Программирование распределенных систем

Роман Елизаров, 2022

elizarov@gmail.com

Лекция 2

ГЛОБАЛЬНОЕ СОСТОЯНИЕ, СОГЛАСОВАННЫЕ СРЕЗЫ И ПРЕДИКАТЫ

Согласованное глобальное состояние (срез)

- Если у распределенной системы нет «глобального состояния», то как запомнить её состояние на диске, чтобы можно было продолжить работу после восстановления с диска?
- DEF: Любое $G \subset E$ называется **срезом** тогда и только тогда, когда

$$\forall e \in E, f \in G : e < f \Rightarrow e \in G$$

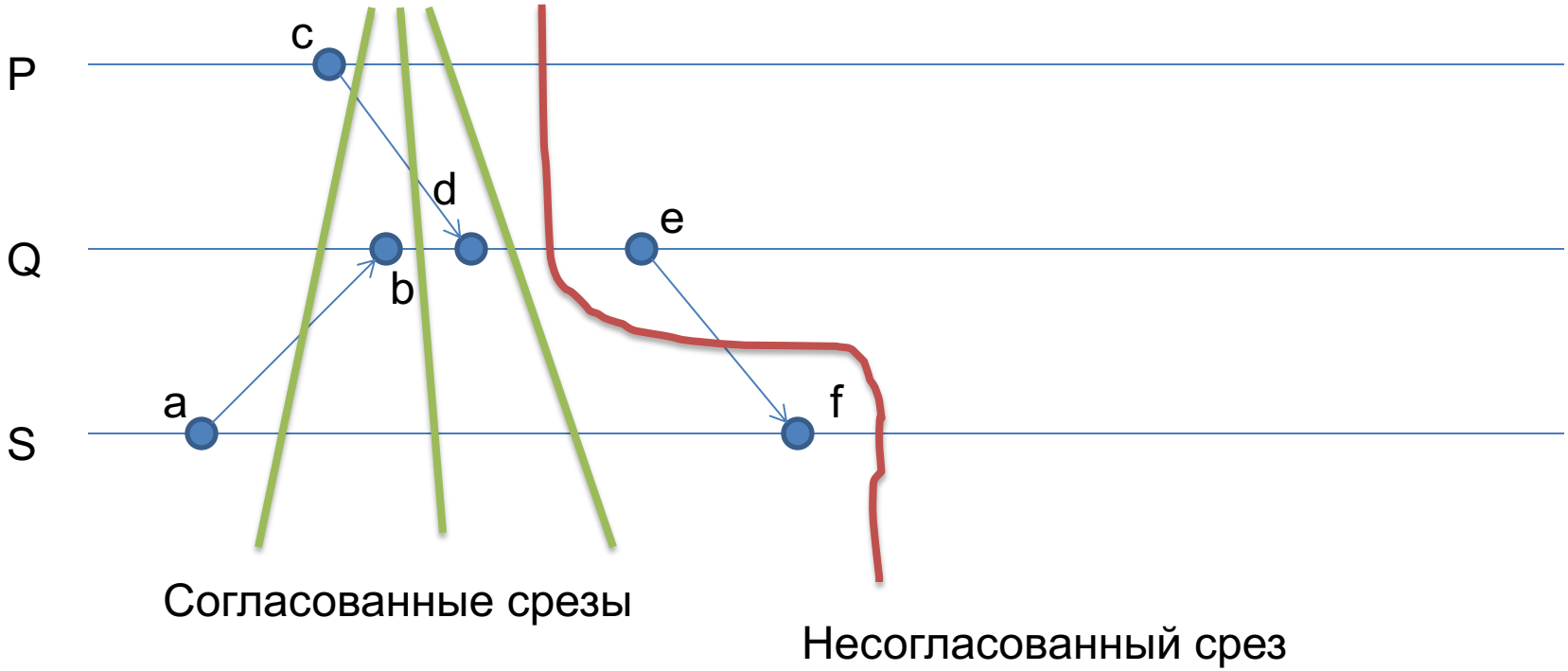
- DEF: G является **согласованным срезом** тогда и только тогда, когда:

$$\forall e \in E, f \in G : e \rightarrow f \Rightarrow e \in G$$

эквивалентное определение:

$$\nexists e \in E \backslash G, f \in G : \; e \rightarrow f$$

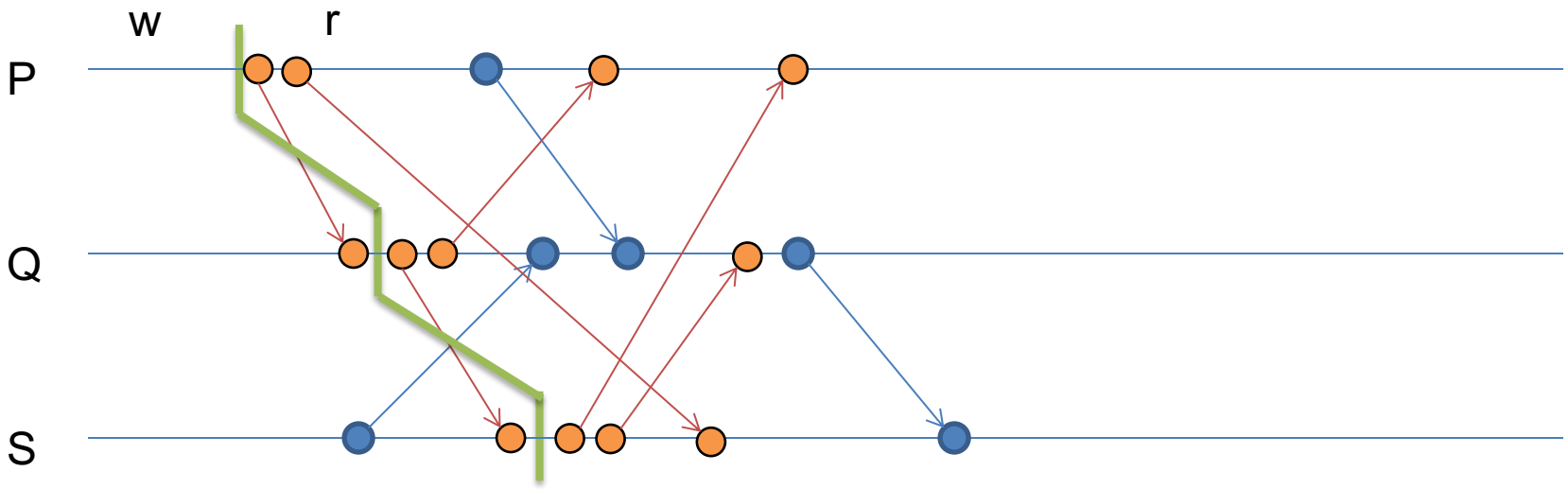
Согласованное глобальное состояние



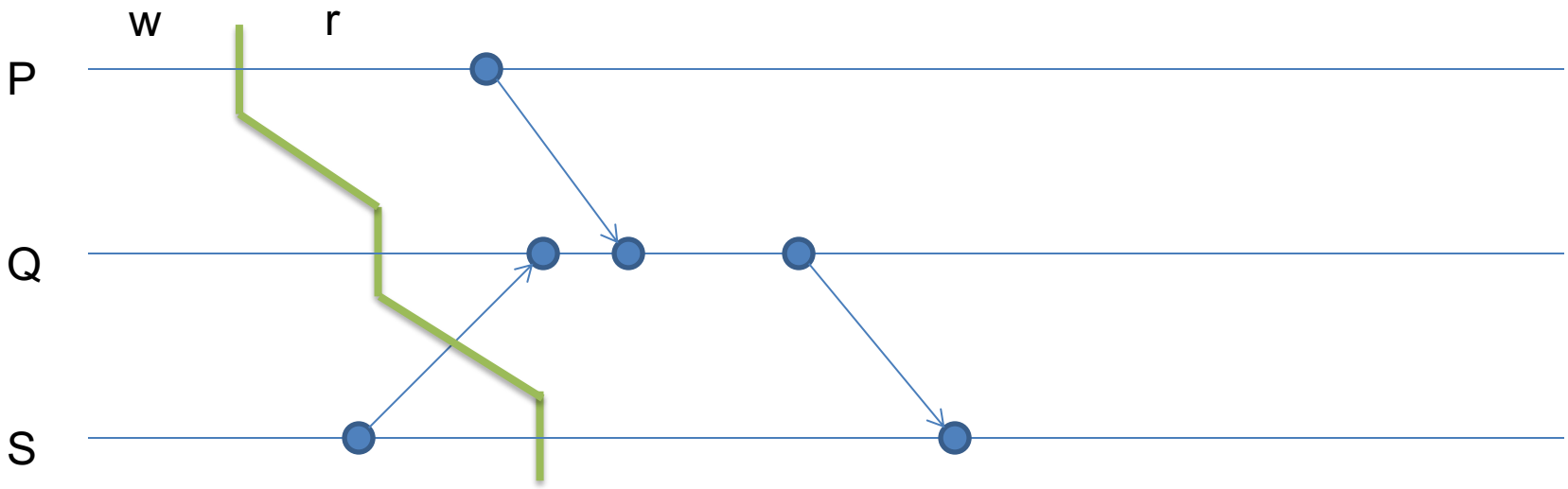
Алгоритм Чанди-Лампорта для согласованного запоминания глобального состояния системы

- Сначала все процессы *белые* (**w**)
- Процесс инициатор:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям
- При получении токена *белый* (**w**) процесс:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям

Алгоритм Чанди-Лампорта



Алгоритм Чанди-Лампорта

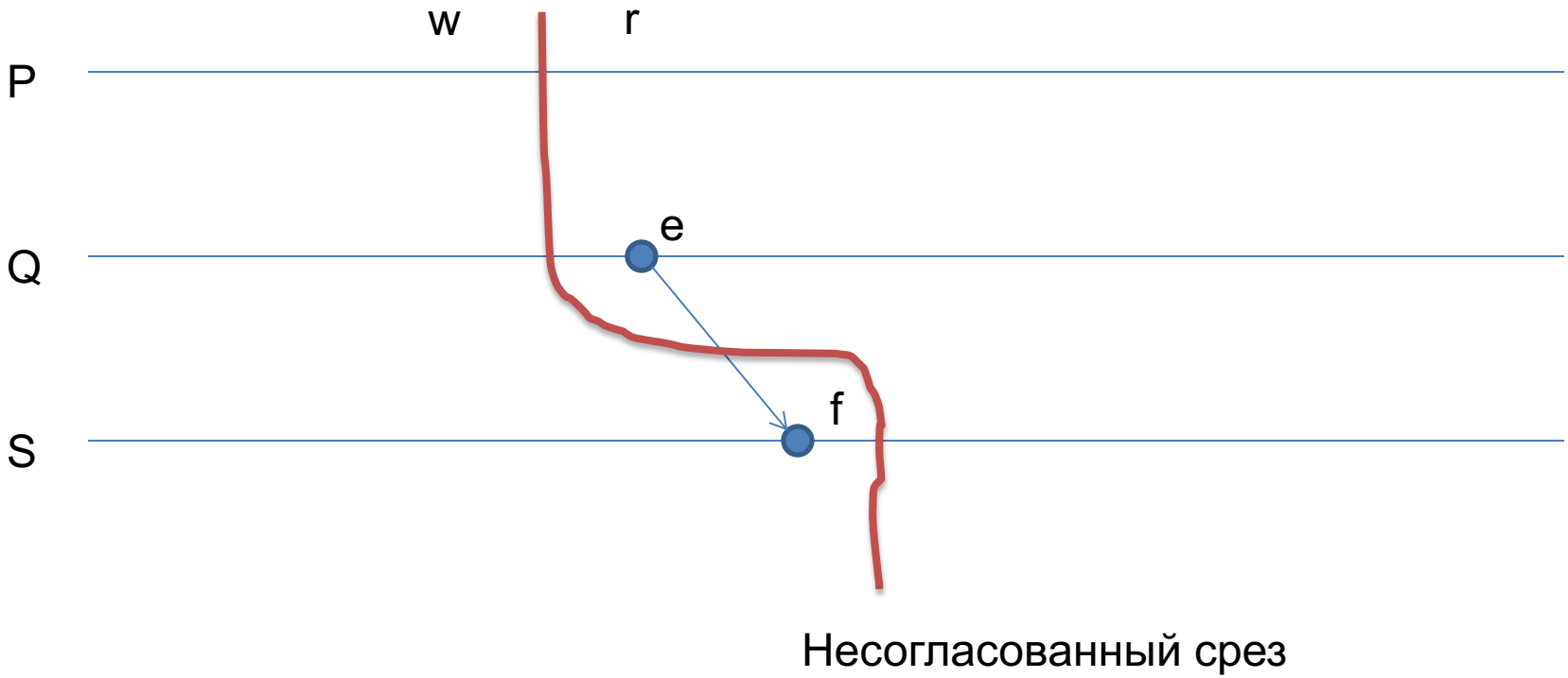


Алгоритм Чанди-Лампорта для согласованного запоминания глобального состояния системы

- Сначала все процессы *белые* (**w**)
- Процесс инициатор:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям
- При получении токена *белый* (**w**) процесс:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям

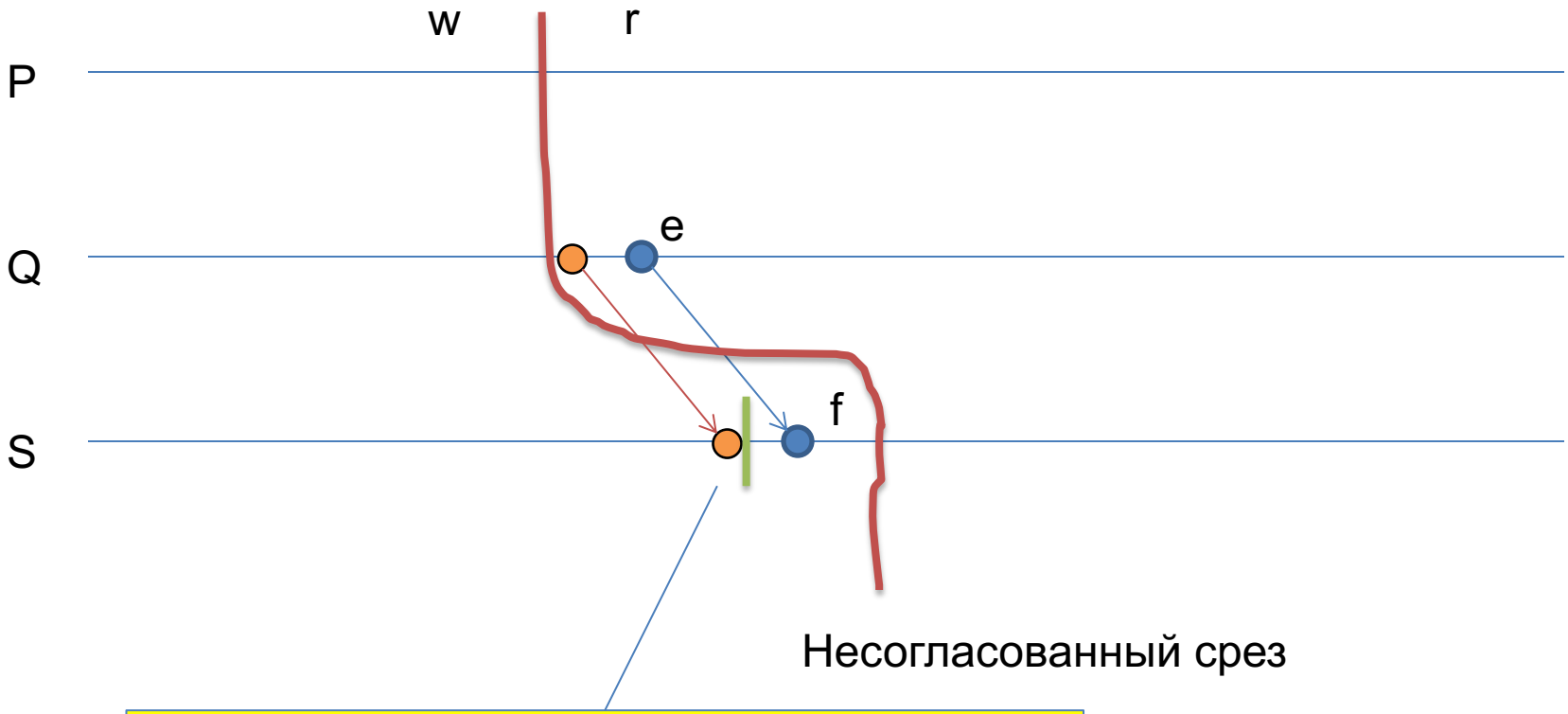
- **Запомненные состояния образуют согласованный срез**

Не может получиться несогласованный срез?



Несогласованный срез

Не может получиться несогласованный срез!



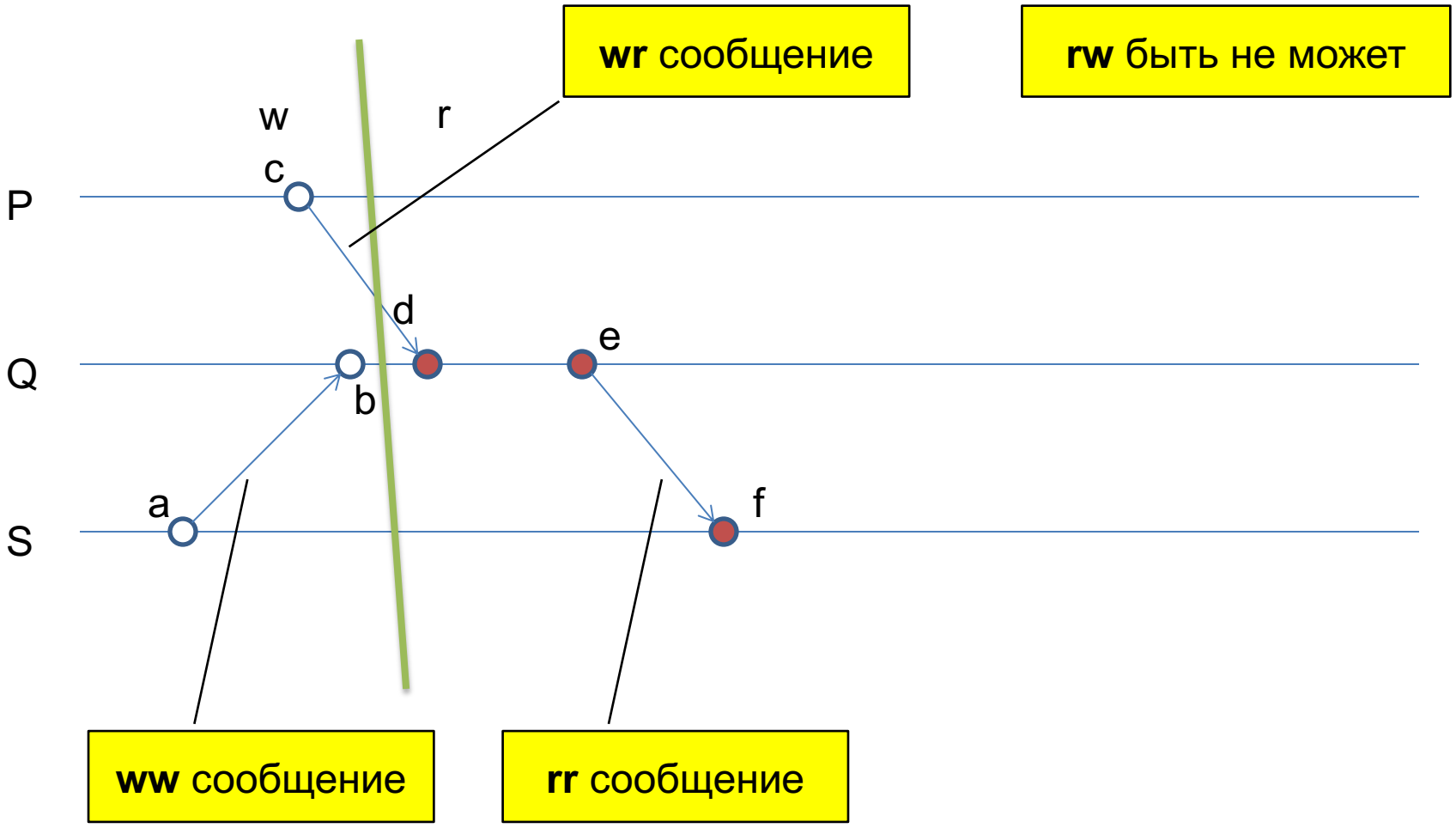
Сообщение посланное первым, должно прийти первым: First-In First-Out (**FIFO**)

Алгоритм Чанди-Лампорта для согласованного запоминания глобального состояния системы

