# Распределенные алгоритмы

ЛЕКТОР: В.А. Захаров

# Лекция 9.

Задача избрания лидера.
Нижние оценки сложности.
Оптимальные выборы.
Алгоритм Галладжера–Хамблета–Спиры (GHS).
Глобальное описание алгоритма GHS.
Подробное описание алгоритма GHS.
Алгоритм Корача–Каттена–Морана.

# Задача избрания лидера

Задача избрания лидера состоит в том, чтобы, исходя из конфигурации, в которой все процессы пребывают в одном и том же состоянии, достичь такой конфигурации, в которой ровно один процесс будет находиться в состоянии **leader** , тогда как все остальные процессы будут пребывать в состоянии **lost** .

## Определение

**Алгоритмом избрания лидера** называется алгоритм, который обладает следующими свойствами.

1. Каждый процесс наделен одним и тем же локальным алгоритмом.
2. Алгоритм является децентрализованным.
3. Алгоритм достигает заключительной конфигурации в каждом вычислении, и в каждой заключительной конфигурации существует ровно один процесс, который находится в состоянии **leader** а все остальные процессы при этом пребывают в состоянии **lost**.

# Нижние оценки сложности алгоритмов избрания лидера

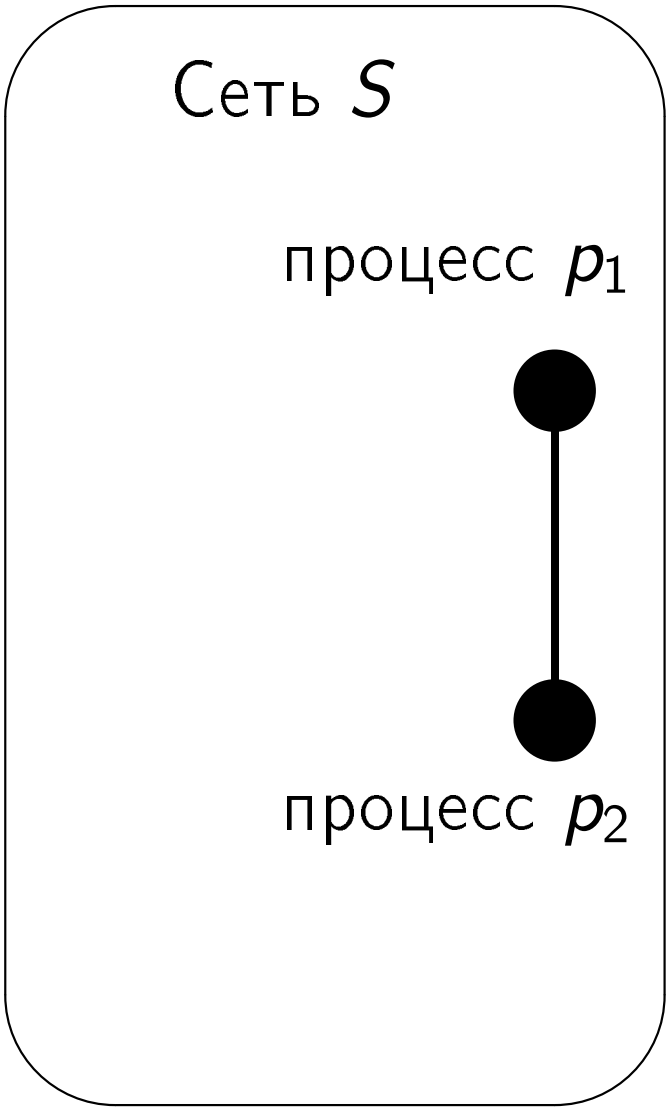
## Теорема.

Всякий алгоритм избрания лидера на основе сравнения для произвольных сетей имеет сложность (и в среднем, и в наихудшем случае) не меньшую, чем $\Omega(|E| + N \log N)$ .

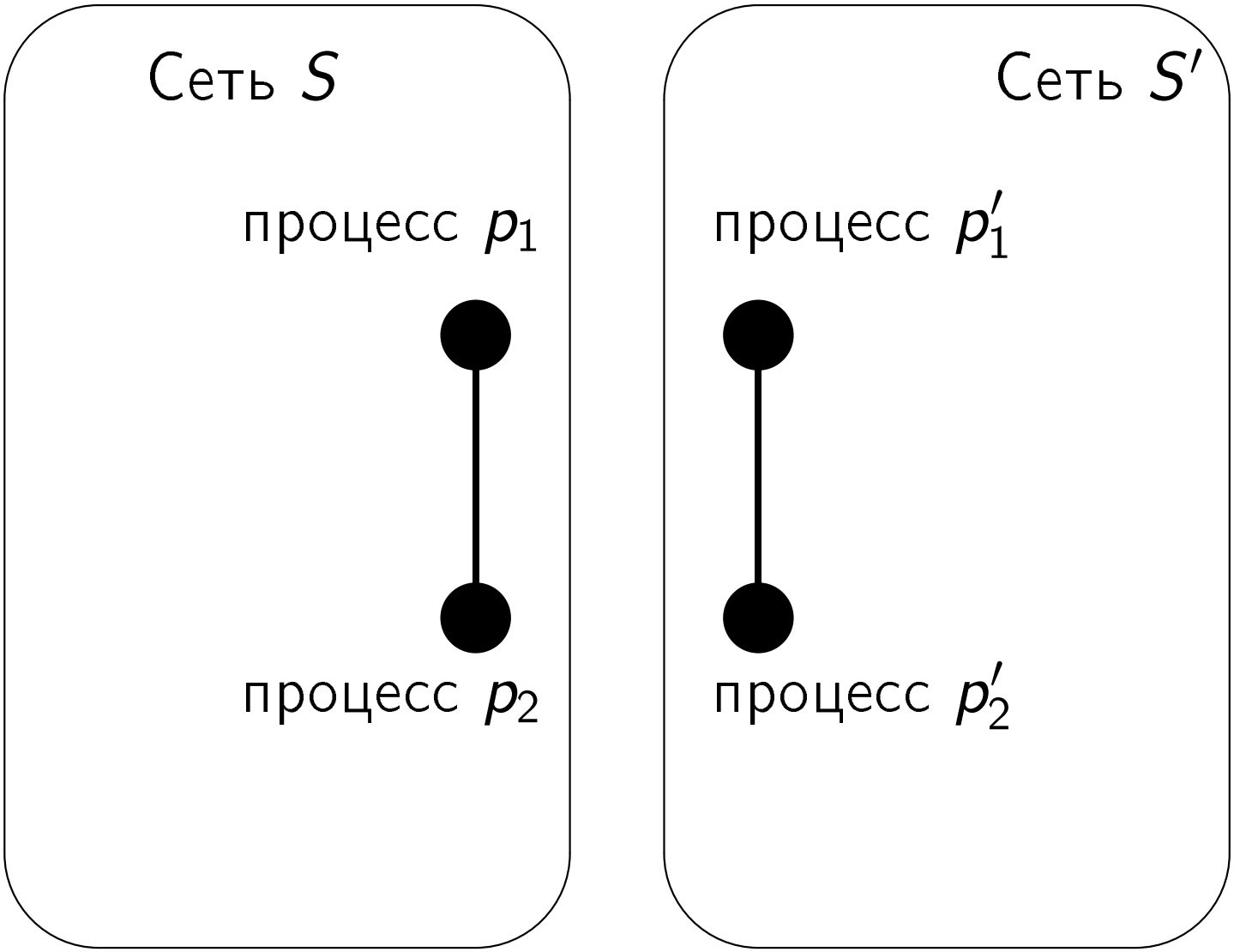
## Следствие.

Всякий децентрализованный волновой алгоритм для произвольных сетей без предварительной осведомленности о соседях имеет сложность по числу обменов сообщениями, не меньшую чем $\Omega(|E| + N \log N)$ .

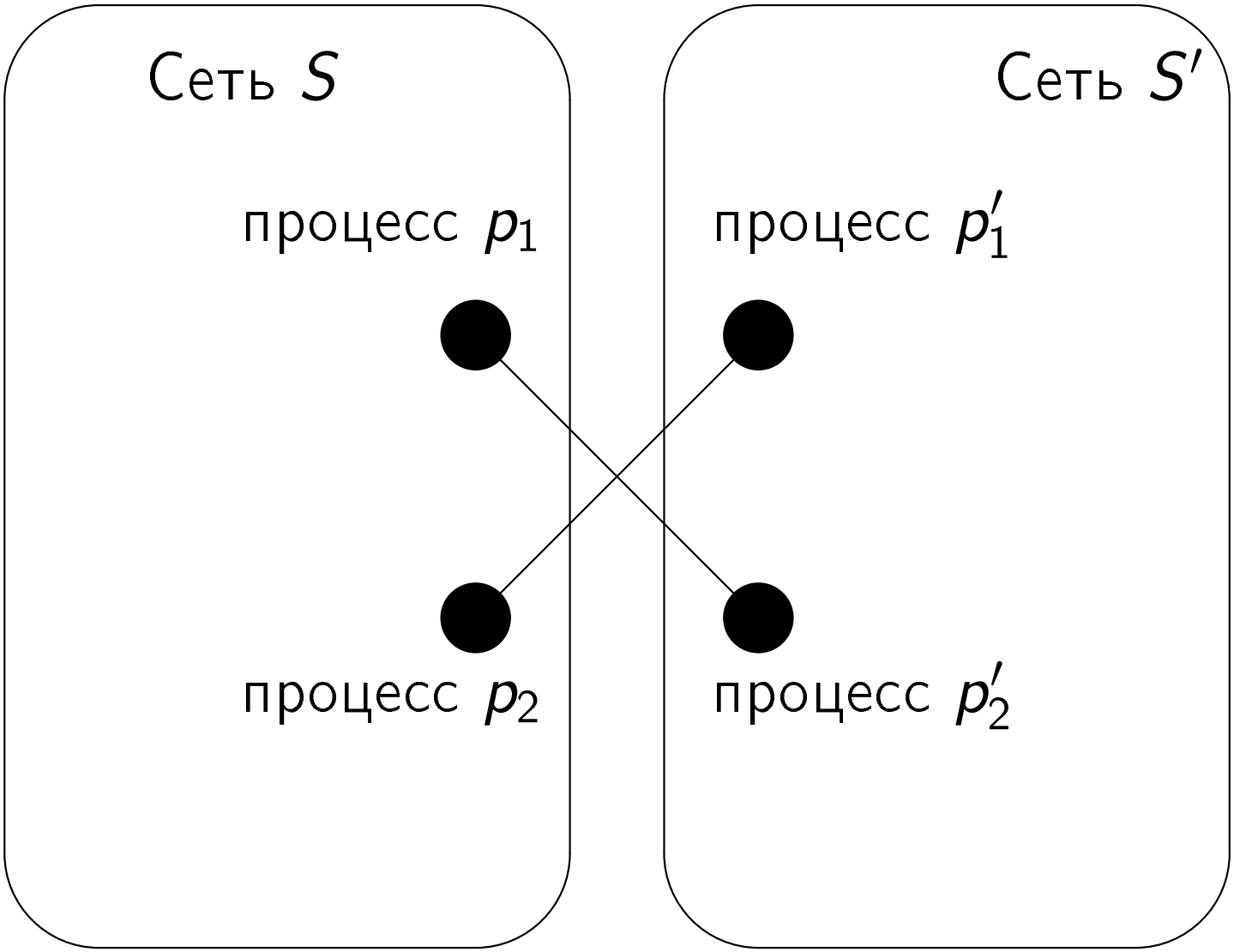
# Нижние оценки сложности алгоритмов избрания лидера



