Распределенные системы: синхронизация

Синхронизация

- Синхронизация часов
- Логические часы
- Голосование
- Взаимное исключение

Синхронизация часов

В централизованных системах время определяется однозначно.

В распределенных системах достичь единого представления времени не так просто.

Пример: работа make.

Физические часы

Встроенные часы обеспечивают локальный «счетчик», позволяющий обеспечить точную синхронизацию локальных процессов.

В распределенных системах каждый участник имеет свои собственные локальные часы, которые не могут быть идеально точно синхронизированы между собой. Происходит рассинхронизация часов (clock skew).

Часы для систем реального времени

Можно использовать внешние физические часы. Причем для эффективности и защищенности желательно иметь несколько таких часов.

Проблемы:

- синхронизация с часами «реального мира»
- синхронизация часов между собой

UTC

Universal Coordinated Time
Универсальное согласованное время —
основа всей системы хранения времени.

Атомная секунда — подсчет переходов атома
цезия-133: 1 сек = 9 192 631 770 переходов.
Используется для обеспечения глобального времени по
атомным часам — TAI.

До изобретения атомных часов использовались
солнечные секунды, которые не являются постоянными
1 день TAI = 86400 сек, что на 3мс меньше солнечного
дня. Для коррекции используются «потерянные
секунды»

Таймер

Каждая машина имеет таймер, который генерирует тики H раз в секунду
Когда время UTC равно t, значение часов машины n равно $C_n(t)$. В идеале $C_n(t) = t$, а $dC/dt = 1$.

Дрейф таймера

Поскольку реальные таймеры имеют ошибку, то соотношение между временем по часам и UTC равно:

1-p <= dC/dt <= 1+p

где p – максимальная скорость дрейфа, определяемая производителем (~10^{-5}).

Если двое часов уходят от реального времени в разные стороны, то разница между ними будет не более **2p*dt**, соответственно, *если мы хотим гарантировать, что пара часов не разойдется более, чем на* **δ**, *то синхронизация часов должна проводиться не реже, чем каждые* **δ/2p** *с.*

Существуют 2 типа распределенных систем:

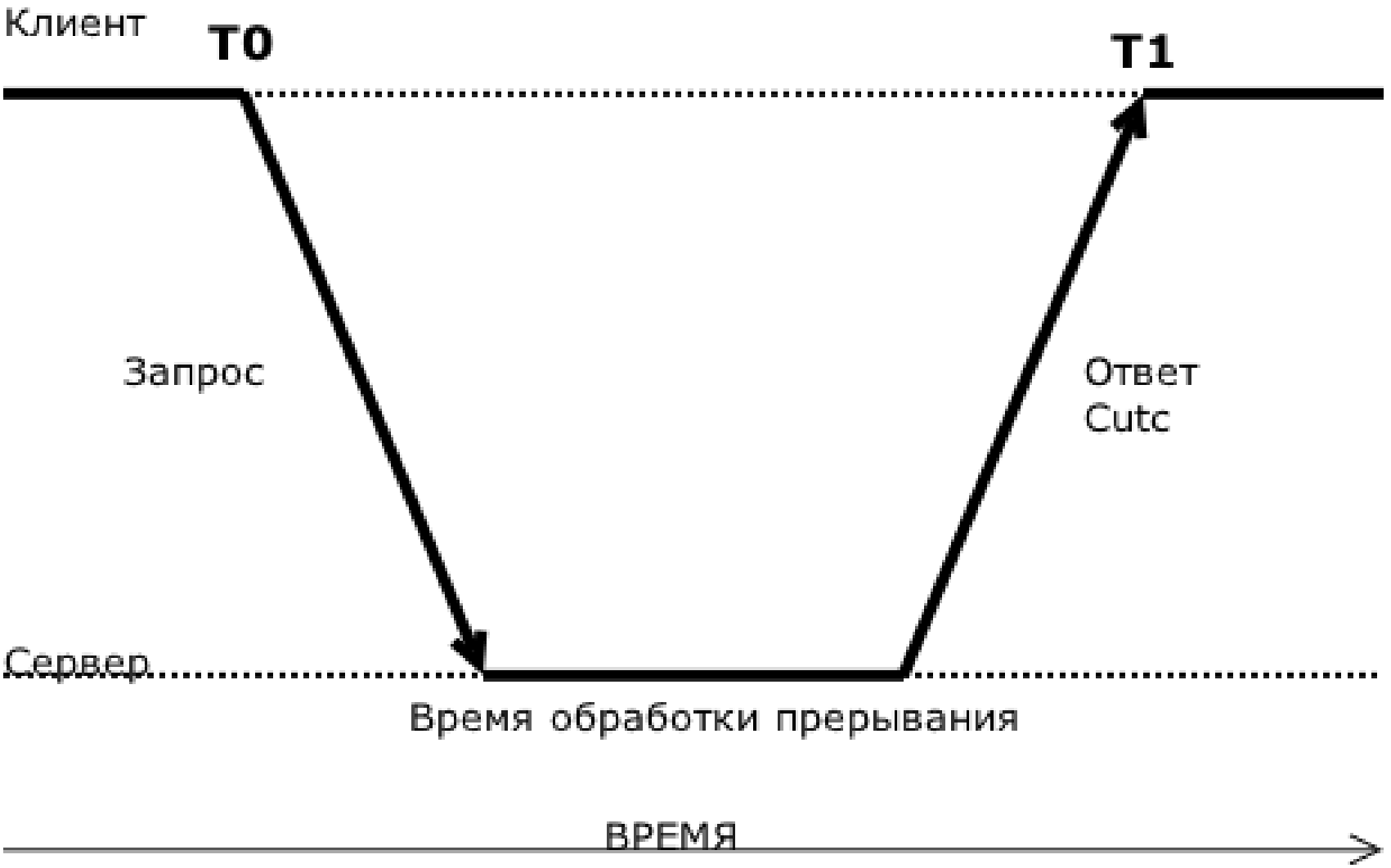
- С подключением к источнику точного времени
- Без подключения к источнику точного времени

Алгоритм Кристиана

Подходит для систем, в которых одна из машин имеет подключение к источнику точного времени — сервер точного времени.

Периодически, но не реже **δ/2p**, каждая машина посылает серверу времени запрос, на которое он отвечает, передавая свое текущее время.

