Параллельные технологии

Обзорное практическое занятие #5

Паттерны параллельного программирования

- Параллелизм циклов
- SPMD
- Producer/Consumer
- Master/Worker
- Fork/Join
- Map/Reduce

Параллелизм циклов

Параллелизм циклов необходим, когда основное время программы тратится на циклы.

Паттерн подразумевает оптимизацию циклов для оптимальной работы в параллельном режиме.

Параллелизм циклов

1) Распределение ветвей циклов по нитям/процессам

```
Начальный цикл:
for(i = 0; i < N; i++){
    A[i] = work(i);
}
```

Параллелизм циклов

1) Распределение ветвей циклов по нитям/процессам

Начальный цикл:

```
for(i = 0; i < N; i++){
    A[i] = work(i);
}
```

Цикл после распределения ветвей:

```
for(i = id; i < N; i += count_threads){
    A[i] = work(i);
}
```

Параллелизм циклов

2) Слияние циклов

```
for(i = 0; i < N; i++){
     A[i] = work1(i);
}
for(j = 0; j < N; j++){
     B[j] = work2(j);
}
```

Параллелизм циклов

2) Слияние циклов

```
for(i = 0; i < N; i++){
     A[i] = work1(i);
}
for(j = 0; j < N; j++){
     B[j] = work2(j);
}
```

```
for(i = 0; i < N; i++){
     A[i] = work1(i);
     B[i] = work2(i);
}
```

Параллелизм циклов

3) Объединение вложенных циклов

```
for(i = 0; i < N; i++) {
    for(j = 0; j < M; j++) {
        A[i][j] = work(i,j);
    }
}
```

Параллелизм циклов

3) Объединение вложенных циклов

```
for(i = 0; i < N; i++) {
    for(j = 0; j < M; j++) {
        A[i][j] = work(i,j);
    }
}
```

```
for(ij = 0; ij < N*M; ij++) {
    i = ij / M;
    j = ij % M;
    A[i][j] = work(i,j);
}
```

Паттерн SPMD

Single program, multiple data

Пример подхода «Разделяй и властвуй»

Строение:

1) Инициализация задачи в одном процессе.
2) Распределение независимых подзадач по процессам.
3) Счет в процессах
4) Сбор результатов и их обработка в одном процессе.

Паттерн SPMD

Примеры использования:

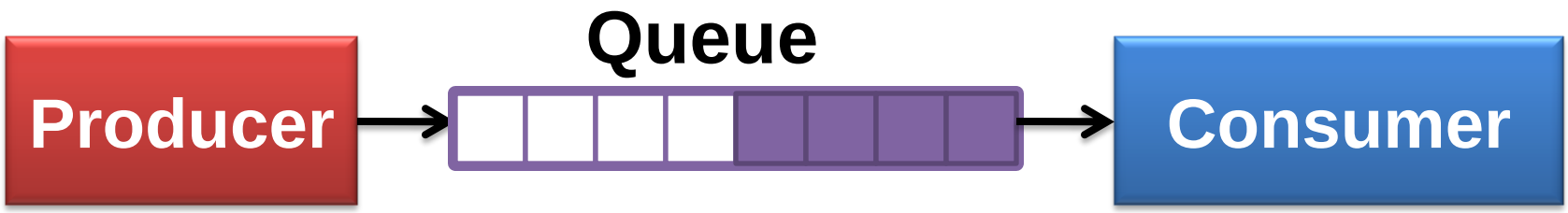
1) Все задачи из домашних заданий
2) Расчет подобластей
3) Обработка большого набора данных
4) Мультипликация
5) Множество других задач

