NATIONAL RESEARCH UNIVERSITY

НИС Машинное обучение и приложения

РАСПРЕДЕЛЁННОЕ ОБУЧЕНИЕ НЕЙРОСЕТЕЙ

Другие оптимизации

Москва, 2021

Шабат Дмитрий, БПМИ192

Как сэкономить память, когда мы работаем с большими массивами вещественных чисел?

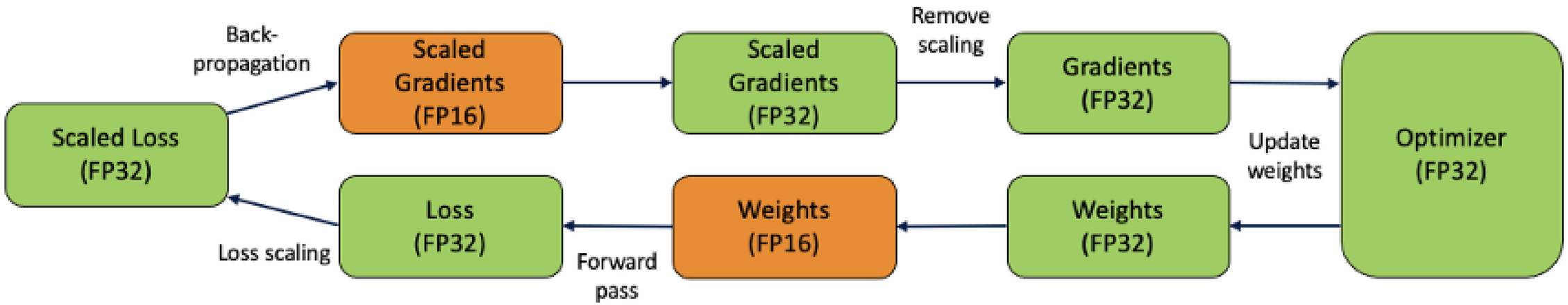
Как сэкономить память, когда мы работаем с большими массивами вещественных чисел?

float (32 bit) → half (16 bit)

В большинстве вычислений будем использовать FP16
В местах, требующих бОльшей точности, будем использовать FP32

Mixed precision training

- Процесс только частично на 16-битной точности
- В итоге сама итоговая модель всё равно будет в 32-битной точности

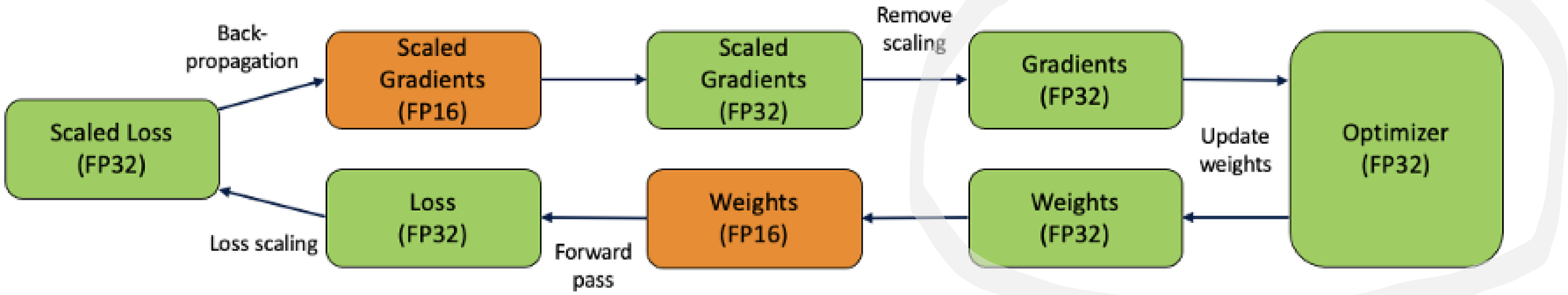


[from https://towardsdatascience.com/understanding-mixed-precision-training-4b246679c7c4]

Mixed precision training

Модель в полном разрешении хранится в Оптимизаторе

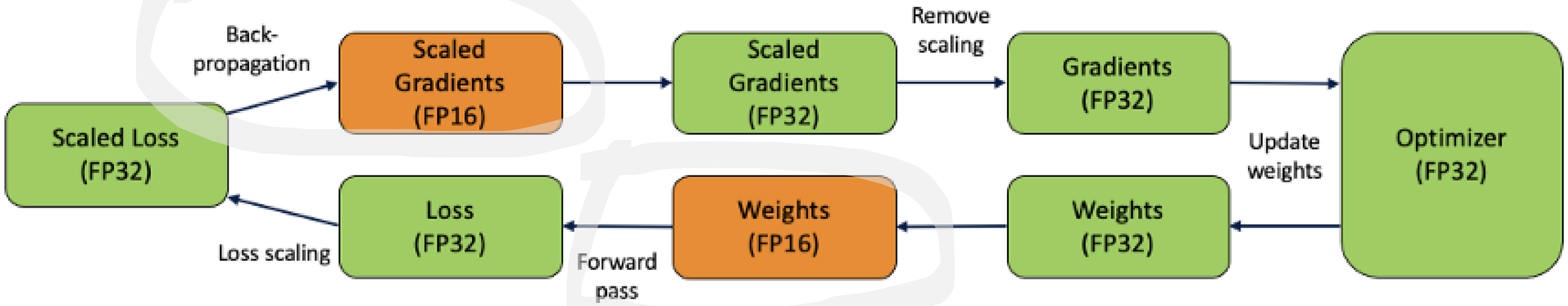
Оттуда она подаётся в процесс обучения



[from https://towardsdatascience.com/understanding-mixed-precision-training-4b246679c7c4]

Mixed precision training

Forward-pass и Back-prop проводятся в FP16

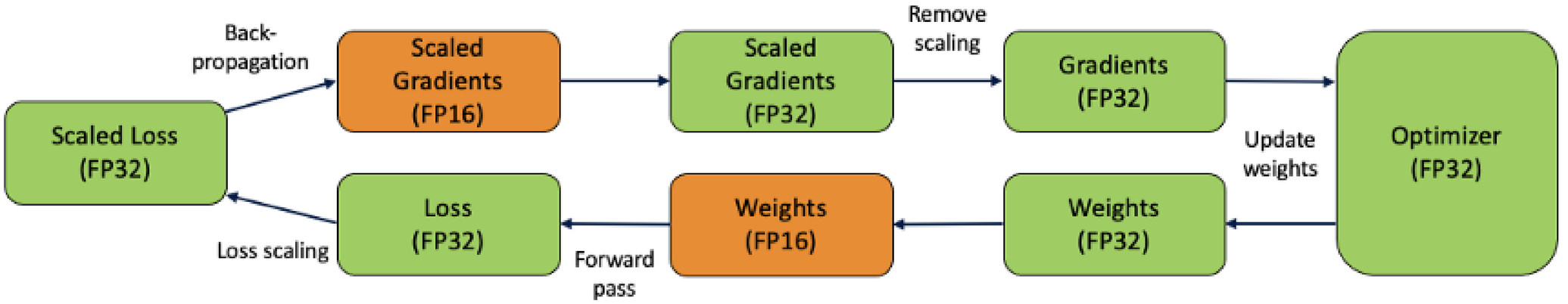


[from https://towardsdatascience.com/understanding-mixed-precision-training-4b246679c7c4]

Mixed precision training

- Можно обучать модели бОльшего размера
- Ускорение вычислений в forward/backward pass'е позволяет в 3 раза сократить время обучения модели
- Быстрее коммуникация между нодами благодаря сжатию градиентов

Важно — поскольку обычно абсолютные значения градиентов занимают малый диапазон, перед переводом в FP16 домножают на какой-то *scale*, чтобы минимально потерять в точности



