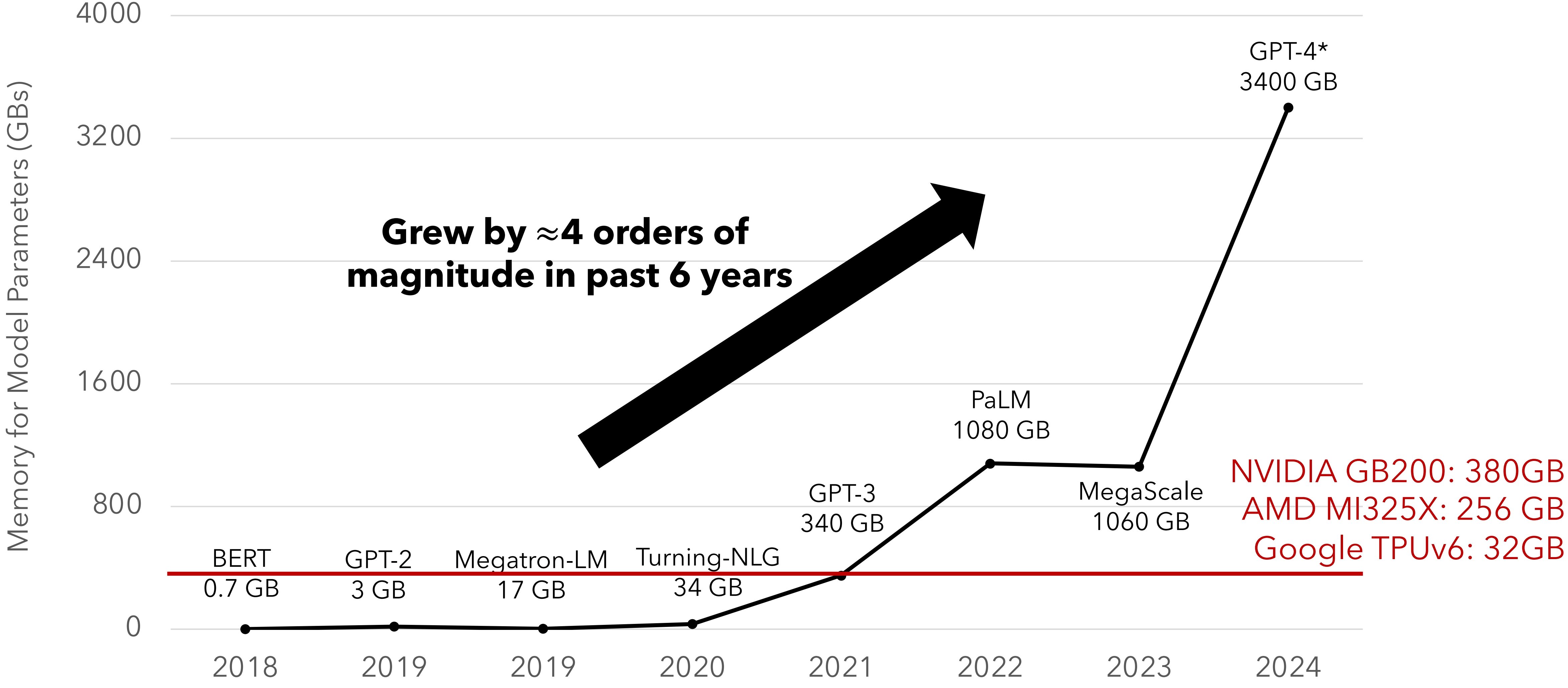
ReCycle: Resilient Training of Large DNNs Using Pipeline Adaptation

Swapnil Gandhi, Mark Zhao,
Athinagoras Skiadopoulos, Christos Kozyrakis

Stanford MAST

Stanford University

Models are **growing**



Assuming 2 bytes per model parameter

AI companies are using 10,000s of accelerators to train these massive models

Failures are **common**

*"During a 54-day snapshot period of pre-training, we experienced a total of **466 job interruptions**….Approximately 78% of the unexpected interruptions are attributed to confirmed hardware issues, such as GPU or host component failures…"*

- Llama Team @ Meta[1]

Similar reports at Google, Microsoft, Amazon, Alibaba, ByteDance, LAION,…

[1] The Llama 3 Herd of Models. https://arxiv.org/pdf/2407.21783

Failures have **large impact**

*"This is a particularly annoying problem to handle as **if one GPU has an issue,** the synchronized nature of distributed training means that **all GPUs get stuck**."*

- LAION Team[1]

*"Estimated 100+ host restarts due to hardware failures over the course of 2 months…**178,000 GPU hours of wasted time due to various malfunctions**"*

- OPT Team[2]

[1] Large Scale Openclip: L/14, H/14 And G/14 Trained On LAION-2B. https://laion.ai/blog/large-openclip/
[2] OPT: Open Pre-trained Transformer Language Models. https://arxiv.org/abs/2205.01068

Practitioners can **prioritize** either…

Performance

or

Resiliency

Practitioners can **prioritize** either…

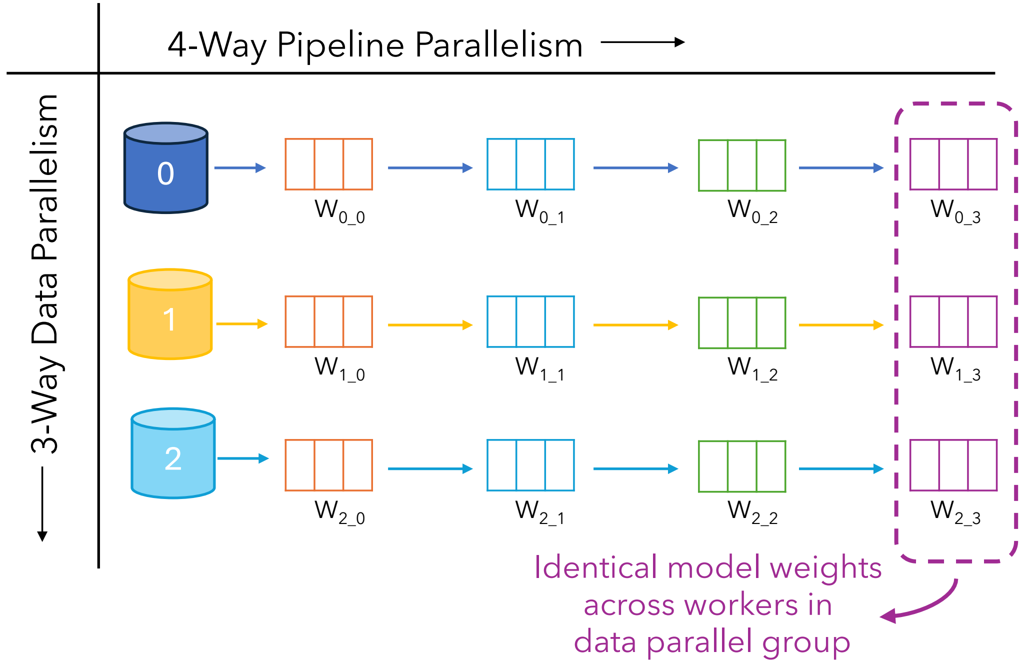
	Using all GPUs for training	**Reserving some GPUs as hot spares**
Performance	✓	✗
Resiliency	✗	✓
	No Overhead in Fault-Free Case	Constant Overhead in Fault-Free Case
	Training stalls when a GPU fails	Hot spare ensures stall-free training
	All or Nothing	All or Nothing

ReCycle prioritizes both

	Using all GPUs for training	**Reserving some GPUs as hot spares**	**ReCycle**
Performance	✓	✗	✓
Resiliency	✗	✓	✓
	No Overhead in Fault-Free Case	Constant Overhead in Fault-Free Case	No Overhead in Fault-Free Case
	Training stalls when a GPU fails	Hot spare ensures stall-free training	Ensures stall-free training without relying on hot spares
	All or Nothing	All or Nothing	Graceful Degradation

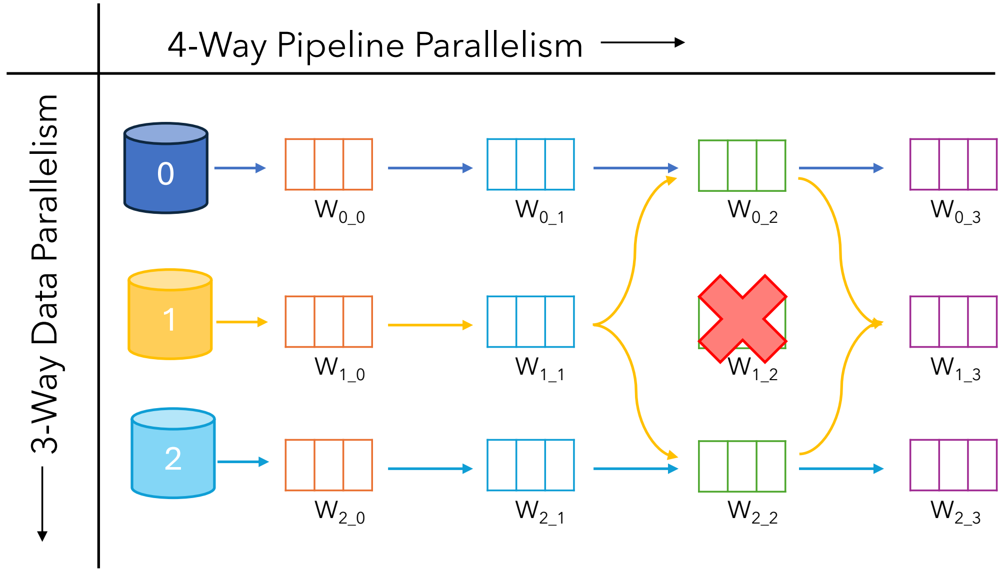
Adaptive Pipelining: Working Around Failures

