

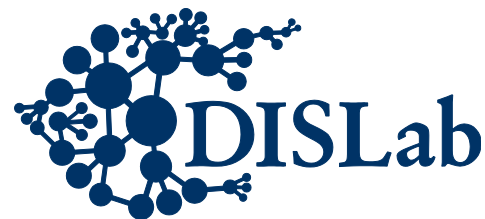
Спецкурс

«Параллельная обработка больших графов»

Лекция 3

к.т.н. Александр Сергеевич Семенов

dislab.org



Определения

- **Дерево** – связный граф без циклов
- **Остовное дерево связного графа** – подграф, являющийся деревом и связывающий все вершины исходного графа
- **Лес** – граф, каждая связная компонента которого является деревом
- **Остовный лес для графа G** – граф, являющийся объединением остовных деревьев всех компонент связности G

Задача построения минимального остовного дерева (Minimum Spanning Tree, MST)

- **Минимальный остовный лес графа G** – остовный лес, вес которого не превосходит вес любого другого возможного остовного дерева графа G
- **Minimum Spanning Tree**
 - Дан неориентированный граф $G = (V, E, W)$
 - $W: E \rightarrow R[0;1]$ – весовая функция
 - Найти минимальный остовный лес графа G

Алгоритмы решения задачи MST

- Прима, 1957
- Крускала, 1950/1930
- Борувки, 1926
- GHS (Gallager, Humblet, Spira), 1983

MST, жадные алгоритмы

MST (G, w)

$A = \{\}$ // множество ребер

while A не является остовным деревом

$(u, v) = \operatorname{argmin} W[(u, v)],$

$u \in$ ребру из $A, v \notin$ ни одному ребру из A

$A = A \cup \{(u, v)\}$

end while

MST, алгоритм Прима

```
MST-Prim (G, w, r) // G – связный граф
for all  $u \in V$  do  $u.key = \text{inf}$ ;  $u.par = \text{undef}$  end for
 $key[r] = 0$ 
 $Q = V$ 
 $A = \{\}$ 
while  $Q \neq \{\}$ 
     $u = \text{ExtractMin}(Q)$ 
     $A = A \cup \{(u.par, u)\}$ 
    for all  $v \in \text{Adj}[u]$ 
        if  $v \in Q$  and  $w(u, v) < key[v]$ 
             $v.key = w(u, v)$ ;  $v.par = u$ 
            // с вызовом  $\text{DecreaseKey}(Q, v, w(u, v))$ 
        end if
    end for
end while
```

Сложность $O(M \lg N)$, $O(M+N \lg N)$

MST, алгоритм Крускала

MST-Kruskal (G, w, r) // G – связный граф

$A = \{\}$

for all $u \in V$ **do** MakeSet(u) **end for**

Sort(E)

for all $(u, v) \in E$

if FindSet(u) \neq FindSet(v)

$A = A \cup \{(u, v)\}$

 Union(u, v)

end if

end for

Сложность $O(M \lg N)$

Система непересекающихся множеств

Также известна как Union-Find

1. **MakeSet(v)** создать отдельное множество из элемента v
2. **FindSet(v)** $\rightarrow u$ найти элемент-представитель множества, содержащего вершину v
3. **Union(u, v)** объединить множества, содержащие элементы u и v

Реализация Union-Find

MakeSet (v)

v.parent = v

FindSet (v)

if v.parent == v

return v

end if

return FindSet(v.parent)

Union (u, v)

x = FindSet(u)

y = FindSet(v)

x.parent = y

Оптимизация Union-Find

MakeSet (v)

v.parent = v

v.rank = 0

FindSet (v):

if v != v.parent

v.parent = FindSet(v.parent)

end if

return v.parent

Оптимизация Union-Find

Union (u,v)

Link(FindSet(u), FindSet(v))

Link (x, y)

