

Базы данных

Распределённые транзакции



Илья Кокорин

kokorin.ilya.1998@gmail.com

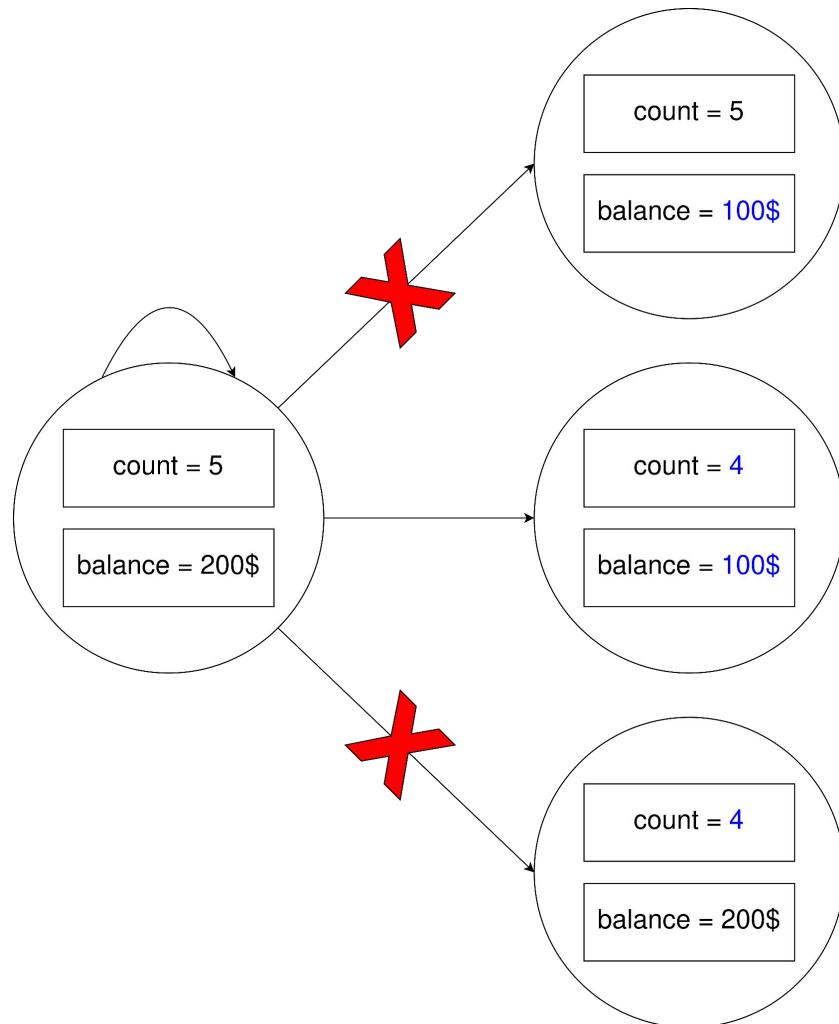
Транзакция

- Единица работы над данными в базе данных
- Списываем деньги со счёта и уменьшаем количество товара на складе

```
1 transaction buy(good_id, user_id):  
2     price := get_price(good_id)  
3     balance := load_balance(user_id)  
4     if balance < price:  
5         rollback  
6     store_balance(balance - price)  
7     count := get_count(good_id)  
8     if count = 0:  
9         rollback  
10    set_count(count - 1)  
11    commit
```

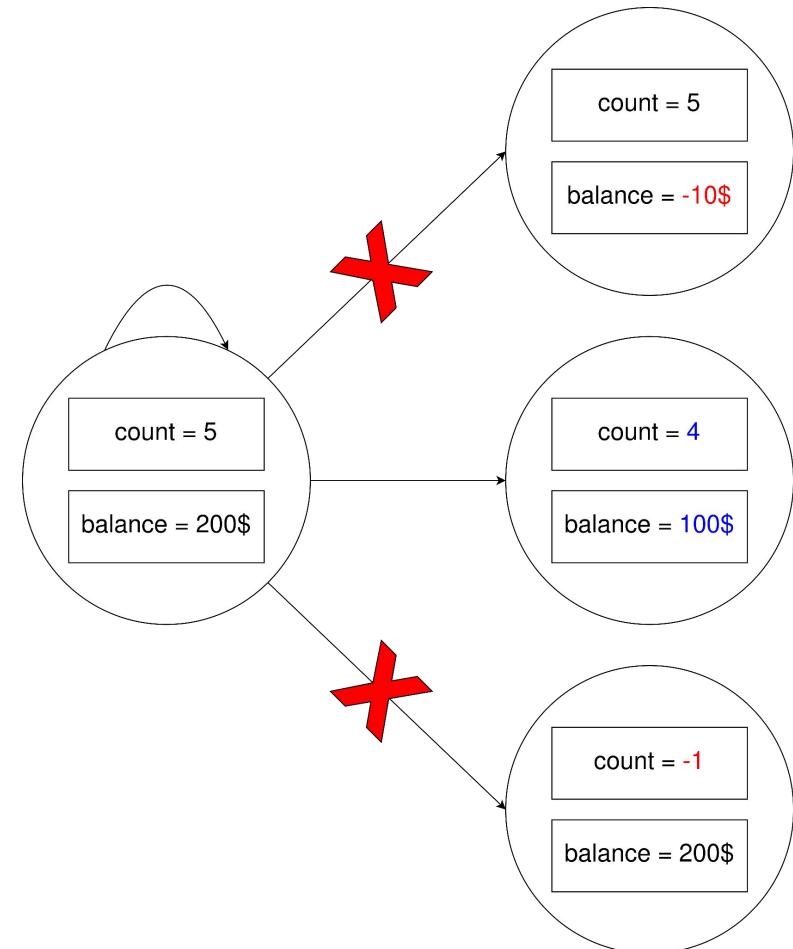
Транзакция: атомарность

- Применяются либо все изменения, либо никакие
- ACID



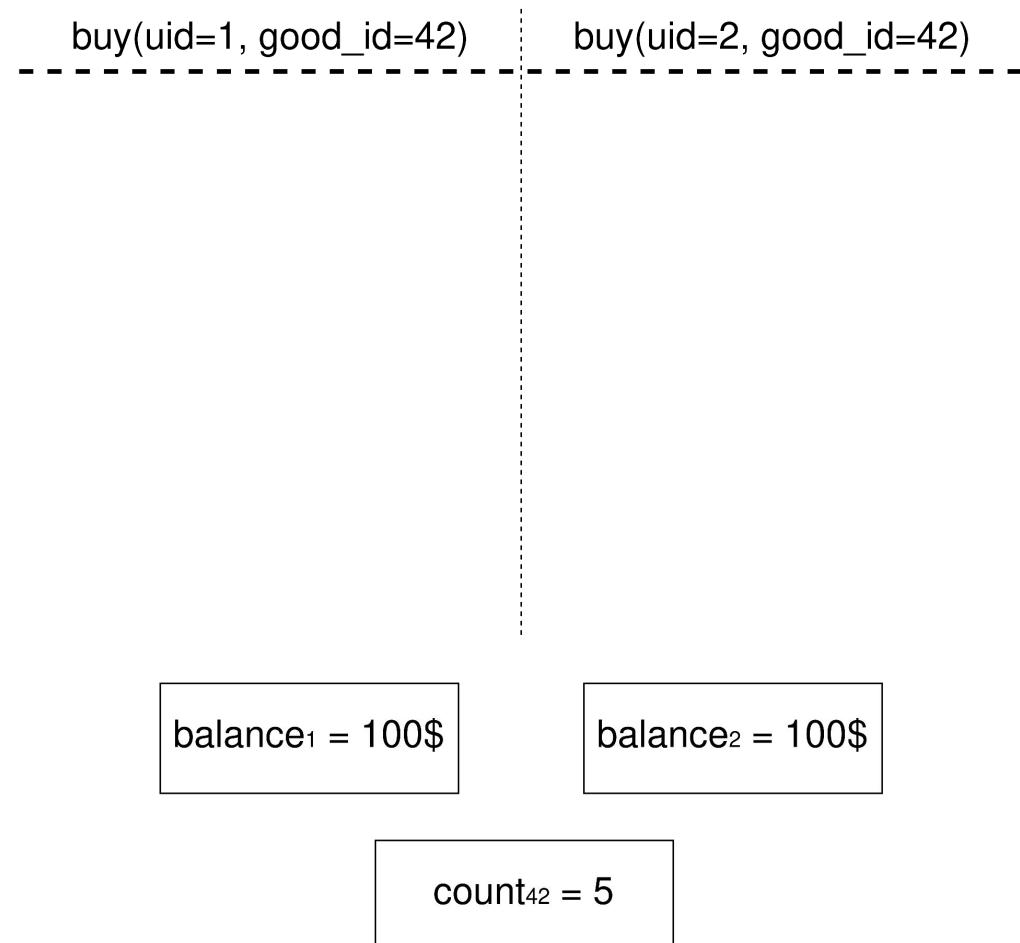
Транзакция: согласованность

- Либо переводит базу данных в согласованное состояние, либо откатывается целиком
- Все инварианты выполнены после завершения транзакции
- ACID



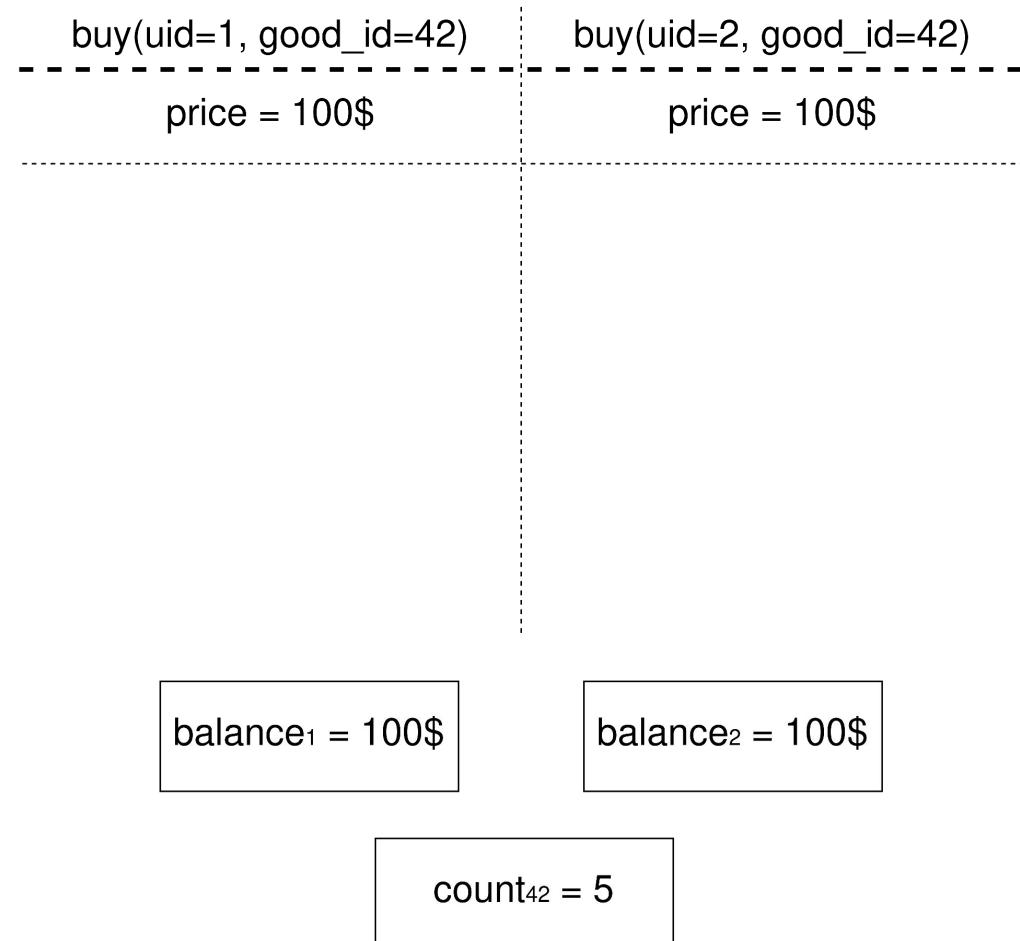
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



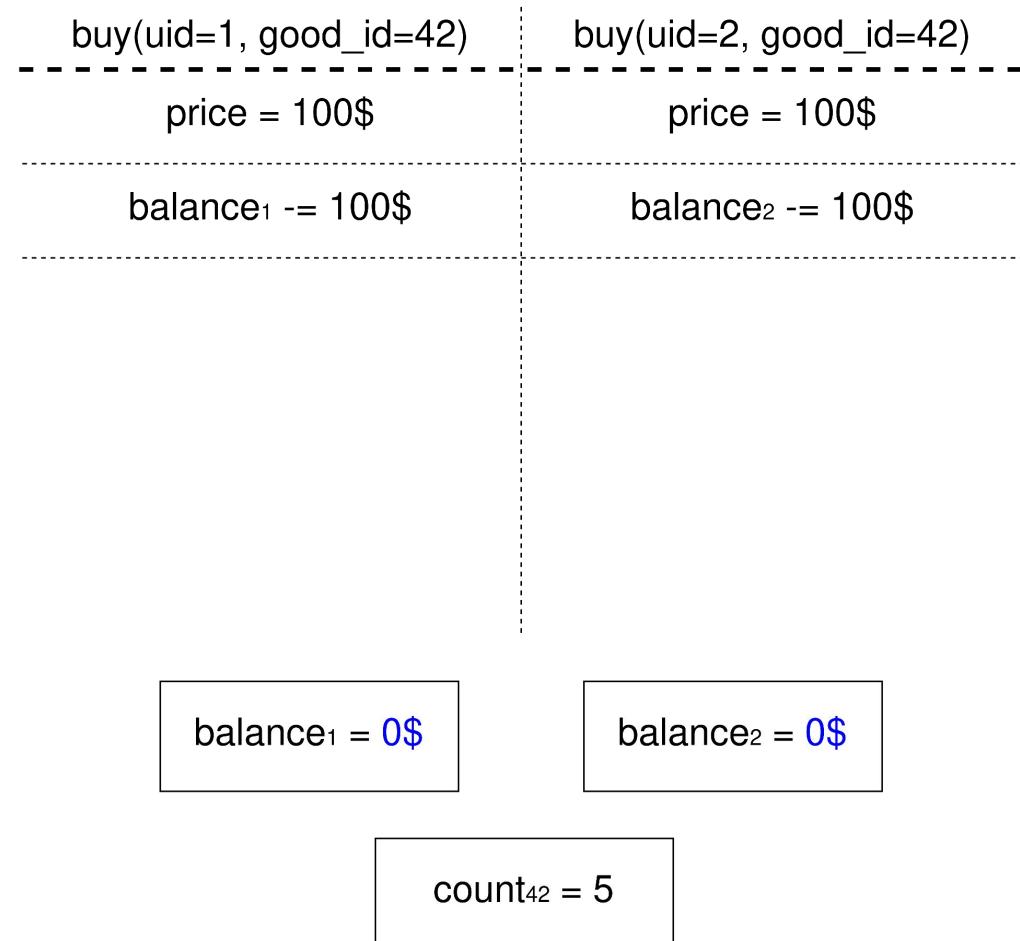
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



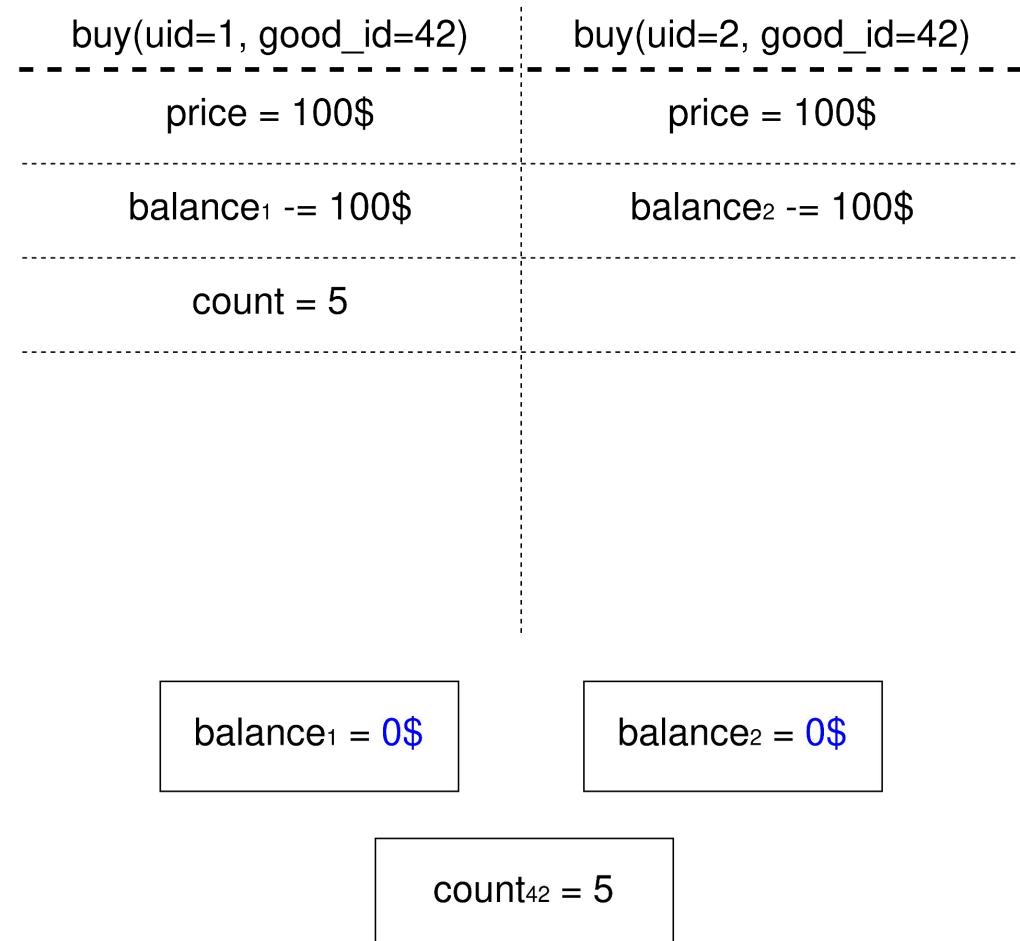
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



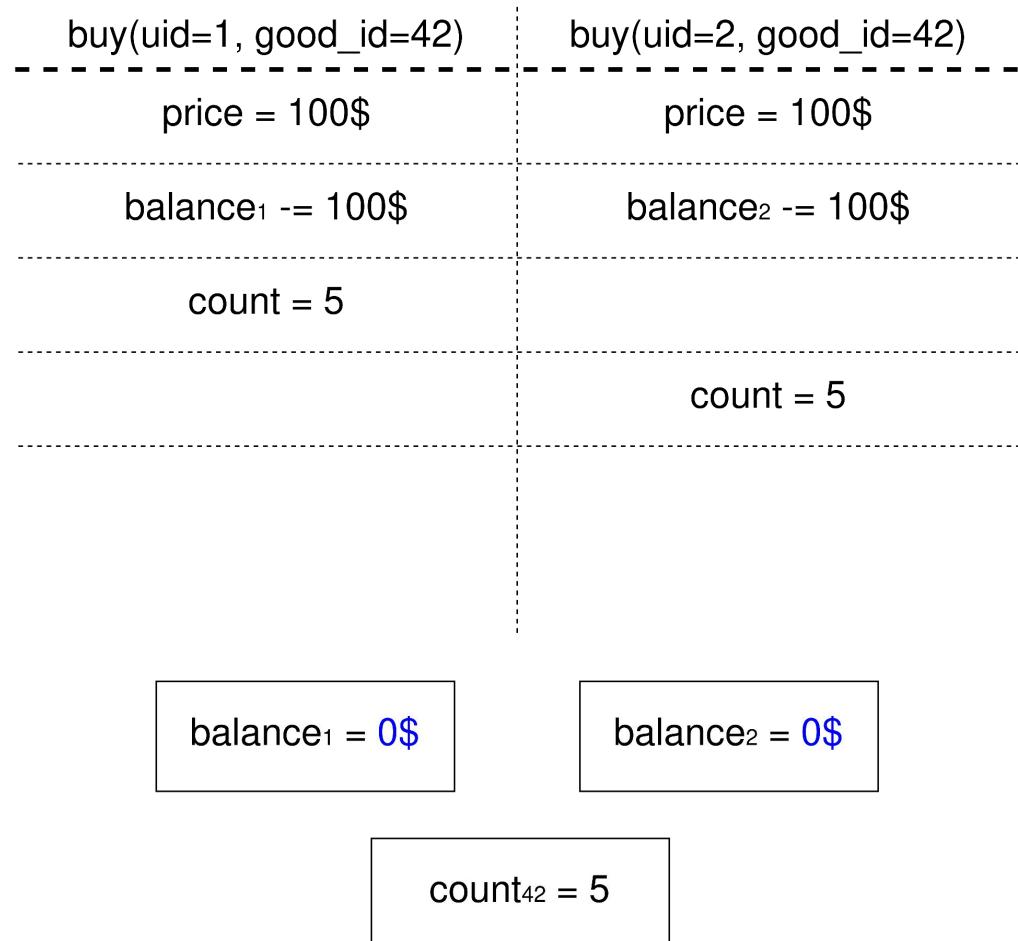
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



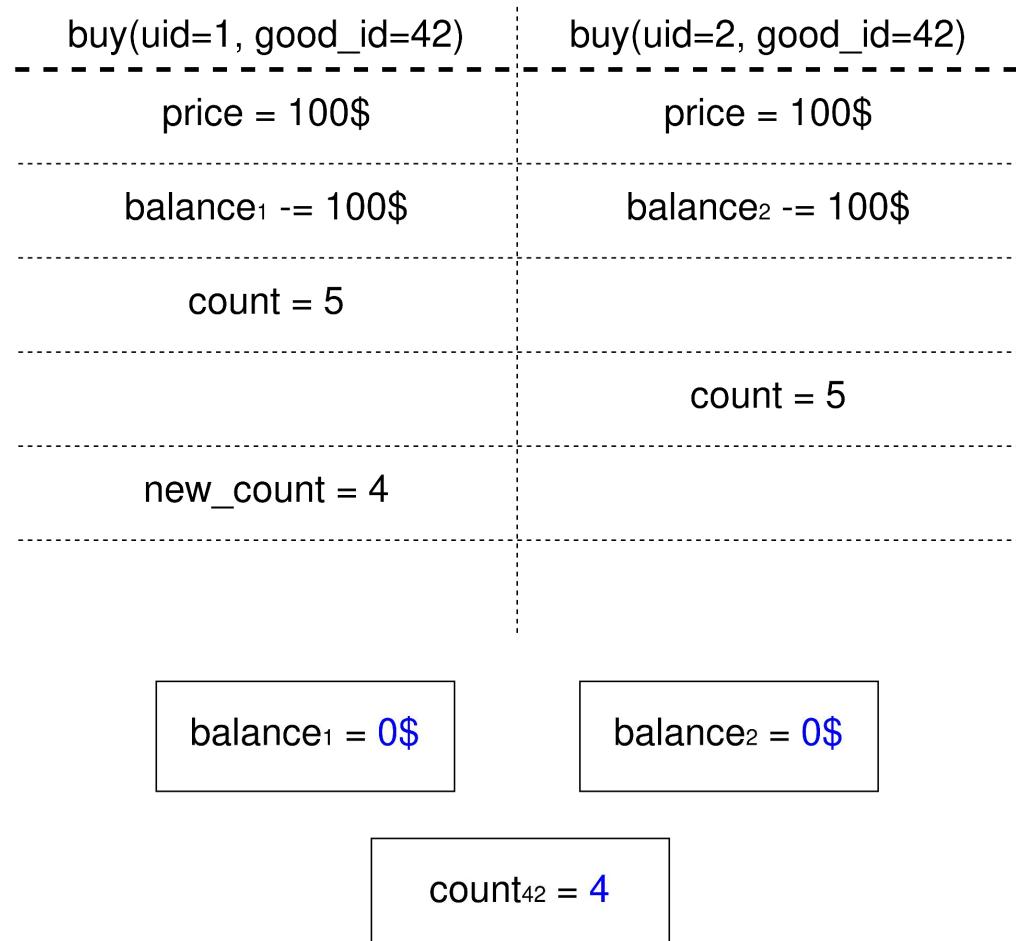
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



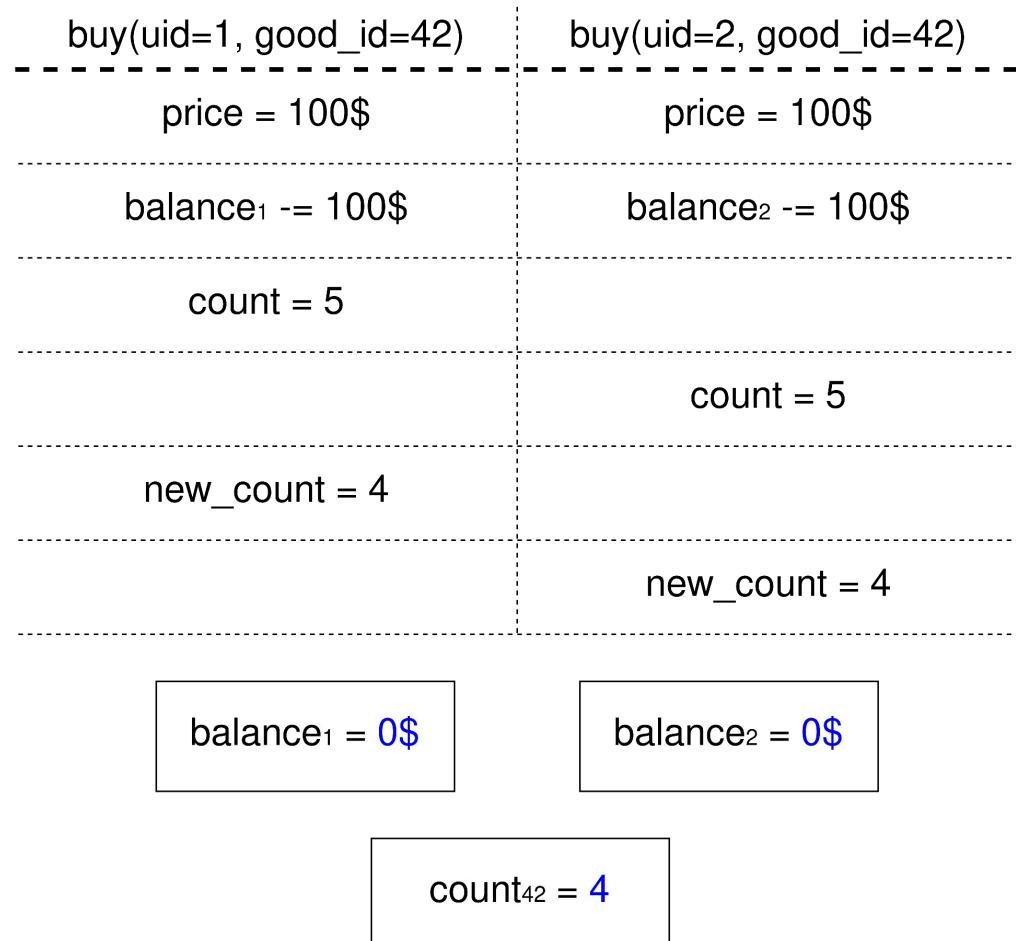
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