Multiple copies of model parameters exist across data-parallel peers

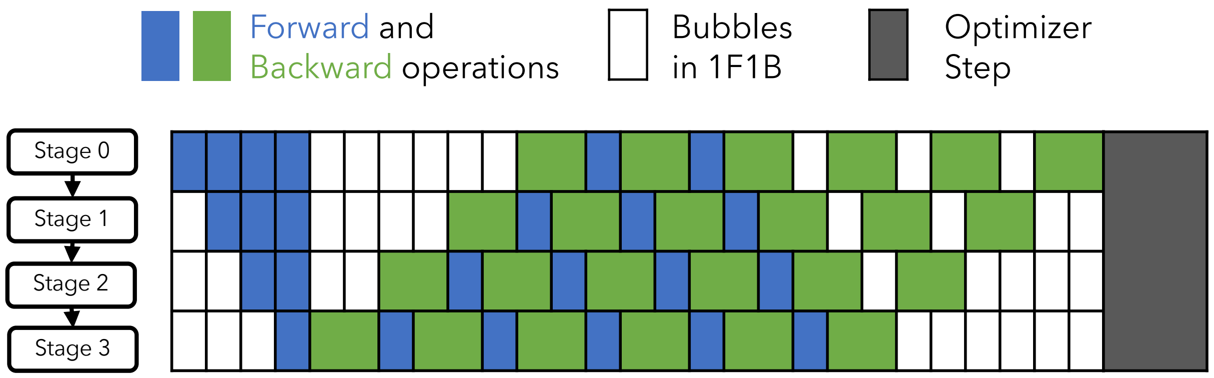


Adaptive Pipelining: Working Around Failures

Multiple copies of model parameters exist across data-parallel peers



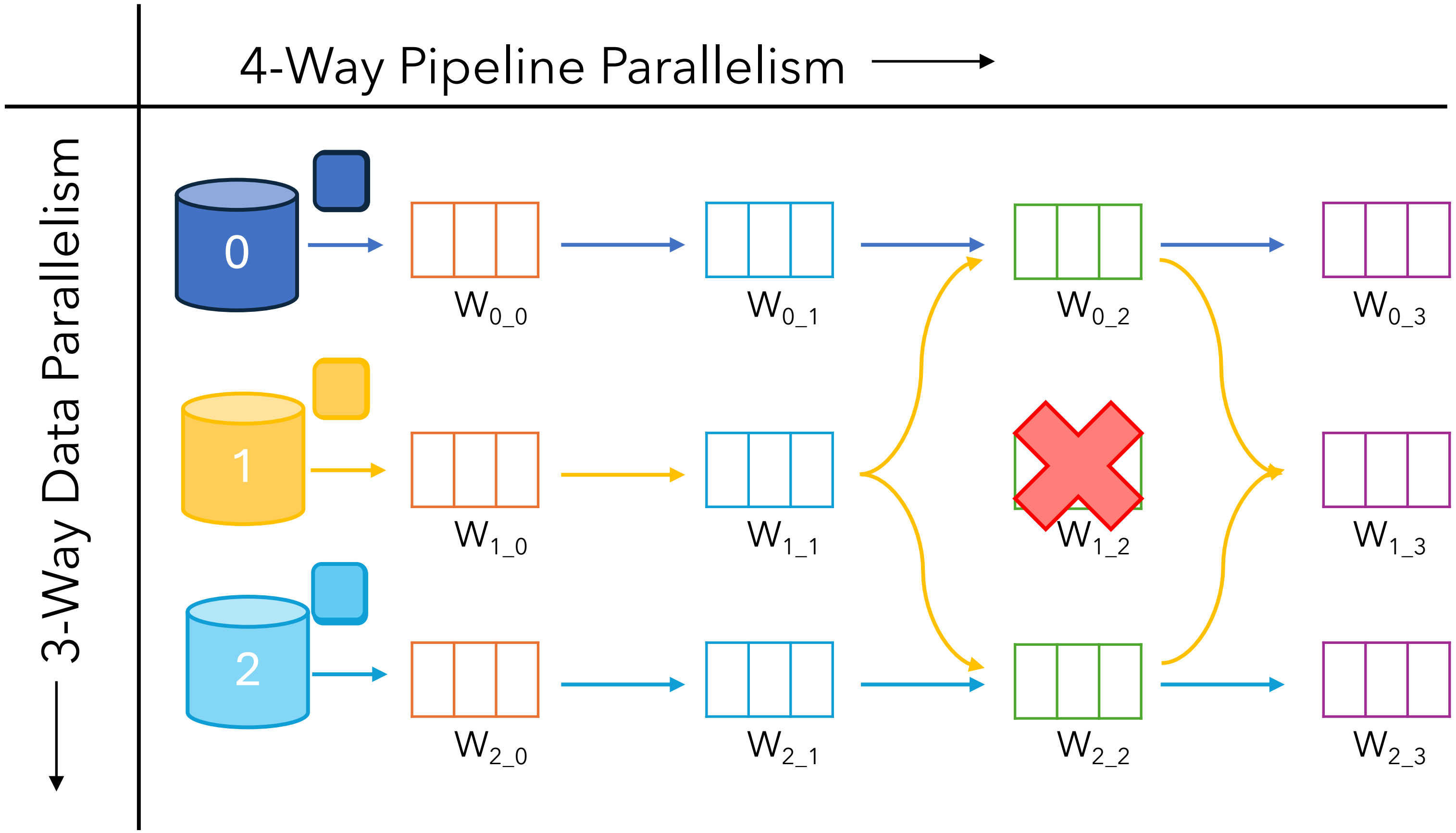
Bubbles are intervals within an iteration where worker idles due to unmet forward and backward operation dependencies



Each worker idle for 30% of the iteration in above 1F1B schedule

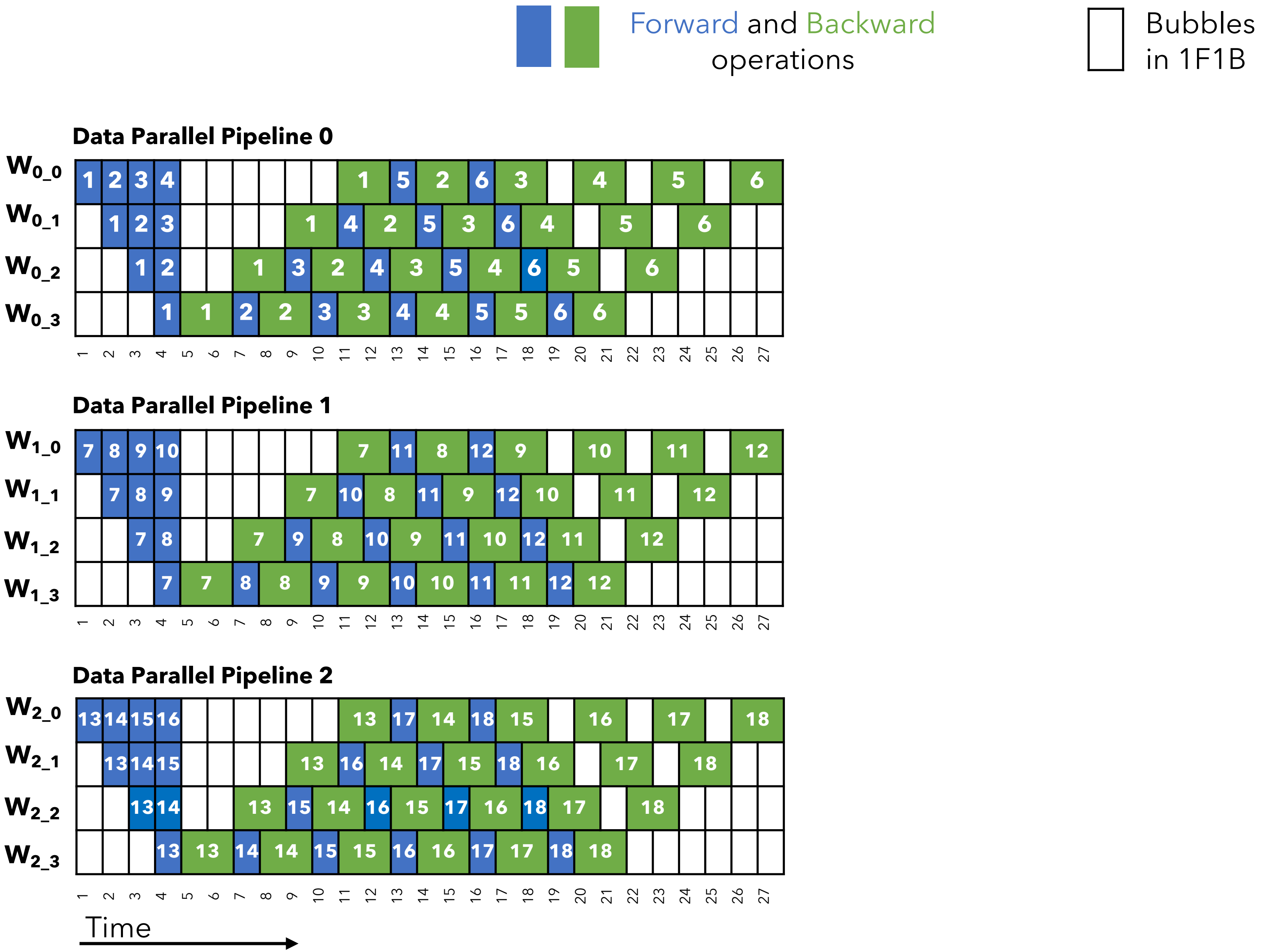
Functional data-parallel peers compensate for a failed worker by utilizing existing pipeline gaps to process re-routed microbatches

