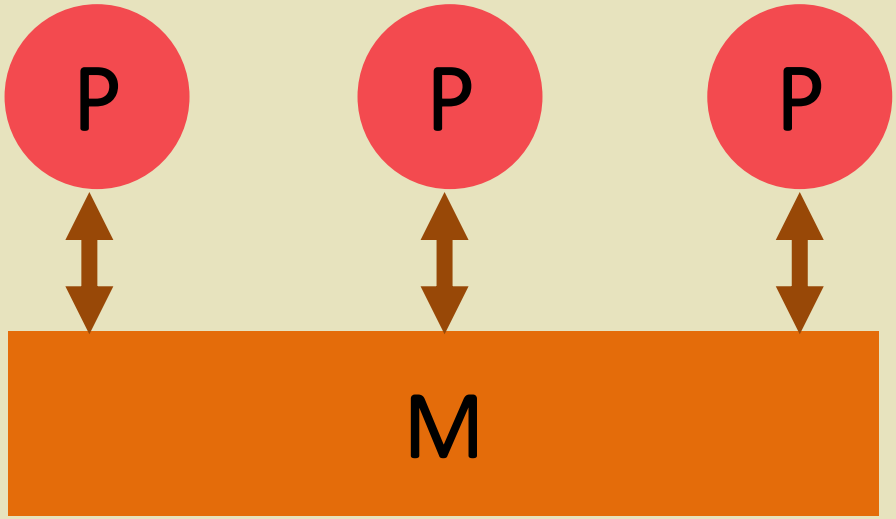
Алгоритмы реализации коллективных операций для распределенных массивов в разделенном глобальном адресном пространстве

Пазников А.А., Курносов М.Г., Куприянов М.С.
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)

II Международная научная конференция по проблемам управления в технических системах (ПУТС-2017)
25-27 октября 2017 г., г. Санкт-Петербург, СПбГЭТУ «ЛЭТИ»

Программные модели вычислительных систем



Общая память

(PThreads, C++-threads, OpenMP, Cilk)

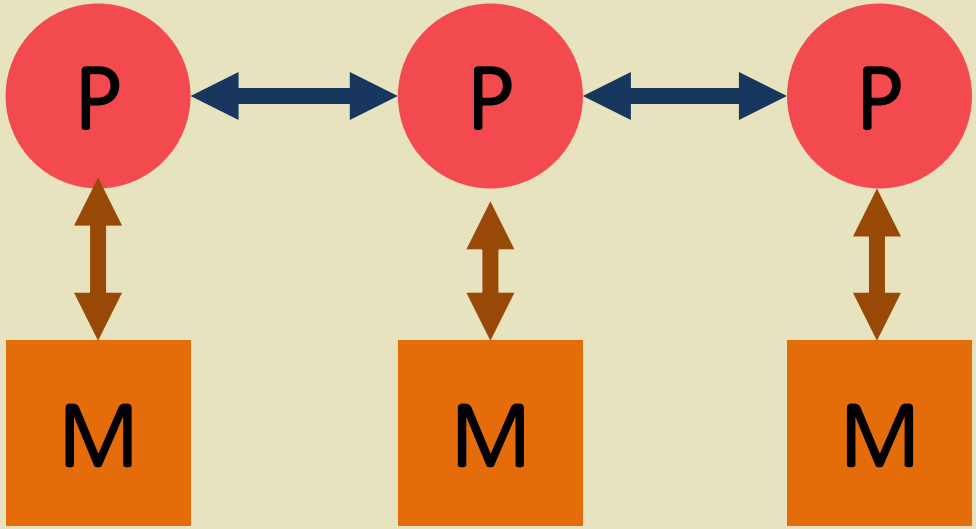
P Задача (процесс, поток)

Доступ к памяти

M Память (адресное пространство)

Передача сообщений

Программные модели вычислительных систем



Распределённая память

(MPI)

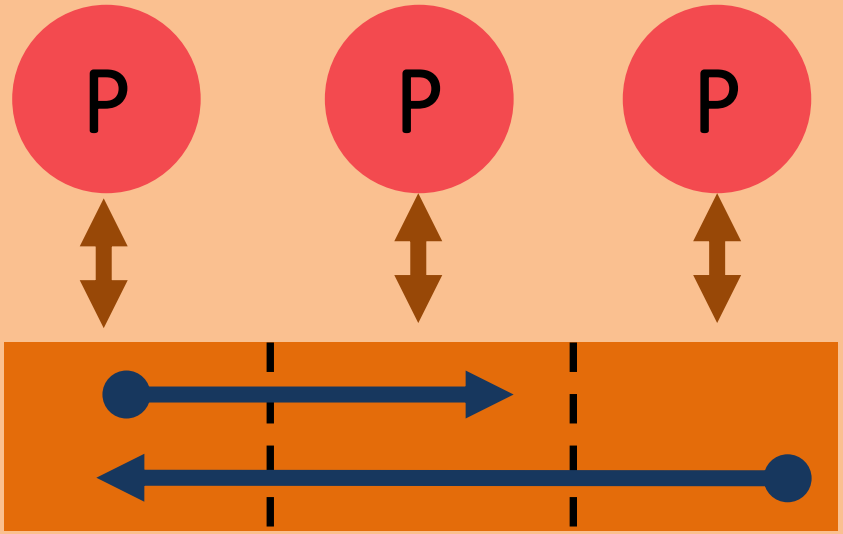
Задача (процесс, поток)

Доступ к памяти

Память (адресное пространство)

Передача сообщений

Программные модели вычислительных систем



- X10
- Chapel
- UPC
- Titanium
- Co-array Fortran
- HPF

Разделяемое общее адресное пространство (Partitioned Global Address Space – PGAS)

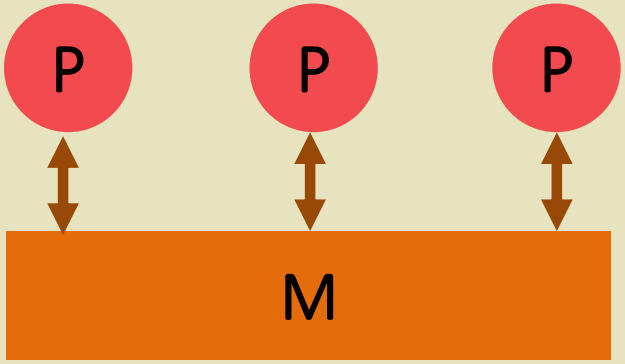
P — Задача (процесс, поток)

Доступ к памяти

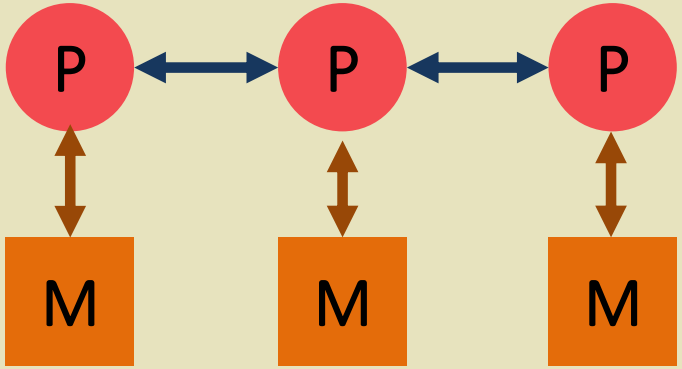
M — Память (адресное пространство)

Передача сообщений

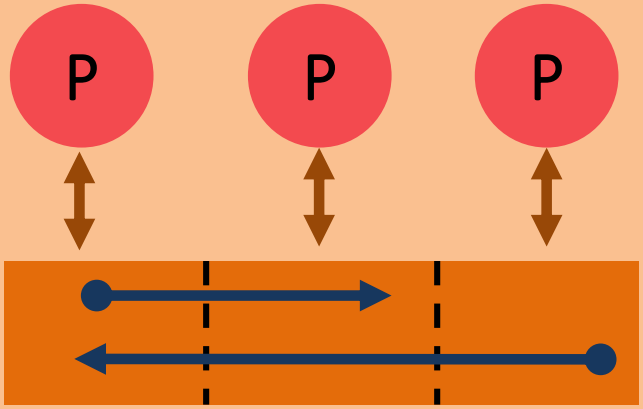
Программные модели вычислительных систем



Общая память
(PThreads, C++, OpenMP, Cilk)



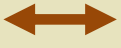
Распределённая память
(MPI)



Разделяемое общее адресное пространство (Partitioned Global Address Space – PGAS)

- X10
- Chapel
- UPC
- Titanium
- Co-array Fortran
- HPF

Задача (процесс, поток)

