

# Введение, логические часы, блокировки



Илья Кокорин

[kokorin.ilya.1998@gmail.com](mailto:kokorin.ilya.1998@gmail.com)

# Presenting myself

- Bachelor of Computer Science
  - (ITMO, 2016 — 2020)
- Master of Computer Science
  - (ITMO, 2020 — 2022)
- PhD in Computer Science
  - (ITMO, 2022 — Present Time)
- Database Developer @ 
  - ex-Yandex, ex-Jetbrains Research, ex-IST Austria
- Teacher of Distributed Systems course @ SPbSU & ITMO

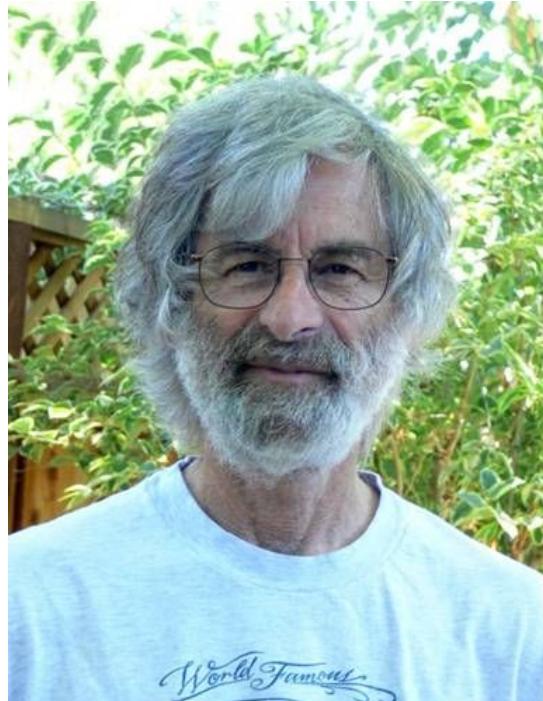


# Требования к курсу

- Языки программирования
  - Java > Go > Python > C++
- Linux
  - Командная строка
  - Системные вызовы
- Параллельное программирование
  - Блокировки
  - Мониторы Хоара
- Основы компьютерных сетей
- Дискретная теория вероятностей
- Основы машинного обучения

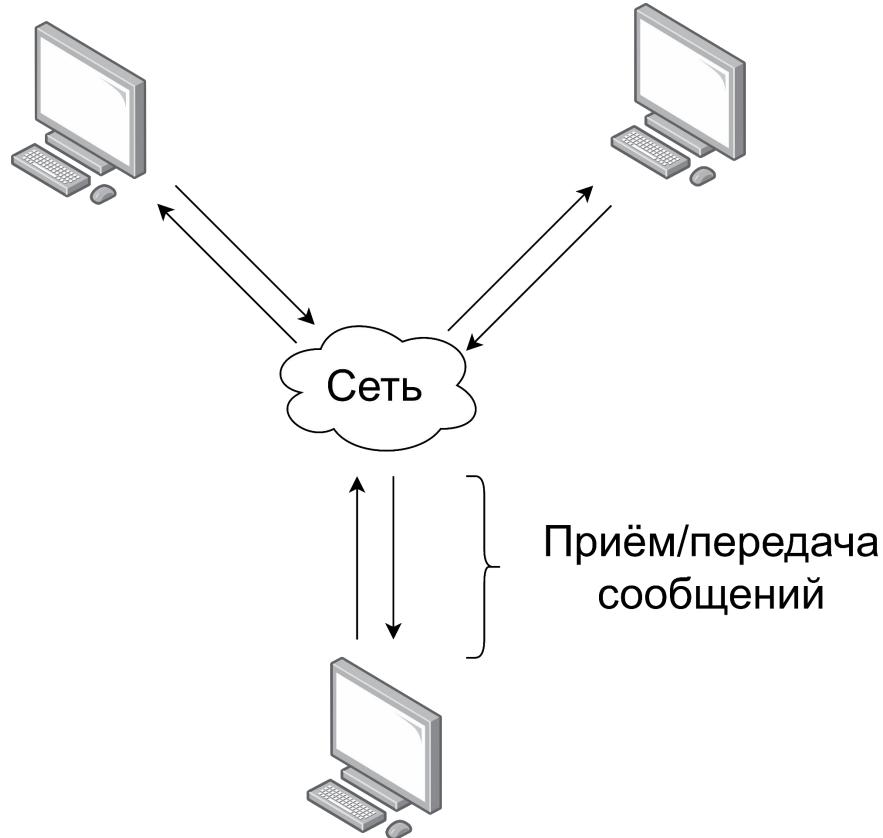
# Распределённая система

- *A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*
  - Leslie Lamport



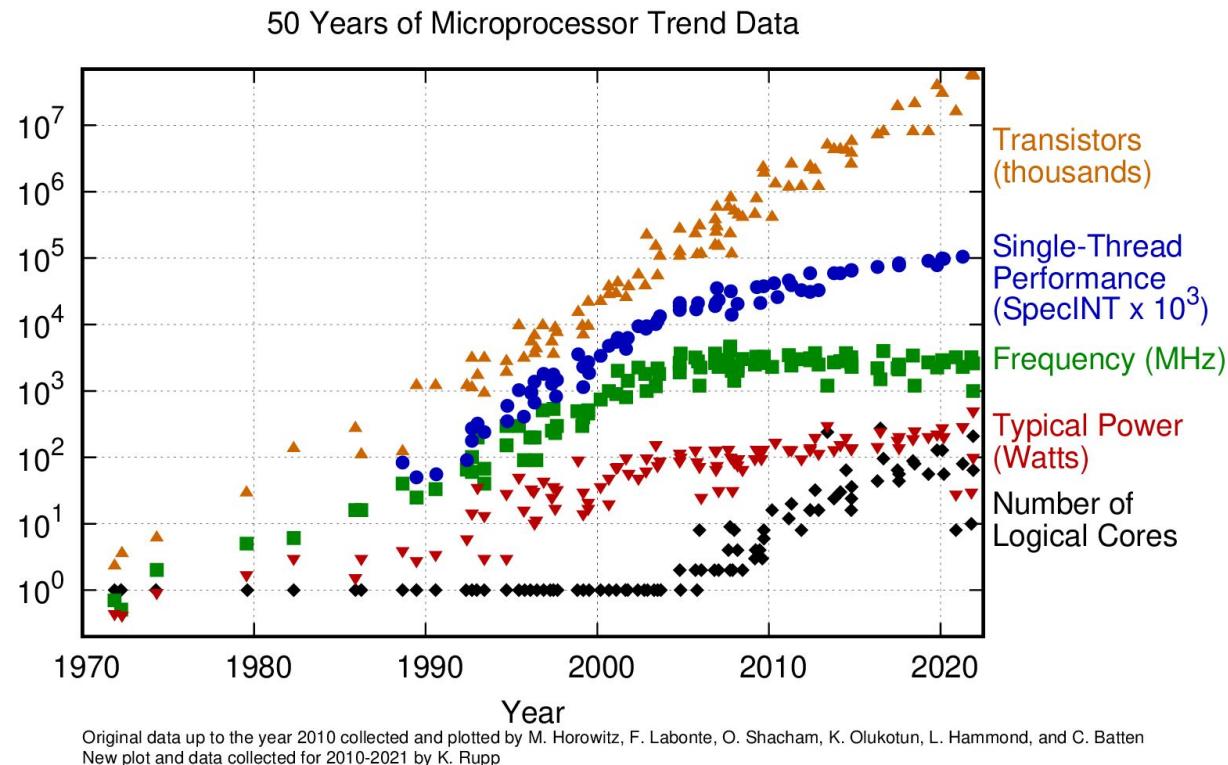
# Распределённая система

- Множество компьютеров, соединённых средой передачи данных
- Отправляют друг другу сообщения
- Решают общую задачу
- У каждого компьютера своя независимая память



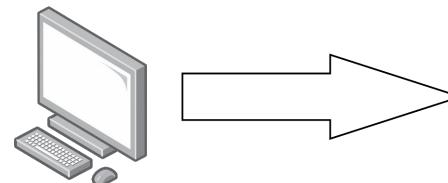
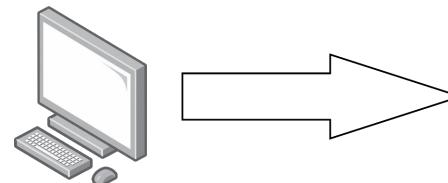
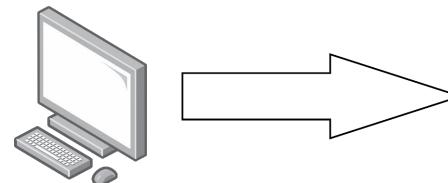
# Мотивация: Free Lunch is Over

- Скорости  
одного  
компьютера  
уже не  
хватает



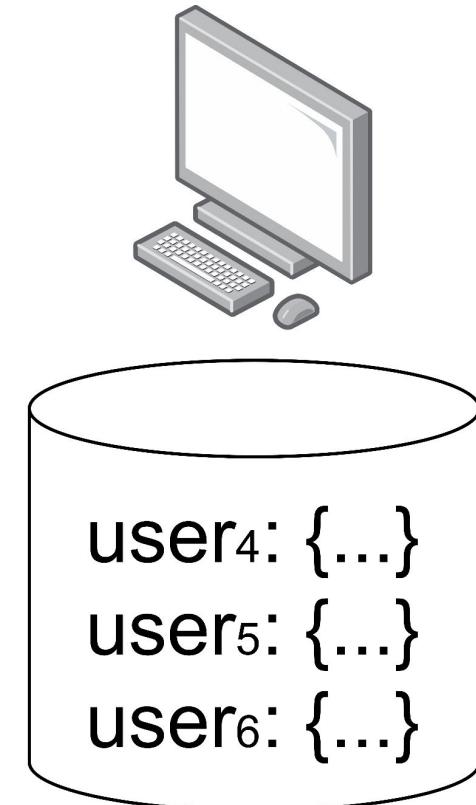
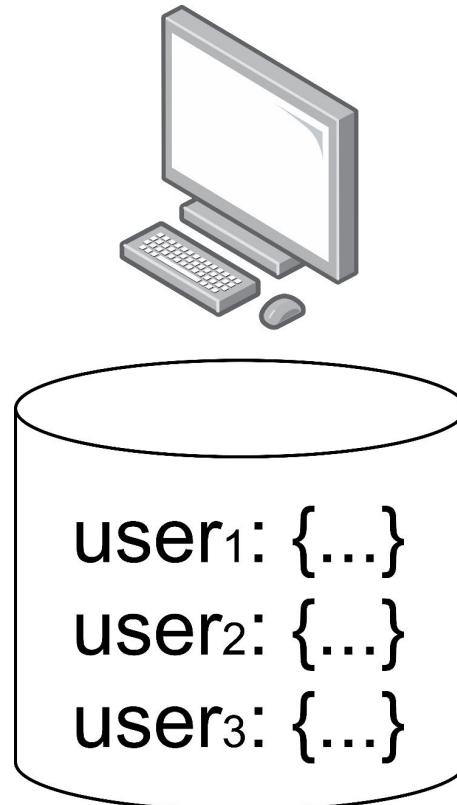
# Мотивация: параллельные вычисления

- Разделим работу на несколько компьютеров и сделаем её быстрее



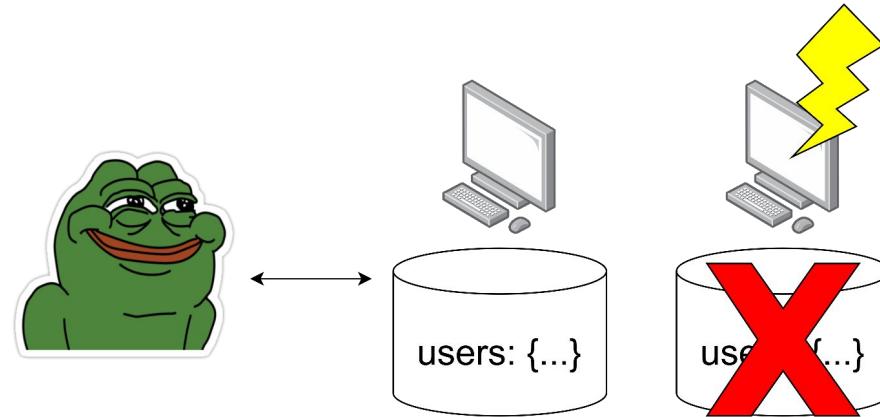
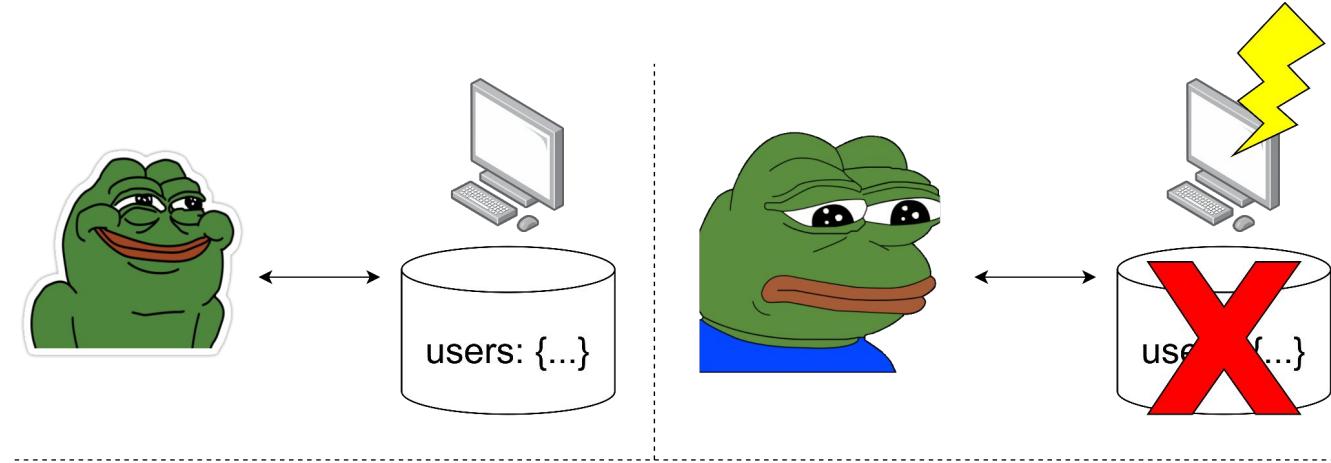
# Мотивация: увеличение доступной памяти

- Храним данных больше, чем доступно памяти на одном узле
- Шардирование
  - Спойлер!



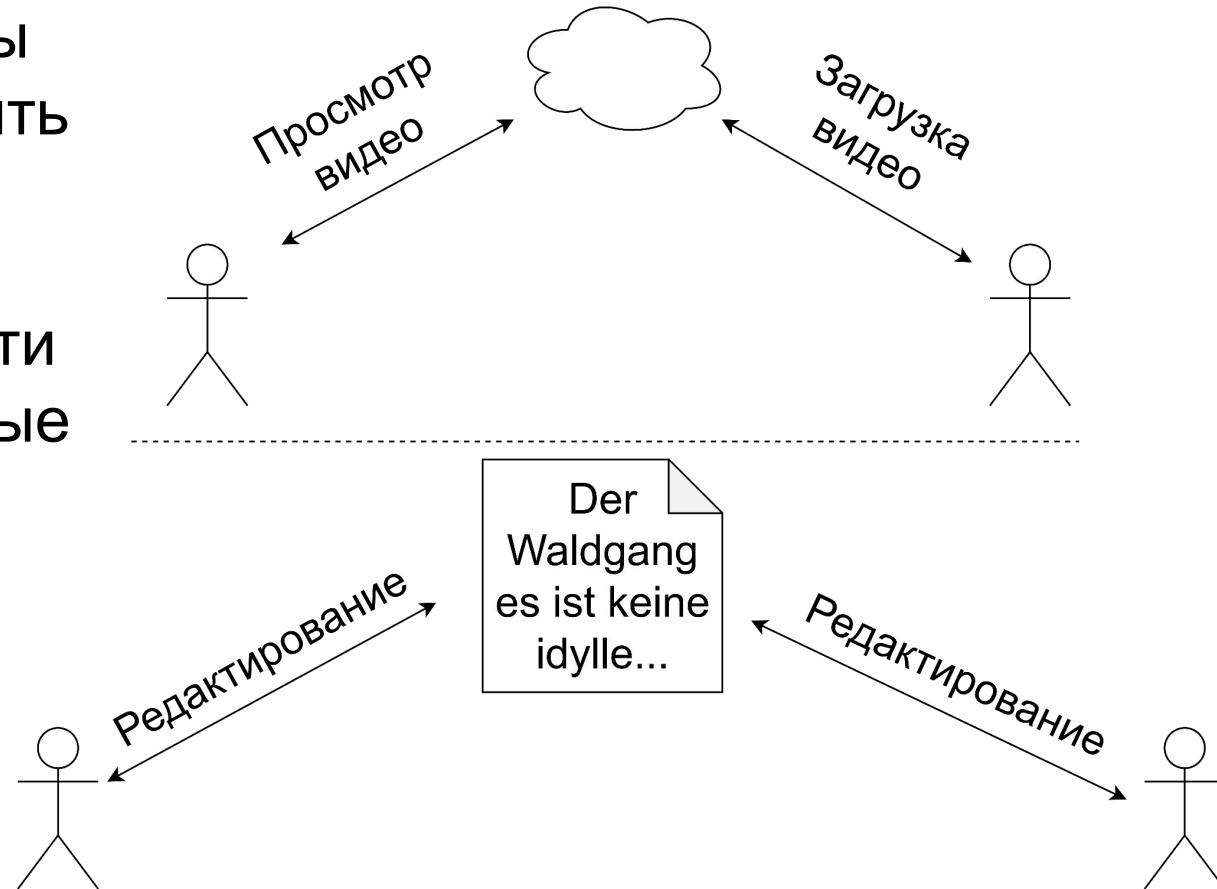
# Мотивация: увеличение надёжности

- Увеличиваем надёжность хранилища данных
- Репликация
  - Спойлер!



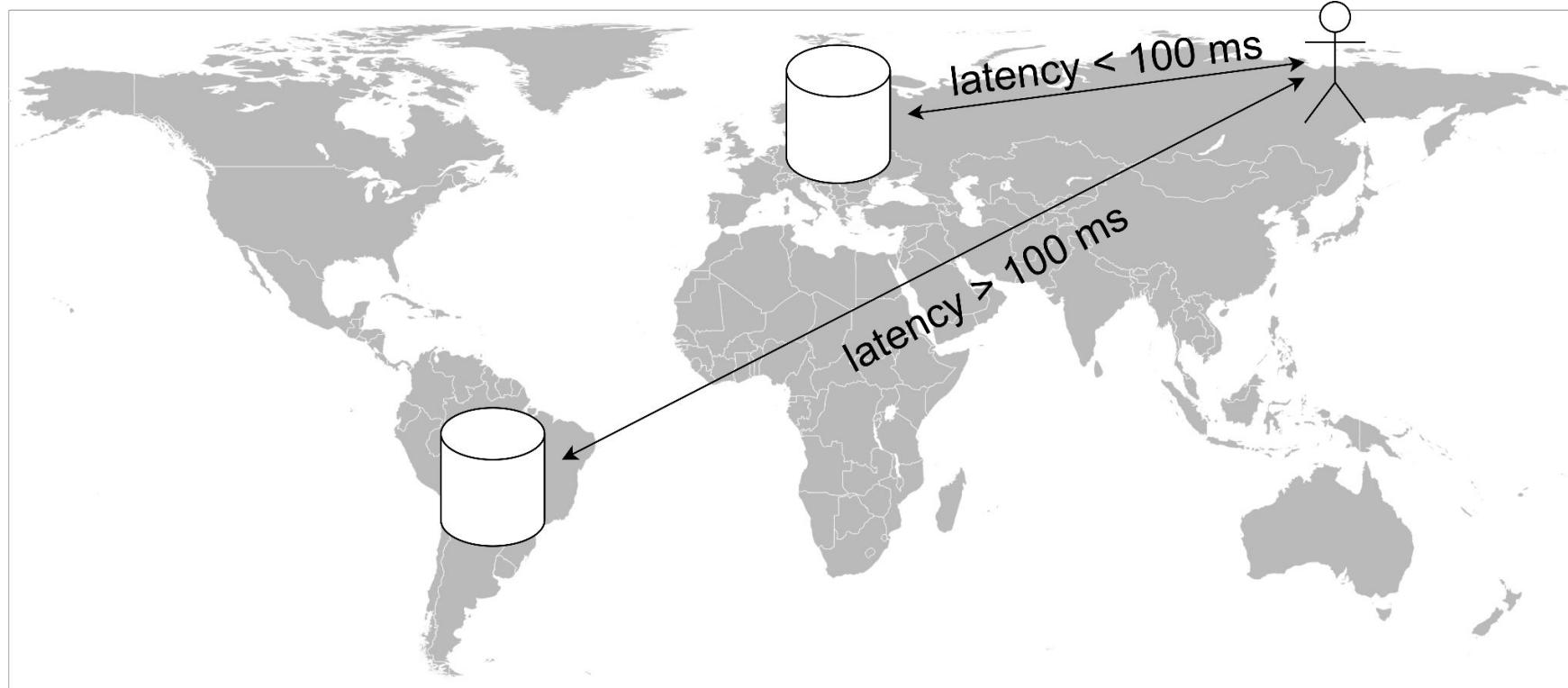
# Мотивация: функциональные требования

- Некоторые сервисы просто обязаны быть распределёнными
  - Видеохостинги
  - Социальные сети
  - Коллаборативные редакторы
  - ...



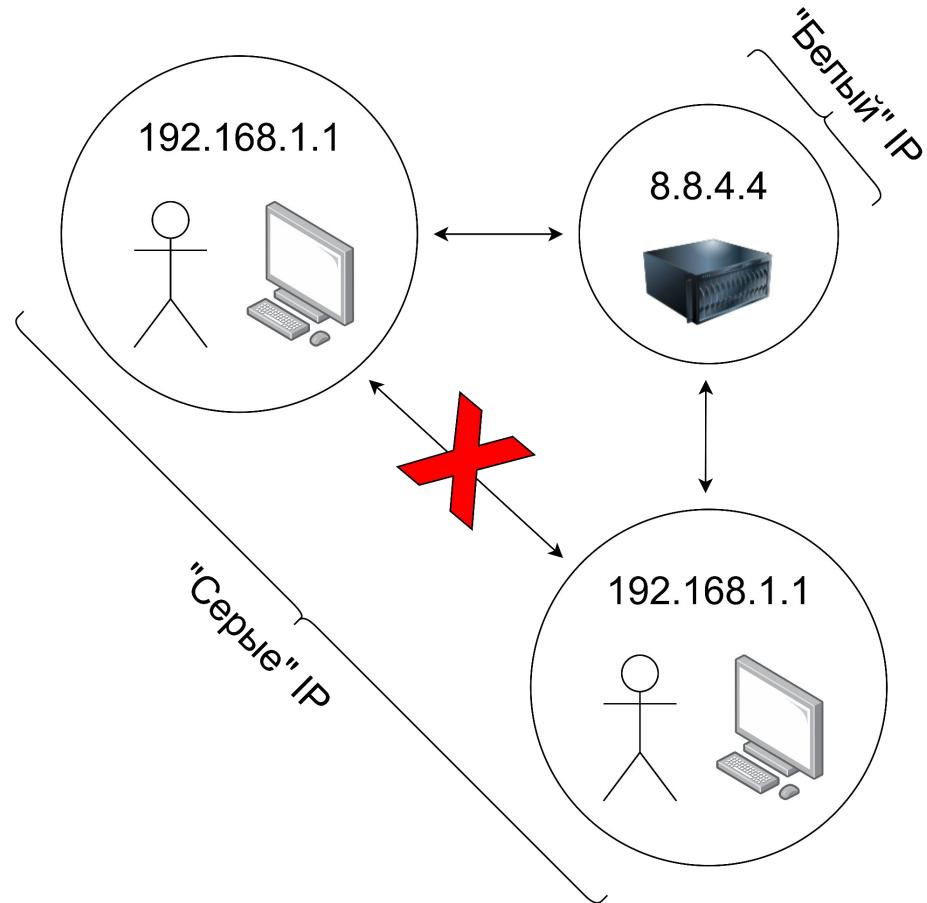
# Географическая распределённость

- Задержка обусловлена расстоянием до узла



# Неполная топология связи

- Частный случай географической распределённости
- Обусловлена существующей сетевой инфраструктурой
  - “Белые” и “серые” IPv4
  - NAT
  - ...



