

# Синхронизация в распределенных системах

# Основные свойства распределенных алгоритмов

Обычно децентрализованные алгоритмы имеют следующие свойства:

- Относящаяся к делу информация распределена среди множества ЭВМ.
- Процессы принимают решение на основе только локальной информации.
- Не должно быть единственной критической точки, выход из строя которой приводил бы к краху алгоритма.
- Не существует общих часов или другого источника точного глобального времени.

Первые три пункта все говорят о недопустимости сбора всей информации для принятия решения в одно место.

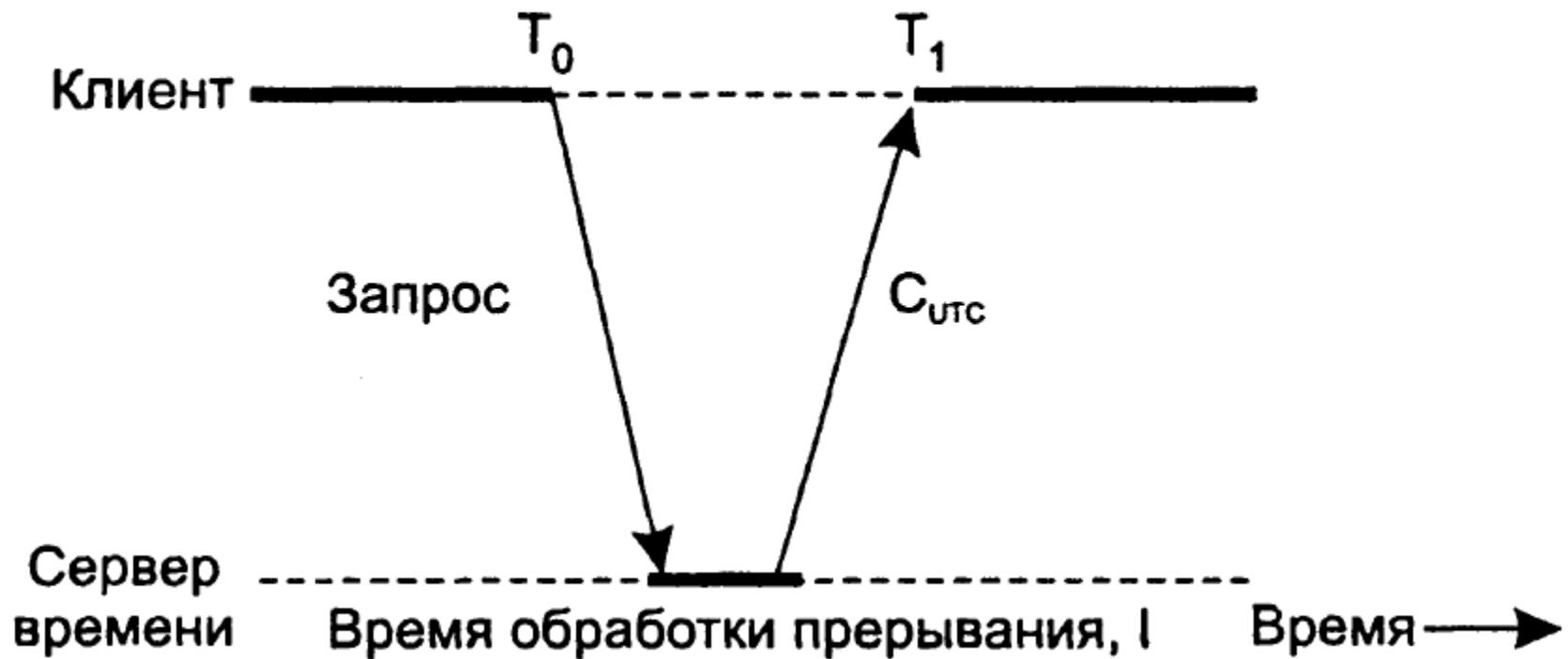
Обеспечение синхронизации без централизации требует подходов, отличных от используемых в традиционных ОС.

Последний пункт также очень важен - в распределенных системах достигнуть согласия относительно времени совсем непросто.

# Синхронизация времени.

## Алгоритм Кристиана

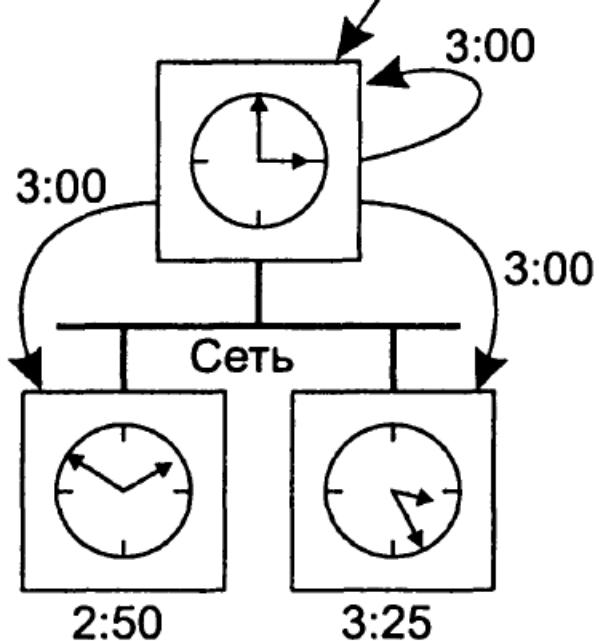
$T_0$  и  $T_1$  отсчитываются по одним и тем же часам



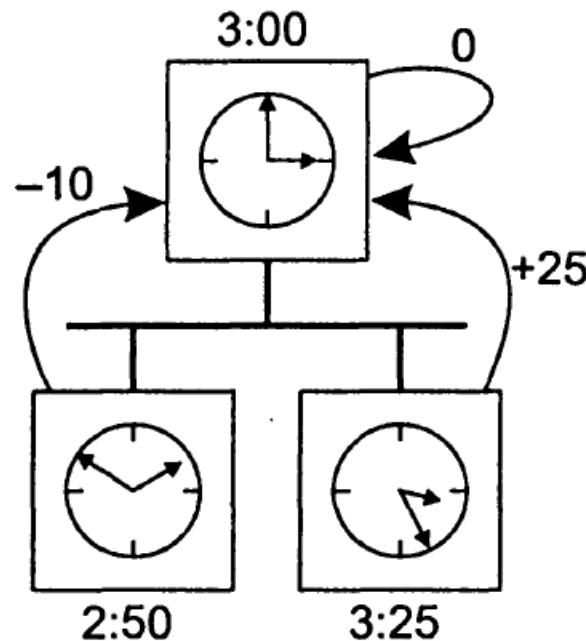
# Синхронизация времени.

## Алгоритм Беркли

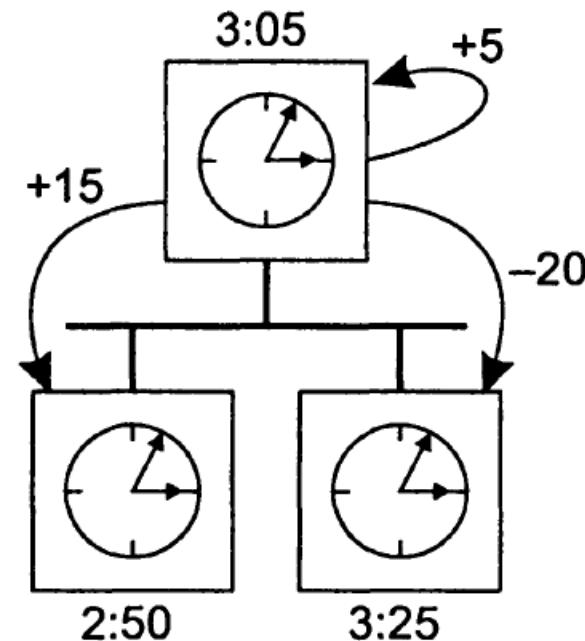
Демон времени



а



б



в

# Логические часы. Lamport

Для синхронизации логических часов Lamport определил отношение «произошло до». Выражение  $a \rightarrow b$  читается как « $a$  произошло до  $b$ » и означает, что все процессы согласны, что сначала произошло событие « $a$ », а затем « $b$ ». Это отношение может в двух случаях быть очевидным:

- 1) Если оба события произошли в одном процессе.
- 2) Если событие « $a$ » есть операция SEND в одном процессе, а событие « $b$ » - прием этого сообщения другим процессом.

Отношение  $\rightarrow$  является транзитивным.

Если два события « $x$ » и « $y$ » случились в различных процессах, которые не обмениваются сообщениями, то отношения  $x \rightarrow y$  и  $y \rightarrow x$  являются неверными, а эти события называют одновременными.

# Логические часы. Lamport

Введем логическое время С таким образом, что  
если  $a \rightarrow b$ , то  $C(a) < C(b)$

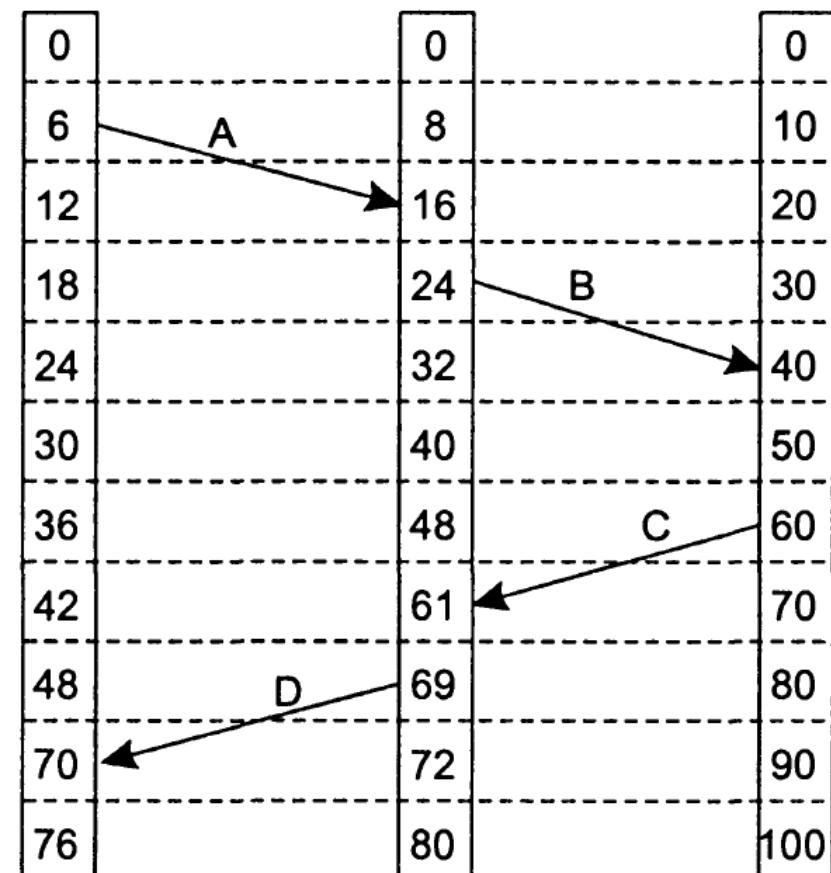
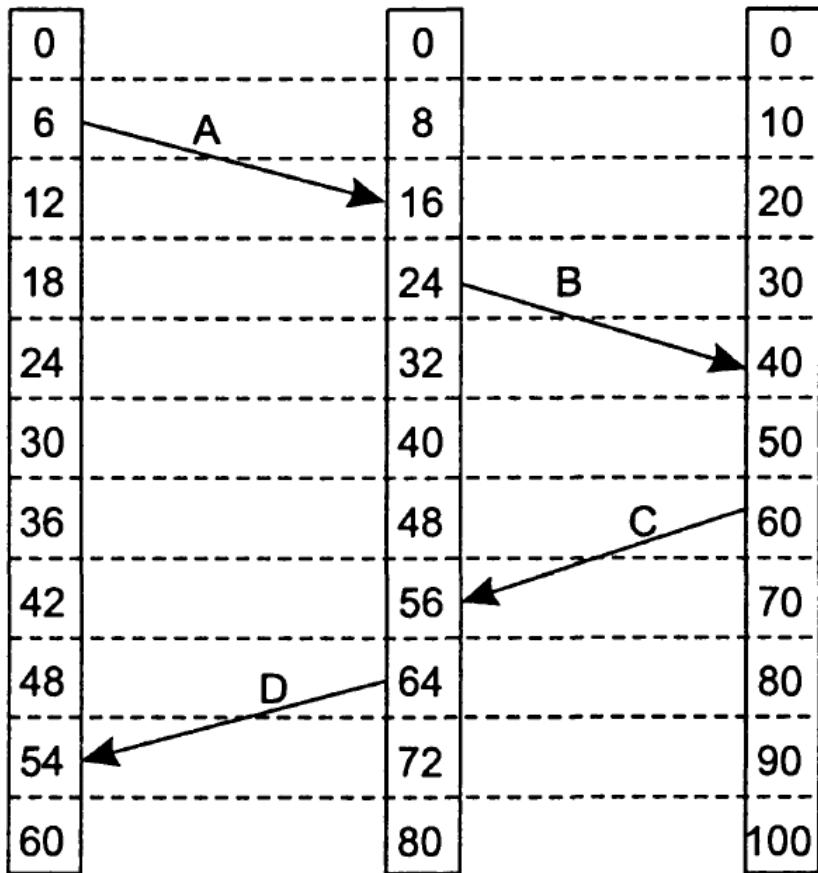
Алгоритм:

- 1) Часы  $C_i$  увеличиваются свое значение с каждым событием в процессе  $P_i$ :  
$$C_i = C_i + d \quad (d > 0, \text{ обычно равно } 1)$$
- 2) Если событие «а» есть посылка сообщения «т» процессом  $P_i$ , тогда в это сообщение вписывается временная метка  $tm=C_i(a)$ .