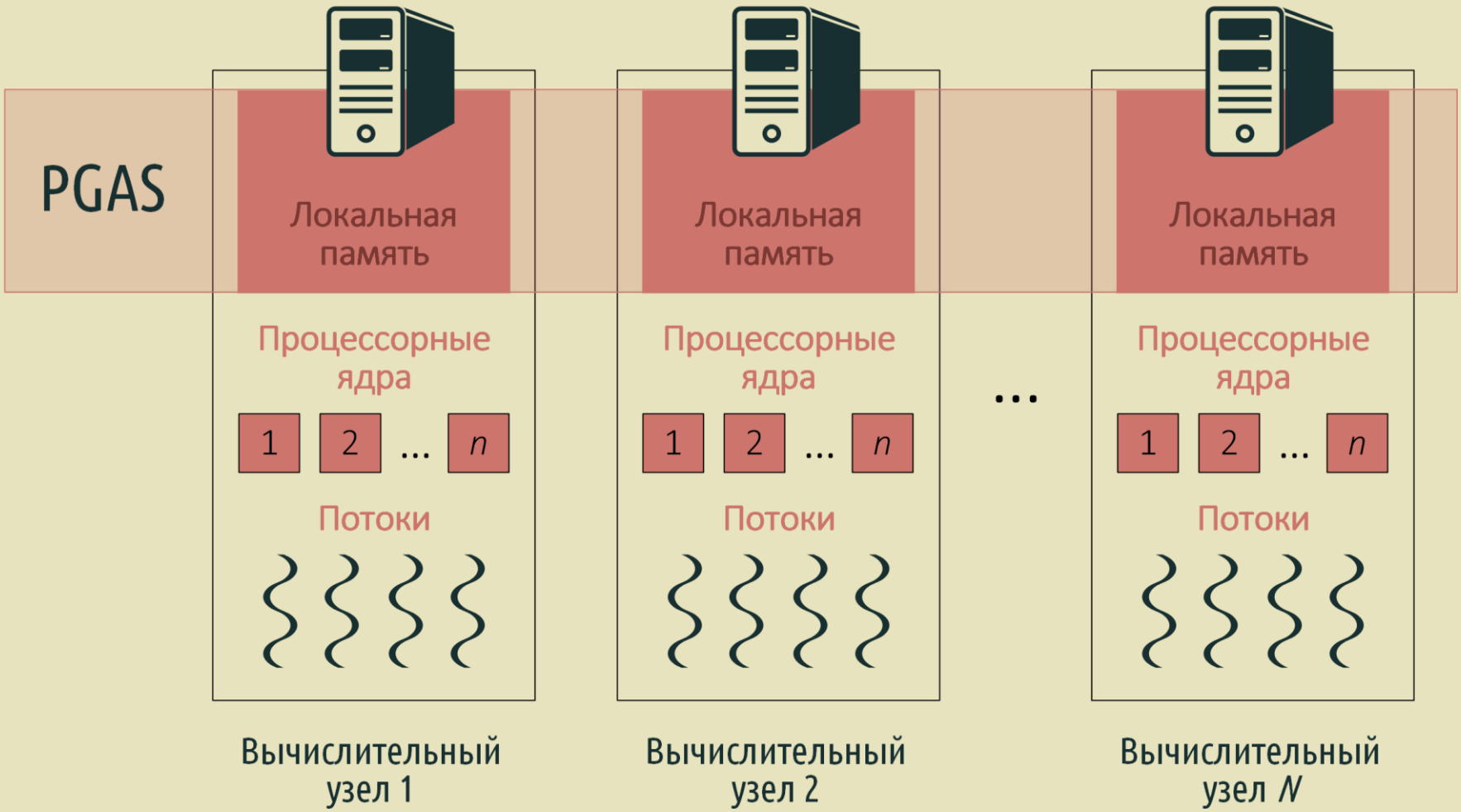


Доступ к памяти

Память (адресное пространство)

Передача сообщений

Глобальное разделённое адресное пространство (PGAS)

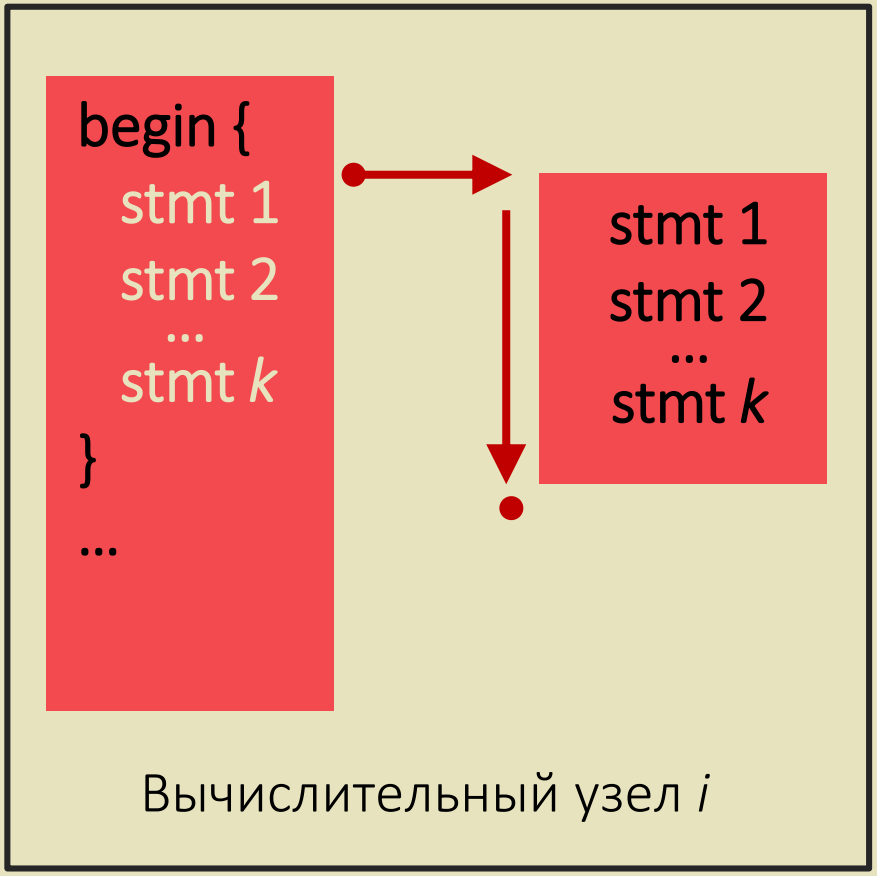


Основные программные конструкции PGAS-языков

- **begin** *S* – асинхронное выполнение инструкций *S* в отдельном потоке на вычислительном узле, с которого вызвана конструкция
- **on** *i S* – выполнение инструкций *S* на узле *i*
- **on** *x S* – выполнение инструкций *S* на том узле, в памяти которого находится объект x
- **coforall** *S* – выполнение каждой итерации тела *S* цикла в независимом потоке
- **sync** *T* – синхронизация группы потоков *T*

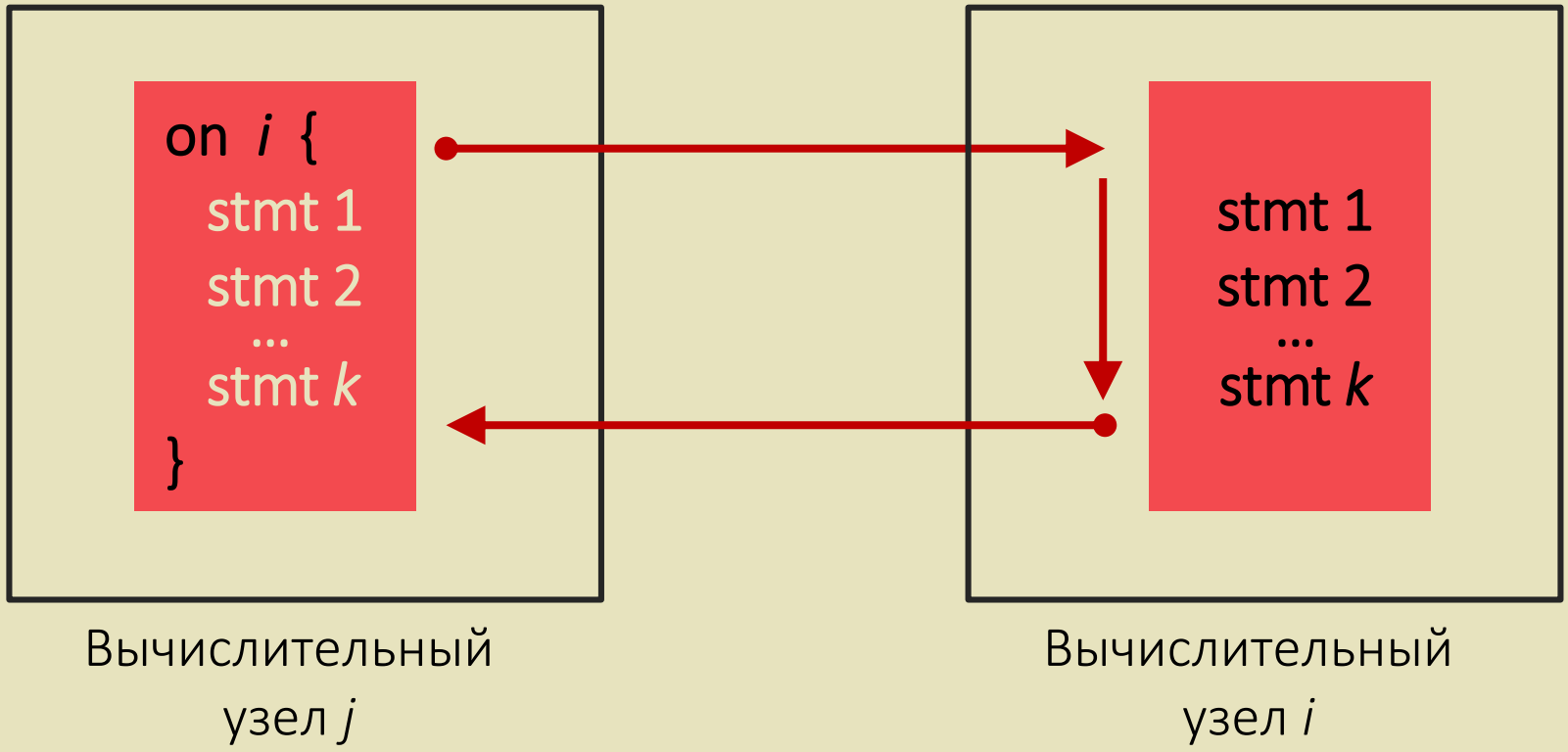
Основные программные конструкции PGAS-языков

- **begin** *S* – асинхронное выполнение инструкций *S* в отдельном потоке на вычислительном узле, с которого вызвана конструкция



