Распределенные вычислительные системы

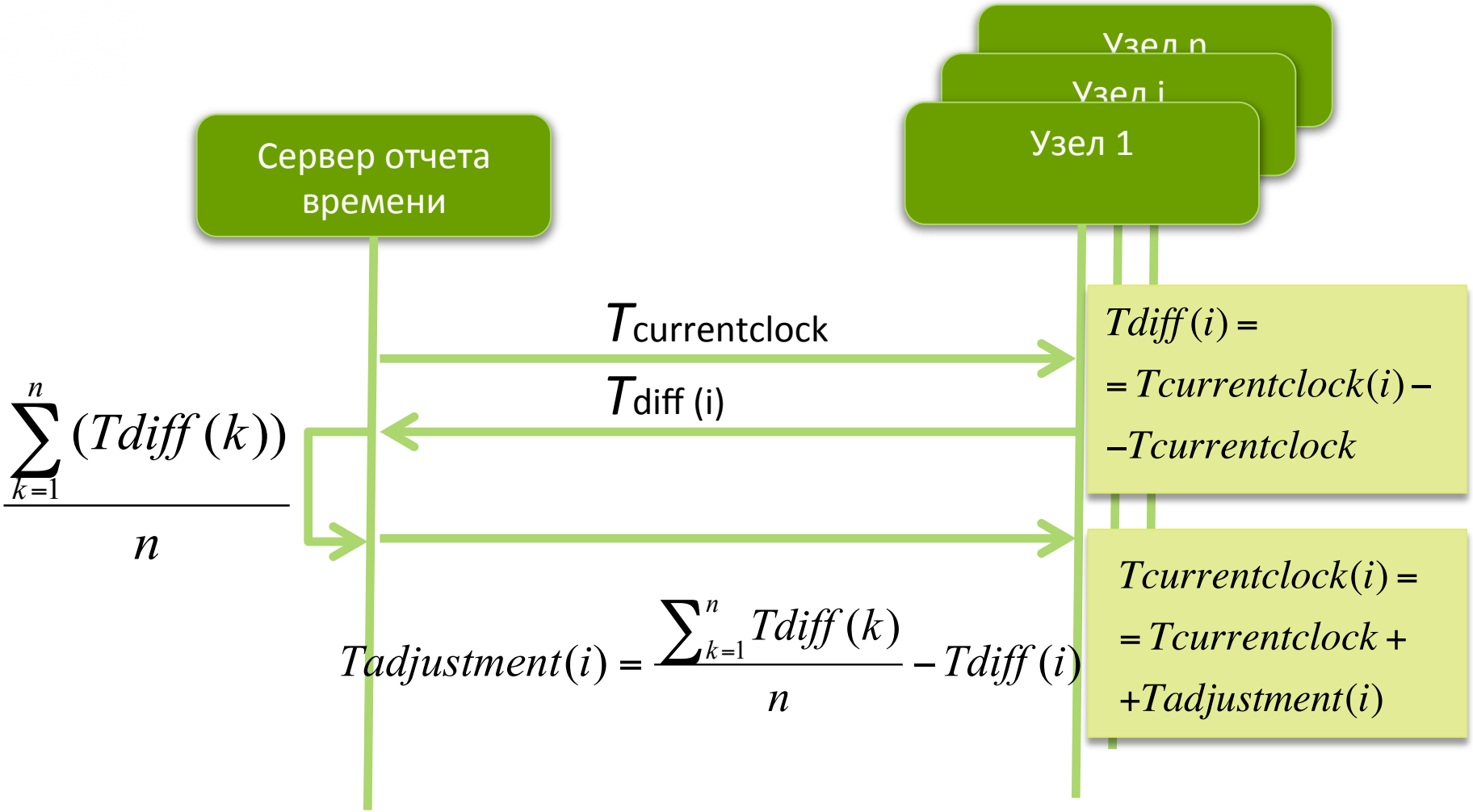
Синхронизация. Согласованность данных

Синхронизация времени в РВС

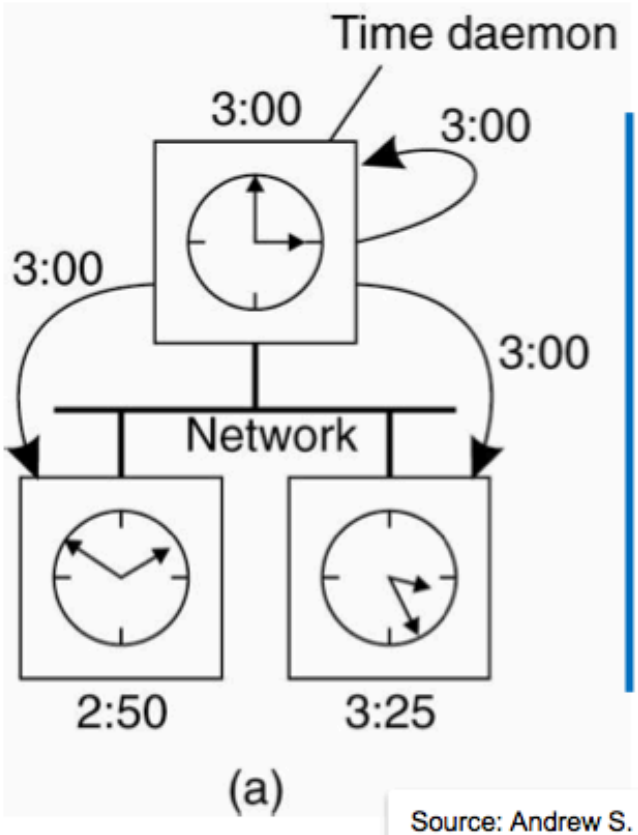
Проблема времени

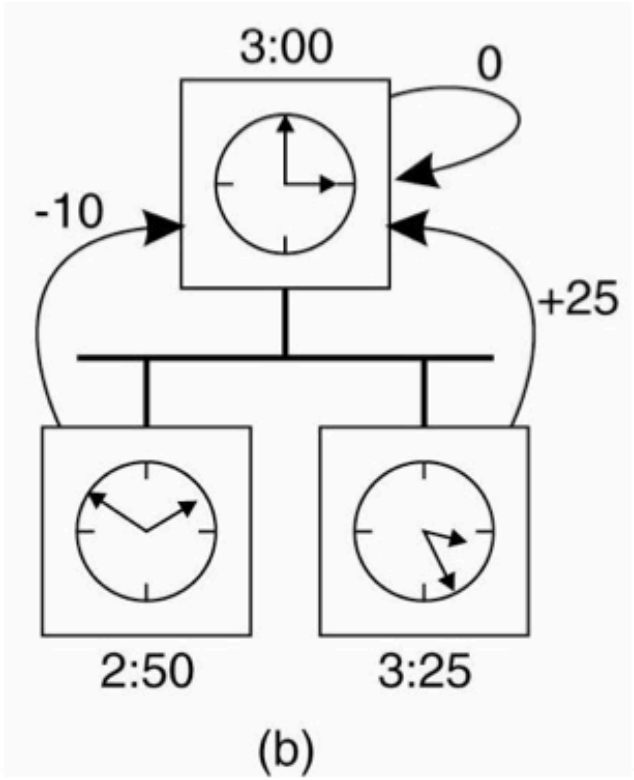
- В РВС невозможно (чрезвычайно затратно) гарантировать одинаковое время во всех узлах, в связи с проблемой «дрейфа часов».
- Для достижения адекватной степени ошибок при работе, РВС используют механизмы синхронизации времени.
- Но как добиться того, чтобы время не дрейфовало вместе с «главными часами»?