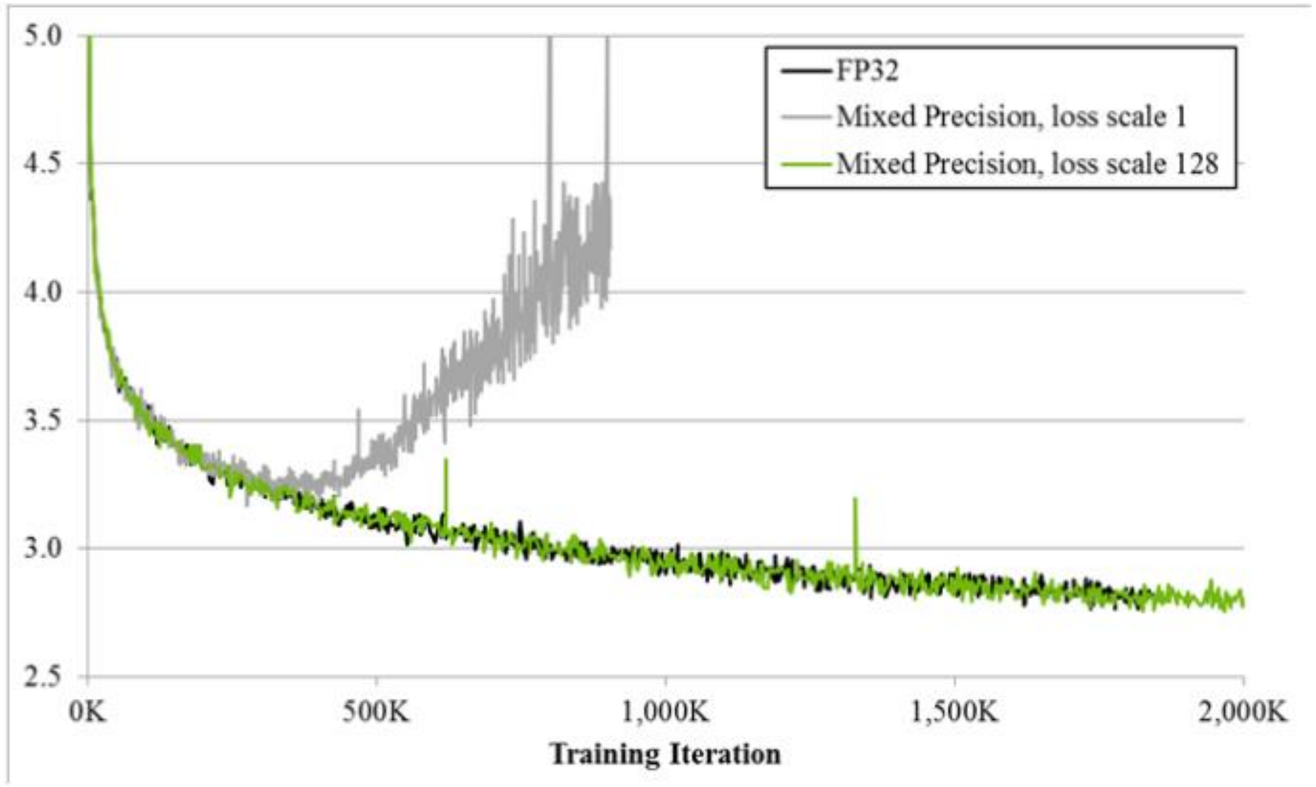
[from https://towardsdatascience.com/understanding-mixed-precision-training-4b246679c7c4]

Mixed precision training

[GPU Tensor Cores](#) добавляют встроенную поддержку для ускоренных вычислений в FP32, FP16, INT8, INT4

В 2018 Nvidia разработала расширение для PyTorch под названием [Apex](#) для **Automatic Mixed Precision**

Пример на обучении **bigLSTM English**

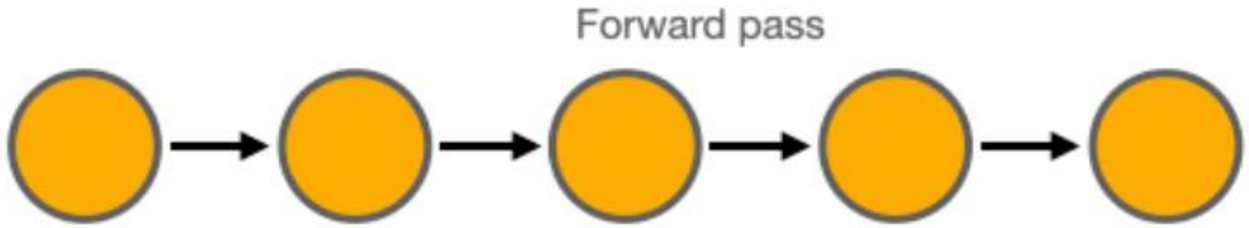


[from https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html]

Gradient checkpointing:
Обменяем время на память

Gradient checkpointing

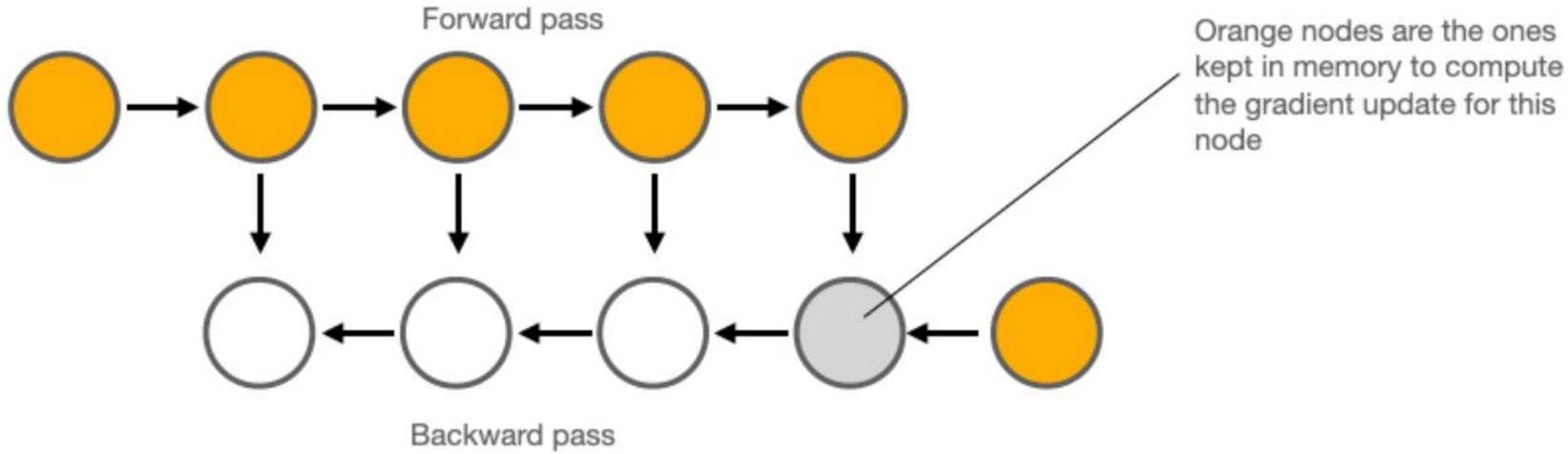
Возьмём за пример последовательную нейросеть
Каждый следующий слой высчитывается напрямую из предыдущего



