

Порядок сообщений



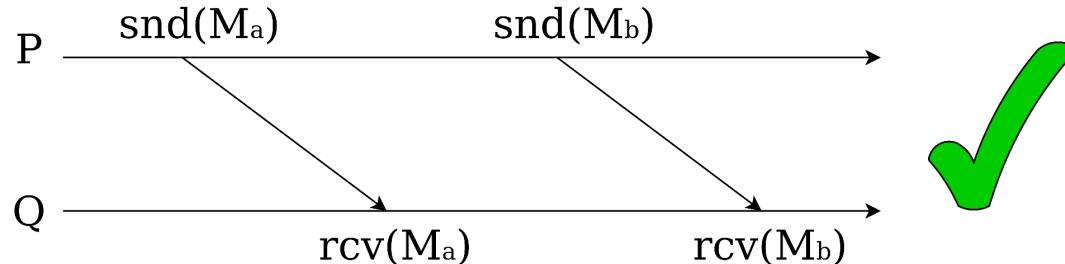
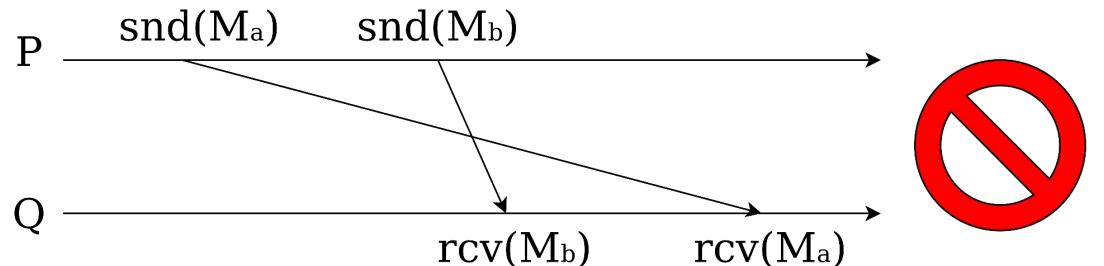
Илья Кокорин

kokorin.ilya.1998@gmail.com

FIFO

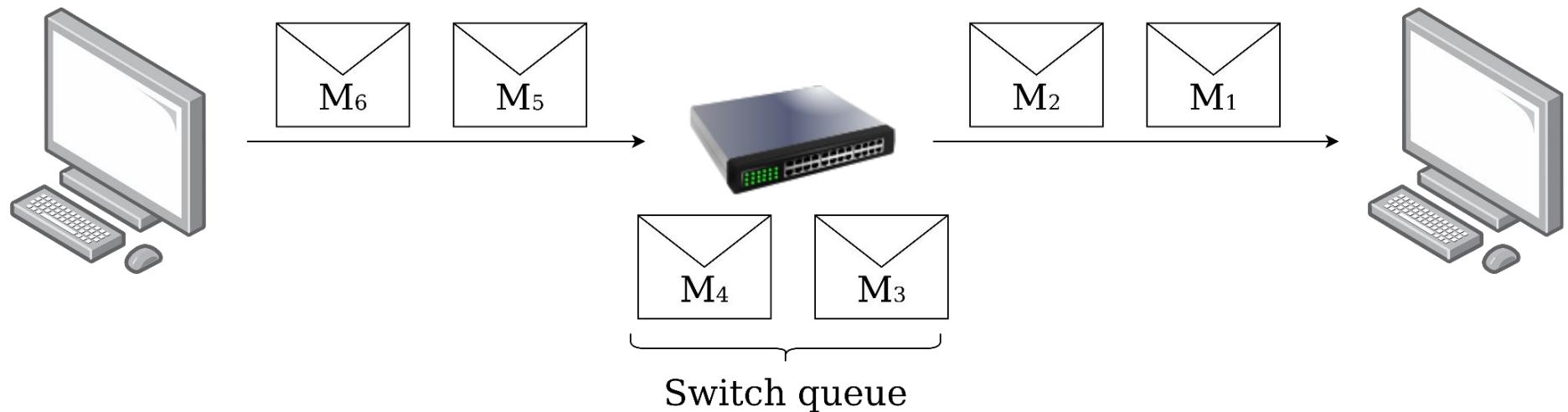
- Гарантируется ли UDP?

$$\forall M_a, M_b \in M_{P \rightarrow Q} : \text{snd}(M_a) < \text{snd}(M_b) \Rightarrow \text{rcv}(M_a) < \text{rcv}(M_b)$$



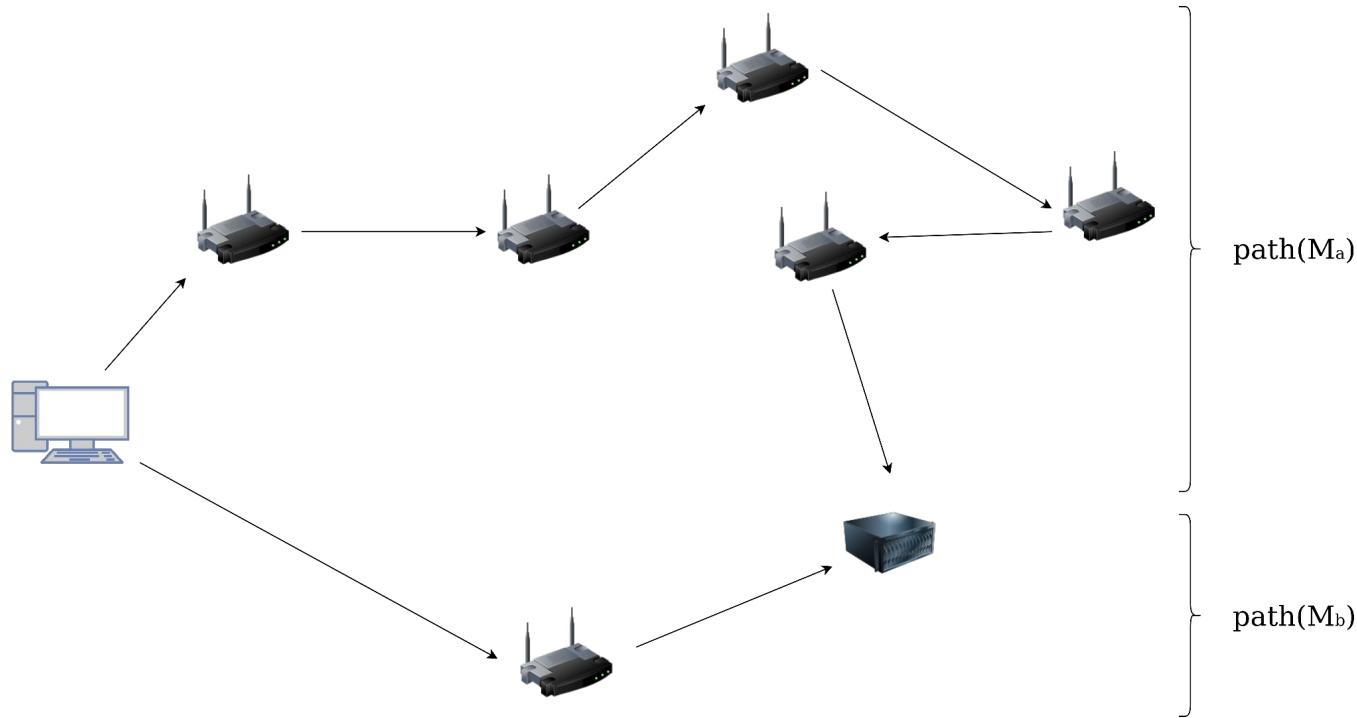
FIFO

- Можно гарантировать на аппаратном уровне
 - По проводу сообщения передаются FIFO
 - В коммутаторе FIFO-очередь
- В локальной сети



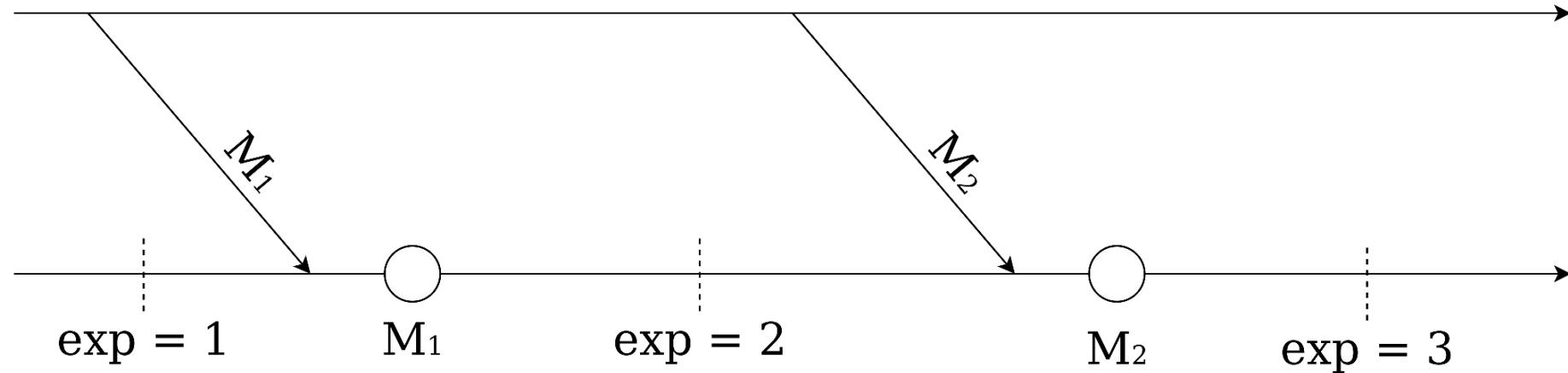
FIFO

- Может быть нарушен в глобальной сети



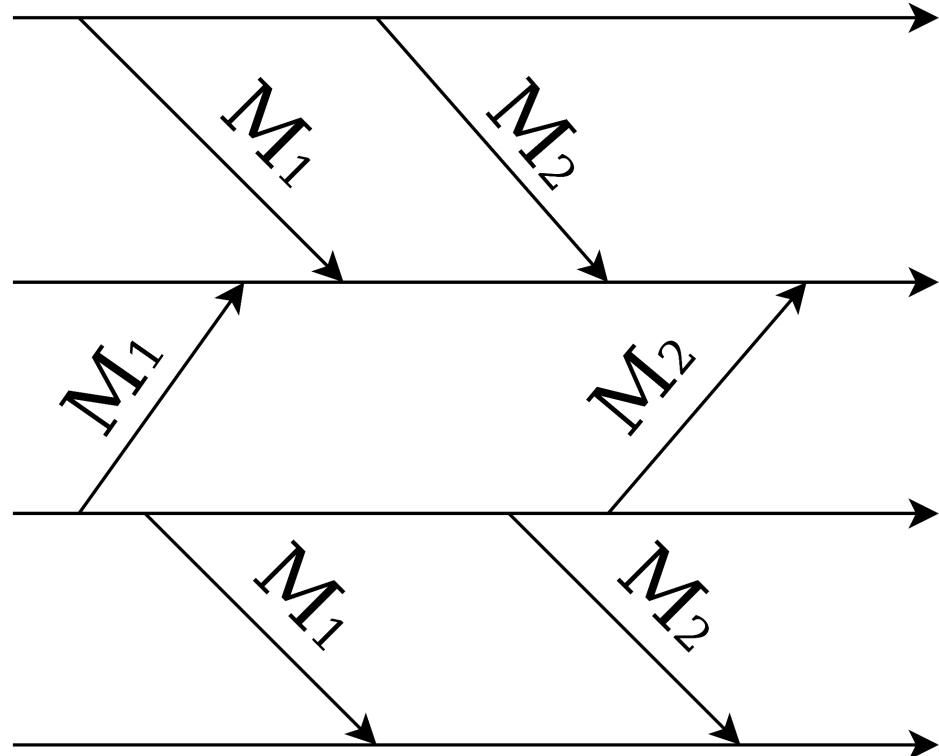
FIFO

- Нумеруем сообщения между каждой парой процессов
- К каждому сообщению прикрепляем номер
- Принимающая сторона обрабатывает сообщение если номер пришедшего равен номеру ожидаемого