Producer/Consumer

Producer/Consumer

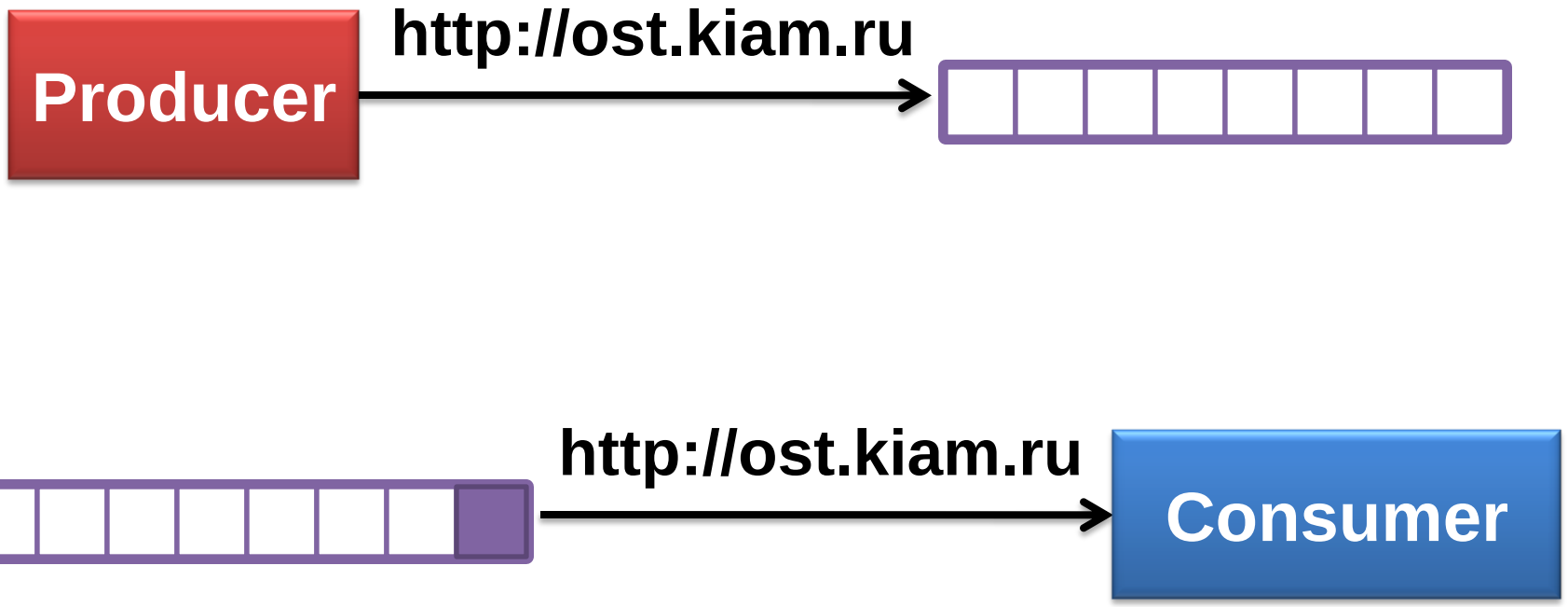
Producer/Consumer

- **Producer** ставит задачи.
- **Consumer** выполняет поставленные задачи
- **Queue** – очередь задач



