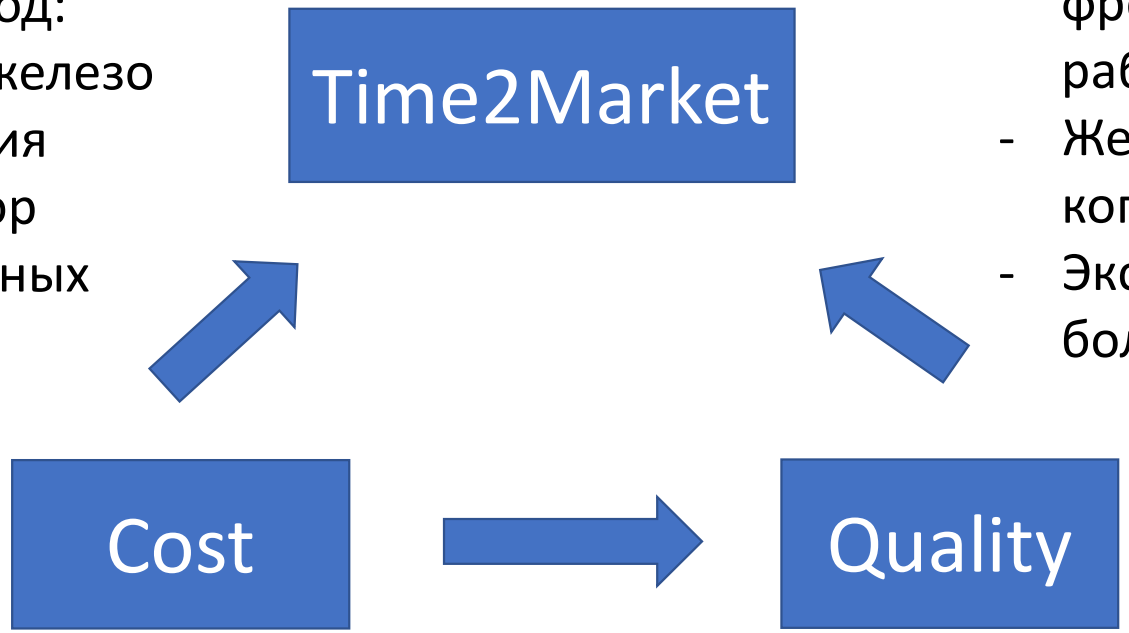
Перенос моделей в распределенную среду

Проблема

Распределенный подход:
- Стоковое дешевое железо
- Простая эксплуатация
- Ограниченный набор моделей в стандартных фреймворках



