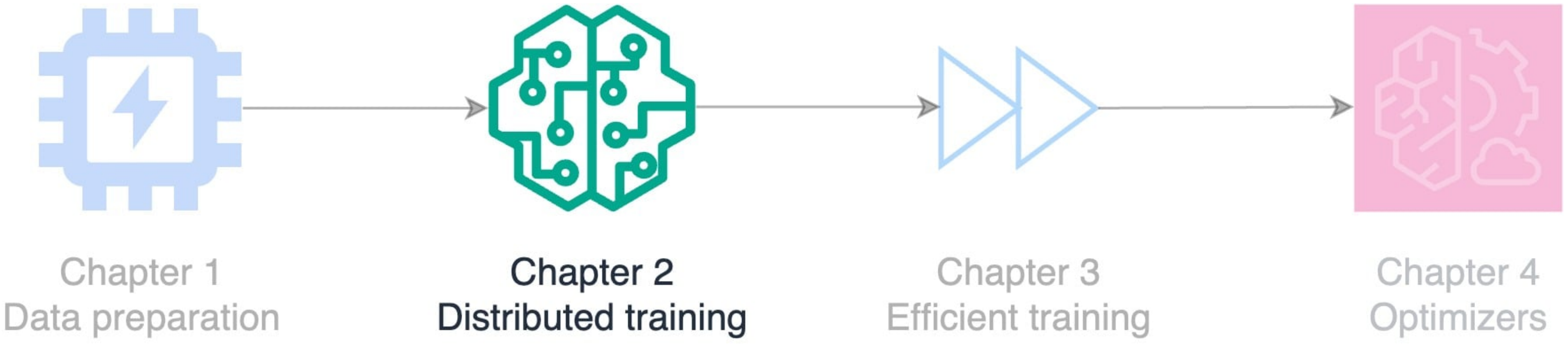
Gradient accumulation

EFFICIENT AI MODEL TRAINING WITH PYTORCH

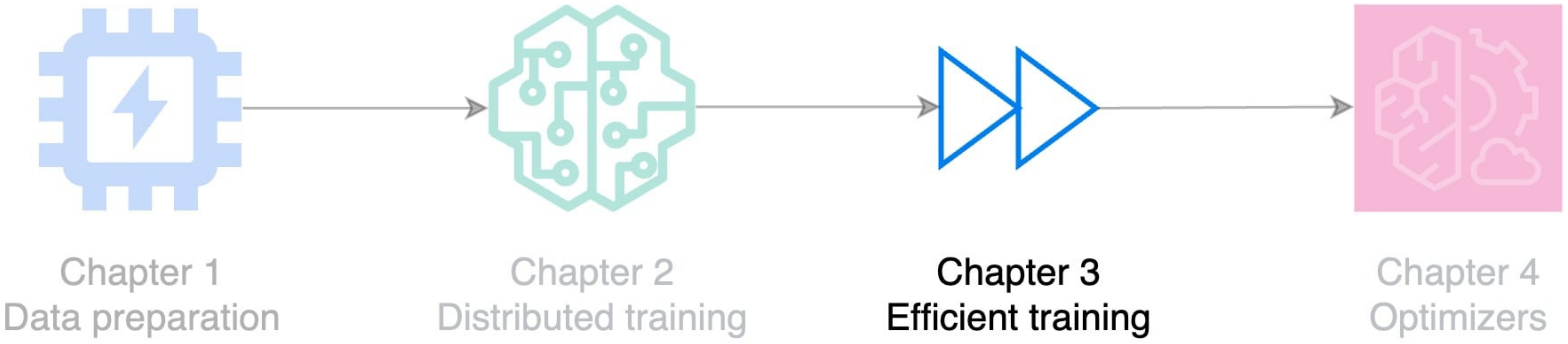
Dennis Lee
Data Engineer

datacamp

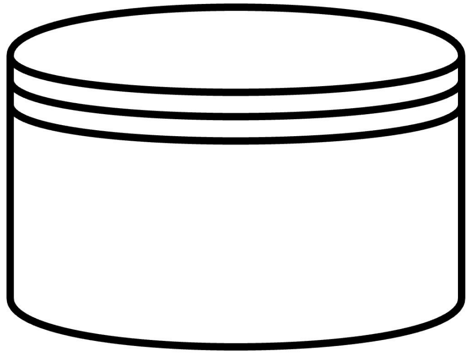
Distributed training



Efficient training



Improving training efficiency

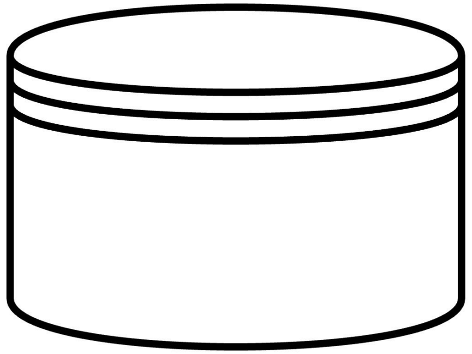


Memory Efficiency

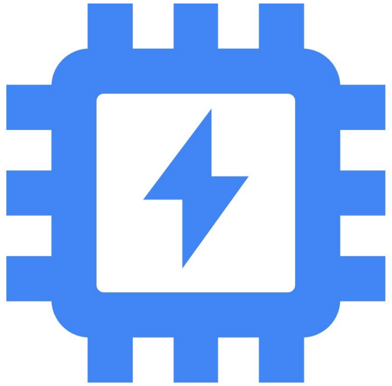
Communication Efficiency

Computational Efficiency

Improving training efficiency



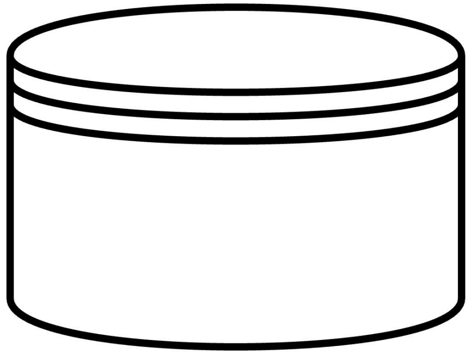
Memory Efficiency



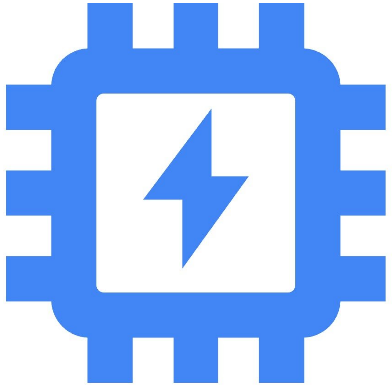
Communication Efficiency

Computational Efficiency

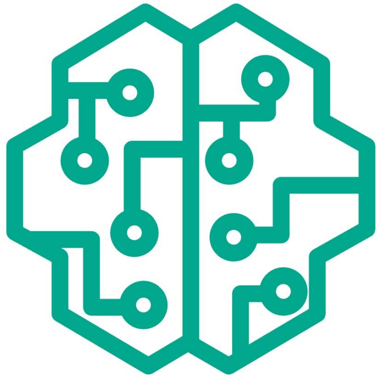
Improving training efficiency



Memory Efficiency

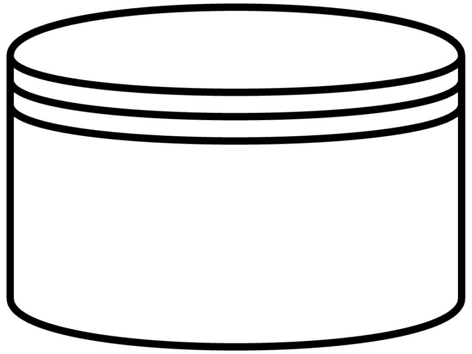


Communication Efficiency



Computational Efficiency

Gradient accumulation improves memory efficiency



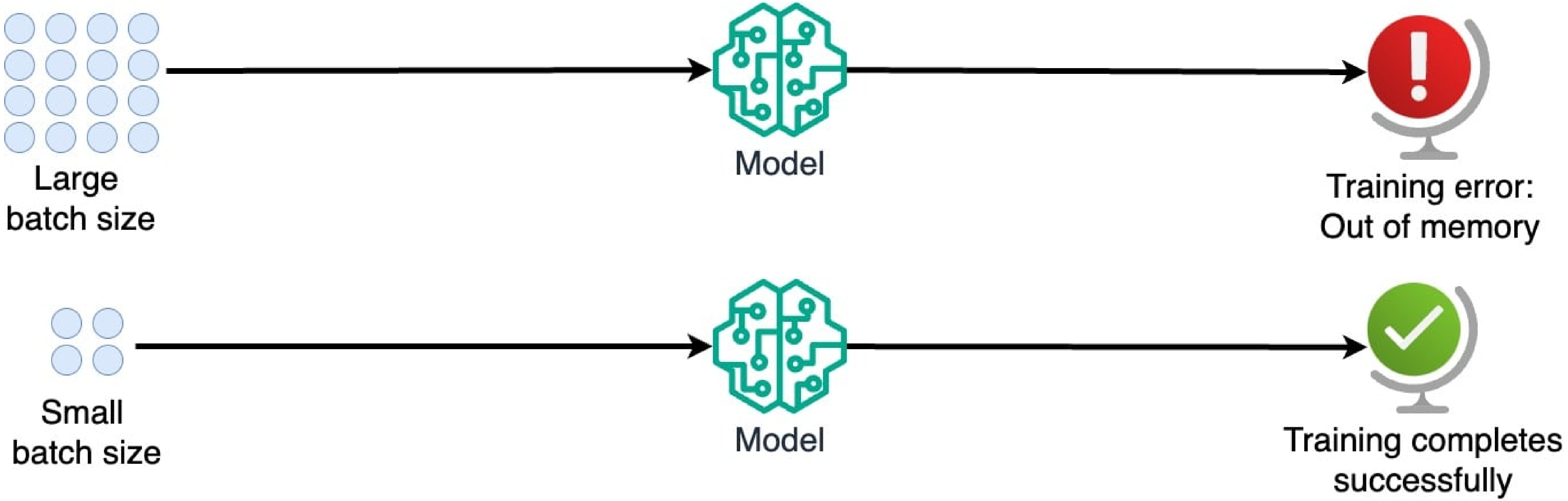
Memory Efficiency

Communication Efficiency

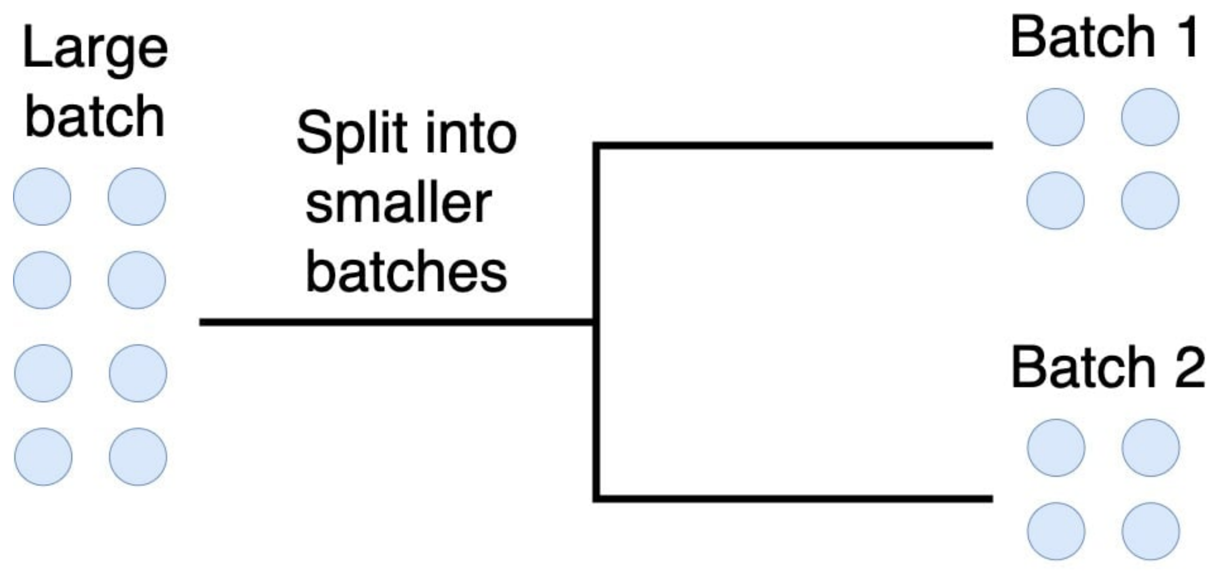
Computational Efficiency

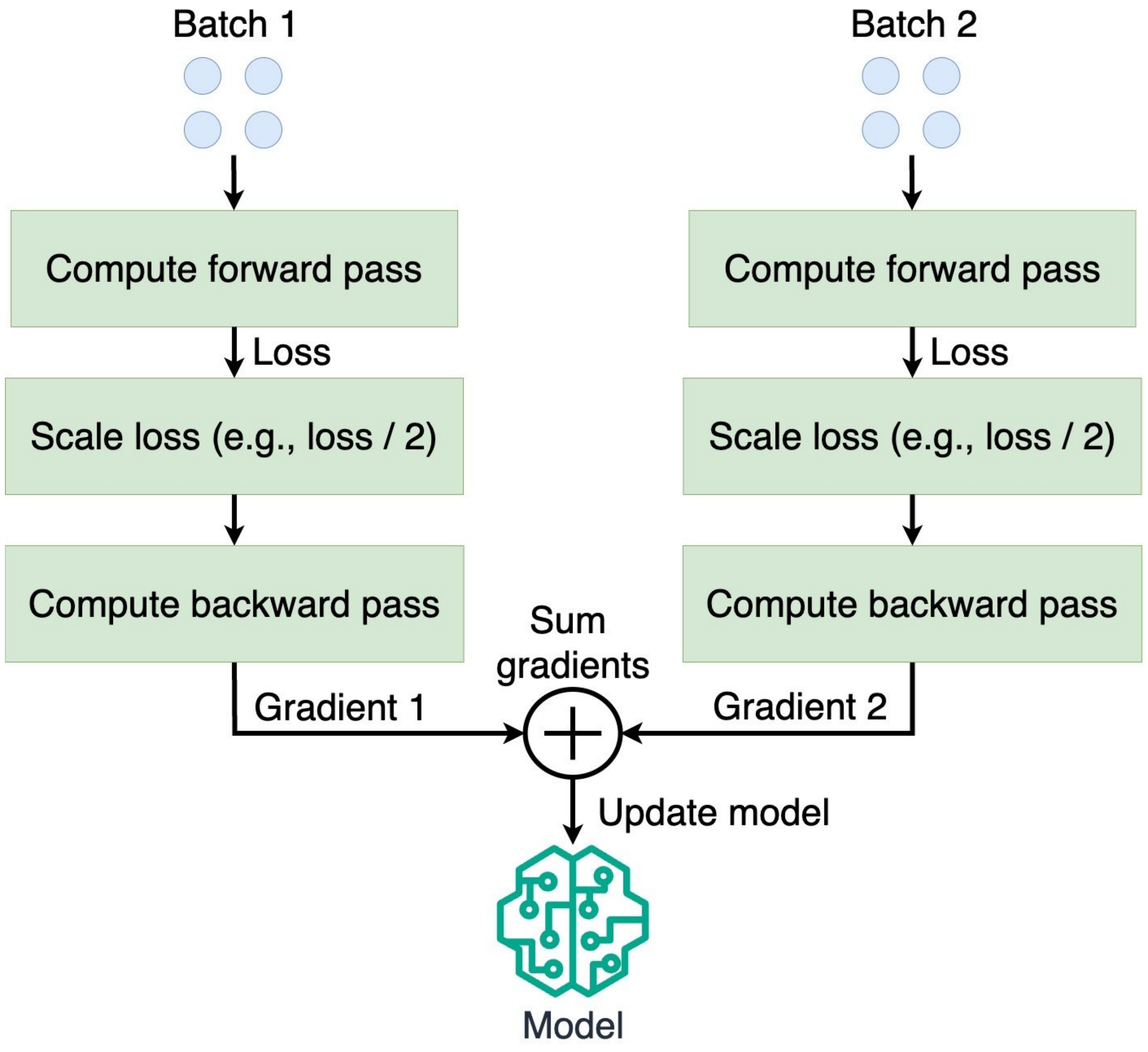
The problem with large batch sizes

- Large batch sizes: Robust gradient estimates for quicker learning
- GPU memory constrains batch sizes



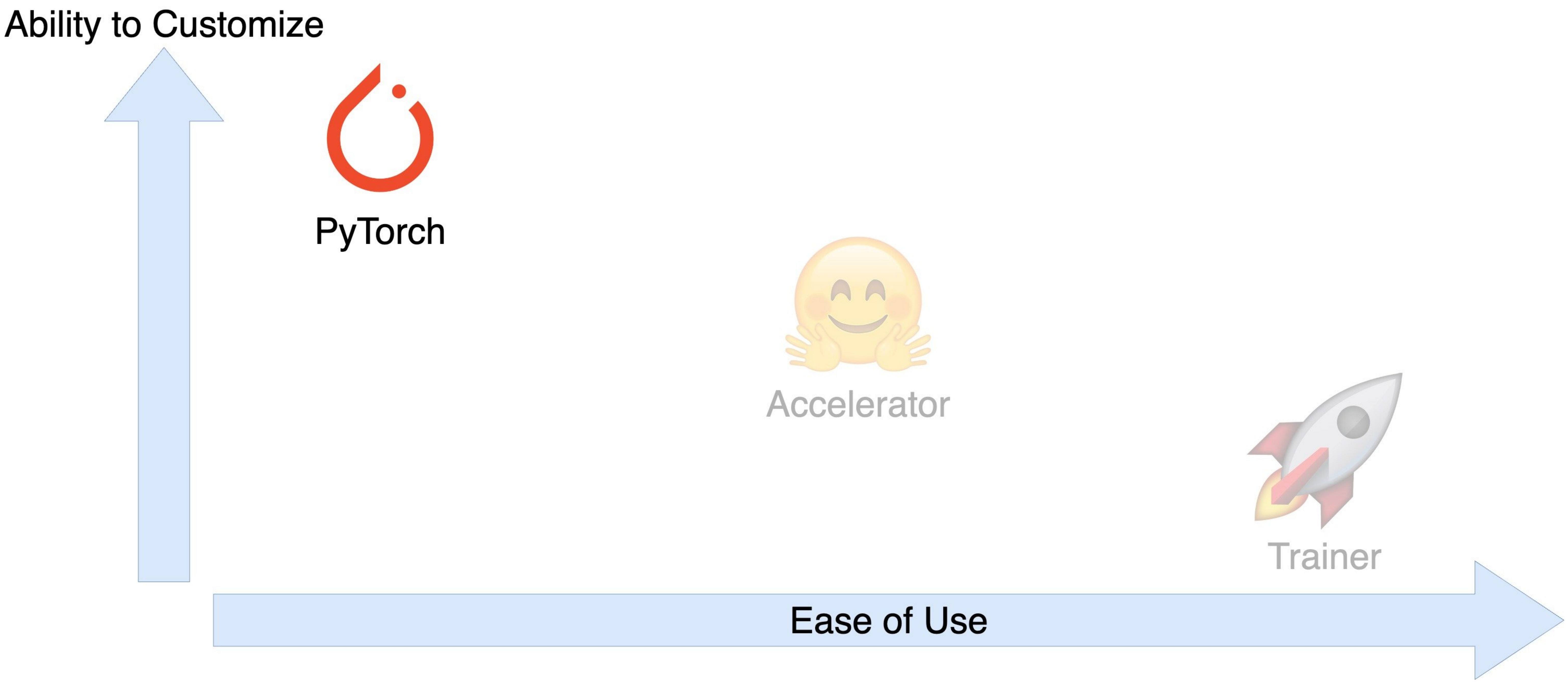
How does gradient accumulation work?



