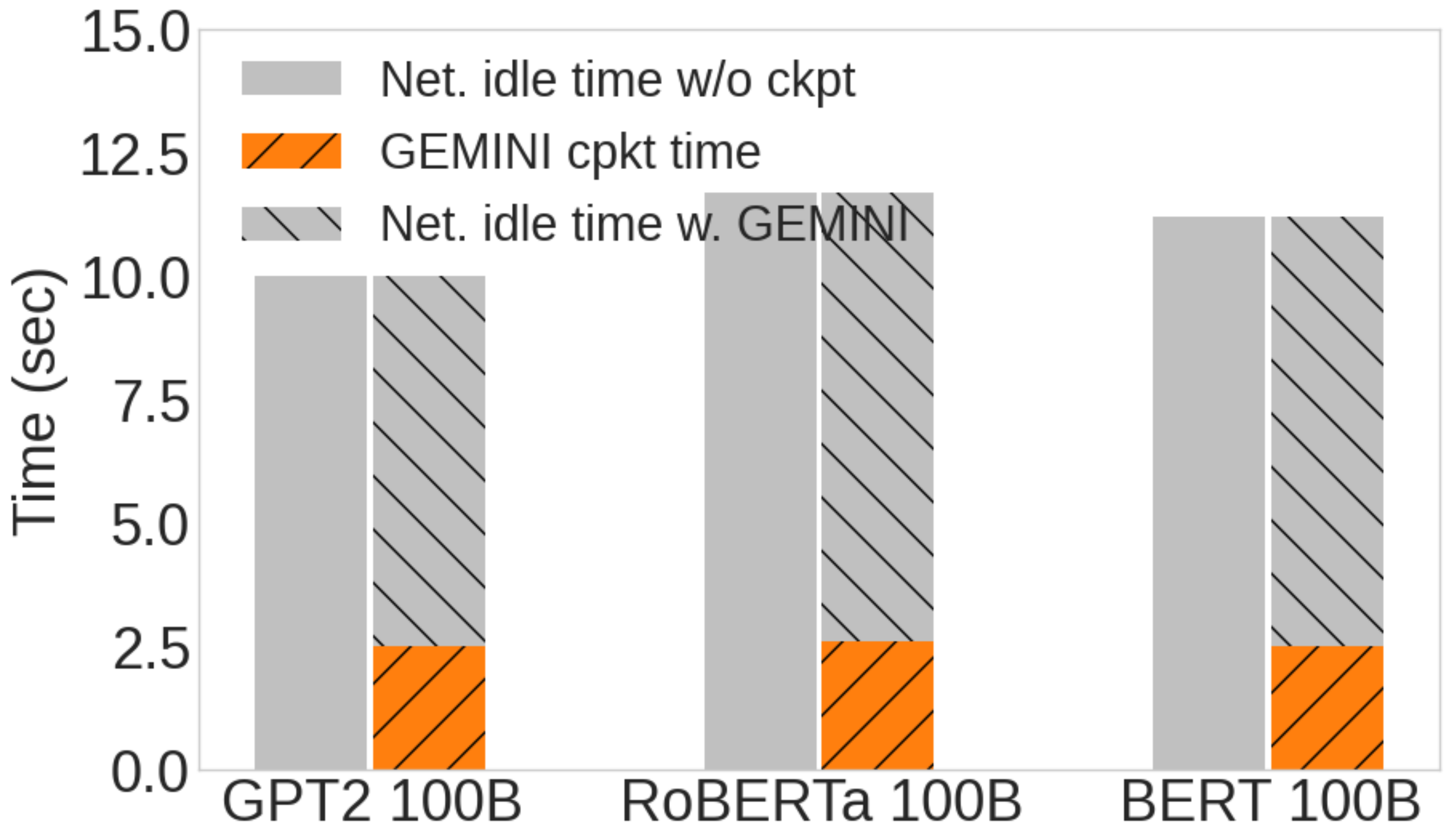
[1] Bloom: A 176b-parameter open-access multilingual language model. arXiv preprint arXiv:2211.05100

Training efficiency

- Training time
  - Gemini checkpoints model states to CPU memory every iteration



Negligible overhead on iteration time



Idle timespans can accommodate checkpoint traffic

Summary

- Large model training suffers from frequent failures
- Gemini checkpoints model states to CPU memory for failure recovery
    - Optimal checkpoint frequency, i.e., every iteration
    - Negligible overhead on training throughput
    - Applicable to different parallelism strategies of training