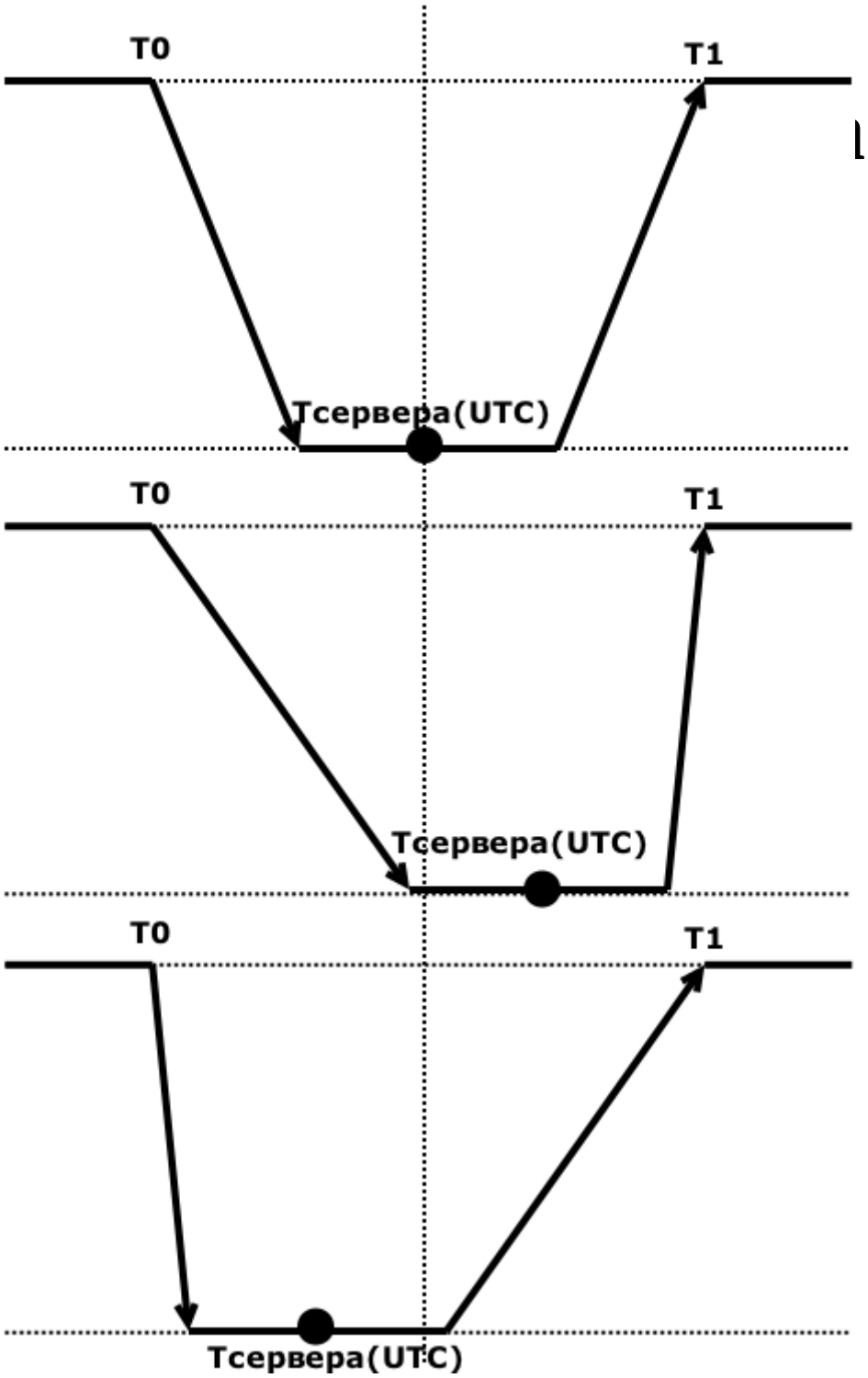
Алгоритм Кристиана



Проблемы алгоритма Кристиана

• Время никогда не течет назад — нельзя просто подставить полученное от сервера время, если это приведет к «переводу стрелок назад»

• Время передачи — ненулевое.
Приблизительно «правильное» можно вычислить с помощью выражения (T1-T0)/2


T0
T1
Tсервера(UTC)
T0
T1
Tсервера(UTC)
T0
T1
Tсервера(UTC)

Алгоритм Беркли

Для машин не имеющих синхронизации с источником точного времени.

Демон является активным и периодически опрашивает все машины в сети.
Усредненные значения рассылает по сети.
Машины опаздывающие – подгоняют время точно, спешащие либо устанавливают, либо замедляются, пока не будет достигнута синхронизация.

Усредняющие алгоритмы

Алгоритмы Кристиана и Беркли — слишком централизованные.

Пример децентрализованного алгоритма:
Каждая машина широковещательно рассылает свое текущее время. Т. к. часы на машинах не строго синхронизированы, то отсылаться они будут в разное время.
Каждая машина прослушивает сеть в течении какого-то времени и собирает широковещательные пакеты от всех машин. Крайние значения отбрасываются, а остальные используются для вычисления среднего времени и установки локальных часов.

Логические часы

Логические часы

Часто необходимо, чтобы все машины использовали единое представление времени. Не обязательно, чтобы оно совпадало с истинным временем.

Существует класс алгоритмов, для которых внутренняя непротиворечивость имеет большее значение, чем то, насколько их время близко к реальному.

Лампорт в статье «Time, Clocks, and the Ordering of Events in a Distributed System» показал, что синхронизация часов в распределенной системе не обязательно должна быть абсолютной.
А если два процесса не связаны между собой, то нет необходимости, чтобы их часы были синхронизированы.

Обычно *имеет значение не точное время выполнения процессов, а его порядок*.

Алгоритм Лампорта

Алгоритм синхронизации логических часов основывается на отметках времени Лампорта (Lamport timestamps).

Для синхронизации логических часов Лампорт определил отношение под названием “происходит раньше”.

Выражение **a->b** читается как “**a** происходит раньше **b**” и означает,
что *все процессы согласны с тем, что первым происходит событие **a**, а позже – **b**.*

Отношение “происходит раньше” исполняется в двух случаях:

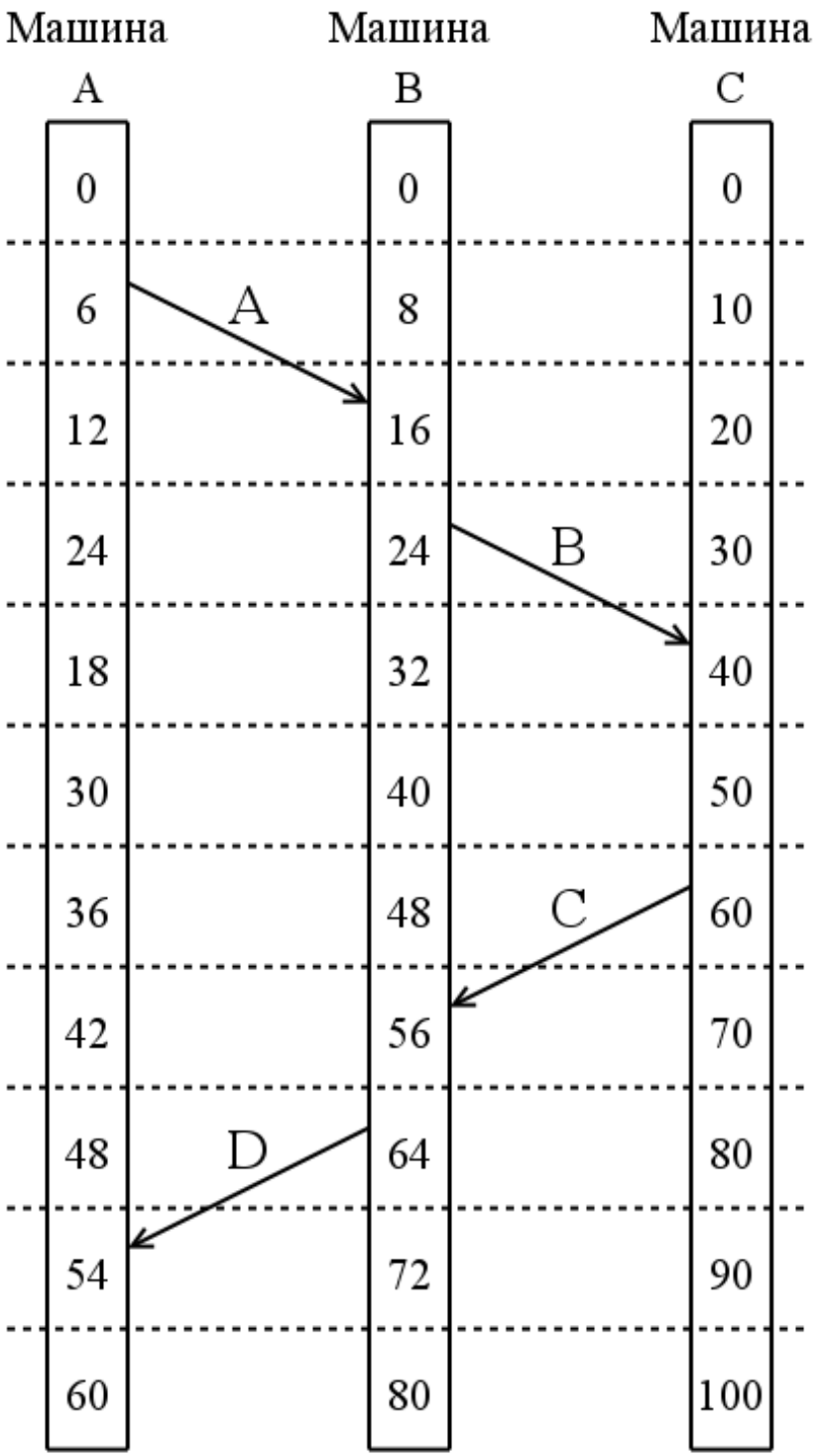
• если **a** и **b** – события, происходящие в одном и том же процессе, и **a** происходит раньше, чем **b**, то отношение **a->b** истинно

• Если **a** – событие отсылки сообщения одним процессом, а **b** – событие получения того же сообщения другим процессом, то отношение **a->b** также истинно.