# Нижние оценки сложности алгоритмов избрания лидера



# Нижние оценки сложности алгоритмов избрания лидера



# Оптимальные выборы

ВОПРОС.

**Можно ли провести выборы лидера в произвольных сетях с использованием $O(|E| + N \log N)$ обменов сообщениями?**

# Оптимальные выборы

ВОПРОС.

**Можно ли провести выборы лидера в произвольных сетях с использованием $O(|E| + N \log N)$ обменов сообщениями?**

Задачи избрания лидера и построения остовного дерева тесно связаны друг с другом.

Пусть

$C_E$ — сложность по числу обменов сообщениями задачи о выборах,

а $C_T$ — для сложность построения остовного дерева.

# Оптимальные выборы

Алгоритм избрания лидера на дереве решает задачу о выборах на древесных сетях с использованием $O(N)$ обменов сообщениями.

Из этой теоремы следует, что $C_E \leq C_T + O(N)$ .

# Оптимальные выборы

Алгоритм избрания лидера на дереве решает задачу о выборах на древесных сетях с использованием $O(N)$ обменов сообщениями.

Из этой теоремы следует, что $C_E \leq C_T + O(N)$ .

Если в нашем распоряжении есть лидер, то остовное дерево можно построить с использованием $2|E|$ обменов сообщениями при помощи централизованного алгоритма обхода сети.

Поэтому справедливо неравенство $C_T \leq C_E + 2|E|$ .

# Оптимальные выборы

Алгоритм избрания лидера на дереве решает задачу о выборах на древесных сетях с использованием $O(N)$ обменов сообщениями.

Из этой теоремы следует, что $C_E \leq C_T + O(N)$ .

Если в нашем распоряжении есть лидер, то остовное дерево можно построить с использованием $2|E|$ обменов сообщениями при помощи централизованного алгоритма обхода сети.

Поэтому справедливо неравенство $C_T \leq C_E + 2|E|$ .

Таким образом, для оптимального выбора лидера необходимо и достаточно уметь оптимально строить остовное дерево.

Алгоритм Галладжера–Хамблета–Спиры (GHS) строит остовное дерево с использованием $2|E| + 5N \log N$ обменов сообщениями.

# Алгоритм Галладжера–Хамблета–Спиры (GHS)

Алгоритм GHS опирается на следующие допущения.

# Алгоритм Галладжера–Хамблета–Спиры (GHS)

Алгоритм GHS опирается на следующие допущения.

1. Каждому ребру приписан уникальный вес $\omega(e)$ . Все веса ребер линейно упорядочены.

# Алгоритм Галладжера–Хамблета–Спиры (GHS)

Алгоритм GHS опирается на следующие допущения.

1. Каждому ребру приписан уникальный вес $\omega(e)$ . Все веса ребер линейно упорядочены.
2. Все узлы пребывают первоначально в состоянии оцепенения и пробуждаются перед началом выполнения алгоритма.
   Некоторые узлы пробуждаются самопроизвольно, другие могут получить сообщение по ходу работы алгоритма, еще пребывая в оцепенении.
   При этом узел, получивший сообщение, вначале выполняет процедуру локальной инициализации, а затем приступает к обработке этого сообщения.

# Минимальные остовные деревья

Рассмотрим взвешенный граф $G = (V, E)$ , и для обозначения веса ребра $e$ будем использовать запись $\omega(e)$ .
Мы будем полагать, что каждое ребро имеет уникальный вес.

Вес остовного дерева $T$ в графе $G$ полагается равным сумме весов всех $N - 1$ ребер, входящих в состав $T$ .
При этом $T$ называется минимальным остовным деревом , или сокращенно MST, если ни одно остовное дерево не имеет вес меньший, чем $T$ .

# Минимальные остовные деревья

## Утверждение 9.1.

Если все веса ребер попарно различны, то существует только одно MST.

# Минимальные остовные деревья

## Утверждение 9.1.

Если все веса ребер попарно различны, то существует только одно MST.

## Доказательство.

Допустим, что есть два MST $T_1$ и $T_2$ , причем $T_1 \neq T_2$ . Рассмотрим ребро $e$ наименьшего веса, которое содержится в одном из этих деревьев, но не содержится в другом.

# Минимальные остовные деревья

## Утверждение 9.1.

Если все веса ребер попарно различны, то существует только одно MST.

## Доказательство.

Допустим, что есть два MST $T_1$ и $T_2$, причем $T_1 \neq T_2$. Рассмотрим ребро $e$ наименьшего веса, которое содержится в одном из этих деревьев, но не содержится в другом. Предположим, что $e \in T_1$. Тогда $T_2 \cup \{e\}$ содержит цикл, и, поскольку в дереве $T_1$ нет циклов, хотя бы одно ребро этого цикла, скажем, ребро $e'$, не содержится в $T_1$. Согласно выбору ребра $e$, справедливо неравенство $\omega(e) < \omega(e')$.

# Минимальные остовные деревья

## Утверждение 9.1.

Если все веса ребер попарно различны, то существует только одно MST.

## Доказательство.

Допустим, что есть два MST $T_1$ и $T_2$, причем $T_1 \neq T_2$. Рассмотрим ребро $e$ наименьшего веса, которое содержится в одном из этих деревьев, но не содержится в другом. Предположим, что $e \in T_1$. Тогда $T_2 \cup \{e\}$ содержит цикл, и, поскольку в дереве $T_1$ нет циклов, хотя бы одно ребро этого цикла, скажем, ребро $e'$, не содержится в $T_1$. Согласно выбору ребра $e$, справедливо неравенство $\omega(e) < \omega(e')$. Но тогда дерево $T_2 \cup \{e\} \setminus \{e'\}$ имеет вес меньший, чем $T_2$, вопреки тому, что $T_2$ является MST. $\square$

# Минимальные остовные деревья

Фрагмент — произвольное поддерево MST. Ребро $e$ называется исходящим ребром фрагмента $F$ , если один из концов $e$ принадлежит $F$ , а другой нет.

# Минимальные остовные деревья

Фрагмент — произвольное поддерево MST. Ребро $e$ называется исходящим ребром фрагмента $F$ , если один из концов $e$ принадлежит $F$ , а другой нет.

## Утверждение 9.2.

Если $F$ является фрагментом, и $e$ — ребро наименьшего веса, исходящее из $F$ , то $F \cup \{e\}$ также является фрагментом.

# Минимальные остовные деревья

Фрагмент — произвольное поддерево MST. Ребро $e$ называется исходящим ребром фрагмента $F$ , если один из концов $e$ принадлежит $F$ , а другой нет.

## Утверждение 9.2.

Если $F$ является фрагментом, и $e$ — ребро наименьшего веса, исходящее из $F$ , то $F \cup \{e\}$ также является фрагментом.

## Доказательство.

САМОСТОЯТЕЛЬНО (руководствуясь доказательством предыдущего утверждения).

# Минимальные остовные деревья

Фрагмент — произвольное поддерево MST. Ребро $e$ называется исходящим ребром фрагмента $F$ , если один из концов $e$ принадлежит $F$ , а другой нет.

## Утверждение 9.2.

Если $F$ является фрагментом, и $e$ — ребро наименьшего веса, исходящее из $F$ , то $F \cup \{e\}$ также является фрагментом.

## Доказательство.

САМОСТОЯТЕЛЬНО (руководствуясь доказательством предыдущего утверждения).

Алгоритмы начинают работу, располагая фрагментами, состоящими из единственного узла, и последовательно наращивают фрагменты, до тех пор пока не будет завершено построение MST.

# Глобальное описание алгоритма GHS

Всякое вычисление алгоритма GHS складывается из следующих шагов.

# Глобальное описание алгоритма GHS

Всякое вычисление алгоритма GHS складывается из следующих шагов.

1. Формируется такое семейство фрагментов, что объединение их содержит все узлы сети.

# Глобальное описание алгоритма GHS

Всякое вычисление алгоритма GHS складывается из следующих шагов.

1. Формируется такое семейство фрагментов, что объединение их содержит все узлы сети.
2. Первоначально это семейство состоит из всех узлов сети, каждый из которых рассматривается как граф с одним узлом.

# Глобальное описание алгоритма GHS

Всякое вычисление алгоритма GHS складывается из следующих шагов.

