Taking Advantage of Low Precision to Accelerate Training and Inference Using PyTorch

Presented by:
Myle Ott and Sergey Edunov
Facebook AI Research (FAIR)

Talk ID: S9832

Overview

Mixed precision training in PyTorch:

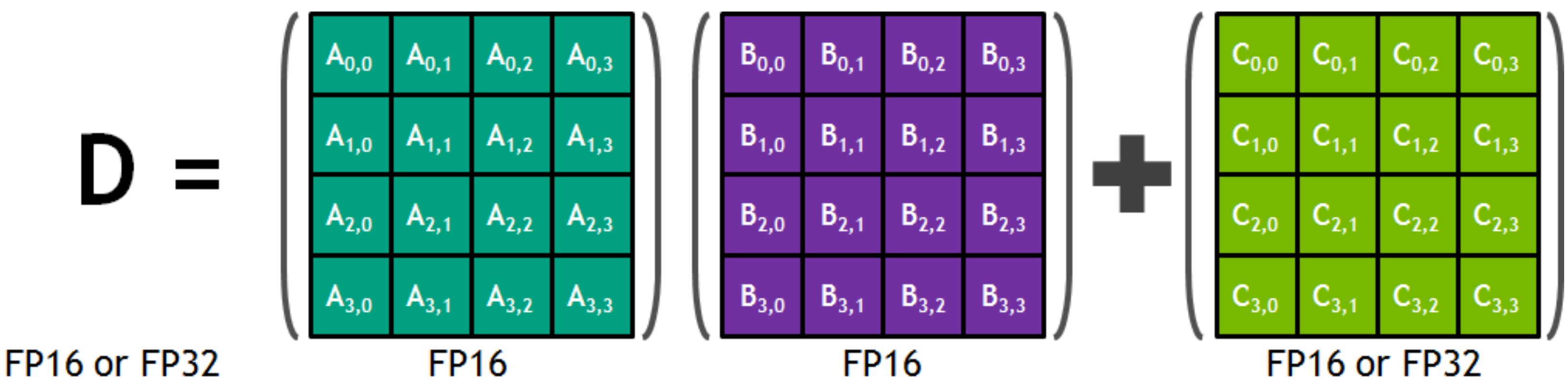
- 3-4x speedups in training wall time
- Reduced memory usage ==> bigger batch sizes
- No architecture changes required

Case study: Neural Machine Translation

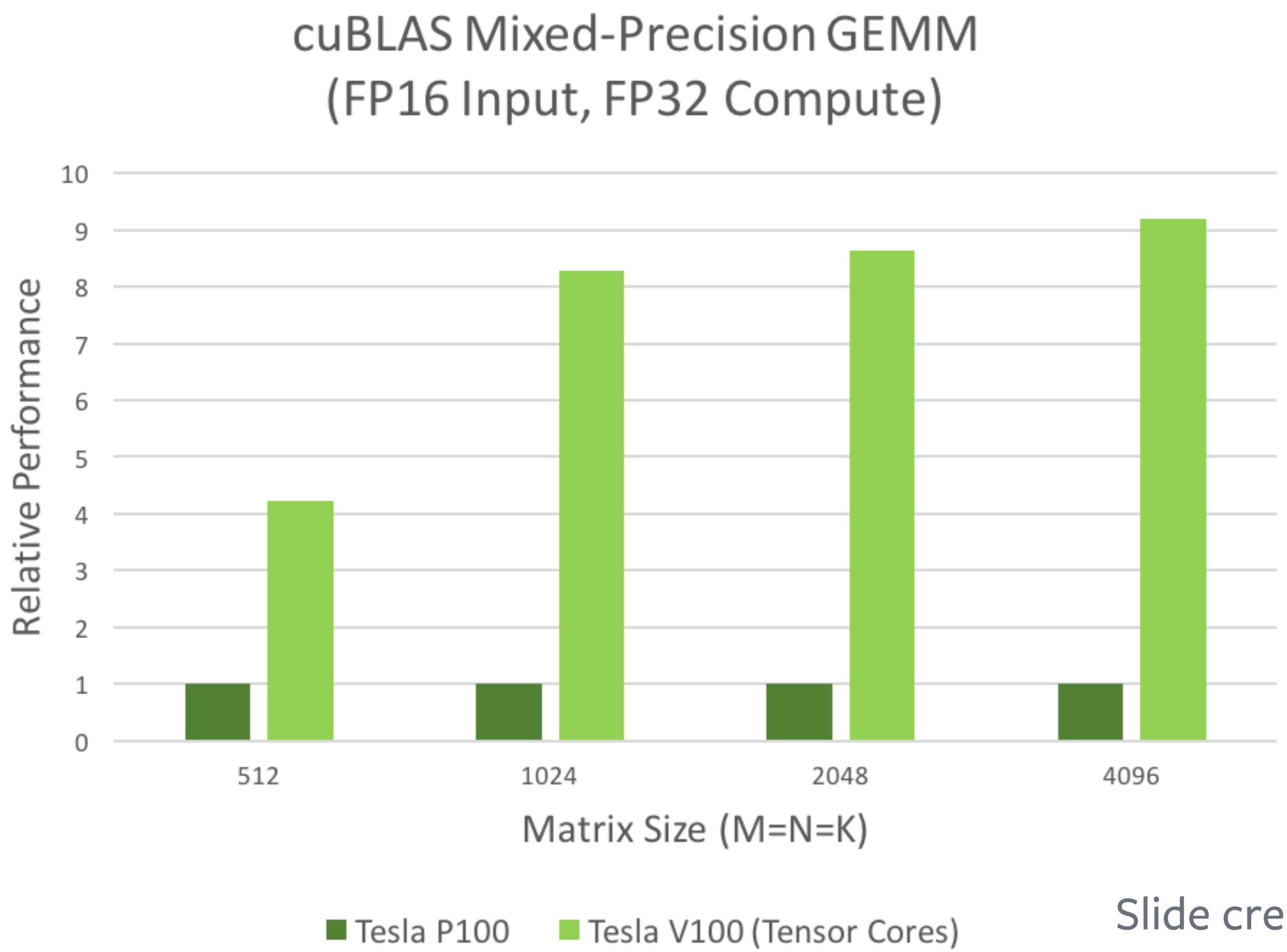
- Train models in 30 minutes instead of 1 day+
- Semi-supervised training over much larger datasets

What are Tensor Cores?

- Optimized hardware units for mixed precision matrix-multiply-and-accumulate: D = A * B + C



Slide credit: Nvidia



Slide credit: Nvidia

If only it were this easy...

```
model.half()
```

Why not pure FP16?

FP16 has insufficient range/precision for some ops

Better to leave some ops in FP32:

- Large reductions, e.g., norms, softmax, etc.
- Pointwise ops where $|f(x)| >> |x|$, e.g., exp, pow, log, etc.

Why not pure FP16?

In practice, **pure FP16 hurts optimization**.

According to Nvidia:

- Sum of FP16 values whose ratio is $>2^{11}$ is just the larger value
- Weight update: if `w >> lr*dw` then update doesn't change `w`

Why not pure FP16?

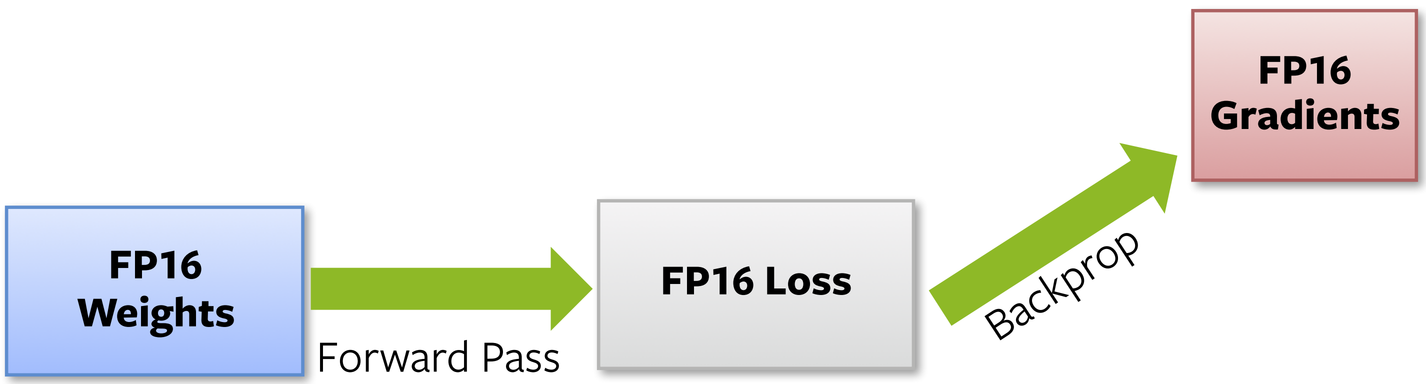
Solution: mixed precision training

Optimize in FP32 and use FP16 for almost* everything else

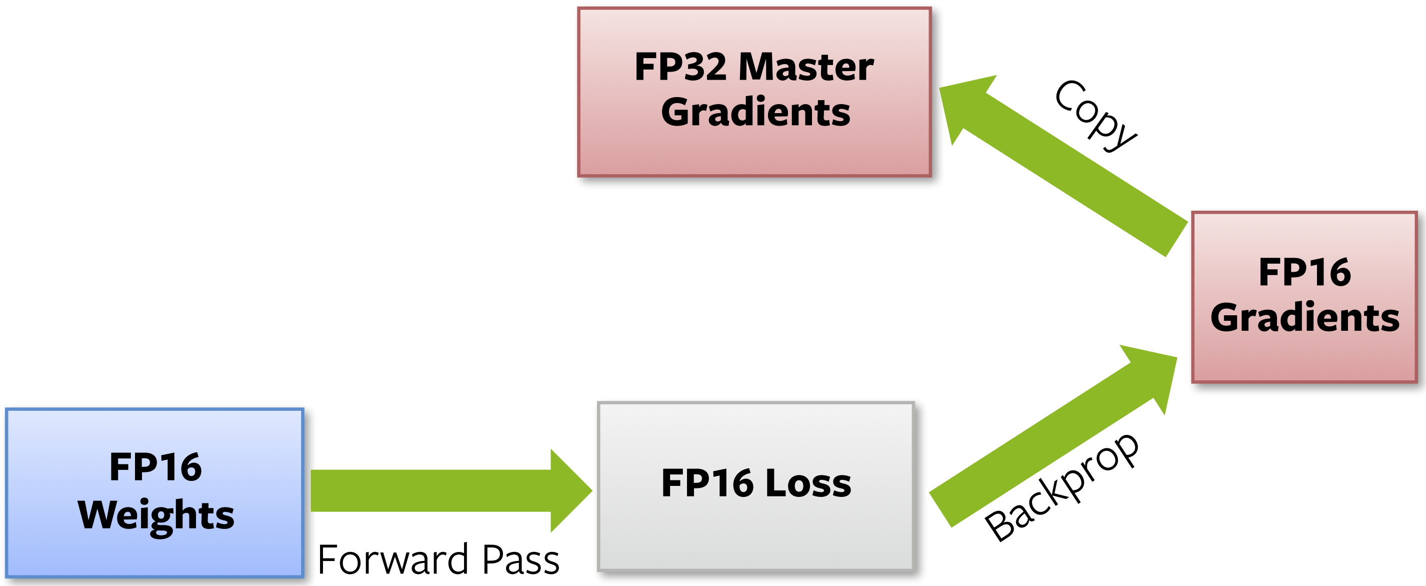
* Some operations should still happen in FP32:

- Large reductions, e.g., norms, softmax, etc.
- Pointwise ops where |f(x)| >> |x|, e.g., exp, pow, log, etc.