Gradient checkpointing

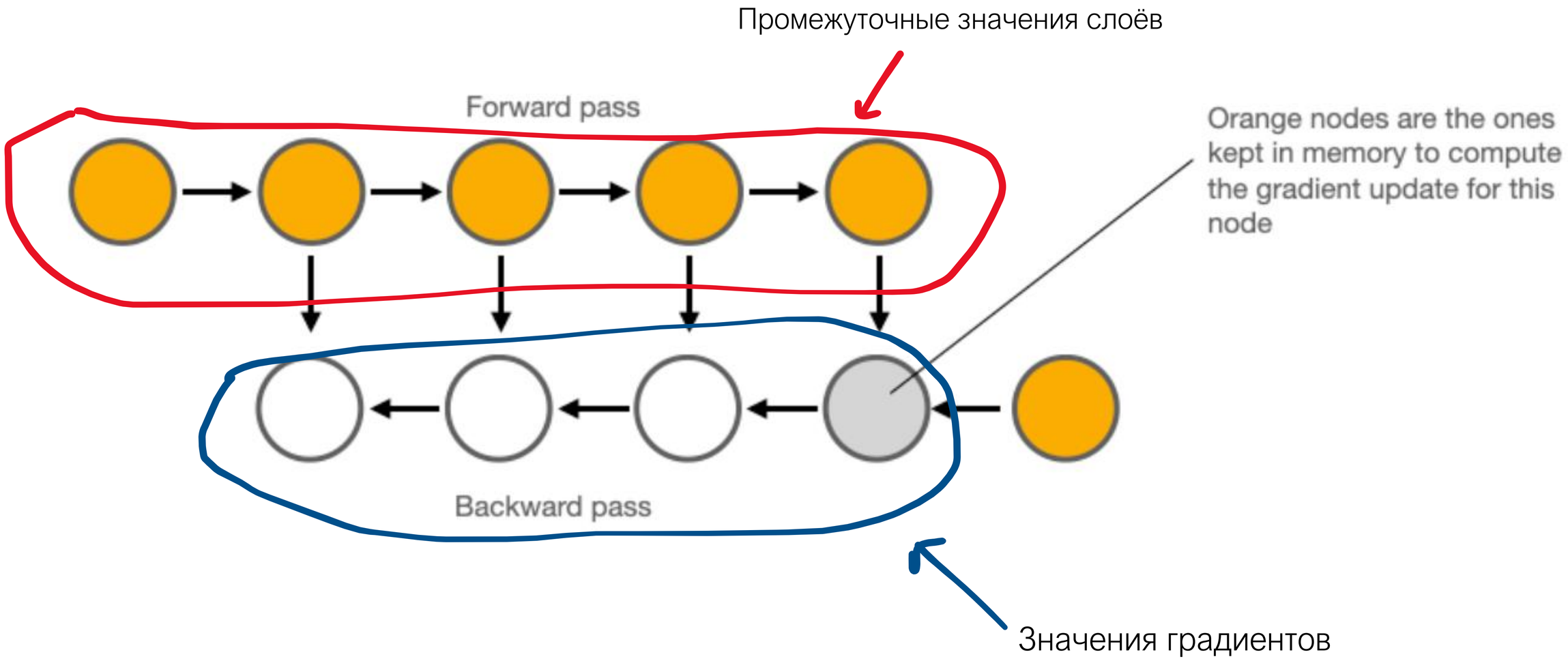
Много памяти во время обучения тратится на хранение промежуточных значений в сети

Поэтому надо использовать *torch.no_grad()* !



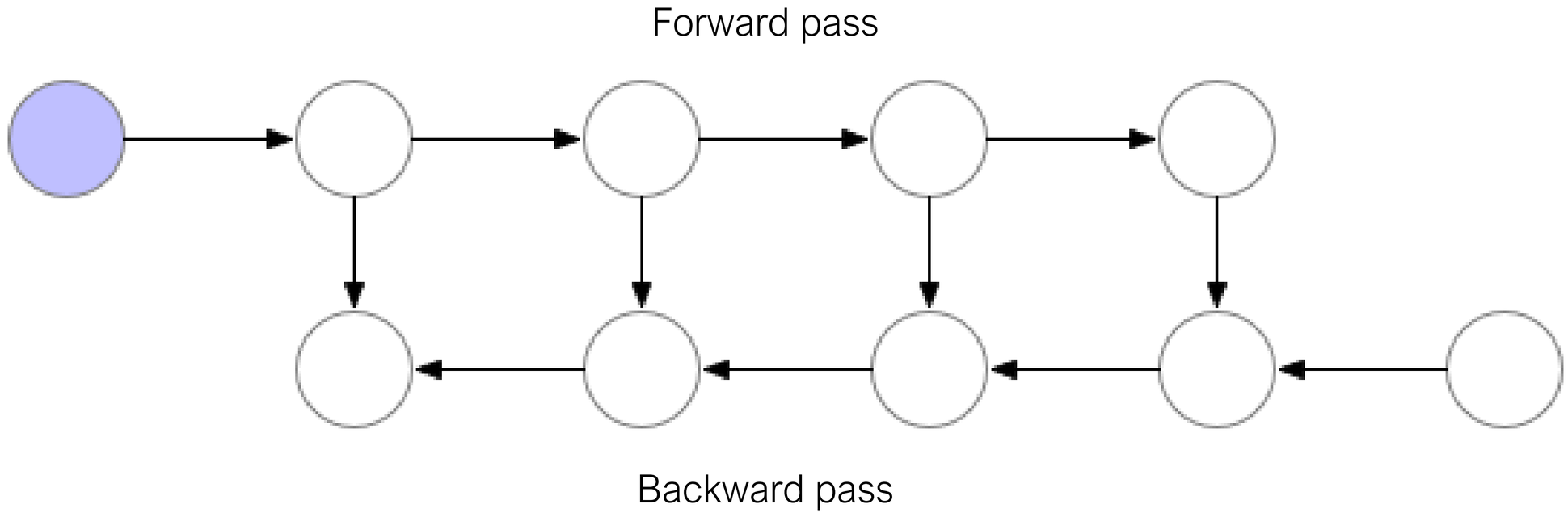
[from https://twitter.com/rasbt/status/1341430378834382859]

