Многопоточные паттерны

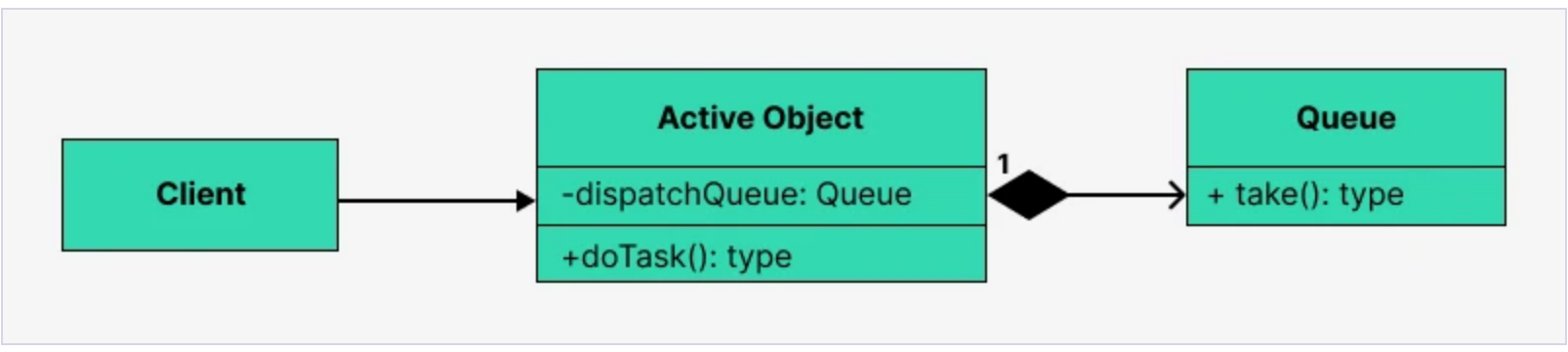
JSP & Servlets
17 уровень, 2 лекция

ОТКРЫТА

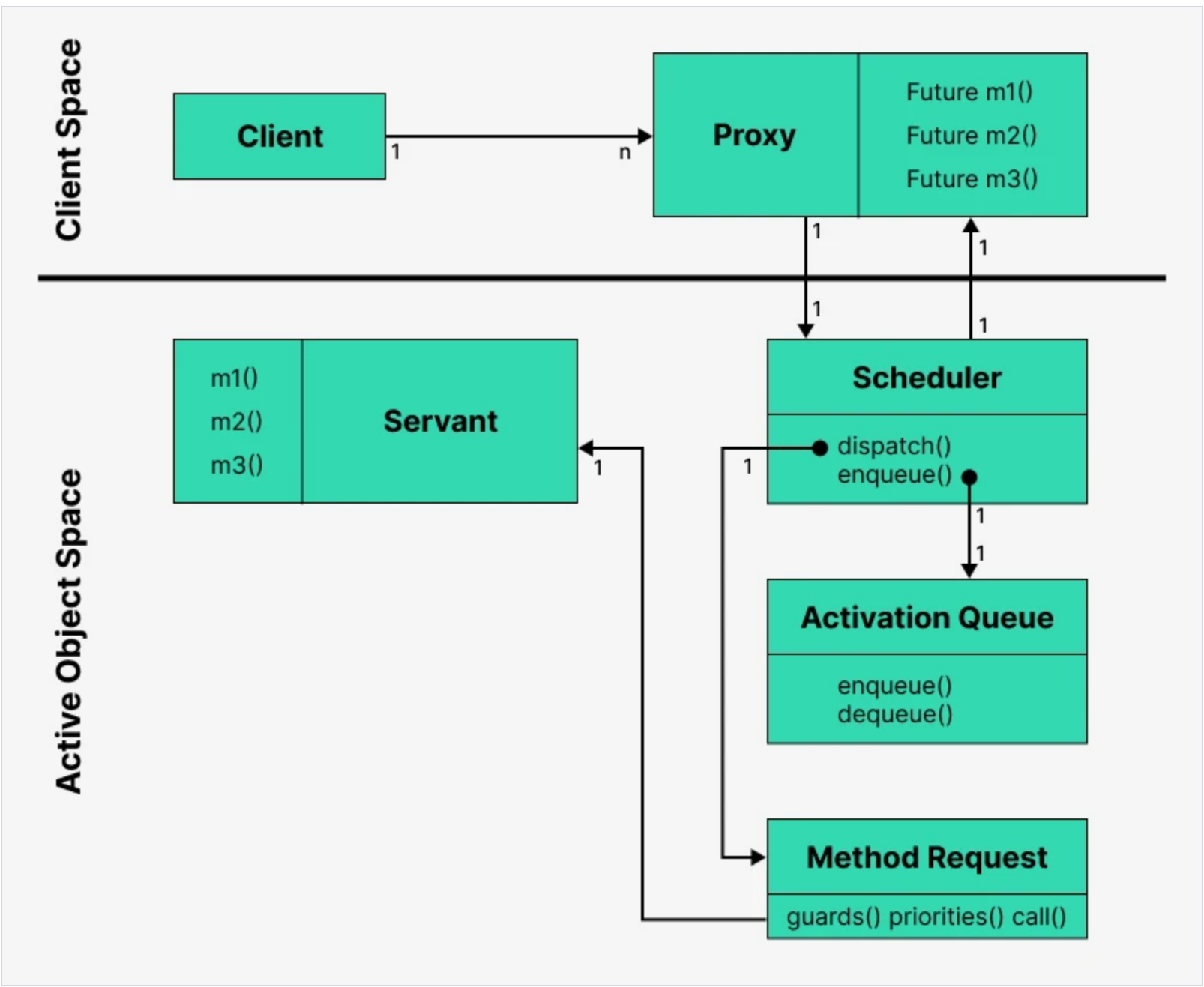
3.1 Active object

Активный объект (Active object) — это шаблон проектирования, который отделяет поток выполнения метода от потока, в