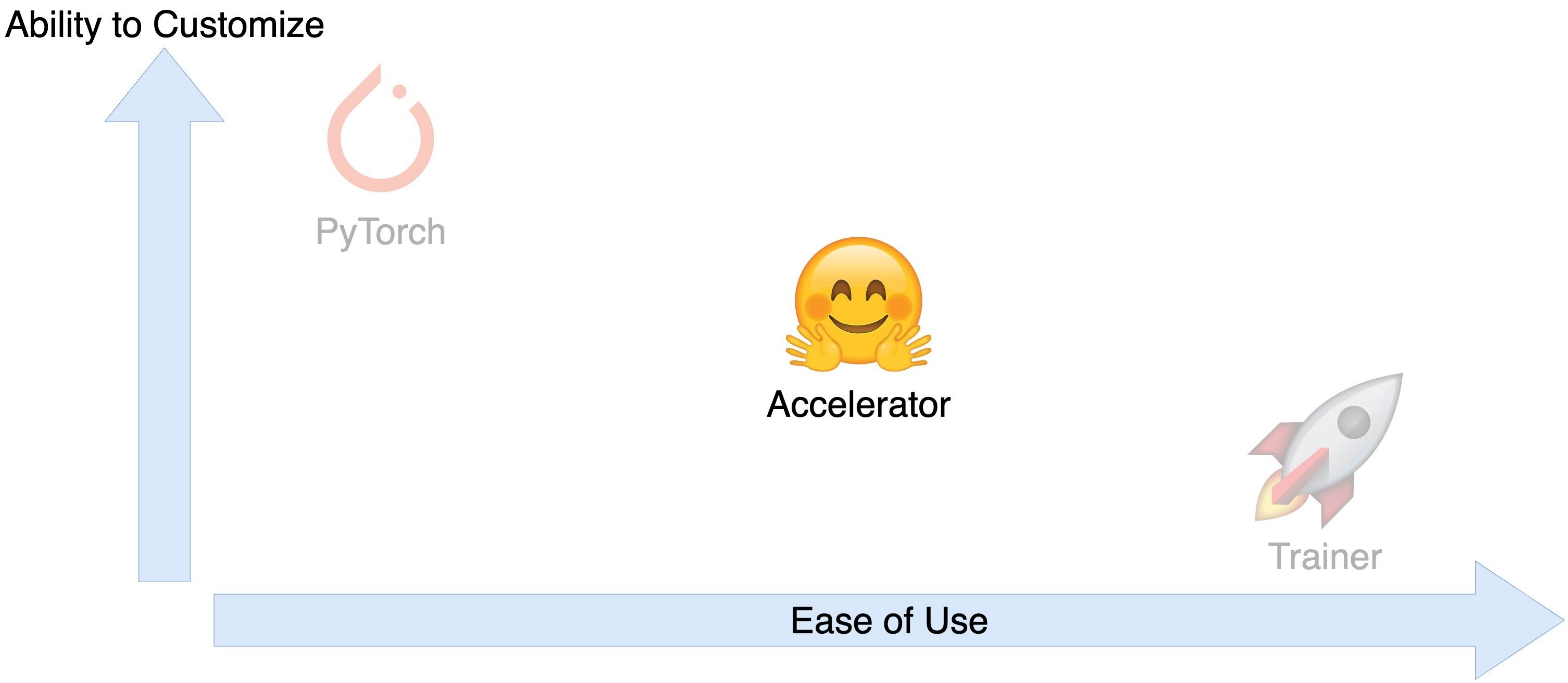


- Gradient accumulation: Sum gradients over smaller batches
- Effectively train the model on a large batch
- Update model parameters after summing gradients

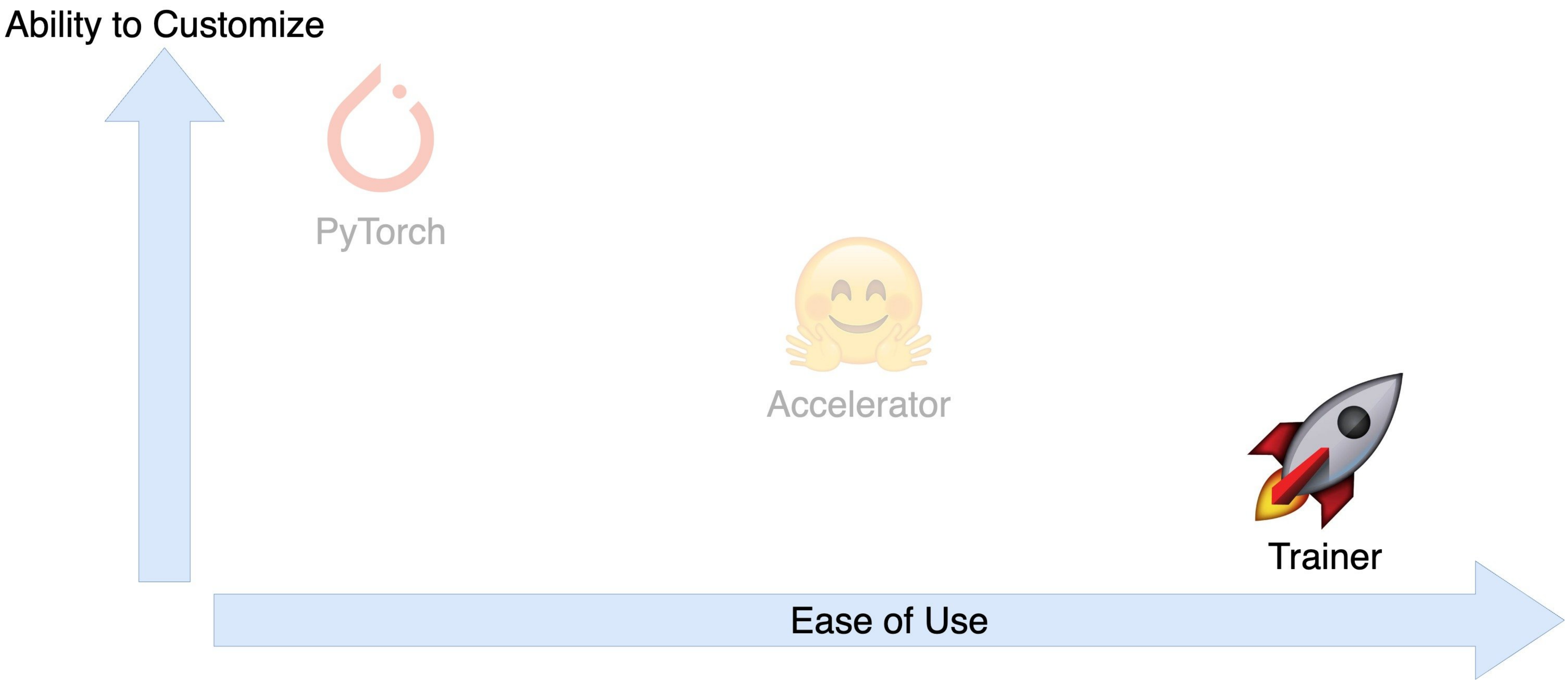
PyTorch, Accelerator, and Trainer



PyTorch, Accelerator, and Trainer

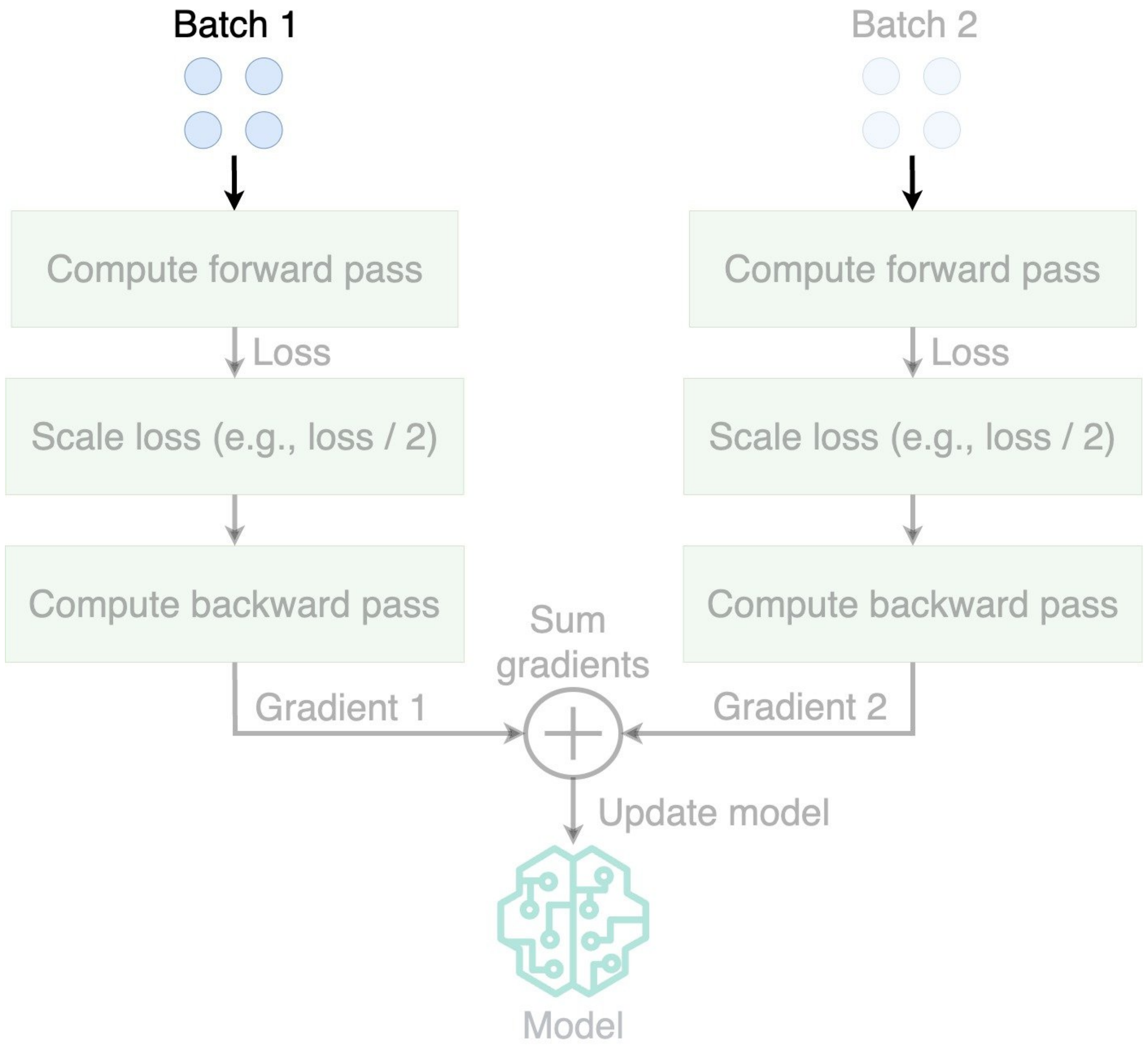


PyTorch, Accelerator, and Trainer



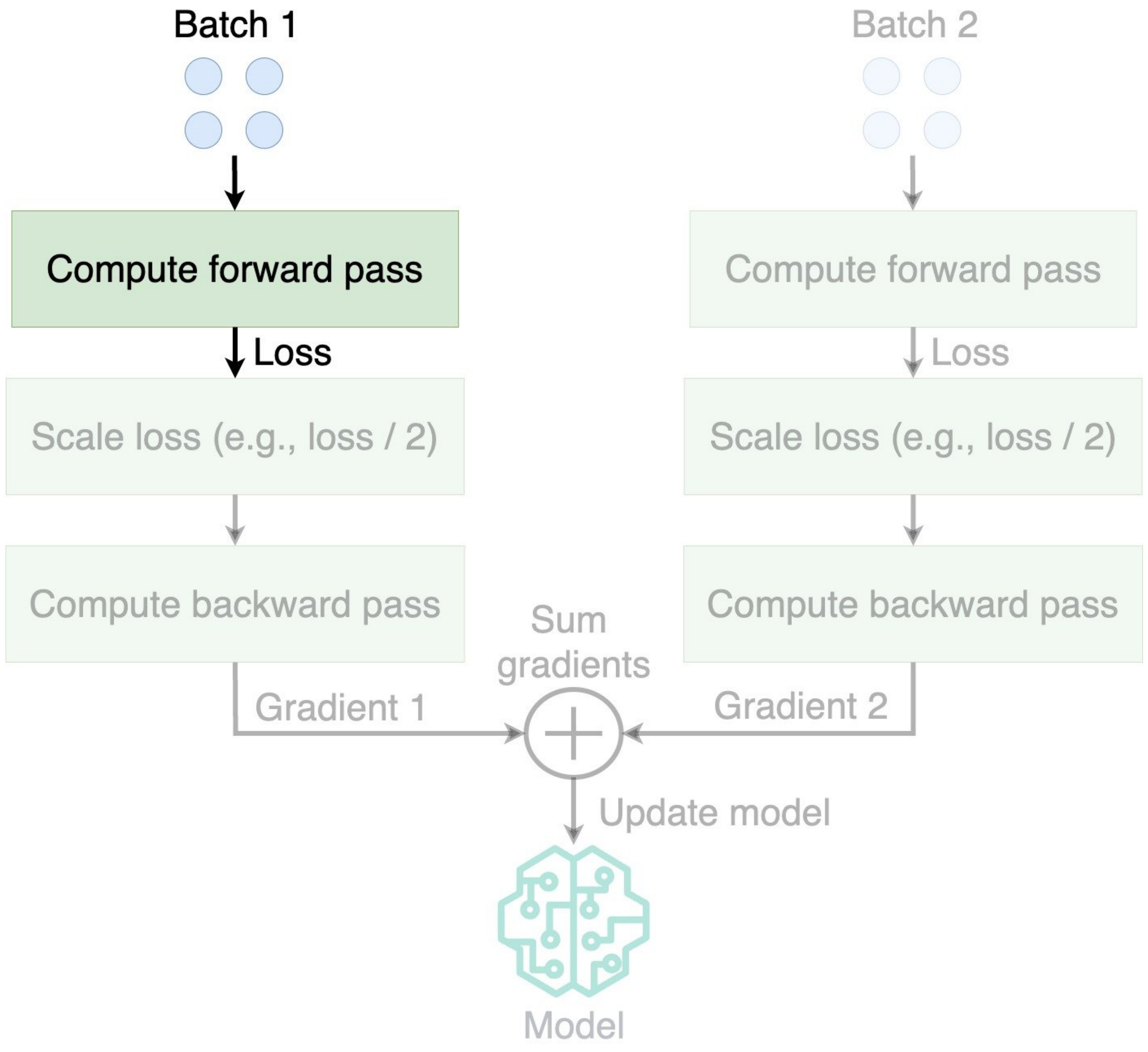
Gradient accumulation with PyTorch

```
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
```



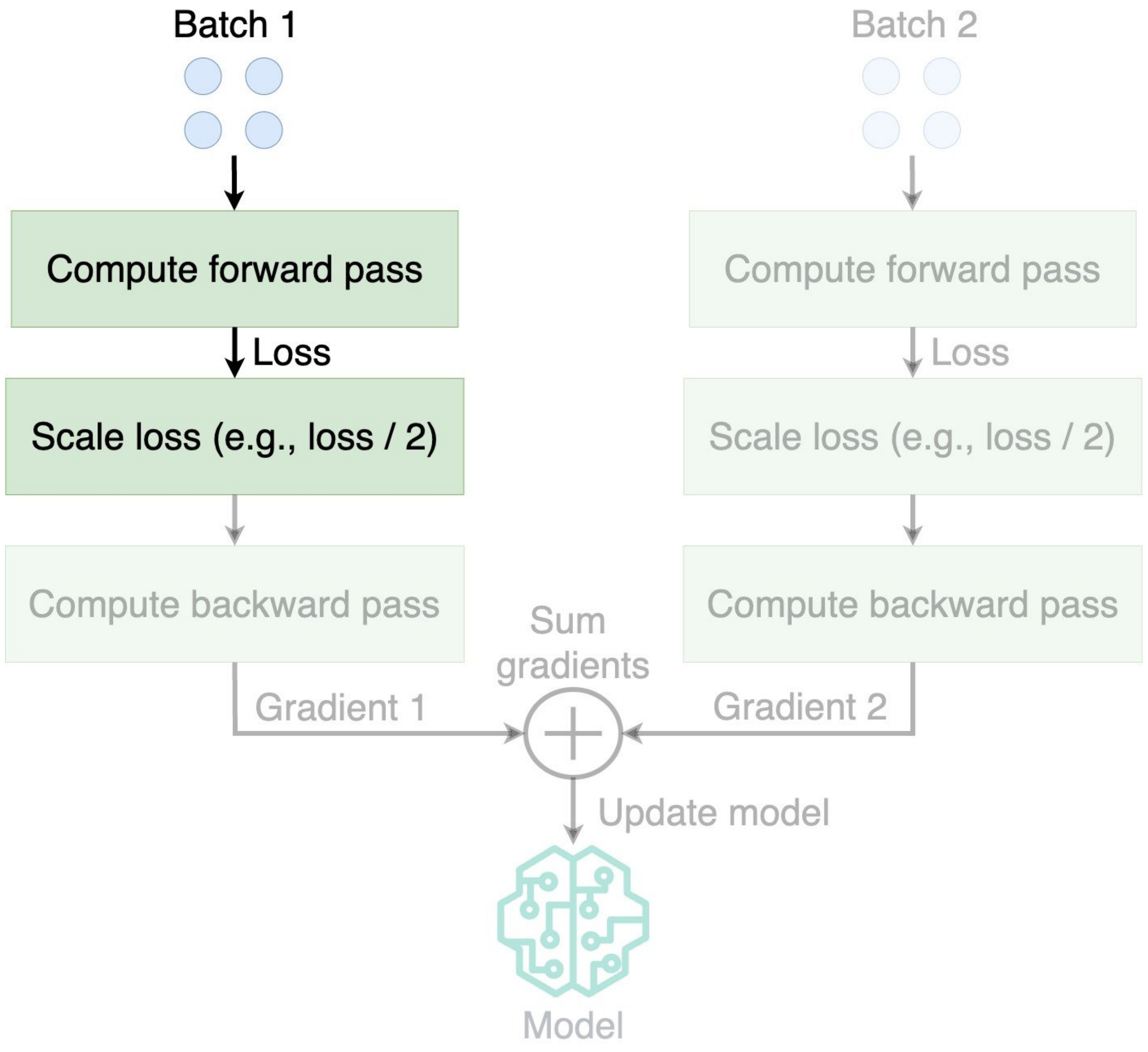
Gradient accumulation with PyTorch

```python
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
```



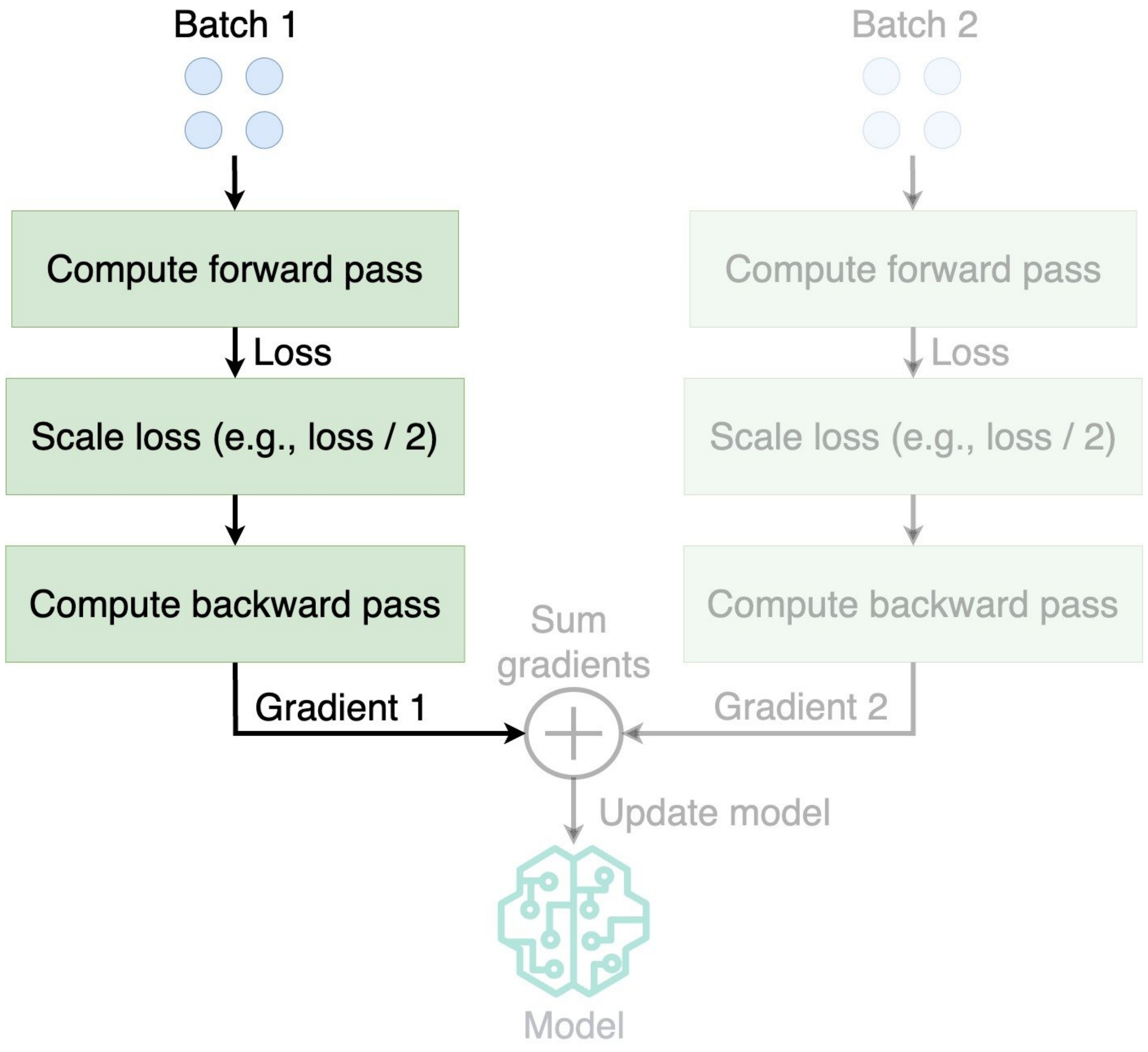
Gradient accumulation with PyTorch

```
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
    loss = loss / gradient_accumulation_steps
```



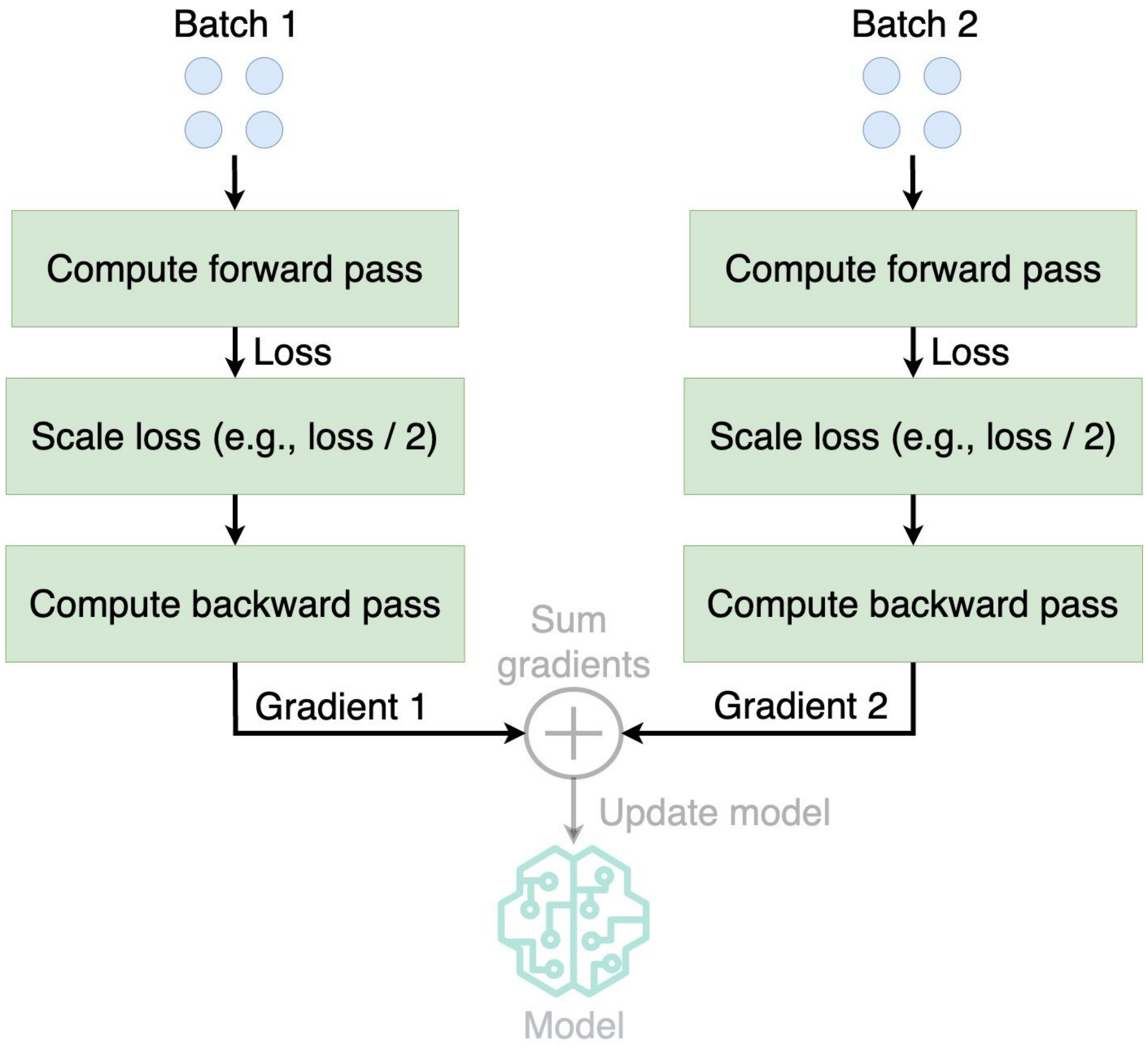
Gradient accumulation with PyTorch

```
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
    loss = loss / gradient_accumulation_steps
    loss.backward()
```



