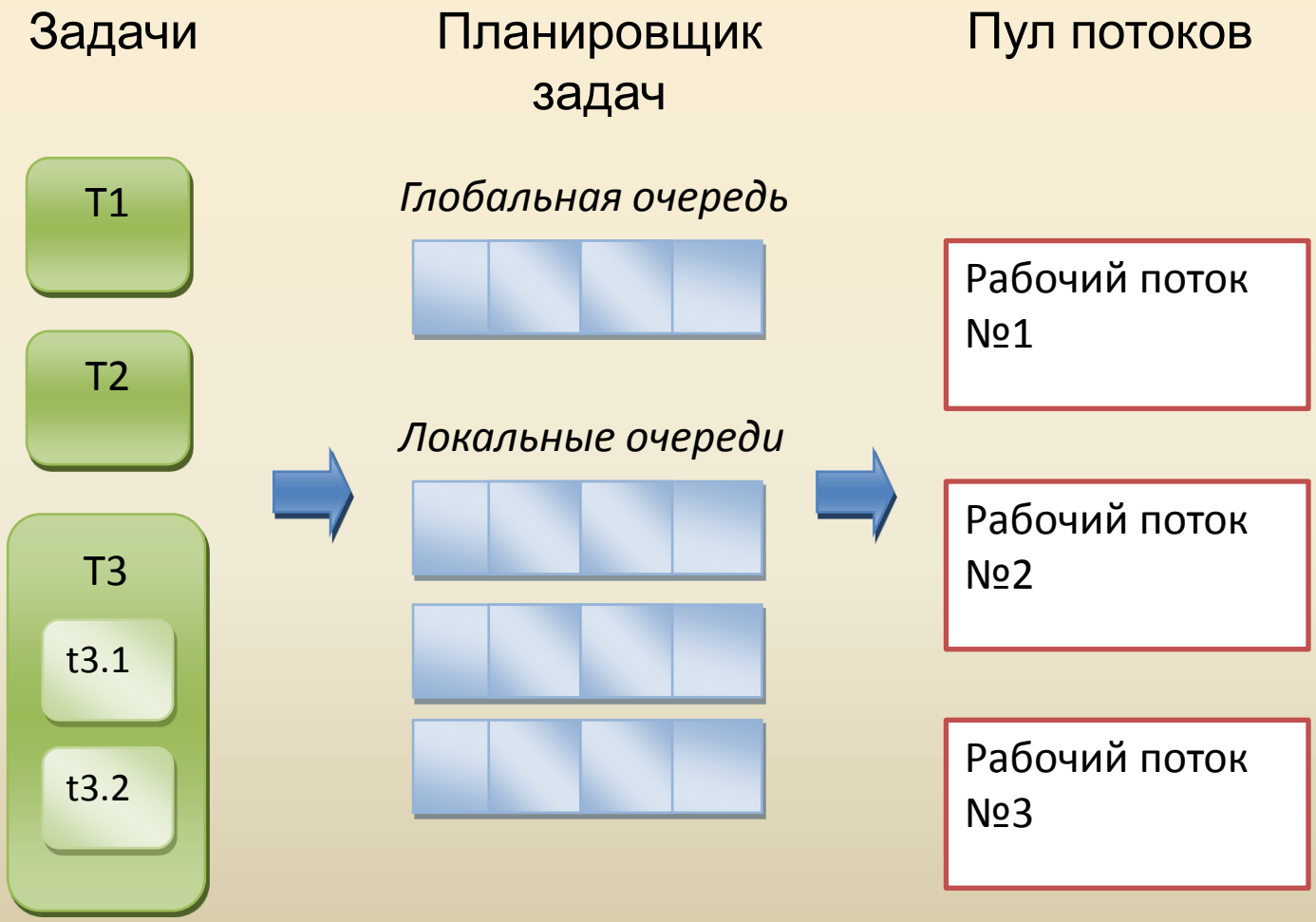
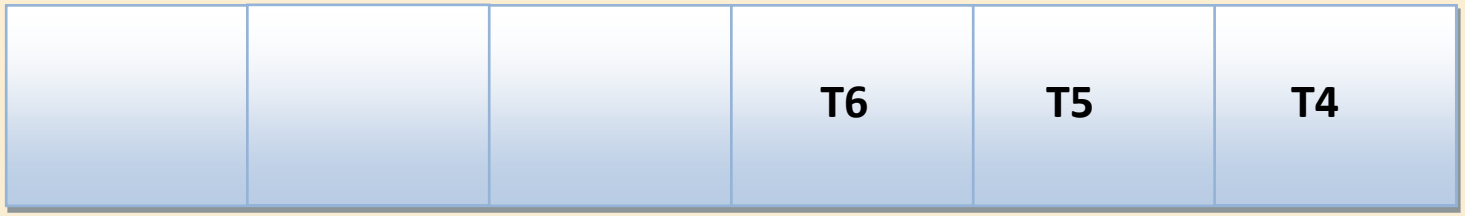
Планировщик задач