Gradient checkpointing



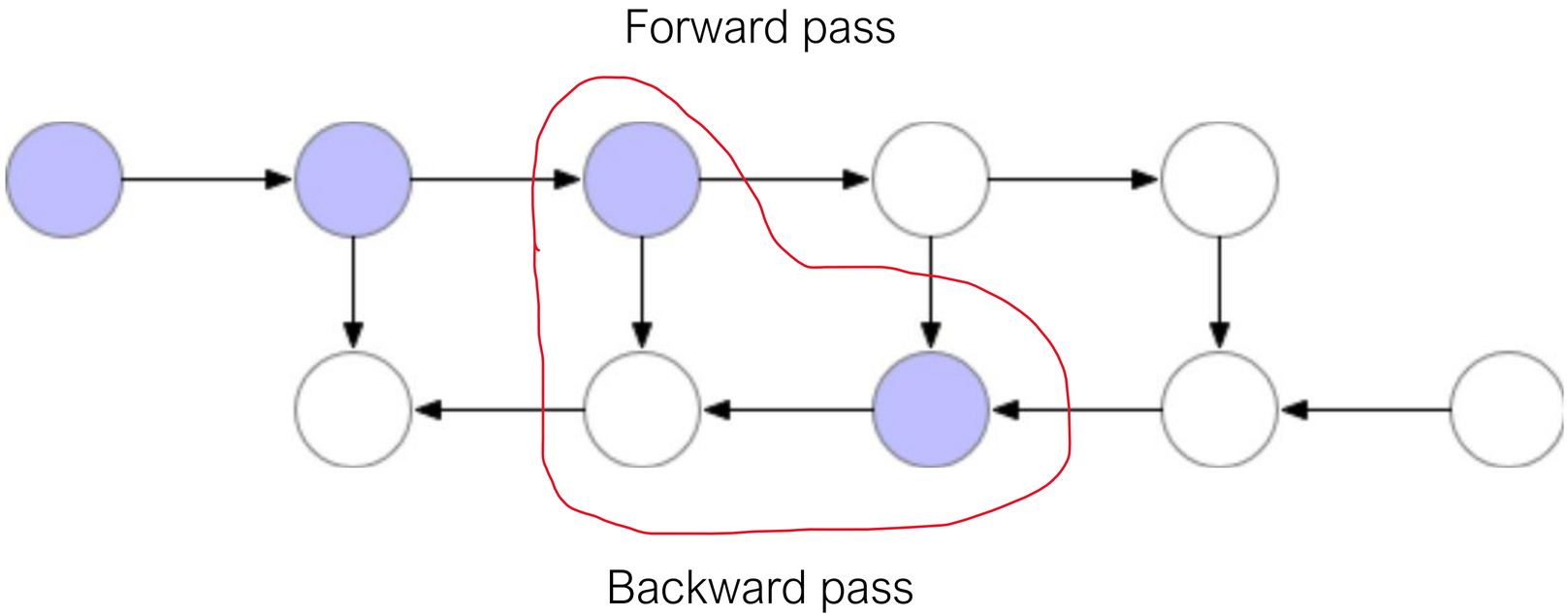
Gradient checkpointing

Примерно так выглядит один Forward+Backward pass



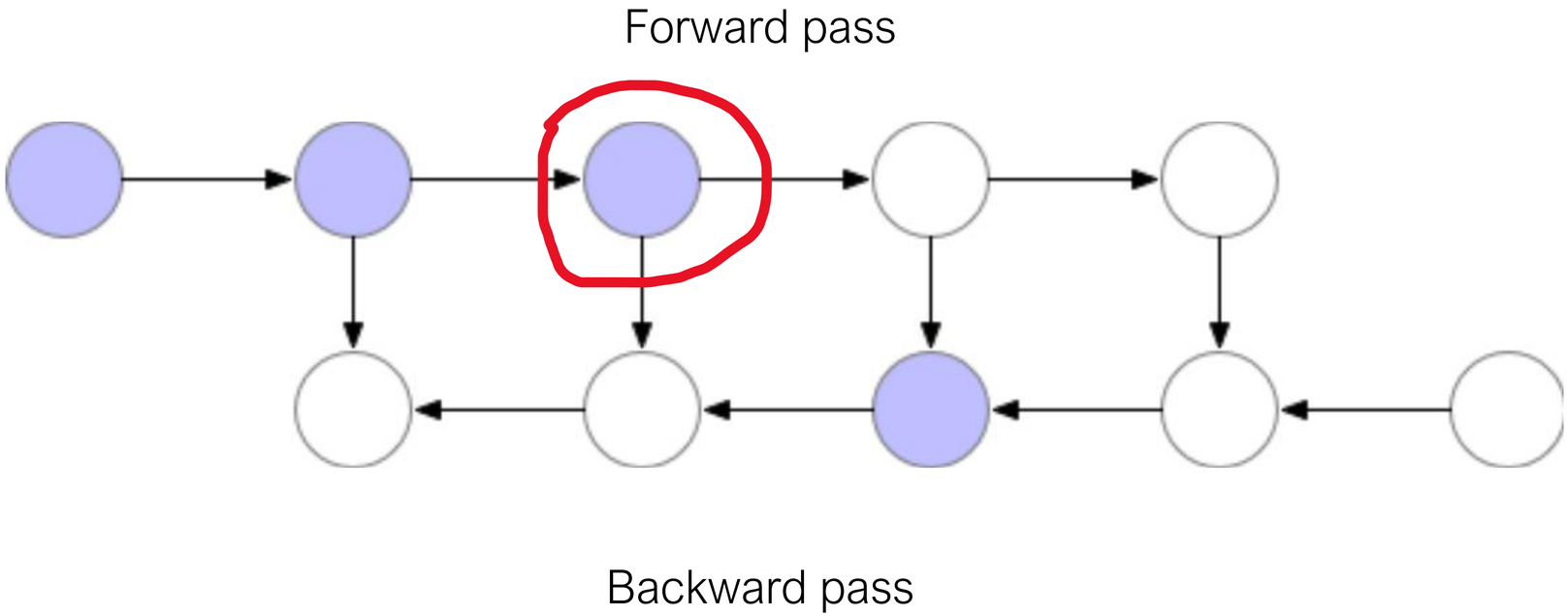
Gradient checkpointing

Посмотрим на один конкретный шаг Backprop’а



Gradient checkpointing

Значение только этого слоя нам нужно на этом шаге



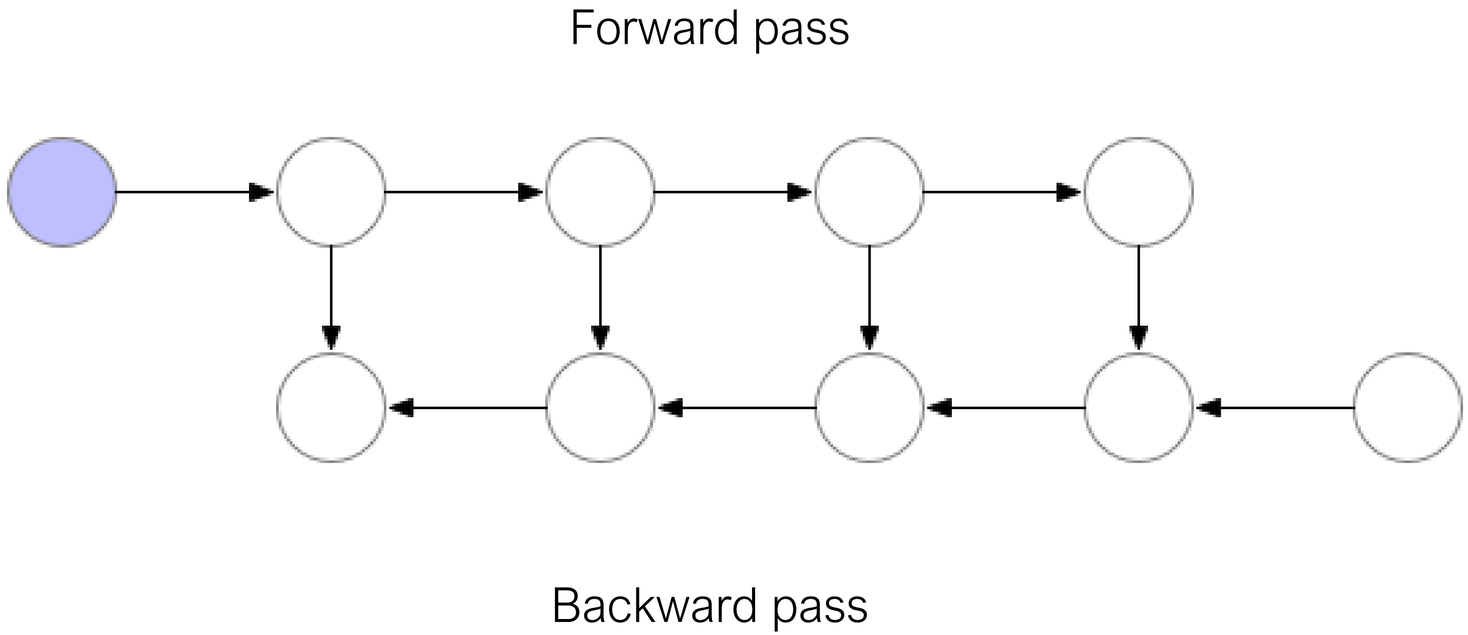
Gradient checkpointing

БИНГО!