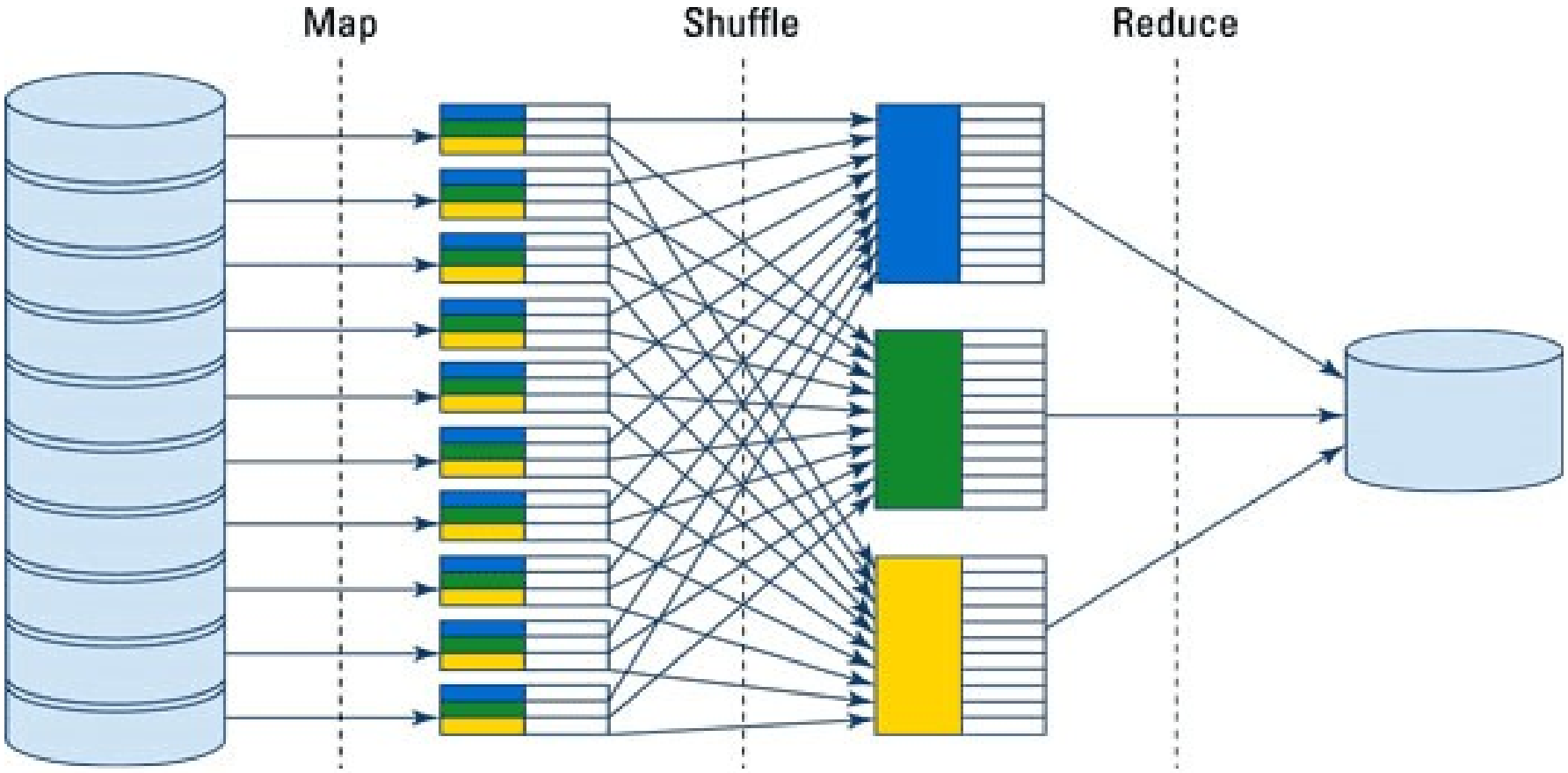
Массивно-параллельный подход:
- Популярные ML-фреймворки работают из коробки
- Железо влетает в копеечку
- Эксплуатация еще больше

Комбинированный подход:
- Сначала делаем R&D по распределению алгоритма
- Тренируем модели быстро, дешево и качественно

