Распределенные системы: непротиворечивость и репликация

Репликация данных

Необходимо для **повышения надежности** и **увеличения производительности**.

Основная проблема — сохранение непротиворечивости данных во всех копиях.

Репликация - за или против?

Плюсы:

• Надежность — дублирование данных.

• Производительность — масштабирование по размеру и географическая.

Минусы:

• Проблемы с непротиворечивостью данных (консистентностью данных).

Пример — устаревание кэша прокси-сервера.

• Проблема масштабируемость vs производительность.

Репликация — как метод масштабирования

Масштабирование ведет к проблемам с производительностью, но размещение реплик данных поблизости от потребителя увеличивает ее.

Для сохранения актуальности копий потребуется дополнительная пропускная способность сети. Кроме того изменения в одной из копий должны приводить к изменениям в других копиях.

Актуальность копий?

Основная проблема в том, что обновления должны проводится, как одна атомарная операция или транзакция.

Фактически означает, что все реплики должны договорится о моменте, когда точно обновлять их локальную копию данных.

А может просто снять ограничения? Например не требовать атомарности и полностью синхронизированных реплик. Зависит от задачи.

Модели непротиворечивости, ориентированные на данные

Непротиворечивость обсуждается в контексте операций чтения-записи над совместно используемыми данными, находящимися в хранилище данных (разделяемая память или распределенная ФС).

Хранилище данных может быть физически разнесено по нескольким машинам. Операция является записью, если она изменяет данные, в противном случае – это операция чтения.

Модель непротиворечивости

По сути — это договоренность между процессами и хранилищем данных, которая гласит:

если процессы согласны соблюдать некоторые правила, хранилище соглашается работать правильно.

Т.е. процесс выполняющий операцию чтения данных ожидает, что операция вернет значение, соответствующее результату последней операции записи данных.

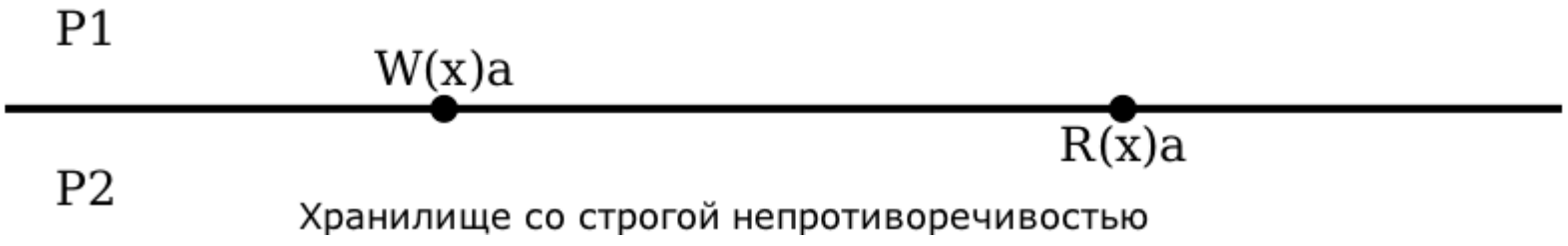
Строгая непротиворечивость (strict consistency)

Это наиболее жесткая модель, которая определяется следующим условием:
всякое чтение данных **X** возвращает значение, соответствующее результату последней записи **X**.

Косвенно подразумевает наличие глобального времени и значительных издержек на блокировки либо нарушение физических принципов.

Нотация

W(x)a — запись процессом в ячейку **x** значения **a**.
R(x)a — чтение процессом из ячейки **x** значения **a**.



Хранилище со строгой непротиворечивостью

Хранилище без строгой непротиворечивости

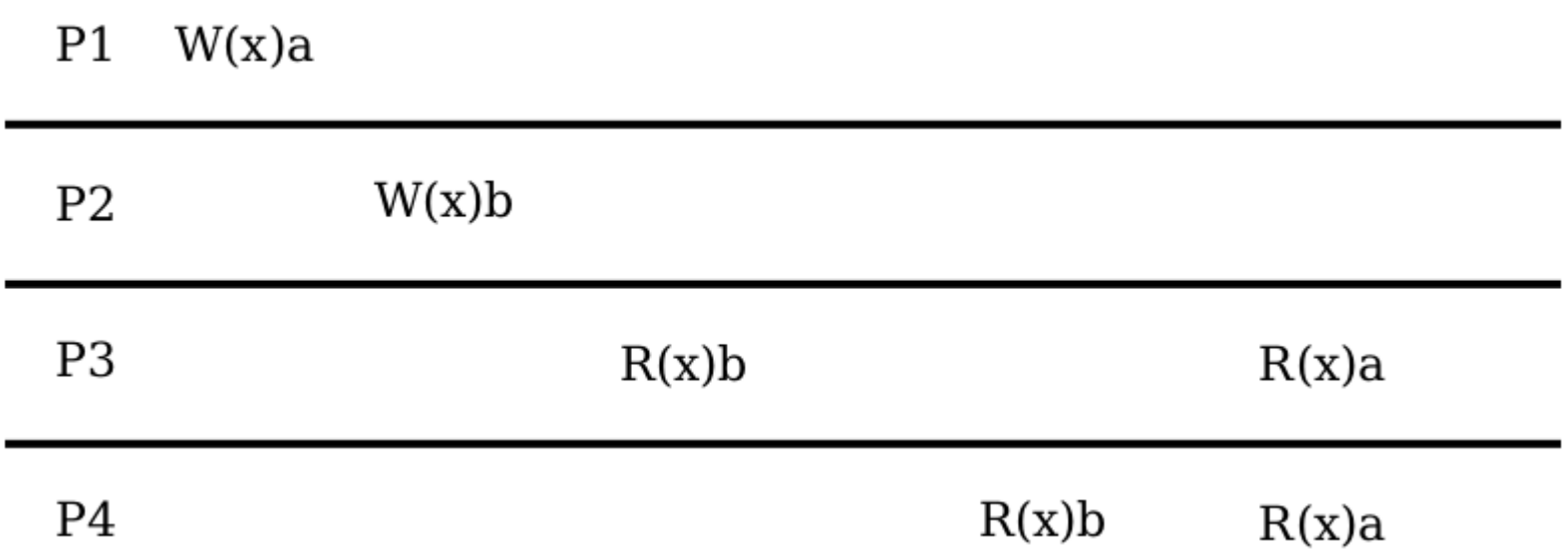
Последовательная непротиворечивость (sequential consistency)