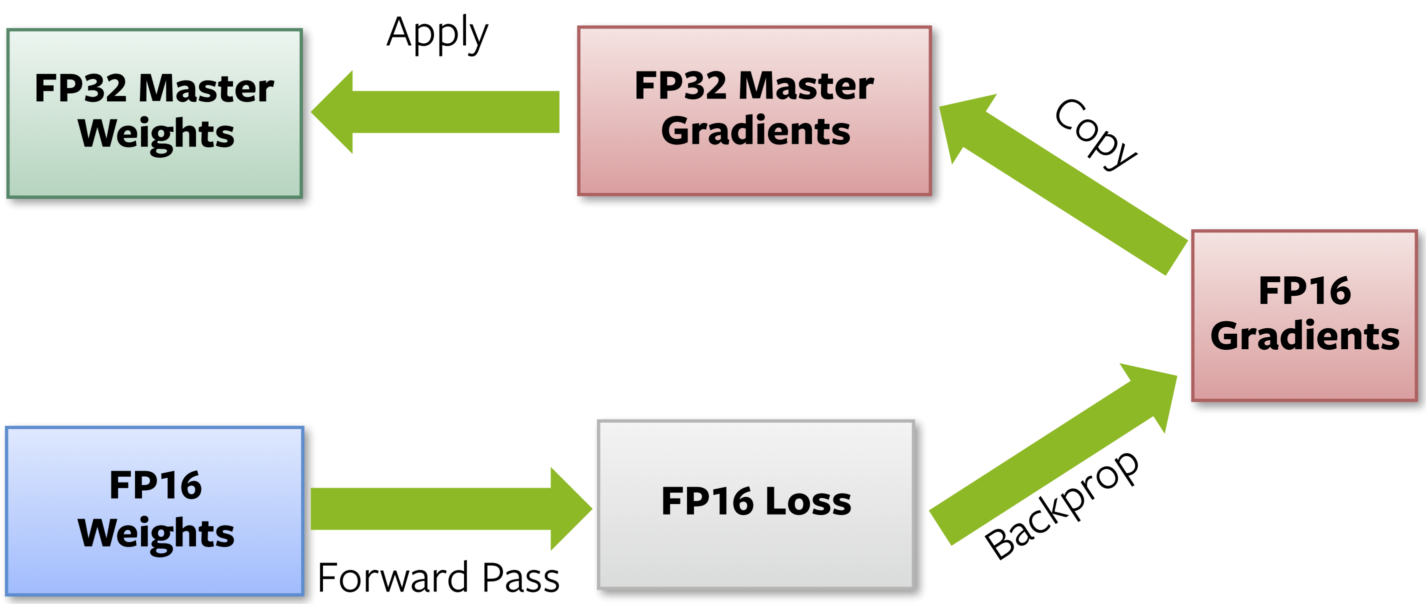
Optimizing in FP32



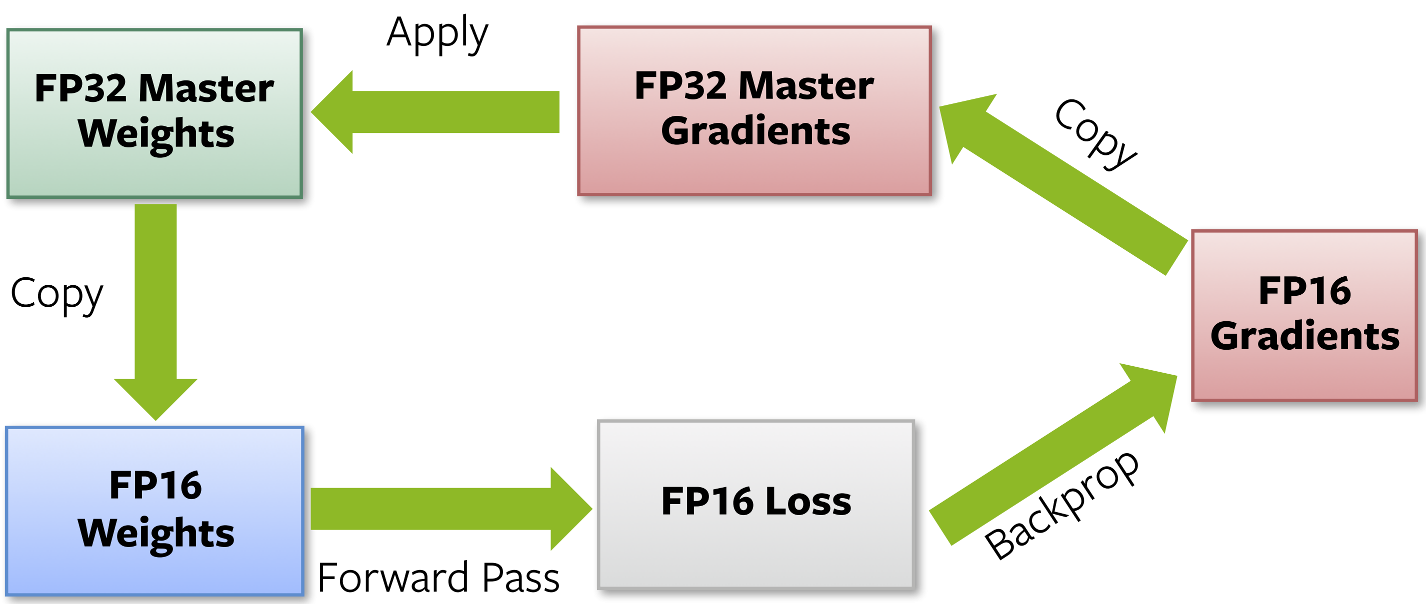
Optimizing in FP32



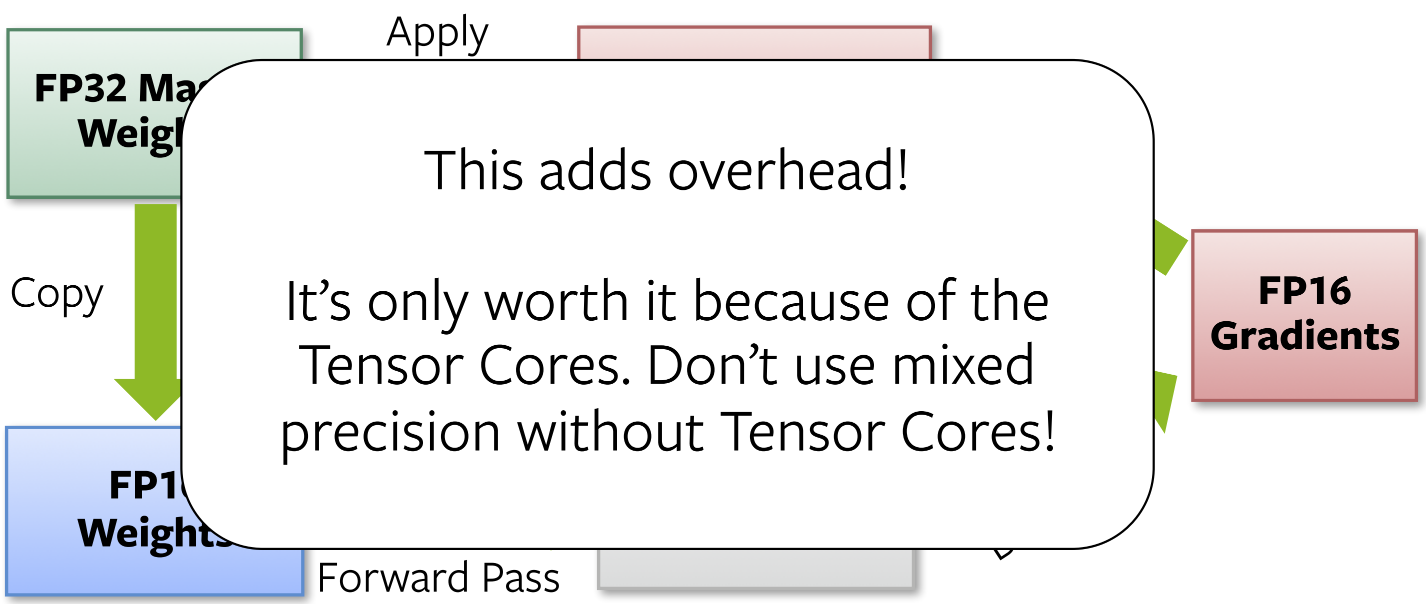
Optimizing in FP32



Optimizing in FP32

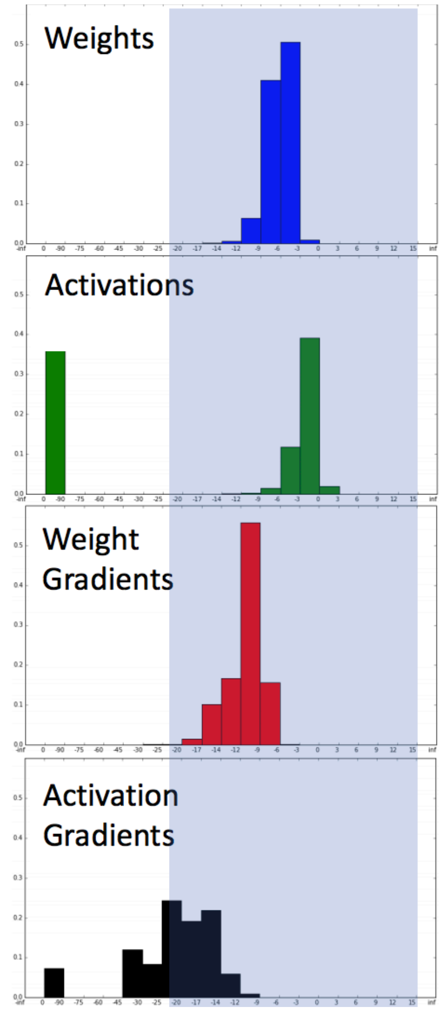


Optimizing in FP32



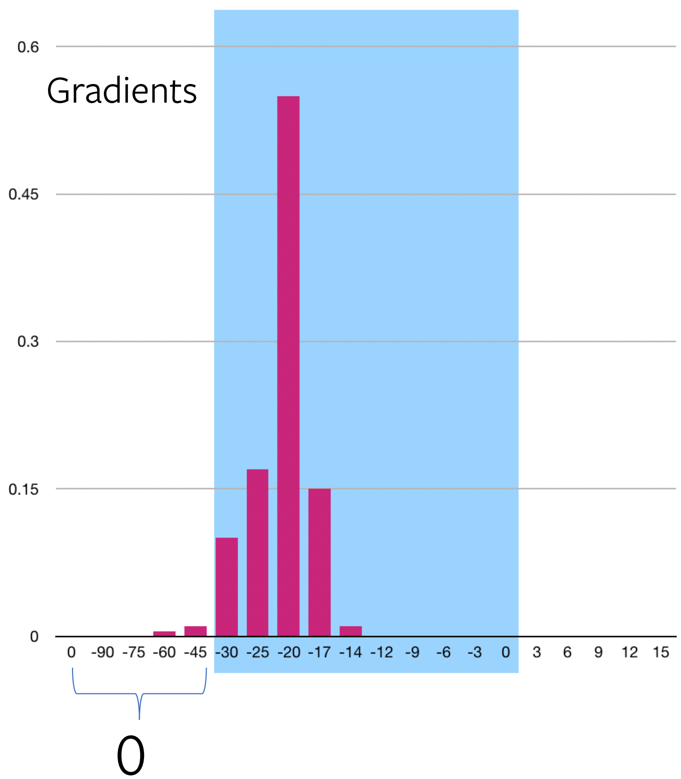
Gradient underflow

- FP16 has a smaller representable range than FP32 (shown in blue)
- In practice gradient are quite small, so there's a risk of underflow



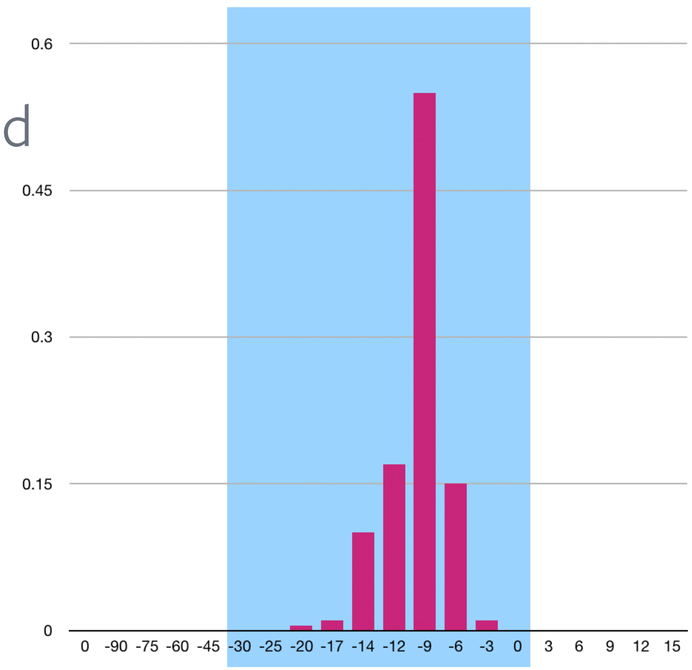
Gradient underflow

If we scale the loss up by K,
by the chain rule of derivatives,
gradients will be K times bigger

