

Распределённые файловые системы

HDFS

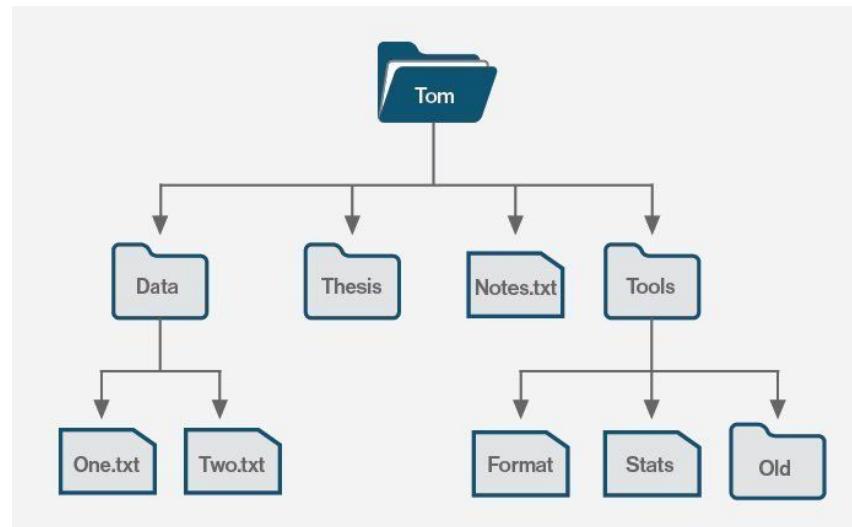


Илья Кокорин

kokorin.ilya.1998@gmail.com

Постановка задачи

- Хотим хранить неструктурированные файлы
 - Легко и просто
 - Когда храним сотни гигабайт
- А мы хотим хранить петабайты данных

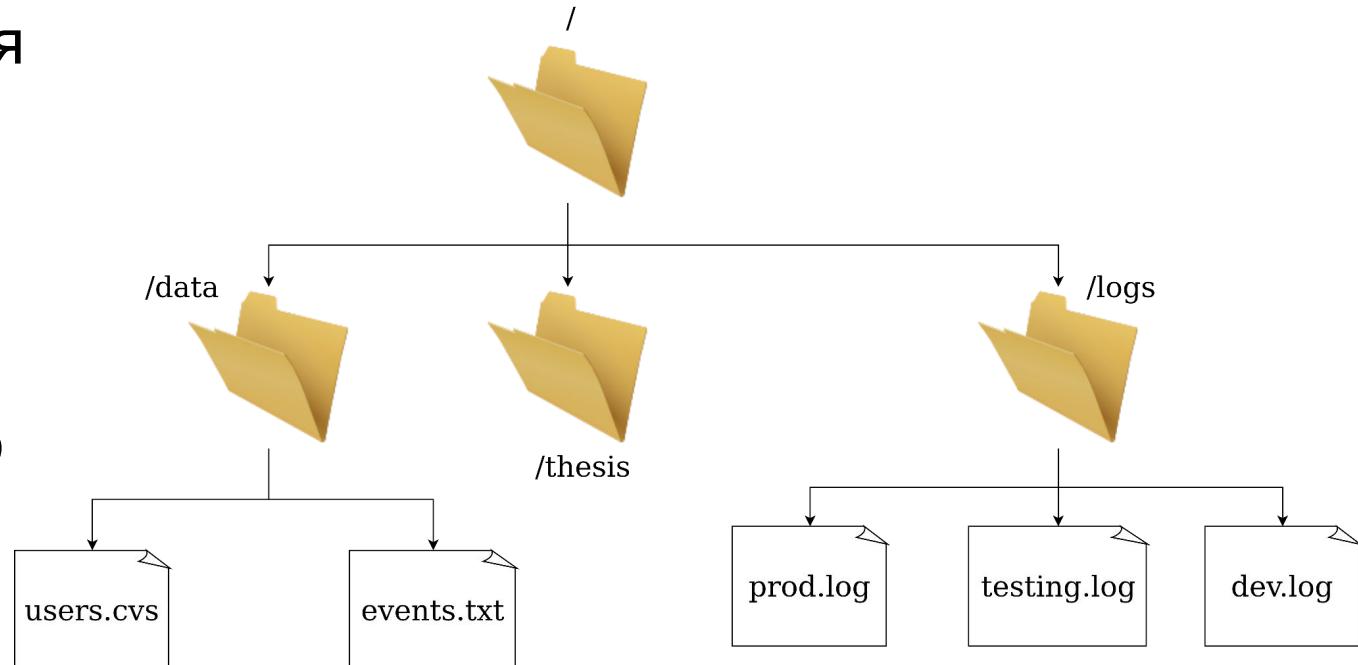


Типы узлов в HDFS

- Два типа узлов в кластере
- NameNode
 - Хранит только метаинформацию
 - Существует в единственном экземпляре
- DataNode
 - Хранит содержимое файлов
 - В кластере могут быть тысячи таких узлов

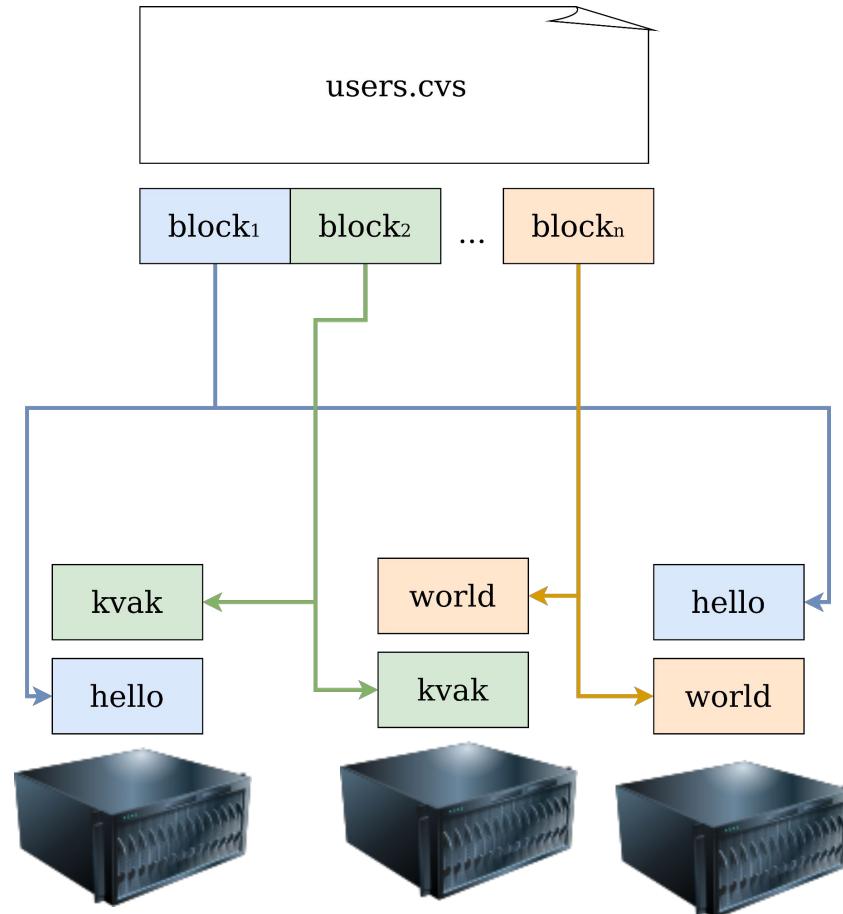
Хранение метаинформации

- Вся метаинформация хранится на NameNode
- Структура файловой системы
- Права доступа
- Даты создания
- ...
- Всё, кроме содержимого
- Нельзя хранить много файлов



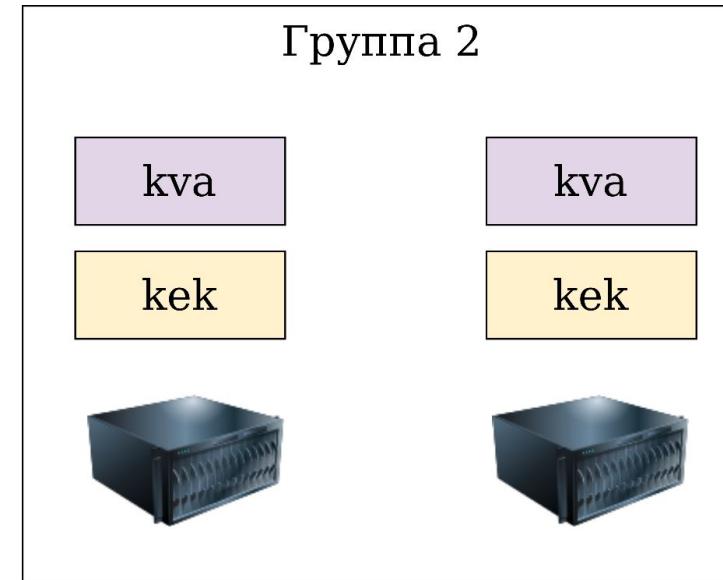
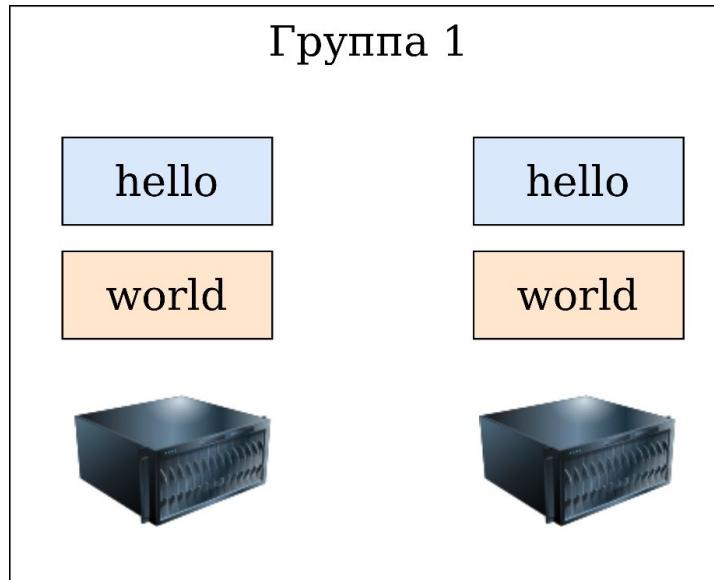
Хранение содержимого файлов

- Файл состоит из блоков
- Каждый блок хранится на нескольких DataNode
- NameNode знает, где лежит каждый блок
 - Сам NameNode хранит только ссылки, не хранит данные
- DataNode только хранит блоки
 - Не знает, какому файлу принадлежит хранимый блок



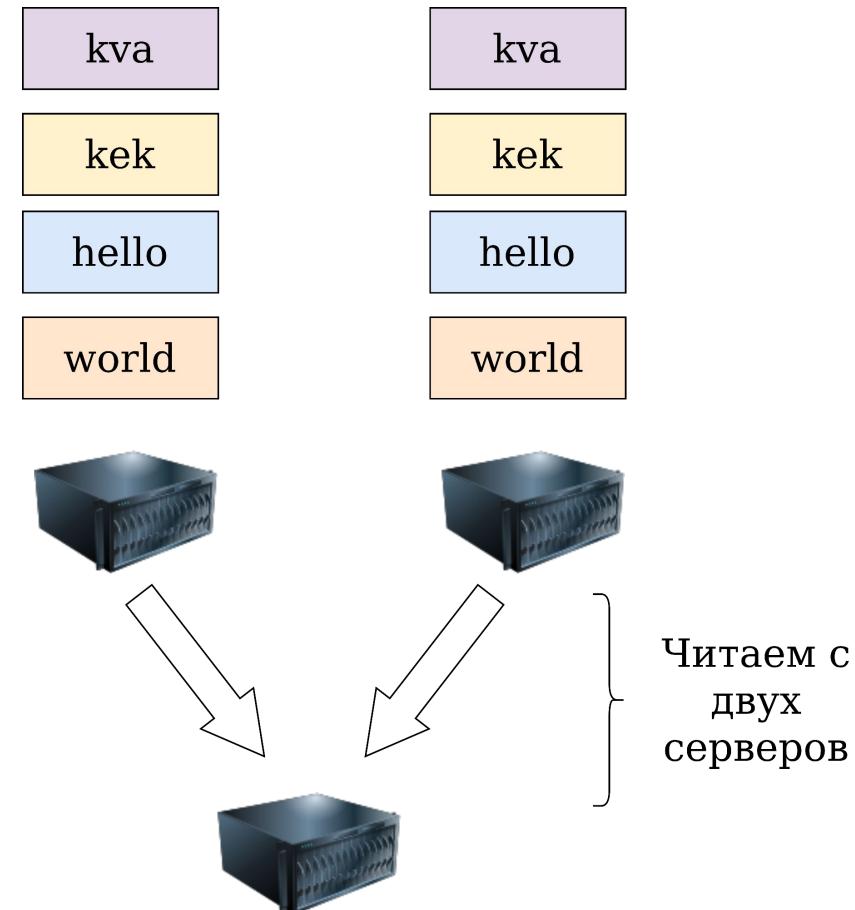
Стратегия размещения: разбиение на группы

- Разбиваем DataNode на группы
- В каждой группе K серверов
- Хранят идентичные блоки
- Нельзя изменить фактор репликации всего одного файла



Стратегия размещения: разбиение на группы

- Добавляем в группу новый сервер
- Чтение осуществляется всего с 2-3 серверов
- Быстро скачать сотню Гб с двух серверов - нетривиальная задача



Стратегия размещения: отсутствие групп

- Каждый узел хранит случайный набор блоков
- Каждый блок хранится на требуемом числе узлов