## Частичные отказы

- Пусть у нас есть единственный компьютер
- Вероятность его сбоя в произвольный день равна  $10^{-4}$ 
  - Это **очень** высокая надёжность
- Посчитать математическое ожидание количества дней безаварийной работы

## Частичные отказы

- Пусть у нас есть единственный компьютер
- Вероятность его сбоя в произвольный день равна  $10^{-4}$ 
  - Это **очень** высокая надёжность
- Посчитать математическое ожидание количества дней безаварийной работы
  - Геометрическое распределение
  - $$\sum_{k=0}^{\infty} (1-p)^k \cdot p \cdot k = \frac{1-p}{p} = \frac{1-10^{-4}}{10^{-4}} = 9999$$
  - 27 лет безаварийной работы в среднем

## Частичные отказы

- Есть  $n$  узлов
- Вероятность сбоя каждого равна  $p$
- Сбои независимы
- Посчитать вероятность того, что хотя бы один узел откажет

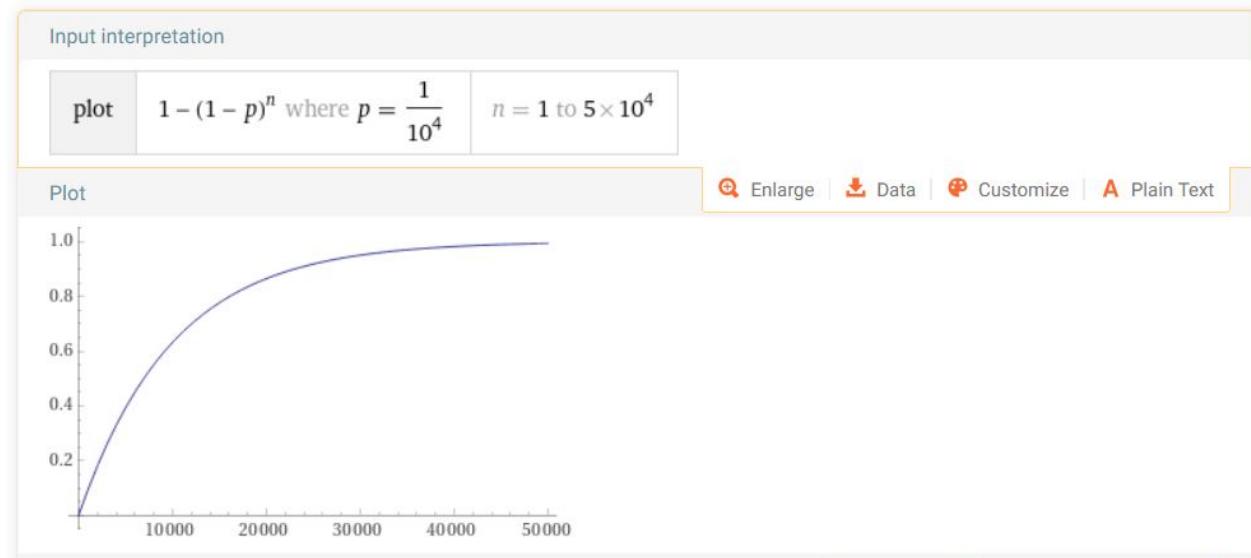
# Частичные отказы

Вероятность безаварийной работы одного узла равна  $1 - p$

Вероятность безаварийной работы всех узлов равна  $(1 - p)^n$

Вероятность сбоя хотя бы одного узла равна  $1 - (1 - p)^n$

- У Google более миллиона серверов
  - Наверное



## Частичные отказы

- *First, component failures are the norm rather than the exception. ... The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures.*
  - The Google File System



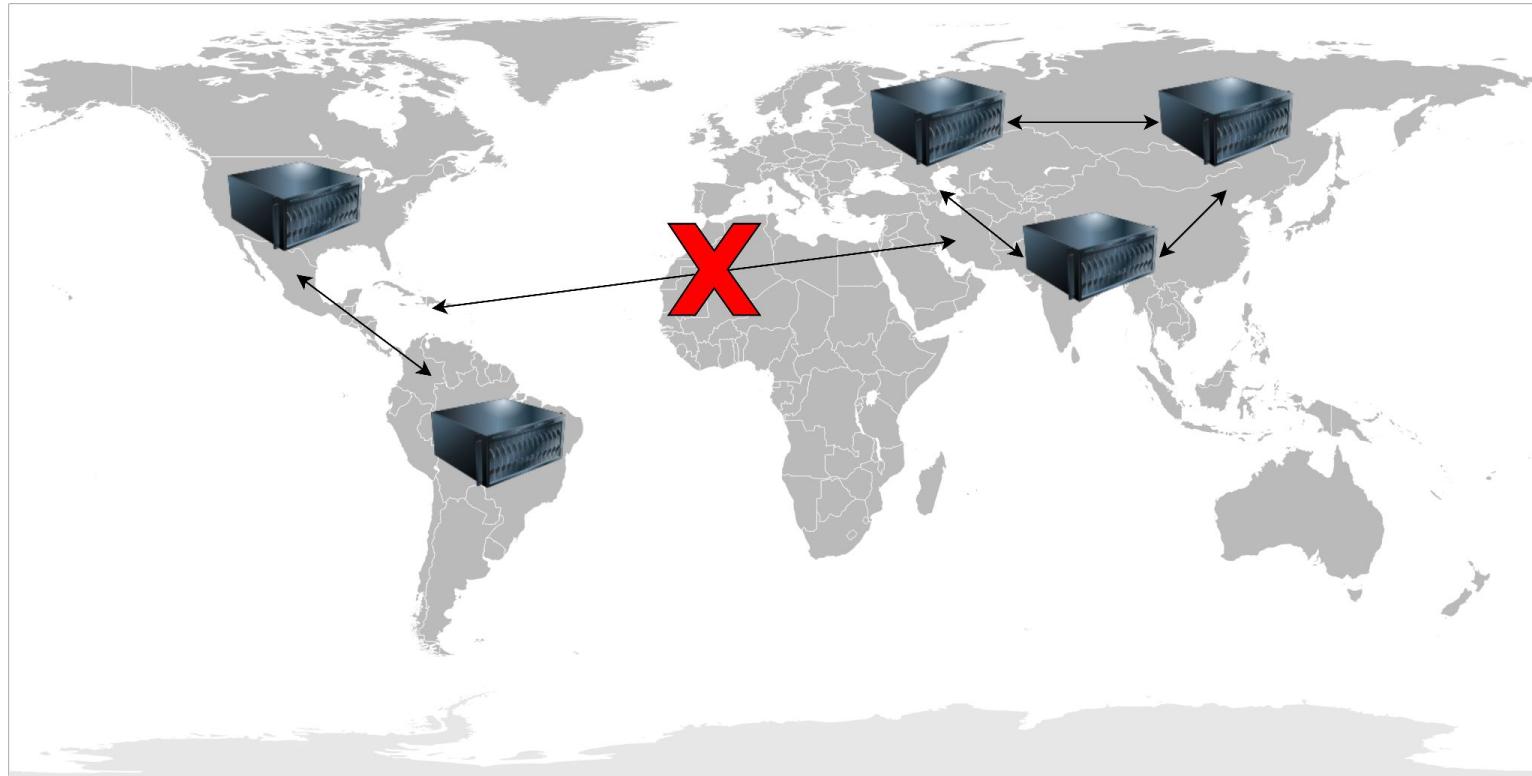
# Частичные отказы: обрывы связи

- Частный случай частичного отказа
- Это не шутка



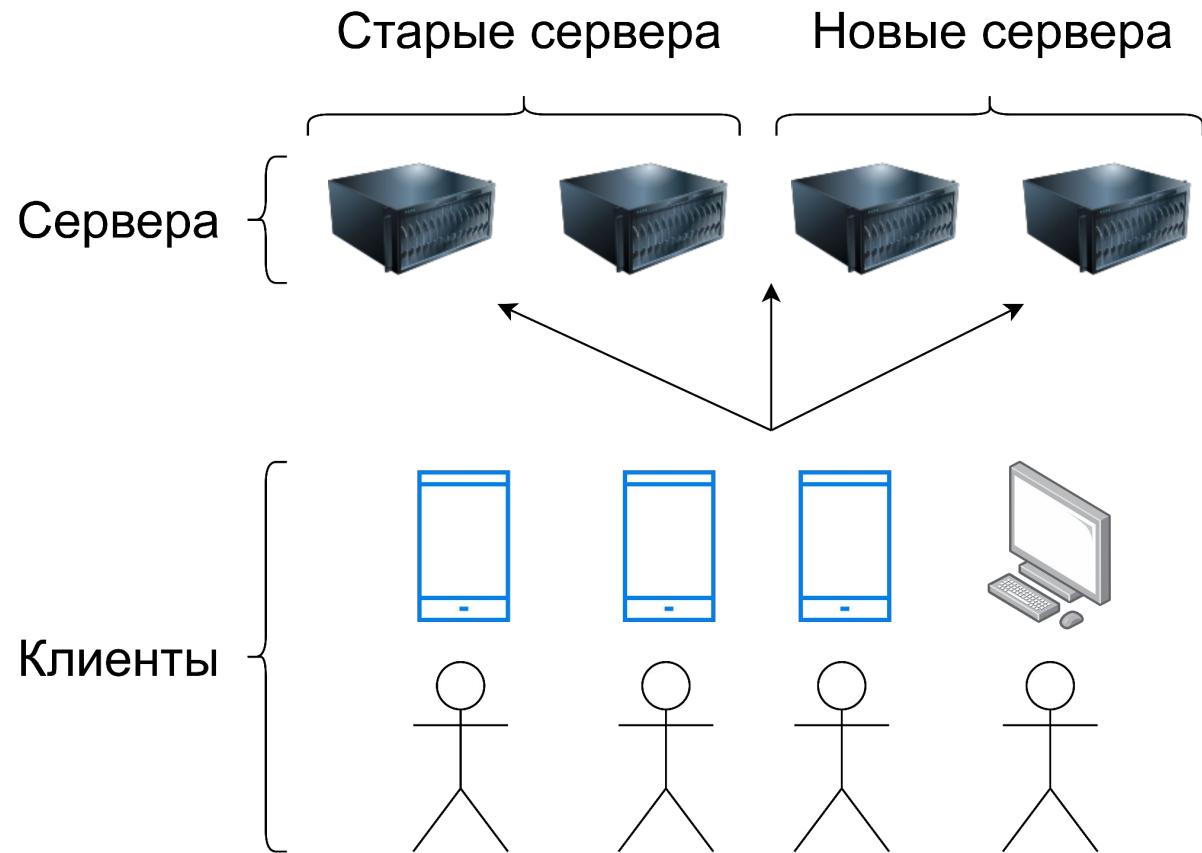
# Частичные отказы: обрывы связи

- Ведут к неполной топологии связи



# Гетерогенность железа

- У клиентов  
мобильные  
телефоны и  
слабые ПК
- В датацентрах  
стоят мощные  
сервера
- Обновить весь  
парк серверов  
одновременно  
нельзя



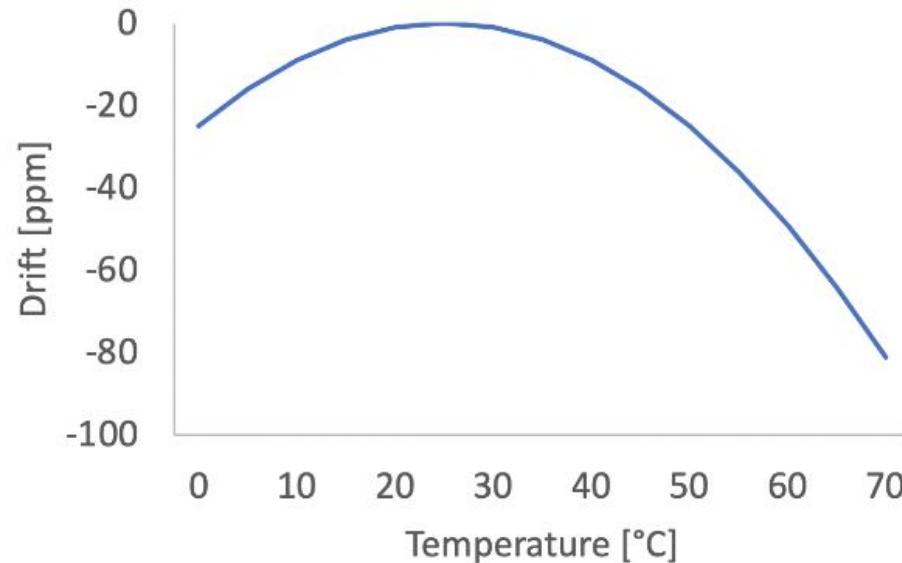
# Часы

- В современных компьютерах используются кварцевые часы
  - Реже используются атомные
- Кристалл кварца механически резонирует  $\eta$  раз в секунду
  - Отмеряют промежутки времени по  $1/\eta$  секунд



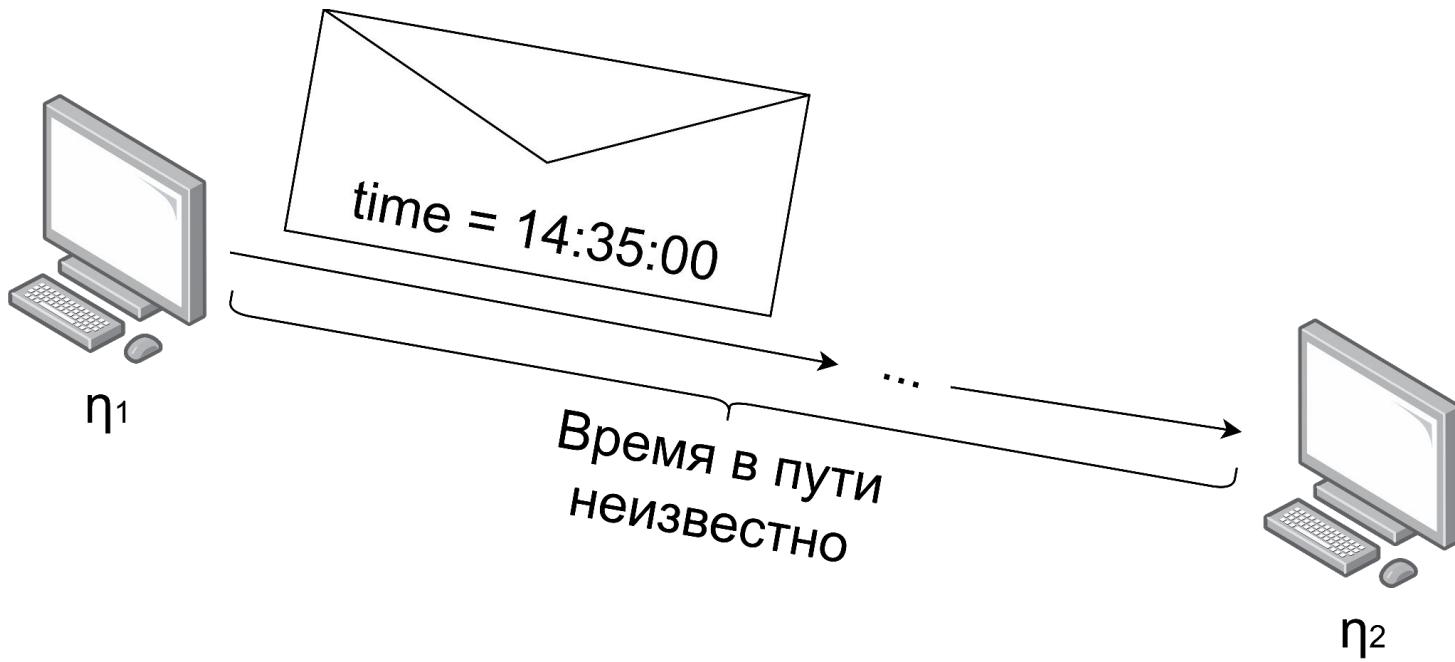
# Clock Drift

- На самом деле кварц резонирует с частотой  $\eta \pm \Delta$ 
  - С атомными часами та же проблема
- Часы на разных узлах начинают расходиться
- ppm - миллионная доля секунды
  - На столько отклоняется каждую секунду
- $20 * 24 * 60 * 60 / 1000000 = 1.7$  сек/день



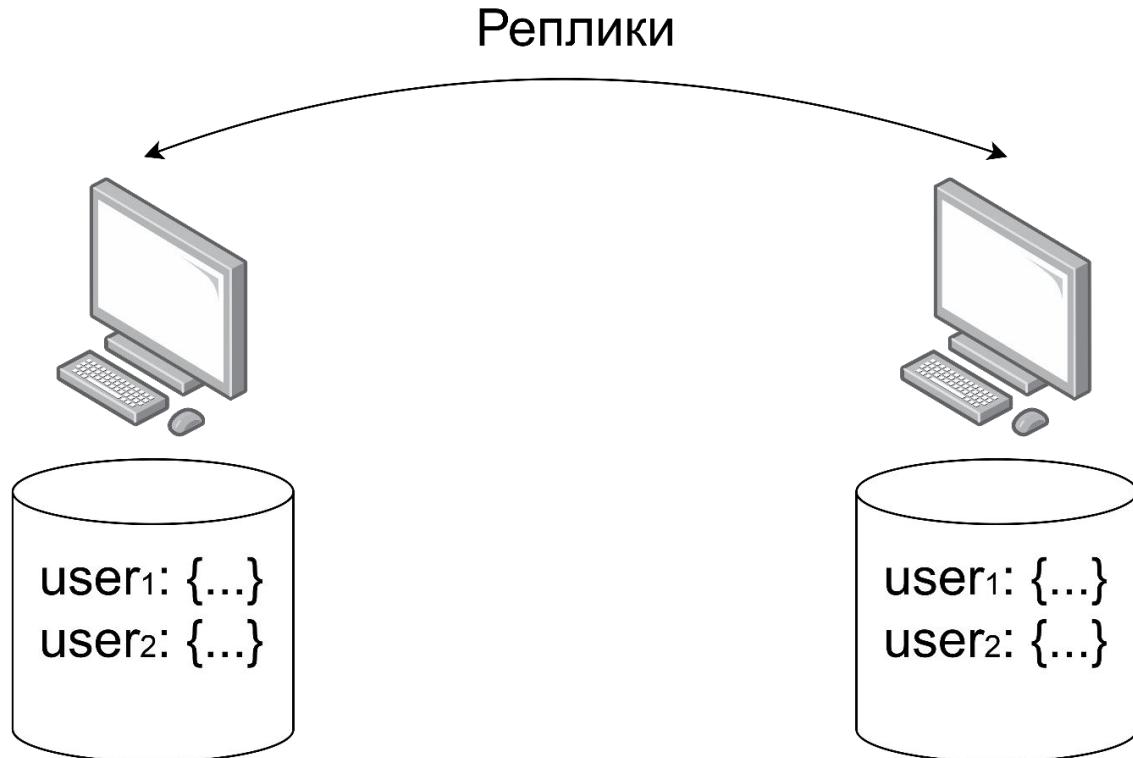
# Невозможность синхронизации часов

- Невозможно абсолютно точно синхронизировать время
- Физическое время в распределённой системе не определено
- Своё время на каждом узле