Считается, что хранилище данных последовательно непротиворечиво, если *результат любого действия такой же, как если бы операции чтения-записи всех процессов в хранилище данных выполнялись бы в некотором последовательном порядке, причем операции каждого отдельного процесса выполнялись бы в порядке, определяемом его программой*.

Фактически все процессы видят одинаковую последовательность операций.

Пример: «классическая» работа с критическими областями — правильно написанные параллельные программы не должны делать никаких предположений об относительных скоростях процессов или очередности выполнения инструкций.

Хранилище с последовательной непротиворечимостью



Хранилище без последовательной непротиворечимости

P1	W(x)a			
P2		W(x)b		
P3			R(x)b	R(x)a
P4			R(x)a	R(x)b

Линеаризуемость (linearizability)

Более слабая модель, чем строгая, но более сильная, чем последовательная.

Результат всякого действия такой же, как если бы все операции всех процессов с хранилищем данных выполнялись бы в некотором порядке, причем операция каждого отдельного процесса выполнялась бы в этой последовательности в порядке, определенном в их программах, и, кроме того, если TSop1(x) < TSop2(x), *то операция* op1(x) *в этой последовательности должна предшествовать операции* op2(y).

Пример: при работе параллельных процессов инициализация должна быть раньше вывода

P1: x=1; print(y,z) P2: y=1; print(x,z) P3: z=1; print(x,y)

Причинная непротиворечивость (casual consistency)

Ослабленный вариант последовательной непротиворечивости.

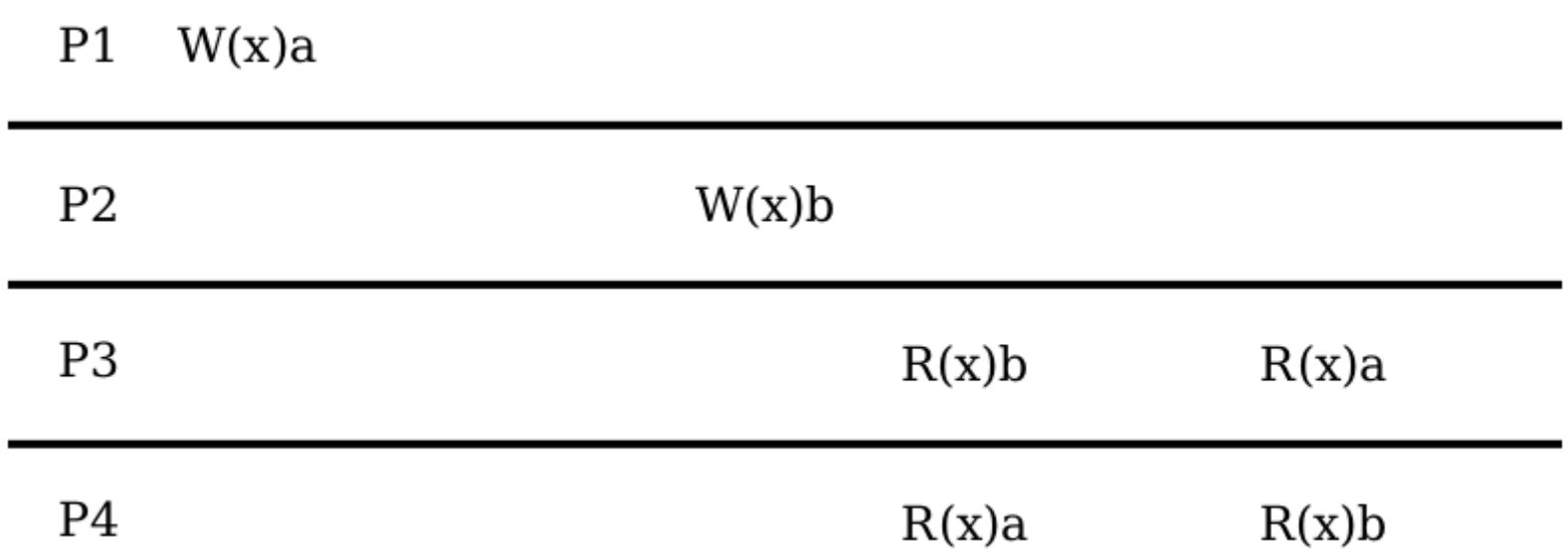
Хранилище поддерживает причинную непротиворечивость, если: *операции записи, которые потенциально связаны причинно-следственной связью, должны наблюдаться всеми процессами в одинаковом порядке, а параллельные операции записи могут наблюдаться на разных машинах в разном порядке*.

Операции не имеющие причинно-следственной связи являются параллельными.

Причинная непротиворечивость

P1	W(x)a		W(x)c	
P2	R(x)a	W(x)b		
P3	R(x)a		R(x)b	R(x)c
P4	R(x)a		R(x)c	R(x)b

Нарушение причинной непротиворечивости хранилища



Нет нарушения причинной непротиворечивости хранилища

Непротиворечивость FIFO

Более слабая модель, чем причинно-следственная.