Adaptive Pipelining: Working Around Failures



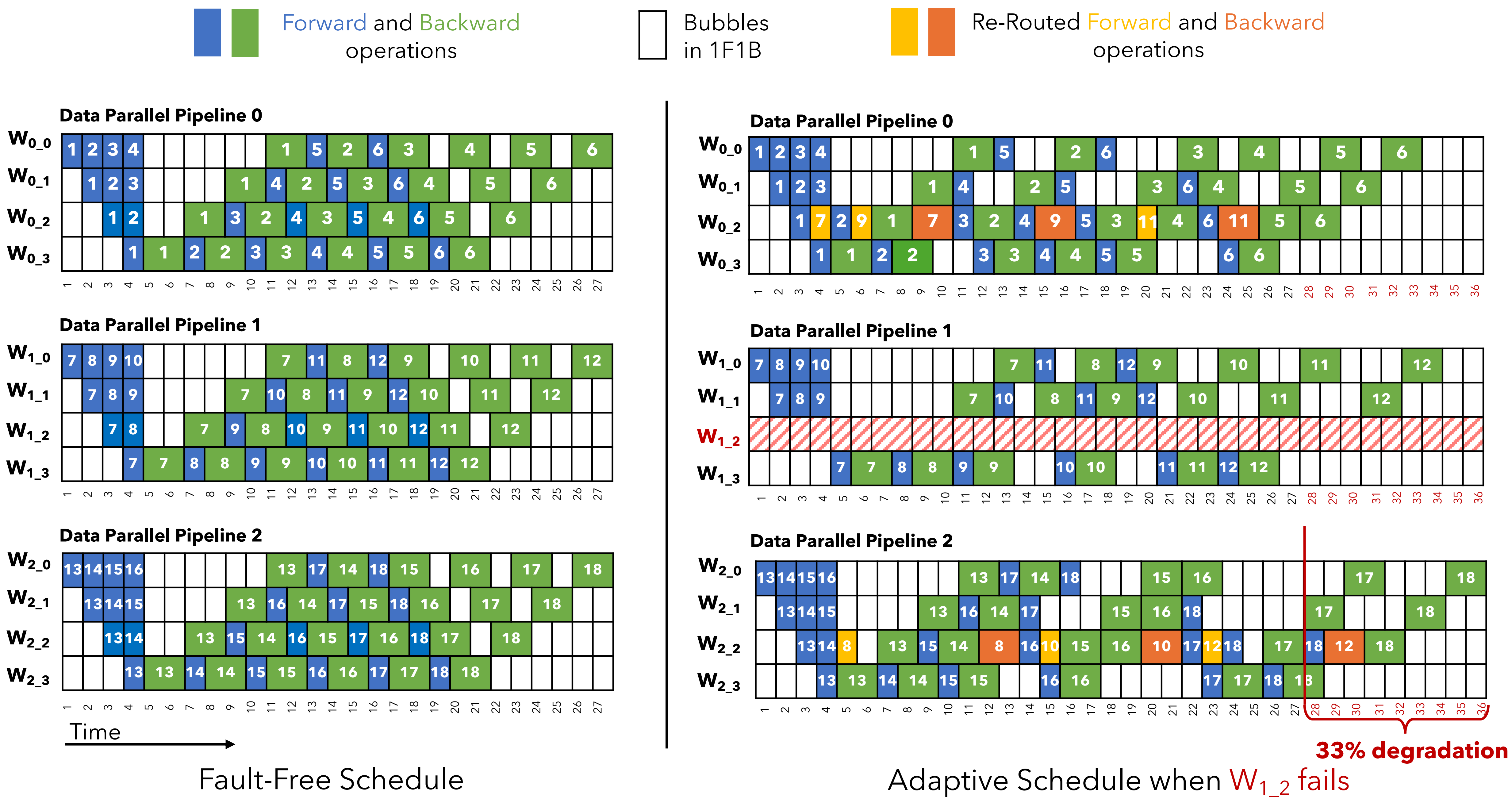
+ **Stall-free training** until at least 1 data-parallel peer is functional
+ **Parallel recovery** using all functional data-parallel peers
+ **No impact** on model convergence

But what about **performance**?



Fault-Free Schedule

3 Data-Parallel Pipelines, 4 Pipeline Stages, 6 Micro batches per Data-Parallel Pipeline



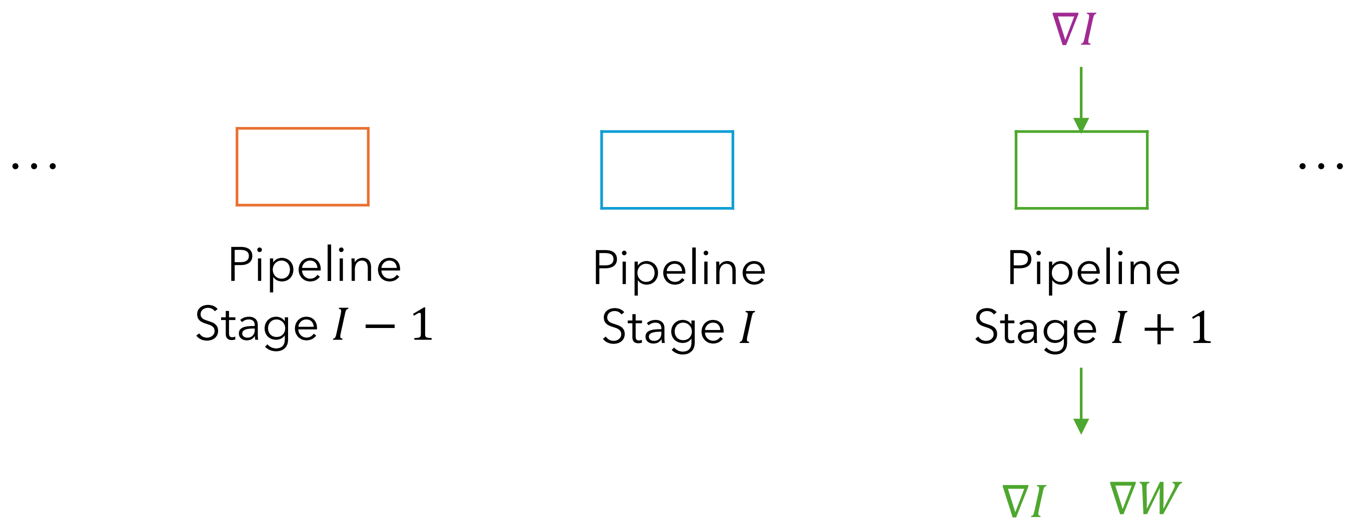
3 Data-Parallel Pipelines, 4 Pipeline Stages, 6 Micro batches per Data-Parallel Pipeline

Can we make Adaptive Pipelining **performant**?