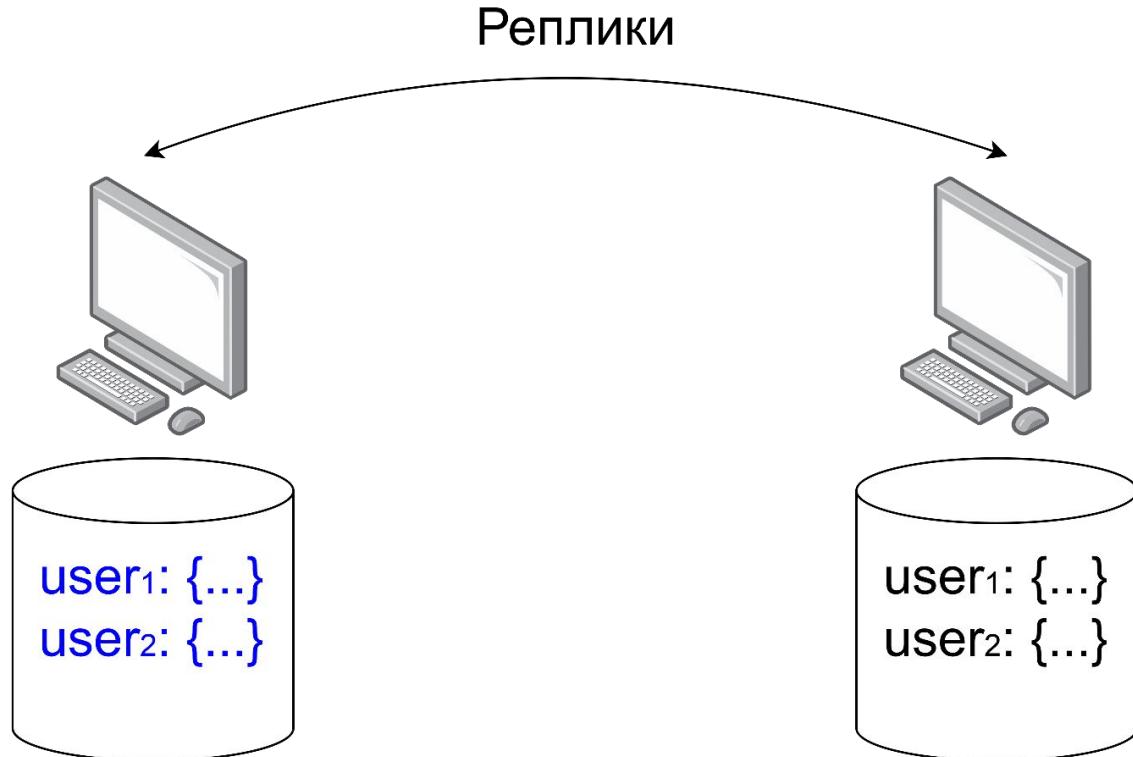
# Неопределённость состояния

- Хотим остановить систему и посмотреть на её состояние



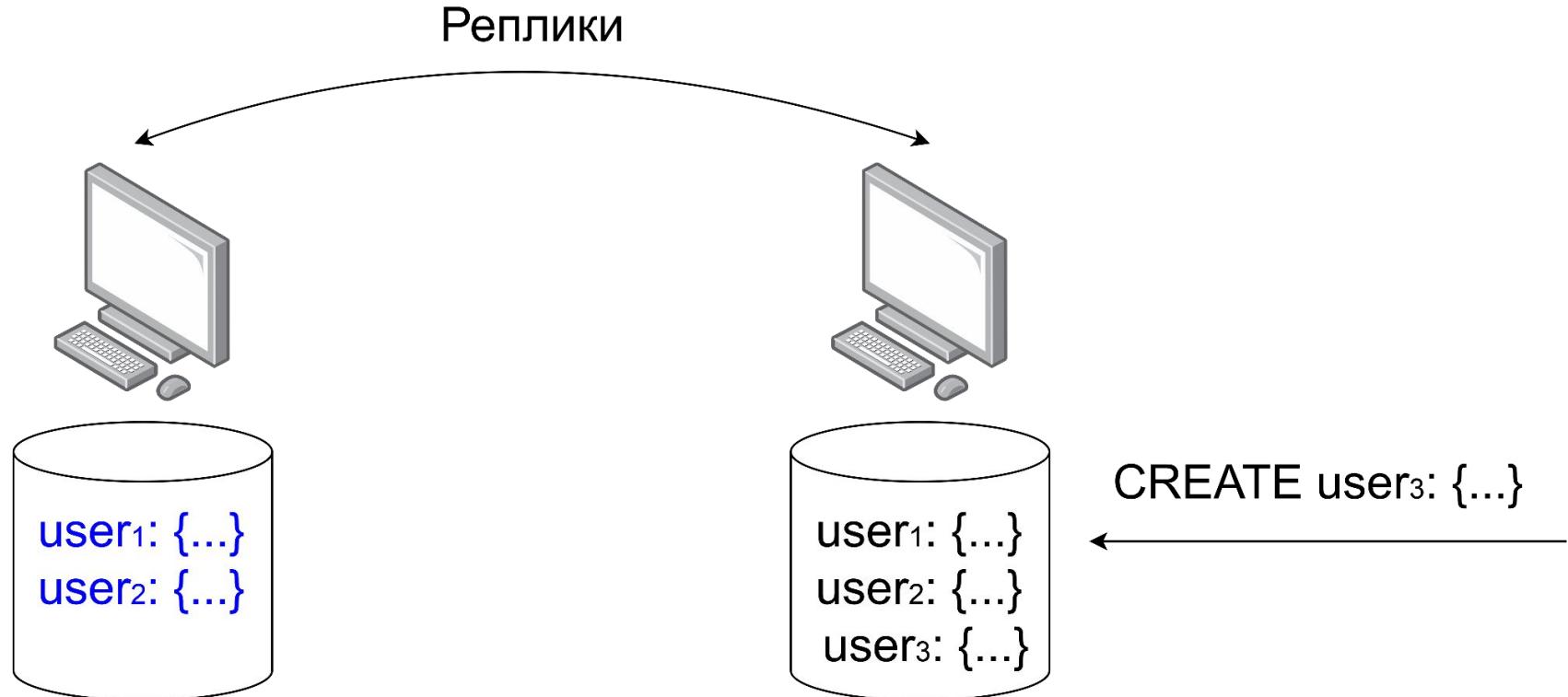
# Неопределённость состояния

- Оба состояния одновременно сохранить не можем



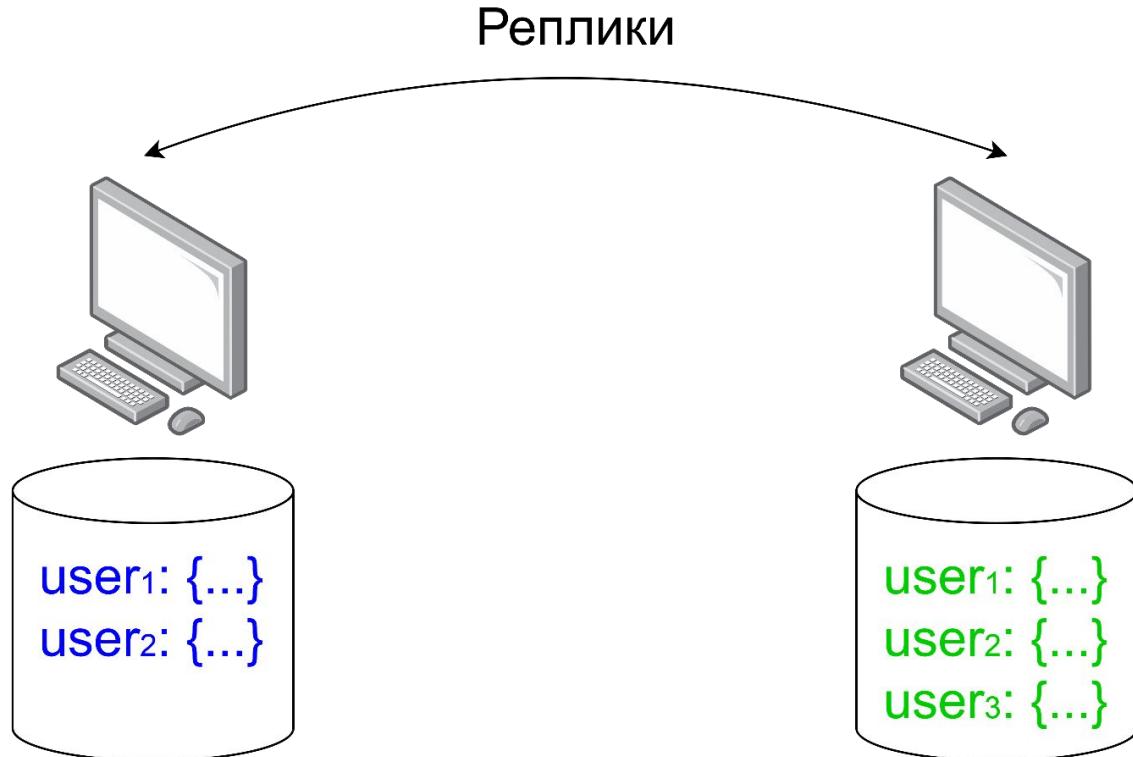
# Неопределённость состояния

- Состояние одного из узлов обновляется



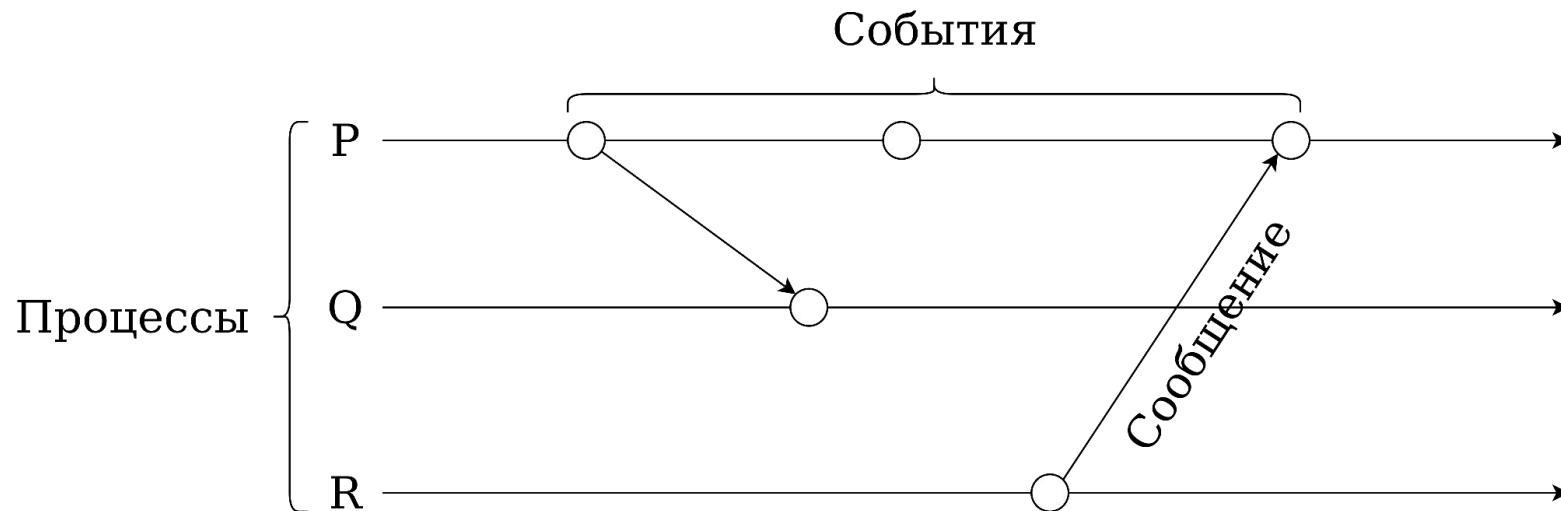
# Неопределённость состояния

- На втором узле состояние успело измениться



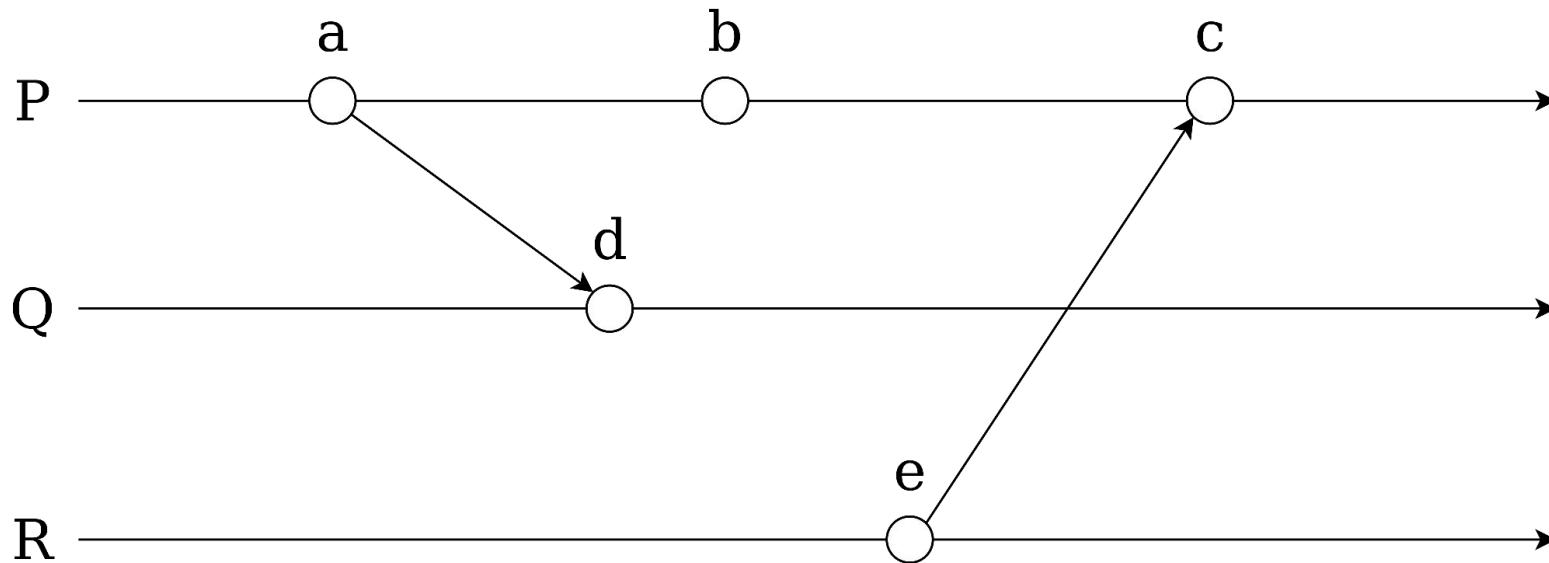
# Модель

- Множество процессов  $P$
- Множество событий  $E$ 
  - $\text{proc}(e) \in P$
- Множество сообщений  $M$ 
  - $\text{snd}(m) \in E, \text{rcv}(m) \in E$



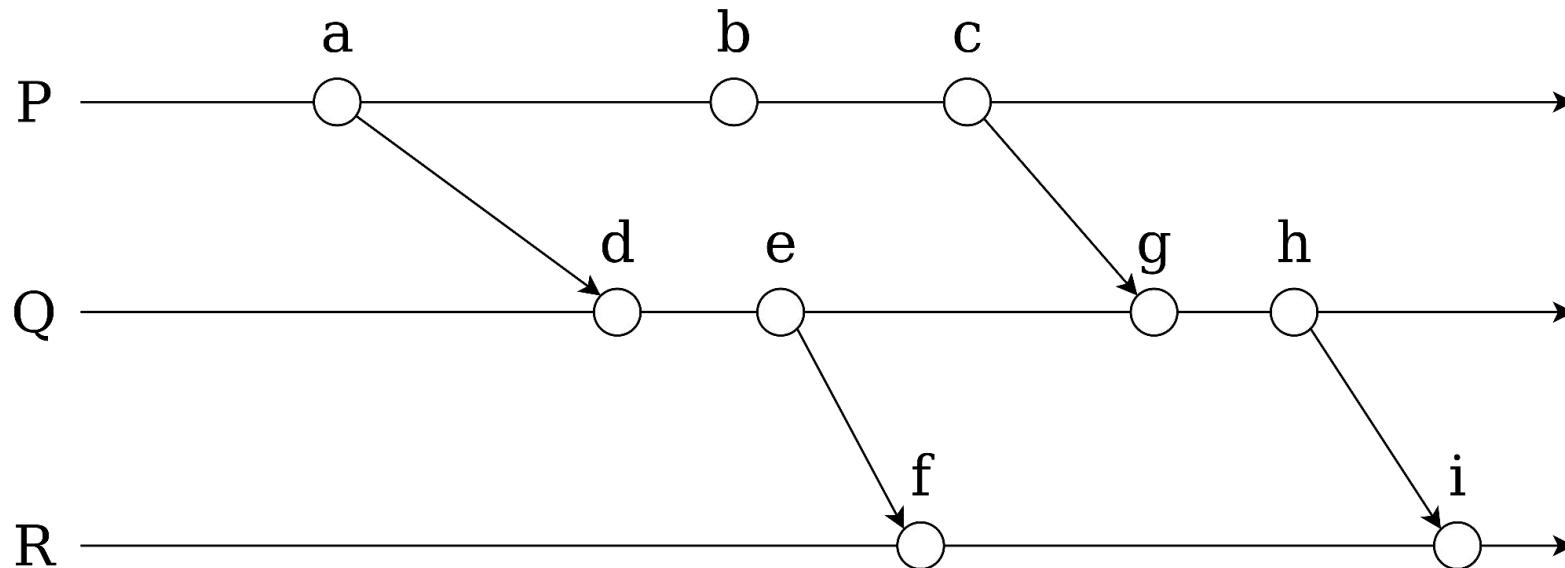
# Отношение прямой зависимости (<)

- Если  $\text{proc}(x) = \text{proc}(y)$ , то  $x < y$  или  $y < x$ 
  - $a < b < c$
- Если  $x = \text{snd}(m)$ ,  $y = \text{rcv}(m)$ , то  $x < y$ 
  - $a < d, e < c$



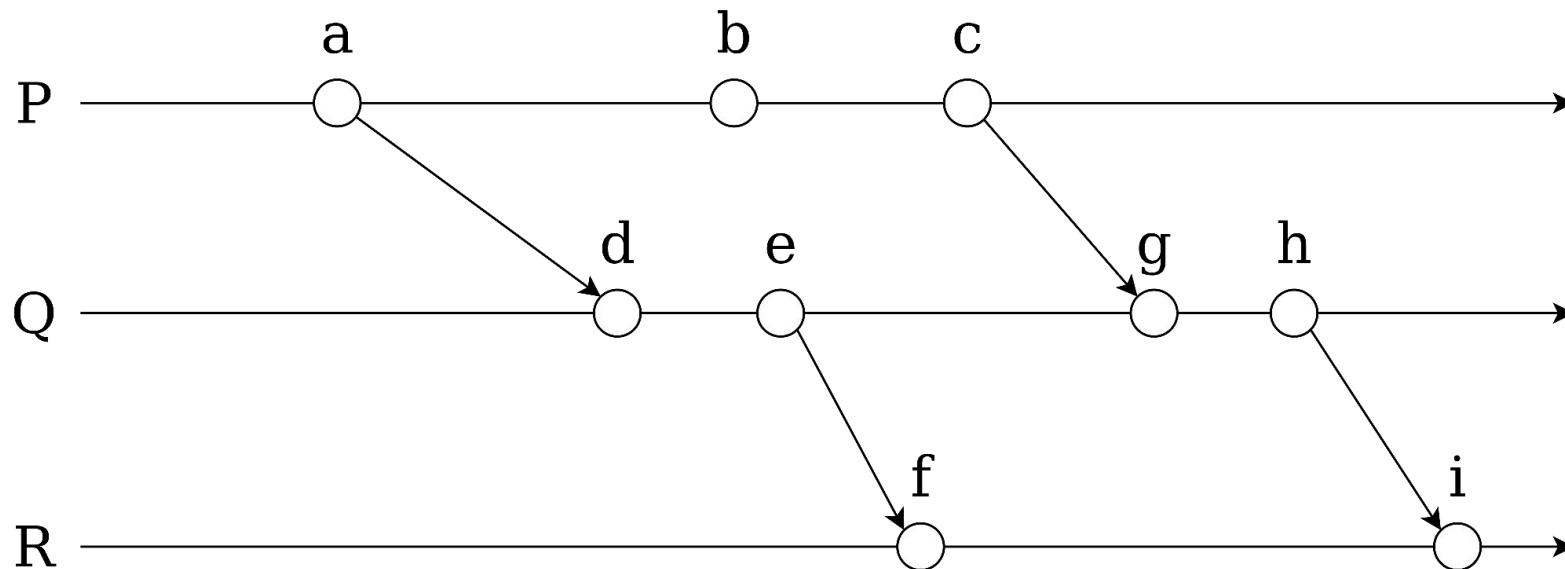
# Отношение произошло-до ( $\rightarrow$ )

- Транзитивное замыкание отношения прямой зависимости
  - Отношение частичного порядка
- $a \rightarrow f$
- $b \rightarrow i$



# Параллельные события

- $x \parallel y$ , если  $x$  не произошло-до  $y$  и  $y$  не произошло-до  $x$
- $b \parallel e$
- $c \parallel f$

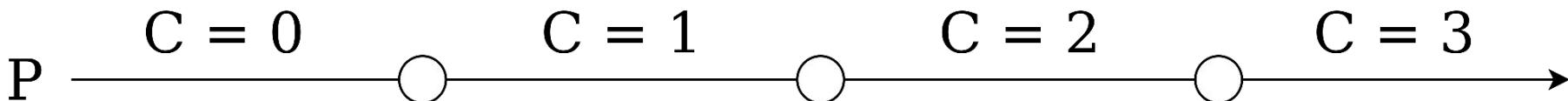


## Логические часы

- Определим натуральнозначную функцию  $C : E \rightarrow \mathbb{N}$ , такую что
  - $\forall e, f \in E : e \rightarrow f \Rightarrow C(e) < C(f)$
- Назовём такую функцию логическими часами
  - $C(e)$  назовём логическим временем события  $e$

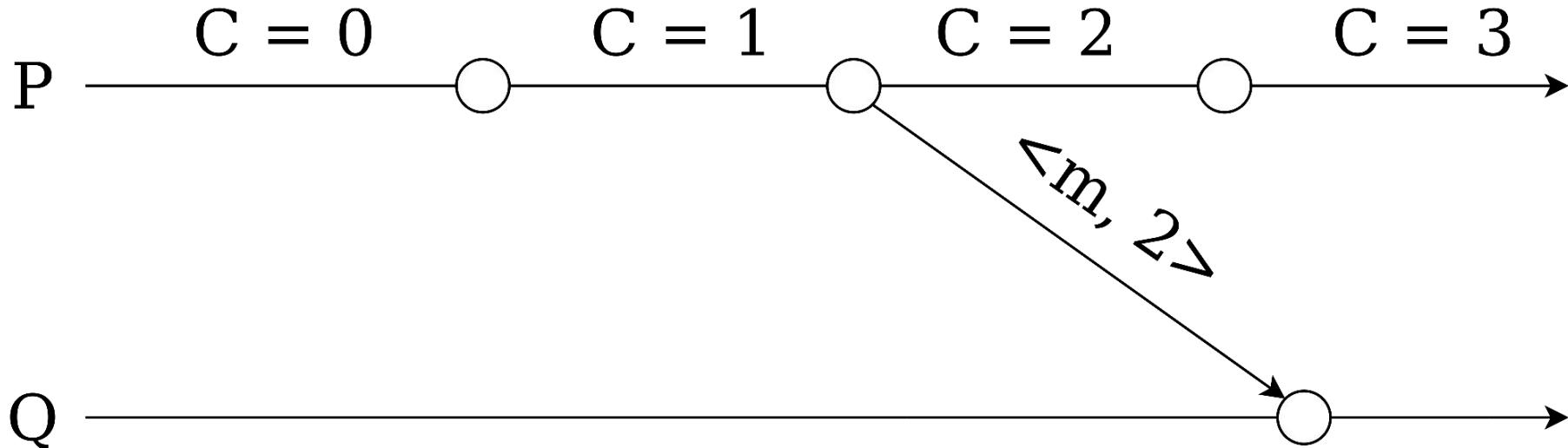
# Логические часы Лампорта

- Каждый процесс локально хранит своё логическое время
  - Изначально  $C := 0$
- При каждом событии процесс увеличивает логическое время на 1



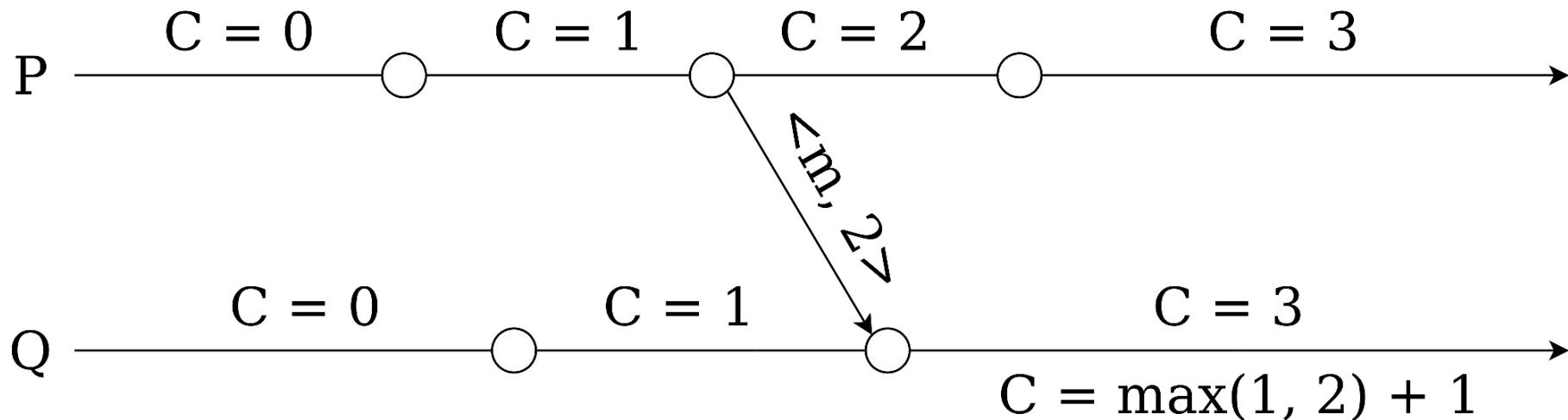
# Логические часы Лампорта

- При каждой посылке сообщения посылаем значение счётчика вместе с сообщением



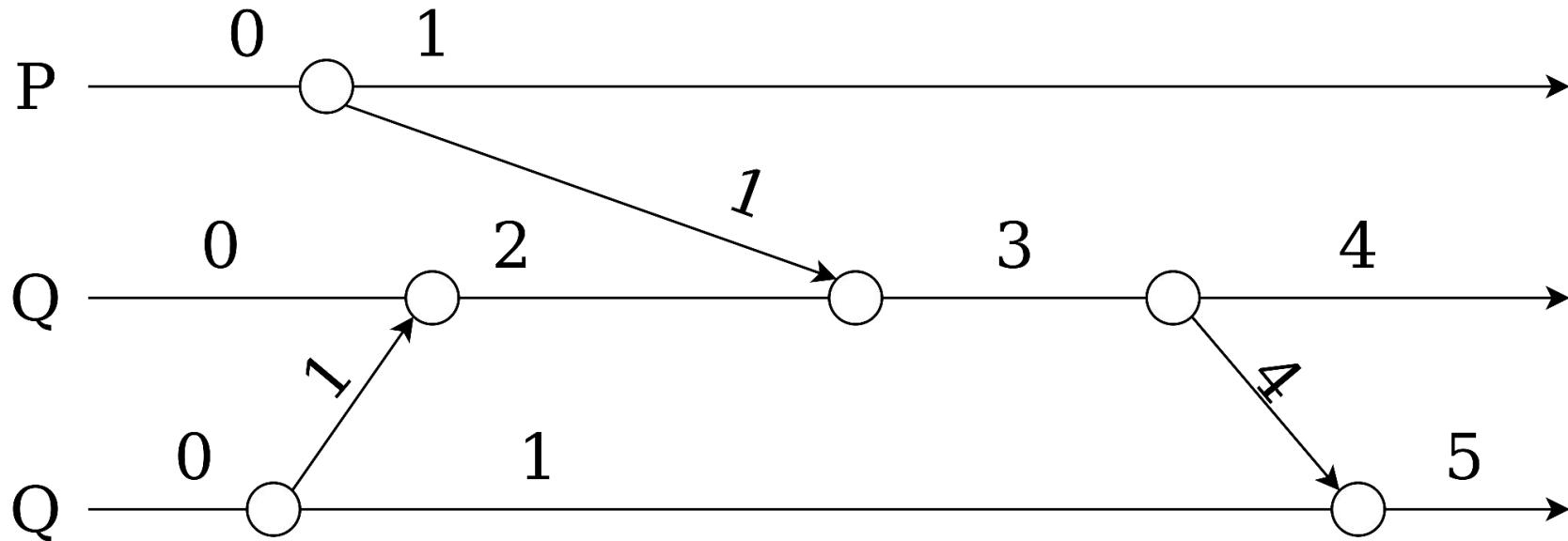
# Логические часы Лампорта

- При каждом получении сообщения берём максимум из локального и полученного счётчиков
  - Увеличиваем результат на 1