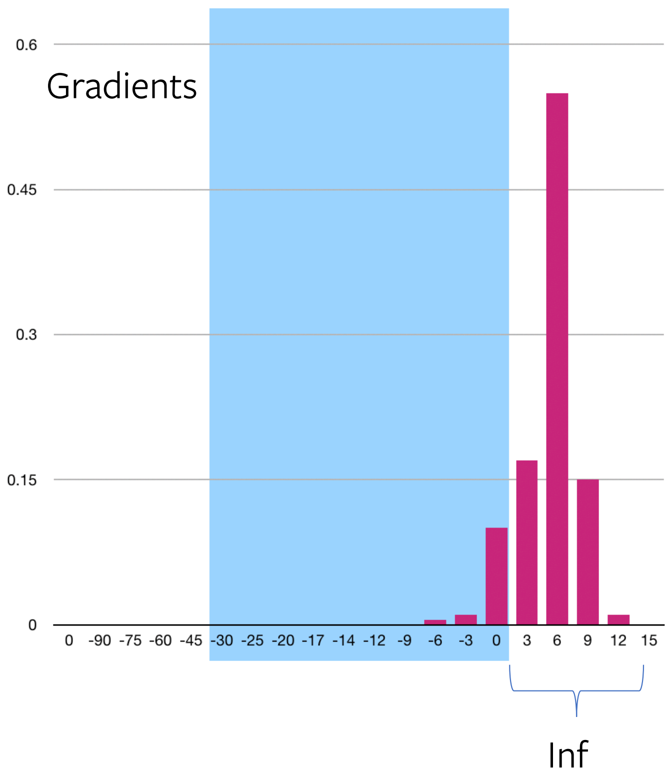


Underflow can **not** be detected

But if we scale loss up

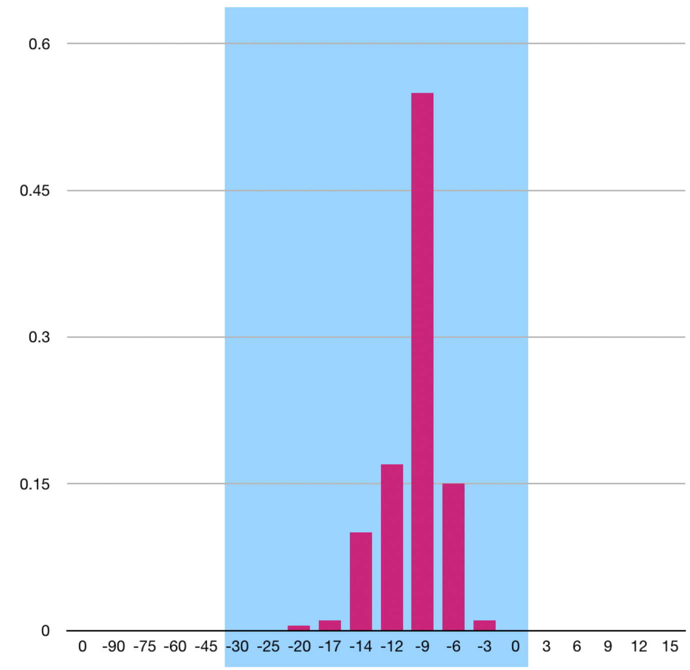


Gradient overflow

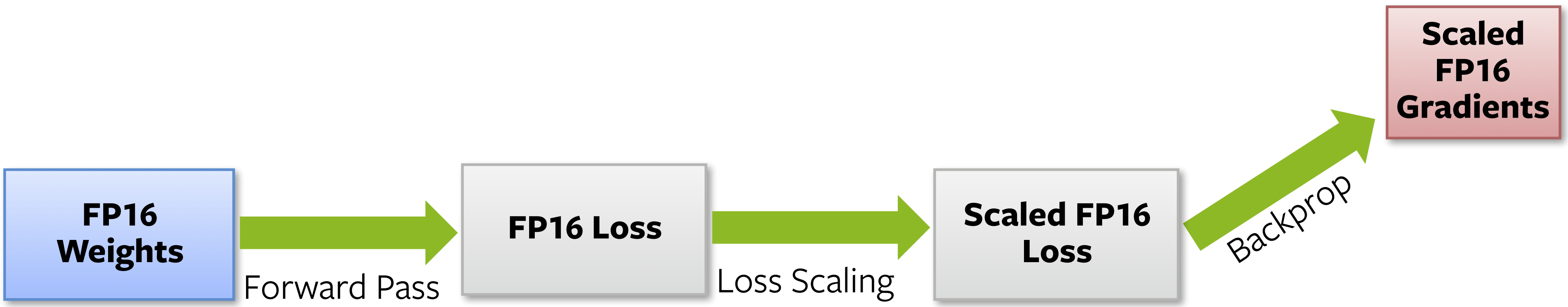


If overflow detected

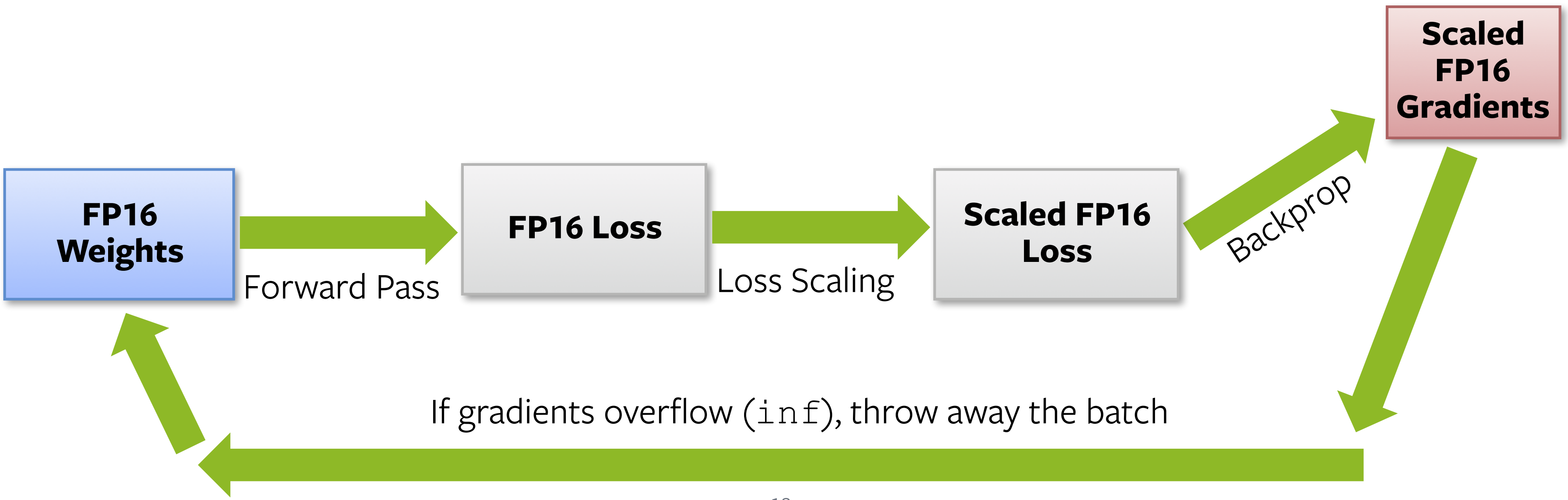
Scale the loss down



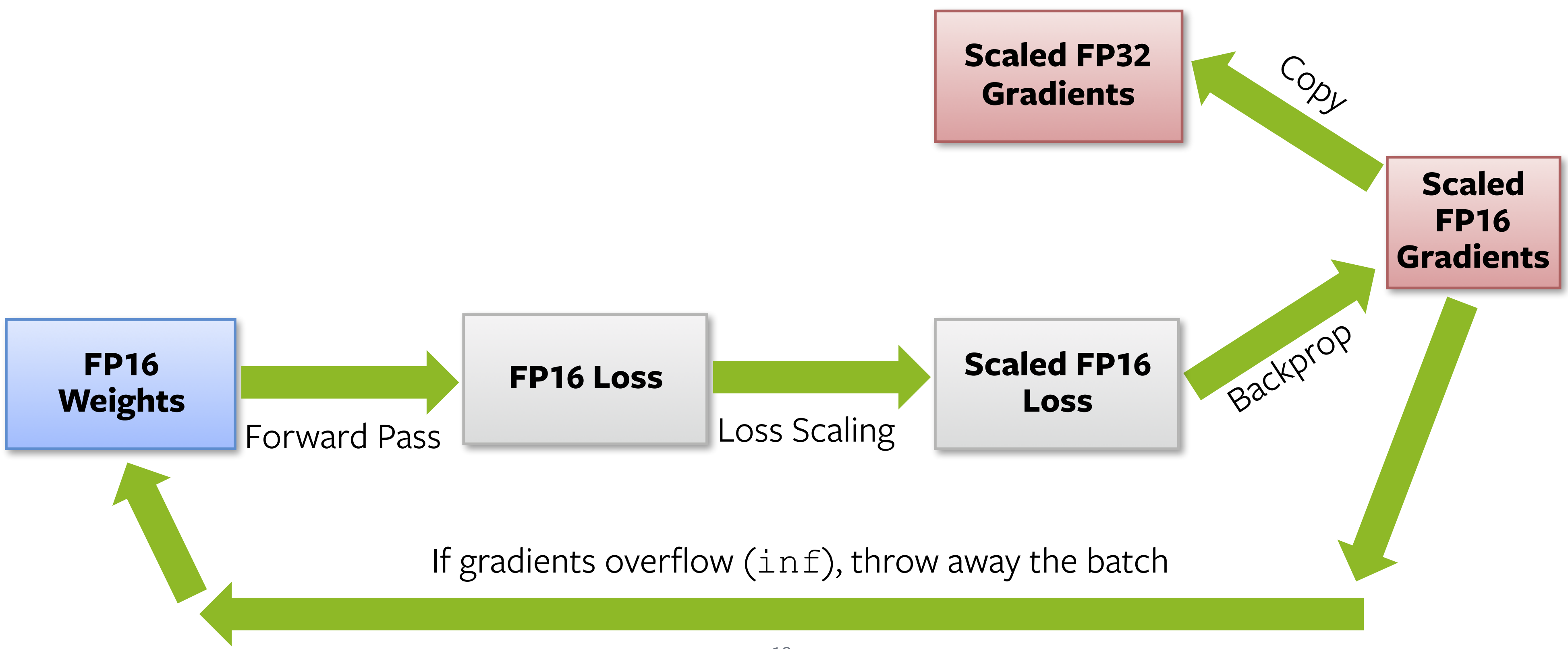
Avoiding under/overflow by loss scaling



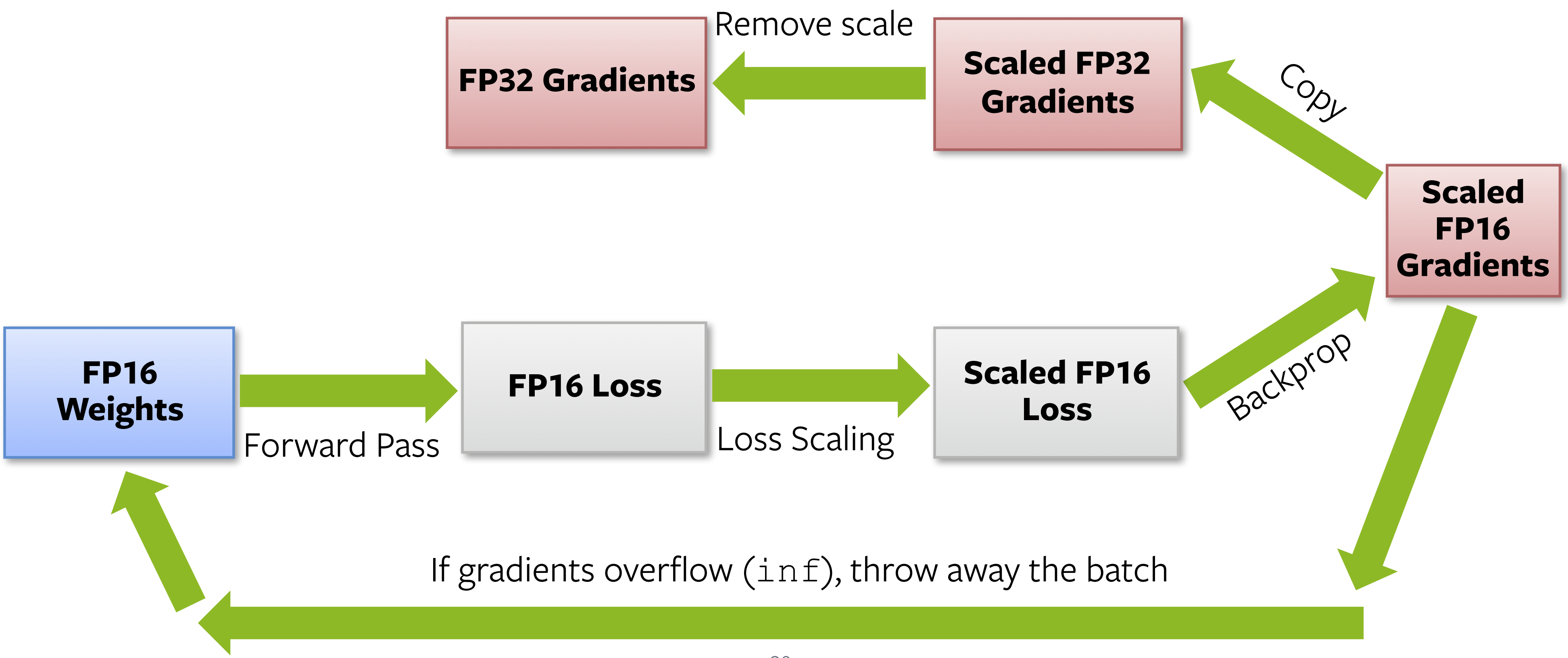
Avoiding under/overflow by loss scaling



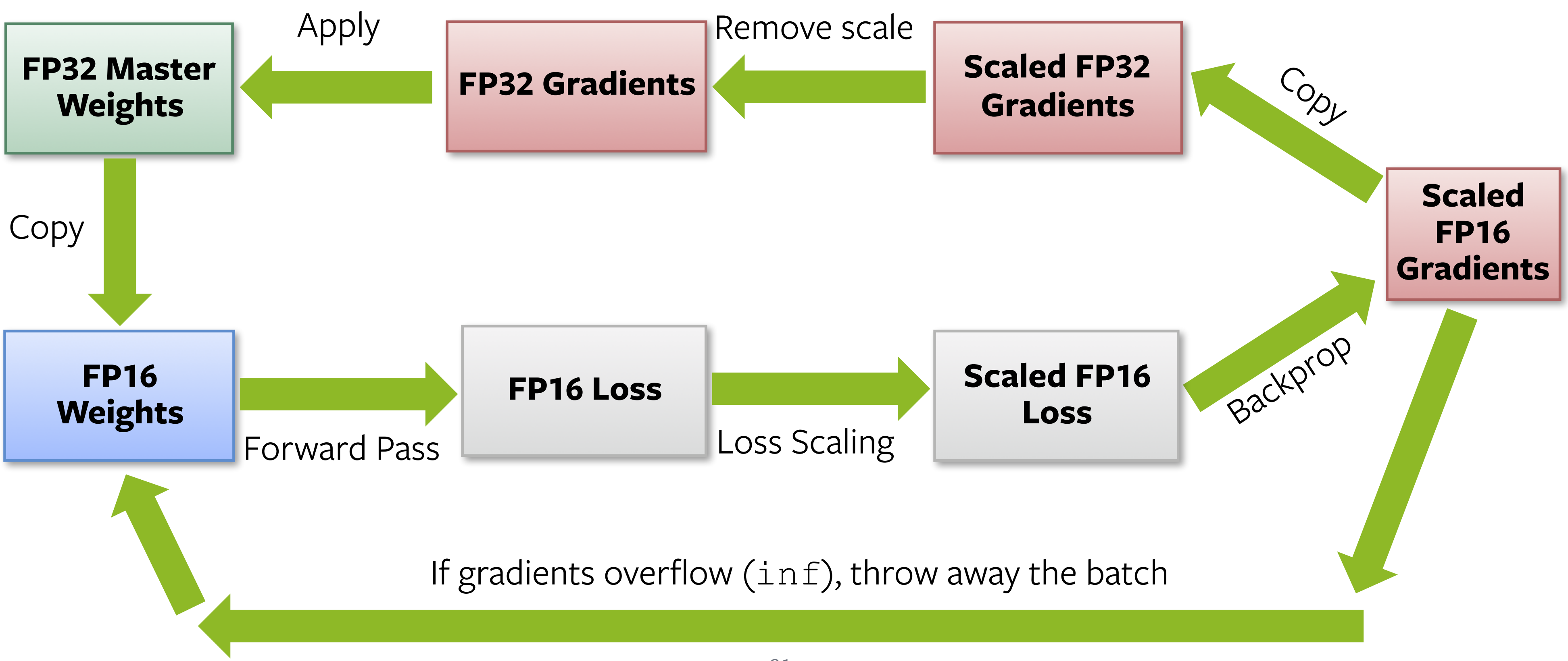
Avoiding under/overflow by loss scaling



Avoiding under/overflow by loss scaling



Avoiding under/overflow by loss scaling



How to pick the scaling constant (K)