hello

world

hello

kva

kek

kva

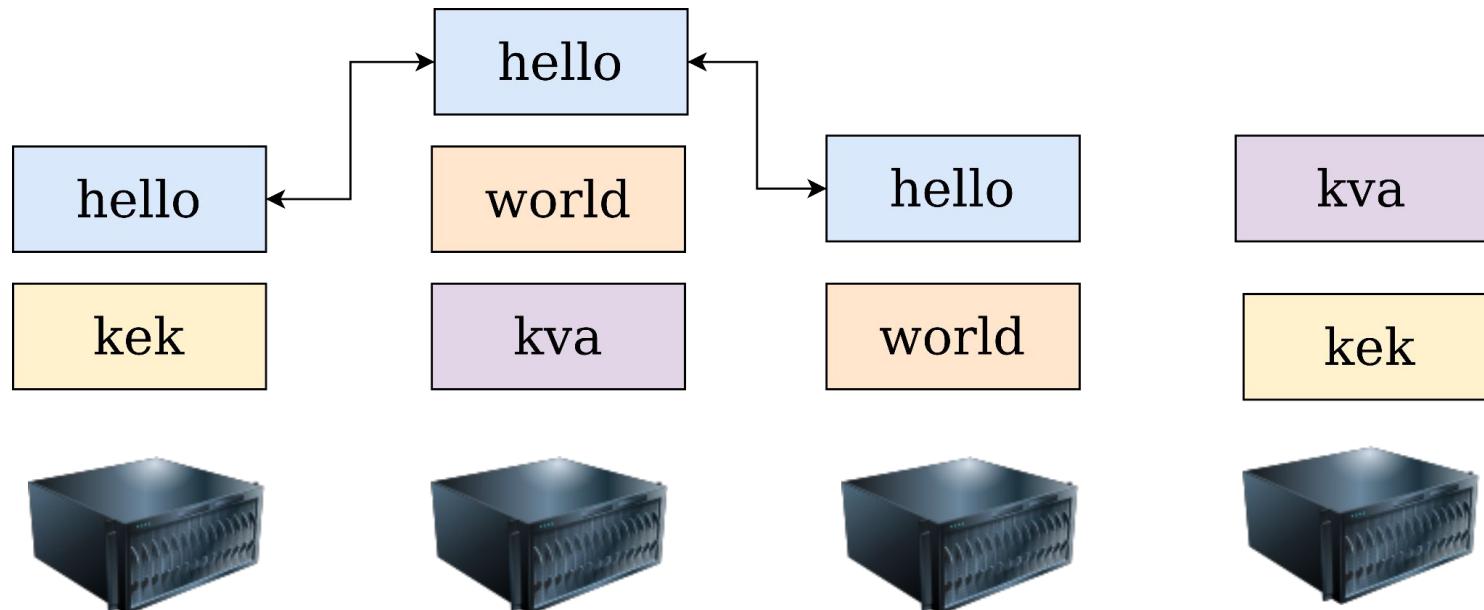
world

kek



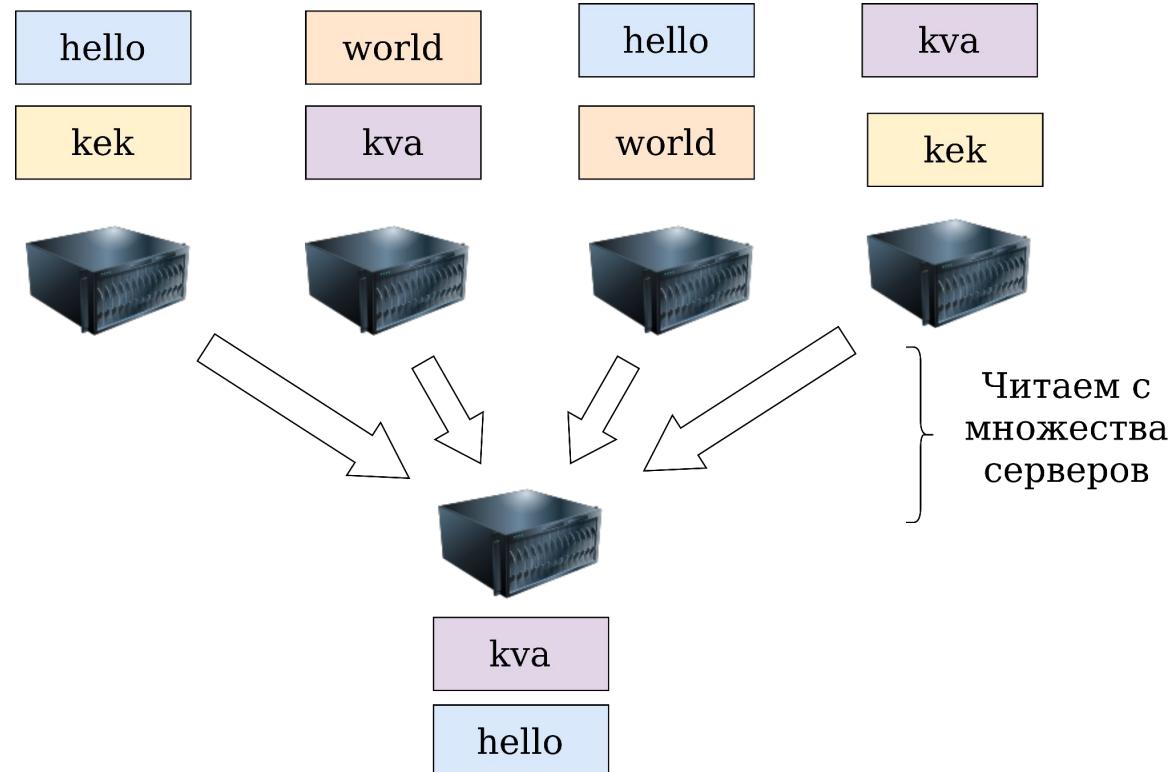
Стратегия размещения: отсутствие групп

- Можно легко изменить фактор репликации у одного файла



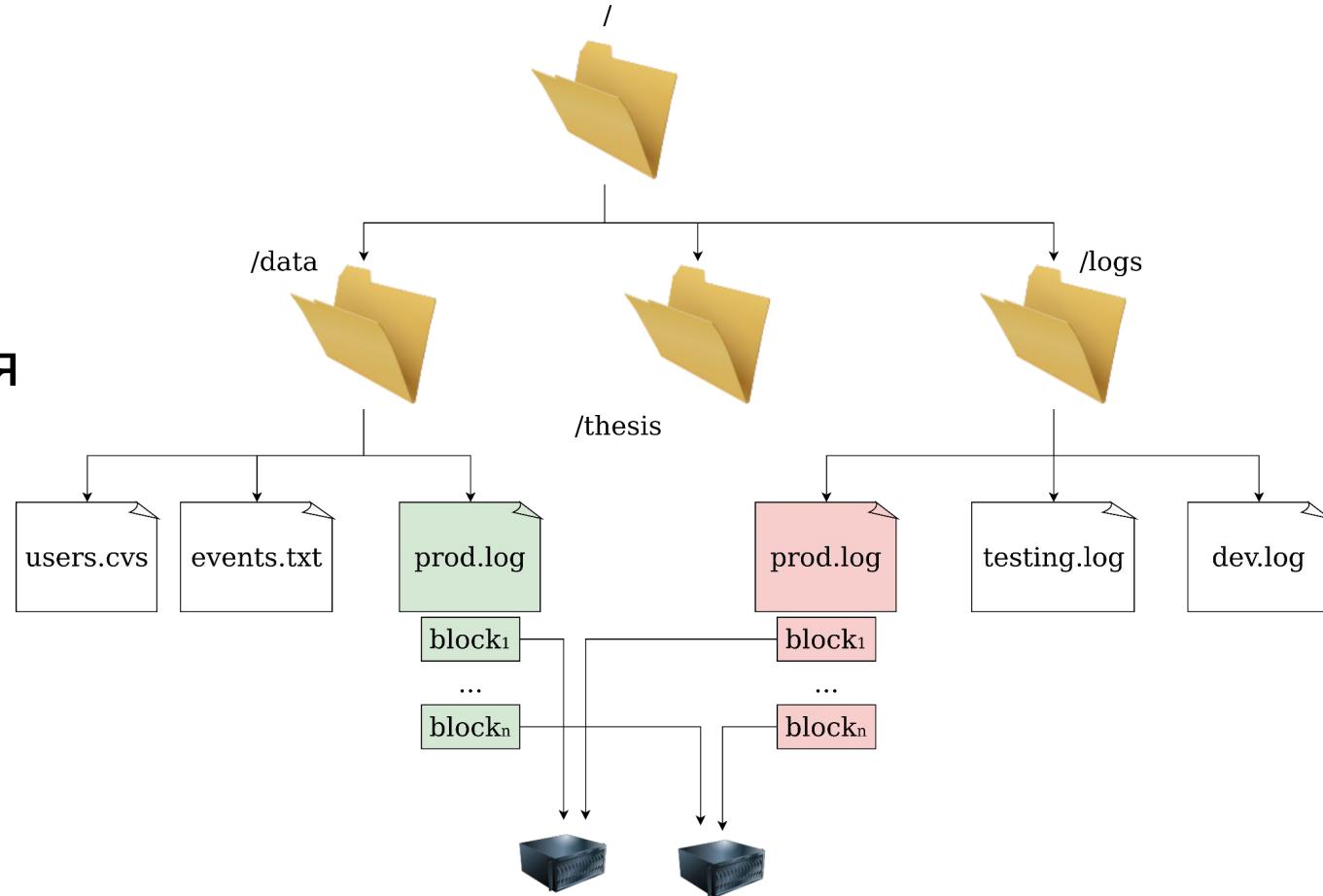
Стратегия размещения: отсутствие групп

- При добавлении нового сервера можно читать данные с множества DataNode
- Со всех, которые хранят нужные ему блоки
- Ускоряем ввод в эксплуатацию новых серверов
- За счёт параллельного чтения



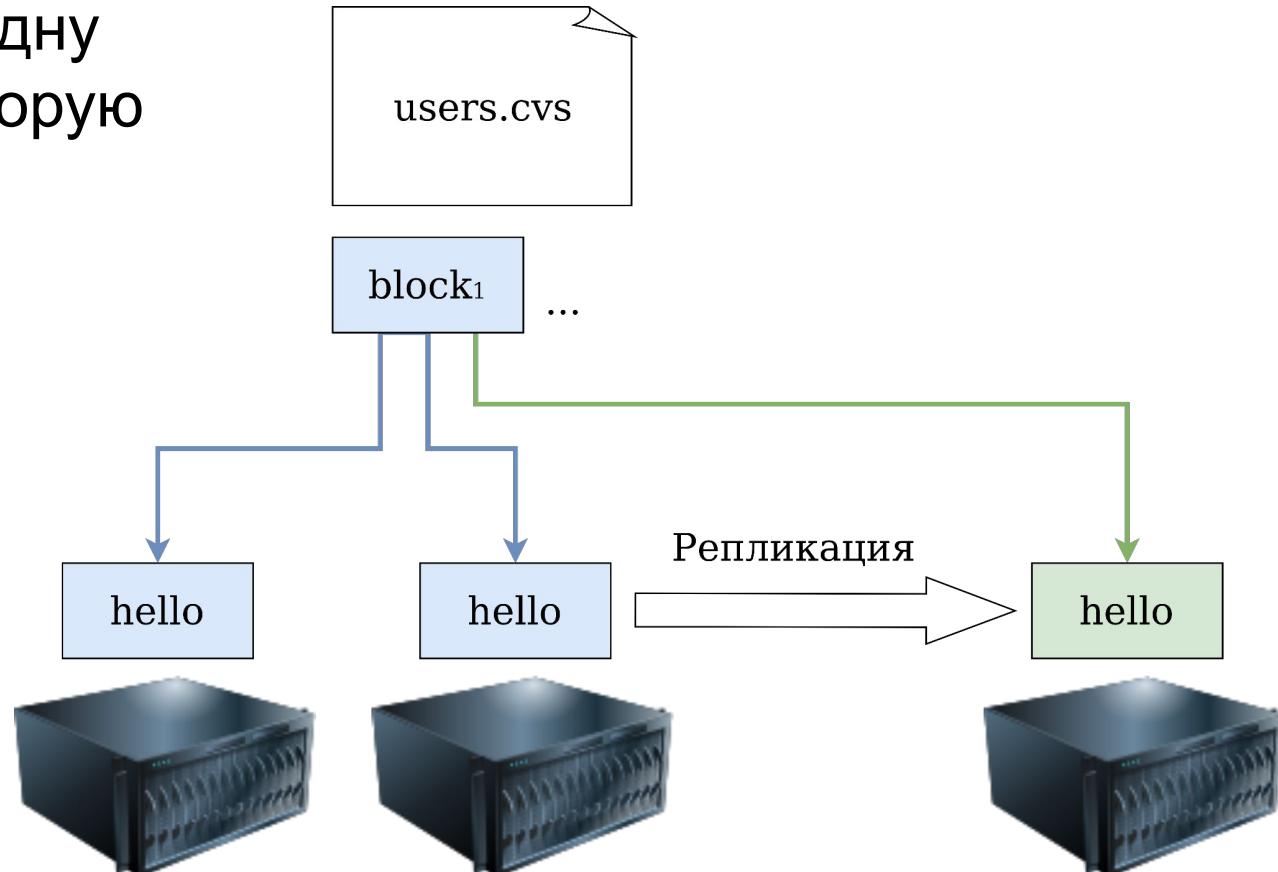
Операции над метаданными

- Происходят в пределах NameNode
- DataNode не задействуются
- Блоки данных ссылаются в те же места



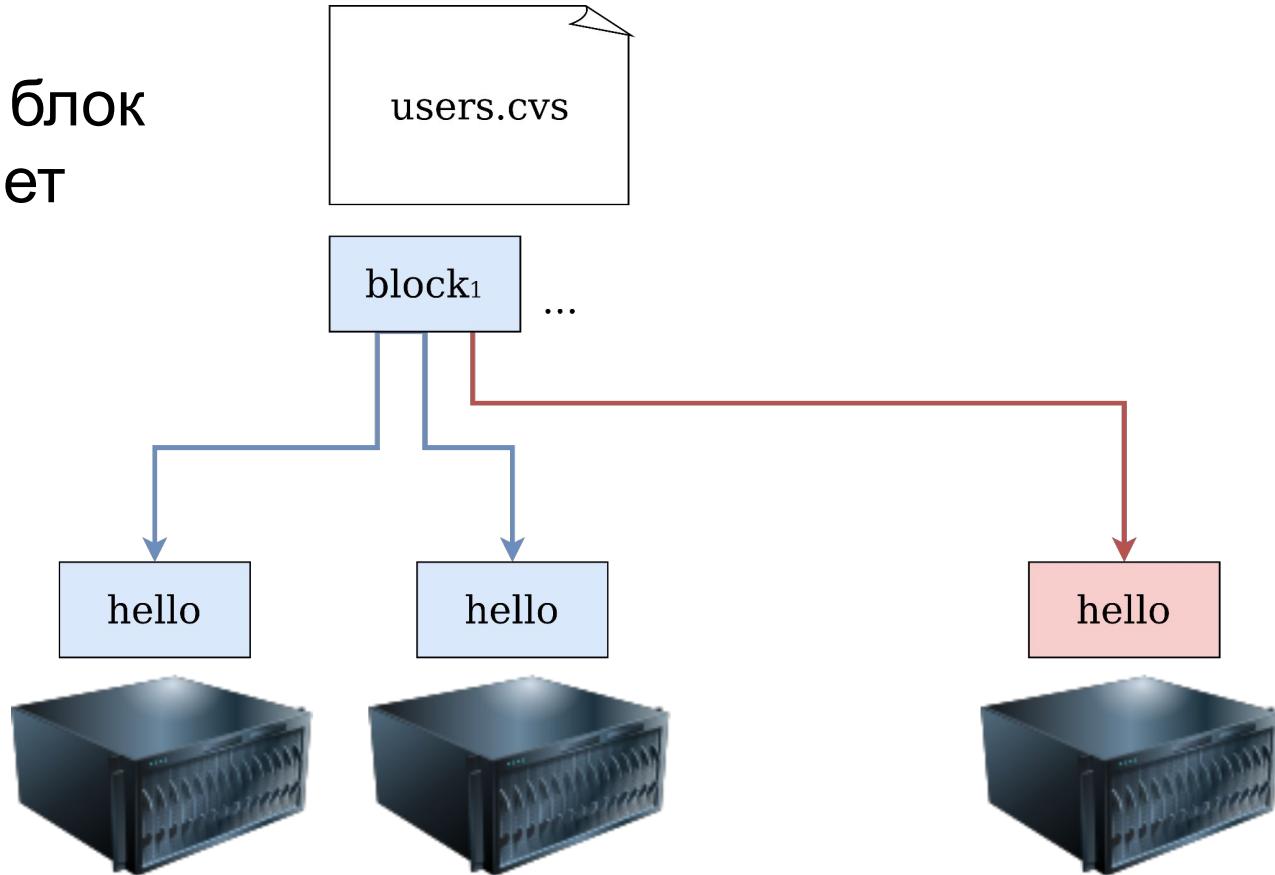
Увеличение числа реплик

- Выбираем ещё одну DataNode, на которую поместим новую реплику
- Репликацией занимаются DataNode
- Данные не проходят через NameNode



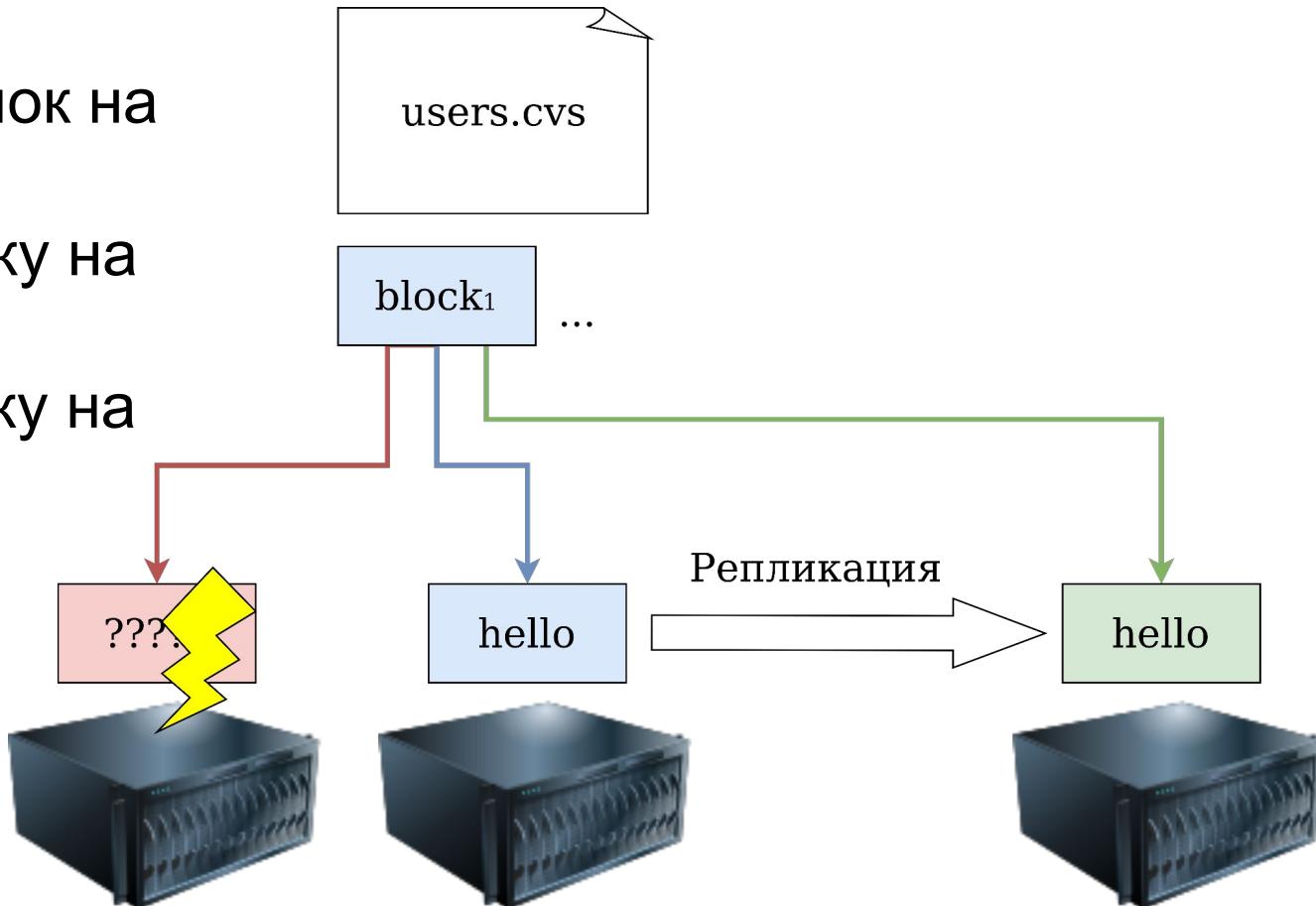
Уменьшение числа реплик

- Даём DataNode команду удалить блок
- NameNode удаляет одну ссылку



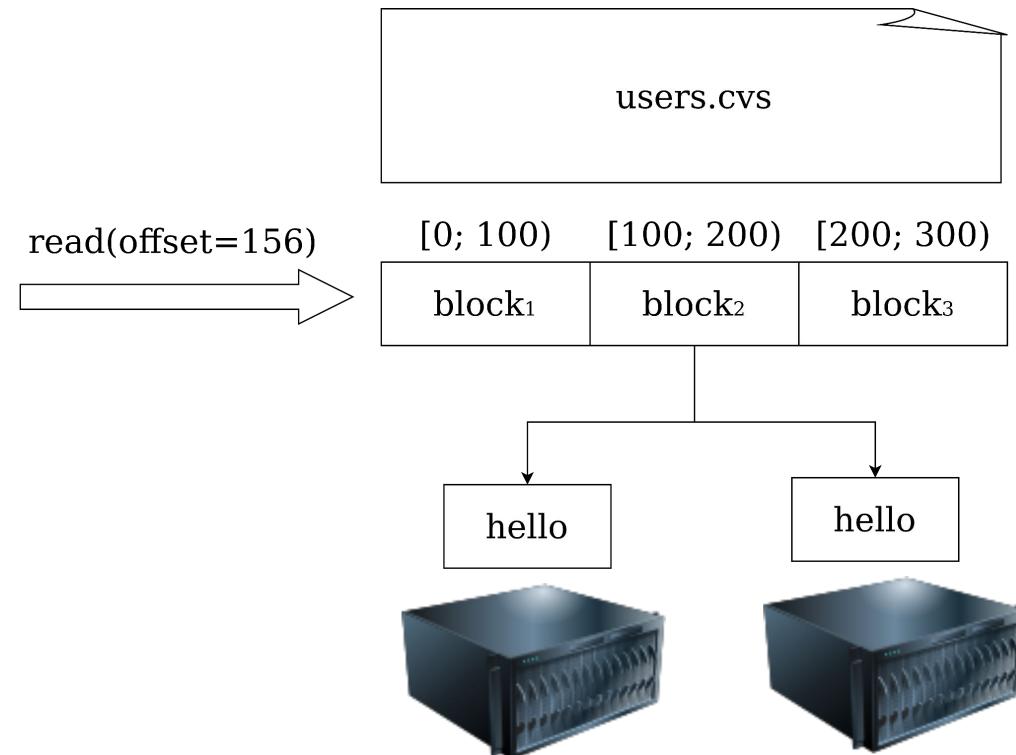
Обработка сбоев DataNode

- Реплицируем потерянный блок на новый узел
- Создаём ссылку на новый блок
- Удаляем ссылку на сбойный блок