В момент получения этого сообщения процессом  $P_j$  его время корректируется следующим образом:

$$C_j = \max(C_j, tm+d)$$

# Логические часы. Lamport



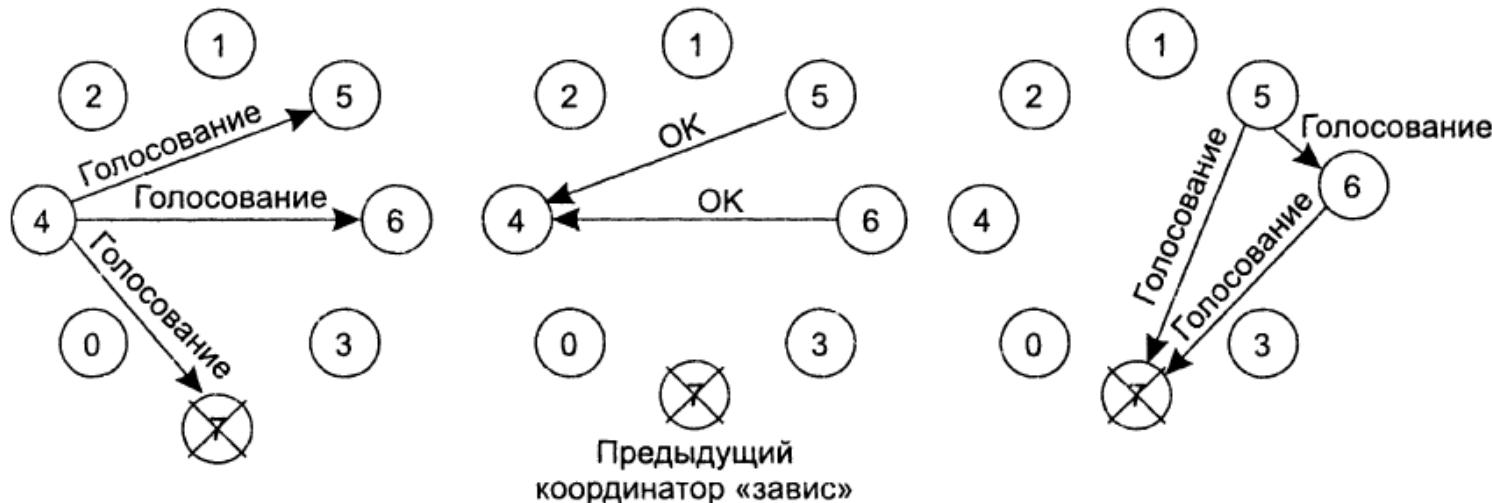
# Выборы координатора.

## Алгоритм задиры

- Если процесс обнаружит, что координатор очень долго не отвечает, то инициирует выборы. Процесс P проводит выборы следующим образом:
- P посылает сообщение «ВЫБОРЫ» всем процессам с большими чем у него номерами.
- Если нет ни одного ответа, то P считается победителем и становится координатором.
- Если один из процессов с большим номером ответит, то он берет на себя проведение выборов. Участие процесса P в выборах заканчивается.
- В любой момент процесс может получить сообщение «ВЫБОРЫ» от одного из коллег с меньшим номером. В этом случае он посылает ответ «OK», чтобы сообщить, что он жив и берет проведение выборов на себя, а затем начинает выборы (если к этому моменту он уже их не вел). Следовательно, все процессы прекратят выборы, кроме одного - нового координатора. Он извещает всех о своей победе и вступлении в должность сообщением «КООРДИНАТОР».
- Если процесс выключился из работы, а затем захотел восстановить свое участие, то он проводит выборы (отсюда и название алгоритма).

# Выборы координатора.

## Алгоритм задиры



а

б

в

г

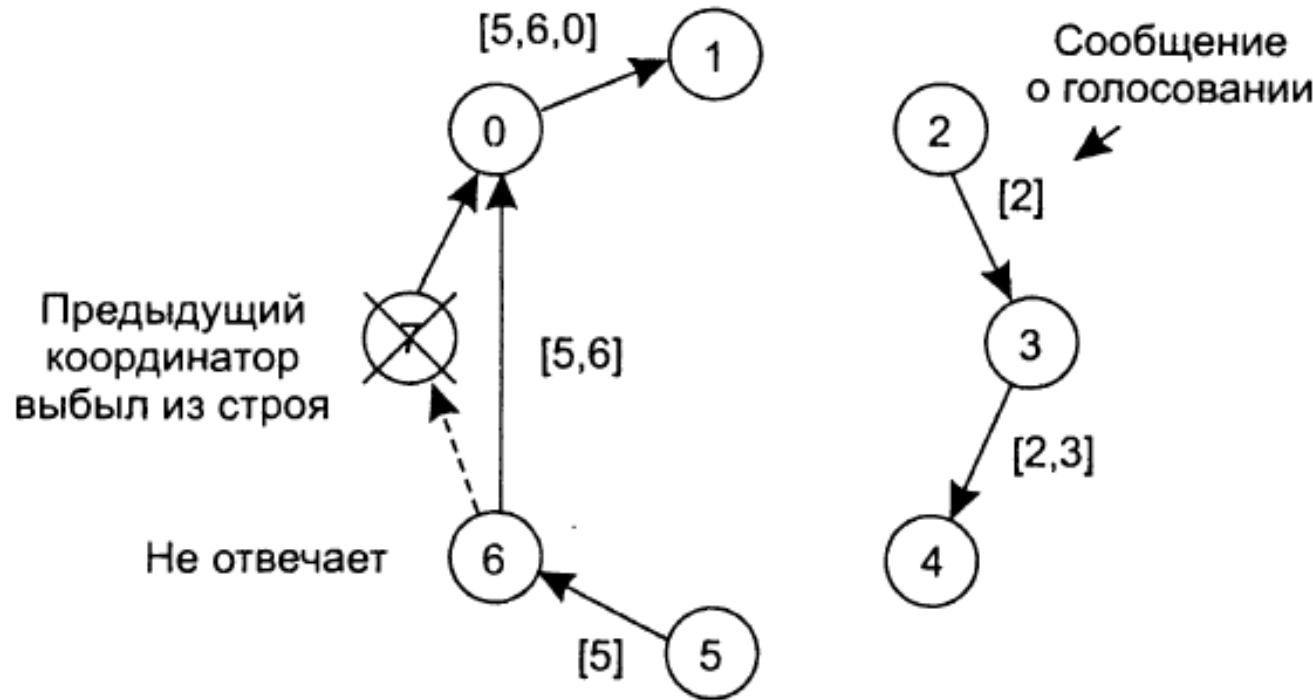
д

# Выборы координатора.

## Круговой алгоритм

- Алгоритм основан на использовании кольца (физического или логического).
- Каждый процесс знает следующего за ним в круговом списке. Когда процесс обнаруживает отсутствие координатора, он посыпает следующему за ним процессу сообщение «ВЫБОРЫ» со своим номером. Если следующий процесс не отвечает, то сообщение посыпается процессу, следующему за ним, и т.д., пока не найдется работающий процесс. Каждый работающий процесс добавляет в список работающих свой номер и переправляет сообщение дальше по кругу.
- Когда процесс обнаружит в списке свой собственный номер (круг пройден), он меняет тип сообщения на «КООРДИНАТОР» и оно проходит по кругу, извещая всех о списке работающих и координаторе (процессе с наибольшим номером в списке).
- После прохождения круга сообщение удаляется.

# Выборы координатора. Круговой алгоритм



# Взаимное исключение.

## Централизованный алгоритм

- Все процессы запрашивают у координатора разрешение на вход в критическую секцию и ждут этого разрешения. Координатор обслуживает запросы в порядке поступления.
- Получив разрешение процесс входит в критическую секцию. При выходе из нее он сообщает об этом координатору.
- Количество сообщений на одно прохождение критической секции - 3.
- Недостатки алгоритма - обычные недостатки централизованного алгоритма (крах координатора или его перегрузка сообщениями).

# Децентрализованный алгоритм на основе временных меток

## *Вход в критическую секцию*

- Когда процесс желает войти в критическую секцию, он посыпает всем процессам сообщение-запрос, содержащее имя критической секции, номер процесса и текущее время.
- После посылки запроса процесс ждет, пока все дадут ему разрешение. После получения от всех разрешения, он входит в критическую секцию.

## *Поведение процесса при приеме запроса*

- Когда процесс получает сообщение-запрос, в зависимости от своего состояния по отношению к указанной критической секции он действует одним из следующих способов.
- Если получатель не находится внутри критической секции и не запрашивал разрешение на вход в нее, то он посыпает отправителю сообщение «OK».
- Если получатель находится внутри критической секции, то он не отвечает, а запоминает запрос.
- Если получатель выдал запрос на вхождение в эту секцию, но еще не вошел в нее, то он сравнивает временные метки своего запроса и чужого. Побеждает тот, чья метка меньше. Если чужой запрос победил, то процесс посыпает сообщение «OK». Если у чужого запроса метка больше, то ответ не посыпается, а чужой запрос запоминается.

# Децентрализованный алгоритм на основе временных меток

## *Выход из критической секции*

- После выхода из секции он посыпает сообщение «OK» всем процессам, запросы от которых он запомнил, а затем стирает все запомненные запросы.
- Количество сообщений на одно прохождение секции -  $2(n-1)$ , где  $n$  - число процессов.
- Кроме того, одна критическая точка заменилась на  $n$  точек (если какой-то процесс перестанет функционировать, то отсутствие разрешения от него всех остановит).
- И, наконец, если в централизованном алгоритме есть опасность перегрузки координатора, то в этом алгоритме перегрузка любого процесса приведет к тем же последствиям.
- Некоторые улучшения алгоритма (например, ждать разрешения не от всех, а от большинства) требуют наличия неделимых широковещательных рассылок сообщений.

Алгоритм носит имя Ricart-Agrawala. Требуется глобальное упорядочение всех событий в системе по времени.

# Алгоритм с круговым маркером

Все процессы составляют логическое кольцо, когда каждый знает, кто следует за ним.

По кольцу циркулирует маркер, дающий право на вход в критическую секцию.

Получив маркер (посредством сообщения точка-точка) процесс либо входит в критическую секцию (если он ждал разрешения) либо переправляет маркер дальше.

После выхода из критической секции маркер переправляется дальше, повторный вход в секцию при том же маркере не разрешается.

# Алгоритм широковещательный маркерный (Suzuki-Kasami)

Маркер содержит:

- очередь запросов;
- массив  $LN[1\dots N]$  с номерами последних удовлетворенных запросов.

## ***Вход в критическую секцию***

- Если процесс  $P_k$ , запрашивающий критическую секцию, не имеет маркера, то он увеличивает порядковый номер своих запросов  $RN_k[k]$  и посыпает широковещательно сообщение «ЗАПРОС», содержащее номер процесса ( $k$ ) и номер запроса ( $S_n = RN_k[k]$ ).
- Процесс  $P_k$  выполняет критическую секцию, если имеет (или когда получит) маркер.

# Алгоритм широковещательный маркерный (Suzuki-Kasami)

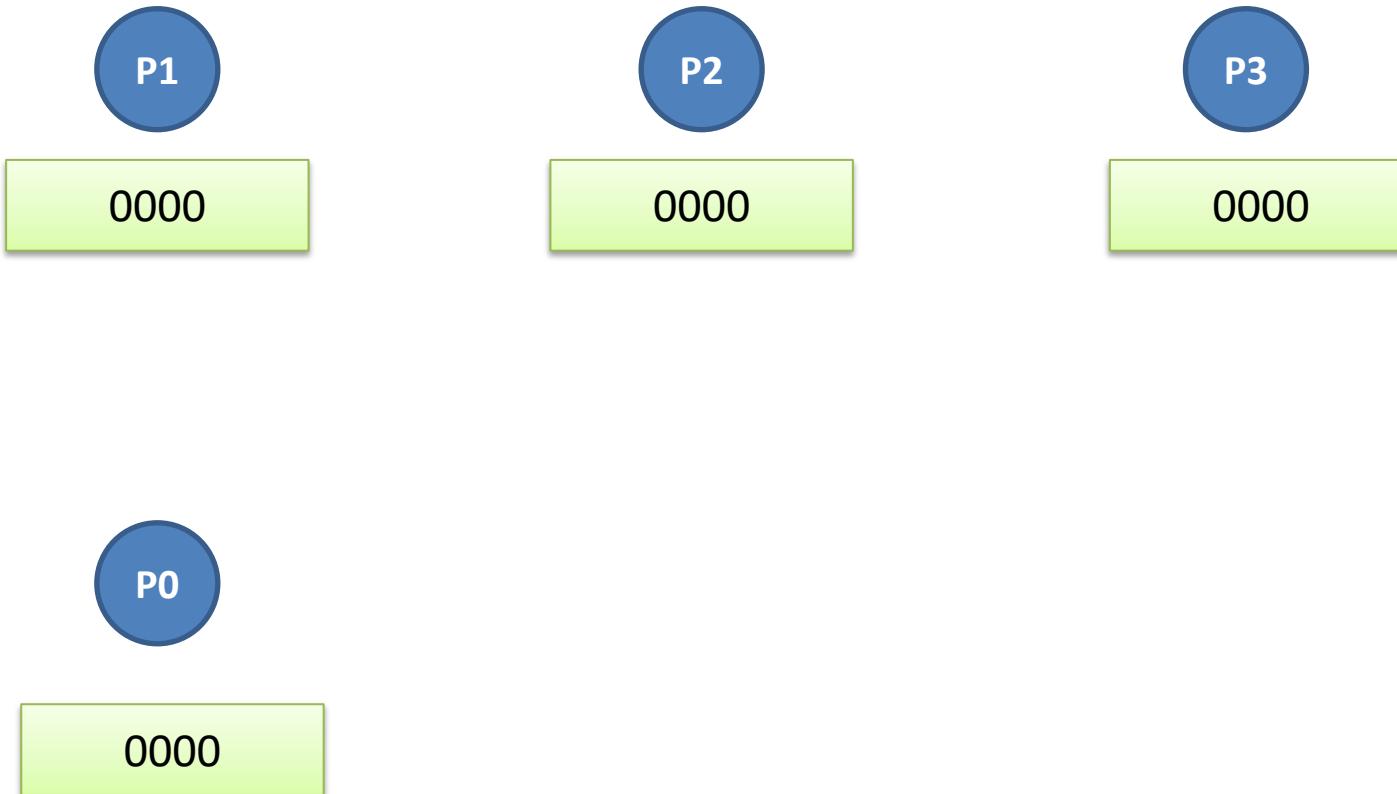
## *Поведение процесса при приеме запроса*

- Когда процесс  $P_j$  получит сообщение-запрос от процесса  $P_k$ , он устанавливает  $RN_j[k] = \max(RN_j[k], S_n)$ .
- Если  $P_j$  имеет свободный маркер, то он его посыпает  $P_k$  только в том случае, когда  $RN_j[k] == LN[k]+1$  (запрос не старый).