Background: Backprop

In conventional backprop, each pipeline stage computes two gradients per layer:

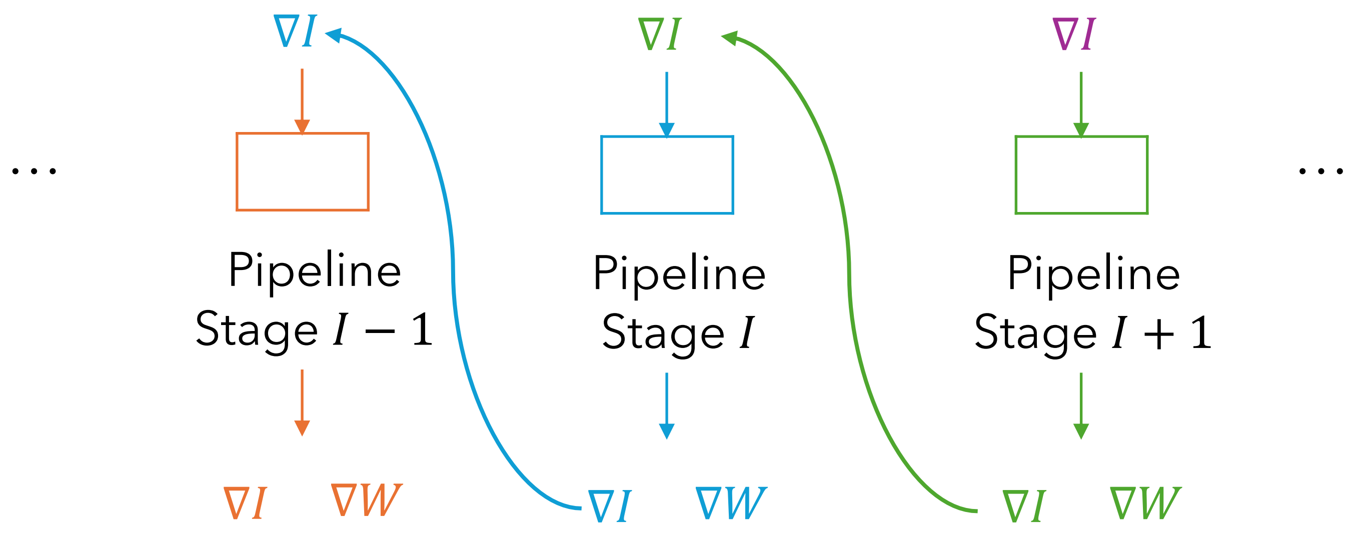
1. Input Gradient (∇I), used to propagate errors back through the network
2. Weight Gradient (∇W), used to update model parameters



Background: Backprop

Dependency: Stage I's gradient computation depends only on the input gradient ∇I from stage $I + 1$'s

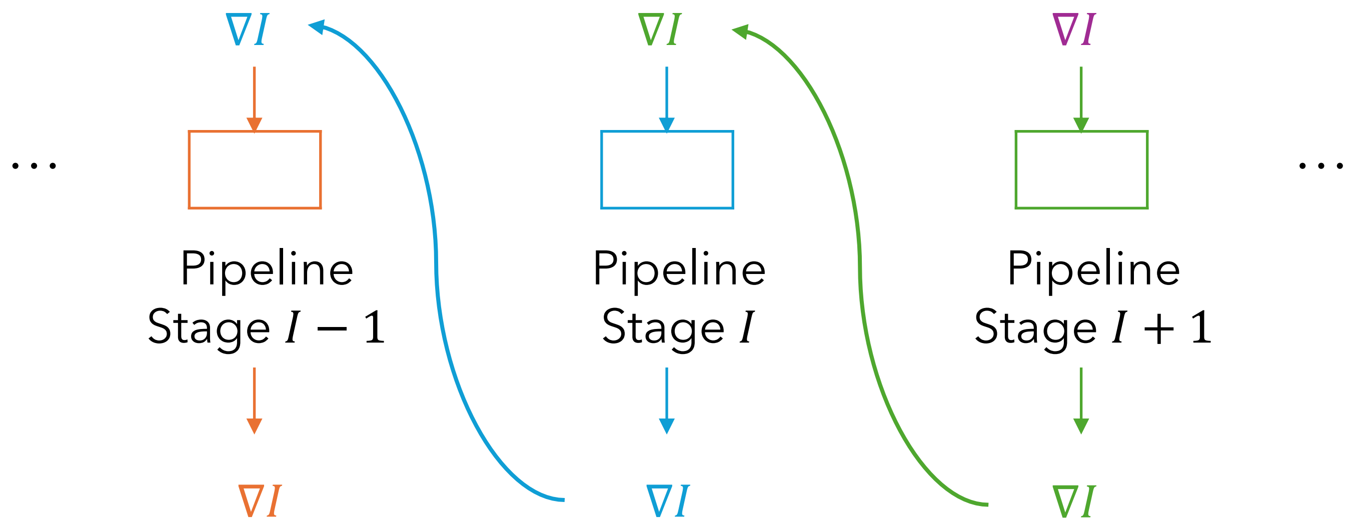
Challenge: These computations are tightly coupled, length-ing computation dependency across pipeline stages



Decoupled Backprop: Filling unused bubbles

Splitting conventional backprop in two distinct phases: B_{Input} and B_{Weight} allows greater scheduling flexibility.

In B_{Input} phase, the input gradients are computed independently, allowing error to be propagated to previous stage without waiting for weight gradient