Операции записи, осуществляемые единичным процессом, наблюдаются всеми остальными процессами в том порядке, в котором они осуществляются, но операции записи, происходящие в разных процессах могут наблюдаться в разных процессах в разном порядке.

Реализация наиболее простая, но практически никаких гарантий порядка записи разных процессов.

Непротиворечивость FIFO

P1	W(x)a					
P2		R(x)a	W(x)b	W(x)c		
P3					R(x)a	R(x)b	R(x)c
P4					R(x)b	R(x)a	R(x)c

А иногда можно сойти с ума. Например, попробовать решить задачу:

P1: x=1; if(y==0) kill(P2)
P2: y=1; if(x==0) kill(P1)

Слабая непротиворечивость (weak consistency)

Не все приложения нуждаются даже в том, чтобы наблюдать запись от других процессов, не говоря уж о том, в каком порядке она сделана.

Например — пиложение использует критическую область для работы с БД, при этом СУБД не знает о ней и может начать репликацию данных в процессе работы приложения с БД.

Слабая непротиворечивость

Можно использовать переменную синхронизации (synchronization variable). Переменная S имеет только одну ассоциированную операцию synchronize(S), которая синхронизирует все локальные копии хранилища данных.

т. е. в ходе синхронизации все локальные операции процесса P распространяются на остальные копии, а операции записи других процессов — на копию данных процесса P.

Свойства модели слабой непротиворечивости

• Доступ к переменным синхронизации, ассоциированным с хранилищем данных, производится на условиях последовательной непротиворечивости.

• С переменной синхронизации не может быть произведена ни одна операция до полного и повсеместного завершения предшествующих ей операций записи.

• С элементами данных не может быть произведена ни одна операция до полного завершения всех операций с переменными синхронизации.

Слабая непротиворечивость

P1	W(x)a	W(x)b	S			
P2				R(x)b	R(x)a	S
P3				R(x)a	R(x)b	S

Допустимая последовательность событий

P1	W(x)a	W(x)b	S		
P2				S	R(x)a

Недопустимая последовательность событий

Свободная непротиворечивость (release consistency)

У слабой непротиворечивости есть проблема: когда осуществляется доступ к переменной синхронизации, хранилище данных не знает — то ли это потому, что завершилась запись совместно используемых данных, то ли началось чтение данных.

Для более эффективной реализации может понадобится два типа синхронизации (и переменных) сообщающих, что происходит вход либо выход из критической области.

Свободная непротиворечивость

Операция *захвата* (acquire) — используется для сообщения хранилищу данных о входе в критическую область, соответственно операция *освобождения* (release) сообщяет о выходе.

Дополнительно могут использоваться *барьеры* — механизм синхронизации, который заставляет дожидаться, пока все процессы не закончат текущую фазу, перед тем как перейти к следующей.

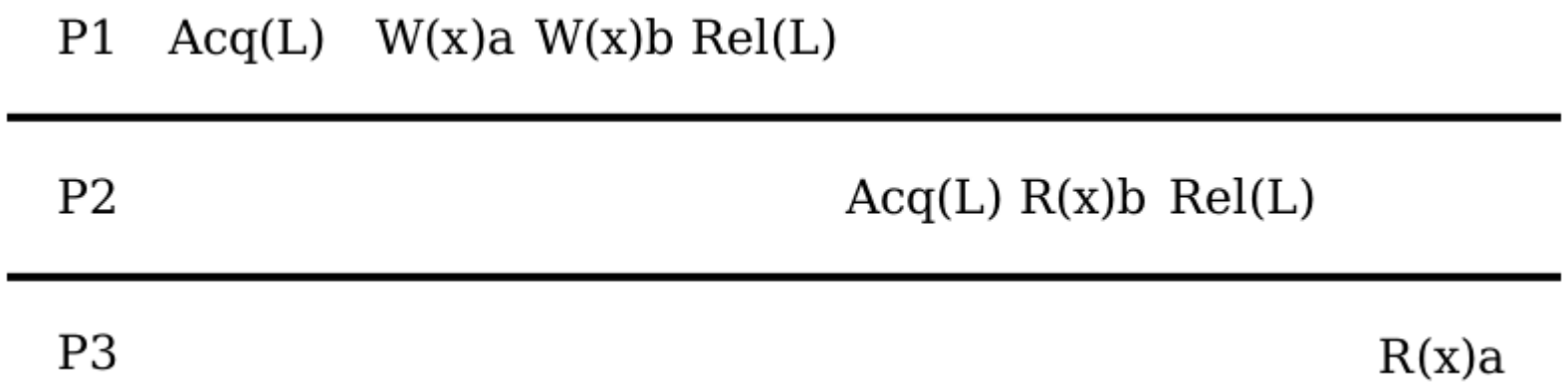
Совместные данные, сохраняющие свою непротиворечивость, являются *защищенными* (protected).

Хранилище данных со свободной непротиворечивостью

Хранилище данных со свободной непротиворечивостью *гарантирует*, что при захвате процессом, хранилище сделает так, что все локальные копии защищенных данных будут актуализированы и станут непротиворечивыми относительно своих удаленных копий. При освобождении, измененные данные будут распространены на другие копии.

Захват *не гарантирует*, что локальные изменения будут разосланы по удаленным копиям. Освобождение не обязательно приведет к импорту изменений из других копий.