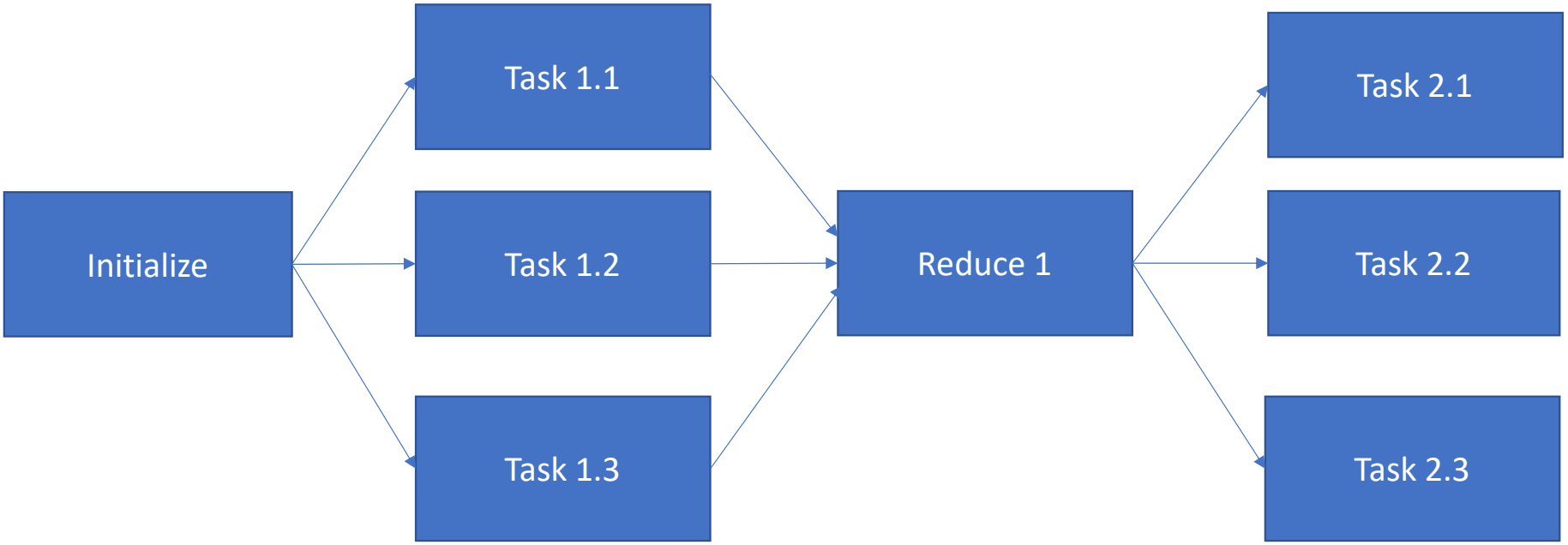
Источники параллелизма в ML

- Параллелизм данных
  - Считаем части суммы градиента параллельно
- Параллелизм модели
  - Обновляем разные части модели параллельно
- Параллелизм задачи
  - Вычисляем модель для каждого «фолда» параллельно