Будем не сохранять эти значения, а каждый раз считать их заново

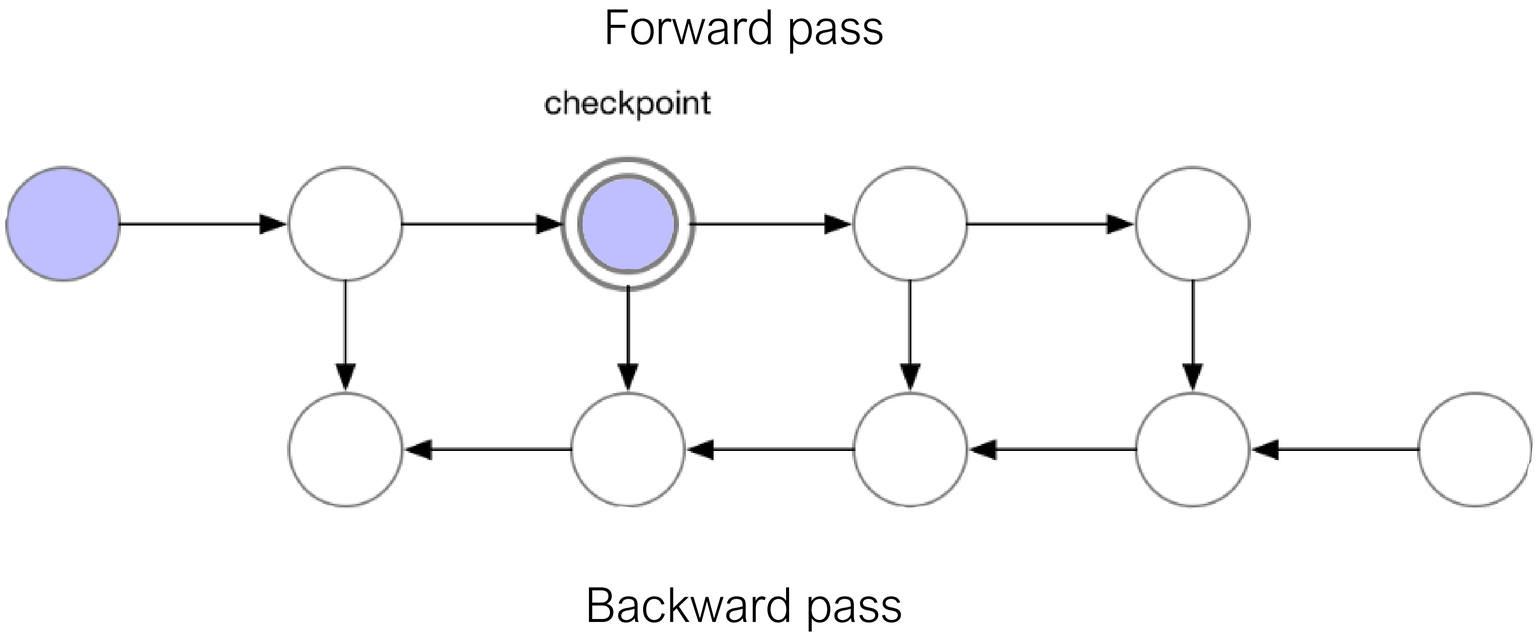
Gradient checkpointing

Для каждого шага Backprop’а просто заново посчитаем это значение



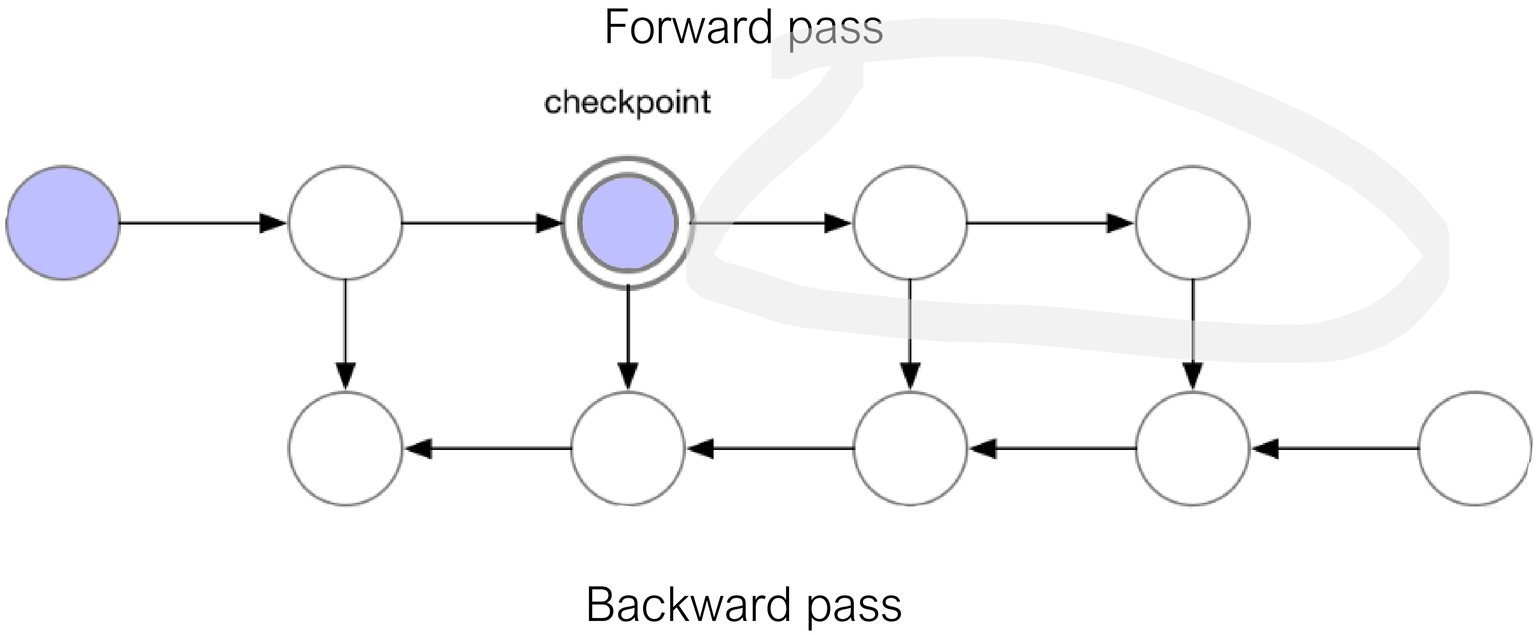
Gradient checkpointing

Также вместо полного пересчёта можем сохранить значения для какого-то слоя посередине, использовать его для пересчёта значений после



Gradient checkpointing

Также вместо полного пересчёта можем сохранить значения для какого-то слоя посередине, использовать его для пересчёта значений после

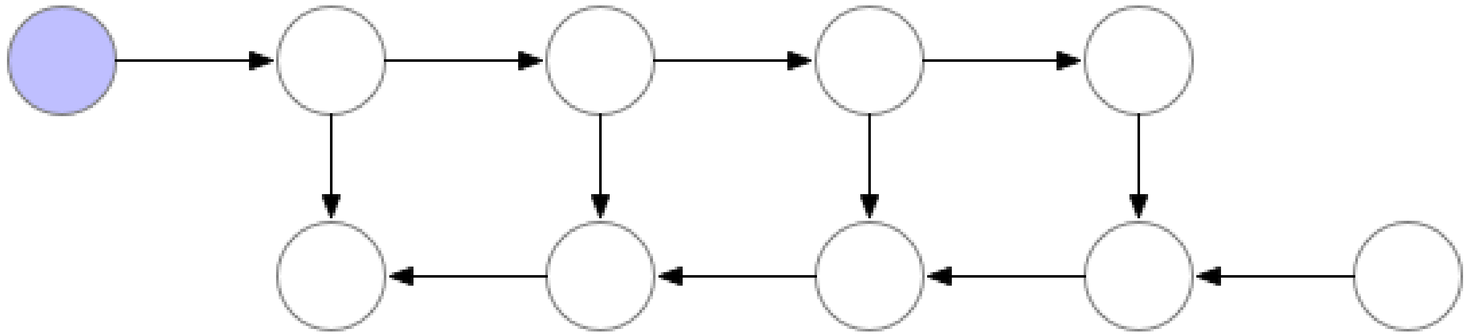


Gradient checkpointing

Для сети с N слоями, используя K чекпоинтов:

- Сокращаем затраты по памяти примерно в $\frac{K}{N}$ раз
- Увеличиваем время работы примерно в $\frac{N}{K}$ раз