Пример последовательности событий при свободной непротиворечивости



Хранилище данных со свободной непротиворечивостью

Хранилище данных является свободно непротиворечивым:

- Перед выполнением операций чтения/записи совместно используемых данных все предыдущие захваты этого процесса должны быть закончены

- Перед освобождением все предыдущие операции чтения/записи этого процесса должны быть закончены

- Доступ к синхронизируемым переменным должен обладать непротиворечивостью FIFO

Ленивая свободная непротиворечивость

Энергичная свободная непротиворечивость (eager release consistency) была описана ранее — проблема в том, что при освобождении данные рассылаются по всем копиям, т. е. по всем *потенциально* заинтересованным копиям.

Ленивая свободная непротиворечивость (lazy release consistency) — в момент освобождения ничего не посылается, вместо этого при захвате происходит актуализация локальных данных. Что позволяет свести к минимуму сетевые операции в некоторых случаях, например когда с данными работает только один процесс в цикле.

Поэлементная непротиворечивость (entry consistency)

В отличие от свободной непротиворечивости, дополнительно требуется, *чтобы каждый отдельный элемент совместно используемых данных был ассоциирован с переменной синхронизации — блокировкой или барьером.*

Такой подход позволяет иметь несколько критических областей, включающих в себя *непересекающиеся* группы совместно используемых элементов данных.

Переменные синхронизации при поэлементной непротиворечивости

Каждая переменная имеет своего текущего владельца— процесс, который захватил ее последним. Владелец может многократно входить в критические области и выходить из них не посылая в сеть сообщений.
Процесс, не являющийся владельцем переменной синхронизации, но желающий захватить ее должен явно затребовать ее у текущего владельца.

Переменной синхронизации могут владеть несколько процессов, но не в эксклюзивном режиме.

Хранилище данных с поэлементной непротиворечивостью

- Захват процессом доступа к переменной синхронизации невозможен, пока не осуществлены все обновления отслеживаемых совместно используемых данных этого процесса
- Пока один процесс имеет эксклюзивный доступ к переменной синхронизации, никакой другой процесс не может захватить эту переменную, в том числе не эксклюзивно
- После эксклюзивного доступа к переменной синхронизации не эксклюзивный доступ любого другого процесса к этой переменной запрещен, пока не будет разрешено владельцем.

Поэлементная непротиворечивость

P1	Acq(Lx)	W(x)a	Acq(Ly)	W(y)b	Rel(Ly)	Rel(Lx)
P2					Acq(Lx)	R(x)a R(y)NIL
P3					Acq(Ly)	R(y)b

Модели непротиворечивости, не требующие операций синхронизации

Непротиворечивость	Описание
Строгая	Абсолютная упорядоченность во времени всех обращений к совместно используемой памяти
Линеаризуемость	Все процессы наблюдают все обращения к совместно используемой памяти в одном и том же порядке. Обращения упорядочены в соответствии с глобальными отметками времени
Последовательная	Все процессы наблюдают все обращения к совместно используемой памяти в одном и том же порядке. Обращения не упорядочены по времени.
Причинная	Все процессы наблюдают все обращения, связанные причинно-следственной связью, к совместно используемой памяти в одном и том же порядке
FIFO	Все процессы наблюдают операции записи любого их процесса в порядке их выполнения. Операции записи различных процессов могут наблюдаться разными процессами в разном порядке

Модели непротиворечивости, использующие операции синхронизации

Непротиворечивость	Описание
Слабая	Совместно используемые данные могут считаться непротиворечивыми только после синхронизации
Свободная	Совместно используемые данные становятся непротиворечивыми после выхода из критической области
Поэлементная	Совместно используемые данные, относящиеся к данной критической области, становятся непротиворечивыми при входе в эту область

Модели непротиворечивости, ориентированные на клиента

Здесь рассматривается специальный класс распределенных хранилищ, которые характеризуются отсутствием одновременных изменений либо легкостью их разрешения.

Потенциальная непротиворечивость

Когда параллельность нужна в урезанном виде — большинство процессов не производят изменения данных (СУБД, DNS, WWW, VCS и т. д.). Ситуация параллельной записи не возникает — требуется решать конфликт чтения-записи.

Такие хранилища характеризуются нечувствительностью к высокой степени нарушения непротиворечивости реплик. Обычно в них долгое время не происходит изменения данных и все реплики становятся непротиворечивыми.

Свойство потенциально непротиворечивого хранилища данных

В отсутствие изменений все реплики постепенно становятся идентичными.

Фактически требуется только, чтобы изменения гарантированно расходились по всем репликам. Конфликты записи легко решаемы, т. к. вносить изменения может сравнительно небольшая группа процессов, поэтому реализация потенциальной непротиворечивости как правило дешева.

Потенциально непротиворечивые хранилища данных работают только, если клиент всегда осуществляет доступ к одной и той же реплике!!!

Пример проблемы: мобильный пользователь, который подключается к различным репликам распределенного хранилища.

Острота проблемы снижается за счет введения *непротиворечивости, ориентированной на клиента (client-centric consistency).*

Предоставляются гарантии непротиворечивости только одному клиенту, относительно параллельного доступа других клиентов никаких гарантий не предоставляется.

Нотации для моделей непротиворечивости, ориентированных на клиента

$\mathbf{x_i[t]}$ — версия элемента данных **x** на локальной копии $\mathbf{L_i}$ в момент времени **t**.

