Время, часы и порядок событий

Сухорослов Олег Викторович

Распределенные системы

Факультет компьютерных наук НИУ ВШЭ

21.11.2020

Зачем нужны часы в PC?

- Измерять длительность некоторого процесса во времени
  - Профайлинг, таймауты, детекторы отказов, планировщики...
- Определять момент времени, когда произошло некоторое событие
  - Логирование событий, время покупки в базе данных...
- Проверять наступил ли некоторый момент времени
  - Проверка валидности цифрового сертификата...
- Определять порядок событий, происходящих на разных узлах системы

Типы часов

- Физические
  - измеряют число прошедших секунд
- Логические
  - измеряют число произошедших событий (например, отправленных сообщений)

Физические часы

- Кварцевые часы
  - Частота колебаний кварцевого генератора известна с некоторой погрешностью
  - Типичный допустимый дрейф часов: 20-50 ppm (~10-25 минут в год)
  - Частота может существенно зависеть от температуры
- Атомные часы
  - 1 секунда = 9 192 631 770 периодов излучения, возникающих при переходах между уровнями состояния атома цезия-133
  - Точность 10^{-14} (1 секунда в 3 миллиона лет)
  - Цена значительно дороже
  - Используются в GPS

Стандарты времени

- Среднее время по Гринвичу (GMT)
  - основано на астрономических наблюдениях
  - скорость вращения Земли не является постоянной
- Международное атомное время (TAI)
  - основано на квантовой механике
- Всемирное координированное время (UTC)
  - TAI с корректировками, учитывающими вращение Земли
  - коррекции в форме leap second применяются 30 июня и 31 декабря

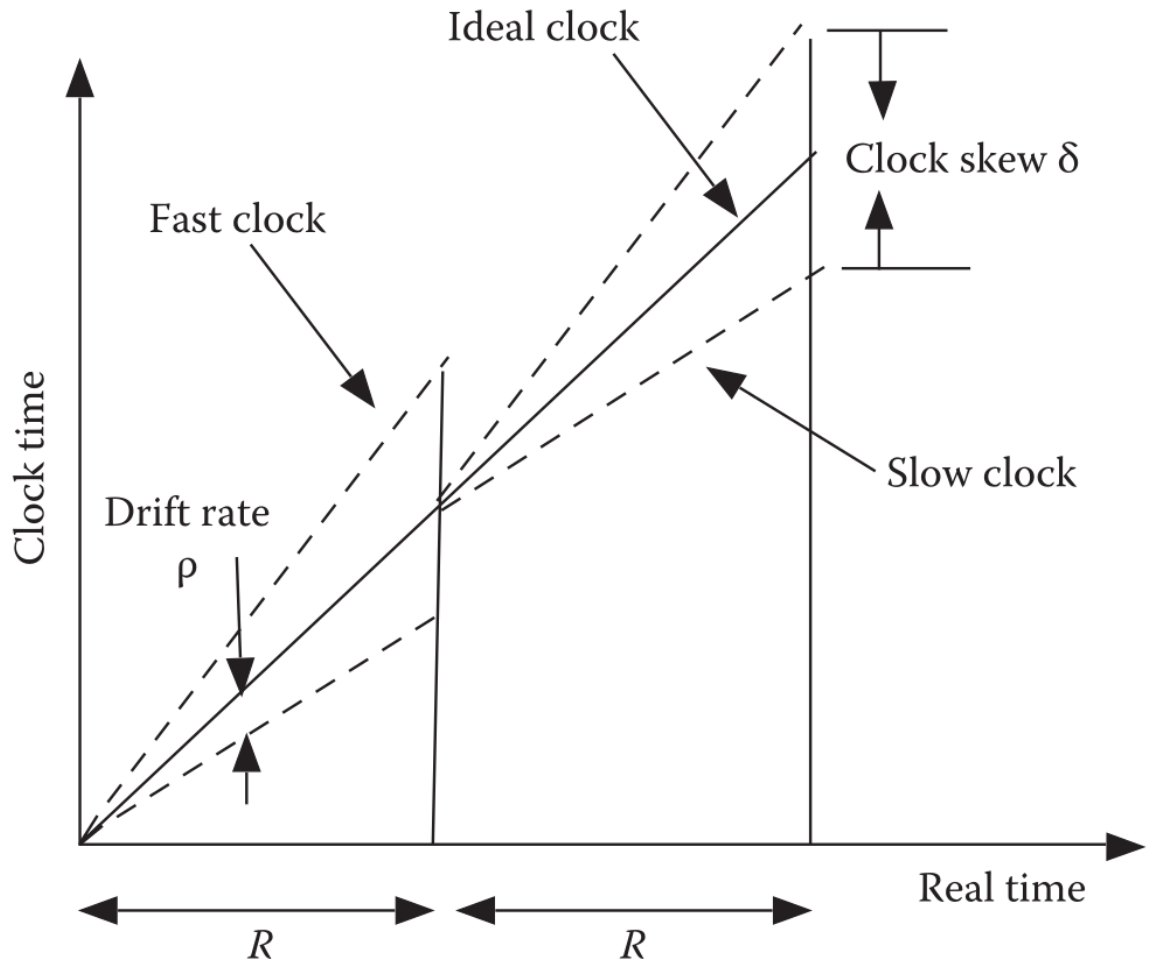
Представление времени на компьютере

- Unix time
  - число секунд с 00:00 1 января 1970 года UTC
  - нет учёта leap seconds
- ISO 8601
  - текстовый формат представления времени в UTC
  - `2020-11-21T16:30:56+03:00`
- Преобразование между форматами
  - Григорианский календарь
  - знание прошлых и будущих leap seconds
- Много ПО просто игнорирует leap seconds
  - но в ОС и РС часто требуется измерять время с высокой точностью

Инцидент 30 июня 2012 года

- Anyone else experiencing high rates of Linux server crashes during a leap second day?
- The Inside Story of the Extra Second That Crashed the Web
- Linux is culprit in leap-second lapses, says Cassandra exec