- TaskScheduler играет связующую роль между пользовательскими задачами и рабочими потоками
- Множество задач приложения выполняется одними и теми же рабочими потоками, число которых динамически оптимизируется планировщиком с учетом возможностей вычислительной системы, фактической загруженностью и прогрессом выполнения задач
- Планировщик включает в себя: очереди задач (одна глобальная и множество локальных очередей), стратегии распределения задач и рабочие потоки, которые фактически выполняют задачи

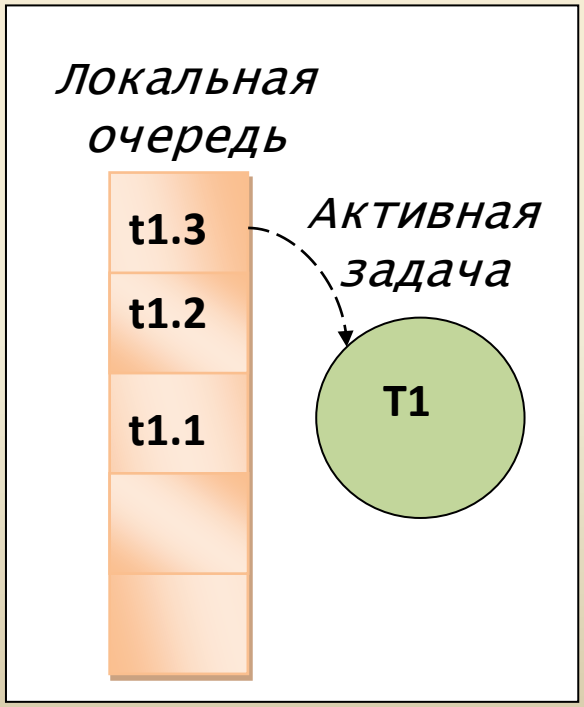
Организация планировщика



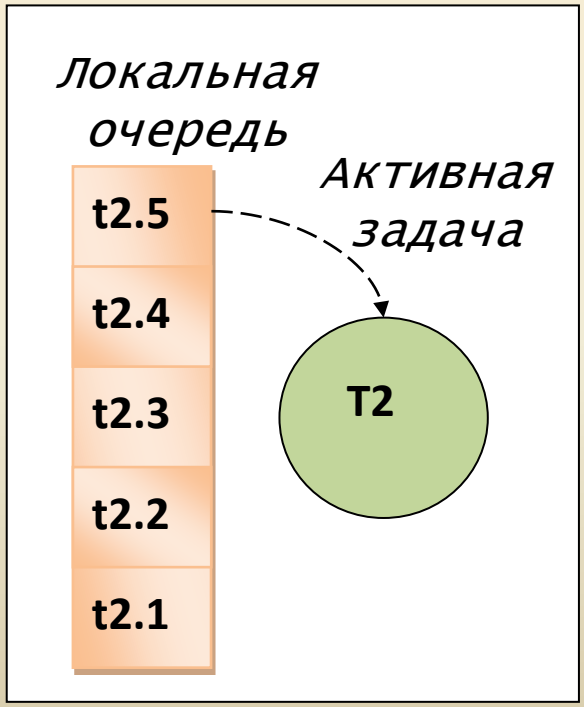
Глобальная очередь задач



Рабочий поток №1



Рабочий поток №2



...

Обработка по умолчанию

- Задачи верхнего уровня (top-most tasks), которые создаются в основном потоке или пользовательских потоках, помещаются в глобальную очередь и обрабатываются по FIFO-принципу.
- Вложенные задачи помещаются в локальные очереди, ассоциированные с фоновым потоком пула, и обрабатываются по LIFO-принципу

Вложенные задачи и LIFO

```
Task t = Task.Factory.StartNew(() =>
{
  // готовим данные для подзадачи 1
  data1 = f1(data);
  // добавляем в очередь подзадачу 1
  Task t1 = Task.Factory.StartNew(() => { work(data1); });
  // готовим данные для подзадачи 2
  data2 = f2(data);
  Task t2 = Task.Factory.StartNew(() => { work(data2); });
  // ожидаем завершения подзадач
  Task.WaitAll(t1, t2);
});
```

PreferFairness

- Порядок обработки задач можно изменить с помощью опции создания задачи PreferFairness
- Планировщик помещает вложенную задачу в глобальную очередь, организованную по принципу FIFO
- Фактический порядок обработки задач зависит от многих факторов (наличие задач в очереди, доступность потоков и др.)