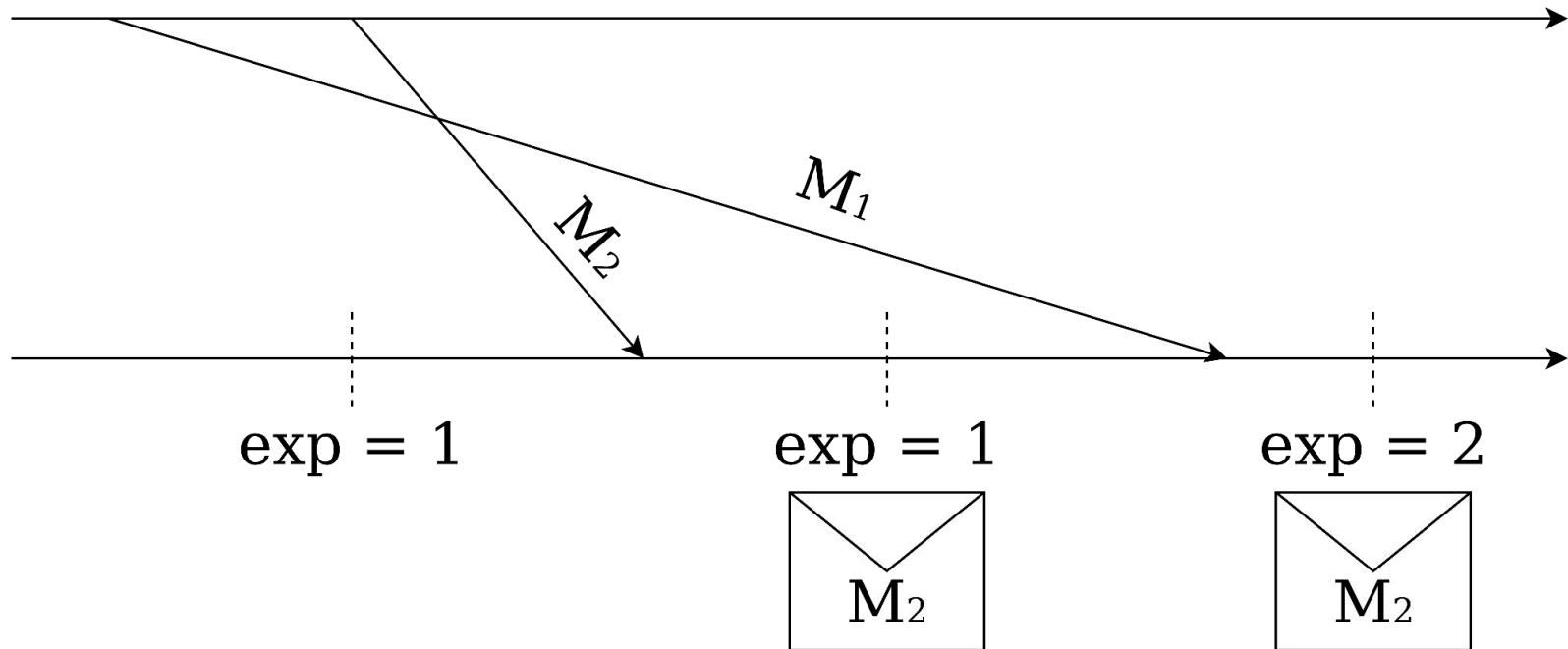
FIFO

- Сообщения между каждой парой процессов нумеруются независимо



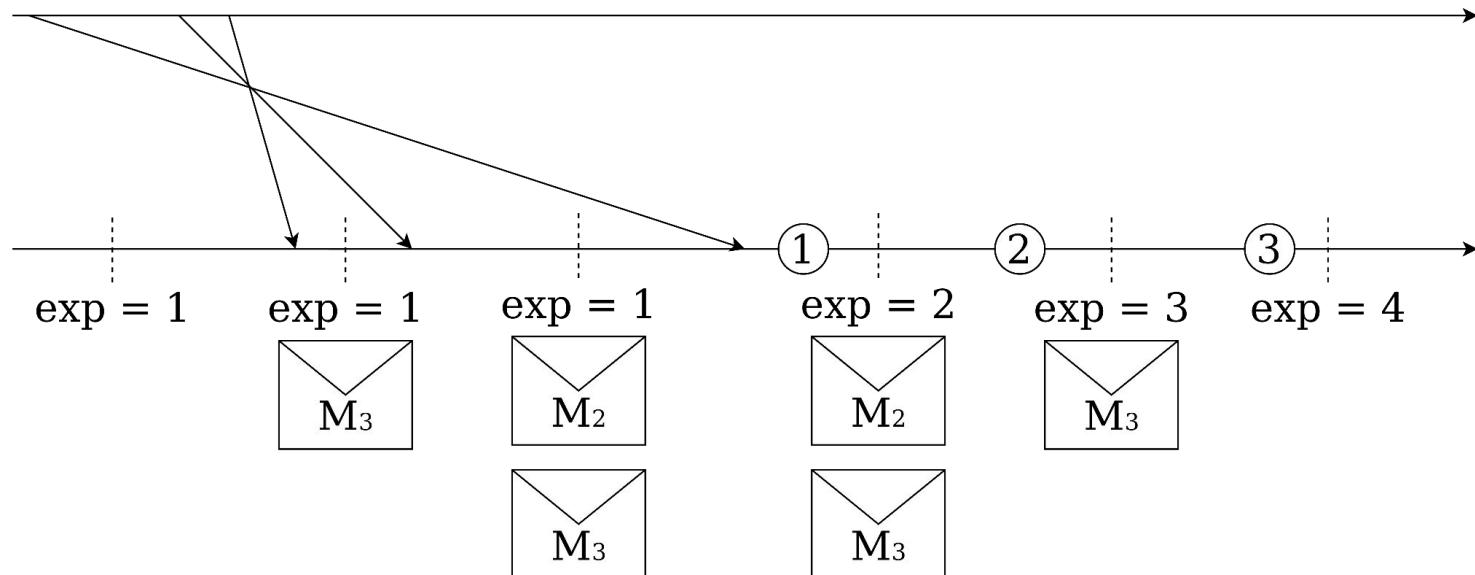
FIFO

- Если приходит сообщение, которого мы не ожидали, сохраняем его
- Ждём прихода ожидаемого



FIFO

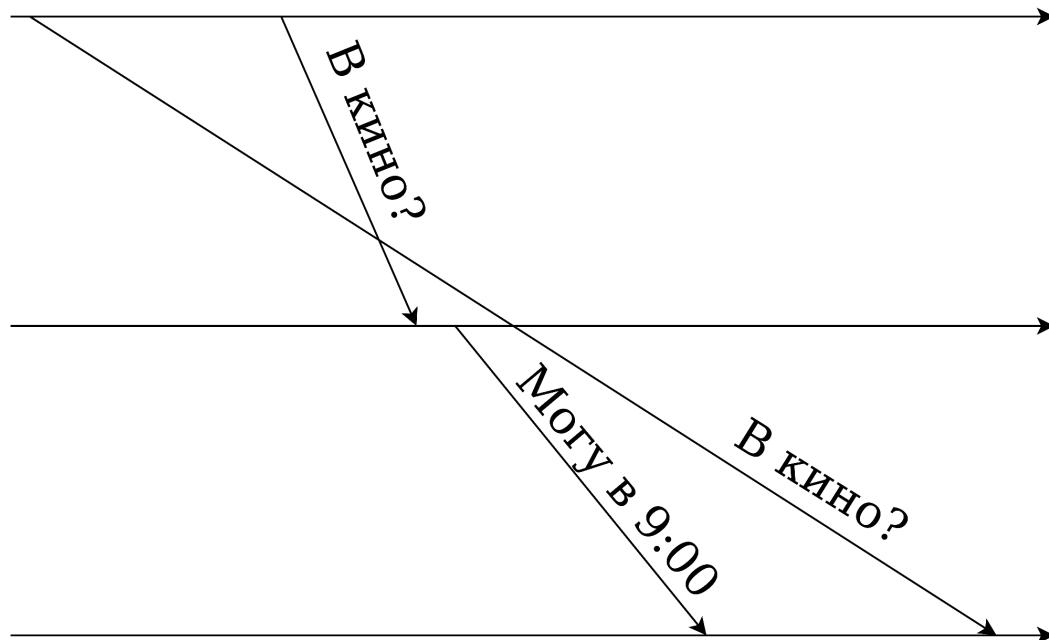
- После обработки сообщения смотрим, нет ли у нас следующего
- Обрабатываем, если есть
- Иначе ждём



Причинная согласованность

- Более сильный порядок

$$\nexists m, n \in M : snd(m) \rightarrow snd(n), rcv(n) \rightarrow rcv(m)$$



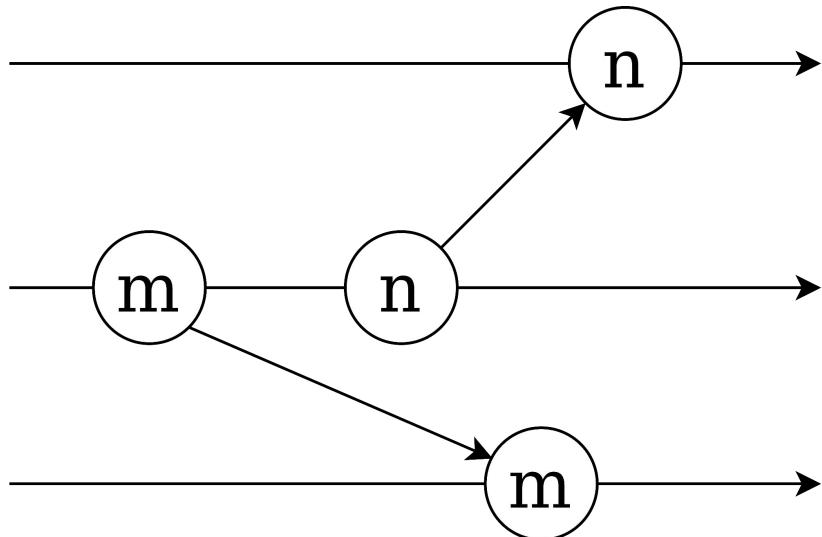
Причинная согласованность

$\exists m, n \in M : snd(m) \rightarrow snd(n), rcv(n) \rightarrow rcv(m)$

не тождественно

$\forall m, n \in M : snd(m) \rightarrow snd(n) \Rightarrow rcv(m) \rightarrow rcv(n)$

- Не хотим ограничивать параллелизм
- m и n могут быть обработаны параллельно

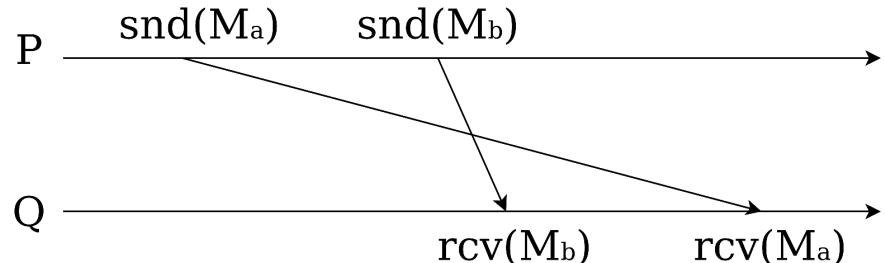


Причинная согласованность

- Из причинной согласованности следует FIFO
- Обеспечив причинную согласованность, обеспечим FIFO

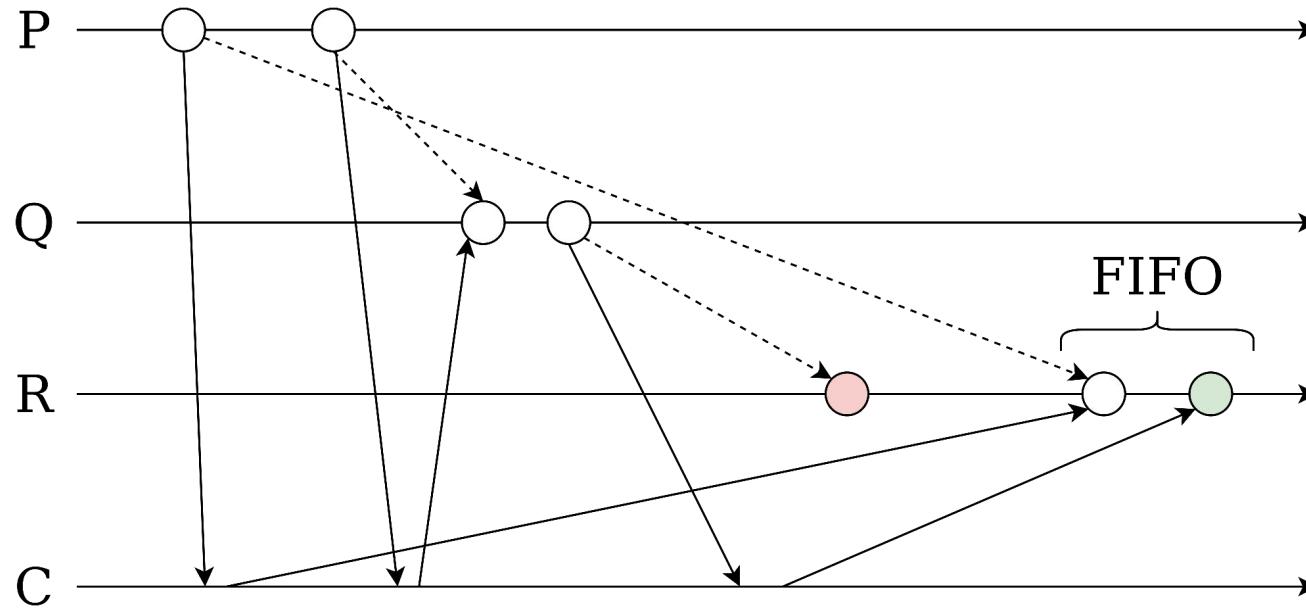
$$\exists m, n \in M : snd(m) \rightarrow snd(n), rcv(n) \rightarrow rcv(m) \Rightarrow \exists m, n \in M^{P \rightarrow Q} : snd(m) < snd(n), rcv(n) < rcv(m)$$

- $snd(M_a) \rightarrow snd(M_b)$
- $rcv(M_b) \rightarrow rcv(M_a)$



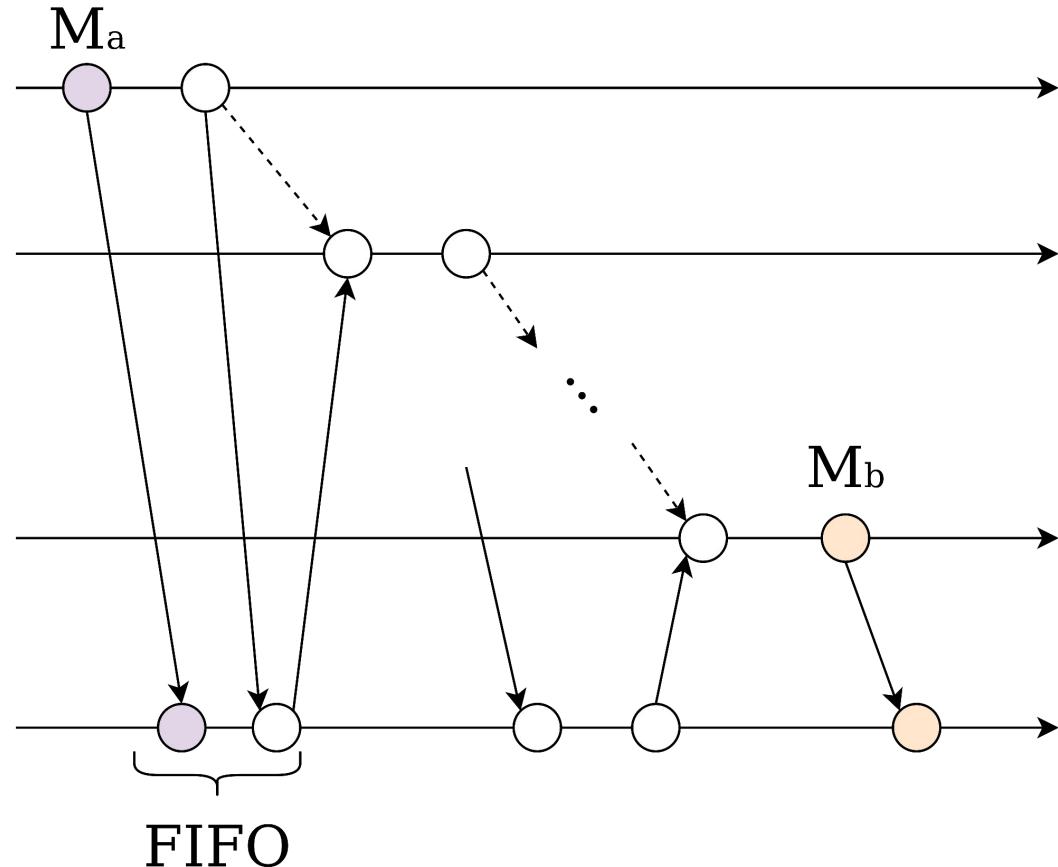
Централизованный алгоритм

- Посыпаем через координатора
- Координатор ретранслирует адресату
- Каналы с координатором FIFO