1. Формируется такое семейство фрагментов, что объединение их содержит все узлы сети.
2. Первоначально это семейство состоит из всех узлов сети, каждый из которых рассматривается как граф с одним узлом.
3. Узлы всякого фрагмента вступают во взаимодействие с целью выявления исходящего из фрагмента ребра с наименьшим весом.

# Глобальное описание алгоритма GHS

Всякое вычисление алгоритма GHS складывается из следующих шагов.

1. Формируется такое семейство фрагментов, что объединение их содержит все узлы сети.
2. Первоначально это семейство состоит из всех узлов сети, каждый из которых рассматривается как граф с одним узлом.
3. Узлы всякого фрагмента вступают во взаимодействие с целью выявления исходящего из фрагмента ребра с наименьшим весом.
4. Как только будет определено исходящее из фрагмента ребро с наименьшим весом, данный фрагмент соединяется с другим фрагментом путем добавления этого исходящего ребра, которое строится в результате взаимодействия этих двух фрагментов.

# Глобальное описание алгоритма GHS

Для эффективной реализации этих шагов нужны.

# Глобальное описание алгоритма GHS

Для эффективной реализации этих шагов нужны.
1. Имя фрагмента.

Чтобы определить исходящее ребро наименьшего веса, необходимо суметь проверить, выводит ли некоторое ребро за пределы фрагмента или оно ведет в узел того же самого фрагмента. Для этого каждый фрагмент будет наделен именем, которое будет известно всем процессам, входящим в этот фрагмент. Процессы будут проверять, является ли ребро внутренним или внешним ребром, сравненивая имена фрагментов, к которым они относятся.

# Глобальное описание алгоритма GHS

2. Соединение старшего и младшего фрагментов.

Когда два фрагмента соединяются, в процессах хотя бы одного из этих фрагментов происходит перемена имени фрагмента; для этого требуется, чтобы изменение имени произошло в каждой точке хотя бы одного из указанных фрагментов. Чтобы осуществить это изменение эффективно, в основу стратегии соединения фрагментов положен следующий принцип: младший фрагмент присоединяется к старшему фрагменту и принимает имя старшего фрагмента.

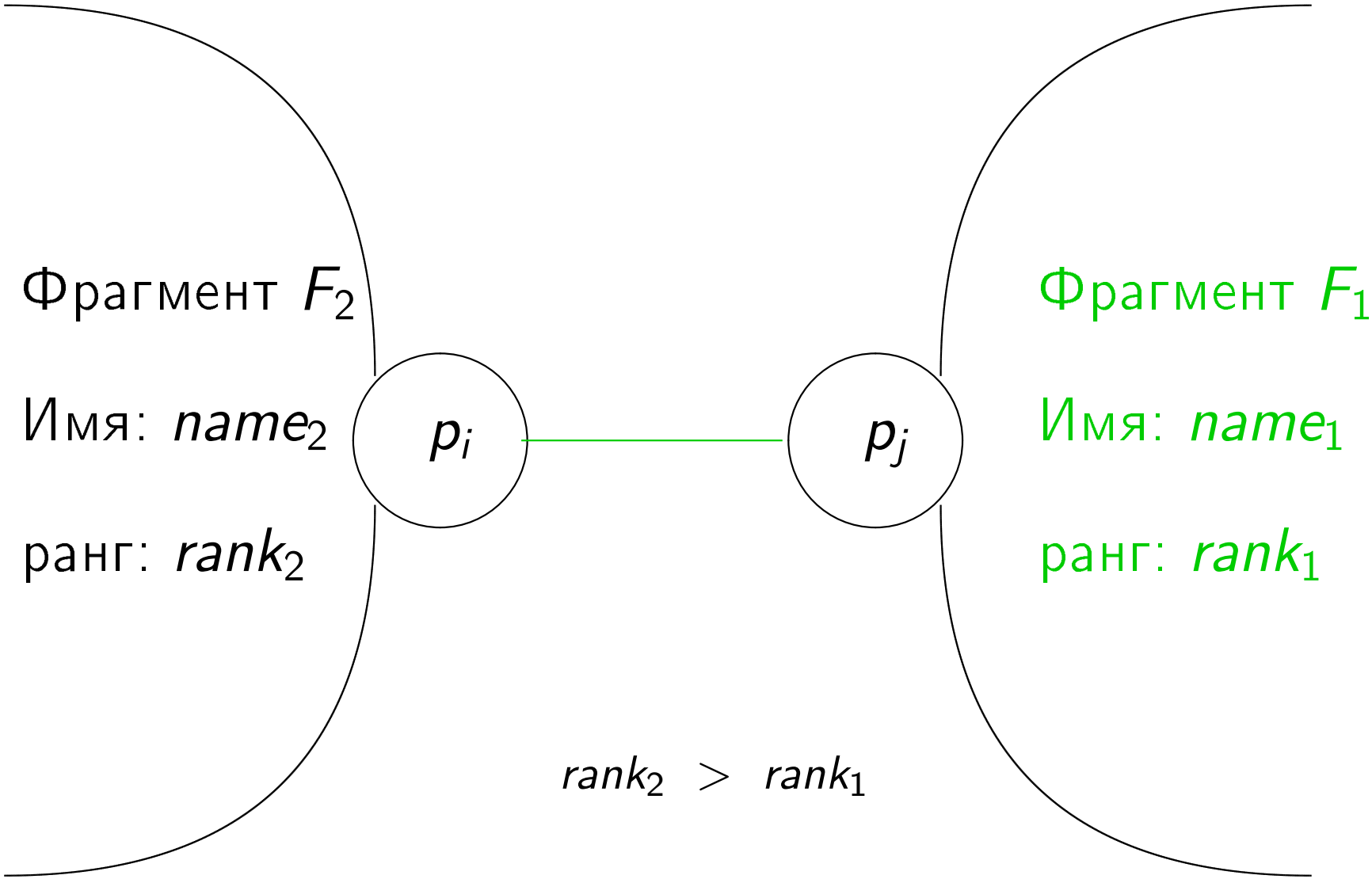
# Глобальное описание алгоритма GHS

3. Ранги фрагментов.

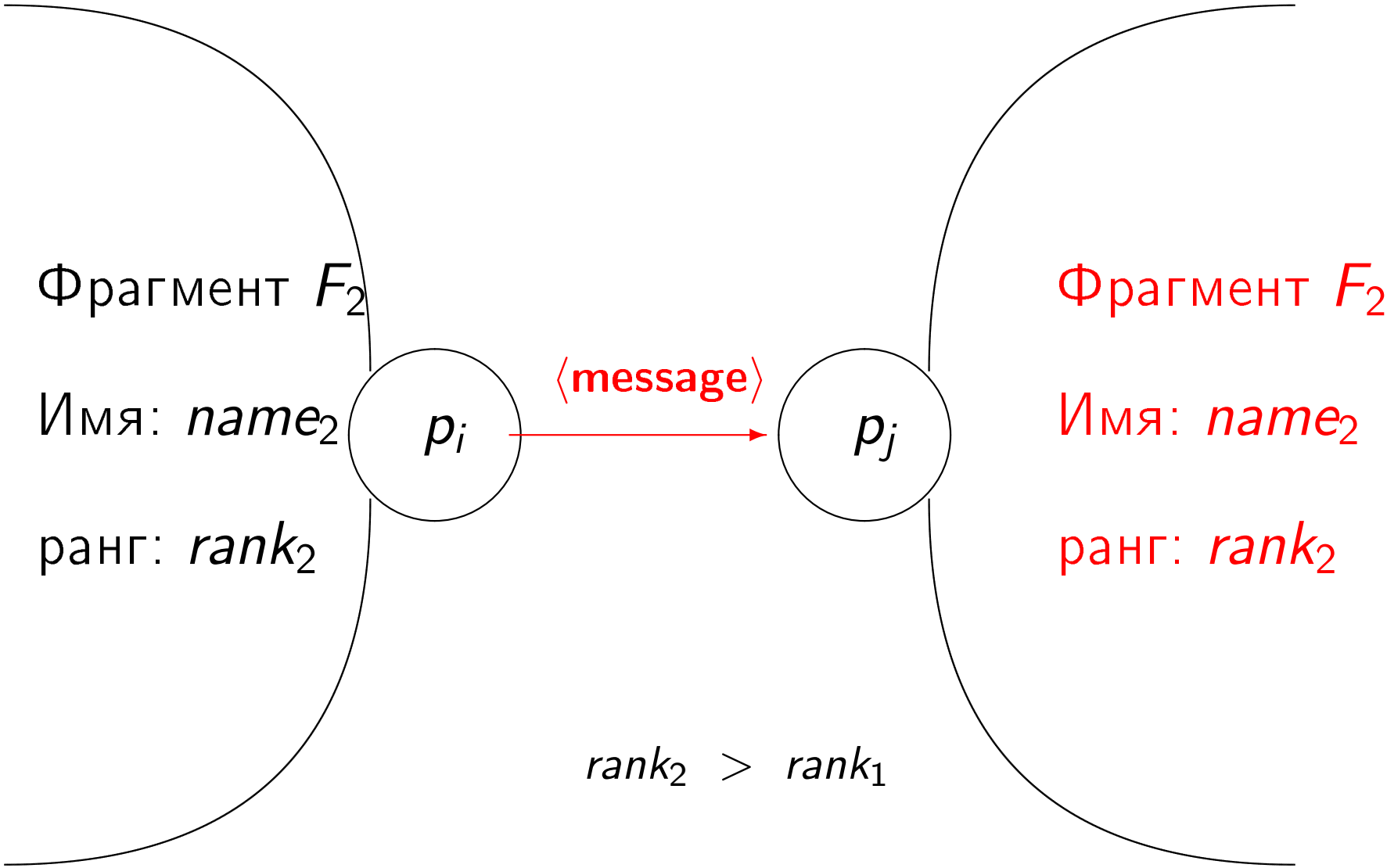
Каждому фрагменту сопоставляется ранг , который первоначально полагается равным 0 для любого фрагмента, состоящего из одного узла.