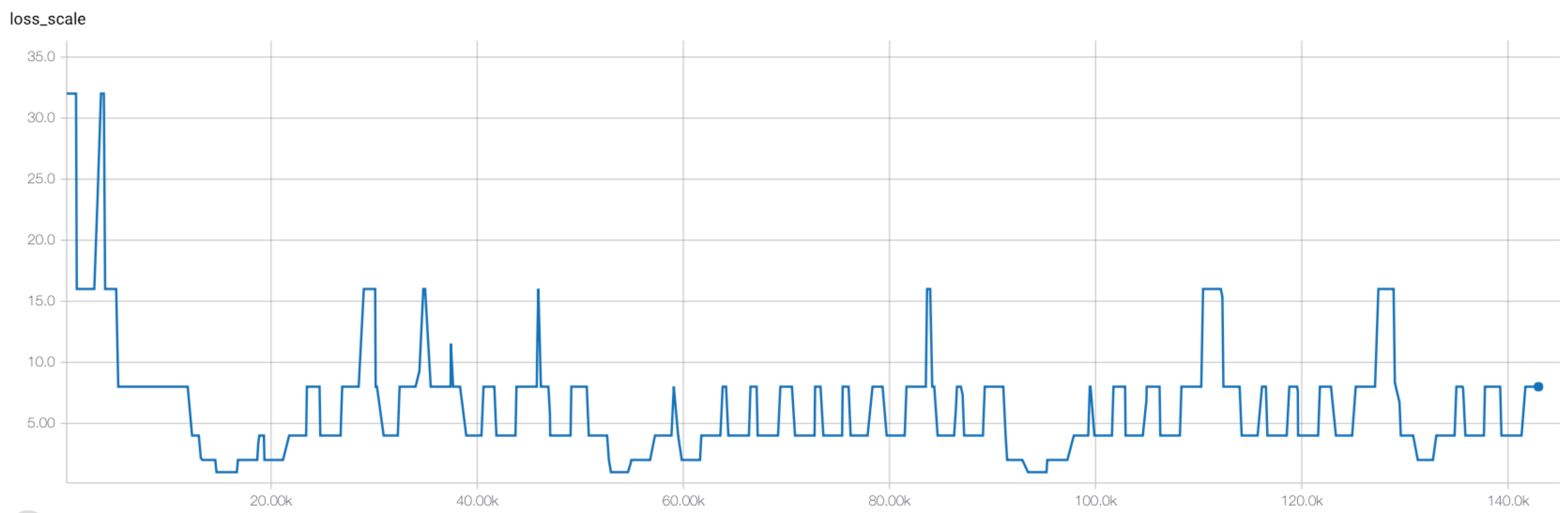
- Too small and gradient will underflow
- Too big and we'll waste compute due to overflow
- In practice the optimal scaling constant changes during training
- We can adjust it dynamically!

Dynamic loss scaling

- Every time the gradient overflows (`inf`), reduce the scaling constant by a factor of 2

- If the gradients haven't overflowed in the last N updates (~1000), then increase the scaling constant by a factor of 2

Dynamic loss scaling



So far...

Tensor Cores make FP16 ops 4-9x faster

Mixed precision training:

- Forward/backward in FP16
- Optimize in FP32
- Requires maintaining two copies of the model weights
- Dynamically scale the loss to avoid gradient under/overflow

One more thing about FP16...

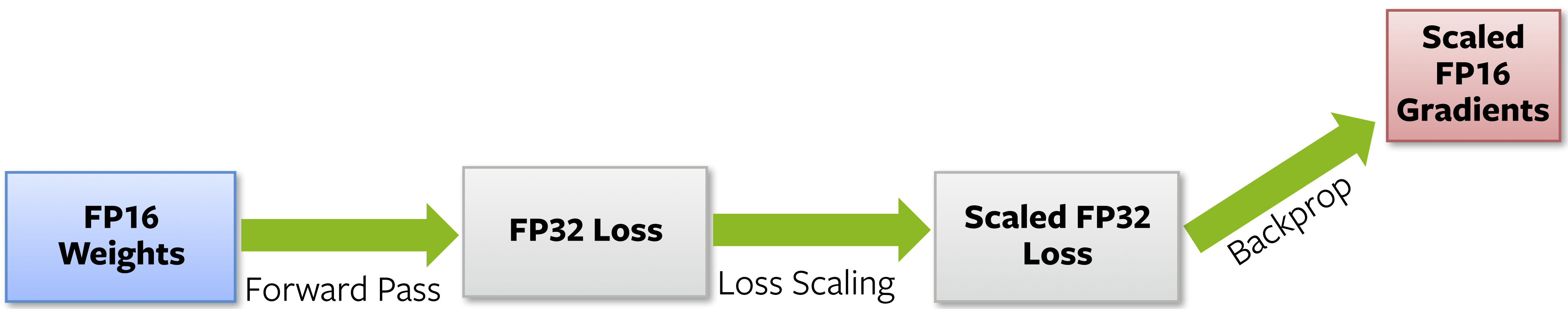
For maximal safety, perform ops that sum many values in FP32

- e.g., normalization layers, softmax, L1 or L2 norm, etc.
- This includes most Loss layers, e.g., CrossEntropyLoss

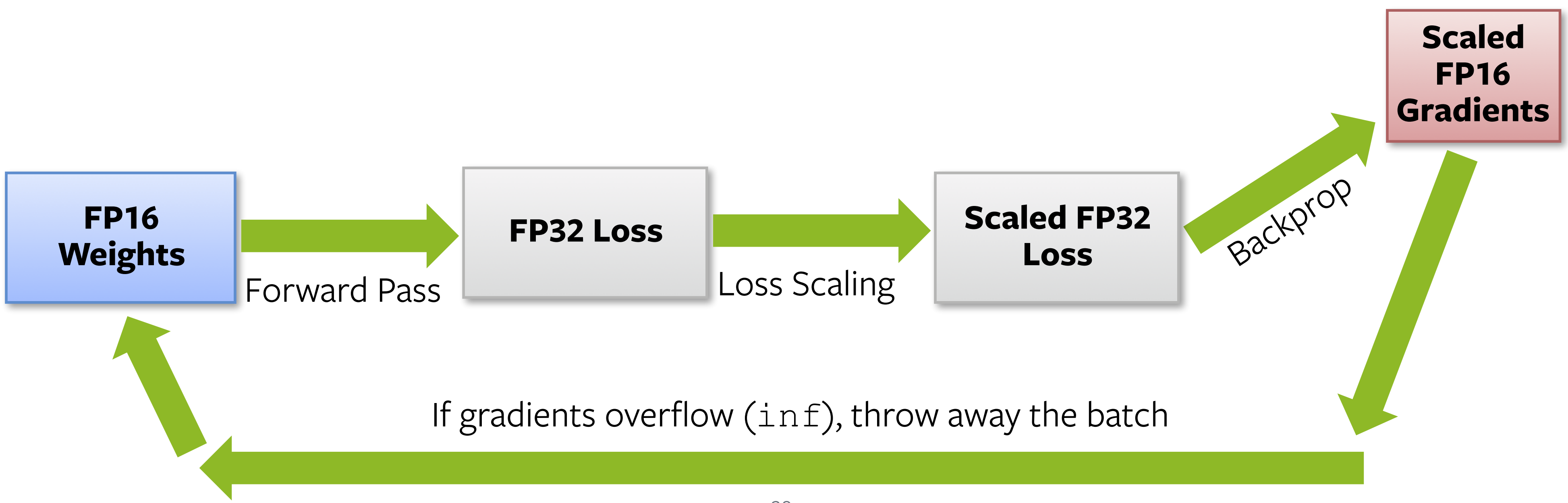
General advice: compute your loss in FP32 too