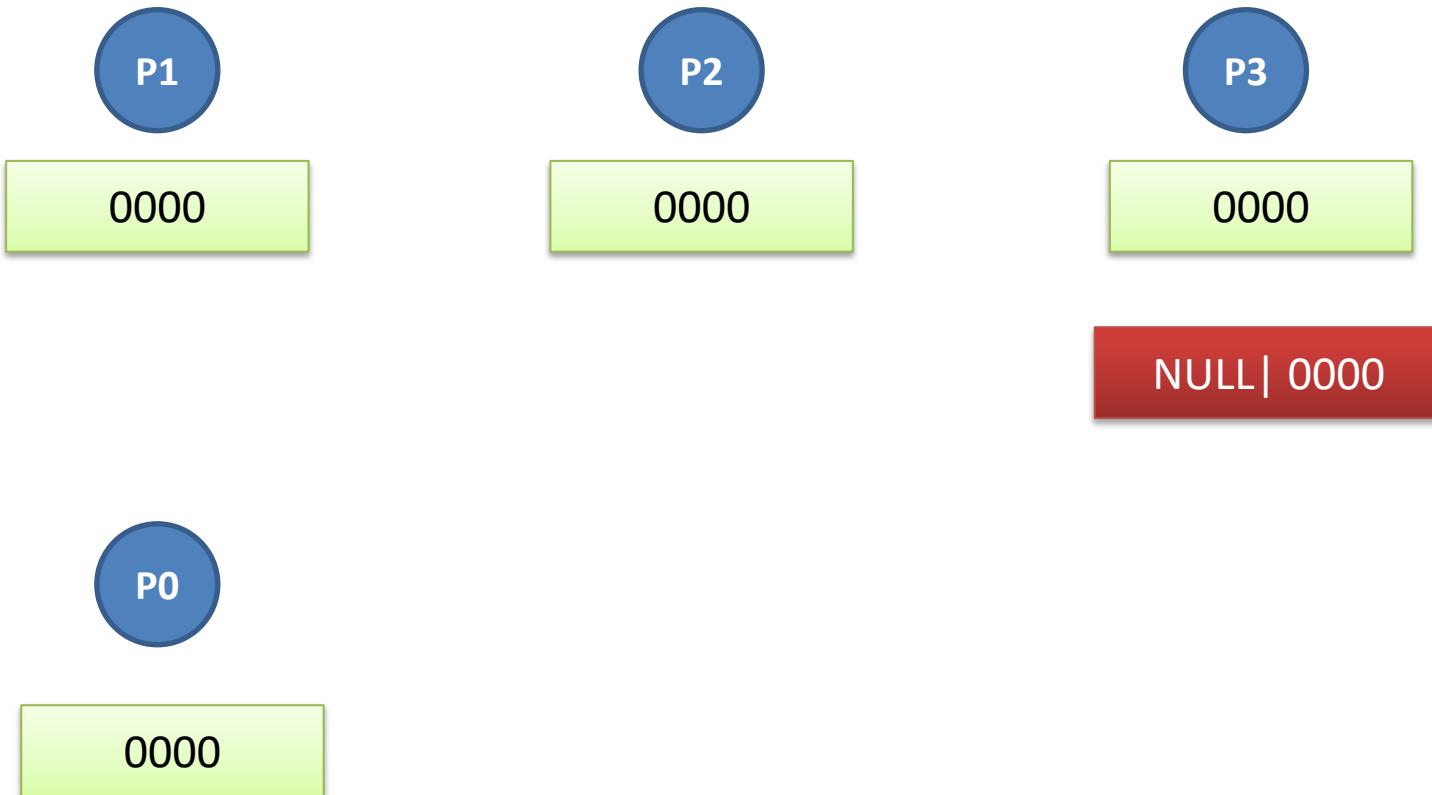
## *Выход из критической секции процесса $P_k$ .*

- Устанавливает  $LN[k]$  в маркере равным  $RN_k[k]$ .
- Для каждого  $P_j$ , для которого  $RN_k[j] = LN[j]+1$ , он добавляет его идентификатор в маркерную очередь запросов (если там его еще нет).
- Если маркерная очередь запросов не пуста, то из нее удаляется первый элемент, а маркер посыпается соответствующему процессу (запрос которого был первым в очереди).

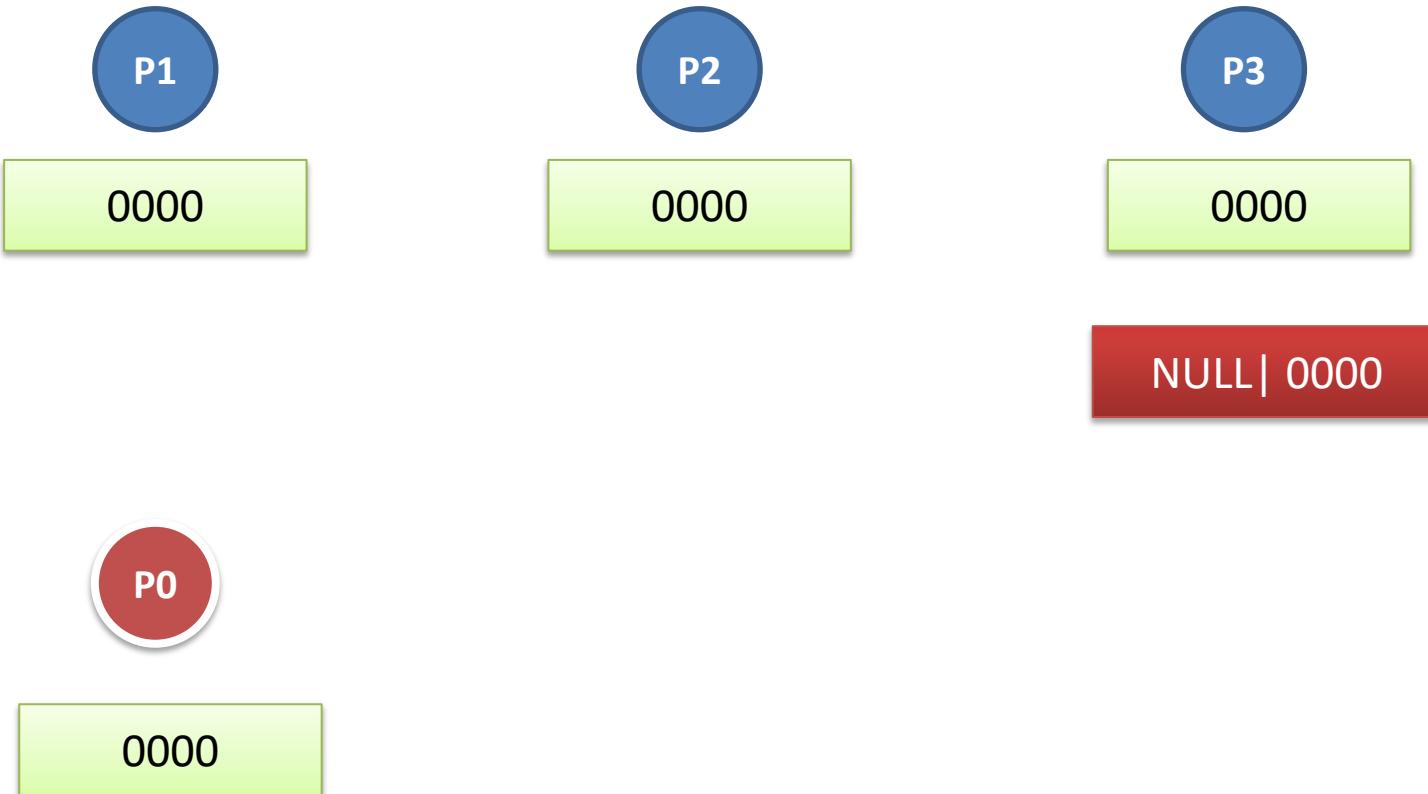
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



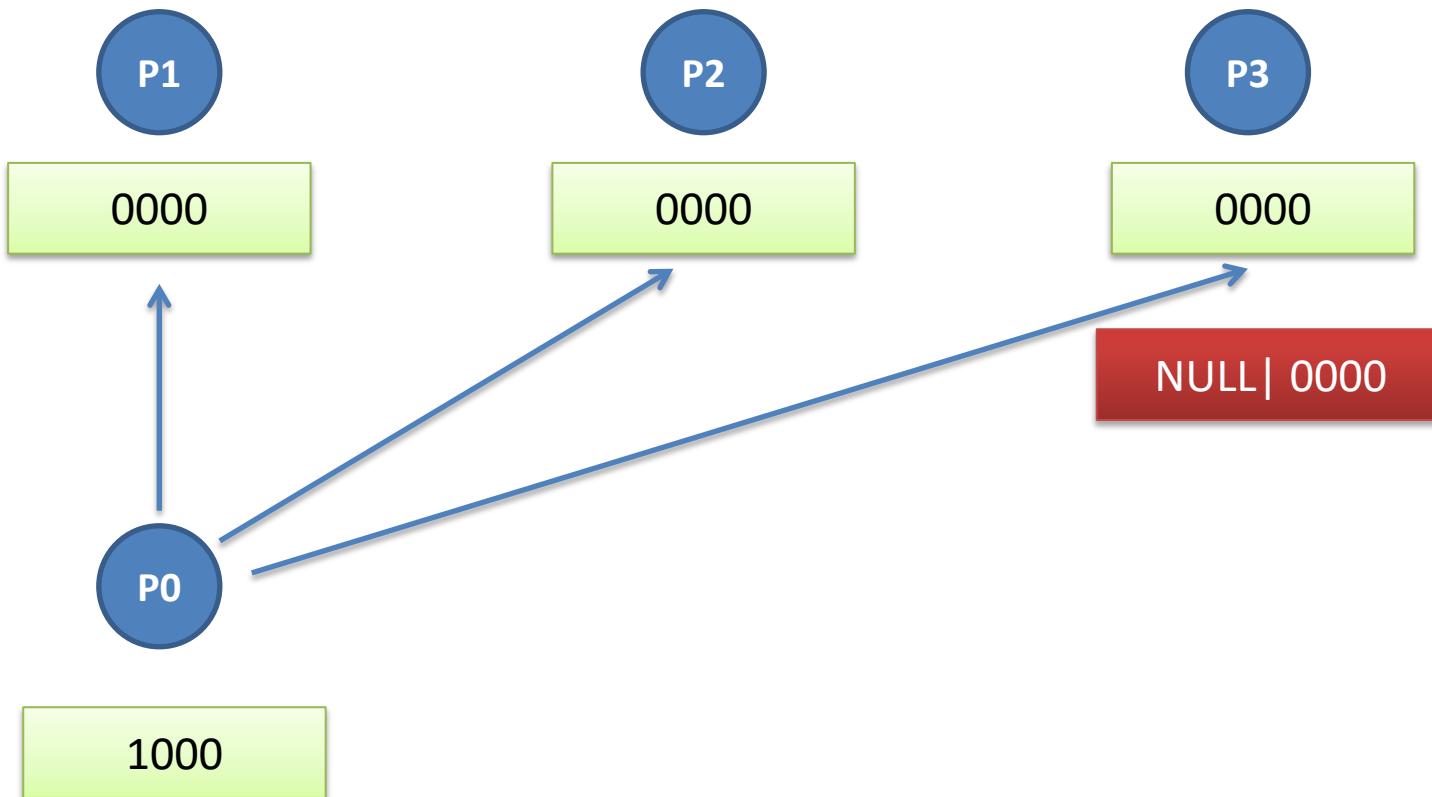
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



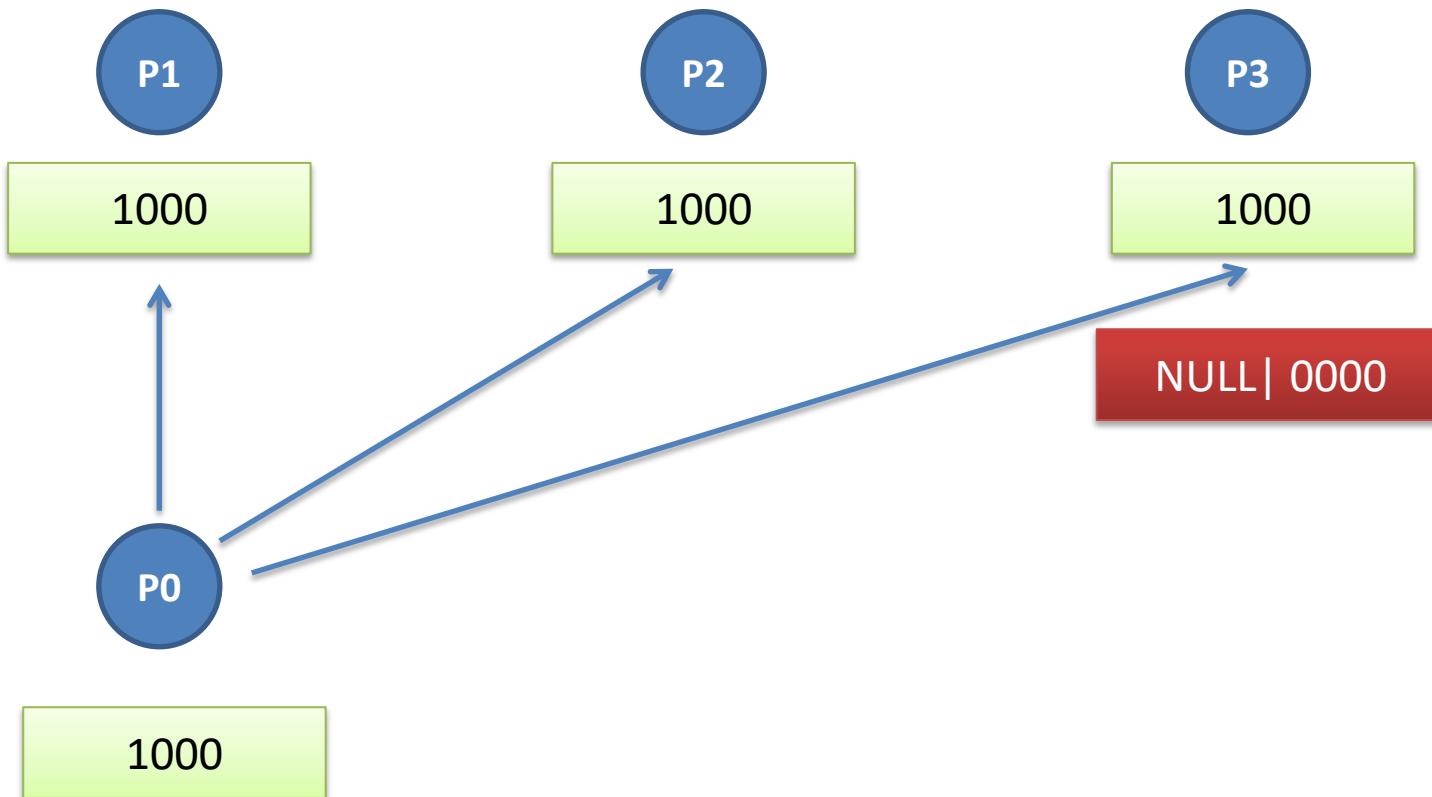
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