Producer/Consumer

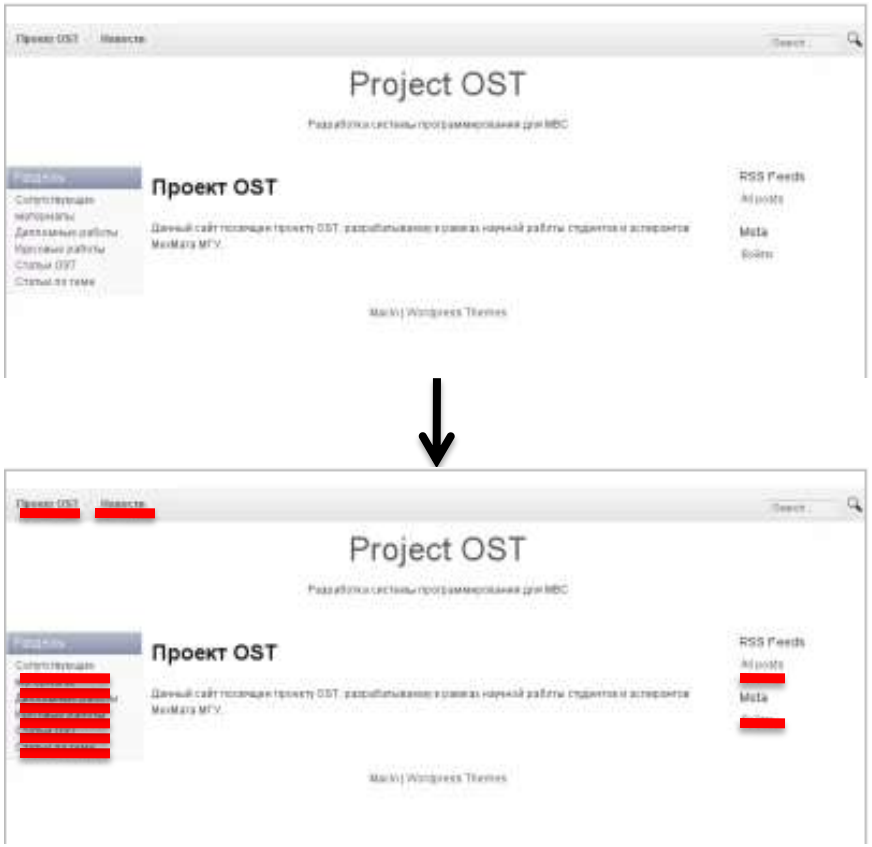
Пример: поисковый робот



Producer/Consumer

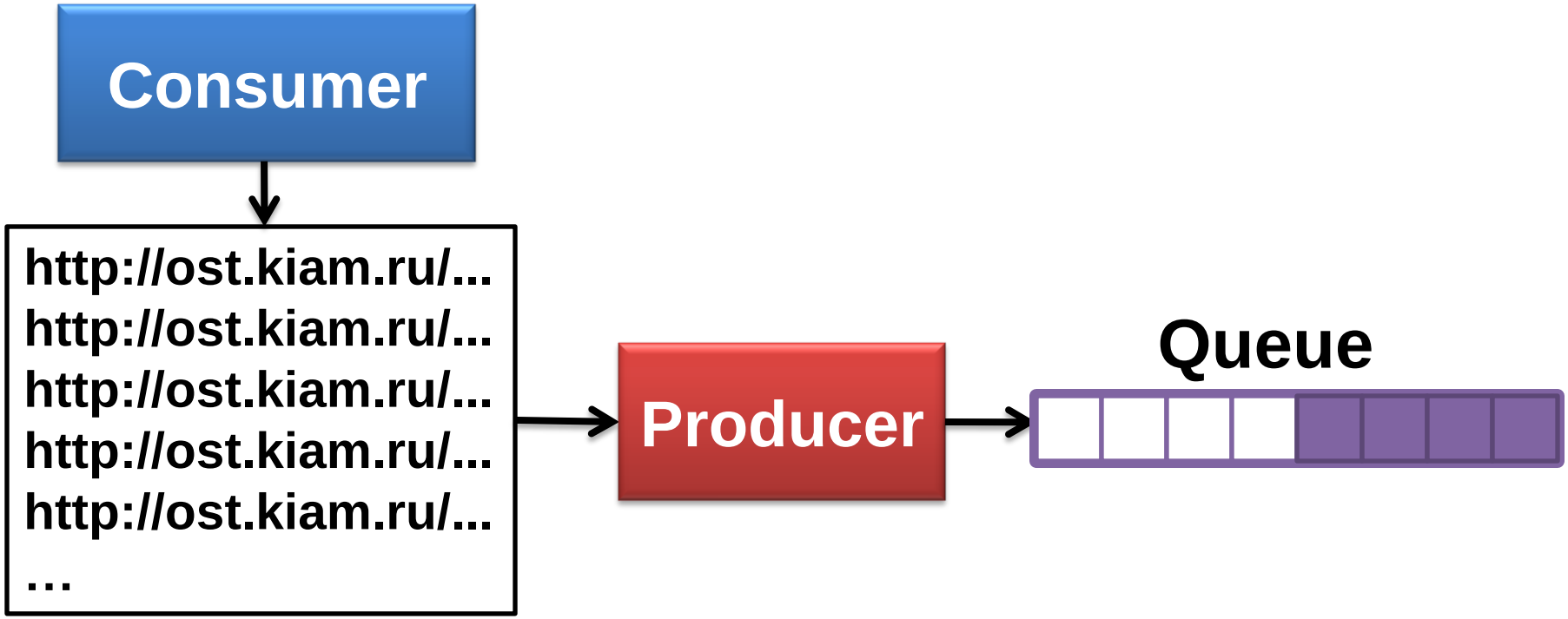