if x.rank > y.rank

 y.parent = x

else

 x.parent = y

if x.rank == y.rank

 y.rank = y.rank + 1

end if

end if

Сложность $O(m \alpha(N)) \leq O(4m)$,

m – количество операций

MST, алгоритм Борувки

MST-Boruvka (G, w, r) // G – связный граф

$T = \{\{v_0\}, \{v_1\}, \dots\}$

while $|T| > 1$

for all component $C_i \in T$

$(u, v) = \operatorname{argmin} W[(u, v)], u \in C_i, v \notin C_i$

$C_i = C_i \cup \{(u, v)\}$

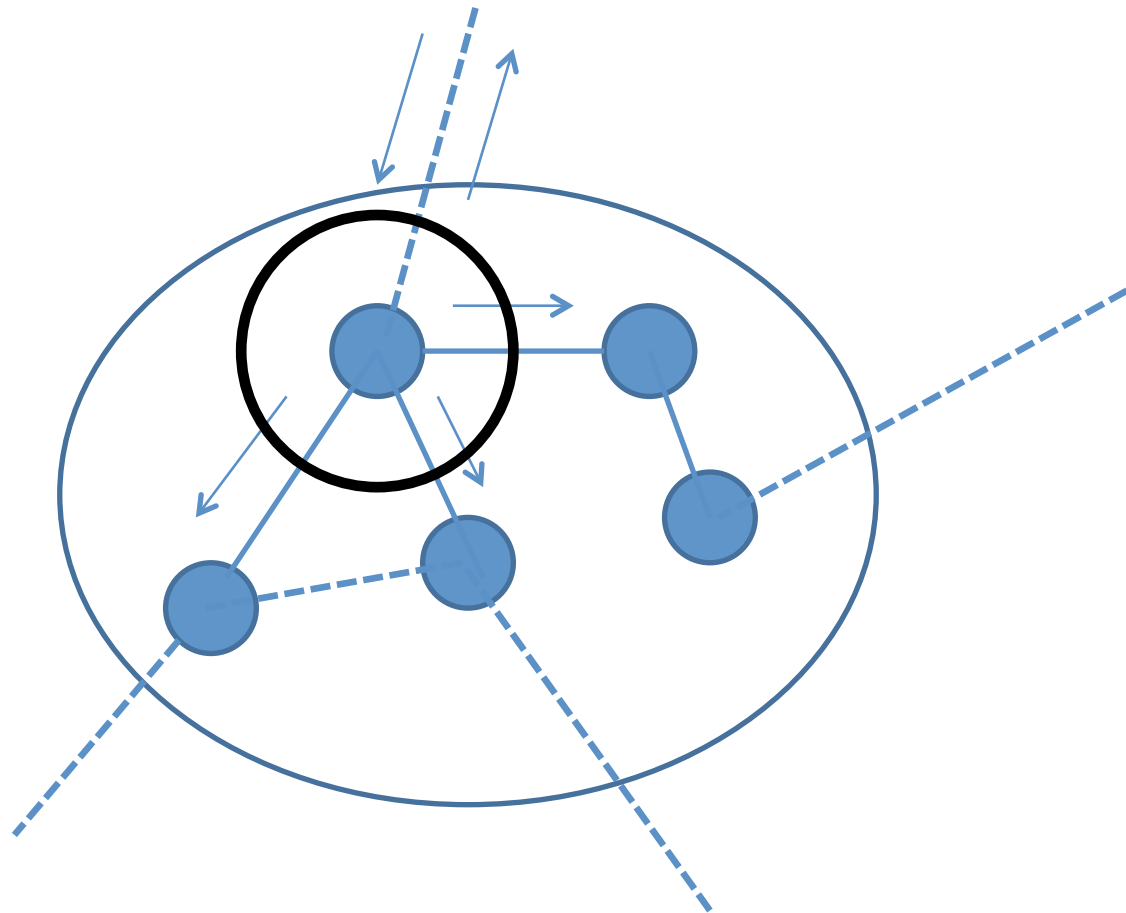
end for

end while

Сложность **$O(M \lg N)$**

Асинхронный алгоритм GHS (R. Gallager, P. Humblet, P. Spira, 1983)

- Веса должны быть различны, граф - СВЯЗНЫЙ



Асинхронный алгоритм GHS (R. Gallager, P. Humblet, P. Spira, 1983)

The Algorithm (As Executed at Each Node)

(1) Response to spontaneous awakening (can occur only at a node in the sleeping state)

execute procedure *wakeup*

(2) procedure *wakeup*

```
begin let m be adjacent edge of minimum weight;
SE(m) ← Branch;
LN ← 0;
SN ← Found;
Find-count ← 0;
send Connect(O) on edge m
end
```

(3) Response to receipt of *Connect(L)* on edge j

```
begin if SN = Sleeping then execute procedure wakeup;
if L < LN
then begin SE(j) ← Branch;
send Initiate(LN, FN, SN) on edge j;
if SN = Find then
find-count ← find-count + 1
end
else if SE(j) = Basic
then place received message on end of queue
else send Initiate(LN + 1, w(j), Find) on edge j
end
```

(4) Response to receipt of *Initiate(L, F, S)* on edge j

```
begin LN ← L; FN ← F; SN ← S; in-branch ← j;
best-edge ← nil; best-wt ← ∞;
for all i ≠ j such that SE(i) = Branch
do begin send Initiate(L, F, S) on edge i;
if S = Find then find-count ← find-count + 1
end;
if S = Find then execute procedure test
end
```

(5) procedure *test*

```
if there are adjacent edges in the state Basic
then begin test-edge ← the minimum-weight adjacent edge in state
Basic;
send Test(LN, FN) on test-edge
end
else begin test-edge ← nil; execute procedure report end
```

(6) Response to receipt of *Test(L, F)* on edge j

```
begin if SN = Sleeping then execute procedure wakeup;
if L > LN then place received message on end of queue
else if F ≠ FN then send Accept on edge j
else begin if SE(j) = Basic then SE(j) ← Rejected;
if test-edge ≠ j then send Reject on edge j
else execute procedure test
end
end
```

(7) Response to receipt of *Accept* on edge j

```
begin test-edge ← nil;
if w(j) < best-wt
then begin best-edge ← j; best-wt ← w(j) end;
execute procedure report
end
```

(8) Response to receipt of *Reject* on edge j

```
begin if SE(j) = Basic then SE(j) ← Rejected;
execute procedure test
end
```

(9) procedure *report*

```
if find-count = 0 and test-edge = nil
then begin SN ← Found;
send Report(best-wt) on in-branch
end
```

(10) Response to receipt of *Report(w)* on edge j

```
if j ≠ in-branch
then begin find-count ← find-count - 1
if w < best-wt then begin best-wt ← w; best-edge ← j end;
execute procedure report
end
else if SN = Find then place received message on end of queue
else if w > best-wt
then execute procedure change-core
else if w = best-wt = ∞ then halt
```

(11) procedure *change-core*

```
if SE(best-edge) = Branch
then send Change-core on best-edge
else begin send Connect(LN) on best-edge;
SE(best-edge) ← Branch
End
```

(12) Response to receipt of *Change-core*

```
execute procedure change-core
```

Объединение фрагментов

- У каждого фрагмента есть переменная – уровень L
- Сначала уровень каждого фрагмента $L = 0$
- Два фрагмента с одинаковым уровнем L могут объединиться во фрагмент с уровнем $L + 1$
- Фрагмент не может присоединиться к фрагменту с меньшим уровнем

Состояния вершин и ребер

Состояния ребер

- Rejected – ребро не является частью MST
- Branch – ребро является частью MST
- Basic – статус ребра еще не определен