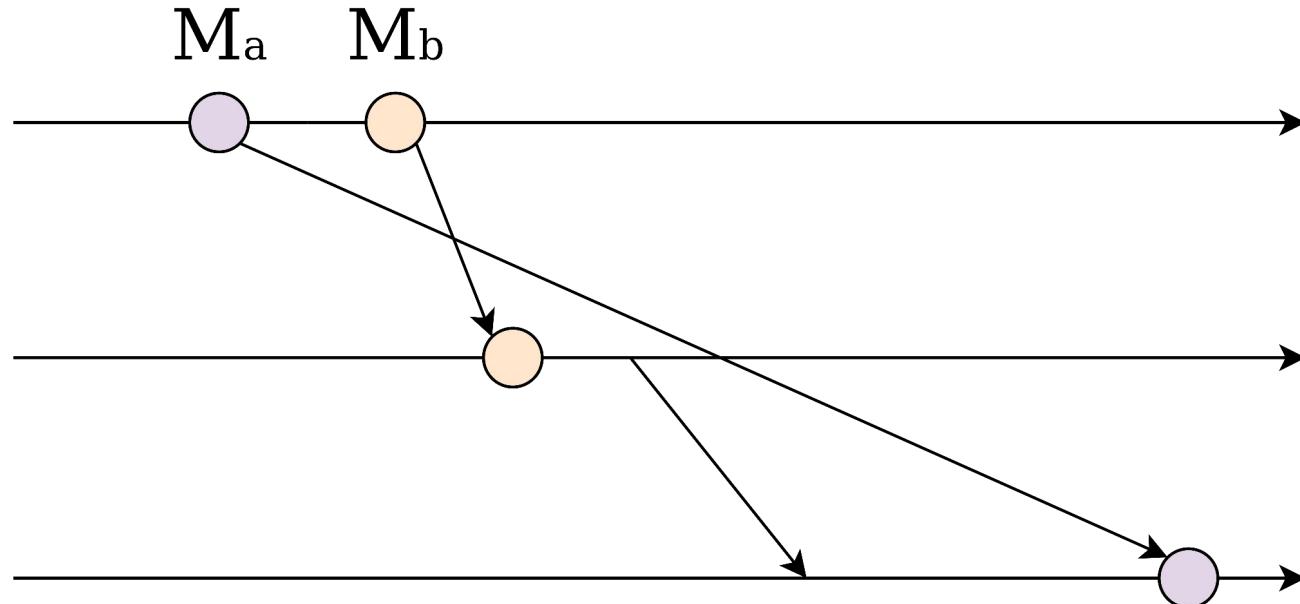
Централизованный алгоритм

- $\text{snd}(M_a) \rightarrow \text{snd}(M_b)$
- M_a оказывается на координаторе раньше M_b
- Каналы с координатором FIFO



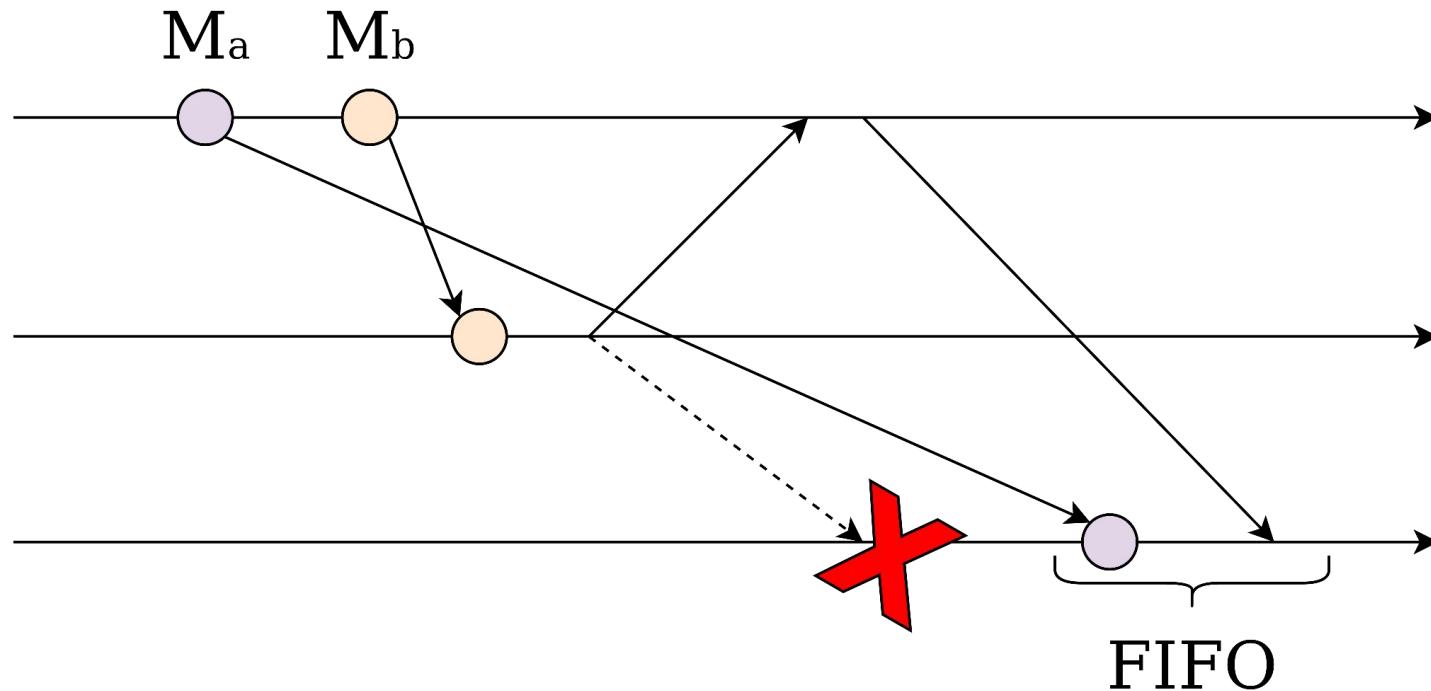
Централизованный алгоритм

- Может ли оказаться, что $\text{rcv}(M_b) \rightarrow \text{rcv}(M_a)$?
- Между получением M_b и M_a существует цепочка сообщений



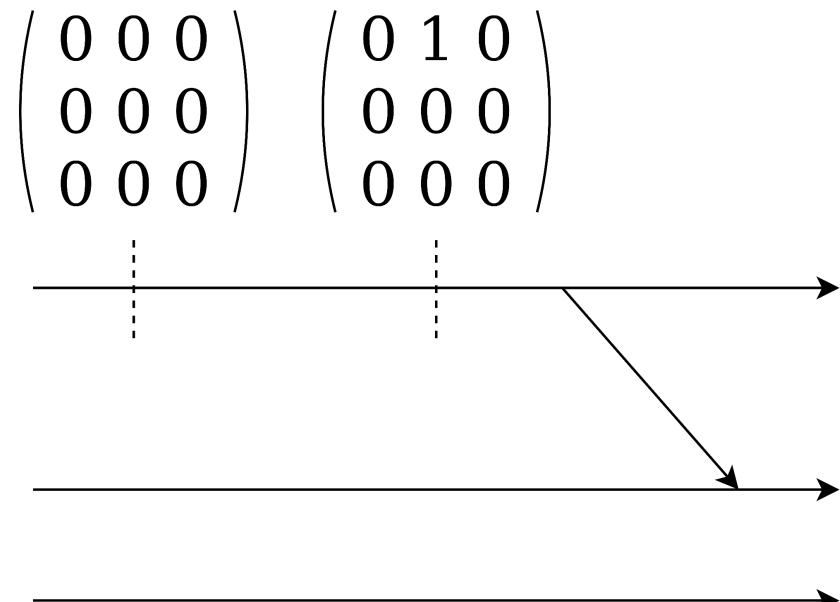
Централизованный алгоритм

- Может ли оказаться, что $\text{rcv}(M_b) \rightarrow \text{rcv}(M_a)$?
- Такая цепочка сообщений не может существовать



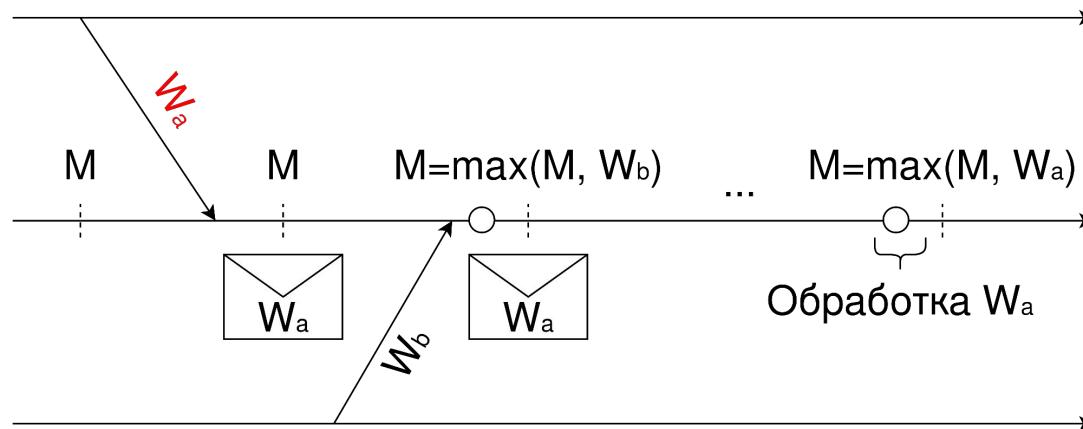
Распределённый алгоритм

- Используем матричные часы
- Размер $n \times n$
 - n — число процессов
- $M_{i,j}$ — количество сообщений, посланных от i -ого процесса j -ому
 - Увеличиваем на 1 перед каждой посылкой
- Прикрепляем матрицу к каждому сообщению



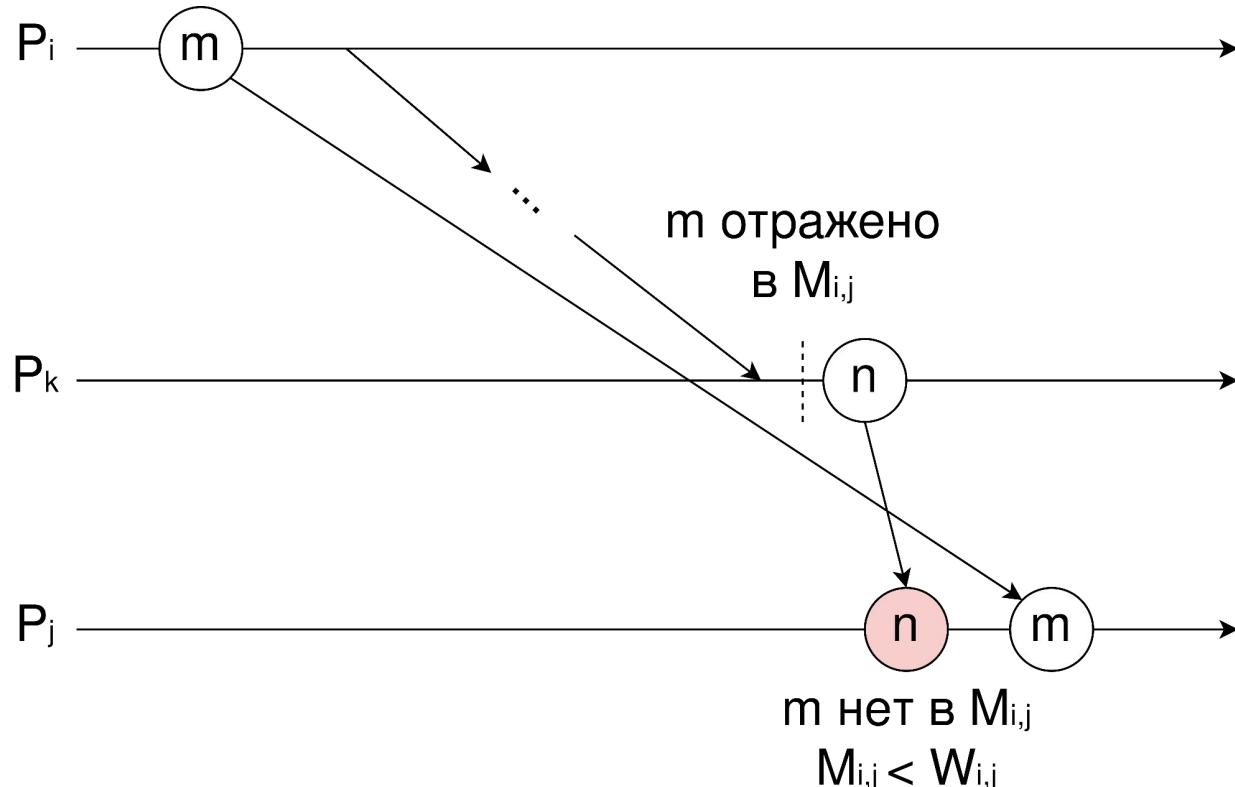
Распределённый алгоритм

- Пусть P_j (матрица M) получил сообщение с матрицей W от P_i
- Обрабатываем сообщение только если $W_{i,j} = M_{i,j} + 1$
 - И $\forall k \neq i : M_{k,j} \geq W_{k,j}$
 - Иначе сохраняем для последующей обработки
- После обработки делаем $M = \max(M, W)$

