Параллельная обработка больших графов

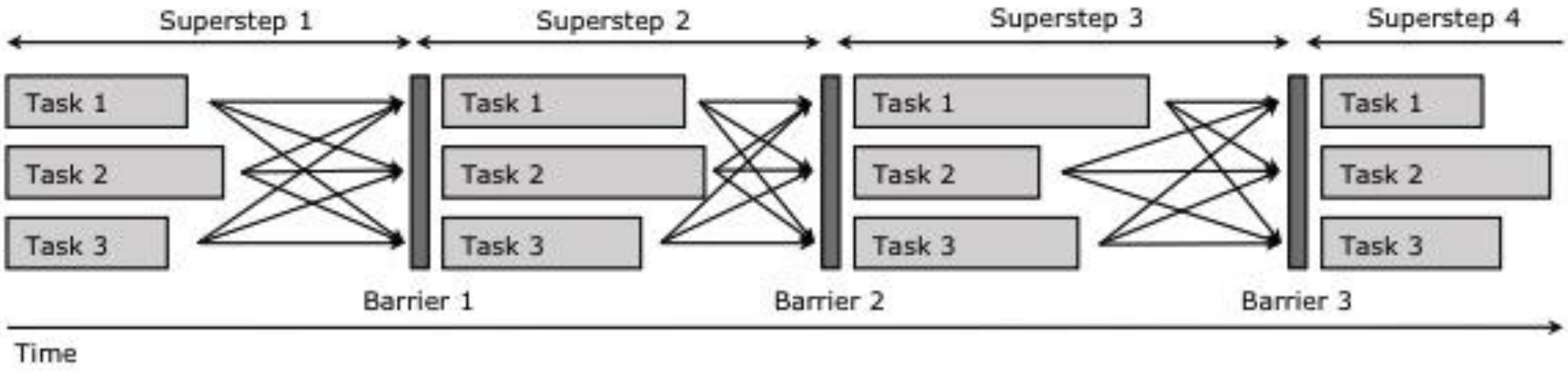
Занятие 3

А.С. Семенов

dislab.org

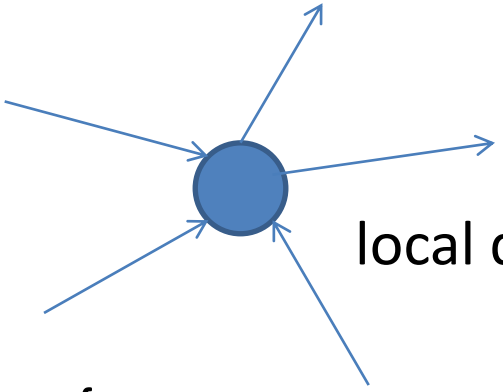
Вычислительные модели

BSP (Bulk Synchronous Parallel), 1988

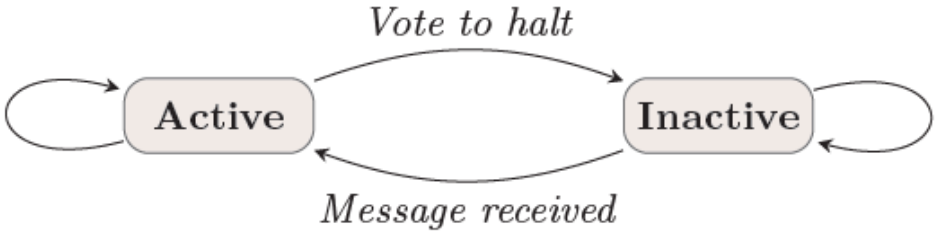


Вычислительные модели

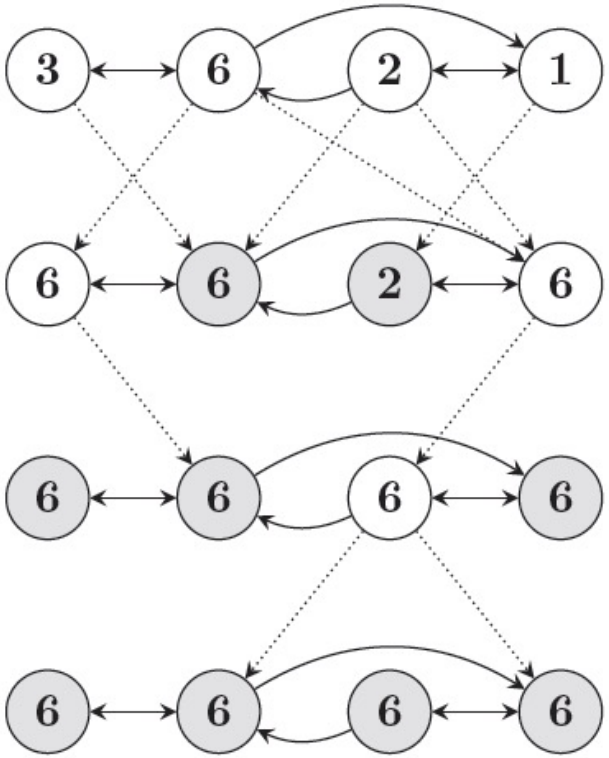
Vertex-centric [Pregel, 2010]



local data



Пример: Maximum value



VertexProgram {

Receive Messages()

do что-нибудь

Send Messages()