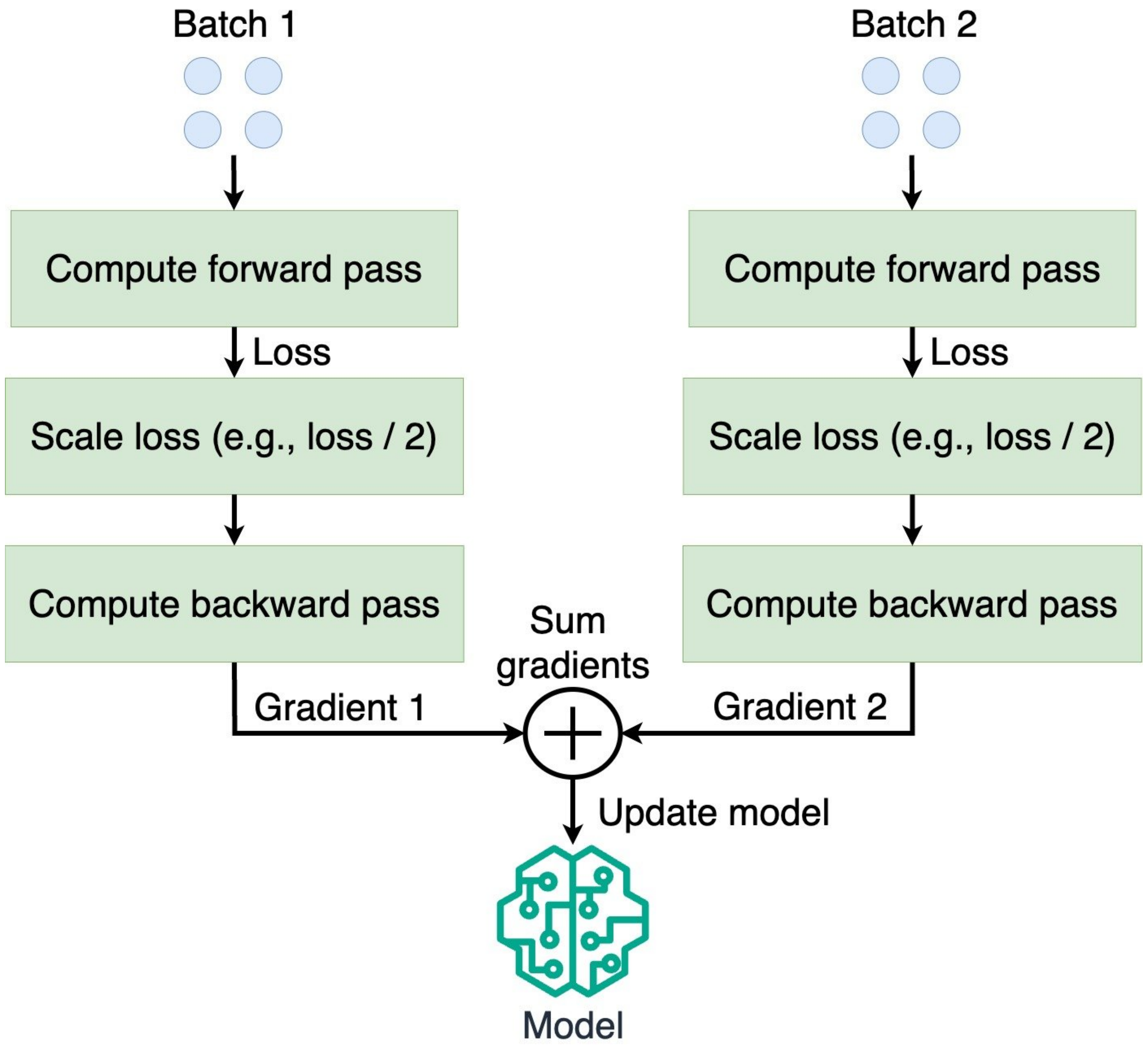
Gradient accumulation with PyTorch

```
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
    loss = loss / gradient_accumulation_steps
    loss.backward()
    if ((index + 1)
        % gradient_accumulation_steps == 0):
```

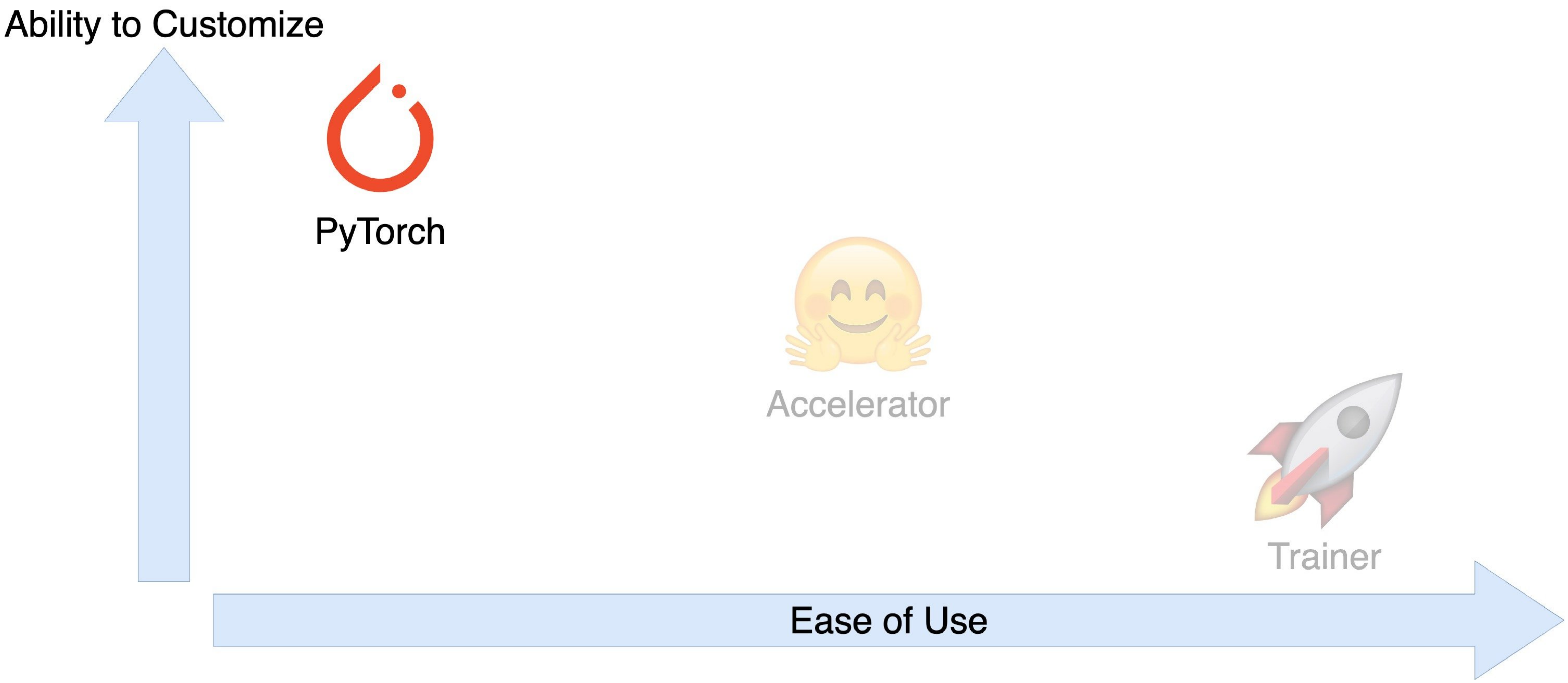


Gradient accumulation with PyTorch

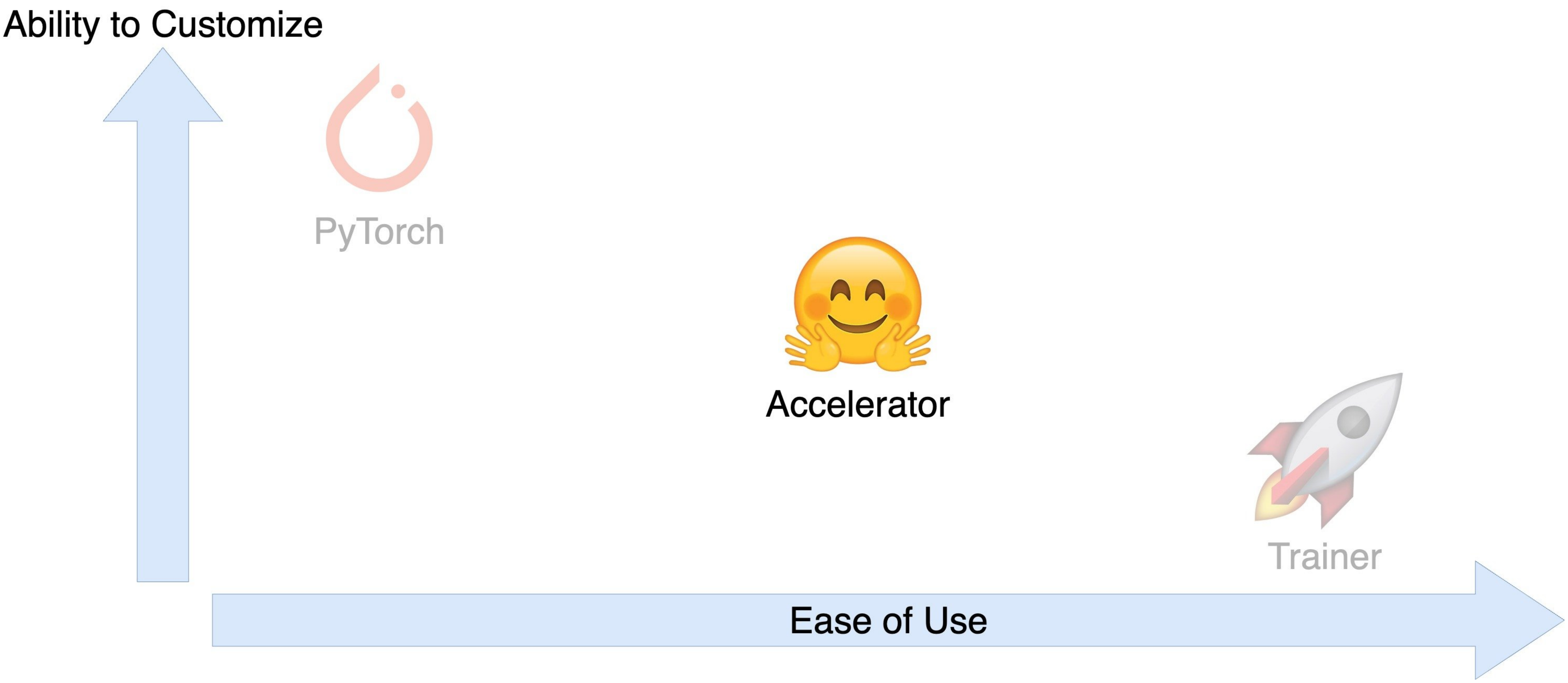
```
for index, batch in enumerate(dataloader):
    inputs, targets = (batch["input_ids"],
                       batch["labels"])
    inputs, targets = (inputs.to(device),
                       targets.to(device))
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
    loss = loss / gradient_accumulation_steps
    loss.backward()
    if ((index + 1)
        % gradient_accumulation_steps == 0):
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```



From PyTorch to Accelerator

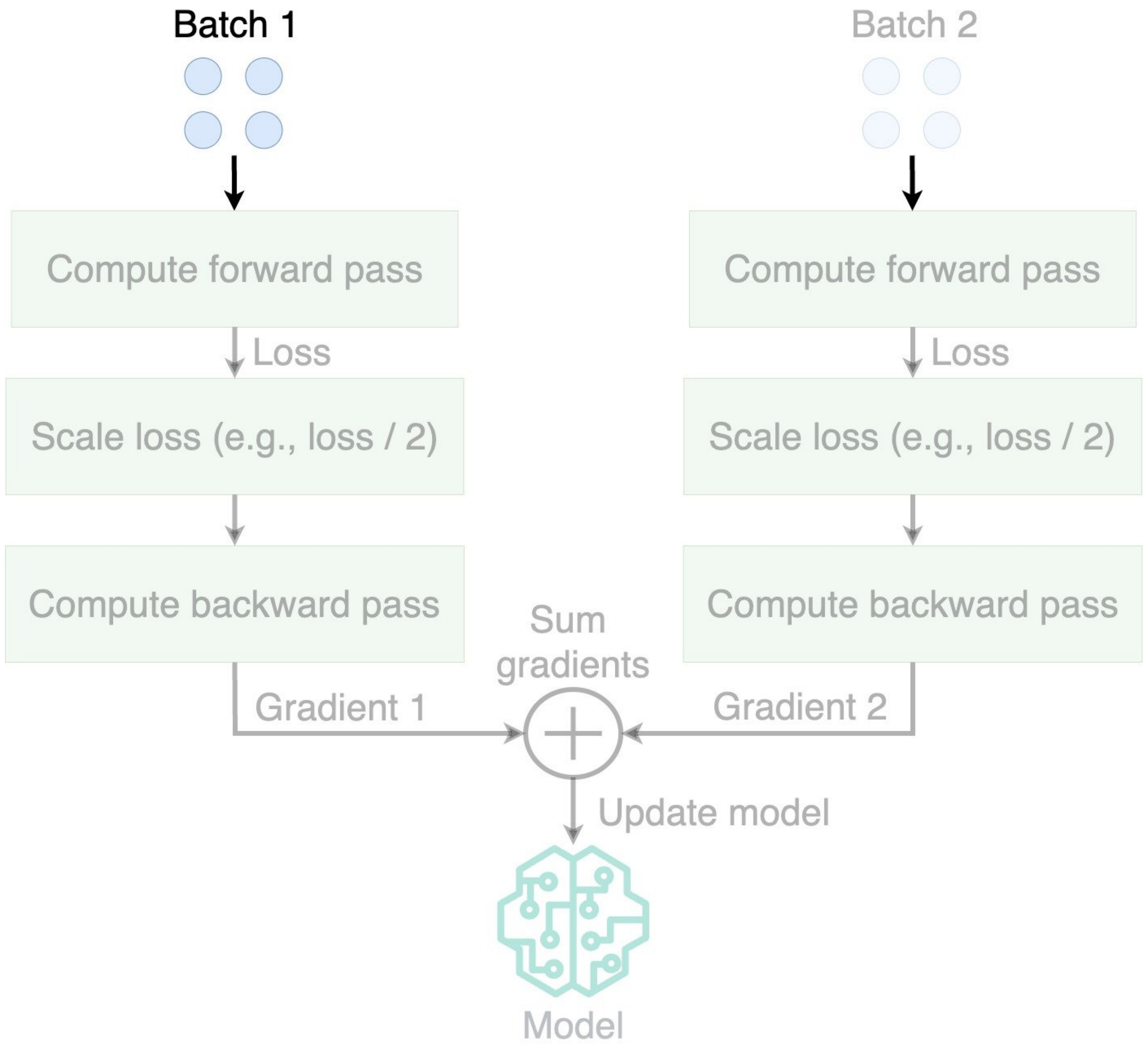


From PyTorch to Accelerator



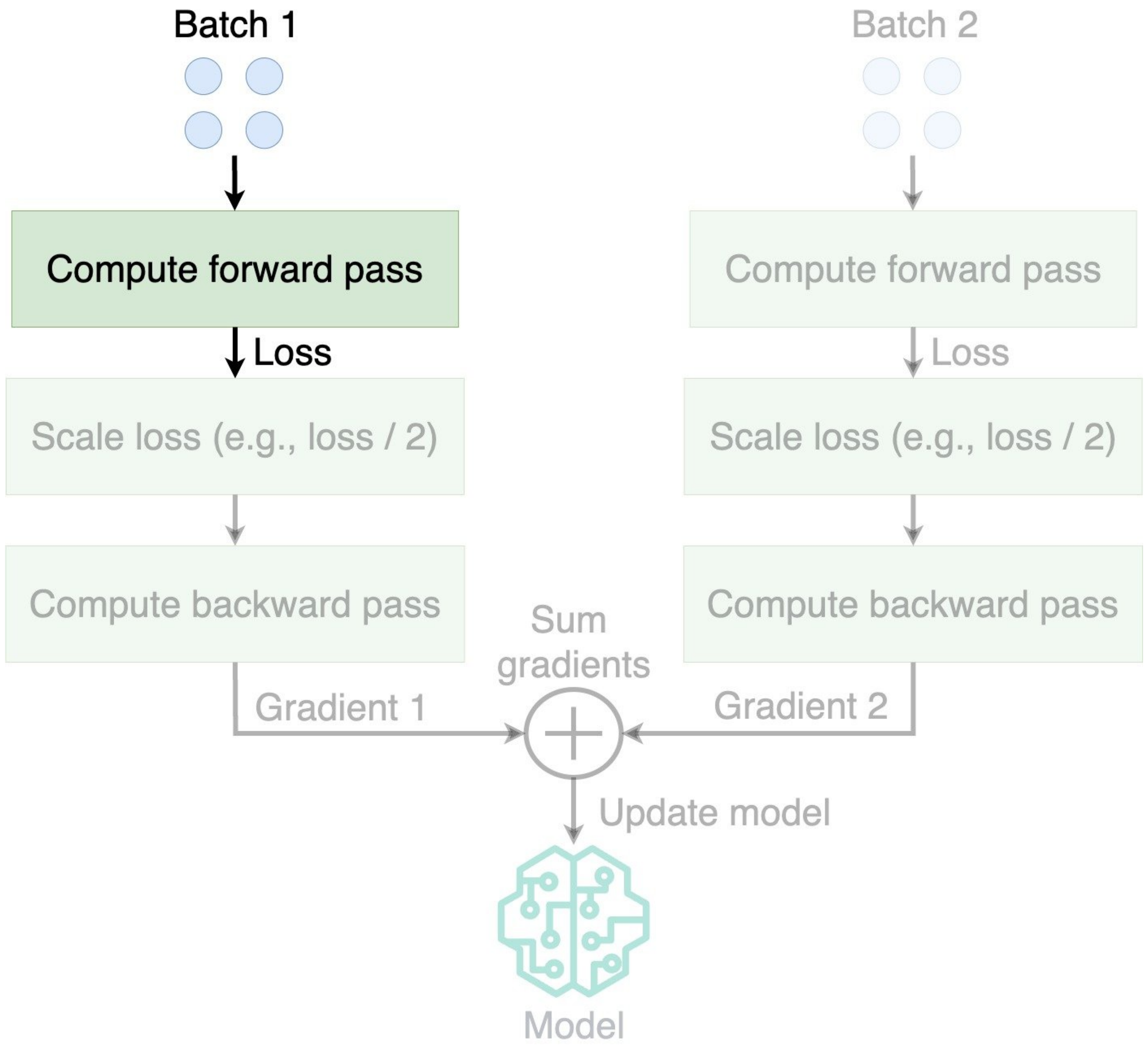
Gradient accumulation with Accelerator

```
accelerator = \
    Accelerator(gradient_accumulation_steps=2)
for index, batch in enumerate(dataloader):

        inputs, targets = (batch["input_ids"],
                           batch["labels"])
```



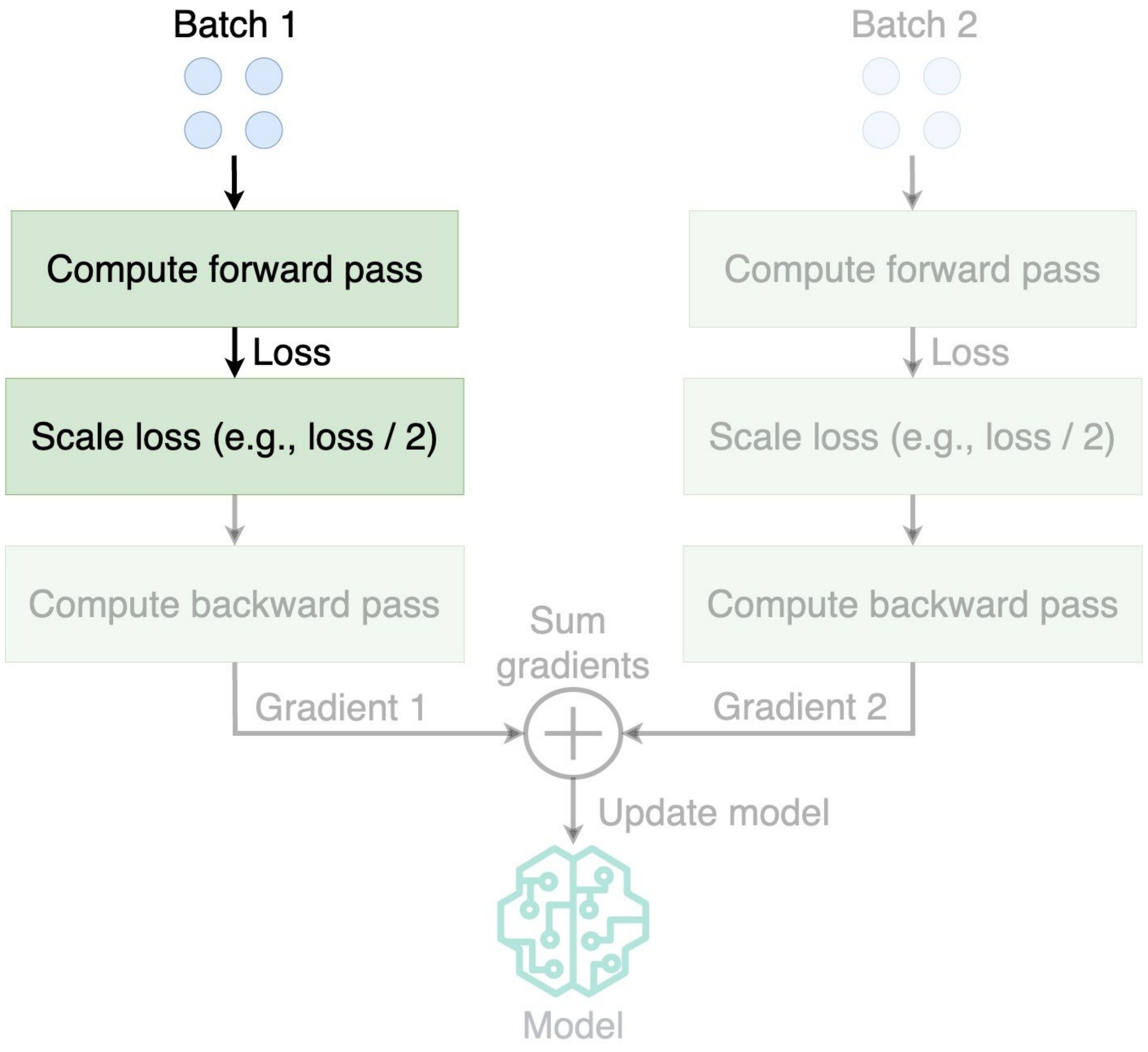
Gradient accumulation with Accelerator