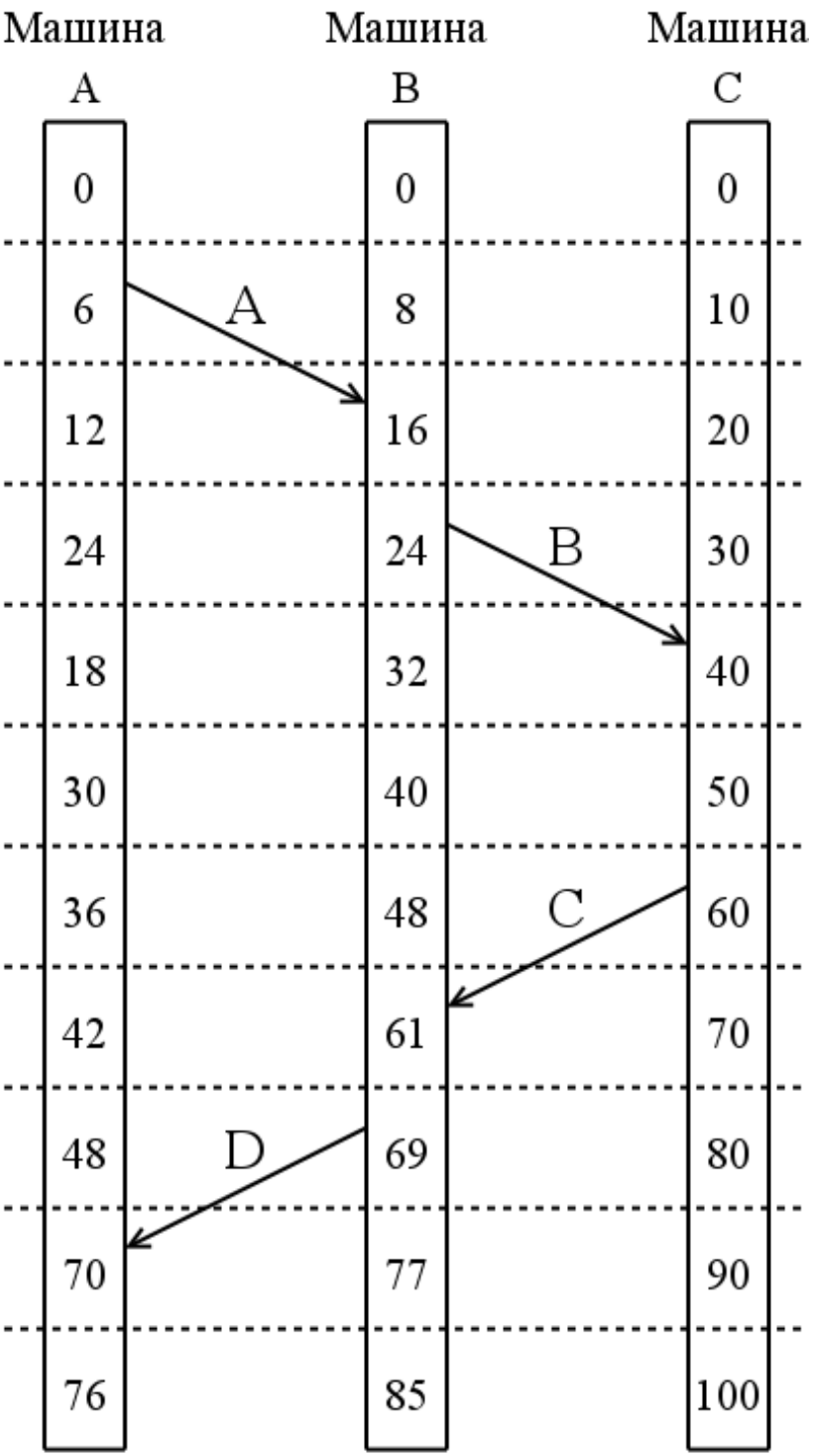
Сообщение не может быть получено раньше или в то же время, т.к. на отсылку требуется конечное ненулевое время.

Отношение “происходит раньше” - транзитивное, т.е. если **a->b** и **b->c**, то **a->c**.

Если события **x** и **y** происходят в разных процессах, которые не обмениваются между собой сообщениями, то отношение **x->y** *не истинно*, так же как и **y->x**.
Такие события называются параллельными (concurrent)


Машина A
Машина B
Машина C
0
6
12
24
18
30
36
42
48
54
60
0
8
16
24
32
40
48
56
64
72
80
0
10
20
30
40
50
60
70
80
90
100
A
B
C
D

а


Машина A
Машина B
Машина C
0
6
12
24
18
30
36
42
48
70
76
0
8
16
24
32
40
48
61
69
77
85
0
10
20
30
40
50
60
70
80
90
100
A
B
C
D

б

Дополнения к отметкам времени Лампорта

Между любыми двумя событиями часы должны «протикать» как минимум один раз.

В некоторых случаях необходимо удовлетворить дополнительное требование: никакие два сообщения не должны происходить одновременно.
Можно использовать составные отметки времени, например: timestamp.processID (10.3, 32.1 и т.п.)

Банковская проблема: как ~~обмануть вкладчика~~ начислить проценты?

Предположим, что у банка существует 2 копии БД в различных городах. У вкладчика 1000$ на счету и он хочет добавить еще 100$ на свой счет.
Одновременно происходят 2 операции:
- добавление денег на счет в банке города A
- начисление процентов (1%) на вклад в городе B

Риторический вопрос: ~~сколько денег получит банк~~ сколько денег будет на счете вкладчика?

1) 1000 + 100 + (1000 + 100)*0.01 = 1111$
2) 1000 + 100 + (1000)*0.01 = 1110$

Полностью упорядоченная групповая рассылка

В «банковской проблеме» необходимо, чтобы операции обновления были выполнены в одинаковом порядке в каждой из копий БД.
Требуется использование полностью упорядоченной групповой рассылки (totally-ordered multicasting) — это операция, при которой все сообщения доставляются всем получателям с одинаковой очередностью.
Предполагается, что все сообщения принимаются всеми участниками (включая отправителя), а связь надежна. Каждое сообщение имеет отметку времени с текущим логическим временем отправителя.

Полностью упорядоченная групповая рассылка

Когда процесс принимает сообщение, оно попадает в локальную очередь в порядке, определяемом отметкой времени.
Получатель рассылает групповое подтверждение всем процессам.

* Интересная особенность: все процессы имеют абсолютно идентичную локальную очередь запросов.

Векторные отметки времени

При использовании отметок времени Лампорта *все события в системе имеют единую последовательность*.
Однако ничего неизвестно о взаимосвязи между двумя сообщениями **a** и **b**, если просто сравнить их временные метки **C(a)** и **C(b)**. Т. е. если **C(a)<C(b)**, то это не означает, что событие **a**, действительно произошло раньше, чем событие **b**.

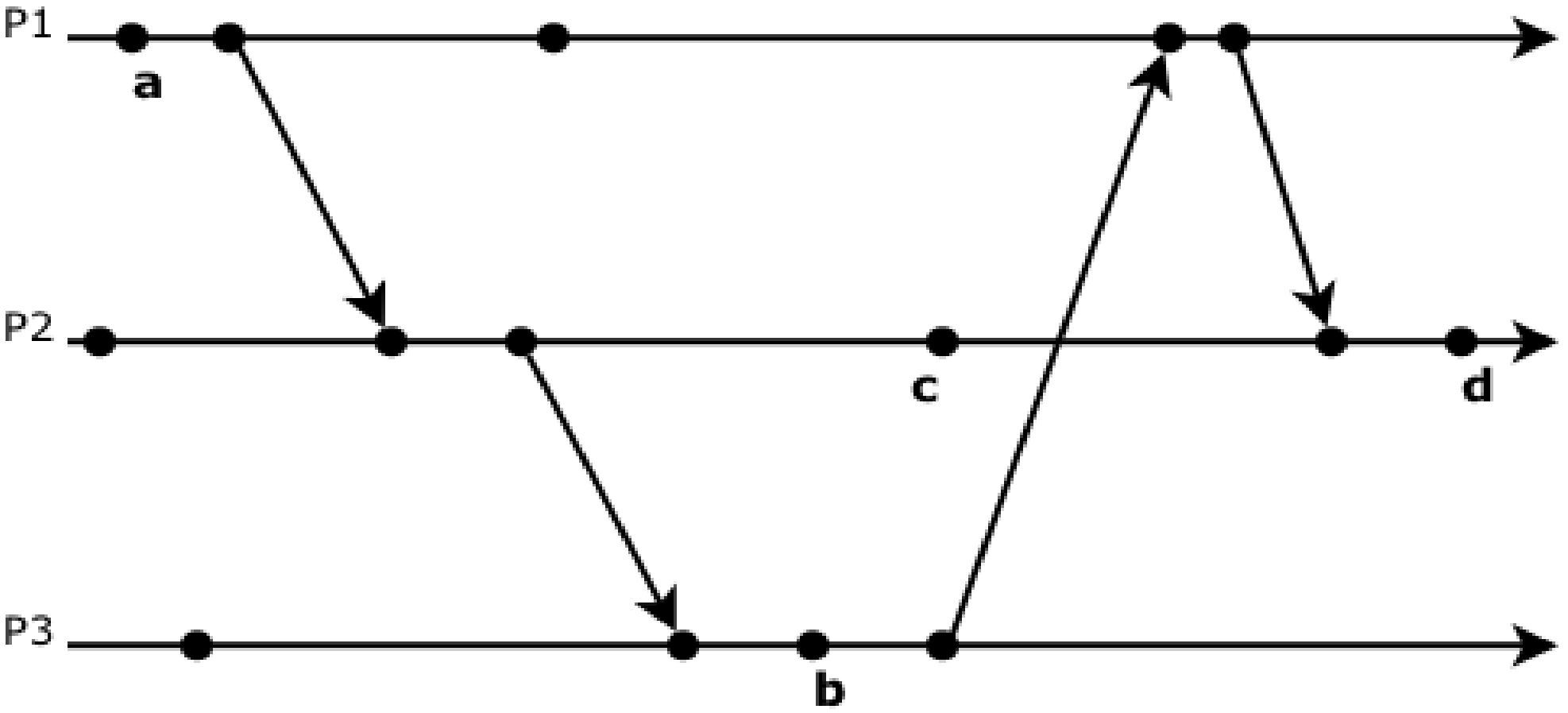
Причинно-следственная связь может быть соблюдена с помощью векторных отметок времени (vector timestamps).