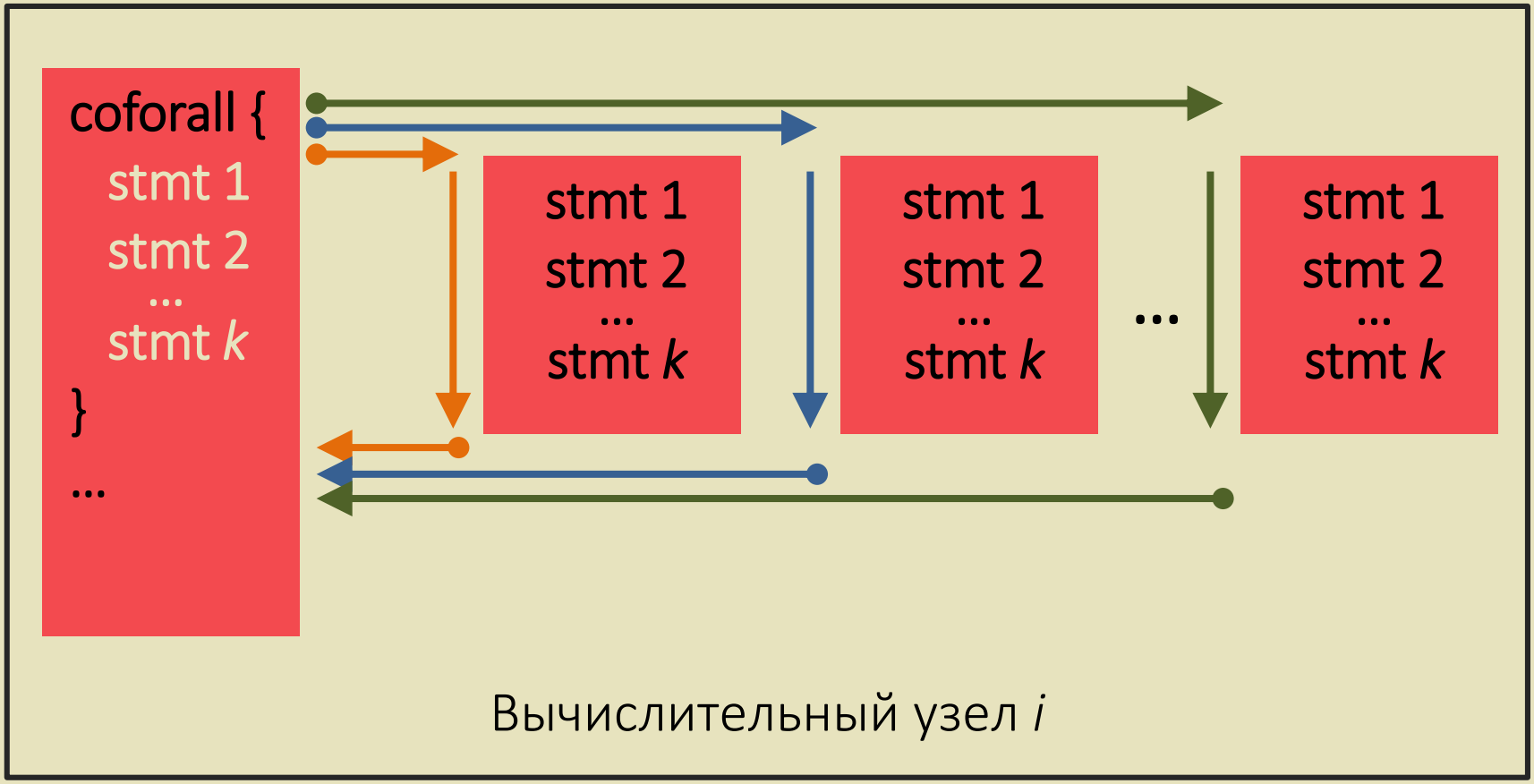
Основные программные конструкции PGAS-языков

- on *i S* – выполнение инструкций *S* на узле *i*



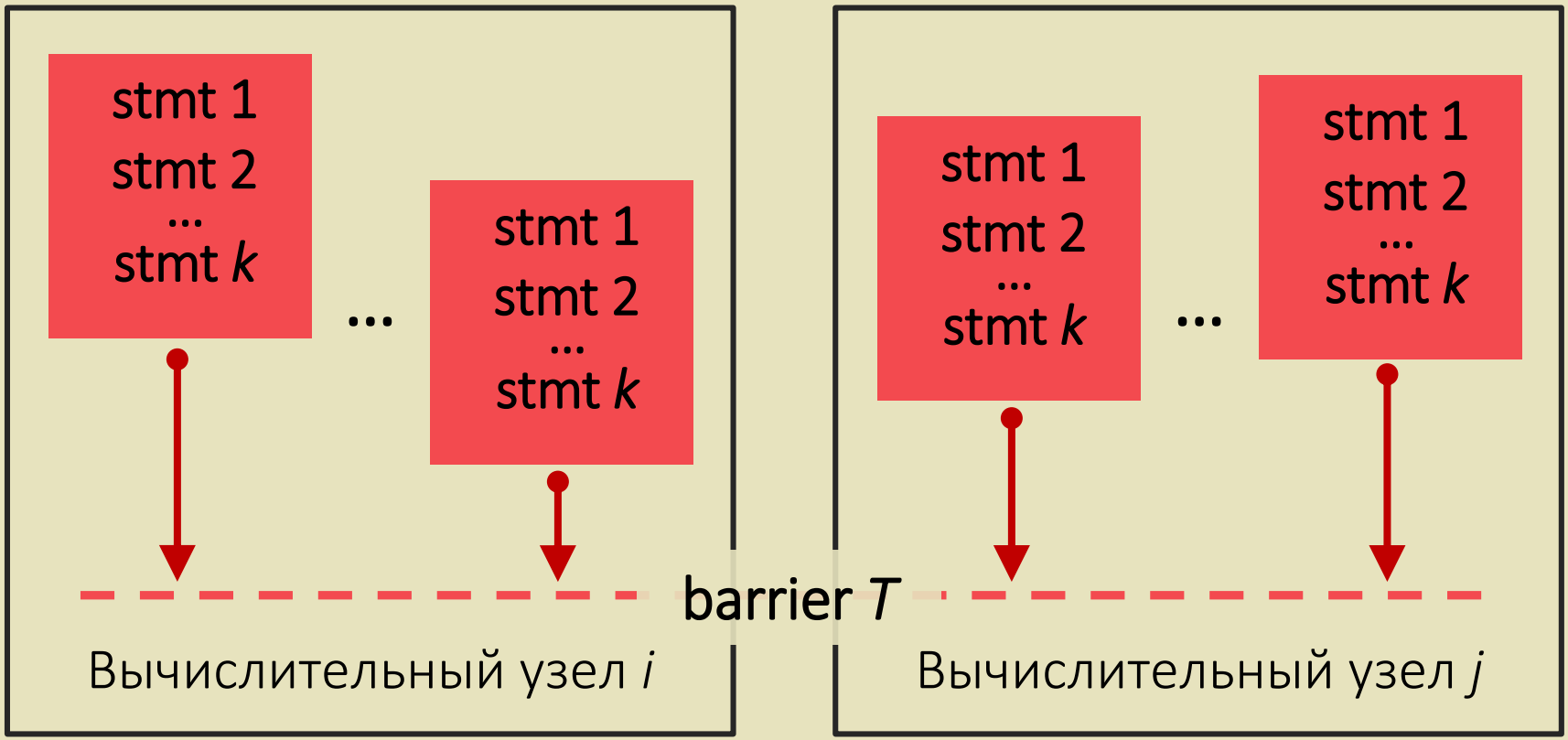
Основные программные конструкции PGAS-языков

- **coforall** *S* – выполнение каждой итерации тела *S* цикла в независимом потоке



Основные программные конструкции PGAS-языков

- **sync *T*** – синхронизация группы потоков *T*



Основные программные конструкции PGAS-языков

Параллелизм уровня потоков

- порождение задач (begin)
- параллельные циклы (coforall)
- примитивы синхронизации (sync)

Параллелизм уровня данных

- распределённые структуры
- коллективные операции (редукция, префиксная сумма)

Основные программные конструкции PGAS-языков

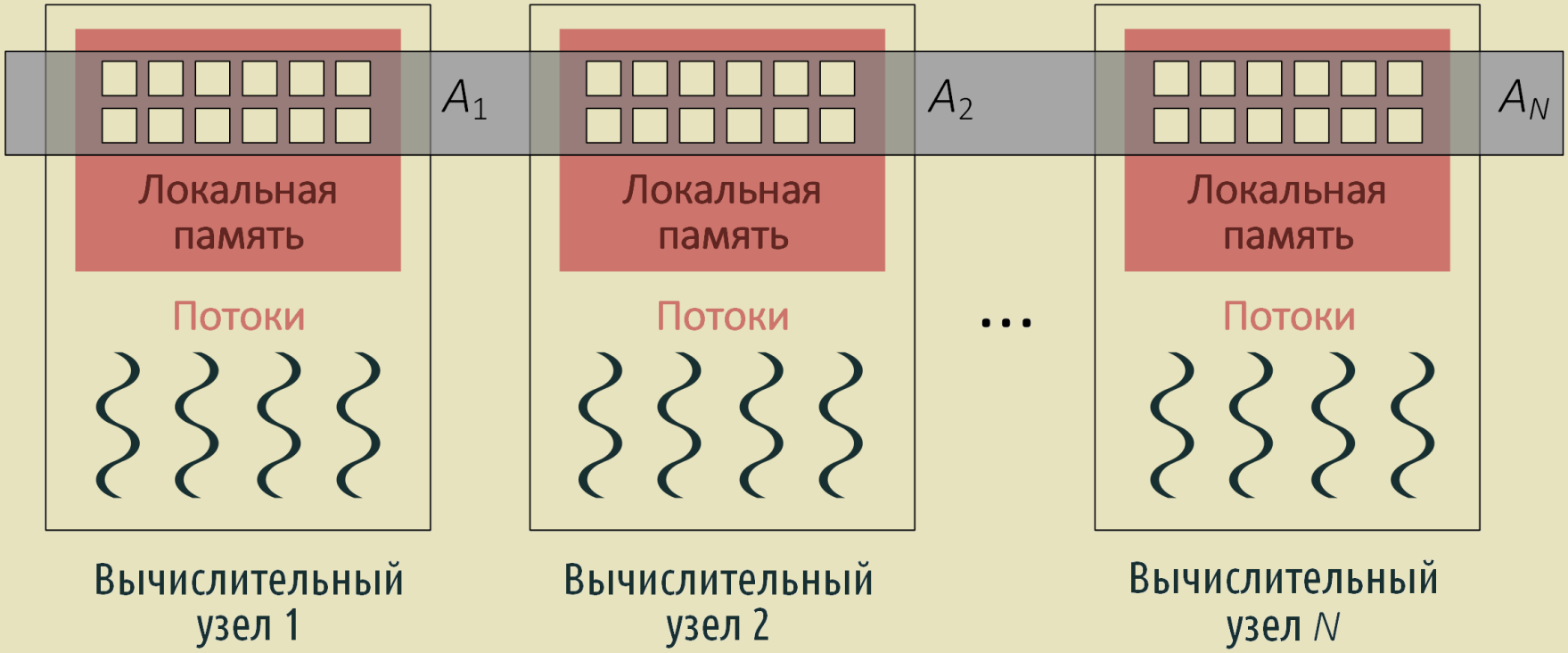
Параллелизм уровня потоков

- порождение задач (begin)
- параллельные циклы (coforall)
- примитивы синхронизации (sync)