Смещение и дрейф часов



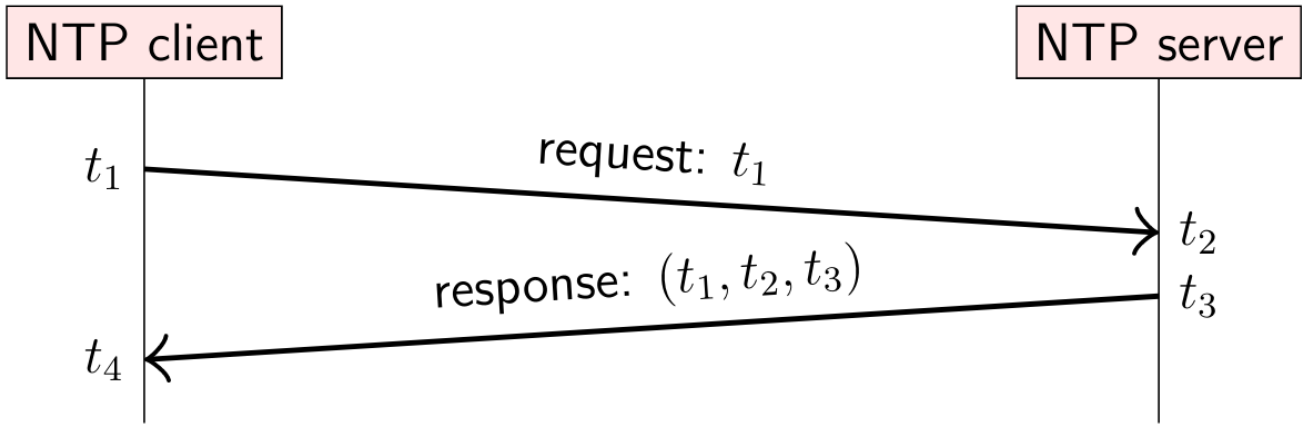
Синхронизация часов

- Цель: поддерживать смещение часов в заданных пределах
- Network Time Protocol (NTP): ~1-100 мс
- Precision Time Protocol (PTP): ~100 мкс
- Google TrueTime: 7 мс

Network Time Protocol (NTP)

- Клиент-серверный протокол поверх UDP
- Иерархическая сеть серверов из нескольких слоев
  - Stratum 0: серверы с непосредственным доступом к точным источникам времени (атомные часы, GPS-приемник...)
  - Stratum 1: серверы напрямую синхронизирующиеся с серверами stratum 0
  - Stratum 2: серверы синхронизирующиеся с серверами stratum 1 ...
- Клиент может взаимодействовать с несколькими серверами
  - Исключение неисправных серверов, усреднение измерений
- Клиент делает несколько запросов к одному серверу
  - Уменьшение ошибки, возникающей из-за изменений сетевой задержки
- Точность синхронизации ~ 1-100 мс

Оценка точного времени



Round-trip network delay: $\delta = (t_4 - t_1) - (t_3 - t_2)$

Estimated server time when client receives response: $t_3 + \frac{\delta}{2}$

Estimated clock skew: $\theta = t_3 + \frac{\delta}{2} - t_4 = \frac{t_2 - t_1 + t_3 - t_4}{2}$

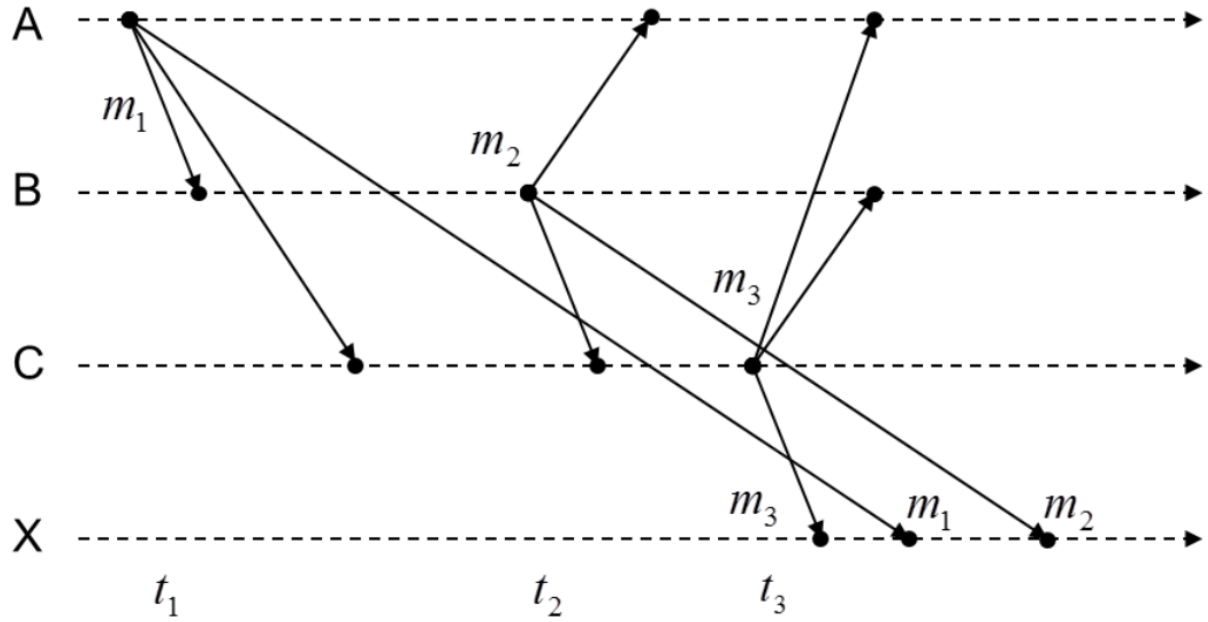
Коррекция часов клиента

- Смещение $\theta < 125$ ms
  - небольшое замедление или ускорение часов
- Смещение 125 ms $\leq \theta < 1000$ s
  - сброс часов клиента на точное время
- Смещение $\theta \geq 1000$ s
  - ничего не делаем, оставляем решение проблемы за администратором
  - необходим мониторинг смещения часов