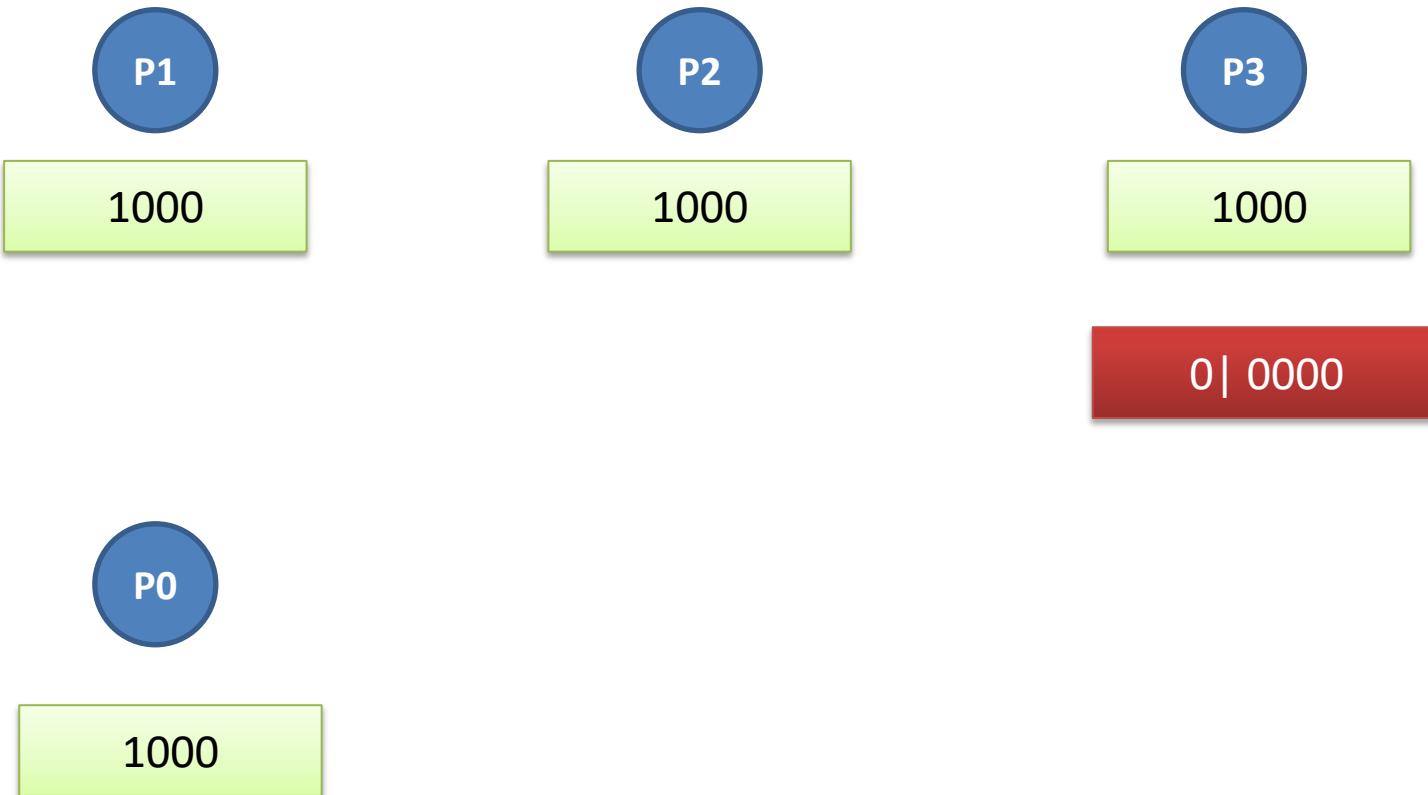
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



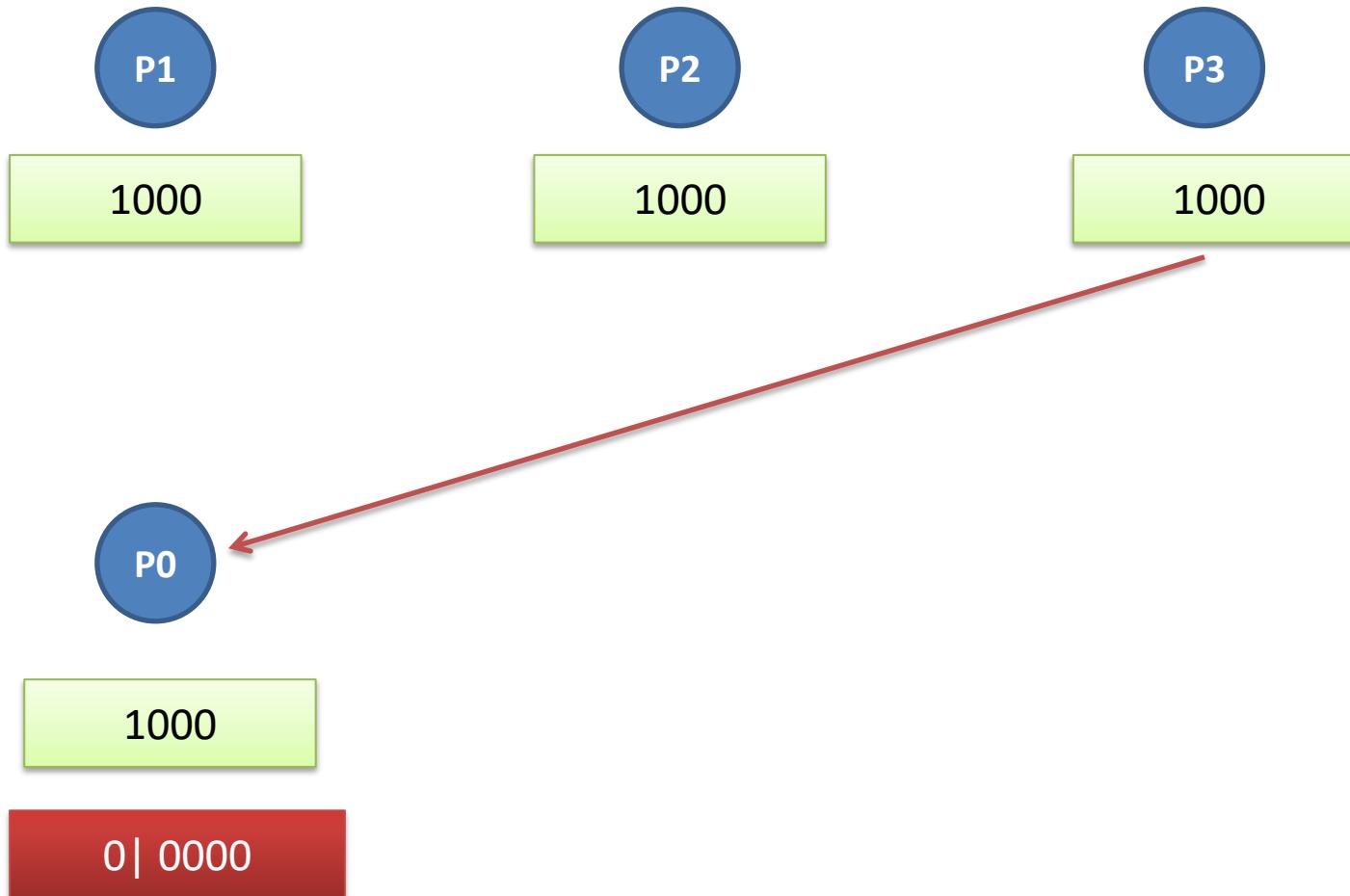
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



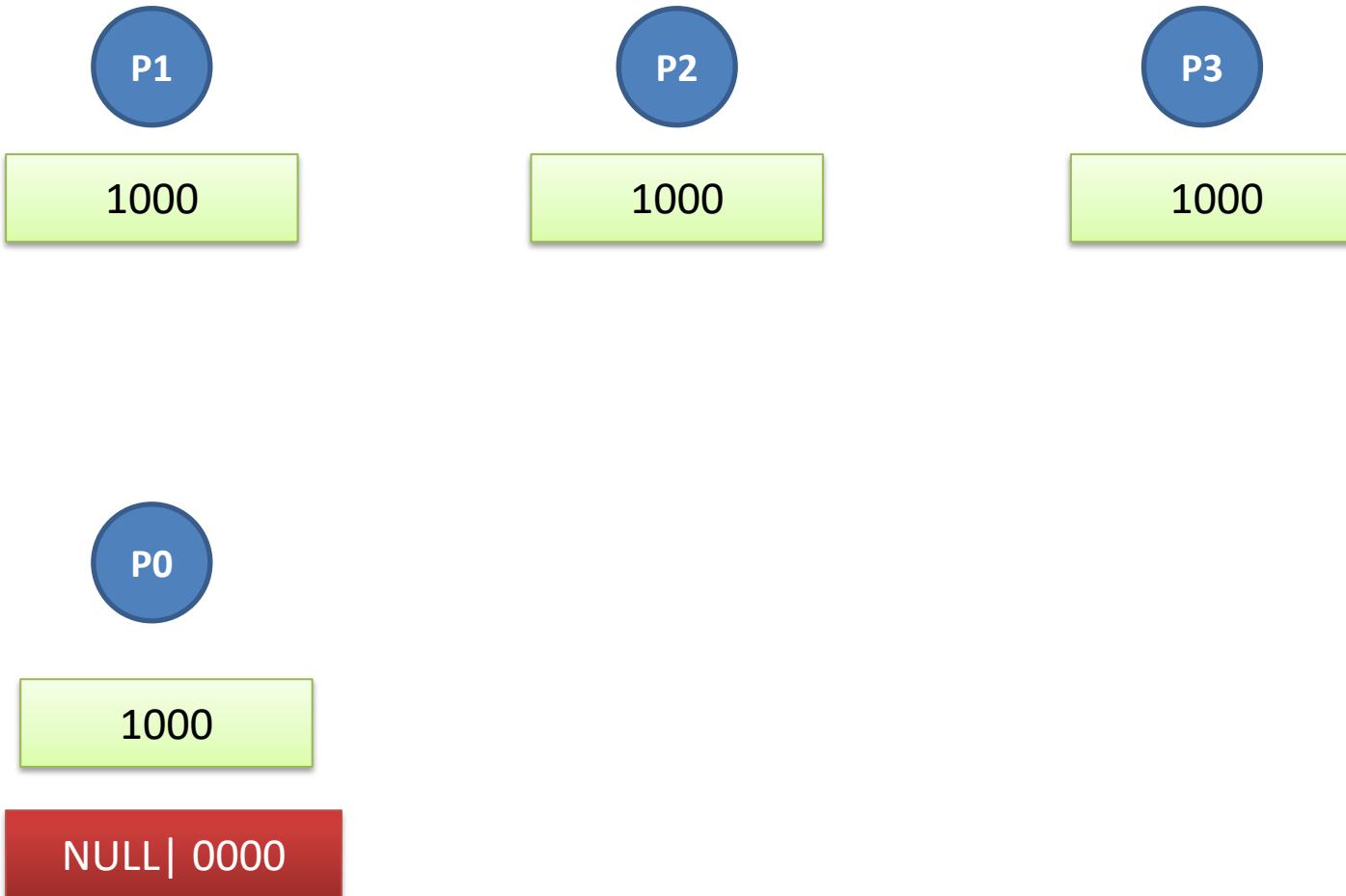
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



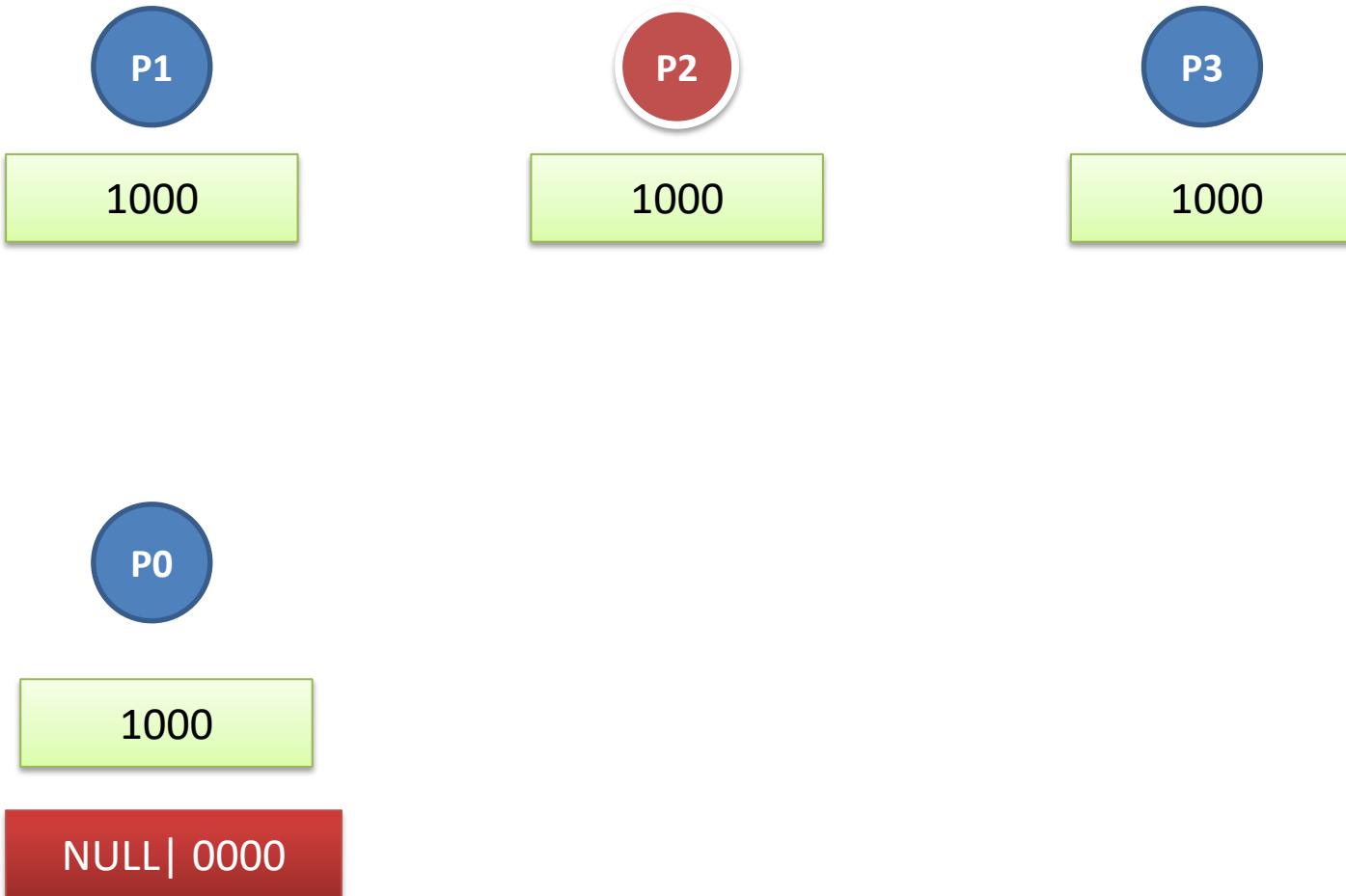
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