```python
accelerator = \
    Accelerator(gradient_accumulation_steps=2)

for index, batch in enumerate(dataloader):

        inputs, targets = (batch["input_ids"],
                           batch["labels"])
        outputs = model(inputs,
                        labels=targets)
        loss = outputs.loss
```



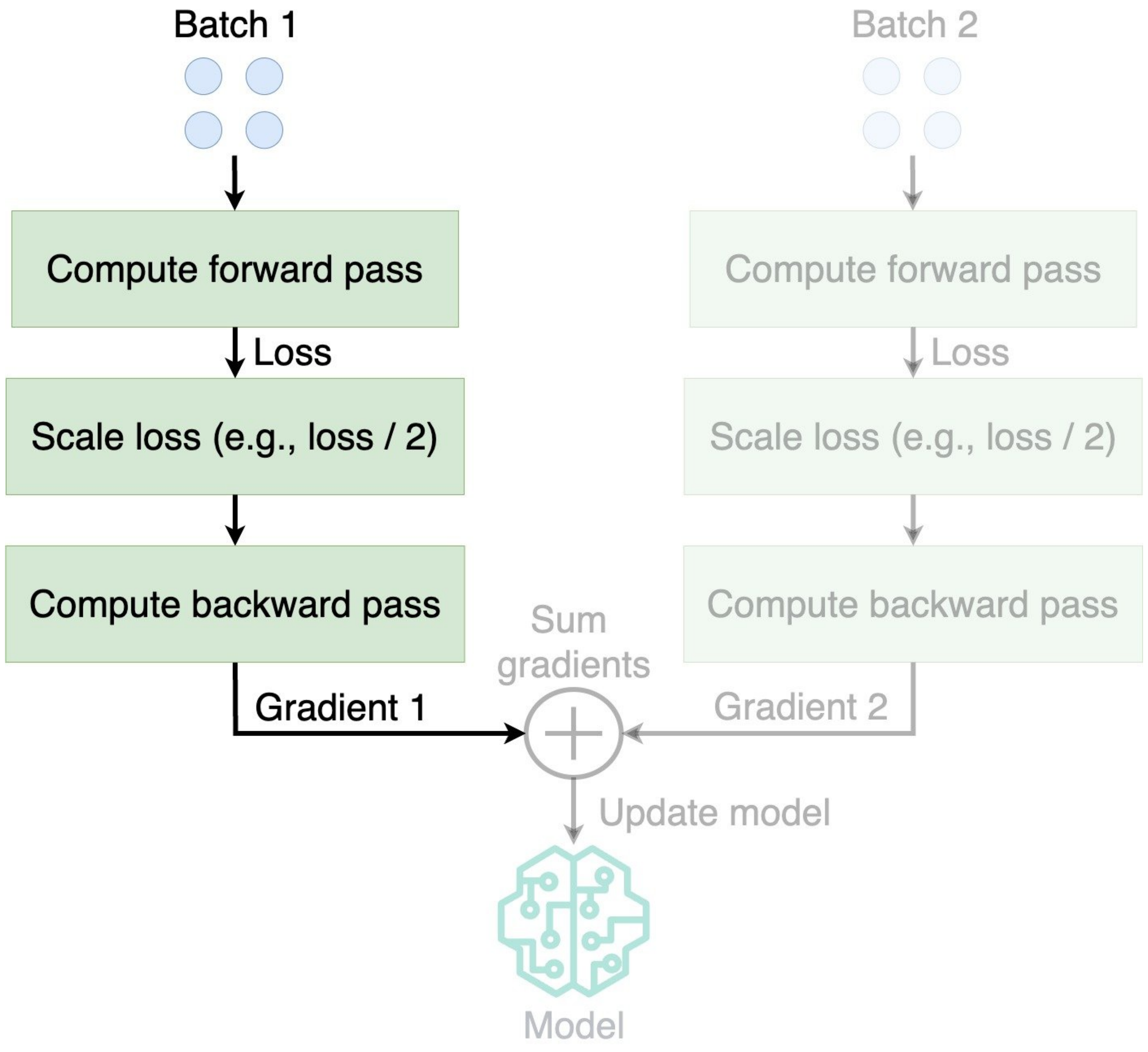
Gradient accumulation with Accelerator

```
accelerator = \
    Accelerator(gradient_accumulation_steps=2)

for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = (batch["input_ids"],
                           batch["labels"])
        outputs = model(inputs,
                        labels=targets)
        loss = outputs.loss
```



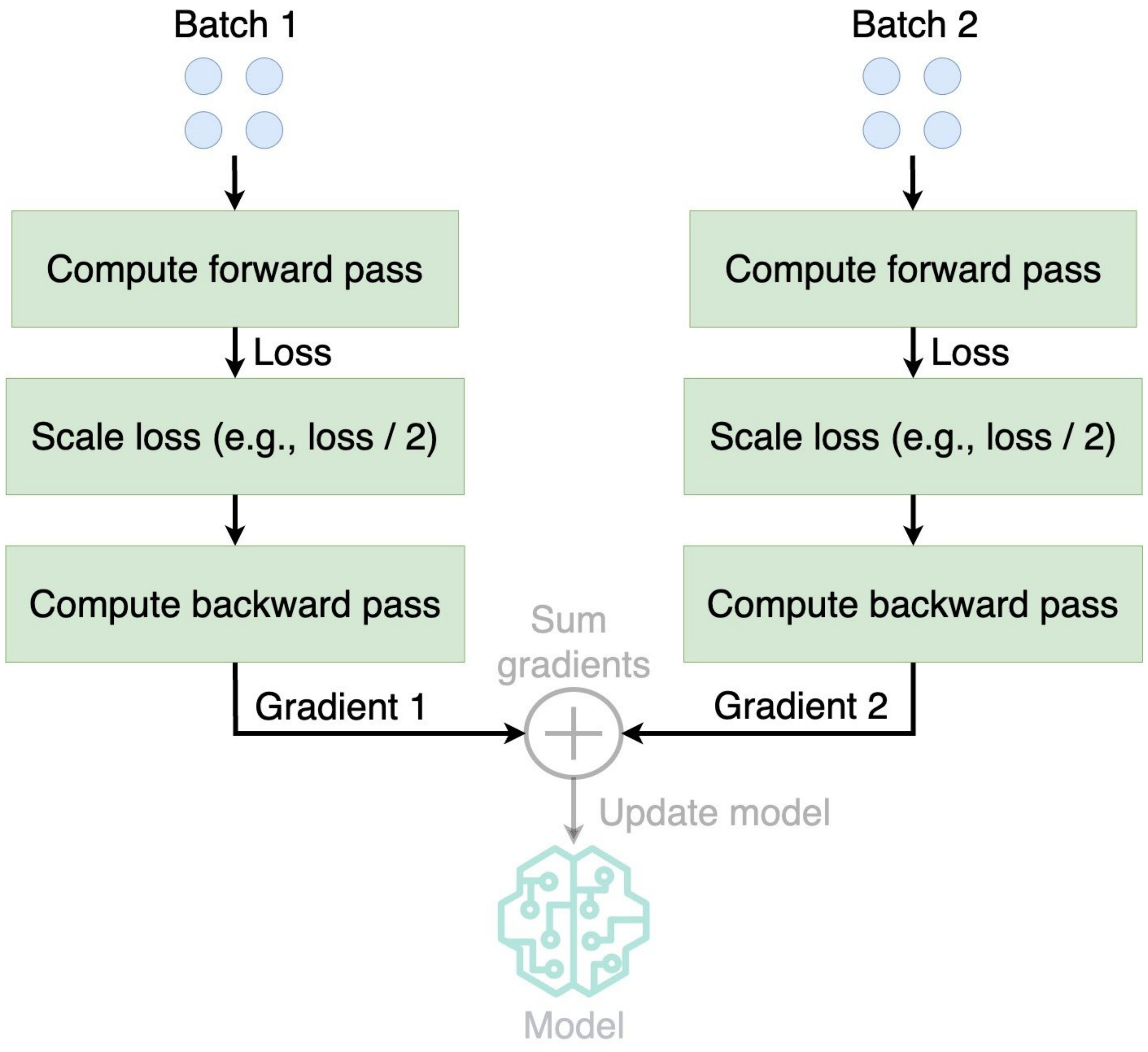
Gradient accumulation with Accelerator

```
accelerator = \
    Accelerator(gradient_accumulation_steps=2)

for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = (batch["input_ids"],
                           batch["labels"])
        outputs = model(inputs,
                        labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
```



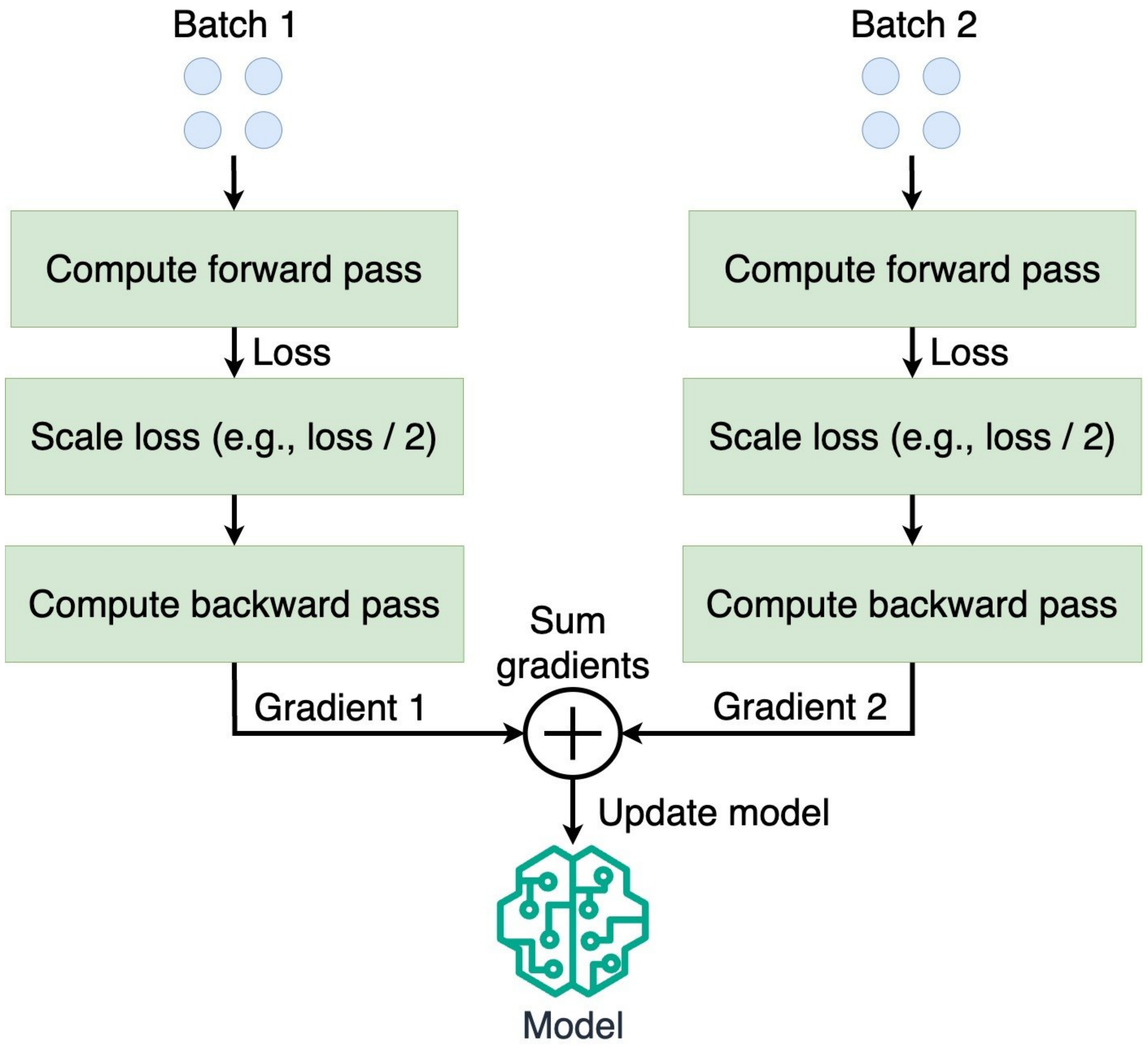
Gradient accumulation with Accelerator

```
accelerator = \
    Accelerator(gradient_accumulation_steps=2)

for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = (batch["input_ids"],
                           batch["labels"])
        outputs = model(inputs,
                        labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
```

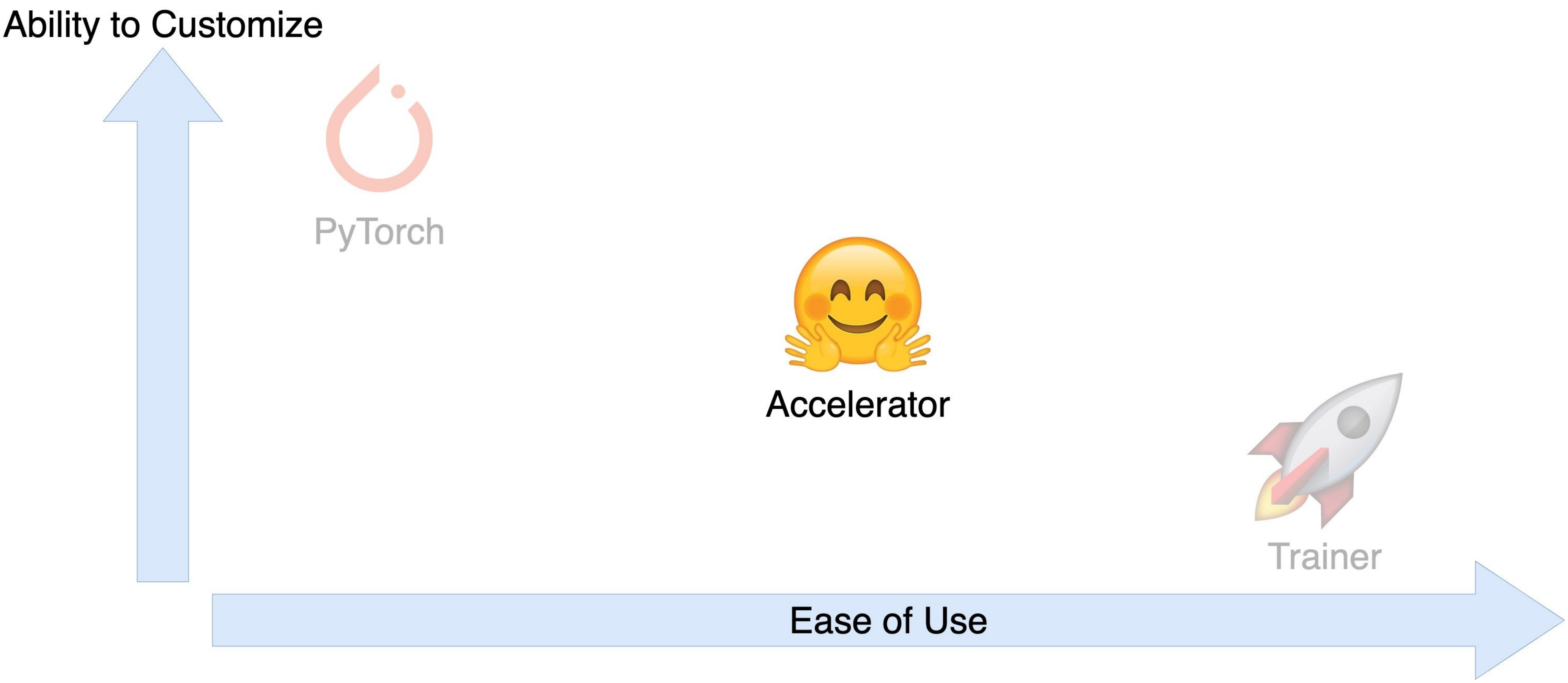


Gradient accumulation with Accelerator

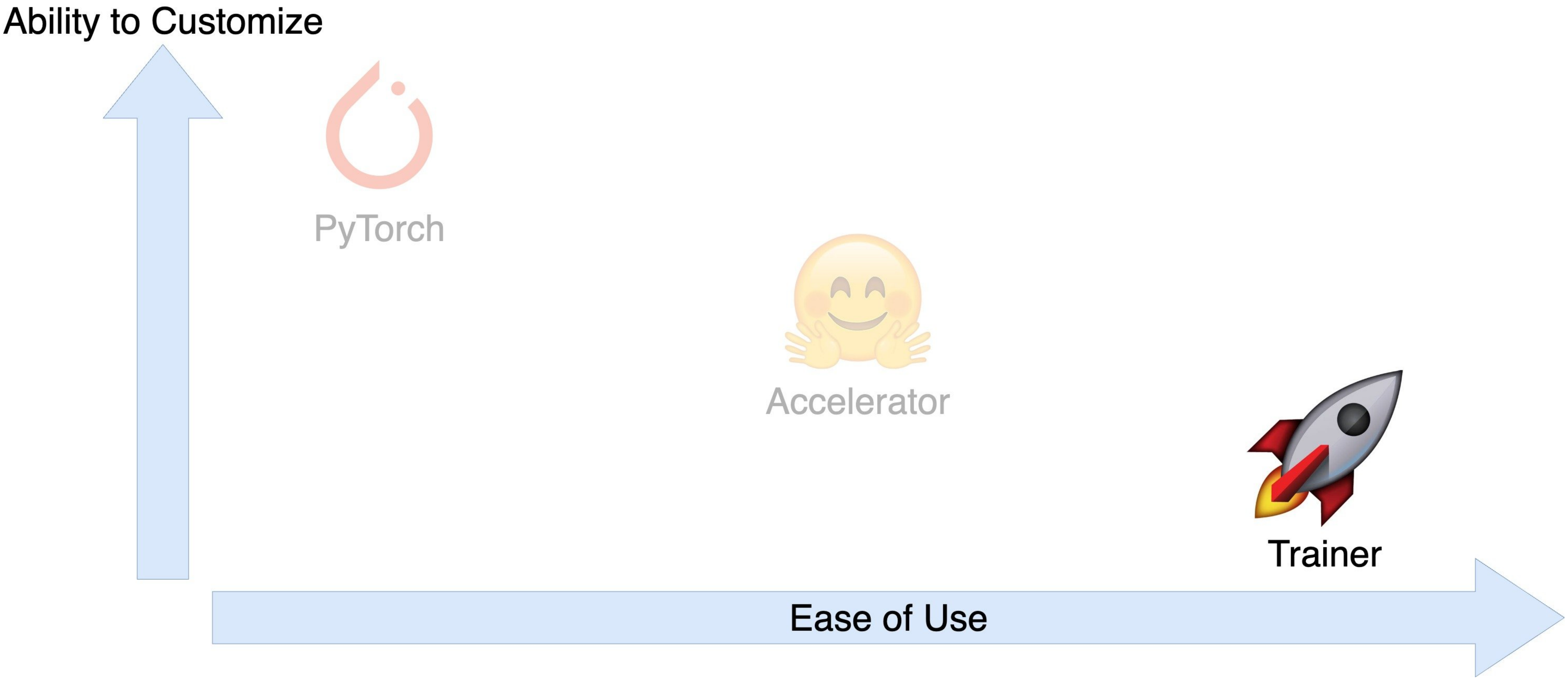
```
accelerator = \
    Accelerator(gradient_accumulation_steps=2)

for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = (batch["input_ids"],
                           batch["labels"])
        outputs = model(inputs,
                        labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```



From Accelerator to Trainer



From Accelerator to Trainer



Gradient accumulation with Trainer

```
training_args = TrainingArguments(output_dir="./results",
                                  evaluation_strategy="epoch",
                                  gradient_accumulation_steps=2)
trainer = Trainer(model=model,
                  args=training_args,
                  train_dataset=dataset["train"],
                  eval_dataset=dataset["validation"],
                  compute_metrics=compute_metrics)
trainer.train()
```

```
{'epoch': 1.0, 'eval_loss': 0.73, 'eval_accuracy': 0.03, 'eval_f1': 0.05}
{'epoch': 2.0, 'eval_loss': 0.68, 'eval_accuracy': 0.19, 'eval_f1': 0.25}
```

Let's practice!