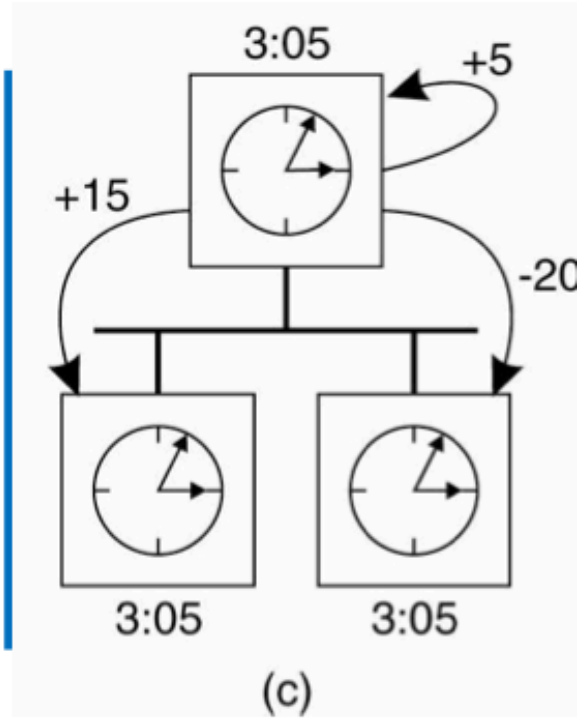
Алгоритм Беркли



Алгоритм Беркли





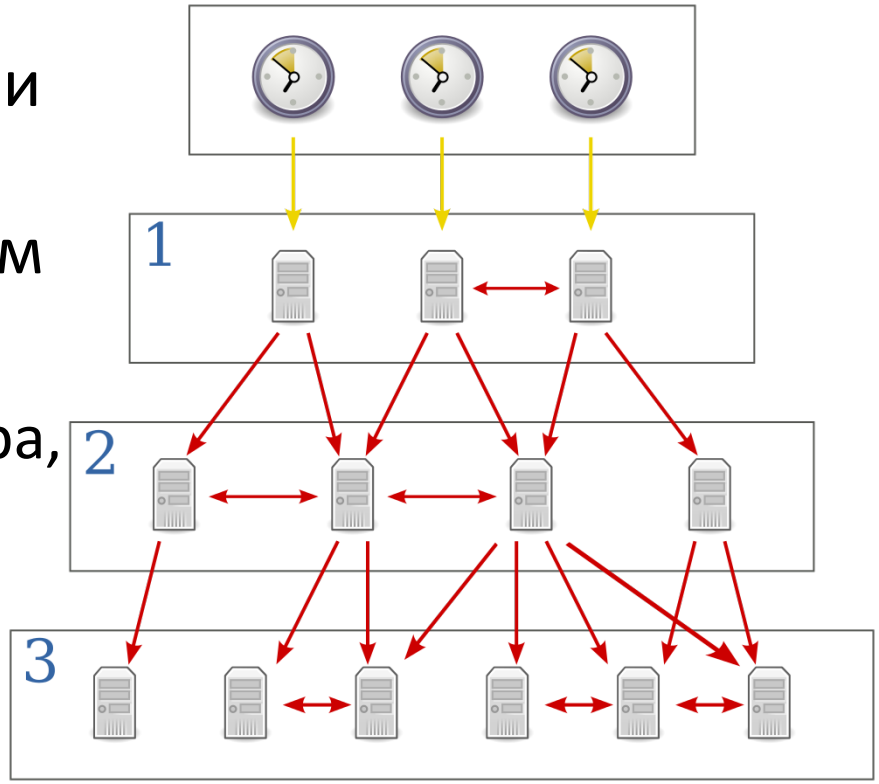


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Иерархическая инфраструктура NTP

- Network Time Protocol (протокол сетевого времени)

- Слой 0: высокоточные приборы счета времени (эталонные часы)
- Слой 1: узлы, к которым подключены эти часы
- Слой 2: основные сервера, получающие время по протоколу NTP
- Слой 3: получают время от серверов слоя 2



Алгоритм протокола NTP

- NTP-клиент запрашивает 3 или более серверов в различных сетях.
- Для учета времени передачи данных, используется следующий метод расчета круговой задержки (round-trip delay – RTD):

Время между отправкой запроса клиентом и получением ответа

Время между получением запроса сервером и отправкой ответа

$$\partial = (t_{ClientGet} - t_{ClientSend}) - (t_{ServerSend} - t_{ServerGet})$$

где:

$t_{ClientSend}$	время на клиенте при отправке запроса
$t_{ClientGet}$	время на клиенте при получении ответа
$t_{ServerGet}$	время на сервере при получении запроса
$t_{ServerSend}$	время на сервере при отправке ответа

Алгоритм протокола NTP

- Сдвиг вычисляется по формуле:

$$\theta = \frac{(t_{ServerGet} - t_{ClientSend}) + (t_{ServerSend} - t_{ClientGet})}{2}$$

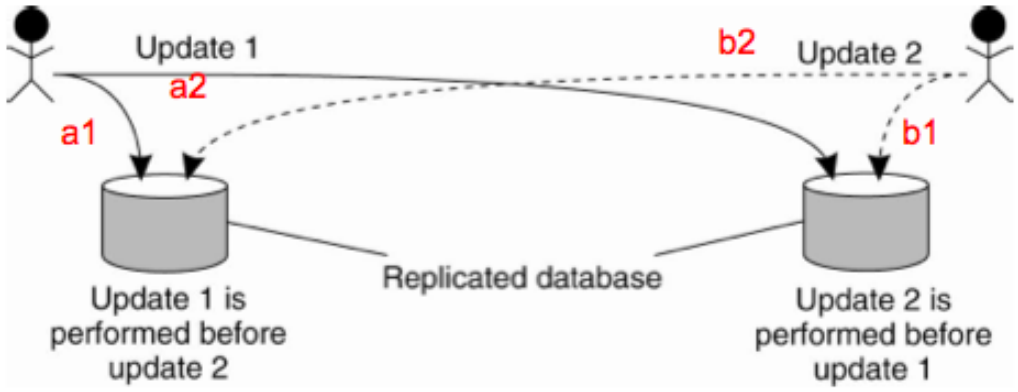
Рассчитанные δ и θ от нескольких NTP-серверов проходят статистический анализ, после чего явно искаженные результаты отбрасываются.

Лучшие 3 NTP-сервера используются для получения сдвига по времени, после чего частота локальных часов корректируется для минимизации сдвига.

Логические часы

Логические часы

- В РВС часто не требуется полной синхронизации всех физических часов на узлах. Достаточно, чтобы просто на каждом узле были физические часы с определенной достаточной точностью.