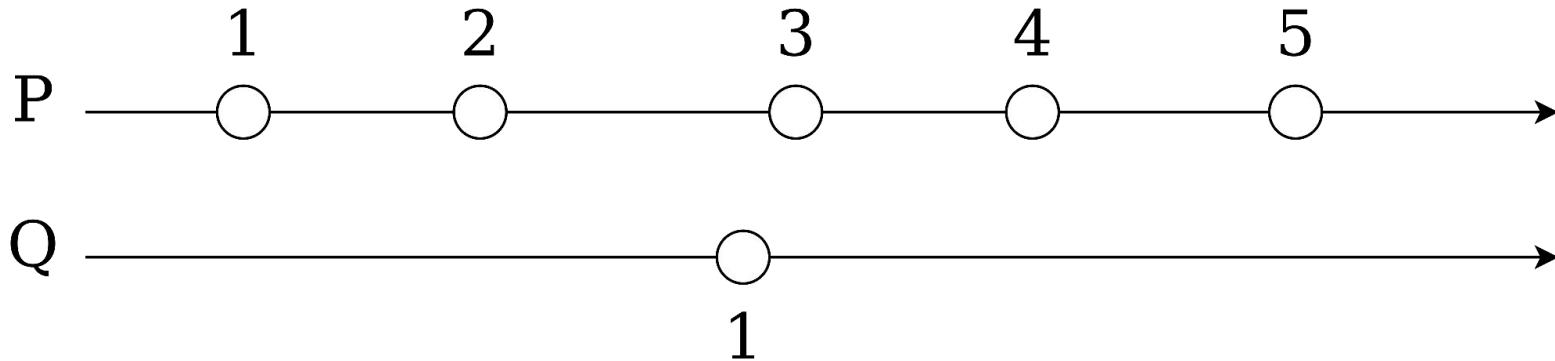
# Логические часы Лампорта

- Логическое время строго возрастает вдоль каждого пути
  - Путь может идти вперёд по событиям процесса
  - И по стрелочкам отправки-получения сообщений
- Свойство логических часов выполняется



# Логические часы Лампорта

- Может ли из  $C(x) < C(y)$  следовать  $x \rightarrow y$ ?
  - Нет



- Из  $C(x) < C(y)$  следует что либо  $x \rightarrow y$ , либо  
 $x \parallel y$ 
  - То есть следует что не  $y \rightarrow x$

## Более точные часы

- Хотим уметь по часам однозначно определять, как связаны два события
- Возможно ли для чисел?

$$\left\{ \begin{array}{ll} x \rightarrow y & \text{если } C(x) < C(y) \\ y \rightarrow x & \text{если } C(y) < C(x) \\ y = x & \text{если } C(y) = C(x) \\ y || x & \text{иначе} \end{array} \right.$$

# Векторные часы

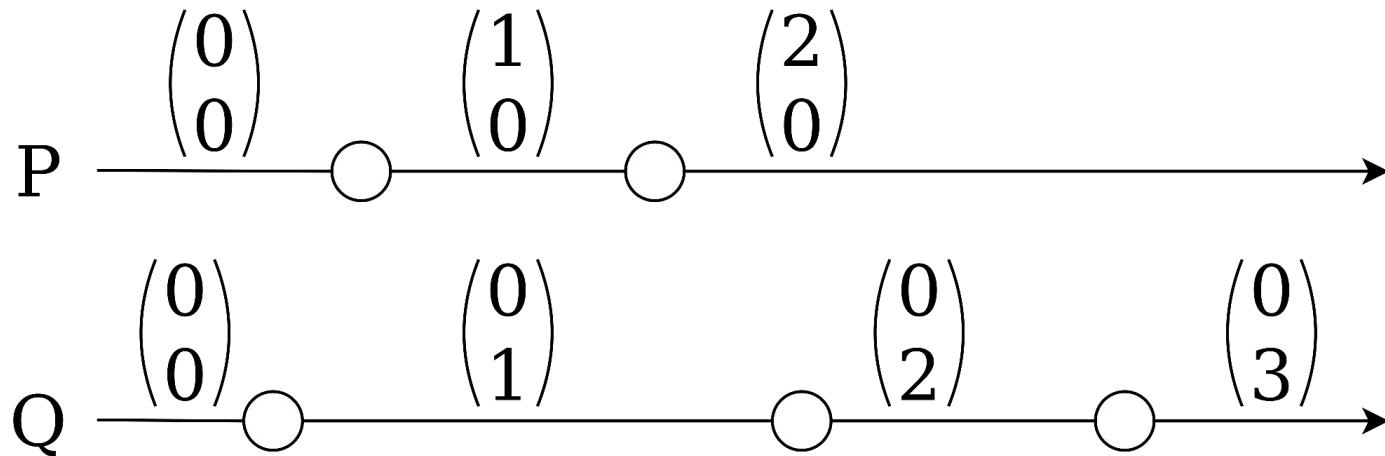
- Используем векторзначную функцию  $V : E \rightarrow \mathbb{N}^n$ 
  - $n$  – число процессов
- Вектора сравниваются покомпонентно
- Вектора могут быть:
  - Меньше или равны
  - Строго меньше
  - Больше или равны
  - Строго больше
  - Равны
  - Не сравнимы

$$\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} < \begin{bmatrix} 3 \\ 0 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} \text{ не сравнимо с } \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

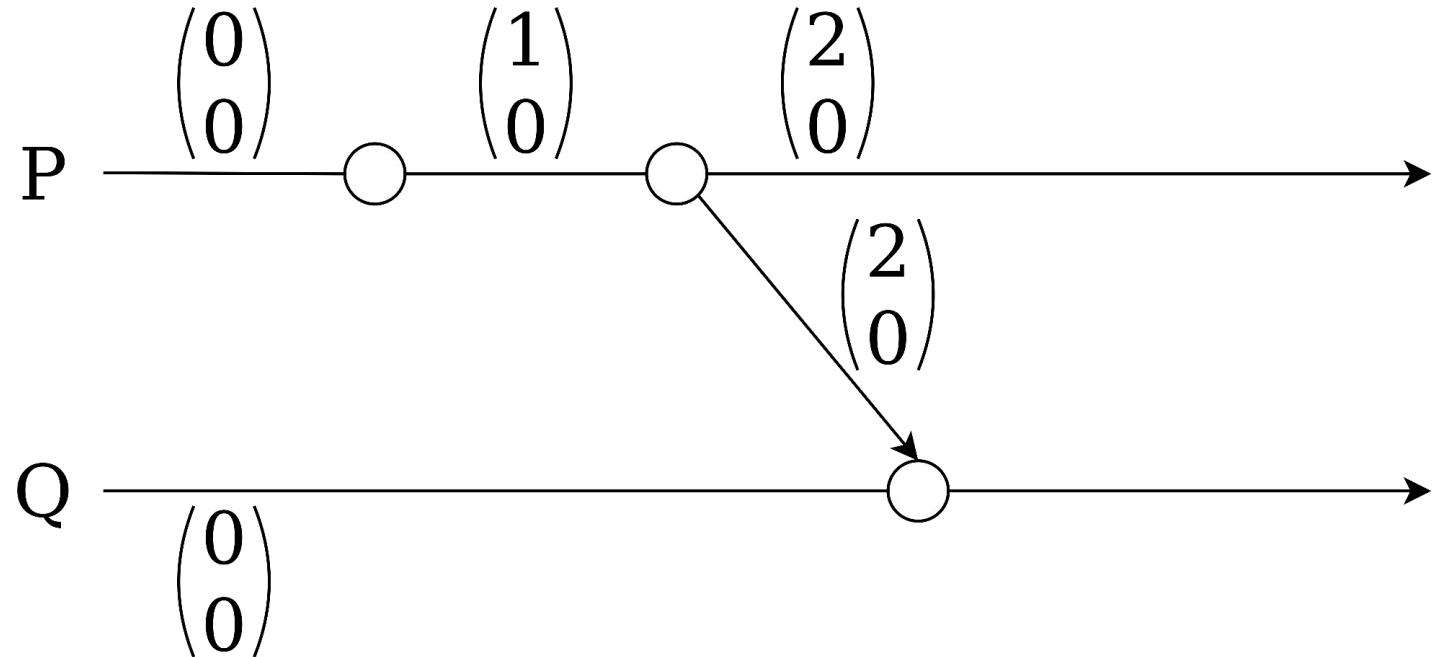
# Векторные часы

- Каждый процесс хранит вектор целых чисел
  - Размерность - количество процессов в системе
  - Изначально  $v[1..n] = 0$
- $i$ -ый процесс увеличивает  $i$ -ую компоненту времени при каждом событии



# Векторные часы

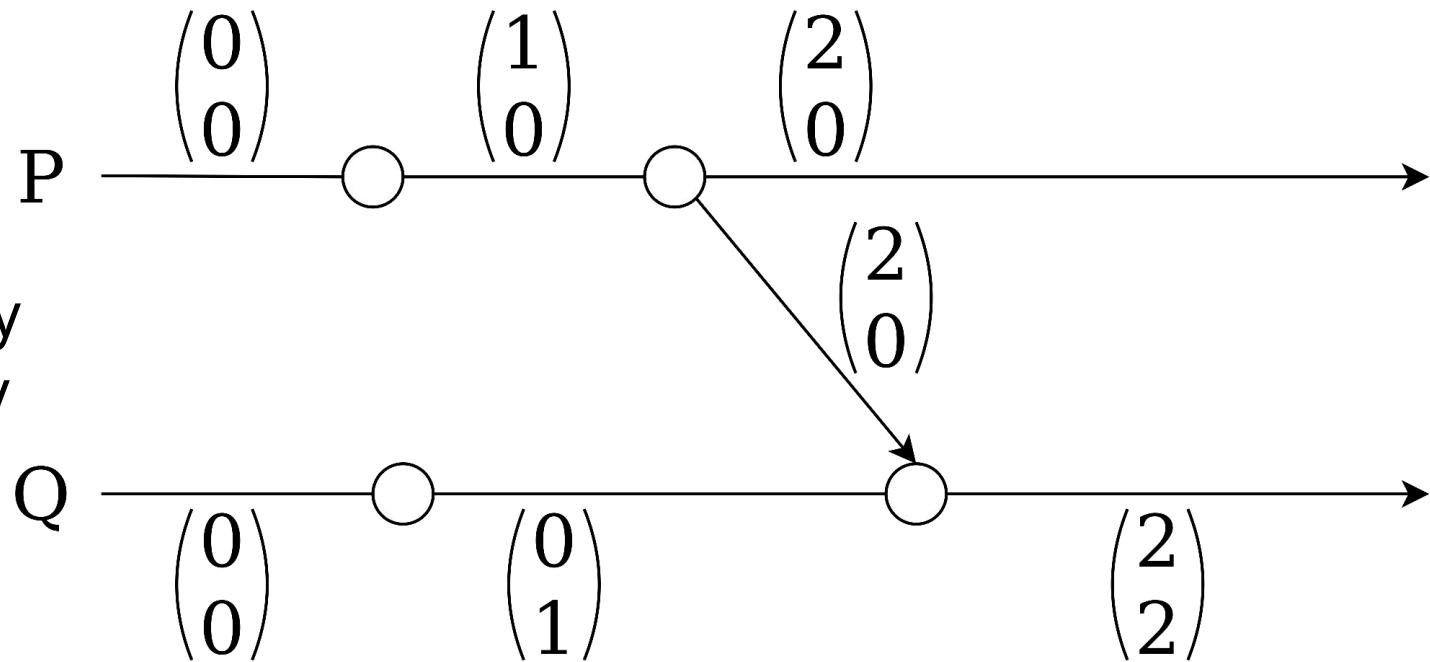
- При каждой посылке сообщения прикрепляем векторное время к сообщению



# Векторные часы

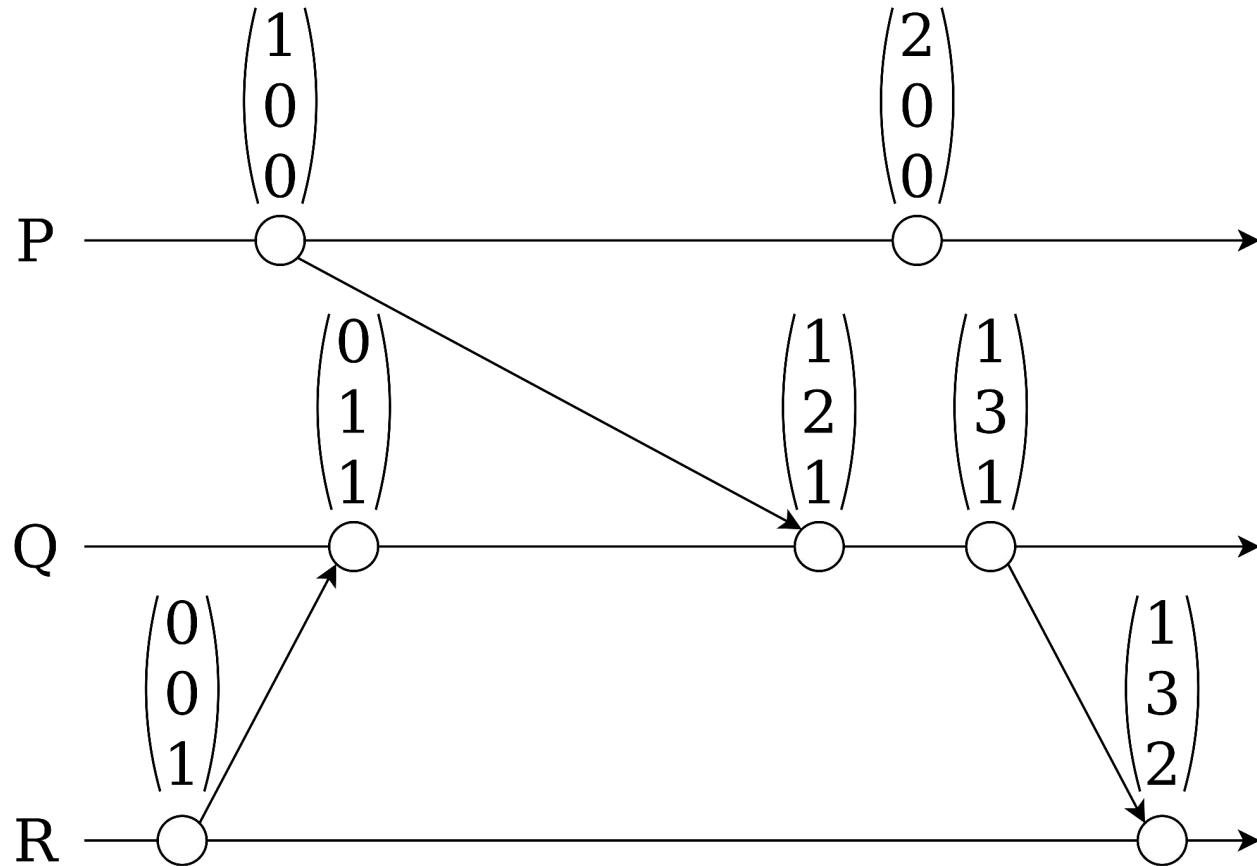
- При получении сообщения берём покомпонентный максимум векторов

- Не забываем увеличить локальную компоненту на единицу



# Векторные часы

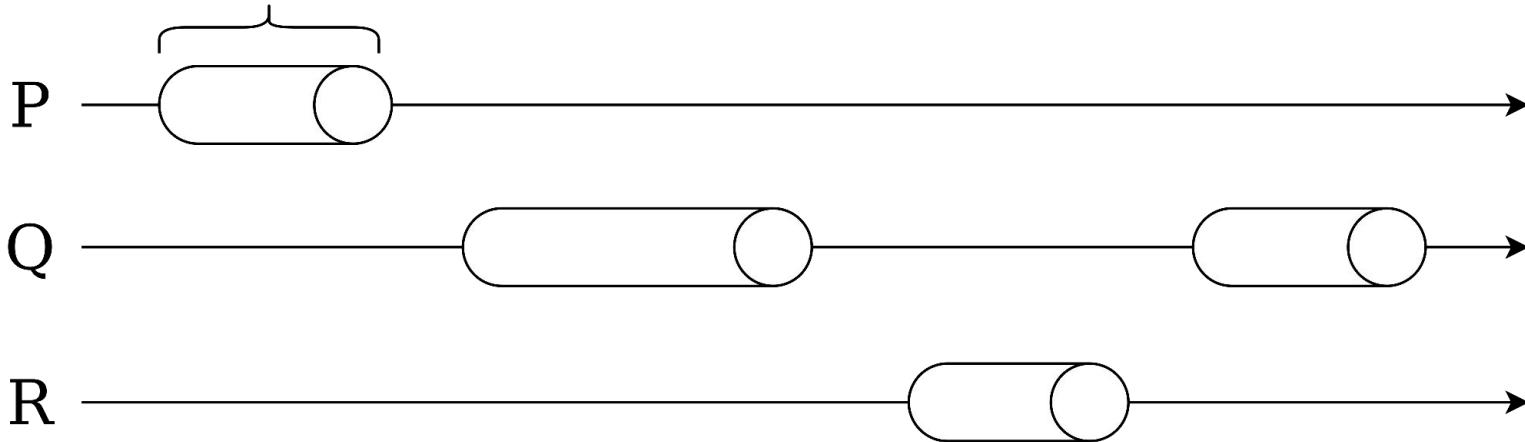
- $v[i]$  обозначает количество событий на  $i$ -ом процессе, произошедших до нашего события



# Взаимное исключение / Критическая секция

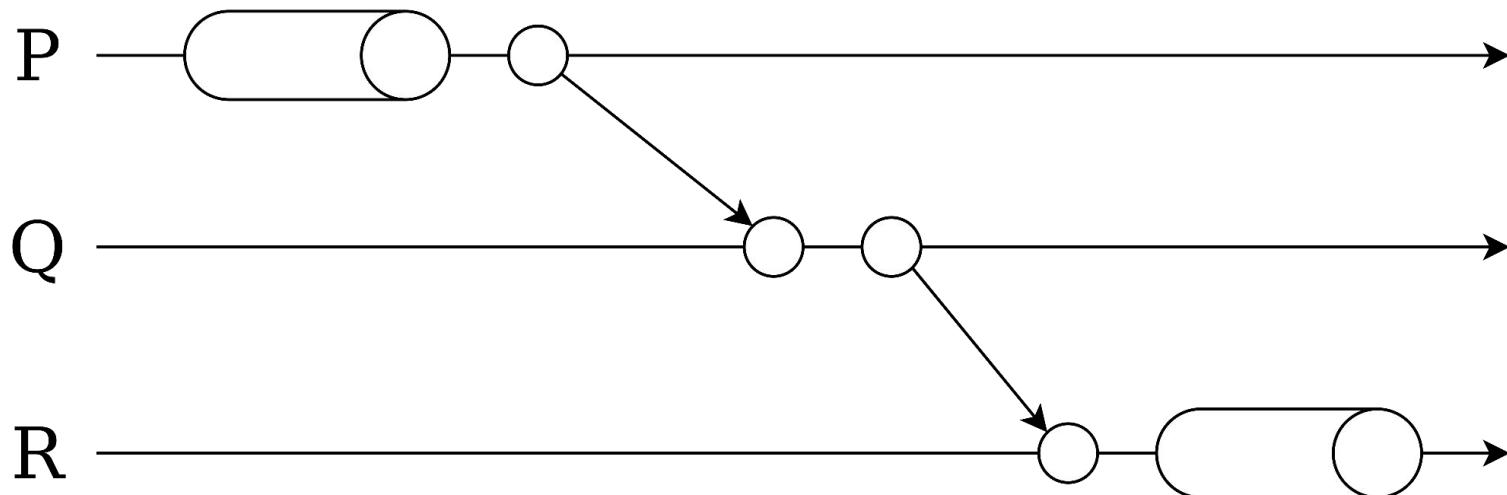
- В критической секции может находиться только один процесс
  - Безопасный доступ к разделяемому ресурсу

Критическая  
секция



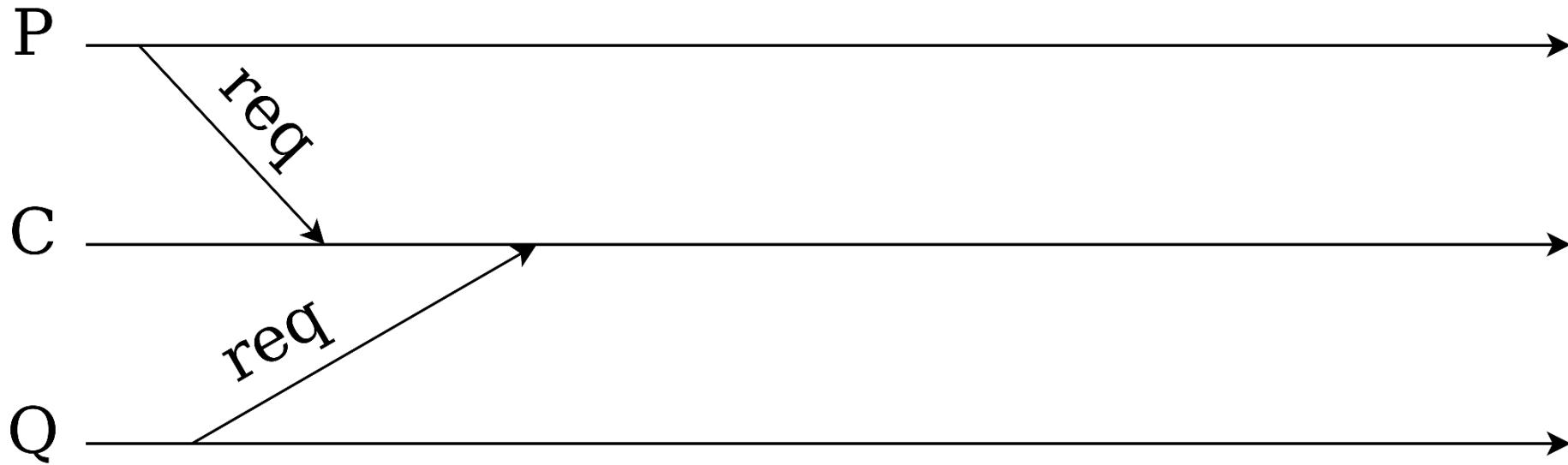
# Взаимное исключение: определение

- Критическая секция состоит из двух событий: Enter и Exit
- Два процесса не должны находиться в КС параллельно
- $\text{Exit}_i \rightarrow \text{Enter}_{i+1}$



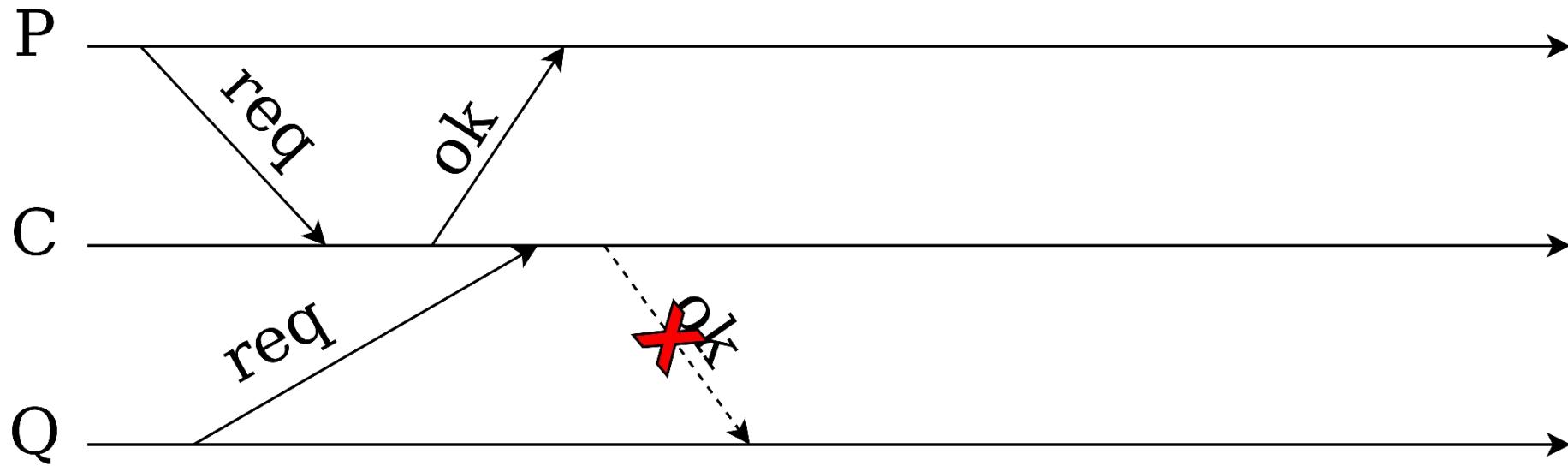
# Централизованный алгоритм

- Есть выделенный процесс-координатор
- Остальные процессы отправляют к нему запросы