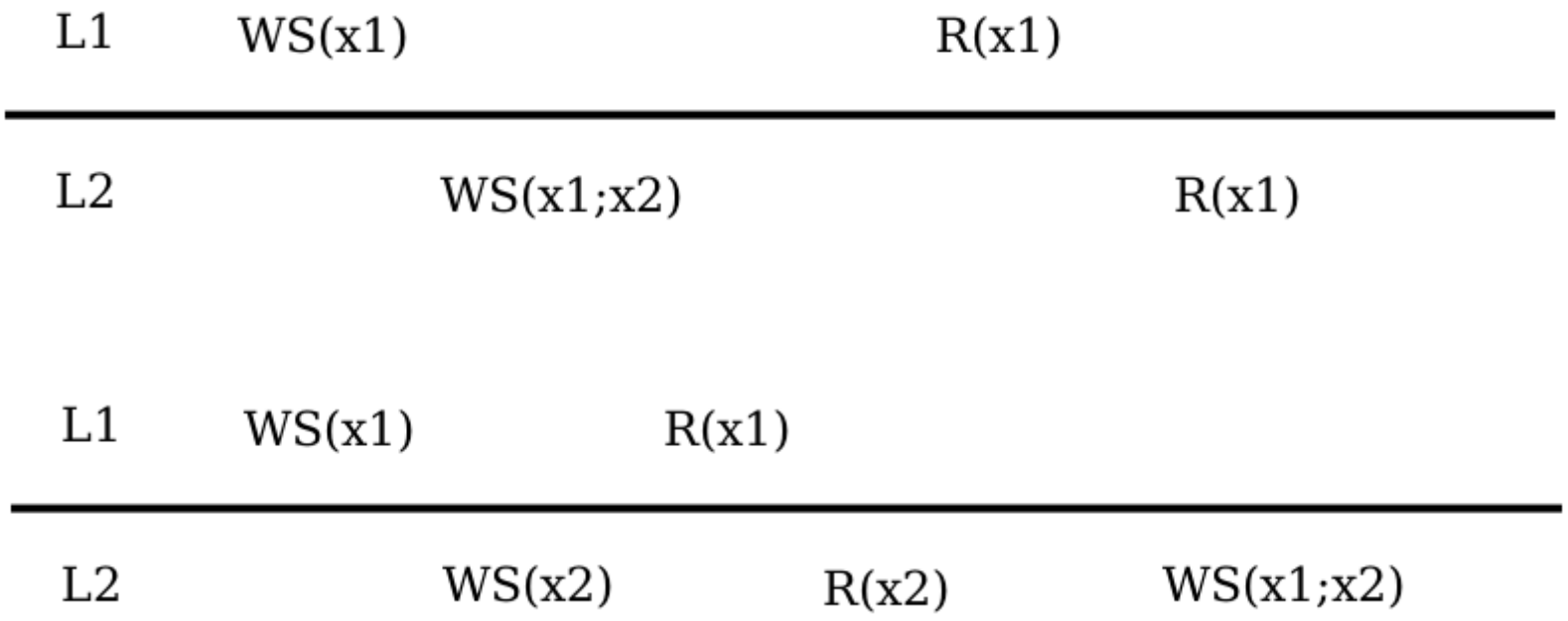
Версия $\mathbf{x_i[t]}$ — результат серии операций записи, произведенных в $\mathbf{L_i}$ после инициализации. Эта серия обозначается, как $\mathbf{WS(x_i[t])}$. Если операции из серии $\mathbf{WS(x_i[t1])}$ также были произведены в локальной копии $\mathbf{L_j}$ в более позднее время **t2**, то запись выглядит, как $\mathbf{WS(x_i[t1];x_j[t2])}$

Монотонное чтение

Хранилище данных обеспечивает непротиворечивость монотонного чтения, если удовлетворяет условию:

Если процесс читает значение элемента данных **x**, любая последующая операция чтения **x** всегда возвращает то же самое или более позднее значение.

Операции чтения, осуществляемые одиночным процессом над двумя различными копиями одного хранилища данных — хранилище с непротиворечивостью монотонного чтения и без