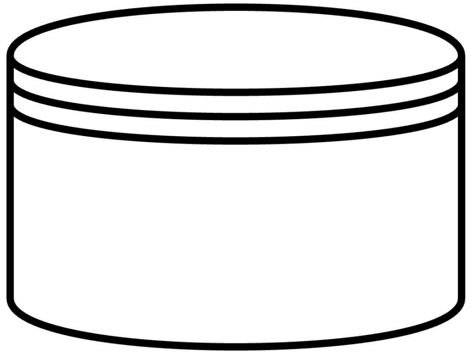
EFFICIENT AI MODEL TRAINING WITH PYTORCH

Gradient checkpointing and local SGD

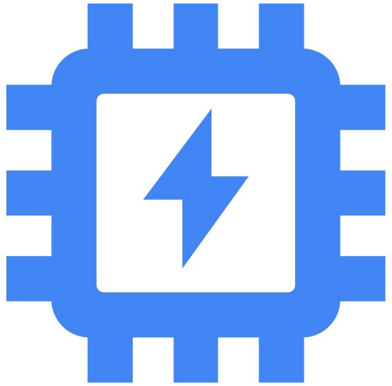
EFFICIENT AI MODEL TRAINING WITH PYTORCH

Dennis Lee
Data Engineer

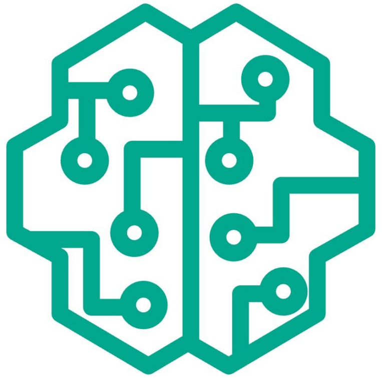
Improving training efficiency



Memory
Efficiency

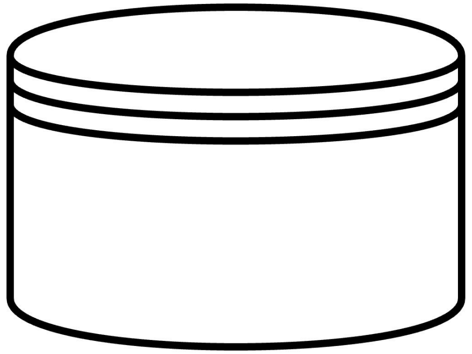


Communication
Efficiency



Computational
Efficiency

Gradient checkpointing improves memory efficiency

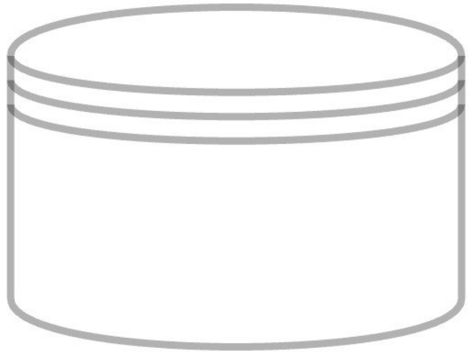


Memory
Efficiency

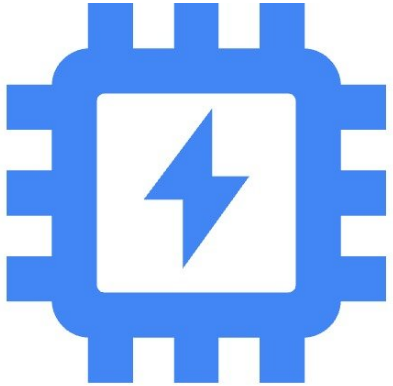
Communication
Efficiency

Computational
Efficiency

Local SGD addresses communication efficiency



Memory
Efficiency

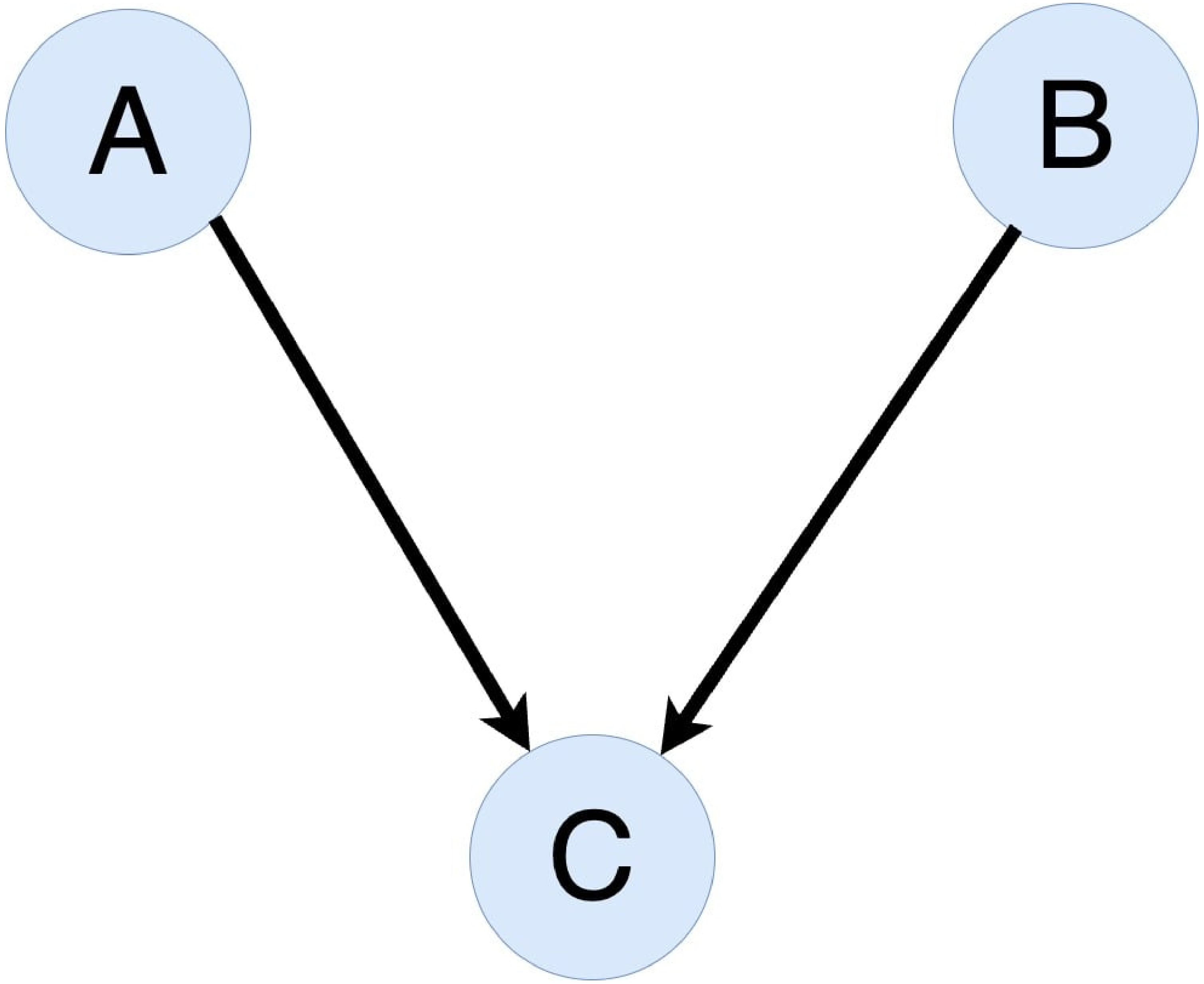


Communication
Efficiency

Computational
Efficiency

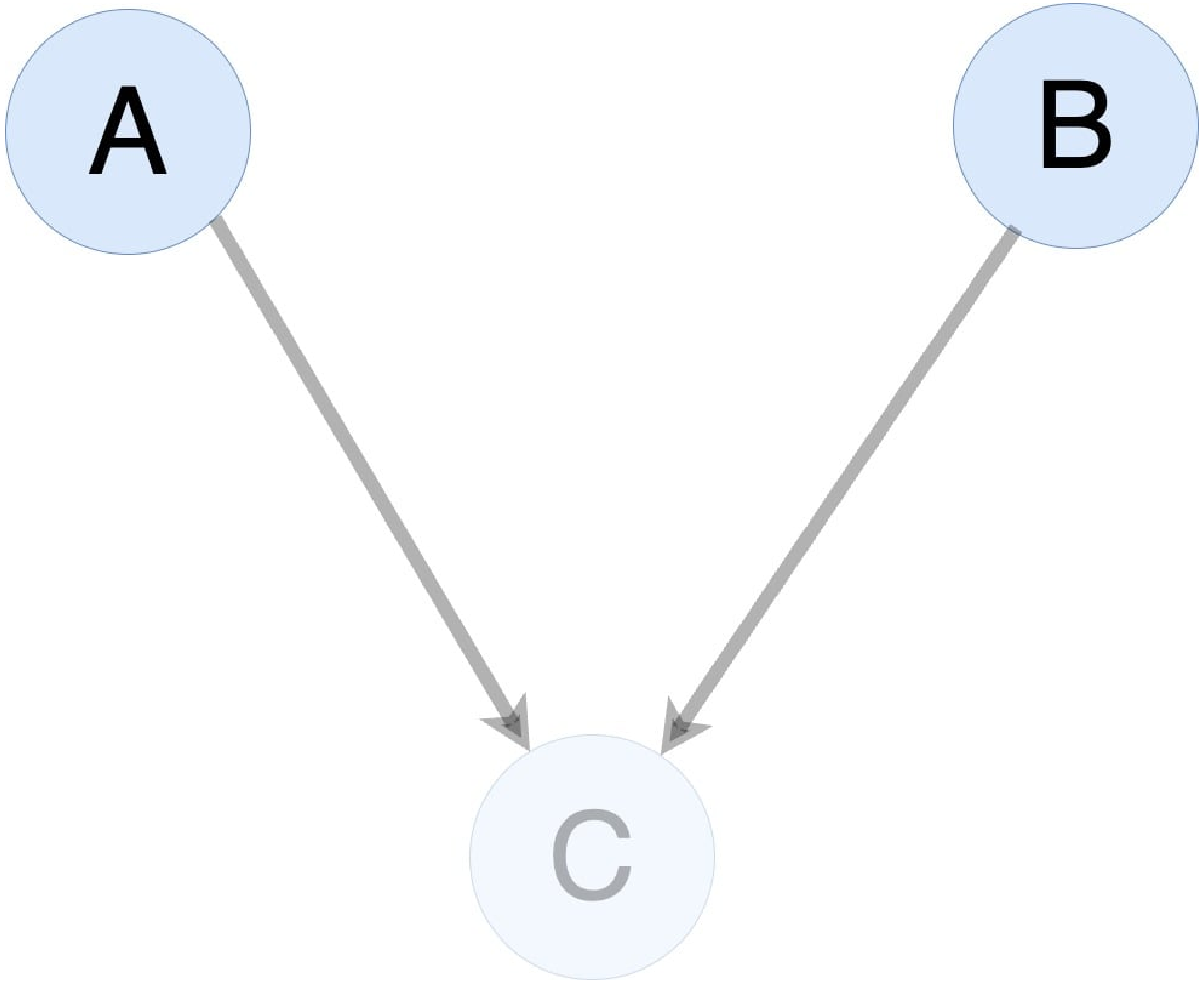
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C



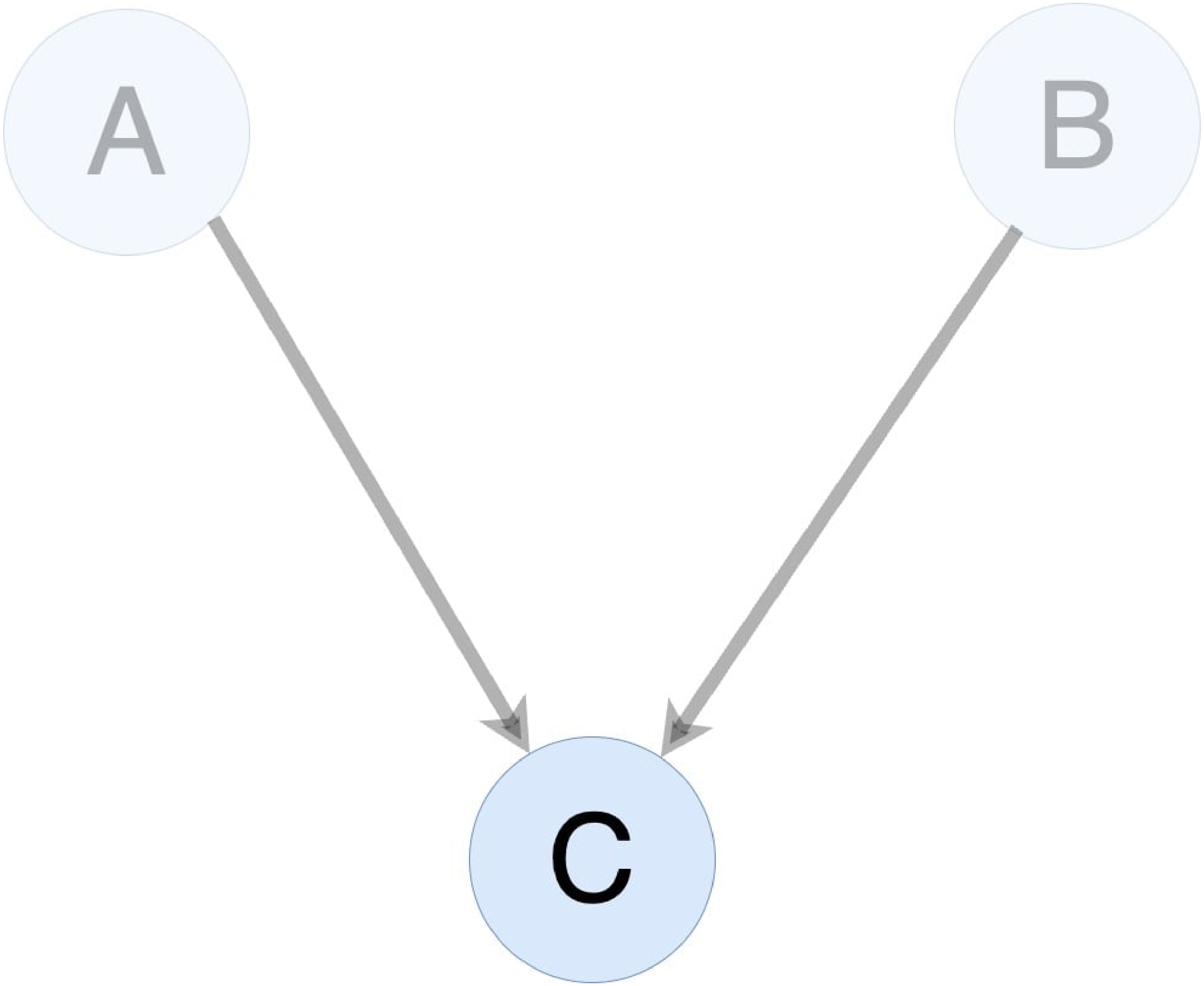
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C



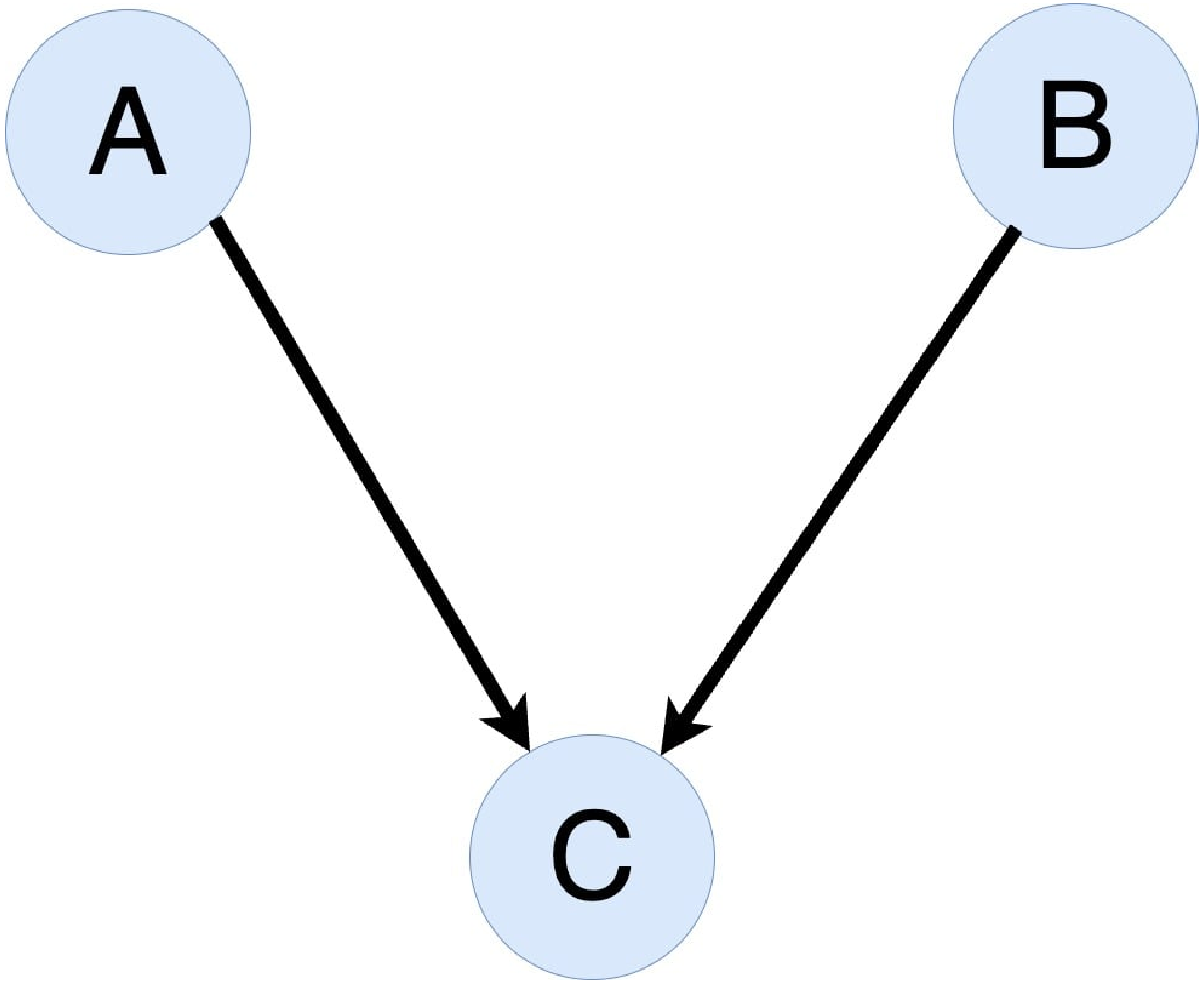
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C
  - A, B not needed for rest of forward pass
- Should we save or remove A and B?



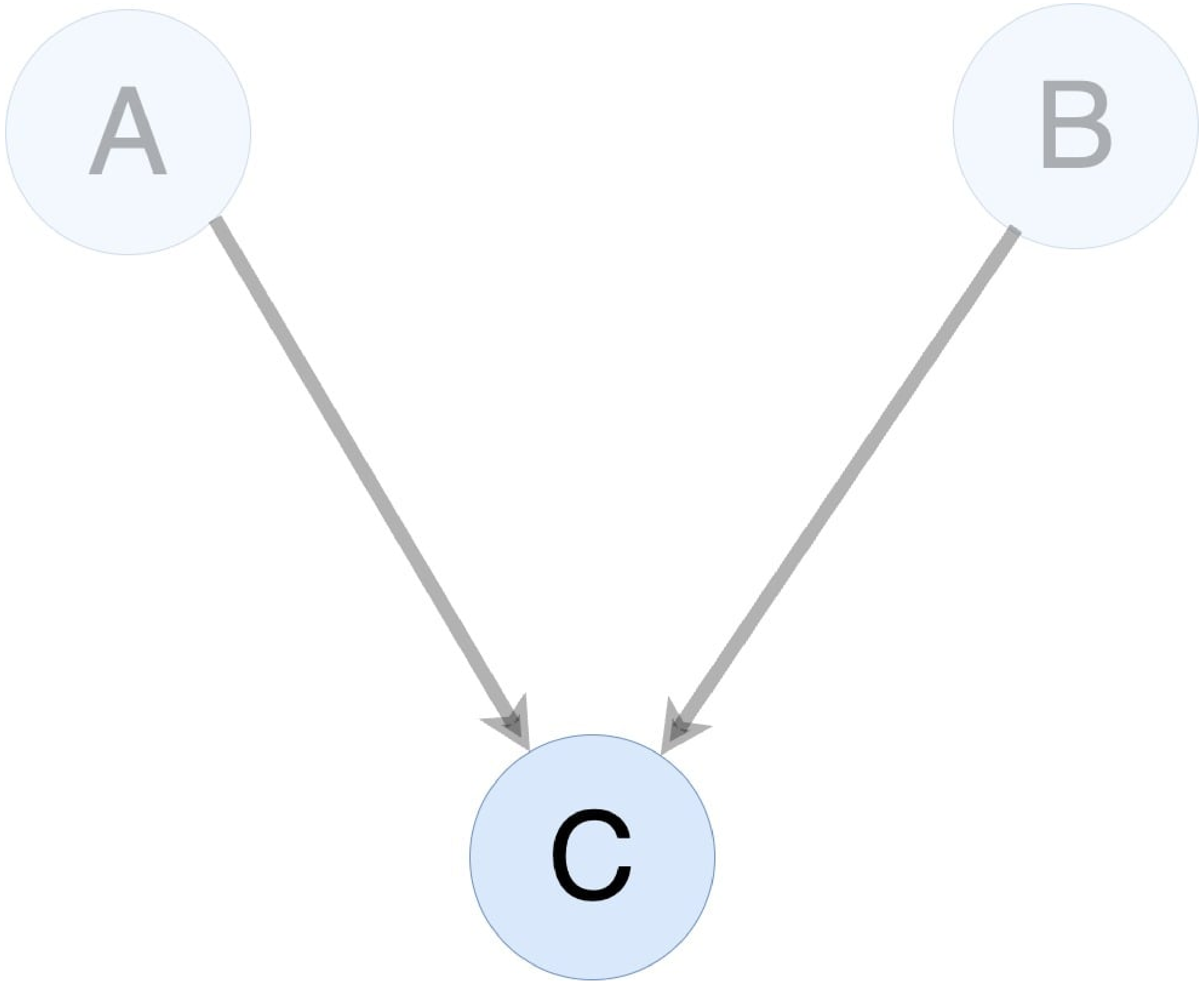
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C
  - A, B not needed for rest of forward pass
- Should we save or remove A and B?
  - No gradient checkpointing: save A, B



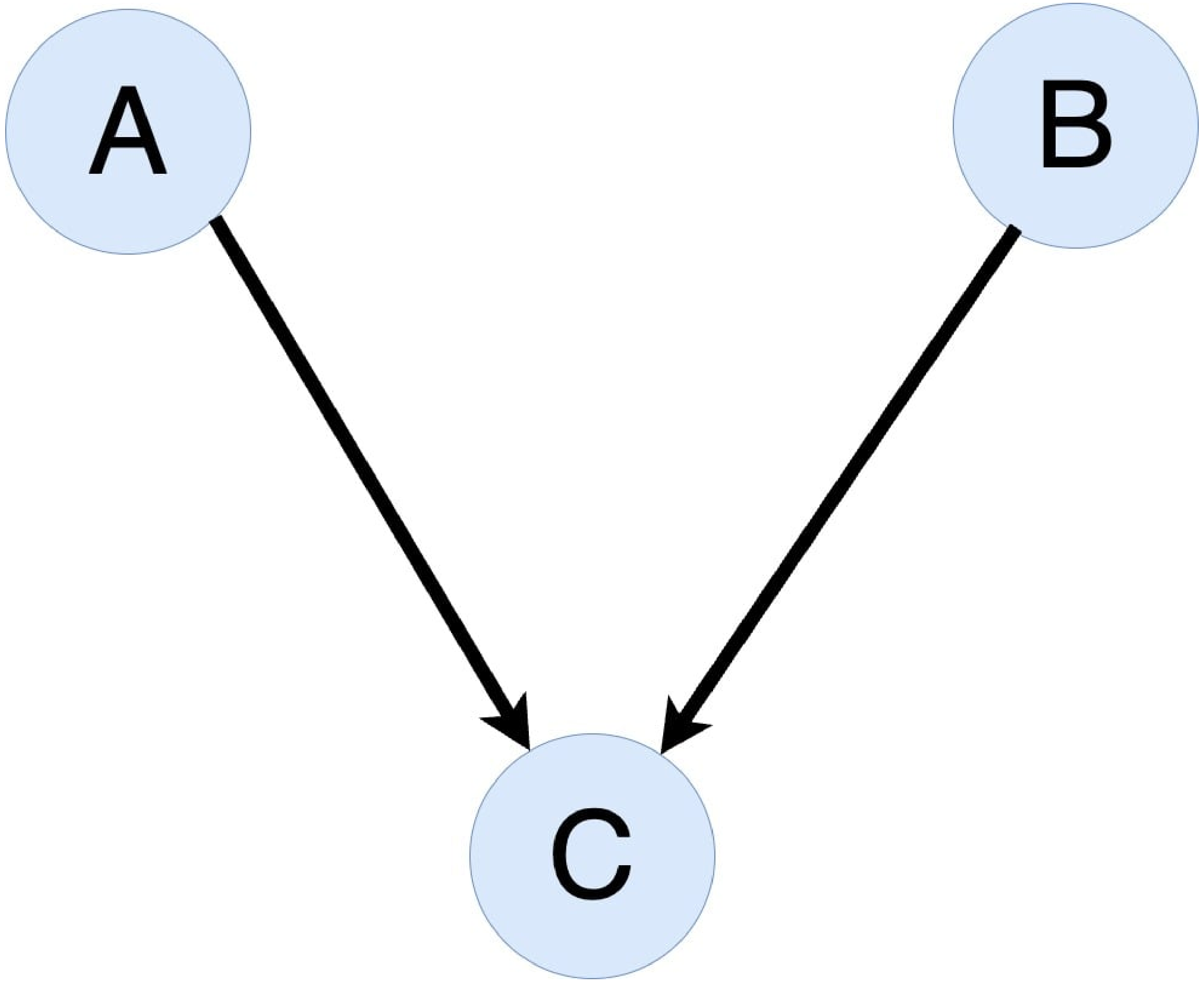
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C
  - A, B not needed for rest of forward pass
- Should we save or remove A and B?
  - No gradient checkpointing: save A, B
  - Gradient checkpointing: remove A, B