Параллелизм уровня данных

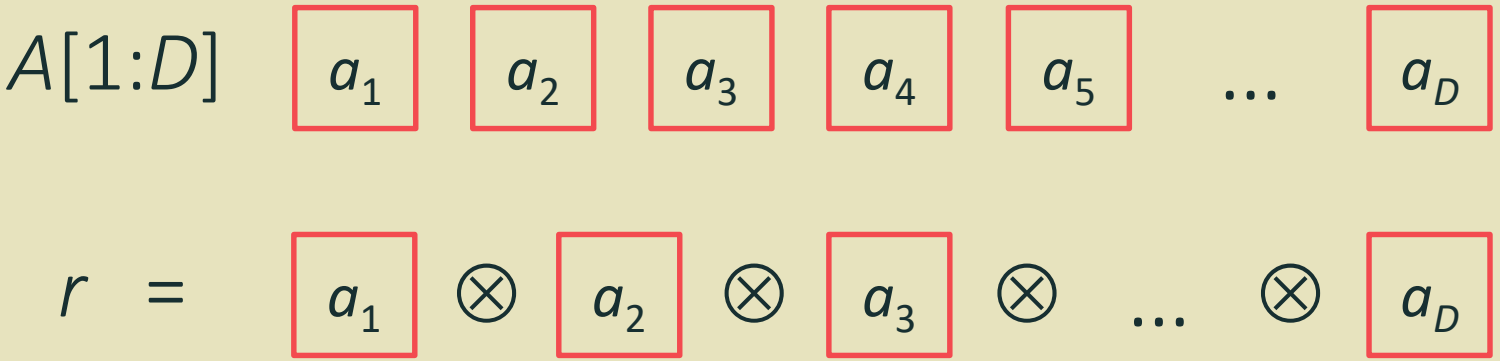
- распределённые структуры
- коллективные операции (редукция, префиксная сумма) над массивами

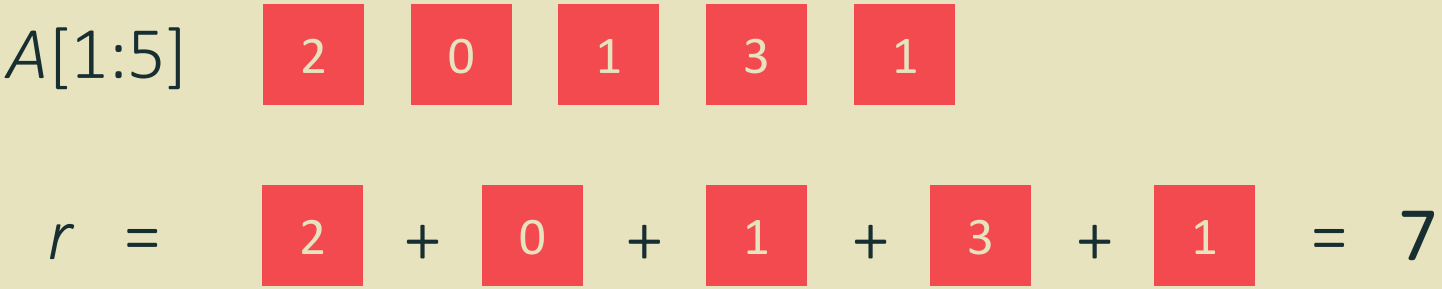
Распределённые массивы (distributed arrays)



Элементы массива *A* распределены между узлами 1, …, *N*

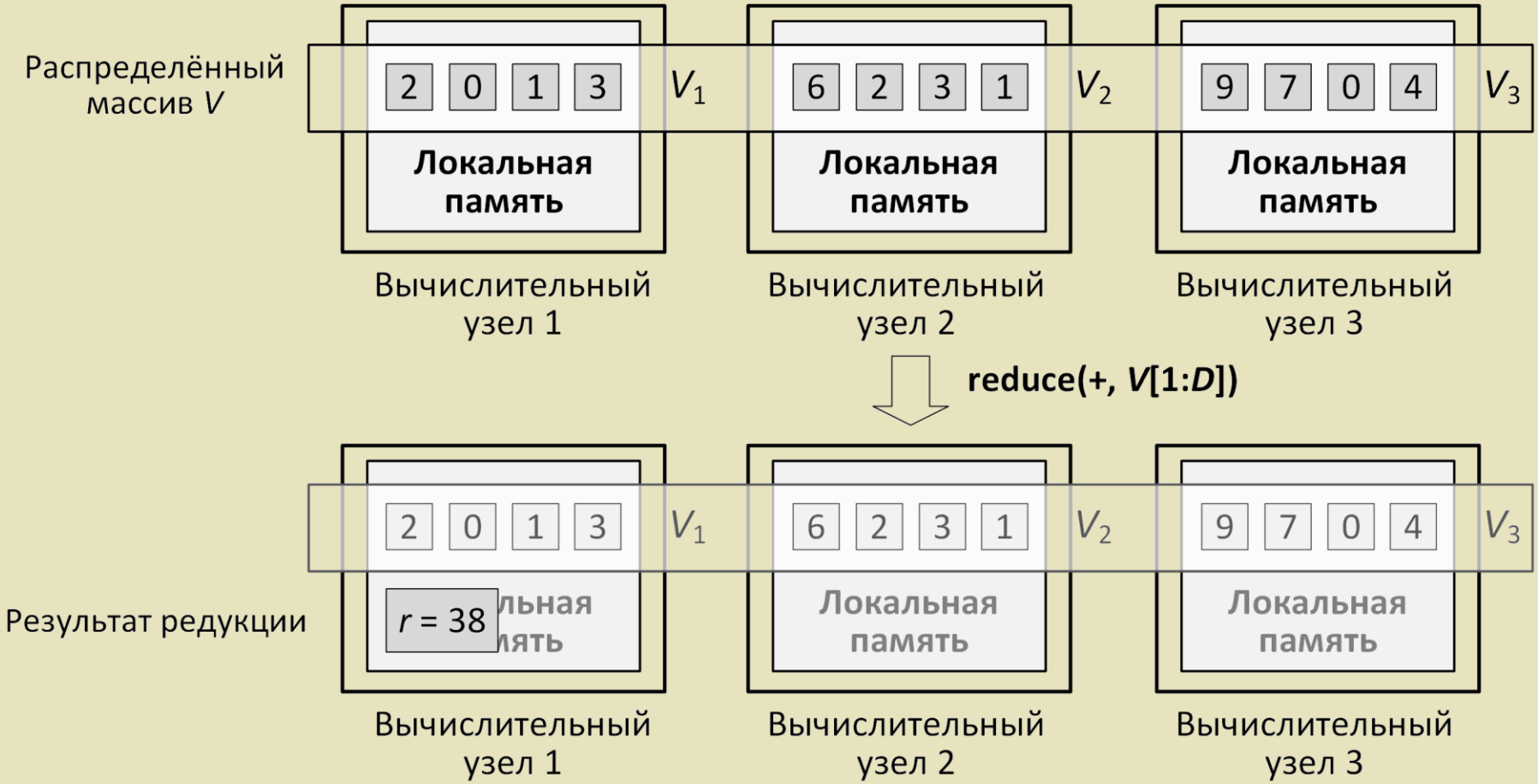
Параллельная редукция (reduce, reduction)





⊗ - ассоциативная операция

Параллельная редукция (reduce, reduction)



Rabenseifner R. **Optimization of Collective Reduction Operations** // Computational Science - ICCS 2004 - Lecture Notes in Computer Science. 2004. Vol. 3036. P. 1-9.

Li S, Hoefler T., Snir M. **NUMA-Aware Shared-Memory Collective Communication for MPI** // HPDC 2013. 2013. P. 85-96.

Пример редукции в Cray Chapel

```
1   use BlockDist;
2
3   proc main {
4     var Space = {1..n, 1..n};
5     var BlockSpace = Space dmapped
6                        Block(boundingBox=Space);
7     var BA: [BlockSpace] real;
8
9     forall (i, j) in Space do
10      BA[i, j] = i * 10 + j;
11
12    var sum = + reduce BA;
13
14    writeln(“REDUCE: ”, sum);
15  }
```

Пример редукции в Cray Chapel

```
1   use BlockDist;
2
3   proc main {
4     var Space = {1..n, 1..n};
5     var BlockSpace = Space dmapped
6                      Block(boundingBox=Space);
7     var BA: [BlockSpace] real;
8
9     forall (i, j) in Space do
10      BA[i, j] = i * 10 + j;
11
12    var sum = + reduce BA;
13
14    writeln(“REDUCE: ”, sum);
15  }
```

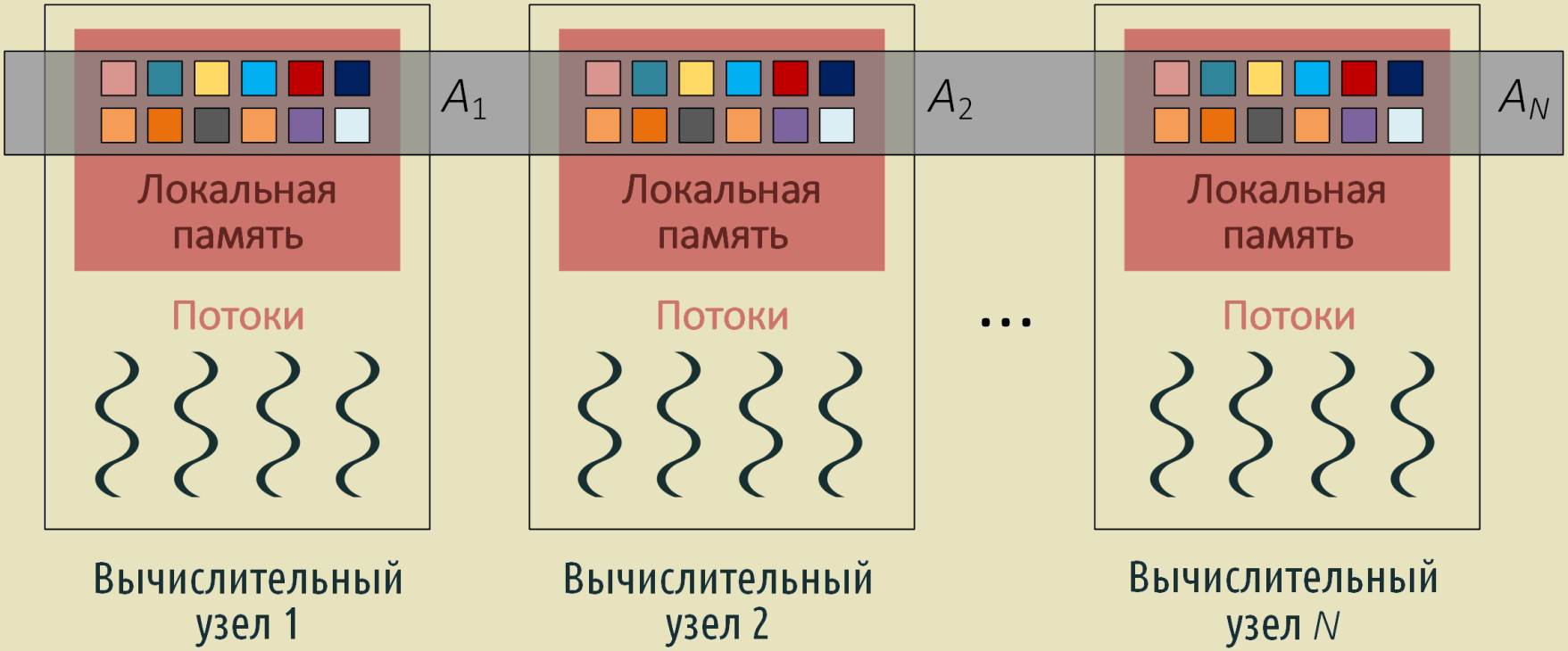
Пример редукции в Cray Chapel

```
1   use BlockDist;
2
3   proc main {
4     var Space = {1..n, 1..n};
5     var BlockSpace = Space dmapped
6                        Block(boundingBox=Space);
7     var BA: [BlockSpace] real;
8
9     forall (i, j) in Space do
10      BA[i, j] = i * 10 + j;
11
12    var sum = + reduce BA;
13
14    writeln(“REDUCE: ”, sum);
15  }
```