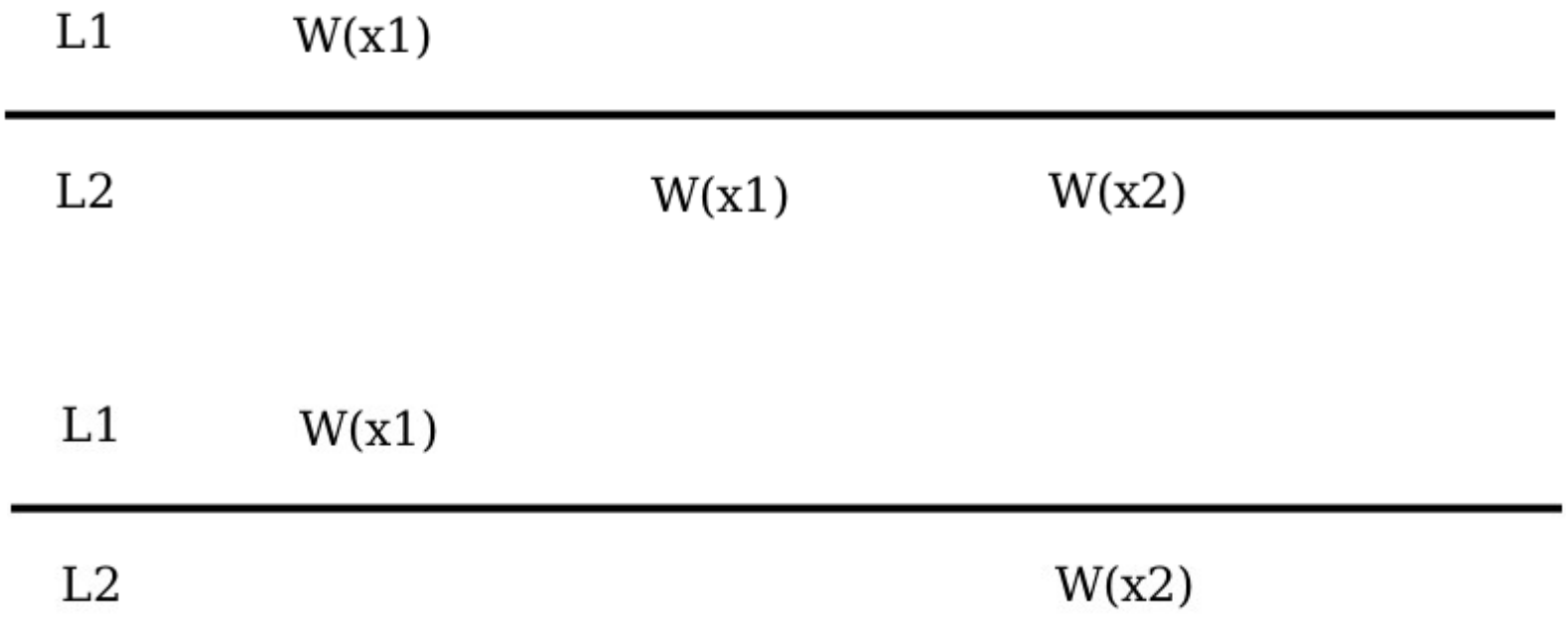
Монотонная запись

Часто важно, чтобы по всем копиям хранилища данных в правильном порядке распространялись операции записи.

Хранилище обладает свойством *непротиворечивости монотонности записи (monotonic-write consistency)*, если операция записи процесса в элемент данных **x** завершается раньше любой из последующих операций записи этого процесса в элемент данных **x**.

Очень похожа на непротиворечивость FIFO. Актуализация копии **x** не обязательна, если каждая операция записи полностью изменяет значение **x**.

Операции записи, осуществляемые одиночным процессом над двумя различными копиями одного хранилища данных — хранилище с непротиворечивостью монотонной записи и без



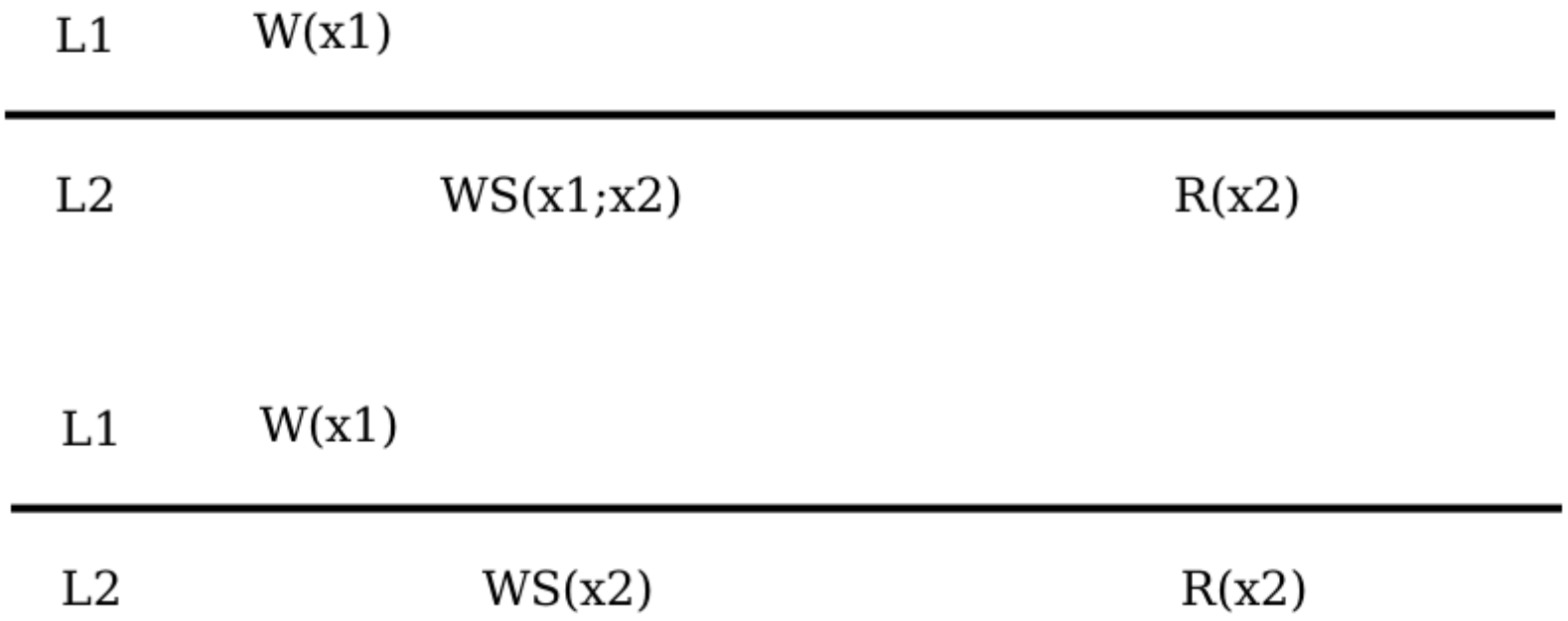
Чтение собственных записей

Хранилище обладает свойством *непротиворечивости чтения собственных записей (read-your-writes consistency)*, если результат операций записи процесса в элемент данных **x** всегда виден последующим операциям чтения **x** этого же процесса.

т. е. операция записи всегда завершается раньше следующей операции чтения этого же процесса.

Пример отсутствия: изменения web-странички и кэш.