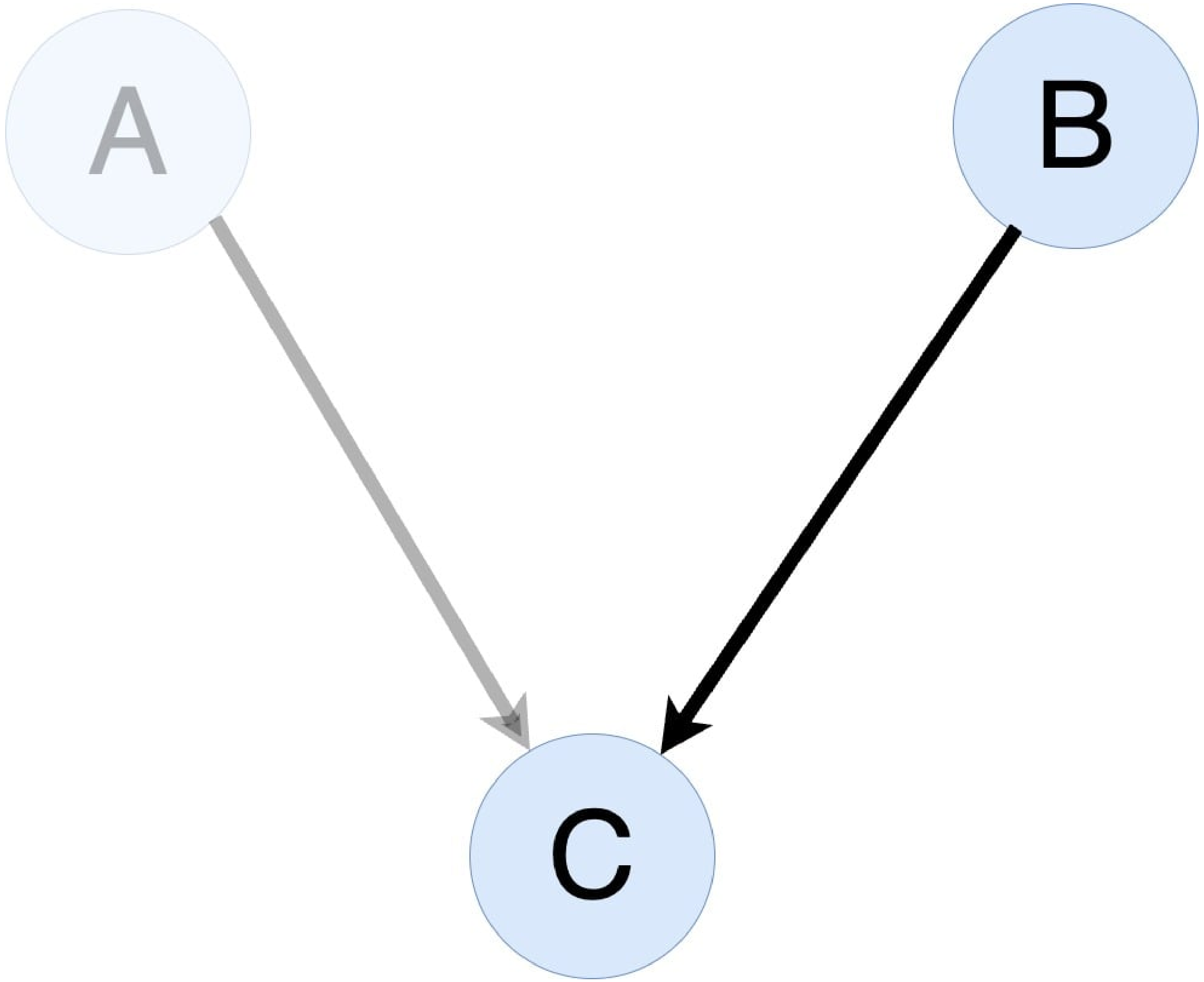
What is gradient checkpointing?

- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C
  - A, B not needed for rest of forward pass
- Should we save or remove A and B?
  - No gradient checkpointing: save A, B
  - Gradient checkpointing: remove A, B
  - Recompute A, B during backward pass

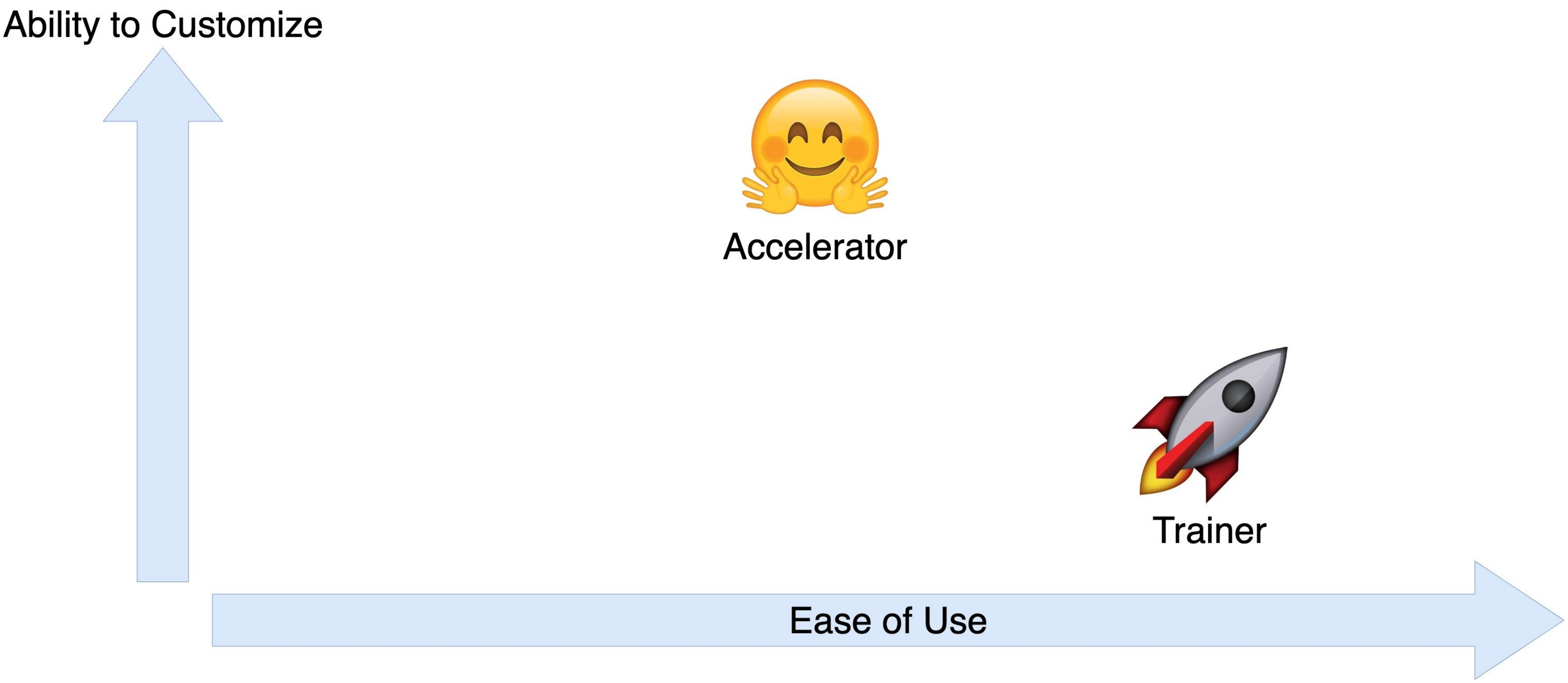


What is gradient checkpointing?

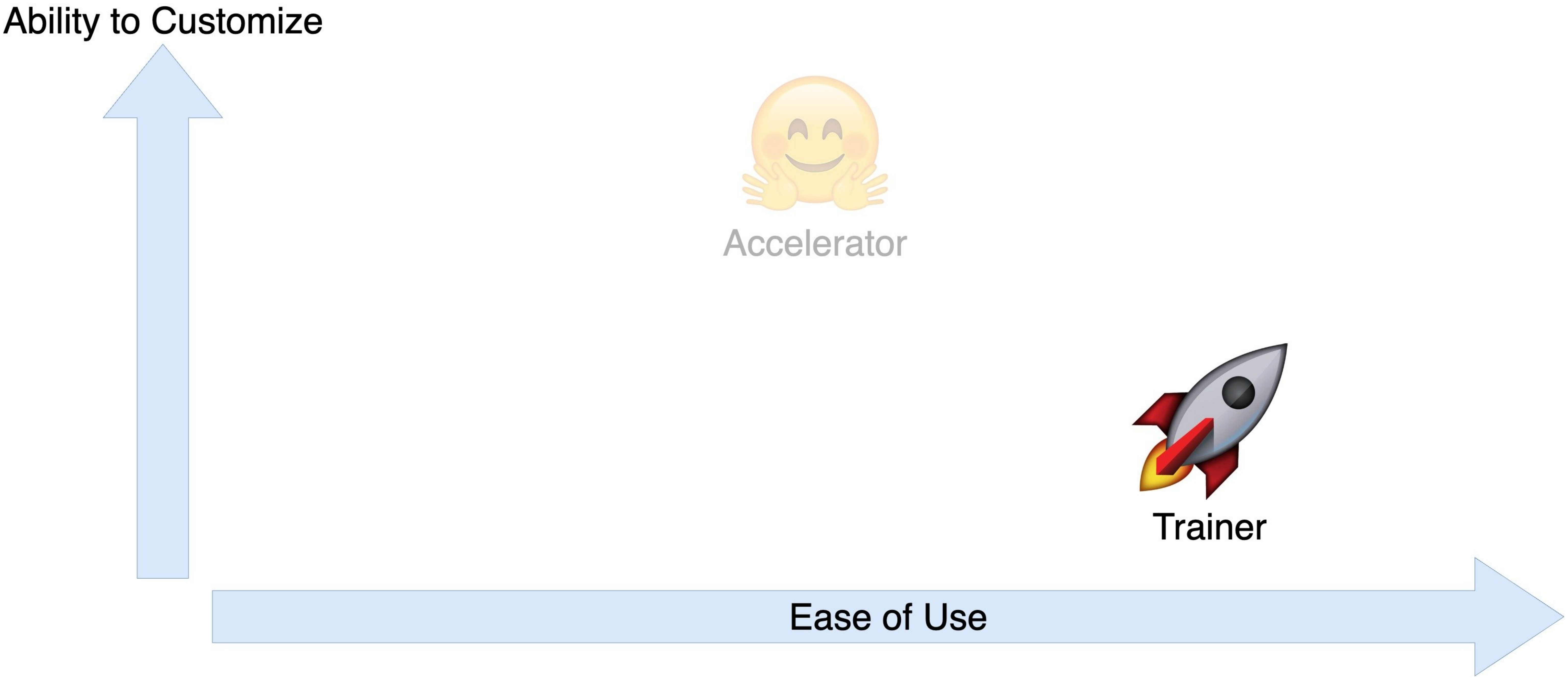
- Gradient checkpointing: reduce memory by selecting which activations to save
- Example: compute A + B = C
  - First compute A, B, then compute C
  - A, B not needed for rest of forward pass
- Should we save or remove A and B?
  - No gradient checkpointing: save A, B
  - Gradient checkpointing: remove A, B
  - Recompute A, B during backward pass
  - If B is expensive to recompute, save it



Trainer and Accelerator



Trainer and Accelerator



Gradient checkpointing with Trainer

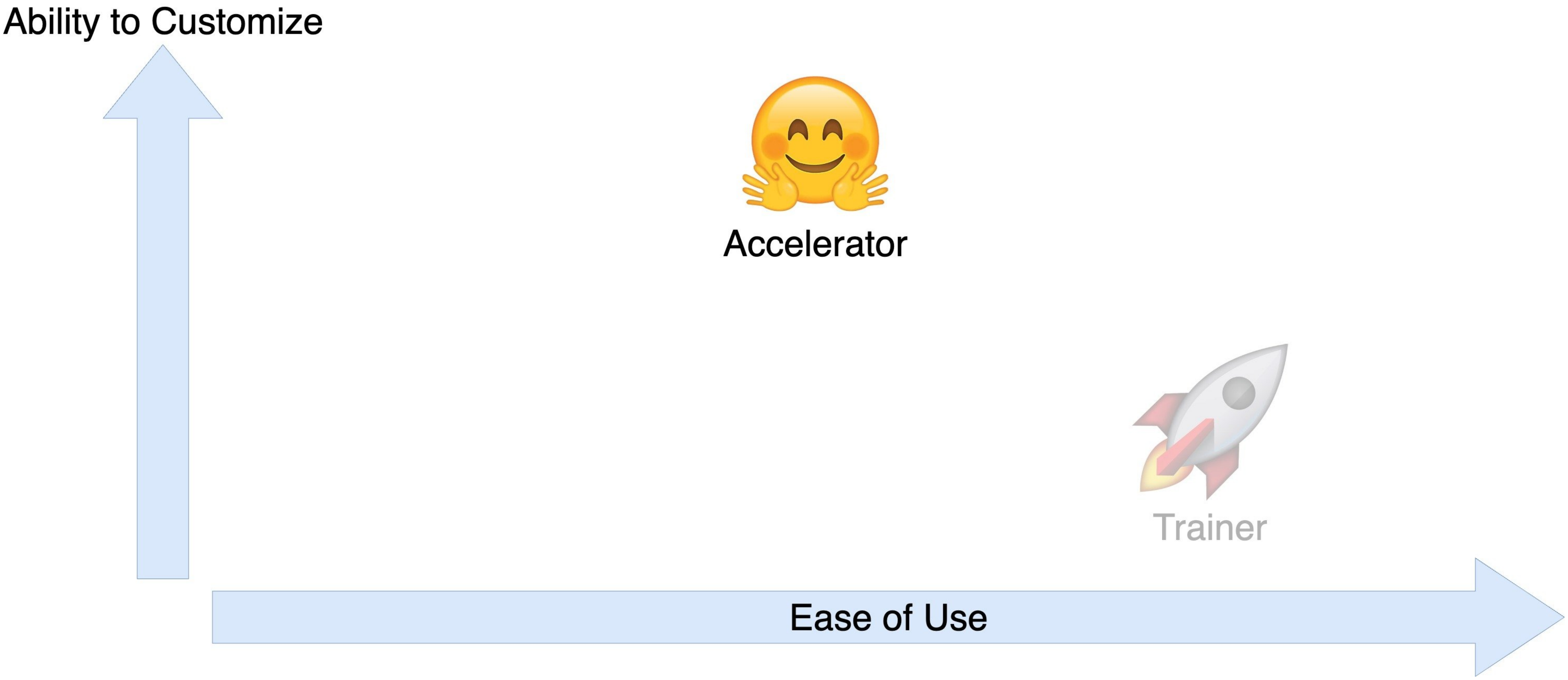
```
training_args = TrainingArguments(output_dir="./results",
                                  evaluation_strategy="epoch",
                                  gradient_accumulation_steps=4)
```

Gradient checkpointing with Trainer

```
training_args = TrainingArguments(output_dir="./results",
                                  evaluation_strategy="epoch",
                                  gradient_accumulation_steps=4,
                                  gradient_checkpointing=True)
trainer = Trainer(model=model,
                  args=training_args,
                  train_dataset=dataset["train"],
                  eval_dataset=dataset["validation"],
                  compute_metrics=compute_metrics)
trainer.train()
```

```
{'epoch': 1.0, 'eval_loss': 0.73, 'eval_accuracy': 0.03, 'eval_f1': 0.05}
```

From Trainer to Accelerator



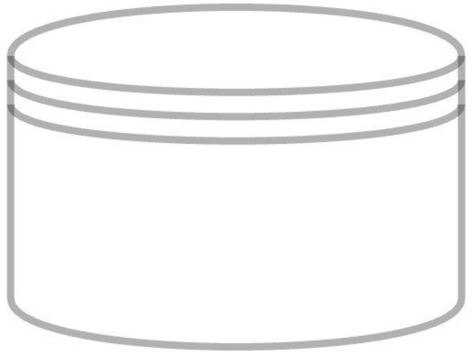
Gradient checkpointing with Accelerator

```python
accelerator = Accelerator(gradient_accumulation_steps=2)



for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = batch["input_ids"], batch["labels"]
        outputs = model(inputs, labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```

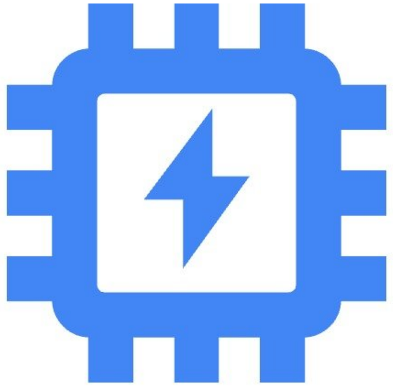
Gradient checkpointing with Accelerator

```
accelerator = Accelerator(gradient_accumulation_steps=2)
model.gradient_checkpointing_enable()

for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = batch["input_ids"], batch["labels"]
        outputs = model(inputs, labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```

Local SGD improves communication efficiency



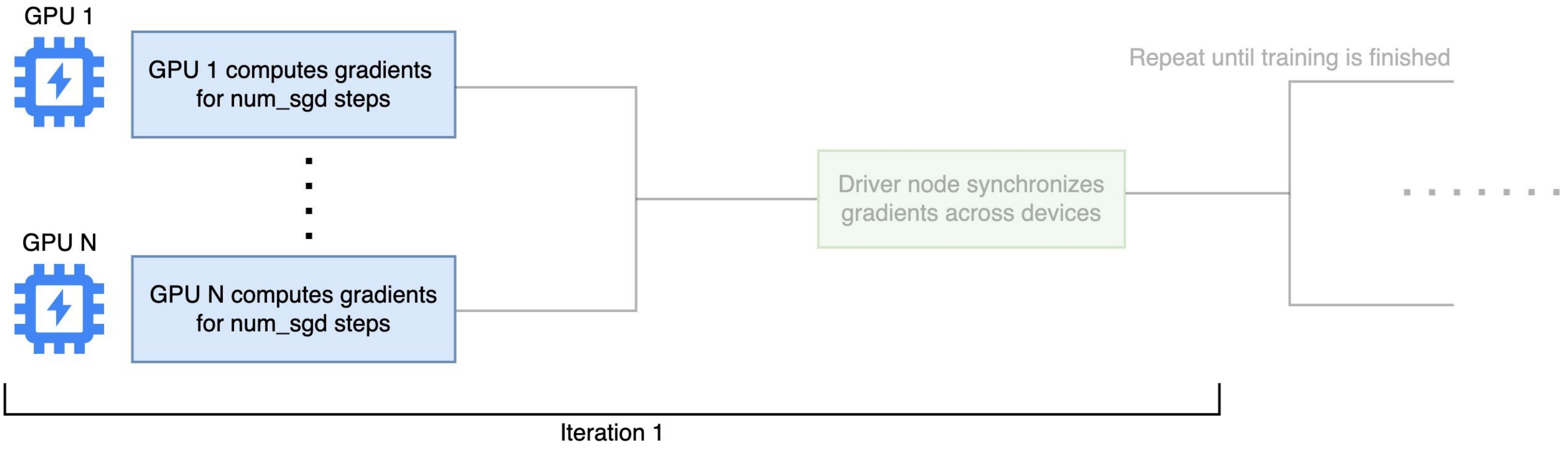
Memory Efficiency



Communication Efficiency

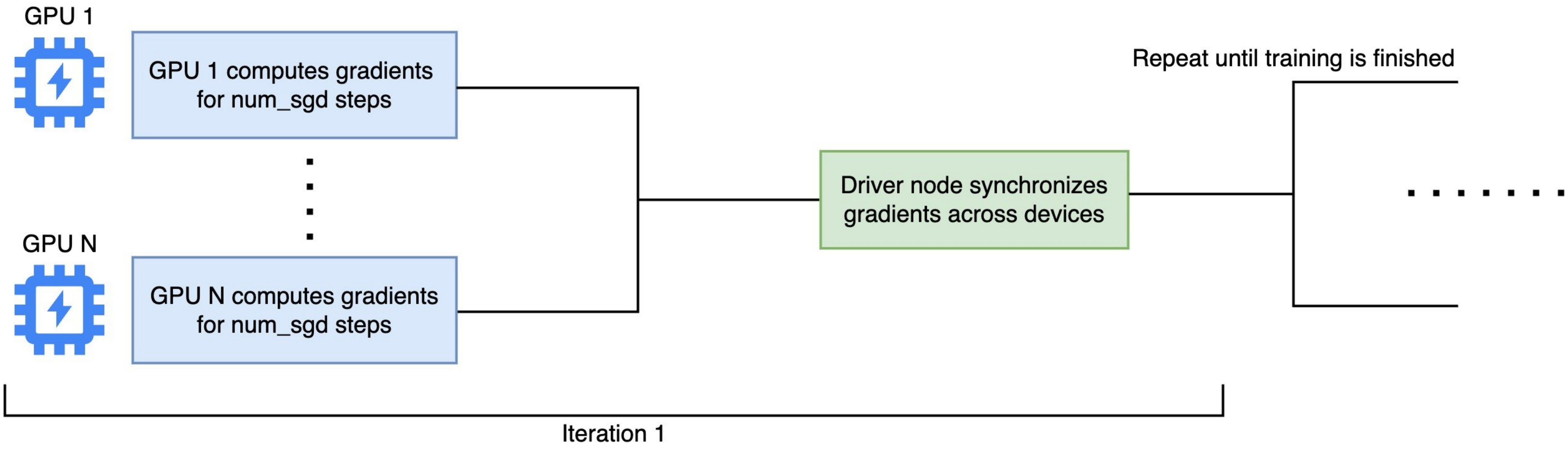
Computational Efficiency

What is local SGD?



- Each device computes gradients in parallel

What is local SGD?