Локальные часы

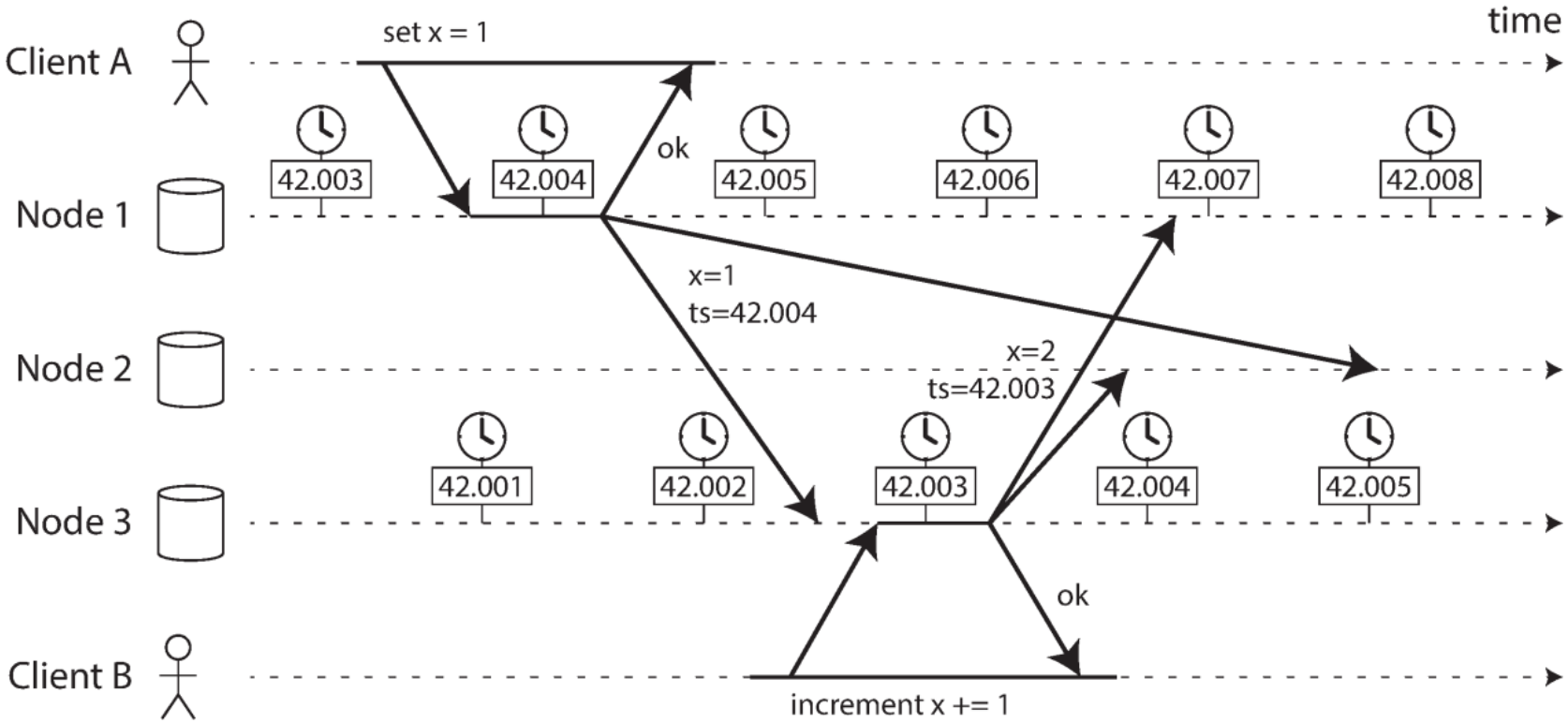
- Time-of-day clocks
  - Привязаны к привычному (wall clock) времени
  - Могут "скакать" назад и вперед (см. NTP, leap seconds)
  - Не подходят для измерения интервалов времени
  - Linux: `clock_gettime(CLOCK_REALTIME)`
- Monotonic clocks
  - Гарантируется монотонное увеличение значения часов
  - Произвольная точка отсчёта (например, время загрузки машины)
  - Не подходит для сравнения времен между разными машинами
  - Обычно более высокое разрешение
  - Linux: `clock_gettime(CLOCK_MONOTONIC)`

Порядок сообщений

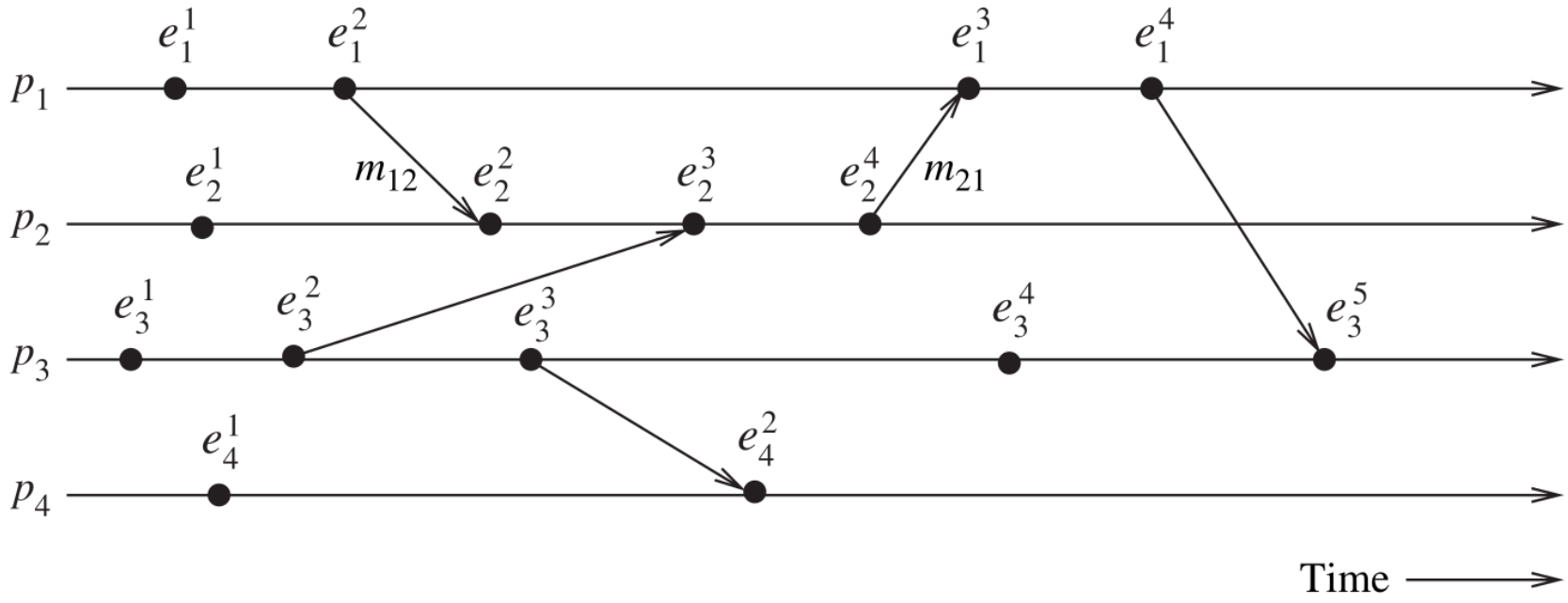


- Сообщение m_2 является ответом на m_1, а m_3 является ответом на m_2
- Узел X получил сообщения в неправильном (с точки зрения логики) порядке
- Поможет ли упорядочить события физическое время?

Последняя запись выигрывает?