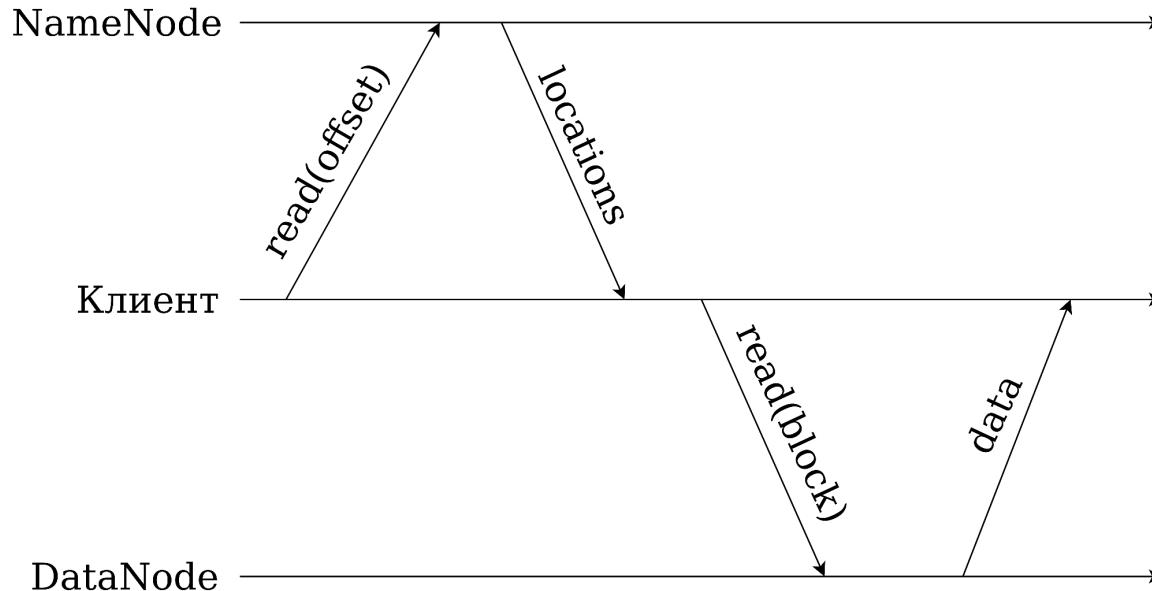
Чтение файлов

- Идём на NameNode
- Спрашиваем, где лежит блок, содержащий нужный нам байт
- Получаем набор реплик
- Читаем с любой из них



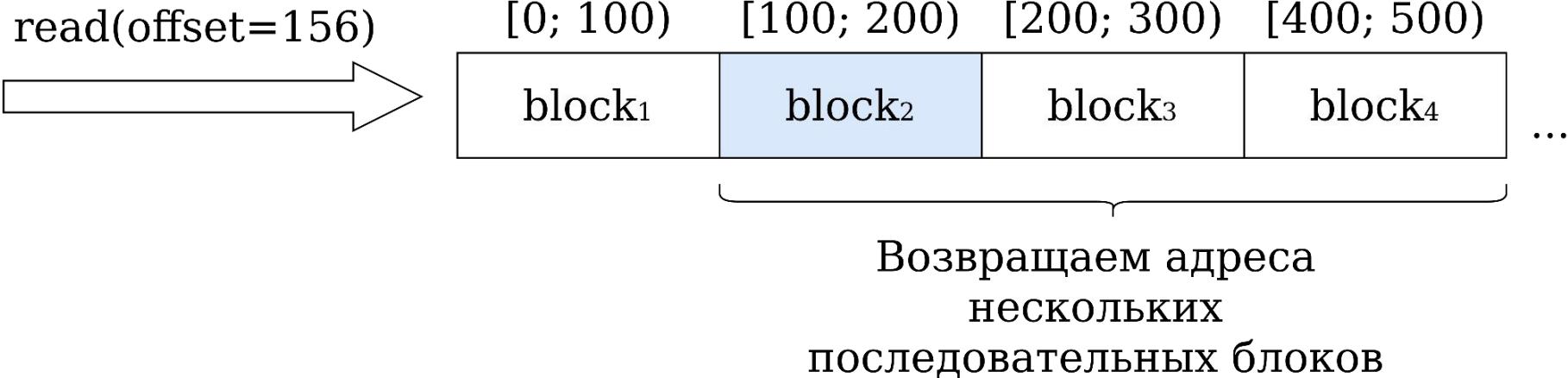
Чтение файлов

- Данные не проходят через NameNode
- NameNode отвечает только за работу с метаданными
- Чтение масштабируется



Чтение файлов: спекулятивное исполнение

- Файлы в основном читаются последовательно
- Скорее всего, после i -ого блока клиент попросит $(i+1)$ -ый, $(i+2)$ -й, и так далее
- Вернём ему их адреса ещё при запросе i -ого блока



Размер блока

- Нельзя делать слишком маленьким
- Будет слишком много блоков
 - Размер блока = 1 Кб
 - Храним 100 Тб данных
 - Получаем сто миллиардов блоков
 - И мёртвую DataNode
- При последовательном чтении каждый новый килобайт нужно читать с нового узла
 - Низкая скорость последовательного доступа

Размер блока

- Нельзя делать слишком большим
- Нужно прочитать и обработать 1 Гб данных
 - При большом размере блока делаем это последовательно
- Десятки / сотни Мб - в самый раз (64, 128, 256)



[0 Мб; 256 Мб]



[128 Мб; 256 Мб]



[256 Мб; 512 Мб]



[512 Мб; 1 Гб]

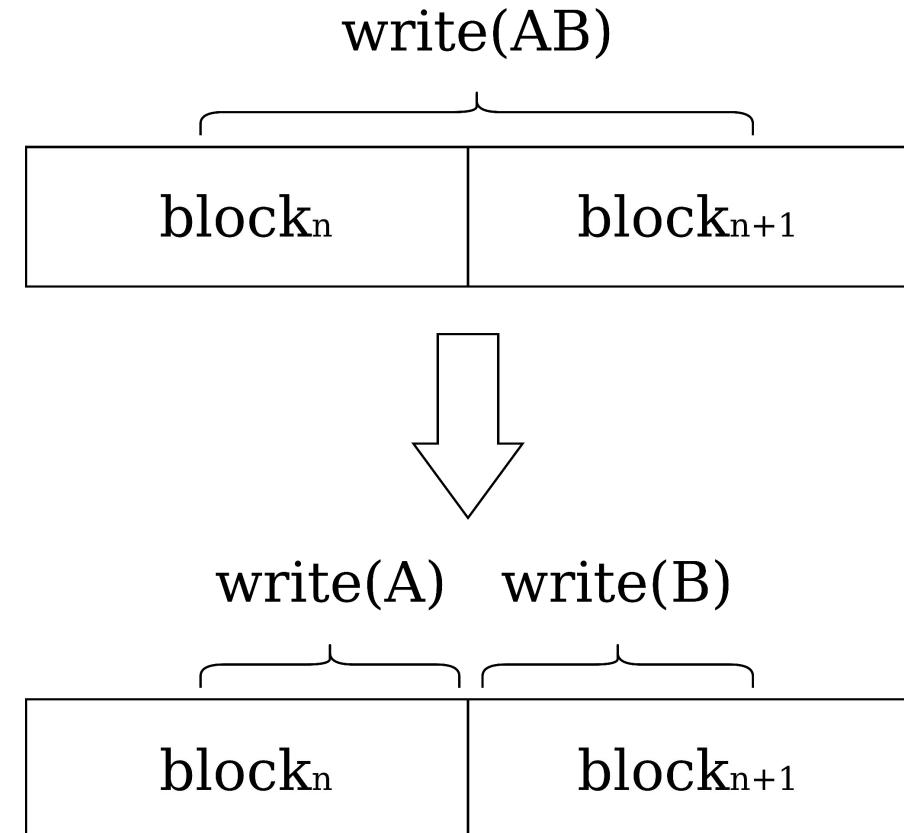


[0 Гб; 1 Гб]



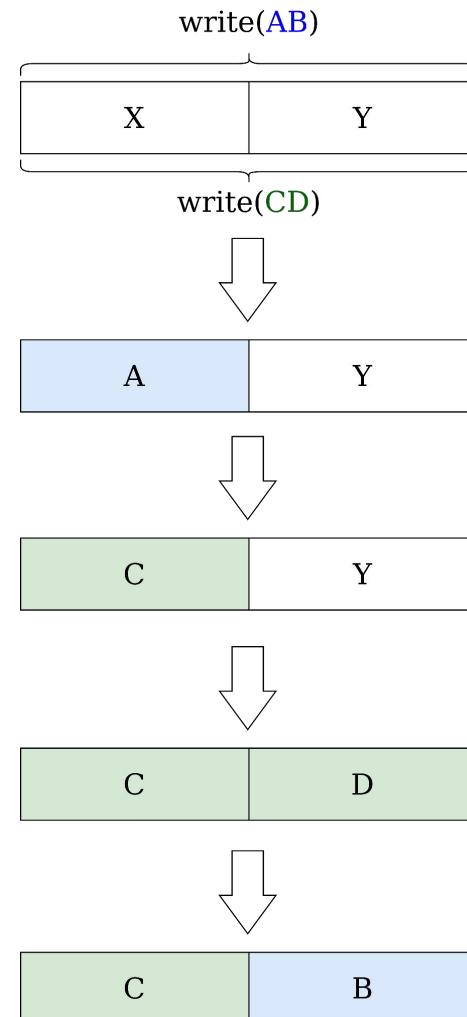
Перезапись файла

- Перезапись может пересекать границы одного блока
- Приходится разбивать её на несколько записей
- Всё очень плохо с консистентностью
 - Первая запись может завершиться успешно, а вторая нет



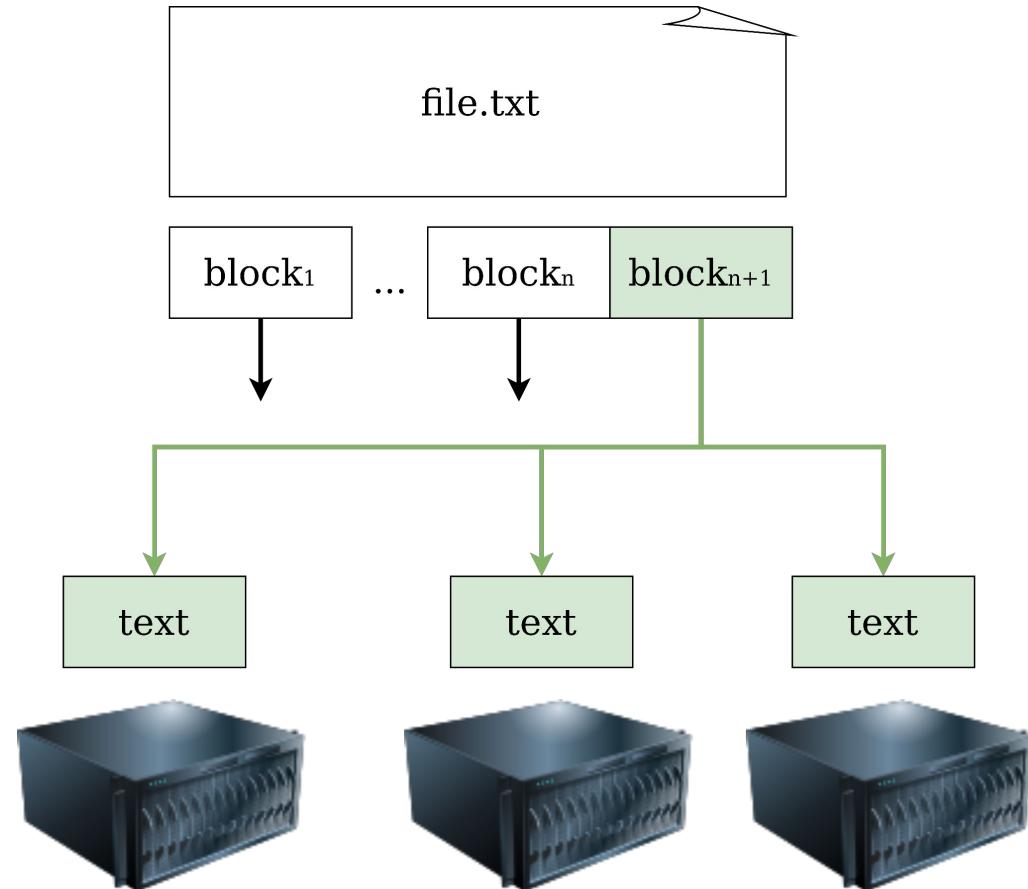
Изменение нескольких блоков

- Может произойти чередование записей
- Пара блоков в неконсистентном состоянии
 - Не в состоянии AB
 - Не в состоянии CD
- HDFS вообще не поддерживает случайную запись
- GFS поддерживает, но не рекомендует пользоваться



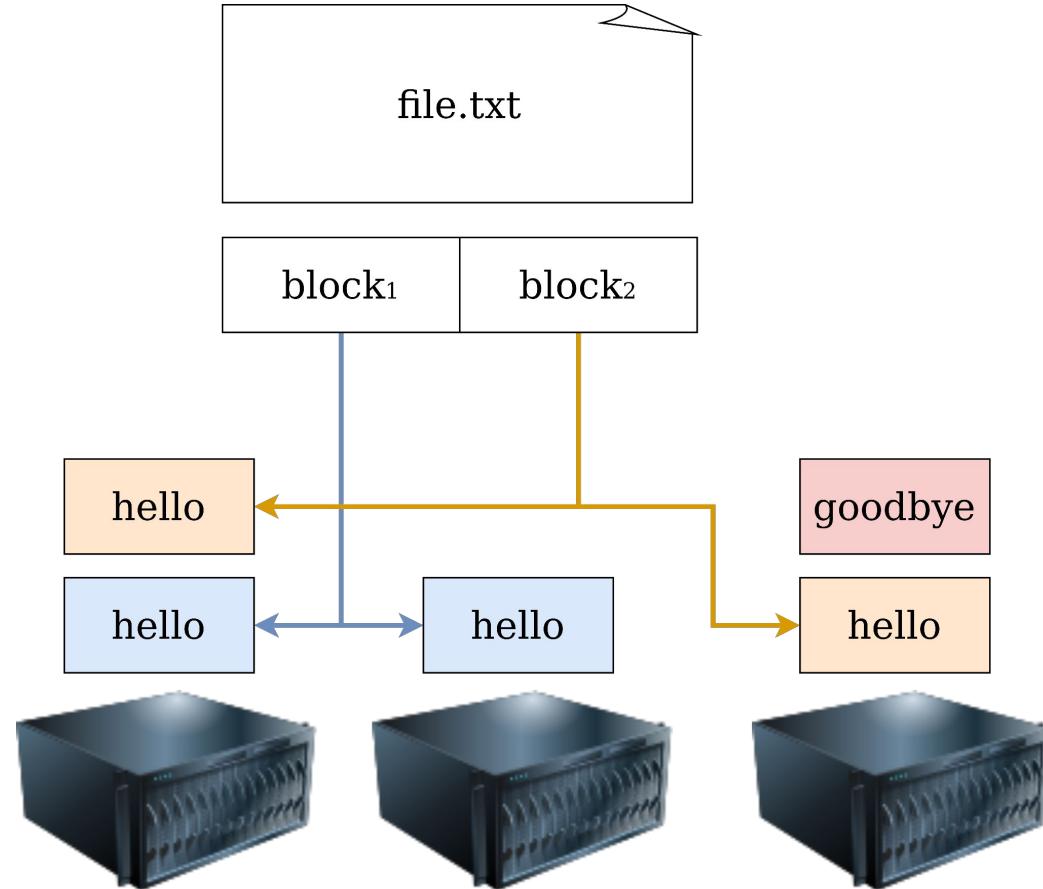
Добавление в конец файла

- Единственная разрешённая операция изменения в HDFS
- Создаём новый блок на нужном числе серверов
- Записываем его на NameNode
- Оставляем ссылки на DataNode, хранящие реплики блока



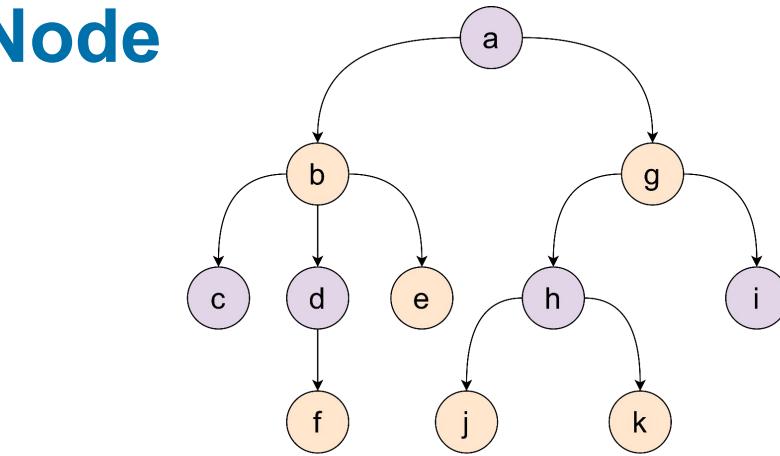
Сборка мусора

- Запись нового блока может не завершиться
- DataNode периодически сообщают NameNode о хранимых блоках
- NameNode командует удалять блоки, которые не являются частью ни одного файла



Масштабирование NameNode

- Дерево со структурой ФС можно шардировать
 - Позволяет обходить дерево как сверху вниз, так и снизу вверх
 - HDFS не умеет
- Реплицируем для надёжности
 - HDFS умеет



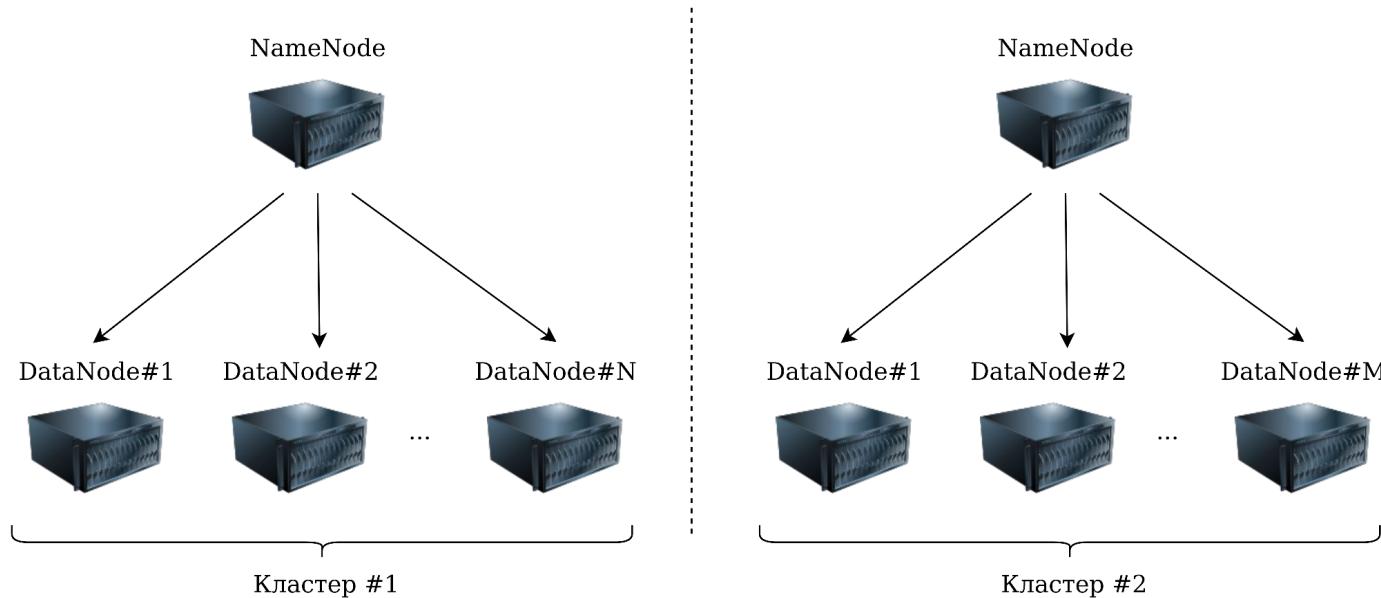
```
Node { a; parent=nil, children=[b, g]; }
Node { d; parent=b, children=[f]; }
Node { c; parent=b, children=[ ]; }
Node { h; parent=g, children=[j, k]; }
Node { i; parent=g, children=[ ]; }
```



```
Node { b; parent=a, children=[c, d]; }
Node { g; parent=a, children=[h, i]; }
Node { e; parent=b, children=[ ]; }
Node { j; parent=h, children=[ ]; }
Node { h; parent=h, children=[ ]; }
```