Пример: поисковый робот

Consumer



Producer/Consumer

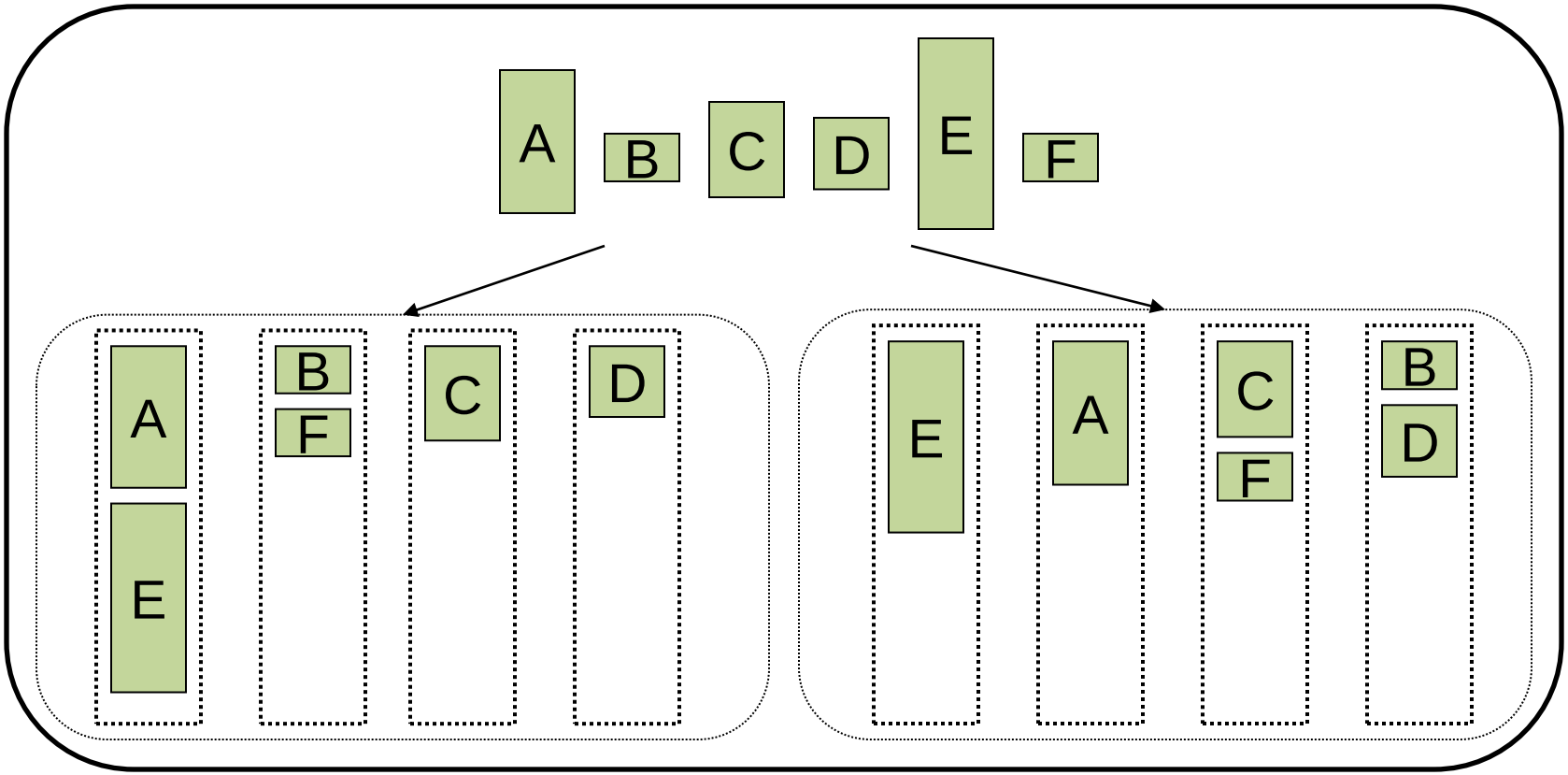
Пример: поисковый робот



Producer/Consumer

Несколько потоков?