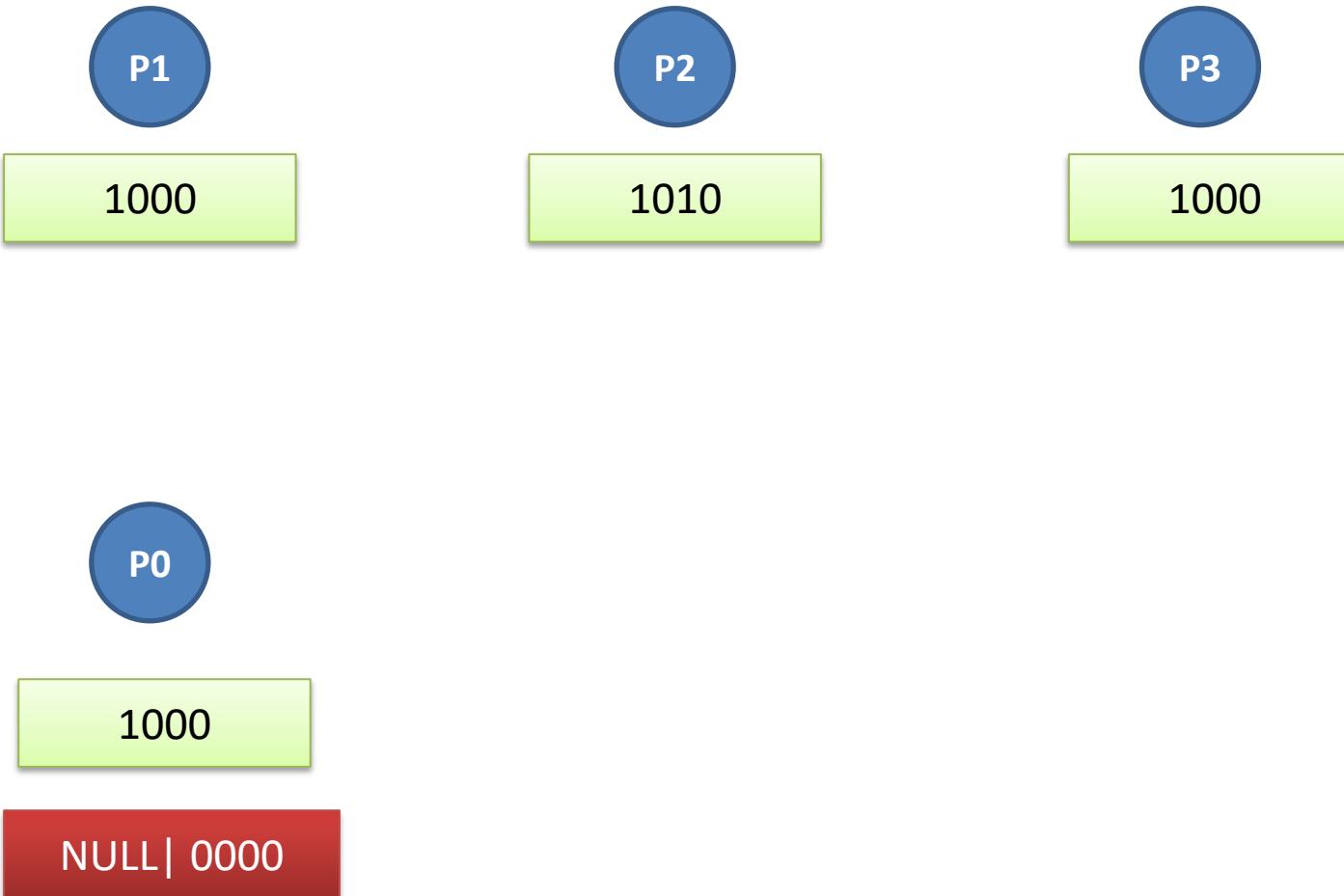
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



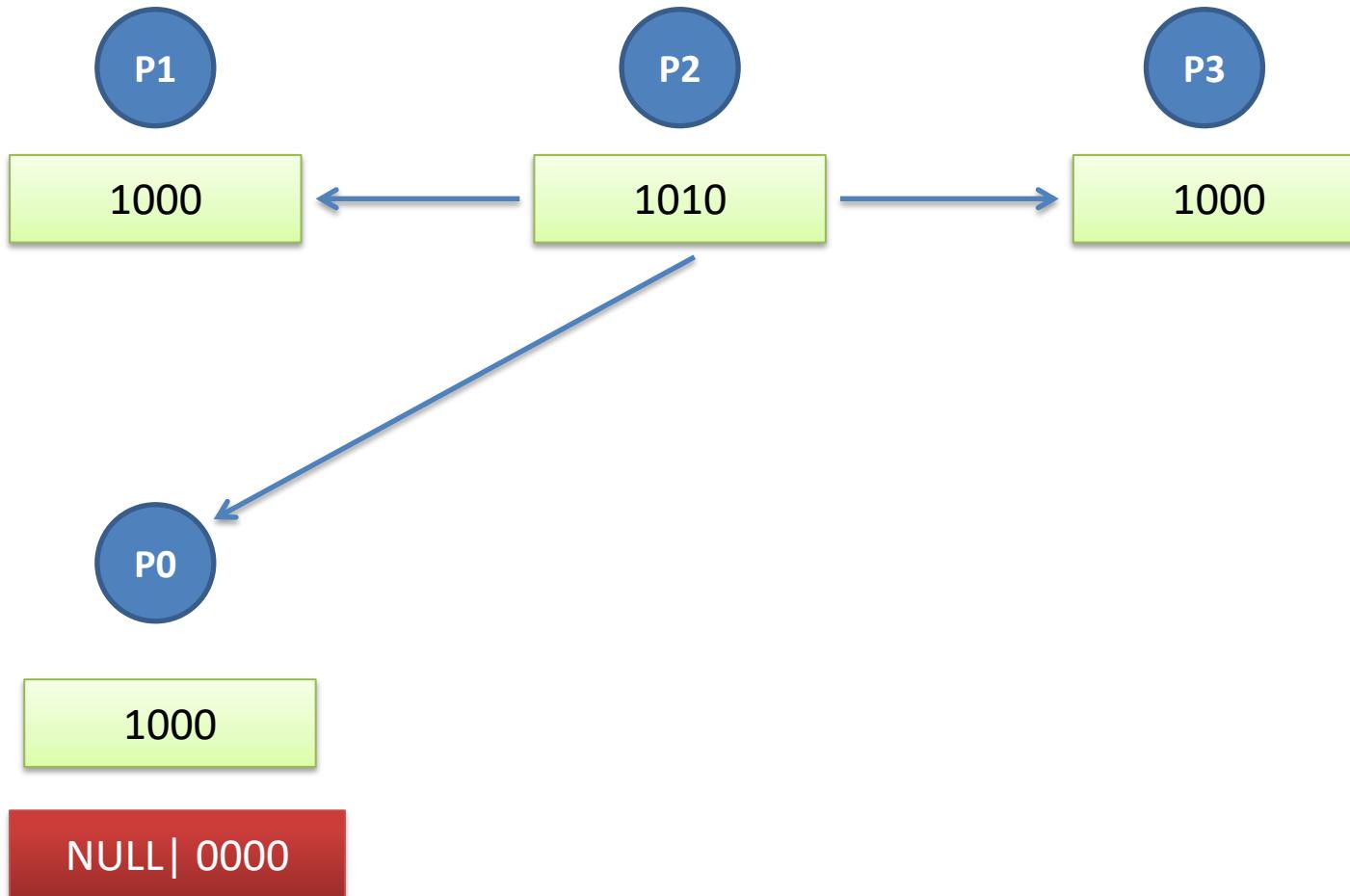
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



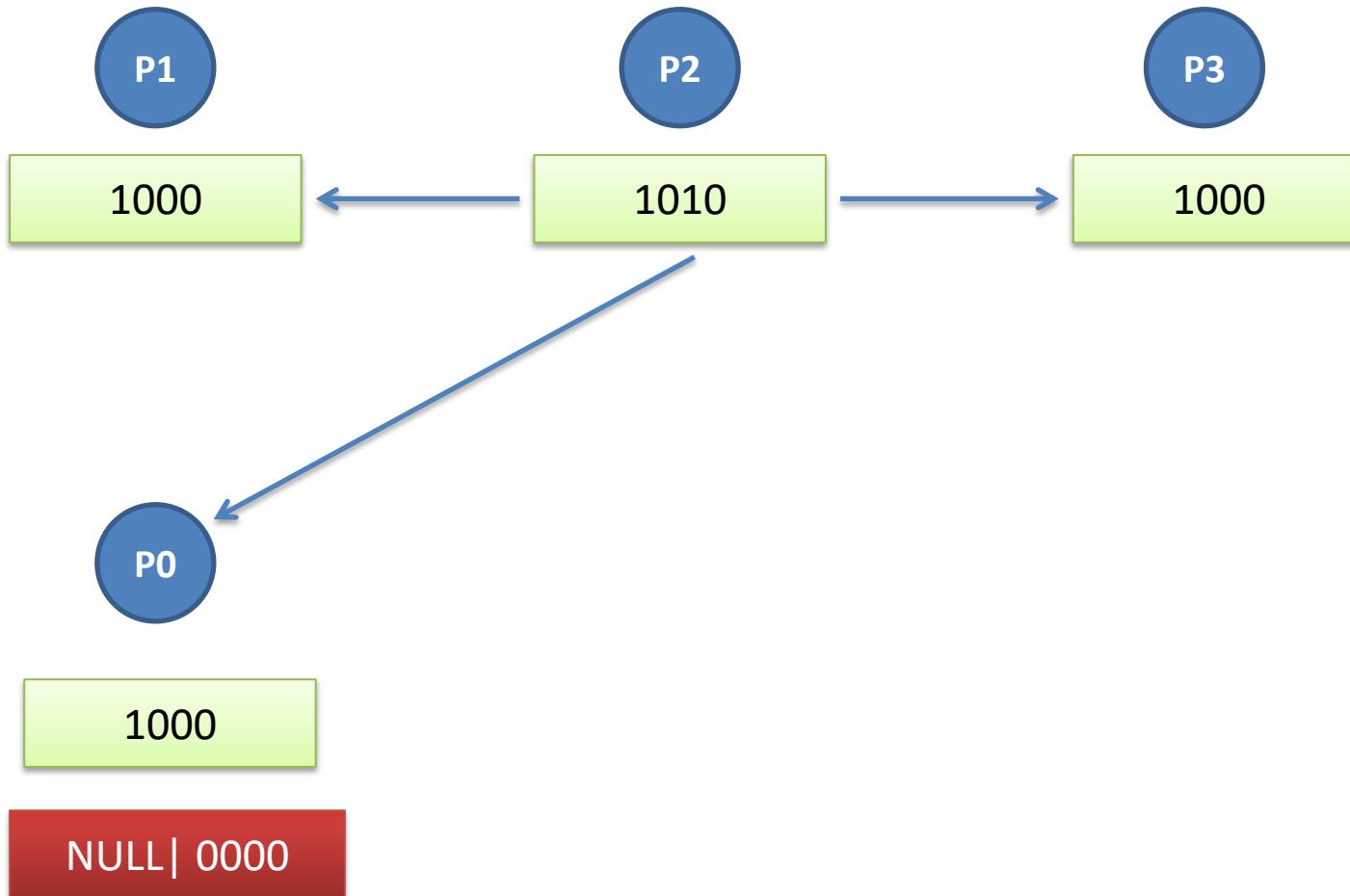
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



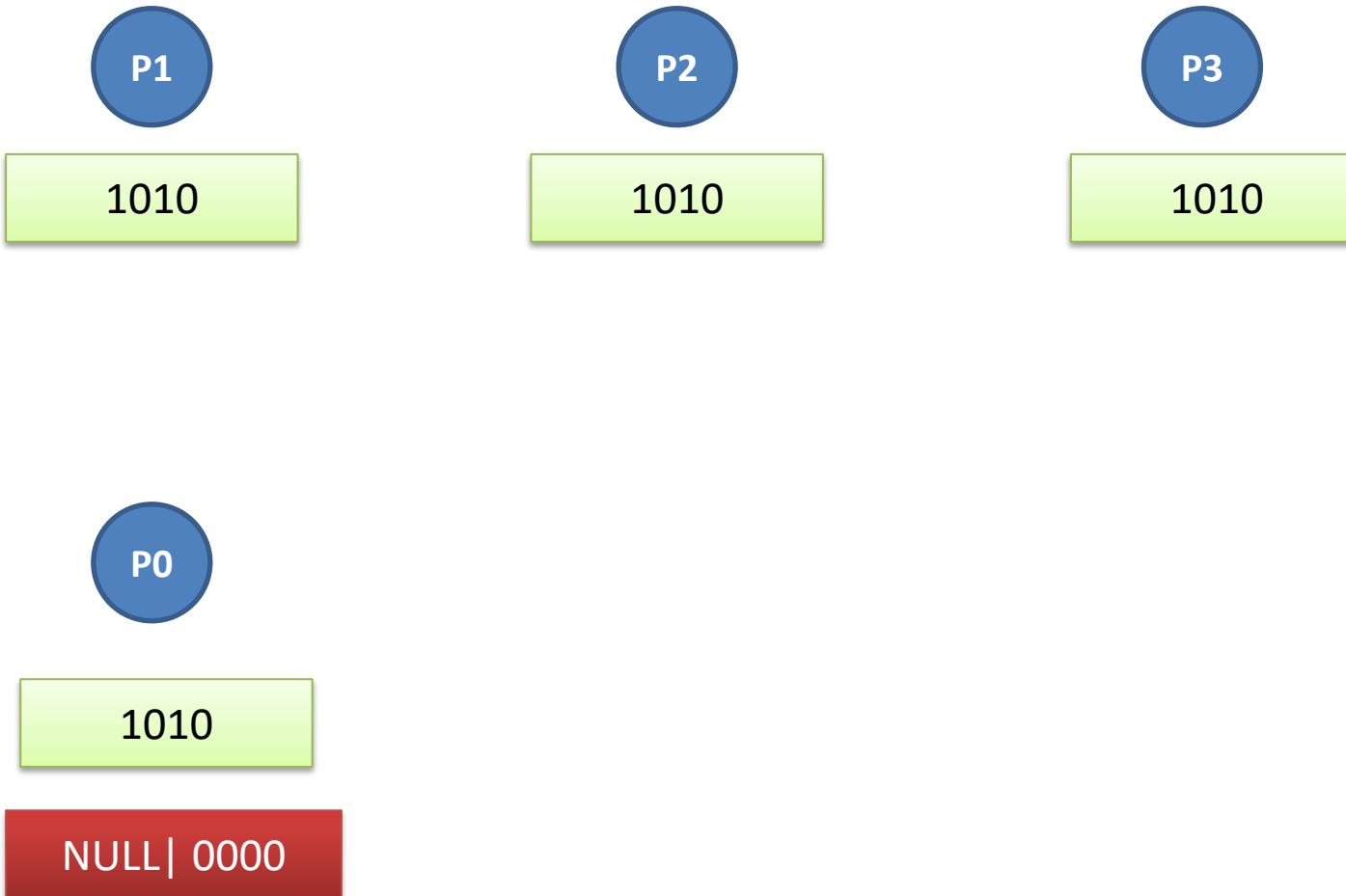
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