Состояния вершин

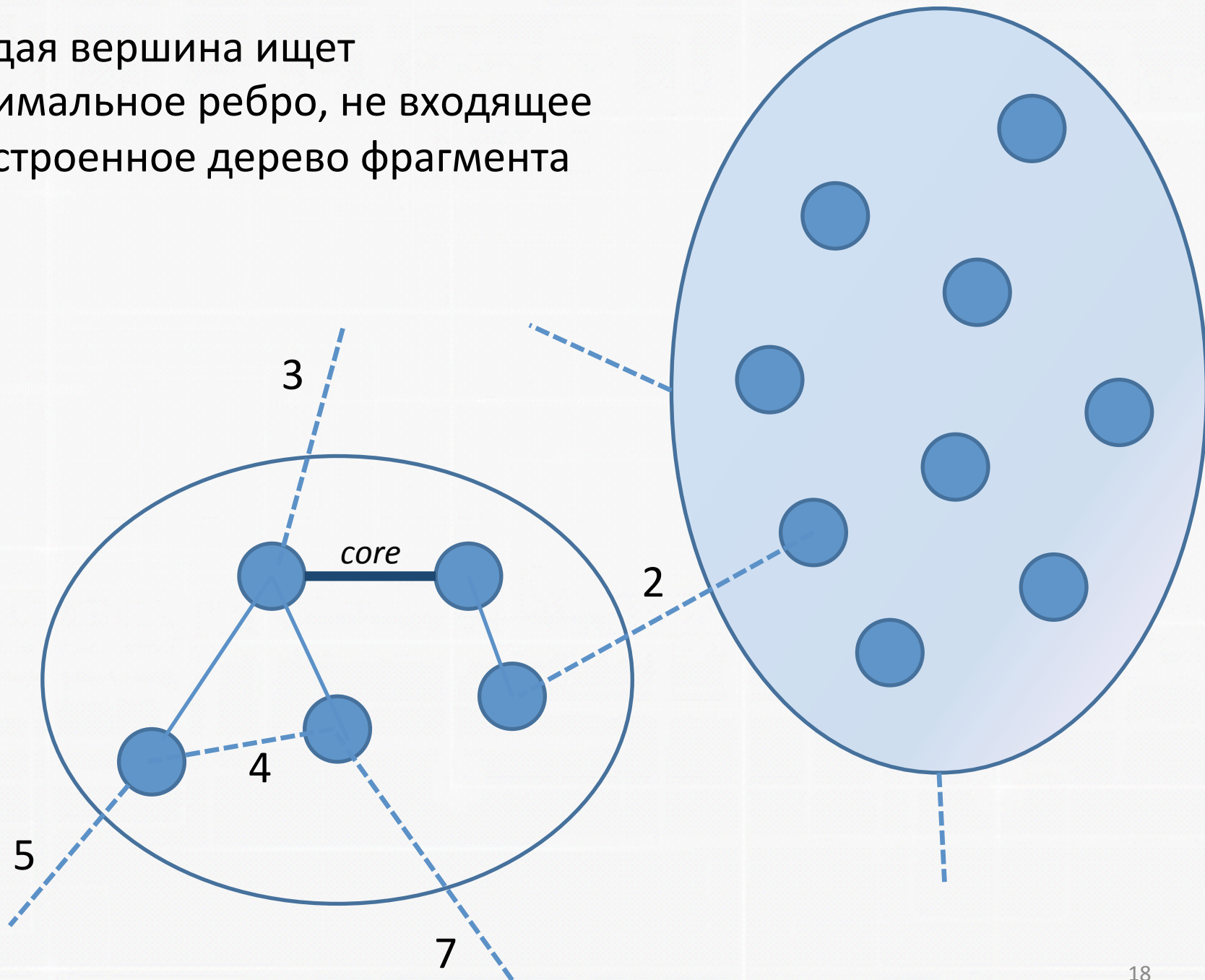
- Sleeping – начальное состояние
- Find – происходит поиск минимального ребра
- Found – минимальное ребро найдено

Сообщения между вершинами

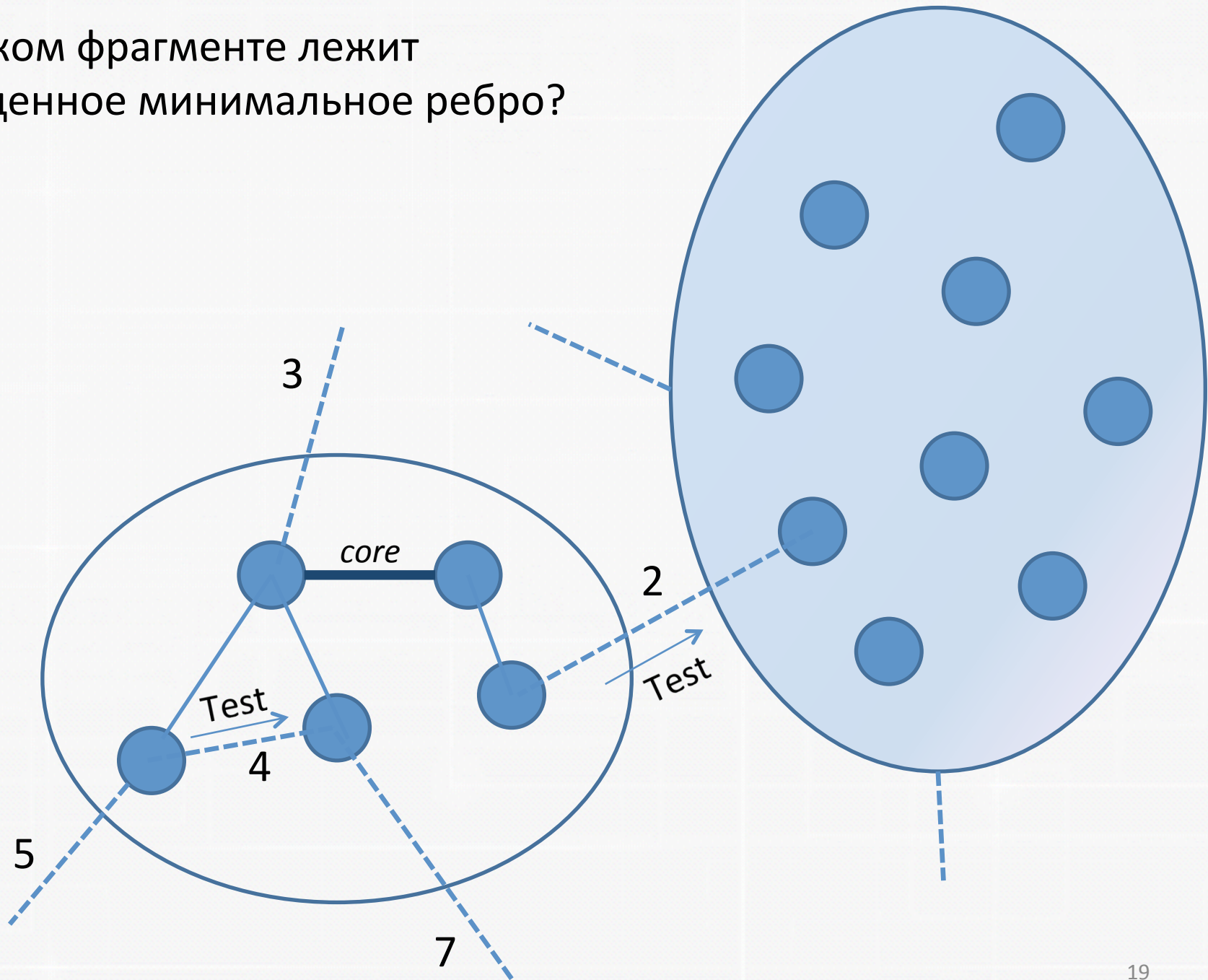
Всего 7 типов сообщений:

- Test (различные фрагменты?)
 - Assert (различные)
 - Reject (один и тот же фрагмент)
- Report (отчет о наименьшем ребре)
- Change core (переход от корня к наилучшему ребру внутри фрагмента)
- Connect (запрос об объединении)
- Initiate (обновление данных и начало нового поиска наилучших ребер)

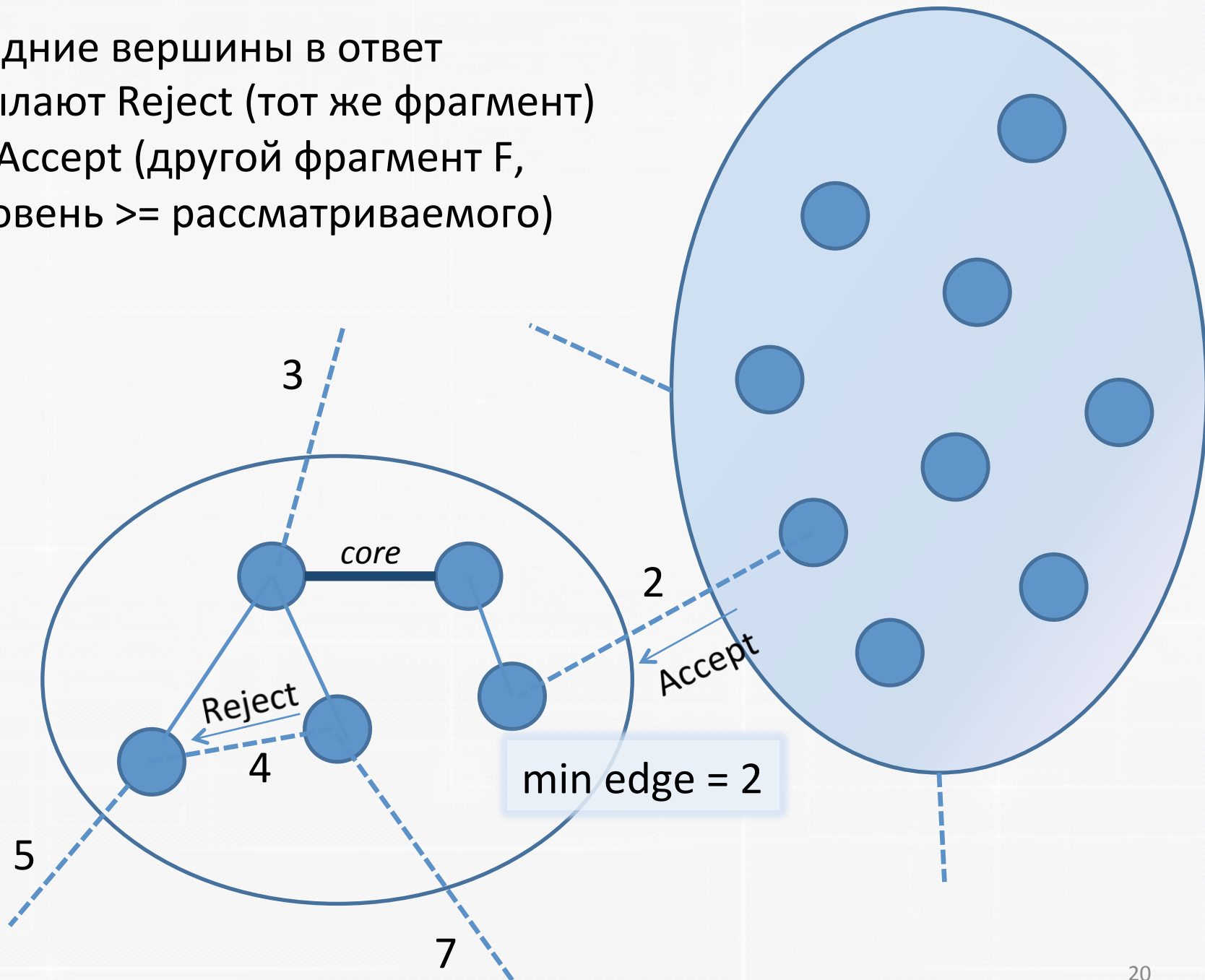
Каждая вершина ищет
минимальное ребро, не входящее
в построенное дерево фрагмента