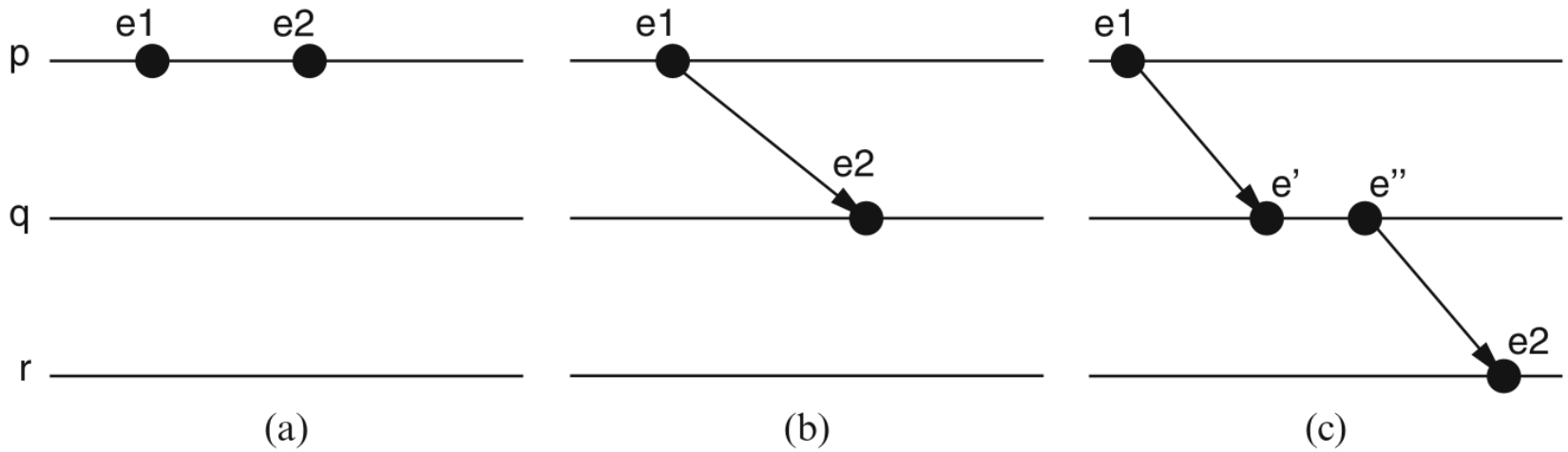


События в РС



- На каждом узле системы происходят события: получение или отправка сообщений, шаг выполнения...
- Порядок событий в рамках узла известен и соответствует локальным часам

Отношение happened-before

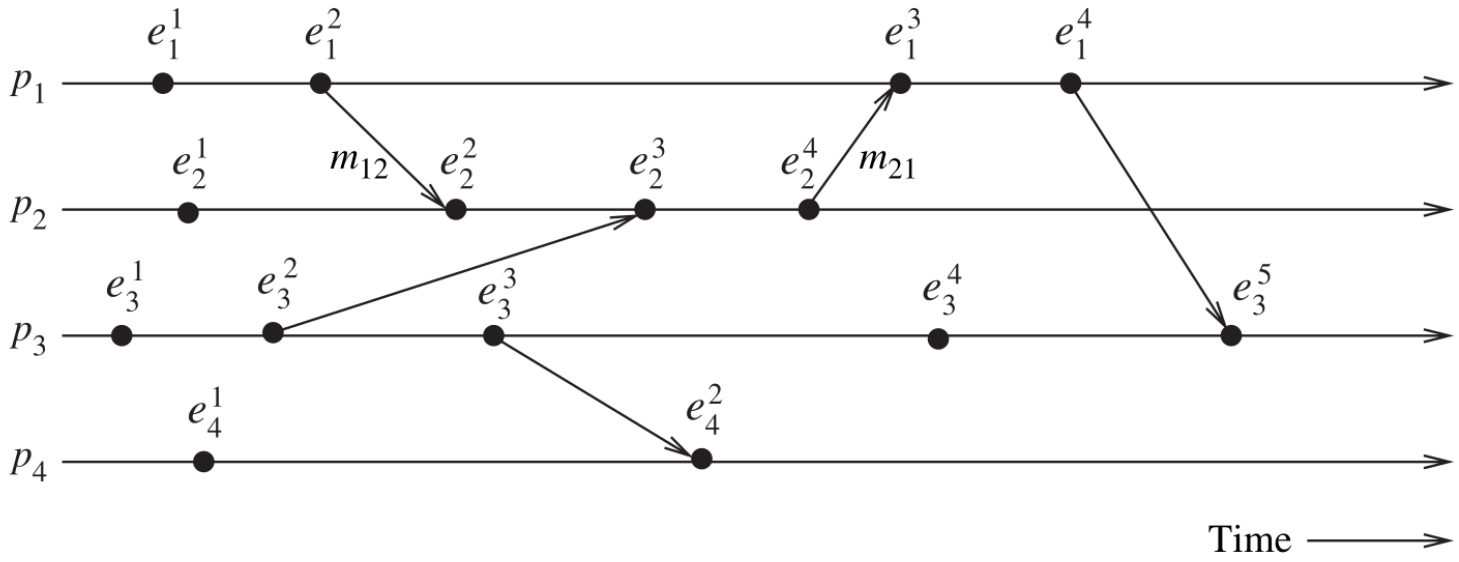


Событие a произошло до события b ($a \rightarrow b$) если выполняется одно из условий:

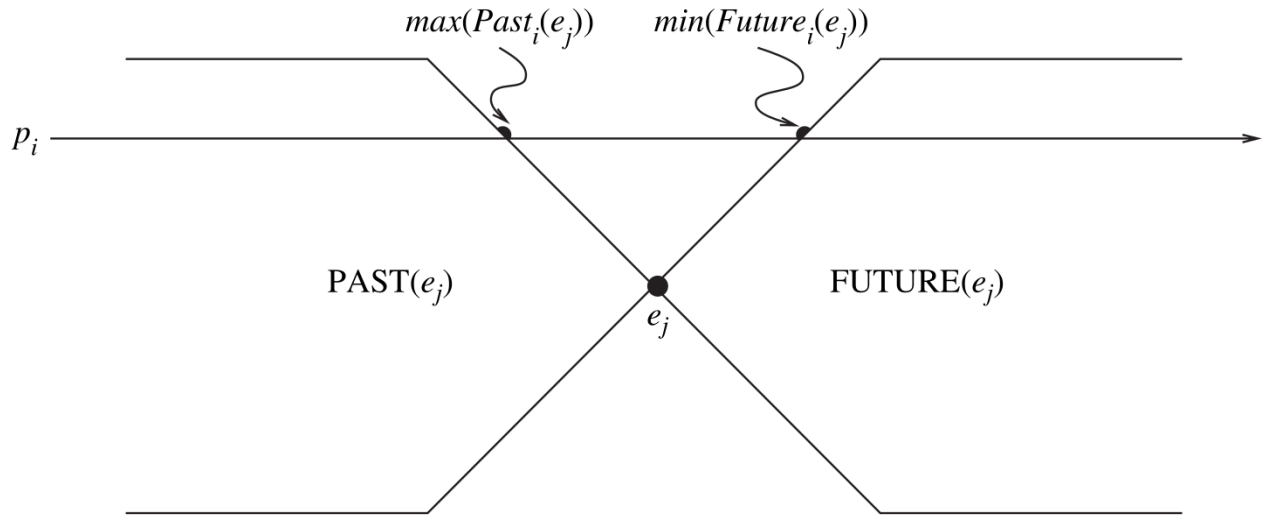
- a и b произошли на одном узле, и a произошло раньше
- a является отправкой сообщения m, а b является получением того же сообщения m
- существует событие c такое что $a \rightarrow c$ и $c \rightarrow b$