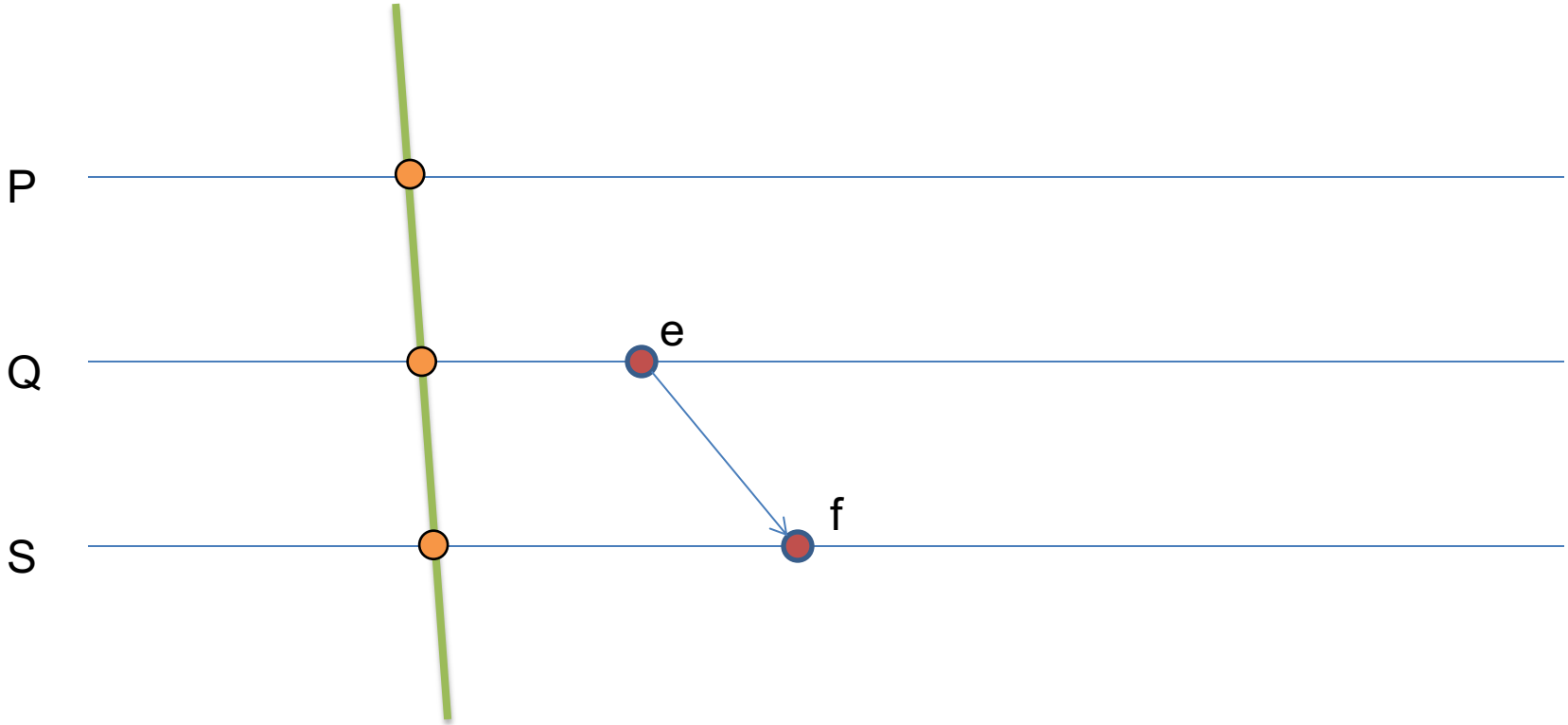
- Сначала все процессы *белые* (**w**)
- Процесс инициатор:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям
- При получении токена *белый* (**w**) процесс:
  - Запоминает свое состояние и становится *красным* (**r**)
  - Посылает токен всем соседям

- **Запомненные состояния образуют согласованный срез**
  - *Если сообщение между процессами идут FIFO*.

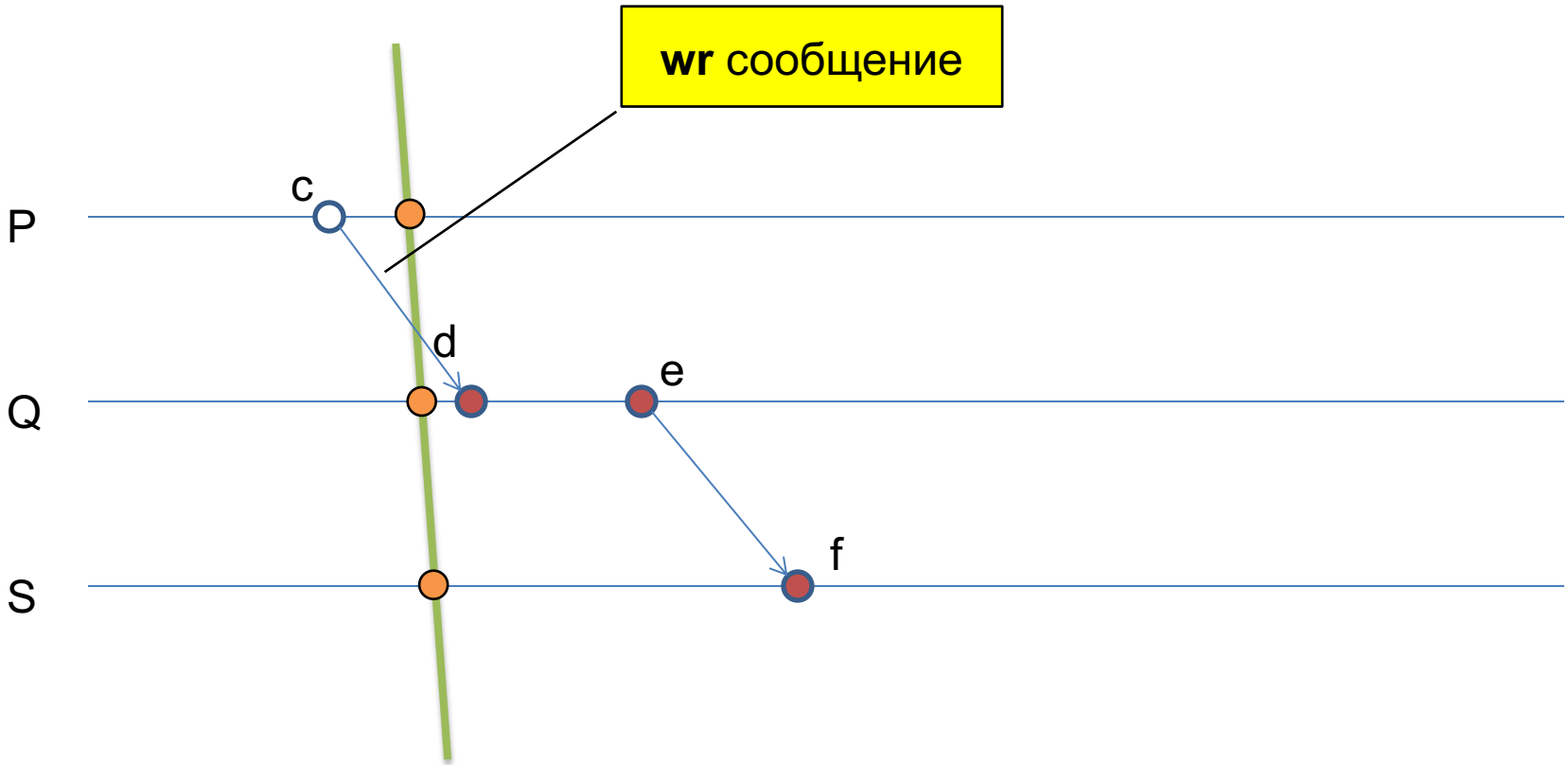
Классификация сообщений



Восстановление системы из запомненного состояния



Сообщения в пути!



Запоминание сообщений

- Чтобы восстановить работу системы из запомненного состояния (checkpoint) надо еще запоминать сообщения в пути

- Нужно запомнить все сообщения m такие что:

$$snd(m) \in G \;\&\; rcv(m) \in E \backslash \mathrm{G}$$

- В алгоритме Чанди-Лампорта это **wr** сообщения

Запоминание сообщений

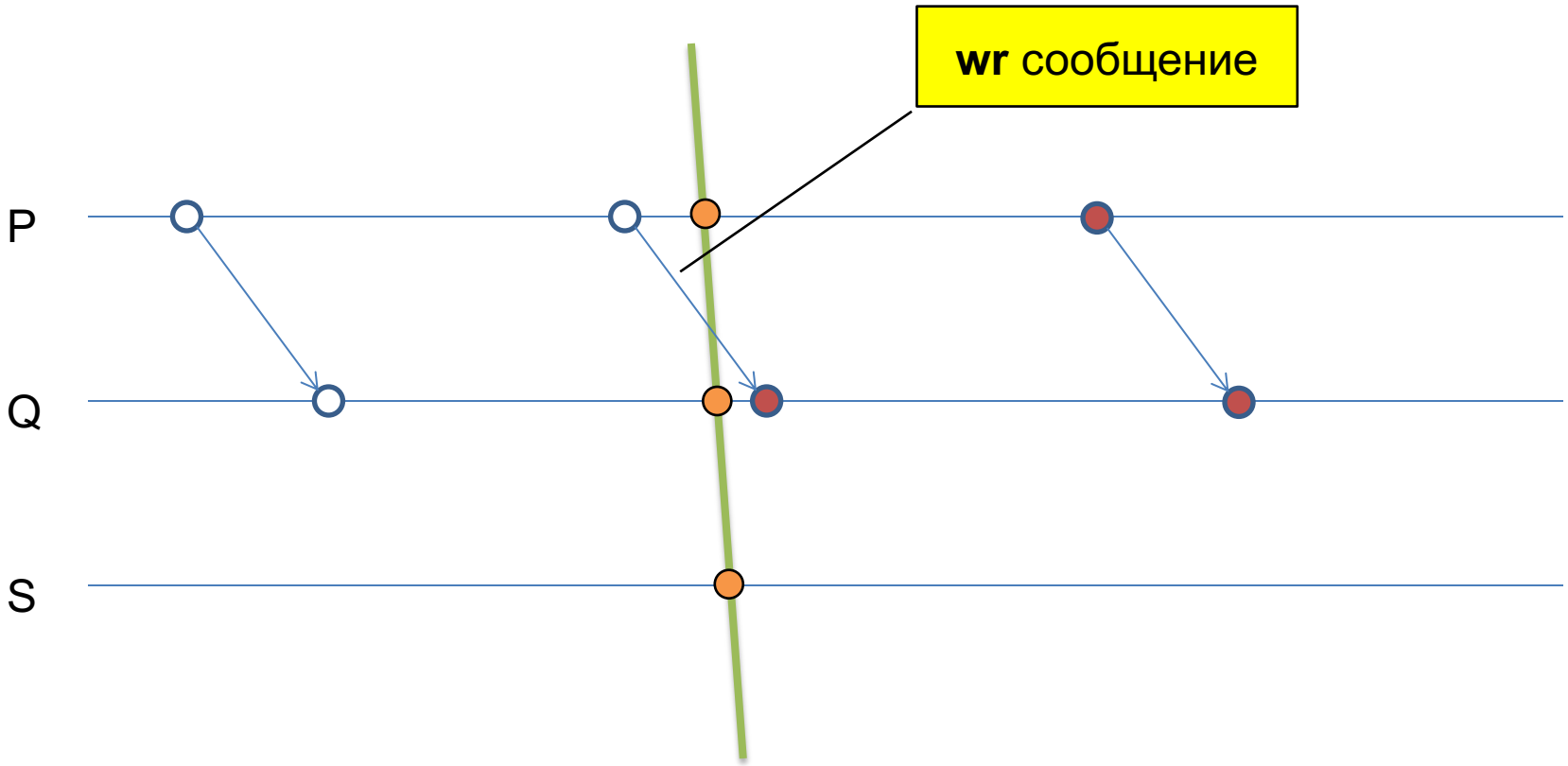
- Чтобы восстановить работу системы из запомненного состояния (checkpoint) надо еще запоминать сообщения в пути