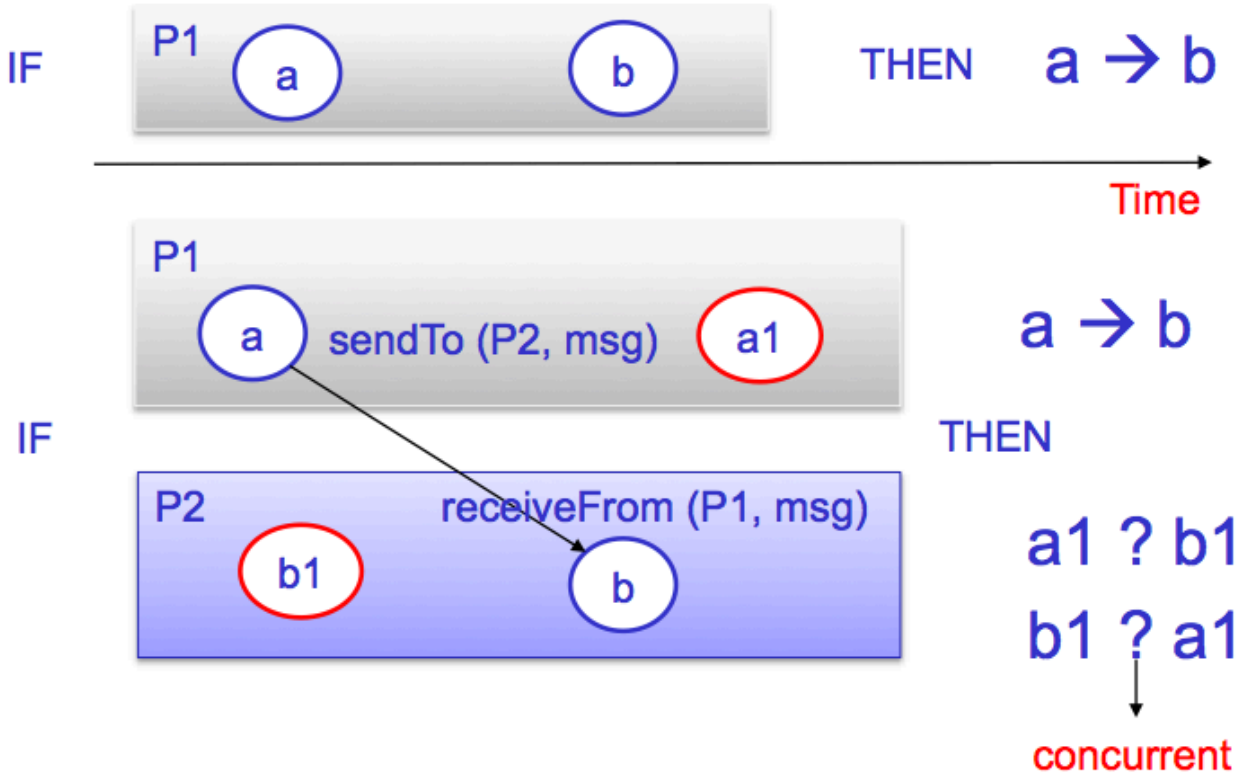


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Нам нужно, чтобы (a1, a2) выполнилось до того, как выполнится (b1, b2).

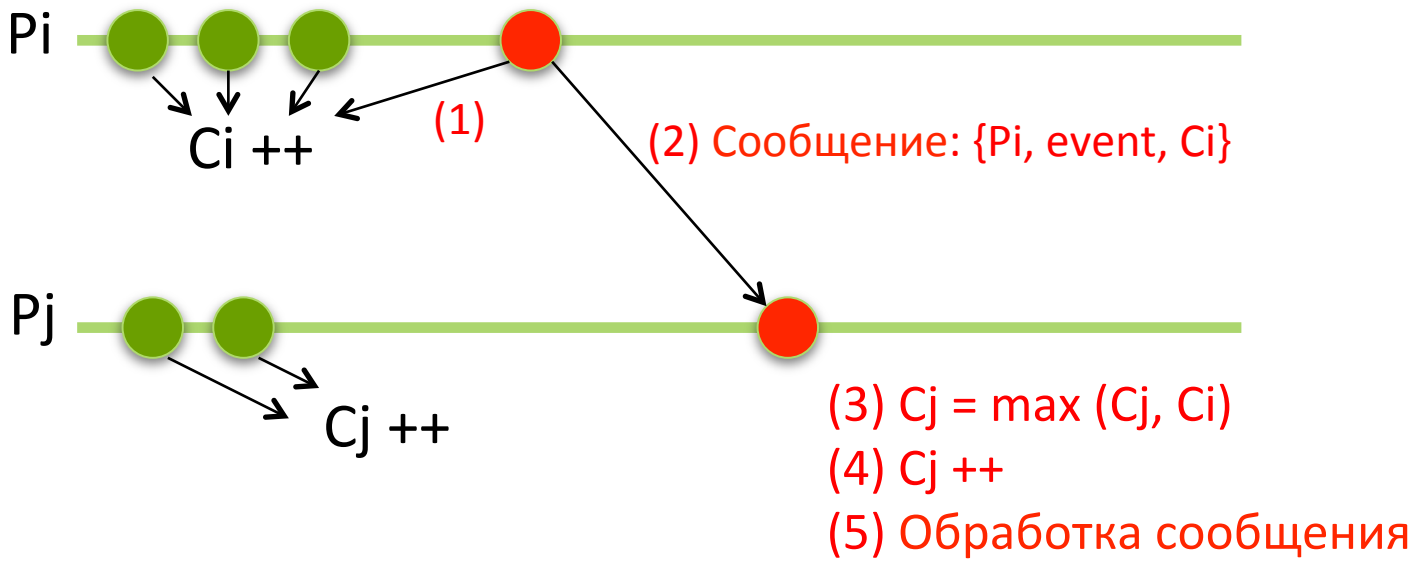
«Произошло до»

- Отношение «Произошло до» (->) между событиями a и и означает, что событие a логически произошло перед событием b. Т.е. есть возможность, что a могло повлиять на b.