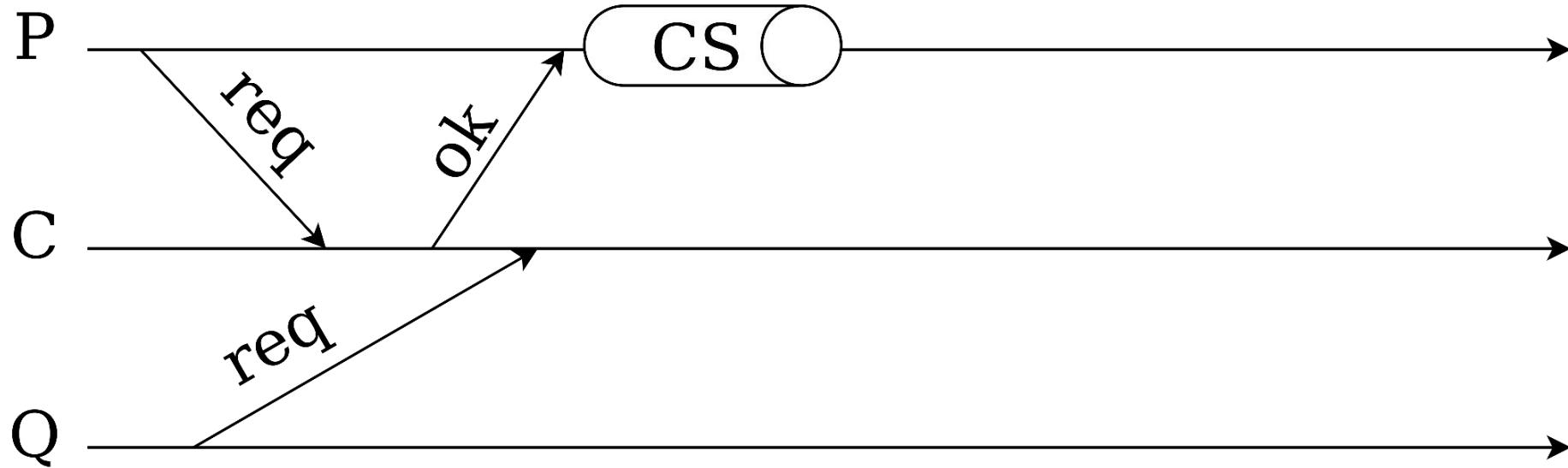
# Централизованный алгоритм

- Координатор отвечает ОК первому запросившему
- Остальным не отвечает ничего
  - Сохраняет эти процессы в очередь



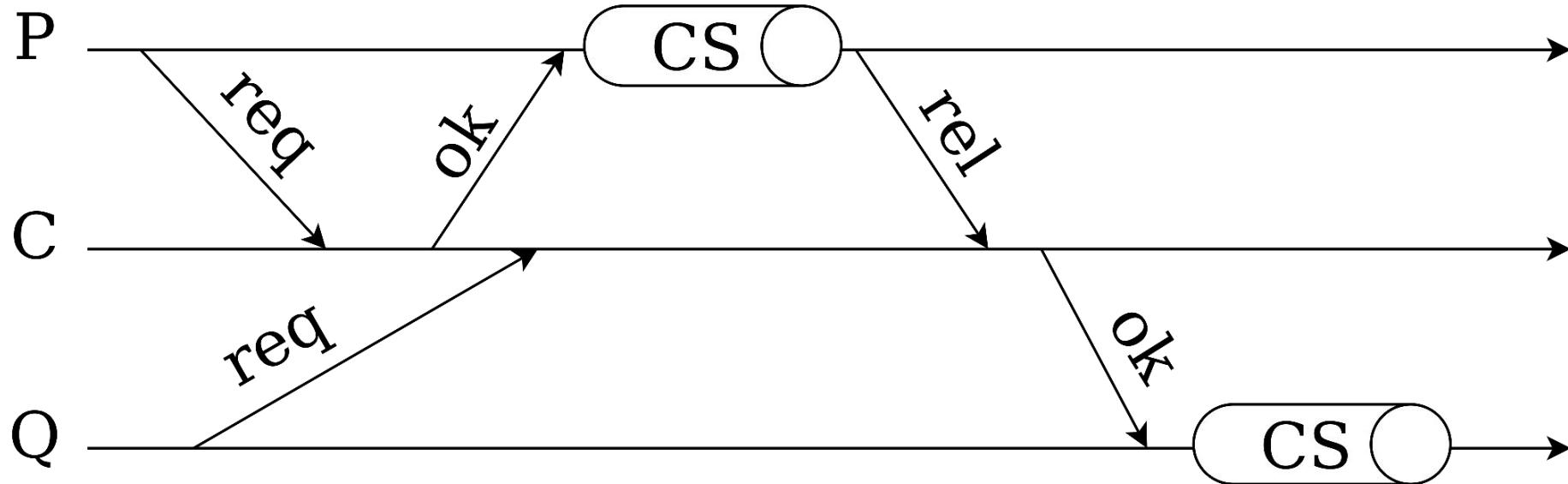
# Централизованный алгоритм

- Процесс заходит в критическую секцию, получив ОК
- Не получив — ждёт



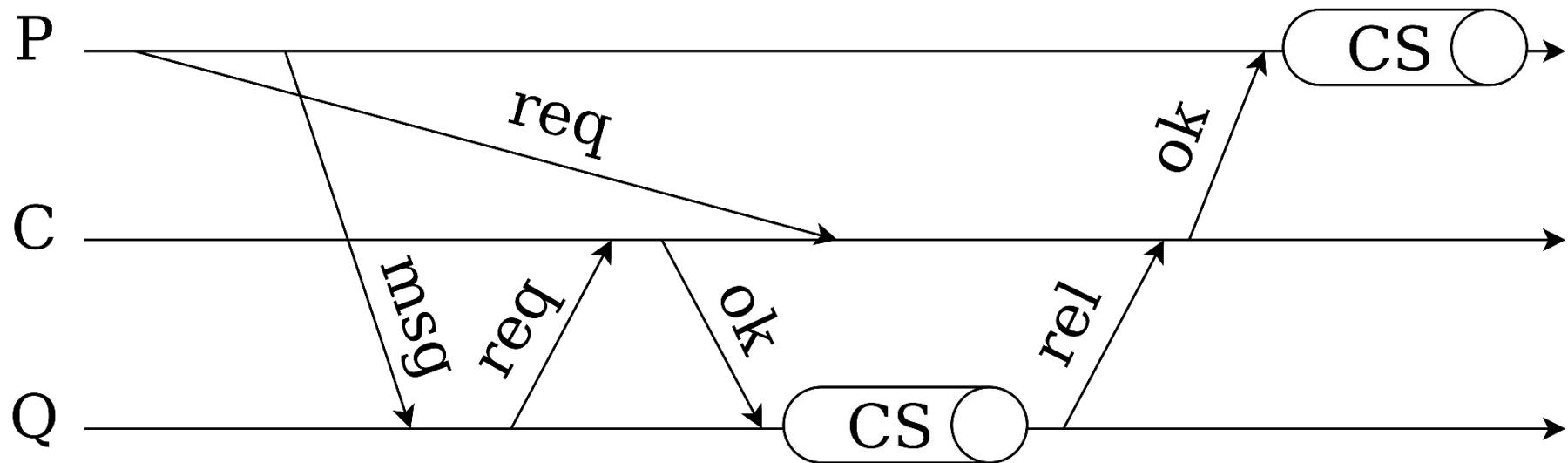
# Централизованный алгоритм

- При выходе из критической секции посылаем сообщение RELEASE координатору
- Получив RELEASE, координатор может выслатить OK следующему ожидающему



# Централизованный алгоритм: честность

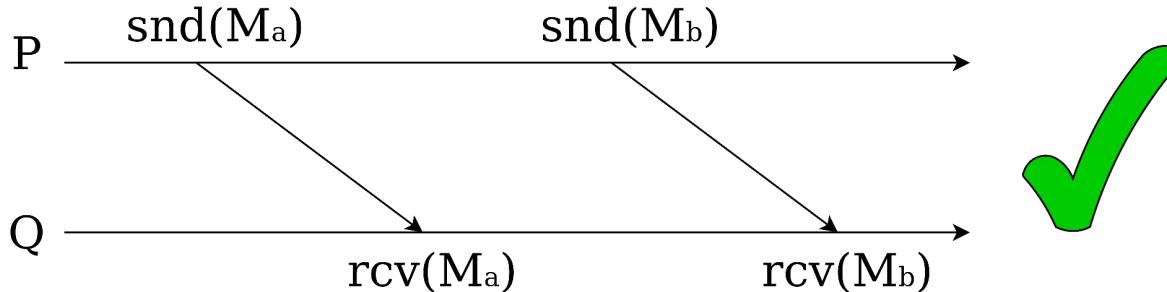
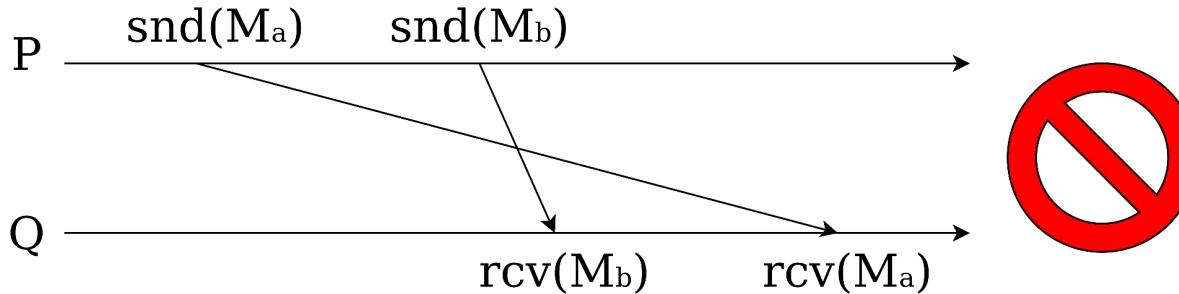
- Из  $\text{Req}_a \rightarrow \text{Req}_b$  следует  $\text{Enter}_a \rightarrow \text{Enter}_b$
- Для централизованного алгоритма не следует
- А ещё централизованные алгоритмы это не очень интересно



# Алгоритм Лампорта: FIFO

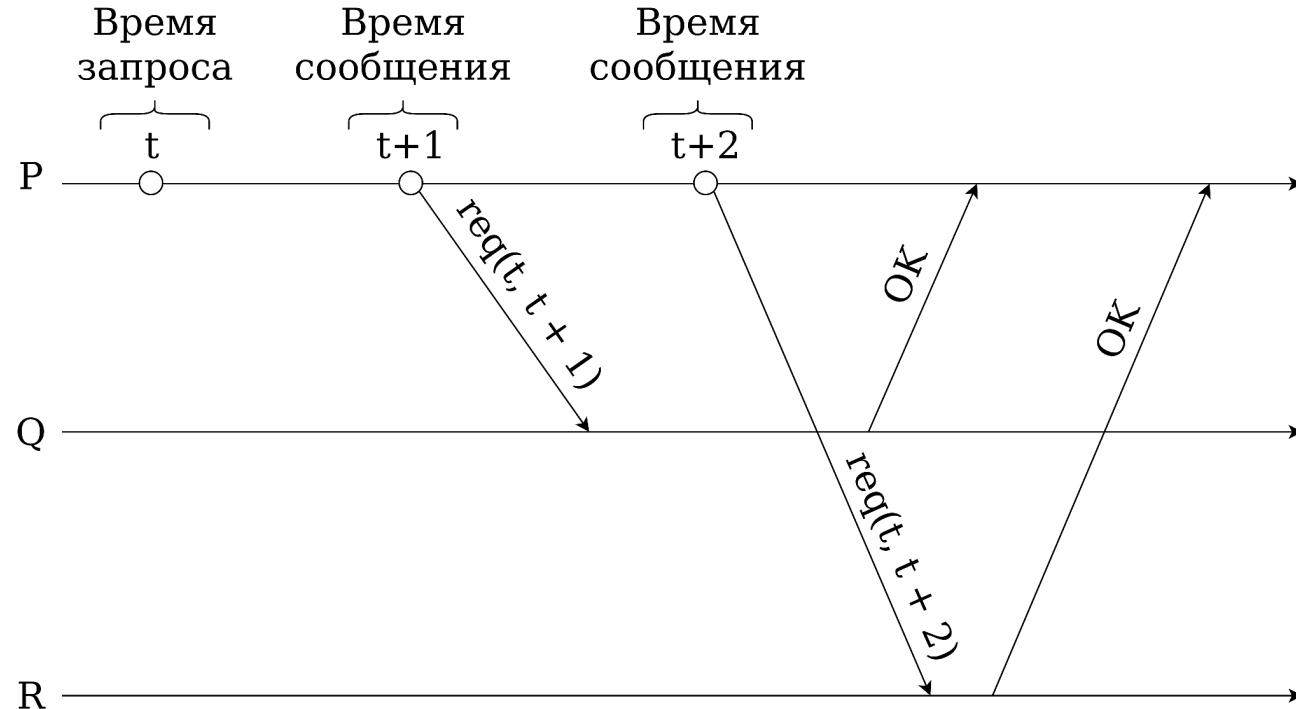
- Порядок доставки сообщений

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



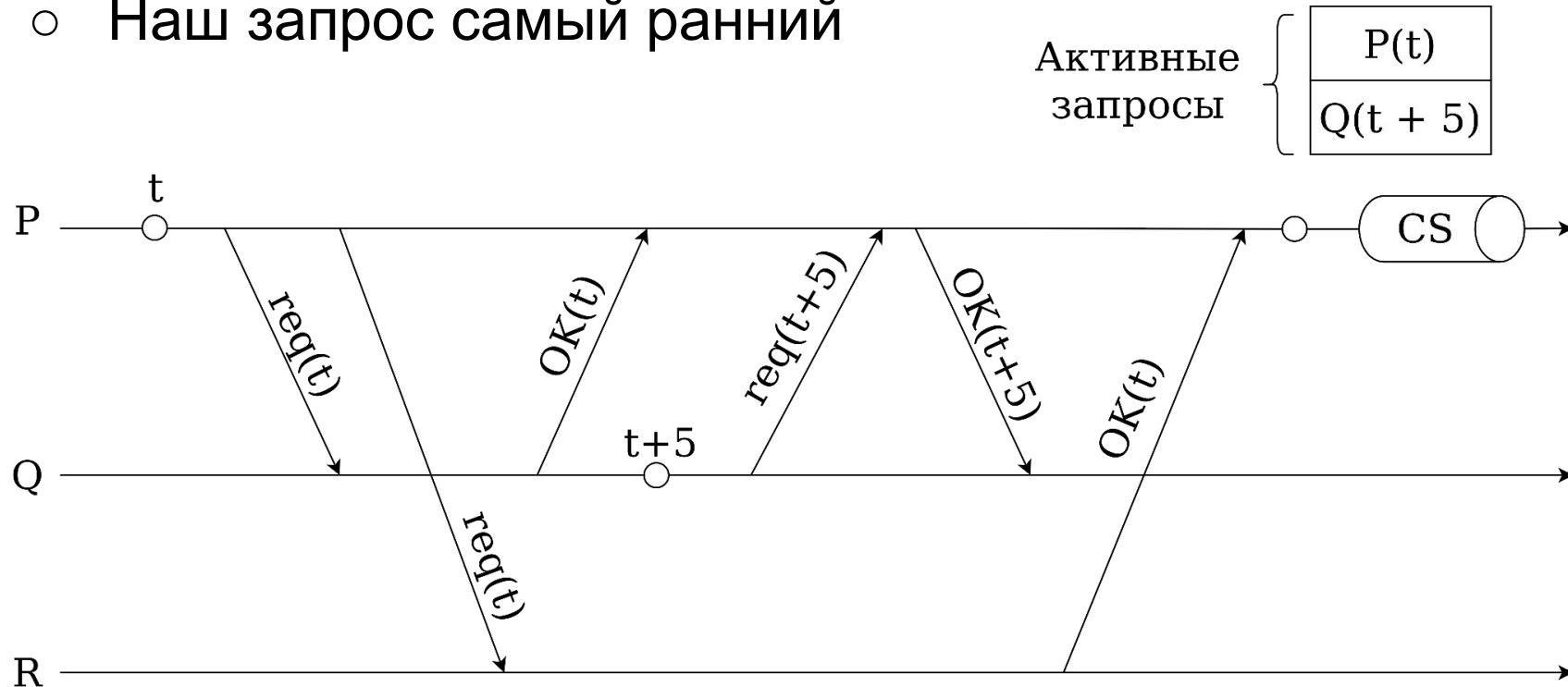
# Алгоритм Лампорта

- Выбираем логическое время запроса и рассылаем всем остальным процессам запрос
- Остальные процессы локально сохраняют наш запрос
- И немедленно отвечают ОК



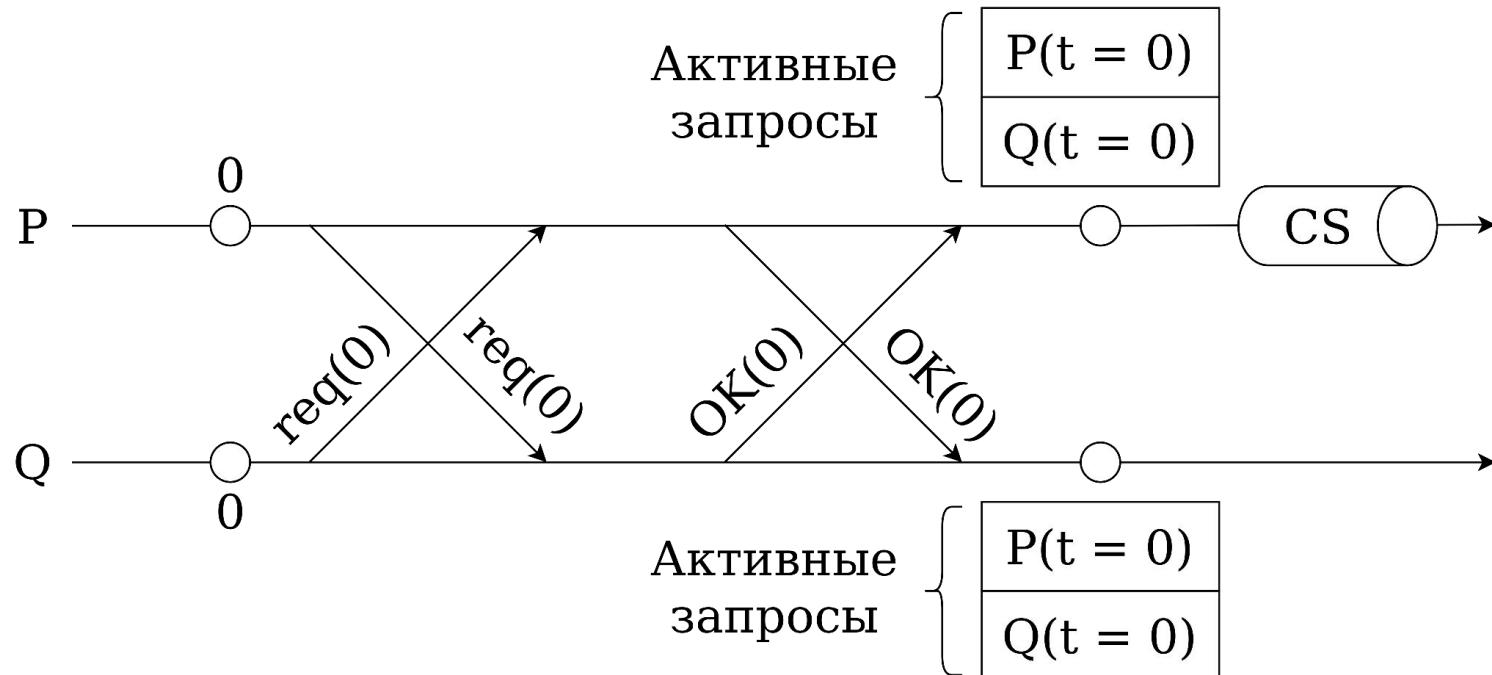
# Алгоритм Лампорта

- Процесс сам решает, когда ему можно входить в КС
  - Получены OK от всех остальных процессов
  - Наш запрос самый ранний



# Алгоритм Лампорта

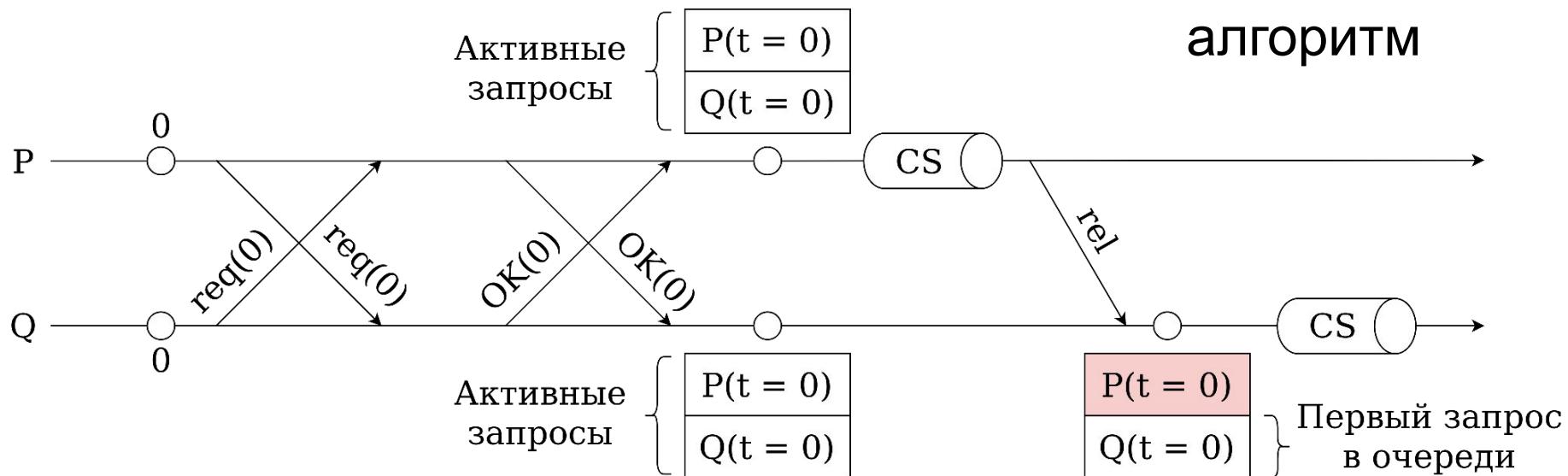
- В случае равенства времени упорядочиваем запросы по идентификатору процесса
  - Считаем что  $\text{id}(P) < \text{id}(Q)$



# Алгоритм Лампорта

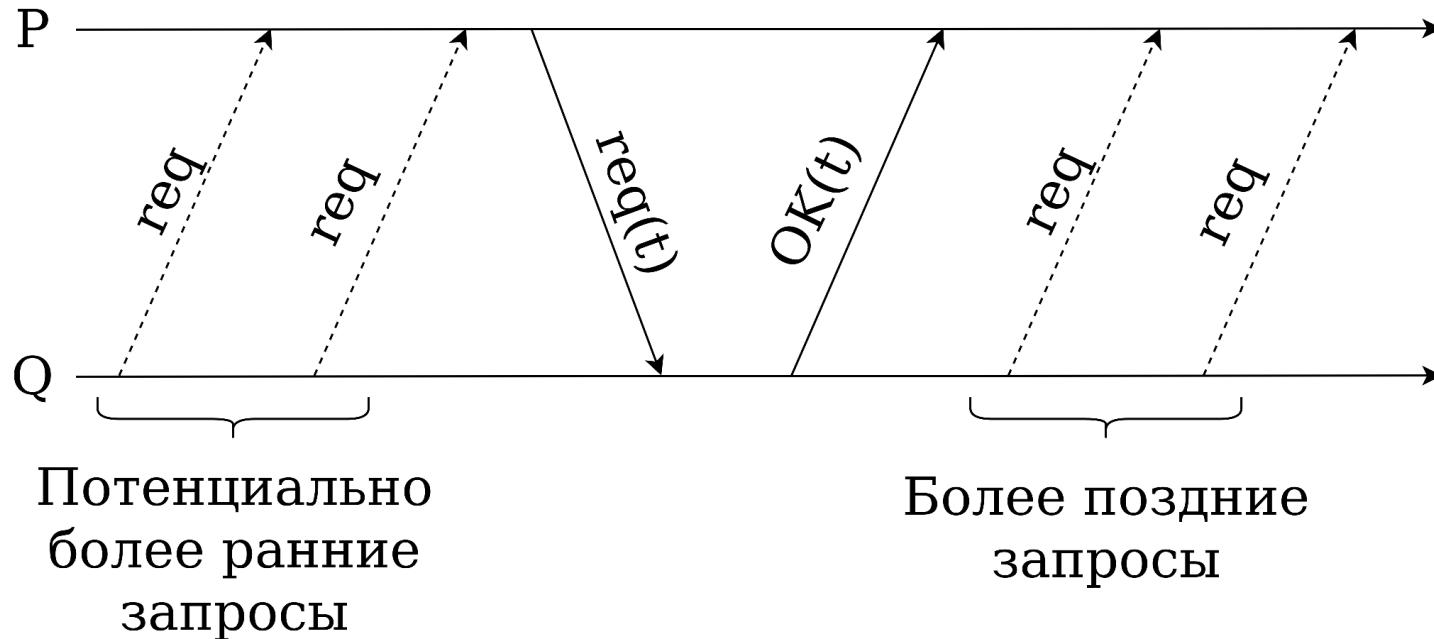
- При выходе из КС рассылаем **всем** процессам RELEASE
- Остальные процессы удаляют наш запрос из очереди
- Один из них теперь может заходить в КС
- $3(n - 3)$  сообщения на КС