Векторные отметки времени

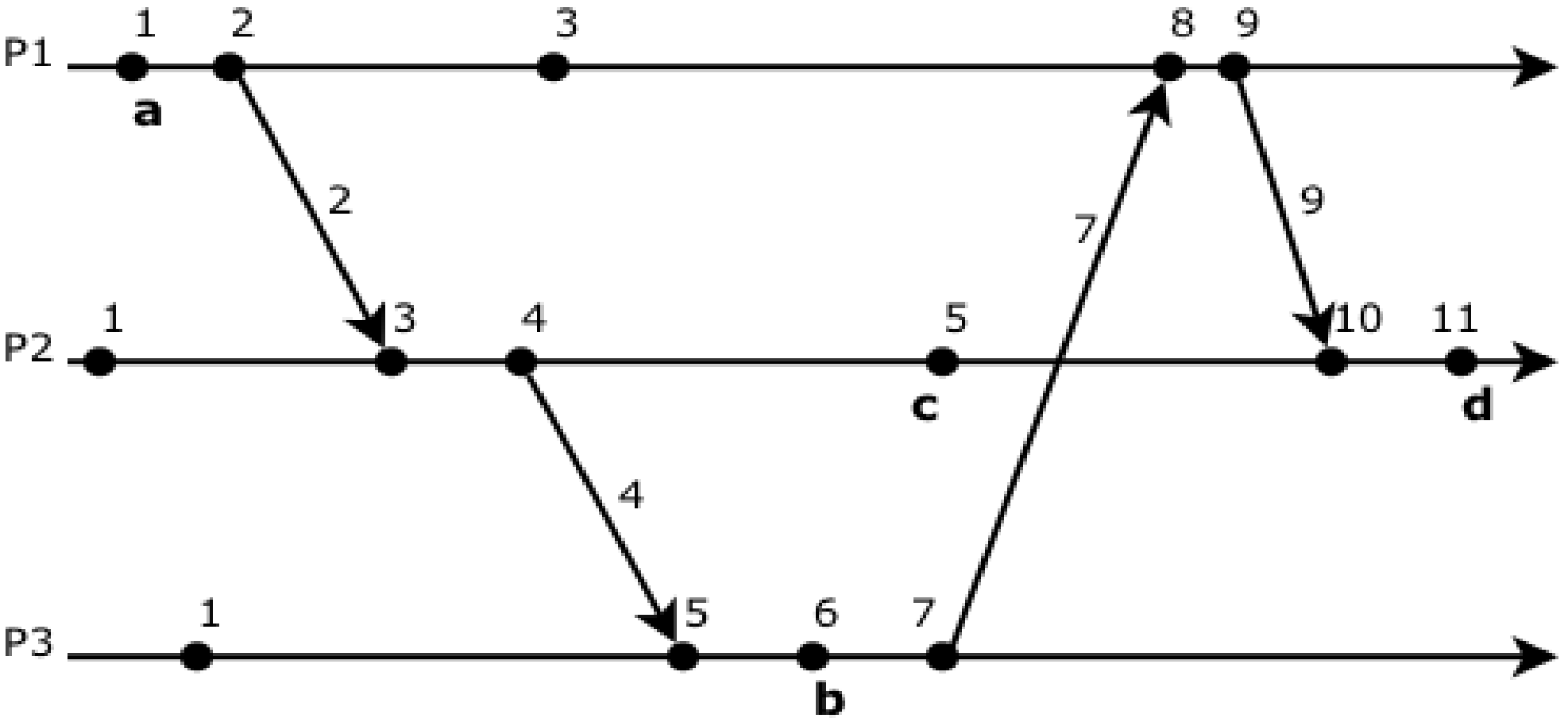
Векторная отметка времени **VT(a)**, присвоенная событию **a**, имеет следующее свойство:
если **VT(a)<VT(b)** для события **b**, значит событие **a** является причинно предшествующим событию **b**.

Пример

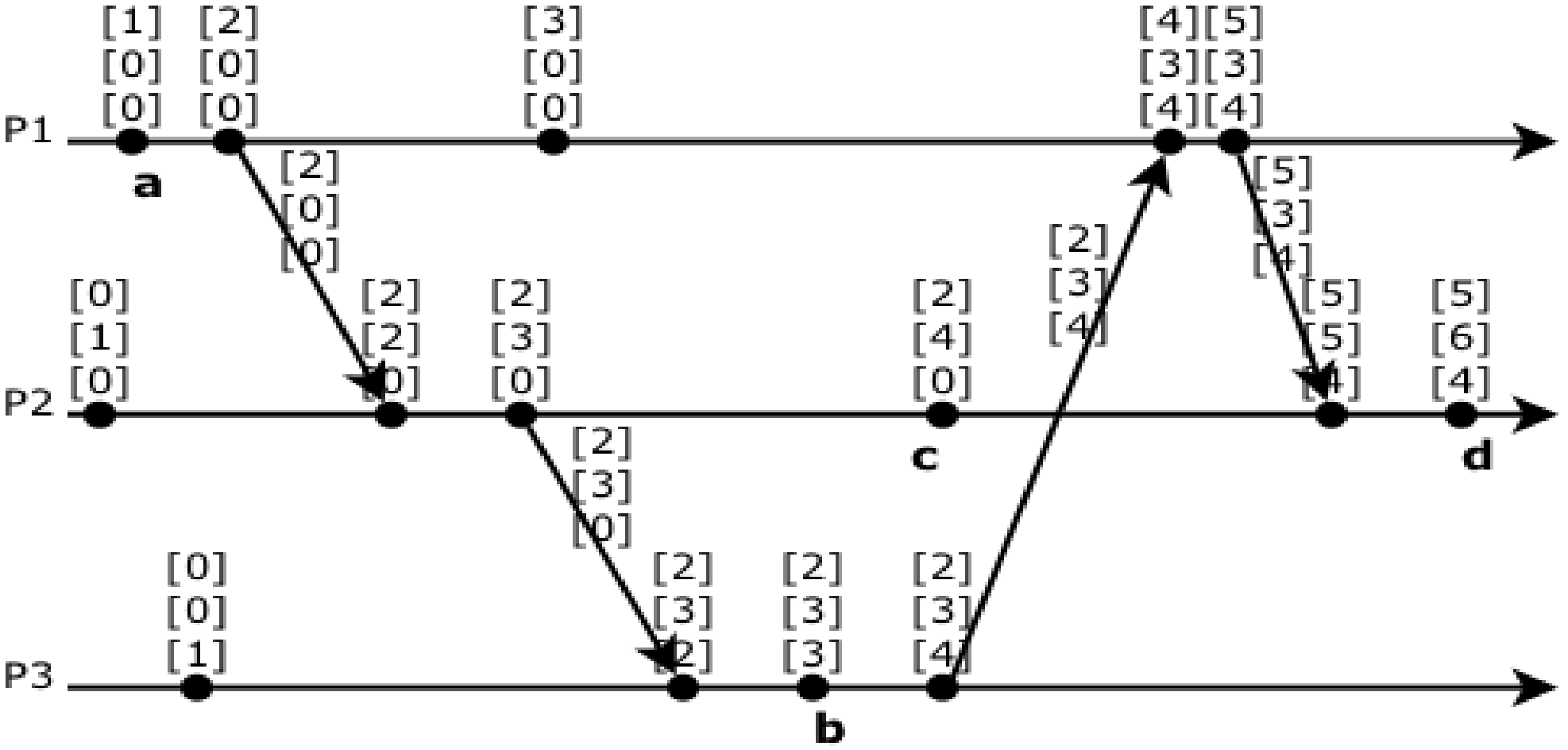
Временная диаграмма взаимодействия в распределенной системе



Использование отметок времени Лампорта



Использование векторных отметок времени



Алгоритмы голосования

Многие распределенные алгоритмы требуют, чтобы один из процессов был координатором, инициатором или выполнял другую роль. Не важно какой именно это будет процесс, главное, чтобы он существовал.