- Нужно запомнить все сообщения m такие что:

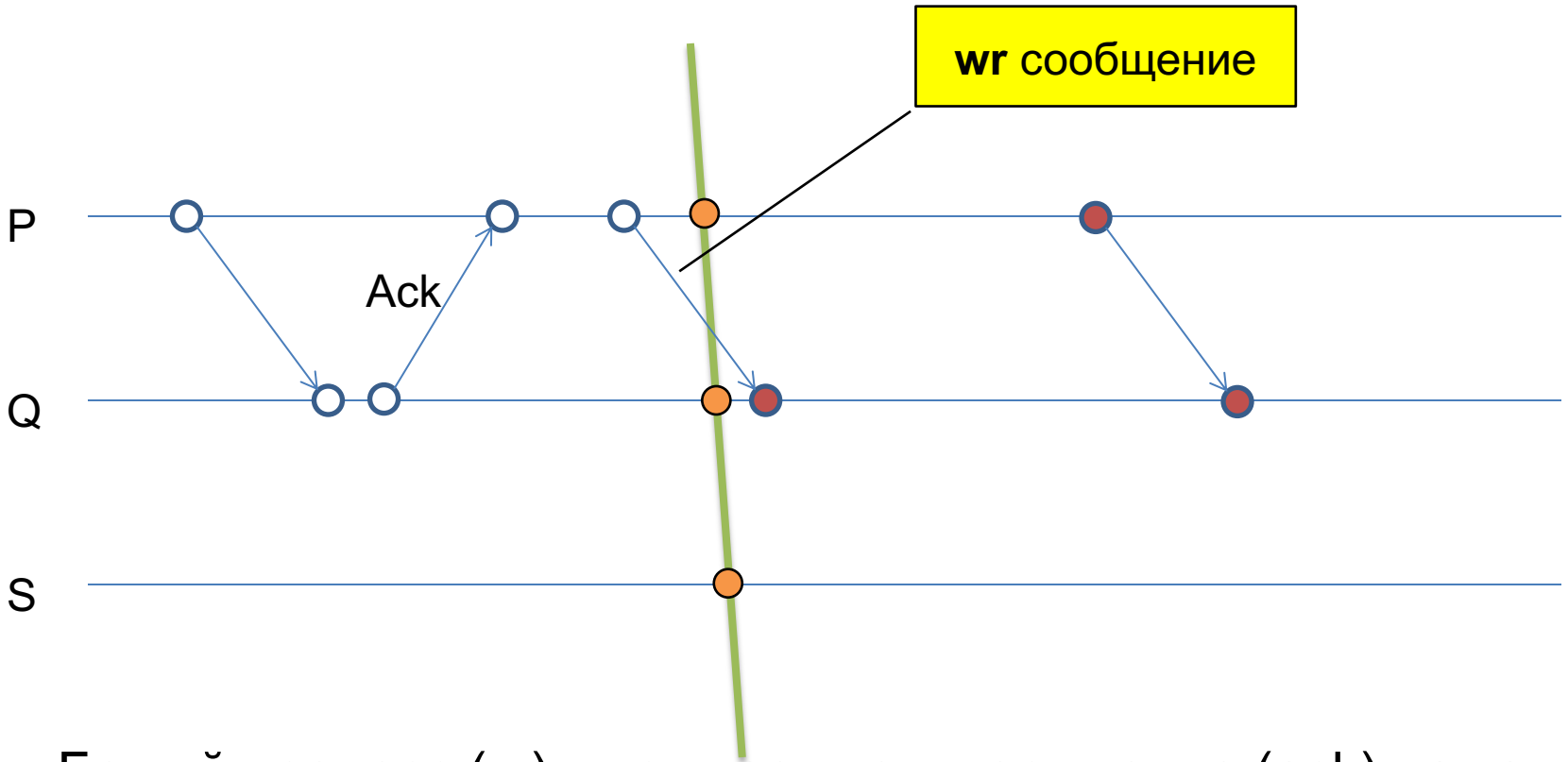
$$snd(m) \in G \;\&\; rcv(m) \in E \backslash \mathrm{G}$$

- **Варианты реализации**
  - Запоминание на стороне отправителя
  - Запоминание на стороне получателя

Запоминание на стороне отправителя

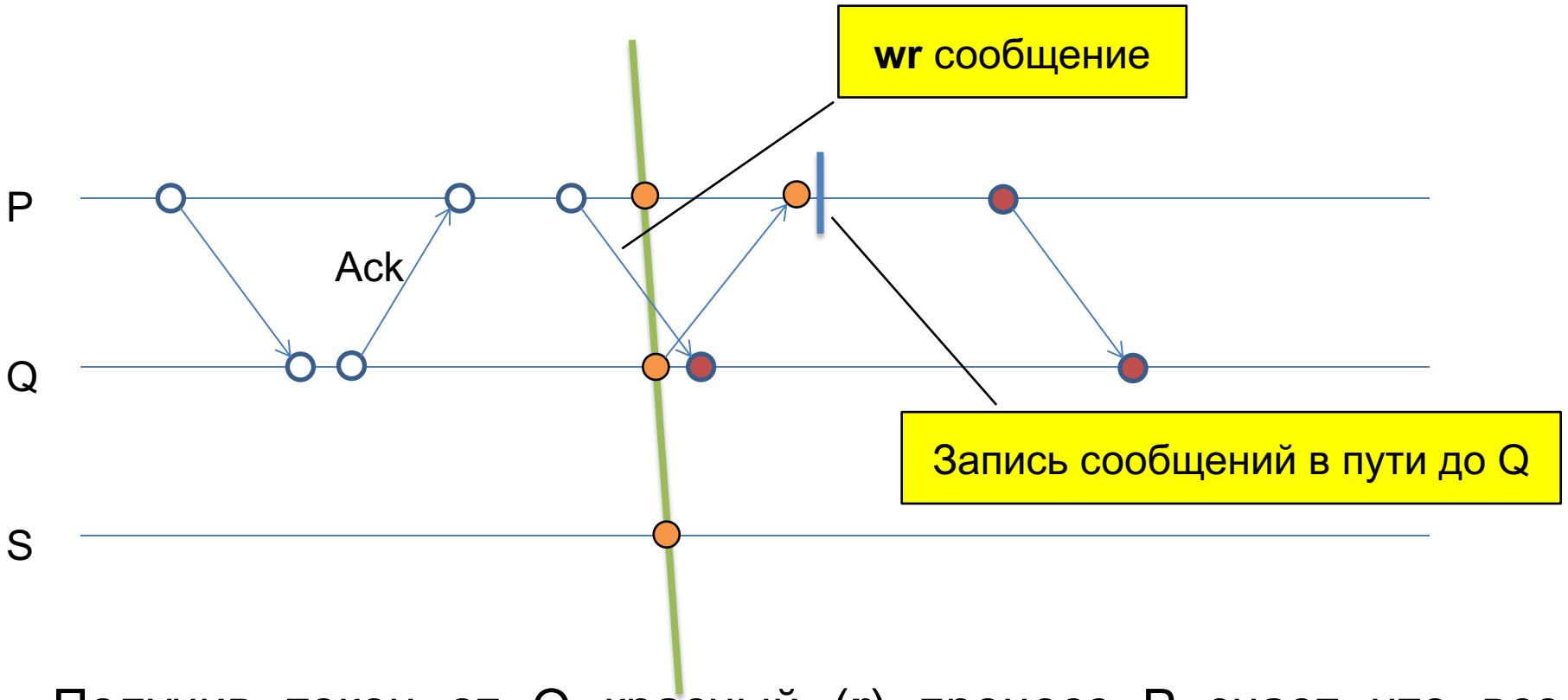


Запоминание на стороне отправителя



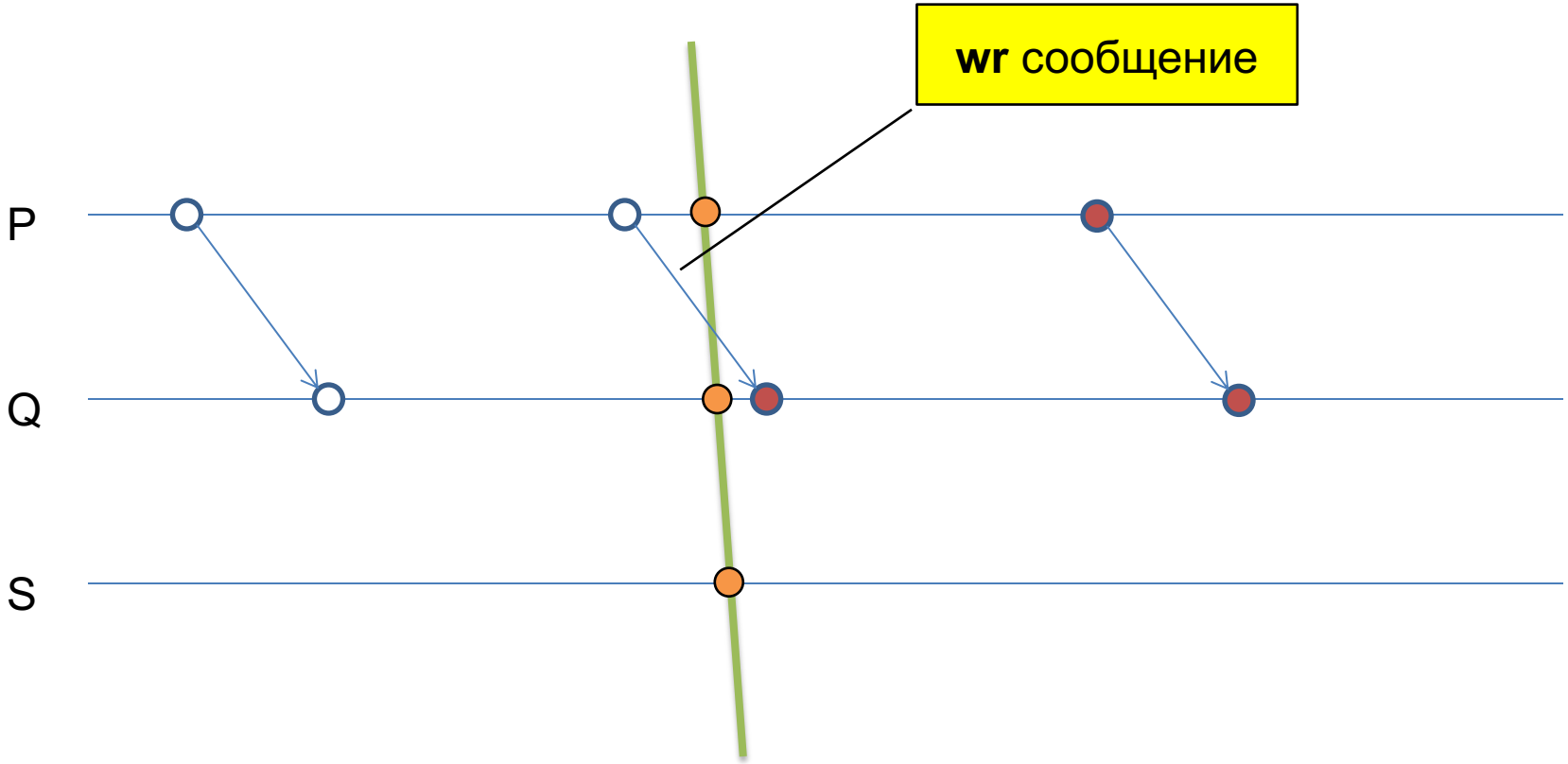
- Белый процесс (**w**) высылает подтверждение (ack) на каждое полученное сообщение
- Пославший процесс знает что его не надо будет хранить и его можно удалить из буфера хранения