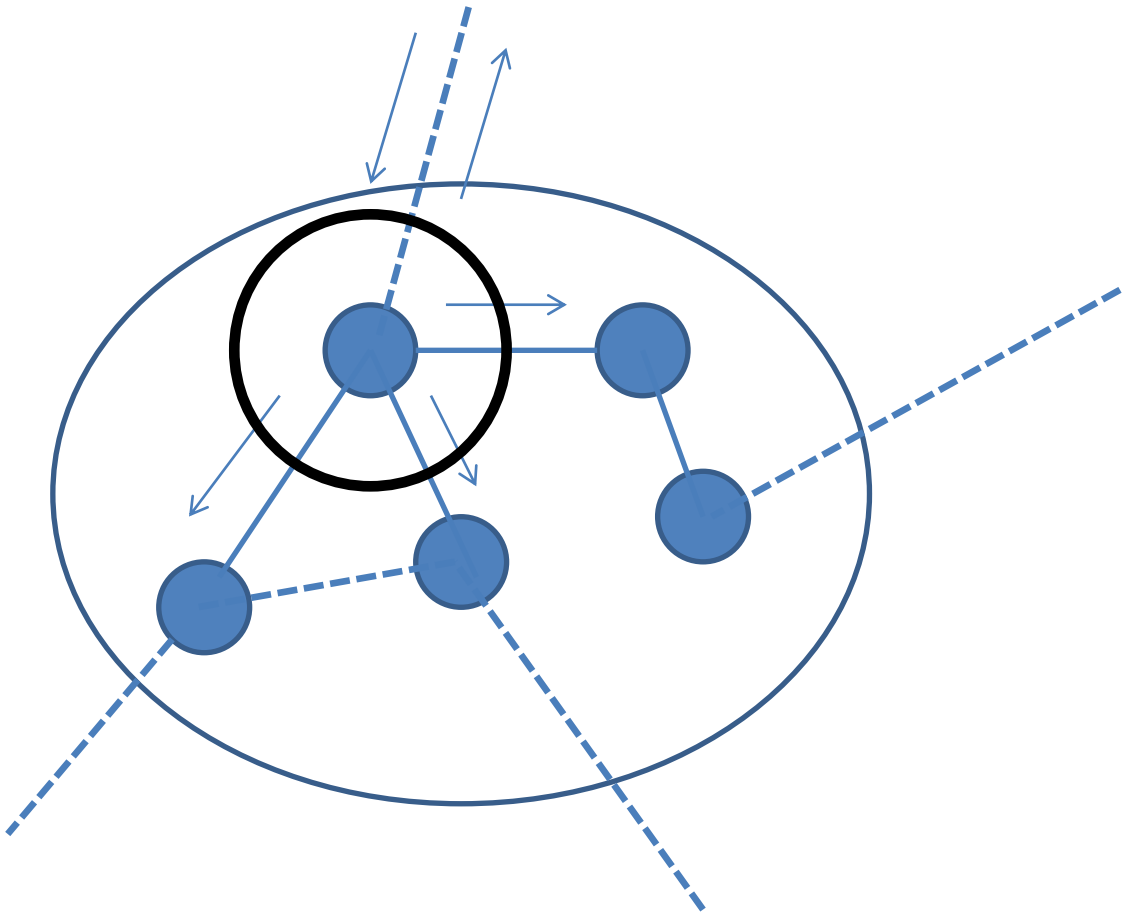
}

- Сообщения не обгоняют друг друга
- Синхронность: отсутствие дедлоков и race conditions

[2010] G. Malewicz etc. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* PDF

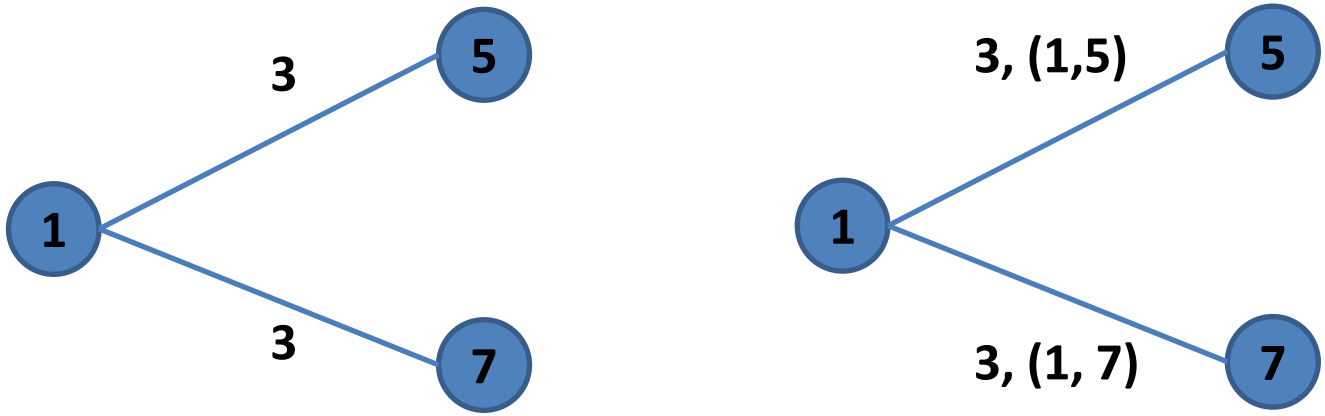
Асинхронный алгоритм GHS (R. Gallager, P. Humblet, P. Spira, 1983)

- Веса должны быть различны, граф - связный



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

- 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

- Фрагмент имеет идентификатор (вес ядрового ребра)
- У каждого фрагмента есть переменная – уровень L
- Сначала уровень каждого фрагмента L = 0
- Два фрагмента с одинаковым уровнем L могут объединится во фрагмент с уровнем L + 1
- Фрагмент не может присоединиться к фрагменту с меньшим уровнем