Если все процессы одинаковы, то способа выбрать один из них не существует. Поэтому применяются различные системы голосования. При этом считаем, что у них есть какие-либо уникальные идентификаторы и они не знают, какие процессы работают, а какие нет.

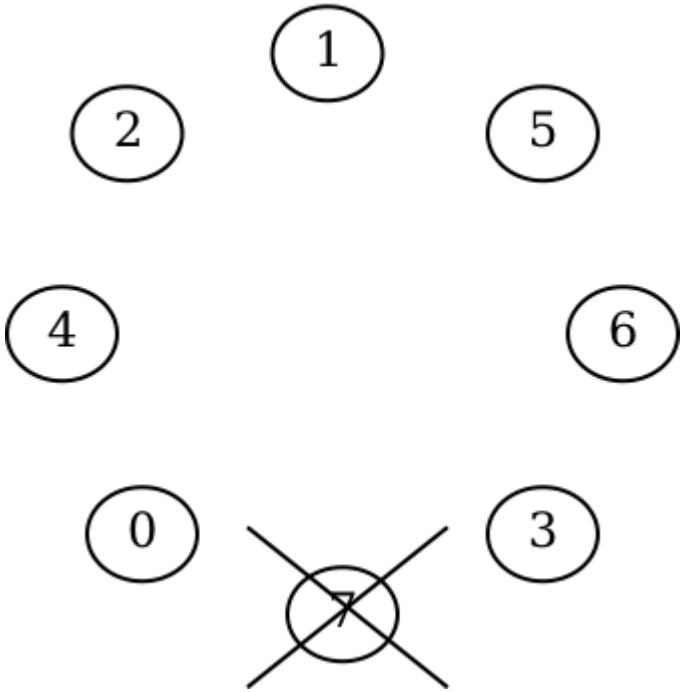
Алгоритм забияки

Когда один из процессов замечает, что координатор не отвечает, он инициирует голосование:

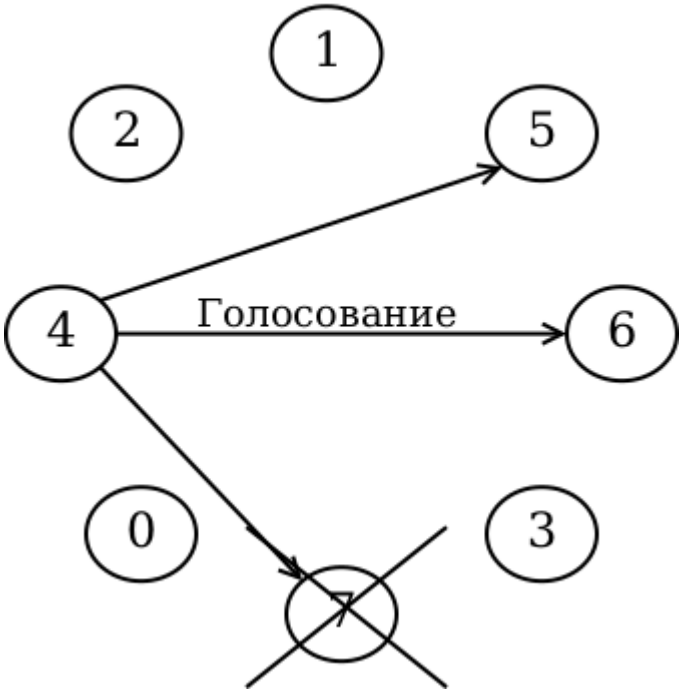
- Процесс рассылает сообщение для всех процессов с большим, чем у него номером.

- Если никто не отвечает, то процесс выигрывает голосование и становится координатором, если же отвечает, то работа текущего процесса на этом закончена