- Each device computes gradients in parallel
- Gradient synchronization: Driver node updates model parameters on each device
- Local SGD: Reduce frequency of gradient synchronization

Local SGD with Accelerator

```python
for index, batch in enumerate(dataloader):
    with accelerator.accumulate(model):
        inputs, targets = batch["input_ids"], batch["labels"]
        outputs = model(inputs, labels=targets)
        loss = outputs.loss
        accelerator.backward(loss)
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
```

Local SGD with Accelerator

```
from accelerate.local_sgd import LocalSGD

with LocalSGD(accelerator=accelerator, model=model, local_sgd_steps=8,
              enabled=True) as local_sgd:
    for index, batch in enumerate(dataloader):
        with accelerator.accumulate(model):
            inputs, targets = batch["input_ids"], batch["labels"]
            outputs = model(inputs, labels=targets)
            loss = outputs.loss
            accelerator.backward(loss)
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
```

Local SGD with Accelerator

```
from accelerate.local_sgd import LocalSGD

with LocalSGD(accelerator=accelerator, model=model, local_sgd_steps=8,
              enabled=True) as local_sgd:
    for index, batch in enumerate(dataloader):
        with accelerator.accumulate(model):
            inputs, targets = batch["input_ids"], batch["labels"]
            outputs = model(inputs, labels=targets)
            loss = outputs.loss
            accelerator.backward(loss)
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
            local_sgd.step()
```

Let's practice!

EFFICIENT AI MODEL TRAINING WITH PYTORCH

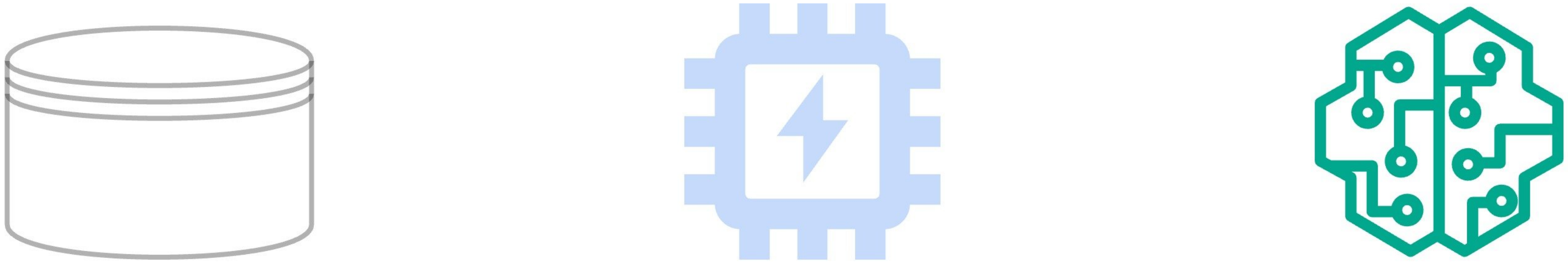
Mixed precision training

EFFICIENT AI MODEL TRAINING WITH PYTORCH

Dennis Lee
Data Engineer

datacamp

Mixed precision training accelerates computation

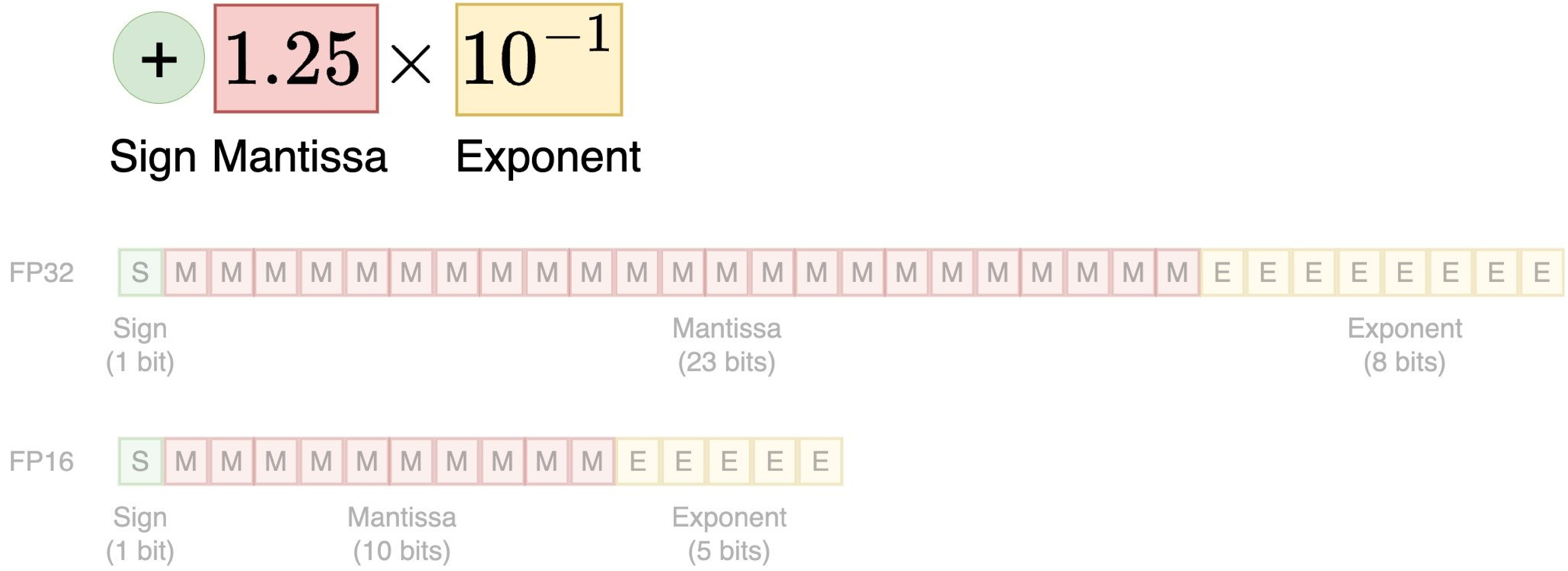


Memory Efficiency

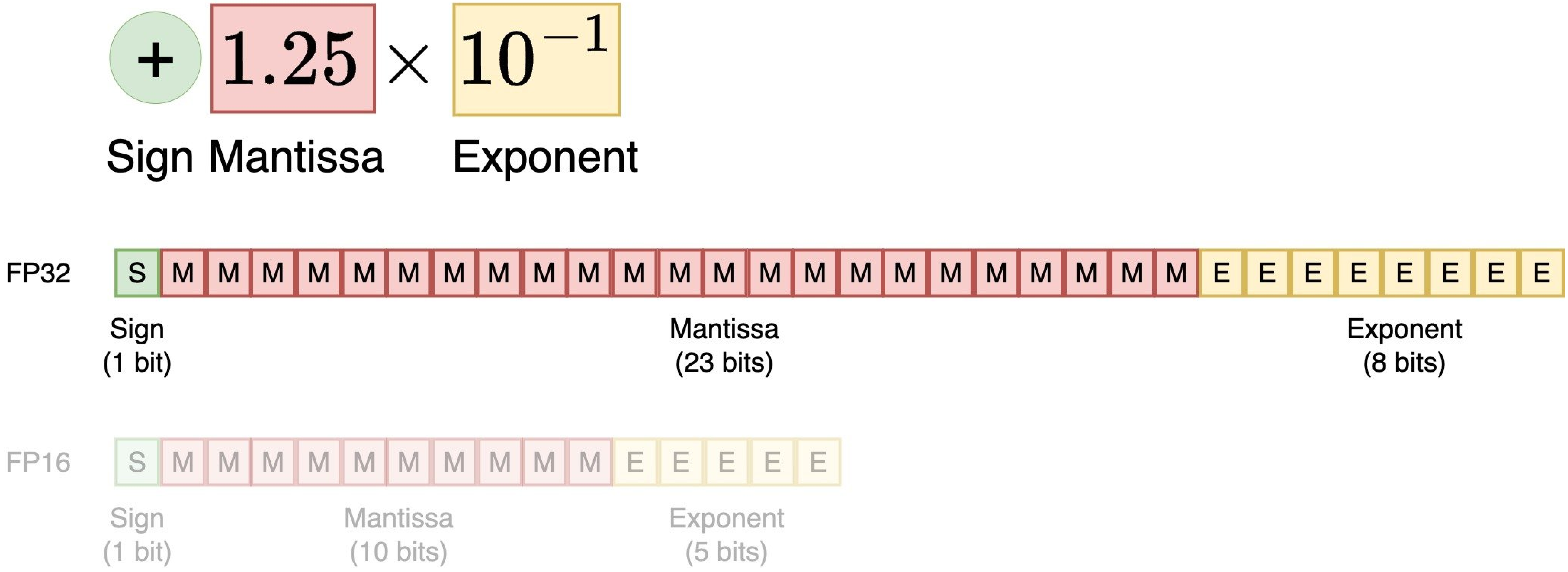
Communication Efficiency

Computational Efficiency

Faster calculations with less precision



Faster calculations with less precision



Faster calculations with less precision

$$\underbrace{+}_{\text{Sign}}\ \underbrace{1.25}_{\text{Mantissa}} \times \underbrace{10^{-1}}_{\text{Exponent}}$$

Sign Mantissa Exponent

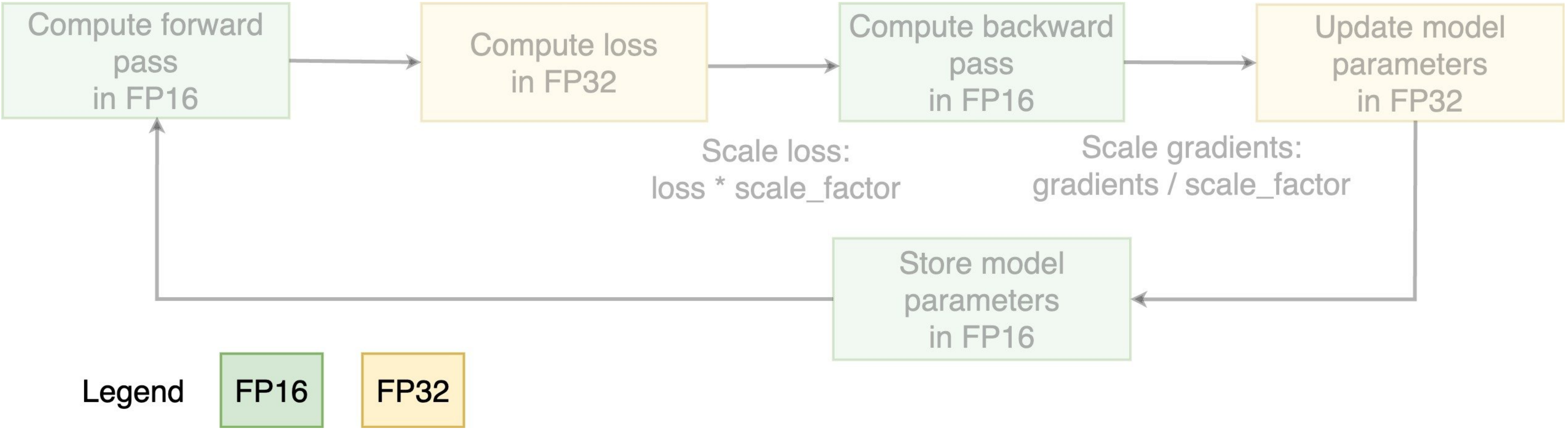
FP32	S	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	E	E	E	E	E	E	E	E

Sign (1 bit) — Mantissa (23 bits) — Exponent (8 bits)

FP16	S	M	M	M	M	M	M	M	M	M	M	E	E	E	E	E

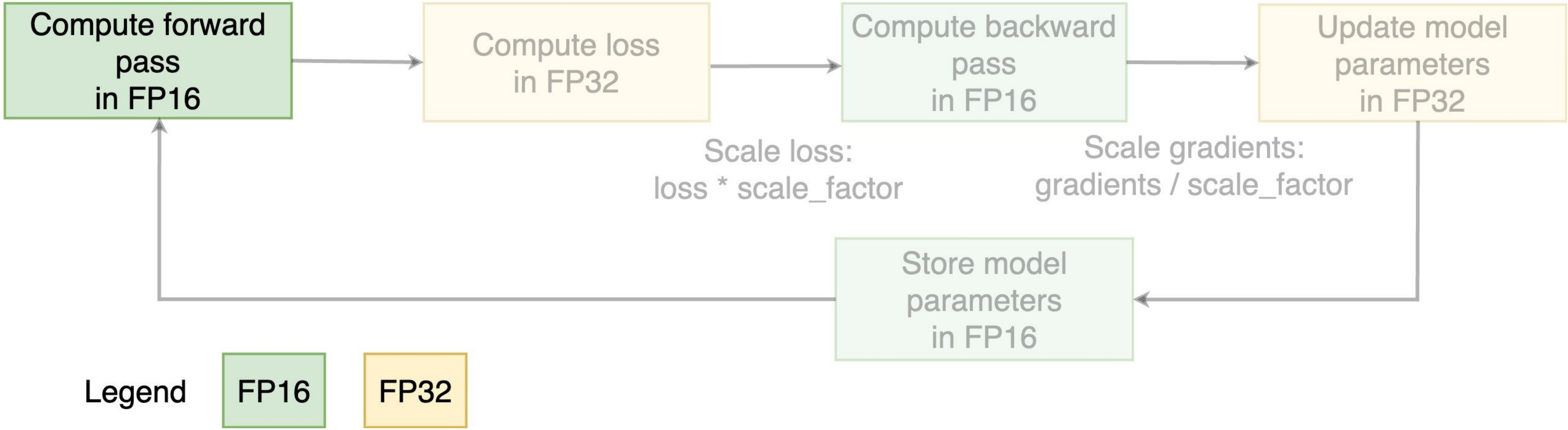
Sign (1 bit) — Mantissa (10 bits) — Exponent (5 bits)

What is mixed precision training?



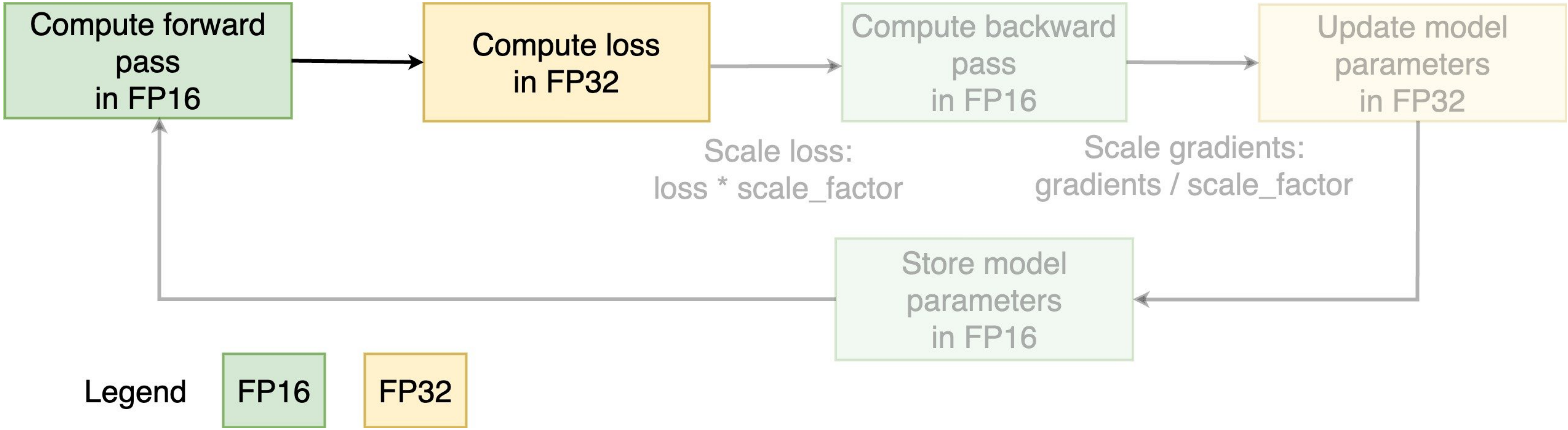
- Mixed precision training: combine FP16, FP32 computations to speed up training

What is mixed precision training?



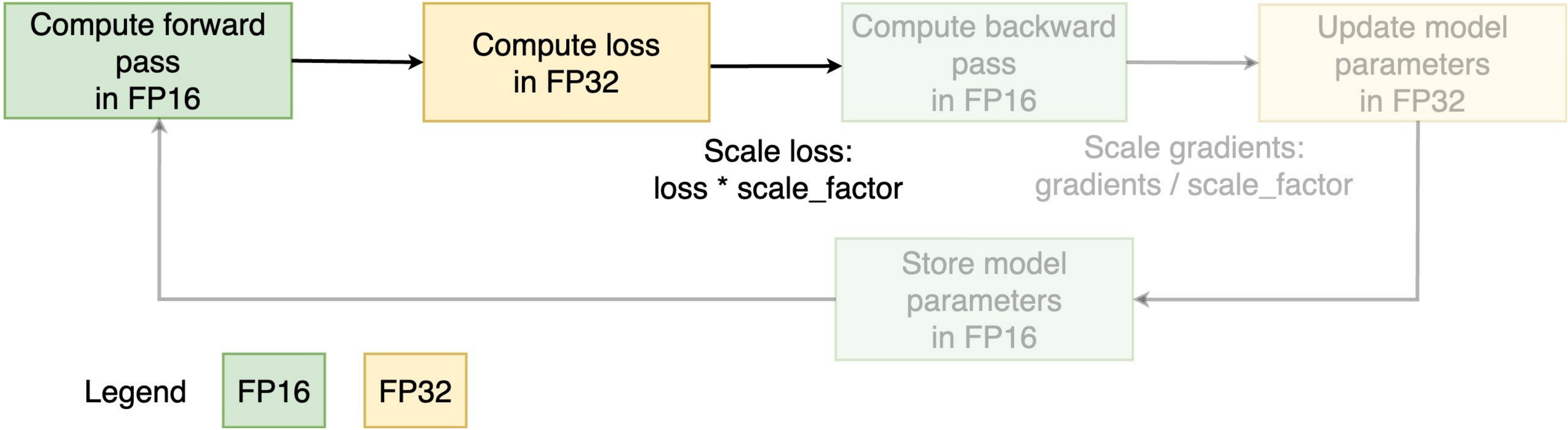
- Mixed precision training: combine FP16, FP32 computations to speed up training

What is mixed precision training?



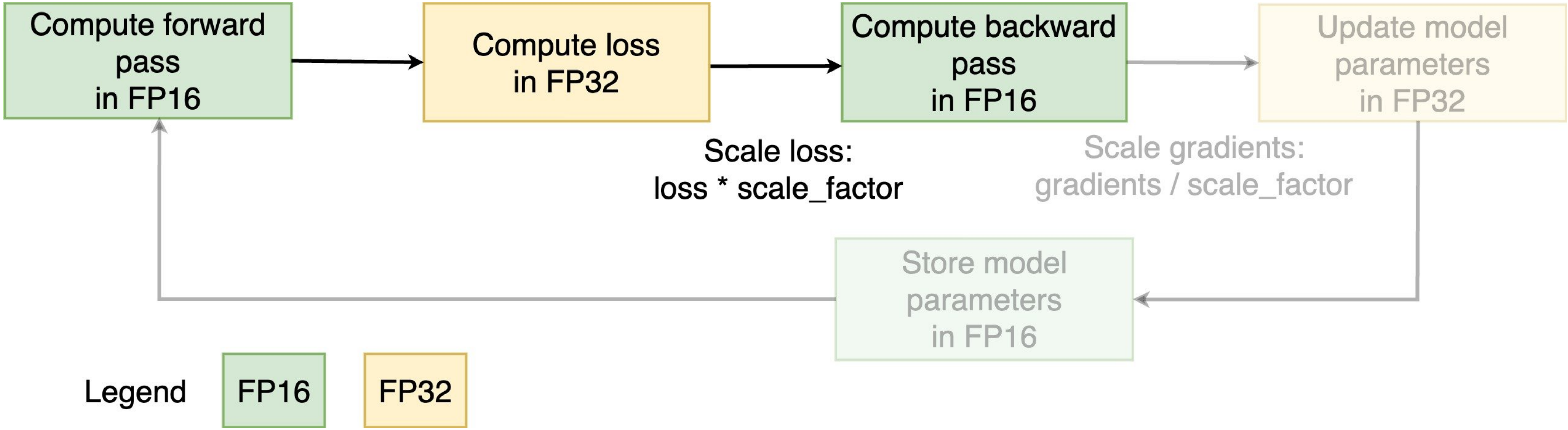
- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision

What is mixed precision training?



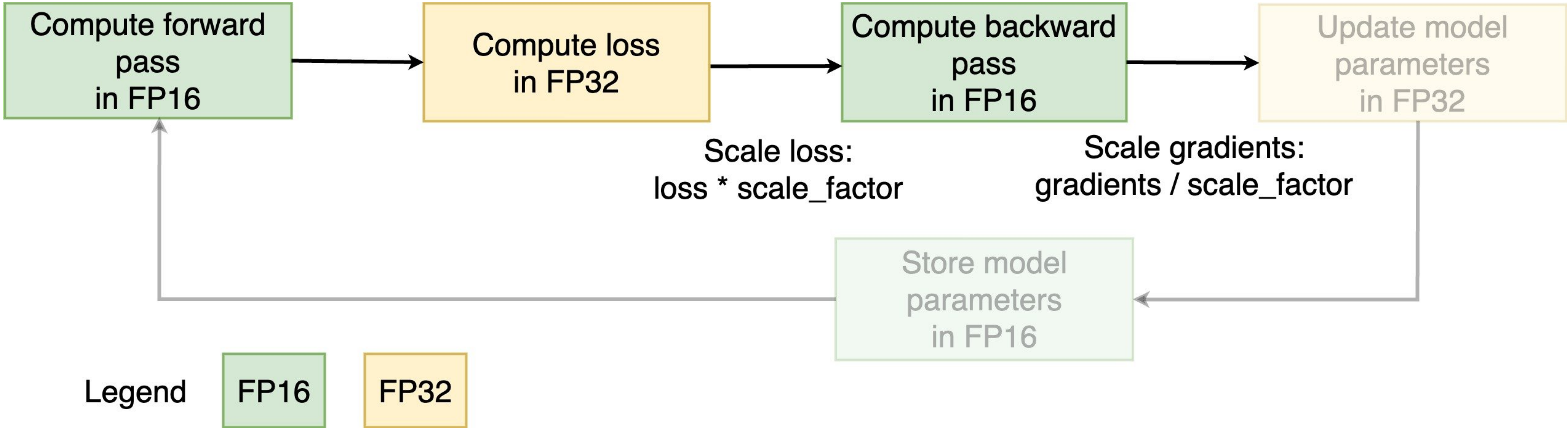
- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision
- Scale loss to prevent underflow

What is mixed precision training?