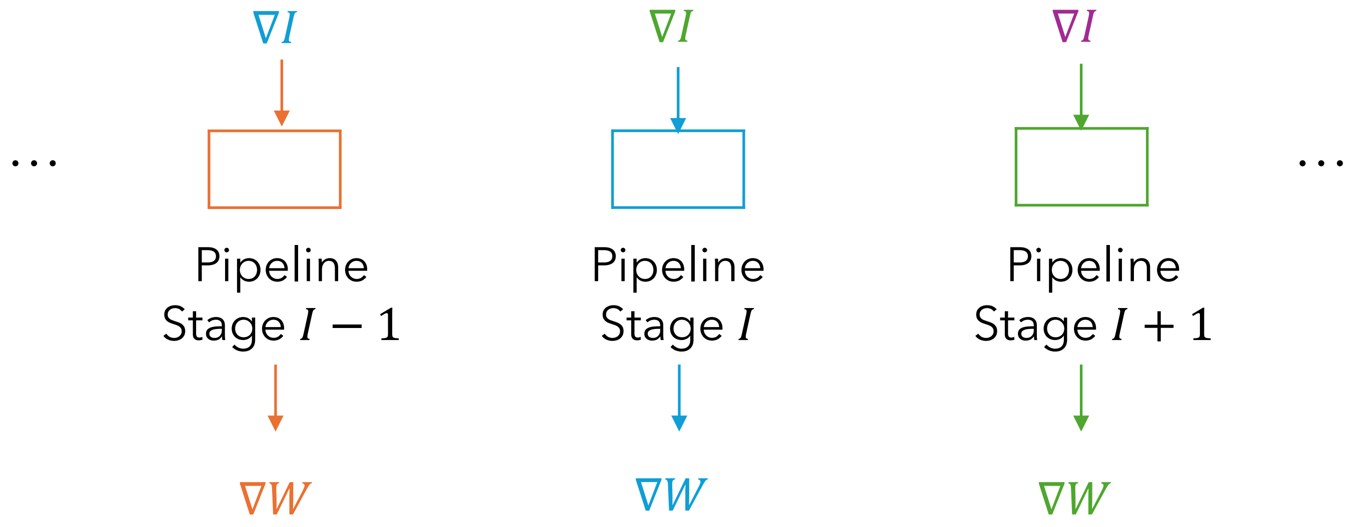


B_{Input} Phase

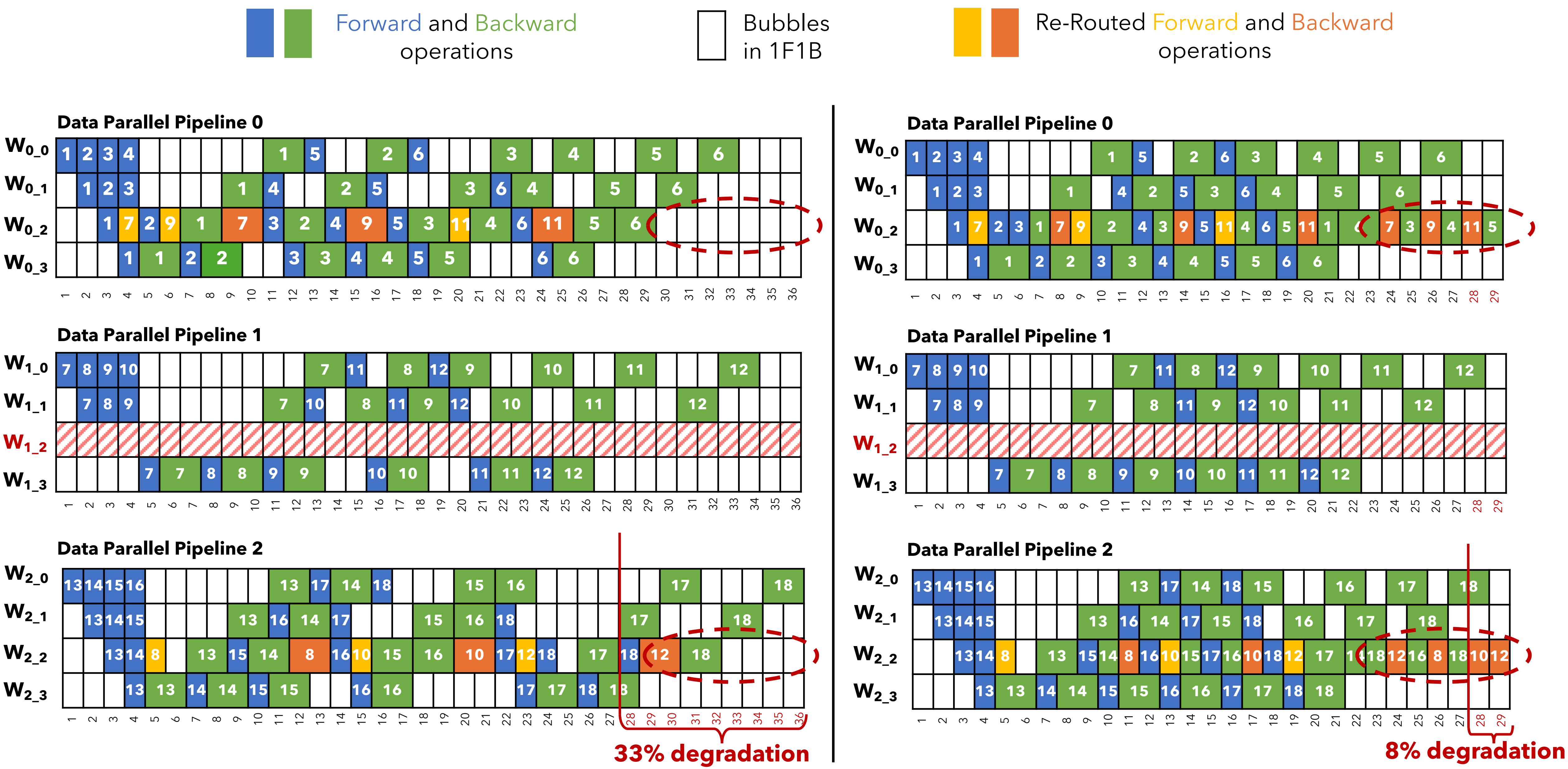
Decoupled Backprop: Filling unused bubbles

Weight gradient ∇W is still computed, but it is performed independently of input gradient ∇I

This decoupling allows two gradient computation to be performed in separate phases, without waiting on each other



B_{Weight} Phase



Adaptive Schedule with Coupled Backprop when W_{1_2} fails

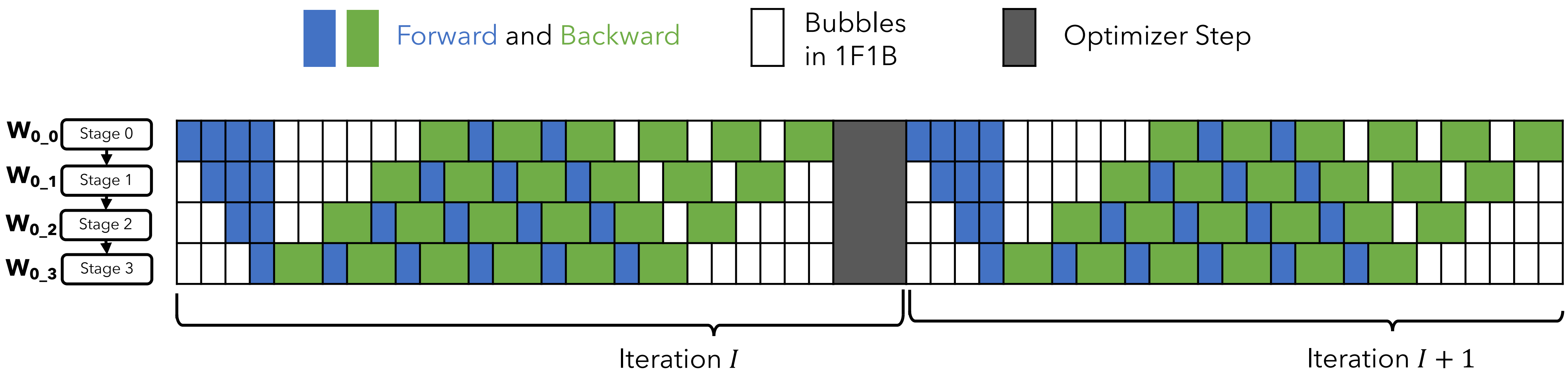
Adaptive Schedule with Decoupled Backprop when W_{1_2} fails

3 Data-Parallel Pipelines, 4 Pipeline Stages, 6 Micro batches per Data-Parallel Pipeline

But what about bubbles at the start!
How can we leverage them?

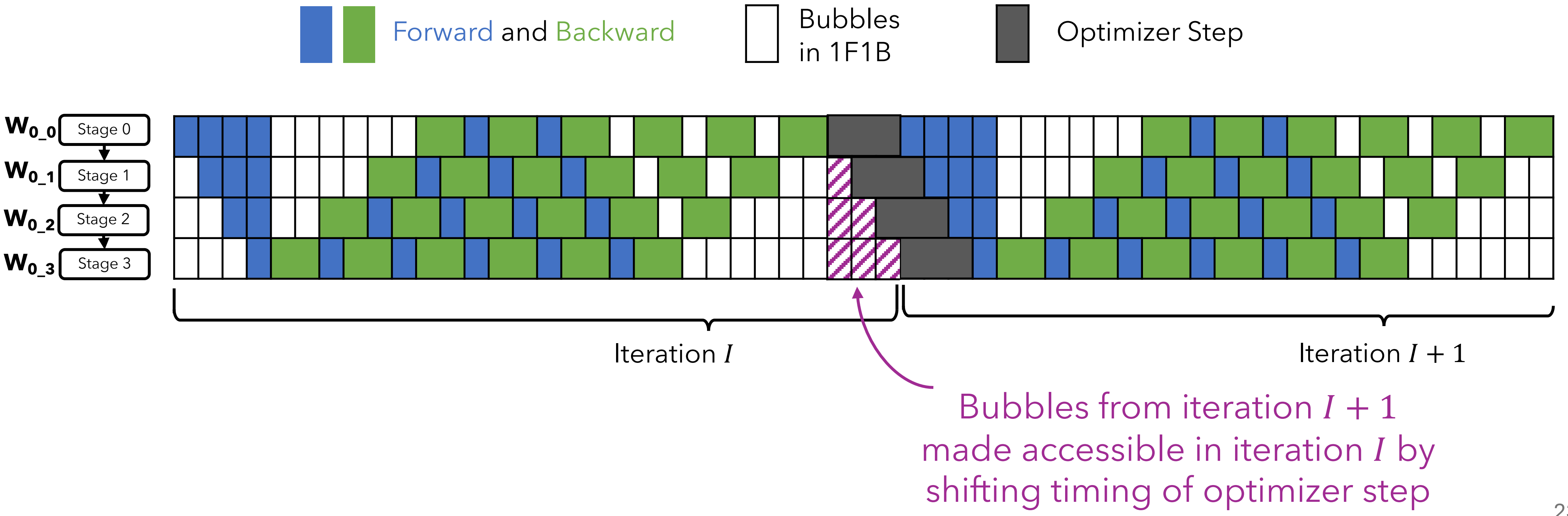
Staggered Optimizer: Accessing more bubbles