Хранилище данных с непротиворечивостью чтения собственных записей и без

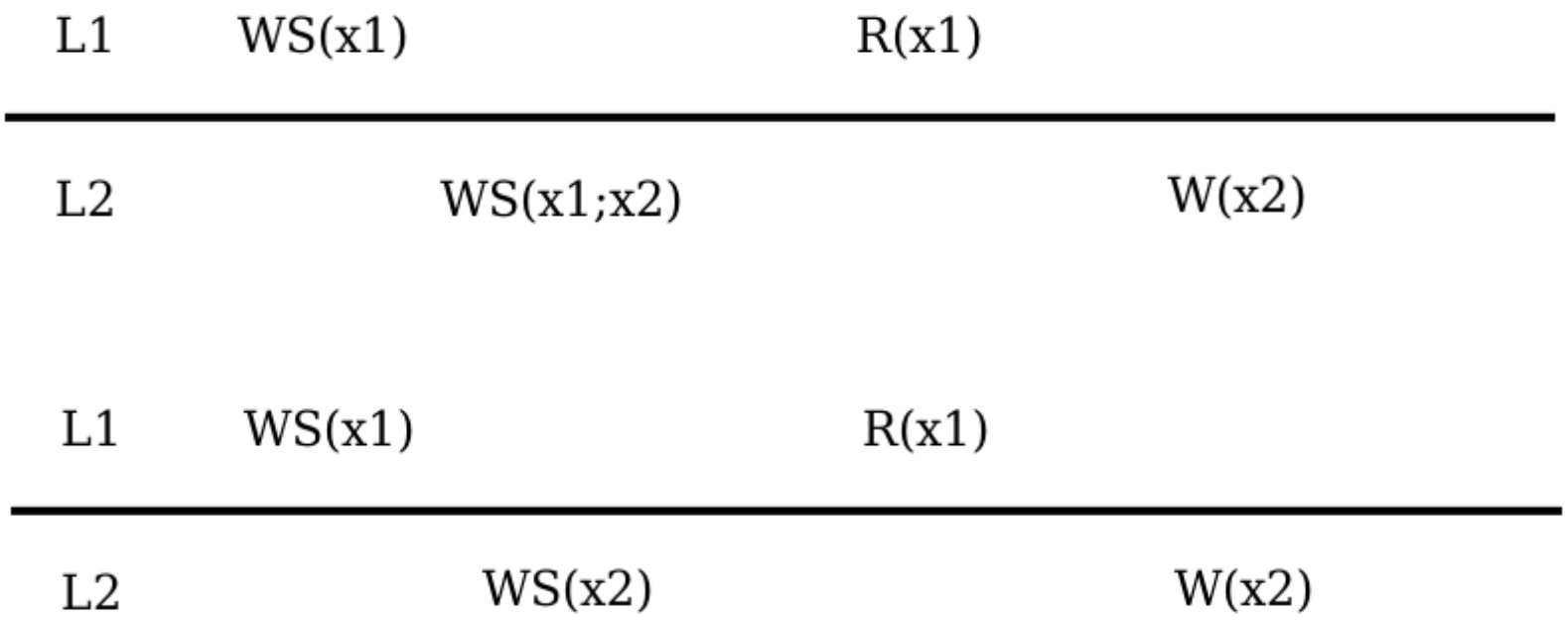


Запись за чтением

Хранилище данных обеспечивает *непротиворечивость записи за чтением (writes-follow-reads consistency)*, если операция записи в элемент данных **x** процесса, следующая за операцией чтения **x** того же процесса, гарантирует, что будет выполняться над тем же самым или более свежим значением **x**, которое было прочитано предыдущей операцией.

Пример: письма с ответами сохраняются в локальной базе только позже оригинального письма.