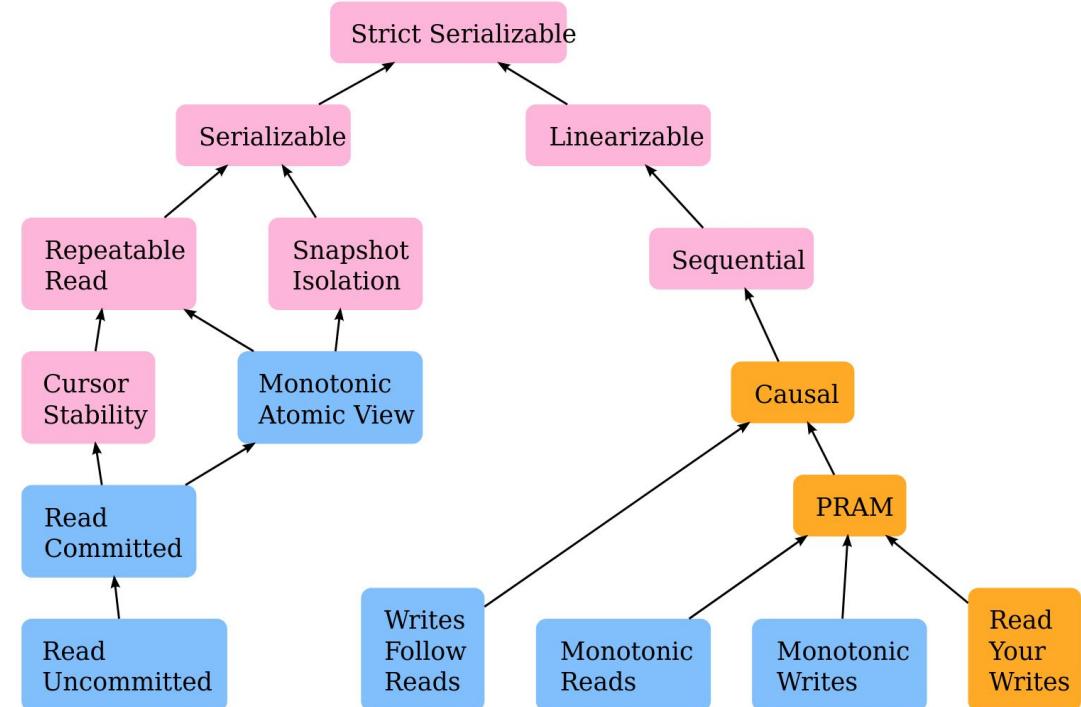
Транзакция: изоляция

- Транзакция исполняется так, будто она в системе единственная
- Нет спецэффектов от параллелизма
- ACID



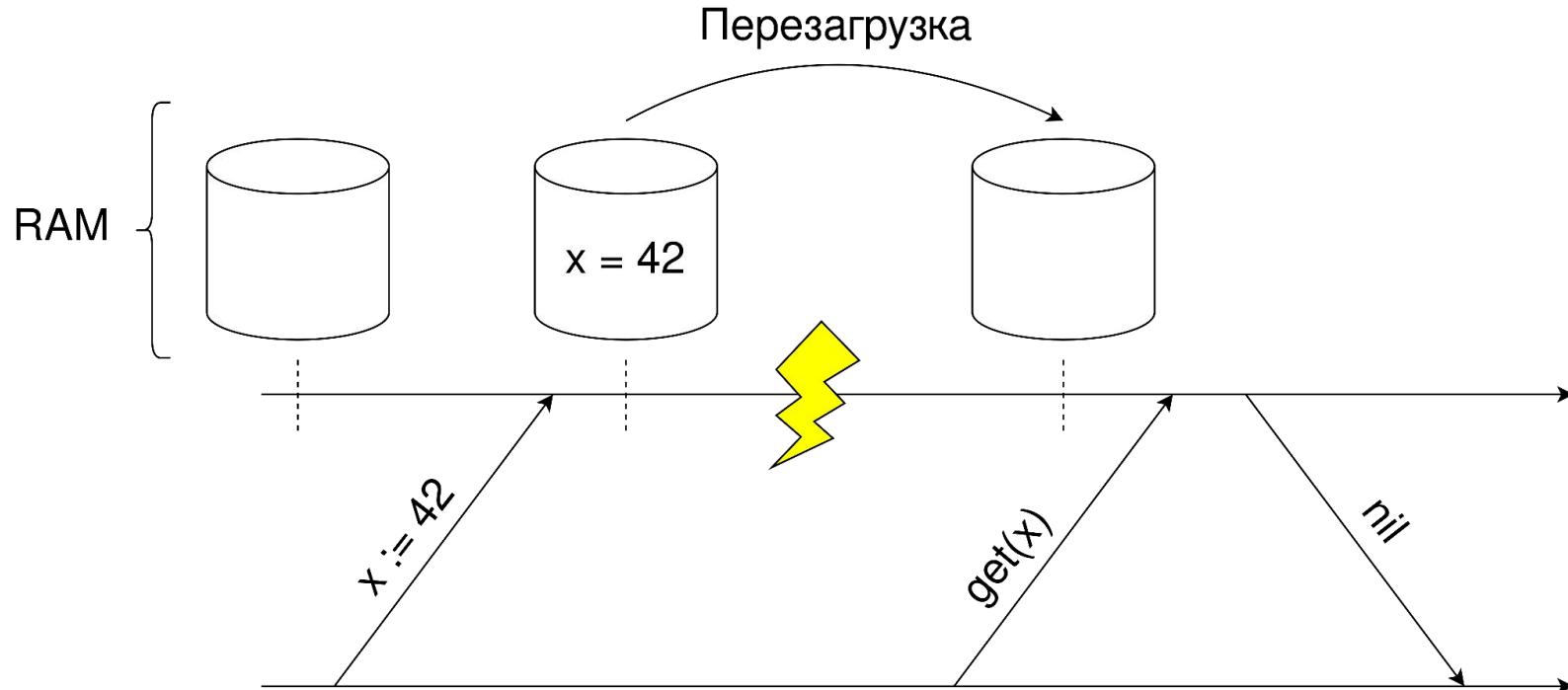
Транзакция: уровни изоляции

- Read Uncommitted
- Read Committed
- Repeatable Read
- *Snapshot Isolation*
- Serializable
- ...



Транзакция: устойчивость

- Результаты успешной транзакции не могут “потеряться”
- Данные должны сохраняться надёжно
- ACID



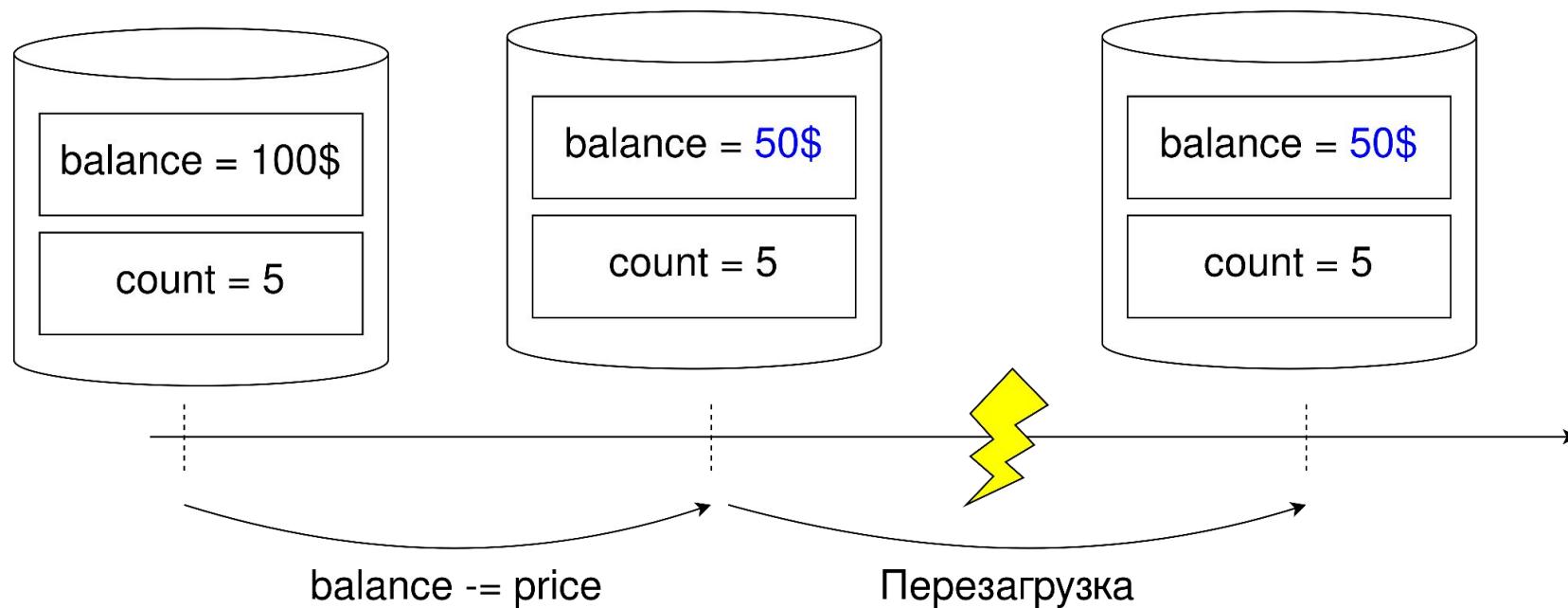
Атомарность через компенсирующие действия

- Если нужно откатить транзакцию, применяем компенсирующие действия
- Отменяем все изменения

```
1 fun buy(good_id, user_id):  
2     price := get_price(good_id)  
3     balance := load_balance(user_id)  
4     if balance < price:  
5         return  
6     store_balance(balance - price)  
7     count := get_count(good_id)  
8     if count = 0:  
9         # restore initial balance  
10        store_balance(balance)  
11        return  
12        set_count(count - 1)
```

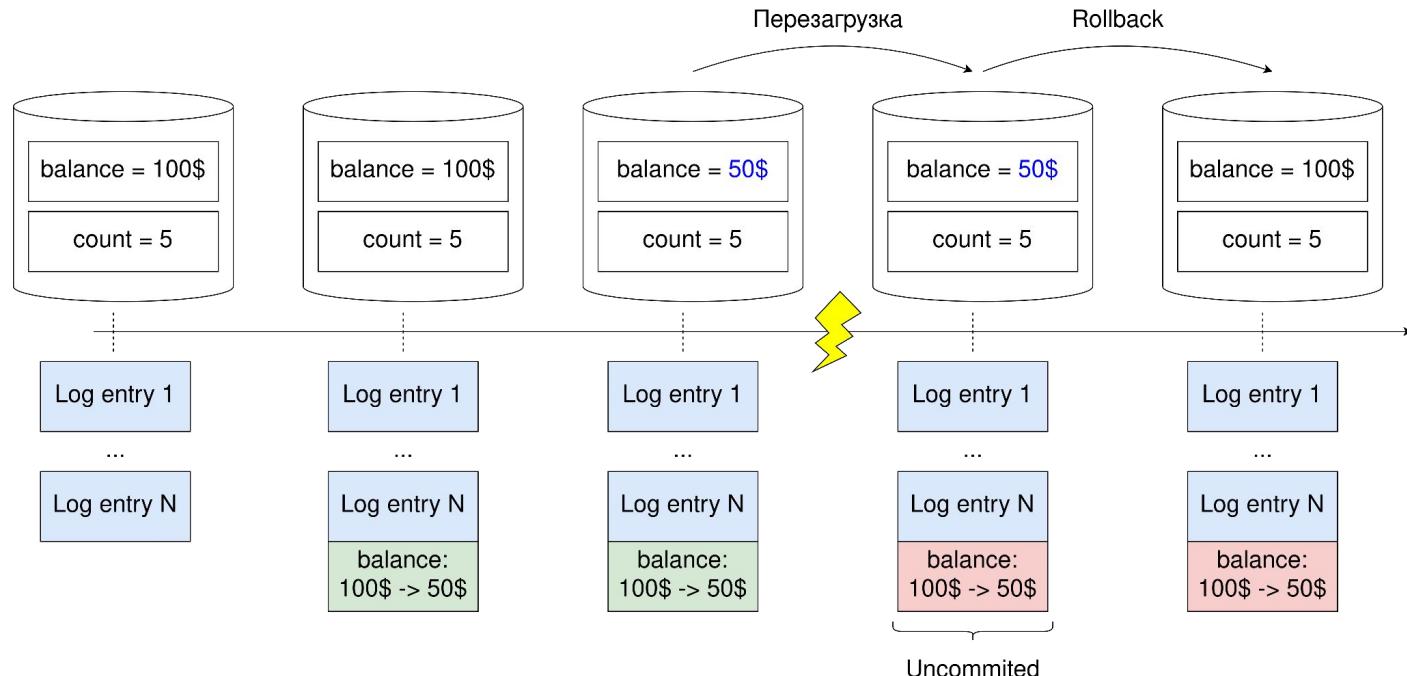
Атомарность через компенсирующие действия

- Не работает из-за аппаратных сбоев
- Регистр RIP перезагружается после сбоя
- Стек программы тоже



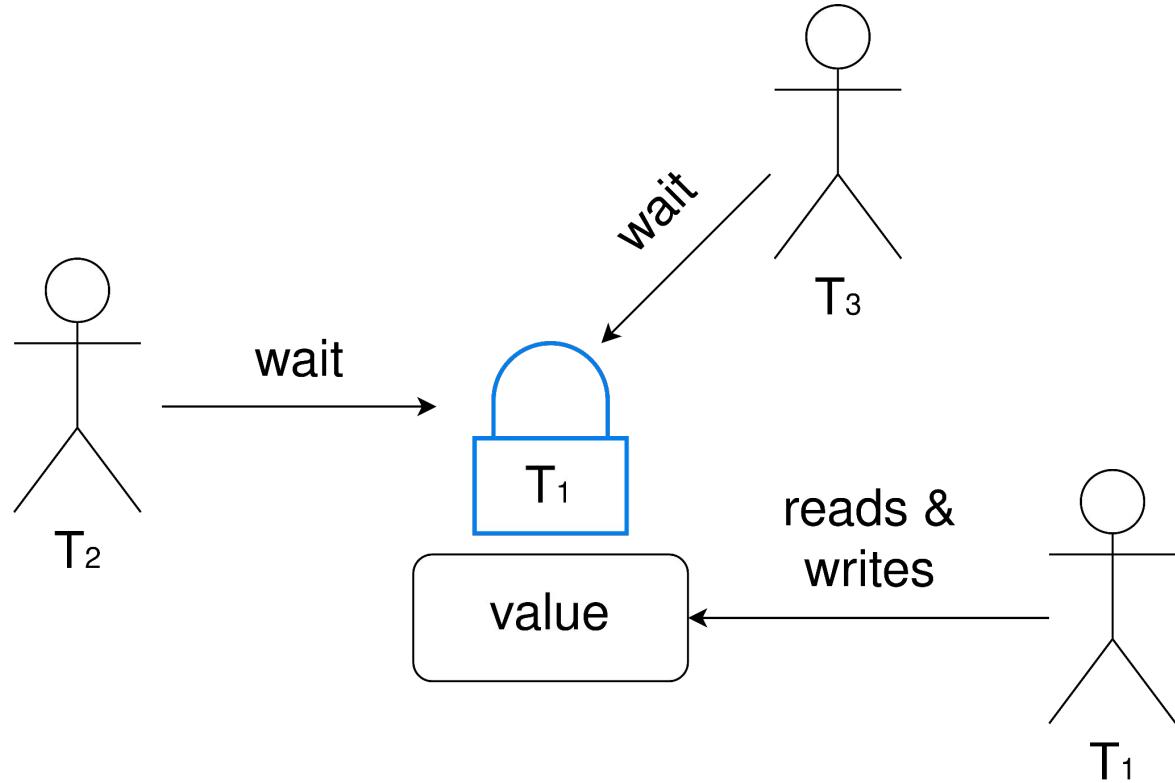
Атомарность через Write-Ahead Log

- Каждое изменение пишем сначала в журнал
- После сбоя откатываем изменения, сделанные незаконченными транзакциями



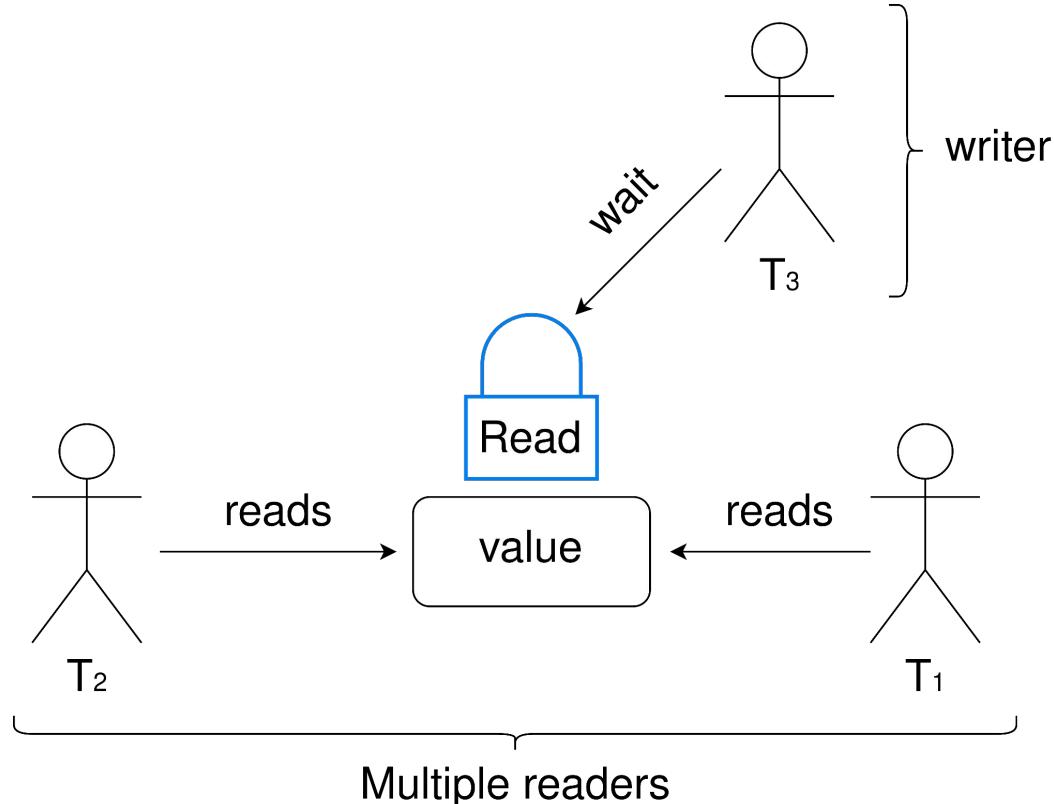
Изоляция: блокировки

- Каждое значение защищено блокировкой
- Читать и писать эту переменную может только транзакция, держащая блокировку
- Остальные транзакции вынуждены ждать освобождения блокировки



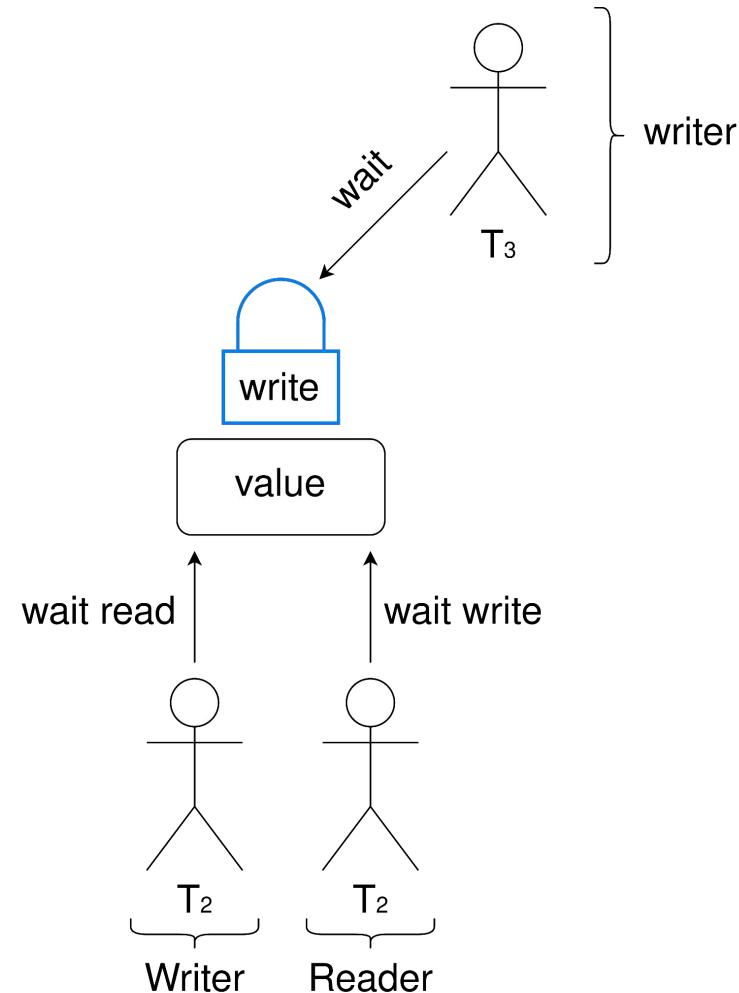
Изоляция: блокировки

- Значение могут читать несколько транзакций под одной и той же блокировкой
- Писать не может НИКТО
- Писатели вынуждены ждать освобождения блокировки
- Пока все читатели закончат читать



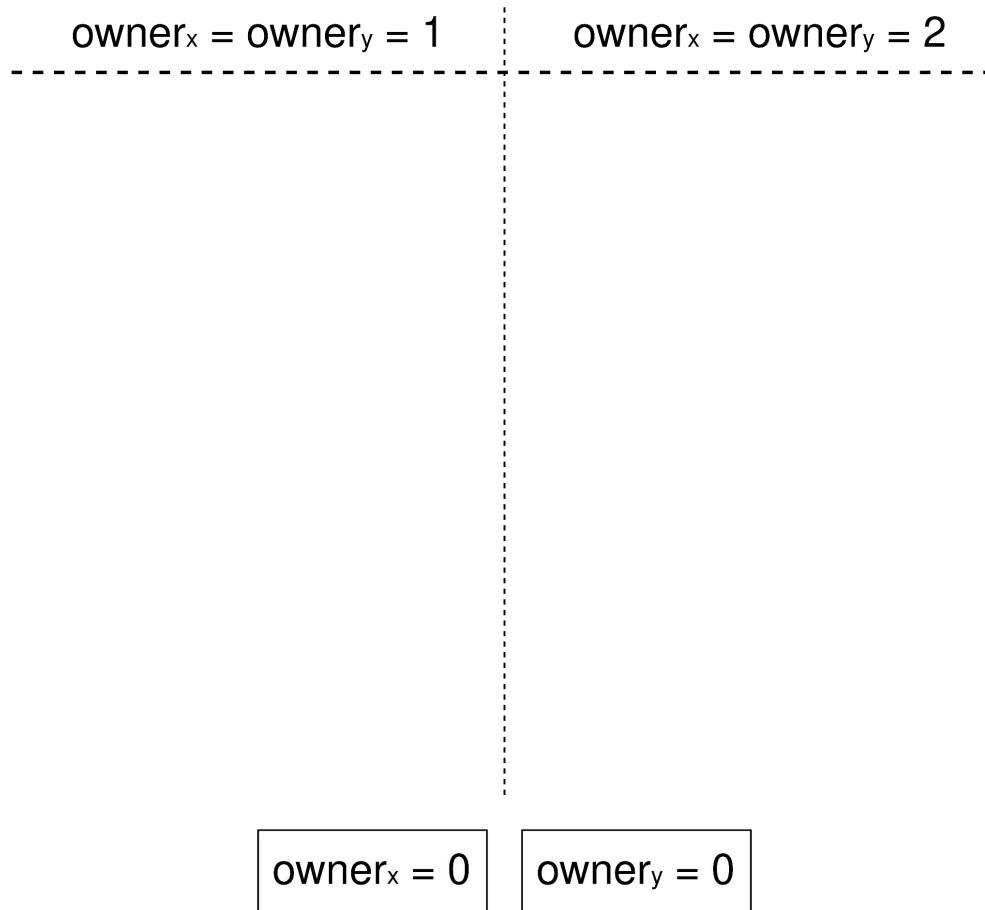
Изоляция: блокировки

- Если хотя бы одна транзакция изменяет значение, другие не могут ни читать, ни писать
- Вынуждены ждать