The full picture

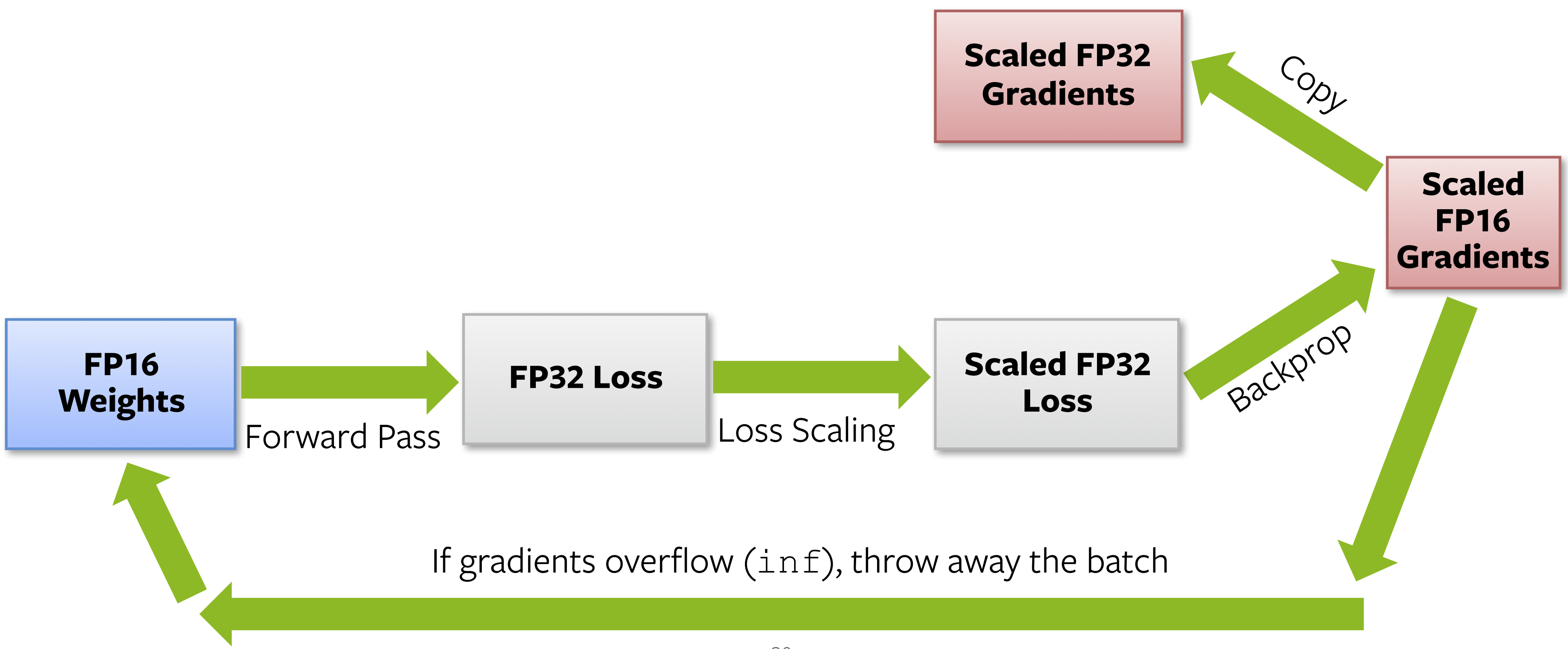
The full picture



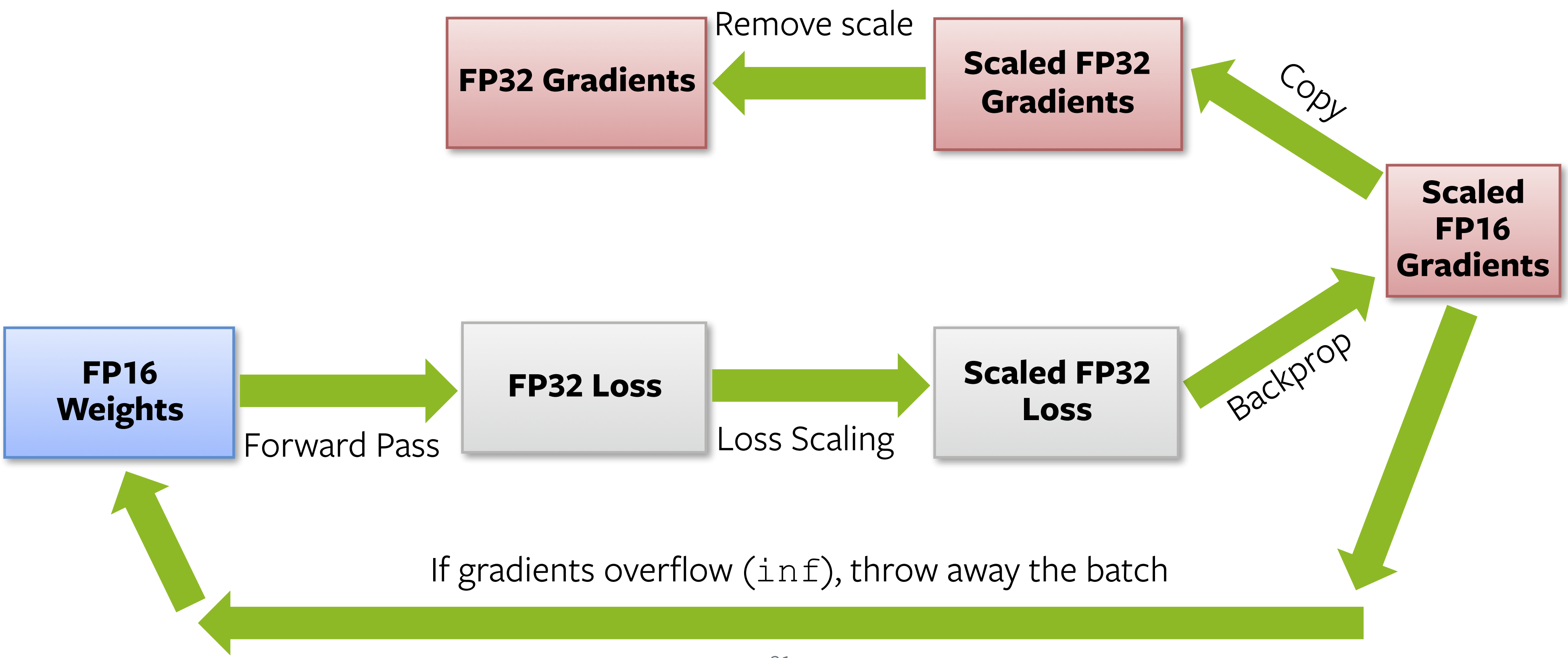
The full picture



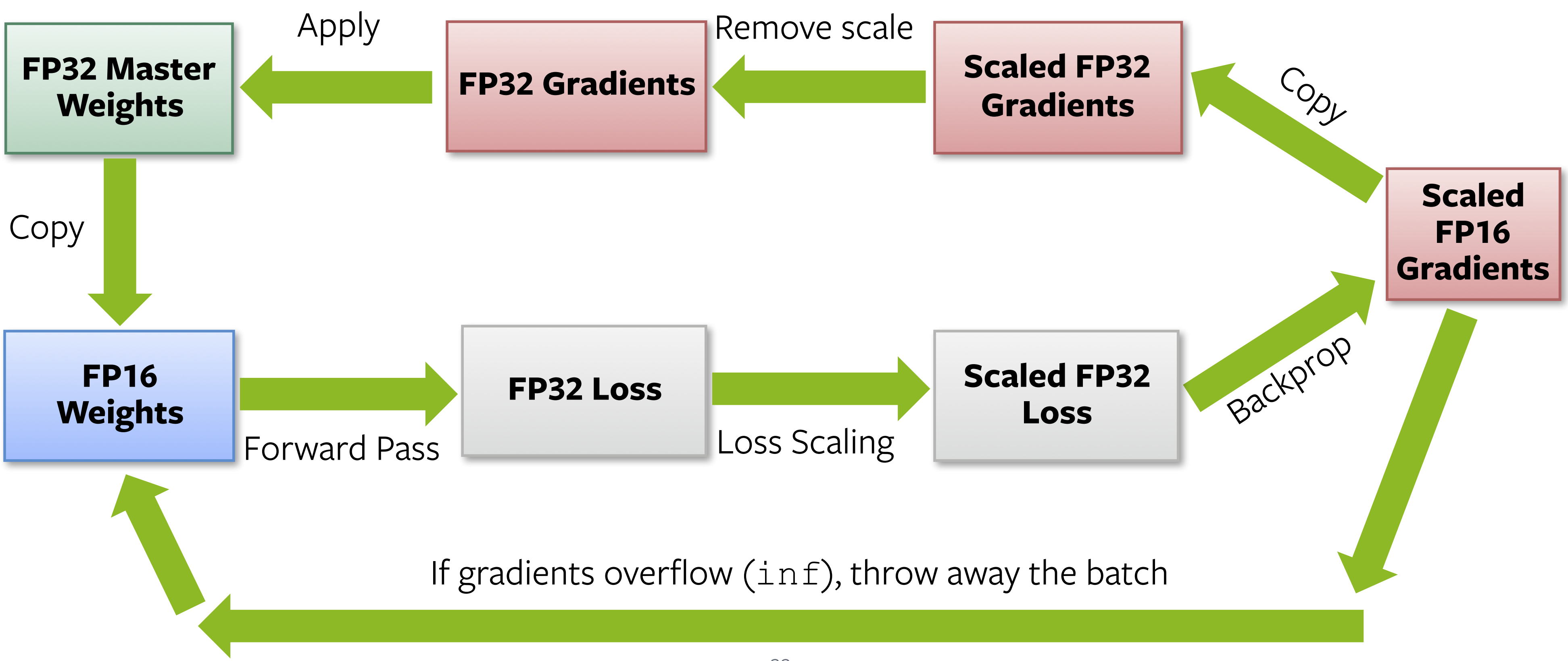
The full picture



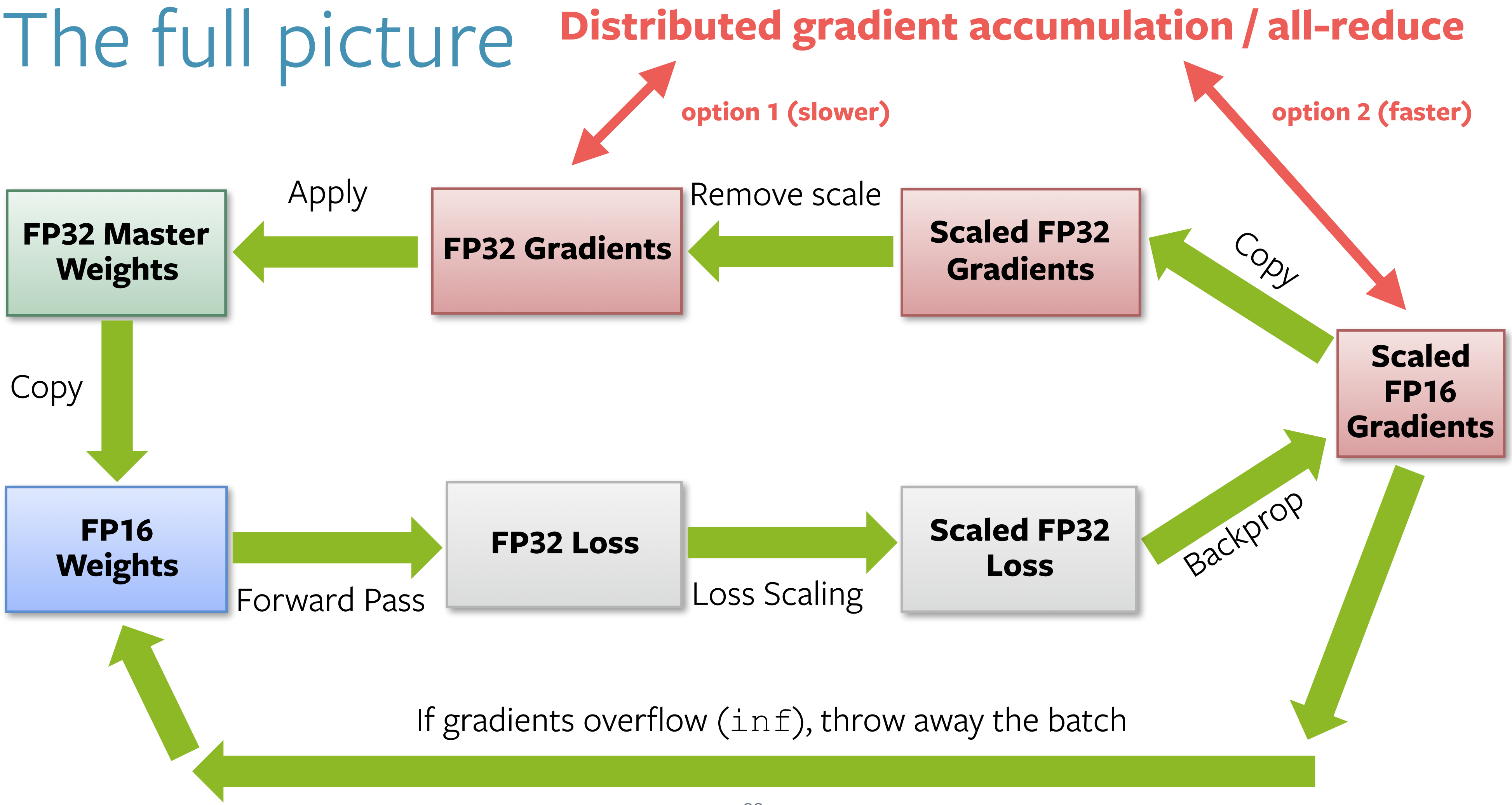
The full picture



The full picture



The full picture



In PyTorch

To automate the recipe, start with Nvidia's `apex.amp` library:

```
from apex import amp
optim = torch.optim.Adam(…)
model, optim = amp.initialize(model, optim, opt_level="O1")
(…)
with amp.scale_loss(loss, optim) as scaled_loss:
     scaled_loss.backward()
optim.step()
```

Making it even faster

`apex.amp` supports different optimization levels

`opt_level="O1"` is conservative and keeps many ops in FP32

`opt_level="O2"` is faster, but may require manually converting some ops to FP32 to achieve good results

More details at: `https://nvidia.github.io/apex/`

Making it even faster

A useful pattern:

```
x = torch.nn.functional.softmax(x, dtype=torch.float32).type_as(x)
```

When `x` is FP16 (i.e., a `torch.HalfTensor`):

- Computes the softmax in FP32 and casts back to FP16

When `x` is FP32 (i.e., a `torch.FloatTensor`):

- No impact on speed or memory

One more thing...

Must have GPU with Tensor Cores (Volta+), CUDA 9.1 or newer

Additionally:

- Batch size should be a multiple of 8
- M, N and K for matmul should be multiples of 8
- Dictionaries/embed layers should be padded to be a multiple of 8

Summary

Mixed precision training gives:

- Tensor Cores make FP16 ops 4-9x faster
- No architecture changes required
- Use Nvidia's `apex` library

Tradeoffs:

- Some extra bookkeeping required (mostly handled by `apex`)
- Best perf requires manual fixes for softmax, layernorm, etc.

Scaling
Machine Translation

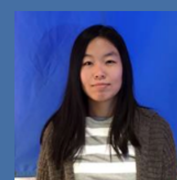

Myle Ott


Sergey Edunov


David Grangier


Michael Auli


Teng Li


Ailing Zhang


Shubho Sengupta

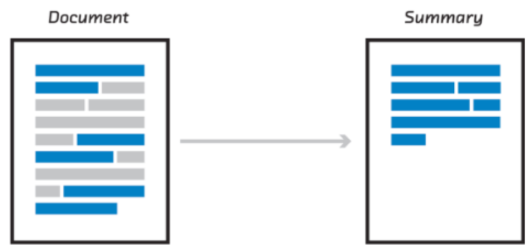
Sequence to Sequence Learning

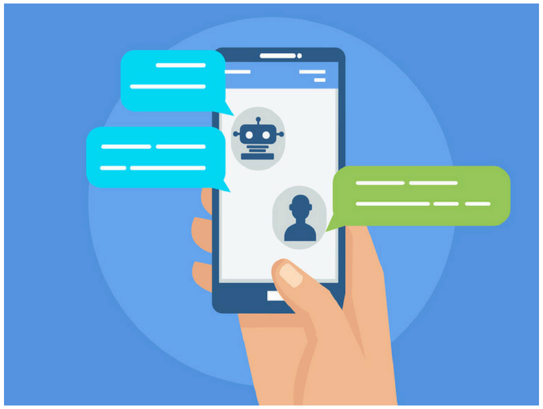
Bonjour à tous ! $\longrightarrow$ Hello everybody!

- **Sequence to sequence** mapping
- Input = sequence, output = sequence
- **Structured prediction** problem

Sequence to Sequence Learning

- machine translation
- text summarization
- writing stories
- question generation
- dialogue, chatbots
- paraphrasing
- ...



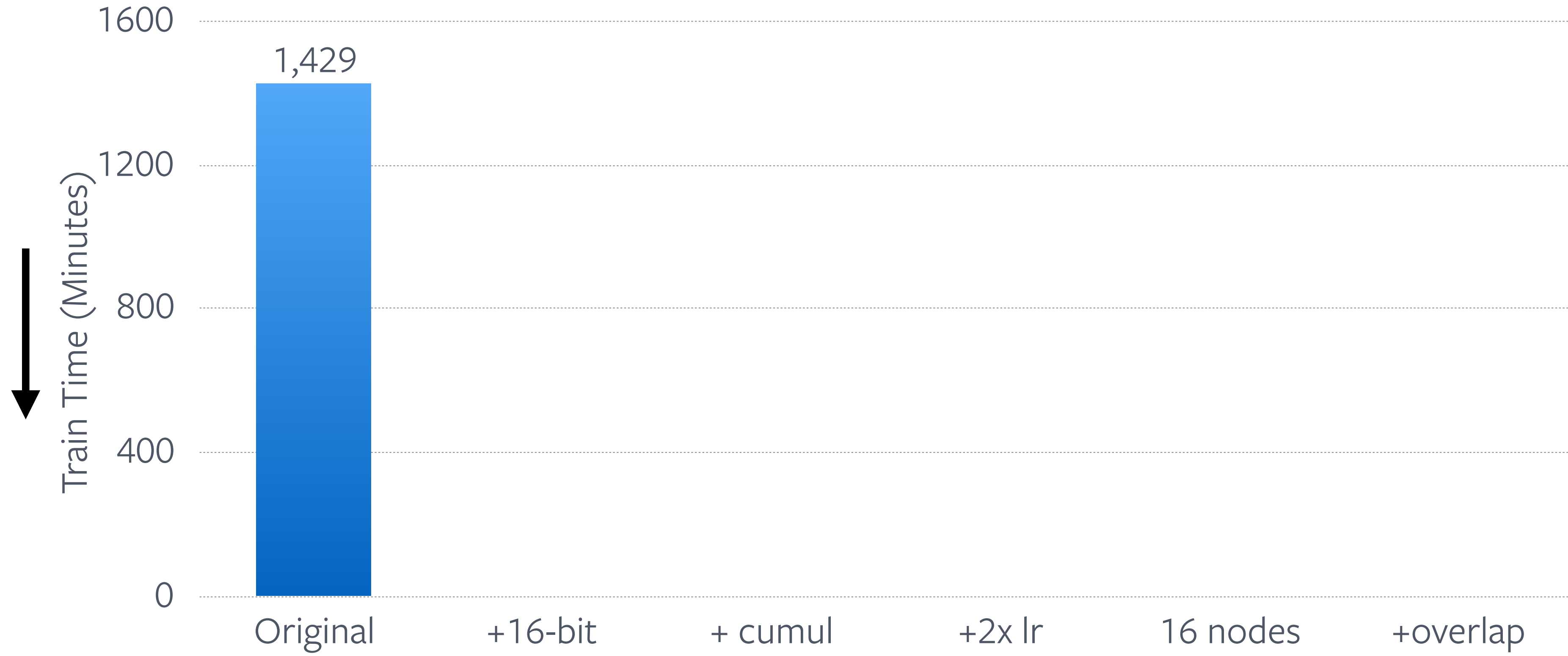


Why do we need to scale?