котором он был вызван. Цель этого шаблона — предоставлять параллельность выполнения, используя асинхронные вызовы методов и планировщик обработки запросов.

Упрощённый вариант:



Классический вариант:



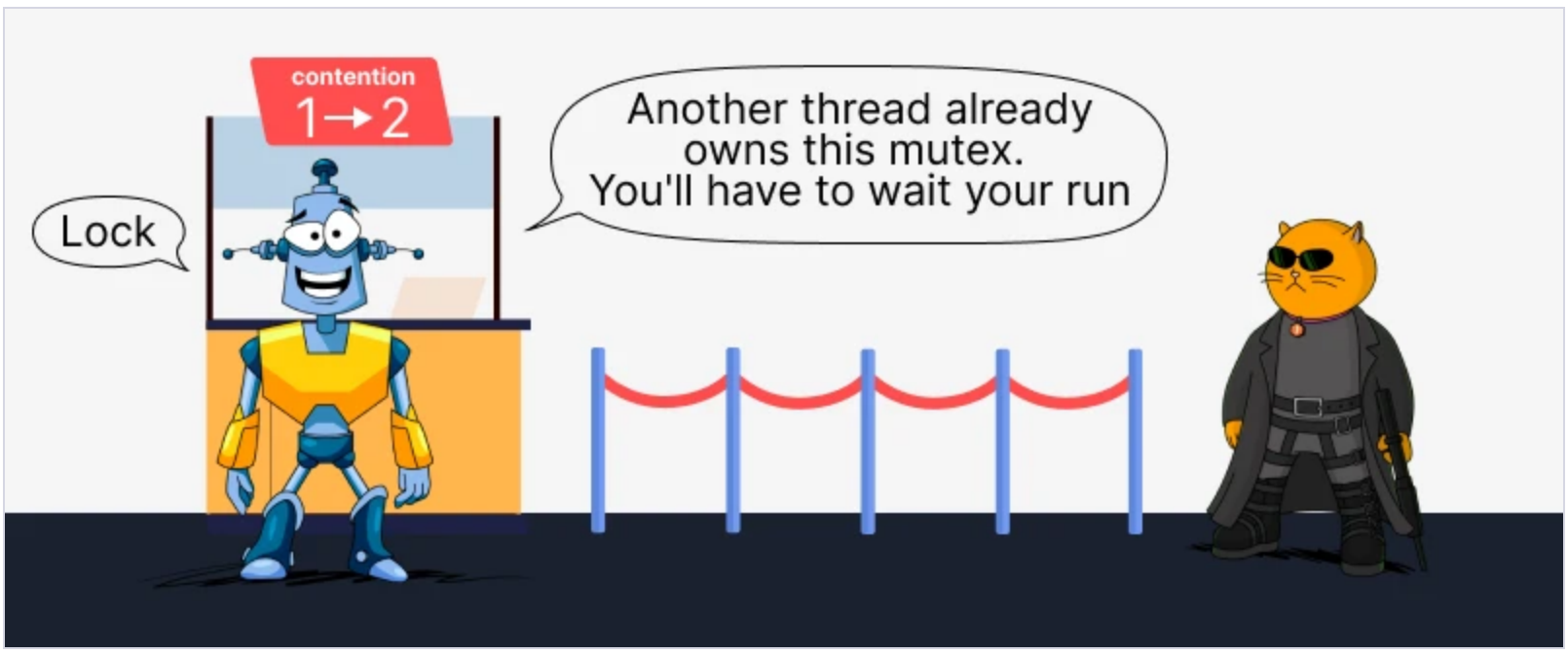
Этот шаблон состоит из шести элементов:

- Объект-заместитель (proxy), который предоставляет интерфейс к публично-доступным методам клиента.
- Интерфейс, который определяет методы доступа к активному объекту.
- Список поступающих запросов от клиентов.
- Планировщик (scheduler), который определяет порядок выполнения запросов.

- Реализация методов активного объекта.
- Процедура обратного вызова (callback) или переменная (variable) для получения клиентом результата.

3.2 Lock

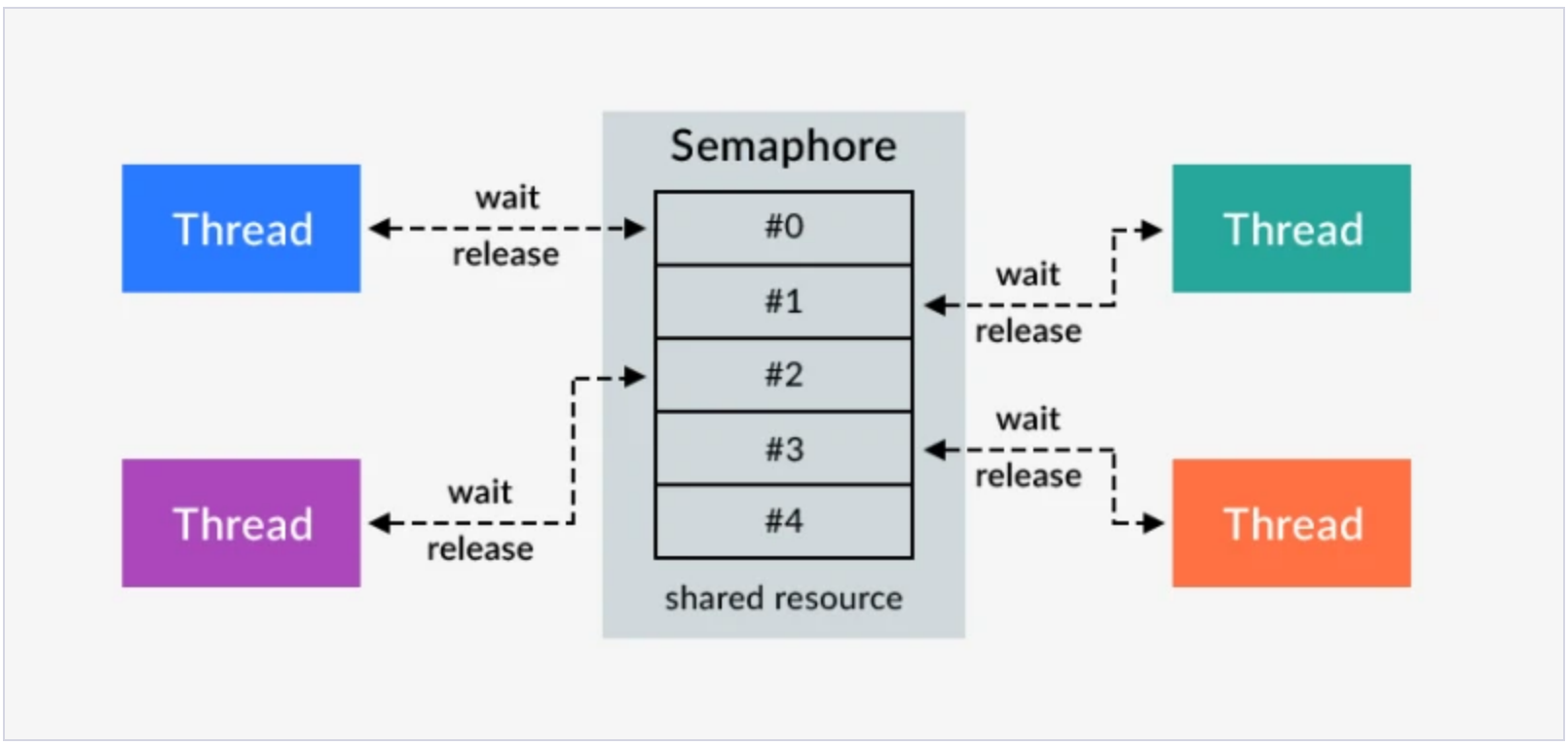
Паттерн Блокировка (Lock) — механизм синхронизации, позволяющий обеспечить исключительный доступ к разделяемому ресурсу между несколькими потоками. Блокировки — это один из способов обеспечить политику управления распараллеливанием.