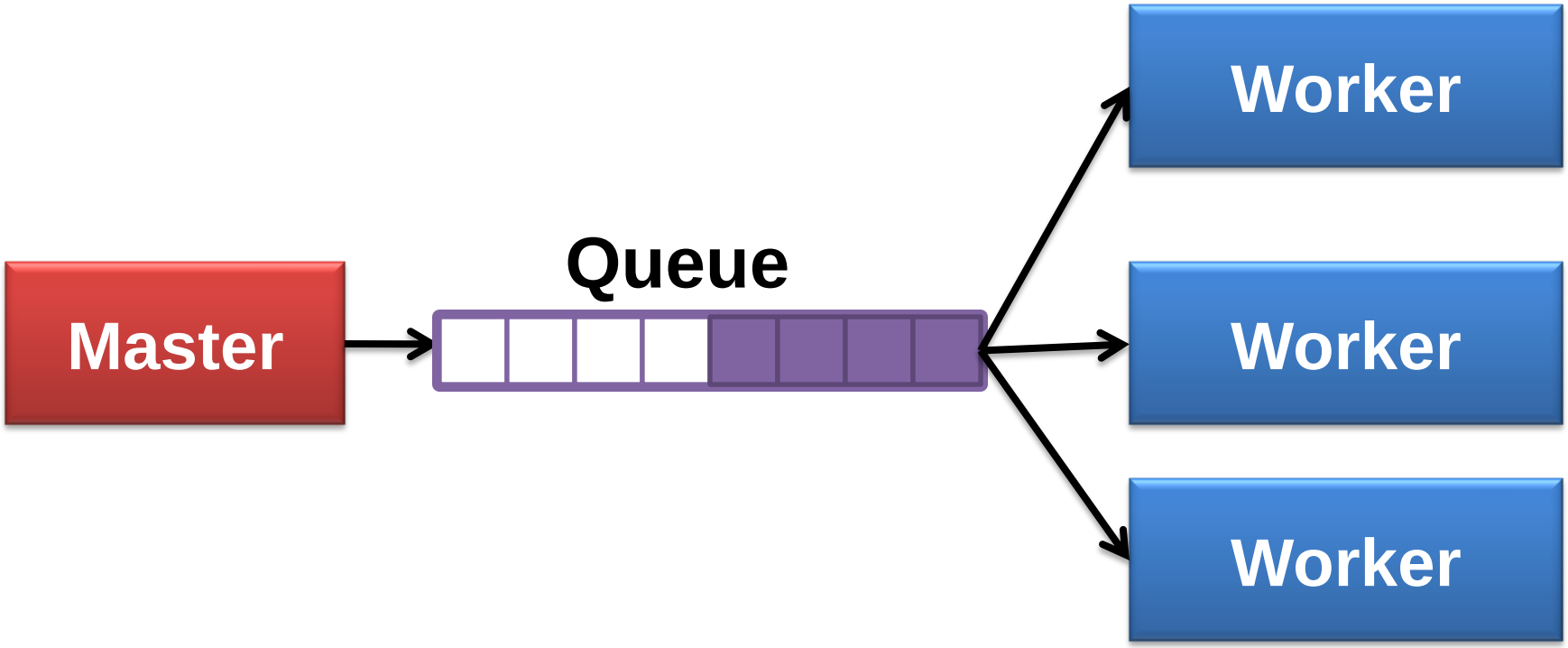
Балансировка



Master/Worker

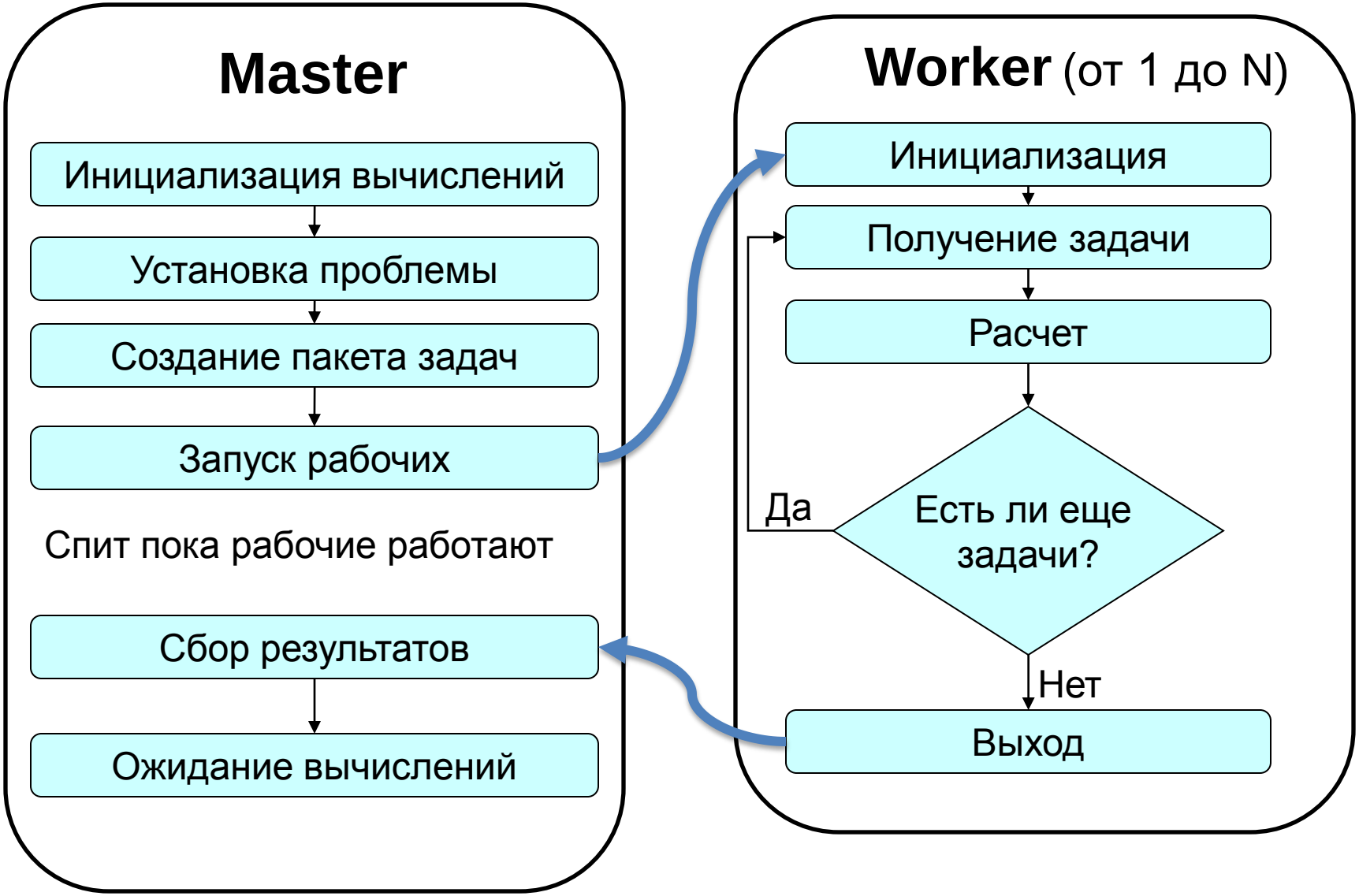