- Large benchmark ~2.4 billion words
  + much more unlabeled data
- Training time: CNNs up to 38 days on 8 M40 GPUs (Gehring et al., 2017)
- Train many models
- Support Multilingual training

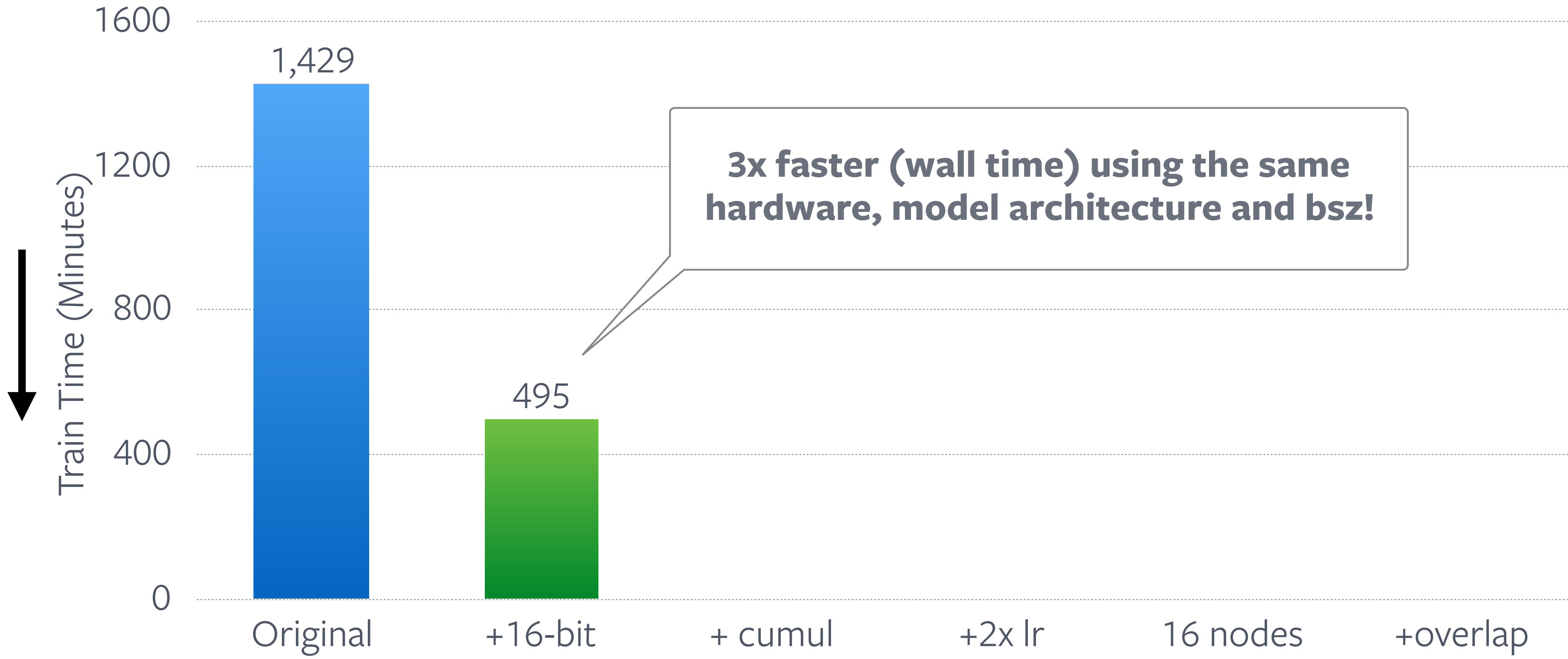
Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



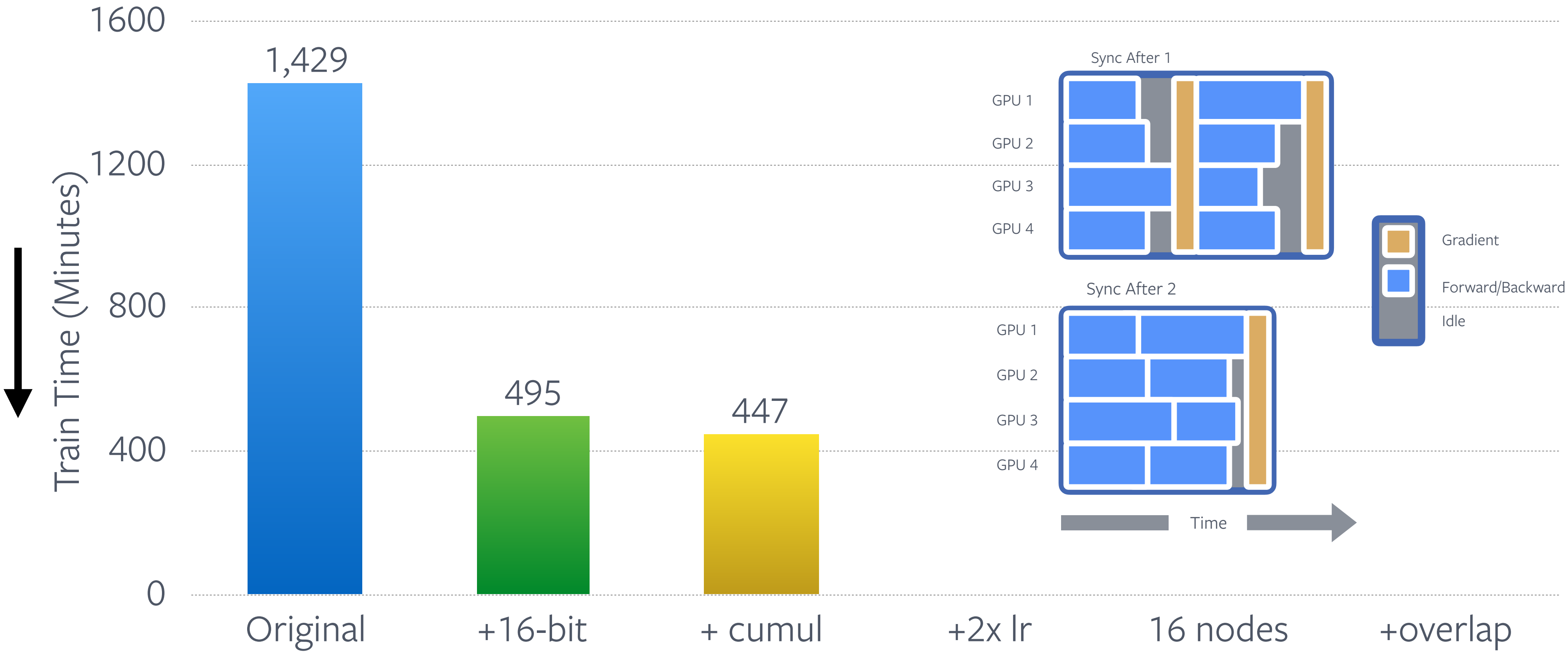
Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



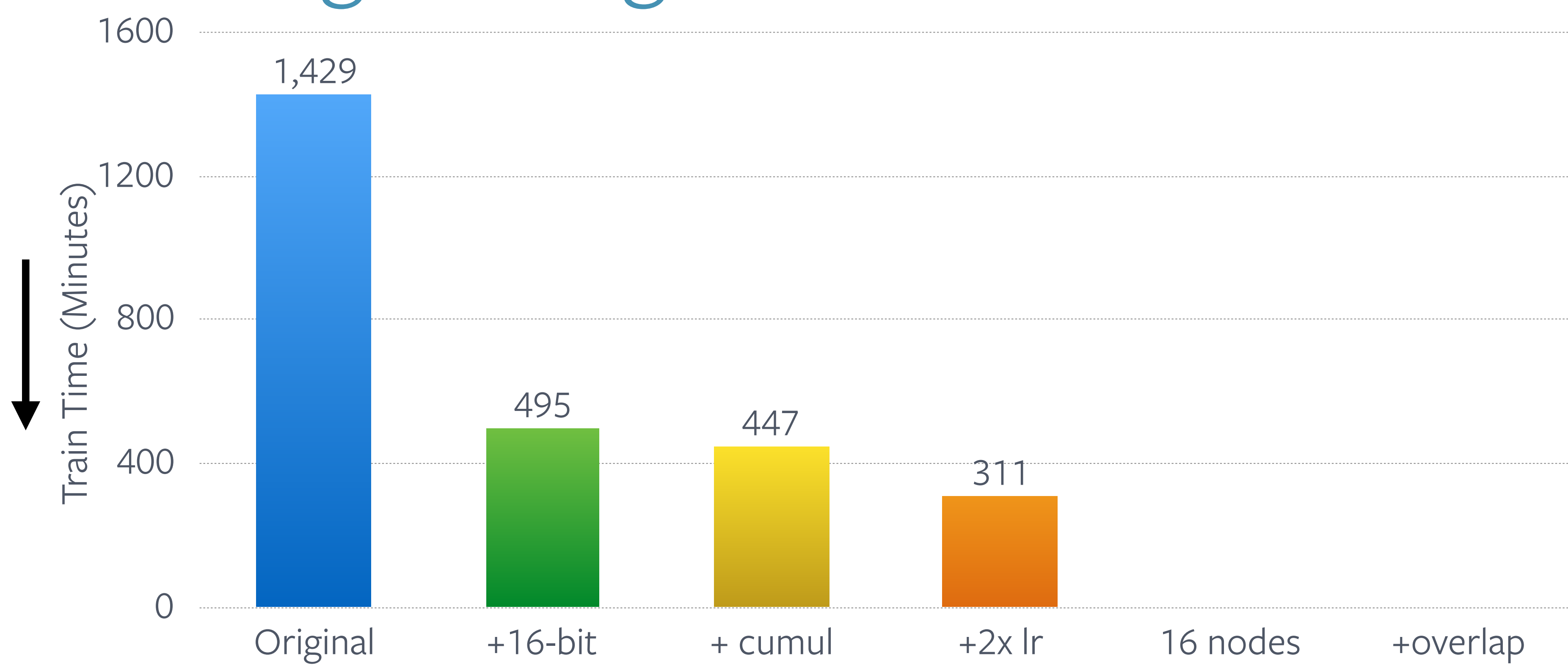
Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



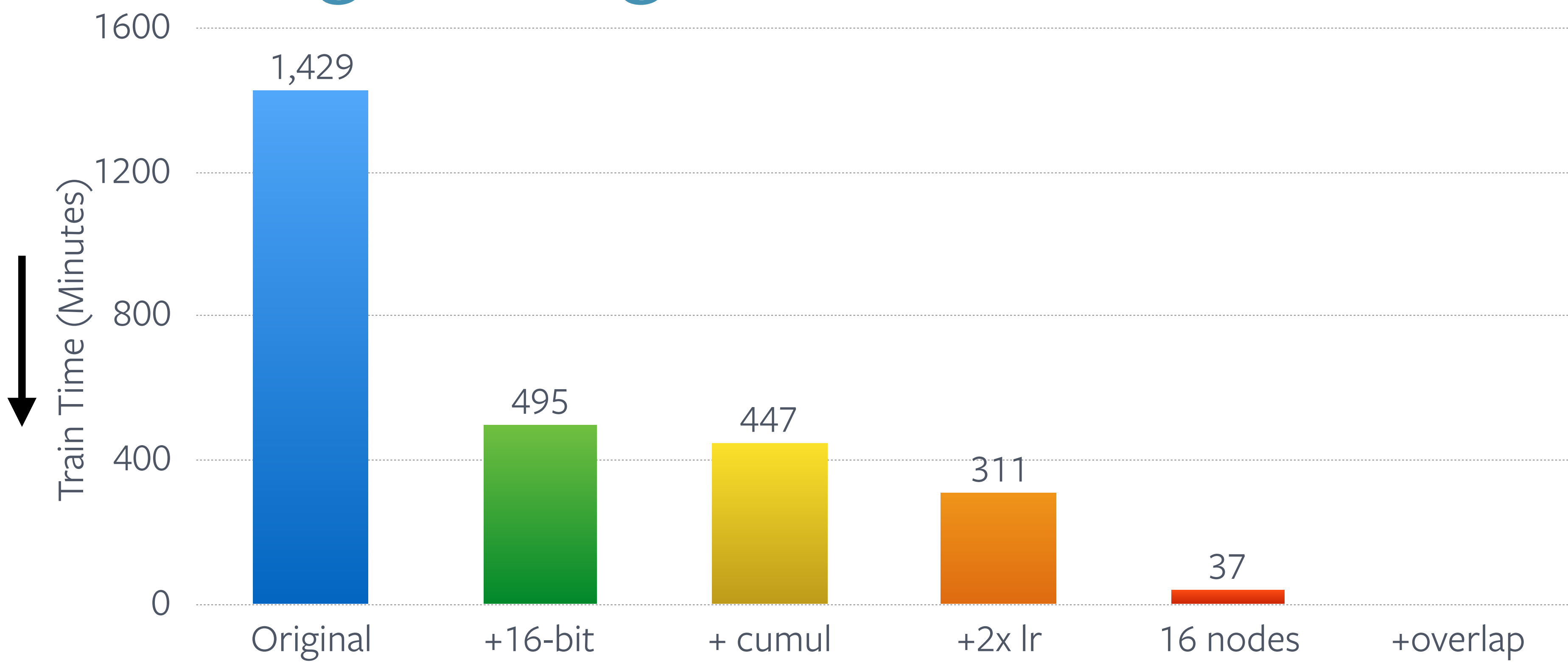
Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