Состояния вершин и ребер

Состояния ребер

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

Состояния вершин

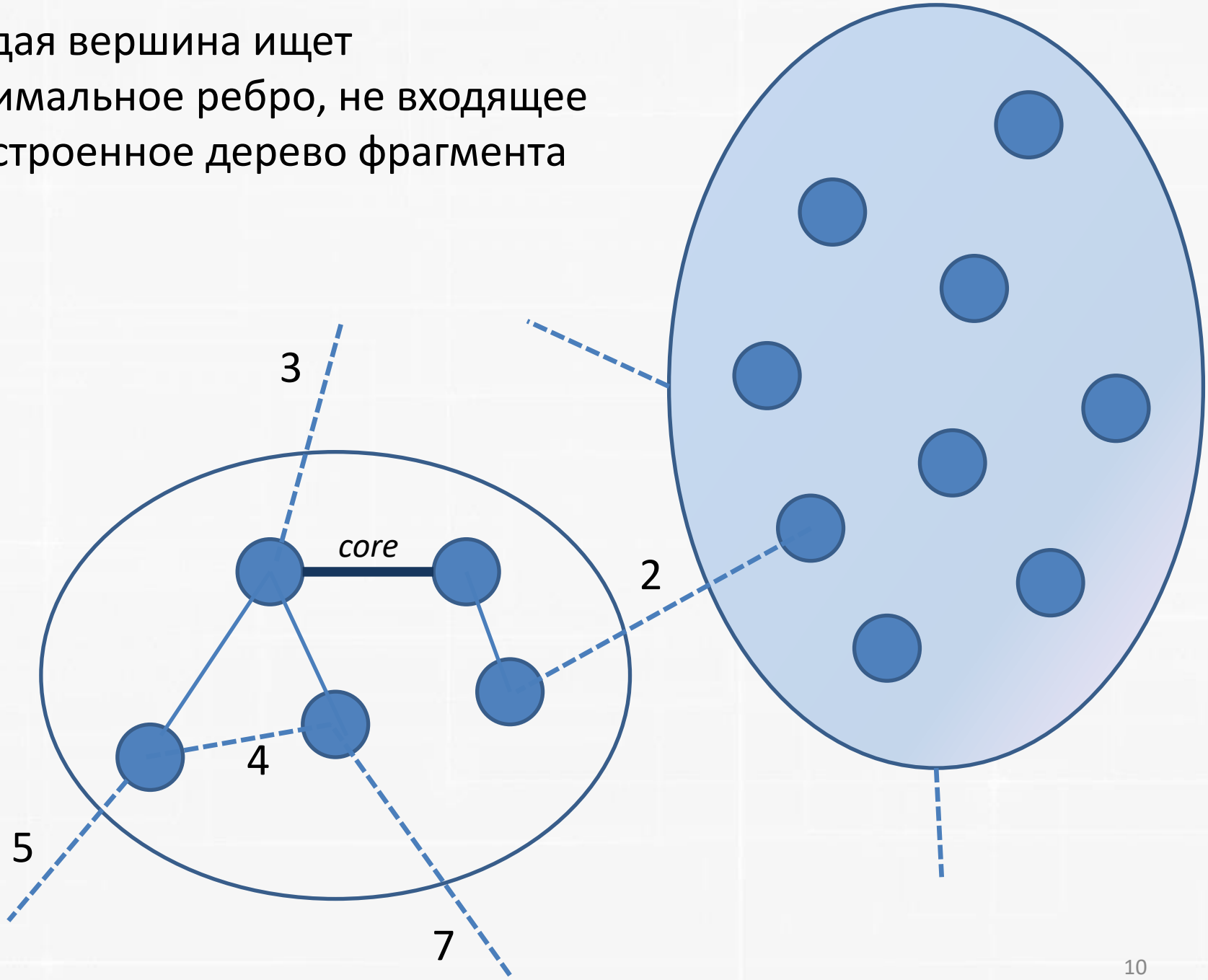
- Sleeping – начальное состояние
- Find – происходит поиск минимального ребра
- Found – минимальное ребро найдено

Сообщения между вершинами

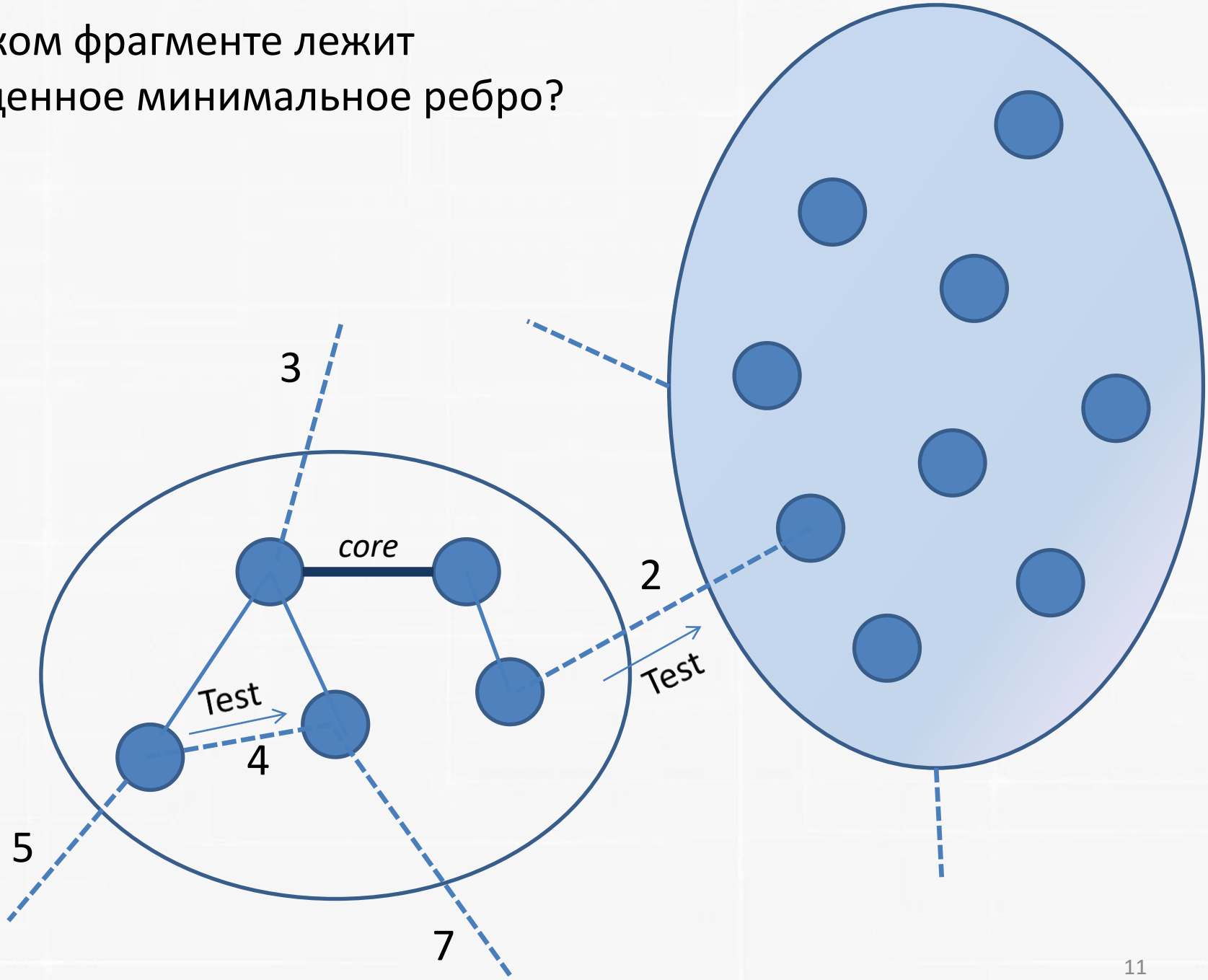
Всего 7 типов сообщений:

- Test (различные фрагменты?)
  - Accept (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)

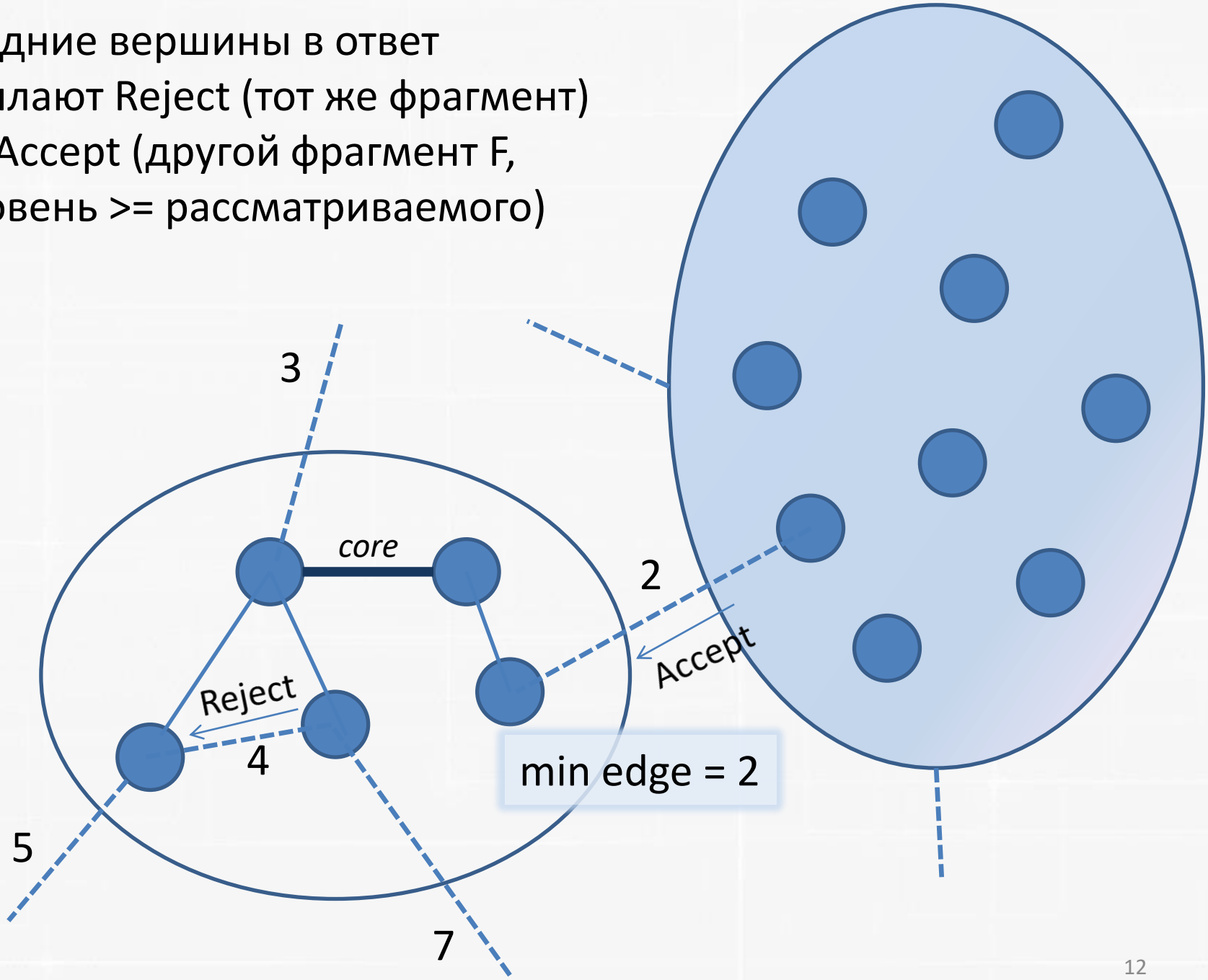
Каждая вершина ищет минимальное ребро, не входящее в построенное дерево фрагмента



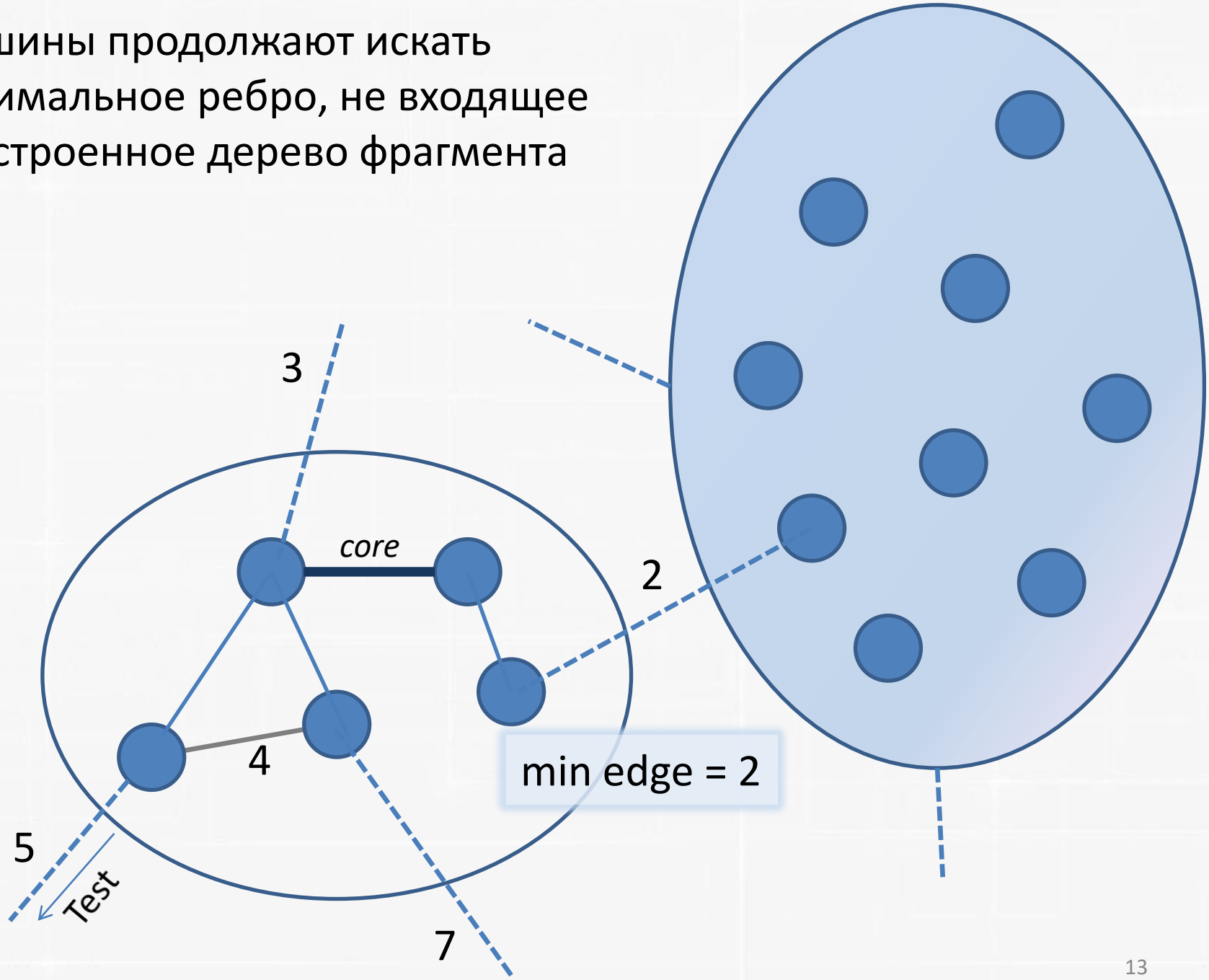
В каком фрагменте лежит найденное минимальное ребро?