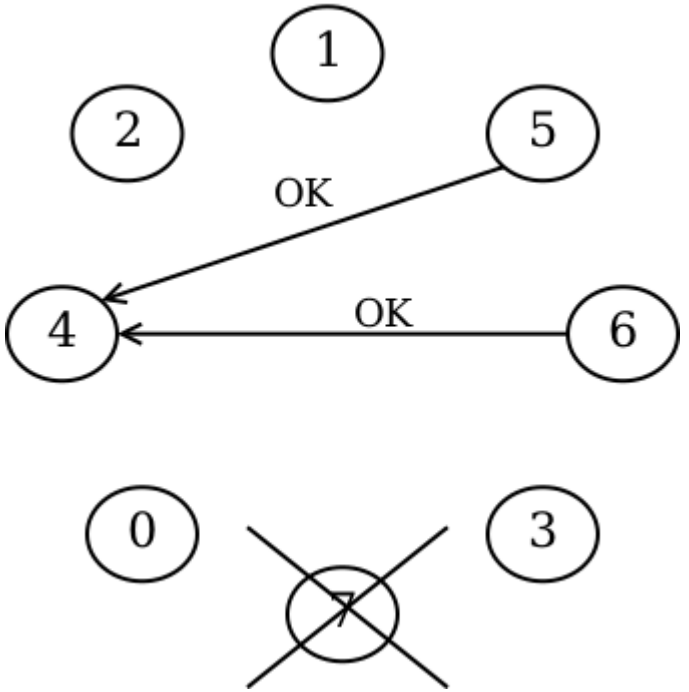
Алгоритм забияки: начальное состояние



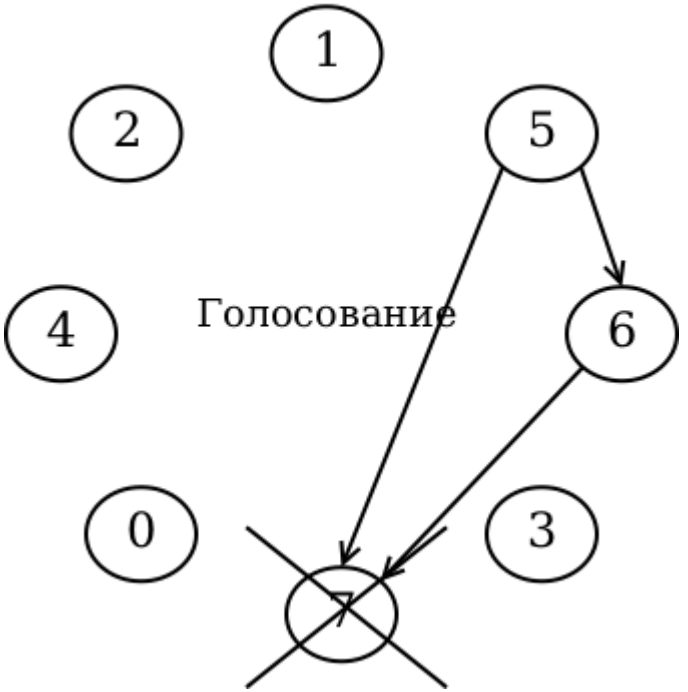
Алгоритм забияки:
Шаг 1



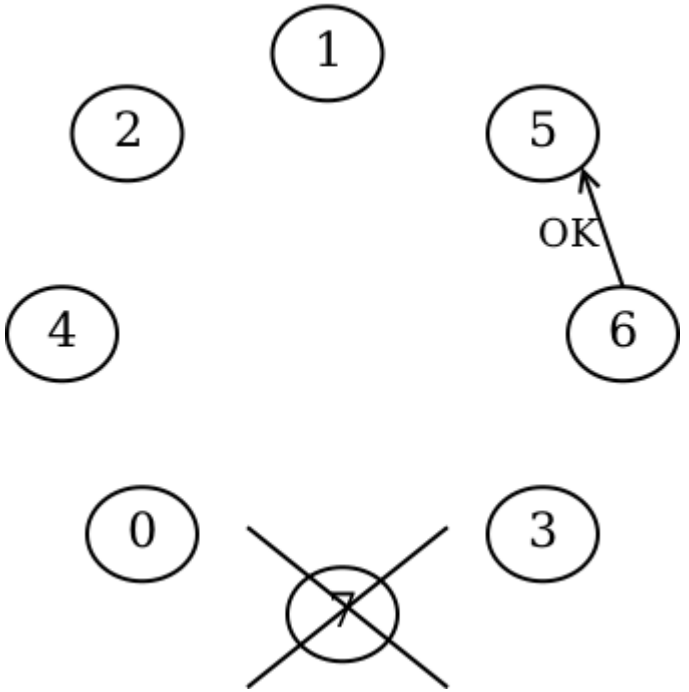
Алгоритм забияки:
Шаг 2



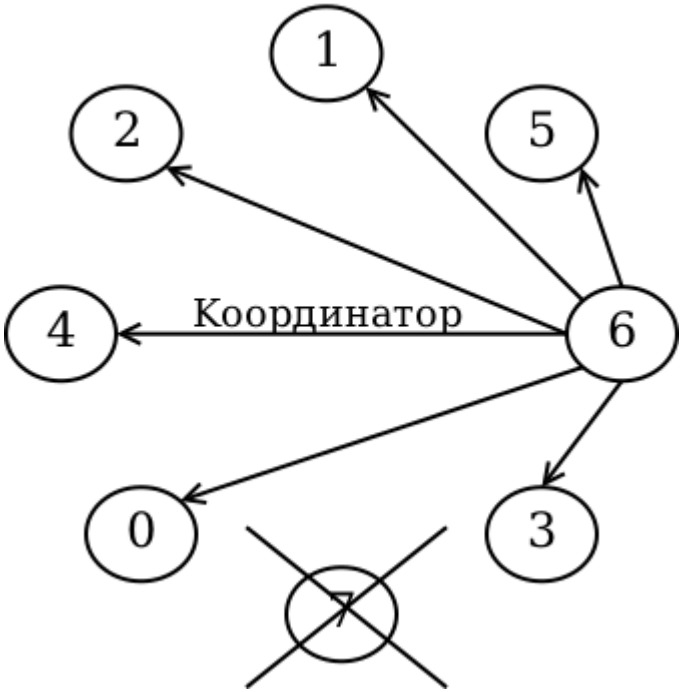
Алгоритм забияки:
Шаг 3



Алгоритм забияки:
Шаг 4



Алгоритм забияки:
Шаг 5

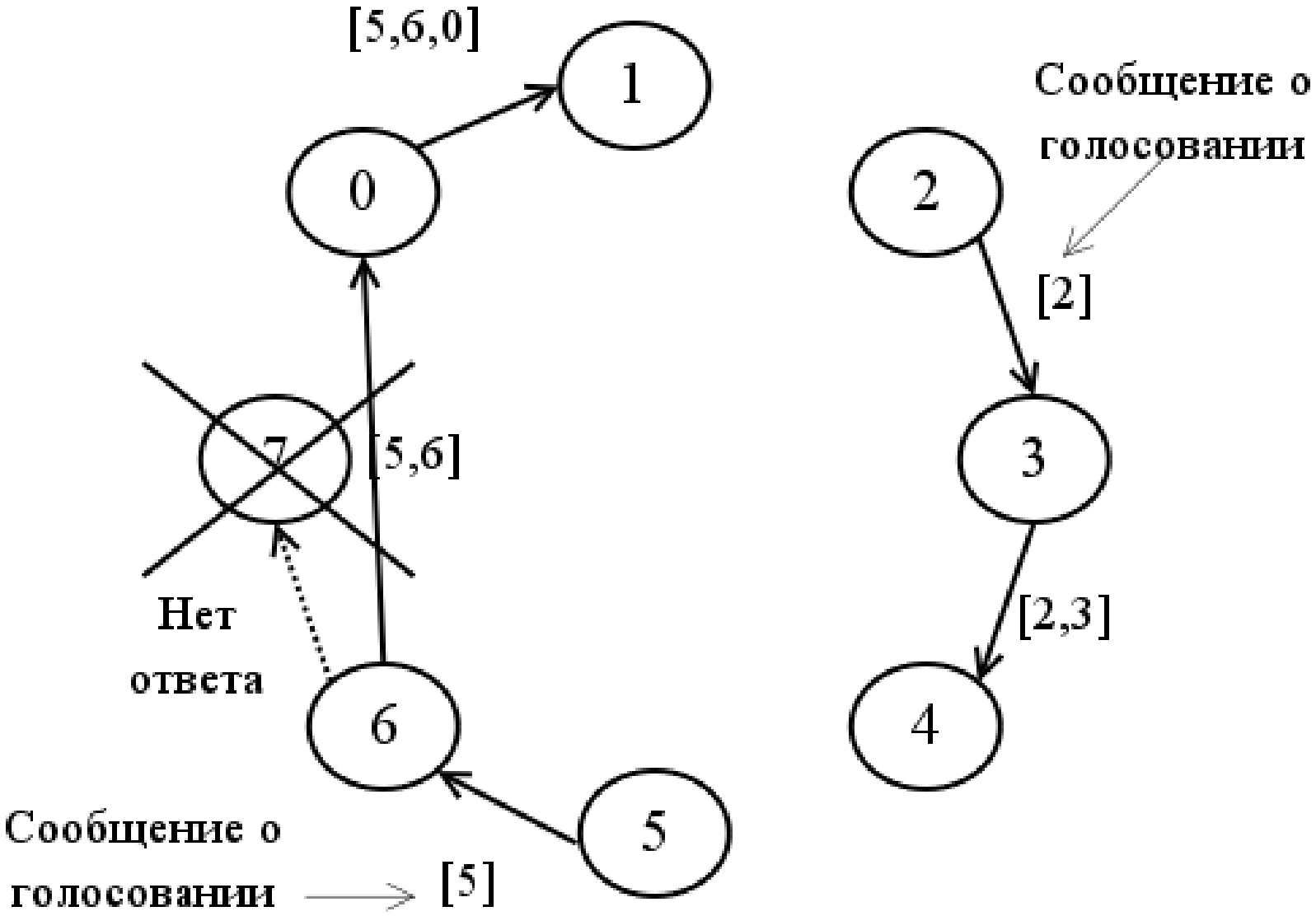


Кольцевой алгоритм голосования

Основывается на использовании кольца.
Маркер не требуется.

- Когда один из процессов обнаруживает, что координатор не работает, он запускает по кольцу сообщение ГОЛОСОВАНИЕ
- Каждый процесс добавляет к сообщению свой номер и пересылает дальше по кольцу адресату, способному принять сообщение
- Начальный процесс получает маркер назад и вычисляет нового координатора, после чего запускает еще одно сообщение КООРДИНАТОР, в котором объявляется координатор.

Кольцевой алгоритм голосования



Взаимное исключение

Системы, состоящие из множества процессов, проще всего программировать, используя критические области.

В обычных операционных системах такие взаимные исключения регулируются с помощью семафоров, мониторов и других конструкций подобного рода.

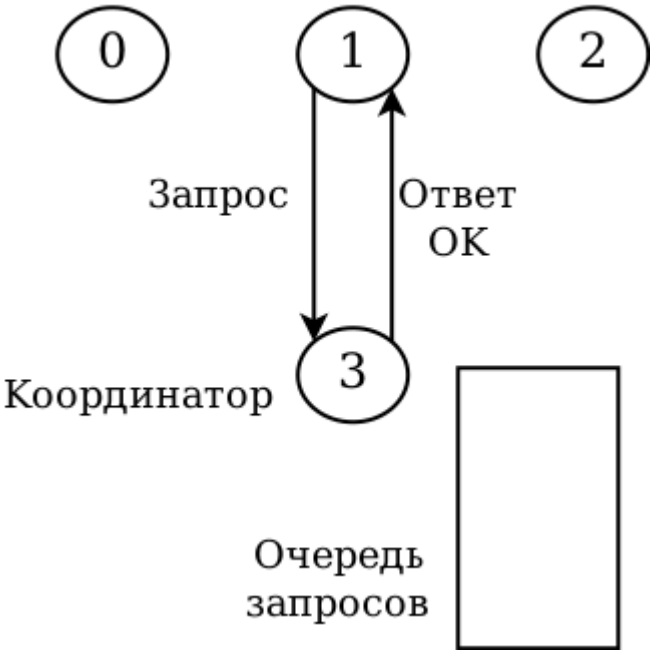
Для распределенных систем существует множество алгоритмов, призванных реализовать такие конструкции, однако практически все они базируются на нескольких алгоритмах.