Фрагмент $F_1$ может соединиться с фрагментом $F_2$ более высокого ранга, после чего новый фрагмент $F_1 \cup F_2$ будет иметь такой же ранг, как и фрагмент $F_2$ , и унаследует имя фрагмента $F_2$ .

# Глобальное описание алгоритма GHS



# Глобальное описание алгоритма GHS

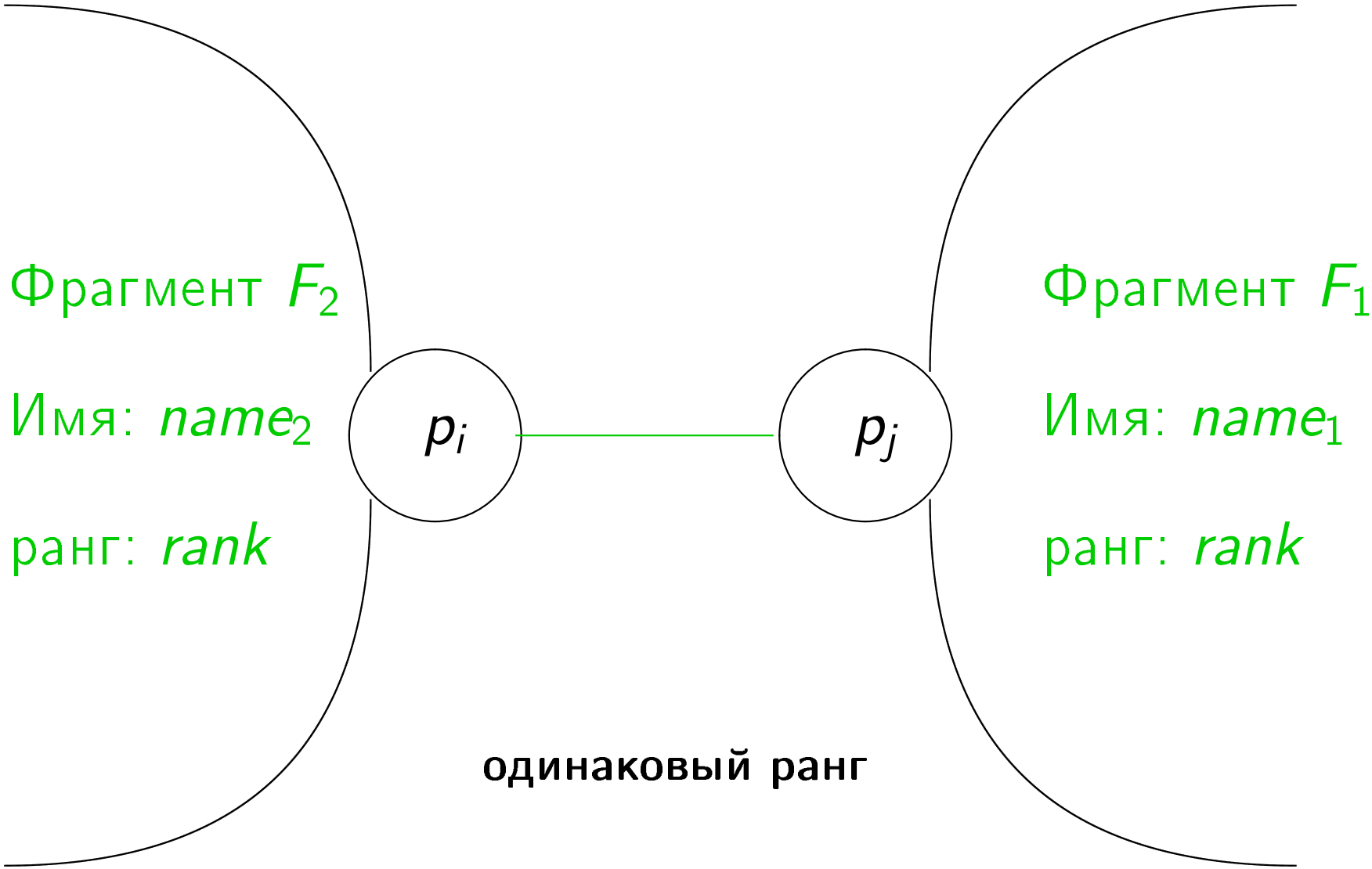


# Глобальное описание алгоритма GHS

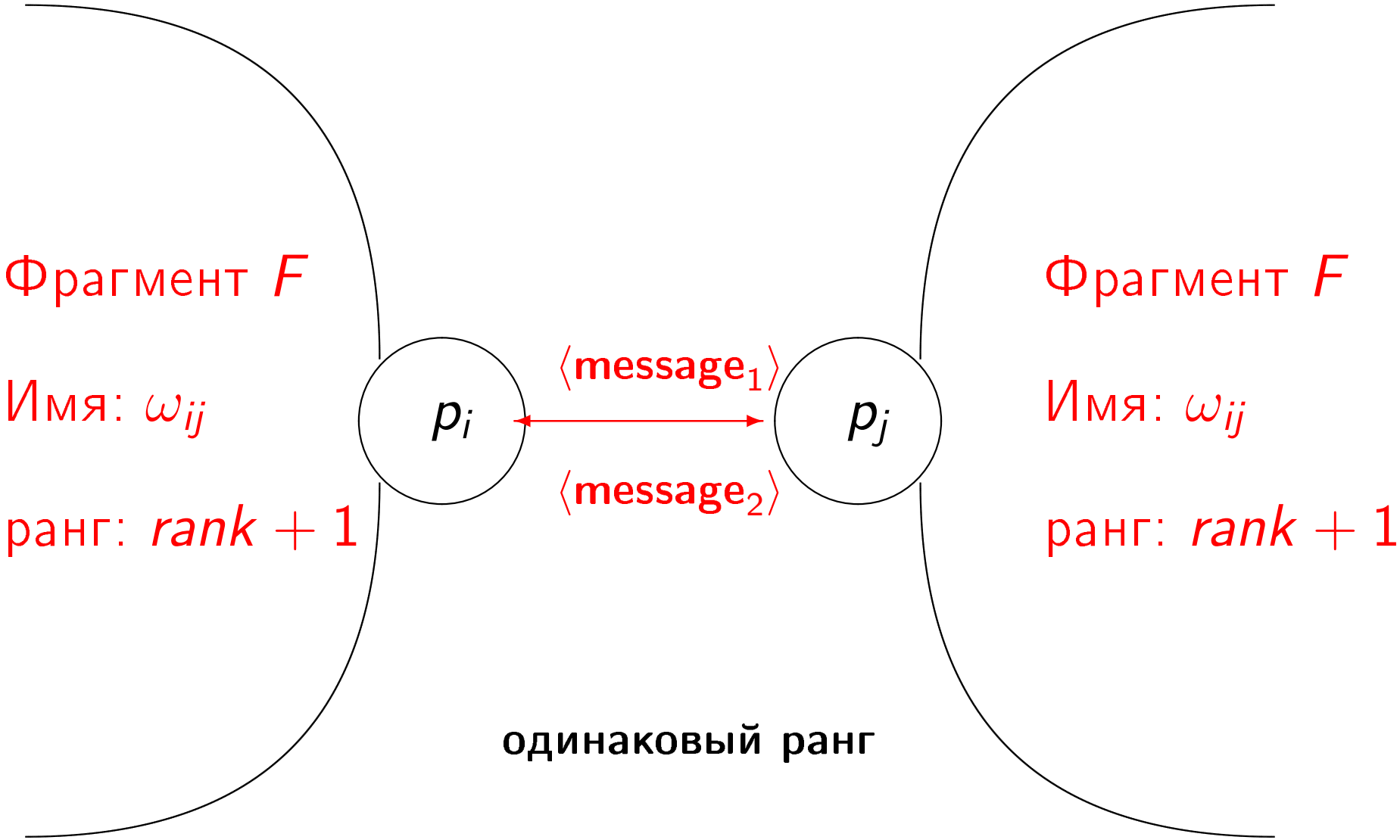
3. Ранги фрагментов.

При соединение двух фрагментов одного и того же ранга новый фрагмент будет носить новое имя и его ранг будет на единицу превосходить ранг соединенных фрагментов. Именем образовавшегося фрагмента будет вес того ребра, которое соединило два исходных фрагмента; указанное ребро назовем стержнем нового фрагмента. Те два узла, которые соединены ребром-стержнем, будут называться стержневыми узлами .

# Глобальное описание алгоритма GHS



# Глобальное описание алгоритма GHS



# Глобальное описание алгоритма GHS

## Утверждение 9.3.

Если соблюдать указанные выше правила соединения фрагментов, то суммарно во всех процессах перемена имени или ранга случится не более $N \log N$ раз.

# Глобальное описание алгоритма GHS

## Утверждение 9.3.

Если соблюдать указанные выше правила соединения фрагментов, то суммарно во всех процессах перемена имени или ранга случится не более $N \log N$ раз.

## Доказательство.

Ранг всякого процесса не понижается, и лишь в том случае, когда он повышается, процесс вынужден переменить имя своего фрагмента. Фрагмент ранга $L$ содержит не менее $2^L$ процессов, и поэтому максимально возможный ранг равен $\log N$ . Отсюда следует, что в каждом отдельном процессе ранг содержащего его фрагмента повышается не более $\log N$ раз. Значит, общее число перемен имени и ранга фрагмента ограничено величиной $N \log N$ .

# Стратегия соединения

Пусть фрагмент $F = (FN, L)$ имеет имя $FN$ и ранг $L$ , и $e_F$ — исходящее из $F$ ребро наименьшего веса.

# Стратегия соединения

Пусть фрагмент $F = (FN, L)$ имеет имя $FN$ и ранг $L$ , и $e_F$ — исходящее из $F$ ребро наименьшего веса.

Правило A. Если $e_F$ ведет в фрагмент $F' = (FN', L')$ , для которого $L < L'$ , то $F$ присоединяется к $F'$ , и образовавшийся новый фрагмент сохраняет имя $FN'$ и ранг $L'$ .