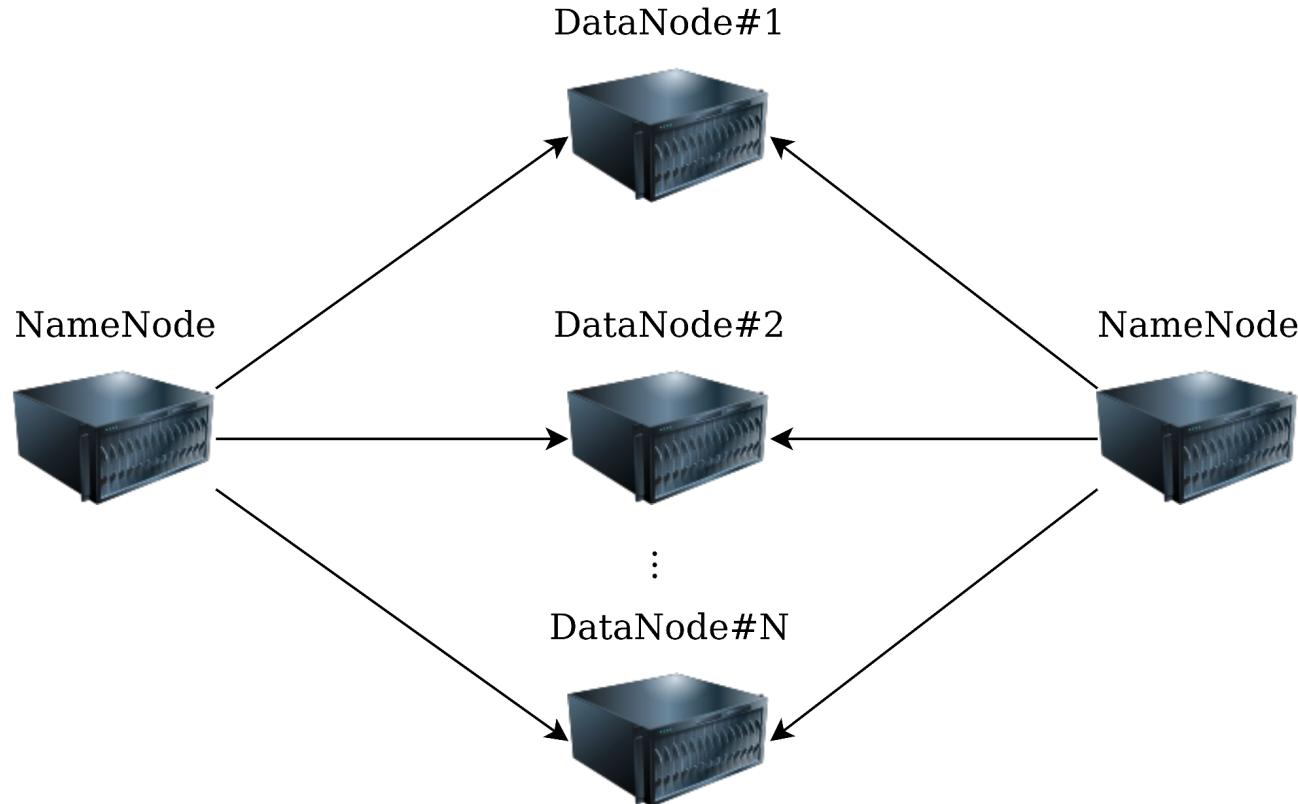
А что тогда HDFS умеет?

- Заводите несколько изолированных кластеров
 - В первом храним логи и метрики
 - Во втором - пользовательские данные
- Сколько DataNode выделить каждому кластеру?



NameNode Federation

- Пул серверов
СОВМЕСТНО
используется
несколькими
NameNode



Недостатки репликации

- Храним 4 логических блока
- Но занимаем 8 физических мест
- С 3 (4, 5,...)-кратной репликацией всё ещё хуже
- Хранить в одной копии не можем
 - Пострадает надёжность

hello

world

hello

kva

kek

kva

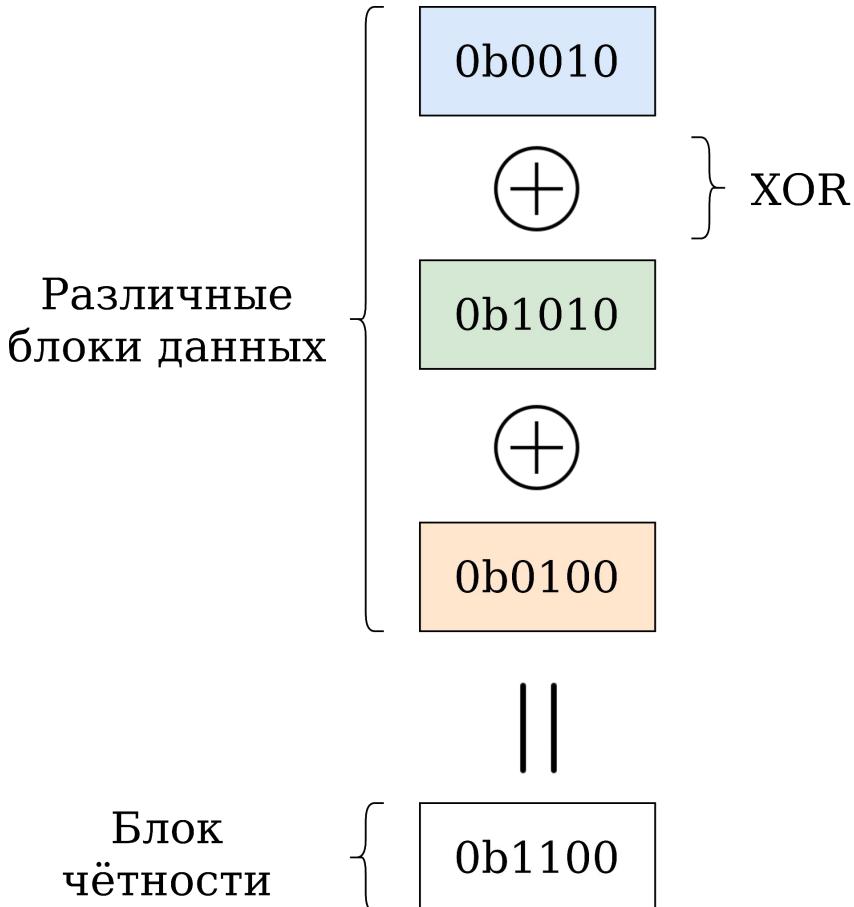
world

kek



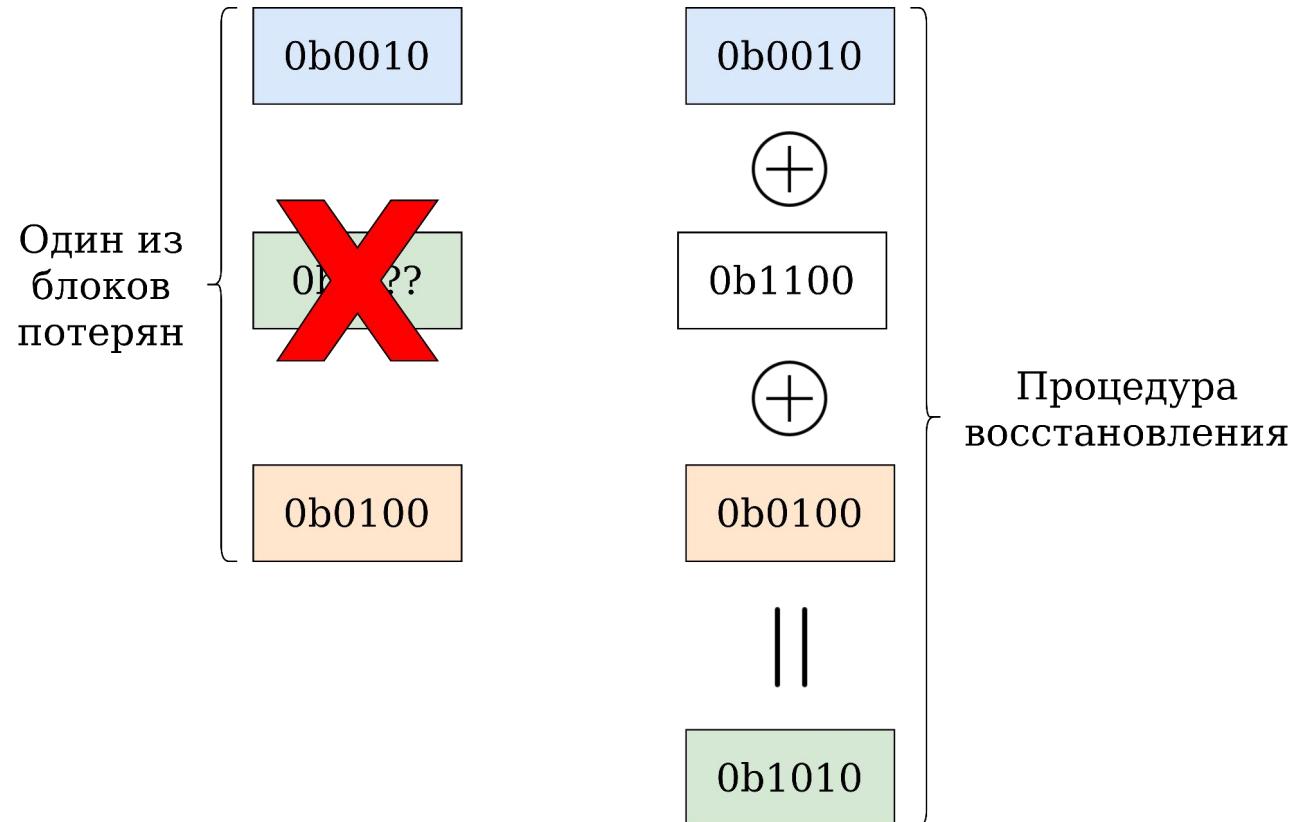
Помехоустойчивое кодирование: XOR

- Берём **N различных** блоков
- Считаем блок чётности по формуле $P = B_1 \oplus B_2 \oplus \dots \oplus B_N$



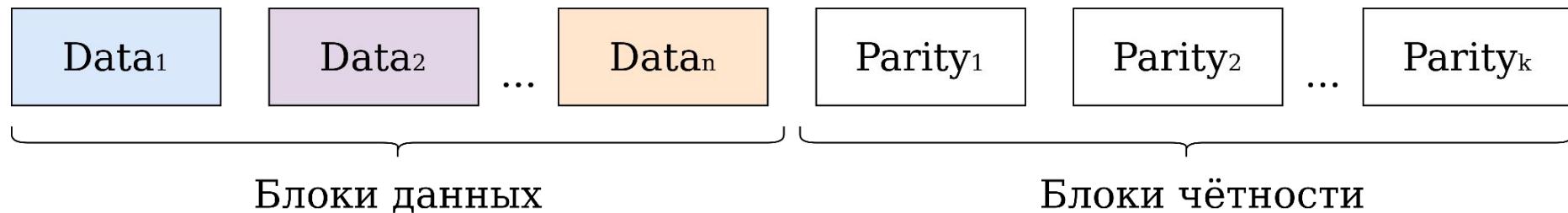
Помехоустойчивое кодирование: XOR

- Переживаем потерю одного блока
- Потерю двух и больше не переживаем
- Храним один служебный блок на N блоков с данными



Коды Рида-Соломона

- Призываю себя на помощь весь аппарат Теории Помехоустойчивого Кодирования
- Используем (N, K) -коды Рида-Соломона
- На N **различных** блоков с данными получаем K блоков чётности
- Можем потерять до K **любых** блоков из $(N + K)$
- И всё равно восстановить их

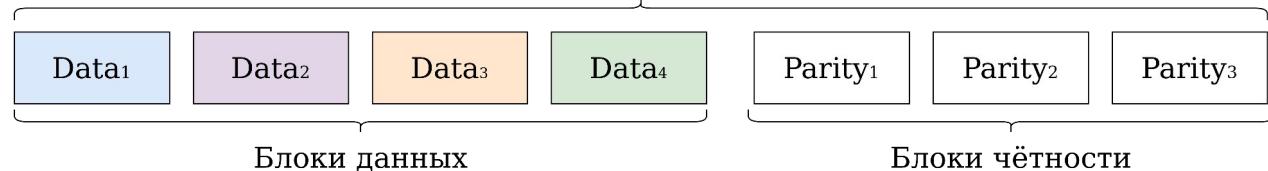


Коды Рида-Соломона

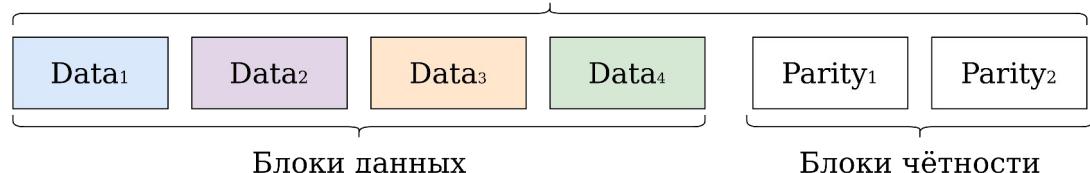
- Как выбрать N и K?
- Представим, что N мы зафиксировали
 - Например, N = 4

- Может
обменять
память на
надёжность

Переживаем потерю 3 блоков
Тратим на 75% больше места



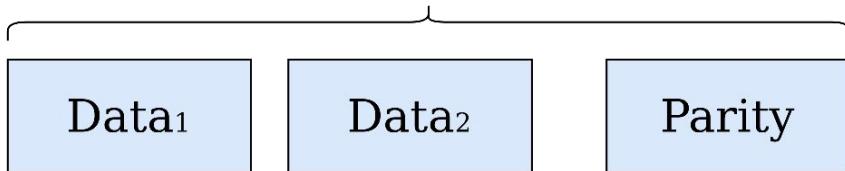
Переживаем потерю 2 блоков
Тратим на 50% больше места



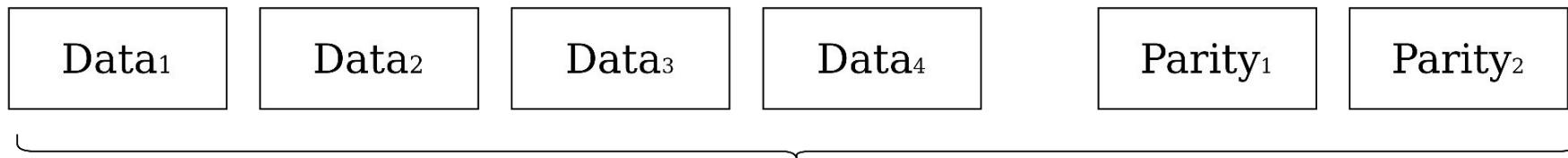
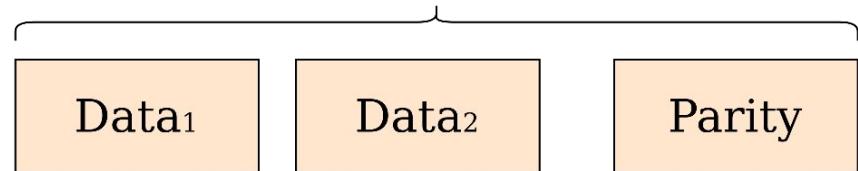
Коды Рида-Соломона

- Как выбрать между (2, 1) и (4, 2)?
 - В обоих случаях тратим на 50% больше памяти
 - Кажется, что надёжность одинаковая
 - Можем пережить потерю до двух блоков из шести

Группа блоков 1



Группа блоков 2

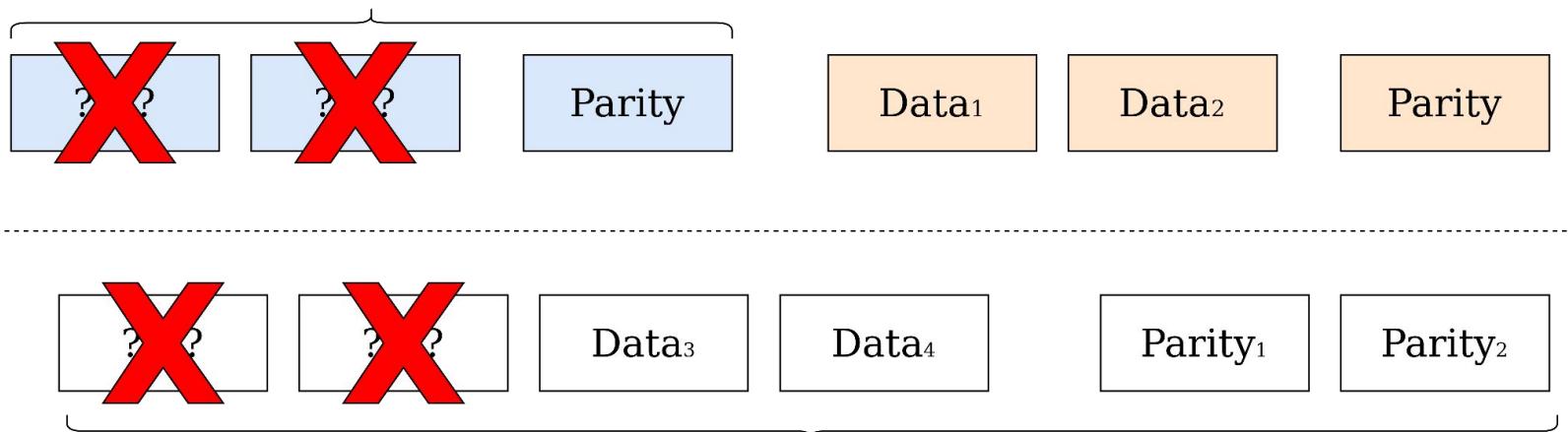


Одна большая группа блоков

Коды Рида-Соломона

- Надёжность в случае (4, 2) выше

Не переживаем потерю
двух соседних блоков



Переживаем потерю
двух соседних блоков

Коды Рида-Соломона

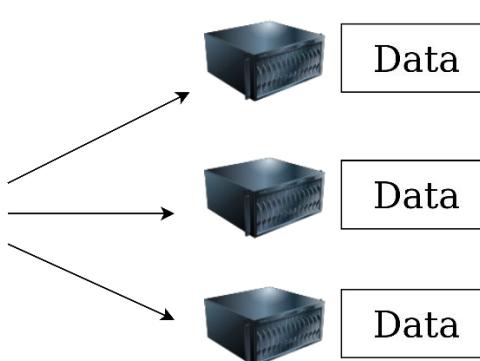
- Почему тогда не использовать код $(1000, 500)$?
- Потому что в (N, K) -коде для восстановления единственного потерянного блока нужно прочитать $\Omega(N)$ блоков
- Выбирая между $(4, 2)$ и $(2, 1)$ мы выбираем между надёжностью и скоростью восстановления