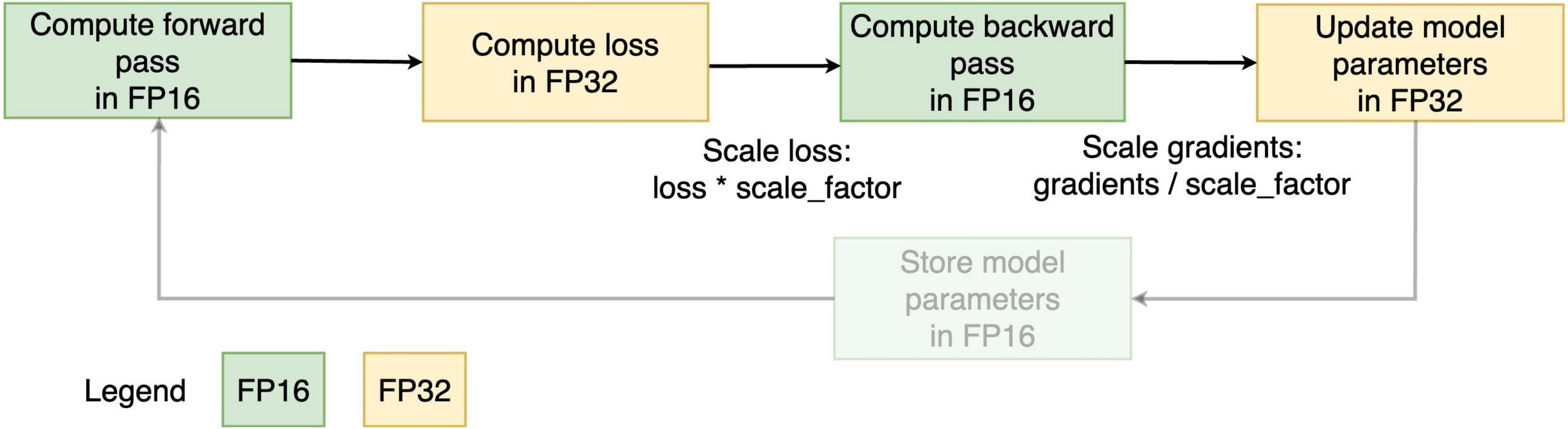
- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision
- Scale loss to prevent underflow

What is mixed precision training?



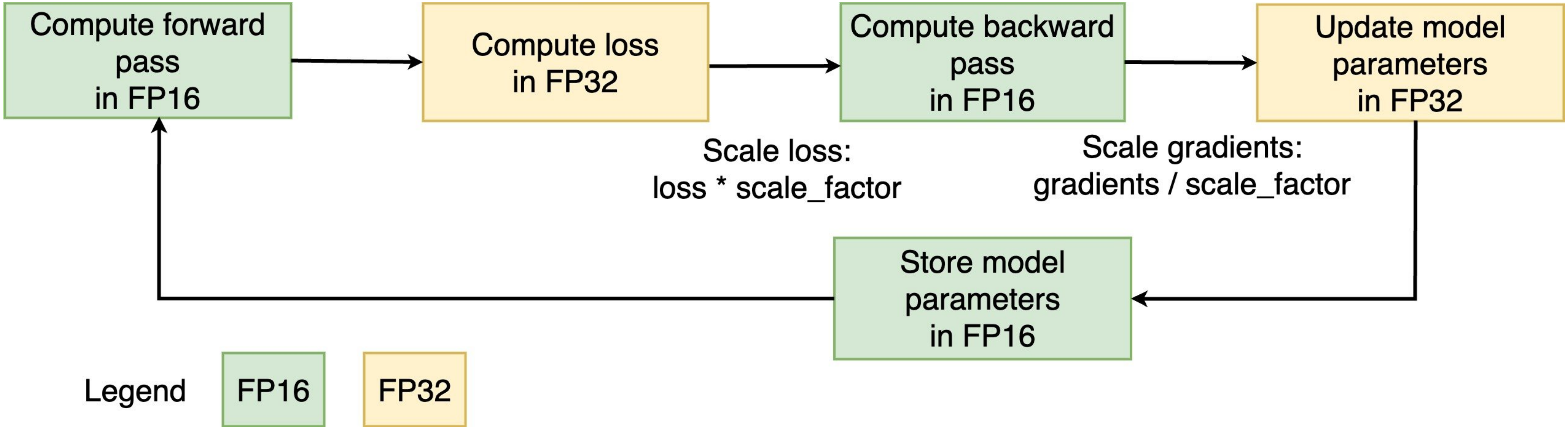
- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision
- Scale loss to prevent underflow

What is mixed precision training?



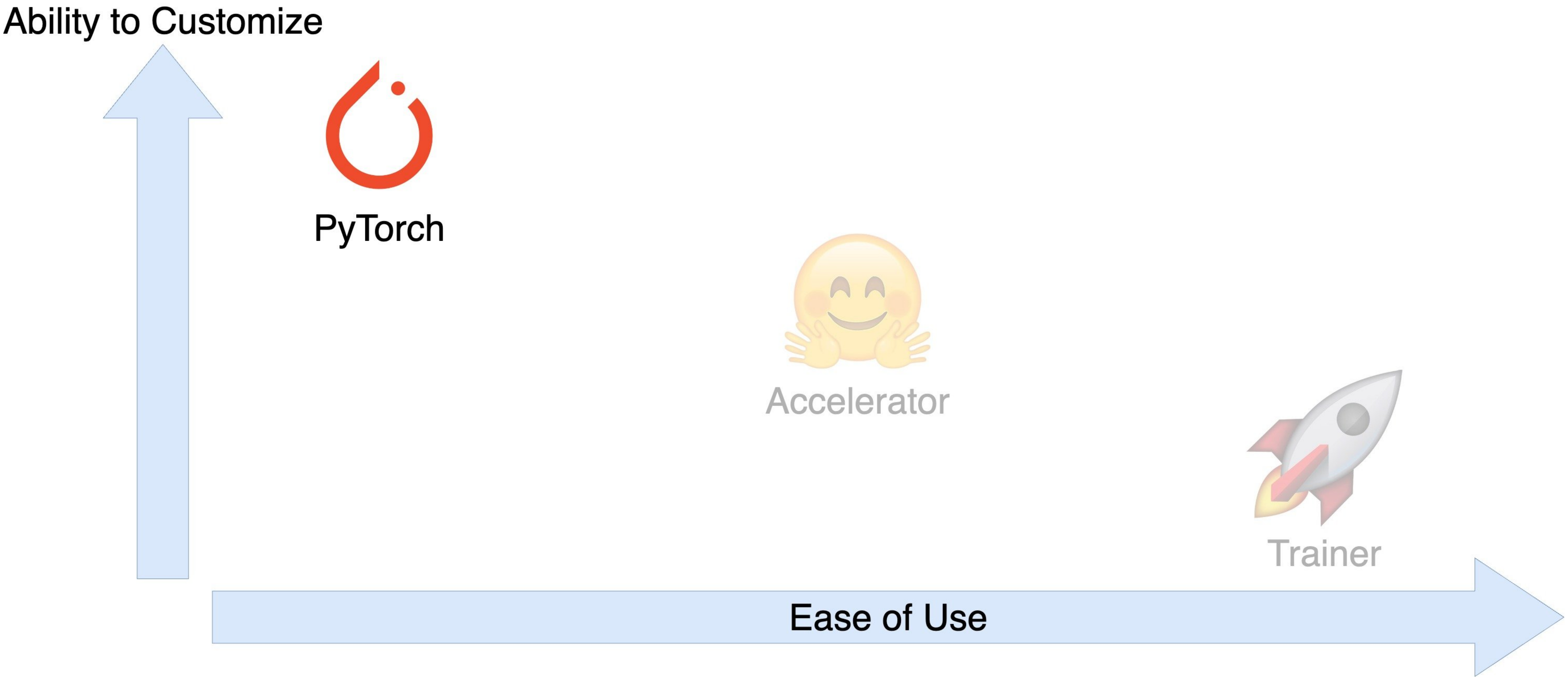
- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision
- Scale loss to prevent underflow

What is mixed precision training?



- Mixed precision training: combine FP16, FP32 computations to speed up training
- Underflow: number vanishes to 0 because it falls below precision
- Scale loss to prevent underflow

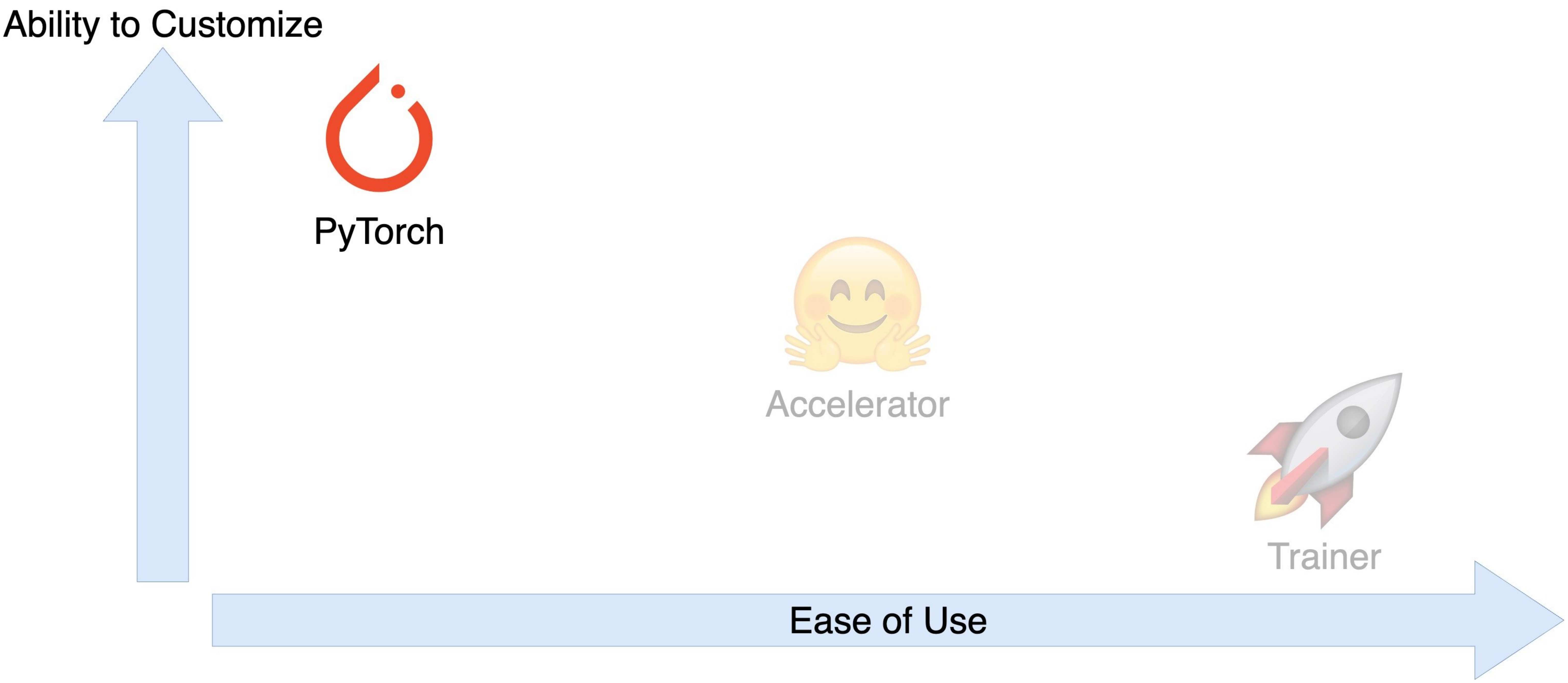
PyTorch implementation



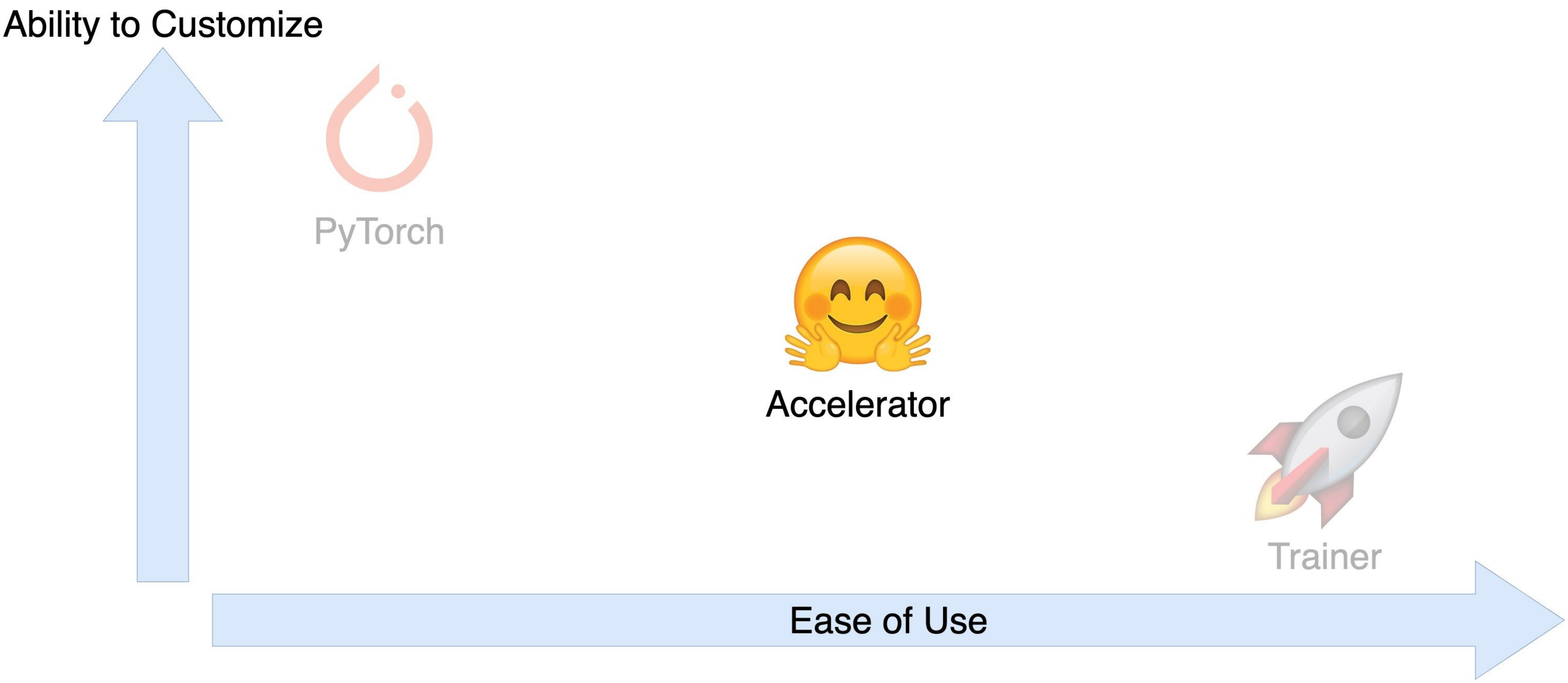
Mixed precision training with PyTorch

```
scaler = torch.amp.GradScaler()

for batch in train_dataloader:
    inputs, targets = batch["input_ids"], batch["labels"]
    with torch.autocast(device_type="cpu", dtype=torch.float16):
        outputs = model(inputs, labels=targets)
        loss = outputs.loss
    scaler.scale(loss).backward()
    scaler.step(optimizer)
    scaler.update()
    optimizer.zero_grad()
```

From PyTorch to Accelerator



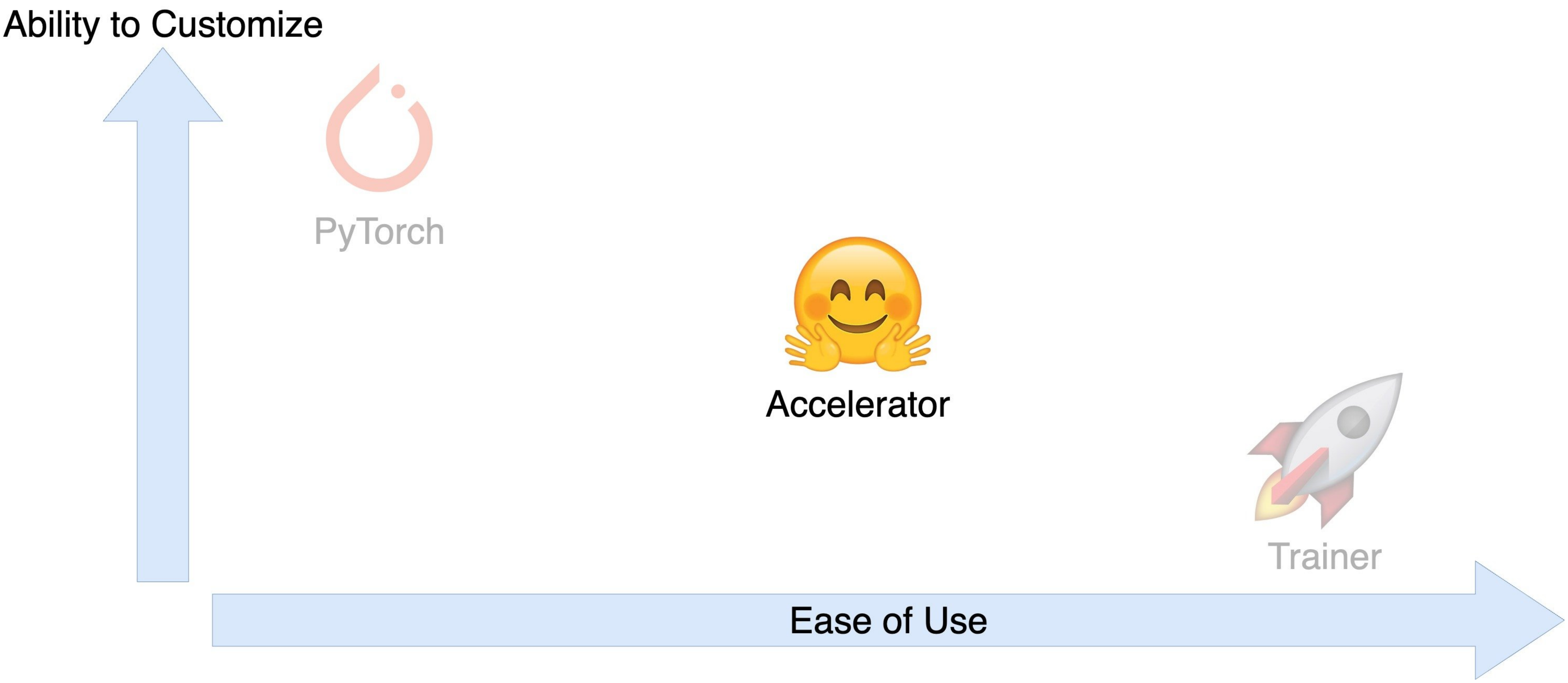
From PyTorch to Accelerator



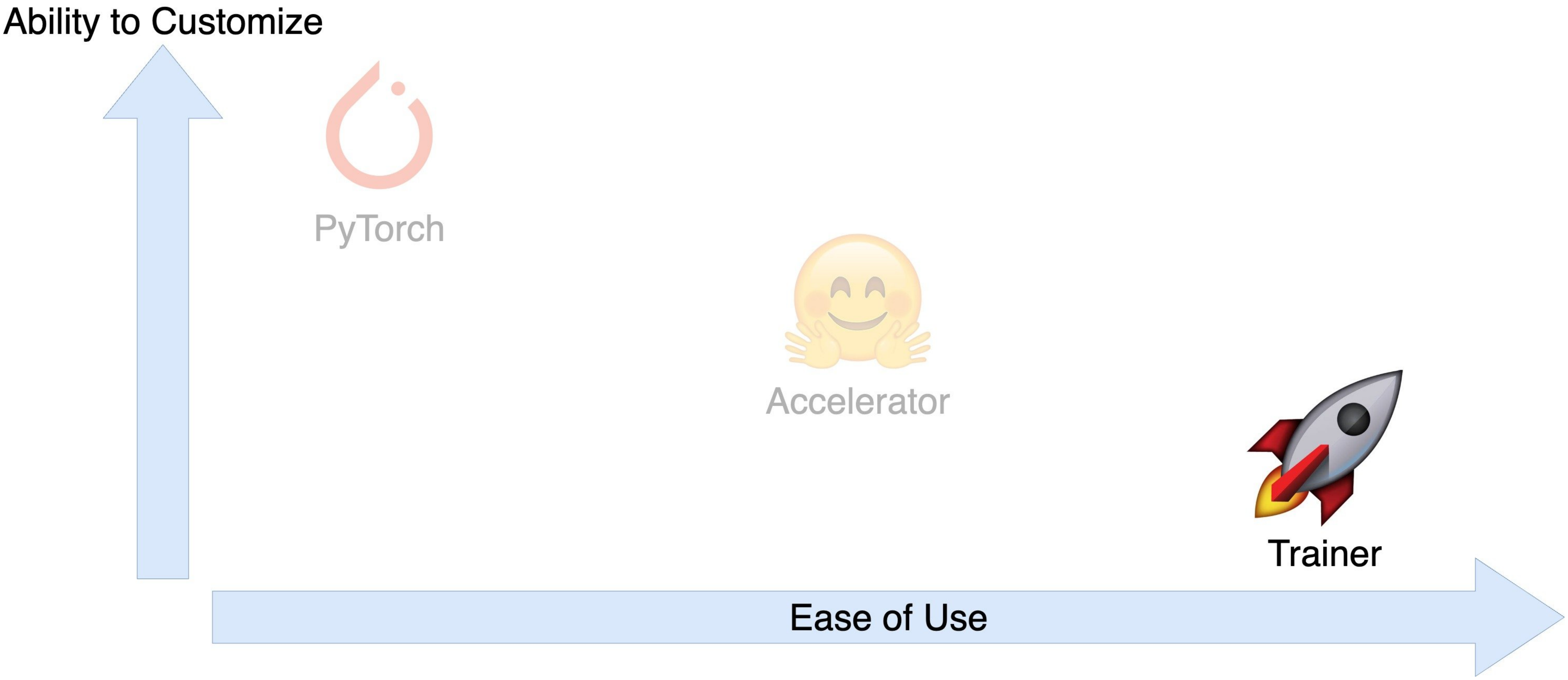
Mixed precision training with Accelerator

```
accelerator = Accelerator(mixed_precision="fp16")

model, optimizer, train_dataloader, lr_scheduler = \
    accelerator.prepare(model, optimizer, train_dataloader, lr_scheduler)

for batch in train_dataloader:
    inputs, targets = batch["input_ids"], batch["labels"]
    outputs = model(inputs, labels=targets)
    loss = outputs.loss
    accelerator.backward(loss)
    optimizer.step()
    optimizer.zero_grad()
```

From Accelerator to Trainer



From Accelerator to Trainer



Mixed precision training with Trainer

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    fp16=True
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset["train"],
    eval_dataset=dataset["validation"],
    compute_metrics=compute_metrics,
)
trainer.train()
```

Let's practice!

EFFICIENT AI MODEL TRAINING WITH PYTORCH