В основном, используется мягкая блокировка, при этом предполагается что каждый поток пытается “захватить блокировку” перед доступом к соответствующему разделяемому ресурсу.

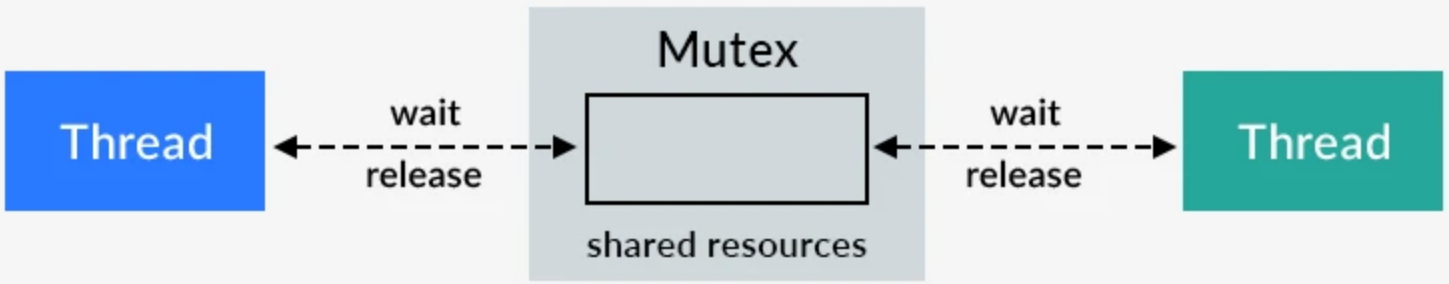
Однако в некоторых системах предоставляется механизм обязательной блокировки, при его использовании попытка несанкционированного доступа к заблокированному ресурсу будет прервана через создание исключения в потоке, который пытался получить доступ.

Семафор — самый простой тип блокировки. С точки зрения доступа к данным не делается никаких различий между режимами доступа: общим (только чтение) или эксклюзивным (чтение и запись). В режиме общего доступа несколько потоков могут запросить блокировку для доступа к данным в режиме “только чтение”. Также используется эксклюзивный режим доступа в алгоритмах обновления и удаления.



Типы блокировок различают по стратегии блокировки продолжения исполнения потока. В большинстве реализаций запрос блокировки препятствует дальнейшему исполнению потока, пока не появится доступ к заблокированному ресурсу.