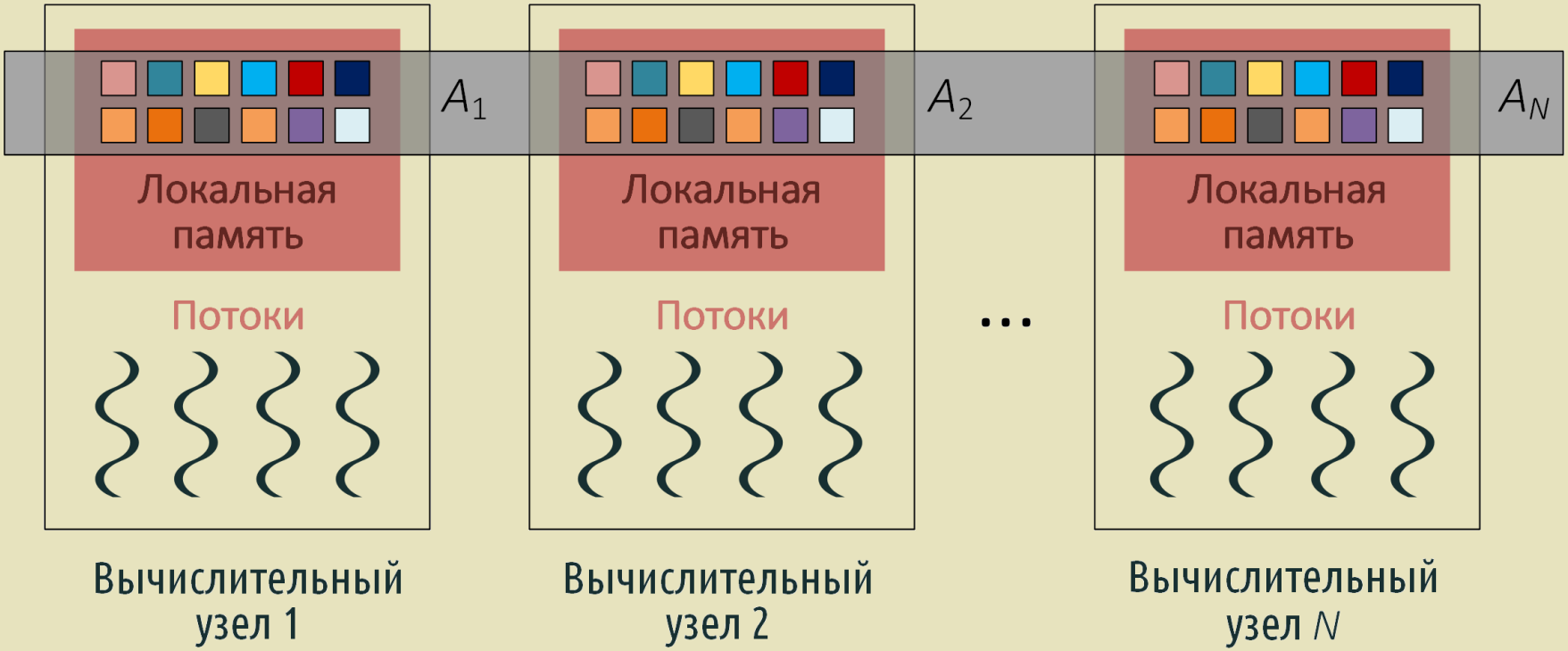
Пример редукции в Cray Chapel

```
1   use BlockDist;
2
3   proc main {
4     var Space = {1..n, 1..n};
5     var BlockSpace = Space dmapped
6                        Block(boundingBox=Space);
7     var BA: [BlockSpace] real;
8
9     forall (i, j) in Space do
10      BA[i, j] = i * 10 + j;
11
12    var sum = + reduce BA;
13
14    writeln(“REDUCE: ”, sum);
15  }
```

Алгоритм редукции DefaultReduce

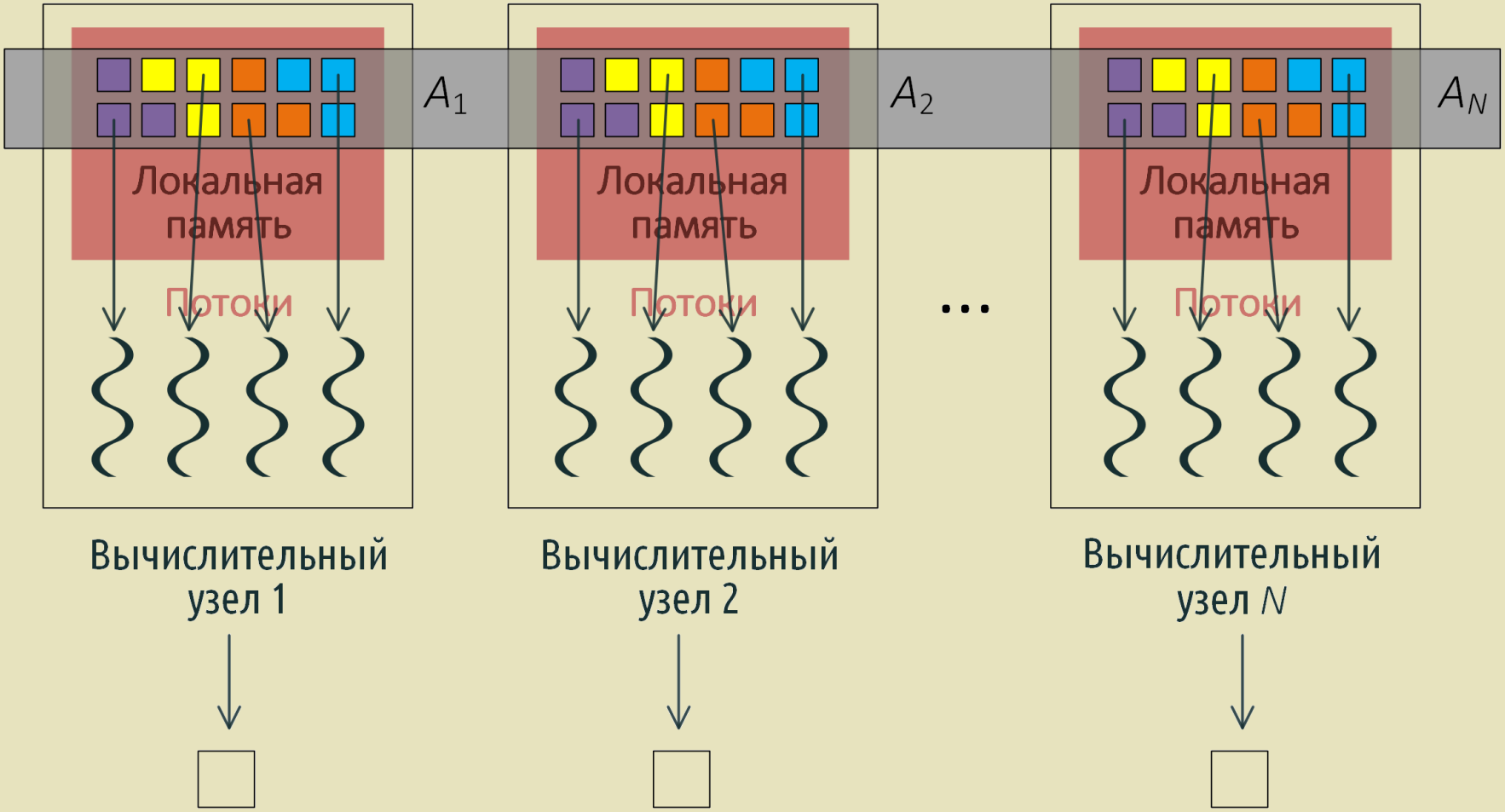


Алгоритм редукции DefaultReduce



Вычислительная трудоёмкость алгоритма DefaultReduce:
$T = O(|A| / N + N)$

Алгоритм редукции BlockReduce (I этап)



Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do      // N – число узлов
2     on i
3       SplitDomain(A_i, n)     // Разбиение на n блоков
4                               // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do     // N – число узлов
2     on i
3       SplitDomain(A_i, n)     // Разбиение на n блоков
4                               // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do      // N – число узлов
2     on i
3       SplitDomain(A_i, n)     // Разбиение на n блоков
4                               // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do      // N – число узлов
2     on i
3       SplitDomain(A_i, n)     // Разбиение на n блоков
4                               // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do      // N – число узлов
2     on i
3       SplitDomain(A_i, n)            // Разбиение на n блоков
4                                      // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