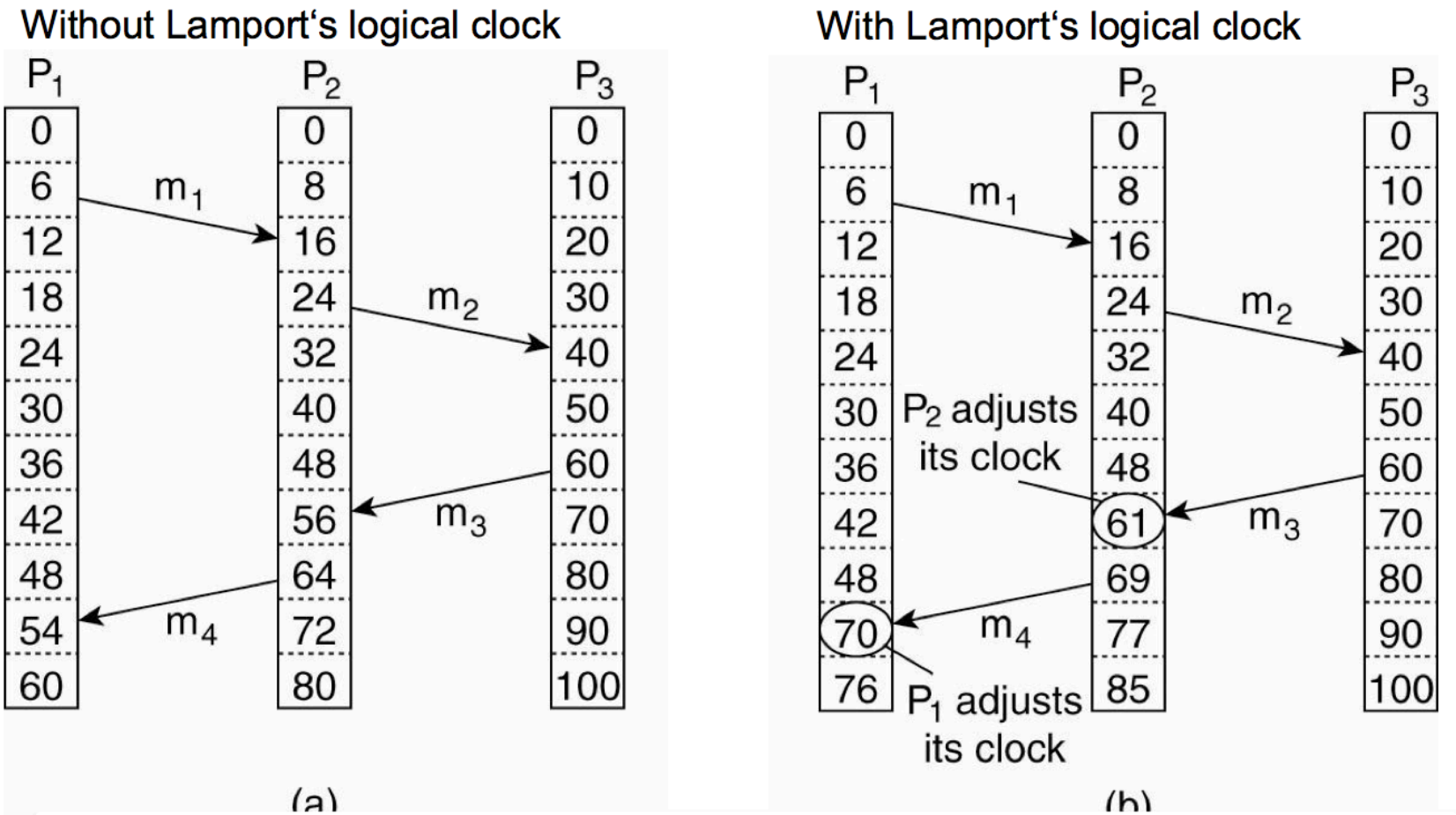


Логические часы Лампорта

- Используются для синхронизации событий в РВС. Каждый процесс Pi имеет собственный счетчик (часы) Ci. Каждое событие увеличивает значение счетчика на 1.
- При отправке сообщения на узел Pj передается событие и текущее значение счетчика Ci. При получении такого сообщения, процесс Pj устанавливает значение своего счетчика как max(Cj,Ci).



Пример работы логических часов Лампорта



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Логические часы Лампорта

- Применение логических часов Лампорта позволяет утверждать, что:

Если a->b, то C(a) < C(b)

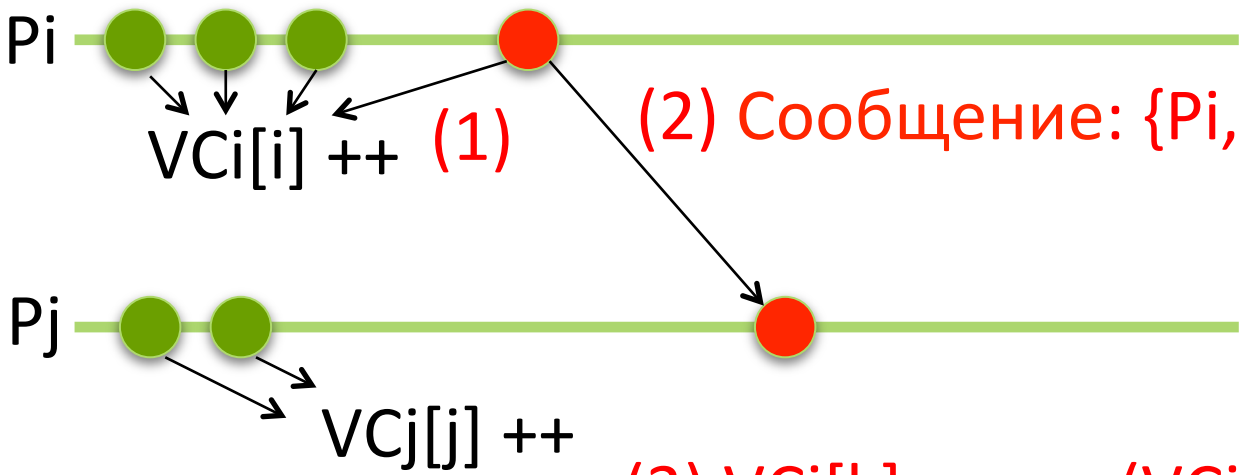
- Более сильное утверждение

{Если C(a) < C(b), то a->b} не может быть обеспечено посредством только меток Лампорта, необходимо применение векторных часов.

Векторные часы

- Процесс Pi поддерживает векторные часы VCi:
  - Значение VCi[i] – это количество событий, произошедших в Pi,
  - Если VCi[j] = k – это означает, что Pi знает о k событиях, произошедших в процессе Pj, который имеет причинно-следственную связь с процессом Pi.
- Каждое сообщение ассоциируется с VC.
- Для данных сообщений a и b, возможно что a воздействует на b, если a.VC < b.VC

Векторные часы



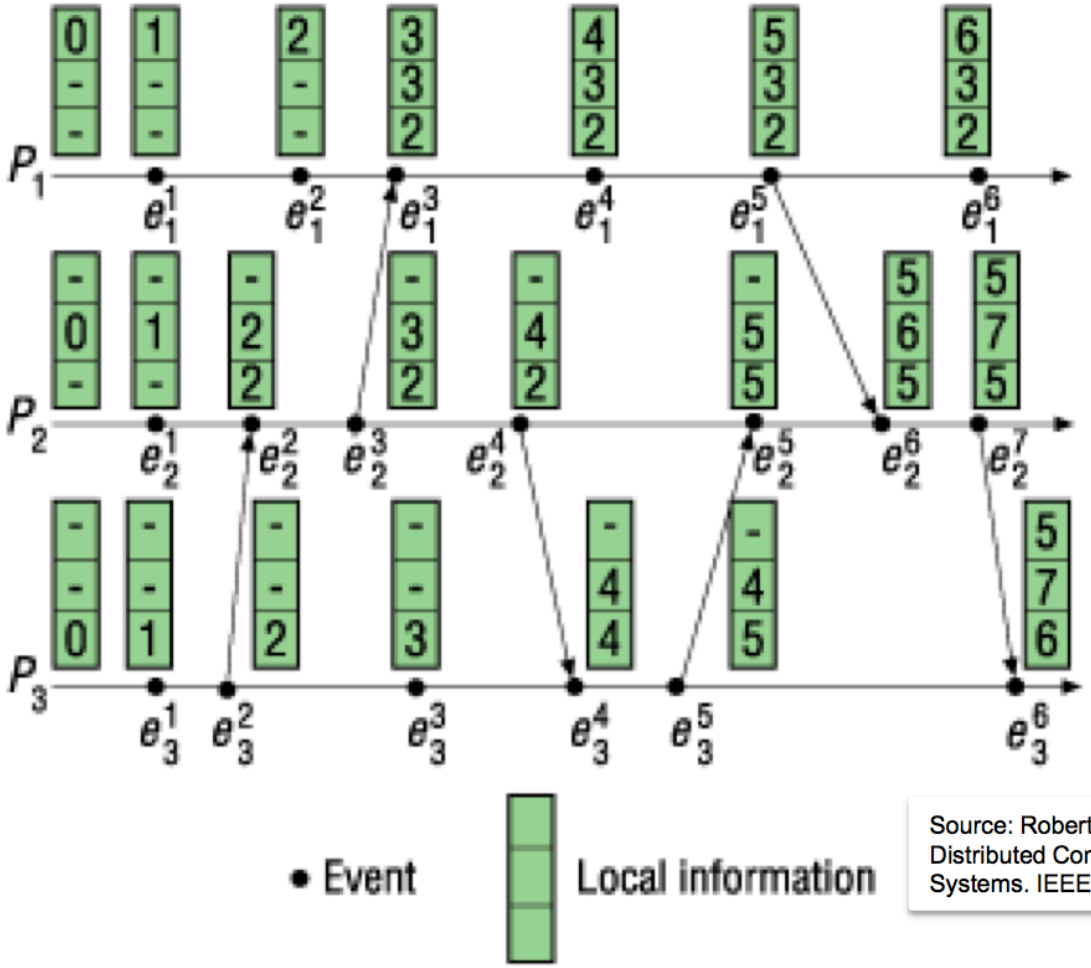
(2) Сообщение: {Pi, event, VCi}

(3) VCj[k] = max (VCj[k], VCi[k])