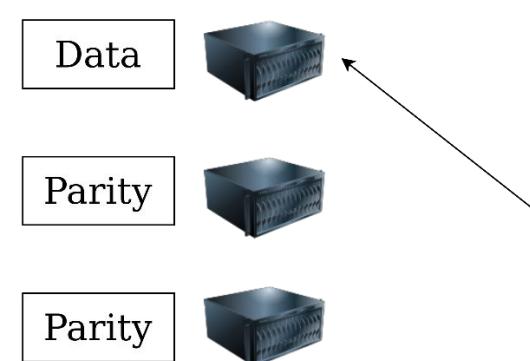
Проблемы помехоустойчивого кодирования

- Нельзя масштабировать чтение
 - В отличие от репликации
 - Только повышать надёжность
- Горячие данные реплицируем
- Холодные данные кодируем

Чтение масштабируется

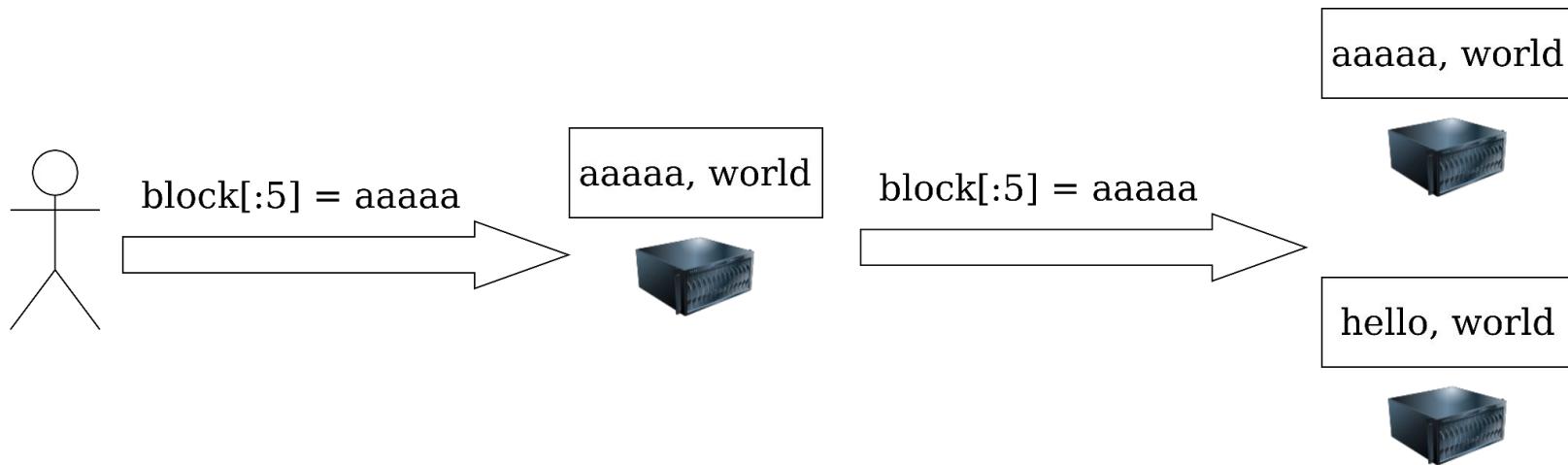


Чтение не масштабируется



Запись в реплицируемый блок

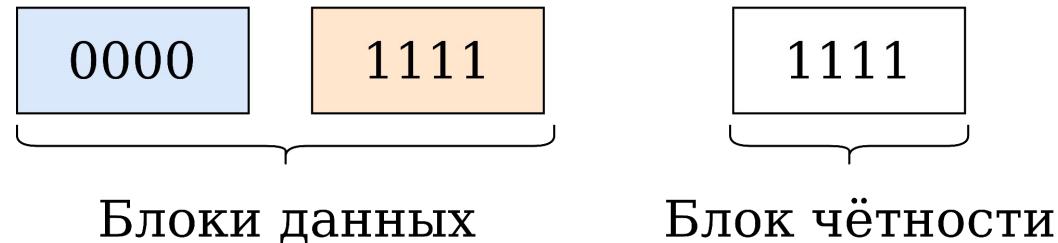
- Запись может не сразу дойти до всех реплик
- Репликация завершится в фоне
- Это нормально



Запись в закодированный блок

- Используем XOR-кодирование

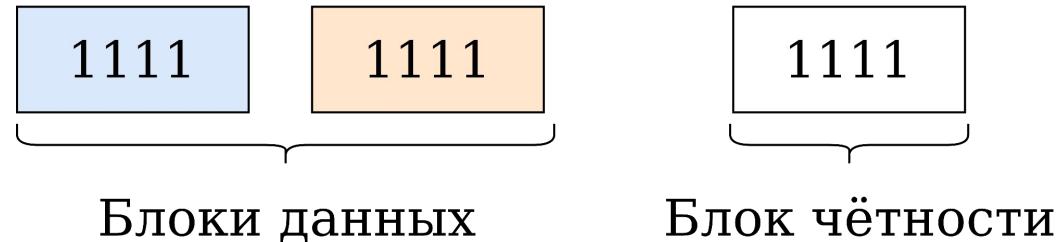
- Представим для простоты: с кодами Рида-Соломона будет та же проблема



Запись в закодированный блок

- Поменяли блок с данными, не успели поменять блок чётности

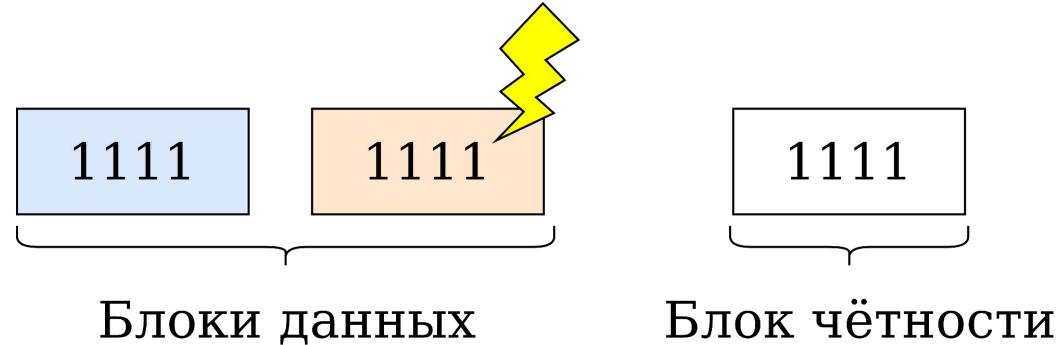
- Неатомарные изменения при записи это нормально



Запись в закодированный блок

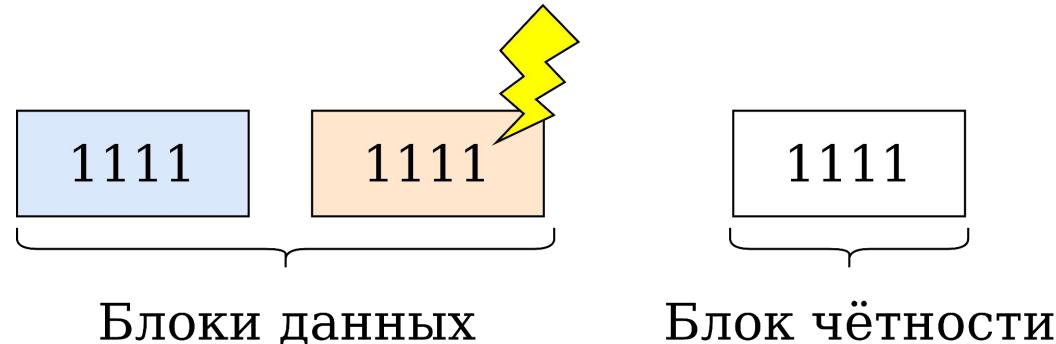
- Потеряли второй блок с данными

- Сбой может произойти в любой момент



Запись в закодированный блок

- Восстанавливаем не то, что хранилось
 - Закодированные блоки являются иммутабельными
- Либо нужна поддержка транзакций