Отношение happened-before

- Отношение *happened-before* определяет только частичный порядок
  - Возможно, что не выполняется ни $a \rightarrow b$, ни $b \rightarrow a$
  - Тогда события a и b называют одновременными ($a \parallel b$)
- Для любых событий a и b выполняется или $a \rightarrow b$ или $b \rightarrow a$ или $a \parallel b$



Причинно-следственная связь

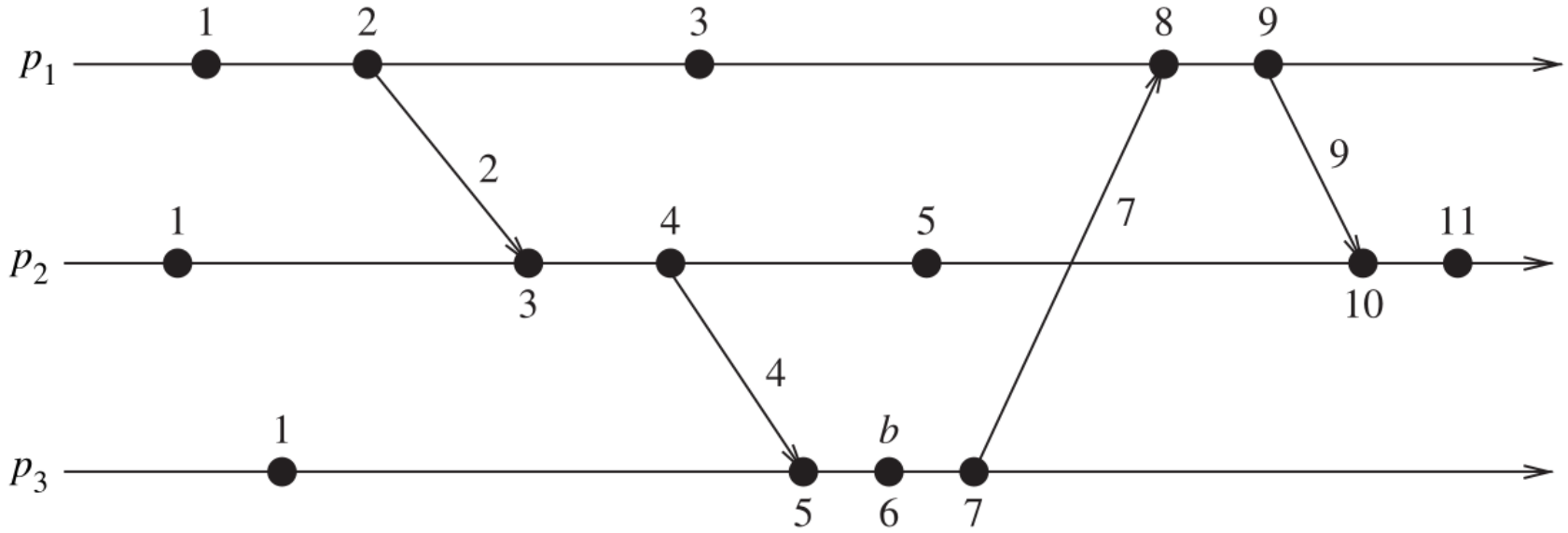


- Отношение *happened-before* указывает на возможную связь между событиями:
  - Если $a \rightarrow b$, то a могло повлиять на (быть причиной) b
  - Если $a \parallel b$, то a не могло повлиять на b
- Линейный порядок $\prec$ является *причинным порядком*, если $(a \rightarrow b) \Longrightarrow (a \prec b)$

Логические часы

- Физические часы
  - измеряют число прошедших секунд
  - сложно использовать в PC для получения причинного порядка событий
- Логические часы
  - измеряют число произошедших событий (например, отправленных сообщений)
  - не применимы для измерений моментов времени или длительности
  - сохраняют причинный порядок: $(a \rightarrow b) \Longrightarrow (T(a) < T(b))$

Часы Лэмпорта



Lamport L. Time, Clocks and the Ordering of Events in a Distributed System (1978)

Часы Лэмпорта: алгоритм

on initialisation **do**
    $t := 0$ ▷ each node has its own local variable t
end on

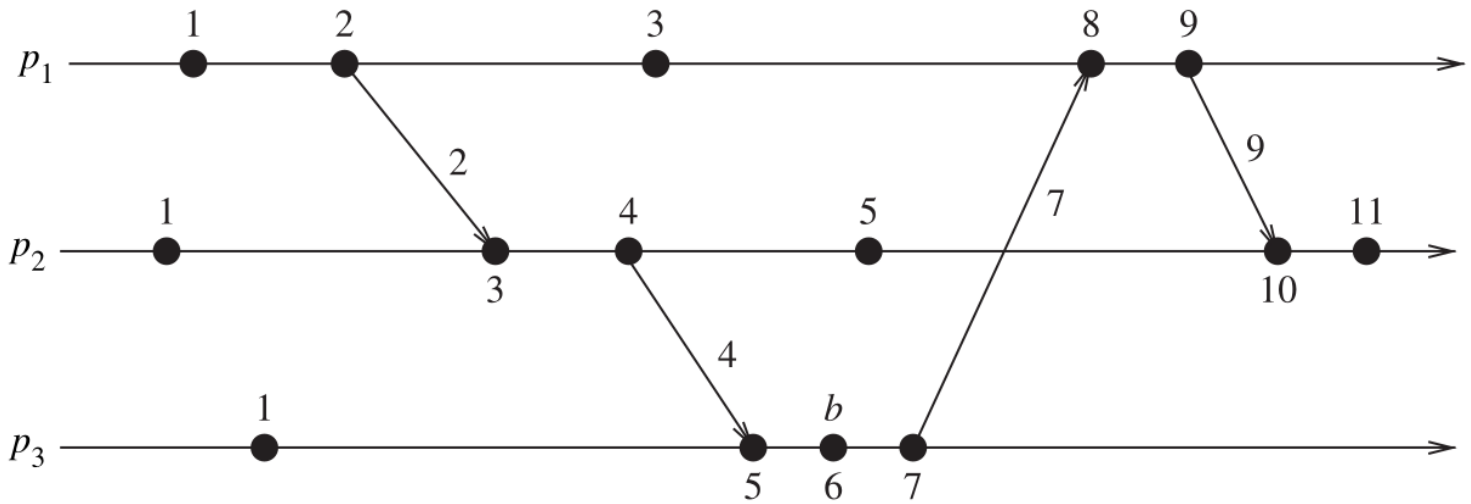
on any event occurring at the local node **do**
    $t := t + 1$
end on

on request to send message m **do**
    $t := t + 1$; send (t, m) via the underlying network link
end on

on receiving (t', m) via the underlying network link **do**
    $t := \max(t, t') + 1$
    deliver m to the application
end on