Централизованный алгоритм

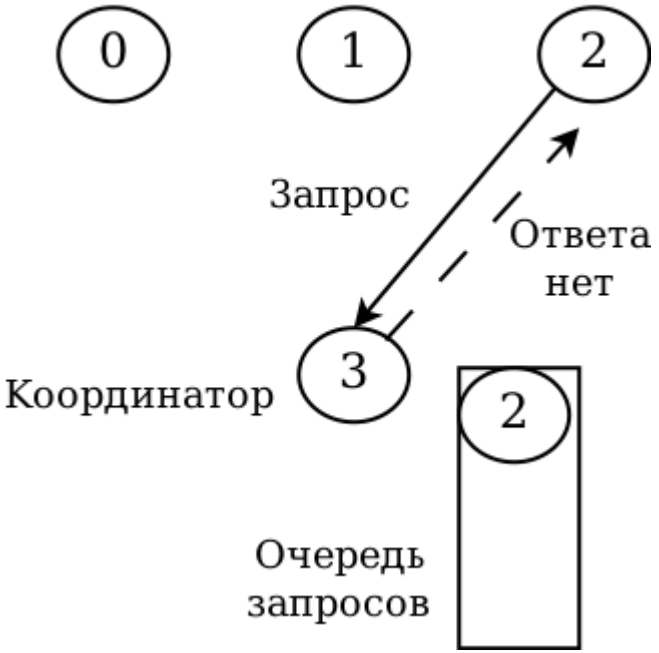
•Один из процессов выбирается координатором.
• Процесс не являющийся координатором, при входе в критическую секцию сначала уведомляет координатора, сообщая, в какую критическую область он собирается войти, и спрашивает на это разрешение.
•Если критическая область не используется другим процессом, то координатор разрешает ее использовать, если же используется – запрещает.

Конкретные способы разрешения/запрета зависят от используемой системы.

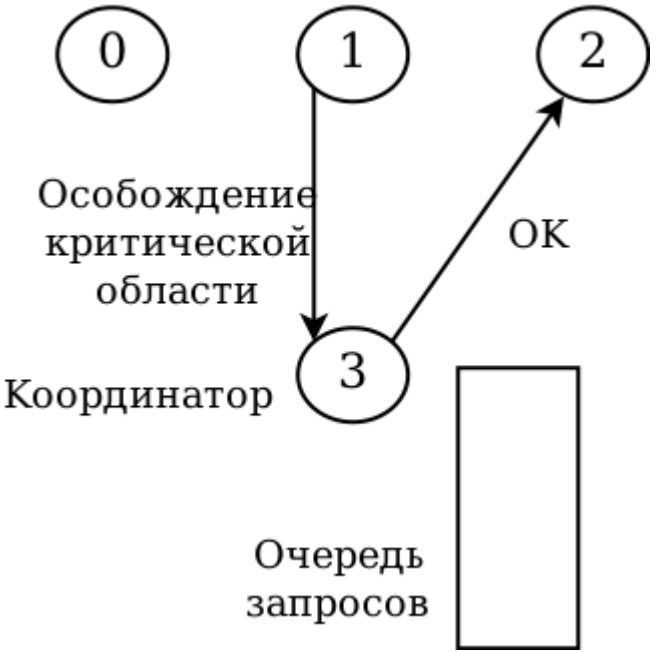
Централизованный алгоритм
Шаг 1



Централизованный алгоритм
Шаг 2



Централизованный алгоритм
Шаг 3



Алгоритм маркерного кольца

Создается логическое кольцо, в котором каждому процессу назначен свой логический номер.

• При инициализации кольца процесс 0 получает маркер (токен). Маркер циркулирует по кольцу.

• Когда процесс получает маркер, он проверяет, не нужно ли ему войти в критическую область. Если нужно, то он входит в критическую область, работает там и выходит. После этого маркер передается дальше. Вход в другую критическую область, используя один и тот же маркер запрещен.

Распределенный алгоритм

В этом алгоритме требуется наличие полной упорядоченности событий в системе (один из способов – алгоритм Лампорта).

1. Когда процесс собирается войти в критическую область, он создает сообщение, в котором указывает свой номер, имя критической области и текущее время. Затем это сообщение рассылается всем процессам.

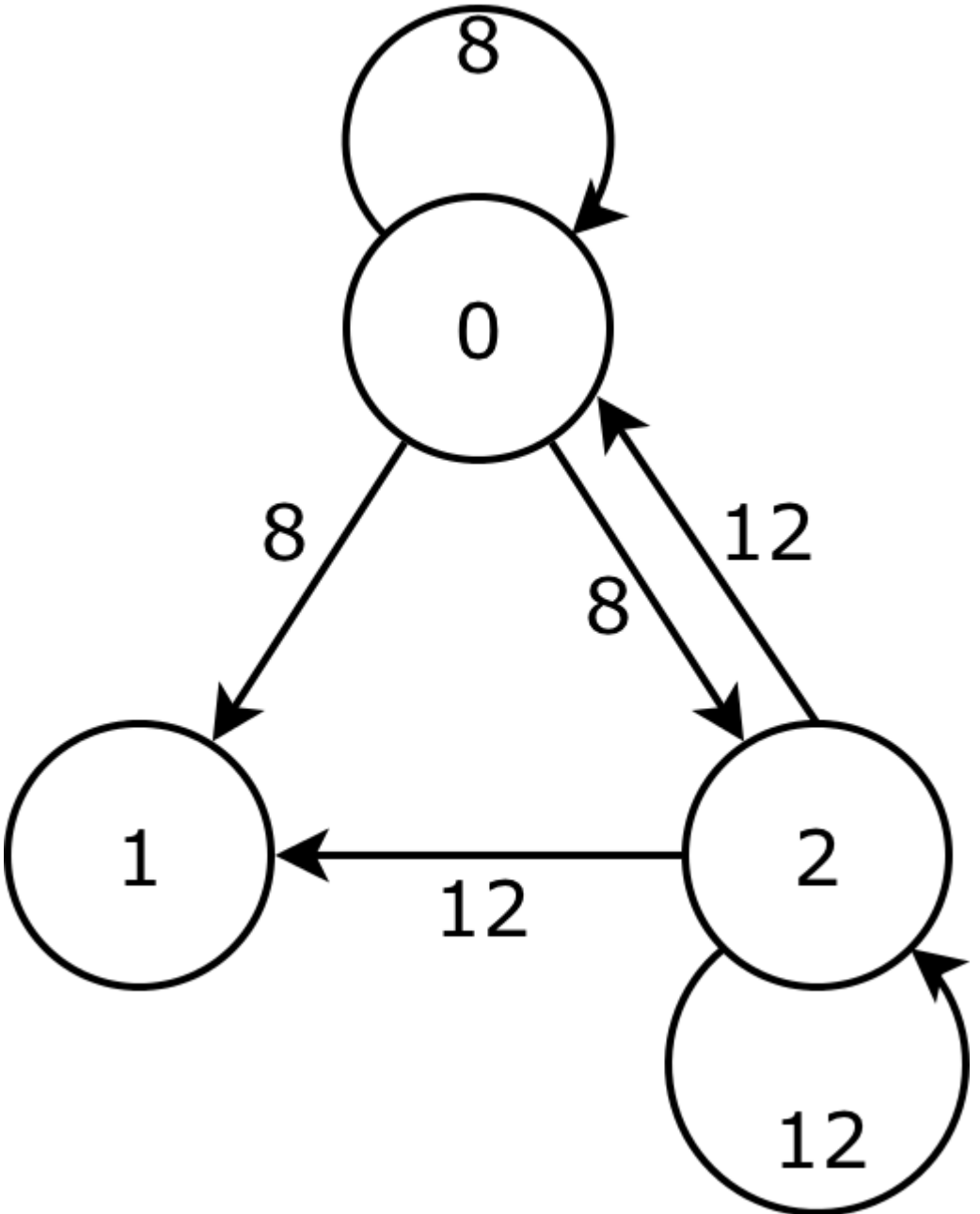
2. При получении процессом сообщения с запросом алгоритм делится на 3 варианта

1) если получатель не находится в критической области и не собирается ее использовать, то отсылает сообщение OK