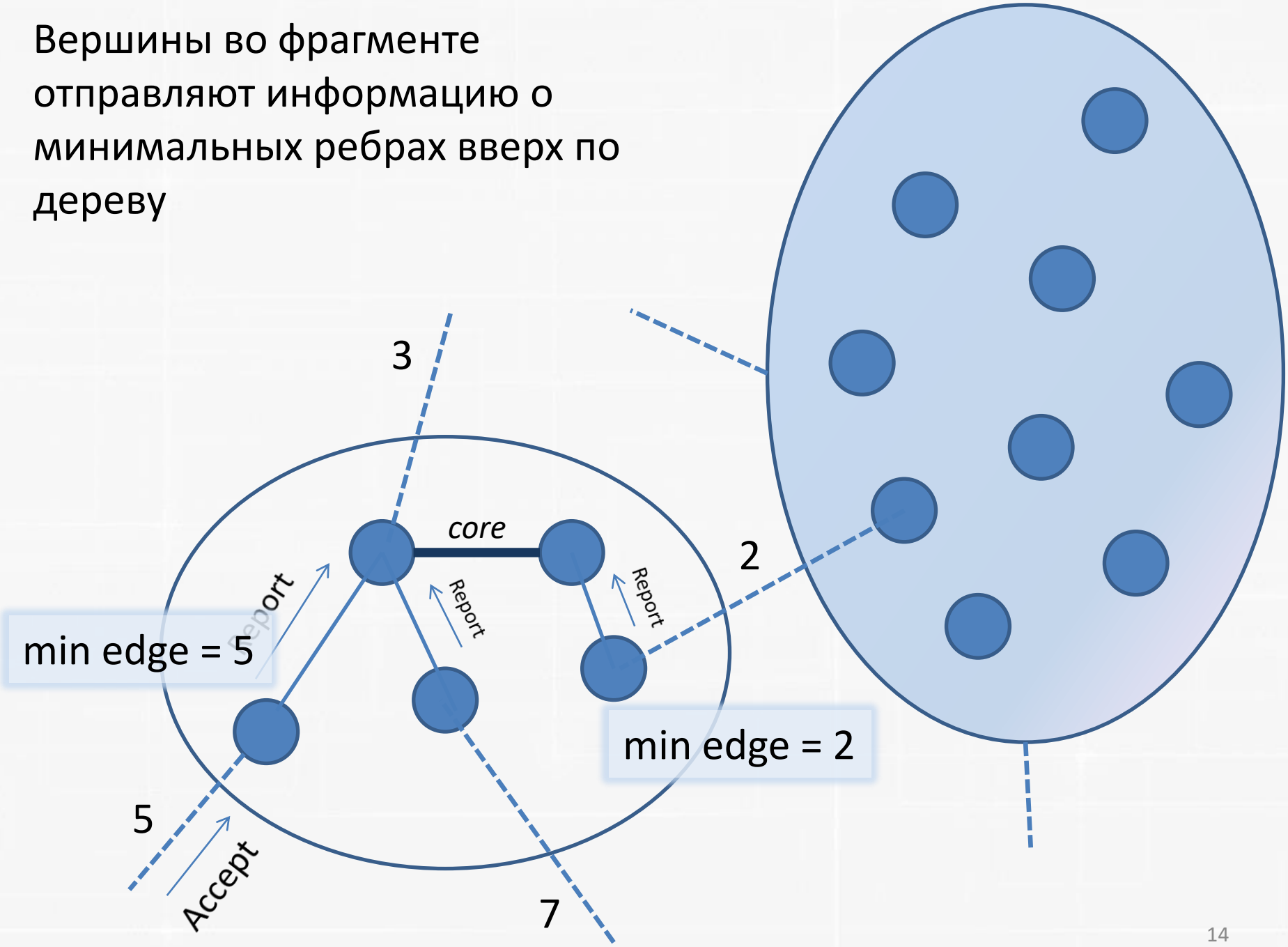
Соседние вершины в ответ посылают Reject (тот же фрагмент) или Accept (другой фрагмент F, F.уровень >= рассматриваемого)