Спинлок — это блокировка, которая ожидает в цикле, пока не появится доступ. Такая блокировка очень эффективна, если поток ожидает блокировку незначительный интервал времени, это позволяет избежать избыточной перепланировки потоков. Затраты на ожидание доступа будут значительными при длительном удержании блокировки одним из потоков.

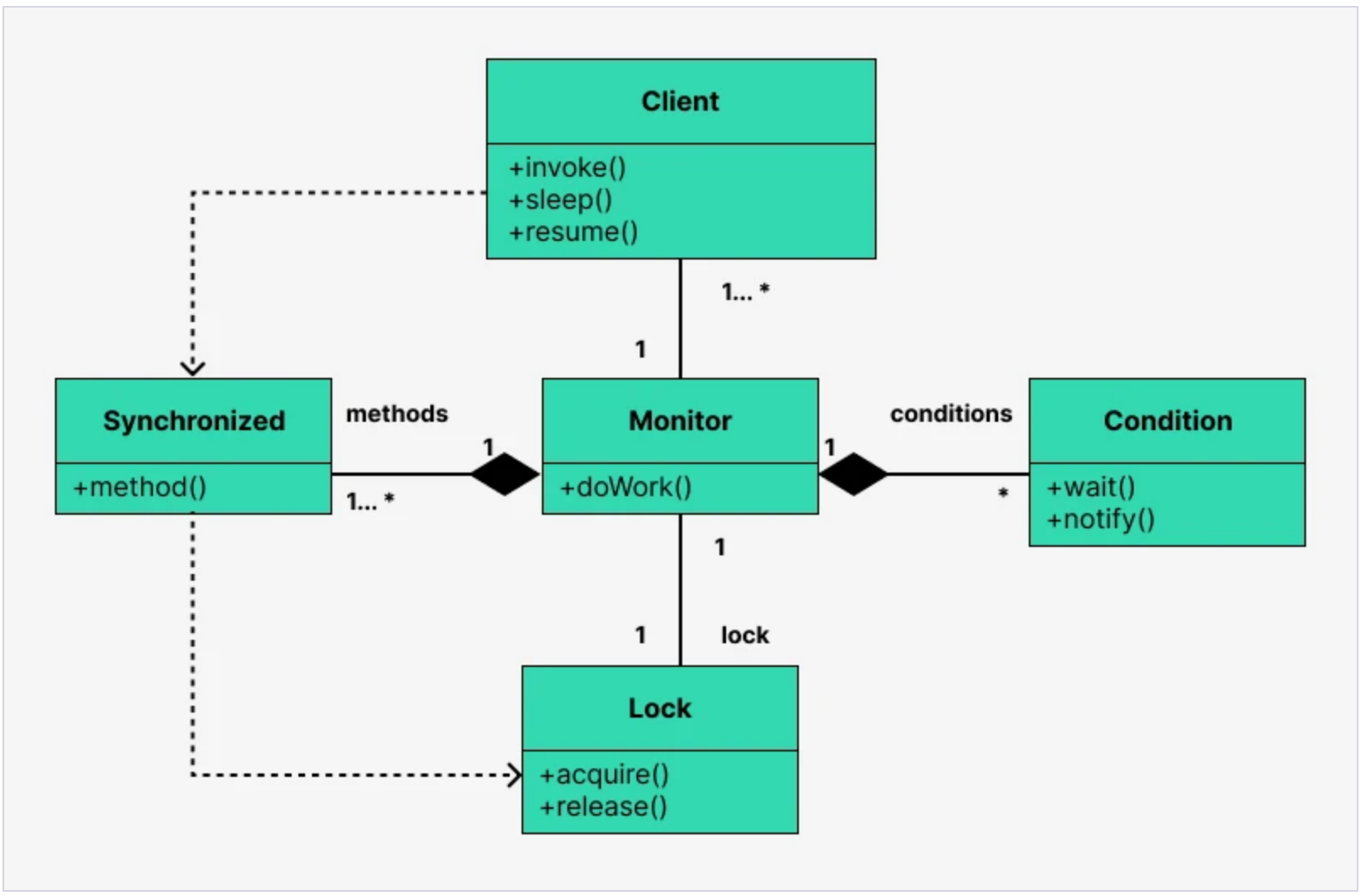


Для эффективной реализации механизма блокировки требуется поддержка на аппаратном уровне. Аппаратная поддержка может быть реализована в виде одной или нескольких атомарных операций, таких как “test-and-set”, “fetch-and-add” или “compare-and-swap”. Такие инструкции позволяют без прерываний проверить, что блокировка свободна, и если это так, то занять блокировку.