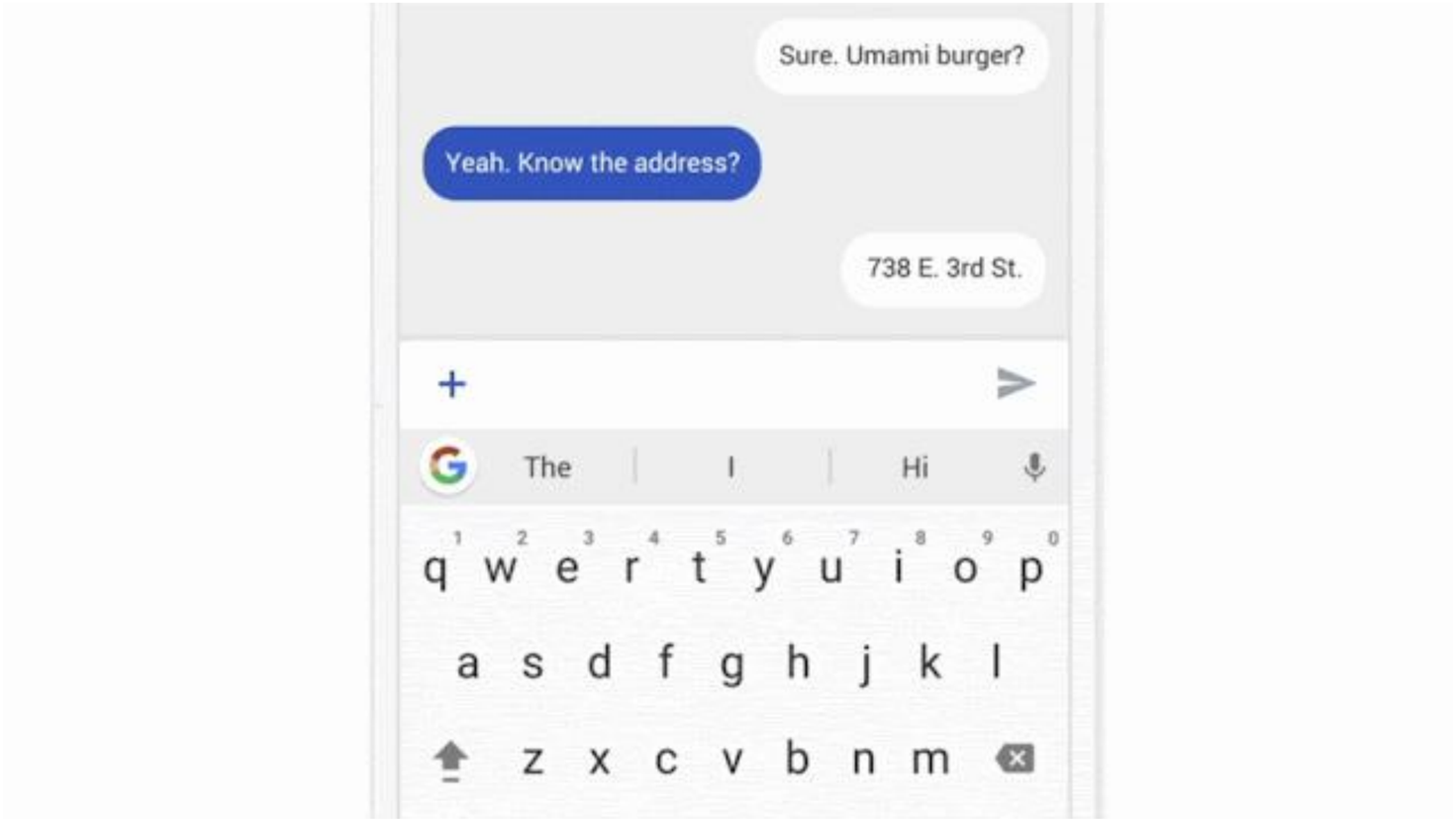
Если вам время и память примерно одинаково ценны, можете показать что оптимально будет использовать K=$\sqrt{N}$



Federated Learning:
Обучаемся на мобильных устройствах бедных пользователей

Federated learning

Причём работа этой модели может быть персонализирована под конкретного пользователя. (Например, умные подсказки гугла в смартфоне)

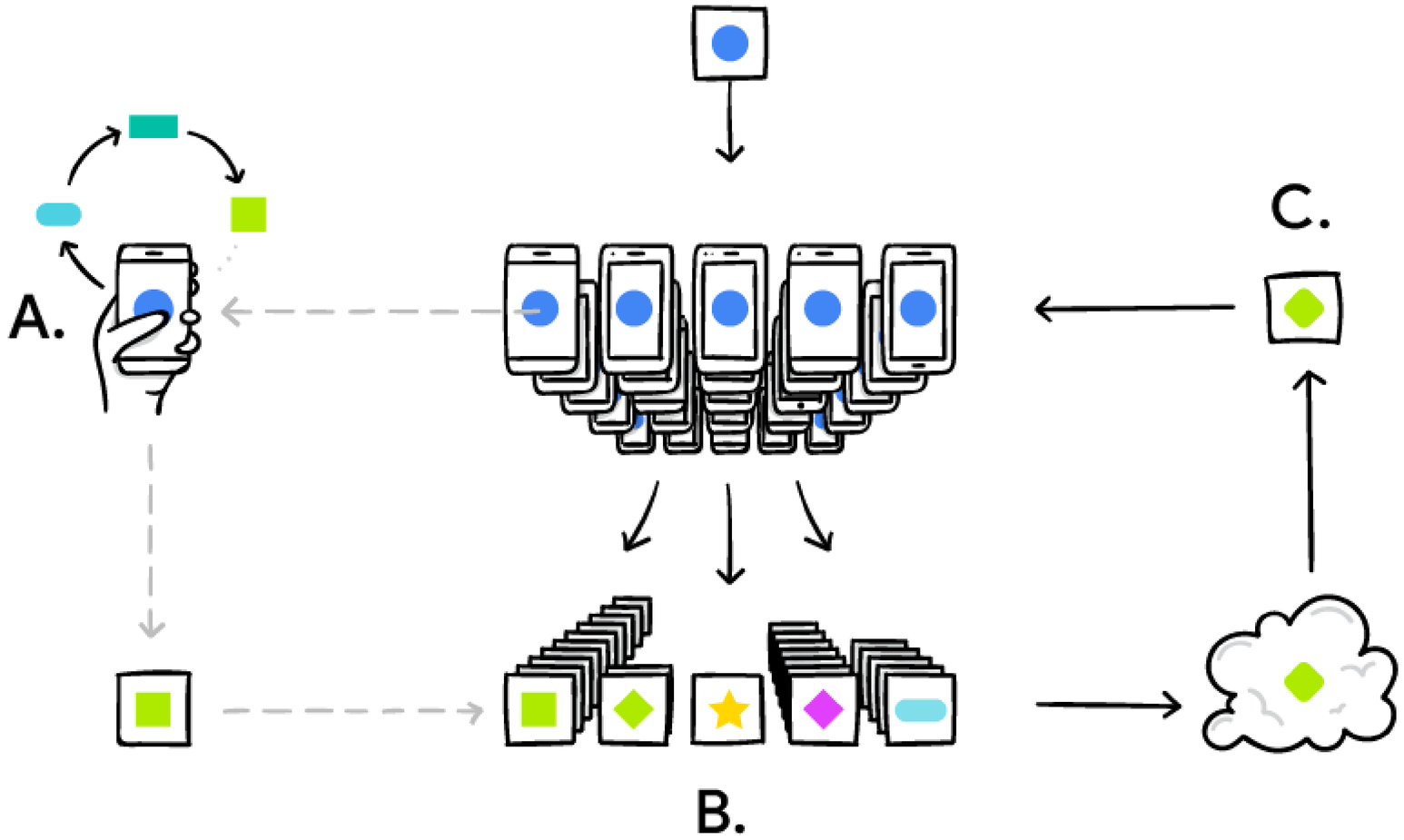


Federated learning

[from https://ai.googleblog.com/2017/04/federated-learning-collaborative.html]