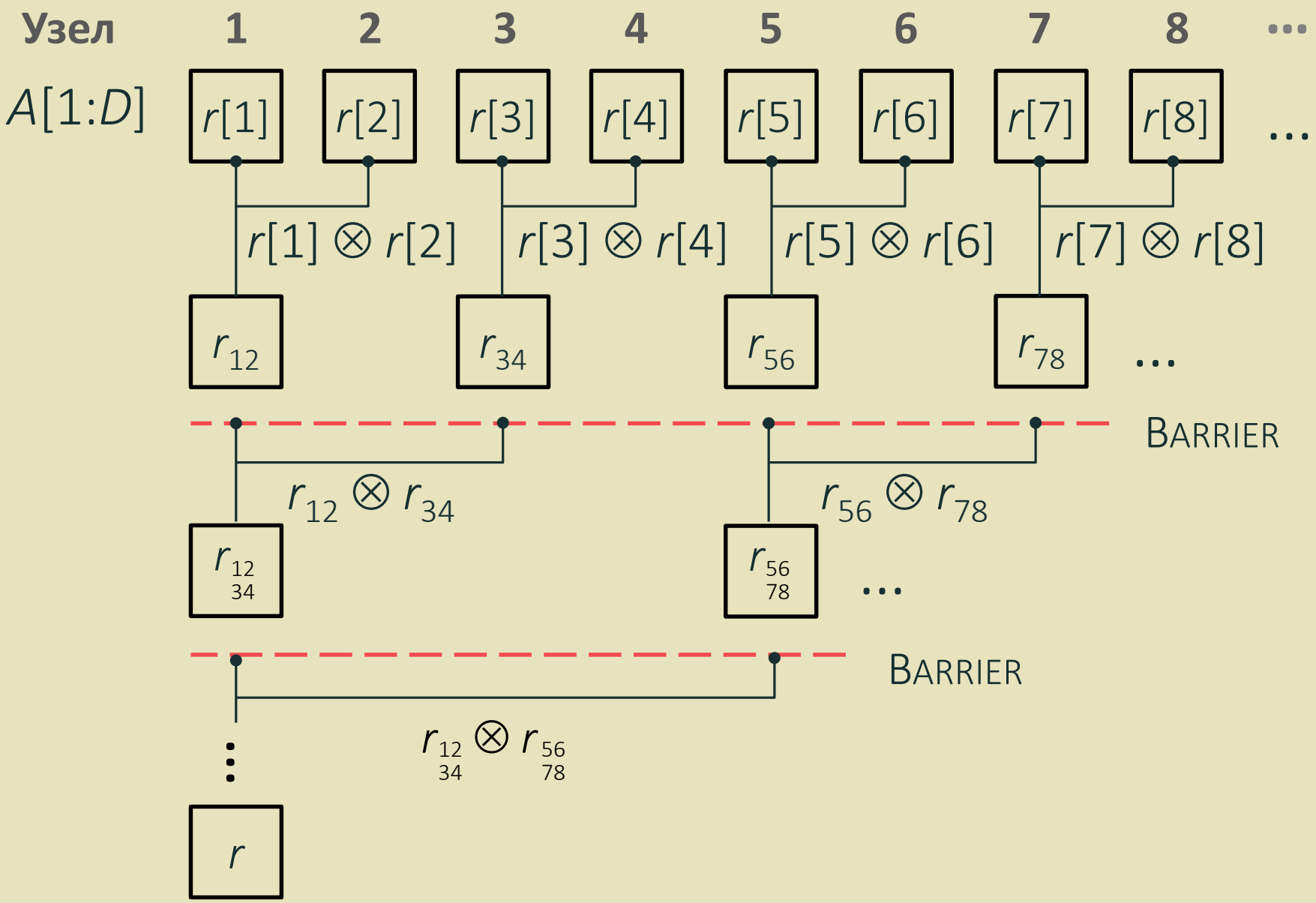
Алгоритм редукции BlockReduce (I этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

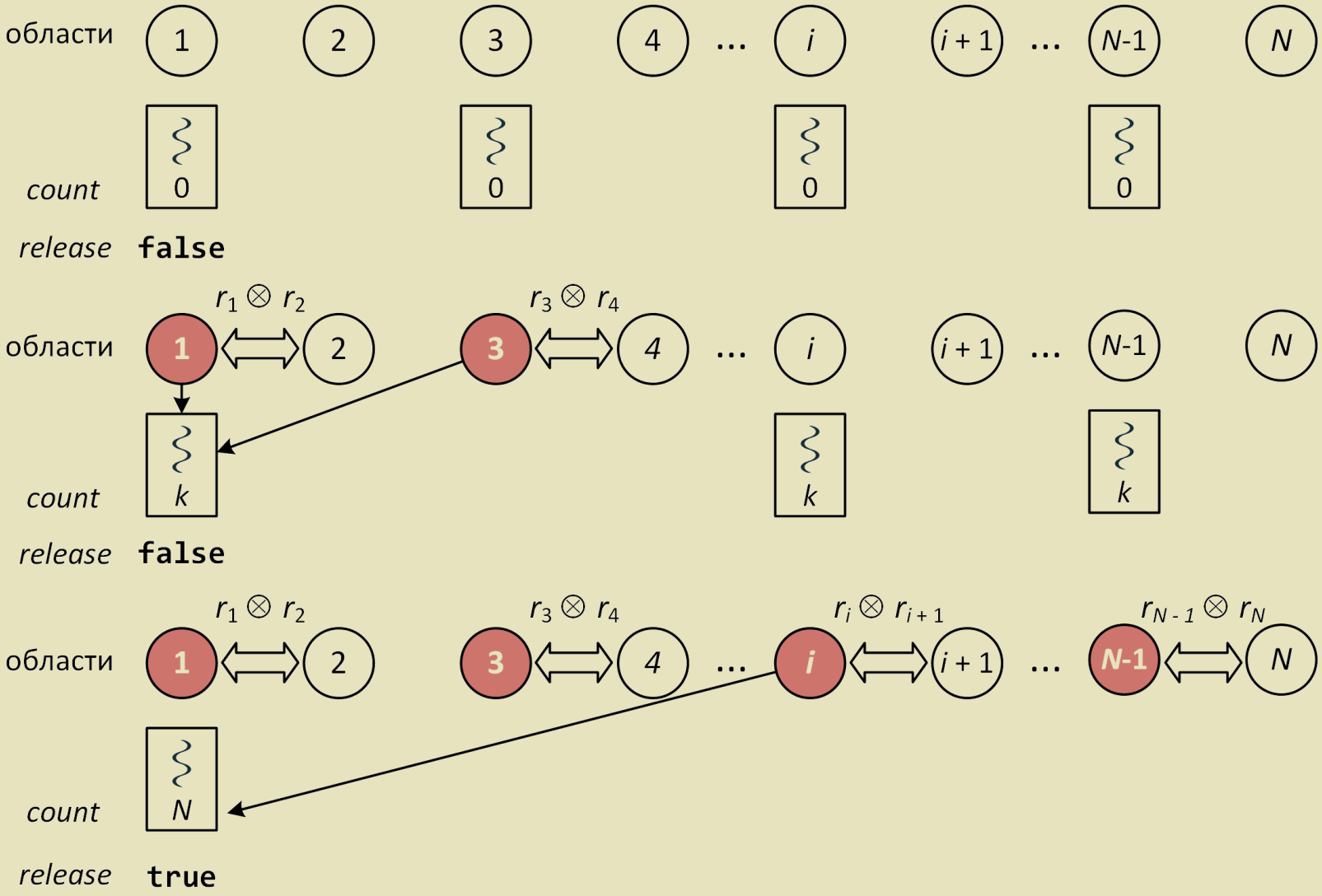
Выходные данные: r – результат применения редукции для A.

```
1   coforall i in [1, 2, …, N] do      // N – число узлов
2     on i
3       SplitDomain(A_i, n)     // Разбиение на n блоков
4                               // по числу процессорных ядер
5       coforall t in [1, 2, …, n] do
6         foreach x in A_it do
7           r_it = r_it ⊗ x
8       foreach t in [1, 2…, n] do
9           r_i = r_i ⊗ r_it
```

Алгоритм редукции BlockReduce (II этап)



Алгоритм редукции BlockReduce (II этап)



Алгоритм редукции BlockReduce (II этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    …
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, …, N])        // Барьерная синхронизация потоков
16      REFRESH(N)                   // Обновление числа N ВУ
17  return r
```

Алгоритм редукции BlockReduce (II этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    ...
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, ..., N])        // Барьерная синхронизация потоков
16      REFRESH(N)                     // Обновление числа N ВУ
17  return r
```

Вычислительная трудоёмкость алгоритма BlockReduce:

$T = O(|A| / N + \log(N))$

Организация экспериментов (алгоритм BlockReduce)

Конфигурация подсистемы

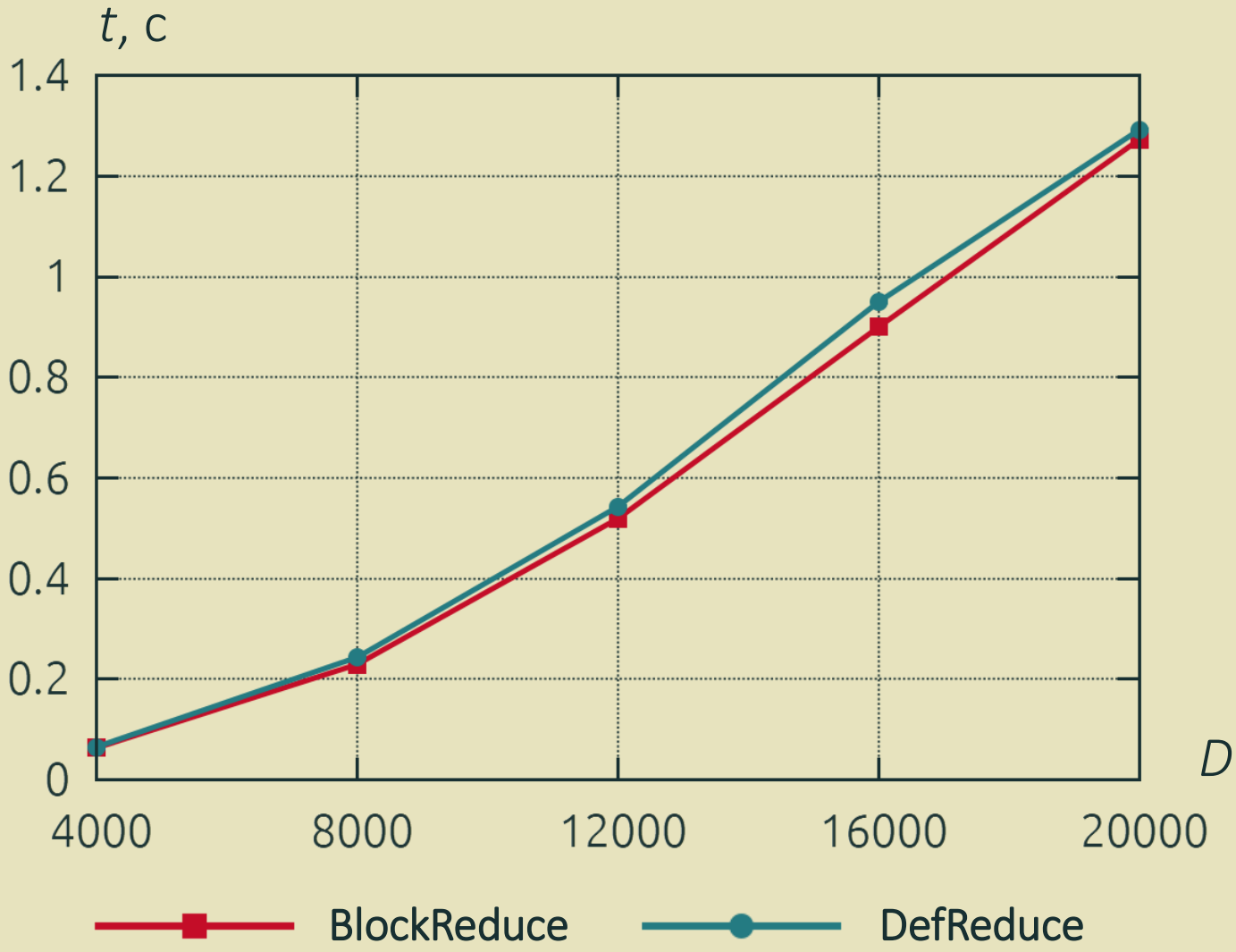
- Кластер A: 6 узлов 2 x Intel Xeon E5620 2.4 GHz (8 процессорных ядер на узел), 8 GiB, сеть Gigabit Ethernet