(4) VCj[j] ++

(5) Обработка сообщения

Векторные часы

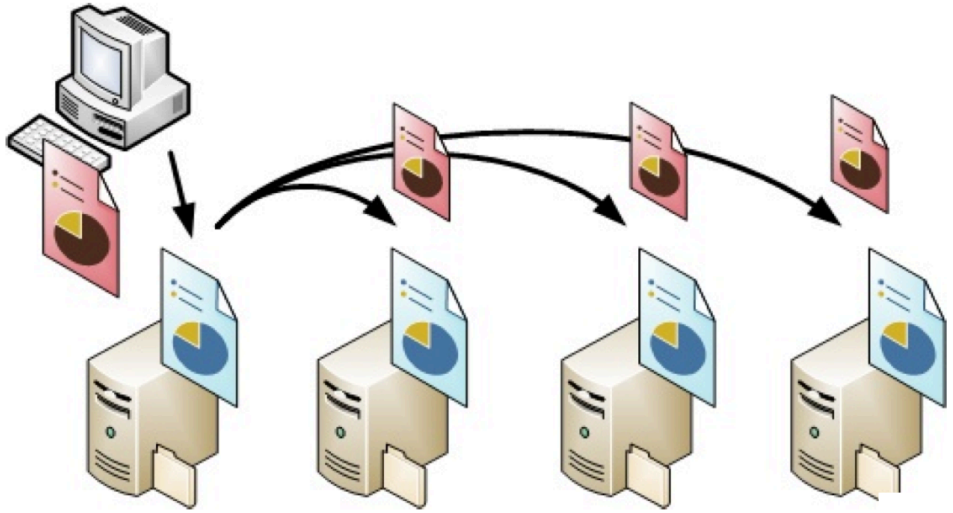


Source: Roberto Baldoni, Michel Raynal: Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems. IEEE Distributed Systems Online 3(2) (2002)

Согласованность и репликация данных

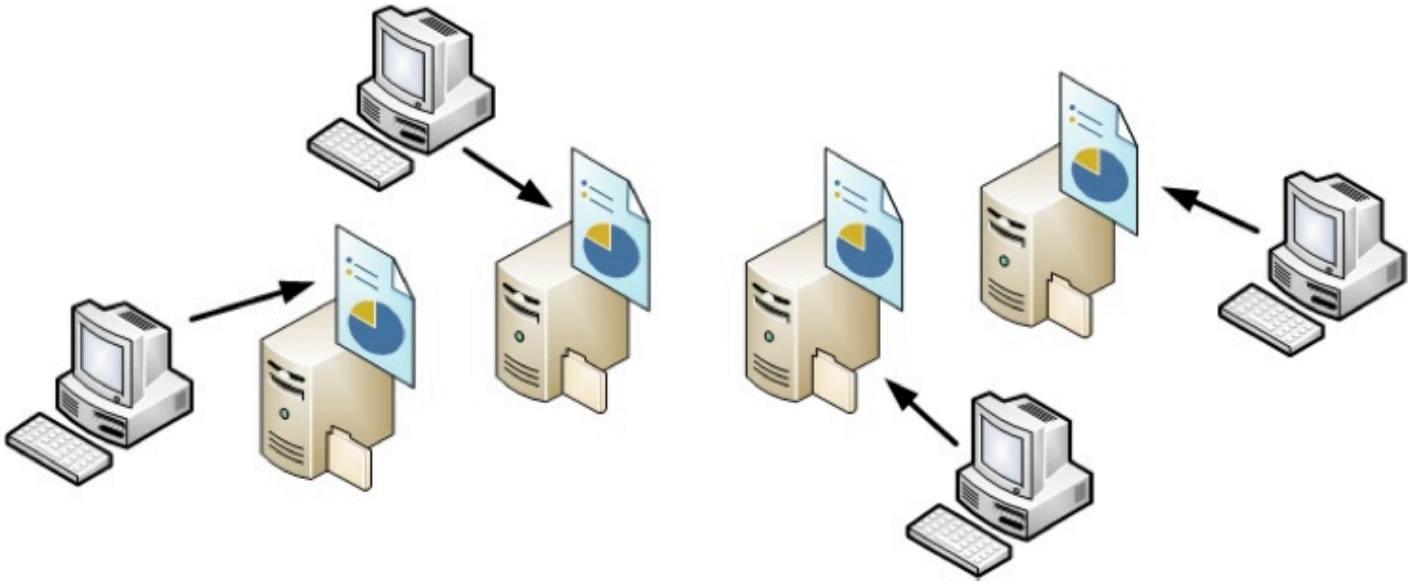
Согласованность и репликация

- ***Репликация -*** это процесс поддержки нескольких копий одного элемента данных в различных локациях.
- ***Согласование данных –*** это процесс поддержки всех реплицированных копий одного элемента данных в одинаковом состоянии при изменении одной из копий.



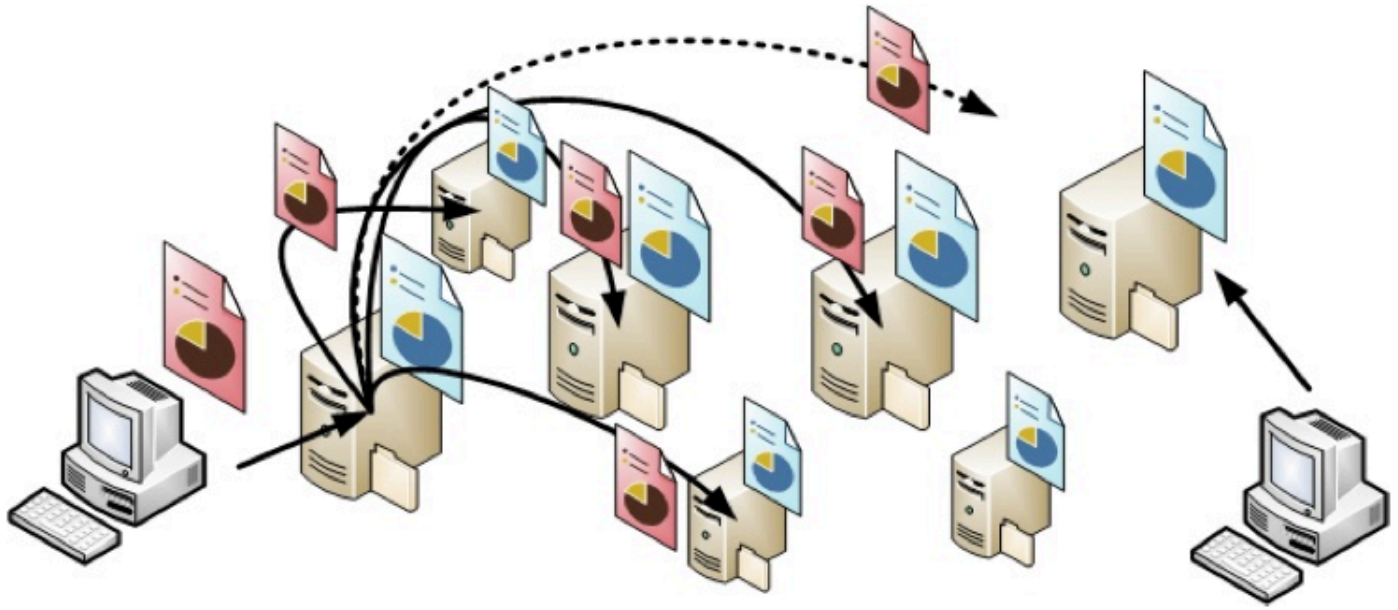
Достоинства репликации

- Чем больше копий данных, тем лучше масштабируемость системы и тем больше клиентов вы можете обслужить.
- Реплики данных можно разместить ближе к клиентам, обеспечивая меньшее время отклика



Недостатки репликации

- Поддержка актуального состояния данных может потребовать большой нагрузки на сеть.
- Обновление данных происходит не мгновенно.