Master – постановщик задач

Worker – исполнитель задач



Master/Worker

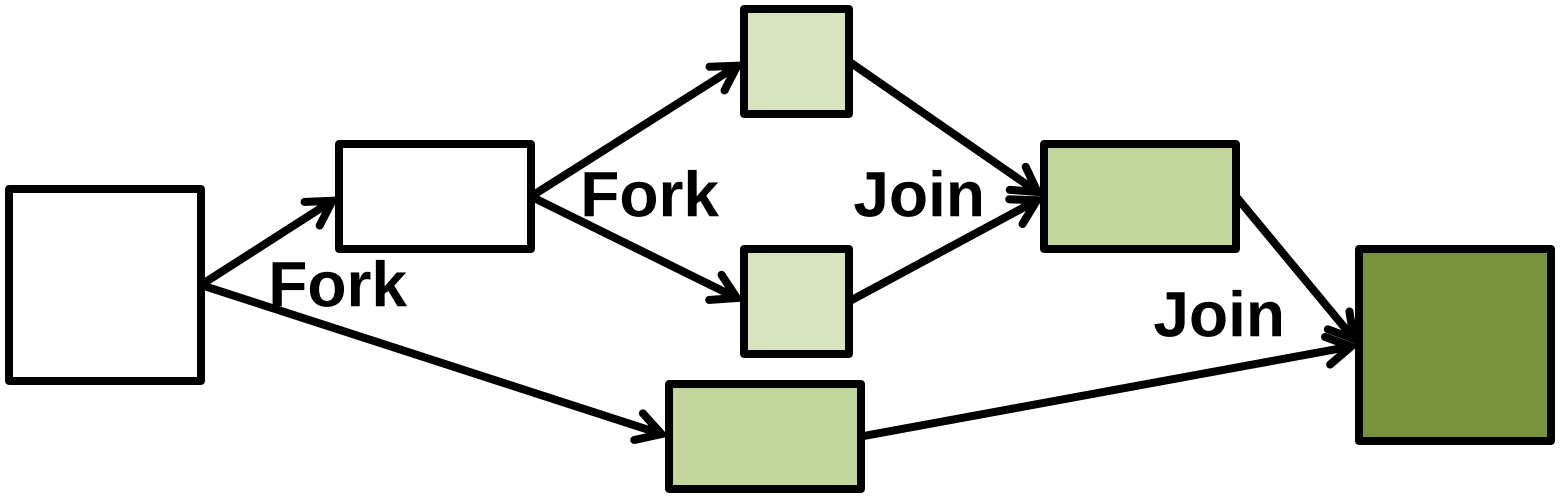
Master/Worker