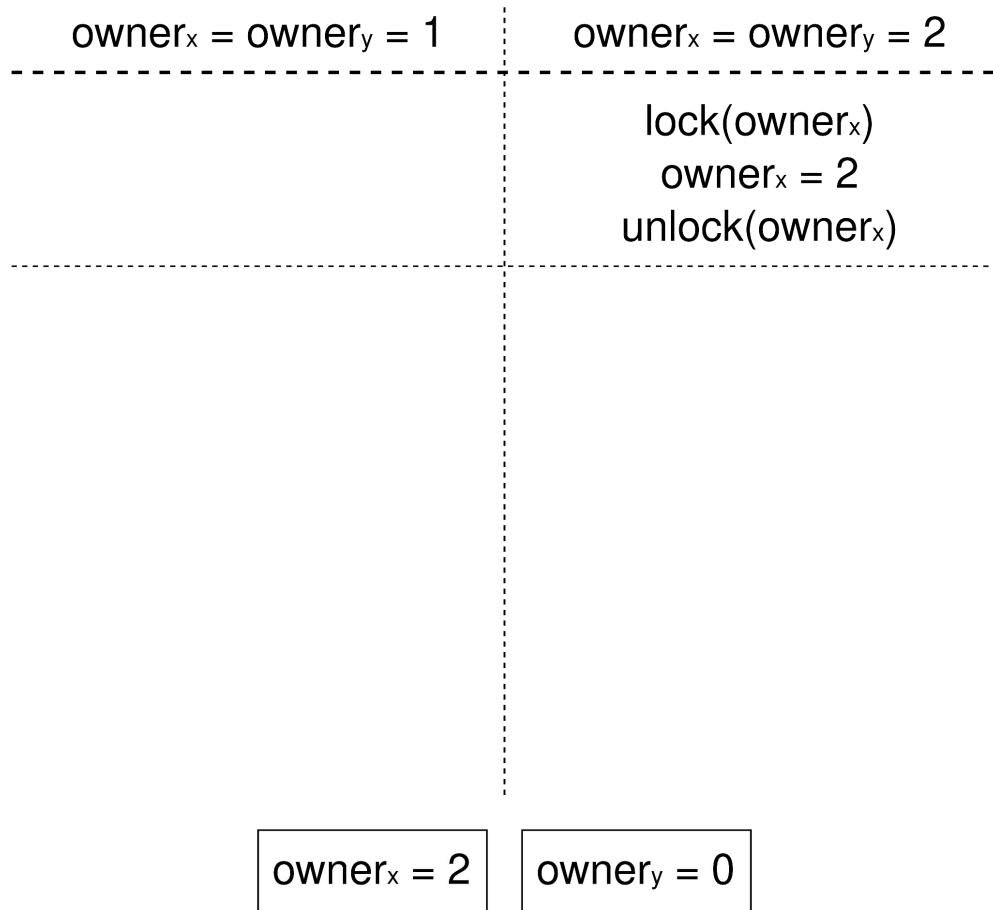
Изоляция: блокировки

- Можно ли отпустить блокировку сразу после чтения/записи?
- Два пользователя хотят забронировать места x и y
- Либо оба места сразу, либо ни одного места



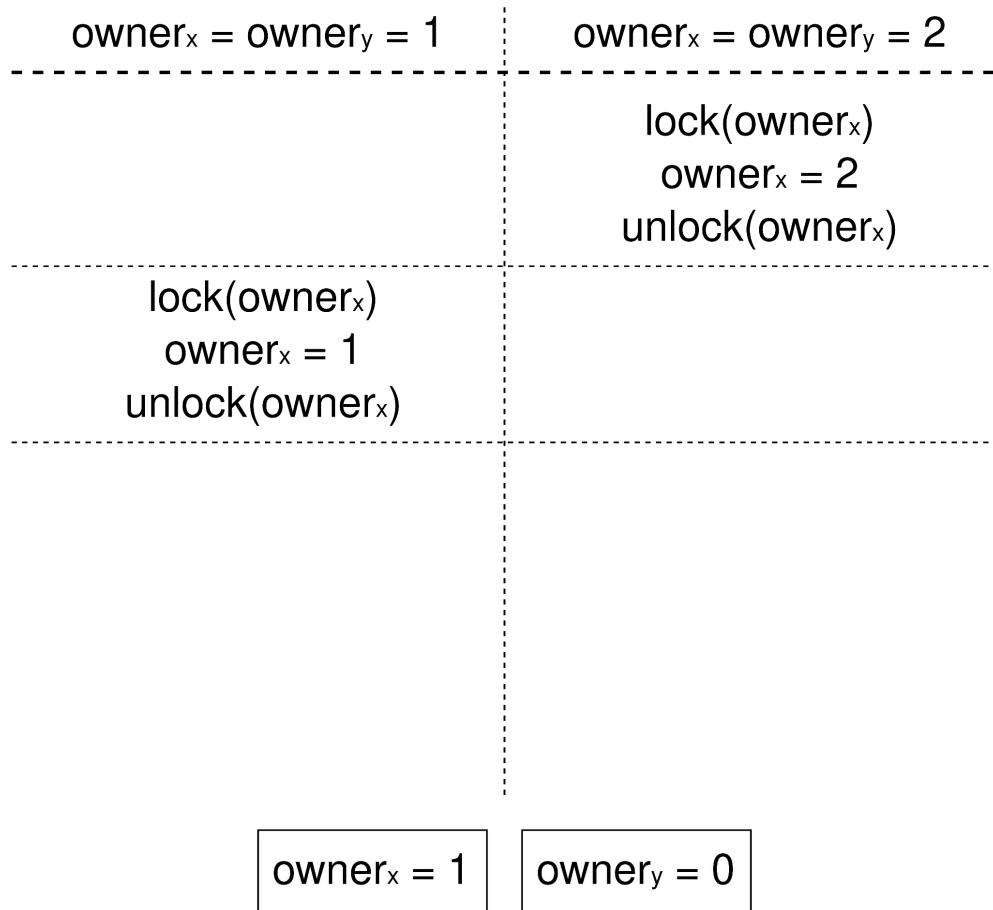
Изоляция: блокировки

- Можно ли отпустить блокировку сразу после чтения/записи?
- Два пользователя хотят забронировать места x и y
- Либо оба места сразу, либо ни одного места



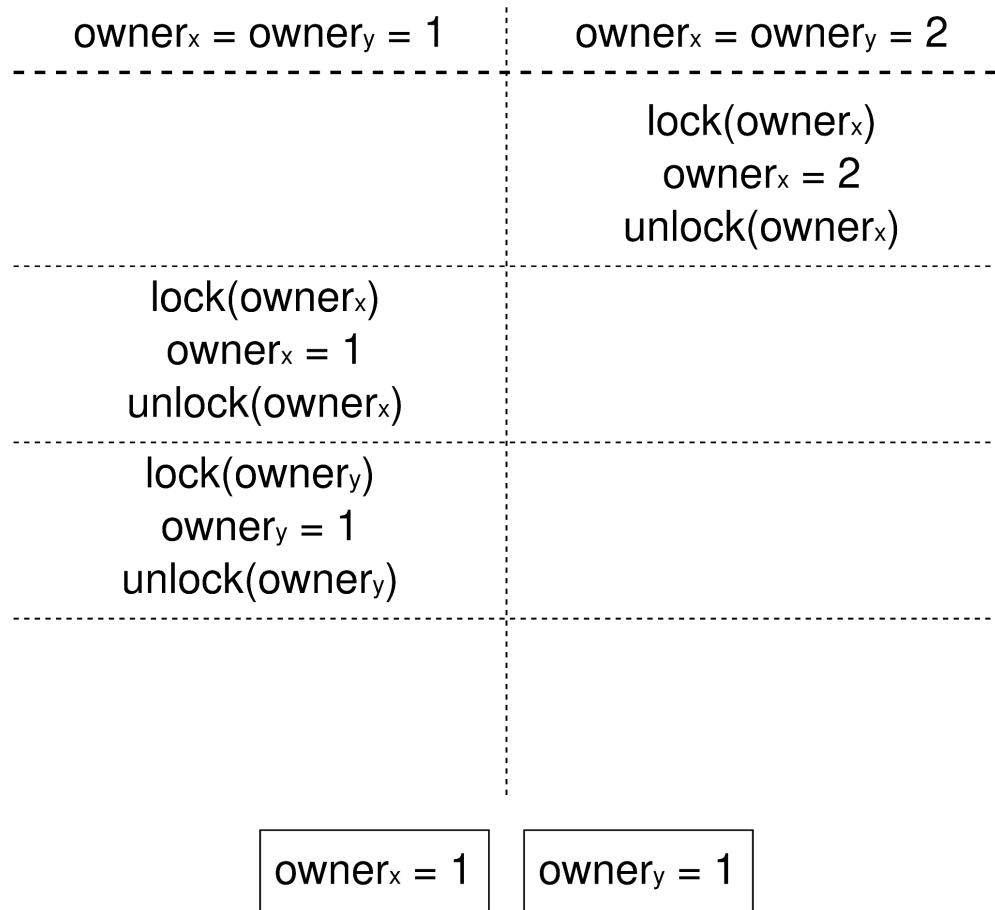
Изоляция: блокировки

- Можно ли отпустить блокировку сразу после чтения/записи?
- Два пользователя хотят забронировать места x и y
- Либо оба места сразу, либо ни одного места



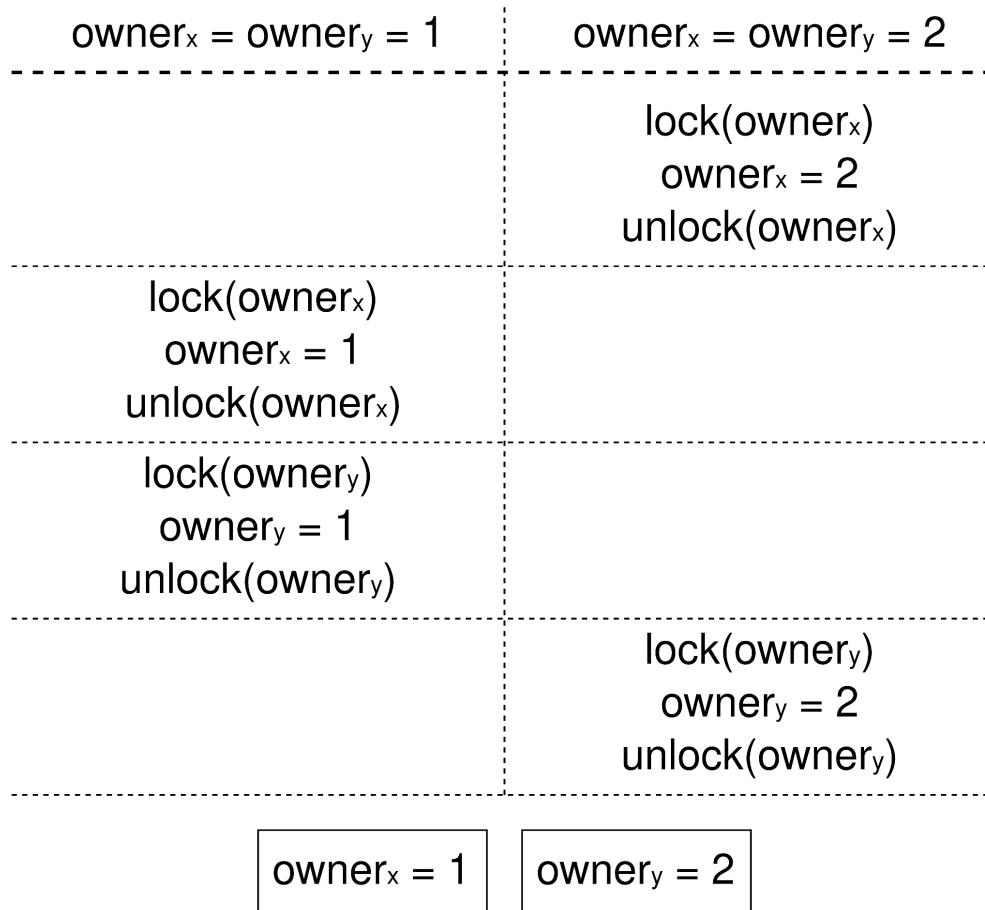
Изоляция: блокировки

- Можно ли отпустить блокировку сразу после чтения/записи?
- Два пользователя хотят забронировать места x и y
- Либо оба места сразу, либо ни одного места



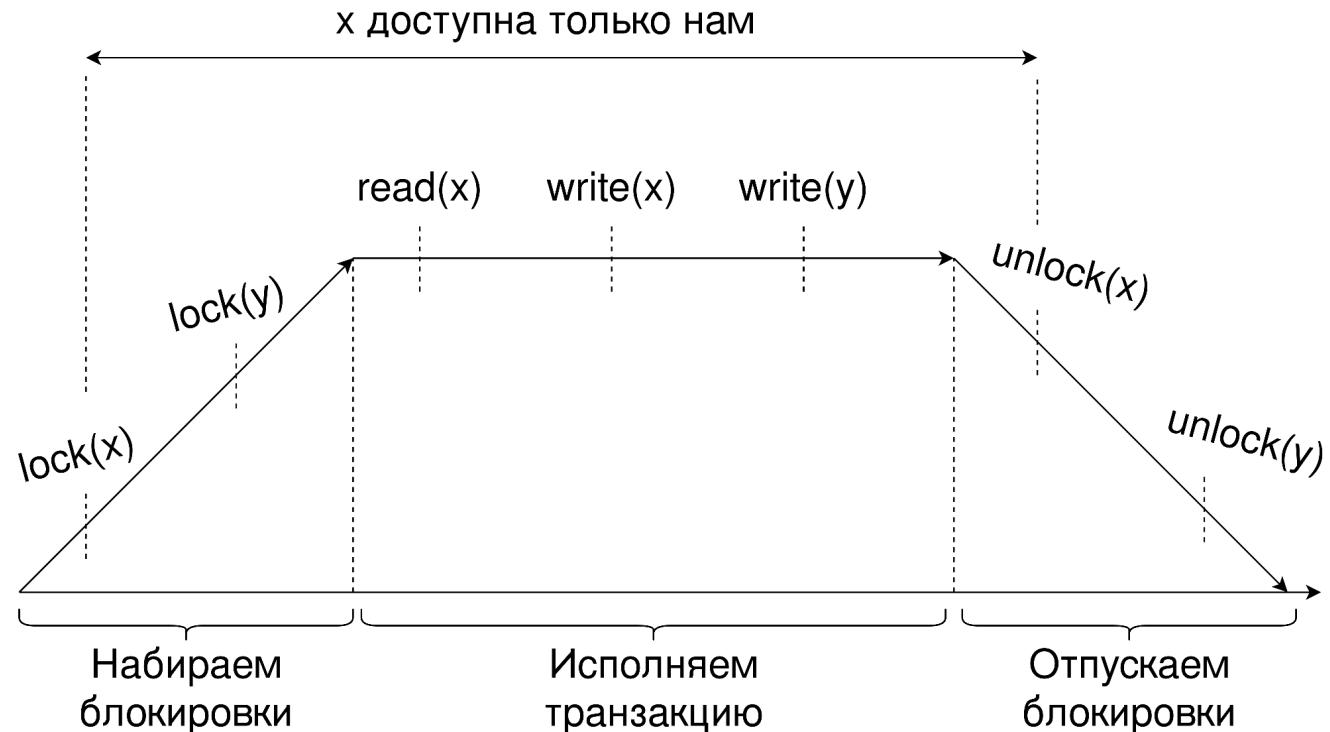
Изоляция: блокировки

- Можно ли отпустить блокировку сразу после чтения/записи?
- Нет
- Каждая из двух транзакций применена частично
- Каждый пользователь забронировал лишь одно место



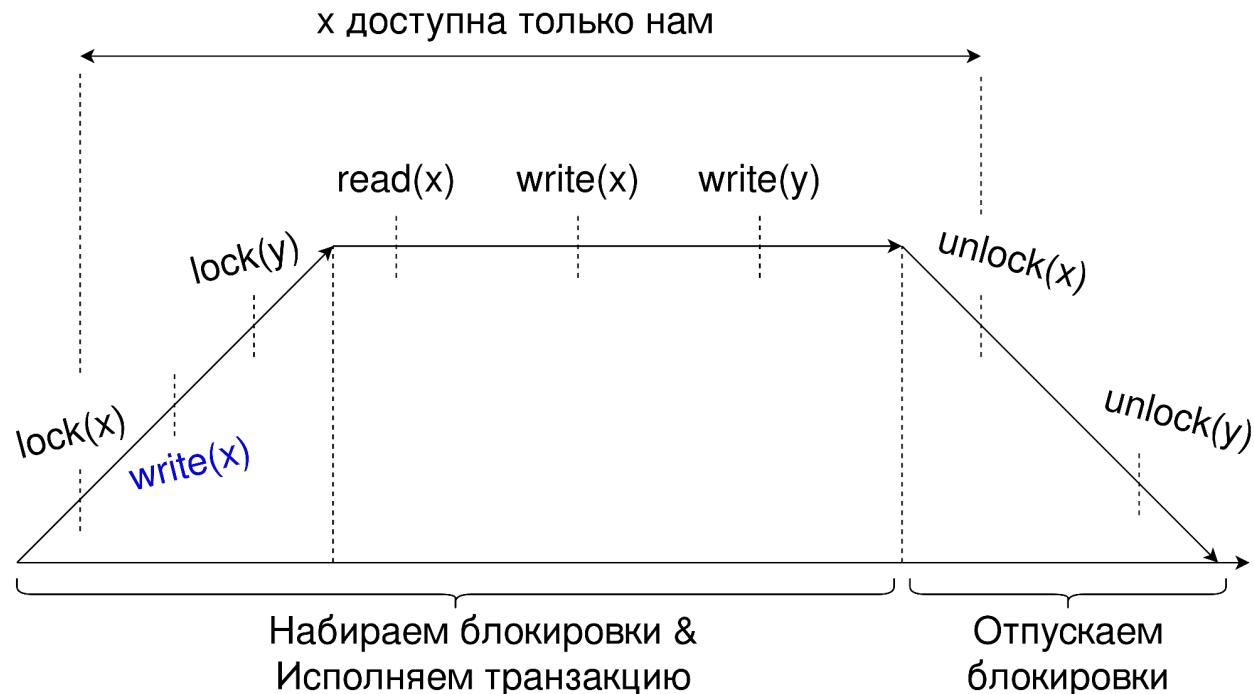
Строгий алгоритм двухфазной блокировки

- Гарантирует линеаризуемость
- Транзакции можно логически упорядочить одну за другой



Строгий алгоритм двухфазной блокировки

- Можно начинать работать с переменной сразу после взятия блокировки
- Не дожидаясь взятия всех блокировок

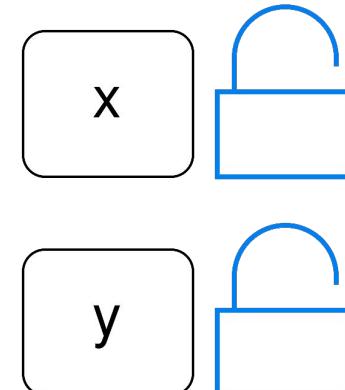
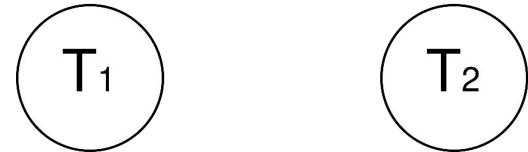


Двухфазная блокировка: взаимная блокировка

- Убийца
прогресса в
системах с
блокировками

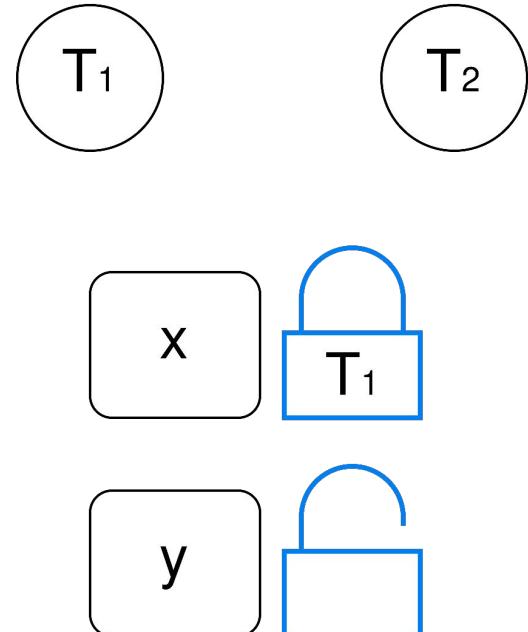
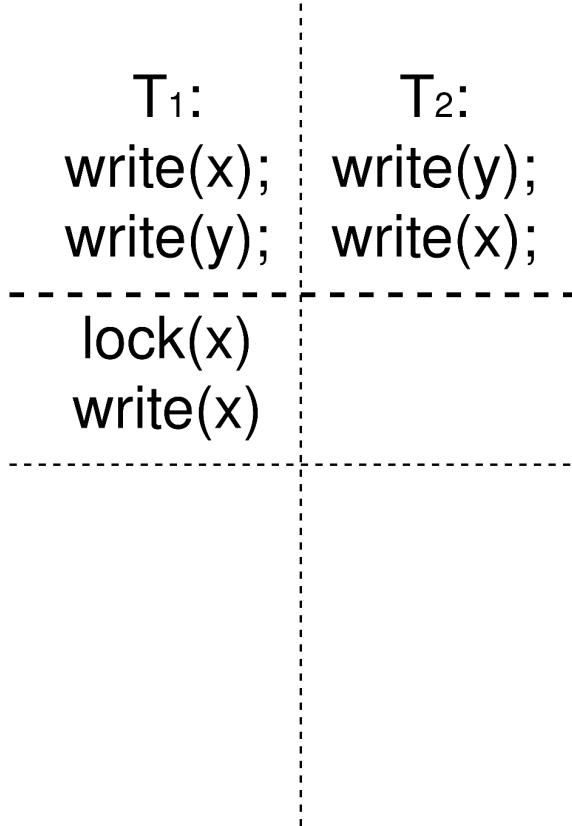
T_1 :
`write(x);`
`write(y);`

T_2 :
`write(y);`
`write(x);`



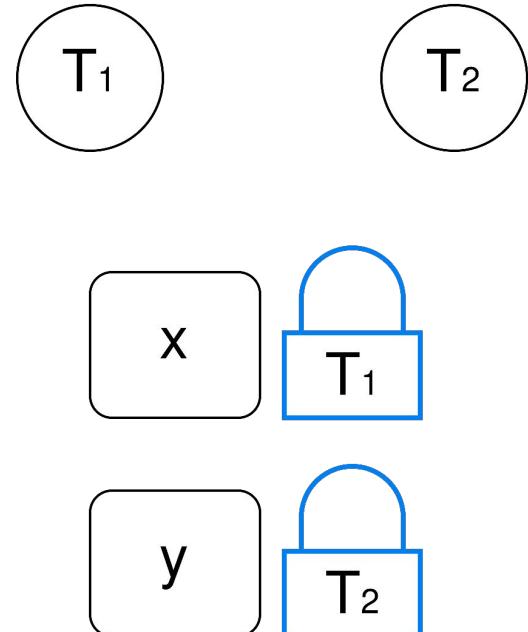
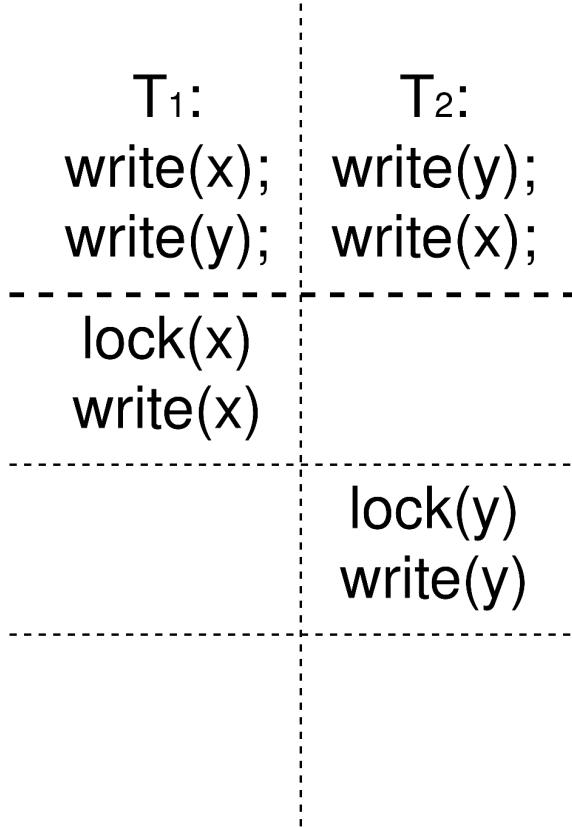
Двухфазная блокировка: взаимная блокировка

- Возможна при определённом порядке захвата блокировок



Двухфазная блокировка: взаимная блокировка

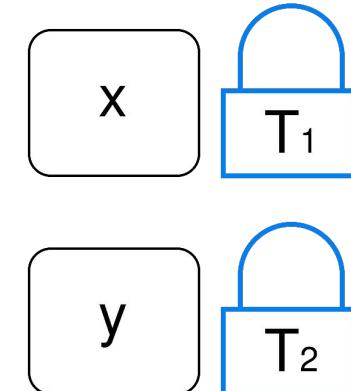
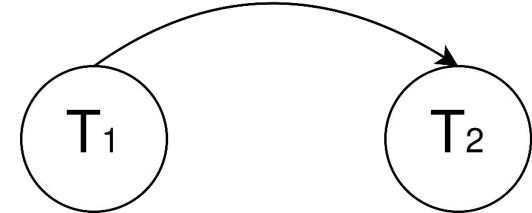
- Возможна при определённом порядке захвата блокировок



Двухфазная блокировка: взаимная блокировка

- Возможна при определённом порядке захвата блокировок

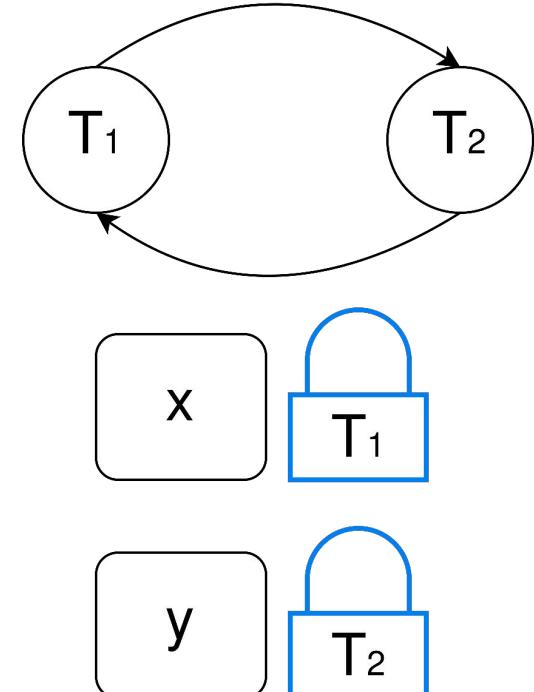
	T ₁ : write(x); write(y);	T ₂ : write(y); write(x);
lock(x) write(x)		
	lock(y) write(y)	
lock(y)		