Запоминание на стороне отправителя

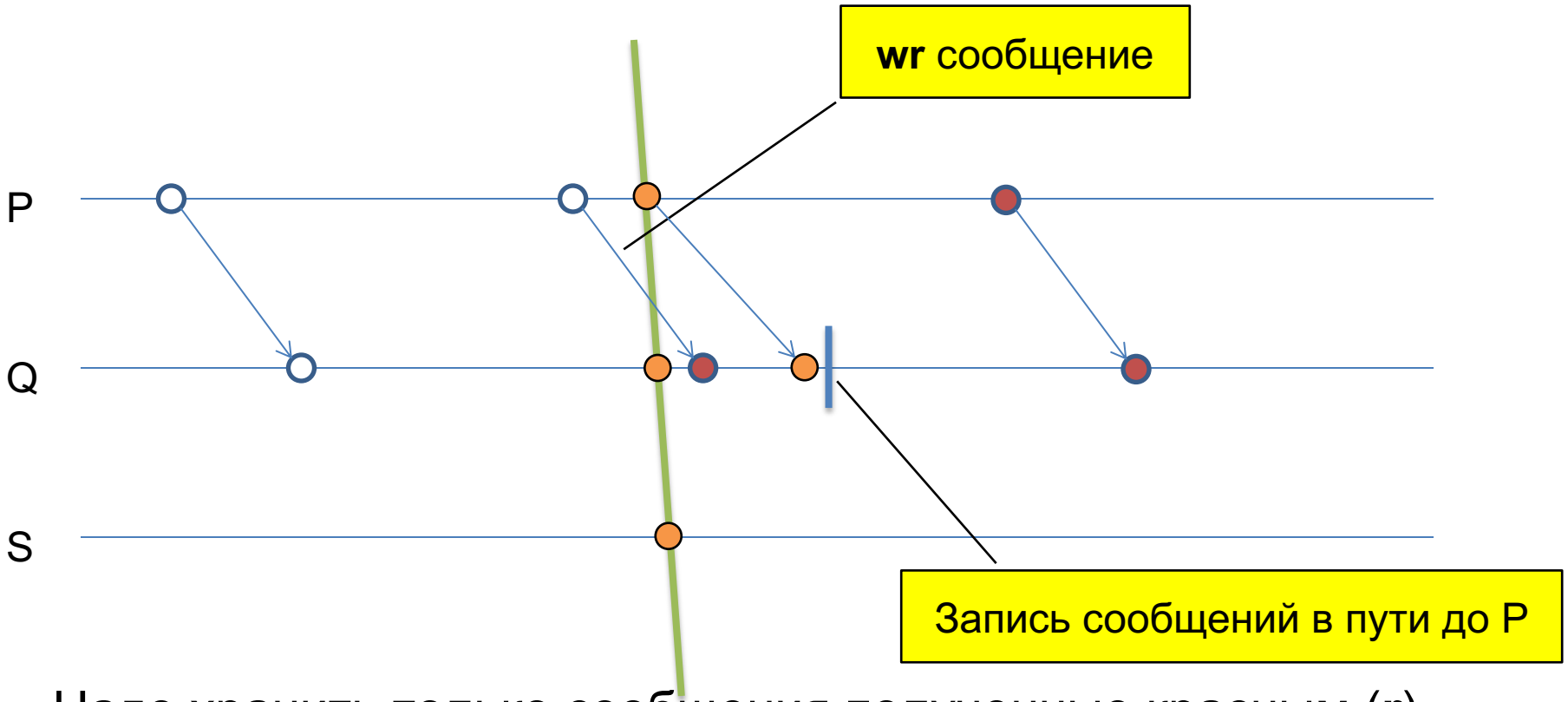


- Получив токен от Q красный (**r**) процесс P знает что все неподтвержденные сообщения, которые посылались процессу Q, надо сохранить

Запоминание на стороне получателя



Запоминание на стороне получателя



- Надо хранить только сообщения полученные красным (**r**) процессом от белого (**w**)
- Получение маркера от процесса означает, что больше таких не будет

Запуск алгоритма много раз

- Можем запускать *несколько экземпляров* одного алгоритма
- Каждый экземпляр алгоритма имеет *уникальный номер*
  - Это пара из *номера инициатора* и *номера алгоритма запущенного этим инициатором*
- Норме экземпляра алгоритма передаётся с каждым сообщением, относящемуся к экземпляру алгоритма

ГЛОБАЛЬНЫЕ СВОЙСТВА (ПРЕДИКАТЫ)

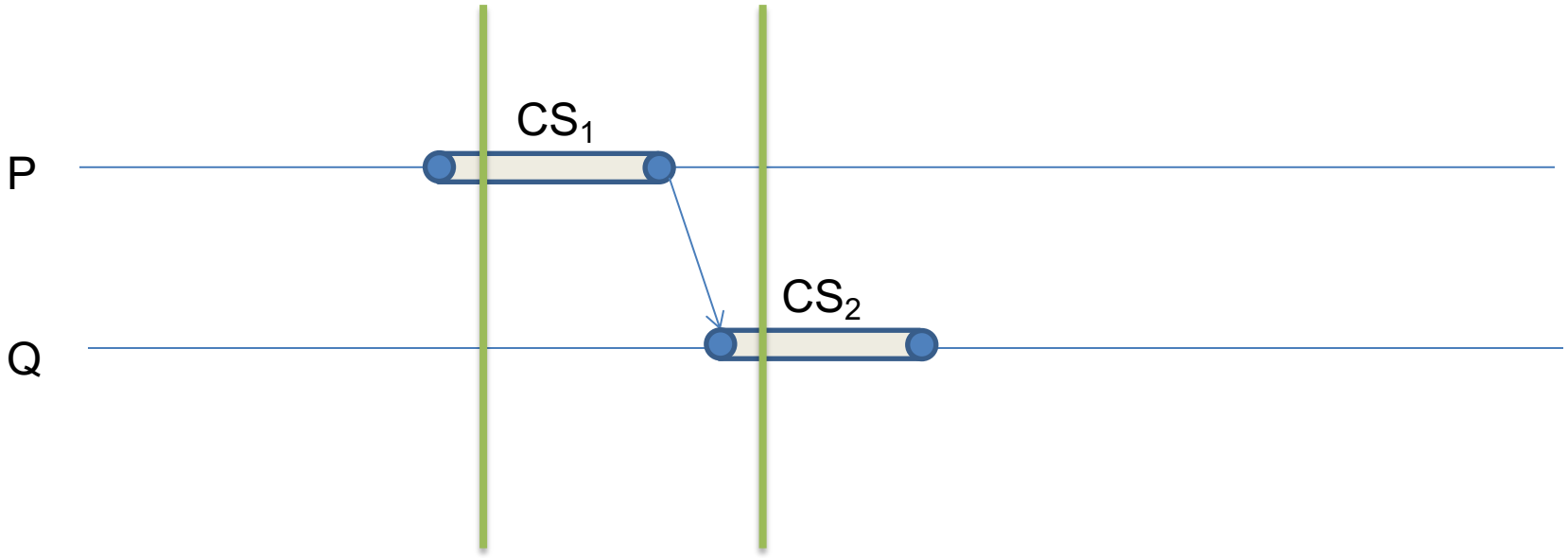
Глобальные свойства

- DEF: **Глобальное свойство** это предикат определенный над состоянием системы в целом

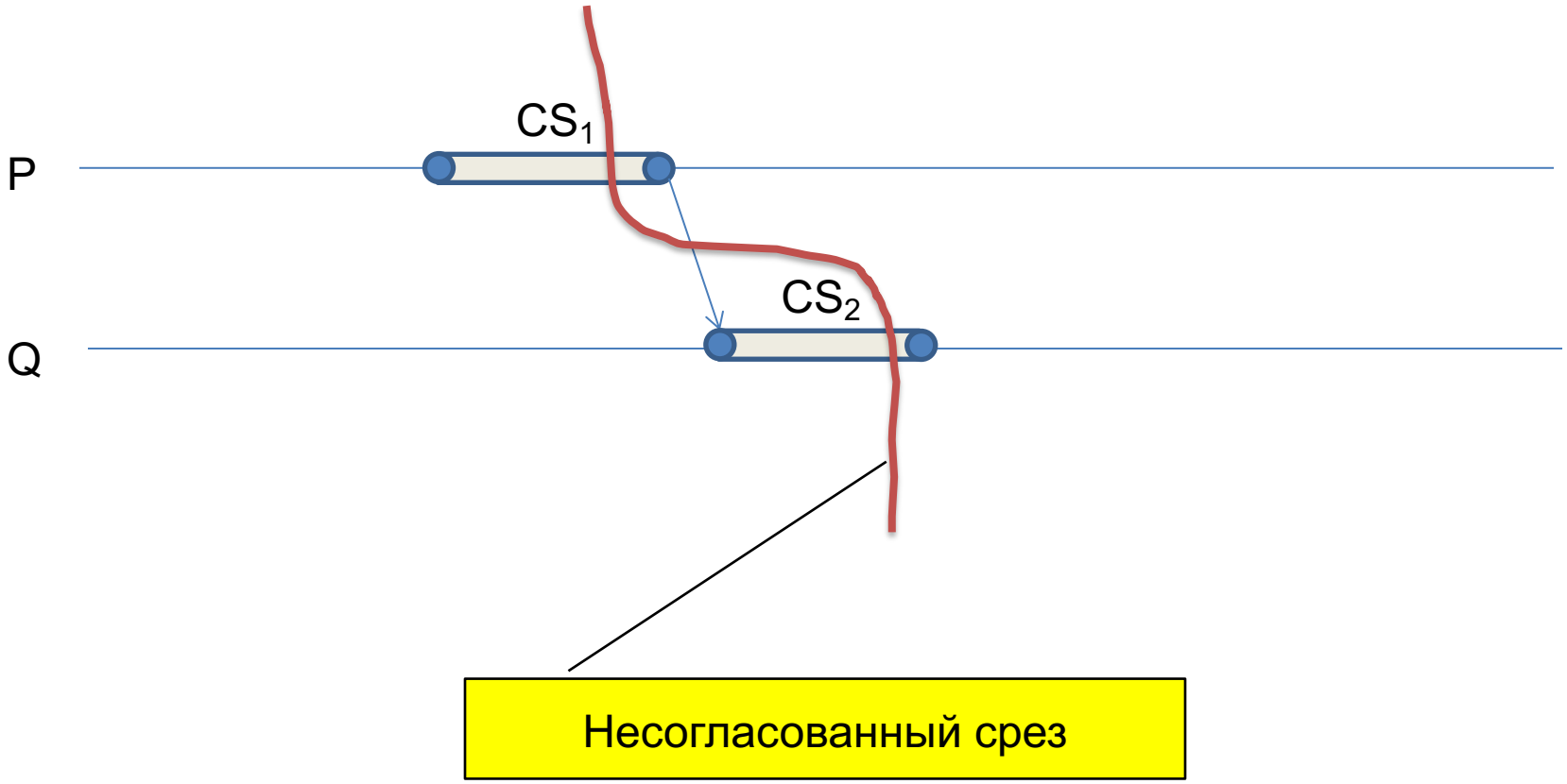
- **Примеры**
  - Какой-то процесс владеет токеном
  - Не более чем один процесс работает с каким-то ресурсом
  - Система попала во взаимное исключение (deadlock)
  - Распределенный алгоритм остановился
  - Согласованность состояния распределенной системы (постоянная сумма денег на счетах в банке и т.п.)