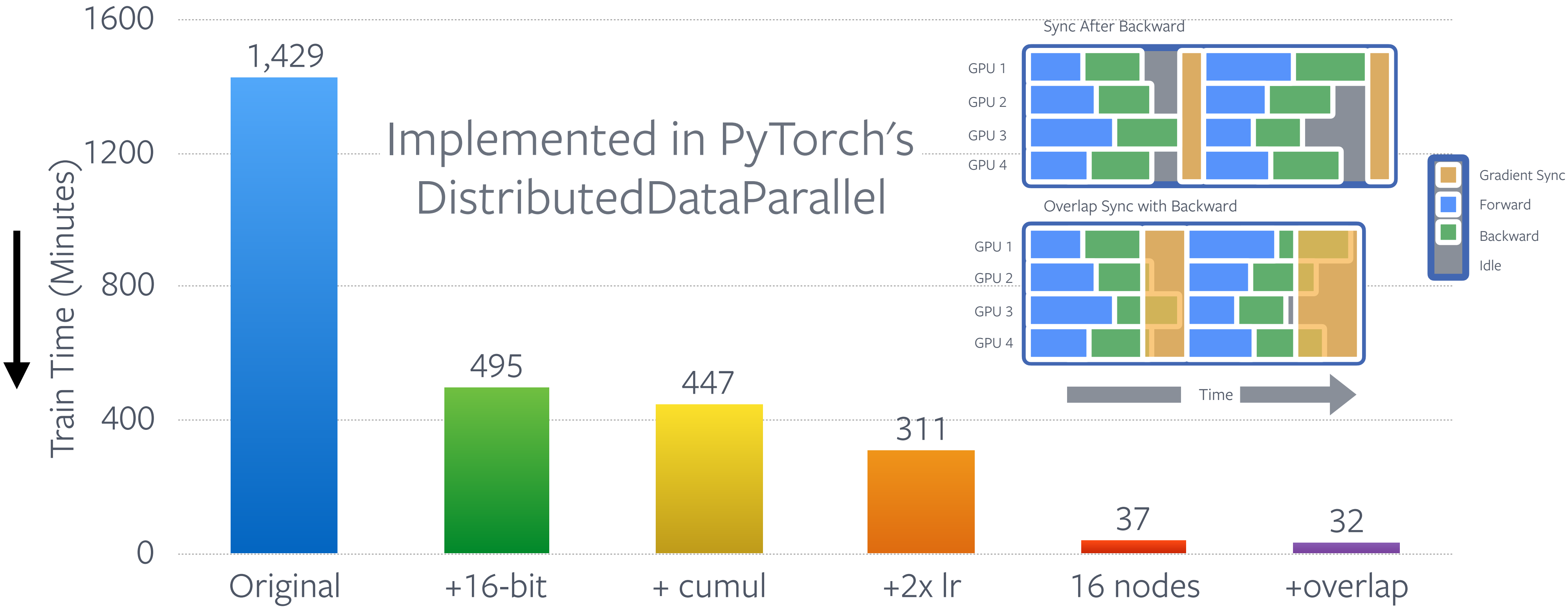
Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



Reducing training time

Time in minutes to train "Transformer" translation model on Volta V100 GPUs (WMT En-De)



Semi-supervised machine translation


Sergey Edunov

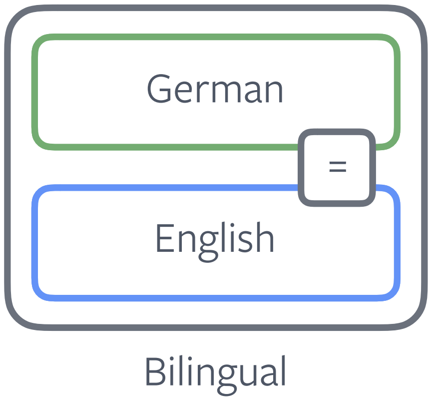

Myle Ott

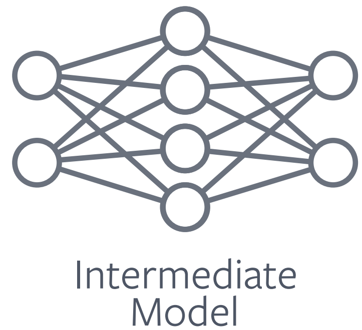

Michael Auli


David Grangier

Data augmentation for Translation

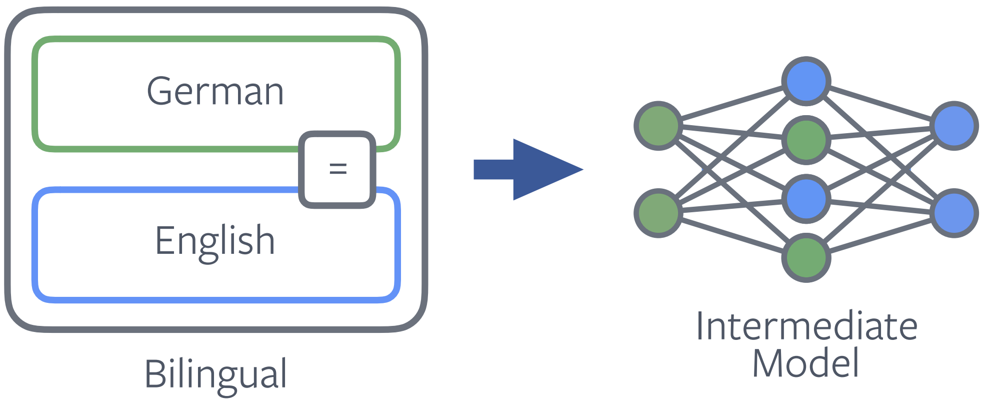
Back-translation (Bojar & Tamchyna, 2011; Sennrich et al., 2016)