Двухфазная блокировка: взаимная блокировка

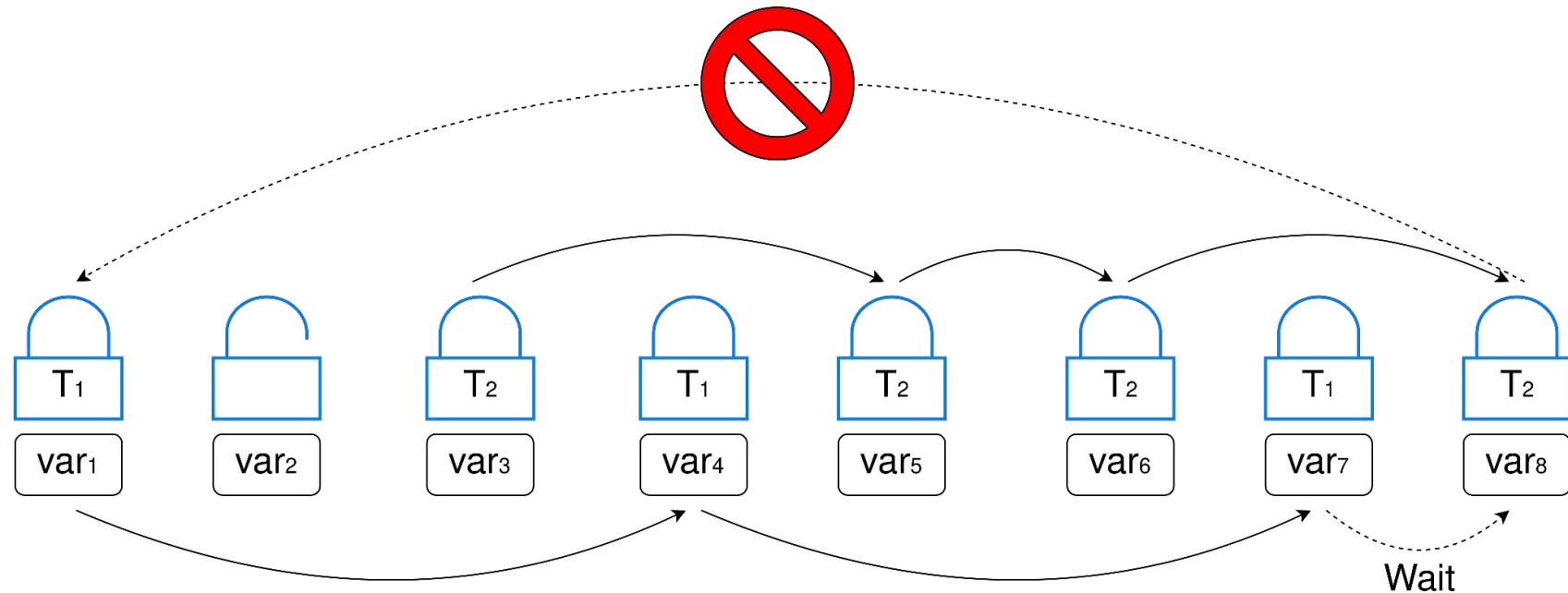
- Цикл в графе ожидания
- Нужно откатывать одну из транзакций
- Необходим deadlock detection
- Локально стабильный предикат

	T ₁ : write(x); write(y); lock(x) write(x)	T ₂ : write(y); write(x); lock(y) write(y)
	lock(y)	
		lock(x)



Иерархическая блокировка

- Упорядочиваем переменные единым образом
- Захватываем блокировки только в этом порядке
- Граф ожидания ацикличен



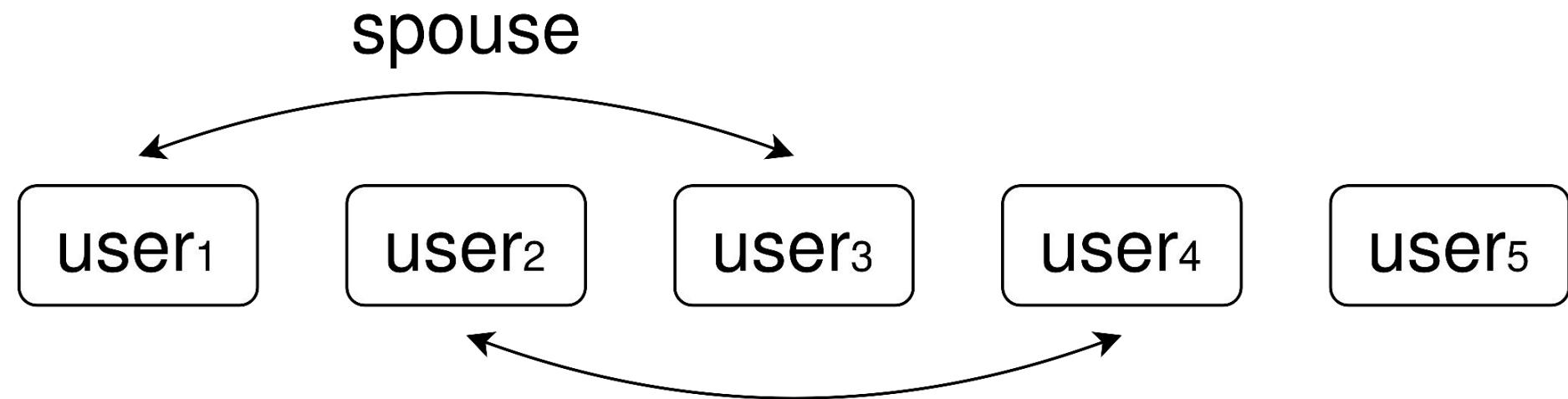
Иерархическая блокировка: невозможность

- Иерархическая блокировка может быть невозможна, если мы заранее не знаем все данные, с которыми будем работать

```
1 transaction family_balance(uid):
2     user := load_user(uid)
3     spouse := load_user(user.spouse_id)
4     if spouse ≠ nil:
5         return user.balance + spouse.balance
6     else:
7         return user.balance
```

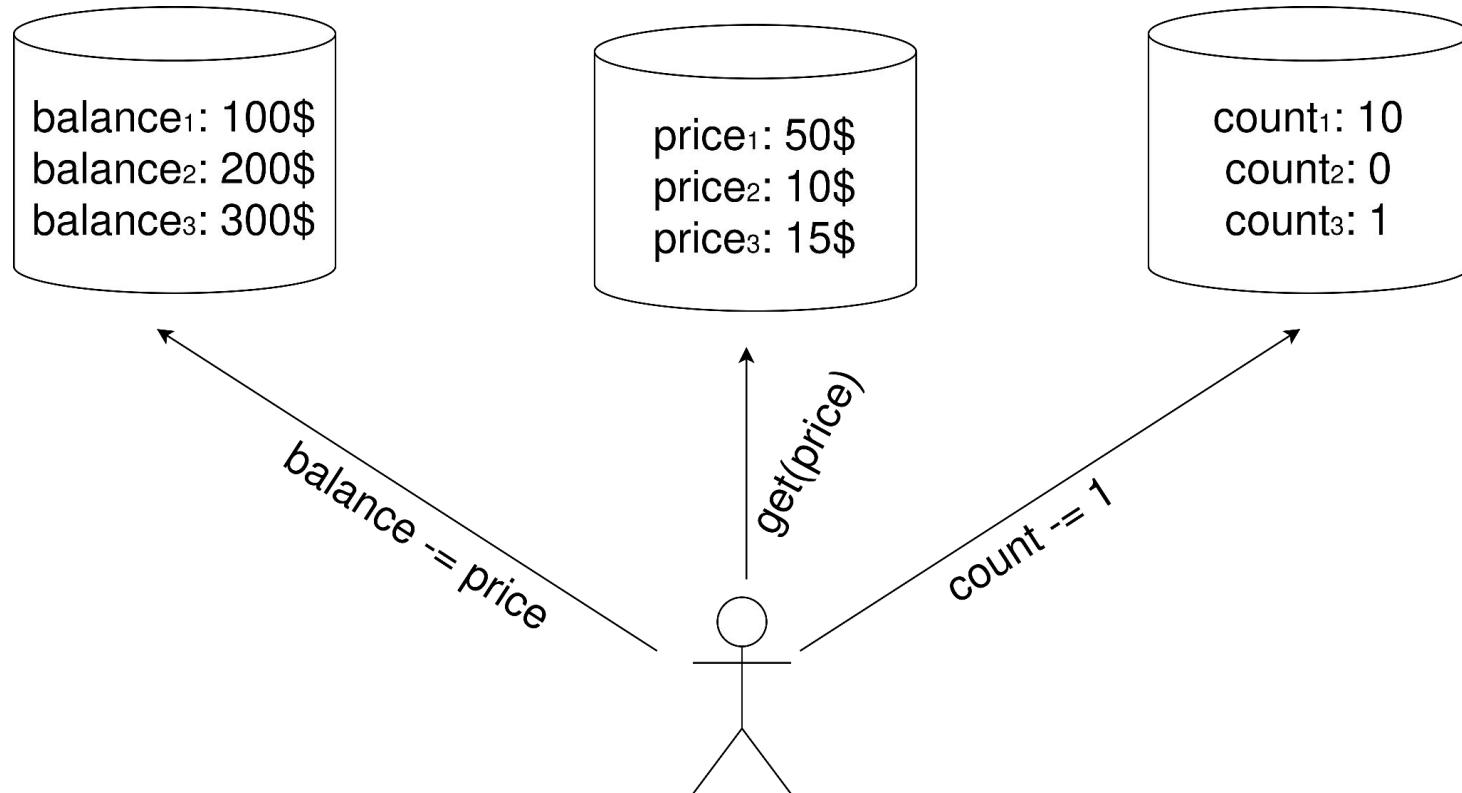
Иерархическая блокировка: невозможность

- В случае вызова `family_balance(uid=3)` загрузим сначала третьего пользователя, потом первого
- Нарушение порядка
- Возможен дедлок
- В общем случае дедлеки неизбежны



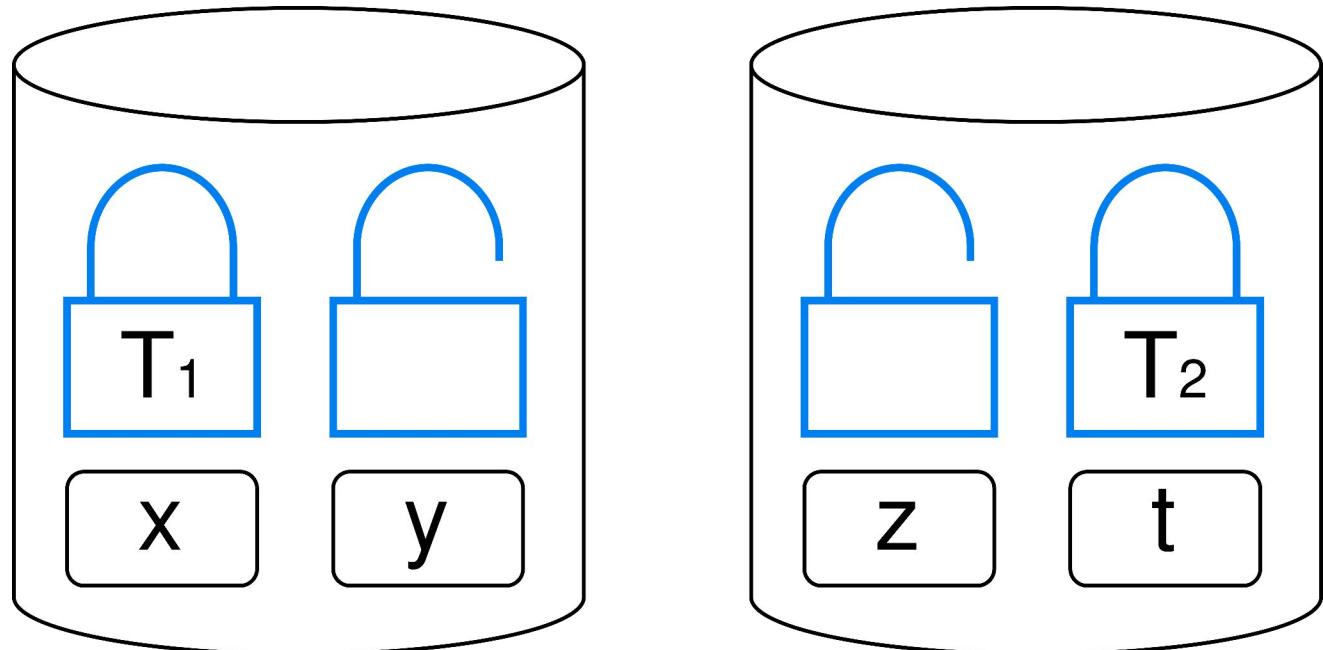
Распределённые транзакции

- Транзакция работает с данными на разных серверах



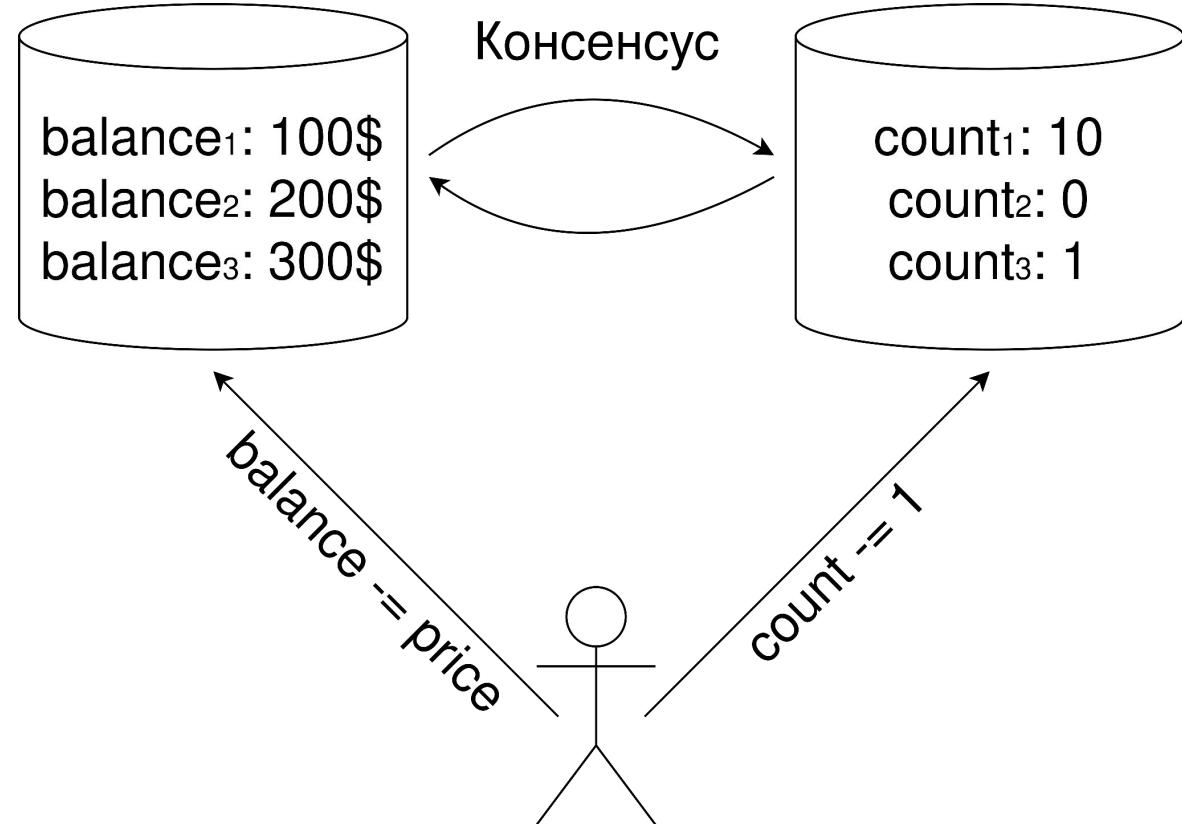
Распределённые транзакции: блокировки

- Каждый сервер единолично следит за блокировками своих данных
- Распределённые блокировки **не нужны**



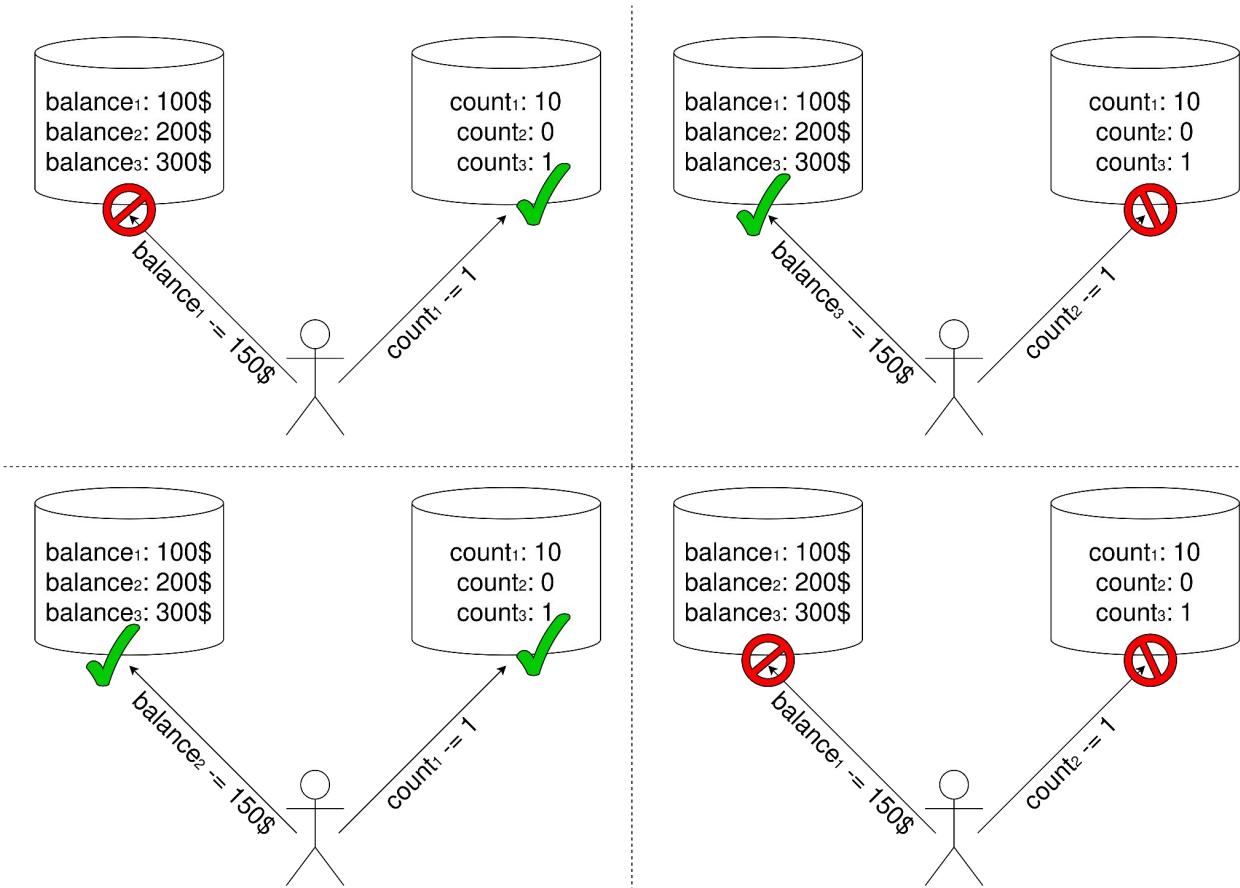
Распределённые транзакции: консенсус

- Транзакция затрагивает данные на нескольких серверах
- На всех затронутых серверах нужно либо применить изменение, либо нет
- Нужно прийти к консенсусу



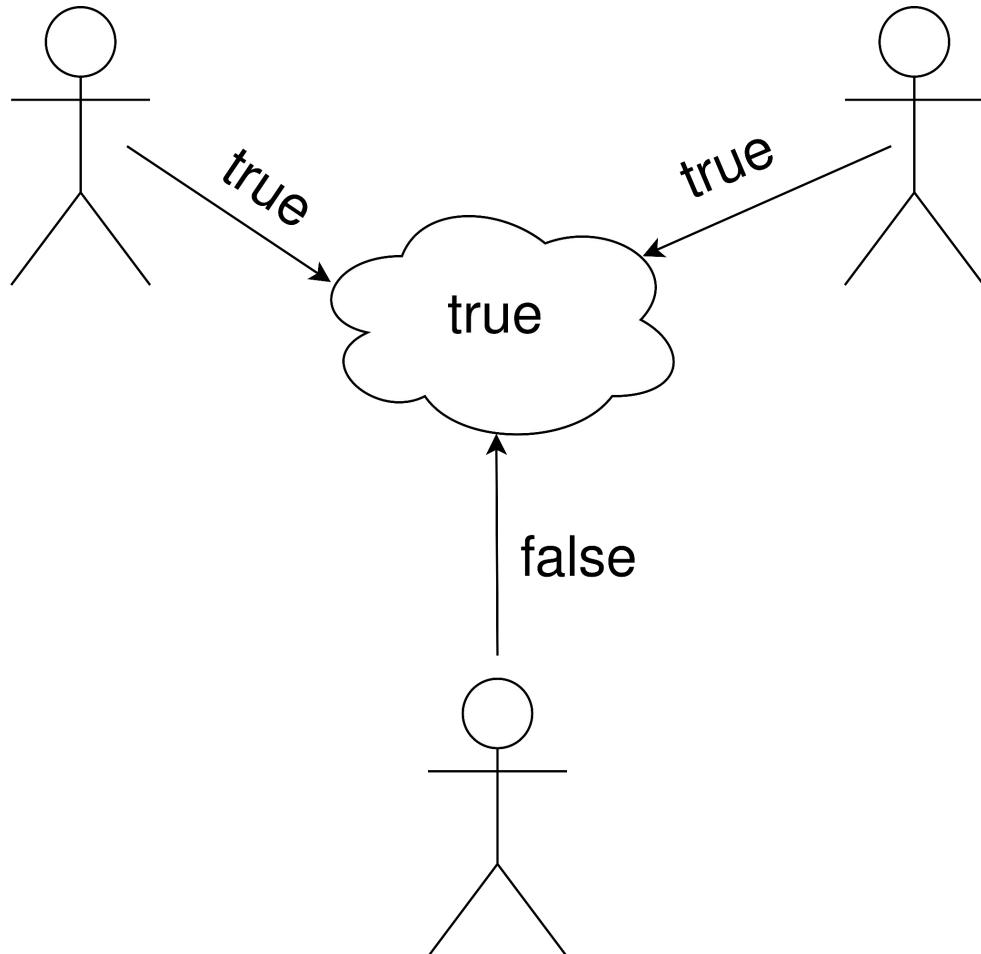
Координированный откат

- Если хотя бы на одном сервере транзакция не может завершиться, её изменения не должны быть применены ни к одному серверу



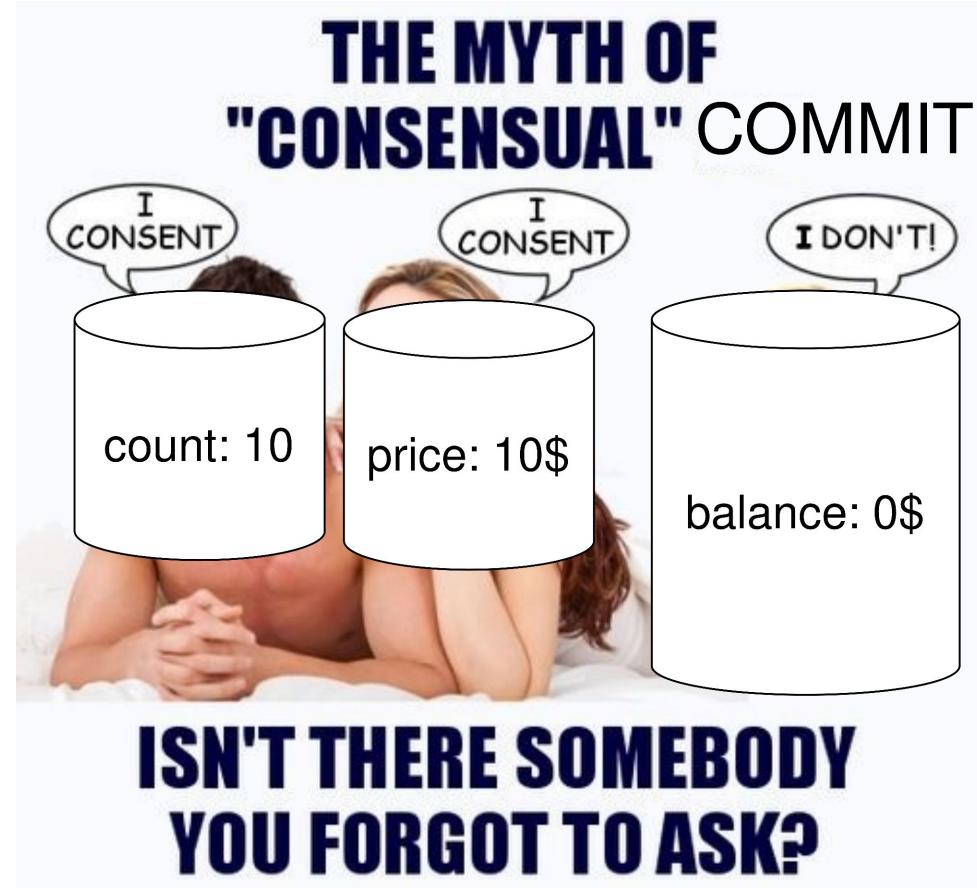
Paxos-like консенсус

- Двое соглашаются коммитить транзакцию, третий нет
- Итогом консенсуса может стать решение о коммите
- Так быть не должно



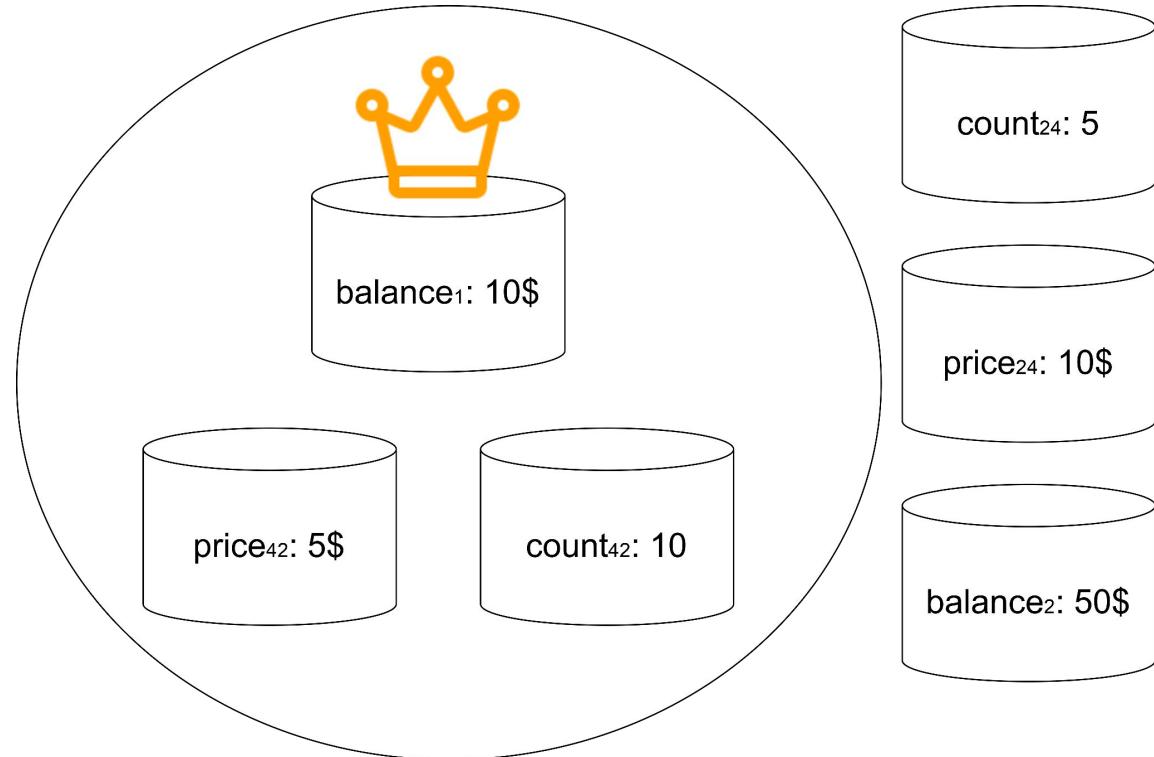
Консенсусный коммит

- Для коммита **все** участники должны согласиться применить транзакцию
- А не какой-то кворум
- Это позволяет нам спроектировать более простой алгоритм консенсуса