- Честный алгоритм



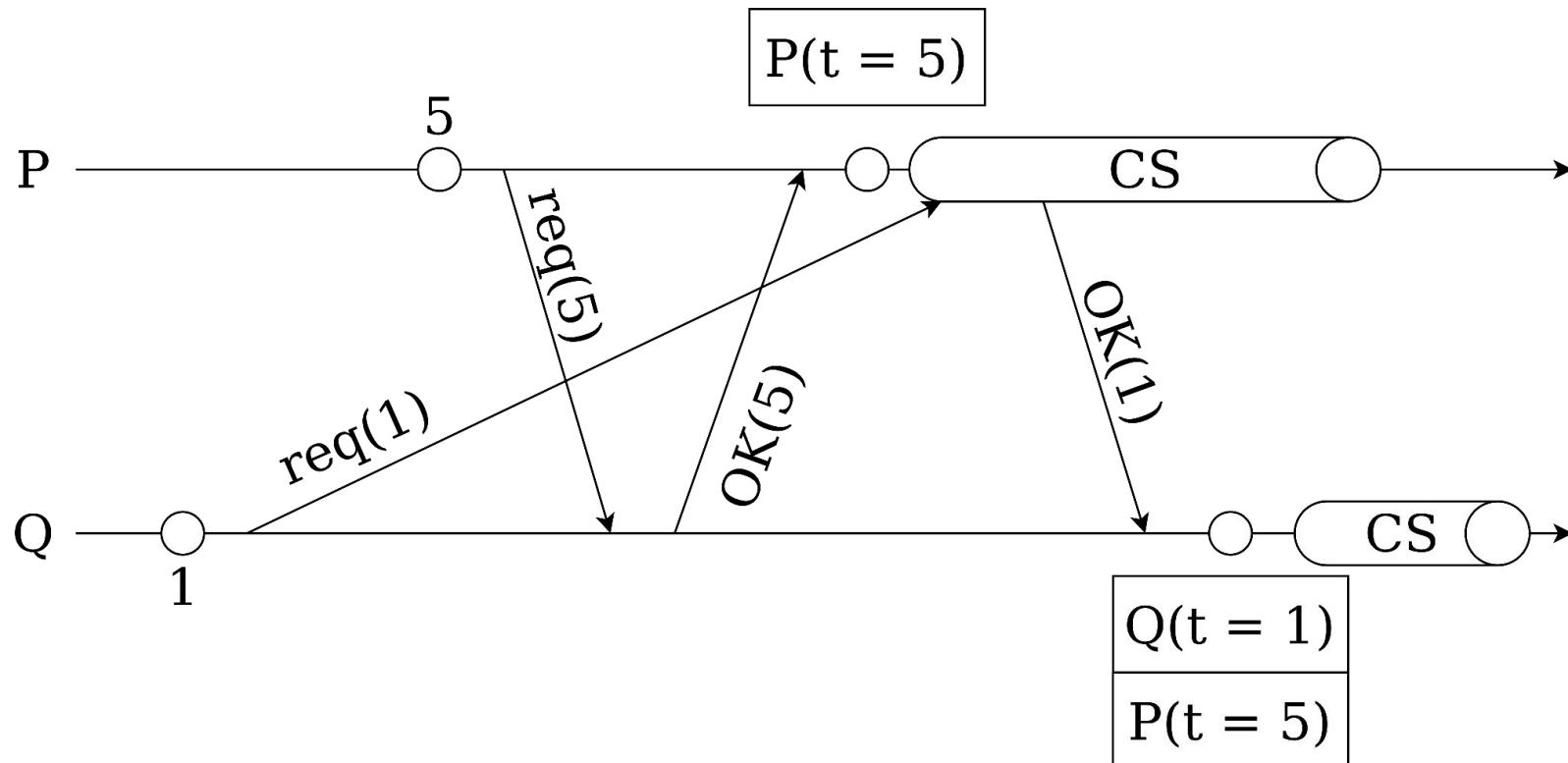
# Алгоритм Лампорта

- К моменту получения ОК мы получили от процесса все запросы с временем меньше нашего
- У следующих запросов время будет больше



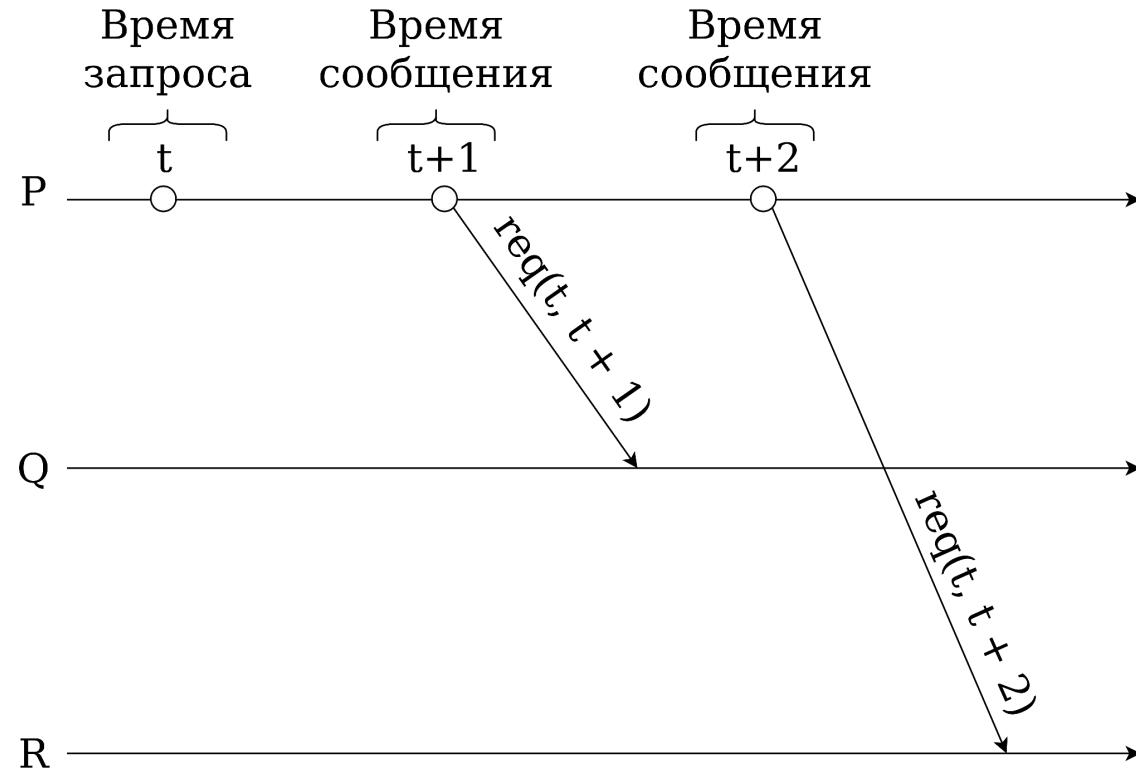
# Алгоритм Лампорта

- Алгоритм ломается при отсутствии FIFO



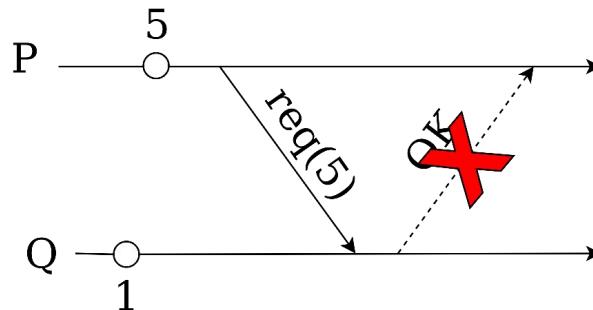
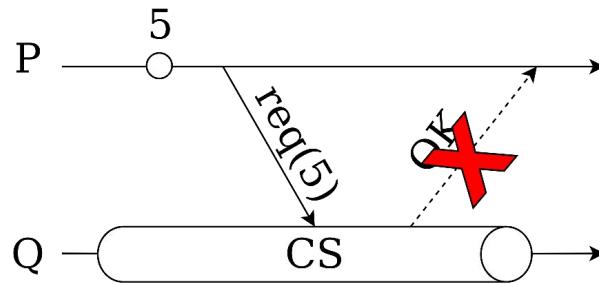
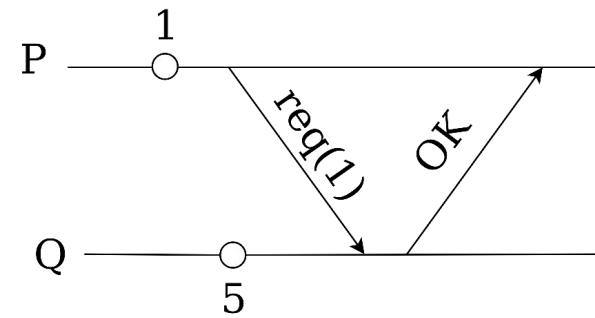
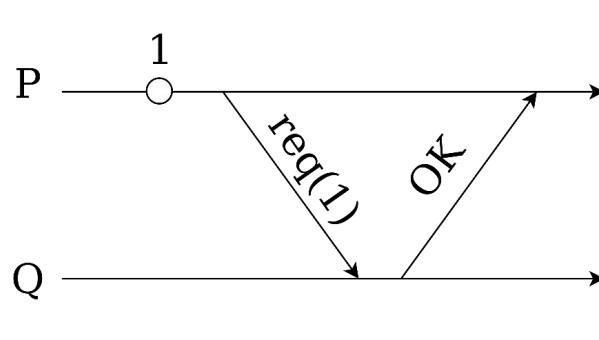
# Алгоритм Рикарта и Агравала

- Оптимизация алгоритма Лампорта
- Начало такое же: рассылаем всем процессам запрос с временем
- OK посыпается по другим правилам



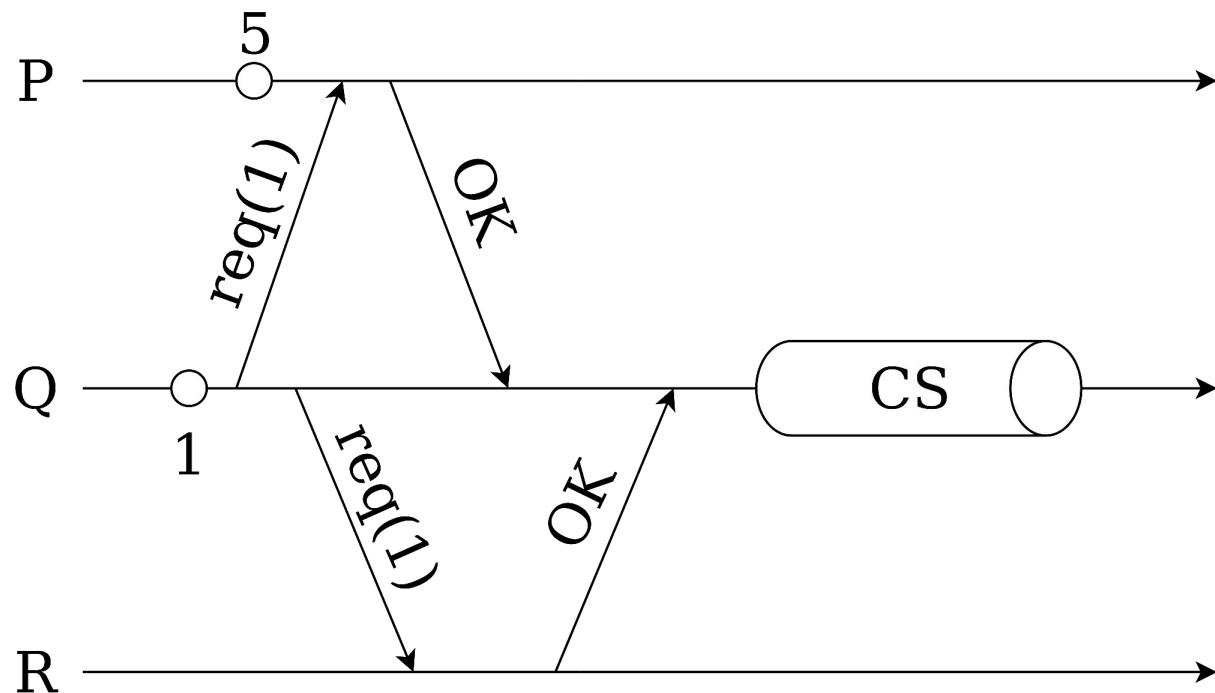
# Алгоритм Рикарта и Агравала

- Отвечаем OK только если сами не хотим в КС
  - Или если у пришедшего запроса меньше время



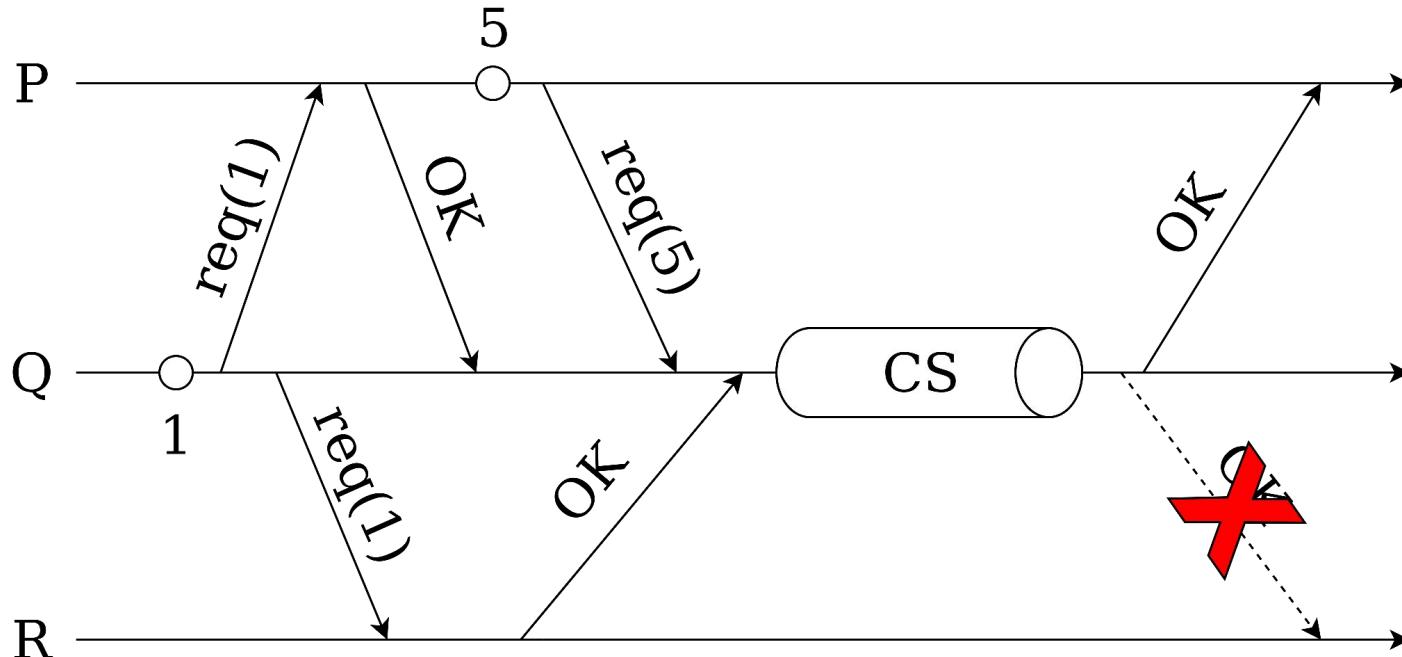
# Алгоритм Рикарта и Агравала

- Заходим в КС когда собрали все ОК
- Если процесс  
ответил нам ОК,  
значит мы  
можем зайти в  
КС раньше него
- Не нужно FIFO



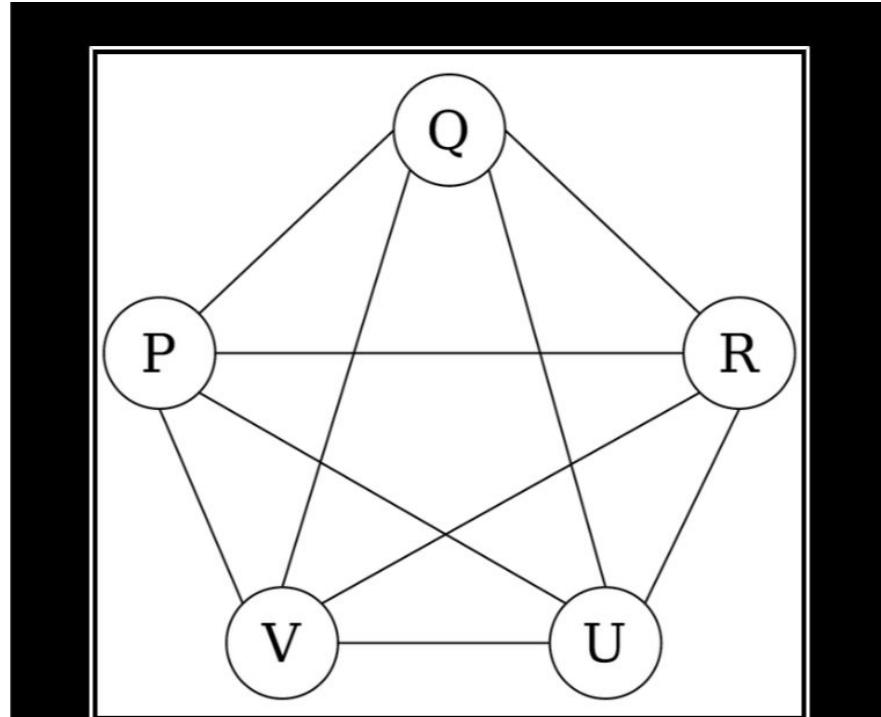
# Алгоритм Рикарта и Агравала

- После выхода из КС отвечаем OK тем, кому не ответили сразу
- Больше никому OK не посылаем
- $2(n - 1)$  сообщений на вход в КС



# Алгоритм обедающих философов

- Представим задачу как полный **ориентированный** граф
- Вершины — процессы
- Между каждой парой процессов ( $P, Q$ ) есть либо ребро  $P \rightarrow Q$ , либо ребро  $Q \rightarrow P$
- Но не оба сразу

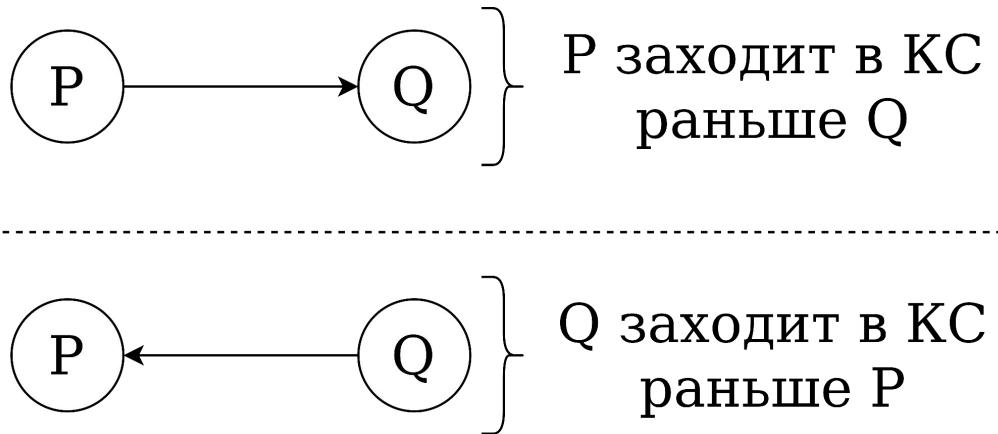
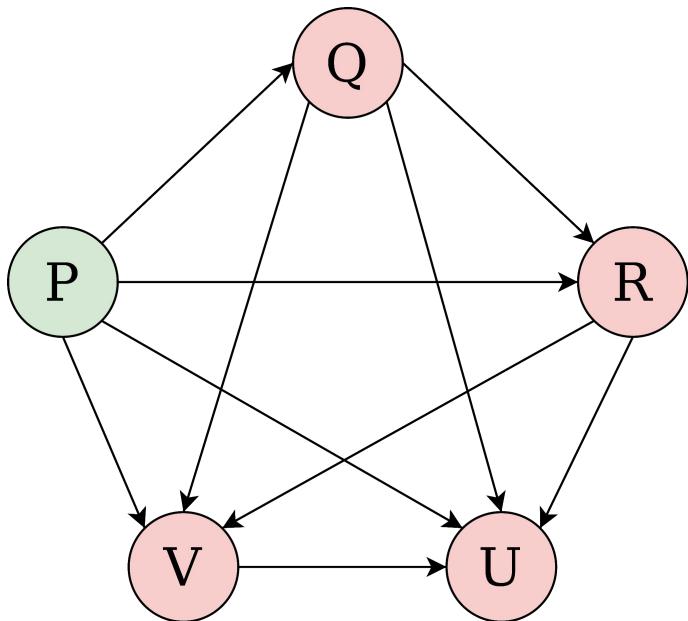


Полный граф

Нет блин худой барон

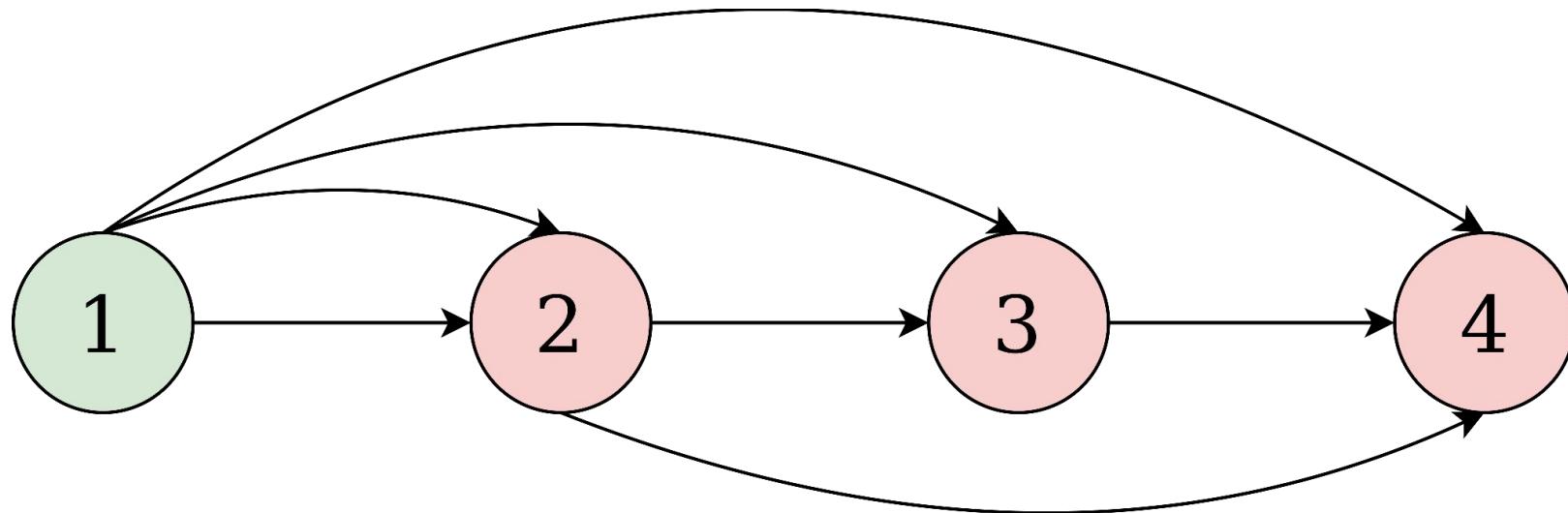
# Алгоритм обедающих философов

- Направление рёбер определяет порядок входа в КС
- Значит, войти в КС может только исток графа