Часы Лэмпорта: свойства

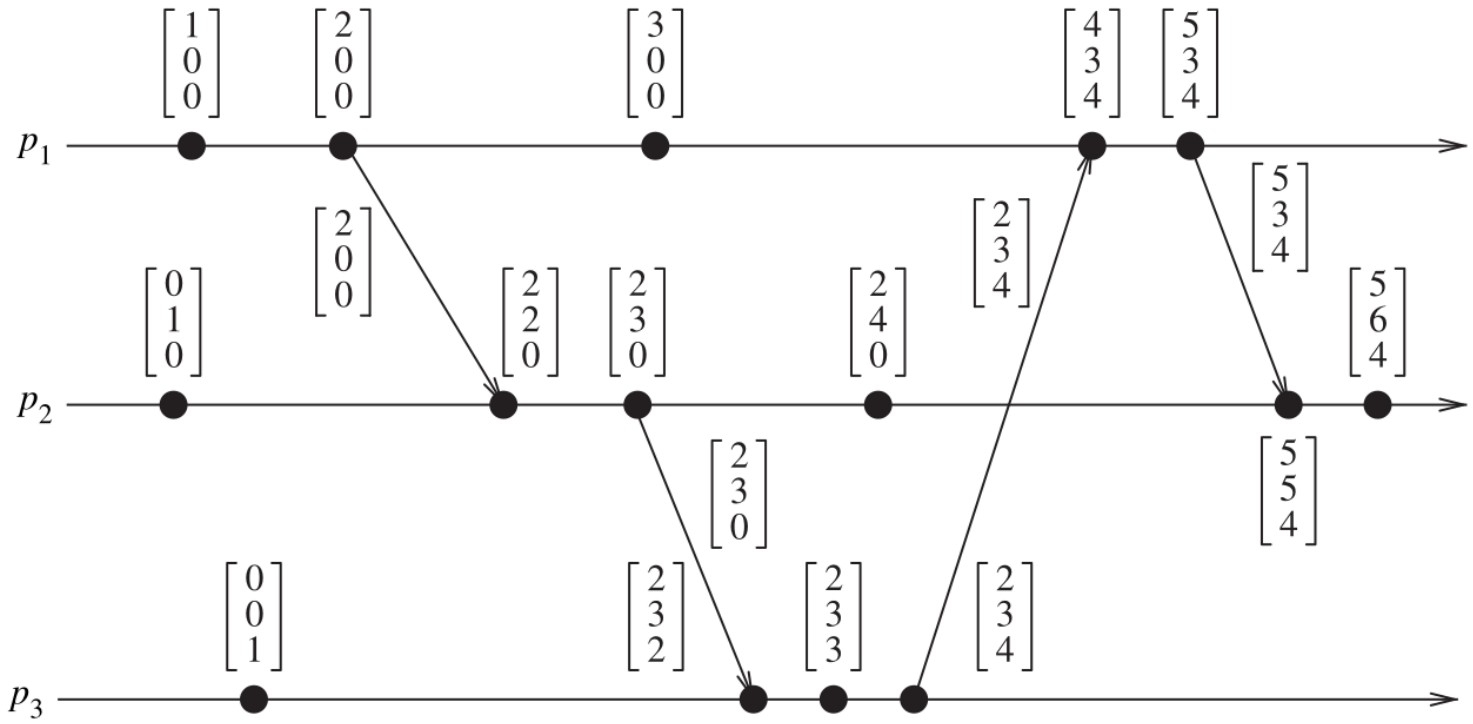
- Если $a \to b$, то $L(a) < L(b)$
- Из $L(a) < L(b)$ не следует $a \to b$ (может быть $a \parallel b$)
- Возможно $L(a) = L(b)$ для $a \neq b$



Линейный порядок событий

- Пусть $N(e)$ - уникальный идентификатор узла, на котором произошло событие e
- Пара $(L(e), N(e))$ уникально идентифицирует событие e
- Используя часы Лэмпорта можно определить линейный (полный) порядок событий: $(a \prec b) \iff (L(a) < L(b) \vee (L(a) = L(b) \wedge N(a) < N(b)))$
- Этот порядок является причинным, то есть $(a \to b) \implies (a \prec b)$

Векторные часы



Fidge C.J. Timestamps in Message-Passing Systems That Preserve the Partial Ordering (1988)
Mattern F. Virtual Time and Global States of Distributed Systems (1988)

Векторные часы: алгоритм

on initialisation at node N_i **do**
    $T := \langle 0, 0, \ldots, 0 \rangle$ ▷ local variable at node N_i
end on

on any event occurring at node N_i **do**
    $T[i] := T[i] + 1$
end on

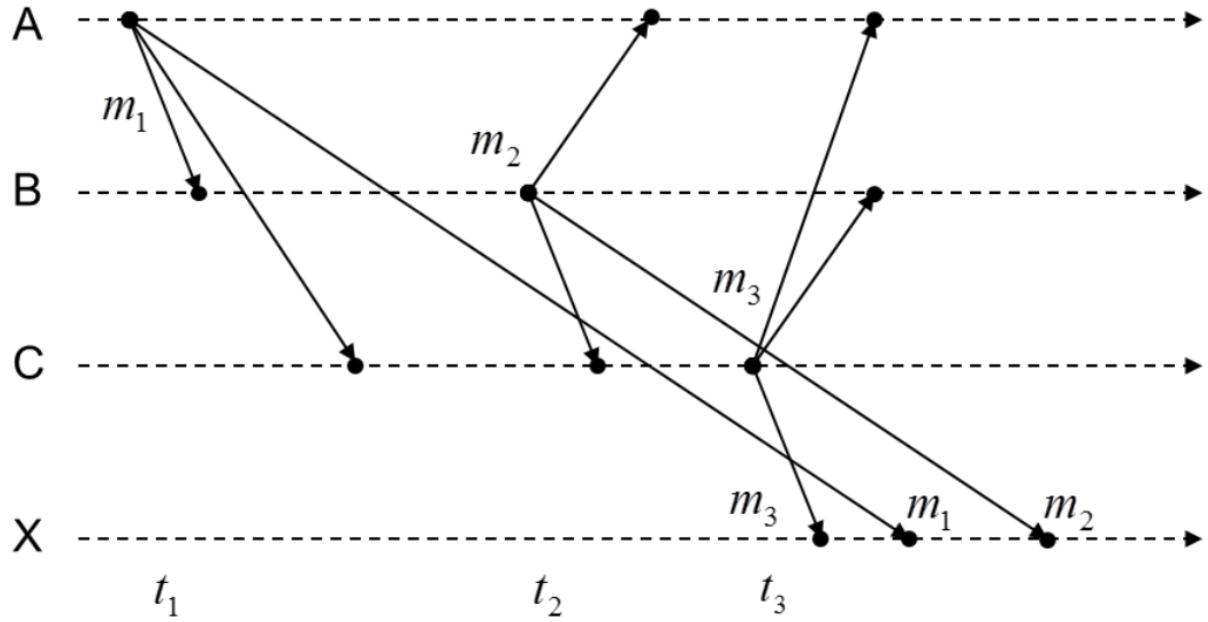
on request to send message m at node N_i **do**
    $T[i] := T[i] + 1$; send (T, m) via network
end on