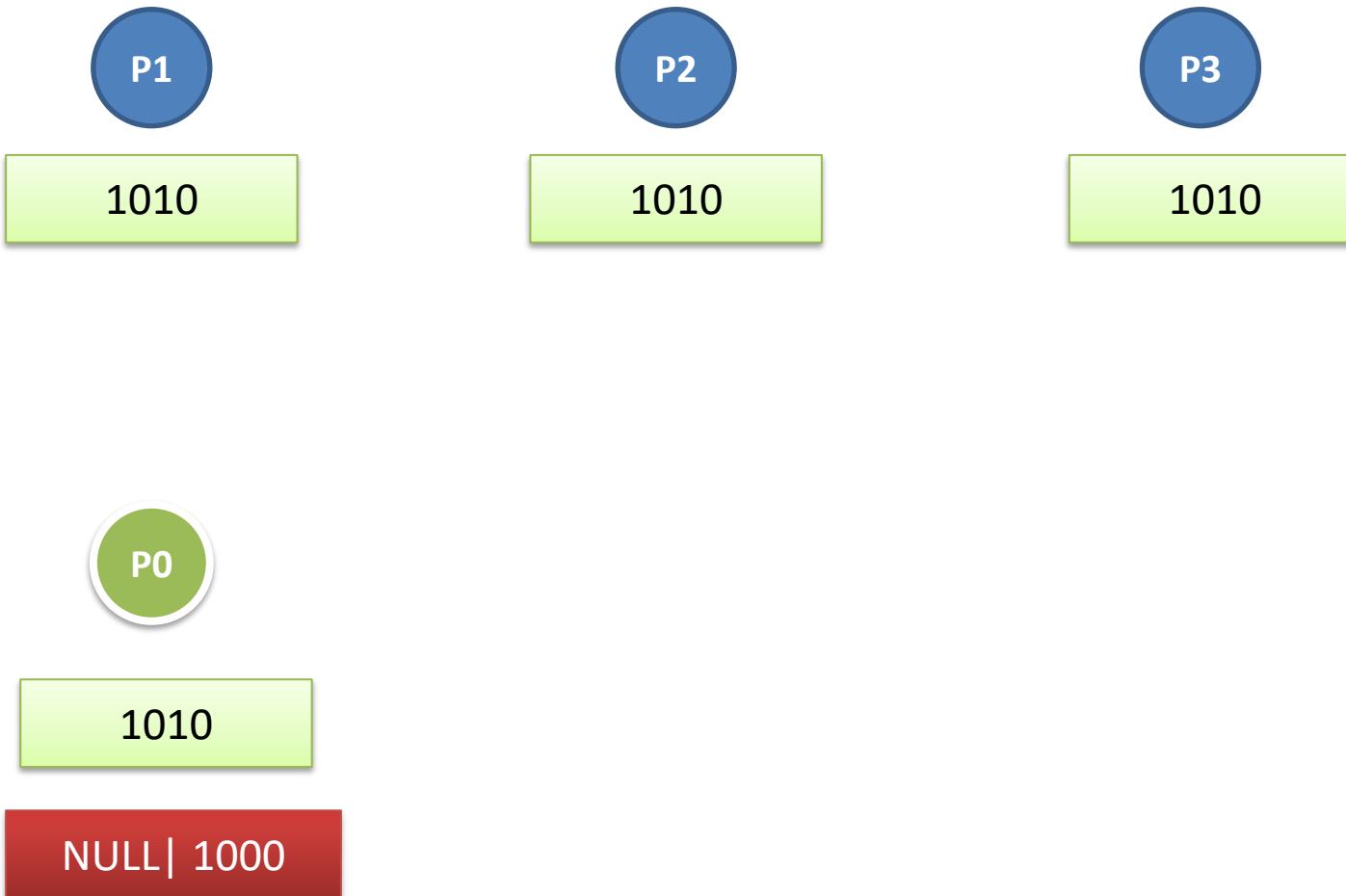
# Алгоритм широковещательный маркерный (Suzuki-Kasami)



# Алгоритм широковещательный маркерный (Suzuki-Kasami)



# Алгоритм широковещательный маркерный (Suzuki-Kasami)



# Алгоритм широковещательный маркерный (Suzuki-Kasami)

P1

1010

P2

1010

P3

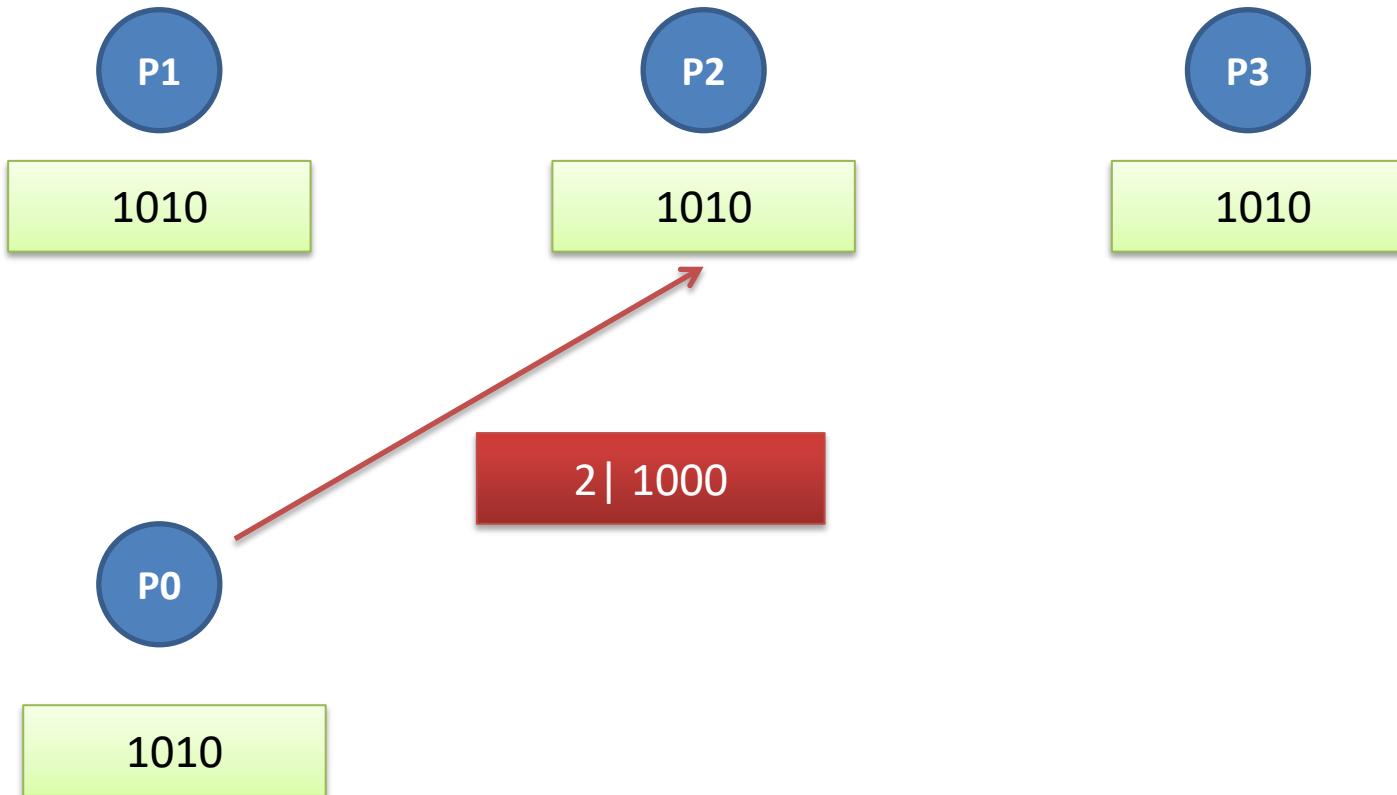
1010

P0

1010

2| 1000

# Алгоритм широковещательный маркерный (Suzuki-Kasami)



# Алгоритм древовидный маркерный (Raymond)

Все процессы представлены в виде сбалансированного двоичного дерева. Каждый процесс имеет очередь запросов от себя и соседних процессов (1-го, 2-х или 3-х) и указатель в направлении владельца маркера.

## *Вход в критическую секцию*

- Если есть маркер, то процесс выполняет КС.
- Если нет маркера, то процесс:
  - 1) помещает свой запрос в очередь запросов;
  - 2) посылает сообщение «ЗАПРОС» в направлении владельца маркера и ждет сообщений.

# Алгоритм древовидный маркерный (Raymond)

## *Поведение процесса при приеме сообщений*

Процесс, не находящийся внутри КС должен реагировать на сообщения двух видов -«МАРКЕР» и «ЗАПРОС».

### **А) Пришло сообщение «МАРКЕР»:**

- М1. Взять 1-ый запрос из очереди и послать маркер его автору (концептуально, возможно себе).
- М2. Поменять значение указателя в сторону маркера.
- М3. Исключить запрос из очереди.
- М4. Если в очереди остались запросы, то послать сообщение «ЗАПРОС» в сторону маркера.

### **Б) Пришло сообщение «ЗАПРОС»:**

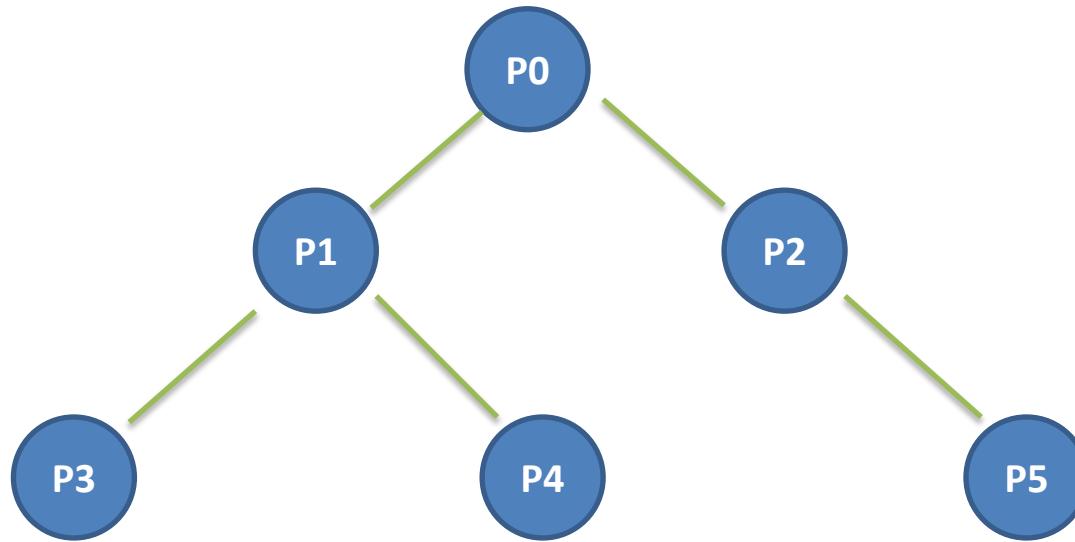
- Поместить запрос в очередь.
- Если нет маркера, то послать сообщение «ЗАПРОС» в сторону маркера, иначе (если есть маркер) - перейти на пункт М1.

# Алгоритм древовидный маркерный (Raymond)

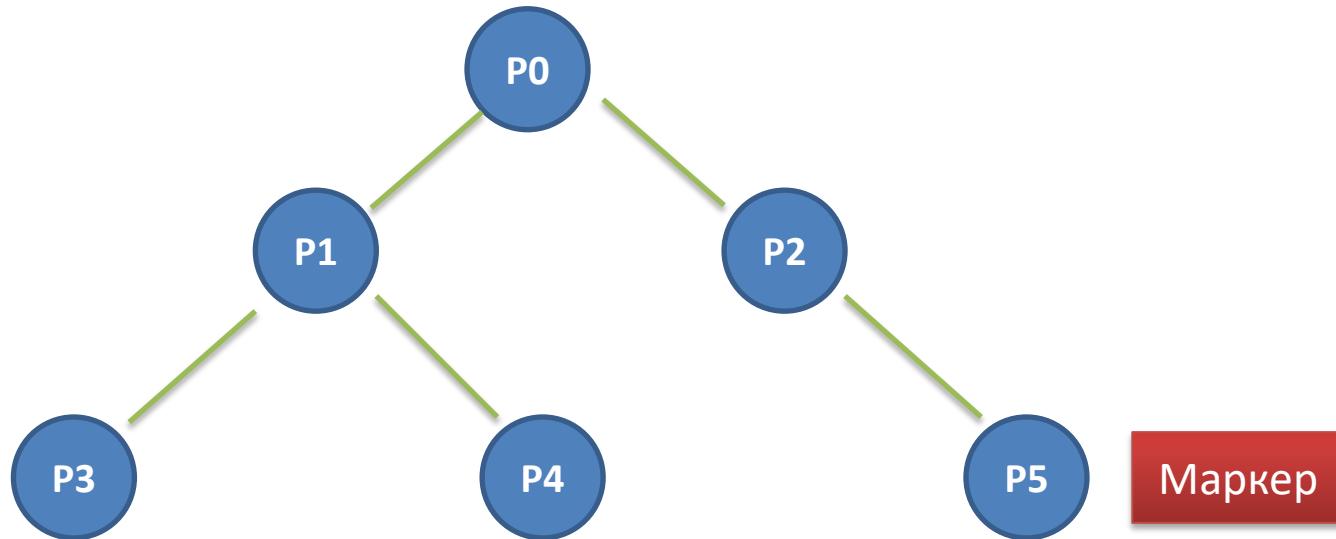
## *Выход из критической секции*

- Если очередь запросов пуста, то при выходе ничего не делается, иначе - перейти к пункту М1.