# Стратегия соединения

Пусть фрагмент $F = (FN, L)$ имеет имя $FN$ и ранг $L$ , и $e_F$ — исходящее из $F$ ребро наименьшего веса.

Правило A. Если $e_F$ ведет в фрагмент $F' = (FN', L')$ , для которого $L < L'$ , то $F$ присоединяется к $F'$ , и образовавшийся новый фрагмент сохраняет имя $FN'$ и ранг $L'$ .

Правило B. Если $e_F$ ведет в фрагмент $F' = (FN', L')$ , для которого $L = L'$ и $e_{F'} = e_F$ , то эти фрагмента соединяются и образуют один новый фрагмент, который будет носить имя $\omega(e_F)$ и иметь ранг $L + 1$ .

# Стратегия соединения

Пусть фрагмент $F = (FN, L)$ имеет имя $FN$ и ранг $L$ , и $e_F$ — исходящее из $F$ ребро наименьшего веса.

Правило A. Если $e_F$ ведет в фрагмент $F' = (FN', L')$ , для которого $L < L'$ , то $F$ присоединяется к $F'$ , и образовавшийся новый фрагмент сохраняет имя $FN'$ и ранг $L'$ .

Правило B. Если $e_F$ ведет в фрагмент $F' = (FN', L')$ , для которого $L = L'$ и $e_{F'} = e_F$ , то эти фрагмента соединяются и образуют один новый фрагмент, который будет носить имя $\omega(e_F)$ и иметь ранг $L + 1$ .

Правило C. В остальных случаях (когда $L > L'$ или $L = L'$ и $e_{F'} \neq e_F$ ) фрагмент $F$ должен ожидать, когда откроется возможность применения правила A или B.

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$ — режим работы процесса $\{\textbf{sleep}, \textbf{find}, \textbf{found}\}$;
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$ — статус канала связи $\{\textbf{basic}, \textbf{branch}, \textbf{reject}\}$;
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$ — имя фрагмента;
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$ — наименьший вес исходящего ребра;
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$ — ранг процесса;
- $father_p$
- $testch_p, bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$ — канал, ведущий в стержневой узел фрагмента;
- $testch_p$, $bestch_p$
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$ — перспективные каналы связи с соседями;
- $rec$

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$
- $stach_p[q]$
- $name_p$
- $bestwt_p$
- $level_p$
- $father_p$
- $testch_p$, $bestch_p$
- $rec$ — счетчик.

# Подробное описание алгоритма GHS

## Переменные.

- $state_p$ — режим работы процесса $\{\textbf{sleep}, \textbf{find}, \textbf{found}\}$;
- $stach_p[q]$ — статус канала связи $\{\textbf{basic}, \textbf{branch}, \textbf{reject}\}$;
- $name_p$ — имя фрагмента;
- $bestwt_p$ — наименьший вес исходящего ребра;
- $level_p$ — ранг процесса;
- $father_p$ — канал, ведущий в стержневой узел фрагмента;
- $testch_p$, $bestch_p$ — перспективные каналы связи с соседями;
- $rec$ — счетчик.

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**connect**, *level*$\rangle$
- $\langle$**initiate**, *level*, *name*, *state*$\rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**connect**, *level*$\rangle$ — сообщение, которое отправляется по ребру наименьшего веса, исходящего из фрагмента, с указанием ранга этого фрагмента;
- $\langle$**initiate**, *level*, *name*, *state*$\rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**connect**, *level*$\rangle$

- $\langle$**initiate**, *level*, *name*, *state*$\rangle$ — сообщение, которое отправляется по ребру наименьшего веса, исходящего из фрагмента,
  при выполнении **правила A** — присоединения фрагмента меньшего ранга, а также
  при выполнении **правила B** — объединения двух фрагментов одинакового ранга;

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**connect**, *level*$\rangle$ — сообщение, которое отправляется по ребру наименьшего веса, исходящего из фрагмента, с указанием ранга этого фрагмента;
- $\langle$**initiate**, *level*, *name*, *state*$\rangle$ — сообщение, которое отправляется по ребру наименьшего веса, исходящего из фрагмента,
  при выполнении **правила A** — присоединения фрагмента меньшего ранга, а также
  при выполнении **правила B** — объединения двух фрагментов одинакового ранга;

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle \textbf{test}, \textit{level}, \textit{name} \rangle$
- $\langle \textbf{reject} \rangle$
- $\langle \textbf{accept} \rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**test**, *level*, *name*$\rangle$ — сообщение, которое отправляется по «свежему» ребру наименьшего веса, исходящего из узла, с указанием имени и ранга фрагмента, которому принадлежит узел;
- $\langle$**reject**$\rangle$
- $\langle$**accept**$\rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle \textbf{test}, \textit{level}, \textit{name} \rangle$
- $\langle \textbf{reject} \rangle$ — сообщение, которое отправляется по ребру в том случае, если это ребро соединяет два узла, принадлежащие одному и тому же фрагменту;
- $\langle \textbf{accept} \rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle \textbf{test}, \textit{level}, \textit{name} \rangle$
- $\langle \textbf{reject} \rangle$
- $\langle \textbf{accept} \rangle$ — сообщение, которое отправляется по ребру в том случае, если это ребро соединяет два узла, принадлежащие разным фрагментам;

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**test**, *level*, *name*$\rangle$ — сообщение, которое отправляется по «свежему» ребру наименьшего веса, исходящего из узла, с указанием имени и ранга фрагмента, которому принадлежит узел;
- $\langle$**reject**$\rangle$ — сообщение, которое отправляется по ребру в том случае, если это ребро соединяет два узла, принадлежащие одному и тому же фрагменту;
- $\langle$**accept**$\rangle$ — сообщение, которое отправляется по ребру в том случае, если это ребро соединяет два узла, принадлежащие разным фрагментам;

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**report**, *bestwt*$\rangle$
- $\langle$**changeroot**$\rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**report**, *bestwt*$\rangle$ — сообщение, которое отправляется по направлению к стержневому узлу с указанием наименьшего веса «свежего» ребра, исходящего из узла;
- $\langle$**changeroot**$\rangle$

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**report**, *bestwt*$\rangle$

- $\langle$**changeroot**$\rangle$ — сообщение, которое отправляется по ребрам присоединяемого фрагмента и указывает то направление, в котором будет располагаться стержневой узел;

# Подробное описание алгоритма GHS

## Сообщения.

- $\langle$**report**, *bestwt*$\rangle$ — сообщение, которое отправляется по направлению к стержневому узлу с указанием наименьшего веса «свежего» ребра, исходящего из узла;
- $\langle$**changeroot**$\rangle$ — сообщение, которое отправляется по ребрам присоединяемого фрагмента и указывает то направление, в котором будет располагаться стержневой узел;

# Подробное описание алгоритма GHS

(1) Первым действием каждого процесса является инициализация данного алгоритма:

**begin** пусть $pq$ — это канал связи процесса $p$, имеющий наименьший вес ;
$stach_p[q] := branch$ ; $level_p := 0$ ;
$state_p := found$ ; $rec_p := 0$ ;
send $\langle \mathbf{connect}, 0 \rangle$ to $q$
**end**

Каждый процесс начинает работу с того, что присвивает себе самый низкий ранг, находит исходящее ребро наименьшего веса, и объявляет о своей готовности присоединиться к соседнему фрагменту.

# Подробное описание алгоритма GHS

(2) После получения сообщения $\langle$**connect**, $L\rangle$ от $q$:

**begin if** $L < level_p$ **then** (* Соединить по правилу A *)

**begin** $stach_p[q] := branch$ ;

send $\langle$**initiate**, $level_p, name_p, state_p\rangle$ to $q$

**end**

**else if** $stach_p[q] = basic$ or $L > level_p$

**then** (* Правило C *)

обработать это сообщение позднее

**else** (* Правило B *)

send $\langle$**initiate**, $level_p + 1, \omega(pq), find\rangle$ to $q$

**end**

После предложения об объединении фрагментов тот узел, который получил это предложение, проверяет ранг соседнего фрагмента,...

# Подробное описание алгоритма GHS

(2) После получения сообщения $\langle \textbf{connect}, L \rangle$ от $q$:

**begin if** $L < level_p$ **then** (* Соединить по правилу A *)

**begin** $stach_p[q] := branch$ ;

send $\langle \textbf{initiate}, level_p, name_p, state_p \rangle$ to $q$

**end**

**else if** $stach_p[q] = basic$ **or** $L > level_p$

**then** (* Правило C *)

обработать это сообщение позднее

**else** (* Правило B *)

send $\langle \textbf{initiate}, level_p + 1, \omega(pq), find \rangle$ to $q$

**end**

и если ранг соседнего фрагмента меньше ранга того фрагмента, которому принадлежит узел $p$ , то узел $p$ отправляет соседнему фрагменту приказ присоединиться «в качестве подчиненного», приняв имя и ранг старшего фрагмента,...

# Подробное описание алгоритма GHS

(2) После получения сообщения $\langle \textbf{connect}, L \rangle$ от $q$:

**begin if** $L < level_p$ **then** (* Соединить по правилу A *)
**begin** $stach_p[q] := branch$ ;
send $\langle \textbf{initiate}, level_p, name_p, state_p \rangle$ to $q$
**end**
**else if** $stach_p[q] = basic$ **or** $L > level_p$
**then** (* Правило C *)
обработать это сообщение позднее
**else** (* Правило B *)
send $\langle \textbf{initiate}, level_p + 1, \omega(pq), find \rangle$ to $q$
**end**

а в противном случае процесс $p$ проверяет, не было ли ребро $pq$ уже «опробовано» процессом $p$ ранее.