on receiving (T', m) at node N_i via the network **do**
    $T[j] := \max(T[j], T'[j])$ for every $j \in \{1, \ldots, n\}$
    $T[i] := T[i] + 1$; deliver m to the application
end on

Векторные часы: свойства

- Правила сравнения значений векторного времени (n - число узлов)
  - $T = T' \iff T[i] = T'[i] \quad \forall i \in \{1, ..., n\}$
  - $T \leq T' \iff T[i] \leq T'[i] \quad \forall i \in \{1, ..., n\}$
  - $T < T' \iff T \leq T' \wedge T \neq T'$
  - $T \parallel T' \iff T \not\leq T' \wedge T' \not\leq T$
- Часы $V(a)$ кодируют набор событий, включая a и его зависимости $e \longrightarrow a$
  - $V(a) \leq V(b) \iff (\{a\} \cup \{e | e \rightarrow a\}) \subseteq (\{b\} \cup \{e | e \rightarrow b\})$
- Это приводит к следующим свойствам векторных часов
  - $V(a) < V(b) \iff a \rightarrow b$
  - $V(a) = V(b) \iff a = b$
  - $V(a) \parallel V(b) \iff a \parallel b$

Порядок сообщений



- Сообщение m_2 является ответом на m_1, а m_3 является ответом на m_2
- Узел X получил сообщения в неправильном (с точки зрения логики) порядке
- Поможет ли упорядочить события логическое время?

Causal Broadcast

- См. лекцию 4, где рассматривались алгоритмы рассылки в группе
- Используется вариант векторных часов
- Элемент $deps[i]$ на узле j содержит число сообщений от i, которые были доставлены на j
- $deps \leq delivered$ означает, что узел уже доставил все сообщения, которые должны предшествовать данному

on initialisation **do**
$sendSeq := 0;\ delivered := \langle 0, 0, \ldots, 0 \rangle;\ buffer := \{\}$
end on

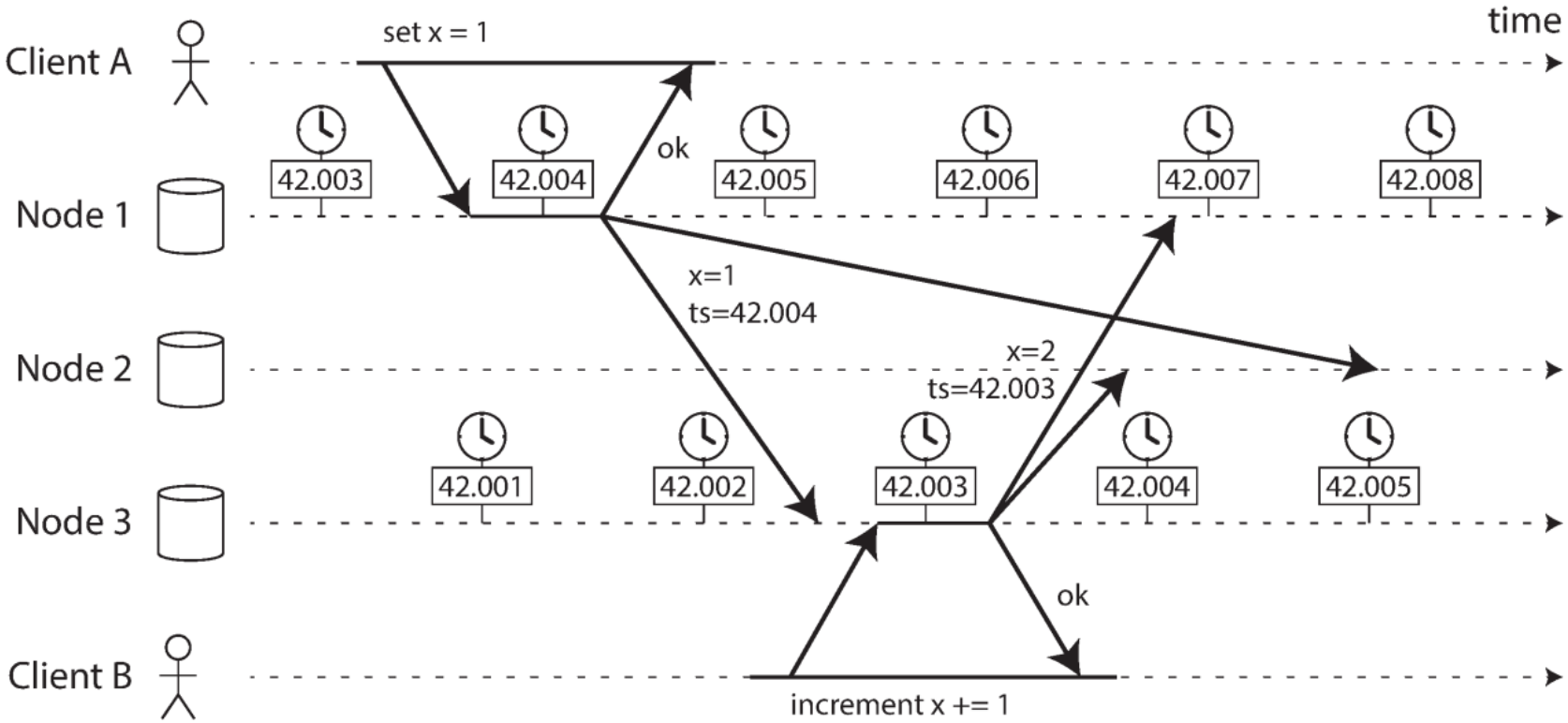
on request to broadcast m at node N_i **do**
$deps := delivered; deps[i] := sendSeq$
send $(i, deps, m)$ via reliable broadcast
$sendSeq := sendSeq + 1$
end on

on receiving msg from reliable broadcast at node N_i **do**
$buffer := buffer \cup \{msg\}$
while $\exists (sender, deps, m) \in buffer.\ deps \leq delivered$ **do**
deliver m to the application
$buffer := buffer \setminus \{(sender, deps, m)\}$
$delivered[sender] := delivered[sender] + 1$
end while
end on