Поддержание производительности и масштабируемости

- Основная задача: для поддержки *согласованности* метрик необходимо, чтобы все *конфликтующие операции* производились со всеми копиями в одном и том же порядке.
- Конфликтующие операции:
  - *Конфликт чтение-запись:* если операции чтения и записи конфликтуют между собой;
  - *Конфликт запись-запись:* две конкурирующие операции записи.
- Проблема: гарантирование глобального порядка выполнения конфликтующих ситуаций может быть очень затратной задачей, снижающей масштабируемость.
- Решение: снижение требований к согласованности, таким образом надеясь, что глобальной синхронизации можно будет избежать.

Непрерывная согласованность

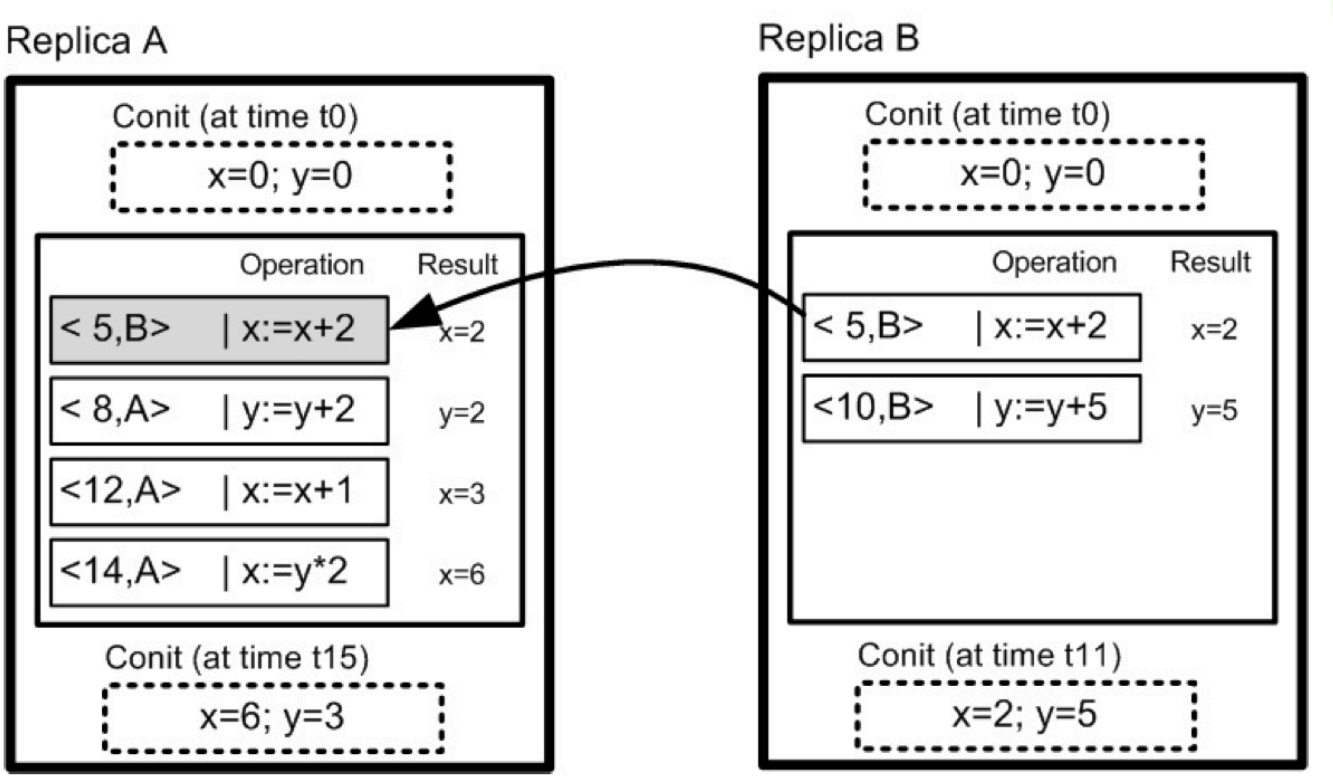
Непрерывная согласованность и «Конит»

- Мы можем говорить о «степени согласованности» данных (т.е. одни данные могут быть более согласованными, чем другие)
- Для определения несогласованности, можно ввести понятие «Единица согласованности» (Consistency unit: «Co-nit»).
- Конит определяет единицу данных, в отношении которых определяется согласованность либо несогласованность.
  - например, на бирже Конитом может служить запись, представляющая отдельную акцию;
  - Конитом может быть информация о прогнозе погоды.
  - Или же веб-страница, строка БД, вся БД.

Типы несогласованности

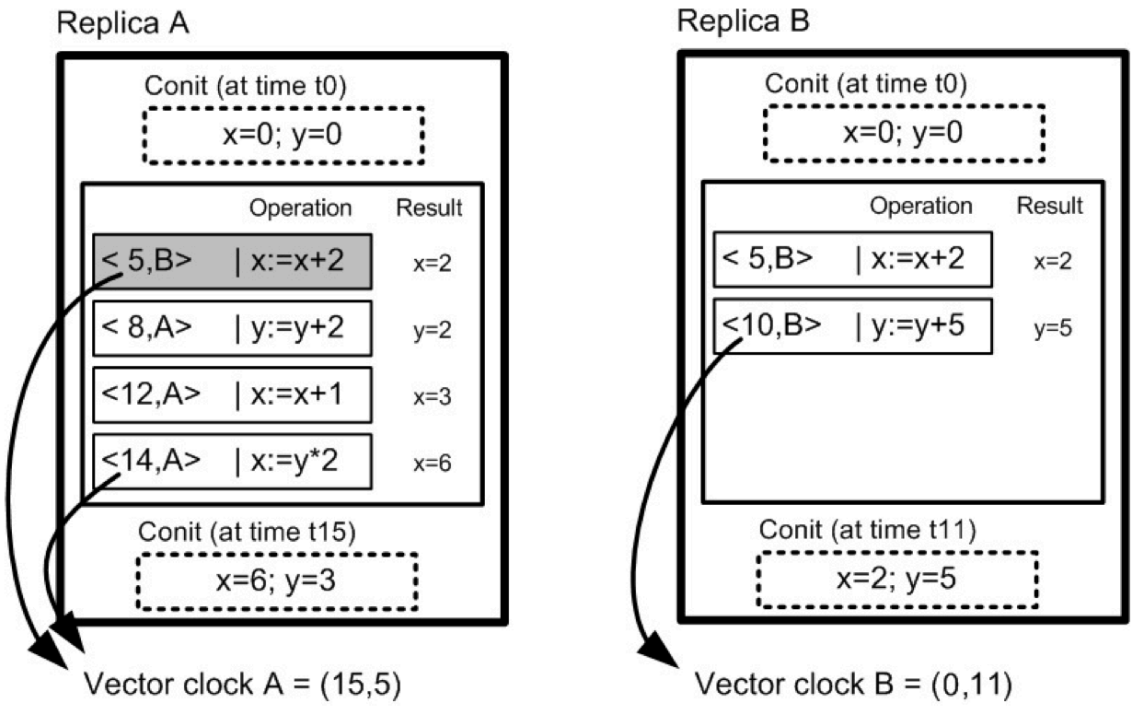
- Можно выделить 3 шкалы несогласованности данных:
  - численные расхождения: используются в приложениях, работающих с цифрами. Могут быть абсолютными ($0.02), относительными (0.5%) или же оцениваться количеством обновлений, которые необходимо применить к реплике, чтобы согласовать ее со остальными
  - расхождения устаревания: учитывается время последнего обновления реплики (прогноз погоды можно обновлять раз в час).
  - порядок обновлений: есть приложения, где реплики могут находиться в различных состояниях обновления в одно и то же время. Недостатком таких систем является то, что, возможно, при синхронизации реплик потребуется откат транзакций и повторное применение обновлений.