ML на распределенных данных



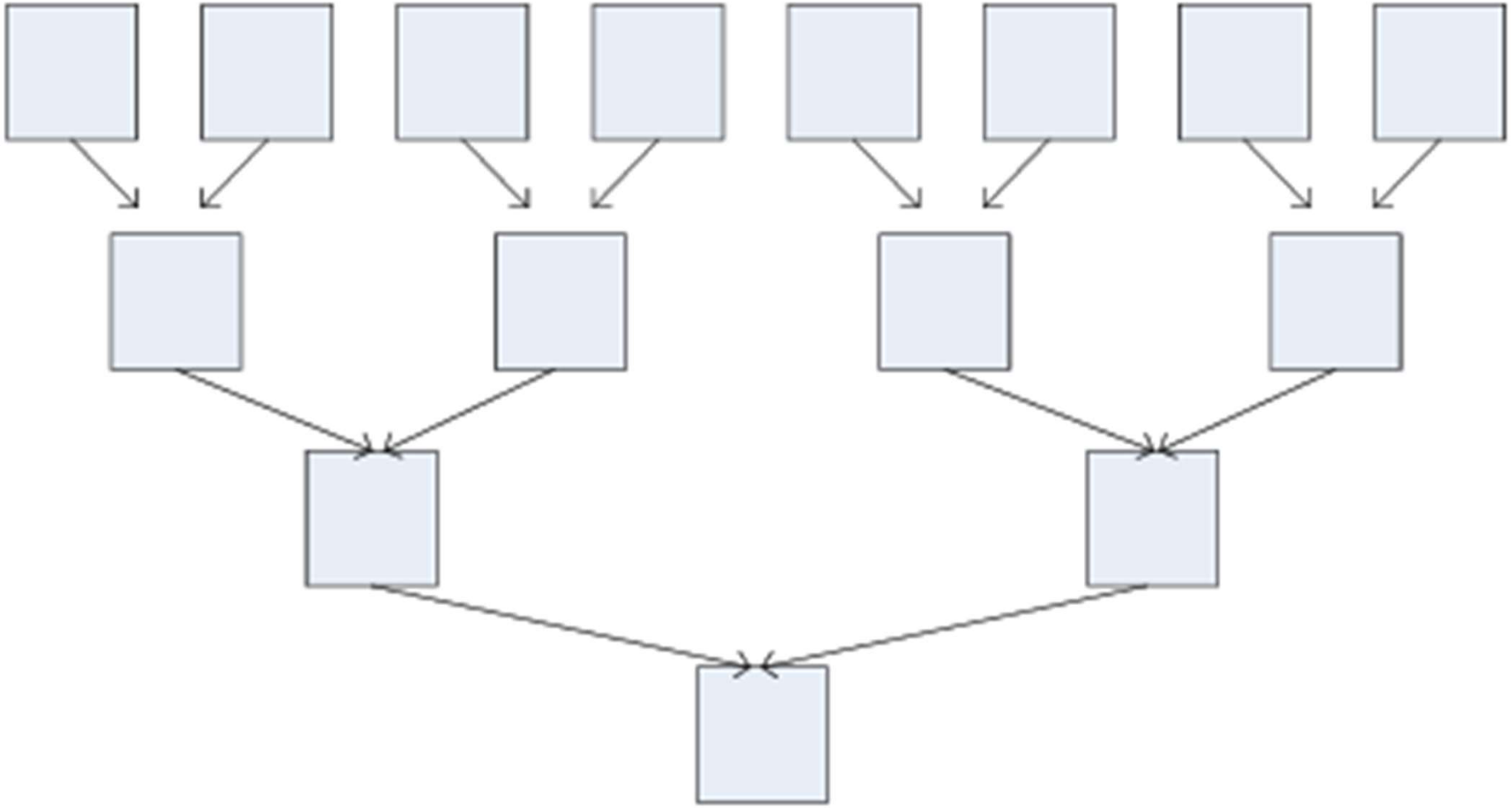
Градиентный спуск на MapReduce



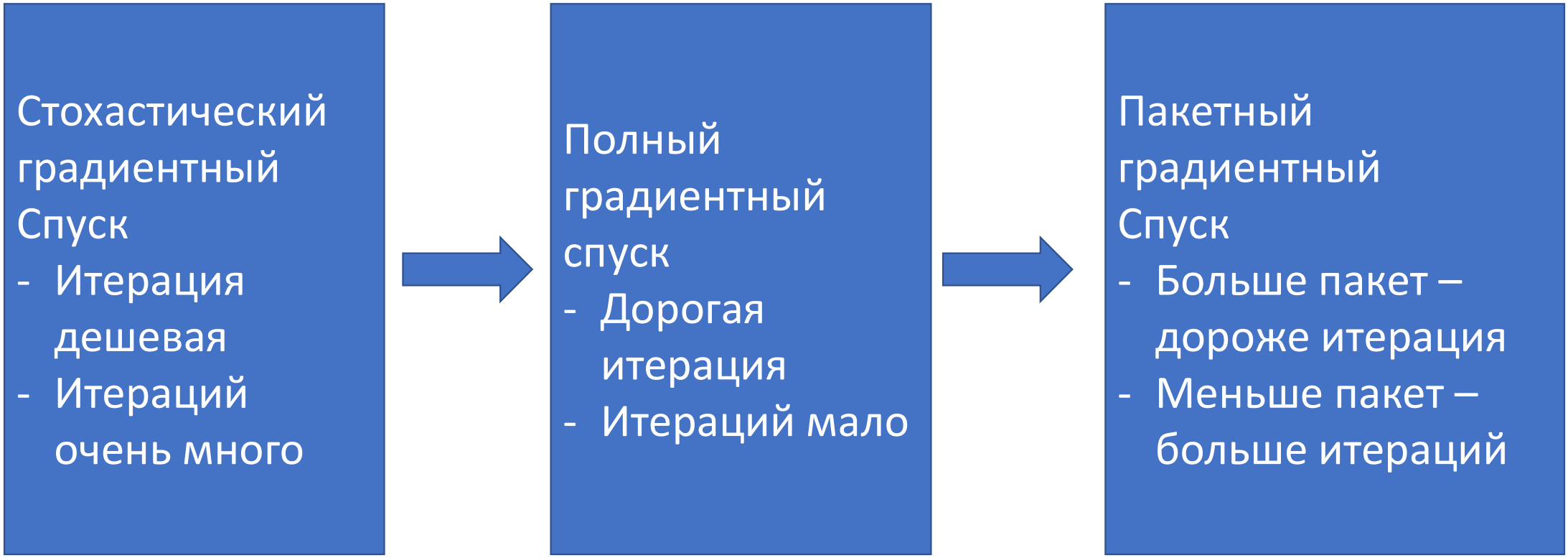
Градиентный спуск на MapReduce: проблемы

- Драйвер – узкое место
  - Даем много памяти
  - Стараемся использовать treeReduce
- Финиш по самому медленному
  - Настроить locality.wait
  - Настроить speculation
- Время итерации >0.2s