Total Order Broadcast

- Все узлы должны получить рассылаемые сообщения в одном порядке
- Реализация на основе часов Лэмпорта
  - Добавим к каждому рассылаемому сообщению значение часов
  - Будем доставлять сообщения в ранее описанном линейном порядке на основе значений часов и идентификаторов узлов
- Как узнать, что мы (узел) уже получили все сообщения с временем $\leq T$?
  - Если сообщения от узлов приходят в порядке их отправки (FIFO link), то надо дождаться сообщения с временем $> T$ от *каждого* узла
  - Для этого можно использовать подтверждения или новые рассылки
- Проблема: отказ любого узла блокирует доставку сообщений

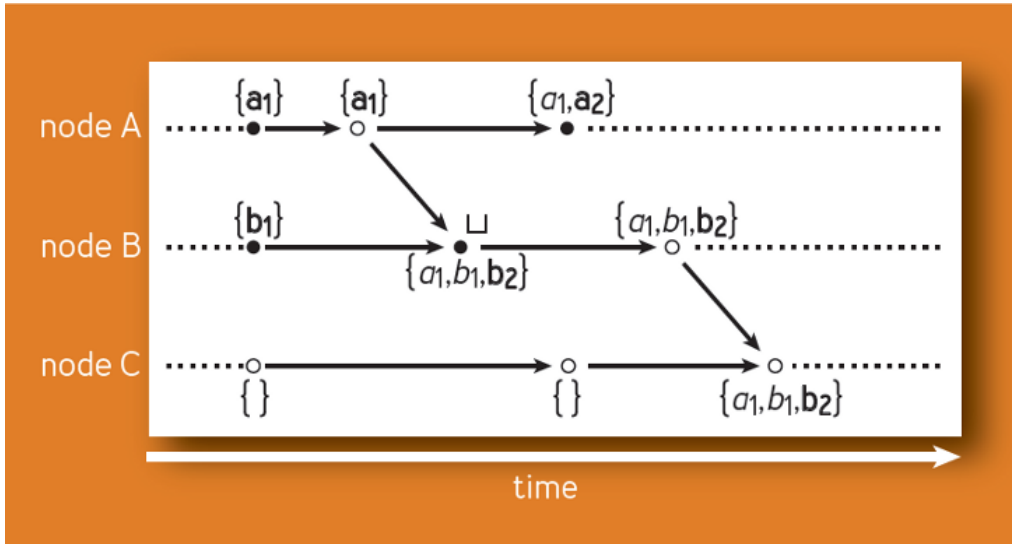
Последняя запись выигрывает?

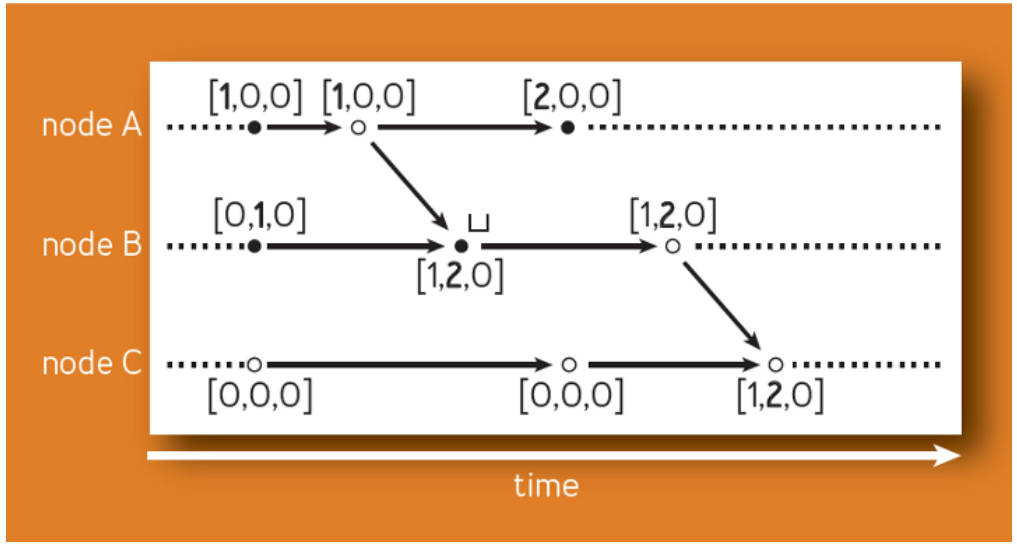


Обнаружение конфликтов при репликации

- Изменения могут приходить на реплики в разном порядке
- Надо уметь определять, когда эти изменения логически связаны
  - изменение B произошло до изменения A
  - значение B заменяет собой A, то есть является более новой версией данных
  - можем понимать, когда надо применить изменение, а когда его надо игнорировать
- Если изменения одновременные, то произошел конфликт
  - изменения B и A не связаны отношением happened-before
  - надо хранить оба значения (siblings), до тех пор, пока конфликт не будет разрешен

Векторы версий (version vectors)





Parker D. Detection of mutual inconsistency in distributed systems (1983)

Векторы версий (version vectors)

- Механизм, используемый для отслеживания логического порядка между версиями реплицируемых данных
- Подход аналогичен векторным часам
  - размер вектора равен числу реплик, изначально все значения равны 0
  - элемент $V[i]$ соответствует числу изменений, выполненных на реплике i
- Обновления векторов
  - при изменении данных на реплике i значение $V[i]$ увеличивается на 1
  - при получении изменения от другой реплики, векторы объединяются (максимум)
- Отслеживание порядка и конфликтов
  - если $V_{local} < V_{update}$, то $local \rightarrow update$ и изменение применяется
  - если $V_{local} \parallel V_{update}$, то возник конфликт (после его разрешения надо увеличить $V[i]$ на 1)
  - в противном случае это изменение уже есть, и его можно игнорировать

Автоматическое разрешение конфликтов

- Conflict-free Replicated Data Types (CRDTs)
  - Operation-based
  - State-based
- Operational Transformation (OT)

См. Kleppmann M. Distributed Systems (video, notes)

Литература

- Kleppmann M. Distributed Systems (часть 3)
- van Steen M., Tanenbaum A.S. Distributed Systems: Principles and Paradigms. Pearson, 2017. (разделы 6.1-6.2)
- Sheehy J. There is No Now: Problems with simultaneity in distributed systems

Дополнительно

- Упомянутые статьи