Каждый телефон имеет локальную копию модели

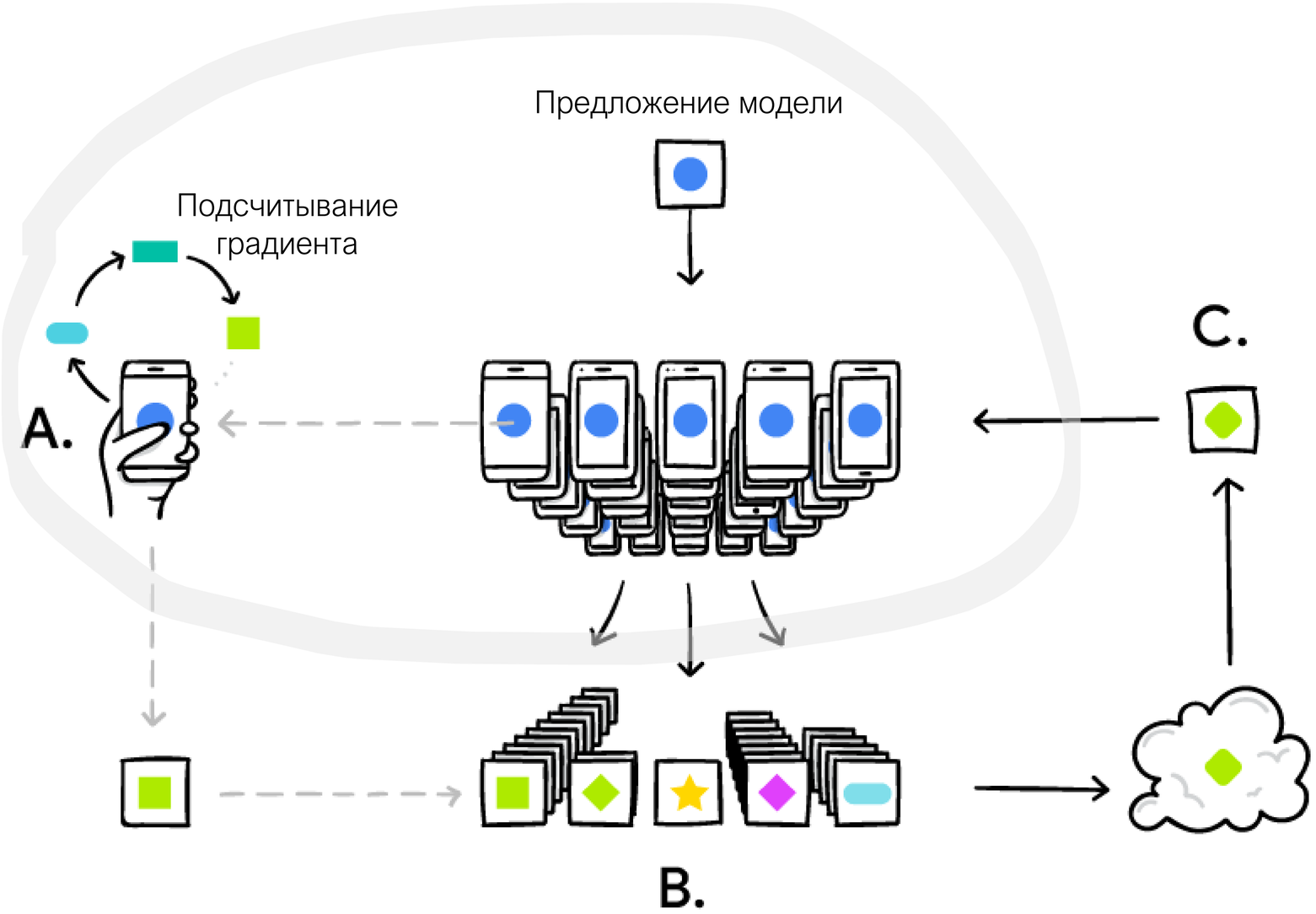
1. Делает персонализированные обновления на основе действия пользователей
2. Персонализированные обновления отправляются в облако где они смешиваются
3. Это глобальное обновление отправляется всем пользователям



Federated learning

Каждый телефон имеет локальную копию модели

1. **Делает персонализированные обновления на основе действия пользователей**
2. Персонализированные обновления отправляются в облако где они смешиваются
3. Это глобальное обновление отправляется всем пользователям

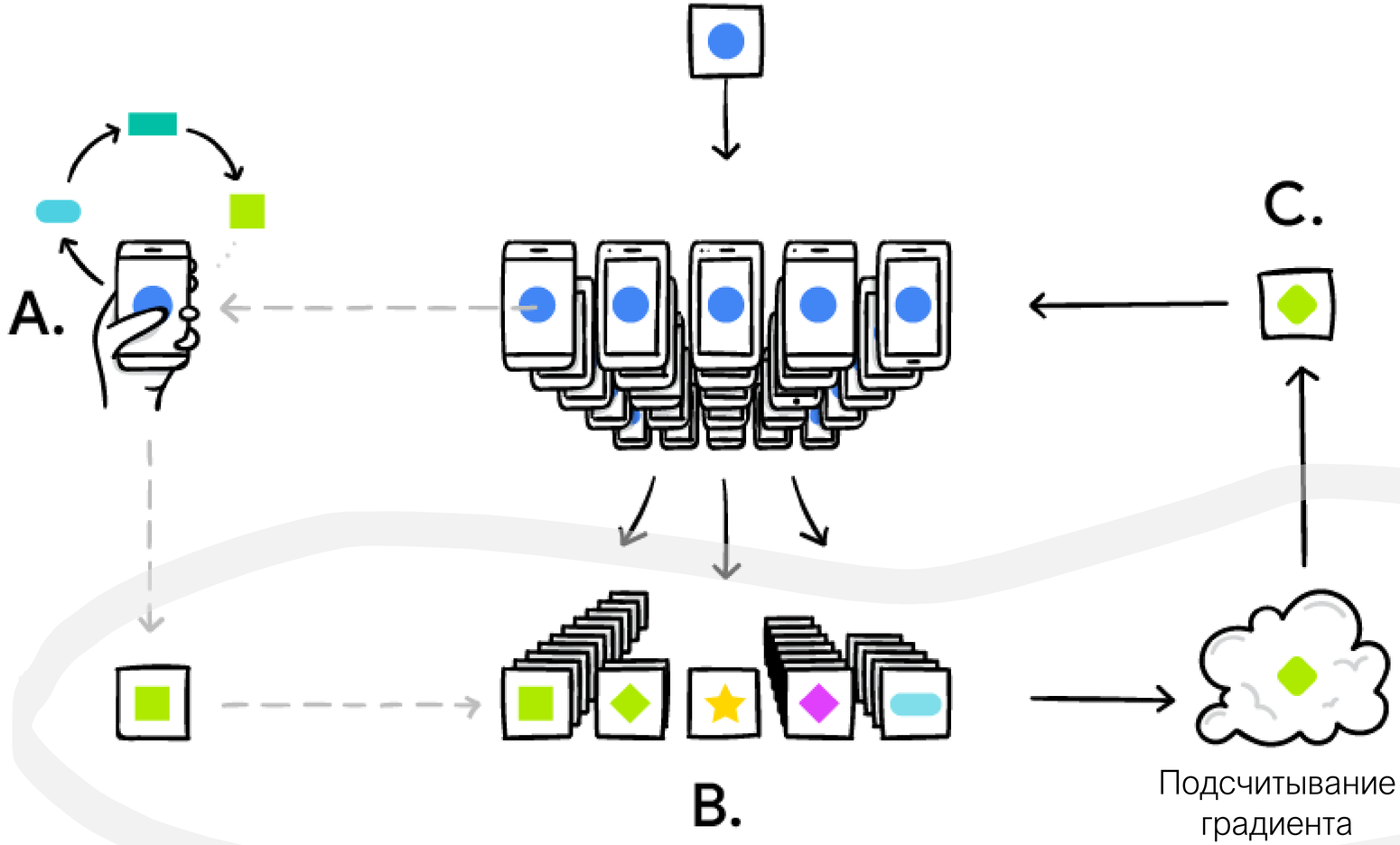


Federated learning

[from https://ai.googleblog.com/2017/04/federated-learning-collaborative.html]