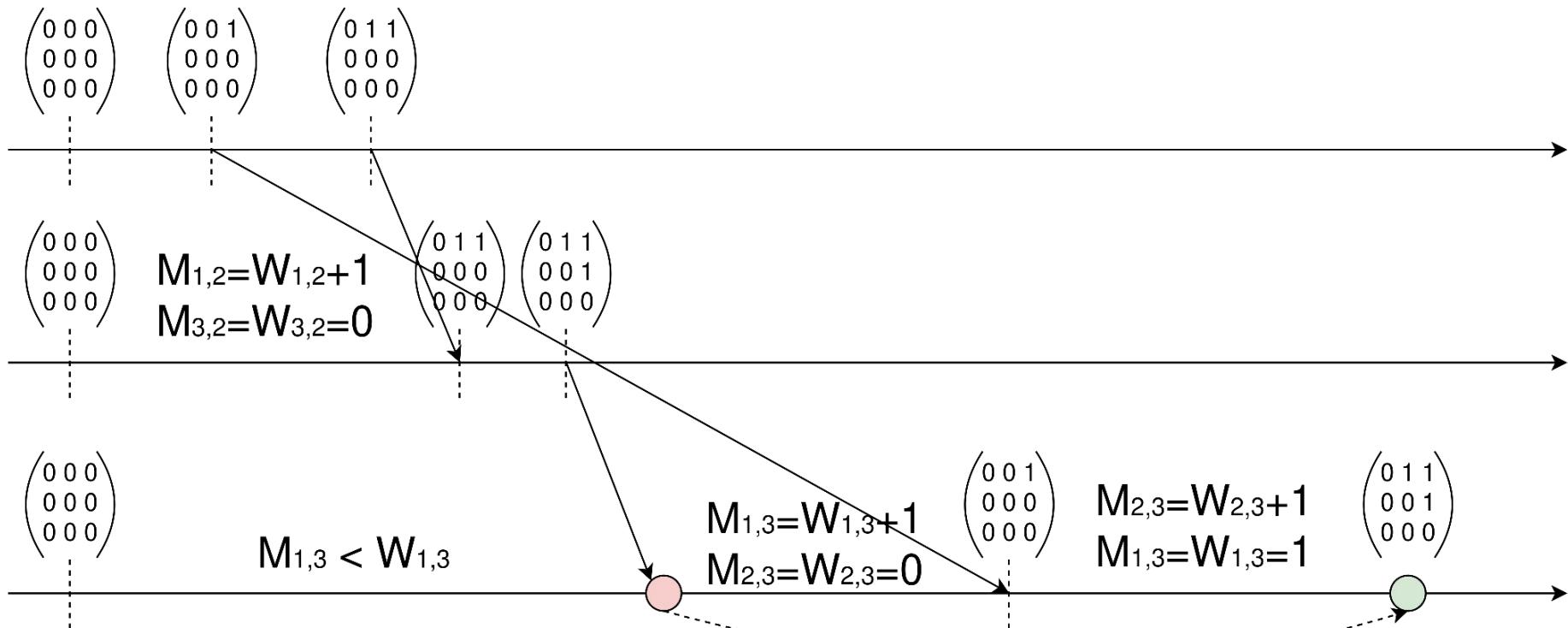


Распределённый алгоритм

- P_k уже знает о сообщении m
- P_j пока нет
- Не может получить

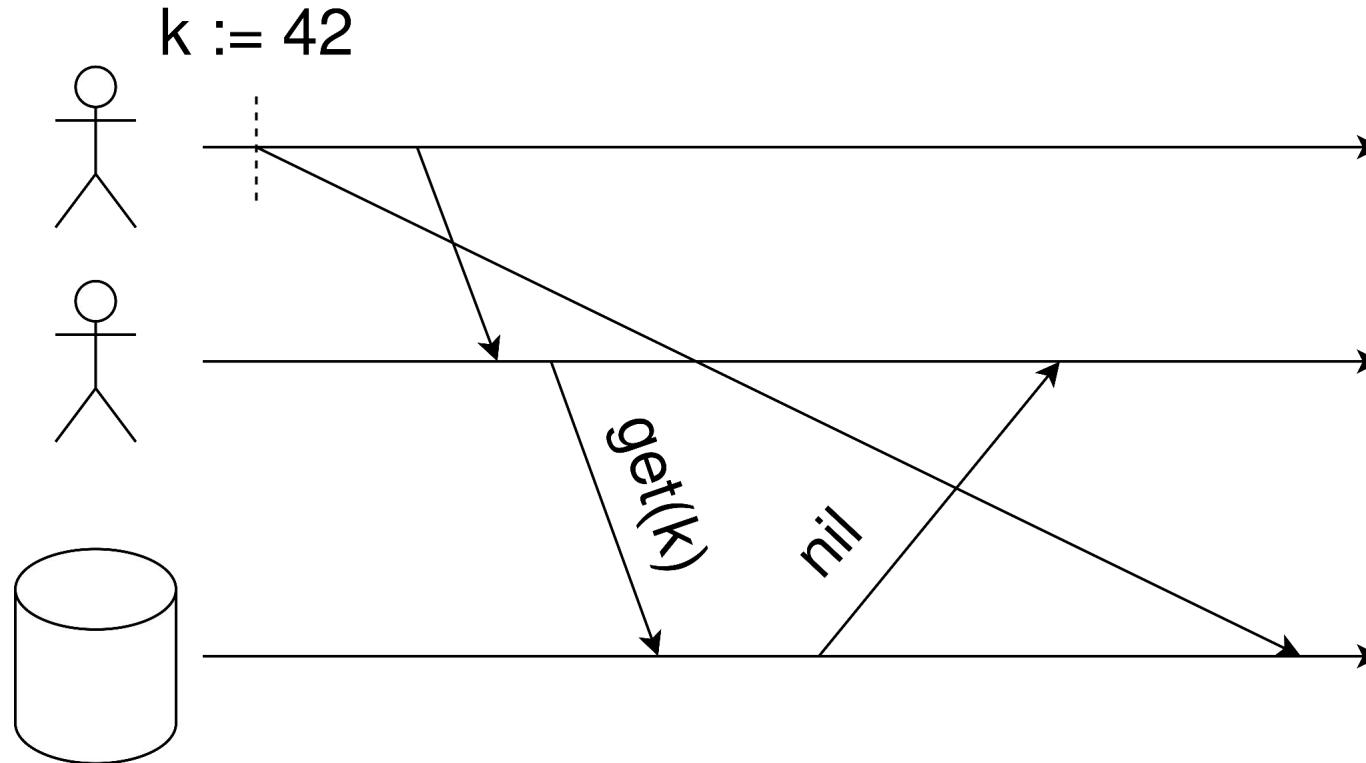


Распределённый алгоритм



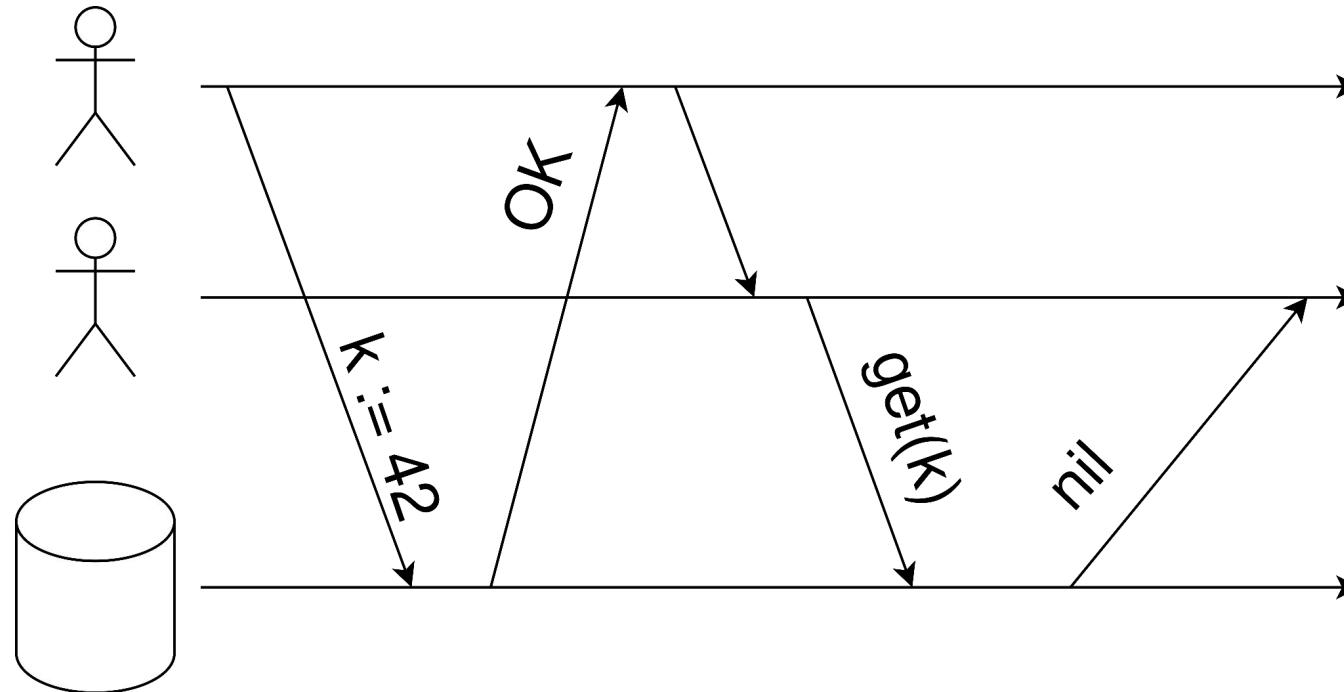
Причинная согласованность на практике

- Клиент не видит запись в базу данных



Причинная согласованность на практике

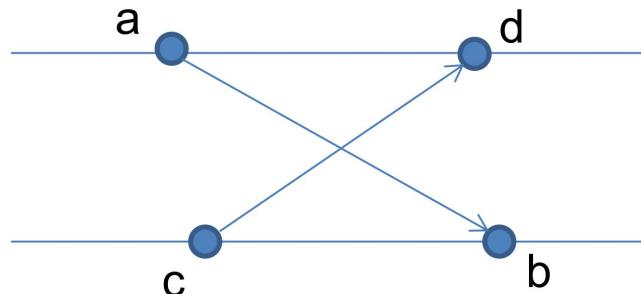
- Можно решать проблему практическими хаками
- Ждём подтверждения от СУБД



Синхронный порядок

- Каждому сообщению m можно сопоставить время $T(m)$
 - Такое, что $T(\text{snd}(m)) = T(\text{rcv}(m))$
 - Из $e \rightarrow f$ следует $T(e) < T(f)$
- Сильнее причинного порядка

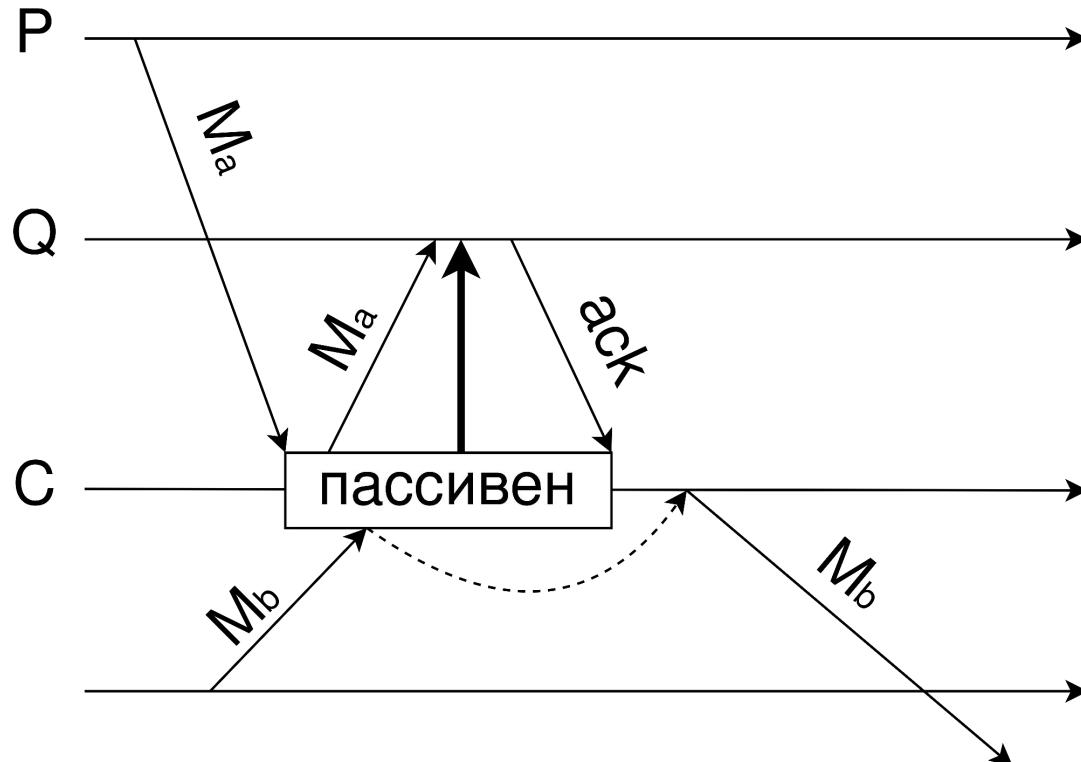
$$\left\{ \begin{array}{l} T(a) = T(b) \\ T(c) = T(d) \\ T(a) < T(d) \\ T(c) < T(b) \end{array} \right.$$



$$\left\{ \begin{array}{l} T(a) = T(b) = x \\ T(c) = T(d) = y \\ x < y \\ y < x \end{array} \right.$$

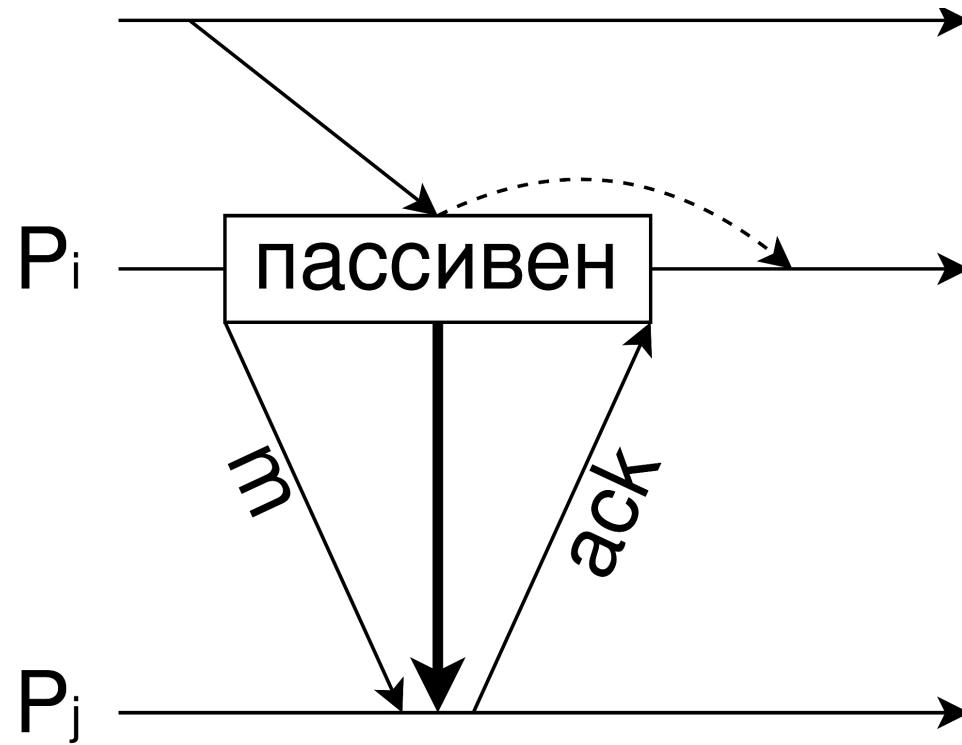
Централизованный алгоритм

- Все сообщения посыпаем через координатора
- Координатор, получив сообщение, пересыпает его
- Ждёт подтверждения
- В это время пассивен
- Обработку других сообщений откладывает
- В системе одно сообщение в каждый момент времени