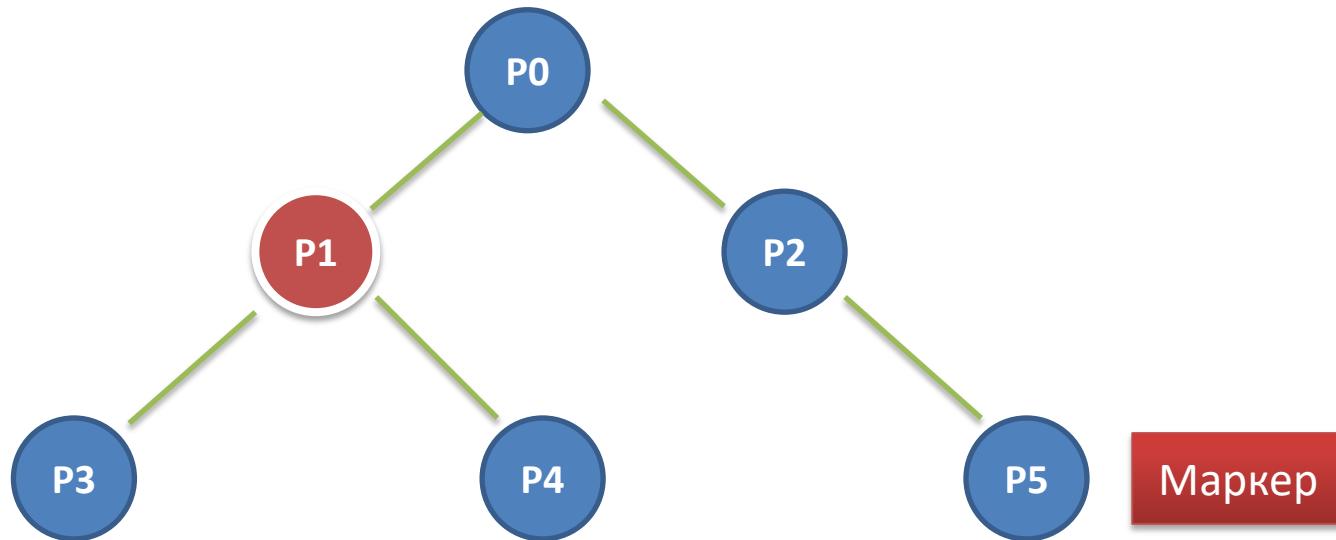
# Алгоритм древовидный маркерный (Raymond)



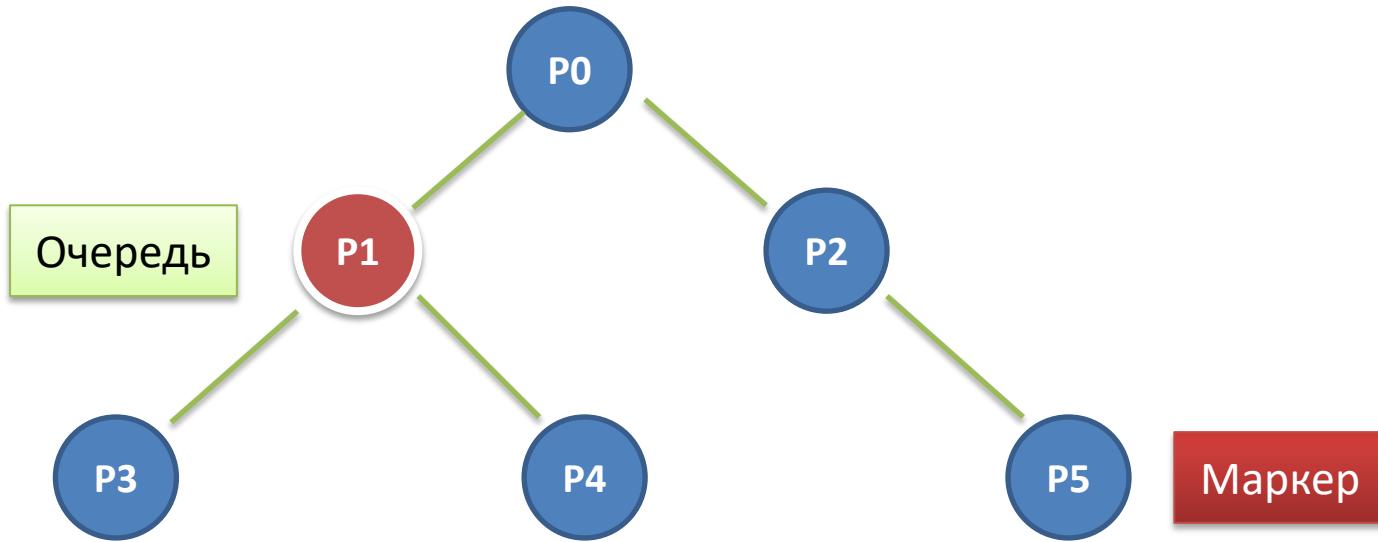
# Алгоритм древовидный маркерный (Raymond)



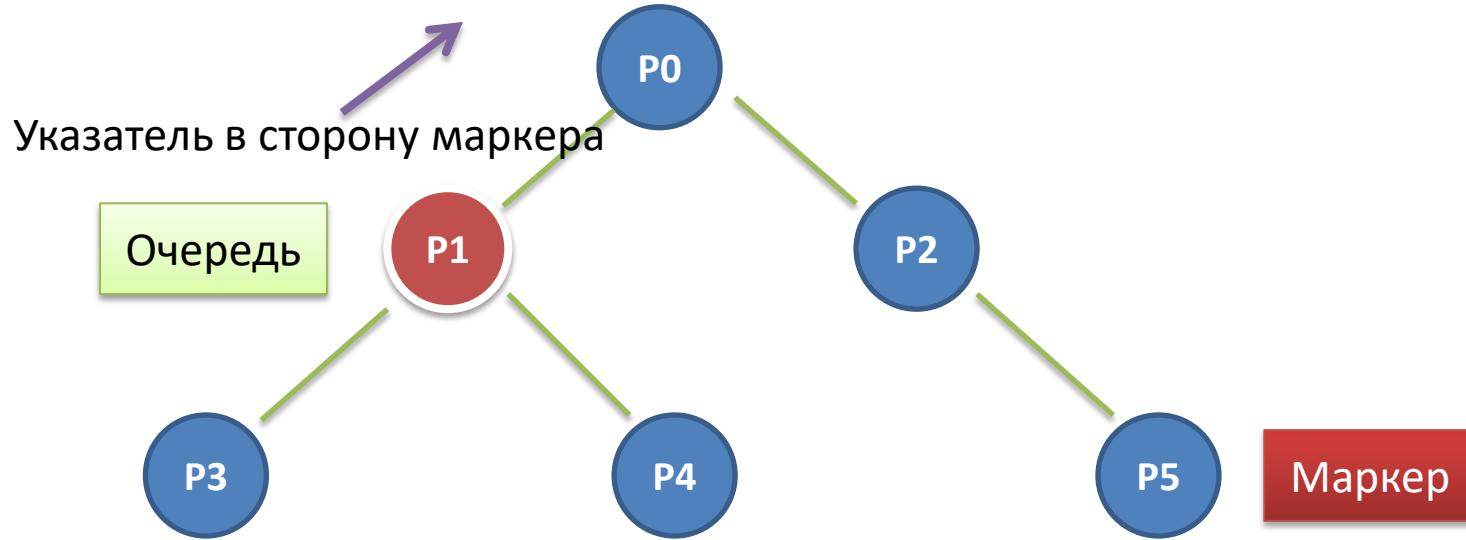
# Алгоритм древовидный маркерный (Raymond)



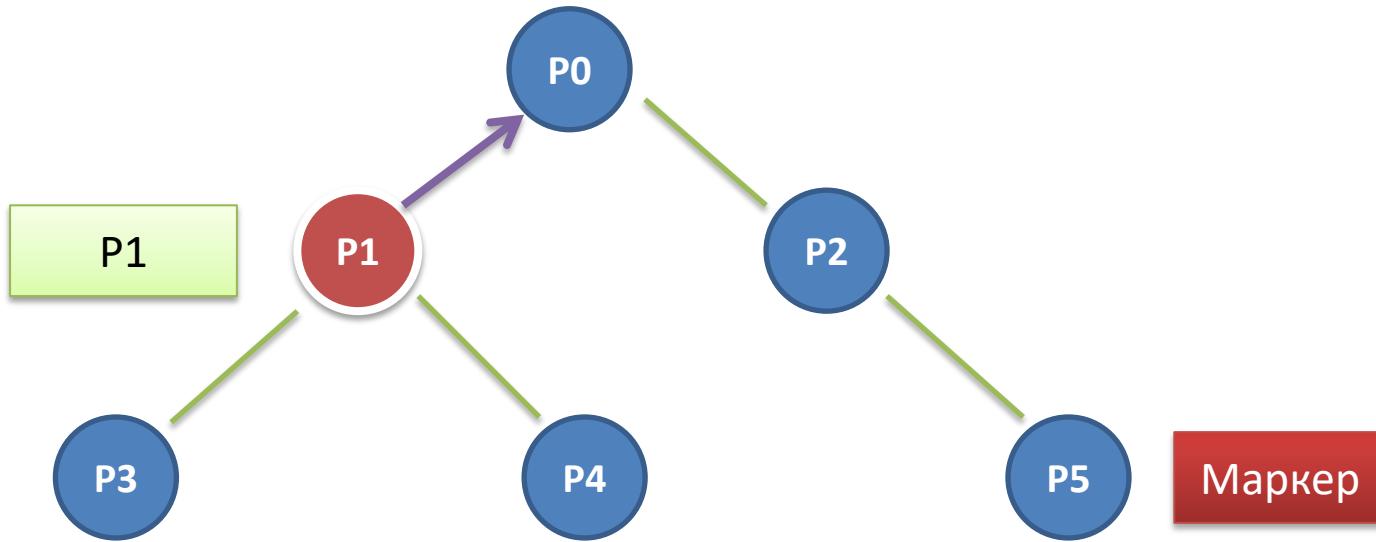
# Алгоритм древовидный маркерный (Raymond)



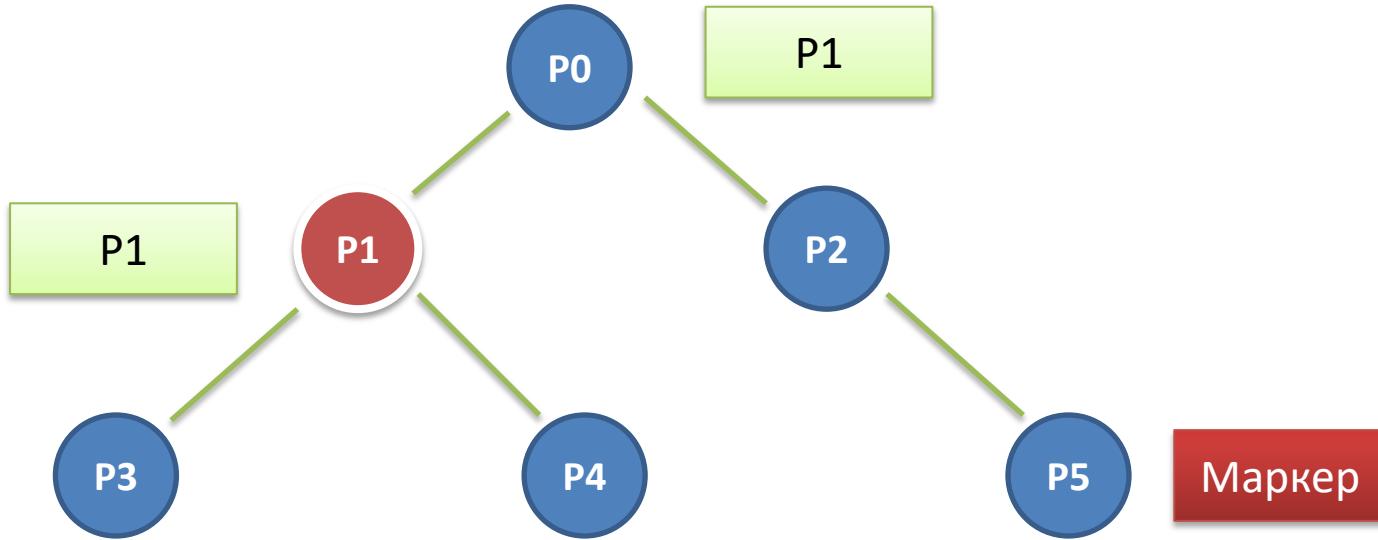
# Алгоритм древовидный маркерный (Raymond)



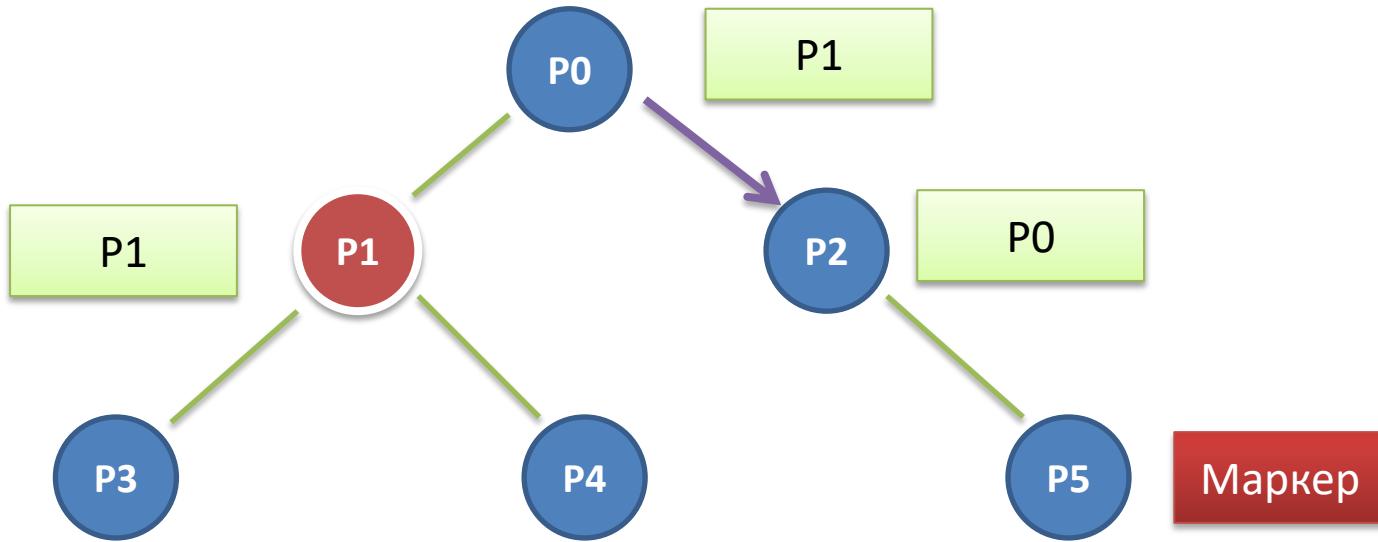
# Алгоритм древовидный маркерный (Raymond)