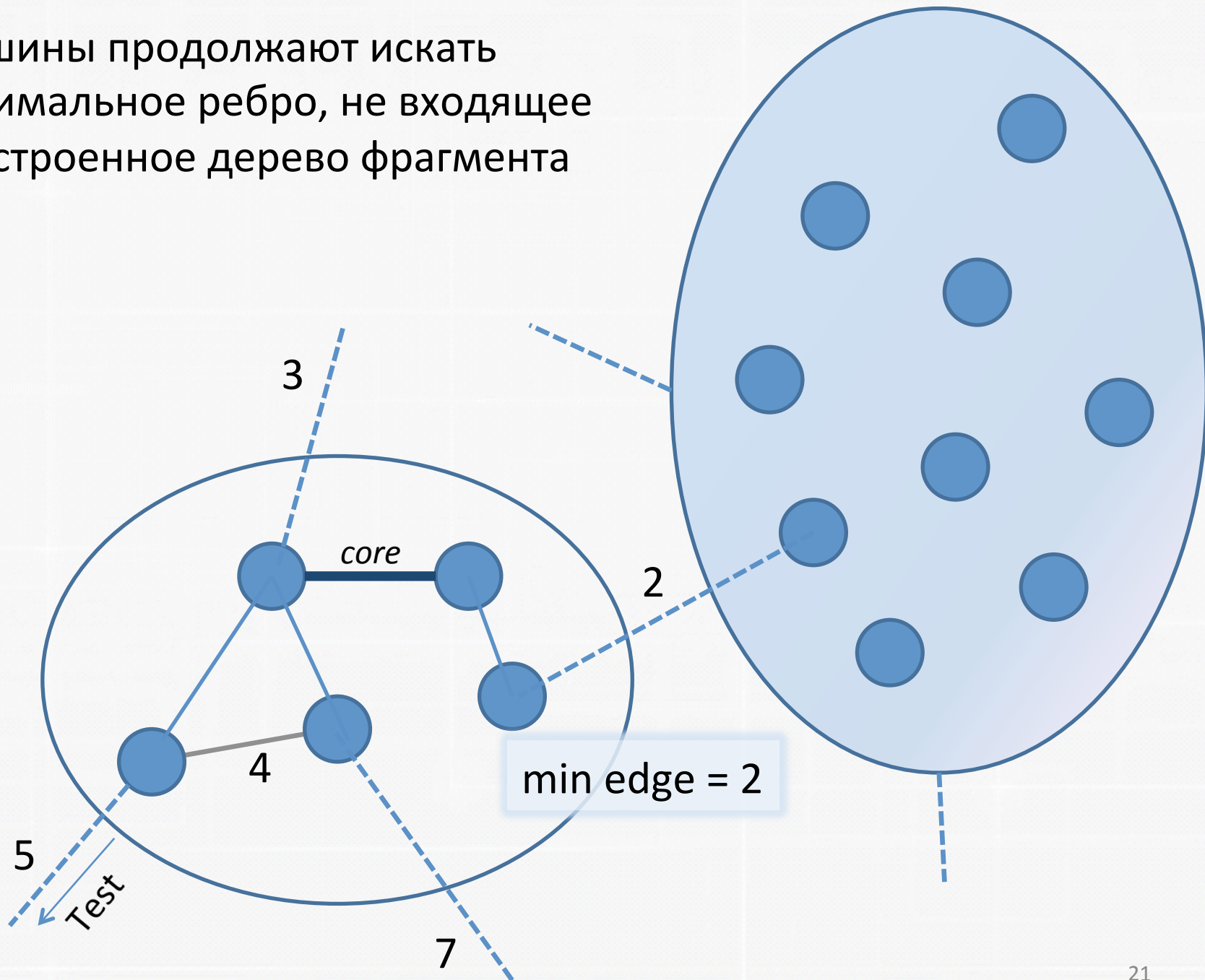
В каком фрагменте лежит найденное минимальное ребро?



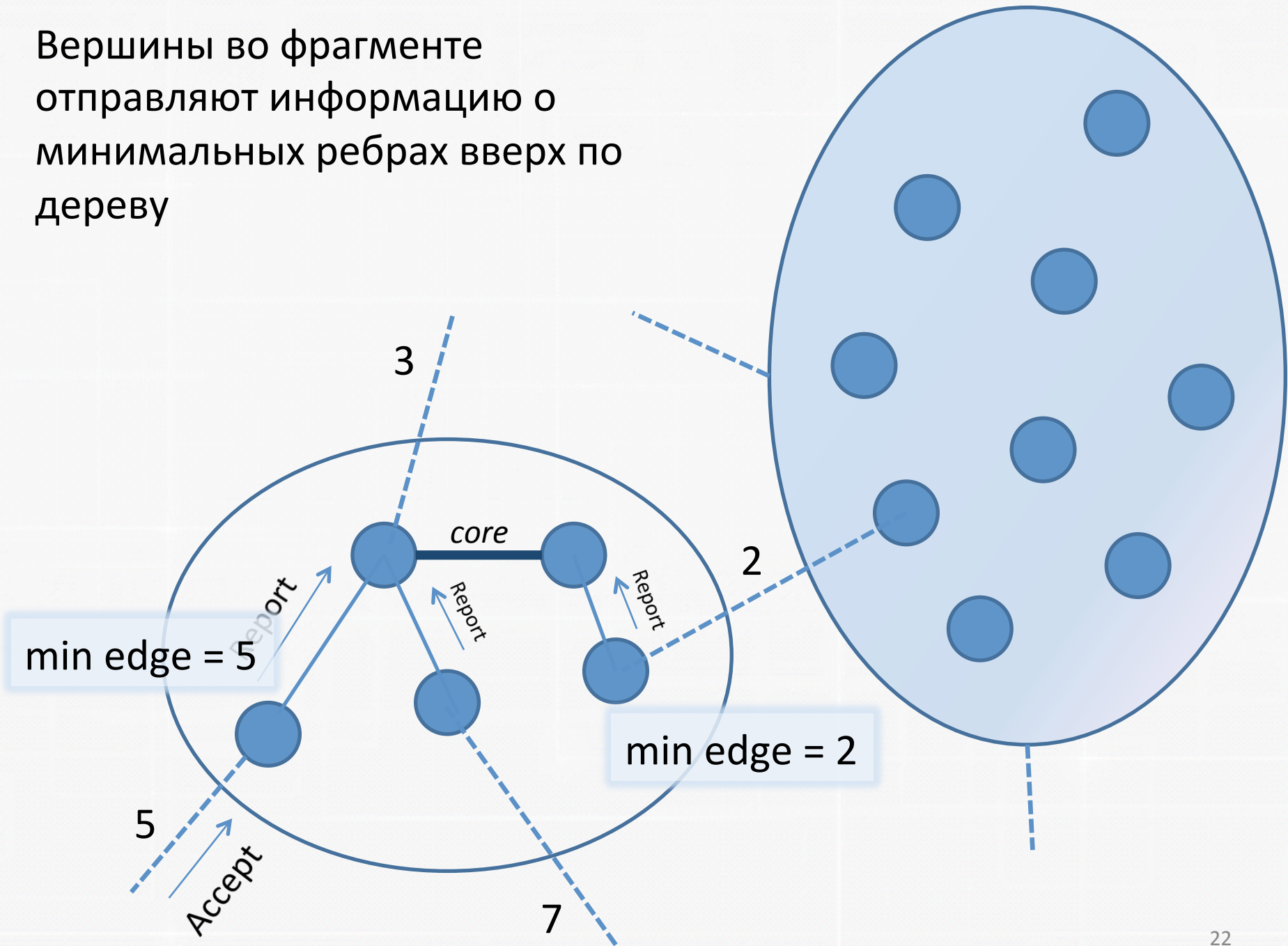
Соседние вершины в ответ посылают Reject (тот же фрагмент) или Ассерт (другой фрагмент F, F.уровень \geq рассматриваемого)