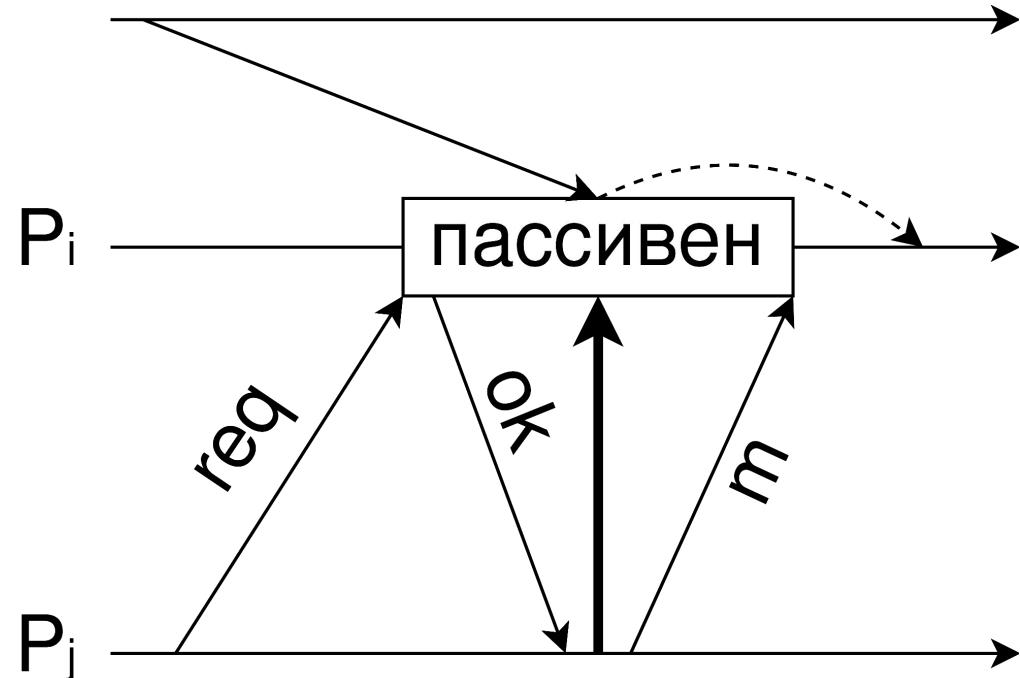
Распределённый алгоритм

- Упорядочим процессы по номерам, $i > j$
- Отправляет сообщение
- Ждёт подтверждения
- В это время пассивен
- P_i участвует только в одной передаче сообщения одновременно



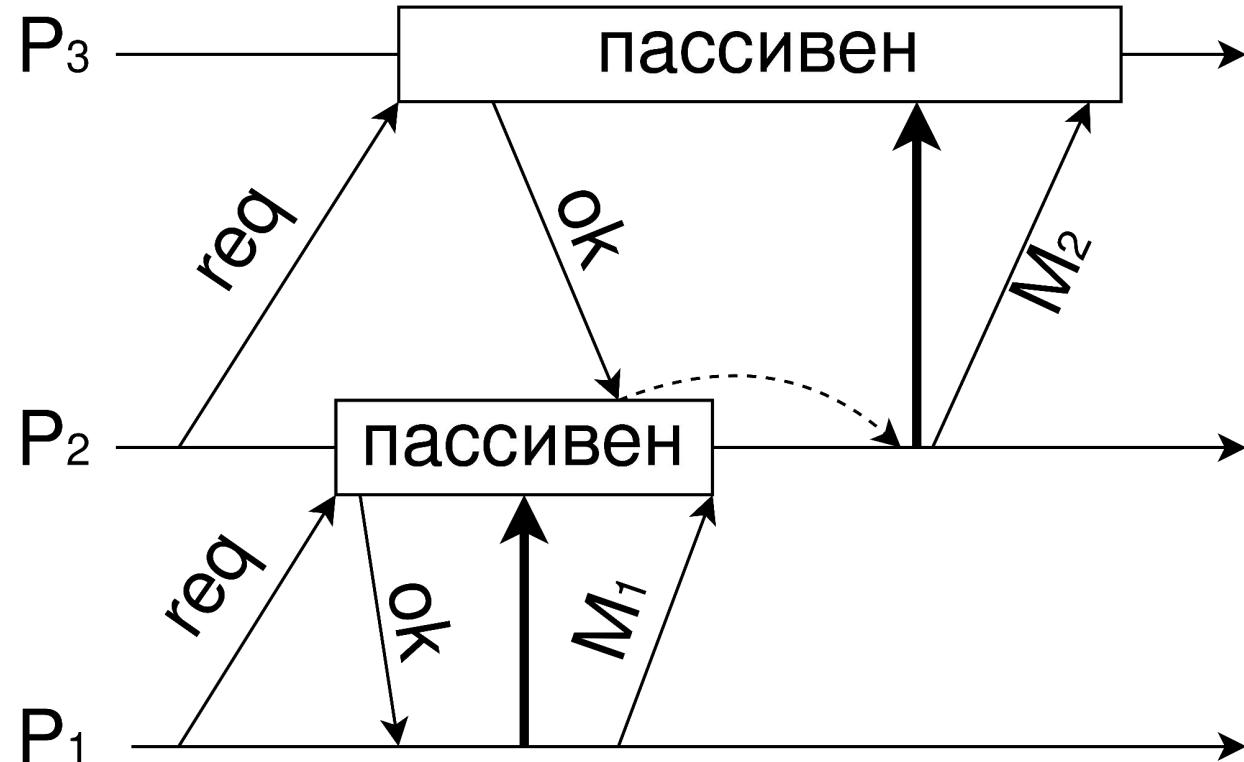
Распределённый алгоритм

- Упорядочим процессы по номерам, $i > j$
- Просит разрешения отправить сообщение
- Отправляет, получив его
- “Большой” процесс становится пассивным, выдав разрешение
- Вновь становится активным, получив сообщение



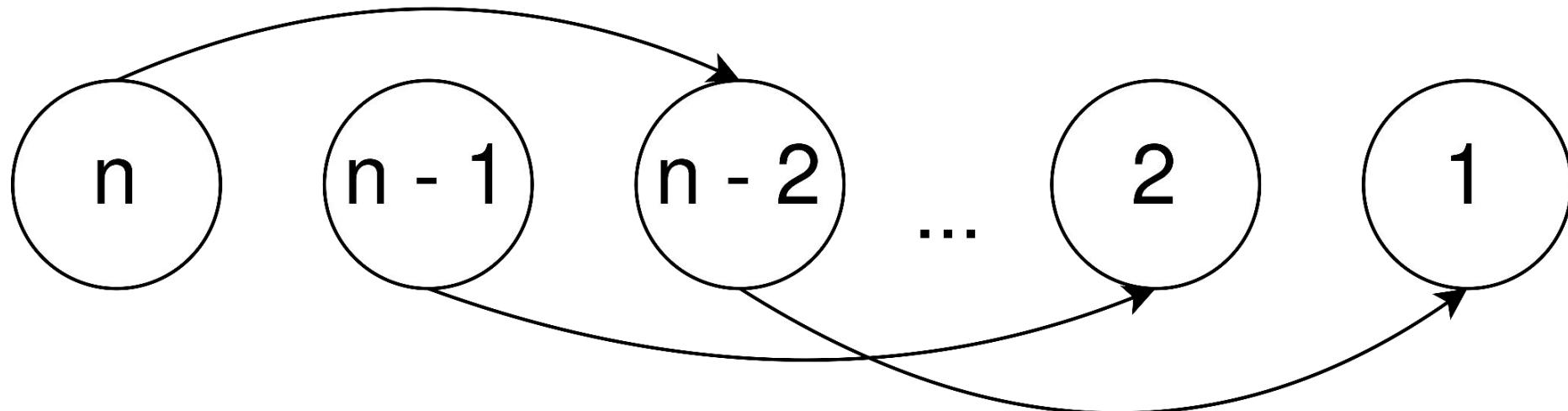
Распределённый алгоритм

- “Меньший” процесс не блокируется и может участвовать в передаче других сообщений



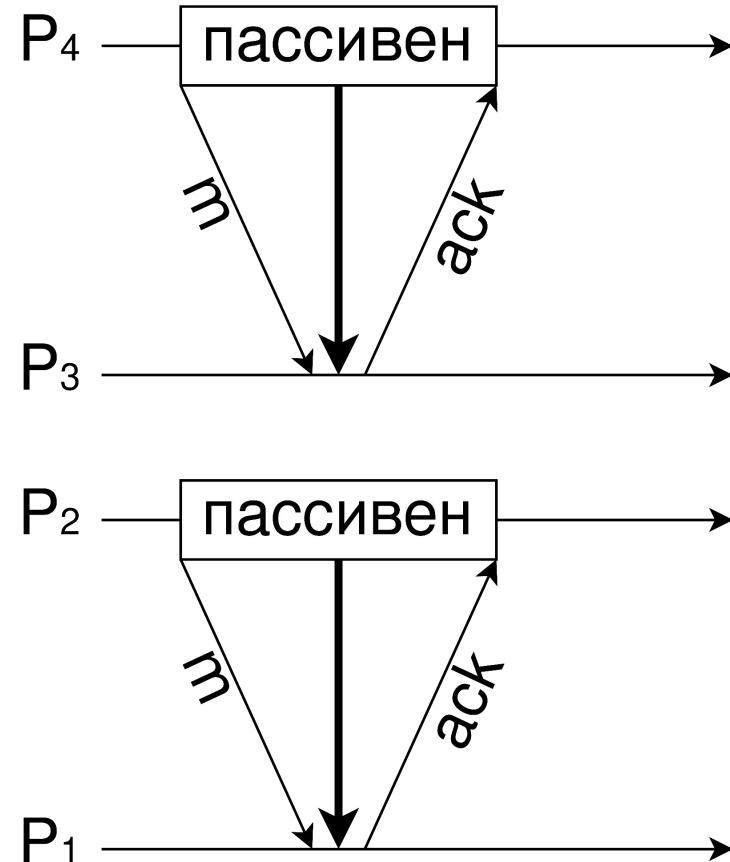
Распределённый алгоритм

- Дедлок в такой схеме невозможен
- При посылке сообщения от P_i до P_j ждёт только процесс с большим номером
- Граф ожидания ацикличен



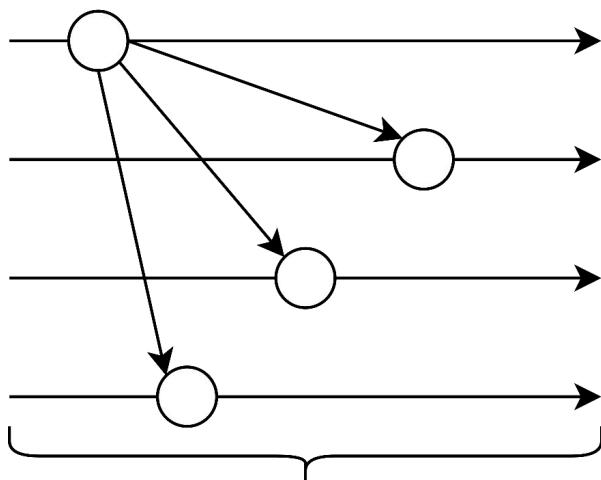
Распределённый алгоритм

- Лучше централизованного
- Непересекающиеся пары процессов могут общаться параллельно

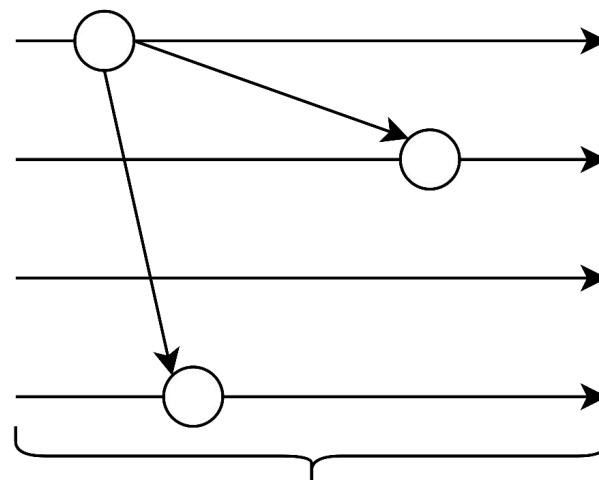


Посылка множеству адресатов

- Broadcast — рассылка всем остальным процессам
- Multicast — рассылка произвольному множеству процессов