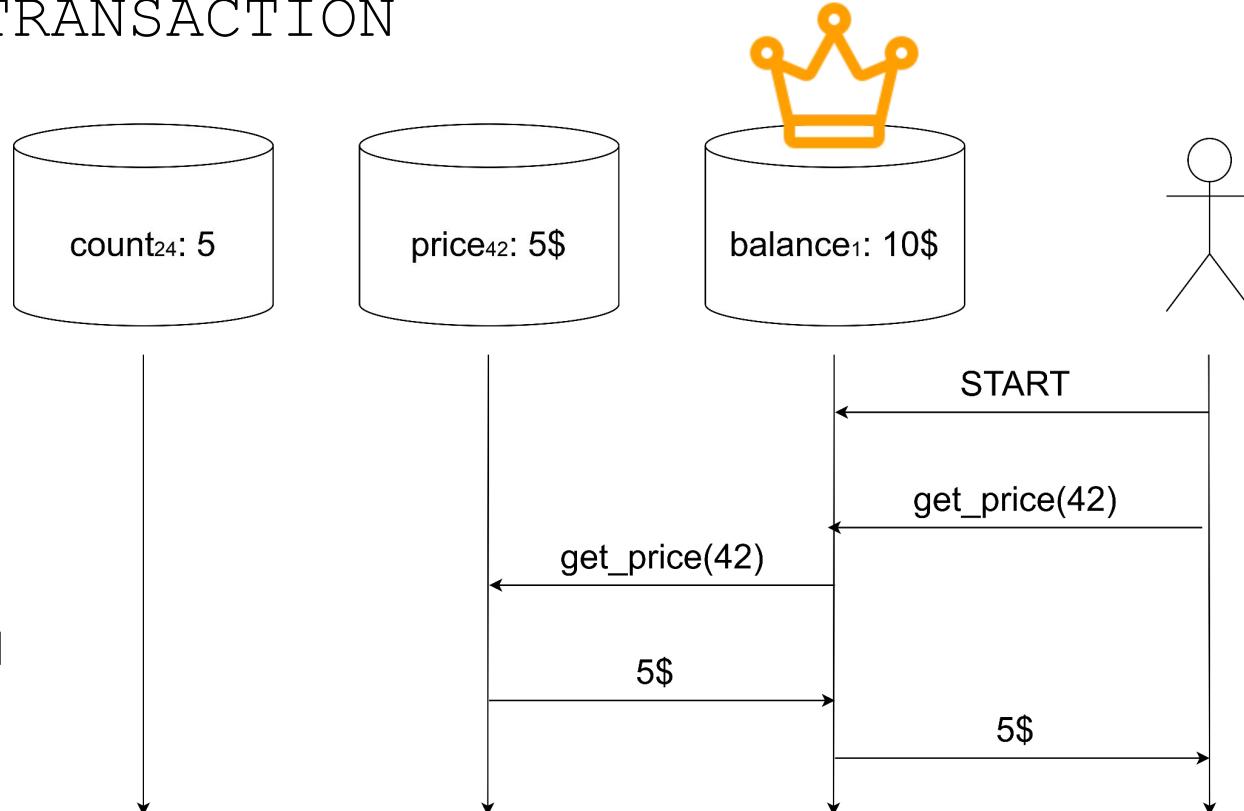
Двухфазный коммит

- Для каждой транзакции на момент коммита известны все участники
- Известно, кто из участников координатор транзакции
- Координатор принимает решение о статусе транзакции



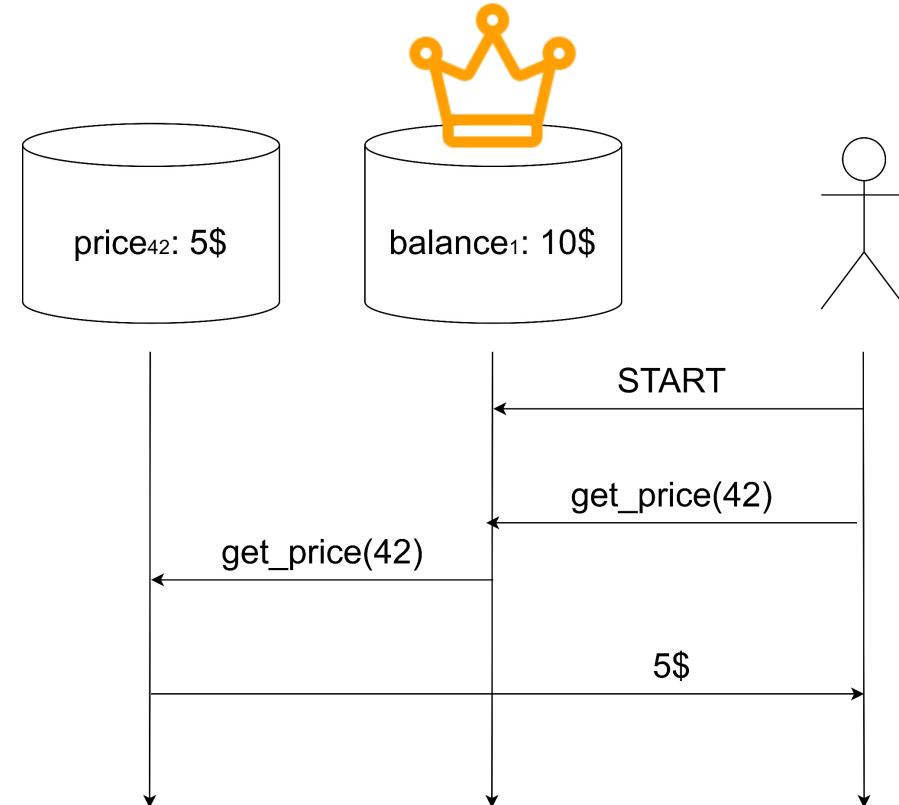
Выбор лидера и состав участников

- Координатором станет сервер, получивший от клиента команду START TRANSACTION
- Клиент направляет все команды через координатора
- Координатор знает состав участников
- Выступает в роли прокси



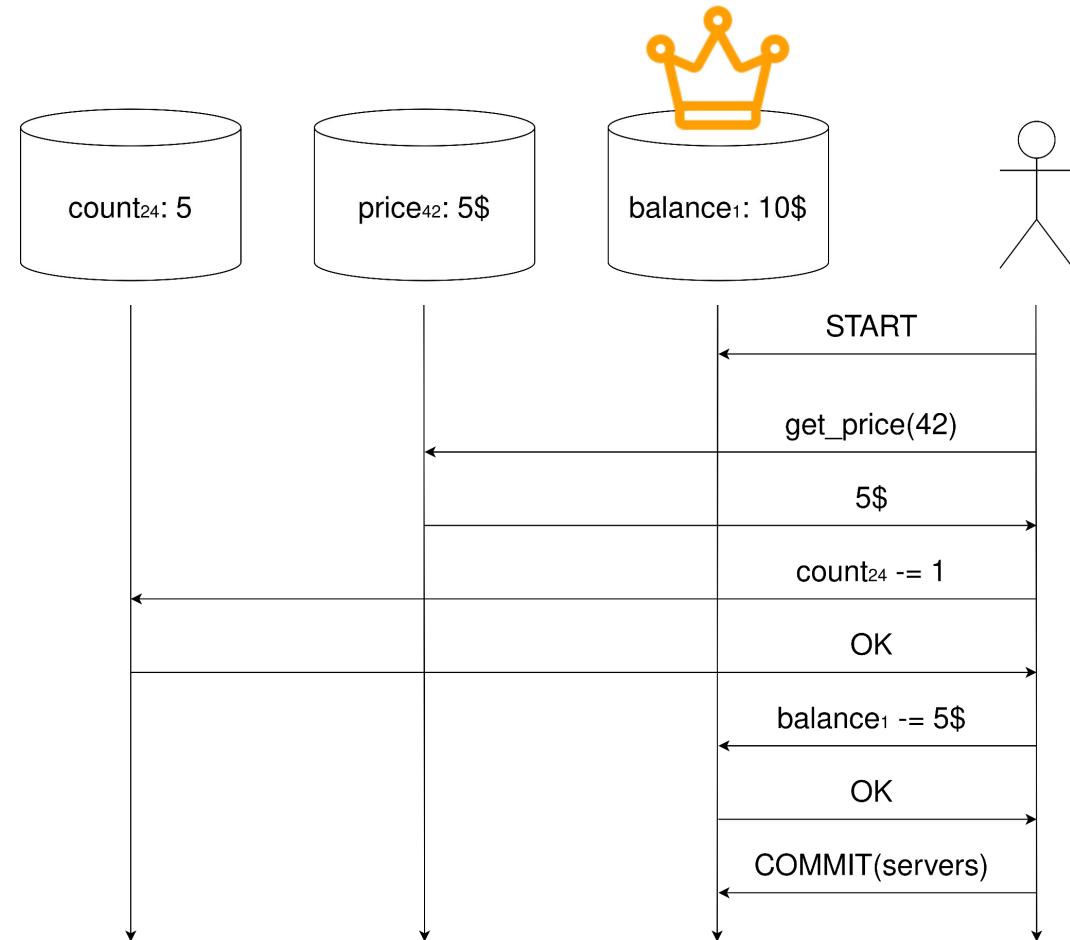
Direct server return

- Сервер может направлять **ответ** напрямую клиенту
- Запросы всё ещё ходят через координатора
 - Чтобы координатор знал всех участников
- Ответ обычно гораздо больше запроса
- Экономим пропускную способность координатора



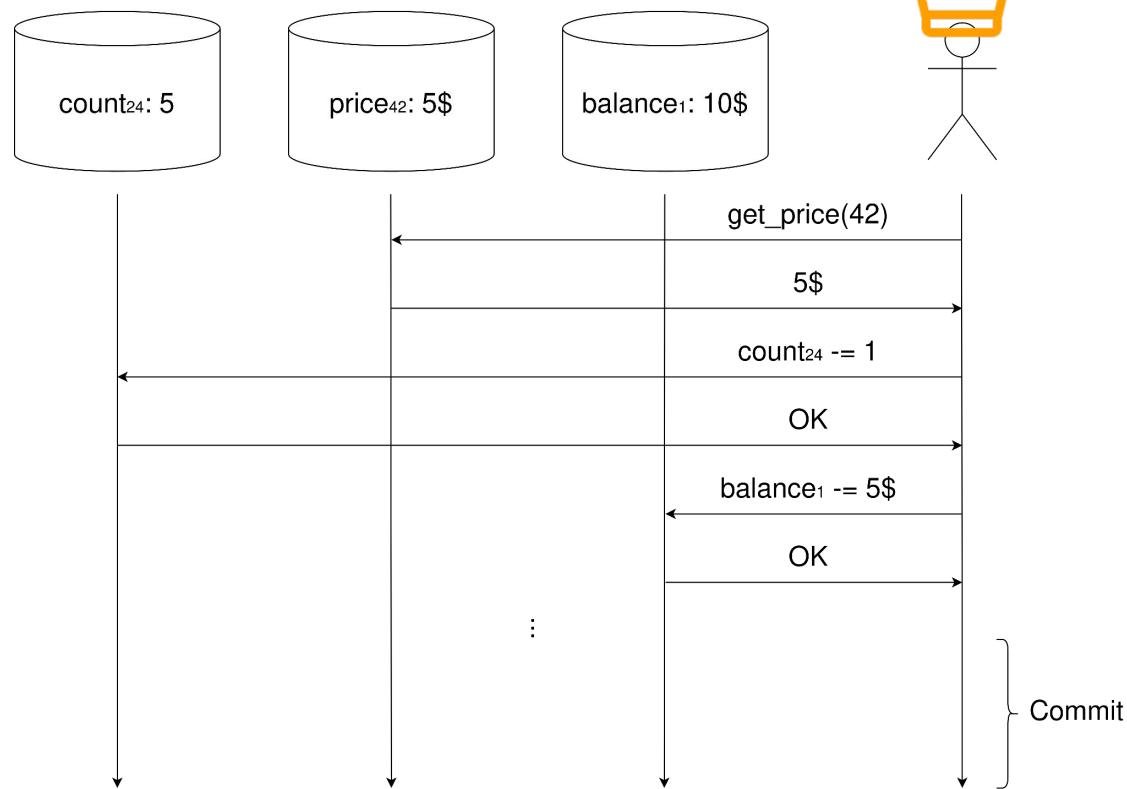
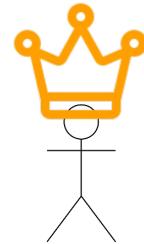
Выбор лидера и список участников

- Клиент может сам сообщить лидеру список участников в момент коммита
- Участники это те сервера, с которыми клиент общался в ходе транзакции



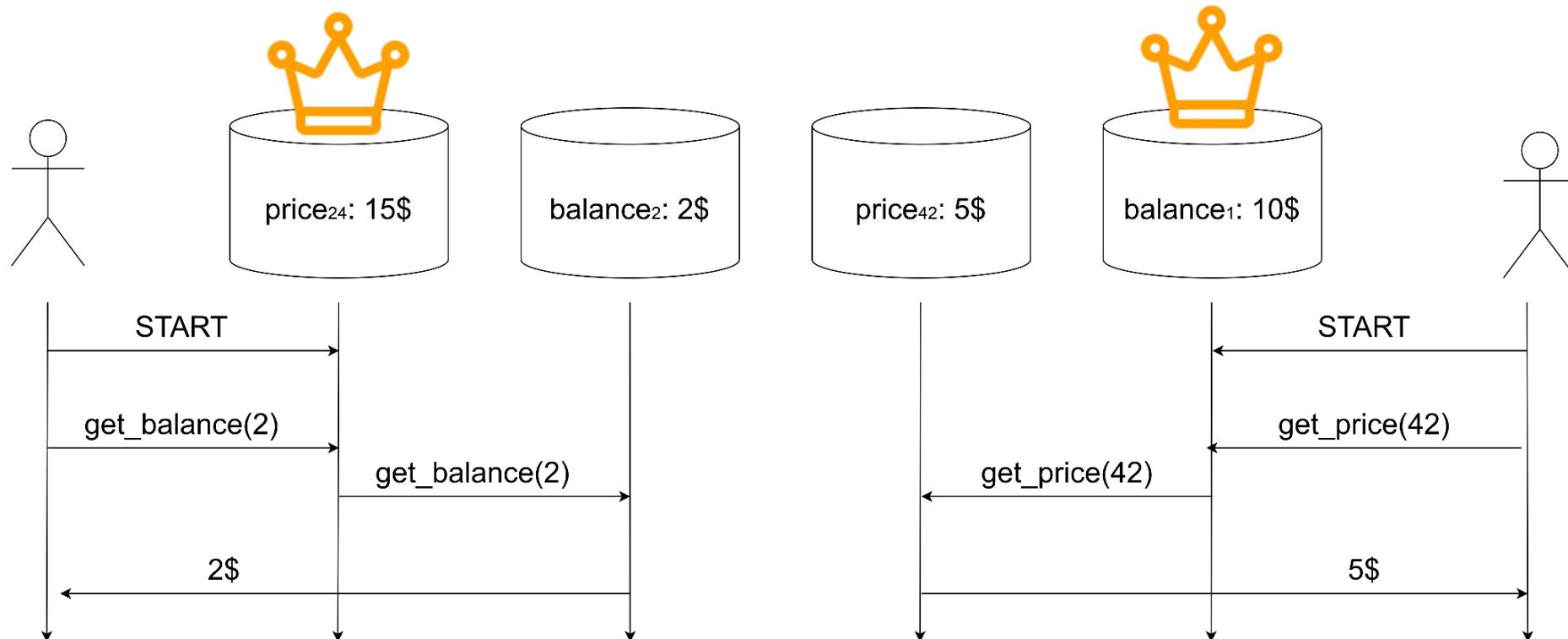
Выбор лидера и список участников

- Клиент может сам играть роль координатора
- Так как знает всех участников



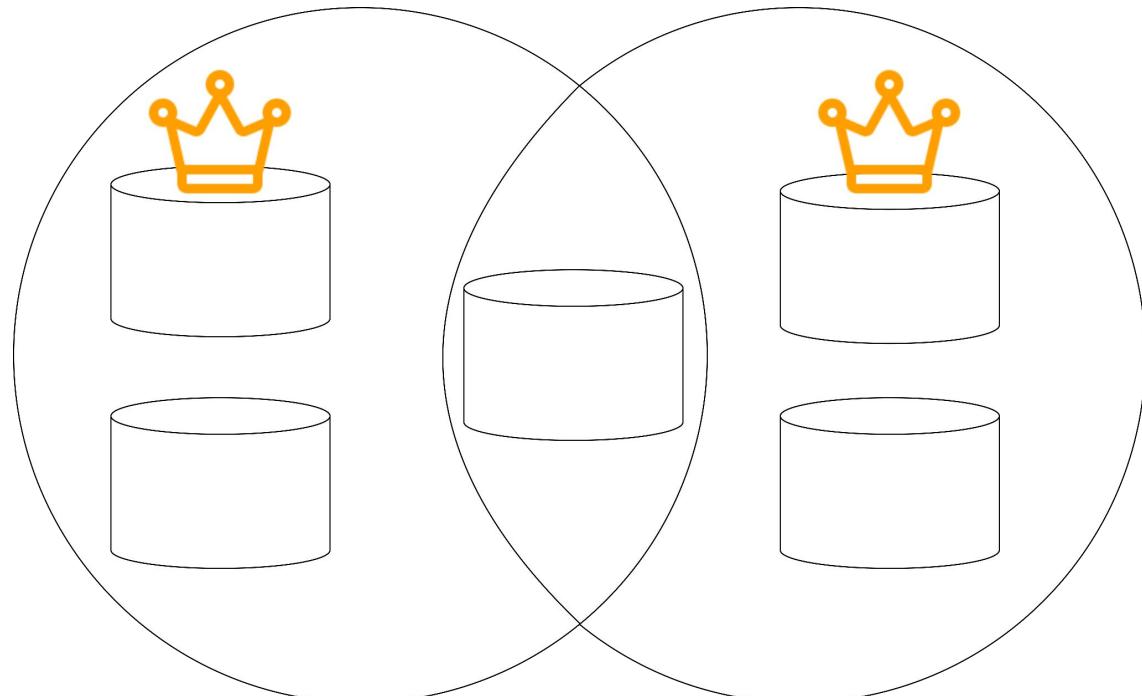
Множество координаторов

- У разных транзакций **разные** координаторы
- Система не централизованная



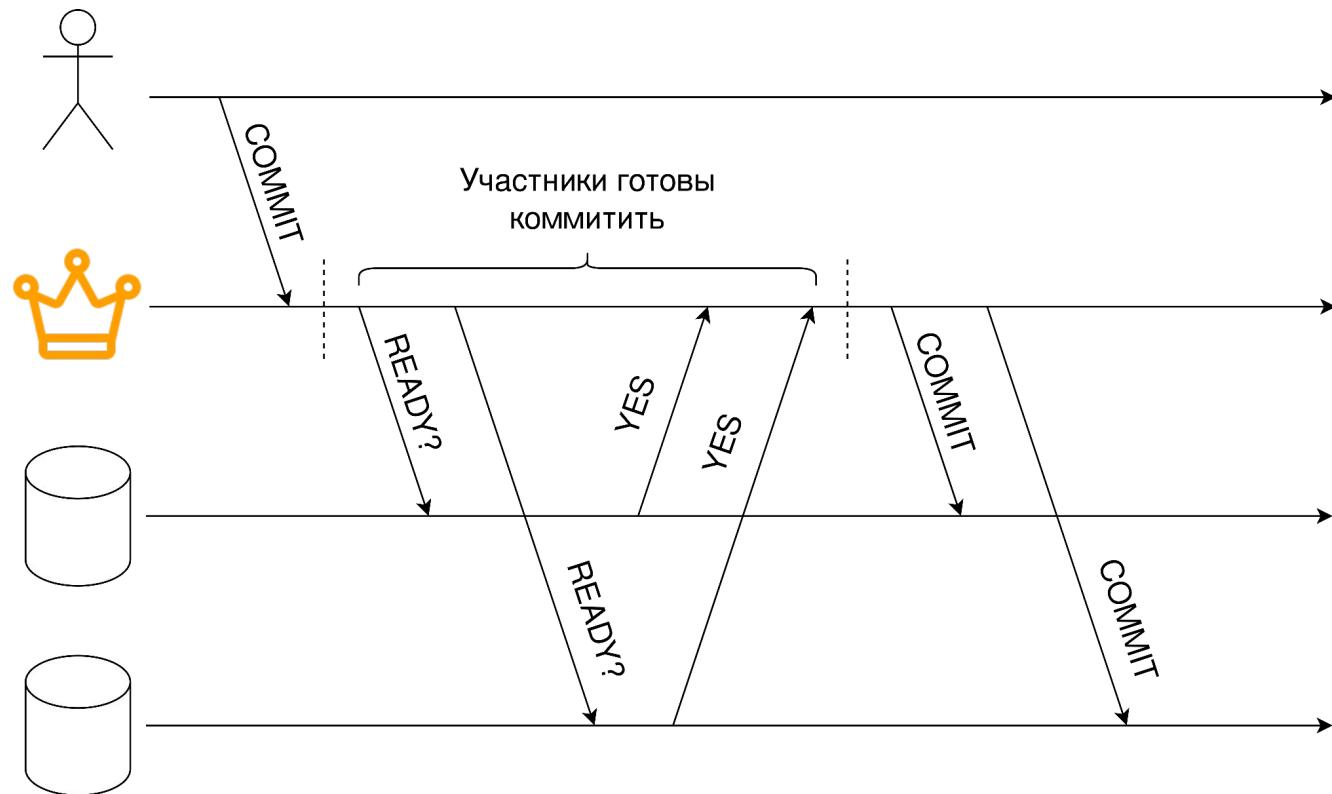
Множество координаторов

- Множество участников двух транзакций может пересекаться
- Координатор одной транзакции может быть обычным участником другой
- Кворумы тут ни при чём
- Потому что могут и не пересекаться



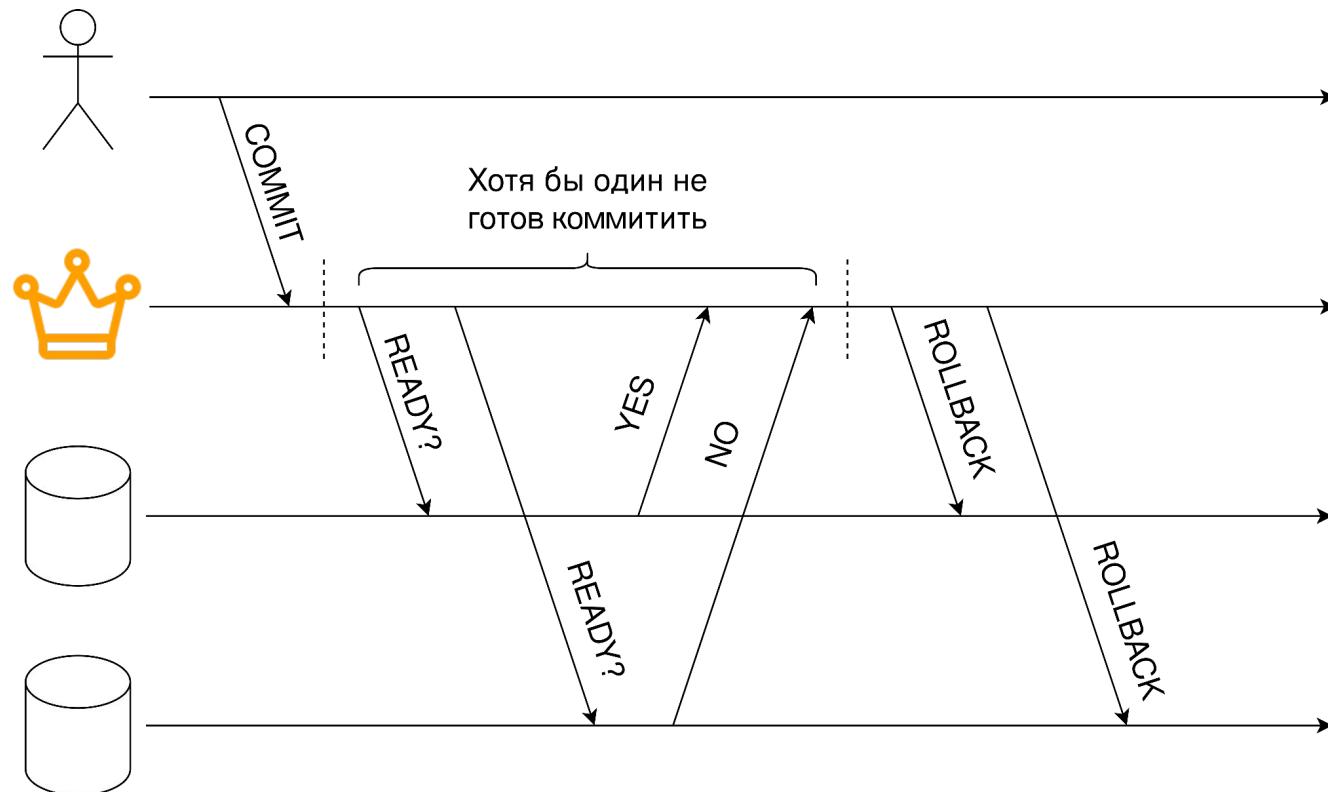
Двухфазный коммит

- Координатор опрашивает всех участников, готовы ли они коммитить
- Получив от всех участников ответ “да”, рассыпает решение о коммите