# Подробное описание алгоритма GHS

(2) После получения сообщения $\langle \textbf{connect}, L \rangle$ от $q$:

**begin if** $L < level_p$ **then** (* Соединить по правилу A *)
**begin** $stach_p[q] := branch$ ;
send $\langle \textbf{initiate}, level_p, name_p, state_p \rangle$ to $q$
**end**
**else if** $stach_p[q] = basic$ **or** $L > level_p$
**then** (* Правило C *)
обработать это сообщение позднее
**else** (* Правило B *)
send $\langle \textbf{initiate}, level_p + 1, \omega(pq), find \rangle$ to $q$
**end**

Если это ребро не было еще «опробовано» процессом $p$ , то обработка предложения объединиться откладывается до тех пор, пока процесс $p$ не приступит к оценке ребра $pq$ ,....

# Подробное описание алгоритма GHS

(2) После получения сообщения $\langle \textbf{connect}, L \rangle$ от $q$:

**begin if** $L < level_p$ **then** (* Соединить по правилу A *)
**begin** $stach_p[q] := branch$ ;
send $\langle \textbf{initiate}, level_p, name_p, state_p \rangle$ to $q$
**end**
**else if** $stach_p[q] = basic$ **or** $L > level_p$
**then** (* Правило C *)
обработать это сообщение позднее
**else** (* Правило B *)
send $\langle \textbf{initiate}, level_p + 1, \omega(pq), find \rangle$ to $q$
**end**

а иначе (т.е. если ребро $pq$ было уже учтено), то процесс $p$ приказывает соседнему фрагменту равноправно объединиться «на более высоком уровне», приняв в качестве нового имени для объединенного фрагмента вес стержневого ребра $pq$ .

# Подробное описание алгоритма GHS

**Вопрос.**

А откуда известно процессу $p$, что в последнем случае ранги фрагментов равны?
Возможен ли такой случай, когда соседний фрагмент (которому принадлежит узел $q$) имеет ранг, превосходящий ранг того фрагмента, которому принадлежит узел $p$?
И что в таком случае делает процесс $p$?

# Подробное описание алгоритма GHS

(3) После получения сообщения $\langle \textbf{initiate}, L, F, S \rangle$ от $q$:

**begin** $level_p := L$ ; $name_p := F$ ; $state_p := S$ ; $father_p := q$ ;
$bestch_p := udef$ ; $bestwt_p := \infty$ ;
**forall** $r \in Neigh_p : \ stach_p[r] = branch \wedge r \neq q$ **do**
send $\langle \textbf{initiate}, L, F, S \rangle$ to $r$ ;
**if** $state_p = find$ **then begin** $rec_p := 0$ ; $test$ **end**
**end**

После приказа об объединении, процесс $p$ изменяет свое имя, ранг, статус и направление к стержневому узлу,...

# Подробное описание алгоритма GHS

(3) После получения сообщения $\langle \textbf{initiate}, L, F, S \rangle$ от $q$:

**begin** $level_p := L$ ; $name_p := F$ ; $state_p := S$ ; $father_p := q$ ;
$bestch_p := udef$ ; $bestwt_p := \infty$ ;
**forall** $r \in Neigh_p : \ stach_p[r] = branch \wedge r \neq q$ **do**
send $\langle \textbf{initiate}, L, F, S \rangle$ to $r$ ;
**if** $state_p = find$ **then begin** $rec_p := 0$ ; $test$ **end**
**end**

«сбрасывает» свои текущие показатели поиска исходящего ребра наименьшего веса,...

# Подробное описание алгоритма GHS

(3) После получения сообщения $\langle \textbf{initiate}, L, F, S \rangle$ от $q$:

**begin** $level_p := L$ ; $name_p := F$ ; $state_p := S$ ; $father_p := q$ ;
$bestch_p := udef$ ; $bestwt_p := \infty$ ;
**forall** $r \in Neigh_p : stach_p[r] = branch \wedge r \neq q$ **do**
send $\langle \textbf{initiate}, L, F, S \rangle$ to $r$ ;
**if** $state_p = find$ **then begin** $rec_p := 0$ ; $test$ **end**
**end**

отправляет своим соседям по фрагменту приказ о присоединении в связи с образованием нового фрагмента,...

# Подробное описание алгоритма GHS

(3) После получения сообщения $\langle \textbf{initiate}, L, F, S \rangle$ от $q$:

**begin** $level_p := L$ ; $name_p := F$ ; $state_p := S$ ; $father_p := q$ ;
$bestch_p := udef$ ; $bestwt_p := \infty$ ;
**forall** $r \in Neigh_p : \ stach_p[r] = branch \wedge r \neq q$ **do**
send $\langle \textbf{initiate}, L, F, S \rangle$ to $r$ ;
**if** $state_p = find$ **then begin** $rec_p := 0$ ; $test$ **end**
**end**

и в том случае, если статус узла требует продолжать поиск ребра наименьшего веса, то присоединяется к этому поиску, запустив процедуру *test*.

# Подробное описание алгоритма GHS

**Вопрос.**

А зачем процесс $p$ «сбрасывает» текущие показатели поиска (переменные *bestch* и *bestwt*), а не продолжает поиск, сохраняя достигнутые показатели?

Будет ли алгоритм работать корректно, если этого «сброса» показателей поиска не делать?

# Подробное описание алгоритма GHS

(4) **procedure** *test*:
**begin if** $\exists q \in Neigh_p : \ stach_p[q] = basic$ **then**
**begin** $testch_p := q$ with $stach_p[q] = basic$
and $\omega(pq)$ minimal ;
send $\langle$**test**, $level_p, name_p\rangle$ to $testch_p$
**end**
**else begin** $testch_p := udef$ ; *report* **end**
**end**

Узел $p$ , стремясь отыскать исходящее из него ребро наименьшего веса, просматривает одно за другим в порядке возрастания веса всех примыкающих к нему ребер, имеющих статус *basic*.

# Подробное описание алгоритма GHS

(4) **procedure** *test*:
**begin if** $\exists q \in Neigh_p$ : $stach_p[q] = basic$ **then**
**begin** $testch_p := q$ with $stach_p[q] = basic$
and $\omega(pq)$ minimal ;
send $\langle$**test**, $level_p, name_p\rangle$ to $testch_p$
**end**
**else begin** $testch_p := udef$ ; *report* **end**
**end**

Если из узла $p$ исходит хотя бы одно «свежее» ребро (имеющее статус *basic*), то процесс $p$ выбирает ребро наименьшего веса, и отправляет по этому каналу запрос об имени и ранге фрагмента, которому принадлежит узел $q$ , присоединенный к этому ребру.

# Подробное описание алгоритма GHS

(4) **procedure** *test*:
**begin if** $\exists q \in Neigh_p$ : $stach_p[q] = basic$ **then**
**begin** $testch_p := q$ with $stach_p[q] = basic$
and $\omega(pq)$ minimal ;
send $\langle$**test**, $level_p, name_p\rangle$ to $testch_p$
**end**
**else begin** $testch_p := udef$ ; *report* **end**
**end**

А если все исходящие ребра уже были опробованы ранее, то узел $p$ запускает процедуру составления доклада *report* о результатах поиска.

# Подробное описание алгоритма GHS

(5) После получения сообщения $\langle \textbf{test}, L, F \rangle$ от $q$:

**begin if** $L > level_p$ **then** (* С ответом придется подождать! *)
обработать это сообщение позднее
**else if** $F = name_p$ **then** (* внутреннее ребро *)
**begin if** $stach_p[q] = basic$ **then** $stach_p[q] := reject$ ;
**if** $q \neq testch_p$
**then** send $\langle \textbf{reject} \rangle$ to $q$
**else** *test*
**end**
**else** send $\langle \textbf{accept} \rangle$ to $q$
**end**

При получении запроса об имени и ранге фрагмента, которому принадлежит узел $p$ , проводится проверка ранга того фрагмента, которому принадлежит узел, отправивший запрос. Если запрос пришел от фрагмента более высокого ранга, то отклик откладывается (ПОЧЕМУ??? Узел $p$ в чем то не уверен? ).

# Подробное описание алгоритма GHS

(5) После получения сообщения $\langle \textbf{test}, L, F \rangle$ от $q$:

**begin if** $L > level_p$ **then** (* С ответом придется подождать! *)
обработать это сообщение позднее
**else if** $F = name_p$ **then** (* внутреннее ребро *)
**begin if** $stach_p[q] = basic$ **then** $stach_p[q] := reject$ ;
**if** $q \neq testch_p$
**then** send $\langle \textbf{reject} \rangle$ to $q$
**else** *test*
**end**
**else** send $\langle \textbf{accept} \rangle$ to $q$
**end**

А если ранг фрагмента, от которого поступил запрос не превосходит ранга фрагмента, которому принадлежит узел $p$ , то узел проверяет имя того фрагмента от которого поступил запрос. И если $p$ обнаруживает, что узел $q$ принадлежит тому же фрагменту, то ...

# Подробное описание алгоритма GHS

(5) После получения сообщения $\langle \textbf{test}, L, F \rangle$ от $q$:

**begin if** $L > level_p$ **then** (* С ответом придется подождать! *)
обработать это сообщение позднее
**else if** $F = name_p$ **then** (* внутреннее ребро *)
**begin if** $stach_p[q] = basic$ **then** $stach_p[q] := reject$ ;
**if** $q \neq testch_p$
**then** send $\langle \textbf{reject} \rangle$ to $q$
**else** *test*
**end**
**else** send $\langle \textbf{accept} \rangle$ to $q$
**end**

это означает, что ребро $pq$ — внутреннее ребро фрагмента, и ему присваивается статус *reject*. После этого в адрес узла $q$ отправляется либо сообщение об этом $\langle \textbf{reject} \rangle$, либо запускает процедуру *test* (если узел $q$ был узлом, в который вело ребро наименьшего веса из узла $p$ ).

# Подробное описание алгоритма GHS

**Вопрос.**

А почему процесс $p$ не отправляет процессу $q$ сообщения $\langle \textbf{reject} \rangle$ в противном случае? Не вводит ли он тем самым процесс $q$ в заблуждение о статусе ребра $pq$? И зачем ему нужно снова запускать процедуру *test*?