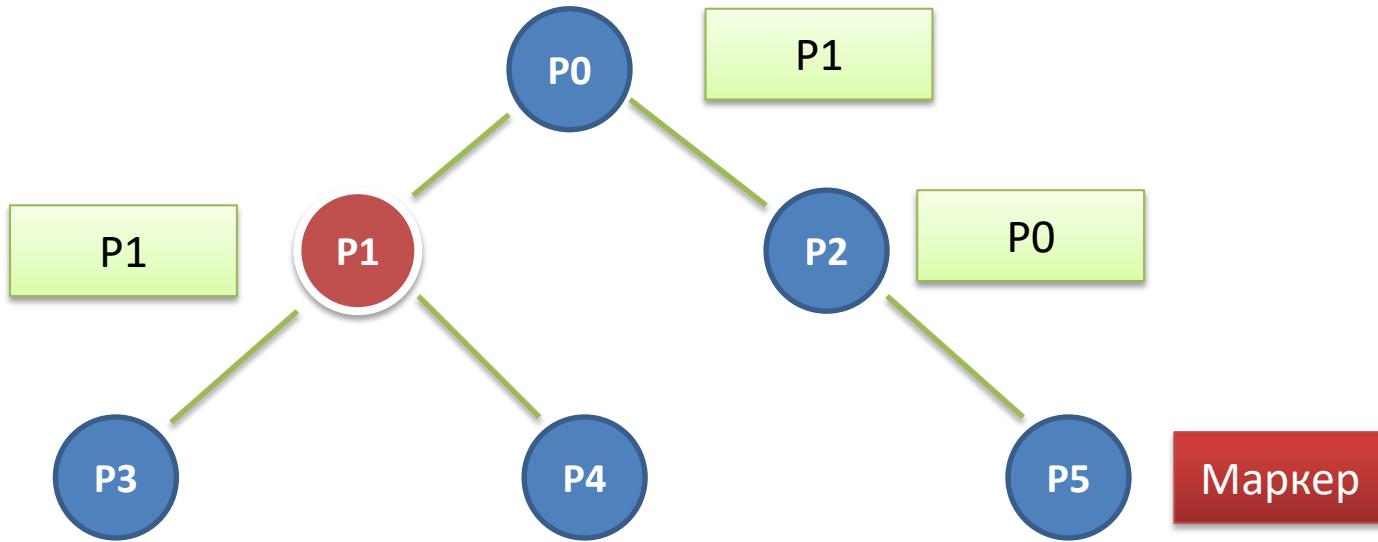
# Алгоритм древовидный маркерный (Raymond)



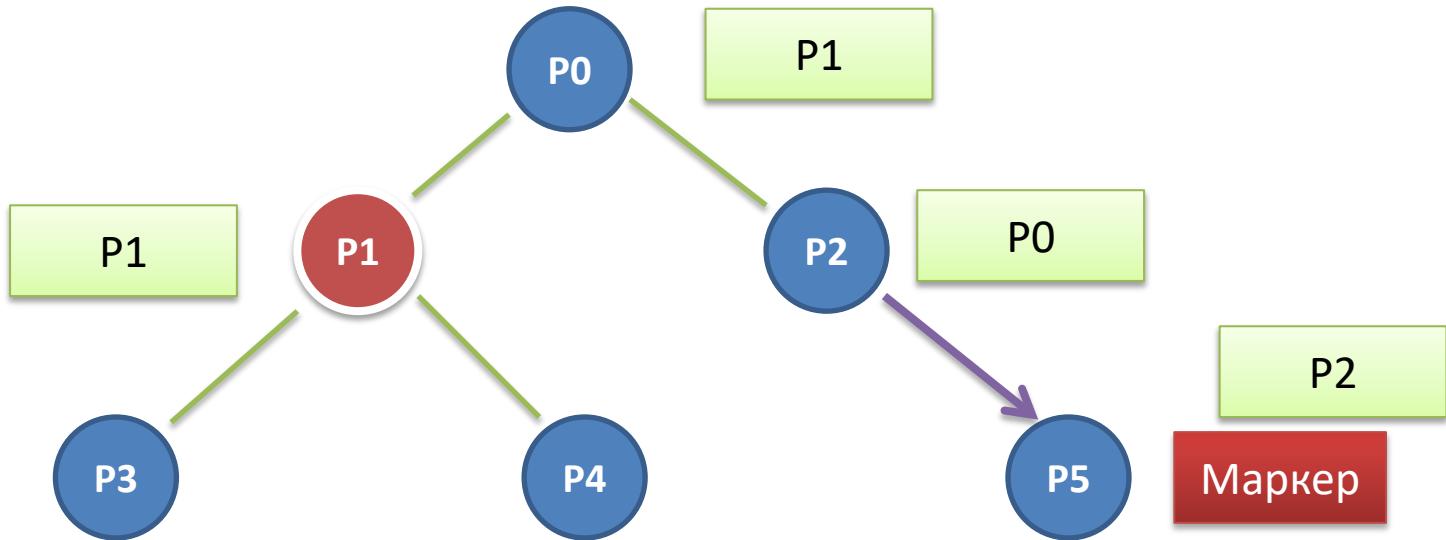
# Алгоритм древовидный маркерный (Raymond)



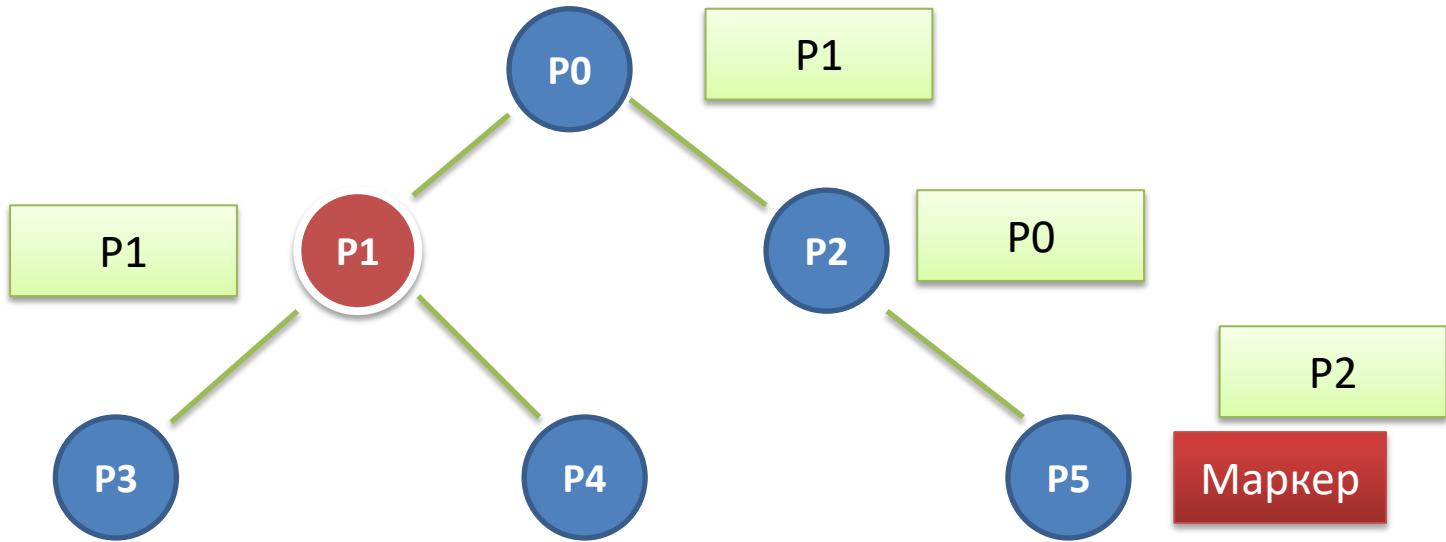
# Алгоритм древовидный маркерный (Raymond)



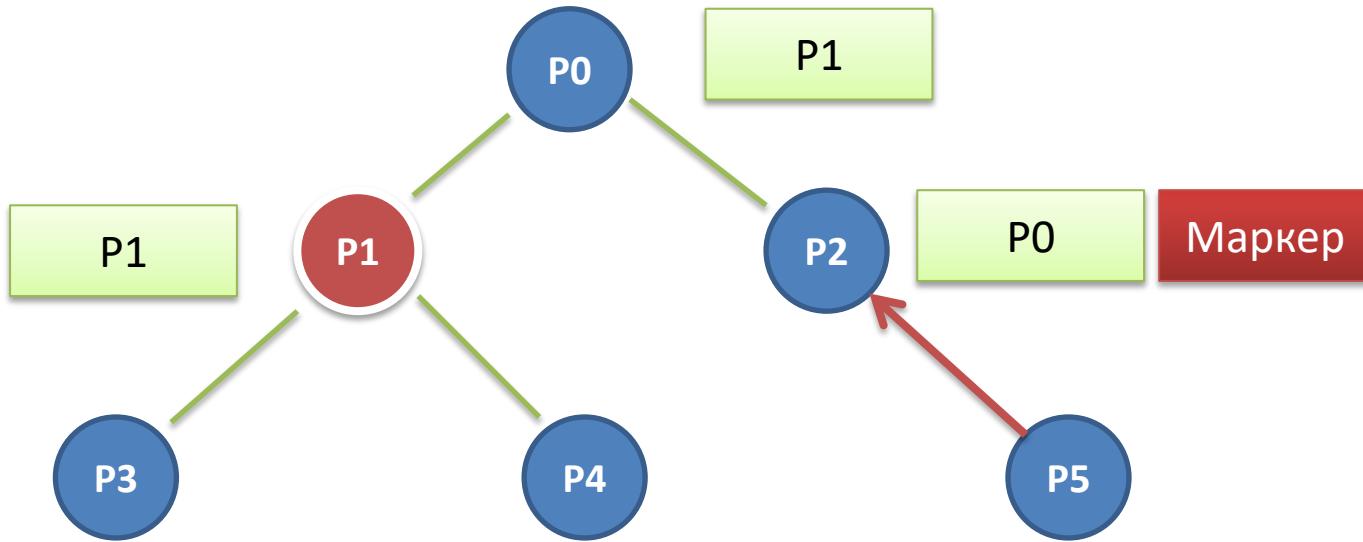
# Алгоритм древовидный маркерный (Raymond)



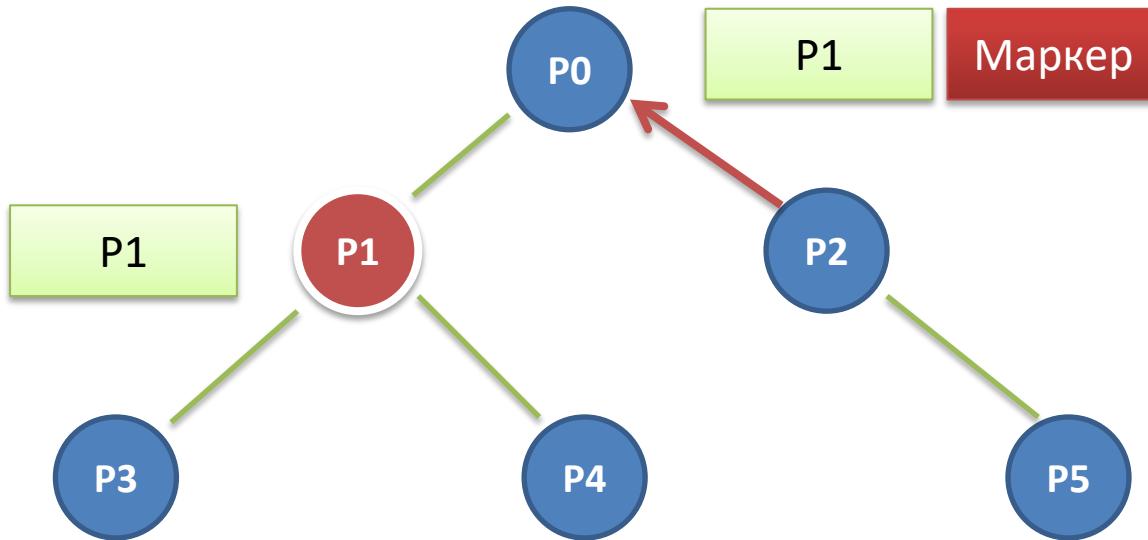
# Алгоритм древовидный маркерный (Raymond)



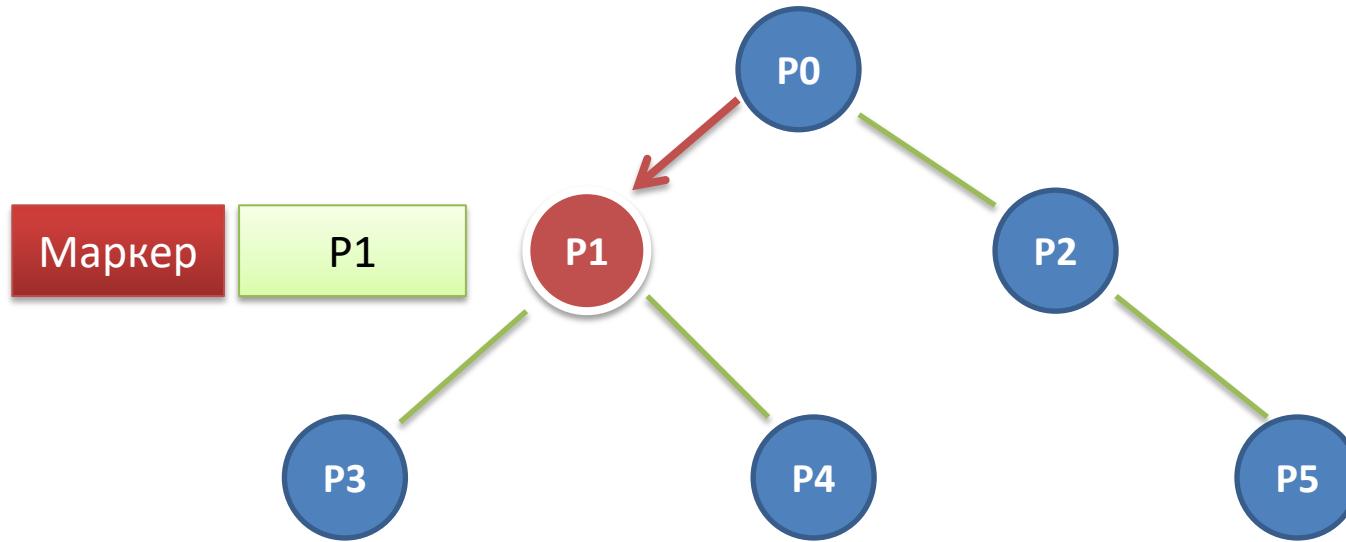
# Алгоритм древовидный маркерный (Raymond)



# Алгоритм древовидный маркерный (Raymond)



# Алгоритм древовидный маркерный (Raymond)



# Алгоритм древовидный маркерный (Raymond)