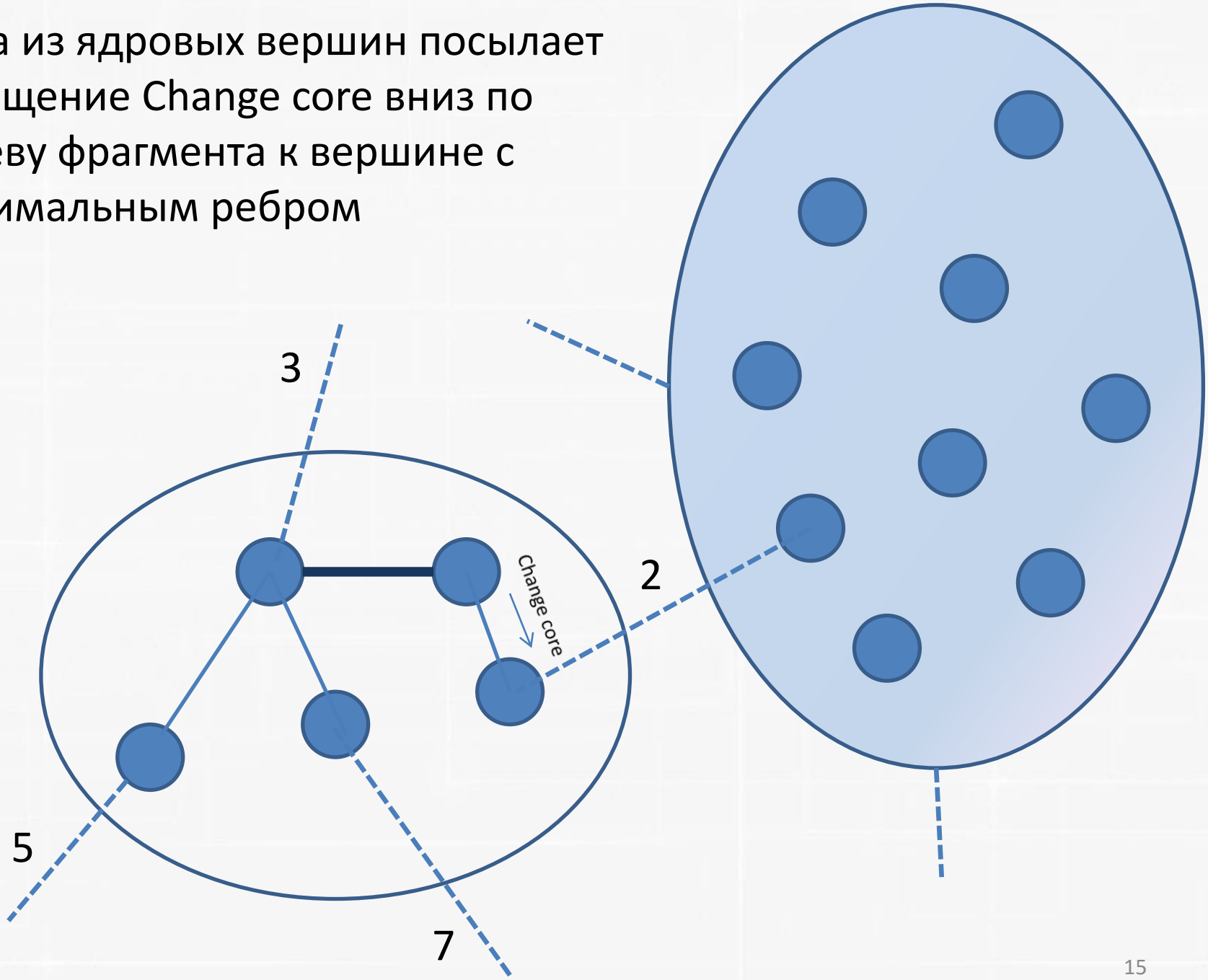
Вершины продолжают искать минимальное ребро, не входящее в построенное дерево фрагмента



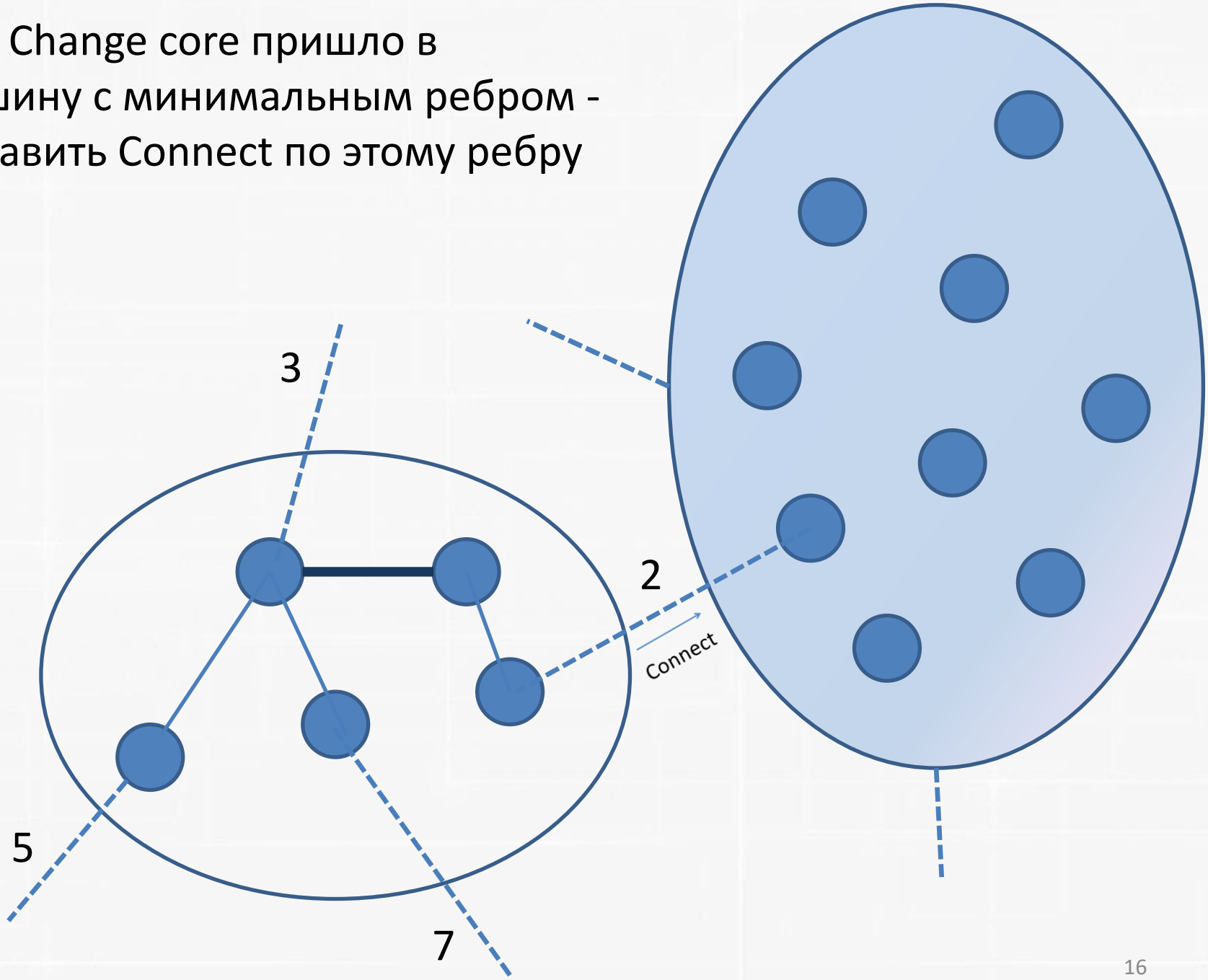
Вершины во фрагменте отправляют информацию о минимальных ребрах вверх по дереву