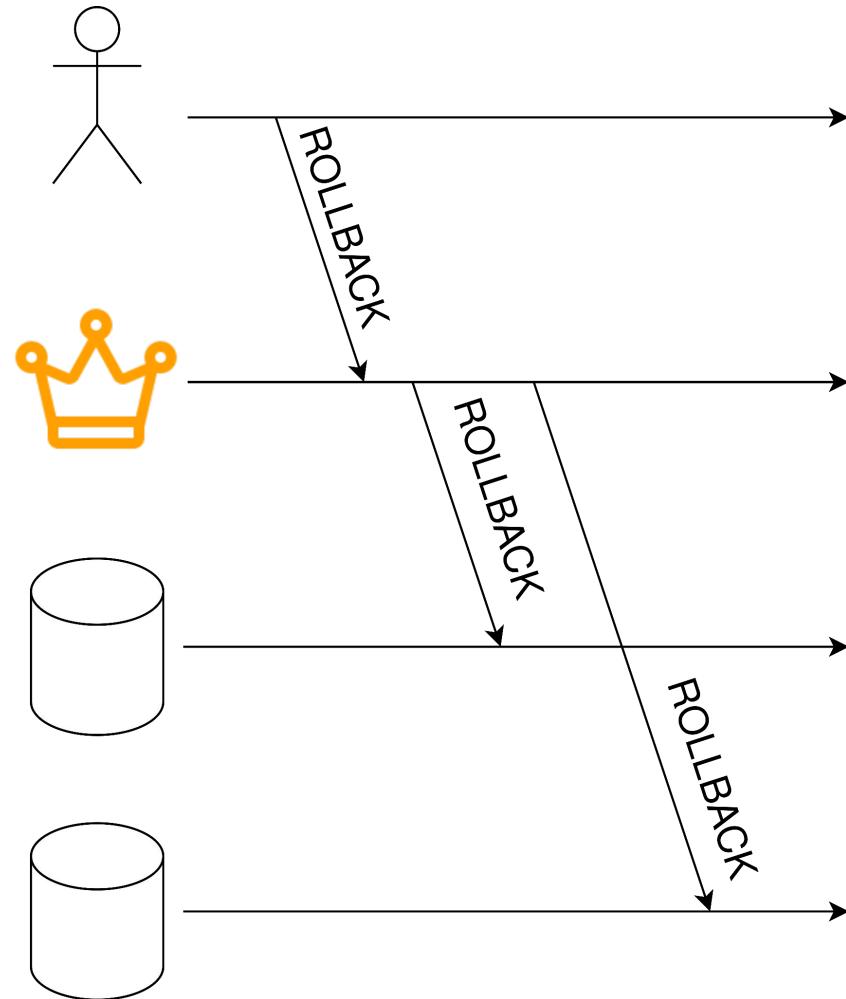
Двухфазный коммит

- Если хотя бы один участник не готов коммитить, откатываем транзакцию
- Сообщаем всем участникам об откате



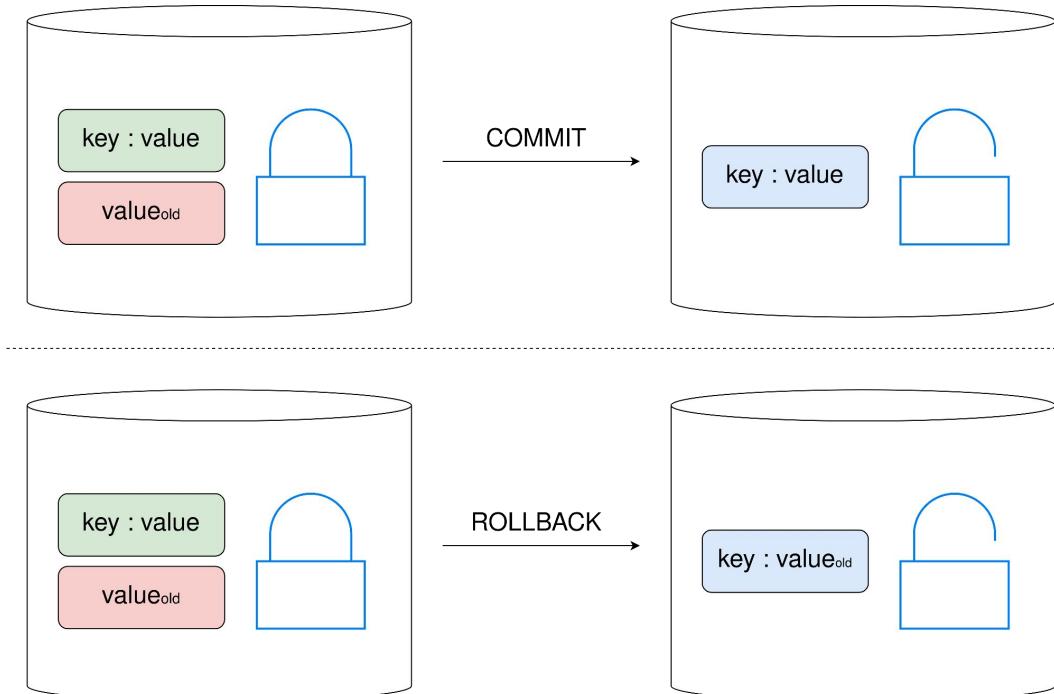
Двухфазный коммит

- Если клиент хочет откатить транзакцию, координатор сообщает всем участникам об этом



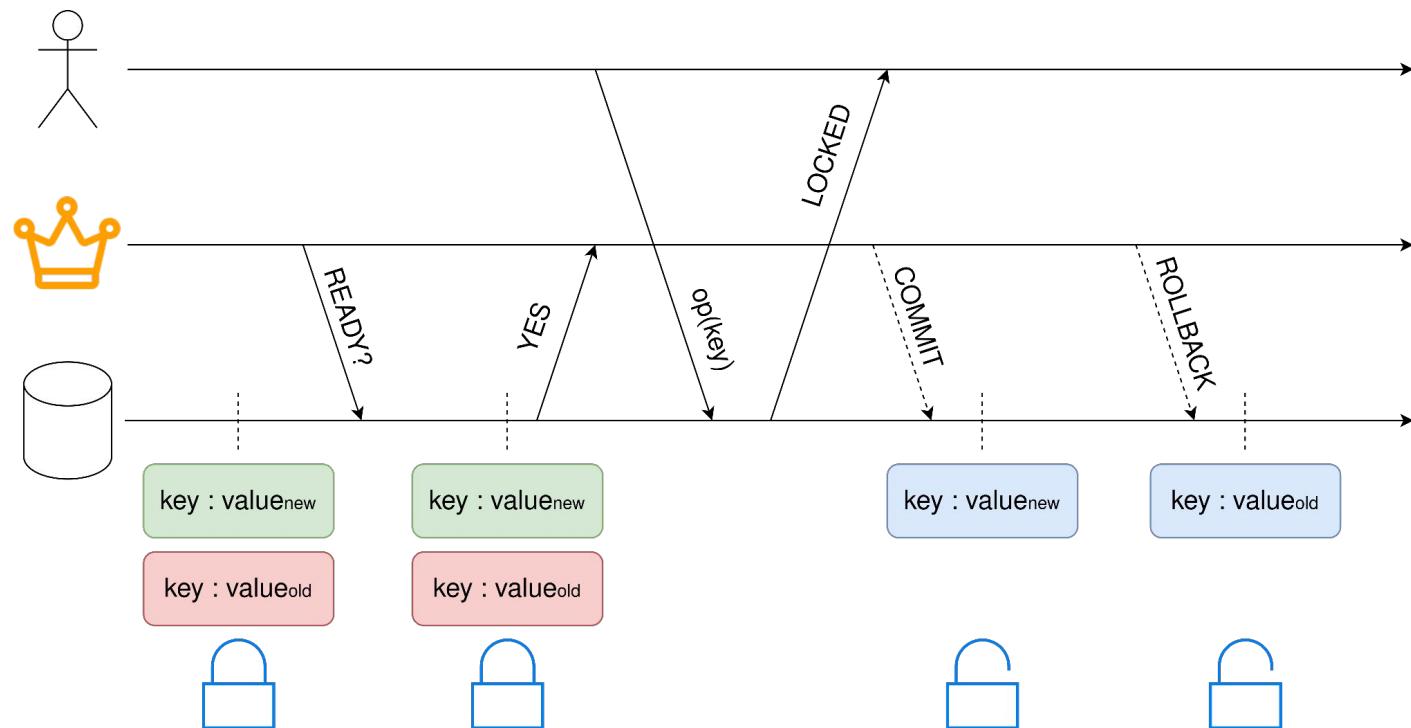
COMMIT vs ROLLBACK

- В ходе транзакции при записи сохраняем старые значения
- Освобождаем блокировку, когда координатор решает коммитить или откатывать
- Либо восстанавливаем старое значение, либо оставляем новое



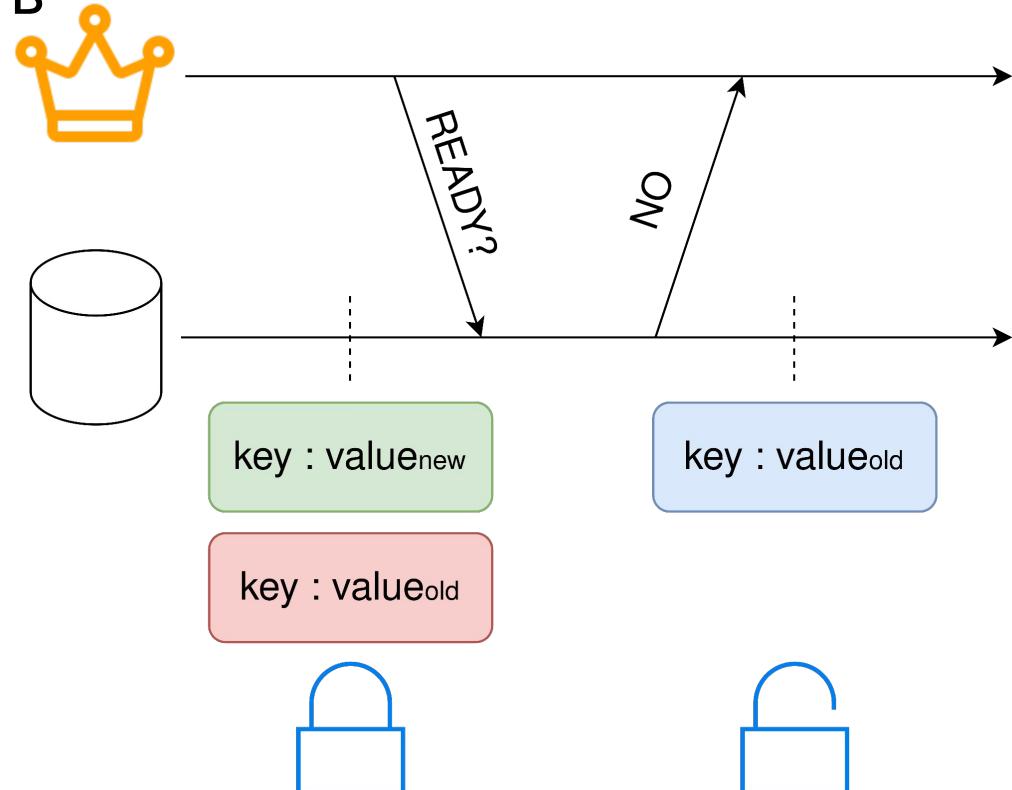
Двухфазный коммит

- Участник не может снимать блокировки после ответа о том, что он готов коммитить
- Не знает, каким будет итоговое решение



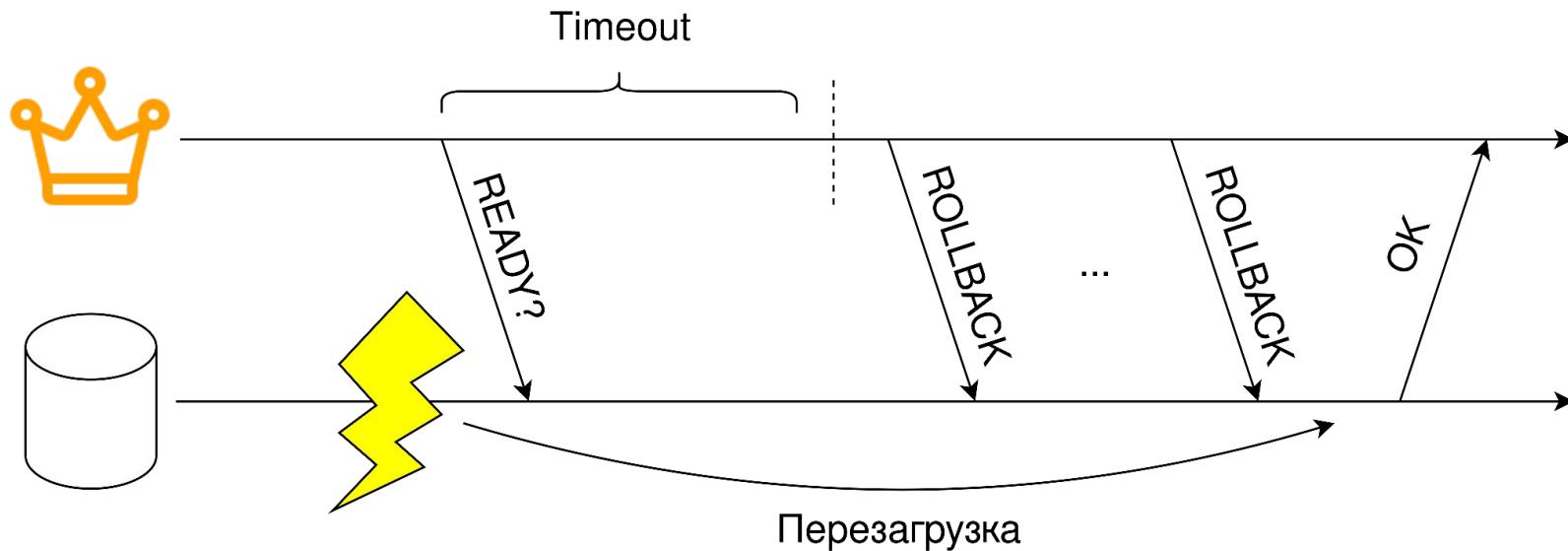
Двухфазный коммит

- Участник может снимать блокировки после ответа о том, что он не готов коммитить
- Восстанавливает старые значения для всех изменённых ключей
- Итоговое решение точно будет ROLLBACK



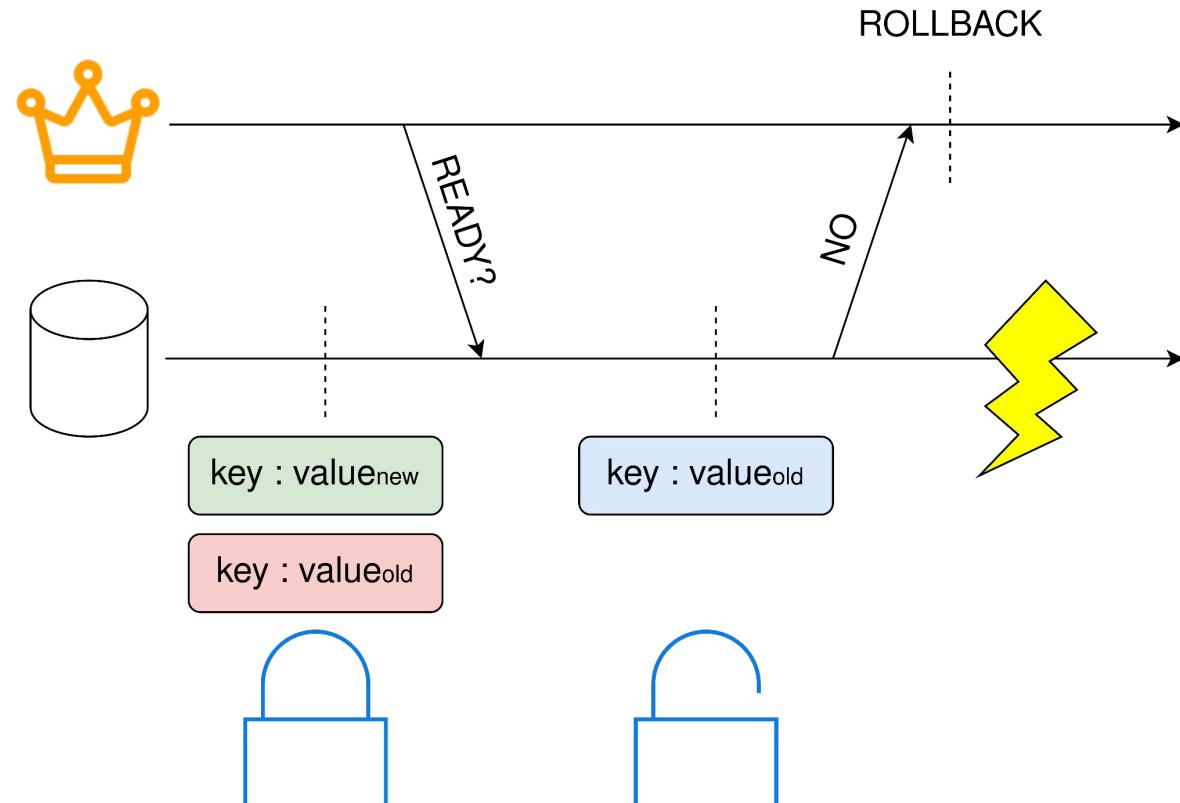
Сбой участника

- При сбое участника до того, как он дал ответ, решаем, что транзакция будет откочена
- Доводим до участника решение, пока не перезагрузится и не подтвердит откат



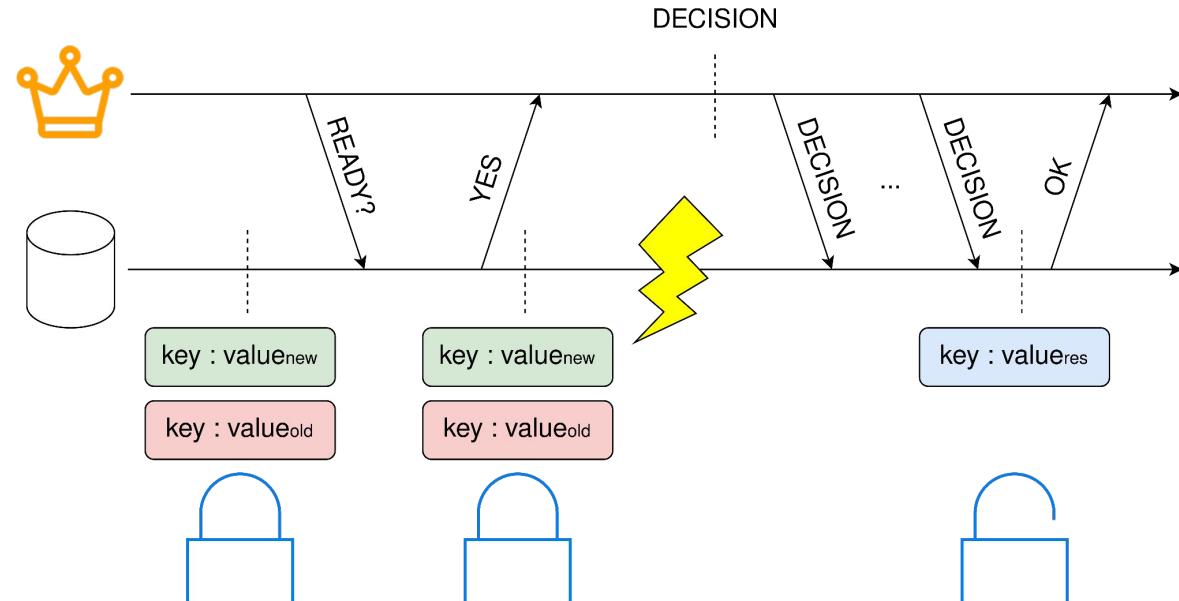
Сбой участника

- Сбой участника после того, как он ответил “нет” ни на что не влияет
- Участник уже точно отпустил блокировки и восстановил старые значения



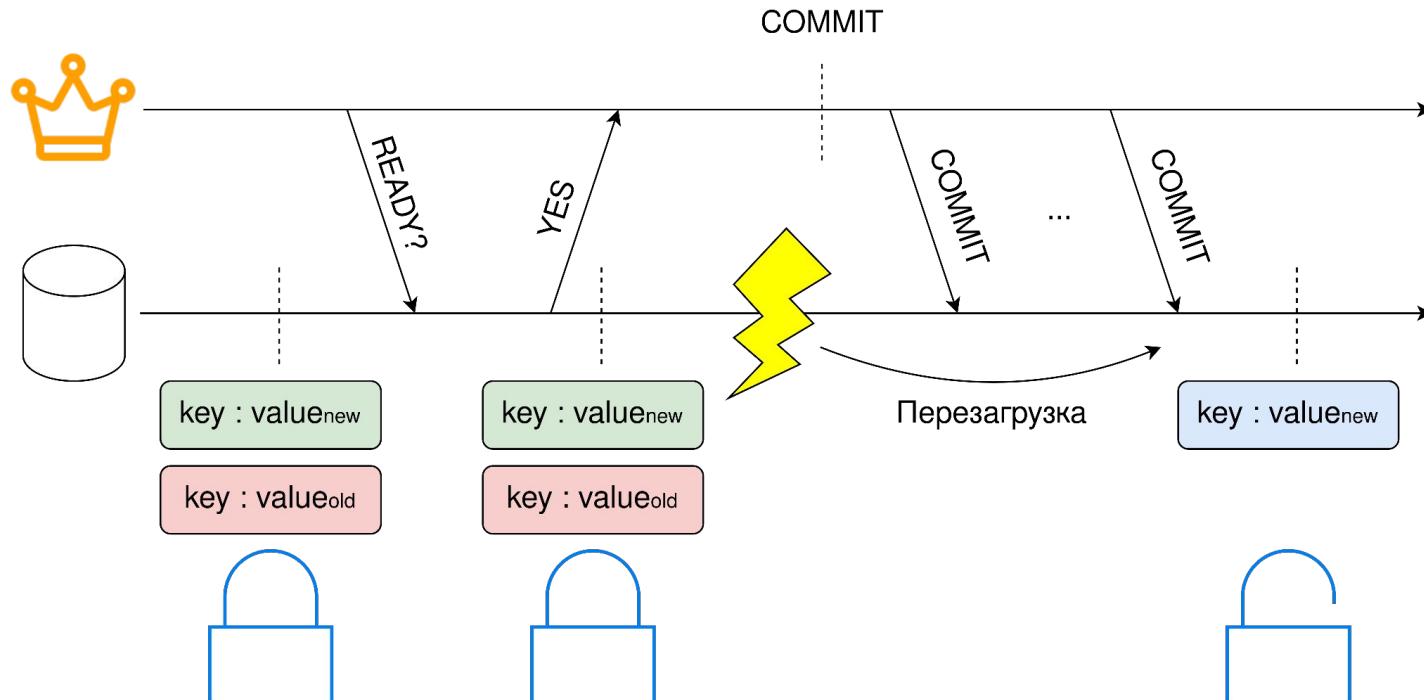
Сбой участника

- Если участник отказал после ответа “да”, то он не снял блокировки
- Нужно довести до него решение
- Подождать, пока не перезагрузится



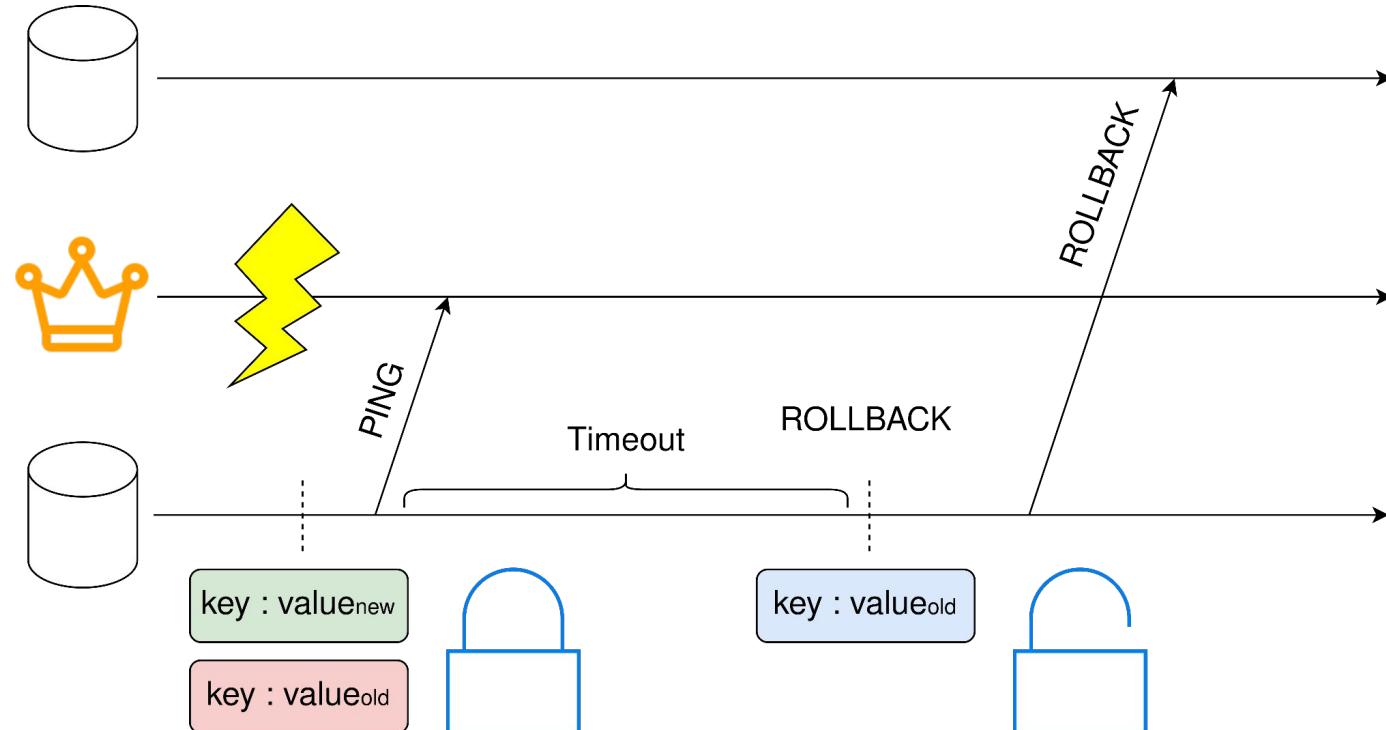
Сбой участника

- Информация о транзакциях, которые участник готов коммитить, должна сохраняться надёжно, чтобы можно было коммитить после сбоя