Broadcast

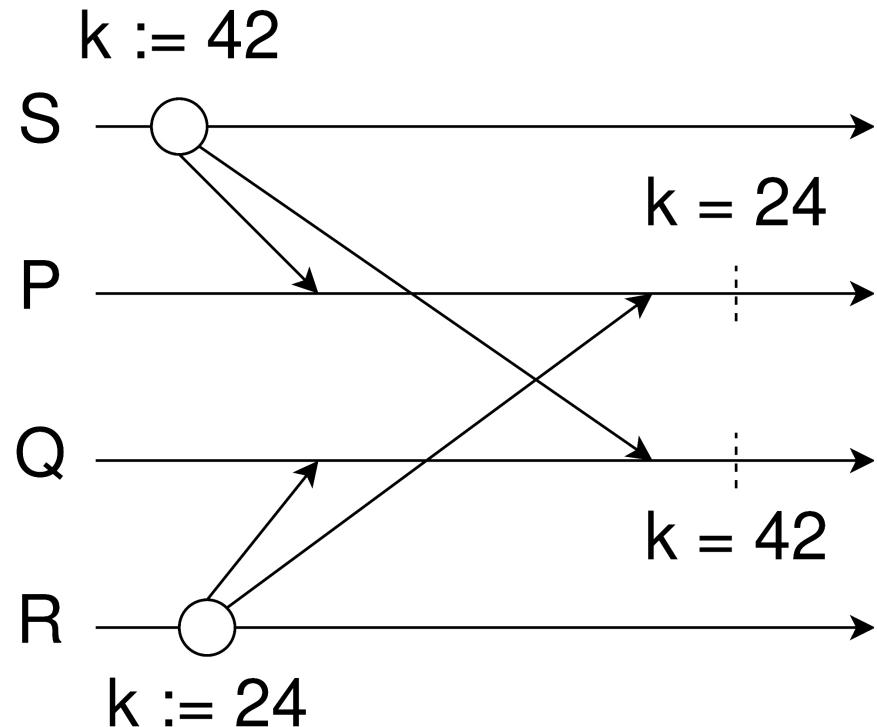


Multicast

Общий порядок

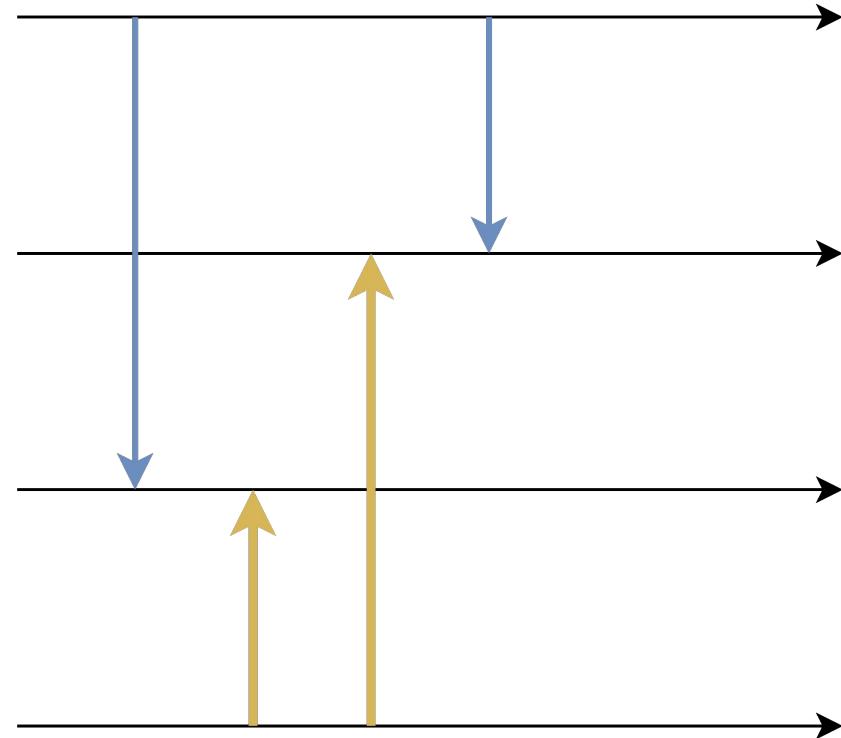
$$\forall m, n \in \mathbb{M}, P, Q \in \mathbb{P} : rcv_P(n) < rcv_P(m), rcv_Q(m) < rcv_Q(n)$$

- Применим только при рассылке множеству адресатов



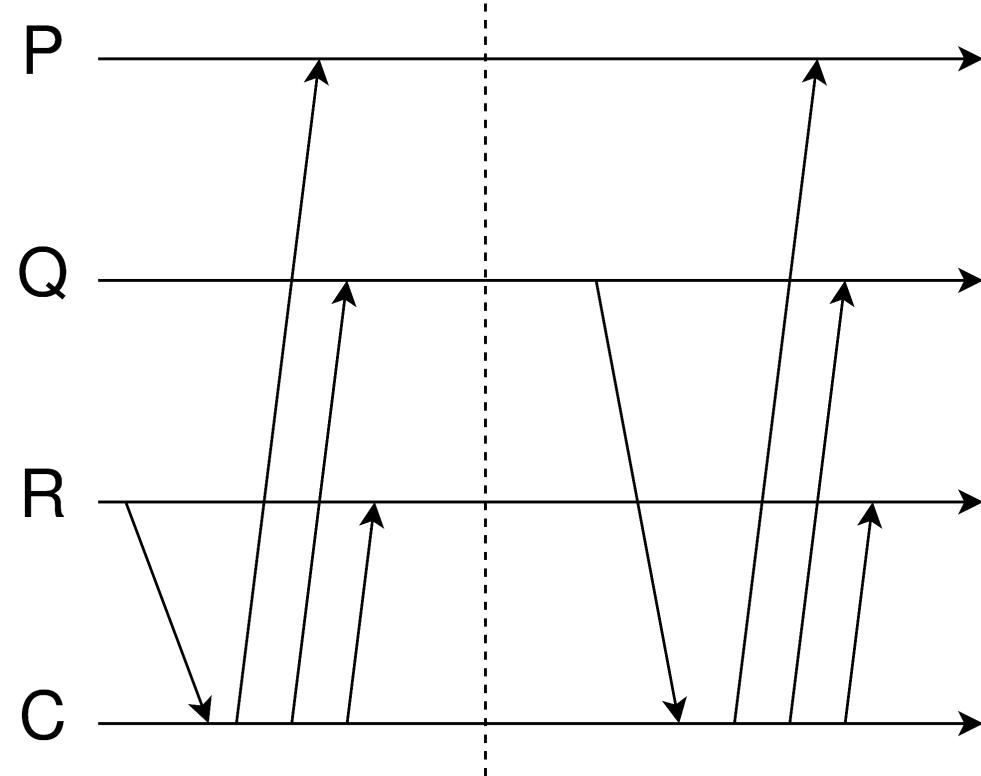
Рассылка множеству адресатов

- Нельзя реализовать даже с помощью синхронного порядка
- ```
for i in 1..n:
 if i != my_pid
 sync_send(msg)
```



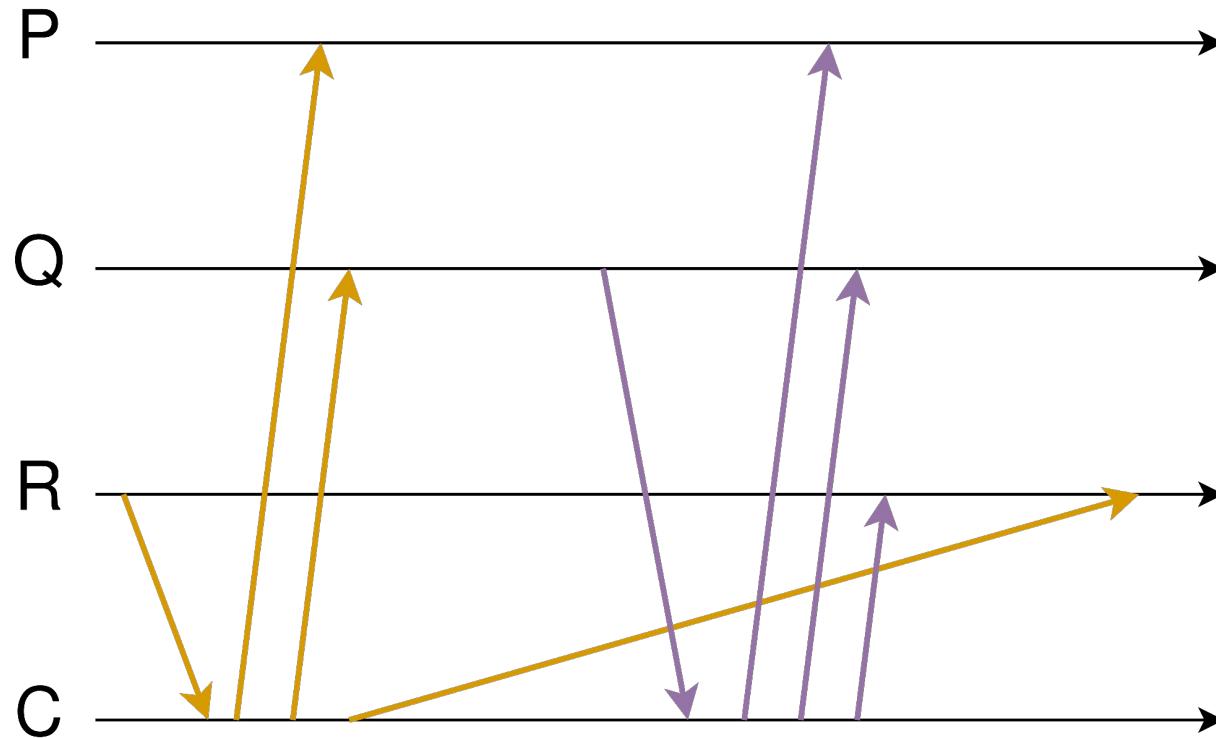
# Централизованный алгоритм

- Посыпаем сообщение координатору
- Координатор делает рассылку



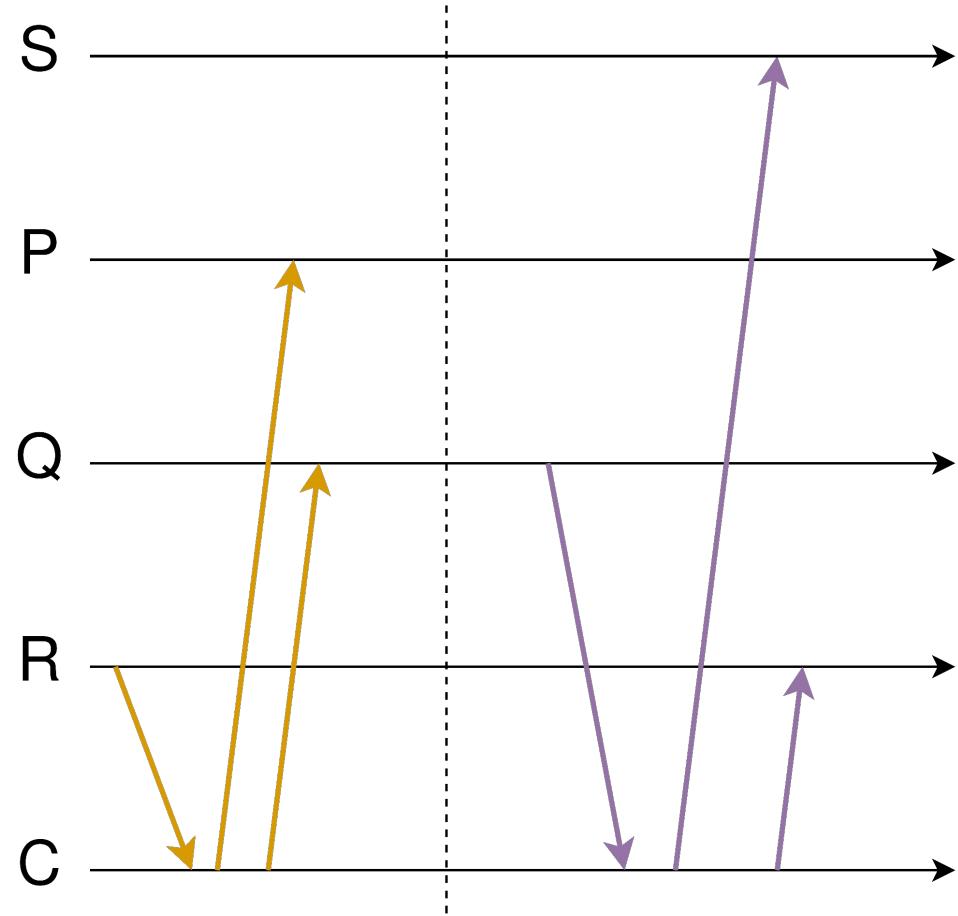
# Централизованный алгоритм

- Каналы от и до координатора должны быть FIFO



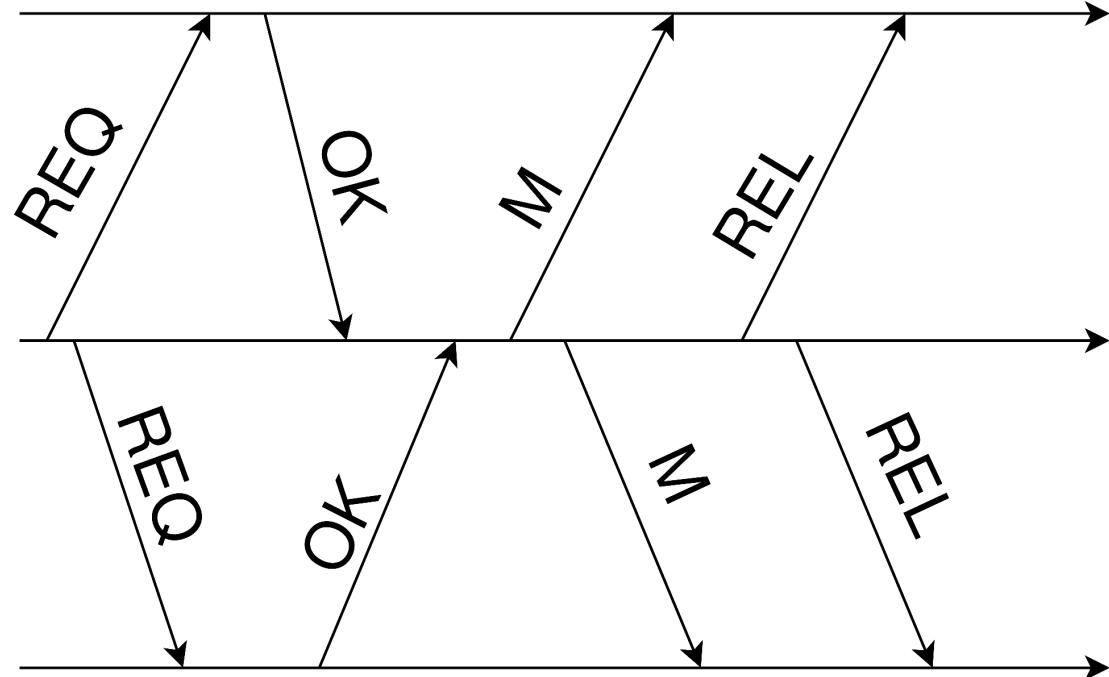
# Централизованный алгоритм

- Можно использовать для multicast
- Сообщения попадают к каждому получателю в том же порядке, в котором они попадают к координатору
- Координатор не принимает следующее сообщение, не разослав предыдущее