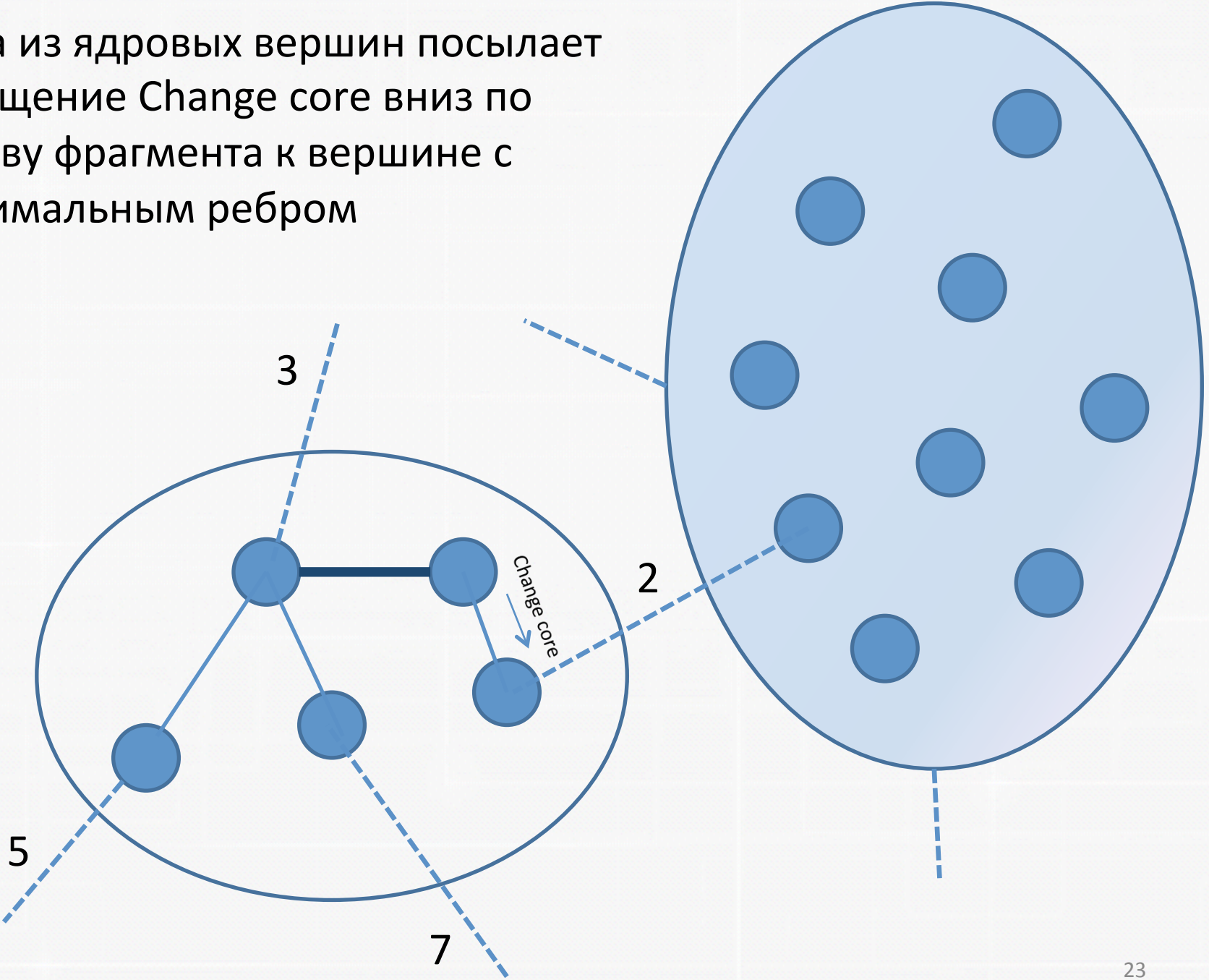
Вершины продолжают искать минимальное ребро, не входящее в построенное дерево фрагмента