$$\begin{array}{c} 1111 \\ + \\ 1111 \end{array} = \boxed{0000}$$

Добавление новых блоков

- Храним блоки в (3, 2)-коде

Data₄

Data₅

Data₆

Parity

Parity

Data₁

Data₂

Data₃

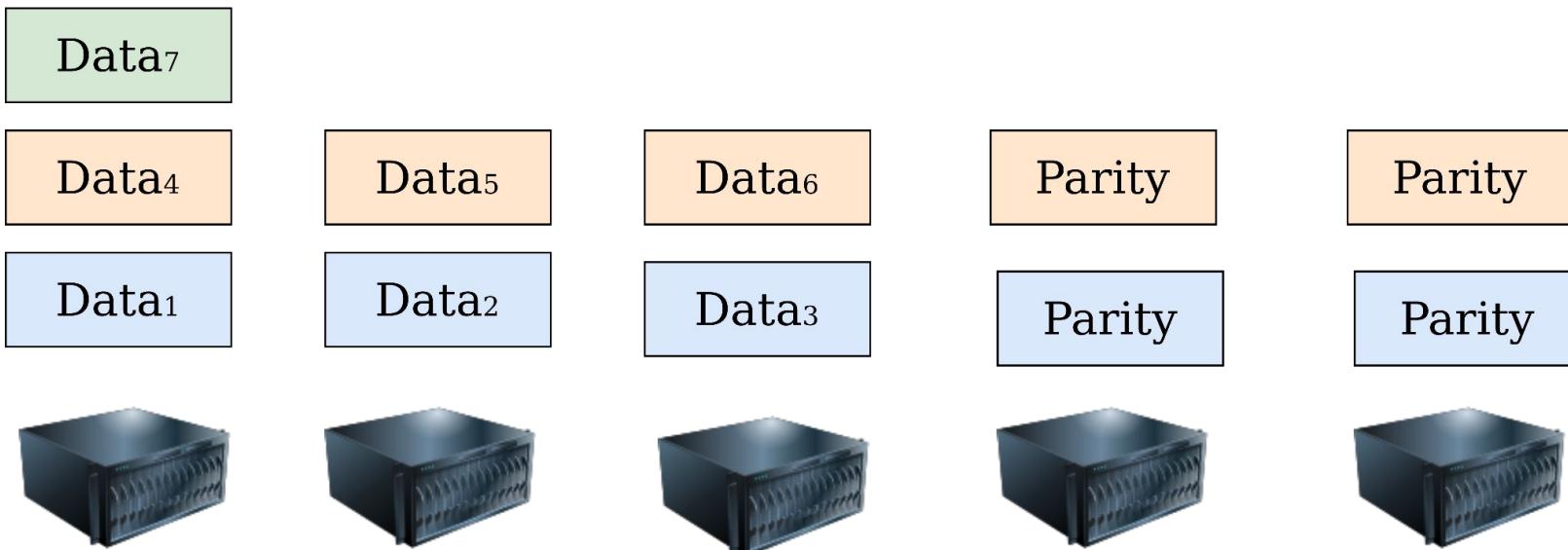
Parity

Parity



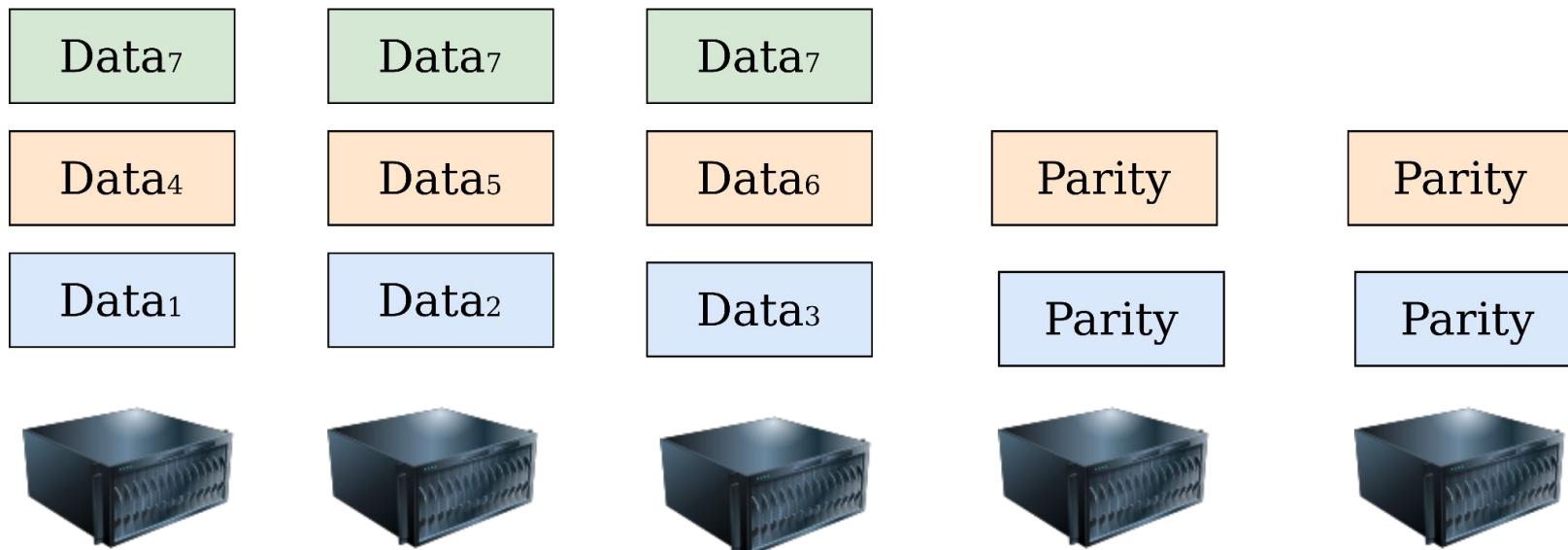
Добавление новых блоков

- Добавили новый блок
- Ещё двух блоков у нас нет



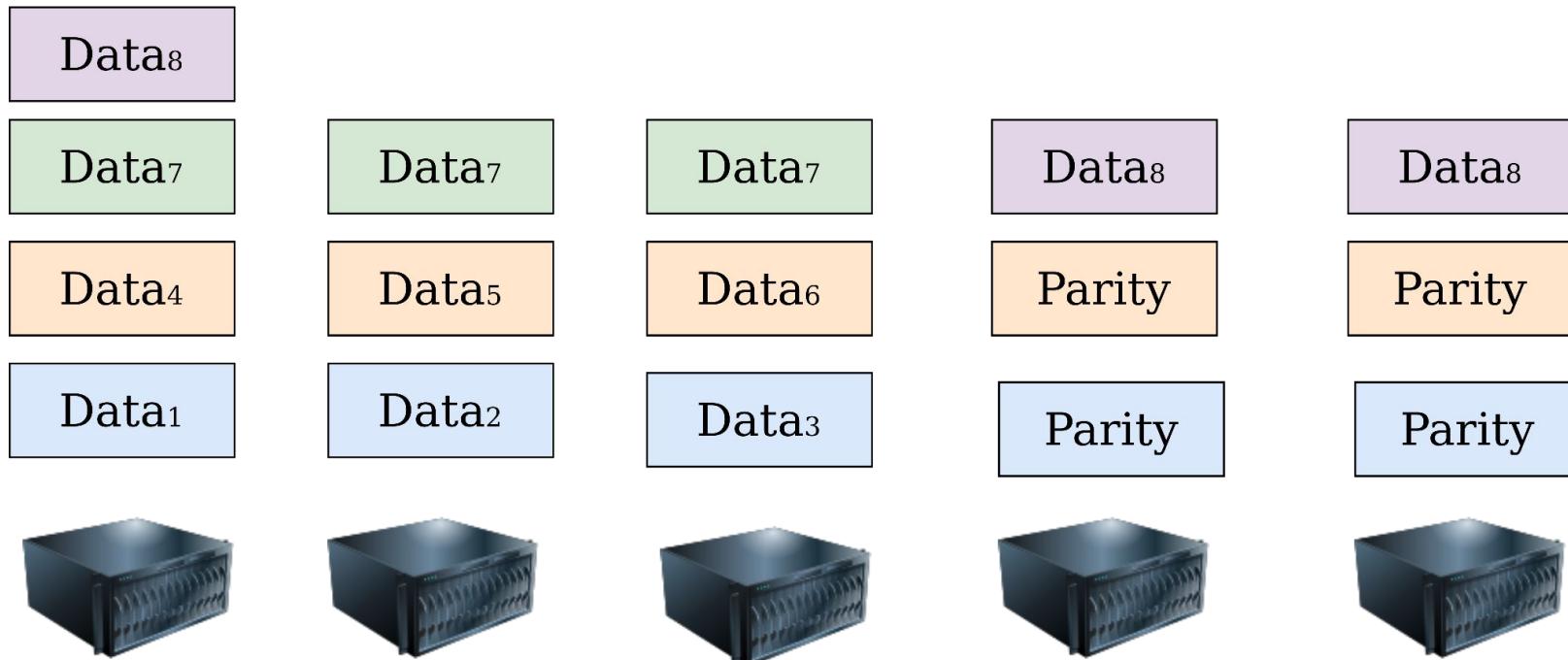
Добавление новых блоков

- Реплицируем новый блок



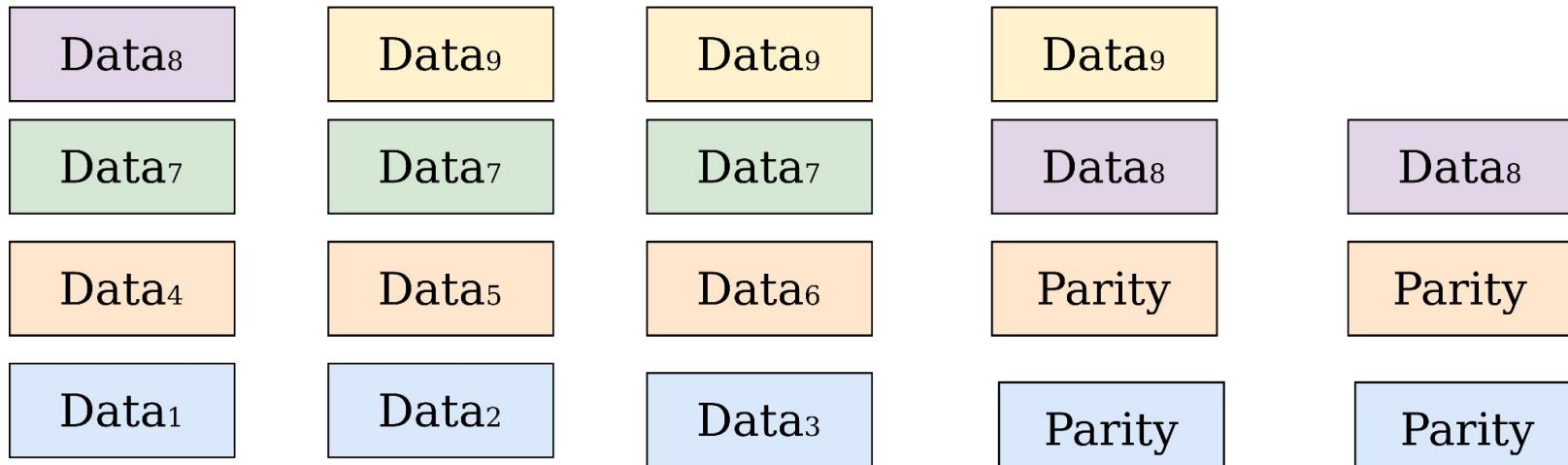
Добавление новых блоков

- И следующий блок



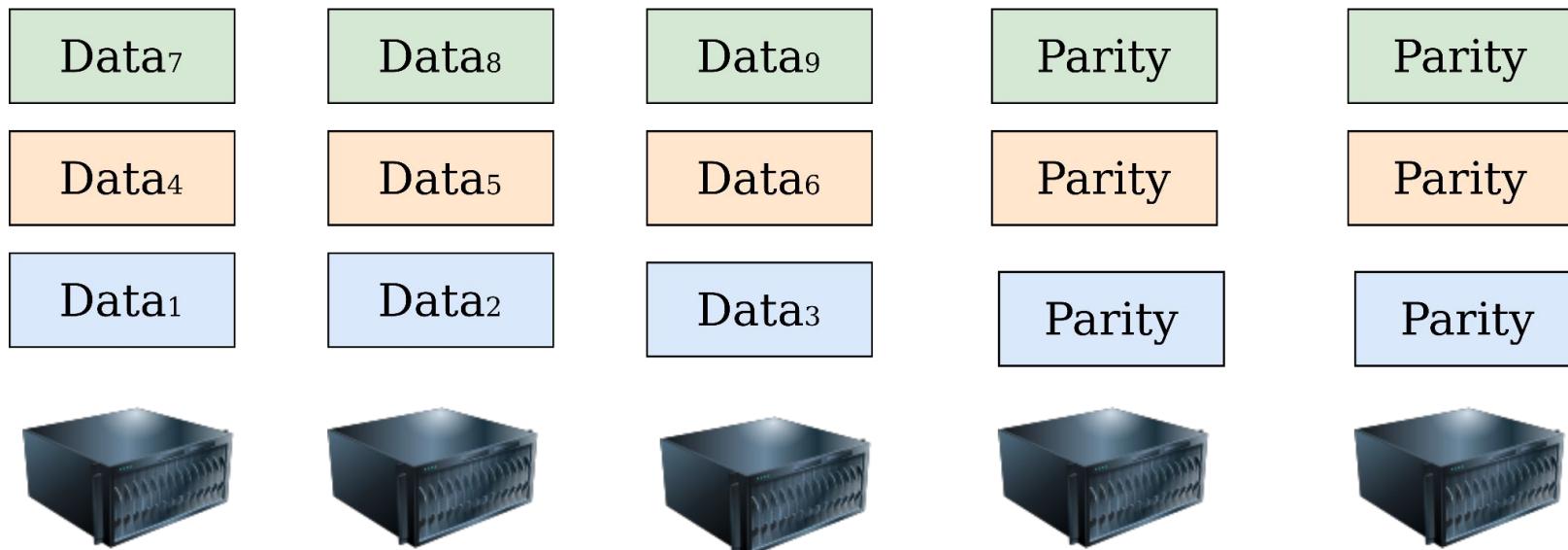
Добавление новых блоков

- И ёщё один
 - Теперь у нас есть три новых блока



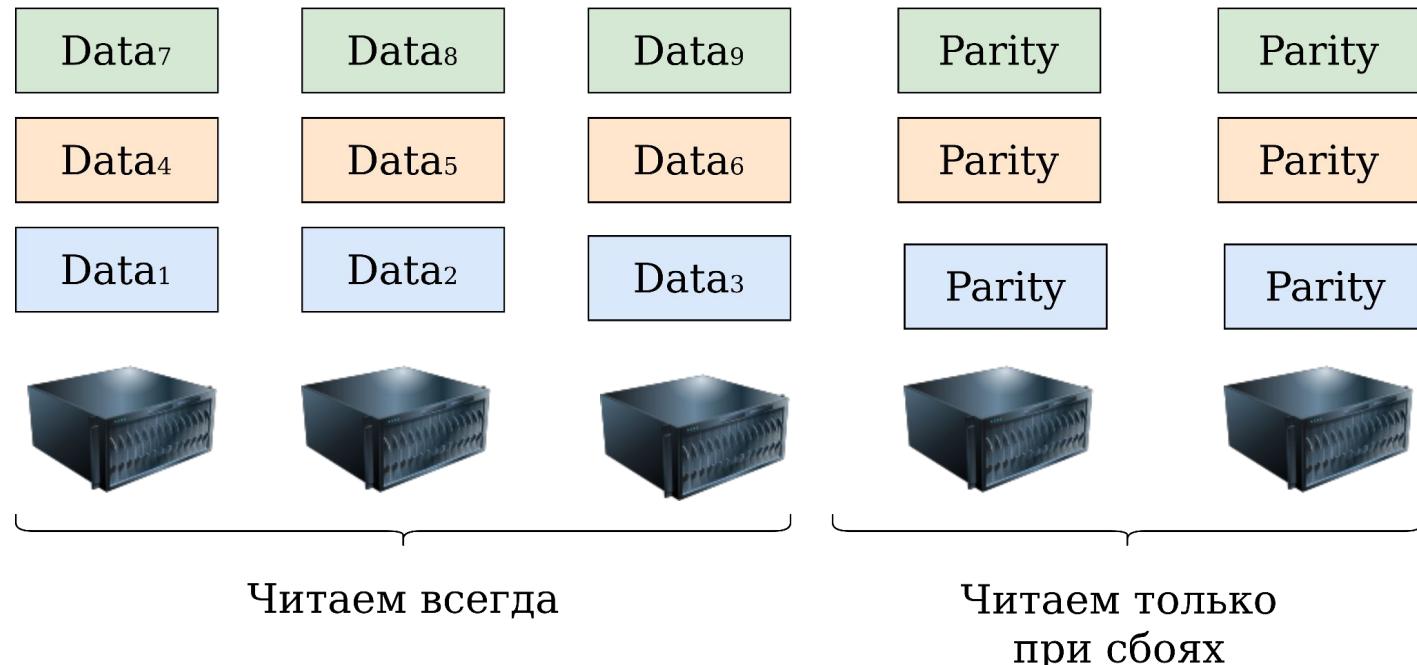
Добавление новых блоков

- Кодируем новую тройку блоков
 - Уменьшая размер занимаемого места почти в 2 раза
- Обычно этим занимается фоновый процесс
 - Периодически сканирующий HDFS



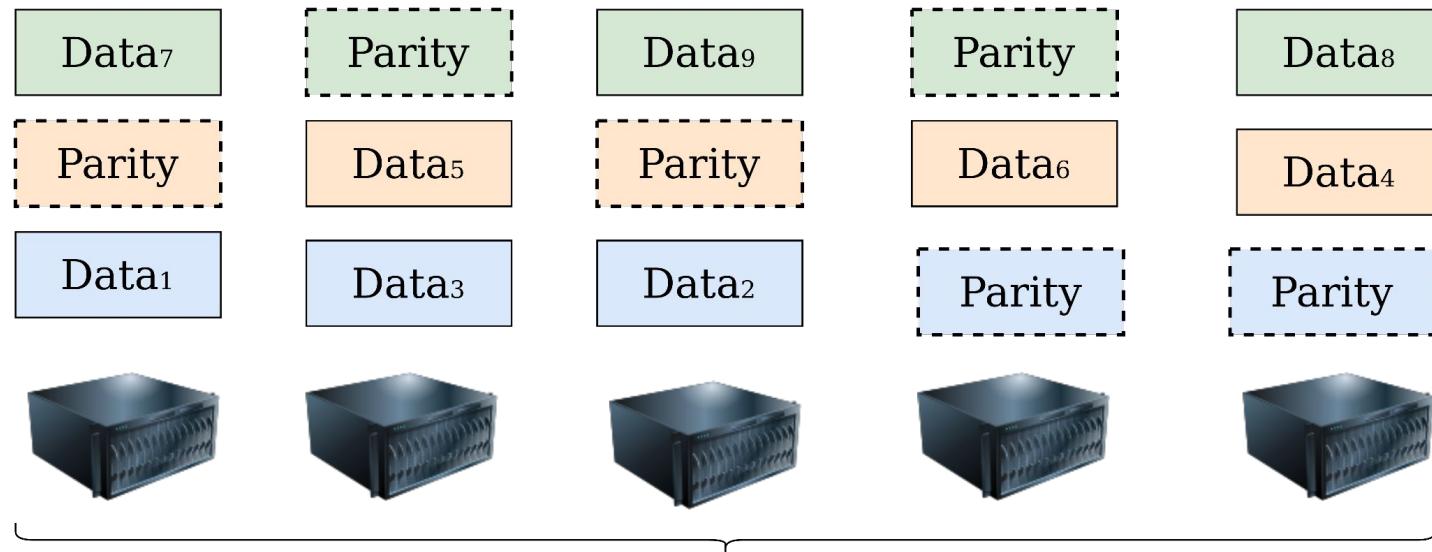
Несбалансированность чтения

- Три сервера запрашиваем при чтении блоков данных
 - Часто
- Остальные два — только при сбоях
 - Редко



Сбалансируем чтения

- Разложим блоки данных и блоки чётности в случайном порядке
 - Вы файловую систему в киосках заряжаете, что ли?



Примерно равное
распределение нагрузки

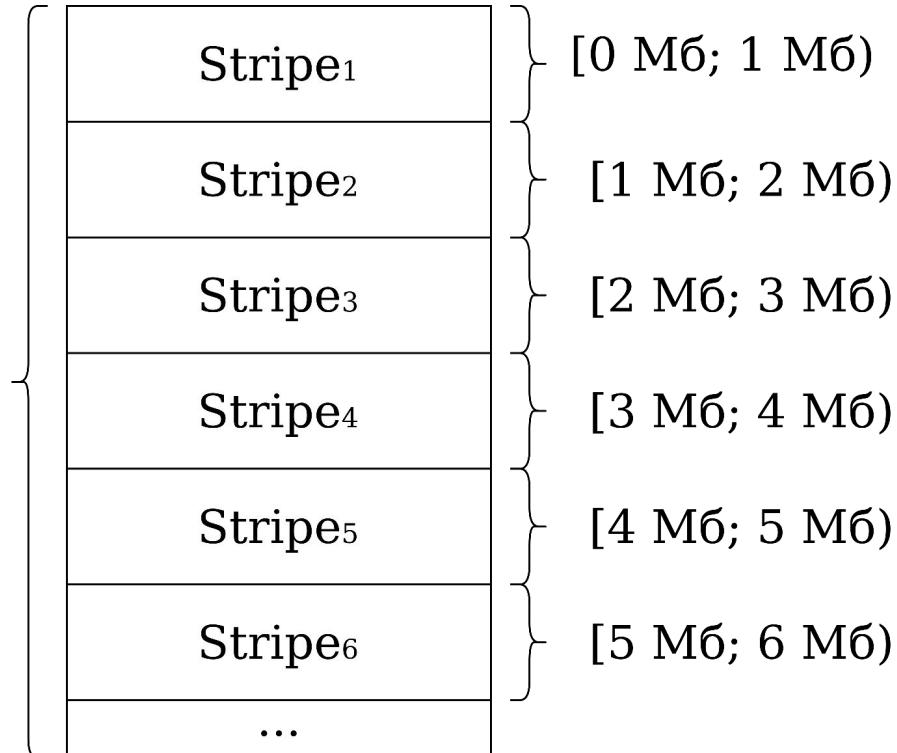
Striped layout

- Разделим каждый блок на полосы

- Так же, как файл разделён на блоки

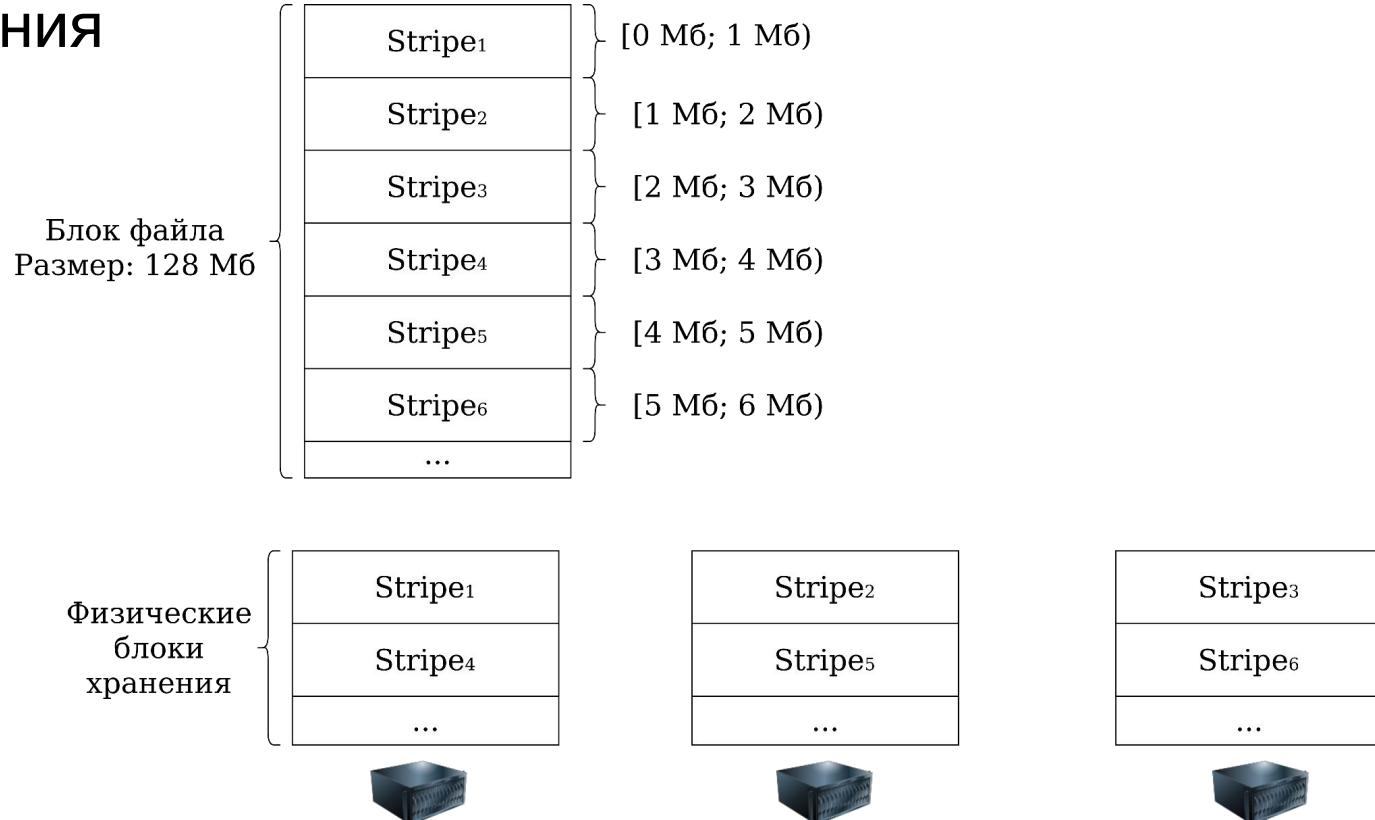
Блок файла
Размер: 128 Мб

- Размер полосы меньше размера блока
 - Например 1 Мб



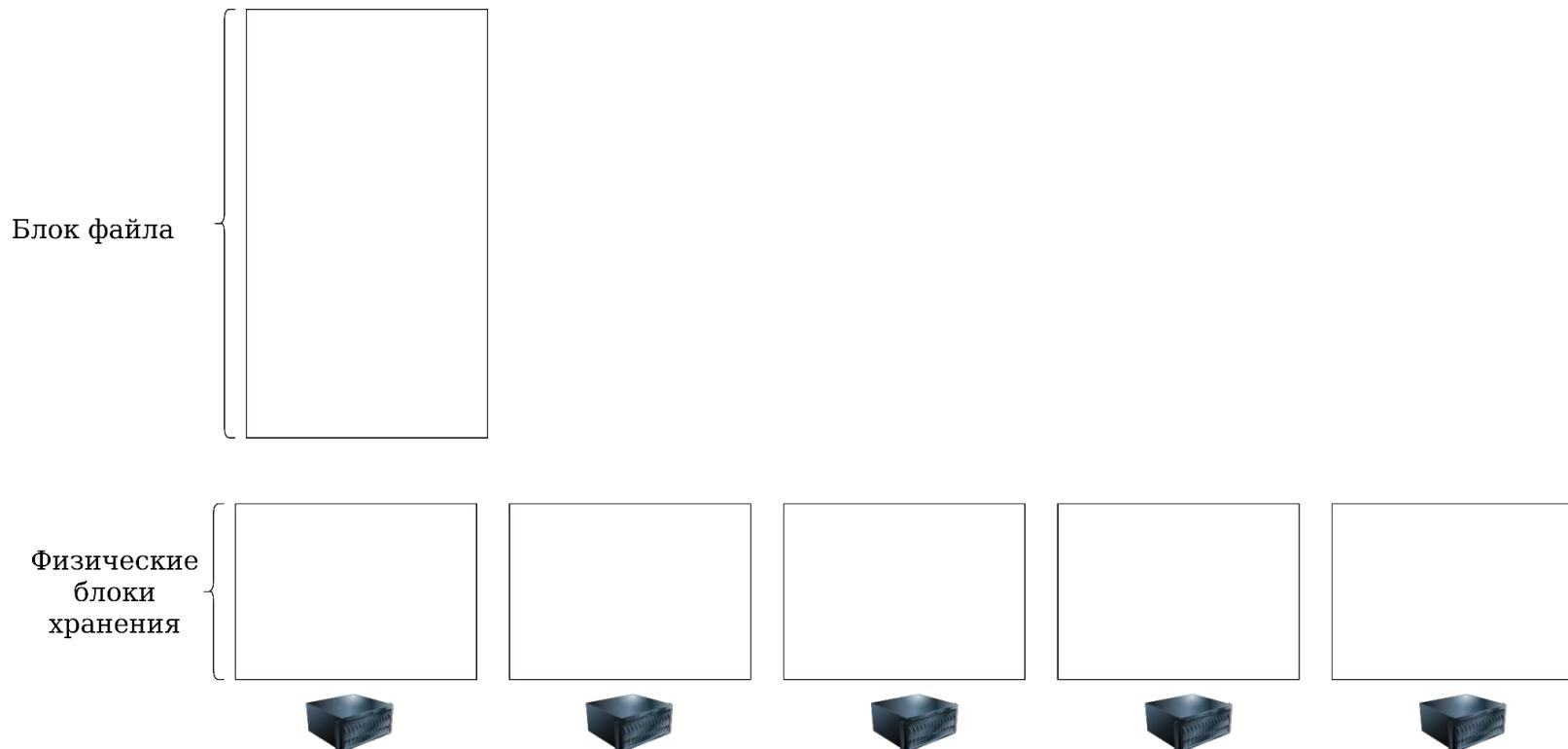
Striped layout: хранение

- Храним логический блок файла в нескольких физических блоках хранения
- Храним по полосам
- Чередуя блоки хранения