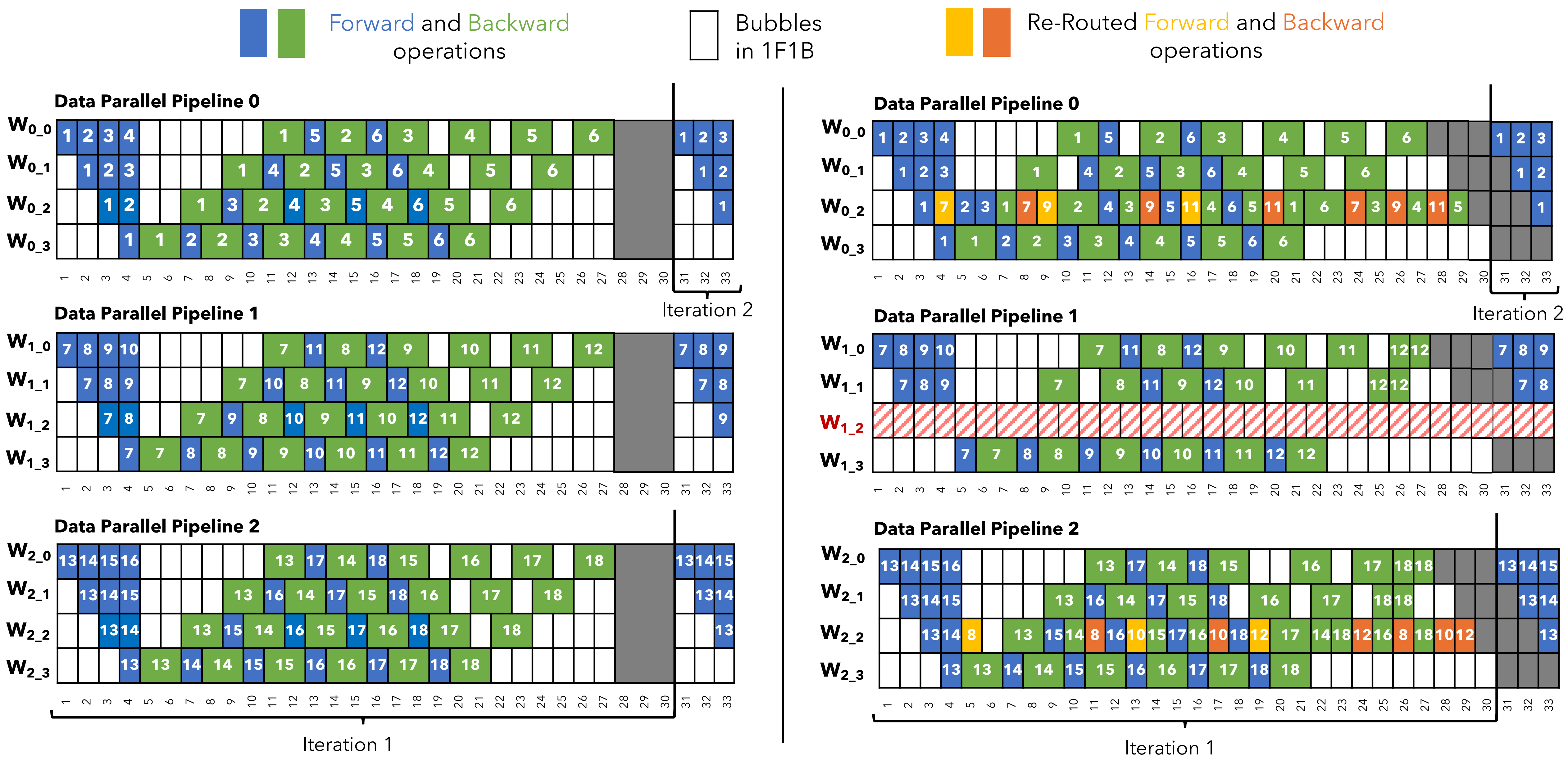
Optimizer step for different pipeline stages operate independently of one another, but are currently coupled together



Staggered Optimizer: Accessing more bubbles

We decouple them and adjust the timing of the optimizer step, shifting bubbles from next iteration's start into current iteration





Fault-Free Schedule

Adaptive Schedule + Decoupled Backprop + Staggered Optimizer when W_{1_2} fails

3 Data-Parallel Pipelines, 4 Pipeline Stages, 6 Micro batches per Data-Parallel Pipeline

Adaptive Pipelining

+

Decoupled BackProp

+

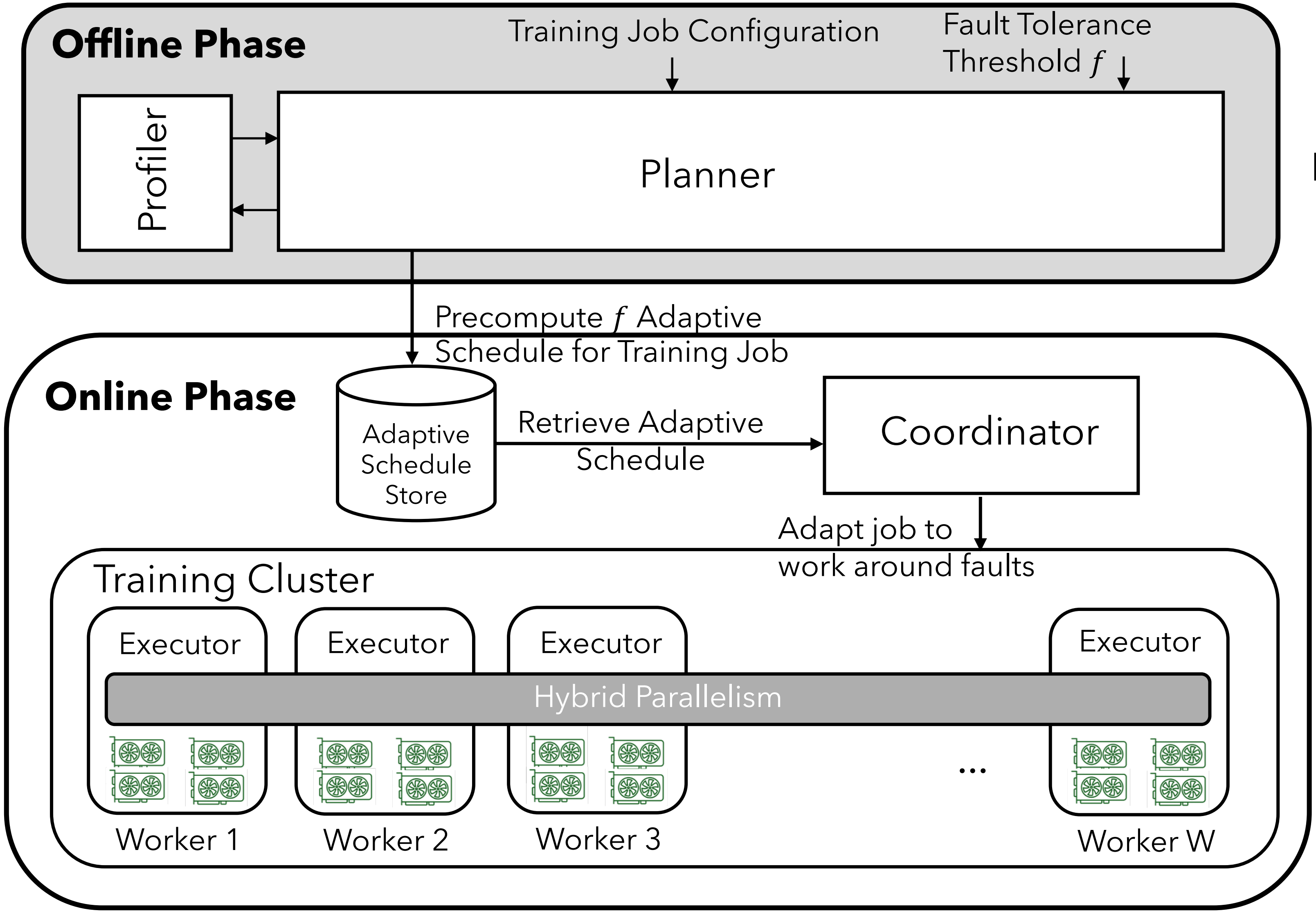
Staggered Optimizer

=

Zero Overhead

despite W_{1_2} failure

ReCycle Prototype



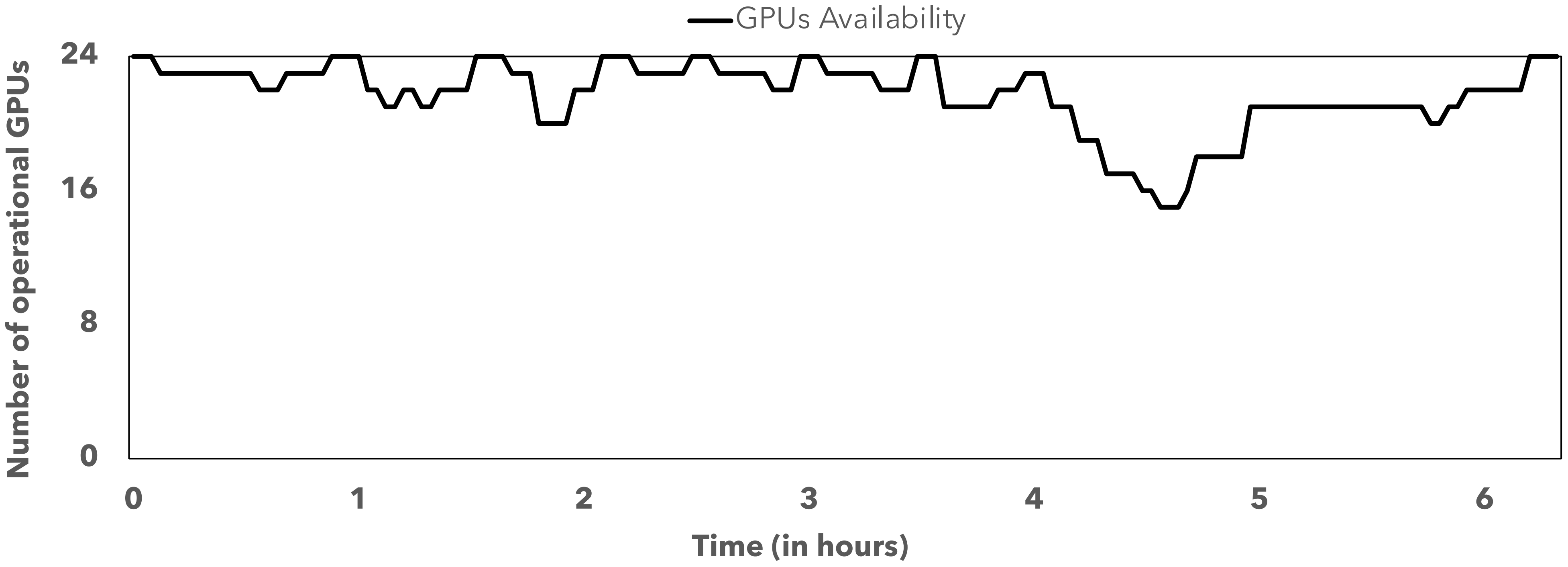
More Details above them in the paper

Uses Dynamic Programming and Mixed Integer Linear Programming to implement ReCycle Techniques