- Но что такое «**состояние системы**» – это должен быть *согласованный срез*

Пример – взаимное исключение



Пример – взаимное исключение



Стабильные и нестабильные предикаты

- **Стабильные предикаты**
  - DEF: Предикат $P(G)$ **стабильный** если для *согласованных срезов* G и H выполняется:

$$P(G)\ \&\ G \subset H \Rightarrow P(H)$$

  - Пример: потеря токена, взаимная блокировка и т.п.
- **Алгоритм «в лоб»**
  - Берем периодически согласованный срез (Чанди-Лампорт)
    - если предикат верен, значит будет верен и в дальнейшем
  - Но построение согласованного среза «дорого»
    - $O(N^2)$ сообщений
  - Научимся эффективней в специальных случаях

Нестабильные предикаты

- Вообще непонятно
  - Согласованных срезов много. Как перебрать все?
- DEF: **Локальный предикат** – это предикат по состоянию одного процесса.
  - Если предикат это дизъюнкция локальных предикатов, то всё просто (в одном процессе «истина», то глобальная истинна).
  - А если конъюнкция? Как определить истинность нестабильного конъюнктивного предиката, если есть разные срезы?

СЛАБЫЙ КОНЪЮНКТИВНЫЙ ПРЕДИКАТ

Слабый конъюнктивный предикат

- Если предикат P имеет вид конъюнкции *локальных* предикатов над состоянием каждого процесса:

$$P = L_1 \,\&\, L_2 \,\&\, \ldots L_n$$

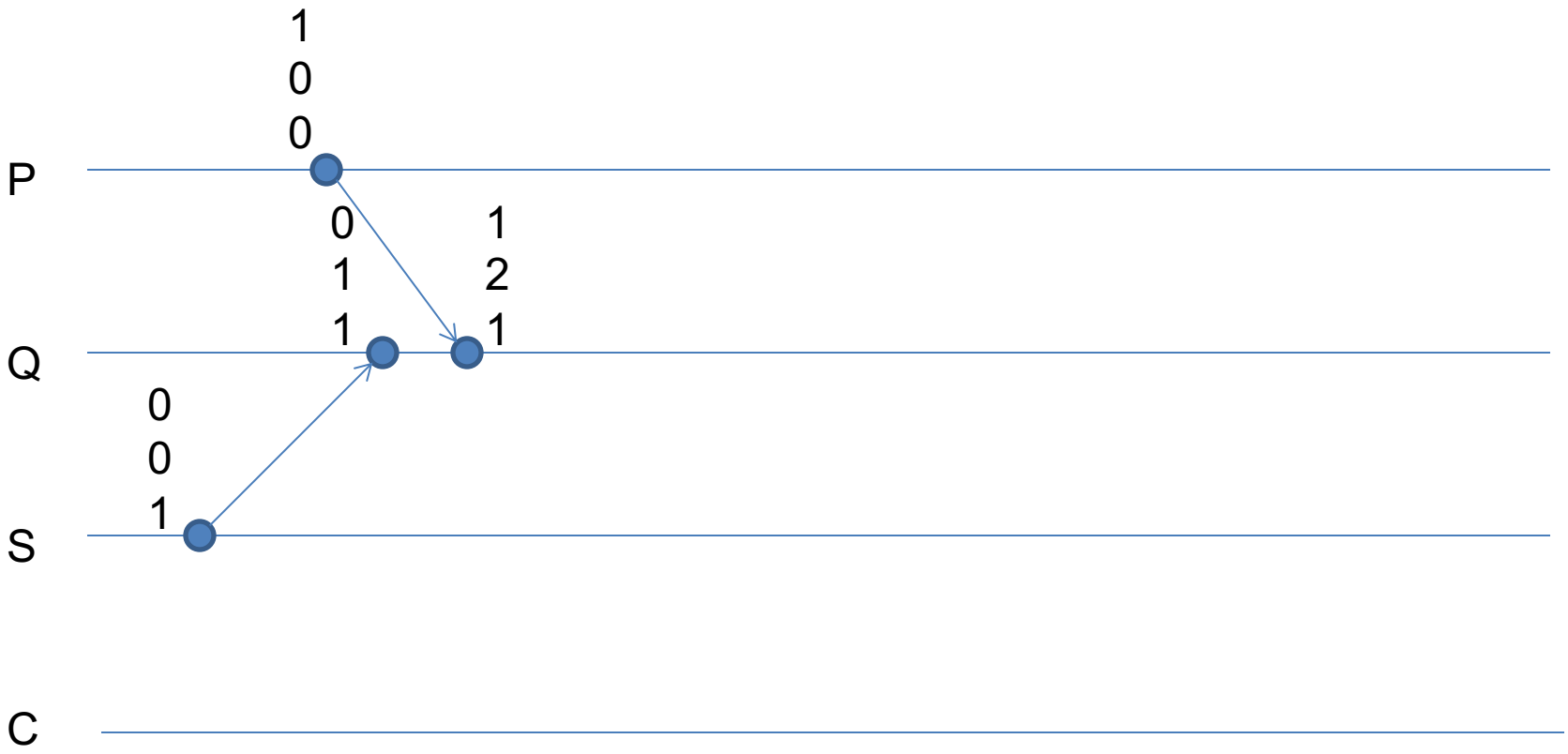
  - Пример предиката: «в системе нет координатора» (локальное условие «я не координатор»)

- DEF: **Слабый конъюнктивный предикат** истинен, если он истинен *на хотя бы одном согласованном срезе*.
- Сложные предикаты, являющиеся логической комбинаций локальных предикатов, всегда можно представить в нормальной дизъюнктивной форме и рассматривать как дизъюнкция слабых конъюнктивных предикатов

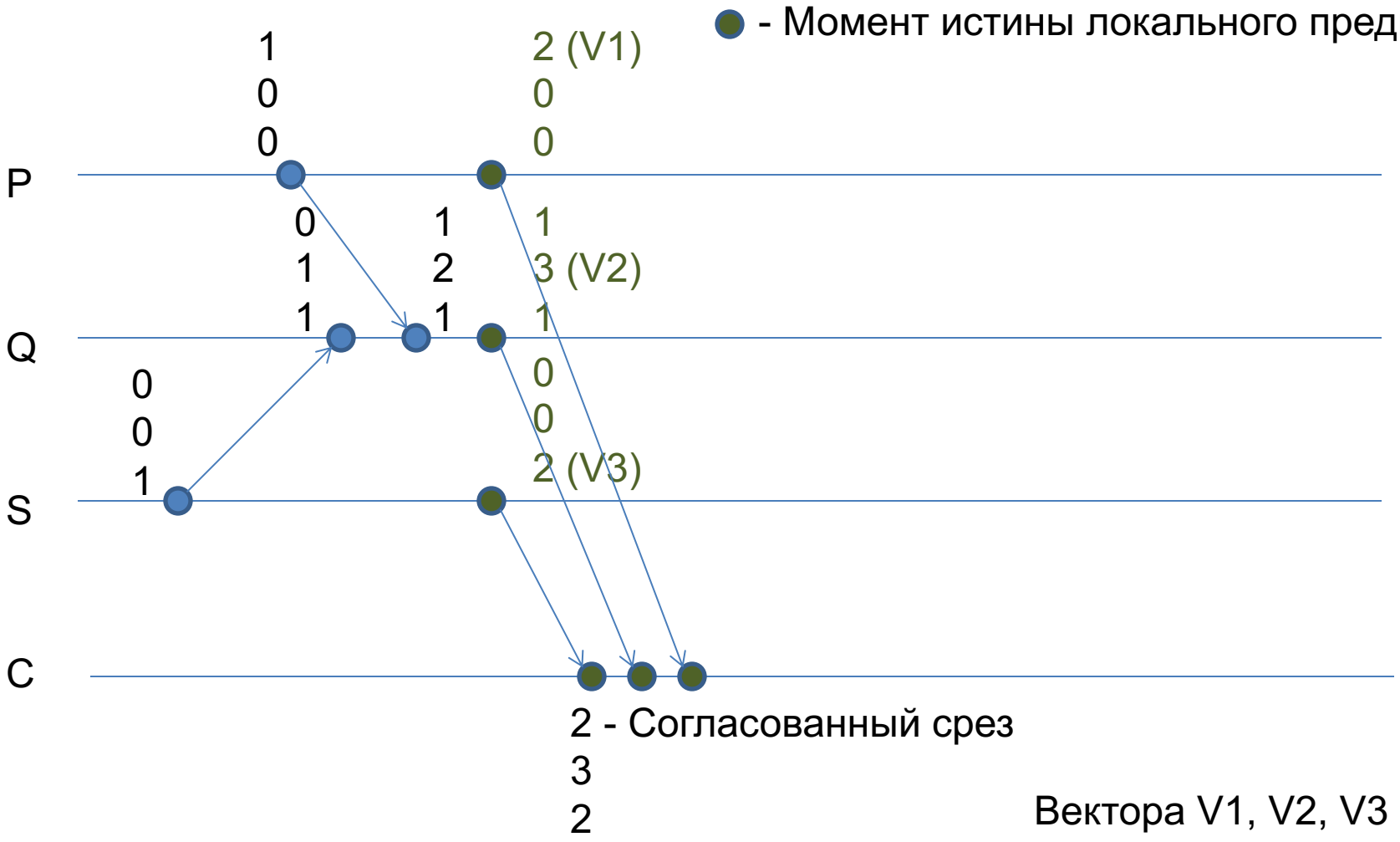
Слабый конъюнктивный предикат: централизованный алгоритм

- Каждый работающий процесс отслеживает свое векторное время VC.
- При наступлении истинности локального предиката L [увеличиваем свою компоненту вектора времени] и посылаем сообщение координатору C, [указывая векторное время, когда это произошло].
- В этом случае, любой срез можно однозначно задать вектором.
  - Координатор поддерживает в памяти *срез-кандидат* и очередь необработанных сообщений от каждого процесса.