Tree-reduce pattern



Пространство для оптимизации

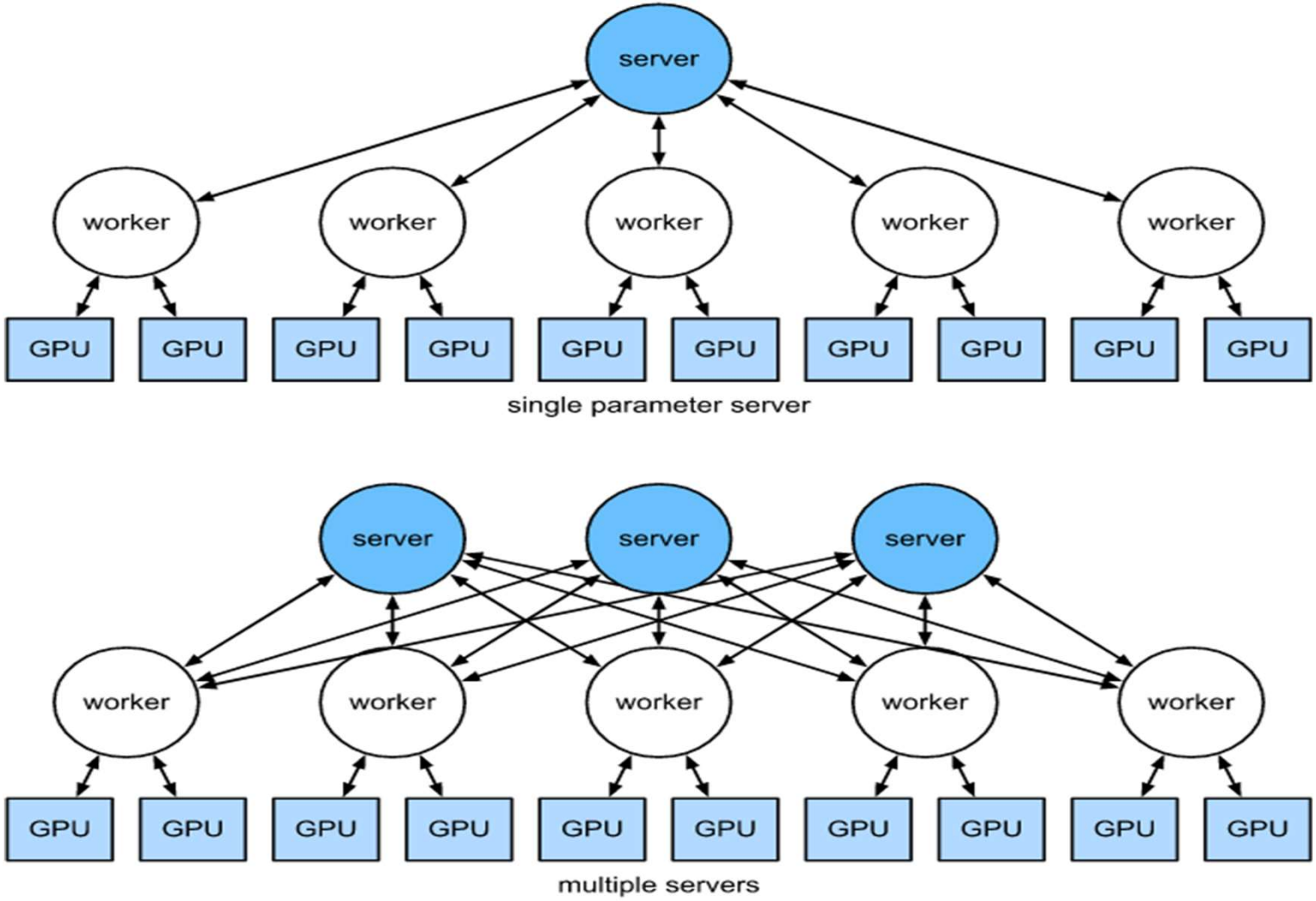


Размер модели

Устоявшаяся практика (не совсем правильная)

1. Берем большие данные
2. Применяем распределенный ETL
3. Получаем маленькие данные
4. Применяем централизованный ML
5. Получаем качественную модель