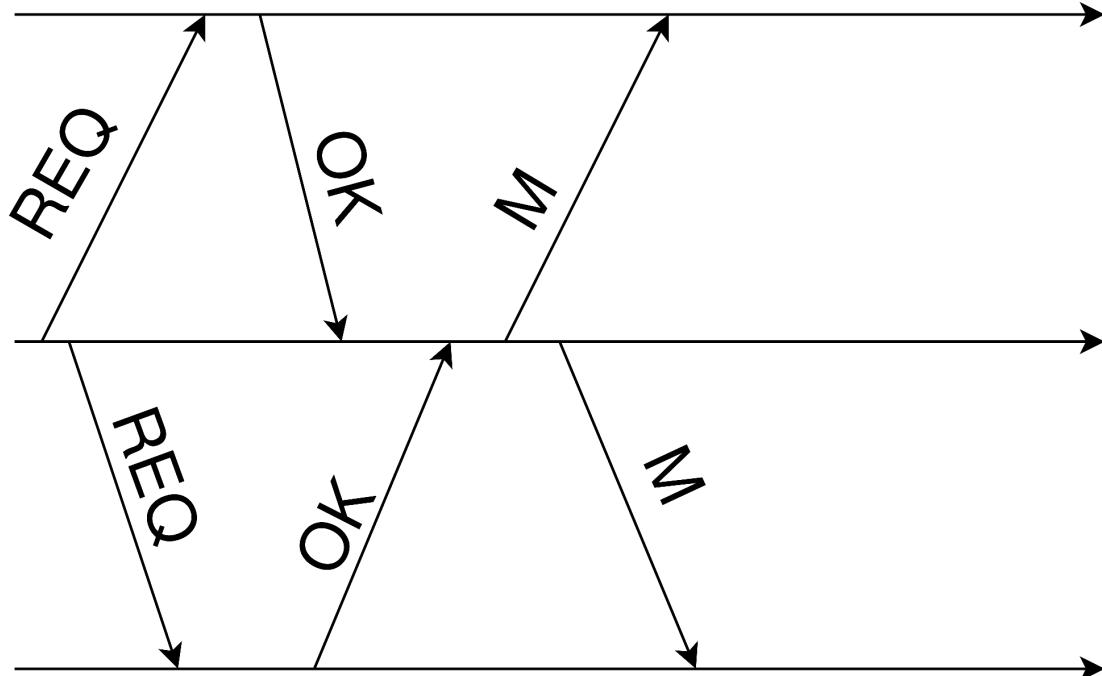
# Broadcast через блокировку

- Берём блокировку
- Делаем рассылку
- Отпускаем блокировку
- Нужно FIFO между всеми парами процессов чтобы REL не пришёл раньше сообщения



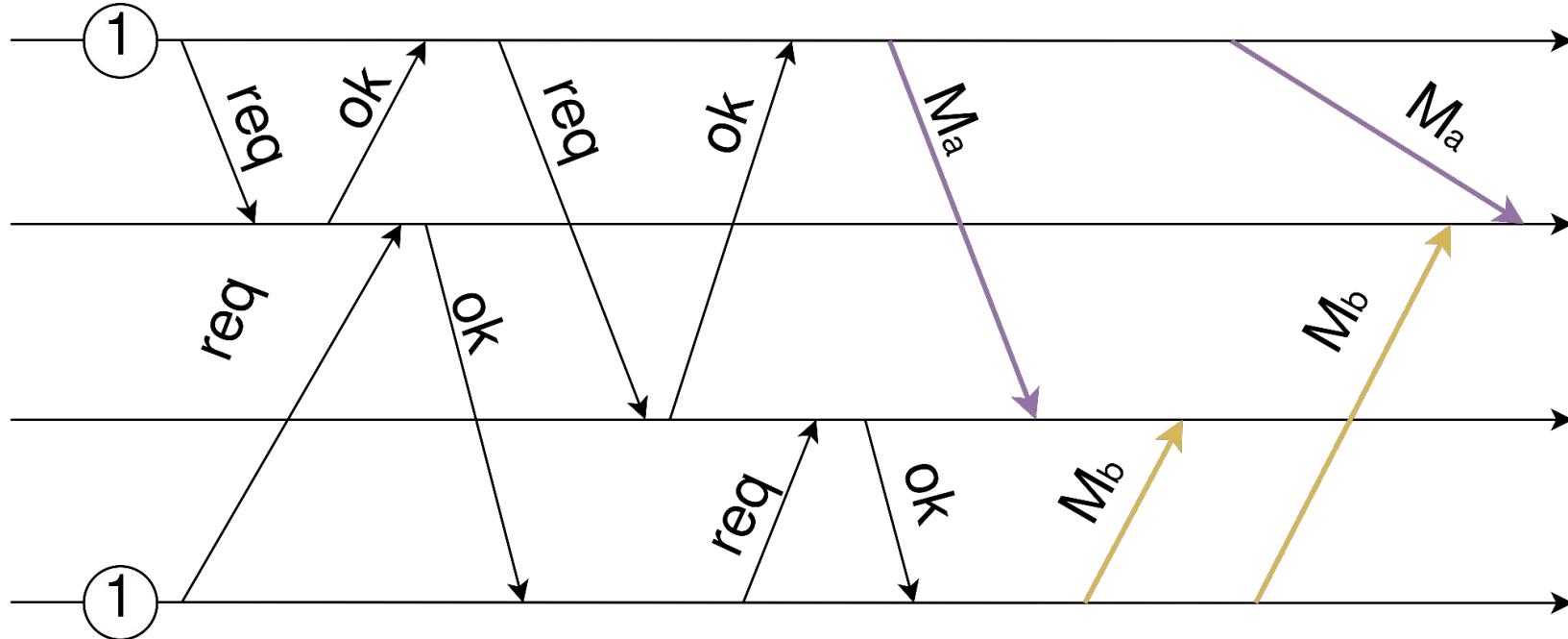
# Алгоритм Лампорта

- Оптимизируем алгоритм с блокировкой
- Используем сообщение для отпускания блокировки
- На  $n - 1$  сообщение меньше



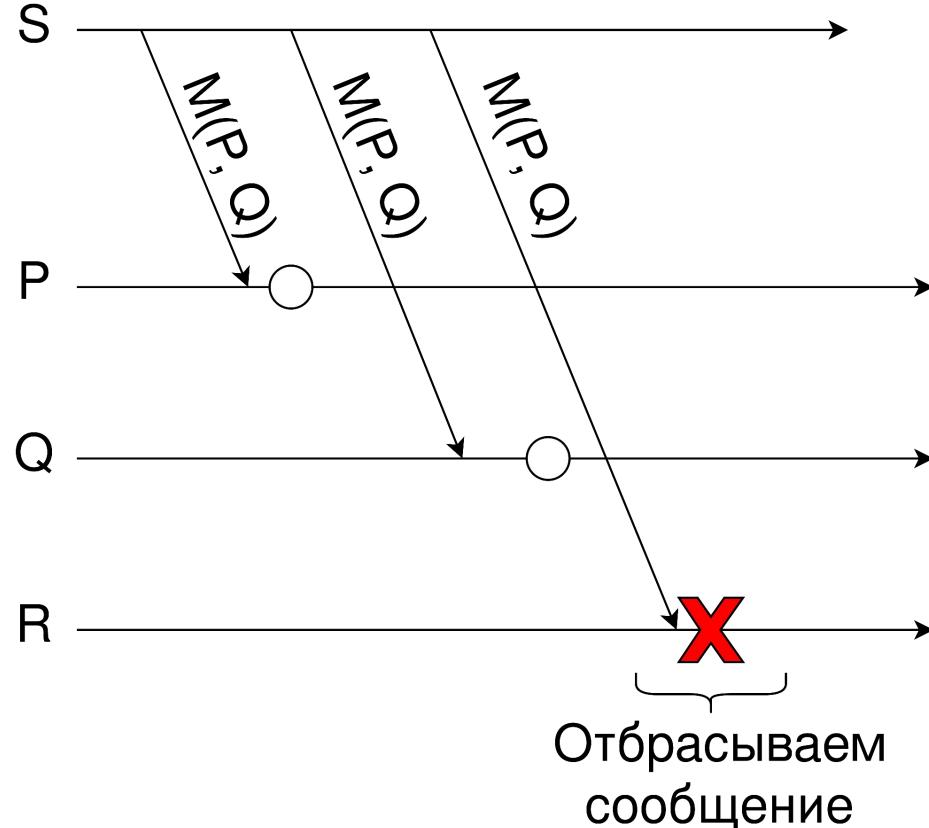
# Алгоритм Лампорта

- Применим только для broadcast
- В случае multicast ломается



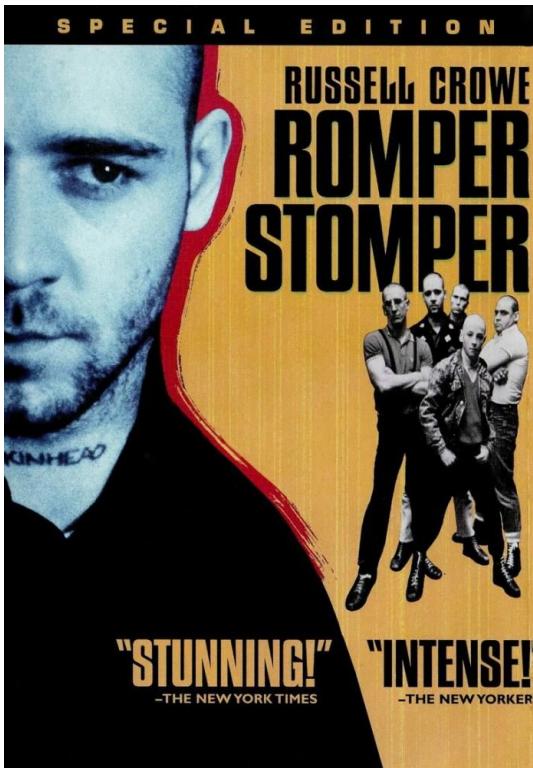
# Реализация multicast с помощью broadcast

- Просто отбросим ненужные сообщения при получении
- Вместо их обработки
- Неэффективно с точки зрения количества передаваемых данных



# Алгоритм Скина

- Нет, не этого

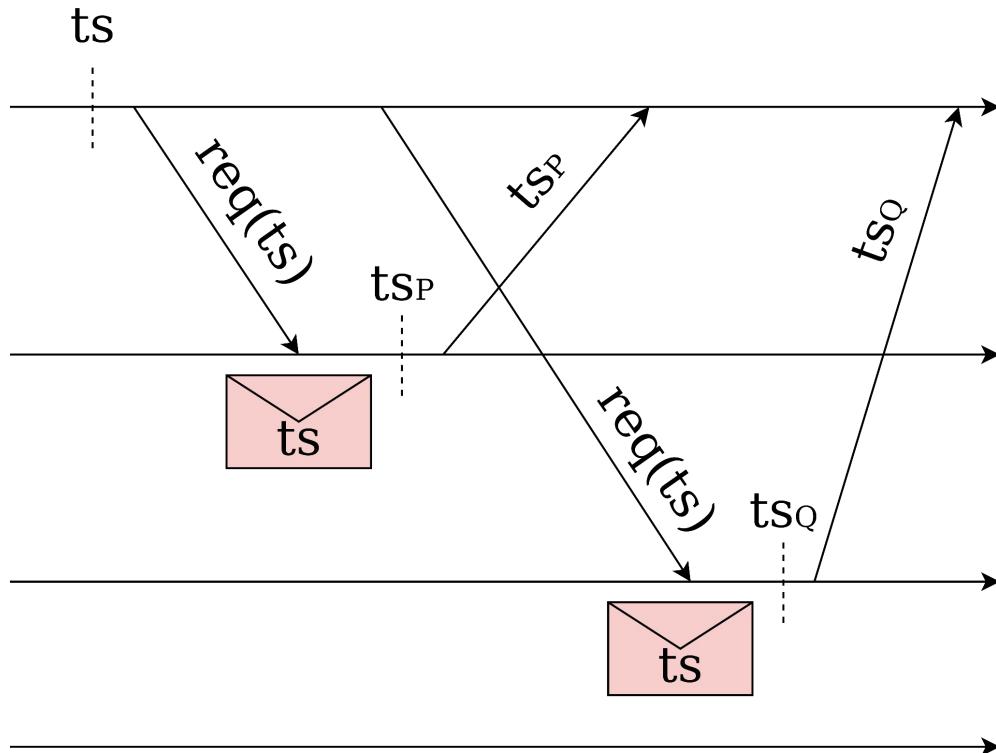


- А Дейла Скина  
(Dale Skeen)



# Алгоритм Скина: multicast

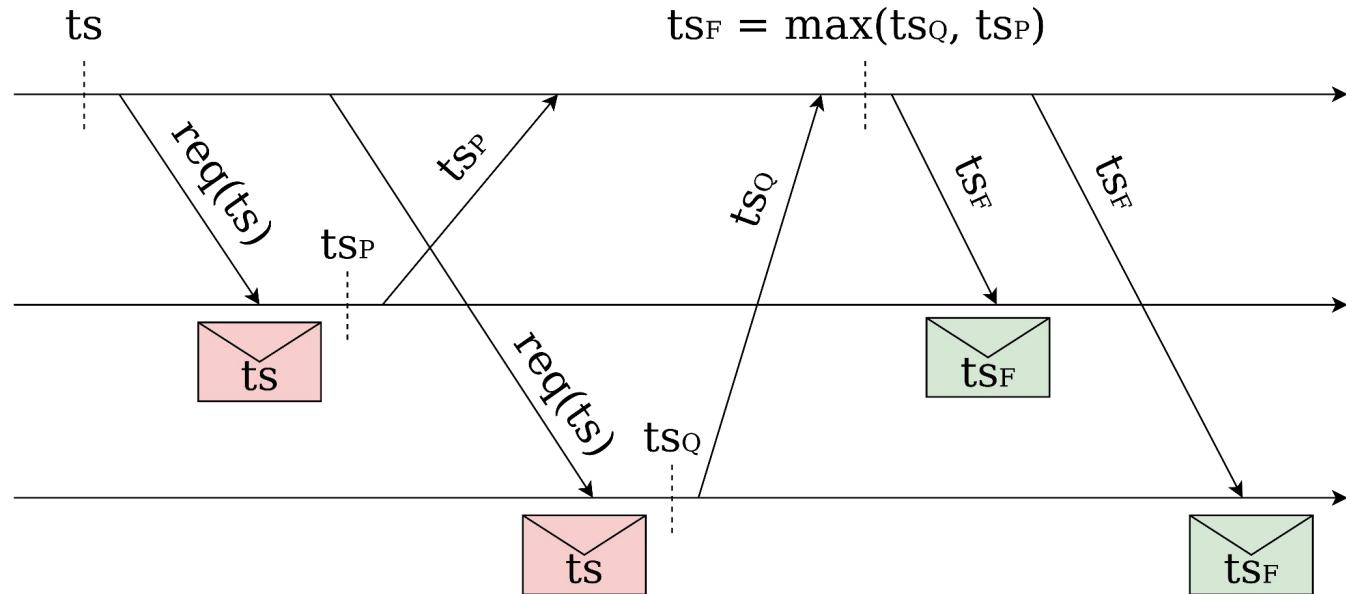
- Инициатор рассыпает всем **получателям** сообщение со своим временем Лампорта
- Получатели принимают его и добавляют в локальную очередь
- Отвечают локальным временем Лампорта
  - $ts_P > ts$ ,  $ts_Q > ts$