Векторные часы

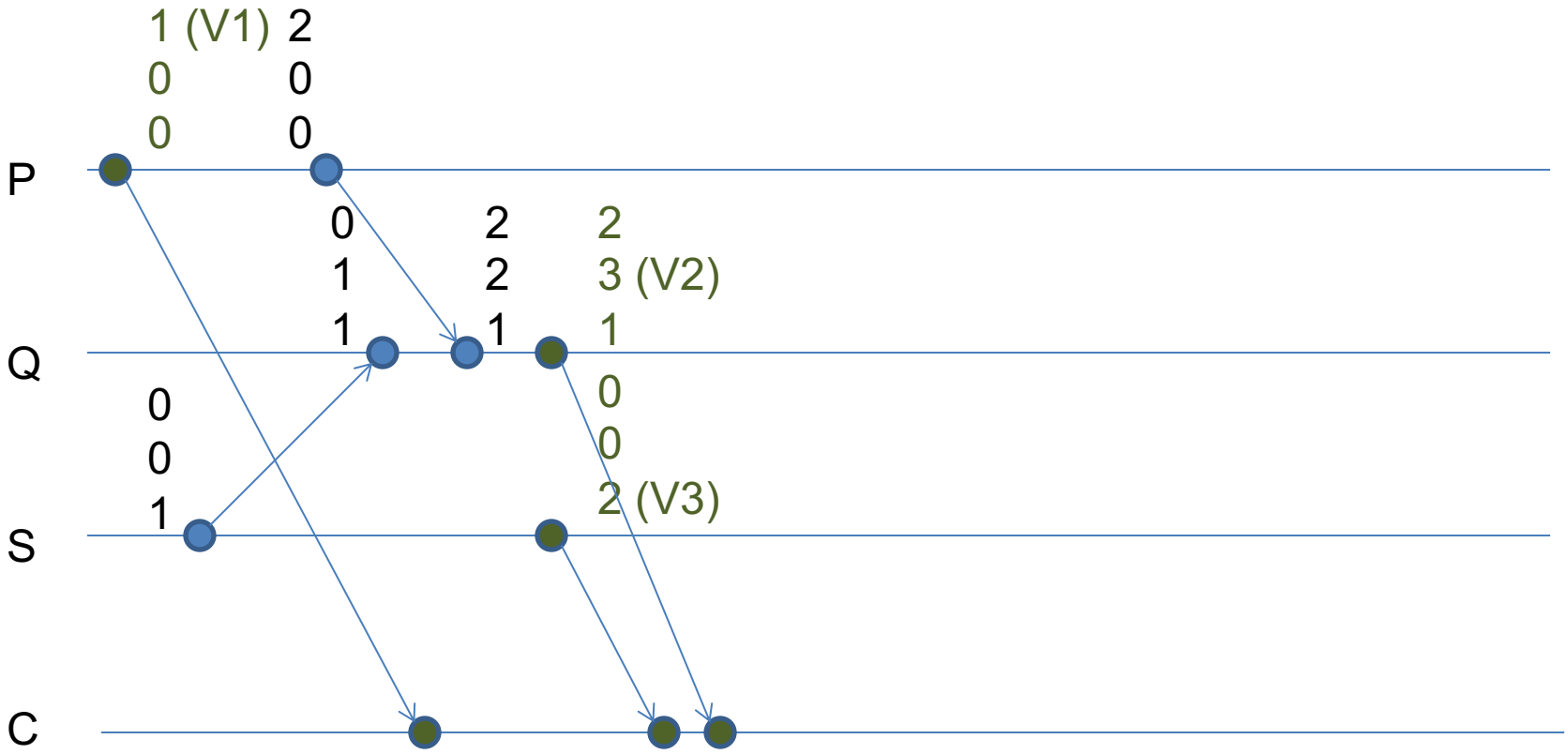


Согласованный срез-кандидат



Вектора V1, V2, V3
попарно несравнимы

Несогласованный срез-кандидат



1 – несогласованный срез-кандидат

3

2

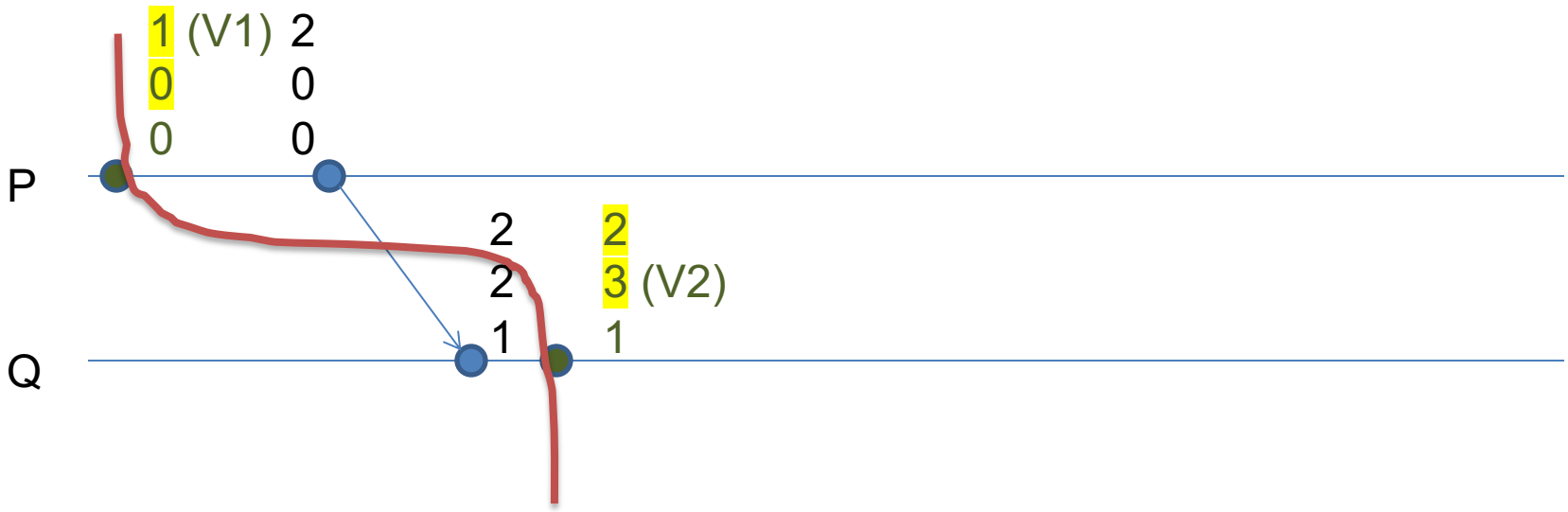
Вектор V1 < V2

Анализ алгоритма

- **Теорема 1**
  - Срез кандидат **согласован** тогда и только когда, когда все вектора в срезе кандидате попарно несравнимы

- **Эквивалентное утверждение:**
  - Срез кандидат **не согласован** тогда и только когда, когда в срезе кандидате найдутся два вектора $V_1 < V_2$
- **Наблюдение**:
  - Для сравнения векторов на 2-х процессах достаточно только соответствующих компонент
  - Для хранения среза кандидата достаточно только одного вектора с соответствующими компонентами

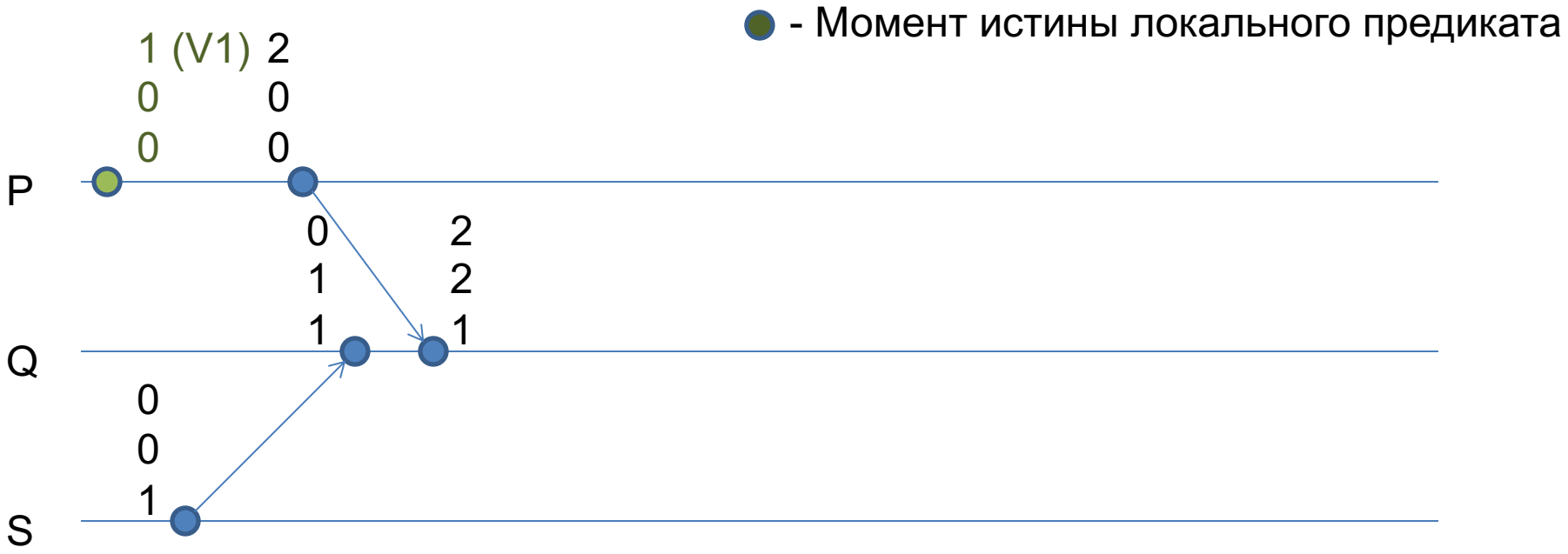
Несогласованный срез-кандидат



Слабый конъюнктивный предикат: централизованный алгоритм (2)

- Координатор хранит **вектор** среза-кандидата и флажок для каждой его компоненты:
  - Красный – этот элемент не может быть частью согласованного среза; Зеленый – наоборот
- Начальное состояние: всё по нулям, красное
- Обрабатываем приходящие сообщения только от красных процессов (сообщения от зеленых ставим в очередь)
  - Сравниваем пришедший вектор попарно с другими процессами (сравниваются только две соответствующие компоненты!),
    - Если новый вектор больше (нарушилась попарная несравнимость!), то делаем меньший процесс красным
    - После обработки сообщения от процесса его делаем зеленым
- Всё зеленое, значит нашли согласованный срез!