Хранилище данных с непротиворечивостью записи после чтения и без



Протоколы распределения

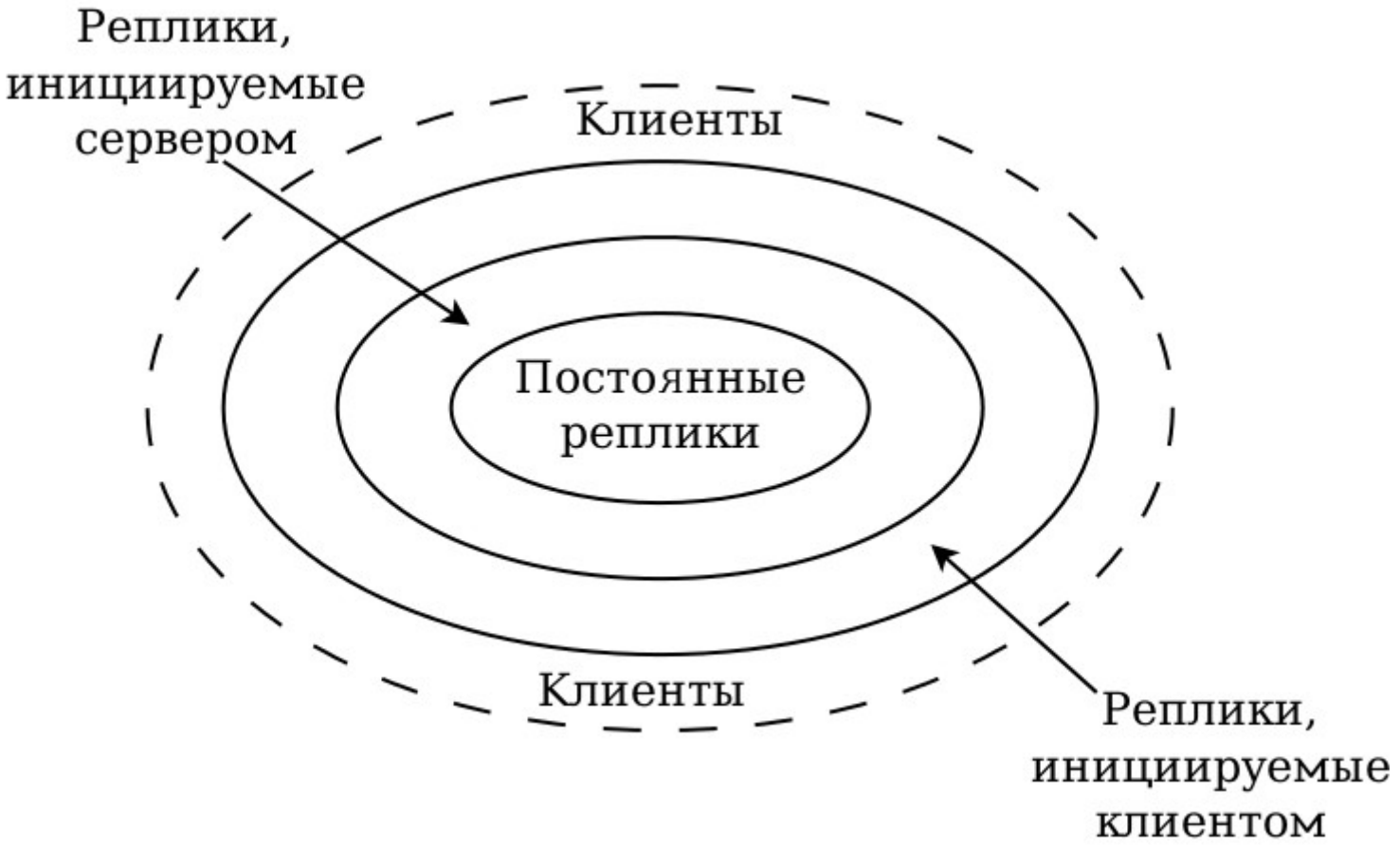
Типы копий

Постоянные реплики – можно рассматривать как исходный набор реплик, образующих распределенное хранилище (кластерное хранилище)

Реплики инициируемые сервером – копии, которые создаются для повышения производительности и создание которых инициируется хранилищем данных – иногда называют *выдвинутым кэшем*.

Реплики инициируемые клиентом – локальное ус-во хранения данных, используемое для клиентом для временного хранения копии запрошенных данных – *клиентский кэш*.

Размещение реплик



Распространение обновлений

Состояние против операций:

- Распространяется только извещение об обновлении*
- Передаются данные из одной копии в другую**
- Распространяется операция обновления по всем копиям — активная репликация.

* Извещения распространяются по *протоколам несостоятельности (invalidation protocols)* — копии информируются о наличии обновления, а также, что копии данных стали неправильными.

** Применяется, когда отношение операций чтения к операциям записи относительно велико

Распространение обновлений

Продвижение против извлечения:

• Продвижение с использованием push-based протоколов — обновления распространяются по другим репликам, не ожидая запросов на обновление.

• При извлечении (pull-based approach) — сервер или клиент обращается с запросом к другому серверу, требуя отправить все доступные обновления.

Смешанная форма распространения обновлений

Аренда (lease) — обещание сервера определенное время поставлять обновление клиенту.
Когда срок аренды истекает, клиент запрашивает у сервера обновления и/или явно продлевает аренду.

Типы аренды:

- Основанная на возрасте элемента данных
- Основанная на частоте обновления клиентом копии в своем кэше
- Основанная на объеме дискового пространства, затрачиваемого сервером на сохранение состояния

Распространение обновлений

Метод рассылки обновлений:

- Целевой (unicasting)
- Групповая рассылка (multicasting)

Эпидемические протоколы

Эпидемические протоколы не разрешают конфликтов обновления, вместо этого они ориентированы на то, чтобы распространить обновление при помощи как можно меньшего количества сообщений.

Для упрощения предполагается, что обновления инициируются одним сервером.

Эпидемические протоколы: терминология

Сервер является *инфицированным (infected)*, если на нем хранится обновление, которое он готов распространять на другие серверы.

Сервер, который еще не получил обновления, называется *восприимчивым (susceptible)*.

Сервер, данные которого были обновлены, но не готовый либо неспособный их распространять, называется *очищенным (removed)*.

Модели распространения обновлений: антиэнтропия

Сервер P случайным образом выбирает другой сервер Q, после чего обменивается с ним обновлениями.

Существует 3 способа обмена обновлениями:

- Сервер P только продвигает свои обновления на сервер Q
- Сервер P только извлекает новые обновления с сервера Q
- Серверы P и Q пересылают обновления друг-другу

При продвижении обновления — обновления имеют небольшую скорость распространения.

Для увеличения скорости распространения для любого метода можно использовать прямое продвижение обновления на несколько серверов. Специальный вариант подобного подхода называется *распространением слухов (rumor spreading)* или *болтовней(gossiping)*: если на сервере P есть обновленный элемент данных X, то он пытается продвинуть обновление на случайный сервер Q. Если сервер Q уже получил это обновление с другого сервера, то P с вероятностью 1/k теряет интерес к распространению обновлений и становится очищенным.

Эпидемические протоколы: удаление данных

Побочный эффект: затрудненное распространение удалений элементов данных, если удаляется вся информация об этом элементе.

Т.о. необходимо использовать запись о произведенном удалении, например с помощью рассылки свидетельств о смерти (death certificates). Проблема в том, что свидетельства также необходимо когда-то удалять, чтобы серверы не накапливали БД об удалении элементов данных.
Частично проблему решает спящее свидетельство о смерти — снабжается отметкой времени и живет в течении времени максимального распространения обновлений.