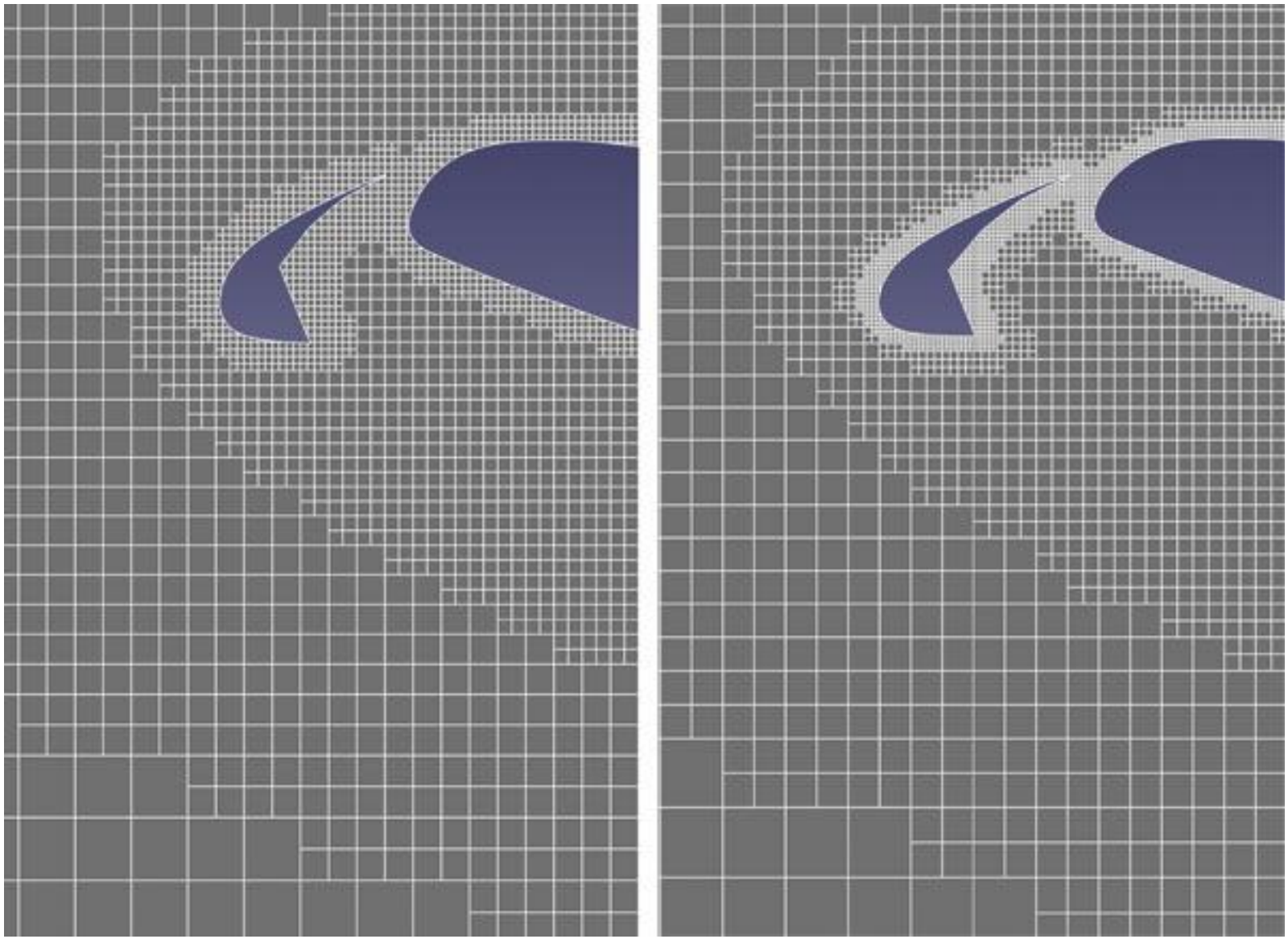
Fork/Join

Fork – процесс разбиения задачи на более мелкие подзадачи.