Описание тестовых программ

- Синтетический тест: 100 операций редукции
- Размер массива $D = 4000, 8000, \ldots, 20000$ элементов
- Chapel-программа **PTRANS** – транспонирование распределённых матриц
- Chapel-программа **miniMD** – молекулярная динамика

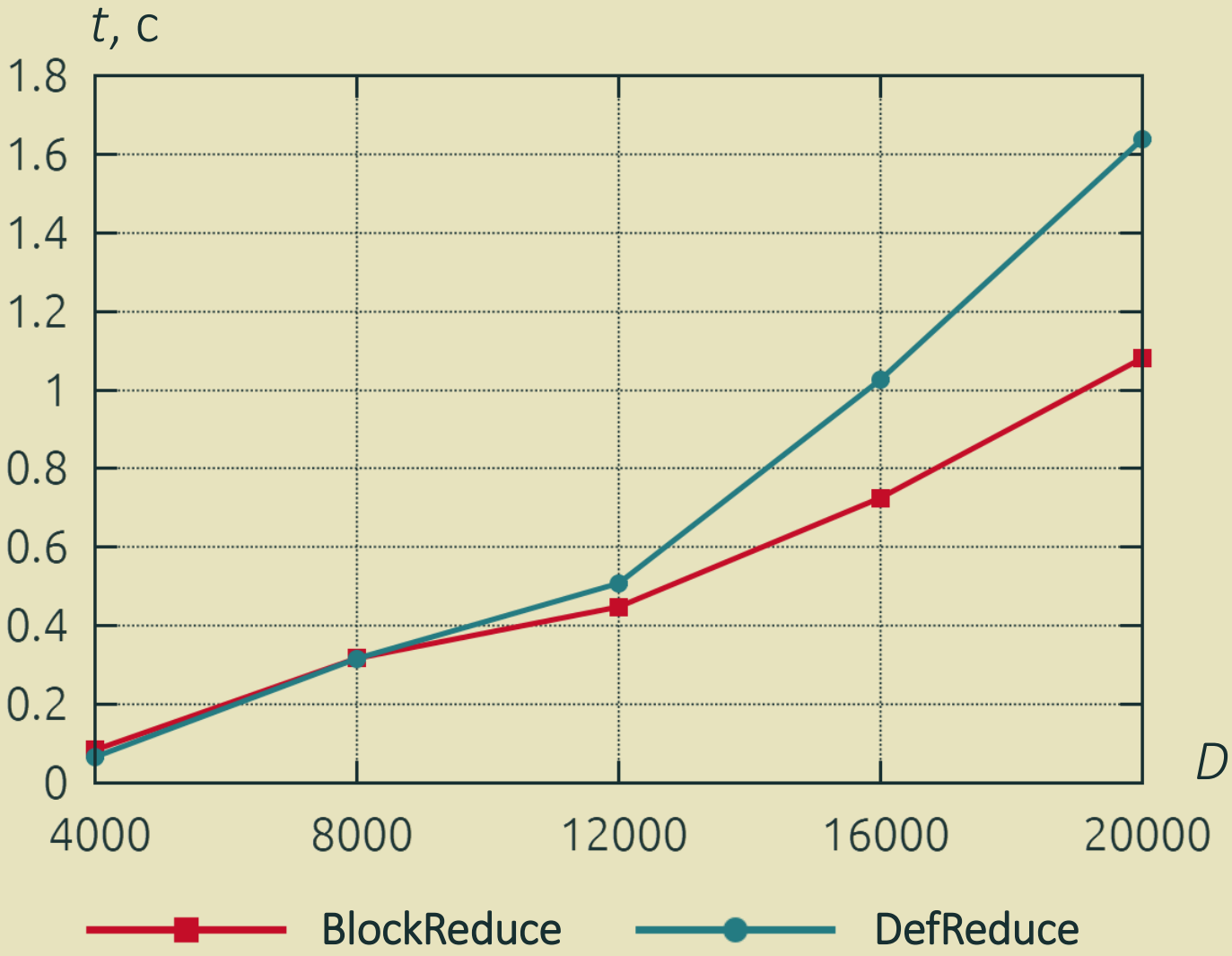
Результаты экспериментов

100 операций редукции массива $A[1:D]$, 2 узла кластера Jet



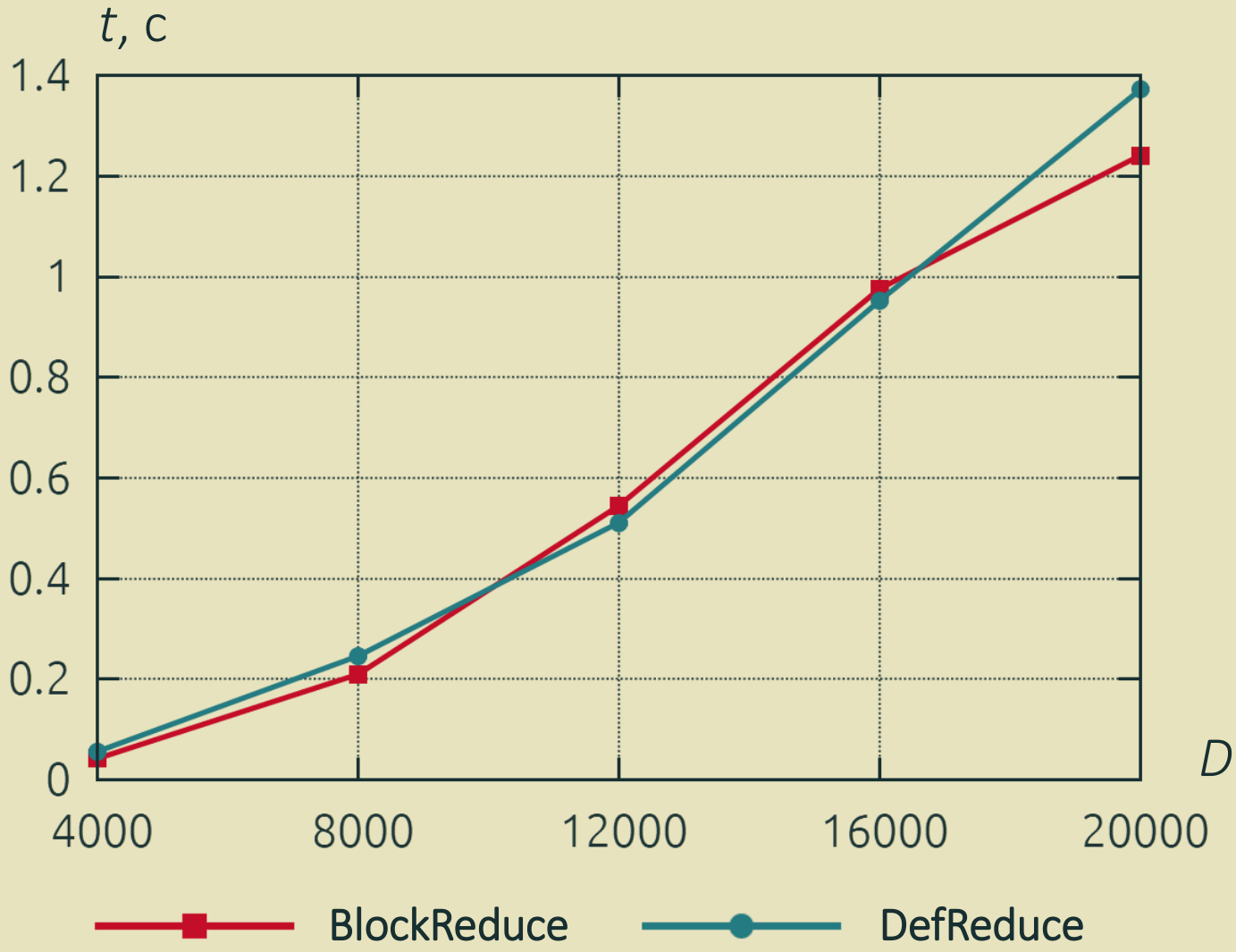
Результаты экспериментов

100 операций редукции массива $A[1:D]$, 4 узла кластера Jet



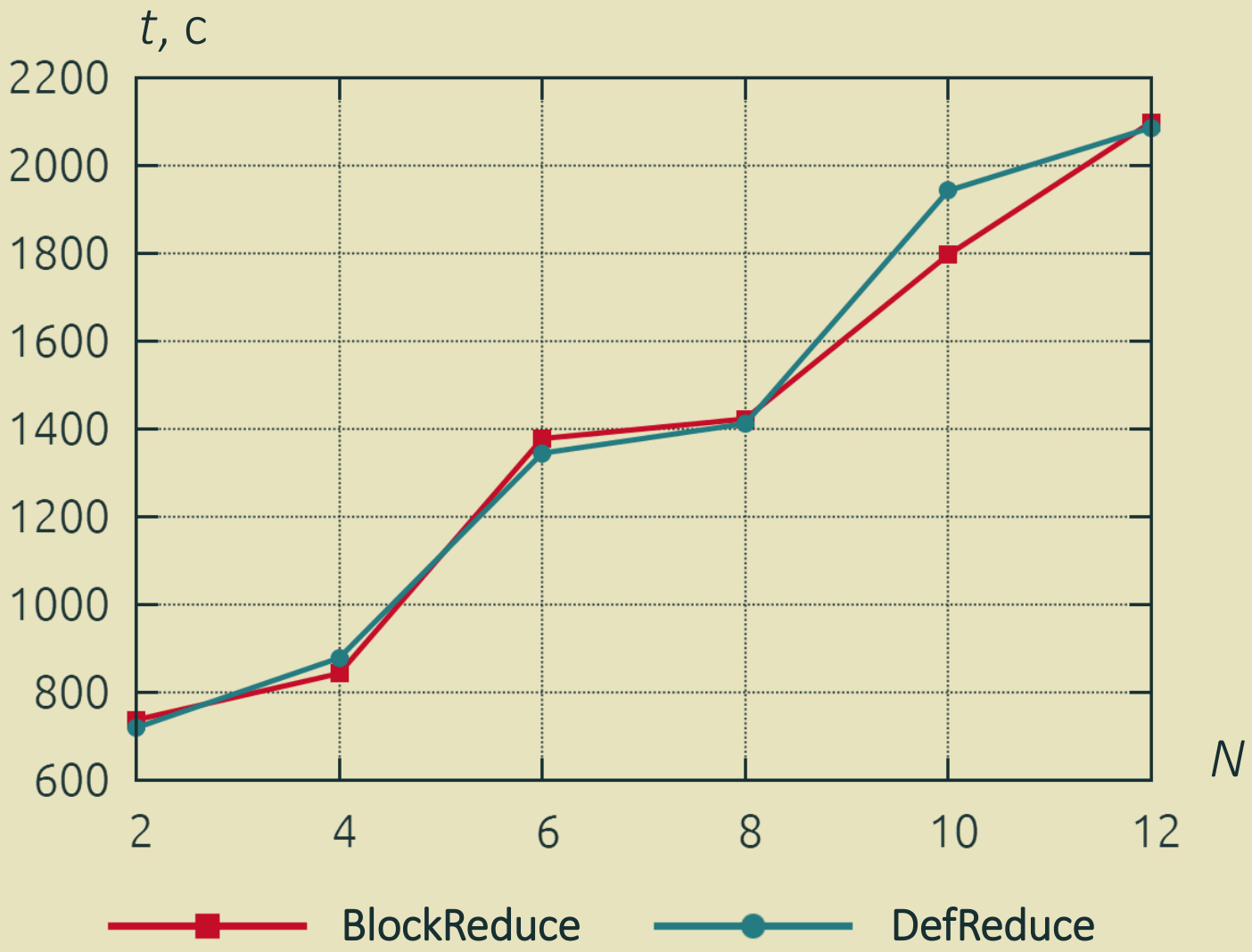
Результаты экспериментов

100 операций редукции массива $A[1:D]$, 8 узлов кластера Jet



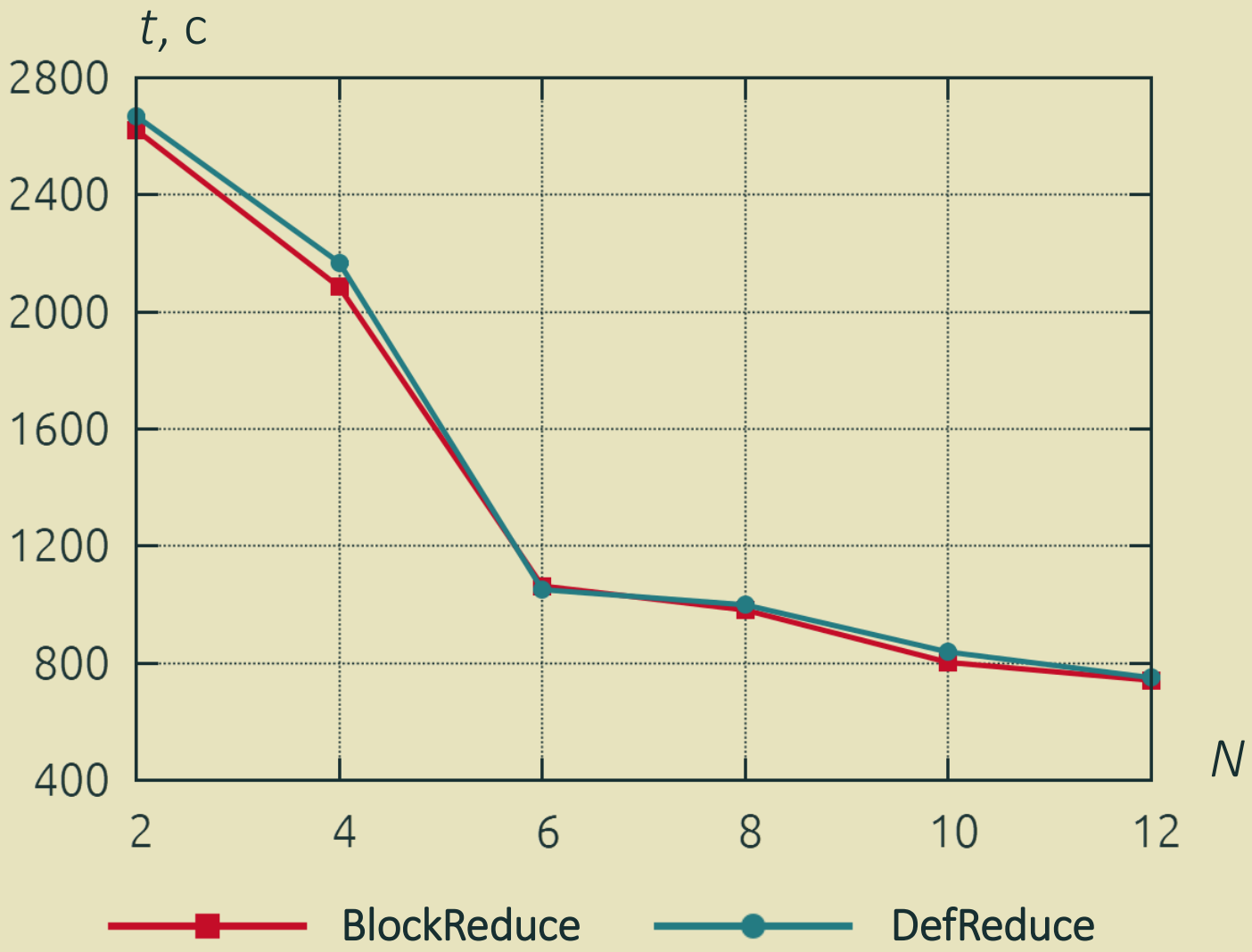
Результаты экспериментов

Программа miniMD, *N* узлов кластера Jet



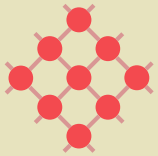
Результаты экспериментов

Программа PTRANS, *N* узлов кластера Jet



Результаты

- Предложен алгоритм редукции BlockReduce в PGAS-языках. Алгоритм реализован в Cray Chapel.
- BlockReduce обеспечивает сокращение времени выполнения редукции на 5-10% для больших размеров массива ($D > 10000$).
- Эффективность алгоритма редукции увеличивается с ростом размера распределённых массивов.



Спасибо за внимание!

https://bitbucket.org/apaznikov/chapel

Пазников А.А. apaznikov@gmail.com

Результаты профилирования DefaultReduce

Результаты профилирования DefaultReduce

Результаты профилирования BlockReduce

Результаты профилирования BlockReduce

Алгоритм редукции BlockReduce (II этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    ...