Fairness

```
static void Main()
 {
    Task tMain = Task.Factory.StartNew(() => {
       t1 = Task.Factory.StartNew(() => ..);
       t2 = Task.Factory.StartNew(() => ..);
       t3 = Task.Factory.StartNew(() => ..,
            TaskCreationOptions.PreferFairness);
       t4 = Task.Factory.StartNew(() => ..,
            TaskCreationOptions.PreferFairness);
    });
 }
```

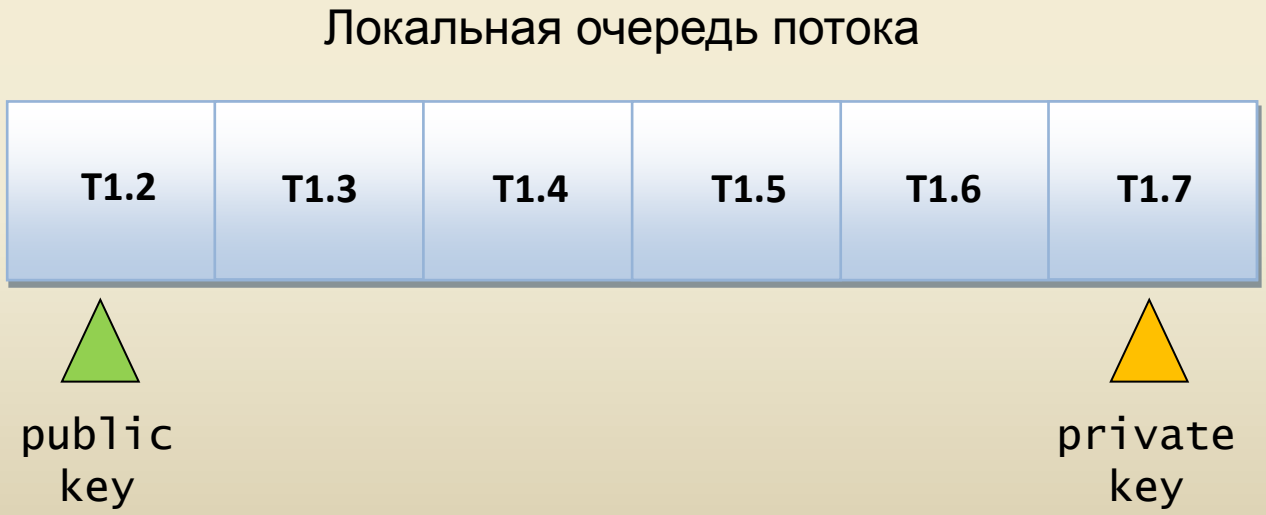
Стратегии планировщика

- Work Stealing
- Inlined threading
- Thread injection

Параметры задачи для изменения стратегий планировщика: LongRunning, PreferFairness

Стратегия заимствования задач

Стратегия *work-stealing* заключается в том, что свободный рабочий поток при условии отсутствия ожидающих задач в глобальной очереди заимствует задачу у одного из загруженных рабочих потоков.



Inlined threading

- Стратегия *inlined execution* применяется в случае блокировки выполняющейся задачи.
- Если выполняющаяся задача блокируется вызовом ожидания завершения задачи, которая находится в локальной очереди этого же рабочего потока, то планировщик выполняет задачу из очереди.
- Такая стратегия позволяет избежать взаимоблокировки рабочего потока (*dead-lock*), которая может возникнуть при определенном порядке вложенных задач в локальной очереди.
- Стратегия *inlined execution* применяется только для задач в локальной очереди блокированного рабочего потока.

Inlined-threading

```
static void Main()
{
   Task tParent = Task.Factory.StartNew(() =>
      {
         tChild1 = Task.Factory.StartNew(() => ..);
         tChild2 = Task.Factory.StartNew(() => ..);

         tChild1.Wait();
      });
}
```

Thread Injection

- Стратегия *thread-injection* применяется для оптимизации числа рабочих потоков.
- Применяются два механизма динамического изменения числа рабочих потоков.
- Первый механизм применяется с целью избежать блокировки рабочих потоков: планировщик добавляет еще один рабочий поток, если не наблюдает прогресса при выполнении задачи.
- Второй механизм стратегии *thread-injection* добавляет или удаляет рабочие потоки в зависимости от результатов выполнения предыдущих задач. Если увеличение числа потоков привело к росту производительности, то планировщик увеличивает число потоков. Если производительность не растет, то число потоков сокращается.

Long tasks & short tasks

- Планировщик не различает, находится ли поток действительно в заблокированном состоянии, ожидая какого-либо события, или поток загружен полезной длительной работой (*long-running task*).
- В случае выполнения «длинных» задач такая стратегия планировщика не улучшает, а даже ухудшает производительность. При росте числа потоков возникает конкуренция за физические ядра вычислительной системы, появляются накладные расходы на переключение контекстов.
- Чтобы избежать неправильных действий планировщика рекомендуется делать более «короткие» задачи, а длинные вычислительно-ёмкие задачи запускать с опцией LongRunning

LongRunning

- Опция при создании «долго-выполняющейся» задачи
- Задача выполняется в отдельном потоке, который не участвует в работе пула.
- Прогресс обработки задачи не контролируется.

```
Task t = Task.Factory.StartNew(VeryLongComputing,
                TaskCreationOptions.LongRunning);
```