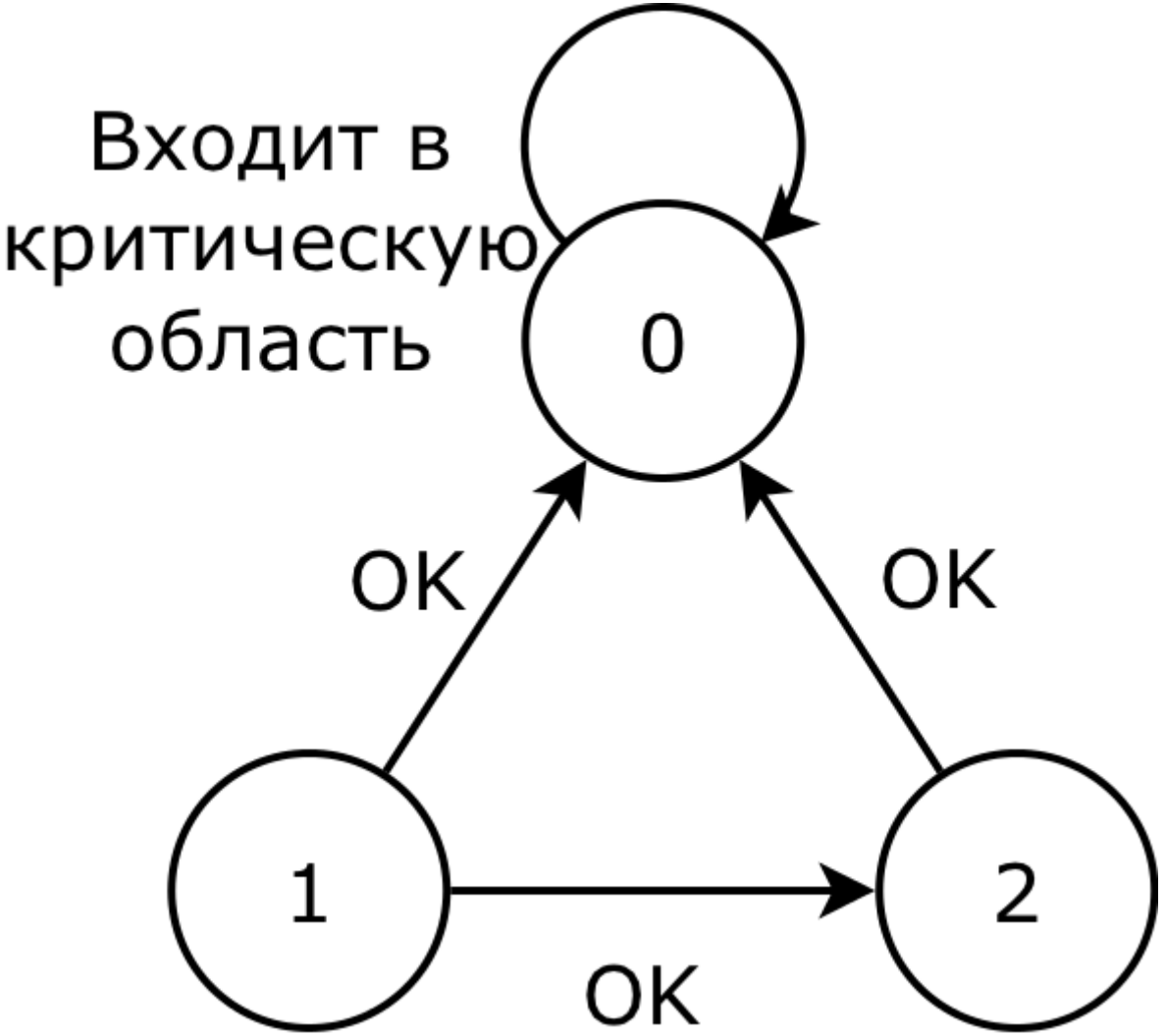
2) если получатель находится в критической области, то не отвечает, а помещает запрос в очередь

3) если получатель собирается войти в критическую область, но еще не сделал этого, он сравнивает метку времени пришедшего сообщения с меткой времени своего сообщения. Выигрывает минимальное. Если пришедшее сообщение имеет меньший номер, то получатель отвечает OK. Если же метка времени меньше на своем сообщении, то пришедшее сообщение ставится в очередь, при этом ничего не отсылается.

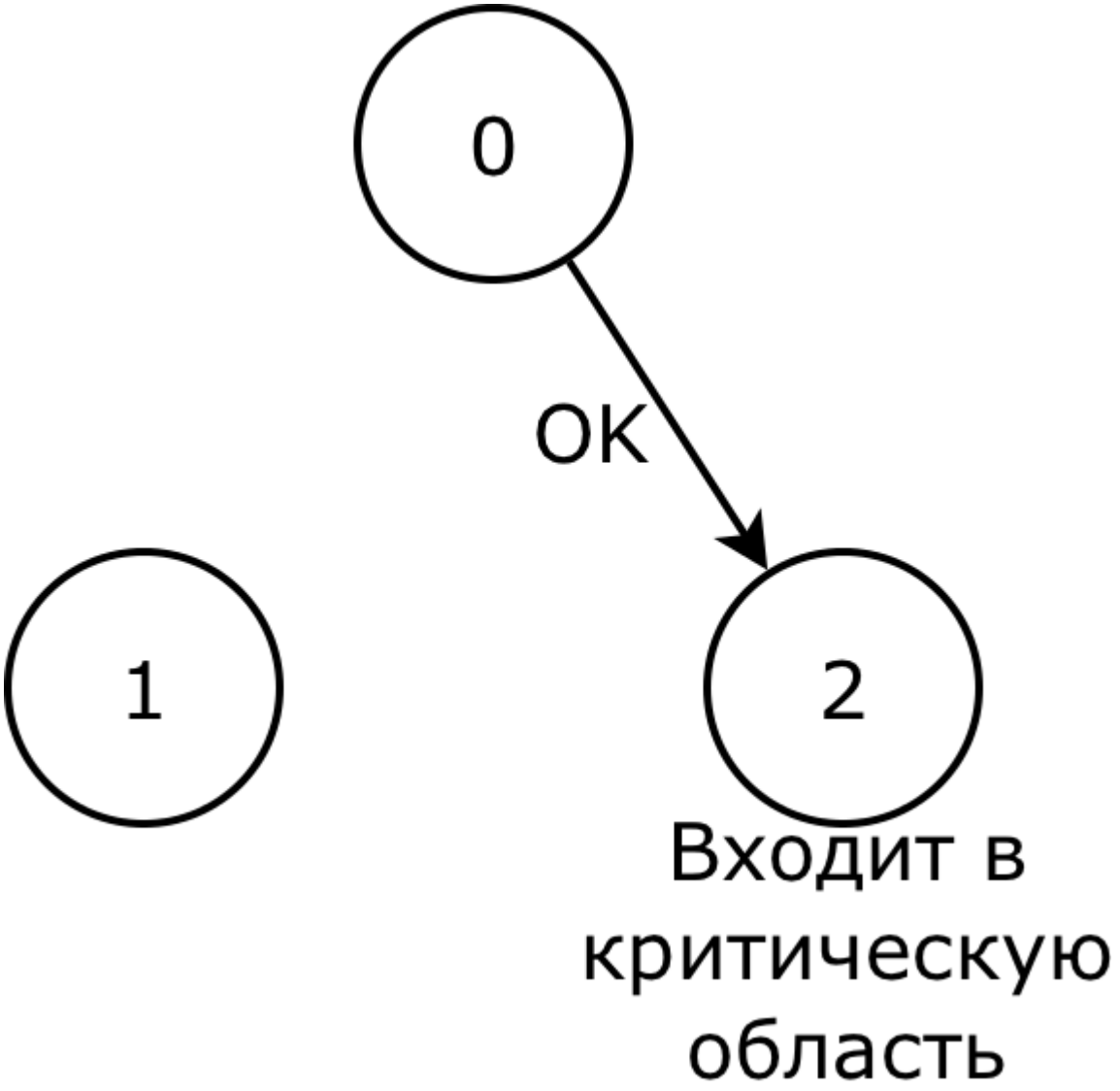
Распределенный алгоритм:
Шаг 1



Распределенный алгоритм: Шаг 2



Распределенный алгоритм: Шаг 3



Сравнение алгоритмов взаимного исключения

Алгоритм	Число сообщений на вход-выход	Задержки перед входом (в сообщениях)	Возможные проблемы
Централизованный	3	2	Сбой координатора
Распределенный	2*(n-1)	2*(n-1)	Сбой в одном из процессов
Маркерного кольца	От 1 до ∞	От 1 до n-1	Потеря маркера, сбой в одном из процессов