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, ..., N])        // Барьерная синхронизация потоков
16      REFRESH(N)                     // Обновление числа N ВУ
17  return r
```

Алгоритм редукции BlockReduce (II этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    ...
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, ..., N])        // Барьерная синхронизация потоков
16      REFRESH(N)                     // Обновление числа N ВУ
17  return r
```

Алгоритм редукции BlockReduce (II этап)

Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    ...
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, ..., N])        // Барьерная синхронизация потоков
16      REFRESH(N)                     // Обновление числа N ВУ
17  return r
```

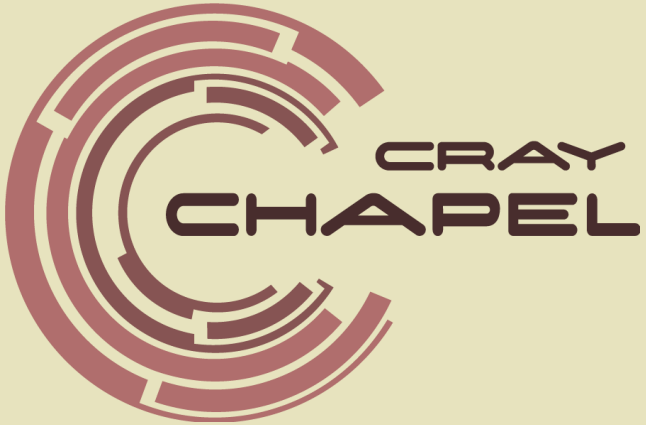
Алгоритм редукции BlockReduce (II этап)

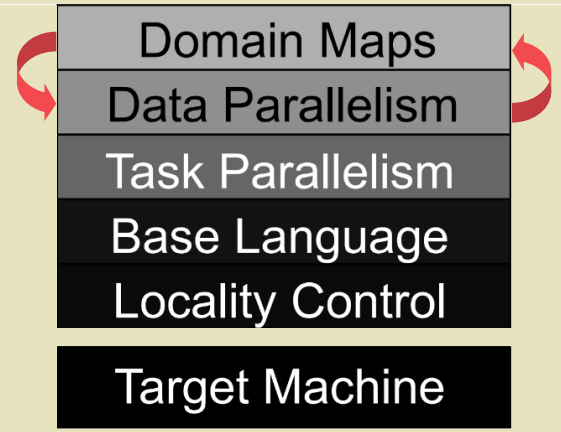
Входные данные: $A[1:D]$ – распределённый массив,
$\otimes$ – ассоциативная операция редукции.

Выходные данные: r – результат применения редукции для A.

```
    …
9   while N > 1 do
10      while i + 1 < N do
11          begin
12              on r_i
13                  r_i = r_i ⊗ r_{i+1}
14          i = i + 2
15      BARRIER([1, 2, …, N])          // Барьерная синхронизация потоков
16      REFRESH(N)                     // Обновление числа N ВУ
17  return r
```

Язык Cray Chapel





- Параллелизм уровня потоков и данных
- Учёт локальности данных
- Низкоуровневое и высокоуровневое управление параллелизмом
- Интуитивно понятный синтаксис