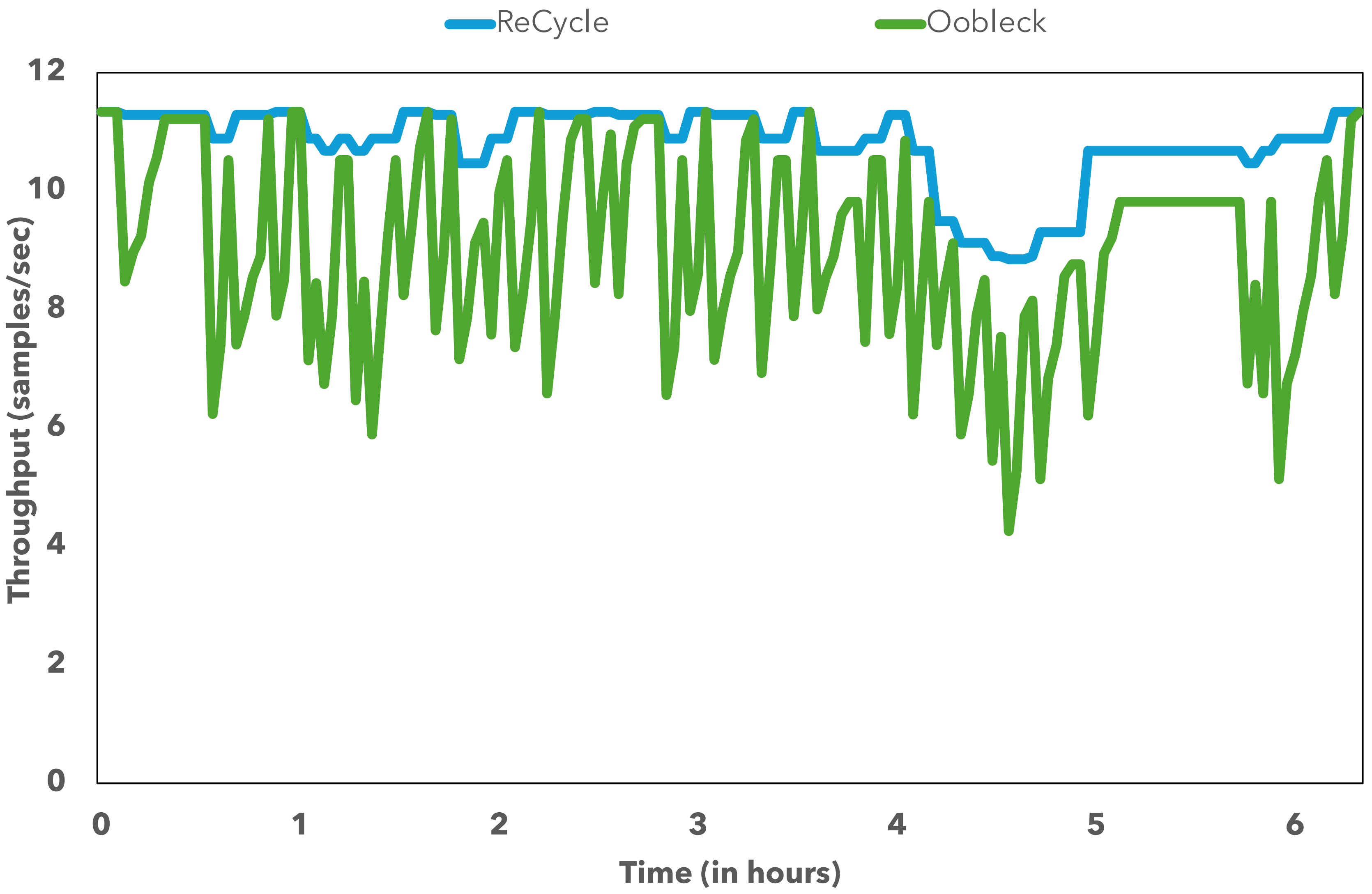
Coordinator retrieves precomputed adaptive schedule from store and instructs executors to follow new schedule

Evaluation

- Implemented on DeepSpeed + Megatron-LM
- Evaluated using 24 NVIDIA A100 GPUs connected via 80Gbps interconnect to train GPT-3 3.5B model using DP=6, PP=4, and TP=1

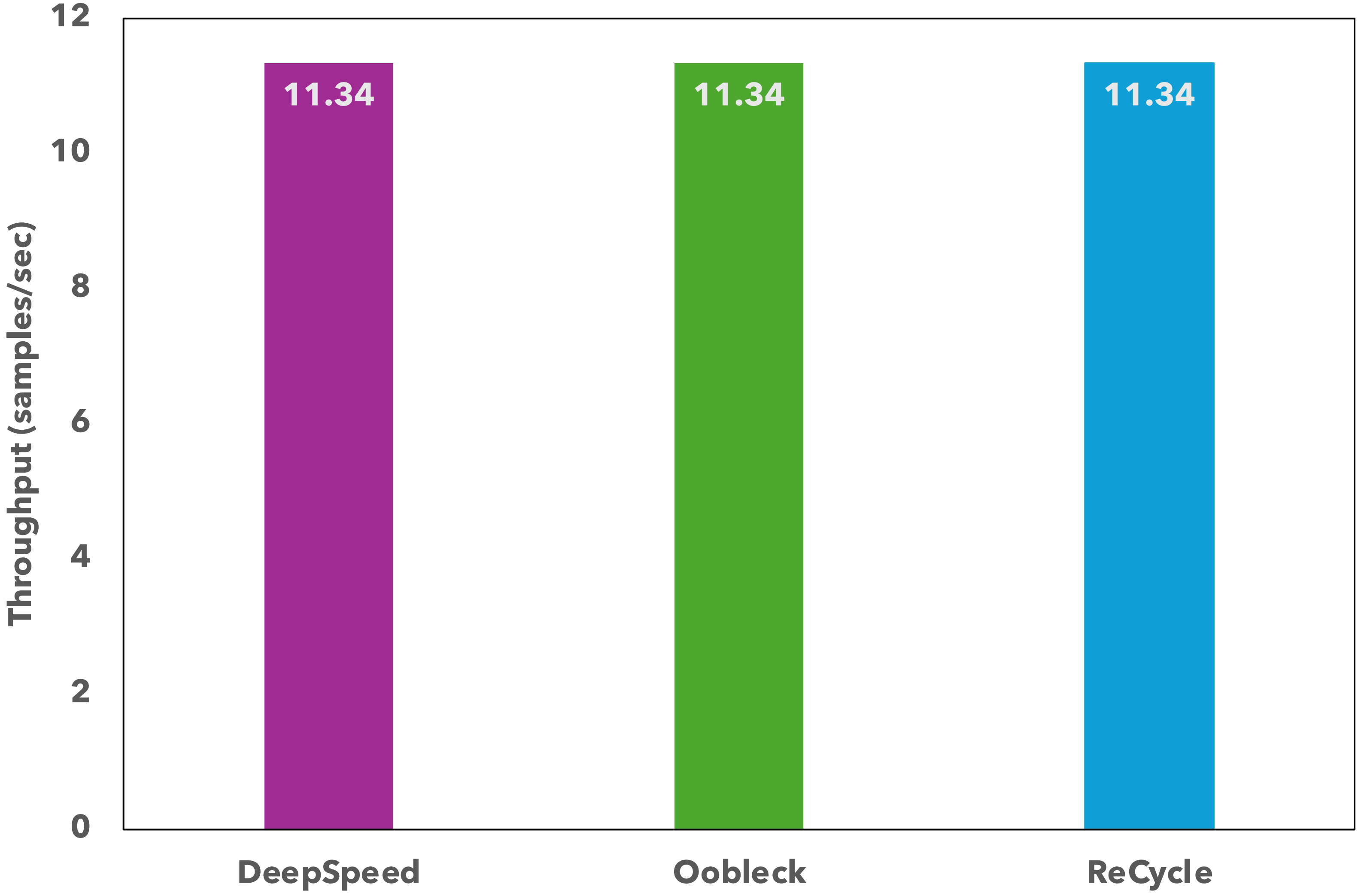


Comparison vs Oobleck [SOSP'23]