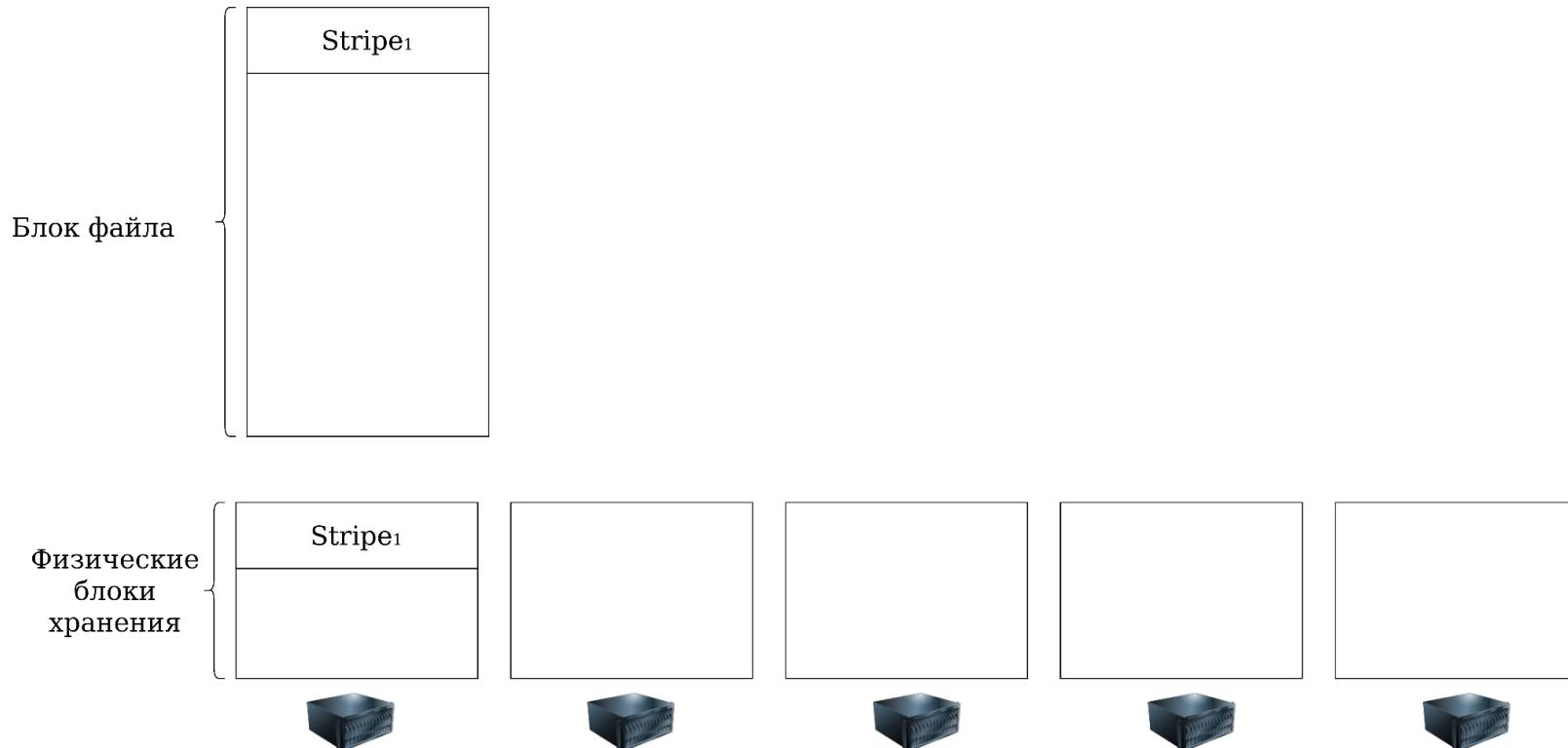
Striped layout: добавление в файл

- Изначально все блоки хранения пустые



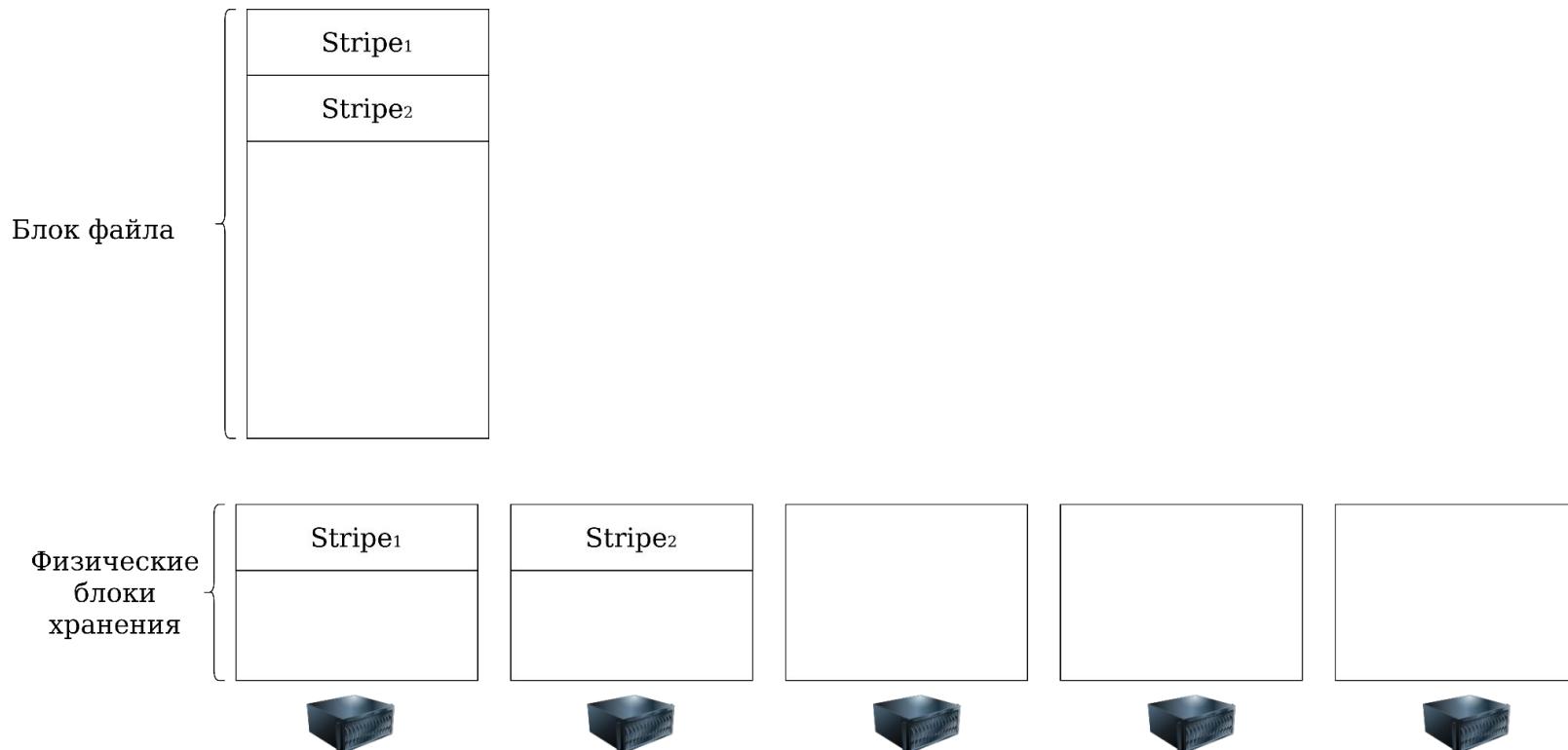
Striped layout: добавление в файл

- Начинаем писать полосы в блоки хранения



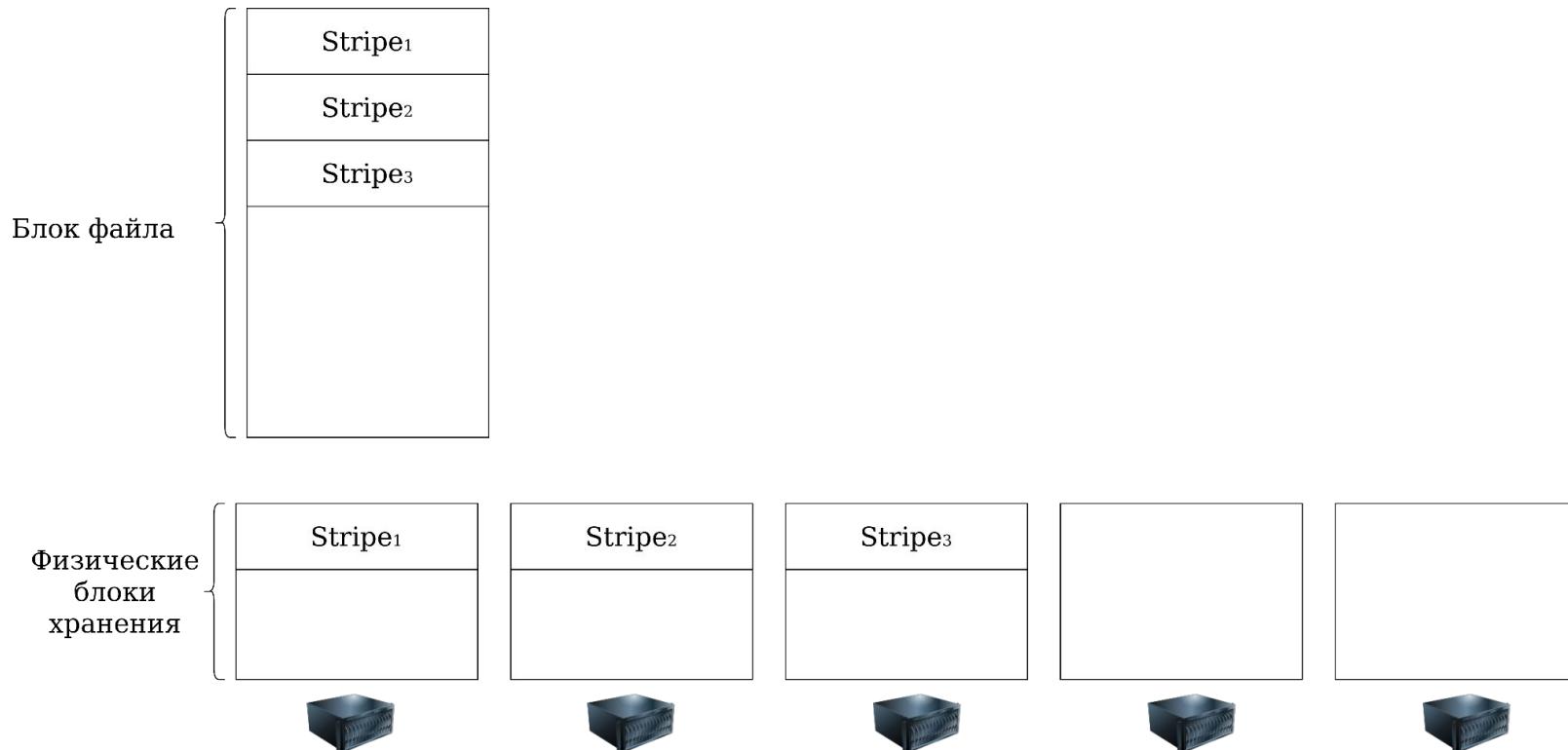
Striped layout: добавление в файл

- Чередуем блоки хранения



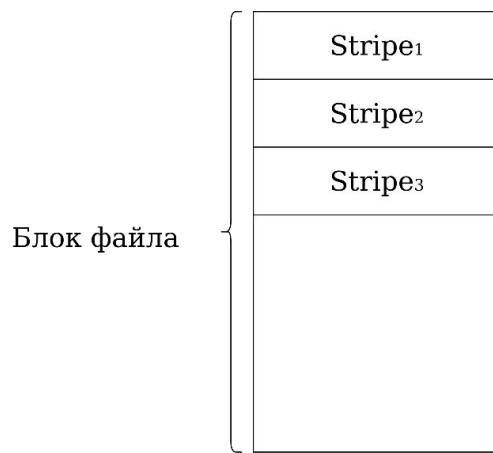
Striped layout: добавление в файл

- Чередуем блоки хранения

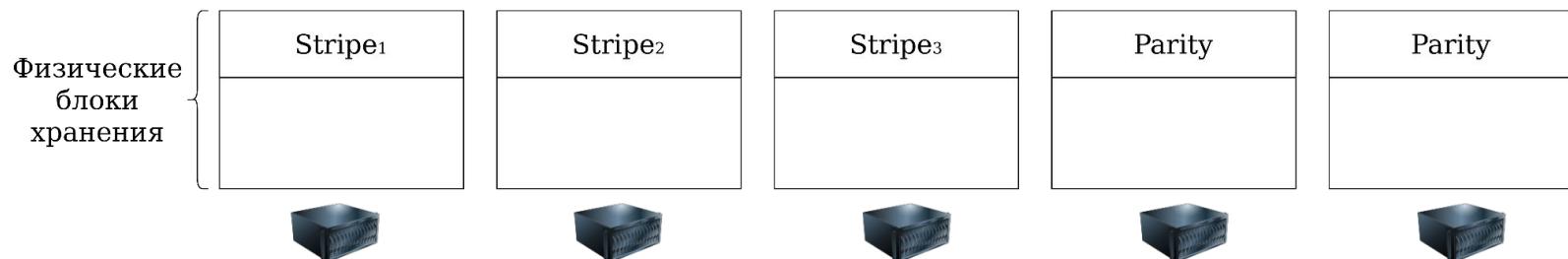


Striped layout: добавление в файл

- У нас есть 3 полосы на разных серверах

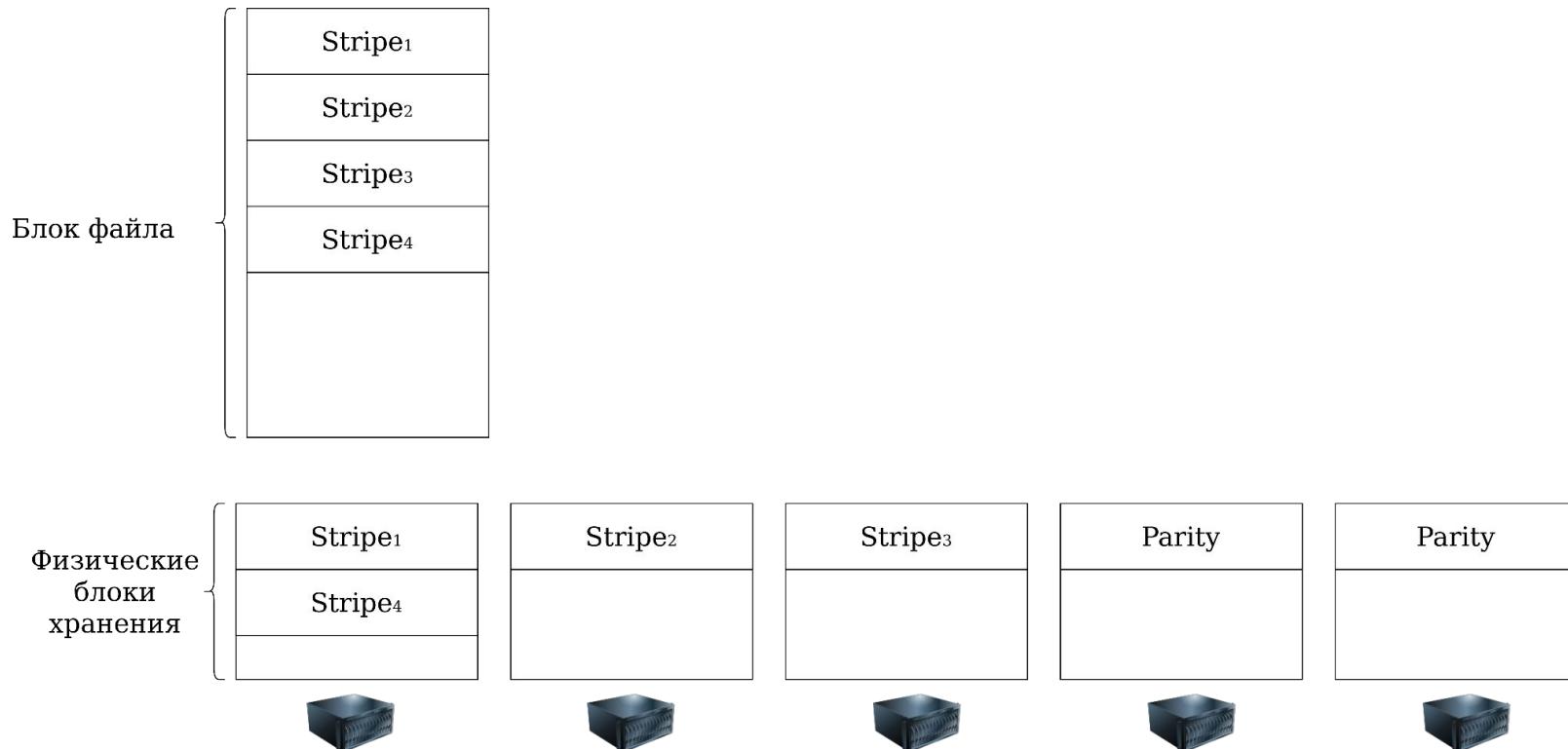


- Можем насчитать 2 полосы чётности



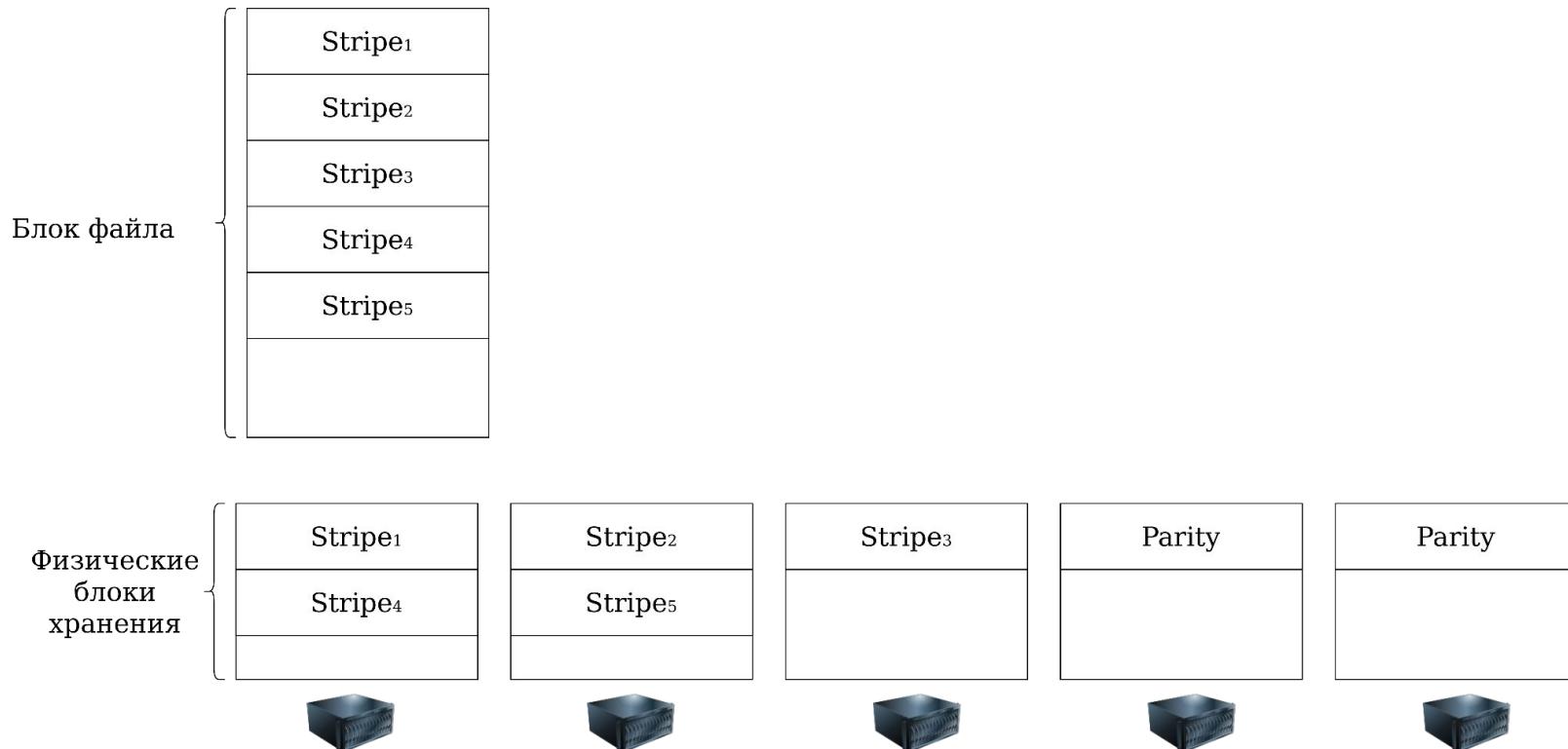
Striped layout: добавление в файл

- Продолжаем запись



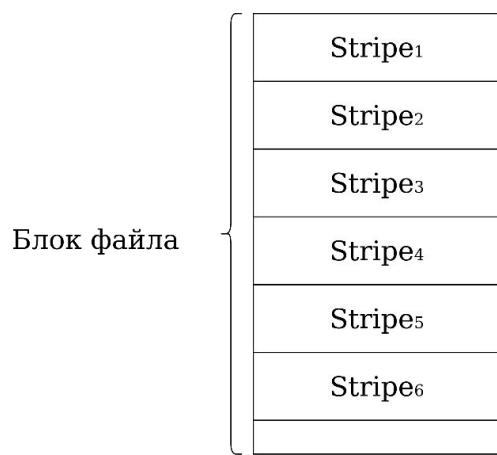
Striped layout: добавление в файл

- Чередуем блоки хранения

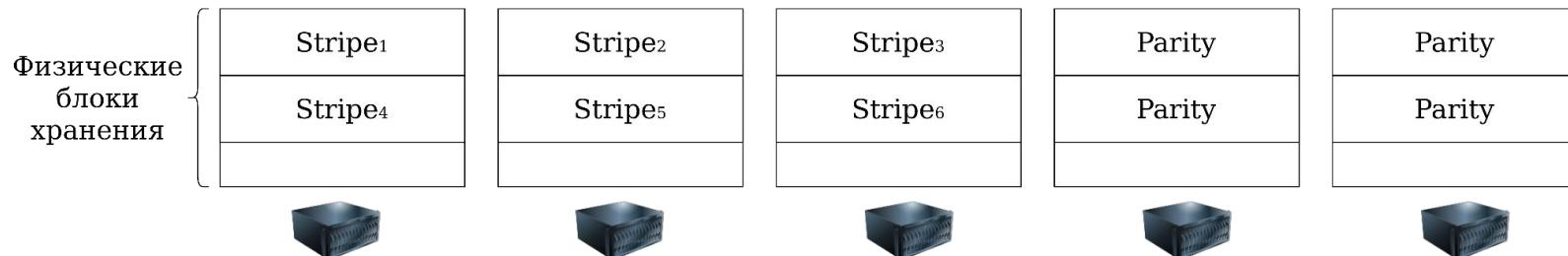


Striped layout: добавление в файл

- Считаем полосы чётности после записи очередных 3 Мб



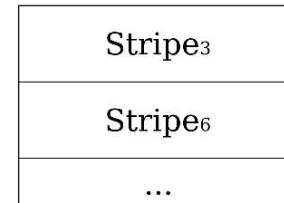
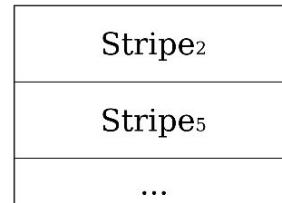
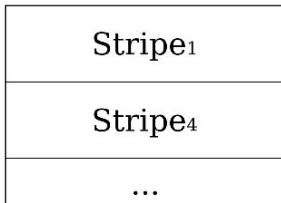
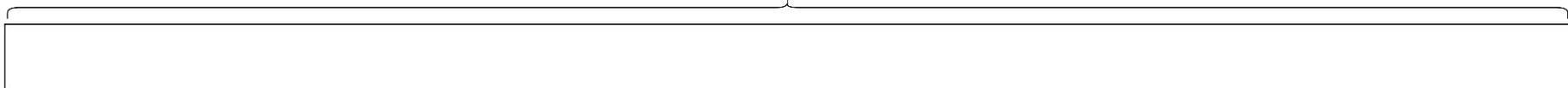
- А не после записи и
репликации трёх блоков по
128 Мб, как раньше



Striped layout: последовательное чтение

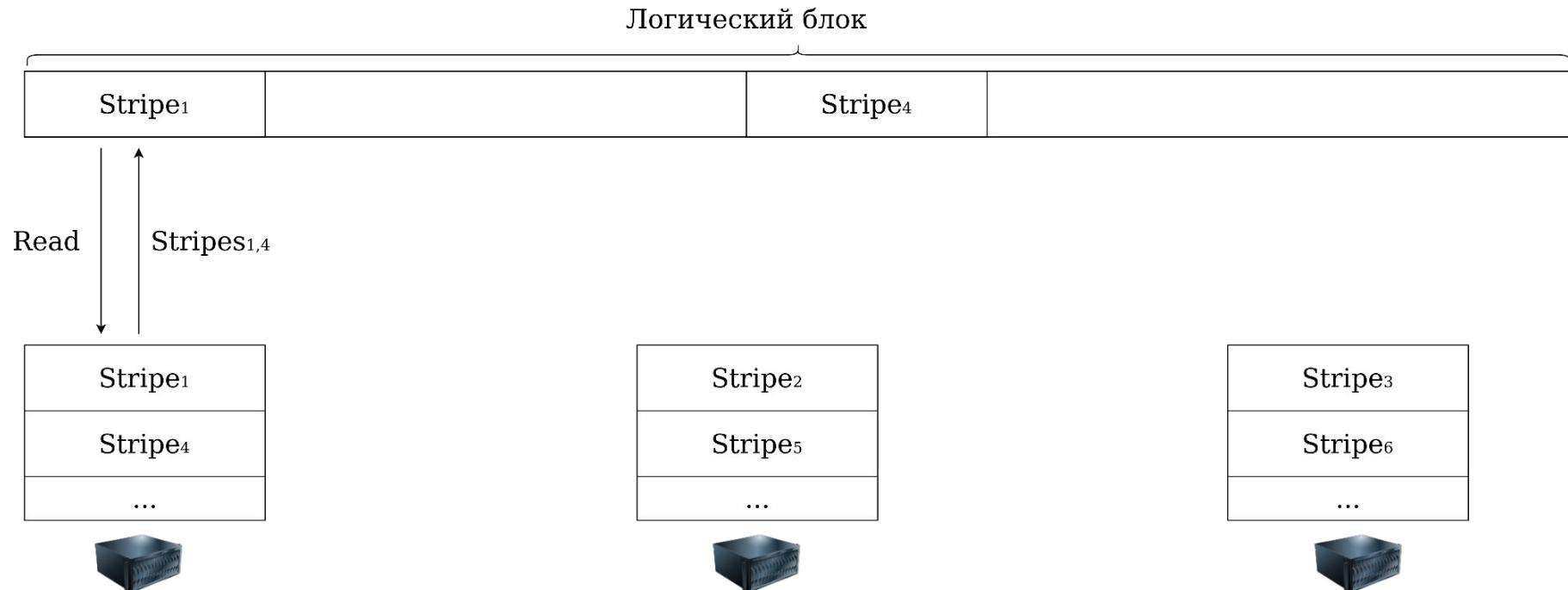
- Чтение усложняется
- Аллоцируем пустой блок

Логический блок



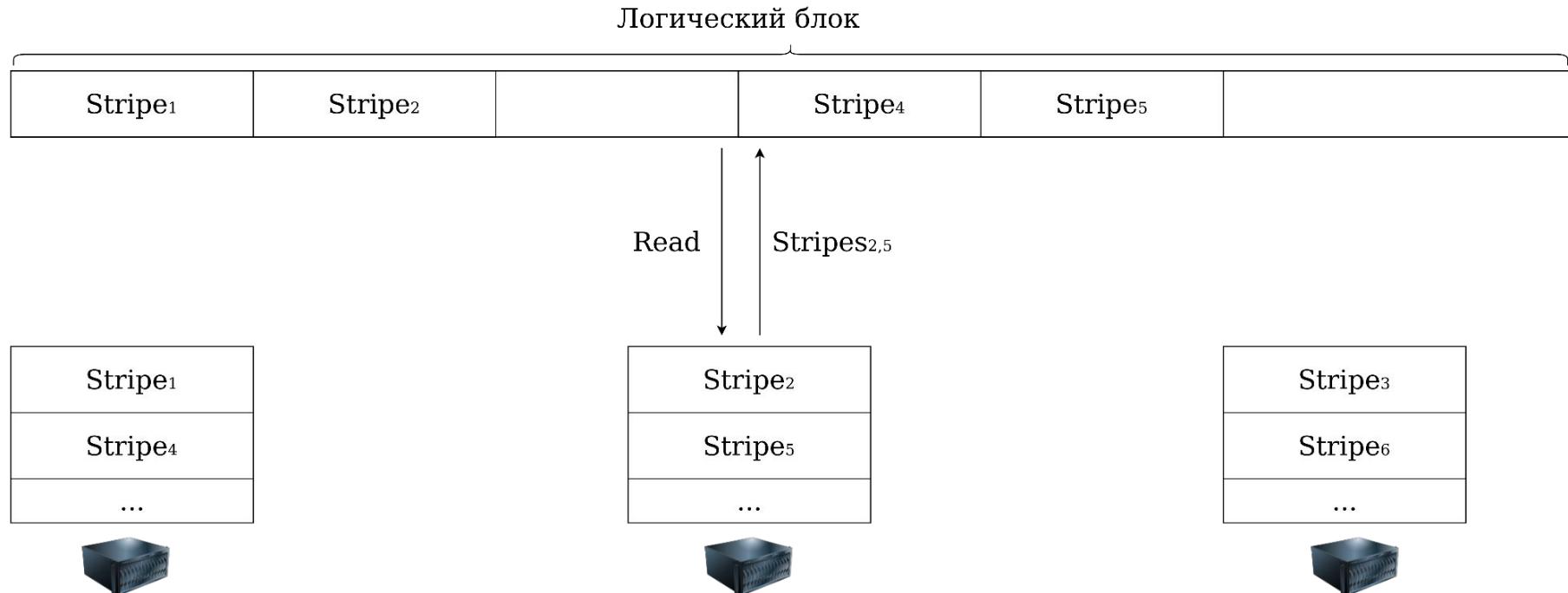
Striped layout: последовательное чтение

- Читаем несколько последовательных полос из первого блока хранения



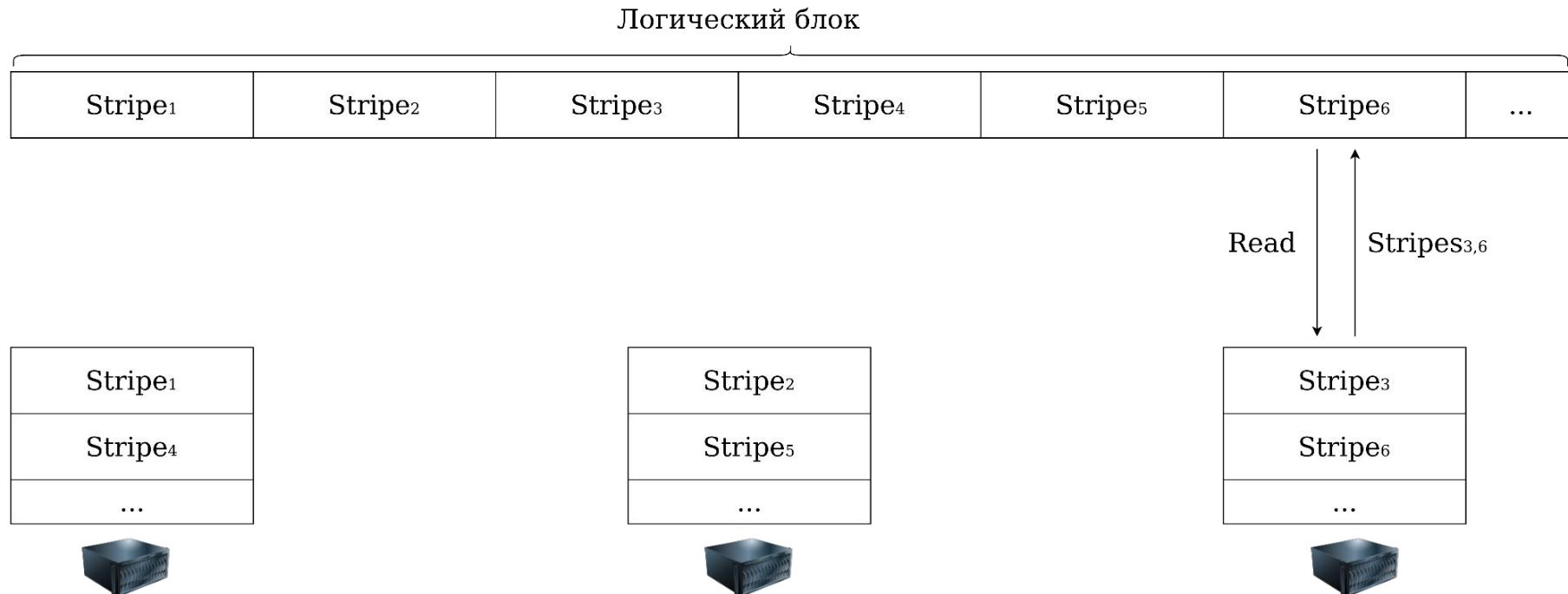
Striped layout: последовательное чтение

- Читаем несколько последовательных полос из второго блока хранения