Каждый телефон имеет локальную копию модели

1. Делает персонализированные обновления на основе действия пользователей
2. **Персонализированные обновления отправляются в облако где они смешиваются**
3. Это глобальное обновление отправляется всем пользователям

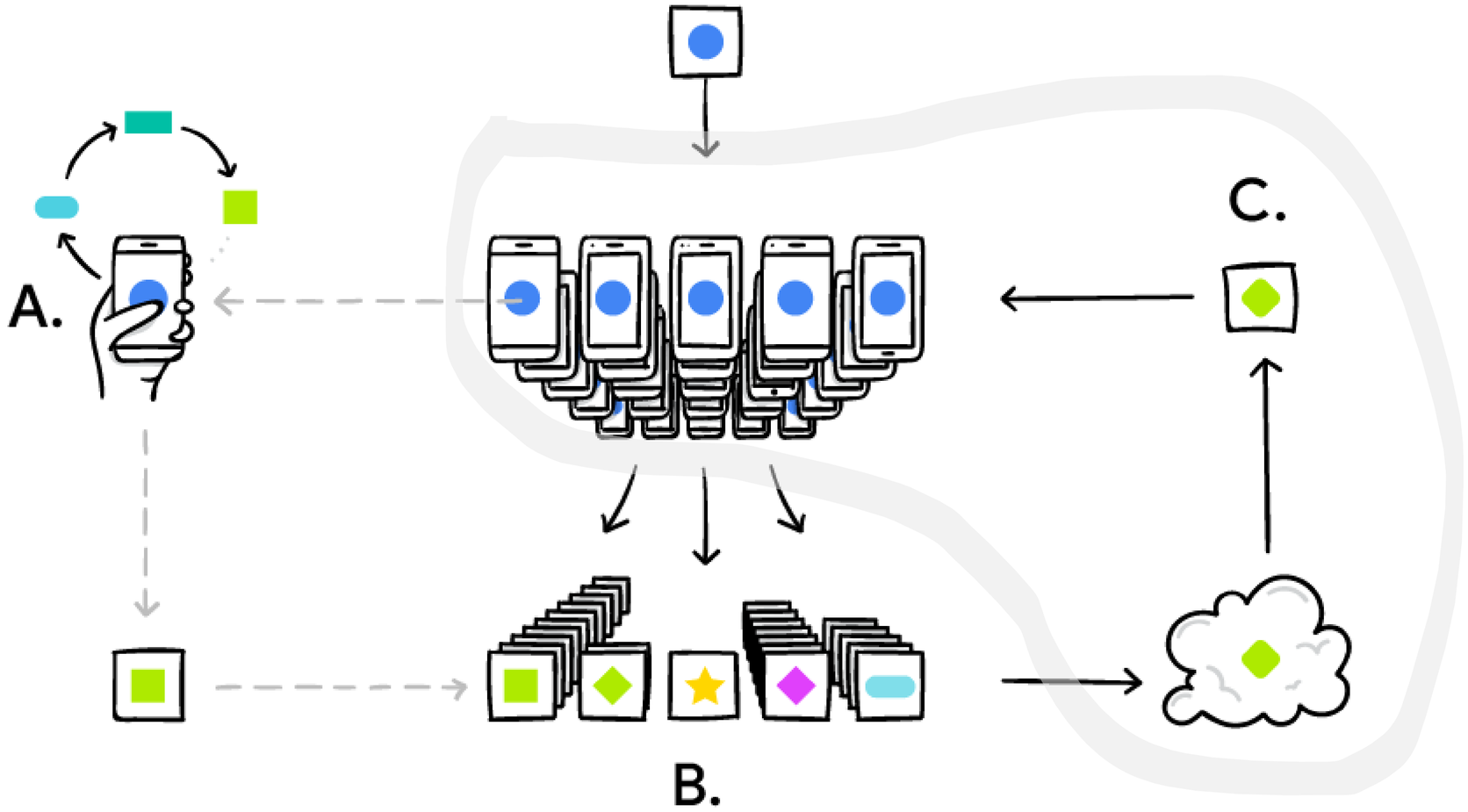


Federated learning

[from https://ai.googleblog.com/2017/04/federated-learning-collaborative.html]

Каждый телефон имеет локальную копию модели

1. Делает персонализированные обновления на основе действия пользователей
2. Персонализированные обновления отправляются в облако где они смешиваются
3. **Это глобальное обновление отправляется всем пользователям**

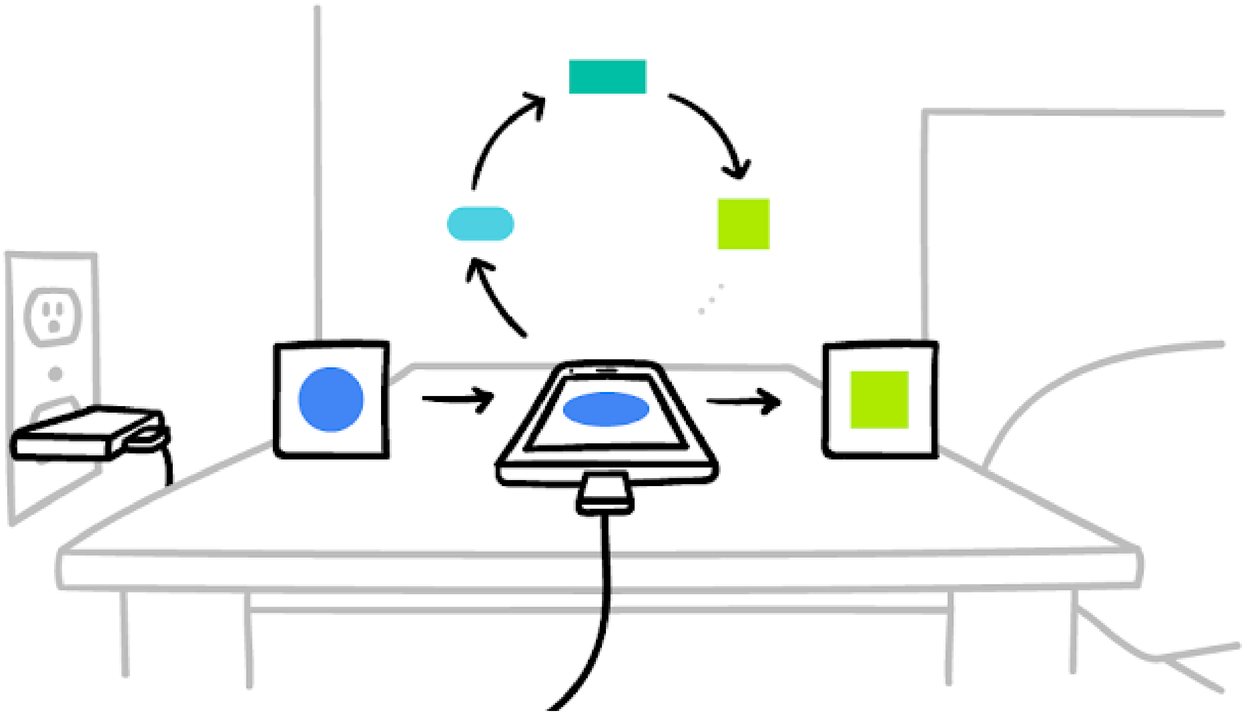


Federated learning

[from https://ai.googleblog.com/2017/04/federated-learning-collaborative.html]

Google в своей статье про это упоминает ещё такое:

- Сжатие используется для передачи обновлений по сети
- Шифрование обновлений используется для того, чтобы третье лицо не смогло восстановить по обновлениям действия пользователя
- Высчитывание обновления происходит в неактивное время телефона, чтобы не вредить пользовательскому опыту



NATIONAL RESEARCH
UNIVERSITY

Спасибо за внимание!