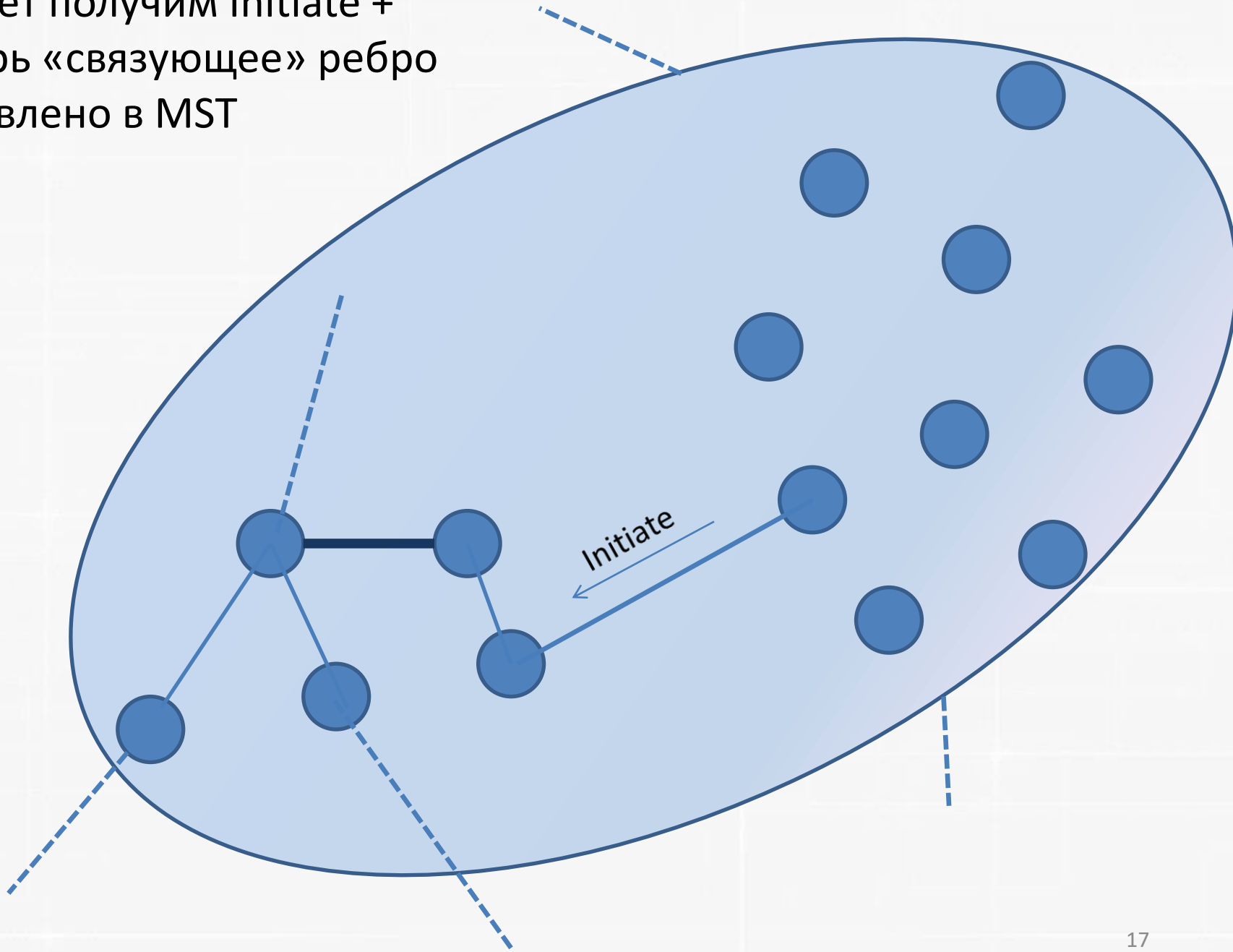
Одна из ядровых вершин посылает сообщение Change core вниз по дереву фрагмента к вершине с минимальным ребром