3.3 Monitor

Паттерн Монитор — высокоуровневый механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к разделяемым ресурсам. Подход к синхронизации двух или более компьютерных задач, использующих общий ресурс, обычно аппаратуру или набор переменных.

При многозадачности, основанной на мониторах, компилятор или интерпретатор прозрачно для программиста вставляет код блокировки-разблокировки в оформленные соответствующим образом процедуры, избавляя программиста от явного обращения к примитивам синхронизации.



Монитор состоит из:

- набора процедур, взаимодействующих с общим ресурсом
- мьютекса
- переменных, связанных с этим ресурсом
- инварианта, который определяет условия, позволяющие избежать состояние гонки

Процедура монитора захватывает мьютекс перед началом работы и держит его или до выхода из процедуры, или до момента ожидания определенного условия. Если каждая процедура гарантирует, что перед освобождением мьютекса инвариант истинен, то никакая задача не может получить ресурс в состоянии, ведущем к гонке.

Именно так в Java работает оператор **synchronized** в паре с методами `wait()` и `notify()`.

3.4 Double check locking

Блокировка с двойной проверкой (Double checked locking) — параллельный шаблон проектирования, предназначающийся для уменьшения накладных расходов, связанных с получением блокировки.

Сначала проверяется условие блокировки без какой-либо синхронизации. Поток пытается получить блокировку, только если результат проверки говорит о том, что получение блокировки необходимо.

```
1  //Double-Checked Locking
2  public final class Singleton {
3  private static Singleton instance; //Don't forget volatile modifier
4
5  public static Singleton getInstance() {
6       if (instance == null) {                //Read
7
8           synchronized (Singleton.class) {    //
9               if (instance == null) {         //Read Write
10                  instance = new Singleton(); //
11              }
12          }
13      }
14   }
```

Как создать объект-синглетон в потокобезопасной среде?

```
1  public static Singleton getInstance() {
2     if (instance == null)
3      instance = new Singleton();
4  }
```

Если ты создаешь объект Singleton из разных потоков, то может возникнуть ситуация, когда создастся несколько объектов одновременно, а это неприемлемо. Поэтому разумно обернуть создание объекта в оператор synchronized.

```
1  public static Singleton getInstance() {
2      synchronized (Singleton.class) {
3          if (instance == null)
4          instance = new Singleton();
5      }
6  }
```

Такой подход будет работать, однако у него есть небольшой минус. После того, как объект создан, каждый раз при попытке его получить в будущем будет выполняться проверка в блоке synchronized, а значит будет лочиться текущий поток и все, что с этим связано. Так что этот код можно немного оптимизировать:

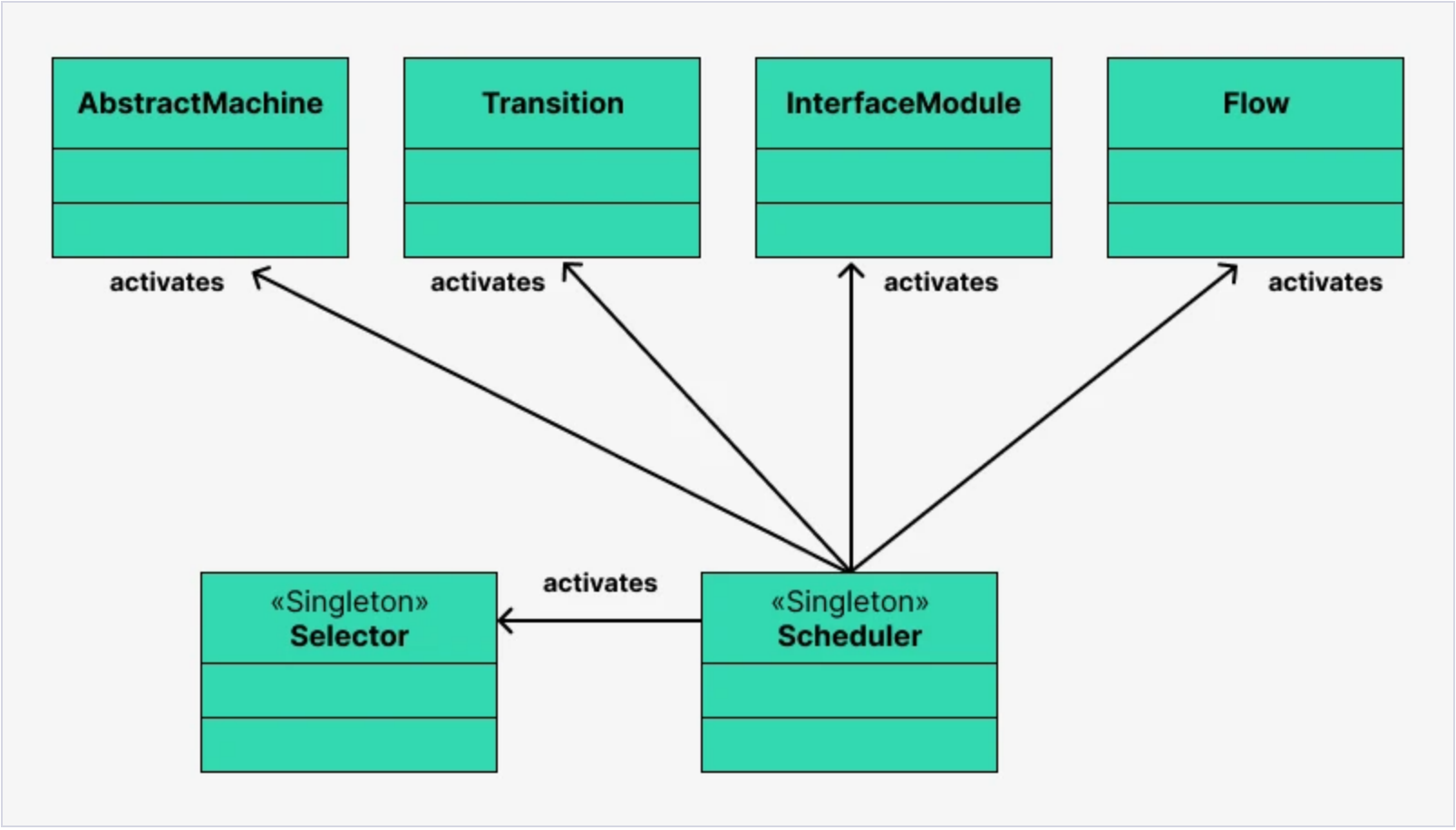
```
1  public static Singleton getInstance() {
2       if (instance != null)
3          return instance;
4
5      synchronized (Singleton.class) {
6          if (instance == null)
7          instance = new Singleton();
8      }
9  }
```

На некоторых языках и/или на некоторых машинах невозможно безопасно реализовать этот шаблон. Поэтому иногда его называют анти-паттерном. Такие особенности привели к появлению отношения строгого порядка “happens before” в Java Memory Model и C++ Memory Model.

Обычно он используется для уменьшения накладных расходов при реализации ленивой инициализации в многопоточных программах, например, в составе шаблона проектирования Одиночка. При ленивой инициализации переменной инициализация откладывается до тех пор, пока значение переменной не понадобится при вычислениях.

3.5 Scheduler

Планировщик (Scheduler) — параллельный шаблон проектирования, обеспечивающий механизм реализации политики планирования, но при этом не зависящий ни от одной конкретной политики. Управляет порядком, в соответствии с которым потокам предстоит выполнить последовательный код, используя для этого объект, который явным образом задаёт последовательность ожидающих потоков.



Более детально с этим шаблоном вы можете ознакомиться в официальной документации. Также есть отличная статья на русском.

‹ Предыдущая лекция

Следующая лекция ›