Пример конита



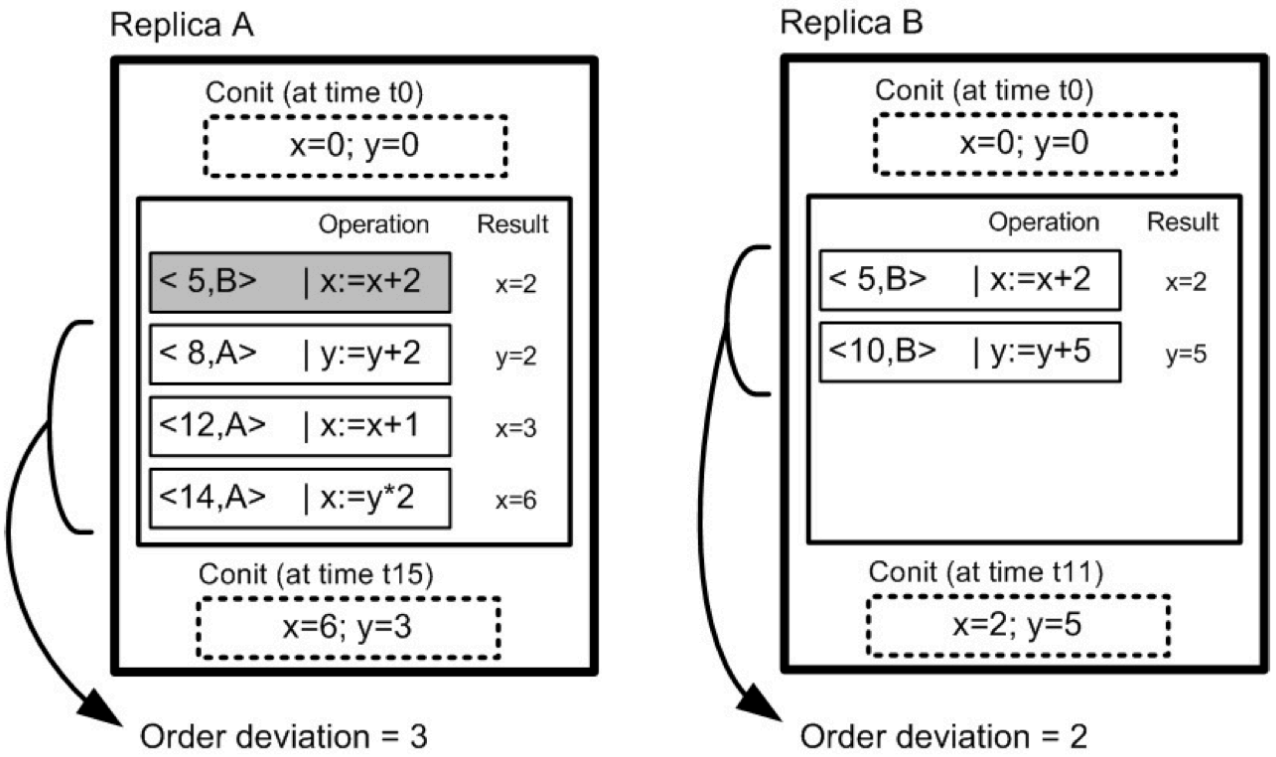
Определим 2 реплики одного набора данных: x=0; y=0.
В реплике B проведем операцию <5, B> и оправим обновление в A.
Об остальных операциях оповещать другие реплики не будем.

Пример конита: векторные часы



Реплика A знает об операции <5, B>. Т.о. она знает, что в B логические часы прошли отметку 5. Об остальных событиях в B реплика A не знает.
Таким образом, векторные часы A: {A:(14+1),B:5}.
B не знает ничего об A, он не получал от него никаких сообщений.
Таким образом, векторные часы B: {A:0,B:(10+1)}.

Пример конита: порядок обновлений



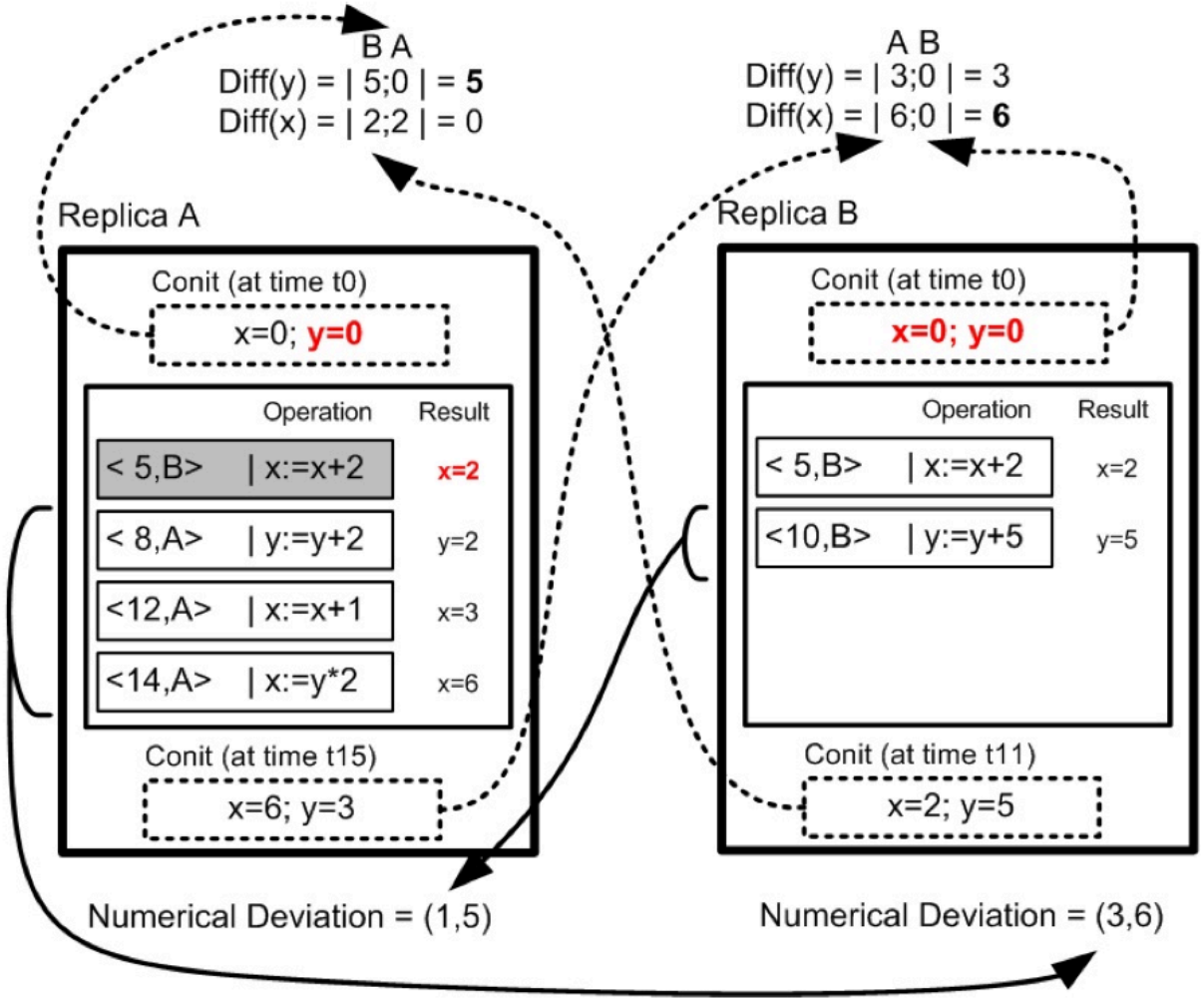
Реплика A получила информацию от B 3 обновления назад.