Join – процесс объединения результатов счета.



Fork/Join



Map/Reduce

Как обрабатывать много данных?

1) **Facebook** 15 терабайт/день
2) **Google** – 20 петабайт/день (2008)
3) **Большой Андронный Коллайдер**
   15 петабайт/год
4) Астрономия и астрофизика
   Large Synoptic Survey Telescope (2015)
   1.28 петабайт/год

Map/Reduce

Проблемы:

1) Очень много данных нужно хранить.
2) Задачи рассчитаны на десятки тысяч компьютеров.
   Требуются очень большие кластеры.
3) Данные не помещаются в оперативную память.

Map/Reduce

Проблемы:

4) Отказы – это норма.

   10 тысяч компьютеров с мат.ожиданием бесперебойной работы ~1000 дней работы это 10 отказов в день

5) Разрабатывать приложения на низком уровне сложно и трудно.
   MPI не устойчив к отказам.

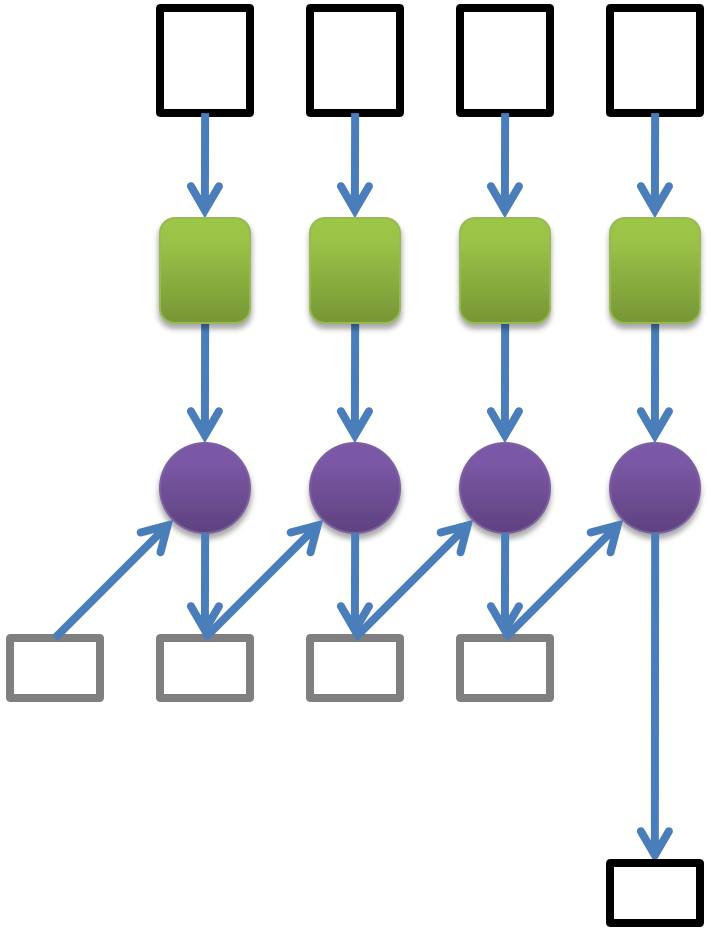
Map/Reduce

Операция **Map**

$(k,v) \Rightarrow [(k_1,v_1),\ldots,(k_L,v_L)]$

Операция **Reduce**

$(k,[v_1,\ldots, v_N]) \Rightarrow$
$[(k_1,v'_1),\ldots,(k_M,v'_M)]$



Map/Reduce

Пример:

Как посчитать самые частые слова в английской/русской википедии?

Map/Reduce

Map: (*название*, *текст*) =>

[(*слово*$_1$, 1), …, (*слово*$_K$, 1)]

Reduce: (*слово*, [1, 1, …,1]) =>

[(*слово*, N)]

N – сумма единиц или количество употреблений *слова* в википедии.