# Подробное описание алгоритма GHS

(5) После получения сообщения $\langle \textbf{test}, L, F \rangle$ от $q$:

**begin if** $L > level_p$ **then** (* С ответом придется подождать! *)
обработать это сообщение позднее
**else if** $F = name_p$ **then** (* внутреннее ребро *)
**begin if** $stach_p[q] = basic$ **then** $stach_p[q] := reject$ ;
**if** $q \neq testch_p$
**then** send $\langle \textbf{reject} \rangle$ to $q$
**else** *test*
**end**
**else** send $\langle \textbf{accept} \rangle$ to $q$
**end**

И, наконец, если узлы $p$ и $q$ принадлежат разным фрагментам, то $p$ сообщает об этом узлу $q$ .

# Подробное описание алгоритма GHS

(6) После получения сообщения $\langle$**accept**$\rangle$ от $q$:

**begin** $testch_p := udef$ ;
  **if** $\omega(pq) < bestwt_p$
    **then begin** $bestwt_p := \omega(pq)$ ; $bestch_p := q$ **end**;
  *report*
**end**

Если получено подтверждение о том, что проверяемое ребро $pq$ соединяет узлы, принадлежащие разным фрагментам, то процесс $p$ завершает поиск наилучшего ребра, считая, что это — ребро $pq$ , и приступает к подготовке отчета в «центр», запуская процедуру *report*.

# Подробное описание алгоритма GHS

**Вопрос.**

А зачем «сбрасывается» значение переменной $testch_p$? А сохранит ли алгоритм корректность, если удалить оператор $testch_p := udef$?

# Подробное описание алгоритма GHS

(7) После получения сообщения $\langle$**reject**$\rangle$ от $q$:
**begin if** $stach_p[q] = basic$ **then** $stach_p[q] := reject$;
*test*
**end**

Если получено подтверждение о том, что проверяемое ребро $pq$ соединяет узлы, принадлежащие одному и тому же фрагменту, то процесс $p$ отмечает это, и приступает к выбору другого исходящего ребра, запуская процедуру *test*.

# Подробное описание алгоритма GHS

(8) **procedure** *report*:
**begin if** $rec_p = \#\{q : stach_p[q] = branch \wedge q \neq father_p\}$
**and** $testch_p = udef$ **then**
**begin** $state_p := found$ ; send $\langle$**report**, $bestwt_p\rangle$ to $father_p$ **en**
**end**

Если процесс $p$ получил от всех соседей по фрагменту (за исключением, того, который располагается по пути в стержневой узел) данные о наилучших ребрах, и сам процесс $p$ не проводит более поиска наилучшего ребра, то он отправляет по направлению к стержневому узлу сообщение с указанием веса этого наилучшего ребра.

# Подробное описание алгоритма GHS

(9) После получения сообщения $\langle \textbf{report}, \omega \rangle$ от $q$:

```
begin if q ≠ father_p
        then (* ответить на сообщение initiate *)
             begin if ω < bestwt_p then
                        begin bestwt_p := ω ; bestch_p := q end;
                   rec_p := rec_p + 1 ; report
             end
        else (* ребро pq — на пути в стержень *)
            if state_p = find
             then обработать это сообщение позднее
             else if ω > bestwt_p
                    then changeroot
                    else if ω = bestwt_p = ∞ then stop
end
```

Если процесс $p$ получил от процесса $q$ (это может быть только его сосед по фрагменту (почему? )) доклад о весе наилучшего ребра, исходящего из узлов соответствующего поддерева, то $p$ выясняет, куда ведет это ребро.

# Подробное описание алгоритма GHS

(9) После получения сообщения $\langle \textbf{report}, \omega \rangle$ от $q$:

```
begin if q ≠ father_p
      then  (* ответить на сообщение initiate *)
            begin  if  ω < bestwt_p then
                       begin bestwt_p := ω ; bestch_p := q end;
                   rec_p := rec_p + 1 ; report
            end
      else (* ребро pq — на пути в стержень *)
           if state_p = find
             then обработать это сообщение позднее
             else if ω > bestwt_p
                    then changeroot
                    else if ω = bestwt_p = ∞ then stop
end
```

Если ребро $pq$ не лежит на пути из $p$ в стержневой узел того фрагмента, которому принадлежит $p$ , то $p$ уточняет вес и направление к наилучшему ребру, и пытается приступить к подготовке отчета, вызывая процедуру *report*.

# Подробное описание алгоритма GHS

(9) После получения сообщения $\langle \textbf{report}, \omega \rangle$ от $q$:

**begin if** $q \neq father_p$

**then** (* ответить на сообщение **initiate** *)

**begin if** $\omega < bestwt_p$ **then**

**begin** $bestwt_p := \omega$ ; $bestch_p := q$ **end**;

$rec_p := rec_p + 1$ ; *report*

**end**

**else** (* ребро $pq$ — на пути в стержень *)

**if** $state_p = find$

**then** обработать это сообщение позднее

**else if** $\omega > bestwt_p$

**then** *changeroot*

**else if** $\omega = bestwt_p = \infty$ **then stop**

**end**

Если доклад, полученный $p$ , поступил от процесса $q$ , который лежит на пути из $p$ в стержень, то в том случае, если $p$ еще не завершил поиск наилучшего ребра, то $p$ откладывает обработку этого сообщения, пока найдет наилучшее ребро.

# Подробное описание алгоритма GHS

(9) После получения сообщения $\langle \textbf{report}, \omega \rangle$ от $q$:

**begin if** $q \neq father_p$

**then** (* ответить на сообщение **initiate** *)

**begin if** $\omega < bestwt_p$ **then**

**begin** $bestwt_p := \omega$ ; $bestch_p := q$ **end**;

$rec_p := rec_p + 1$ ; *report*

**end**

**else** (* ребро $pq$ — на пути в стержень *)

**if** $state_p = find$

**then** обработать это сообщение позднее

**else if** $\omega > bestwt_p$

**then** *changeroot*

**else if** $\omega = bestwt_p = \infty$ **then stop**

**end**

А если вес ребра, найденного $p$ , меньше того веса, который нашел $q$ , то объединение фрагмента, к которому принадлежит $p$ , будет проводиться по тому ребру, которое сумел отыскать $p$ . Поэтому $p$ запускает процедуру *changeroot*.

# Подробное описание алгоритма GHS

(9) После получения сообщения $\langle \textbf{report}, \omega \rangle$ от $q$:

**begin if** $q \neq father_p$

**then** (* ответить на сообщение **initiate** *)

**begin if** $\omega < bestwt_p$ **then**

**begin** $bestwt_p := \omega$ ; $bestch_p := q$ **end**;

$rec_p := rec_p + 1$ ; *report*

**end**

**else** (* ребро $pq$ — на пути в стержень *)

**if** $state_p = find$

**then** обработать это сообщение позднее

**else if** $\omega > bestwt_p$

**then** *changeroot*

**else if** $\omega = bestwt_p = \infty$ **then stop**

**end**

В противном случае объединение фрагмента, к которому принадлежит $p$ , с «равноправным» фрагментом будет проводиться по ту сторону фрагмента, на которой лежит узел $q$ , и поэтому данные, собранные $p$ — неактуальны.

# Подробное описание алгоритма GHS

(10) **procedure** *changeroot*:
**begin if** $stach_p[bestch_p] = branch$
**then** send $\langle$**changeroot**$\rangle$ to $bestch_p$
**else begin** send $\langle$**connect**, $level_p\rangle$ to $bestch_p$ ;
$stach_p[bestch_p] := branch$
**end**
**end**

Эта процедура инициирует объединение фрагмента по найденному исходящему ребру наименьшего веса. Сообщение $\langle$**changeroot**$\rangle$ отправляется по направлению к исходящему ребру наименьшего веса до тех пор, пока...

# Подробное описание алгоритма GHS

(10) **procedure** *changeroot*:
**begin if** $stach_p[bestch_p] = branch$
**then** send $\langle$**changeroot**$\rangle$ to $bestch_p$
**else begin** send $\langle$**connect**, $level_p\rangle$ to $bestch_p$ ;
$stach_p[bestch_p] := branch$
**end**
**end**

... это сообщение не достигнет того узла, из которого исходит ребро наименьшего веса. Тогда из этого узла в соседний фрагмент отправляется предложение $\langle$**connect**, $level_p\rangle$ об объединении.

# Подробное описание алгоритма GHS

(11) После получения сообщения ⟨**changeroot**⟩:
**begin** *changeroot* **end**

# Подробное описание алгоритма GHS

Таким образом каждый локальный алгоритм может выполнять одно из 10 действий (1)–(10) с использованием процедур *test*, *report*, *changeroot*.

Из подробного описания алгоритма должно быть ясно, что ребро, по которому из фрагмента отправляется сообщение $\langle \textbf{connect}, L \rangle$ действительно является исходящим из фрагмента ребром наименьшего веса. Отсюда следует, что MST будет построено правильно, если каждый фрагмент действительно отправляет такое сообщение и соединяется с другим фрагментом, несмотря на ожидание, предусмотренное данным алгоритмом.

# Подробное описание алгоритма GHS

## Теорема.

Алгоритм GHS вычисляет минимальное остовное дерево, используя при этом не более $5N \log N + 2|E|$ обменов сообщениями.

## Доказательство.

Алгоритм построит MST, если мы докажем, что он не будет заблокирован.
Потенциальная возможность для взаимной блокировки возникает в тех ситуациях, когда узлы или фрагменты должны пребывать в ожидании того, что определенные условия будут выполнены в другом узле или в другом фрагменте. Ожидание в стержневых узлах, связанное с сообщениями $\langle \textbf{report}, \omega \rangle$, не приводит к блокировке, потому что каждый стержневой узел рано или поздно получит сводки от всех своих преемников (если только весь фрагмент целиком не пребывает в ожидании другого фрагмента), после чего указанное сообщение будет обработано.

# Подробное описание алгоритма GHS

Доказательство.

Рассмотрим случай, когда сообщение, отправленное из фрагмента $F_1 = (level_1, name_1)$ , достигает некоторого узла фрагмента $F_2 = (level_2, name_2)$ .