ReCycle and Oobleck ensure **stall-free training without relying on hot spares**

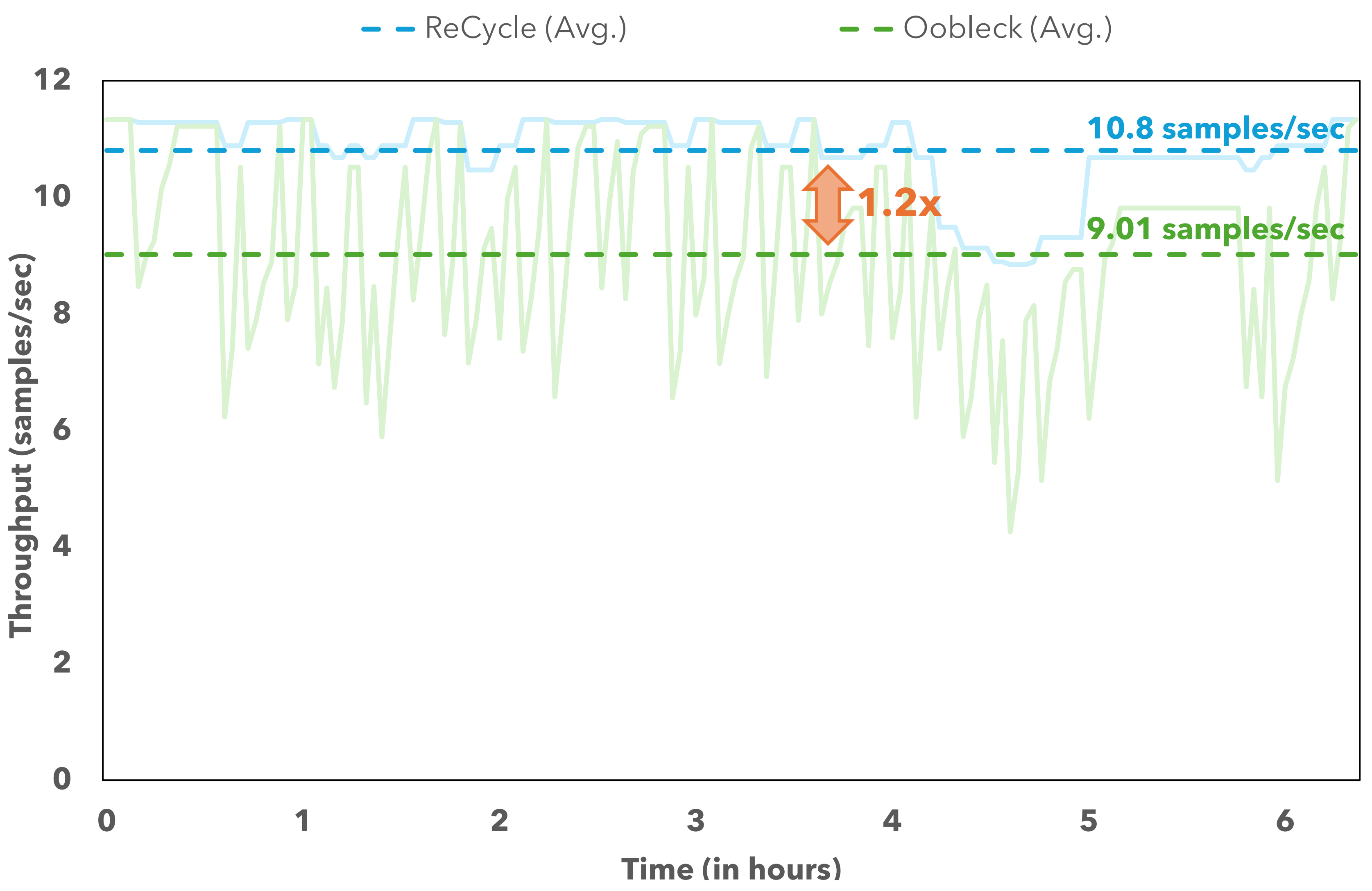
Comparison vs Oobleck [SOSP'23]



ReCycle and Oobleck ensure **stall-free training without relying on hot spares**

ReCycle and Oobleck introduce **no overhead in fault-free case**

Comparison vs Oobleck [SOSP'23]



ReCycle and Oobleck ensure **stall-free training without relying on hot spares**

ReCycle and Oobleck introduce **no overhead in fault-free case**

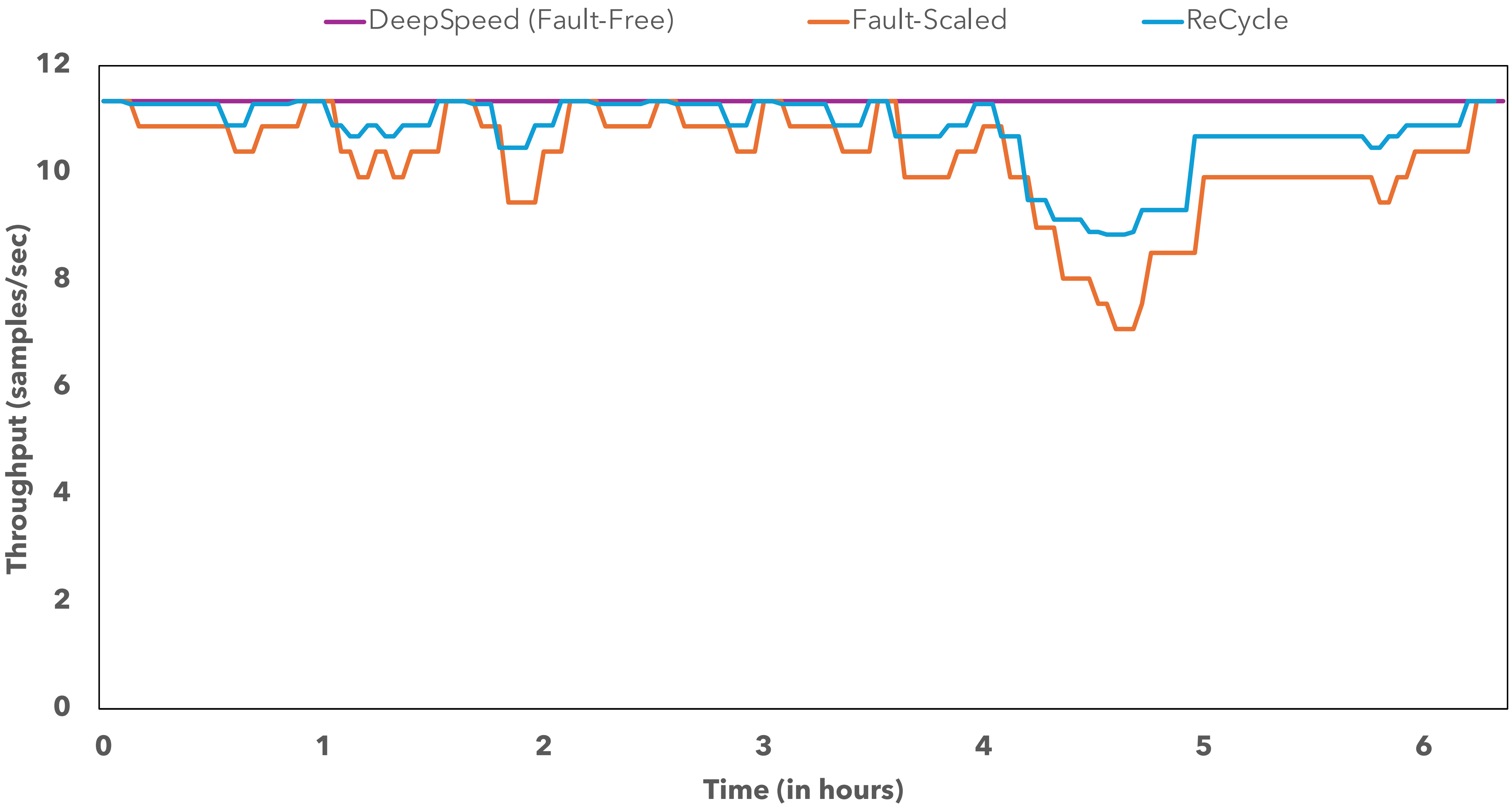
ReCycle delivers **1.2x higher throughput over Oobleck** due to reduced reconfiguration overhead

Comparison vs Fault-Scaled

$$\text{Fault-Scaled Throughput} = \text{Fault-Free Throughput} \times \frac{\text{Operational Resources}}{\text{Resources in Fault-Free Case}}$$

Extrapolates throughput as a linear function of operational resources

Comparison vs Fault-Scaled



ReCycle enables Performant and Resilient Distributed Training

Adaptive Pipelining reroutes computation from failed workers to functioning data-parallel peers, ensuring **stall-free training**

Decoupled BackProp and Staggered Optimizer exploits pipeline bubbles to maintain **high training throughput** in presence of failures

ReCycle maintains **synchronous training semantics**, ensuring model convergence is unaffected

gandhis@stanford.edu