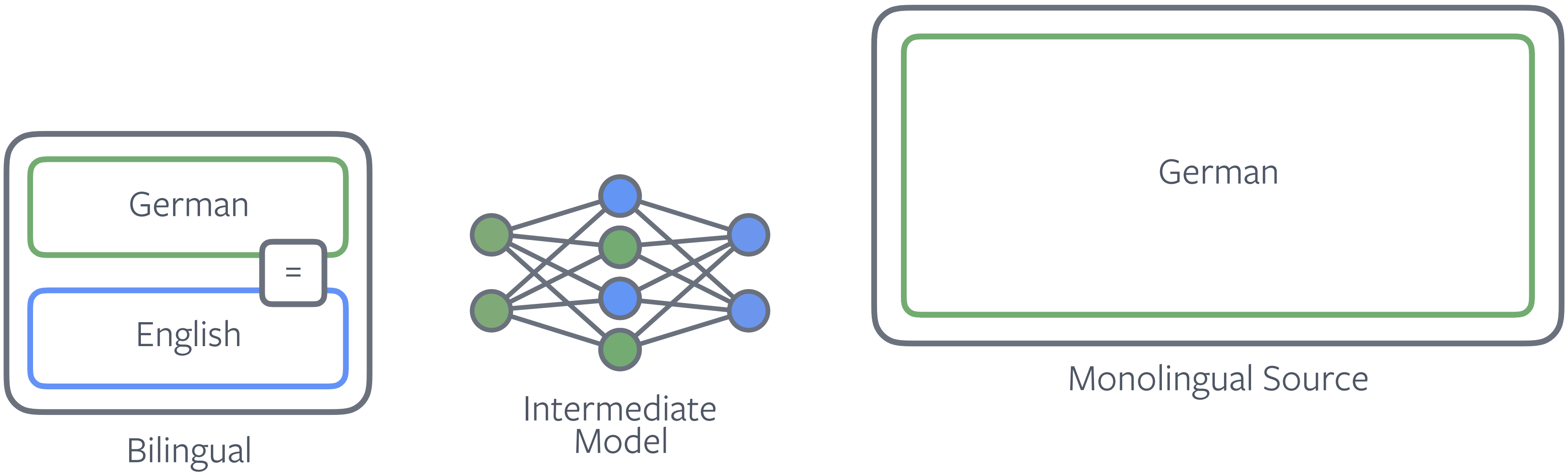


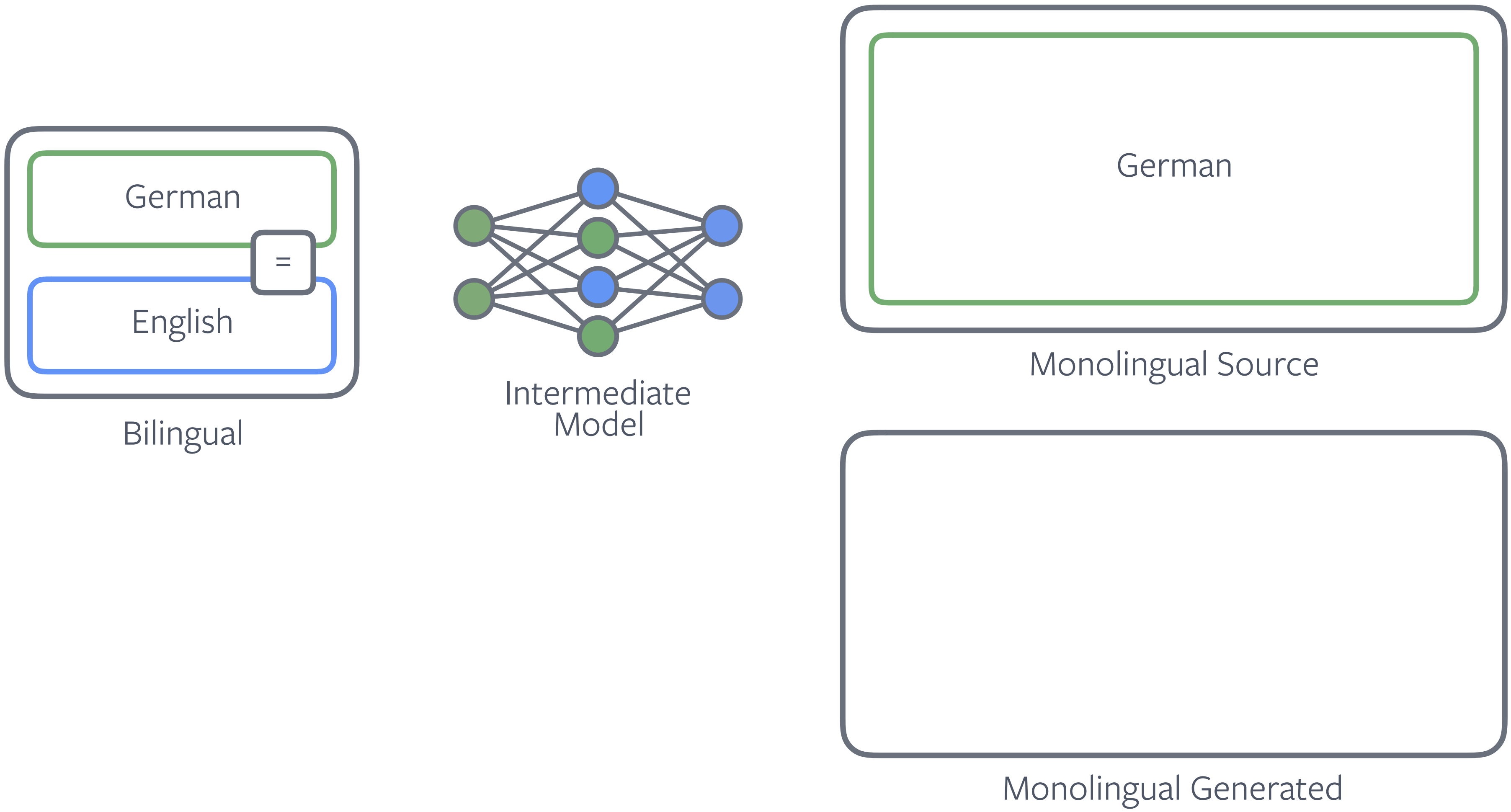


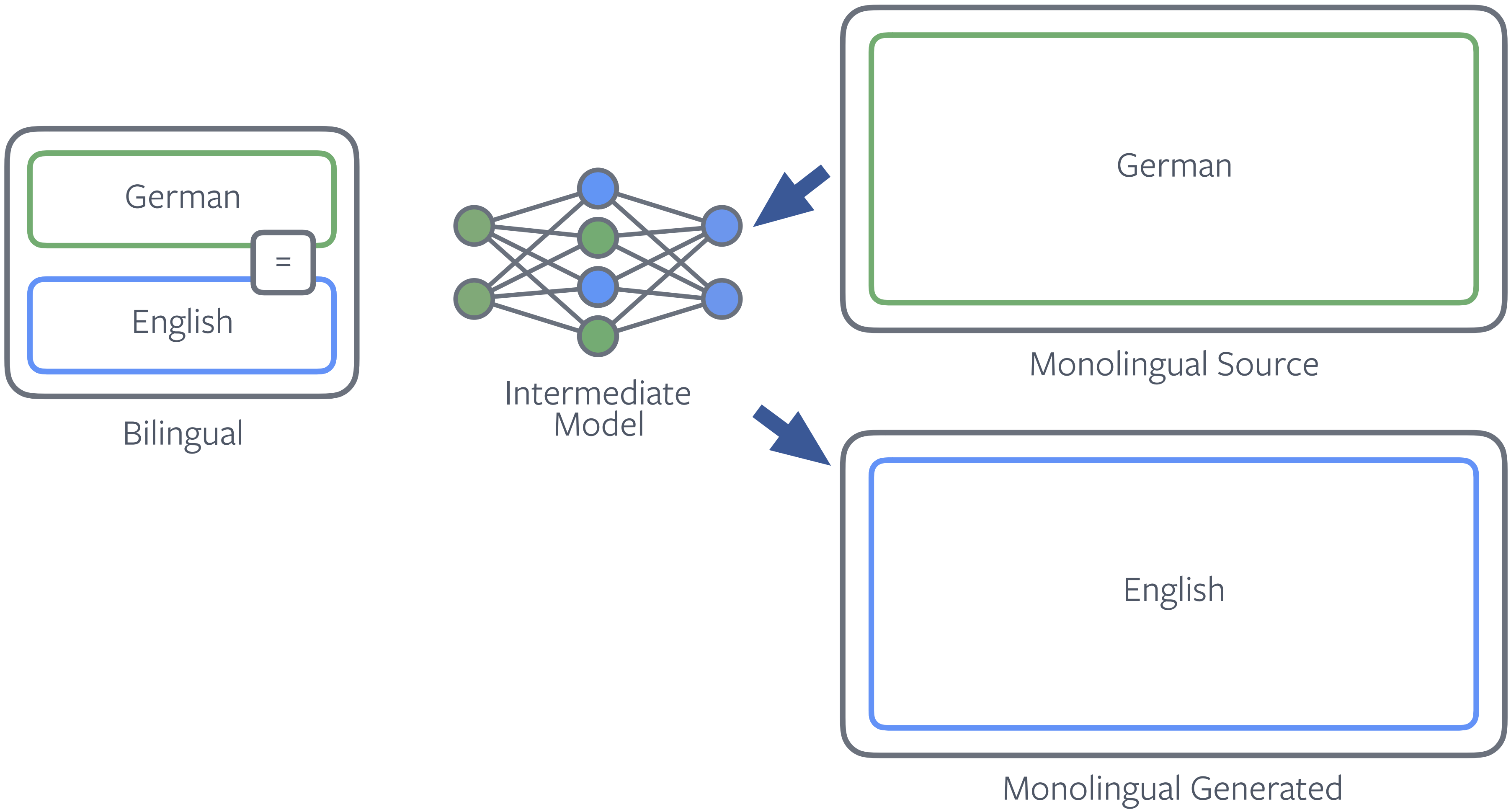
Data augmentation for Translation

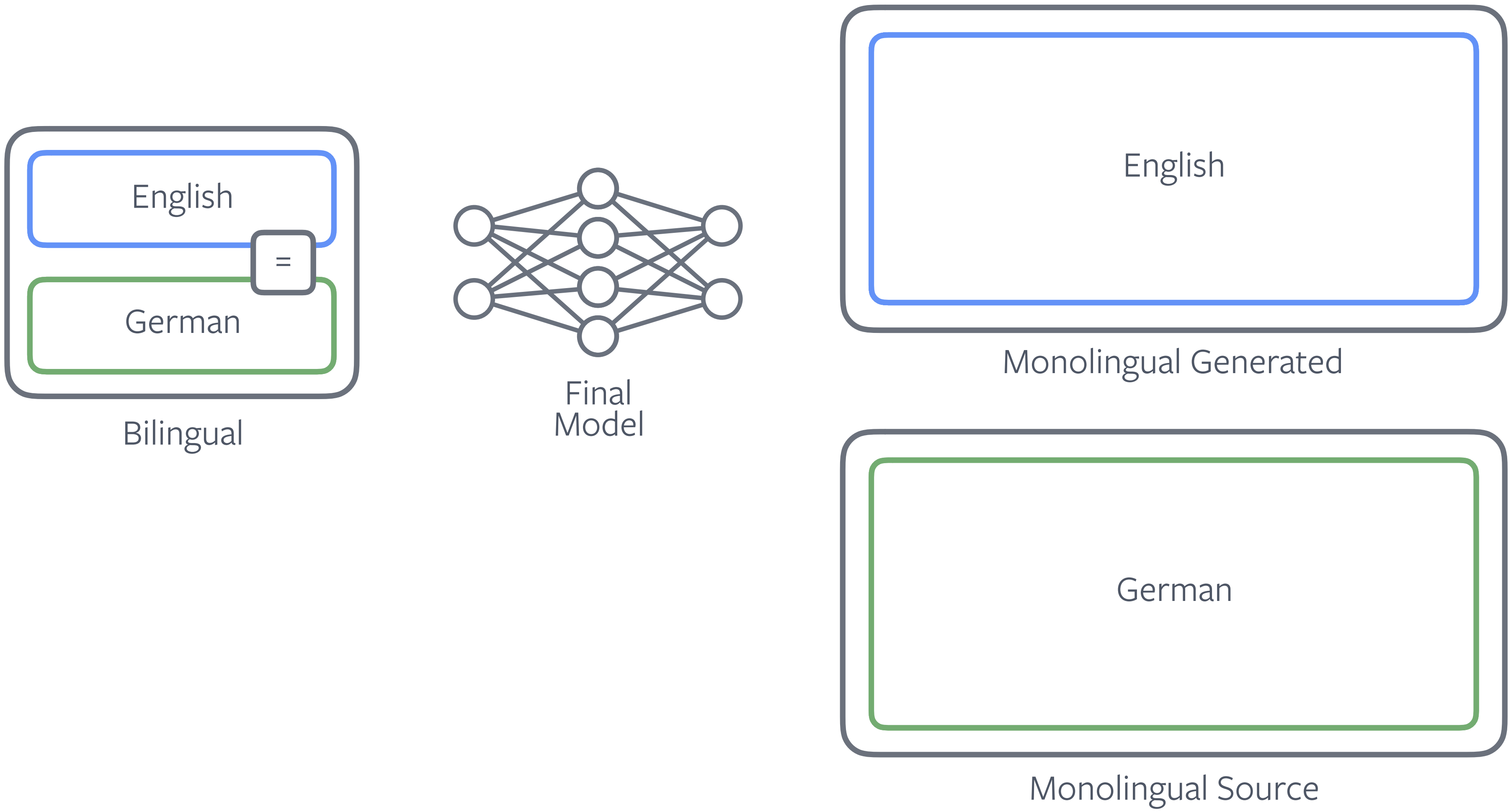
Back-translation (Bojar & Tamchyna, 2011; Sennrich et al., 2016)

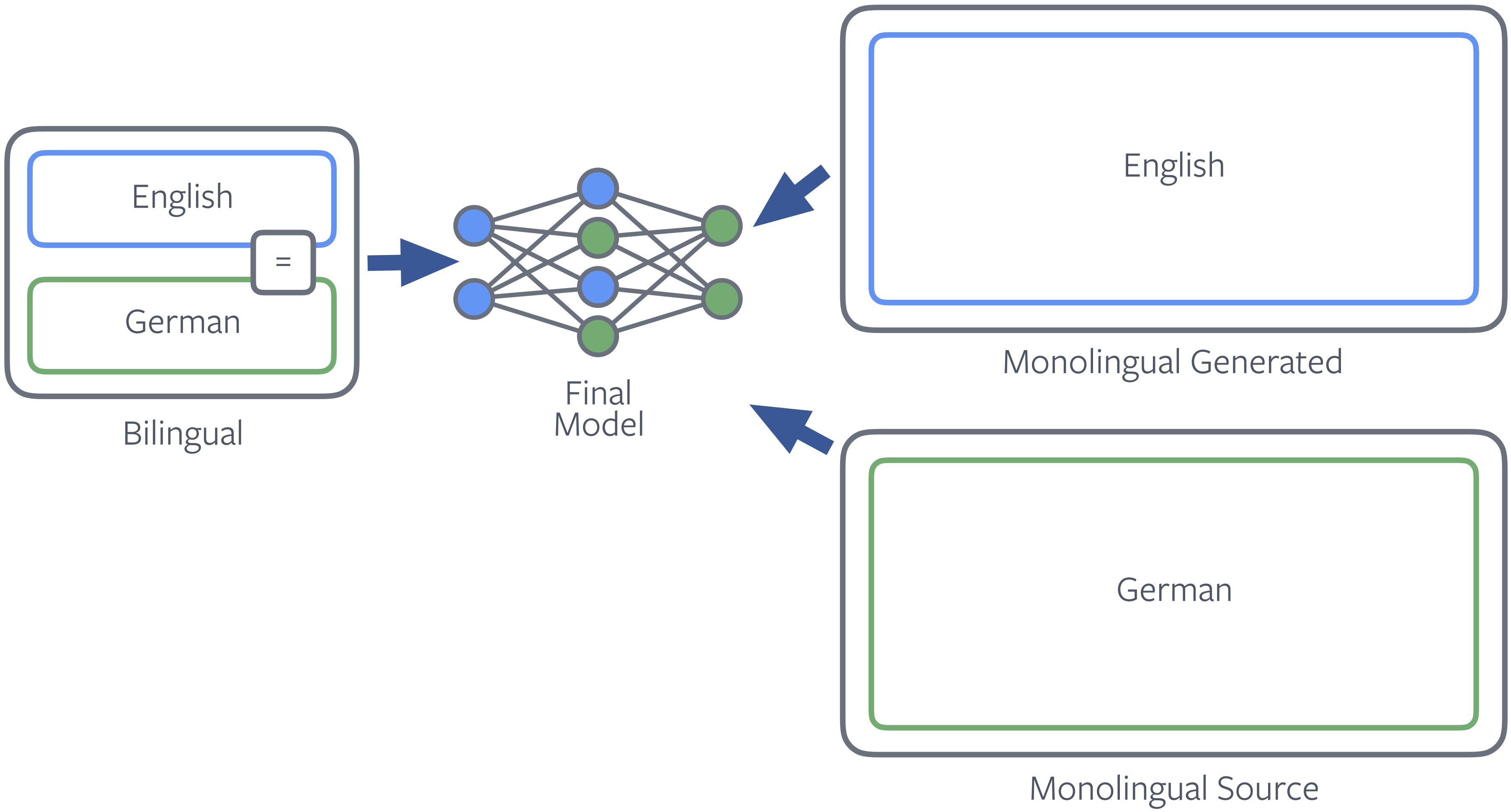



German
=
English
Bilingual
Intermediate
Model
German
Monolingual Source


German
=
English
Bilingual
Intermediate
Model
German
Monolingual Source
Monolingual Generated

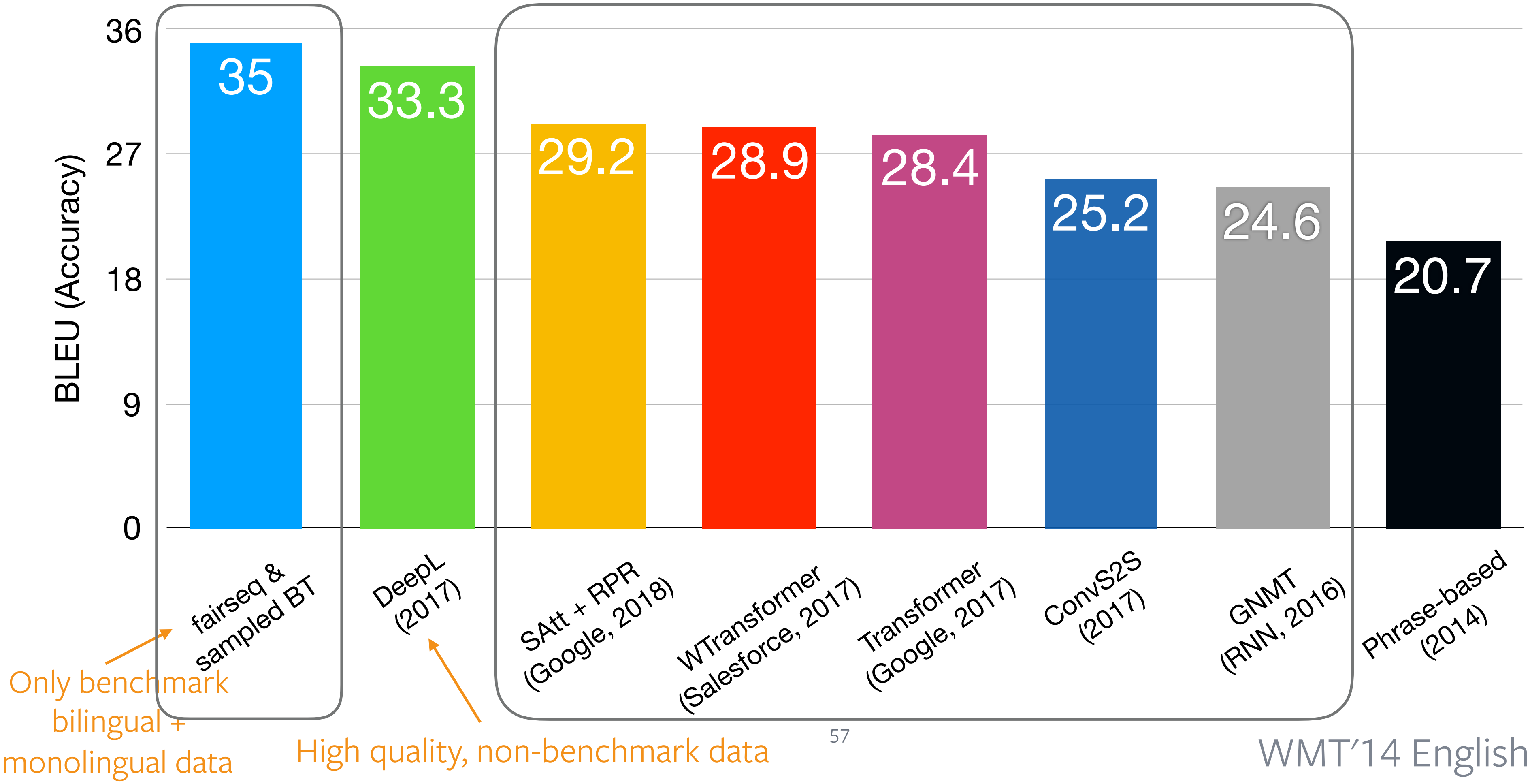

German
=
English
Bilingual
Intermediate
Model
German
Monolingual Source
English
Monolingual Generated


English
=
German
Bilingual
Final
Model
English
Monolingual Generated
German
Monolingual Source


English
=
German
Bilingual
Final
Model
English
Monolingual Generated
German
Monolingual Source

Scaling from 100M to 5.8B words

Model trains in 22.5h
on 128 V100



WMT'14 English-German

WMT'18 Human evaluations

Ranked #1 in the human evaluation of the WMT'18 English-German translation task

English→German

	Ave. %	Ave. z	System
1	85.5	0.653	FACEBOOK-FAIR ⋆
2	82.2	0.561	ONLINE-B
	81.9	0.551	MICROSOFT-MARIAN
	81.6	0.539	MMT-PRODUCTION
	82.3	0.537	UCAM
	80.2	0.491	NTT
	79.3	0.454	KIT
8	77.7	0.396	ONLINE-Y
	76.7	0.377	JHU
	76.3	0.352	UEDIN
11	71.8	0.213	LMU-NMT
12	67.4	0.060	ONLINE-A
13	53.2	−0.385	ONLINE-F
	53.8	−0.416	ONLINE-G
15	36.7	−0.966	RWTH-UNSUPER
16	32.6	−1.122	LMU-UNSUP

Conclusion

Mixed precision training in PyTorch:

- 3-4x speedups in training wall time
- No architecture changes required
- Use Nvidia's `apex` library

Case study: Neural Machine Translation

- Train models in 30 minutes instead of 1 day+
- State-of-the-art translation quality using semi-supervised learning

Thank you! Questions?

Contact Us

Myle Ott
myleott@fb.com

Sergey Edunov
edunov@fb.com

References

- Scaling Neural Machine Translation: `arxiv.org/abs/1806.00187`
- Understanding Back-translation at Scale: `arxiv.org/abs/1808.09381`
- apex: `nvidia.github.io/apex`

Acknowledgements: Nvidia and PyTorch teams for helping us implement and optimize mixed precision training.