# Подробное описание алгоритма GHS

Доказательство.

Рассмотрим случай, когда сообщение, отправленное из фрагмента $F_1 = (level_1, name_1)$ , достигает некоторого узла фрагмента $F_2 = (level_2, name_2)$ .

Сообщение $\langle \textbf{connect}, level_1 \rangle$ должно ожидать обработки, если $level_1 \geq level_2$ и никакое сообщение $\langle \textbf{connect}, level_2 \rangle$ не было отправлено по тому же самому ребру из фрагмента $F_2$ (см. (2)).

# Подробное описание алгоритма GHS

Доказательство.

Сообщение $\langle \mathbf{test}, level_1, name_1 \rangle$ должно ожидать обработки, если $level_1 > level_2$ (см. (5)). Во всех тех случаях, когда фрагмент $F_1$ ожидает фрагмент $F_2$ , выполняется одно из следующих условий:

1. $level_1 > level_2$ ;
2. $level_1 = level_2 \wedge \omega(e_{F_1}) > \omega(e_{F_2})$ ;
3. $level_1 = level_2 \wedge \omega(e_{F_1}) = \omega(e_{F_2})$ и $F_2$ все еще пребывает в поиске исходящего из этого фрагмента ребра наименьшего веса. (Коль скоро $e_{F_1}$ является ребром, исходящим из $F_2$ , неравенство $\omega(e_{F_2}) > \omega(e_{F_1})$ выполняться не будет.)

Таким образом, взаимная блокировка по замкнутому циклу невозможна.

# Подробное описание алгоритма GHS

Доказательство.

Остается оценить сложность алгоритма.

- Оцените, сколько сообщений каждого типа отправляется процессами по ходу работы алгоритма в сети с $N$ узлами и $|E|$ ребрами.

# Подробное описание алгоритма GHS

Доказательство.

Остается оценить сложность алгоритма.

- Оцените, сколько сообщений каждого типа отправляется процессами по ходу работы алгоритма в сети с $N$ узлами и $|E|$ ребрами.
- На основании этой оценки покажите, что сложность алгоритма GHS по числу обменов сообщениями не превосходит величины $2N \log N + 5|E|$.

# Алгоритм Корача–Каттена–Морана

Между задачей о выборах и задачей обхода сети есть тесная взаимосвязь. Поэтому можно использовать общий метод построения эффективного алгоритма избрания лидера для произвольного класса сетей на основе всякого алгоритма обхода сетей.

В основу алгоритма Корача–Каттена–Морана положены идеи двух методов:

- **метода угасания**,
- **алгоритма Петерсона/Долева–Клейва–Роде**.

# Алгоритм Корача–Каттена–Морана

Сходство с **методом угасания** проявляется в том, что инициаторы выборов приступают к обходам сети, используя маркеры, помеченные их отличительными признаками.

В процессе обхода процессы могут поглощать маркеры. И если обход завершится, то инициатор этого обхода будет считаться избранным.

Но для этого завершиться должен только один обход. Все остальные обходы должны угаснуть.

# Алгоритм Корача–Каттена–Морана

Чтобы «гасить» обходы, алгоритм оперирует на разных уровнях , подобно тому как алгоритм Петерсона/Долева—Клейва—Роде разбивает свое вычисление на туры.

Если запущены по крайней мере два обхода, то один из маркеров рано или поздно достигнет того процесса, в котором уже успел побывать другой. Тогда обход, который проводится при помощи вновь прибывшего маркера, будет прерван.

Цель работы алгоритма состоит в том, чтобы привести две маркера вместе в один процесс, где они будут подавлены, и после этого может начаться новый обход.

# Алгоритм Корача–Каттена–Морана

**Принцип 1.**

Вместо туров в алгоритме Корача–Каттена–Морана используются уровни ; две маркера дают начало новому обходу только в том случае, если они находятся на одном и том же уровне, и при этом уровень вновь запущенного обхода будет на единицу больше.

Уровень процесса — это уровень того маркера, обход которого он поддерживает. Уровень процесса может изменяться при посещении процесса другими маркерами.

Если какой-либо маркер встречается с другим маркером более высокого уровня или достигает узла, который уже посетил маркер более высокого уровня, то этот прибывший маркер просто подавляется без всякого влияния на маркер более высокого уровня.

# Алгоритм Корача–Каттена–Морана

**Принцип 2.**

Чтобы привести маркеры одного и того же уровня совместно в один процесс, каждому маркеру предписывается один из трех режимов: захват (**annex**), погоня (**chase**) или ожидание .

Маркер инициируется в режиме **захвата**, и в этом режиме он подчиняется алгоритму обхода.

Взаимодействие с алгоритмом обхода происходит за счет обращения к функции *trav* : эта функция либо указывает того соседа, которому нужно переправить маркер, либо заявляет о принятии решения, если данный обход завершился.

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

1. Алгоритм обхода завершен, и процесс становится лидером.

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

1. Алгоритм обхода завершен, и процесс становится лидером.
2. Маркер подавляется в узле более высокого уровня.

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

1. Алгоритм обхода завершен, и процесс становится лидером.
2. Маркер подавляется в узле более высокого уровня.
3. Маркер достиг узла, в котором его ожидает другой маркер того же уровня; тогда оба маркера подавляются, и из этого узла начинается новый обход.

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

1. Алгоритм обхода завершен, и процесс становится лидером.
2. Маркер подавляется в узле более высокого уровня.
3. Маркер достиг узла, в котором его ожидает другой маркер того же уровня; тогда оба маркера подавляются, и из этого узла начинается новый обход.
4. Маркер достиг узла того же уровня, который посетил последним маркер с большим отличительным признаком или маркер в режиме **погони**; тогда наш маркер переходит в режим **ожидания** и остается в этом узле;

# Алгоритм Корача–Каттена–Морана

**Принцип 3.**

Маркер в режиме **захвата** подчиняется алгоритму обхода, пока не сложится одна из следующих ситуаций.

1. Алгоритм обхода завершен, и процесс становится лидером.
2. Маркер подавляется в узле более высокого уровня.
3. Маркер достиг узла, в котором его ожидает другой маркер того же уровня; тогда оба маркера подавляются, и из этого узла начинается новый обход.
4. Маркер достиг узла того же уровня, который посетил последним маркер с большим отличительным признаком или маркер в режиме **погони**; тогда наш маркер переходит в режим **ожидания** и остается в этом узле;
5. Маркер достиг узла того же уровня, в котором последним побывал маркер в режиме **захвата** с меньшим отличительным признаком; тогда наш маркер переходит в режим **погони** и переправляется по тому же каналу, по которому был отправлен предыдущий маркер.

# Алгоритм Корача–Каттена–Морана

**Принцип 4.**

Маркер в режиме **погони** переправляется в каждом узле по тому каналу, по которому был отправлен самый последний из посетивших этот узел маркеров, если только не складывается одна из следующих ситуаций.

# Алгоритм Корача–Каттена–Морана

**Принцип 4.**

Маркер в режиме **погони** переправляется в каждом узле по тому каналу, по которому был отправлен самый последний из посетивших этот узел маркеров, если только не складывается одна из следующих ситуаций.

1. Маркер достиг процесса более высокого уровня; тогда маркер подавляется.

# Алгоритм Корача–Каттена–Морана

**Принцип 4.**

Маркер в режиме **погони** переправляется в каждом узле по тому каналу, по которому был отправлен самый последний из посетивших этот узел маркеров, если только не складывается одна из следующих ситуаций.

1. Маркер достиг процесса более высокого уровня; тогда маркер подавляется.
2. Маркер достиг процесса, в котором хранится маркер того же уровня, пребывающий в режиме **ожидания**; тогда оба маркера изымаются и в этом процессе начинается новый обход.

# Алгоритм Корача–Каттена–Морана

**Принцип 4.**

Маркер в режиме **погони** переправляется в каждом узле по тому каналу, по которому был отправлен самый последний из посетивших этот узел маркеров, если только не складывается одна из следующих ситуаций.

1. Маркер достиг процесса более высокого уровня; тогда маркер подавляется.
2. Маркер достиг процесса, в котором хранится маркер того же уровня, пребывающий в режиме **ожидания**; тогда оба маркера изымаются и в этом процессе начинается новый обход.
3. Маркер достиг процесса того же уровня, и при этом последний из посетивших этот процесс маркеров пребывал в режиме **погони**; тогда наш маркер переходит в режим **ожидания**.

# Алгоритм Корача–Каттена–Морана

**Принцип 5.**

Маркер в режиме **ожидания** остается в том процессе, которого он достиг, до тех пор пока не сложится одна из следующих ситуаций.

# Алгоритм Корача–Каттена–Морана

**Принцип 5.**

Маркер в режиме **ожидания** остается в том процессе, которого он достиг, до тех пор пока не сложится одна из следующих ситуаций.

1. Маркер более высокого уровня достигнет того же самого процесса; тогда ожидающий маркер будет подавлен.

# Алгоритм Корача–Каттена–Морана

**Принцип 5.**

Маркер в режиме **ожидания** остается в том процессе, которого он достиг, до тех пор пока не сложится одна из следующих ситуаций.

1. Маркер более высокого уровня достигнет того же самого процесса; тогда ожидающий маркер будет подавлен.
2. Маркер того же самого уровня достиг того же самого процесса; тогда оба маркера изымаются, и в данном процессе начинается новый обход более высокого уровня.

# Алгоритм Корача–Каттена–Морана

**Задача.**

- Доказать, что алгоритм Корача–Каттена–Морана является алгоритмом избрания лидера.

# Алгоритм Корача–Каттена–Морана

**Задача.**

- Доказать, что алгоритм Корача–Каттена–Морана является алгоритмом избрания лидера.
- Оценить коммуникационную сложность алгоритм Корача–Каттена–Морана при условии, что в качестве процедуры обхода используется алгоритм Тарри.

КОНЕЦ ЛЕКЦИИ 9.