Реплика B провела уже 2 обновления, не получив от A никаких оповещений за это время.

Пример конита: численные расхождения

- Определим в нашем примере численное расхождение для A как пару:
  - количество операций, которые еще не получила реплика A от реплики B. Мы не знаем о 1 операции <10,B>.
  - максимальная разница между текущим состоянием переменных Bx и By и информацией, которой обладает A о реплике B. A знает о том, что в B было достигнуто состояние x=2, y=0. После этого, в B была операция y:=y+5, о которой не известно в A.
  - Таким образом, численное расхождение для A = (1, 5)
- B не знает ничего об A. Таким образом, есть 3 операции в A, которые не известны в B, которые по максимуму дают расхождение в 6 (мы думаем, что в A x=0, а он равен 6). Таким образом, численное расхождение для B = (3, 6)

Пример конита: численные расхождения

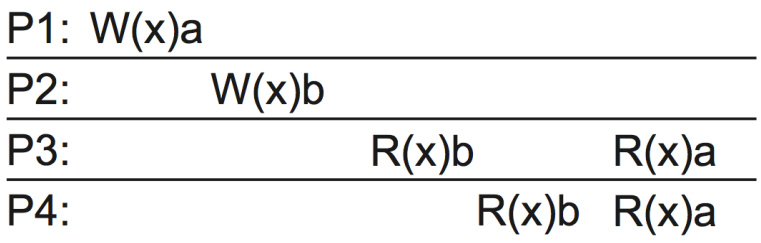


Последовательная согласованность

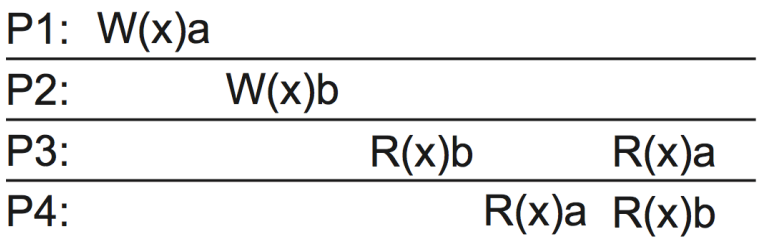
Последовательная согласованность

- Хранилище данных обеспечивает последовательную согласованность, если результат любых действий (чтения – записи) выполненных параллельными процессами соответствует варианту, как если бы операции всех процессов были исполнены в некотором последовательном порядке, при этом операции каждого отдельного процесса должны появляться в этой последовательности в порядке, установленном его программой.
- Т.е. мы не требуем синхронизации глобального времени между всеми процессами, но мы должны иметь возможность выстроить их в один последовательный, согласованный, не противоречащий ничему процесс.
- Для этого необходимо, чтобы каждый процесс-читатель получал обновления в одном и том же порядке.

Пример последовательной согласованности



(a)



(b)

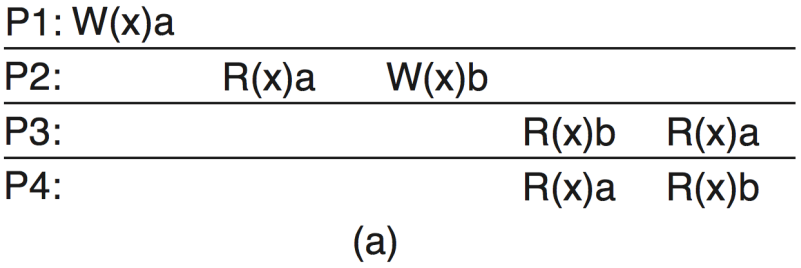
- "W(x)a" означает, что процесс записал в переменную x значение a
- "R(x)a" означает, что процесс считал из переменной x значение a
- В варианте (a) оказалось, что операция P2 выполнилась перед P1, что привело к тому, что P3 и P4 в начале считали b из x, а только потом a. Последовательная согласованность не нарушается.
- В варианте (b) принципы последовательной согласованности нарушаются, т.к. для P3 произошло P2, а потом P1; а для P4 произошло в начале P1 а потом P2. Мы не сможем их выстроить в один последовательный согласованный процесс.

Причинная согласованность

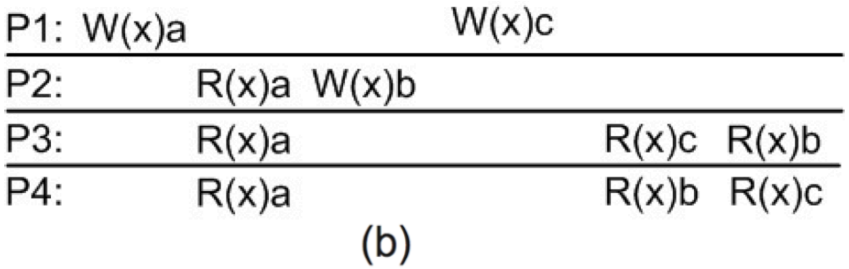
Причинная согласованность

- Причинная согласованность – это ослабление последовательной согласованности.
- Операции записи, которые, потенциально, причинно-зависимы, должны быть видны всем процессам в одном и том же порядке. Параллельные операции записи могут быть видны каждому процессу в разном порядке.
- Главный вопрос – как определить операции, которые причинно-зависимы.

Причинная согласованность



(a)



(b)

- (a) процесс P2 считал a из x, а потом вписал b – операция, потенциально, причинно-зависима (например, x=x+1). Но процессы P3 и P4 считали данные из x в разной последовательности. Система причинно-несогласованна.
- (b) x в начале был присвоен a, потом (параллельно) b и/или c. В этом случае, согласованность не нарушается ни одним из процессов, соответственно, система причинно-согласованна.

FIFO - согласованность

FIFO - согласованность

- Операции записи, которые сделаны одним процессом должны быть видны всем другим процессам в том порядке, в котором они были выполнены.
- При этом, операции записи из разных процессов могут быть видны в различном порядке в разных процессах.

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

Согласование будет обеспечено, только если все процессы соответствуют правилу R(x)b -> R(x)c

FIFO - согласованность

- Данный тип согласованности легко реализовать: все операции в различных процессах определяются как «параллельные».
- Но результаты применения такой согласованности иногда не интуитивны:

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

Если используется FIFO-согласованность, то ОБА процесса могут быть уничтожены (что невозможно при последовательной согласованности).

Итого

- Мы рассмотрели модели синхронизации времени (Алгоритм Беркли, NTP)
- Рассмотрели существующие модели логических часов, включая логические часы Лампорта и векторные часы.
- Рассмотрели модели согласованности данных в распределенных системах: непрерывную согласованность, последовательную согласованность, причинную согласованность, FIFO – согласованность.