# Алгоритм Скина: multicast

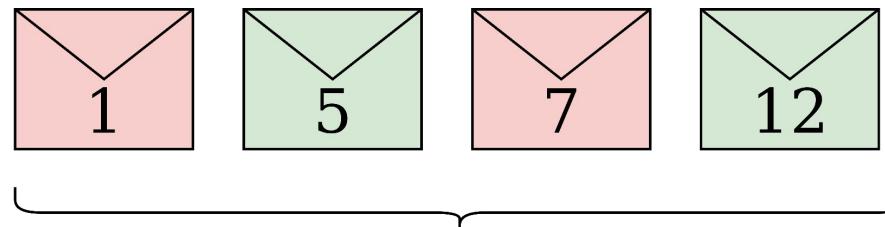
- Инициатор собирает полученные таймстампы
- Берёт из них максимум и рассыпает по получателям
- Получатели помечают сообщение как готовое к получению

- Ставят  $ts_F$  как его время

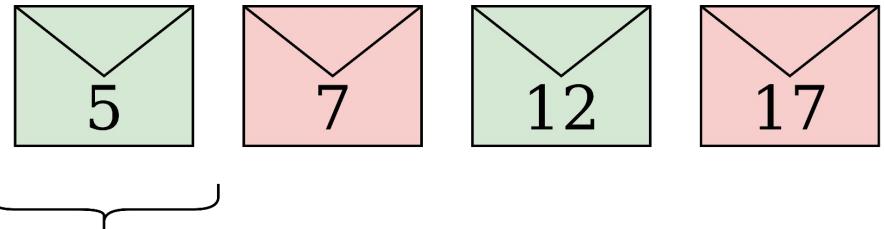


# Алгоритм Скина: multicast

- Сообщение можно обработать если у него минимальное время
- И оно помечено как готовое к получению
  - То есть для него известно окончательное время



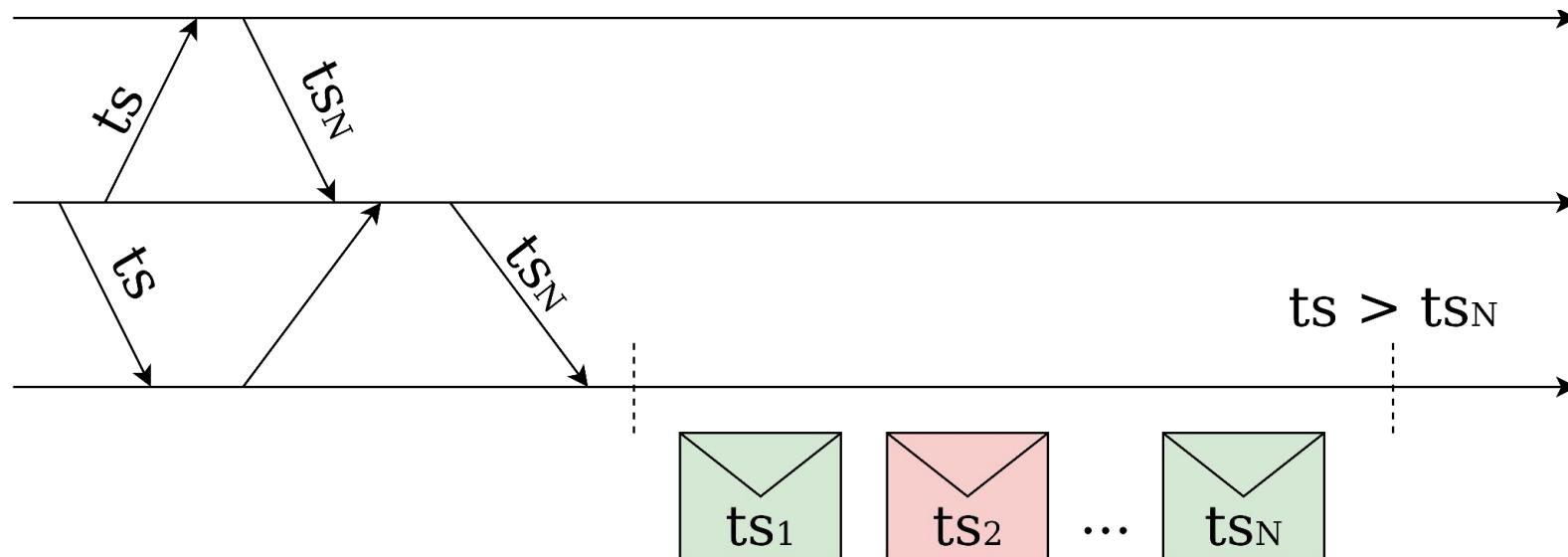
Нельзя обработать



Можно обработать

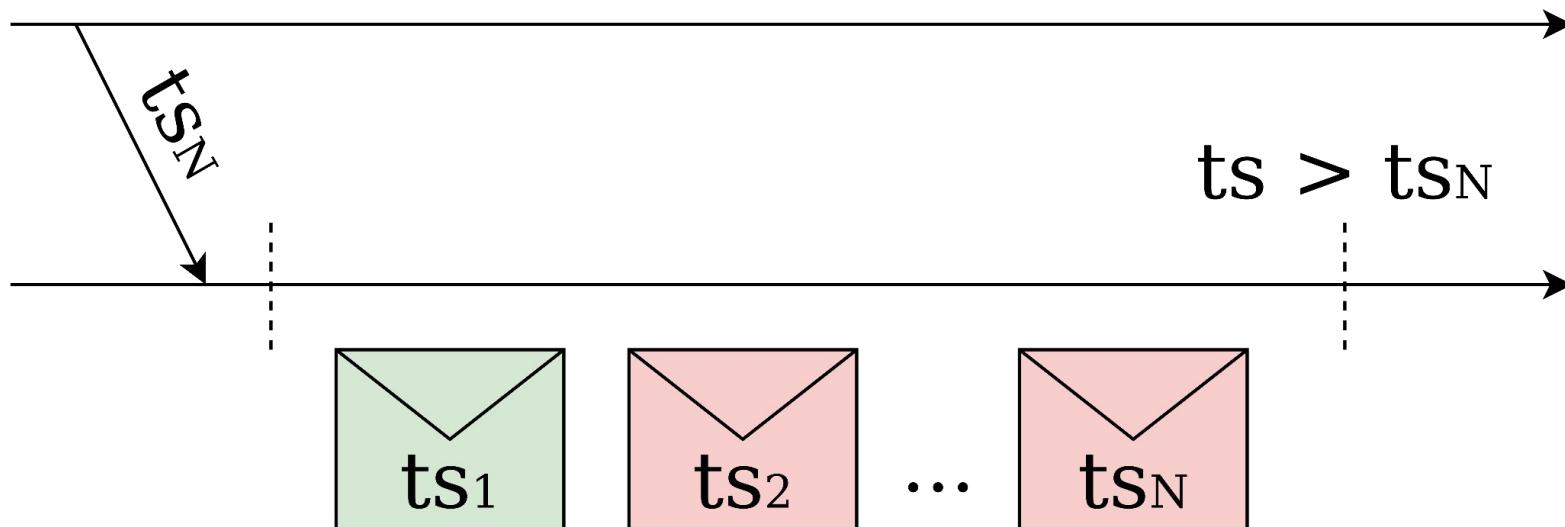
# Алгоритм Скина: доказательство

- Пусть последнее сообщение в очереди можно обработать и его время  $ts_N$
- Это время кто-то приспал
- Значит, локальное время больше  $ts_N$



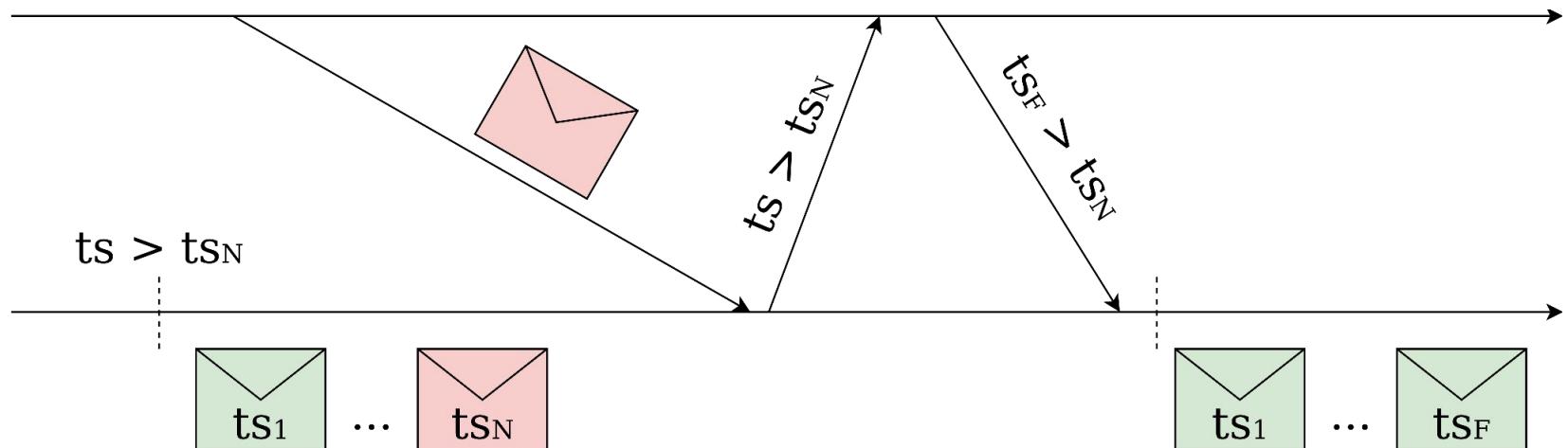
# Алгоритм Скина: доказательство

- Аналогично если последнее сообщение не готово к доставке



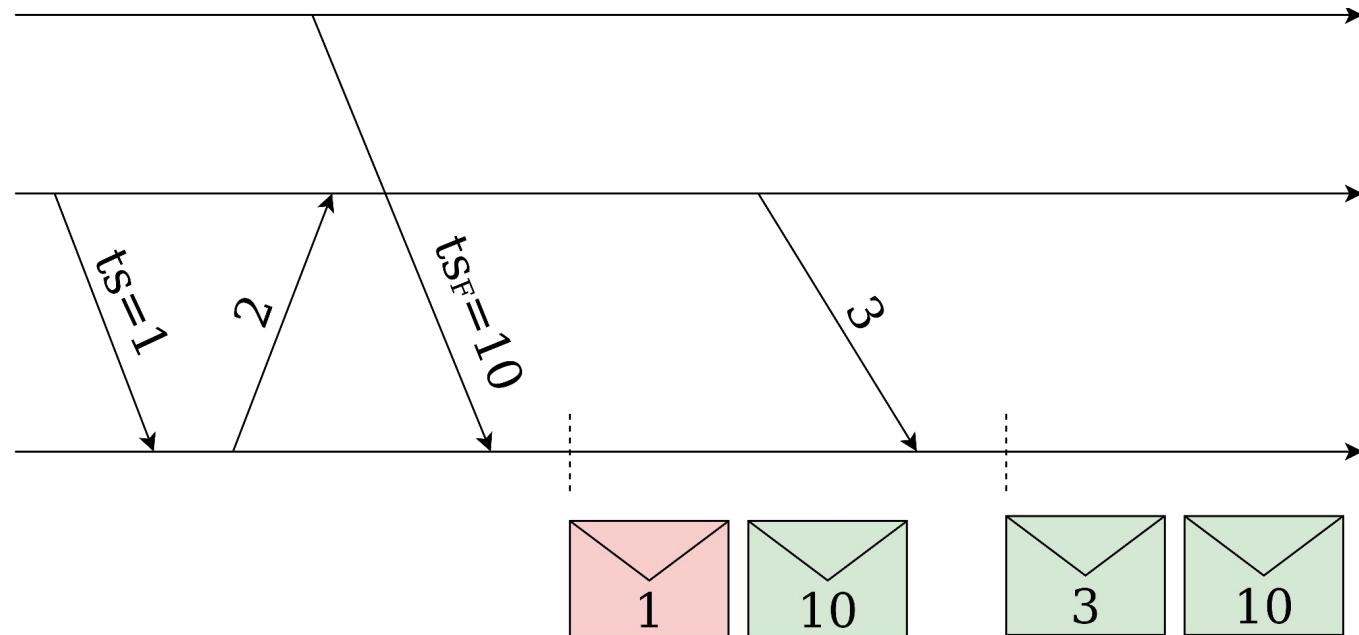
# Алгоритм Скина: доказательство

- Если сообщения нет в локальной очереди, значит мы его ещё не получили
- Когда получим, ответим локальным временем  $ts > ts_N$
- Итоговое время этого сообщения  $ts_F > ts_N > ts_{N-1} > \dots > ts_1$
- Можем обработать первое сообщение, если оно готово



# Алгоритм Скина: доказательство

- Если первое сообщение в очереди не готово к получению, нельзя обрабатывать последующие
- У этого первого сообщения итоговое время может быть меньше



## Что почитать:

- *Raynal M., Schiper A., Toueg S.* The causal ordering abstraction and a simple way to implement it
- *Murty V. V., Garg V. K.* An algorithm for guaranteeing synchronous ordering of messages
- *L. Lamport.* Time, clocks, and the ordering of event in a distributed system.
- *Skeen M. D.* Crash recovery in a distributed database system.

# Thanks for your attention