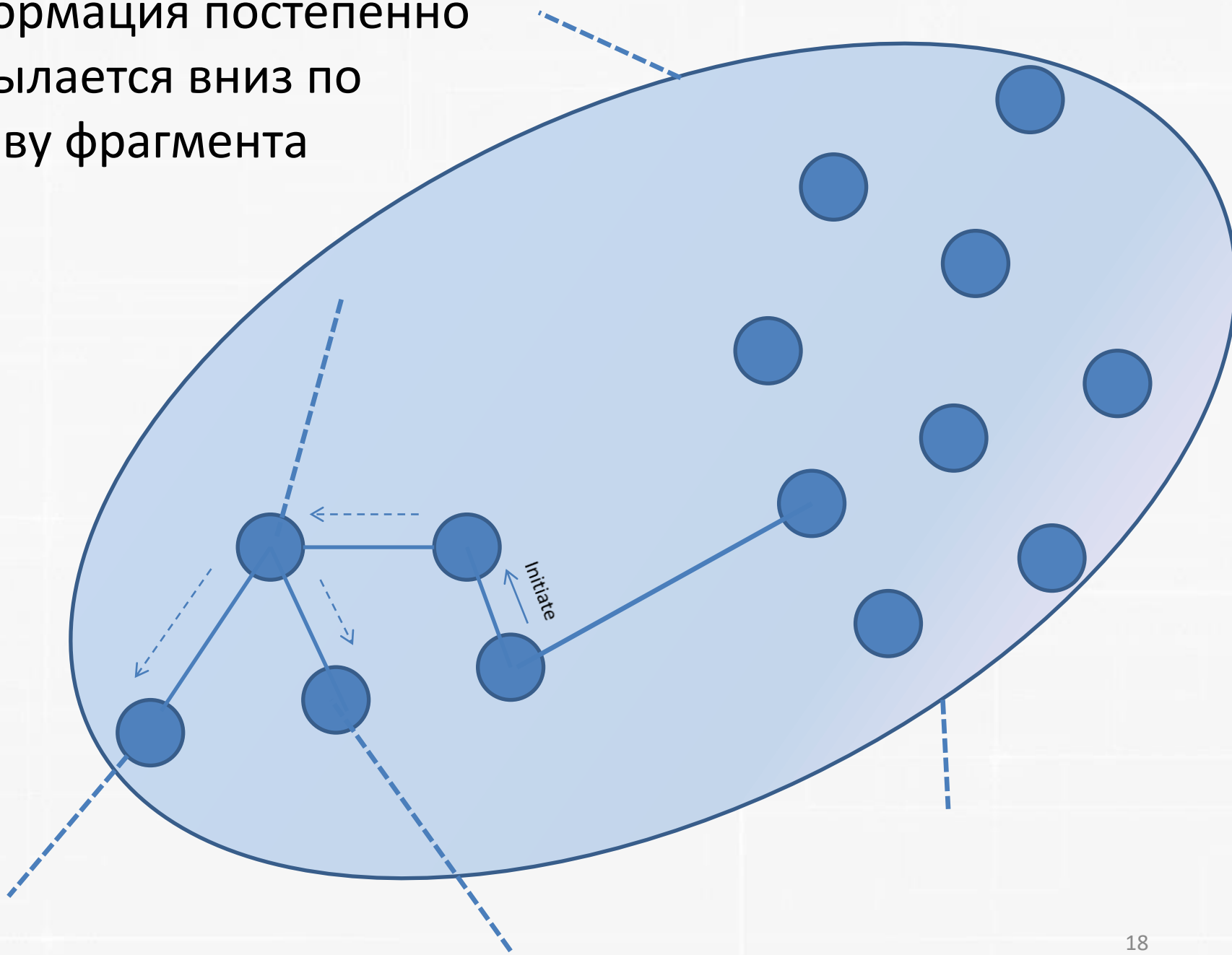
Если Change core пришло в вершину с минимальным ребром - отправить Connect по этому ребру



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



Информация постепенно рассылается вниз по дереву фрагмента



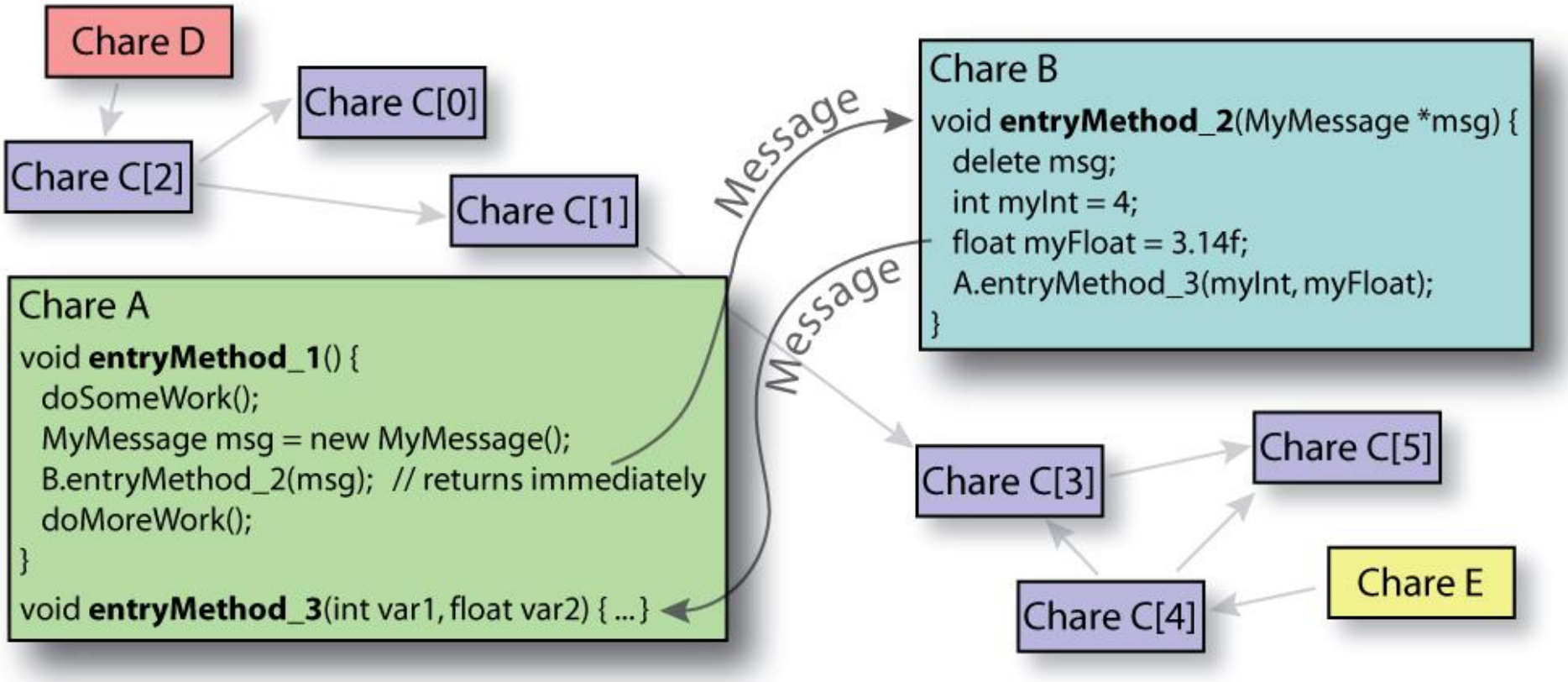
Асинхронный алгоритм GHS (R. Gallager, P. Humblet, P. Spira, 1983)

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

Charm++

- University of Illinois, с 90х годов, open-source
- Текущая версия: 7.0.0
- C++, объектно-ориентированный, порт для Python
- Глобальная видимость объектов
- Управление потоком асинхронных сообщений
- Балансировка нагрузки, библиотека автоматической агрегации

Charm++: application's view



Charm++: system view