Сервер параметров увеличивает модель



Режим выставления барьера

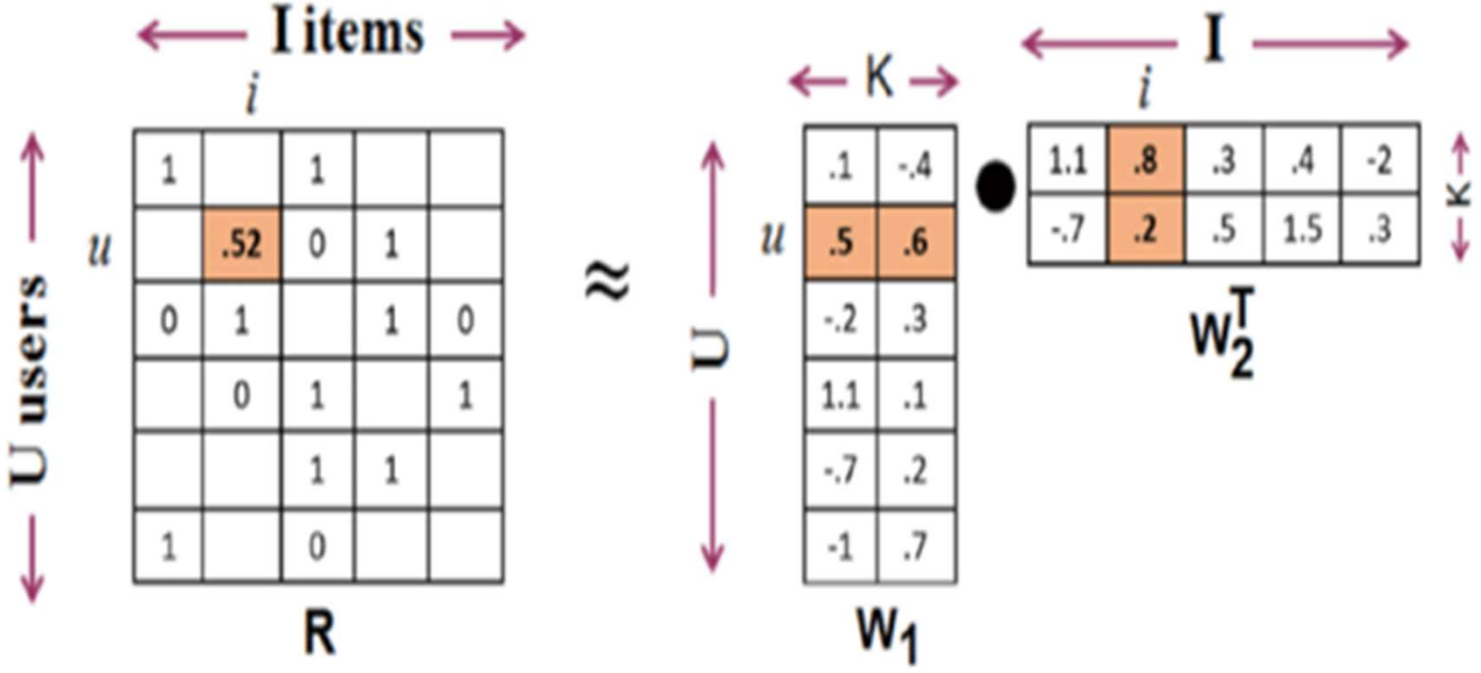
- Обычный планировщик
  - Задачи стартуют независимо
  - Задачи завершаются независимо
  - Задачи рестартуют независимо

- Gang scheduler (Spark 2.4)
  - Задачи стартуют вместе
  - Задачи завершаются вместе
  - Рестартуют тоже всей командой
  - Идеально для Parameter Server