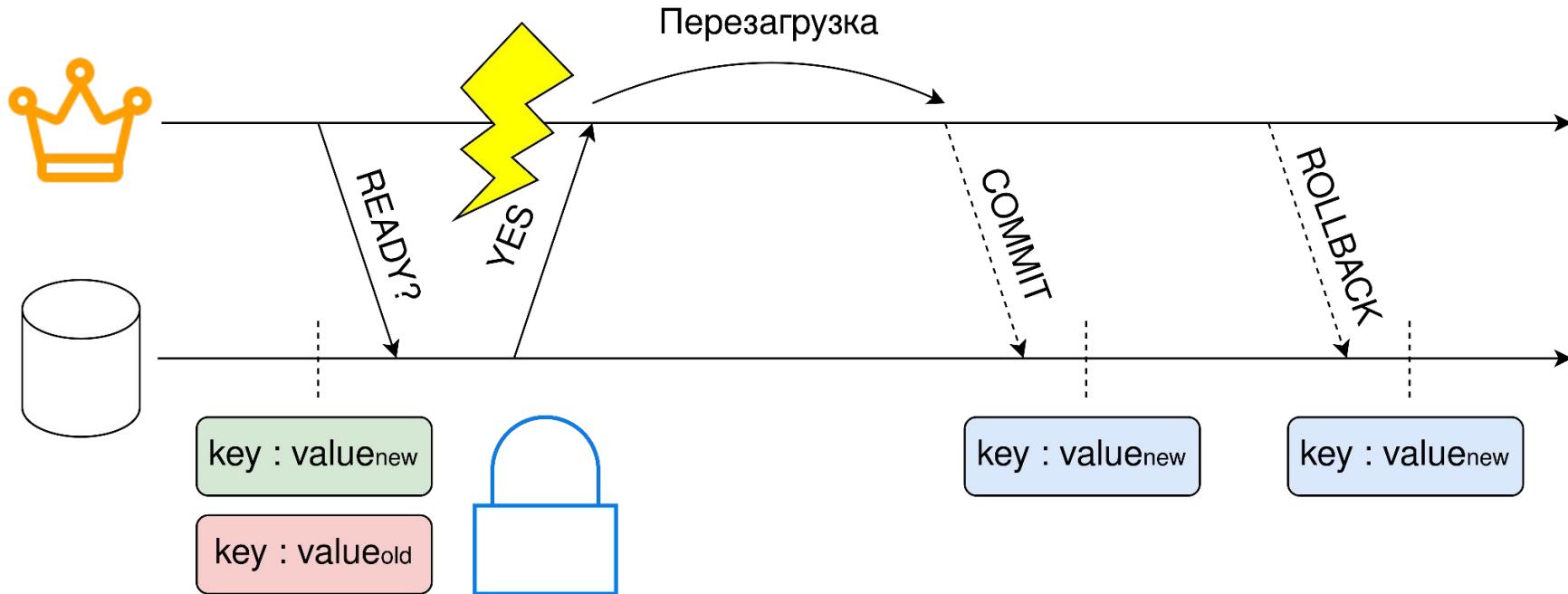
Сбой координатора

- При сбое координатора до ответа участника, участник может решить откатывать транзакцию
- Отпускает блокировки
- Может уведомить других участников
- Чтобы они не ждали



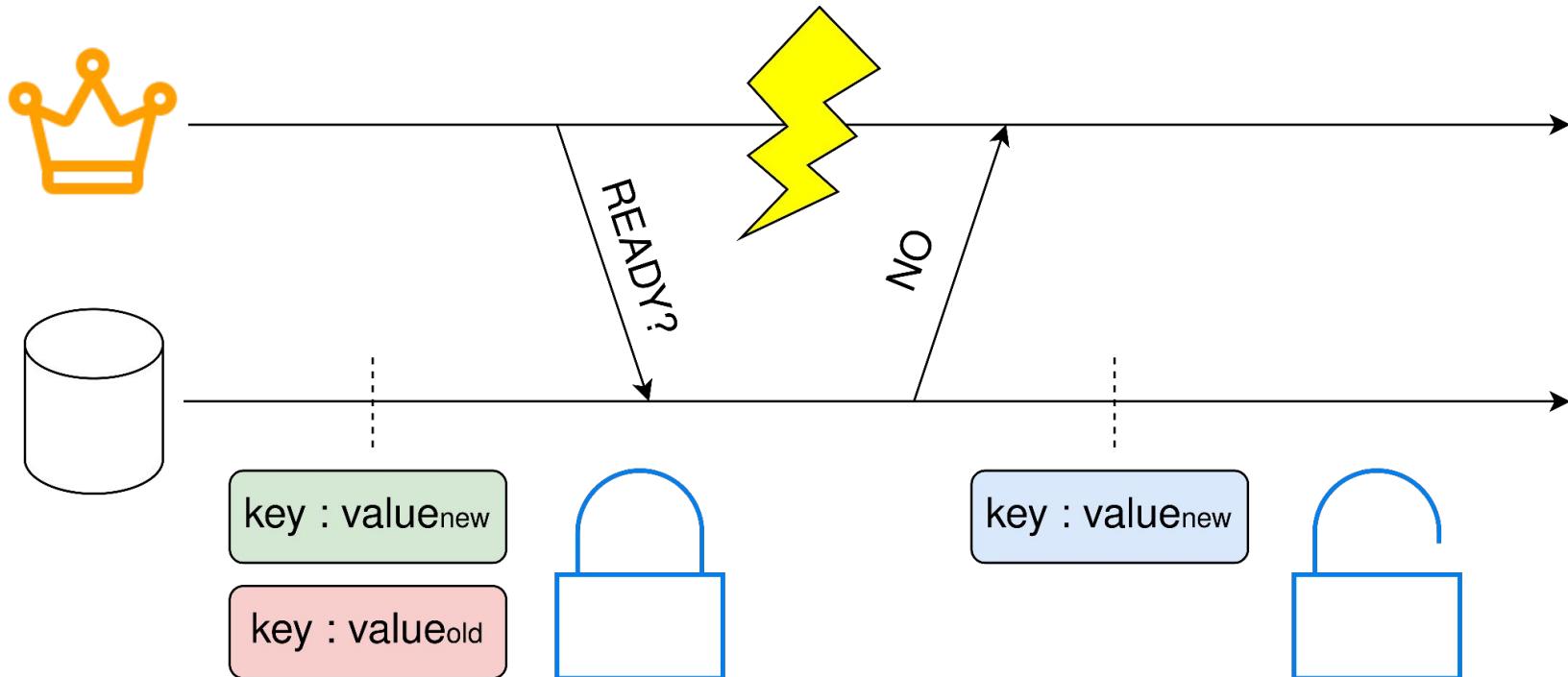
Сбой координатора

- При сбое координатора после положительного ответа, участник не может решить, что делать с транзакцией
- Приходится ждать результата



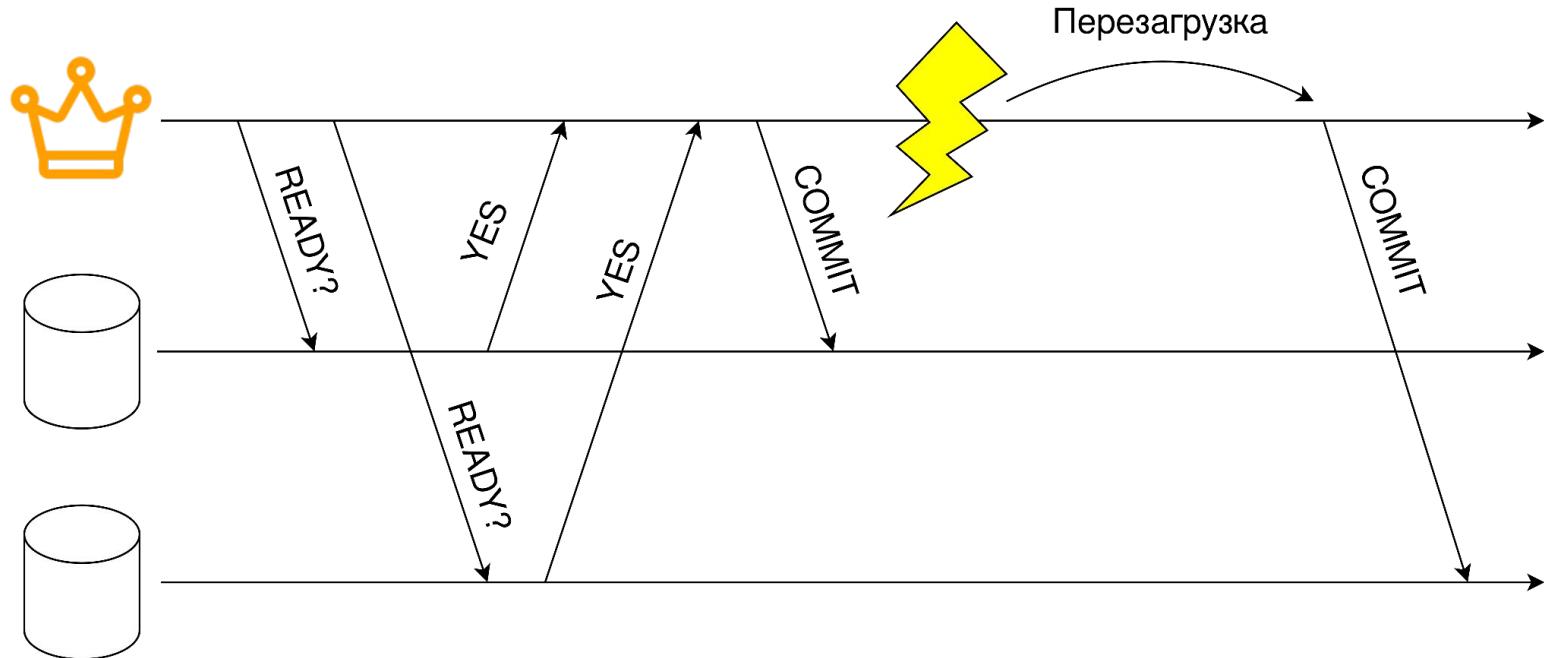
Сбой координатора

- При сбое координатора после отрицательного ответа, участник может откатывать транзакцию



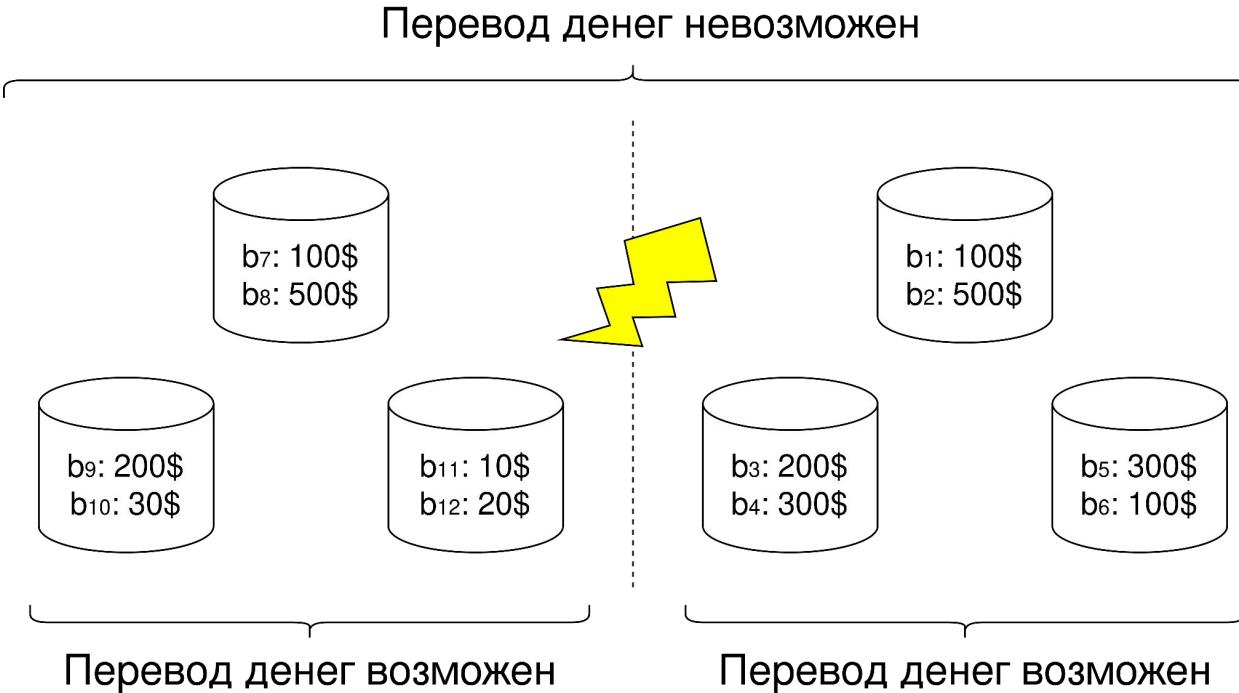
Сбой координатора

- Координатор должен надёжно сохранять транзакции, которые нужно закоммитить
- Чтобы продолжить коммитить даже после сбоя



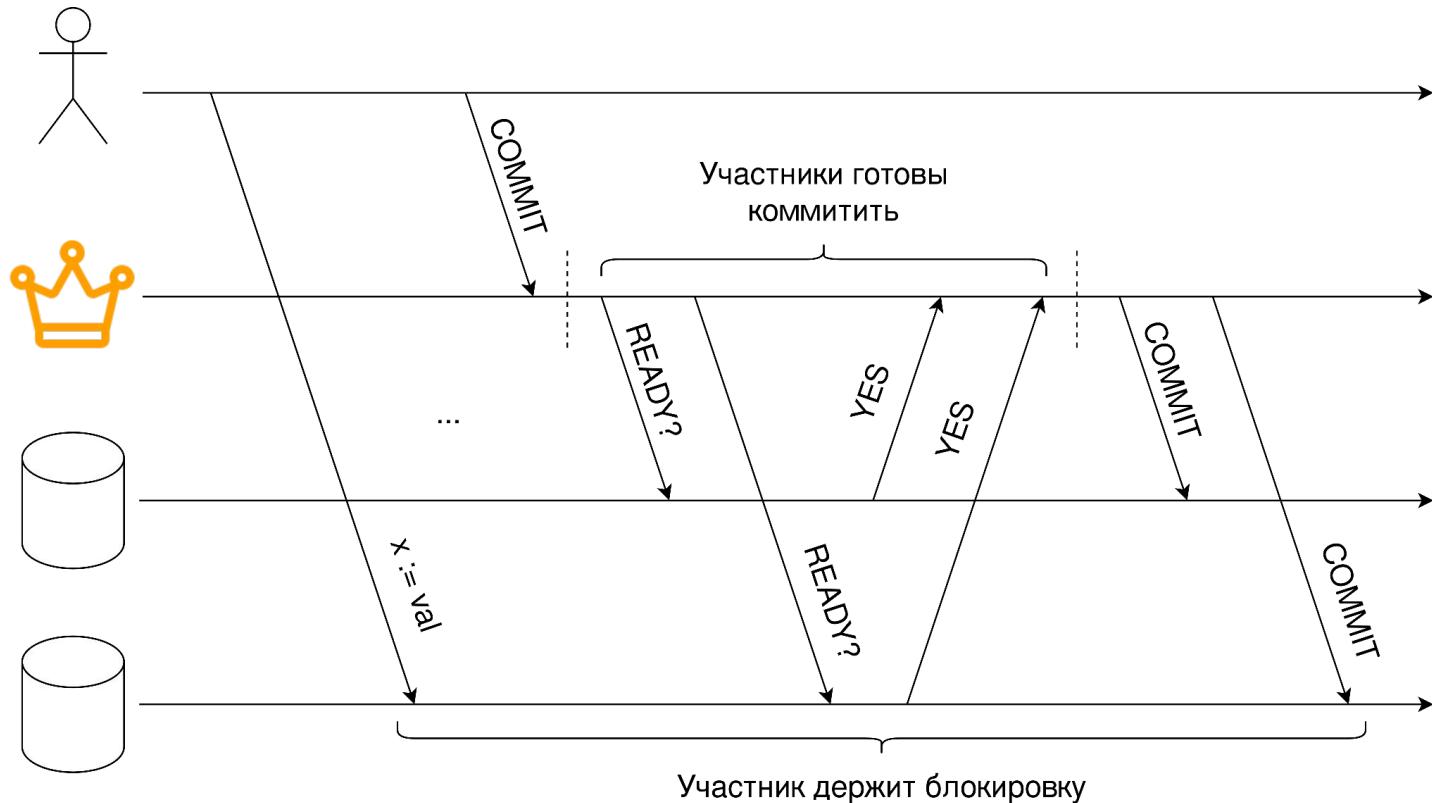
Сбои узлов

- Все участники транзакции должны быть связаны сетью
- Возможно разбиение на несвязанные сегменты сети



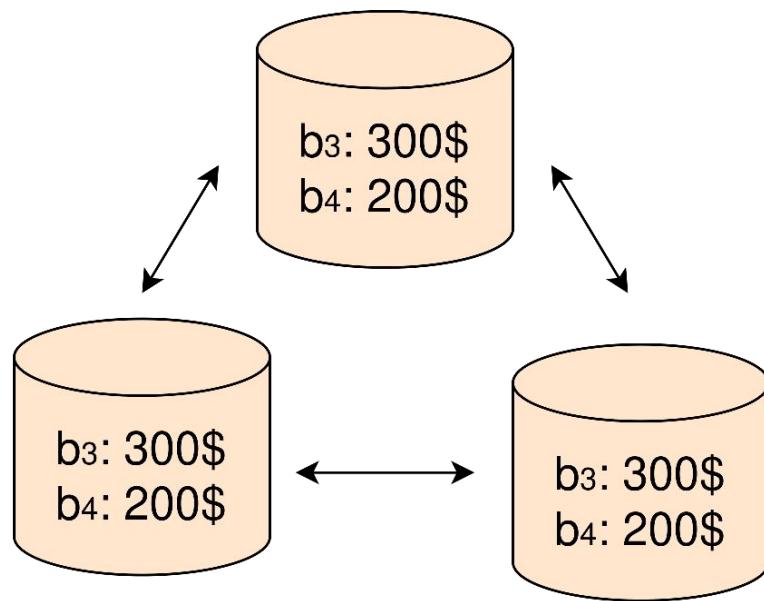
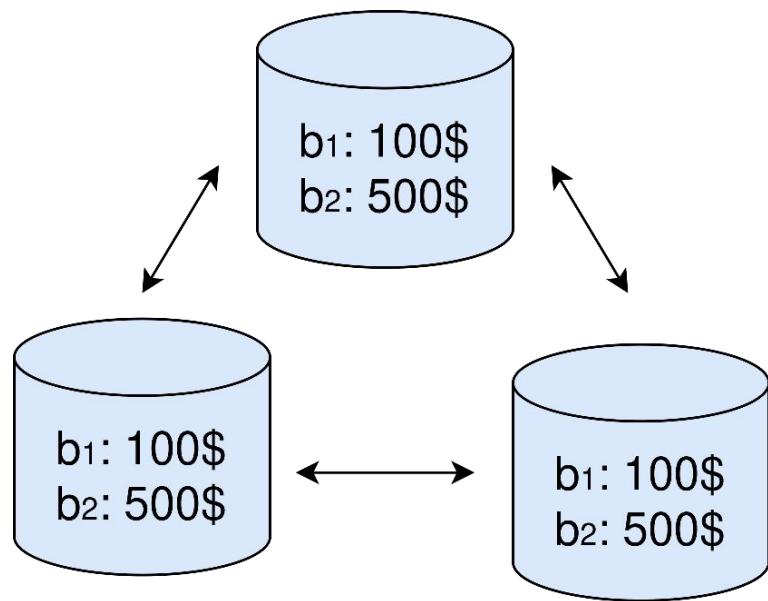
Двухфазный коммит: недостатки

- Участники держат блокировки в течении всей транзакции
- Несколько RTT



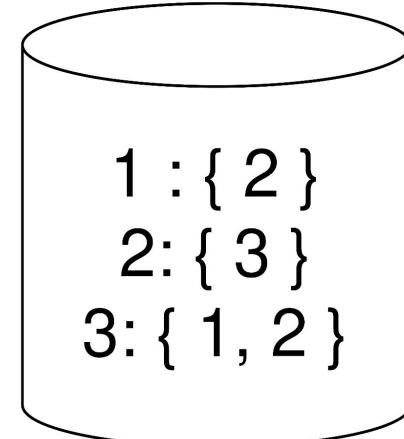
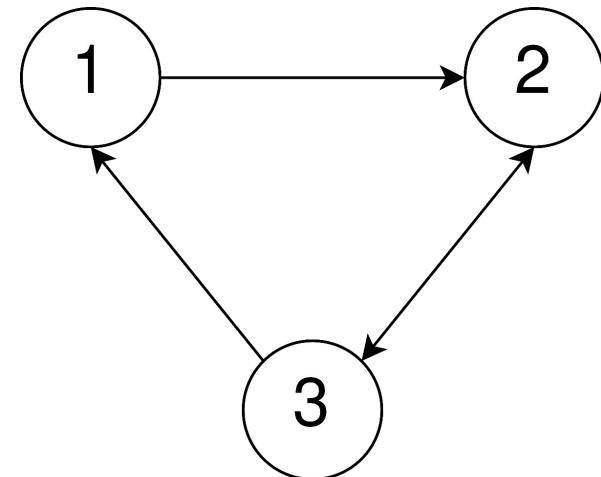
2PC + Репликация

- Каждый участник транзакции может быть набором реплик
- Синхронизируются реплики с помощью Paxos
- Ведут себя как один сервер



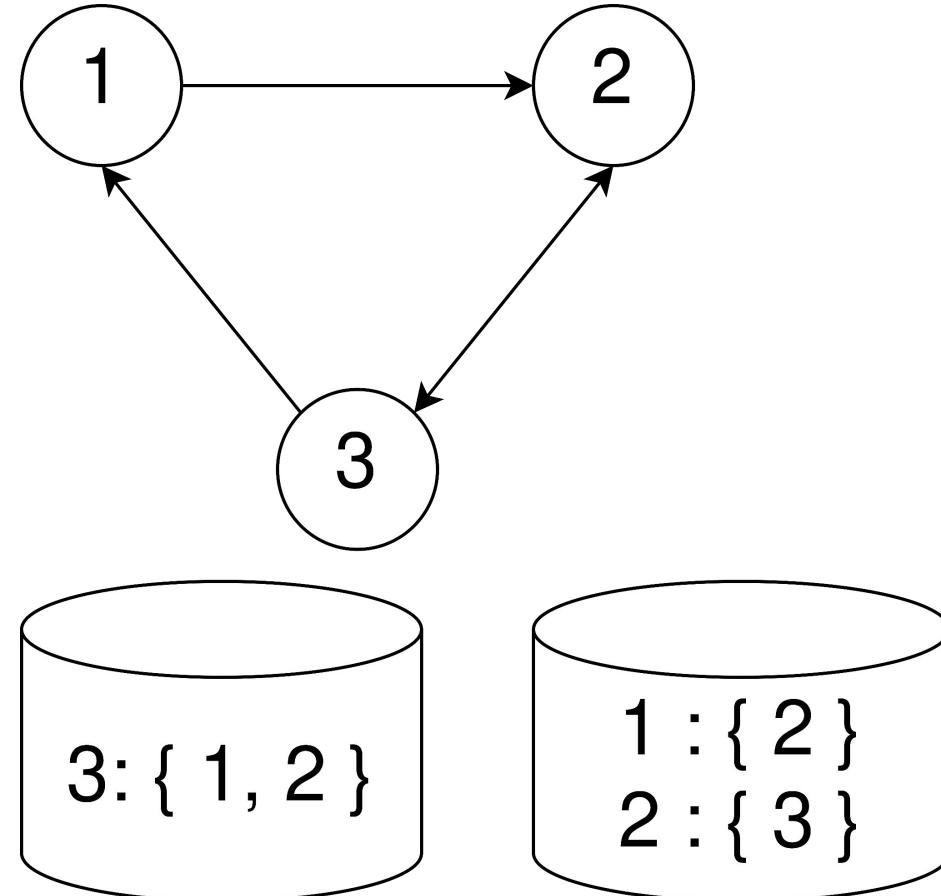
Графовые СУБД

- Вершины - объекты
 - профили людей в социальной сети
- Рёбра - отношения между объектами
 - А → В, если А подписан на В
- Храним граф в виде списков смежности
- В общем случае с рёбрами и вершинами могут быть связаны какие-то данные
 - ФИО пользователя
 - Дата начала подписки



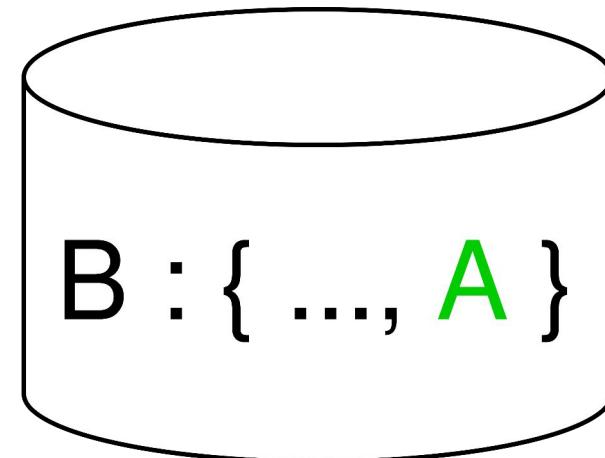
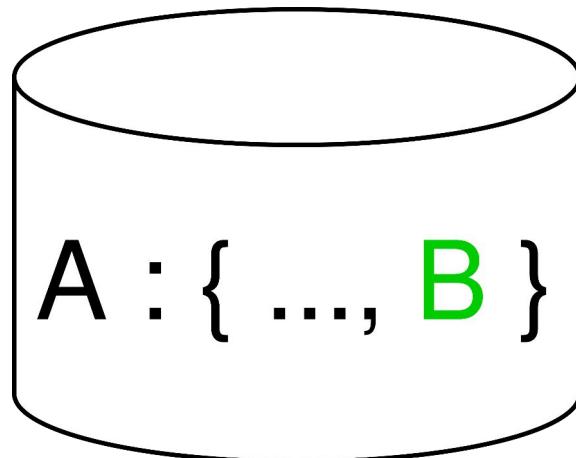
Графовые СУБД: шардирование