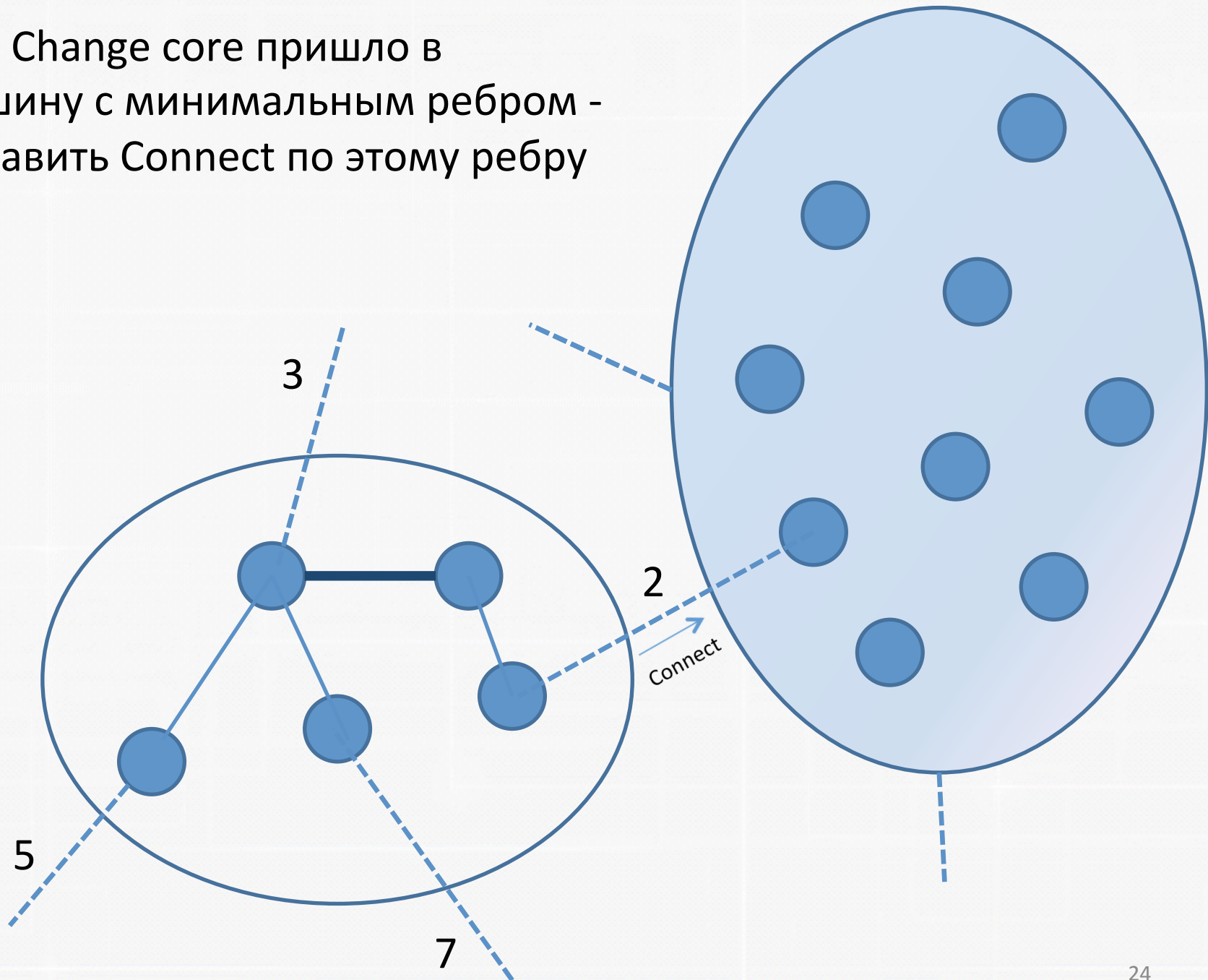
Вершины во фрагменте
отправляют информацию о
минимальных ребрах вверх по
дереву



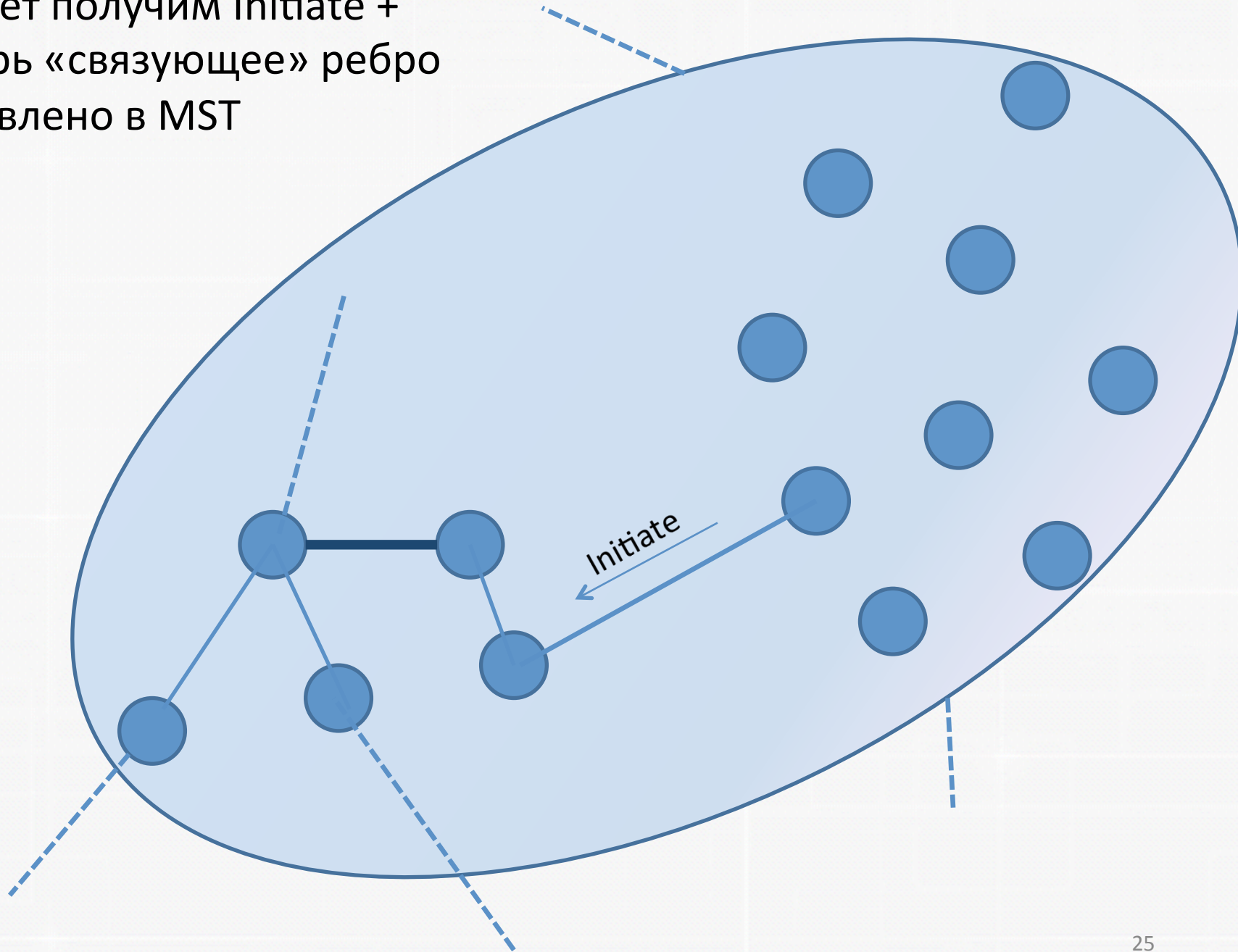
Одна из ядровых вершин посылает сообщение Change core вниз по дереву фрагмента к вершине с минимальным ребром