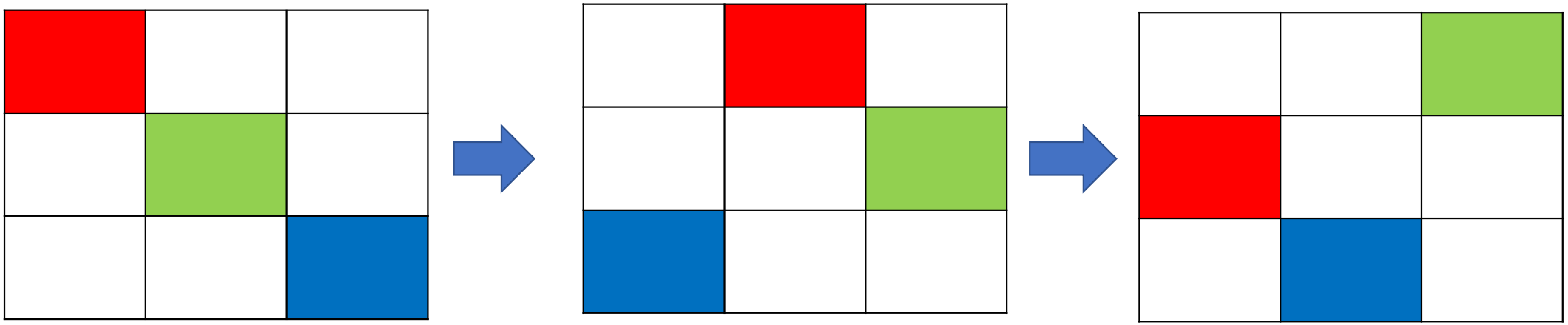
Parameter server

- Модели до нескольких гигабайт
- Добавляет проблем с синхронизацией
- Хорошо работает с «распределенными» моделями
  - Каждое обновление затрагивает только часть параметров

Факторизация матриц



Распределенный блочный SVD



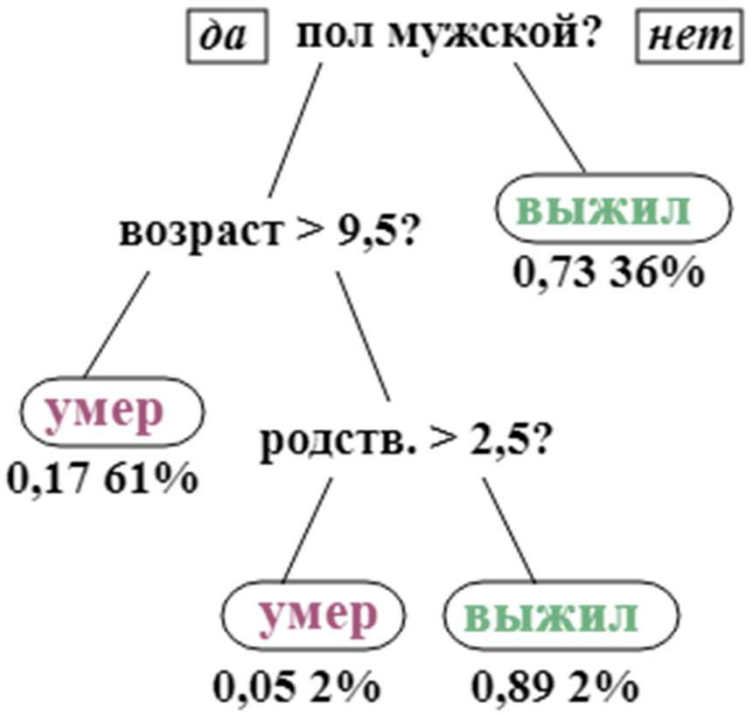
Распределенный блочный SVD

1. Данные нарезаем на столбцы и колонки
2. Кешируем на экзекуторах блоки на пересечениях колонки/столбца с избыточностью
3. Факторами управляем через параметр сервер
4. Задачи раздаем оркестратором

LDA: join модели и данных

- Имеем большой корпус и небольшой словарь
- Ищем тематическую модель для слов и документов
- Комбинируем:
  - Топики документа обновляем независимо от остальных
  - Распределение слов по топикам храним на параметр сервере

Деревья принятия решений



Поиск сплита в узле

- Каждую колонку можно рассматривать независимо

Построение поддеревьев

- Поддеревья строим независимо друг от друга

Выращивание леса

- При бэгинге деревья независимы
- При бустинге деревья растут только последовательно

ML на распределенных задачах

1. Кросс-валидация
   - Нет коммуникации между фолдами при обучении
   - Нет разделяемого изменяемого состояния
   - Большое пересечение по данным для обучения
2. Отбор признаков
3. Мульти-классовая классификация
4. Ансамбли по принципу бэгинга
5. Подбор гиперпараметров

Ключевые выводы

- Избегать распределенного ML
  - Наращивать железо и уменьшать данные
  - Распределенный ETL и централизованный ML
- Небольшие модели тренировать Mapreduce
  - Улучшать качество отбором признаков и тюнингом параметров
- Модели крупнее резать на части
  - Параметр сервер
- Деревья распределяются лучше матриц

Жизненный цикл модели

1. Сбор данных
2. ETL
3. Обучение модели
4. Вывод в ПРОМ (Доход!!!)
5. Эксплуатация (Расходы!!!)