- Шардируем по вершинам
- Множество соседей объекта целиком хранится на одном сервере



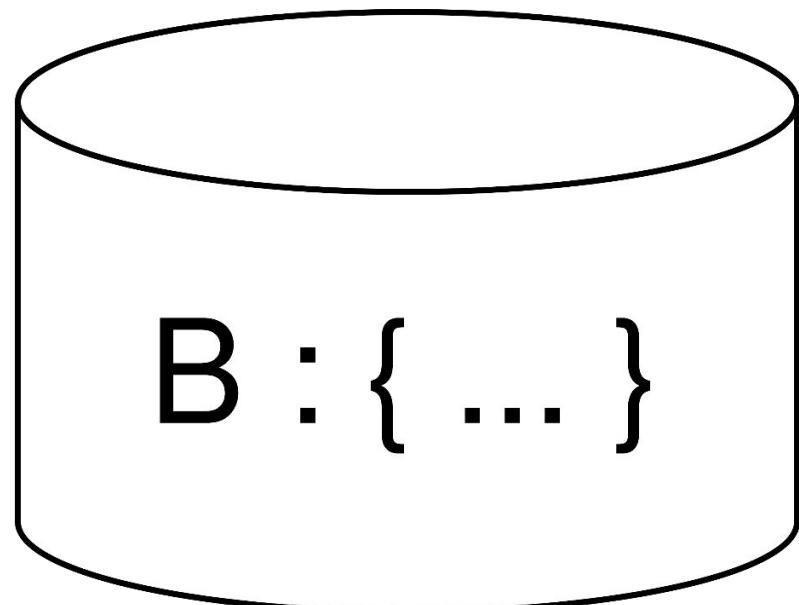
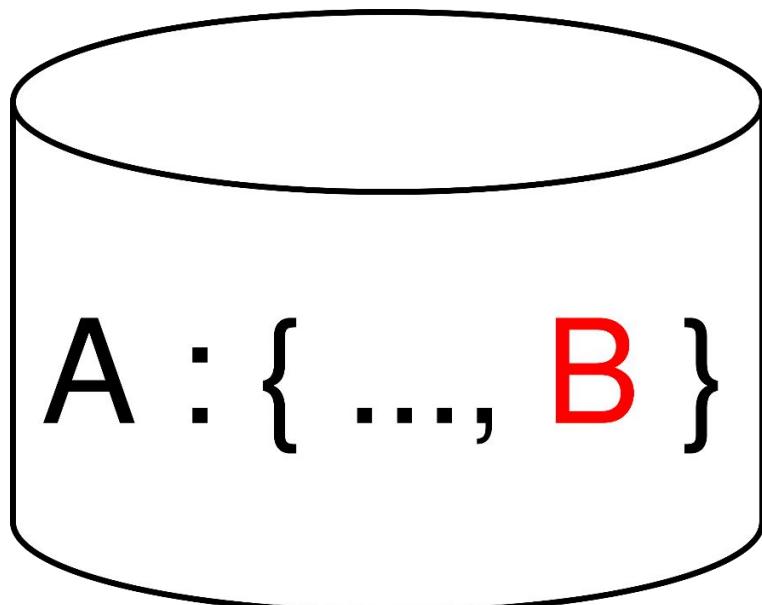
Графовые СУБД: симметричные отношения

- Иногда ребро $A \rightarrow B$ невозможно без обратного ребра
 - Когда эти рёбра описывают симметричное отношение
 - Например, отношение “быть друзьями”
- Когда A и B заключают дружбу, A необходимо добавить в список рёбер, исходящих из B



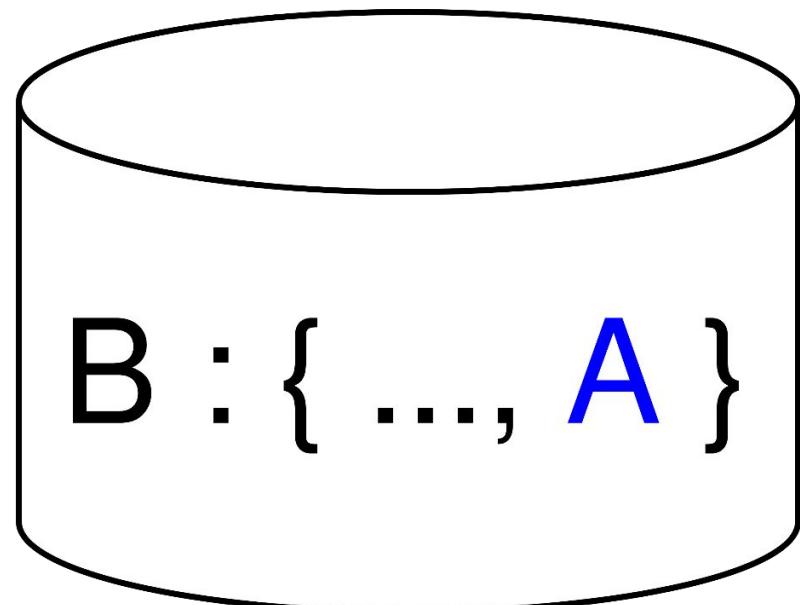
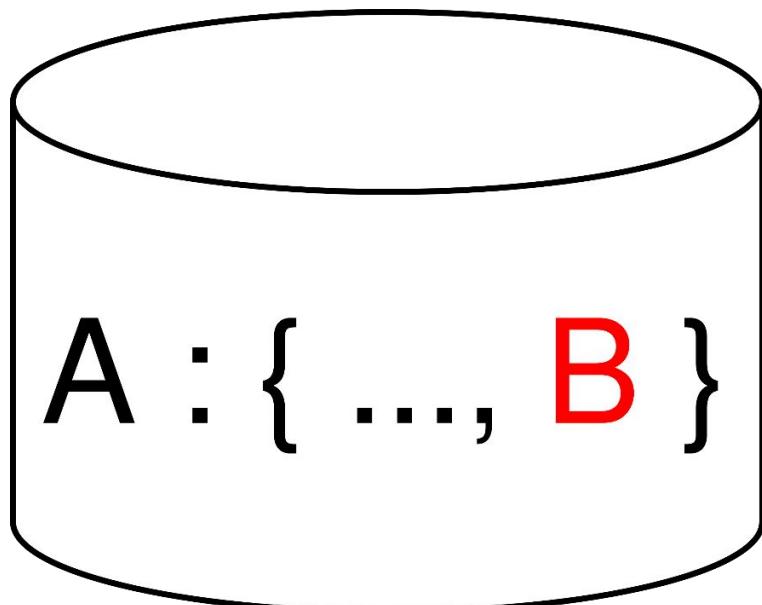
Нетранзакционные обновления

- Добавим ребро A → B на сервер, на котором хранится A
- Пометим его как сломанное



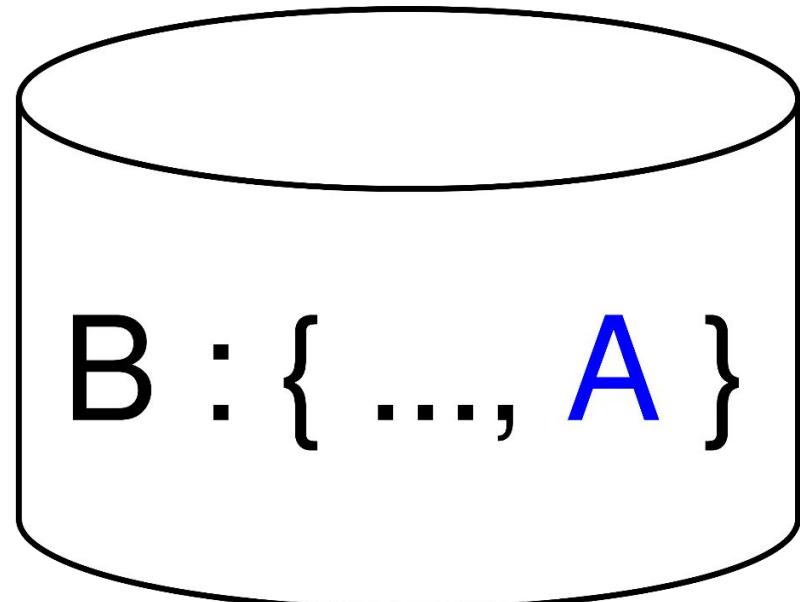
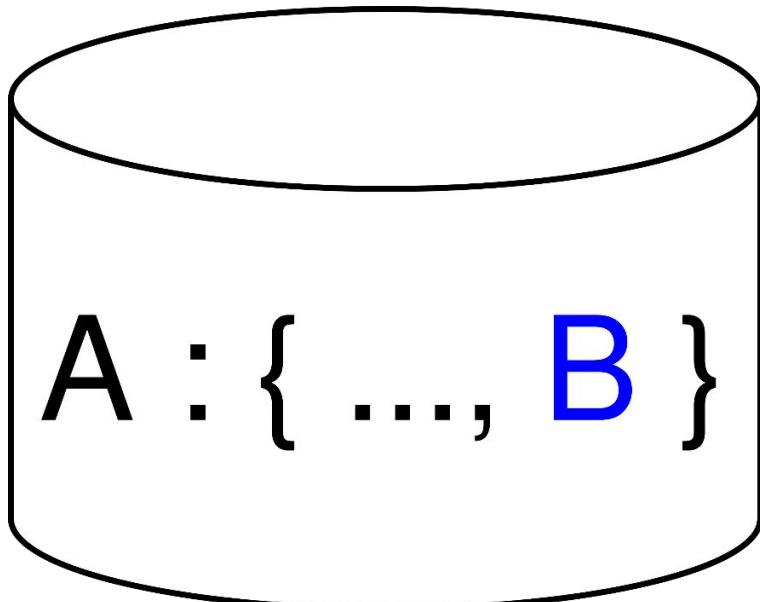
Нетранзакционные обновления

- Добавим ребро $B \rightarrow A$ на сервер, на котором хранится B
- Пометим его как корректное



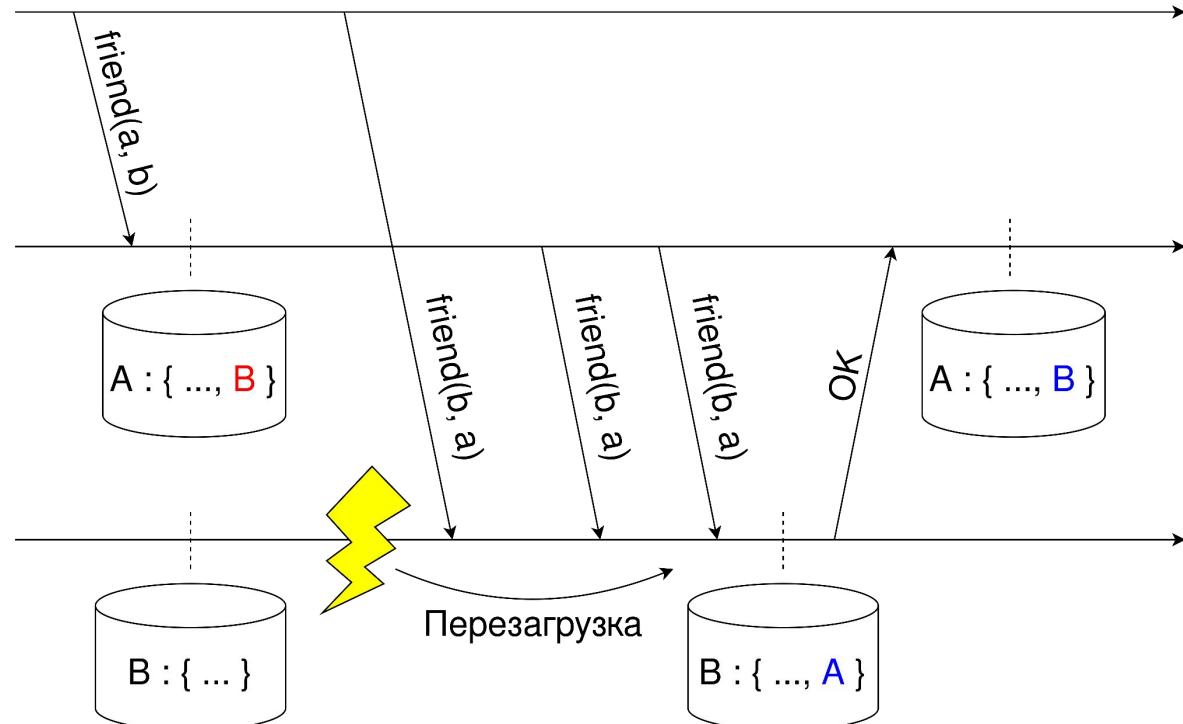
Нетранзакционные обновления

- Пометим ребро A → B как корректное



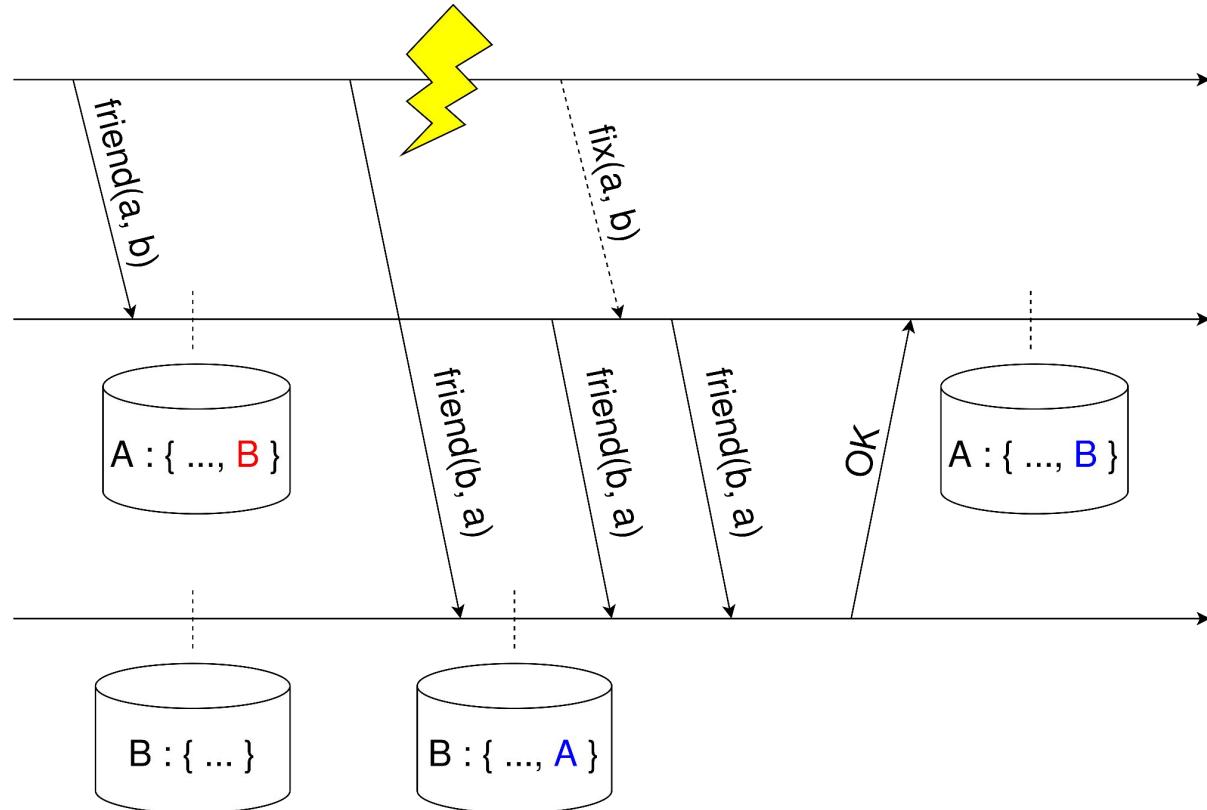
Нетранзакционные обновления

- Фоновый процесс на каждом узле сканирует локальную базу в поисках сломанных рёбер
- И добавляет пару, если нужно



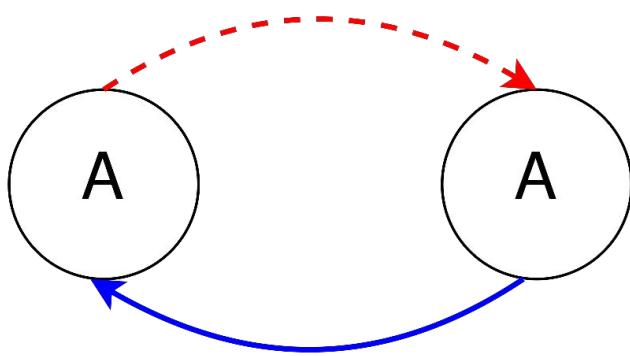
Нетранзакционные обновления

- Добавление ребра должно быть идемпотентной операцией
- Хранилище может повторить добавление парного ребра, если клиент отказал

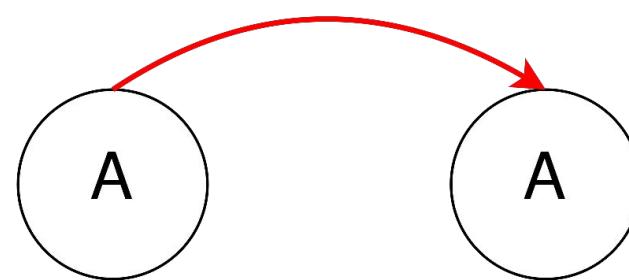


Нетранзакционные обновления

- Либо есть момент, когда парное ребро уже добавлено, а сломанное ребро не помечено как корректное
- Либо когда добавлено ребро, но не добавлено парное



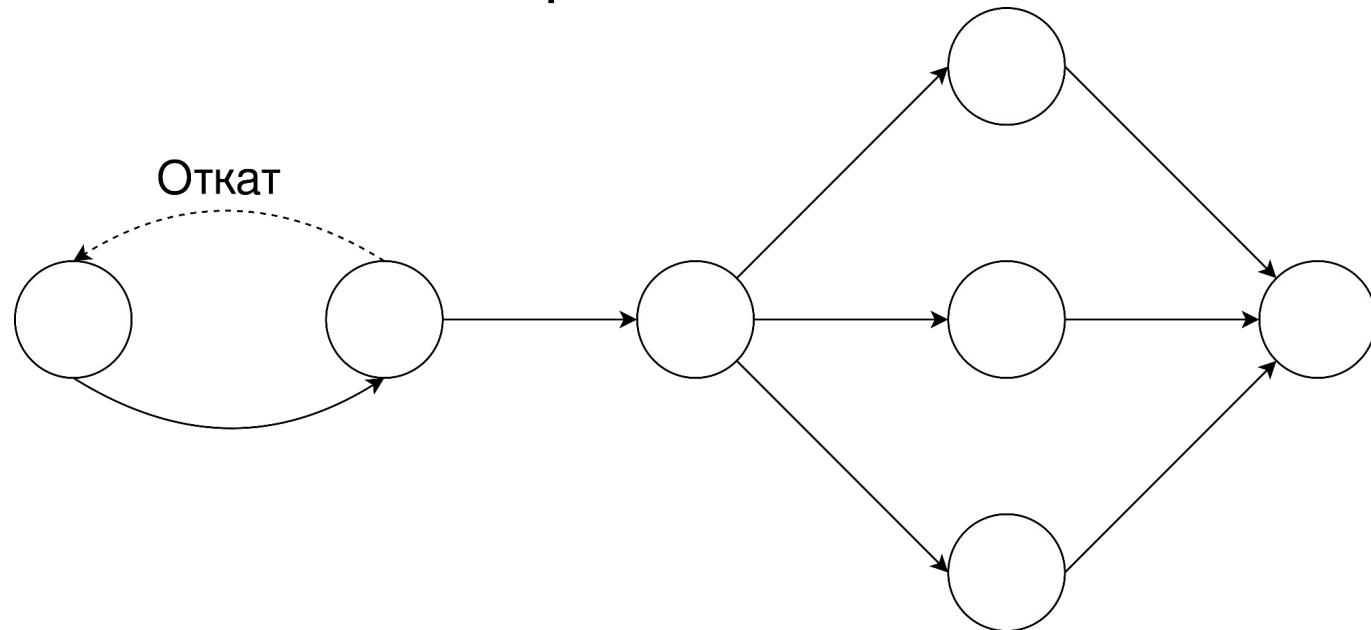
Не показываем
сломанное ребро



Показываем
сломанное ребро

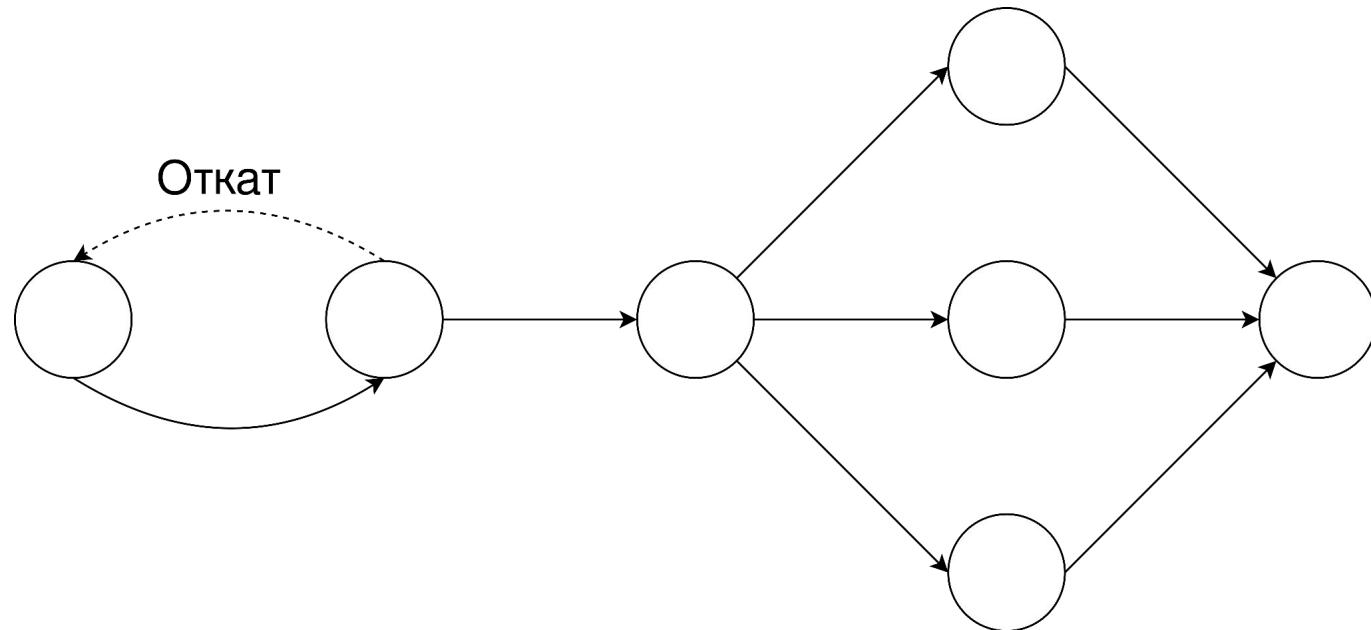
Saga

- Общий вид таких нетранзакционных обновлений
- Некоторые шаги можем повторять до успеха
- Что-то при сбое можем откатывать
- Можно делать какие-то шаги параллельно



Saga

- Некоторым шагам не нужен откат
- DB.Store(NewUUID(), data)

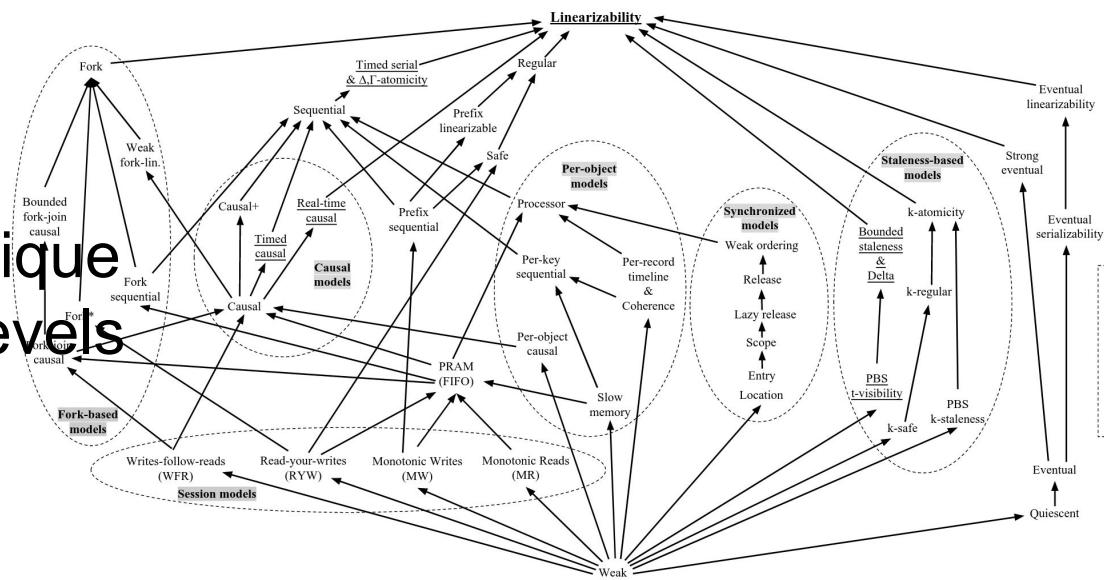


Что почитать: транзакционные и нетранзакционные СУБД

- *Huang D. et al.* TiDB: a Raft-based HTAP database
- *Peng D., Dabek F.* Large-scale incremental processing using distributed transactions and notifications
- *Corbett J. C. et al.* Spanner: Google's globally distributed database
- *Taft R. et al.* CockroachDB: The resilient geo-distributed SQL database
- *Bronson N. et al.* TAO: Facebook's Distributed Data Store for the Social Graph

Что почитать: уровни изоляции

- Jepsen, Consistency models
- Transaction isolation in PostgreSQL
- *Viotti P., Vukolić M.* Consistency in non-transactional distributed storage systems
- *Bailis P. et al.* Highly available transactions: Virtues and limitations
- *Berenson H. et al.* A critique of ANSI SQL isolation levels



Что почитать: разное

- Aksenov V., Kokorin I. et al. Execution of NVRAM Programs with Persistent Stack
- *Mohan C. et al.* ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging
- *Herlihy M. et al.* The art of multiprocessor programming
- *Bernstein P. A., Hadzilacos V., Goodman N.* Concurrency control and recovery in database systems
- Иван Стрелков. Доводим распределённые действия до конца с использованием простейшего паттерна Saga
- Saga pattern in Microservices

Thanks for your attention