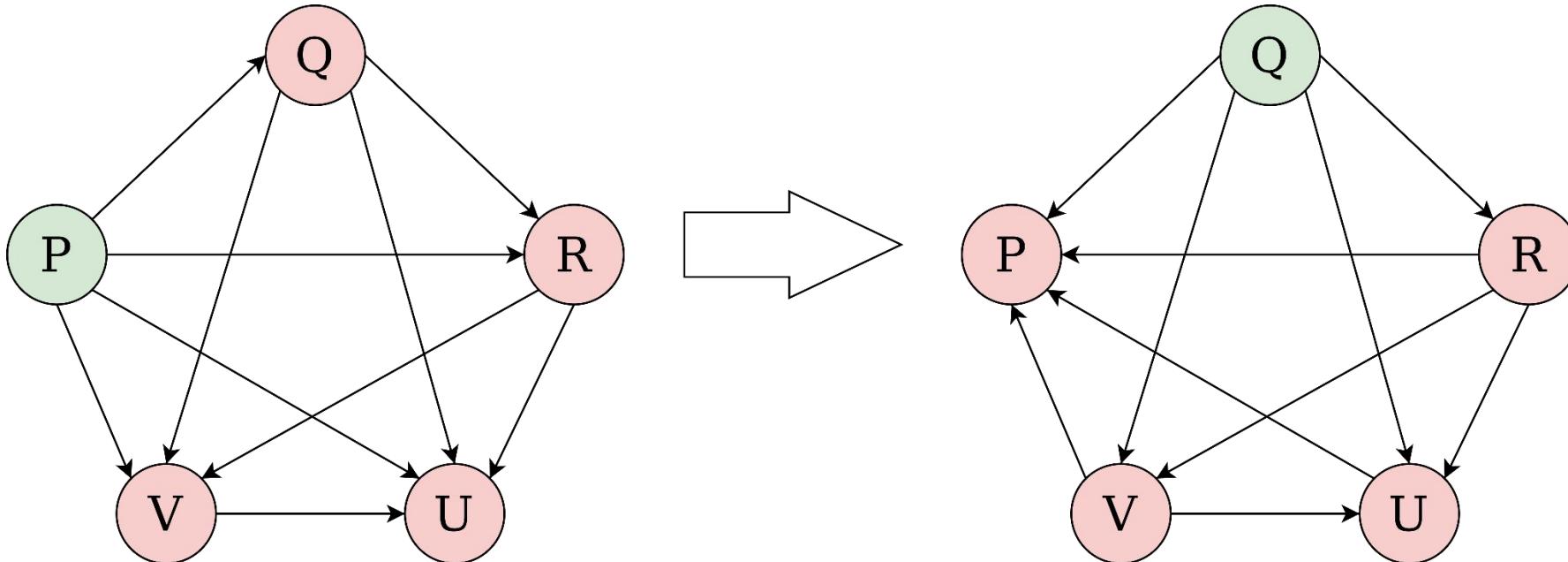
# Алгоритм обедающих философов

- В ациклическом графе всегда есть исток
- Изначально проведём рёбра так, чтобы граф был ациклическим
  - Например, от процесса с меньшим  $pid$  к процессу с большим  $pid$



# Алгоритм обедающих философов

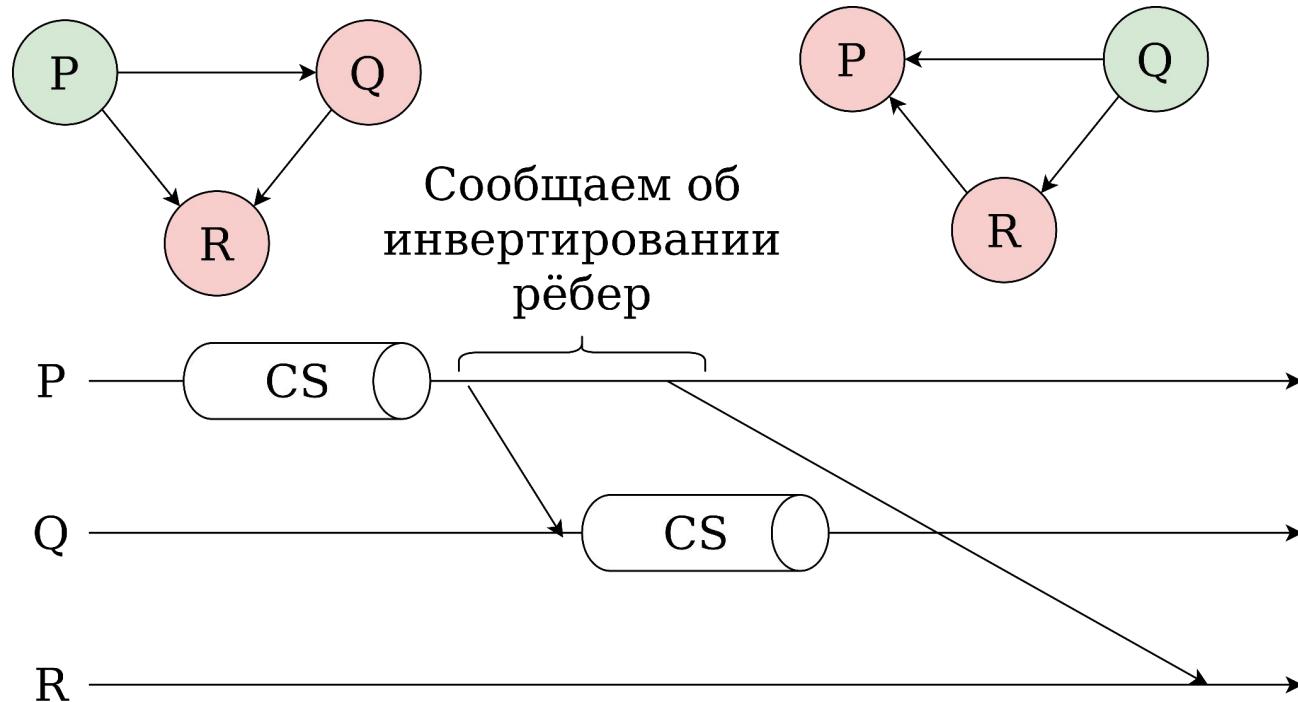
- Если в ациклическом ориентированном графе инвертировать все рёбра у истока, граф останется ациклическим



# Алгоритм обедающих философов

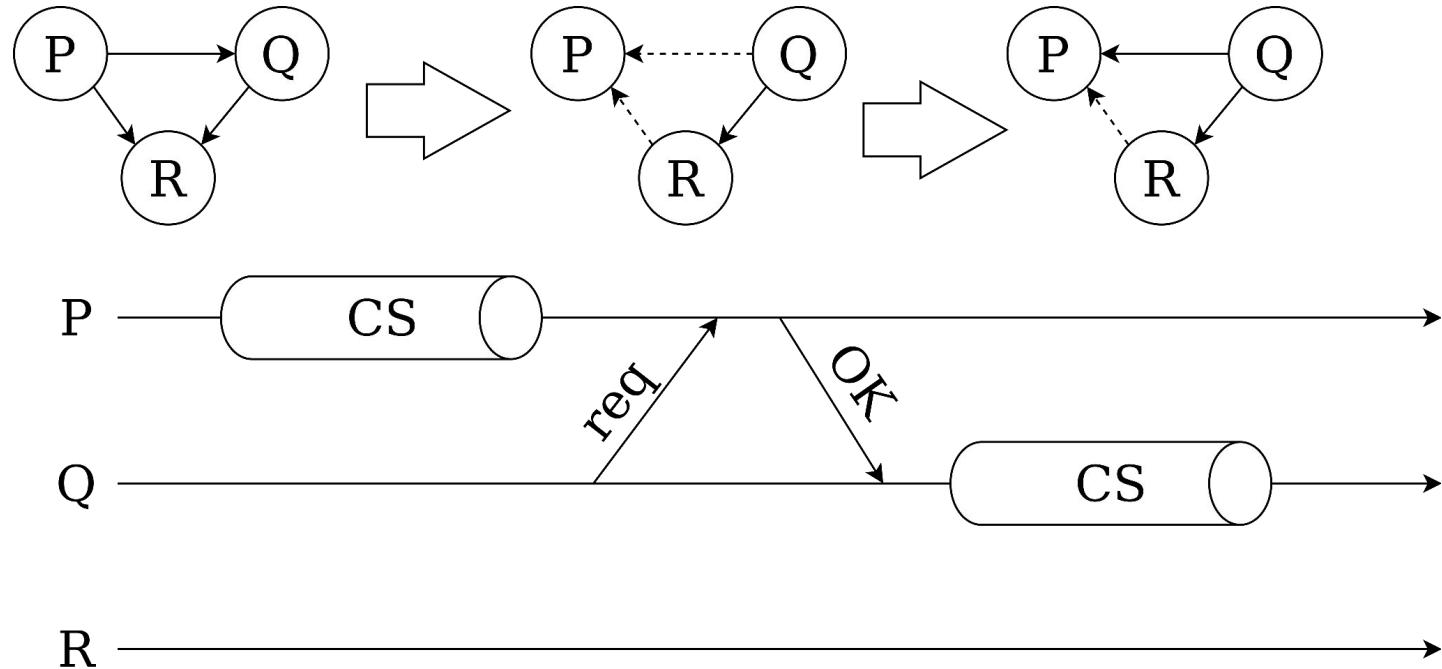
- После выхода из критической секции источник рассыпает всем сообщение об инвертировании рёбер

- Можно ли эффективней?



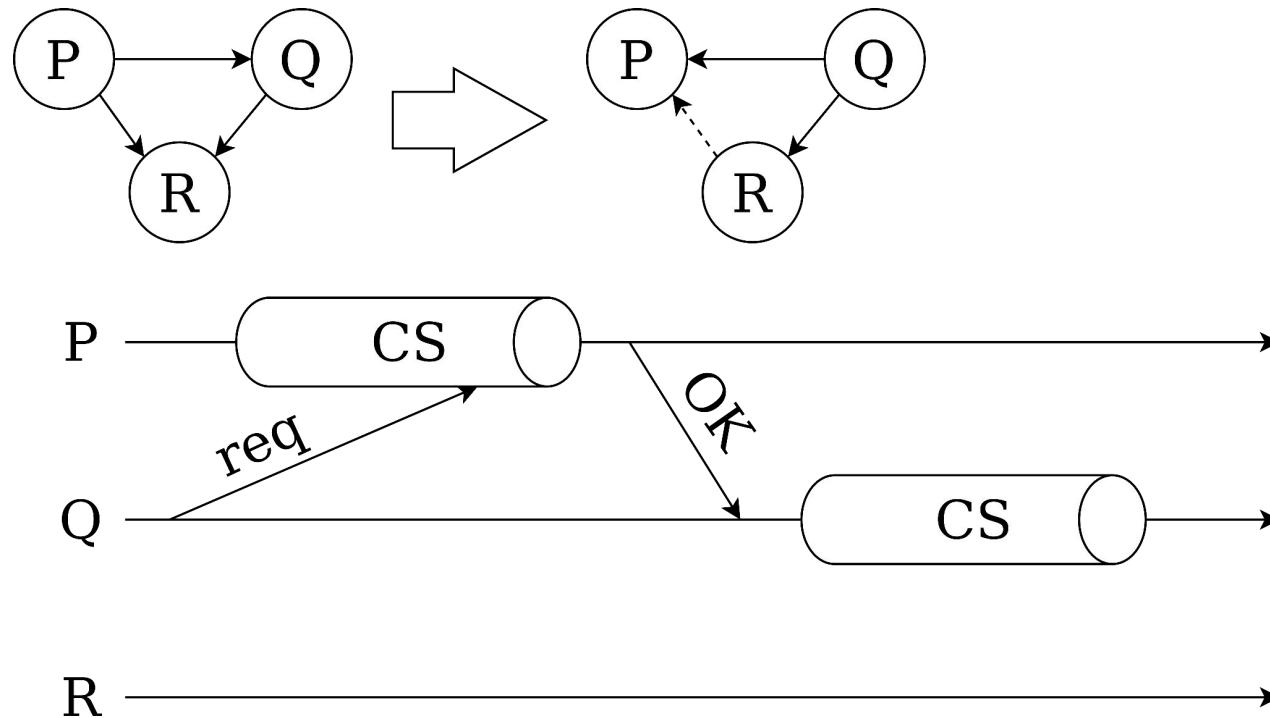
# Алгоритм обедающих философов

- После выхода из КС логически инвертируем рёбра
- Но не сообщаем об этом другим процессам
- “Отдаём”  
рёбра по  
запросу



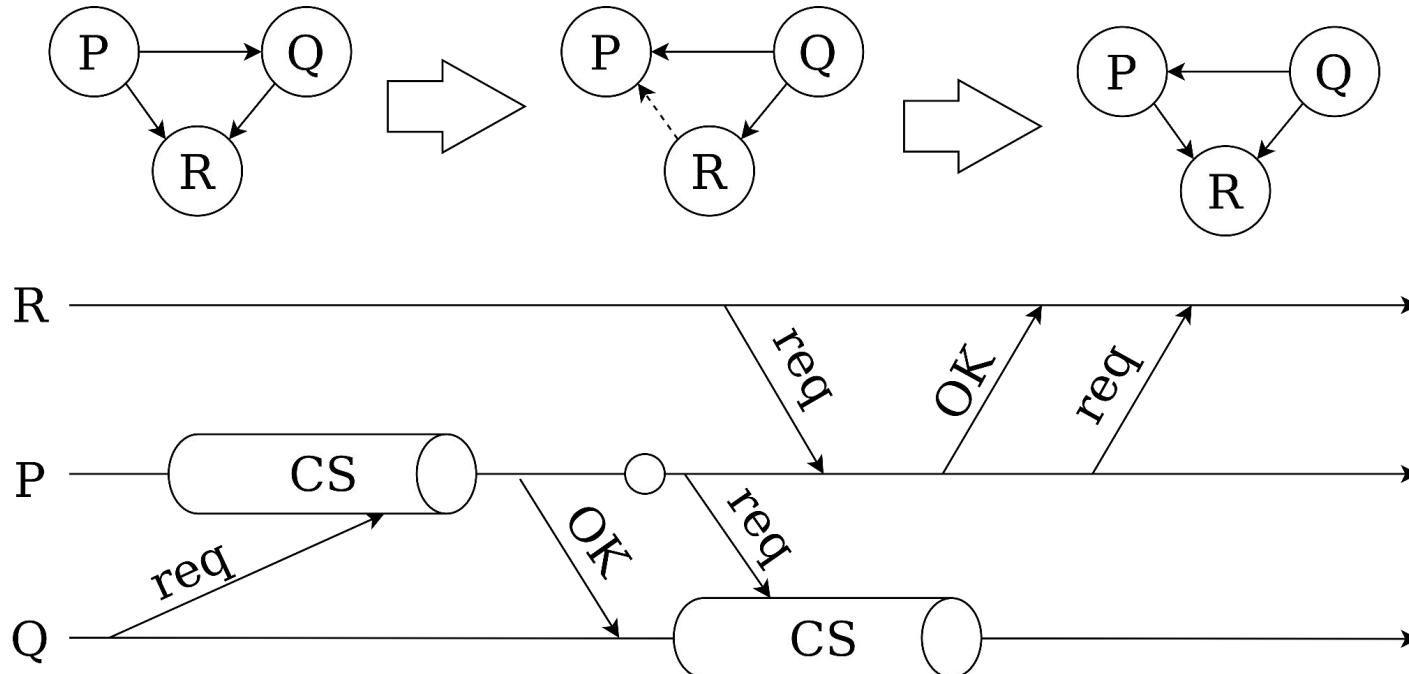
# Алгоритм обедающих философов

- Если запрос пришёл пока мы были в КС, то отдаём ребро сразу после выхода



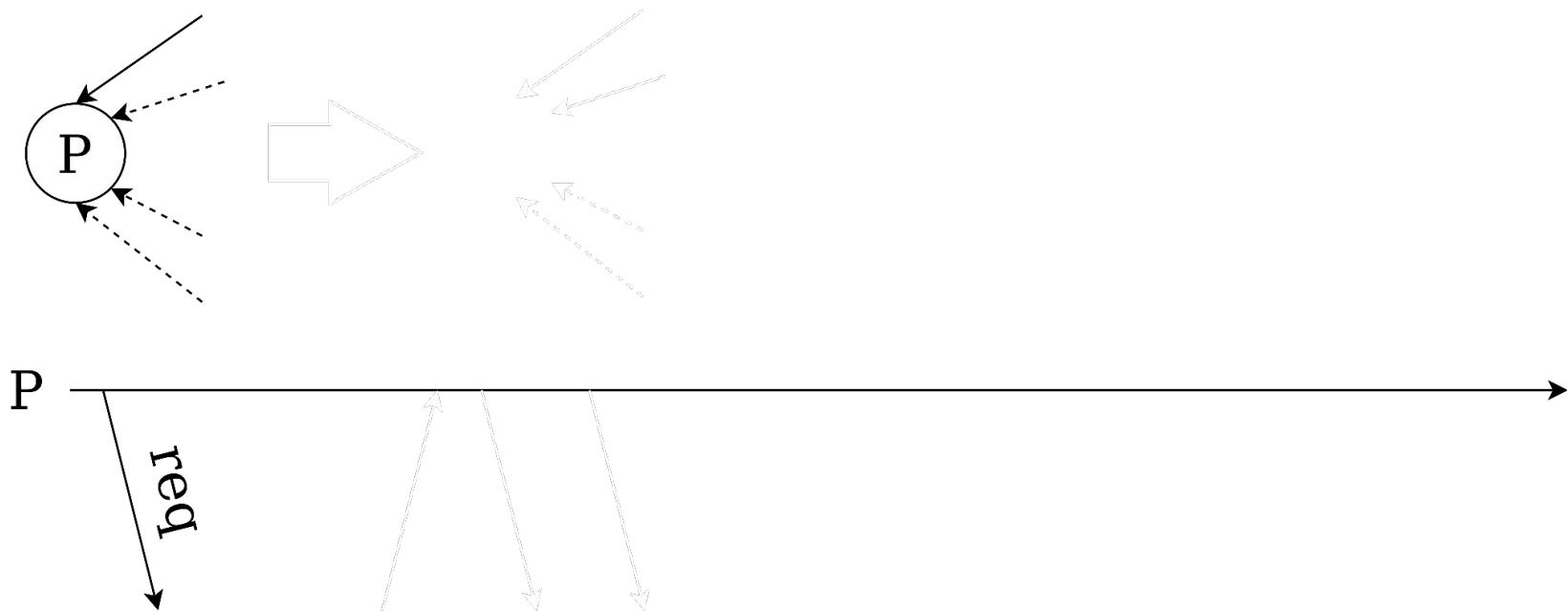
# Алгоритм обедающих философов

- Передаём логически инвертированные рёбра по запросу, даже если сами хотим войти в КС
- И тут же запрашиваем их обратно



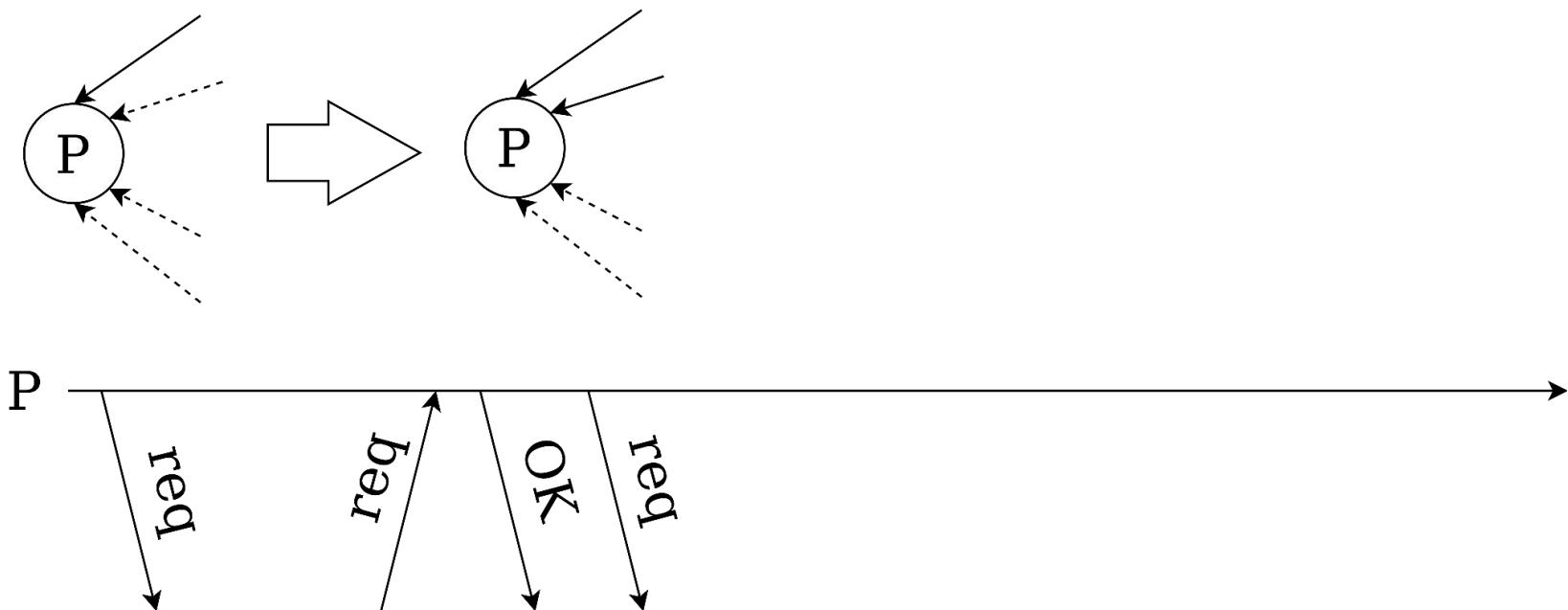
# Алгоритм обедающих философов

- Запрашиваем все рёбра, которыми владеют другие процессы



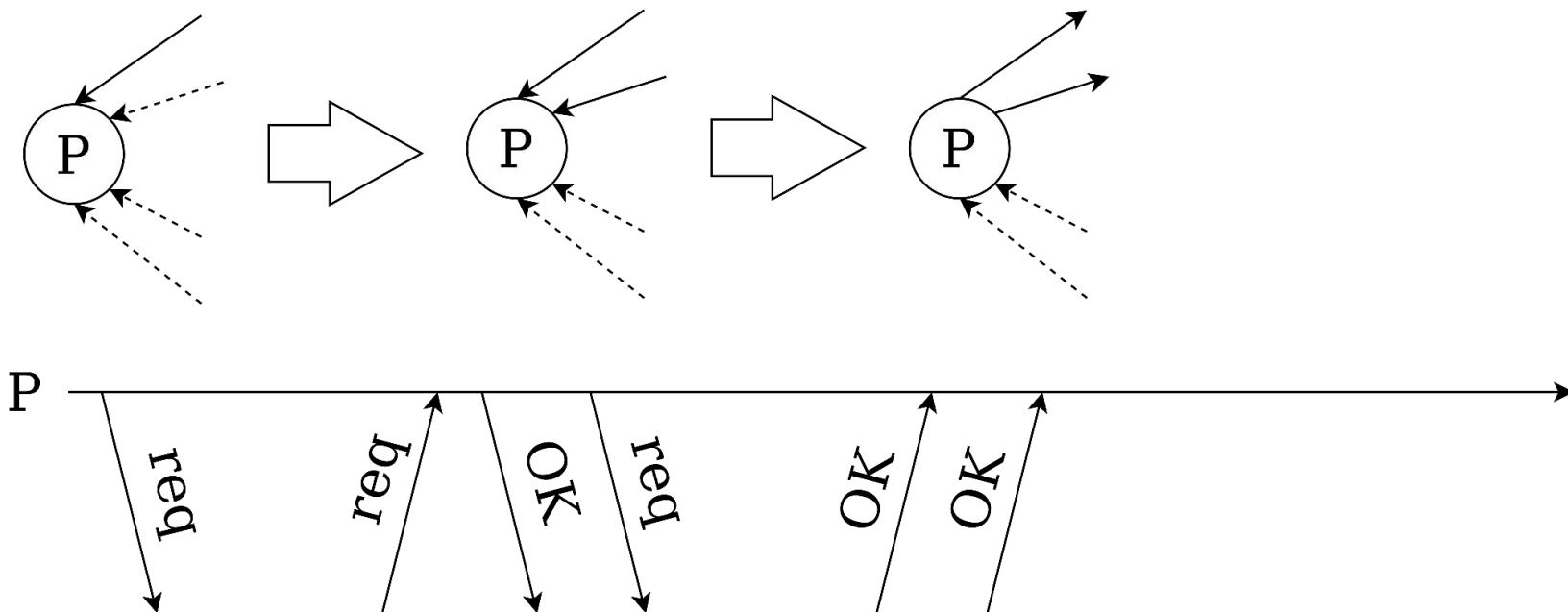
# Алгоритм обедающих философов

- Если в процессе ожидания ответов нам приходит запрос на логически инвертированное ребро — отдаём его
  - И тут же запрашиваем обратно