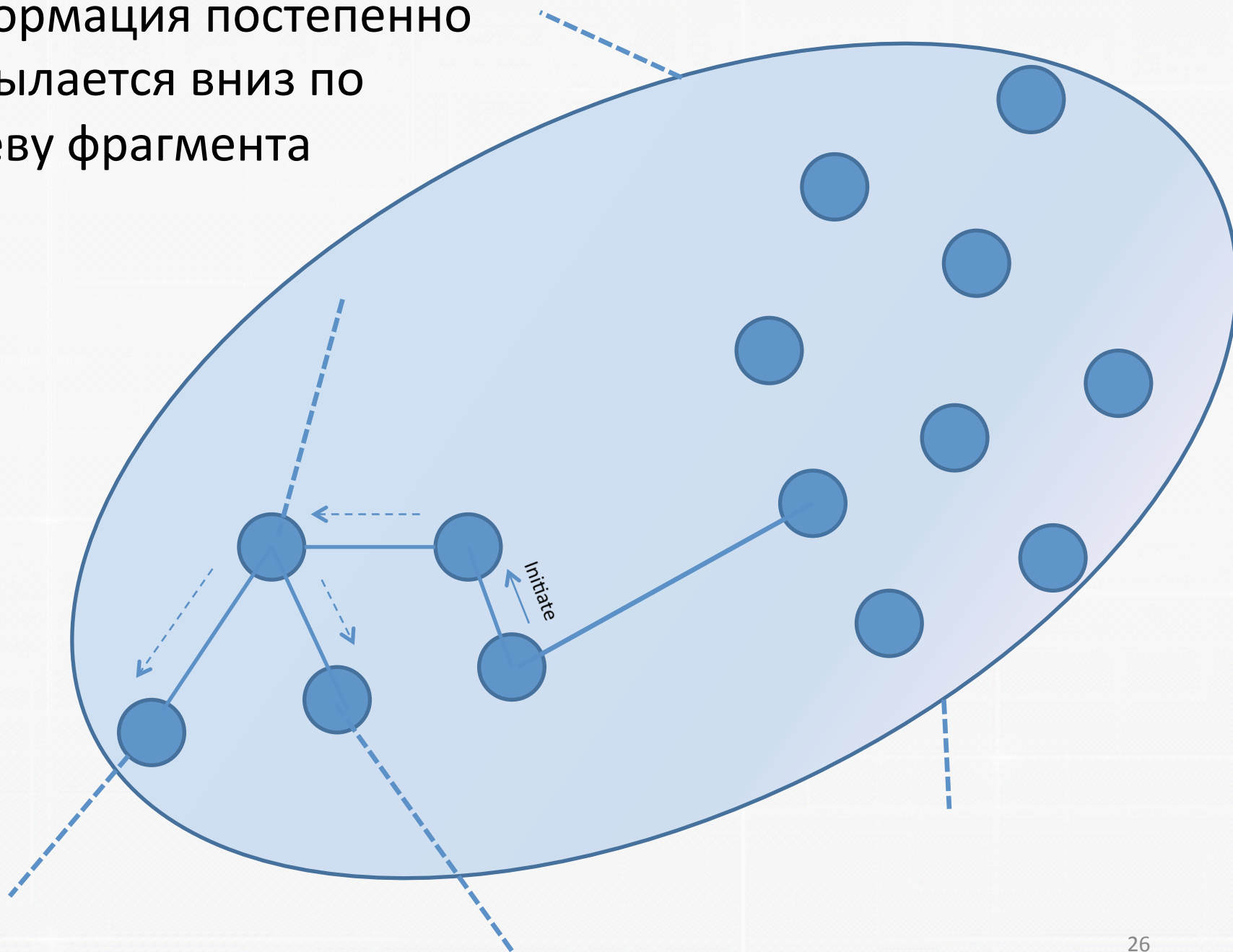
Если Change core пришло в
вершину с минимальным ребром -
отправить Connect по этому ребру



В ответ получим Initiate +
теперь «связующее» ребро
добавлено в MST



Информация постепенно
рассылается вниз по
дереву фрагмента



Асинхронный алгоритм GHS (R. Gallager, P. Humblet, P. Spira, 1983)

- Количество сообщений – не более $5N \log_2 N + 2M$
- Время – $5N \log_2 N$

Вопросы?

alxdr.semenov@gmail.com