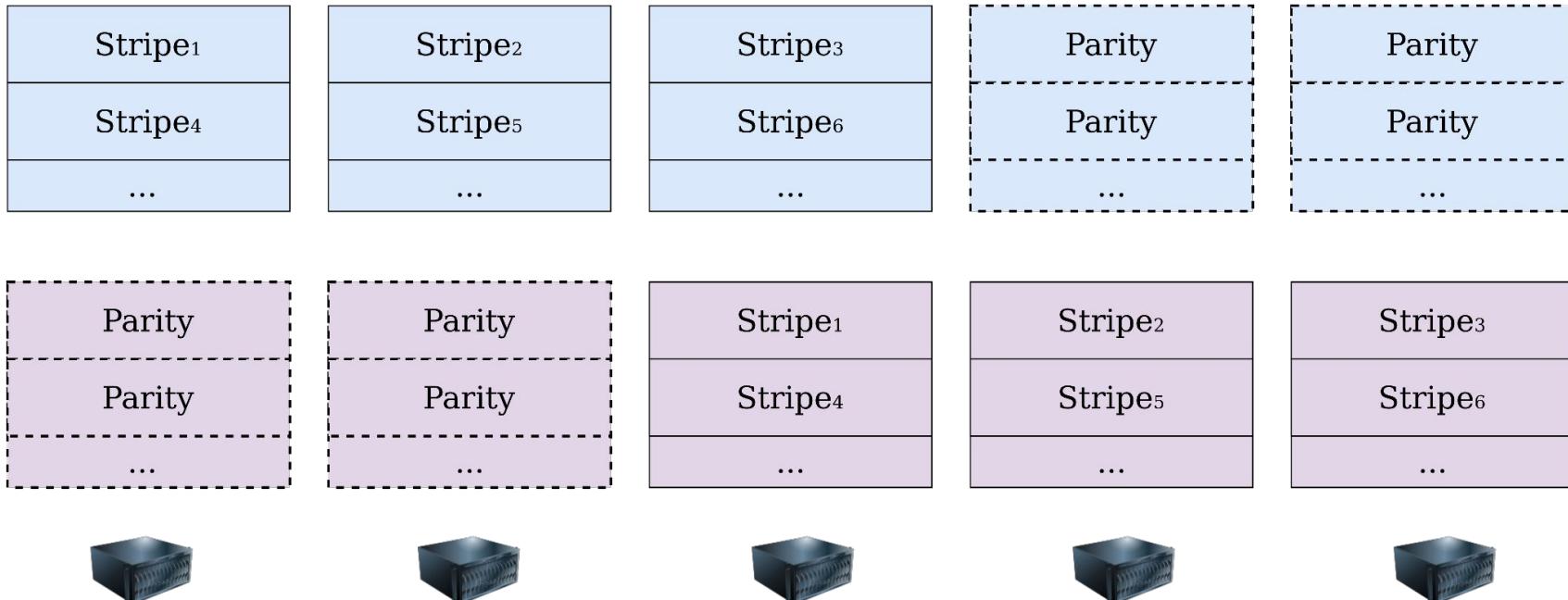
Striped layout: последовательное чтение

- И так далее, пока не заполним блок



Striped layout: сбалансированность чтения

- Каждый логический блок хранит полосы чётности на случайному наборе серверов
 - Чтение достаточно сбалансировано



Что почитать: GFS и HDFS

- *Ghemawat S., Gobioff H., Leung S. T.* The Google File System
- The Architecture of Open Source Applications: HDFS
- HDFS Architecture
- HDFS High Availability Using the Quorum Journal Manager
- HDFS Federation
- Append / HFlush / Read Design
- *White T.* Hadoop: The definitive guide.

Что почитать & посмотреть: помехоустойчивое кодирование

- Introduction to HDFS Erasure Coding in Apache Hadoop
- Денис Ефаров. Hadoop 3: Erasure coding catastrophe
- Кудряшов Б. Д. Основы теории кодирования
- Александр Христофоров. Дешевле, надёжнее, проще: хранение петабайтов видео и фото в Одноклассниках
- Huang C. et al. Erasure coding in windows azure storage
- Как применение кодов избыточности в SDS помогает Яндексу дёшево и надёжно хранить данные
- Maxim Babenko. Erasure Coding at Scale

Что почитать: другие распределённые ФС

- Network File System
- Coda File System
 - *Satyanarayanan M. et al.* Coda: A highly available file system for a distributed workstation environment
 - *Kistler J. J., Satyanarayanan M.* Disconnected operation in the Coda file system
- Сепх
 - *Weil S. A. et al.* Ceph: A scalable, high-performance distributed file system
 - *Weil S. A.* Ceph: reliable, scalable, and high-performance distributed storage

Что почитать & посмотреть: объектные хранилища

- [What is Amazon S3](#)
- [Вадим Цесько. ОК S3: Строим Систему Сам](#)

Thanks for your attention