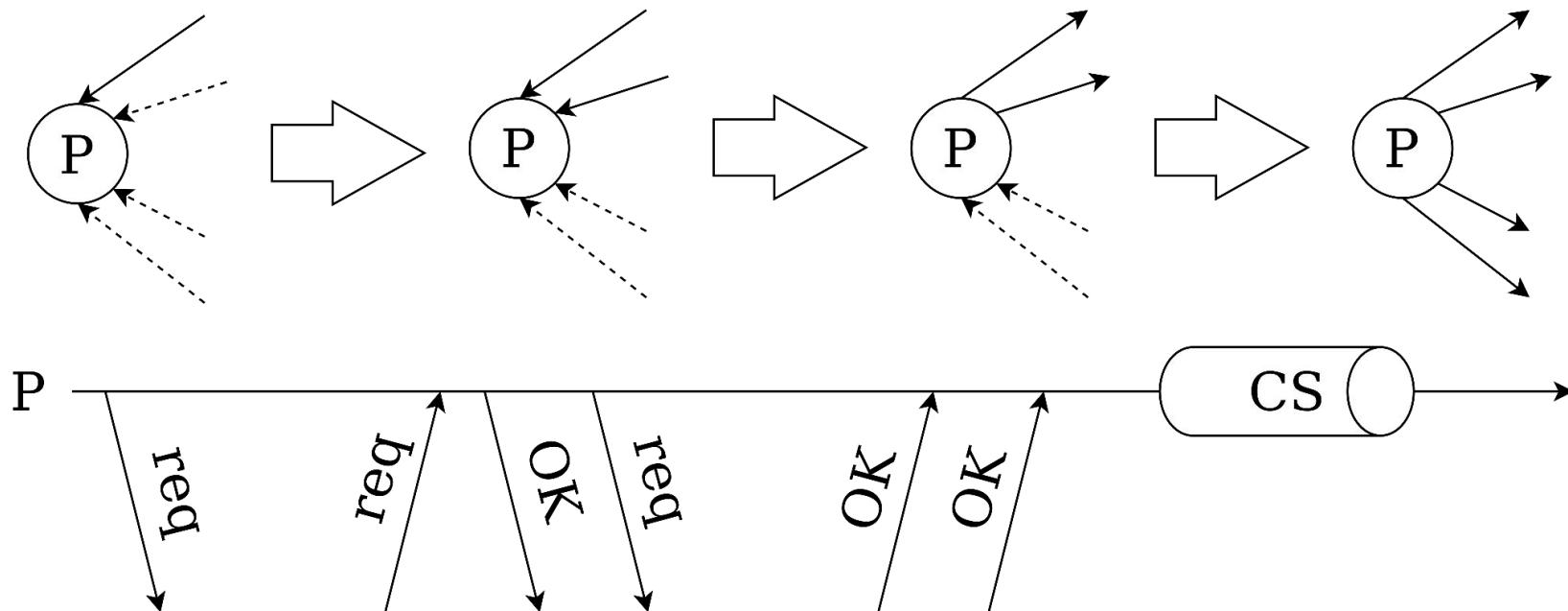
# Алгоритм обедающих философов

- Дожидаемся ответов на все запросы



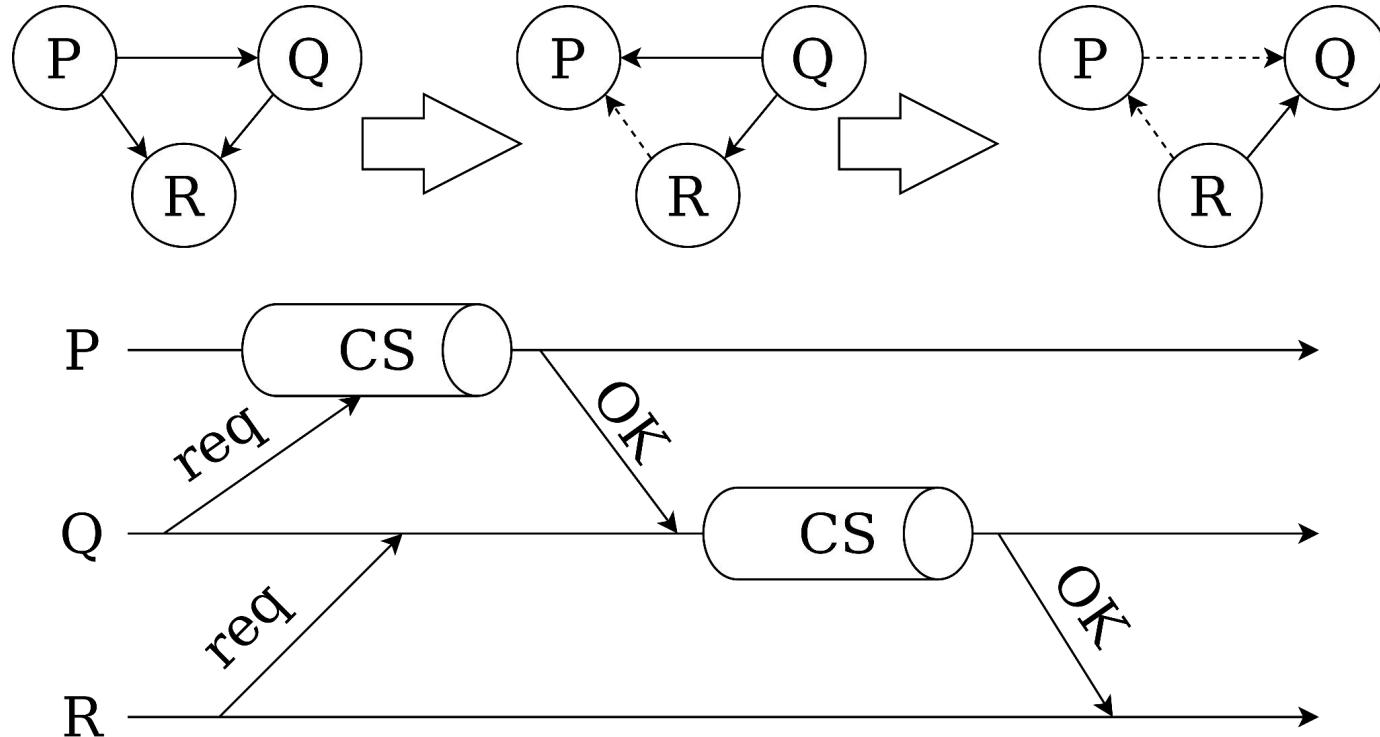
# Алгоритм обедающих философов

- Возвращаем все логически инвертированные рёбра и входим в КС
- Не более  $2(n-1)$  сообщения на вход в КС



# Алгоритм обедающих философов

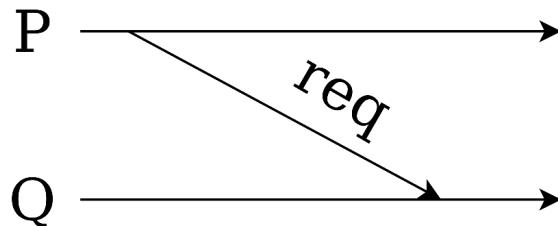
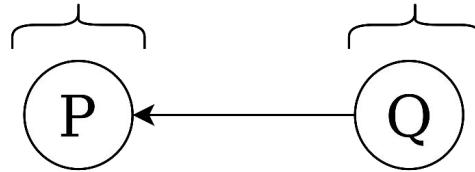
- Если у нас запрашивают ребро, которым мы владеем — не отдаём
- Должны войти в КС раньше этого процессса



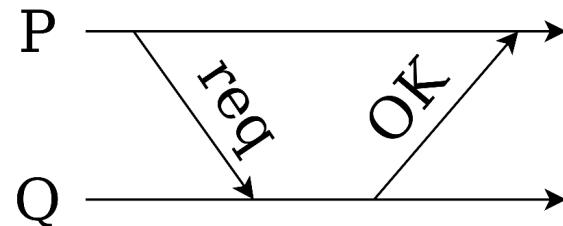
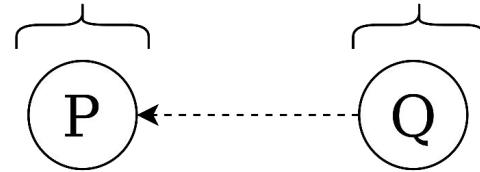
# Алгоритм обедающих философов

- В начале алгоритма можно считать все рёбра логически инвертированными
- Иначе может не быть прогресса
- Чтобы не писать лишние условия

Хочет войти в КС      Не хочет

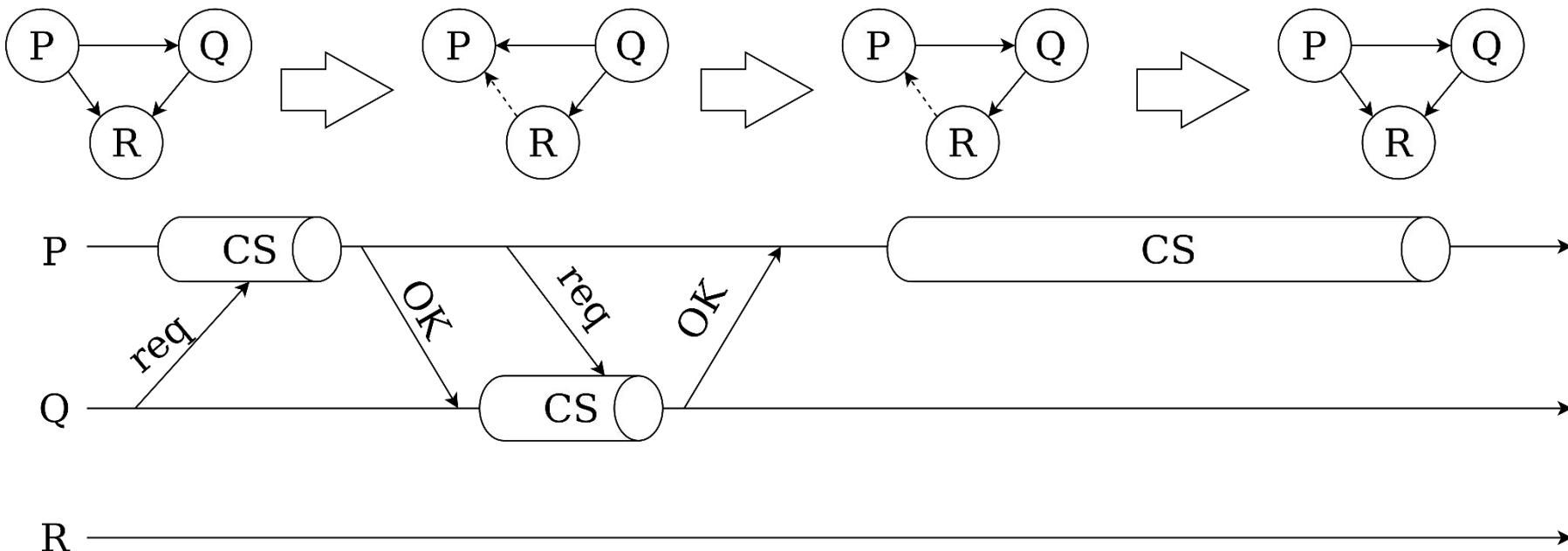


Хочет войти в КС      Не хочет



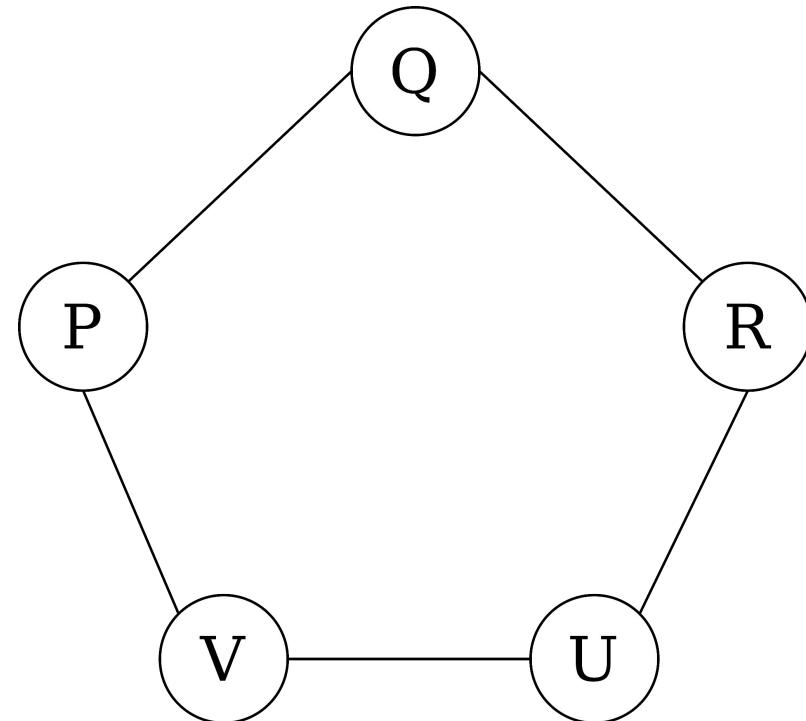
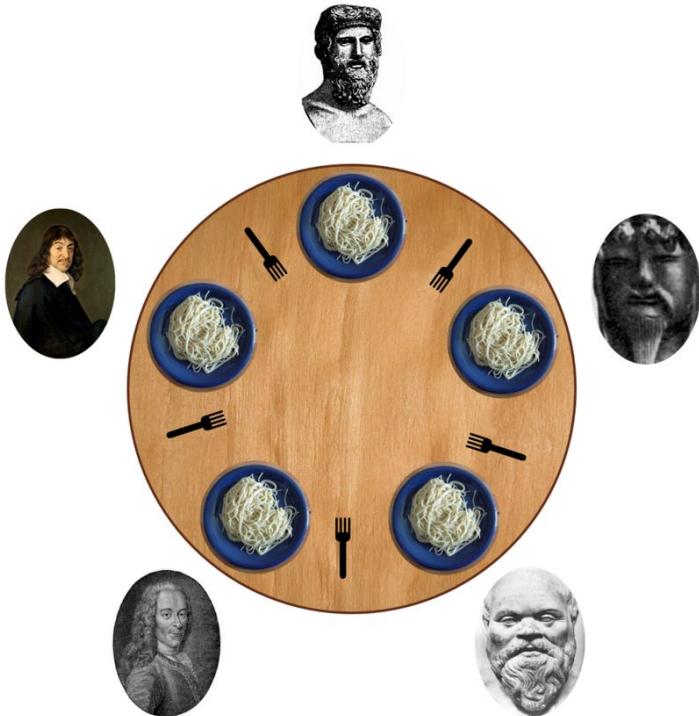
# Алгоритм обедающих философов

- Сообщения передаются только между процессами, заходящими в КС
- Остальные не участвуют в обмене сообщениями



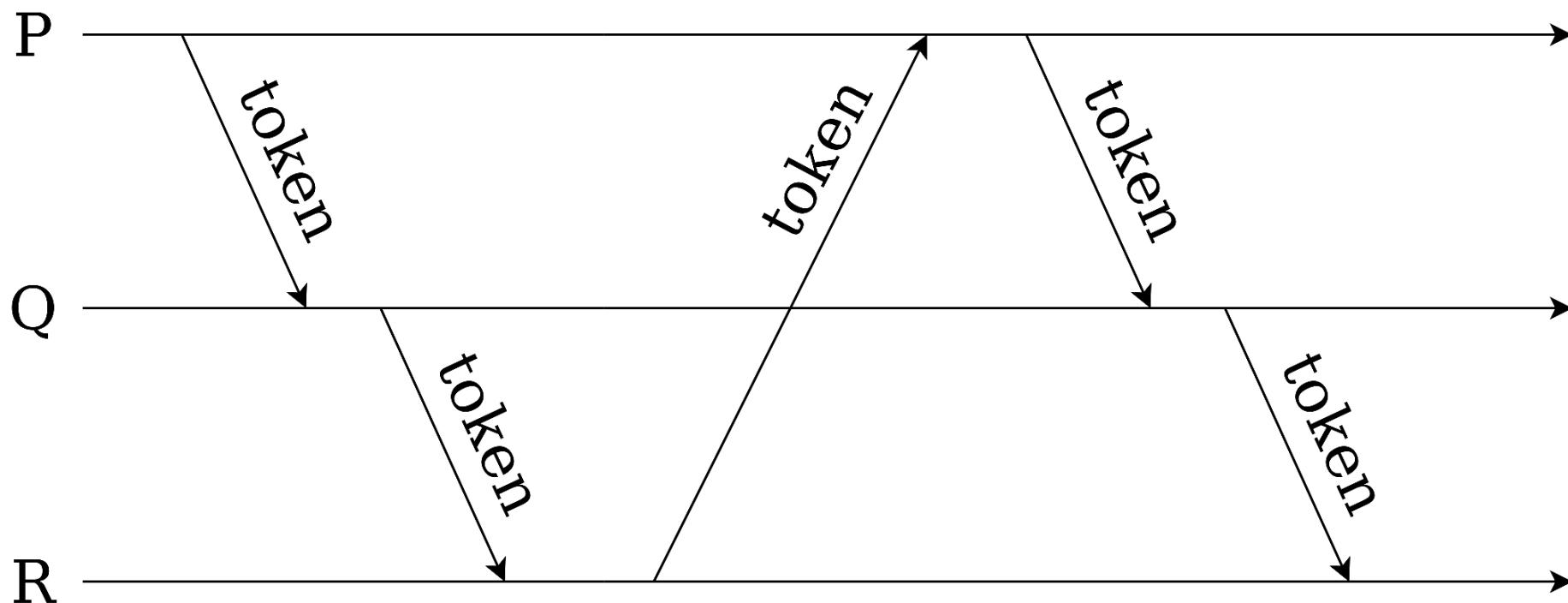
# Алгоритм обедающих философов

- А при чём тут вообще философы?
- Алгоритм обобщается на произвольный граф конфликтов



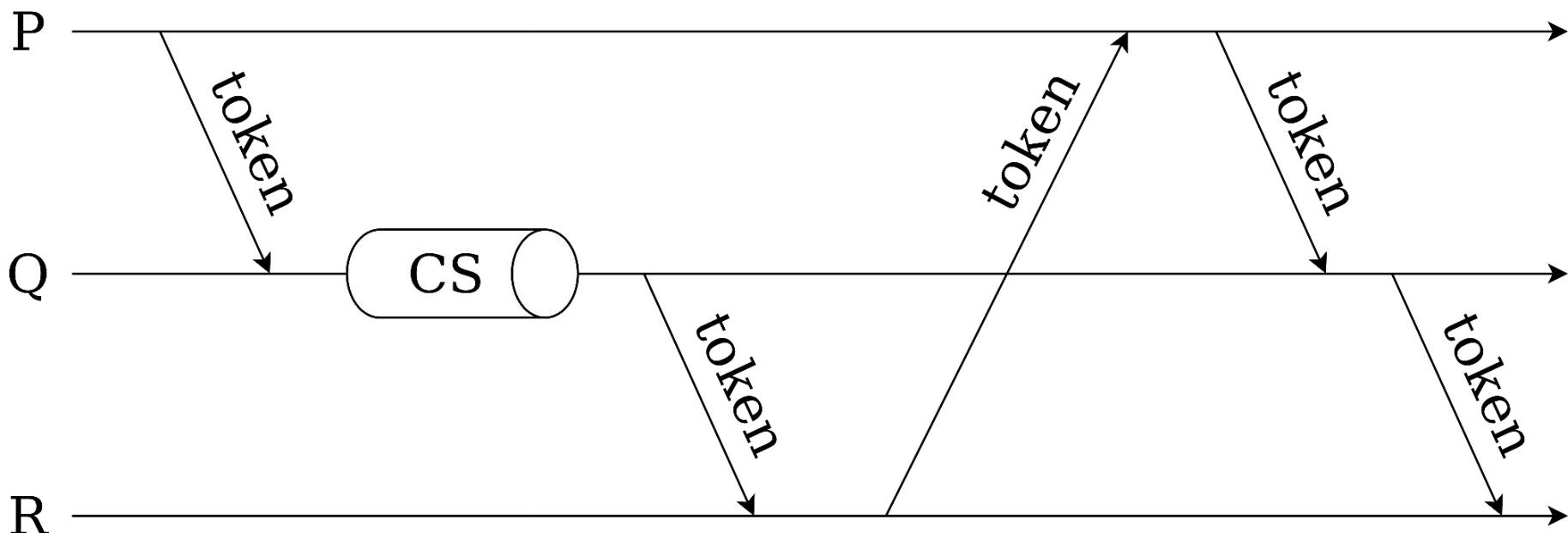
# Алгоритм на основе токена

- Процессы передают по кругу токен
  - Токен — пустое сообщение



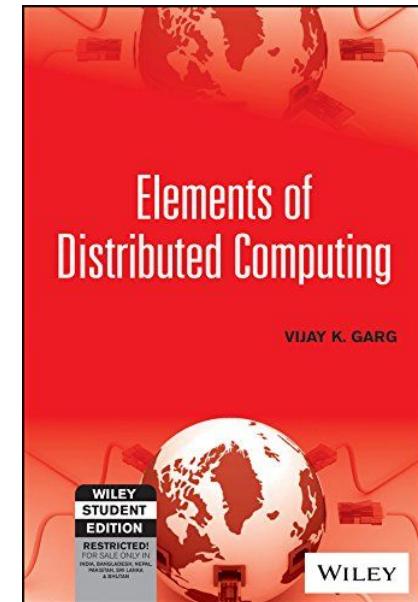
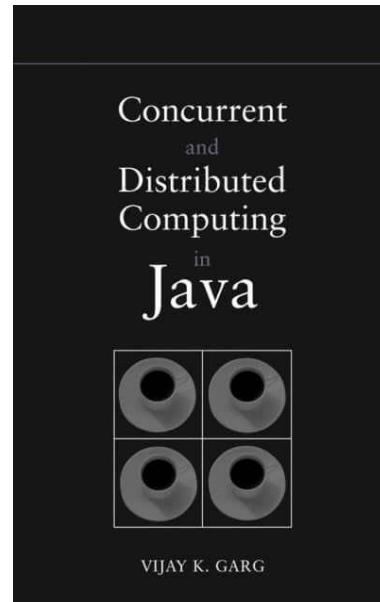
# Алгоритм на основе токена

- Входим в КС если токен у нас
  - Пересылаем токен по кругу при выходе из КС
- Идея схожа с [Token Ring сетями](#) (IEEE 802.5)



# Что почитать: Распределённые системы в целом

- Garg V. K. Concurrent and distributed computing in Java
- Garg V. K. Elements of distributed computing
- [jepsen.io/analyses](http://jepsen.io/analyses)



## Что почитать & посмотреть: часы и взаимное исключение

- *Lamport L.* Time, clocks, and the ordering of events in a distributed system
- *Liskov B., Ladin R.* Highly available distributed services and fault-tolerant distributed garbage collection
- *Ricart G., Agrawala A. K.* An optimal algorithm for mutual exclusion in computer networks
- *Chandy K. M., Misra J.* The drinking philosophers problem
- *Strole N. C.* The IBM token-ring network—A functional overview
- *Martin Kleppmann. Distributed Systems: Physical time*

# Thanks for your attention