Алгоритм


- Момент истины локального предиката
1 (V1)
0
0
2
0
0
P
0
1
1
2
2
1
Q
0
0
1
S

Алгоритм


- Момент истины локального предиката
1 (V1)
0
0
2
0
0
P
0
1
1
2
2
1
2
3 (V2)
1
Q
0
0
1
S

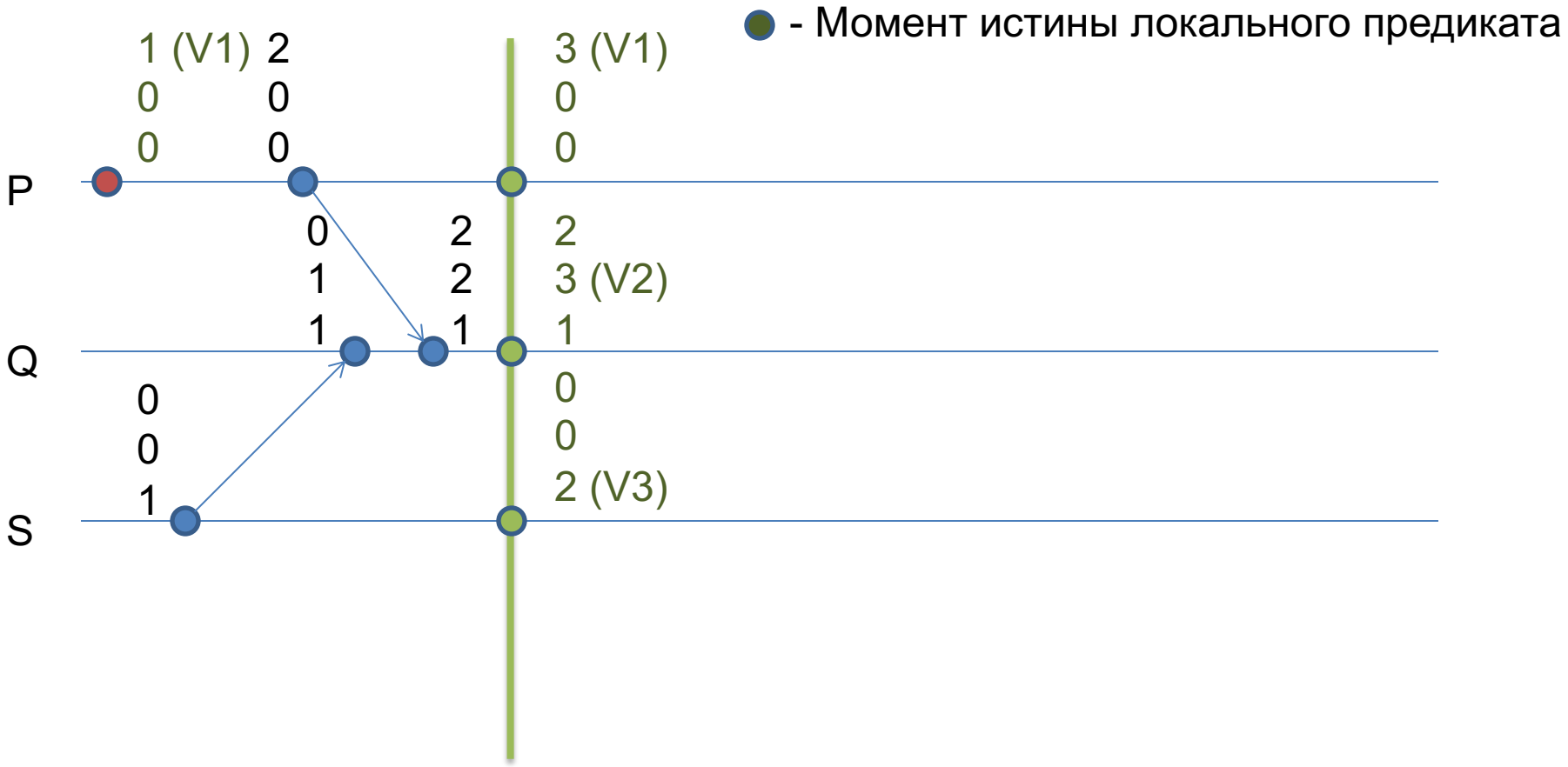
Алгоритм


- Момент истины локального предиката
1 (V1)
0
0
2
0
0
P
0
1
1
2
2
1
2
3 (V2)
1
Q
0
0
1
0
0
2 (V3)
S

Алгоритм


- Момент истины локального предиката
1 (V1)
0
0
2
0
0
3 (V1)
0
0
P
0
1
1
2
2
1
3 (V2)
1
Q
0
0
1
0
0
2 (V3)
S

Согласованный срез-кандидат



Анализ алгоритма

- **Теорема 2**
  - Этот алгоритм корректен.

- **План доказательства**
  - Он никогда не пропустит согласованный срез-кандидат
  - Компонента согласованного среза, становится кандидатом «зеленой» и всегда будет оставаться «зеленой».

Слабый конъюнктивный предикат: распределенный алгоритм

- Каждый процесс имеет своего собственного координатора
- Процессы шлют сообщения (как раньше) своим координаторам, координаторы общаются между собой
  - Координаторы пересылают друг другу срезы-кандидаты и флажки (зеленый/красный)
- Красные координаторы обрабатывают сообщения от своих процессов (как раньше)
  - После обработки сообщения становятся зелеными
  - Если в процессе обработки они пометили красным другой процесс, то шлют сообщение его координатору

СПЕЦИАЛЬНЫЕ СЛУЧАИ СТАБИЛЬНЫХ ПРЕДИКАТОВ

Специальные случаи стабильных предикатов: более эффективные алгоритмы

- Останов системы, например, при поиске кратчайшего пути
  - Обобщим для т.н. **диффундирующих вычислений**

- Взаимная блокировка при распределении ресурсов
  - Обобщим для т.н. **локально-стабильных предикатов**

ОСТАНОВ ДИФФУНДИРУЮЩЕГО ВЫЧИСЛЕНИЯ

Мотивирующая задача

- **Распределенный алгоритм Дейкстры** для поиска кратчайшего пути от инициатора до остальных узлов
  - Каждый узел поддерживает кратчайшее расстояние до источника, в начале у всех узлов $d = \infty$.
  - Инициатор – источник. У него $d = 0$
  - Инициатор шлет всем соседям сообщение «distance 0»
  - Узлы получив сообщение «distance d_r» по ребру стоимости d_e обновляют расстояние у себя
$$d = \min(d, d_r + d_e)$$
  - Если расстояние обновилось, то высылают соседям «distance d»
- Рано или поздно алгоритм найдет все кратчайшие d
- Но как узнать что он закончил работу?

Диффундирующие вычисления (1)

- DEF: В **диффундирующем вычислении** процессы бывают в двух состояниях:
  - Активные и Пассивные
  - Получение сообщения –> Делает процесс активным
  - Только активный процесс может посылать сообщения
  - Активный процесс может в любой момент стать пассивным
  - Алгоритм начинается с одного активного процесса-инициатора

Диффундирующие вычисления (2)

- Проблема останова!
  - **DEF**: Диффундирующее вычисление остановилось если все процесс пассивные и нет сообщений в пути
- Как инициатор может узнать о том, что алгоритм завершился?

Диффундирующие вычисления – алгоритм (1)

- Алгоритм **Дейкстры и Шолтена**:
  - Выстраиваем процессы в дерево
  - Требуем подтверждение (ack) на каждое сообщение
    - Считаем баланс (сколько послали сообщений минус сколько подтверждений получили).
  - Назовем процесс «зеленым» если он
    - пассивен
    - у него нет детей в дереве (childCount == 0)
    - у него нет неподтвержденные сообщения (balance == 0)
      - То есть его исходящий канал пуст
  - В противном случае будет считать процесс «красным»
  - Дерево будет содержать все «красные» процессы
  - Каждый процесс в дереве будет знать parentId

Диффундирующие вычисления – алгоритм (2)

- Алгоритм **Дейкстры и Шолтена**:
  - «Зеленый» процесс (не в дереве) при получении сообщения` становится новым листом в дереве (и «красным»),
    - делает parentId := sourceId (от кого получили сообщение)
    - Шлет сообщение в parent «я твой новый child»
    - Получи это сообщение, parent делает childCount++
  - Пассивный лист дерева (без детей) при отсутствии неподтвержденных сообщений удаляет себя из дерева (становится «зеленым»)
    - Шлет сообщение в parent «я больше не твой child»
    - Получив это сообщение, parent делает childCount—
  - Когда корень дерева (инициатор!) становится «зеленым», то диффундирующее вычисление завершено

ЛОКАЛЬНО-СТАБИЛЬНЫЕ ПРЕДИКАТЫ

Согласованные интервалы

- **DEF**: Пара *срезов* $F, G \subset E$ называется интервалом если $F \subset G$
  - Пишем $[F, G]$
- **DEF**: Интервал называется согласованным если:

$$\forall\, e \in E, f \in F : e \to f \Longrightarrow e \in G$$

- Эквивалентно:

$$\nexists\, e \in E \backslash \mathrm{G}, f \in F : e \to f$$

- Это обобщает определение согласованного среза если $F = G$
- **Теорема**:
  - Интервал $[F, G]$ согласован тогда и только когда когда внутри него есть согласованный срез H:

$$F \subset H \subset G$$

Барьерная синхронизация

- **DEF**: Интервал $[F, G]$ **барьерно-синхронизирован** если:
$$\forall f \in F, g \in E \setminus G : f \rightarrow g$$
- **Теорема**:
  - Любой барьерно-синхронизированный интервал согласован
- **Построение барьерной-синхронизации (3 алгоритма):**
  - Через координатора
  - Посылка каждый-каждому
  - Посылка токена два раза (по кругу)

Локально стабильные предикаты

- **DEF**: Локально-стабильный предикат -- стабильный предикат, определяемый группой процессов, у которых не меняется состояние
- **Пример**: Взаимная блокировка
  - Предикат – есть цикл в графе ожидания
  - Все процесс попавшие в цикл ничего не делают (ждут)
  - Важно! Нужен согласованный срез
- Алгоритм поиска с использованием барьерной синхронизации