Kotlin

Proseminar: Fortgeschrittene Programmierkonzepte

Christian Konersmann, Finn Paul Lippok, Paul Lukas

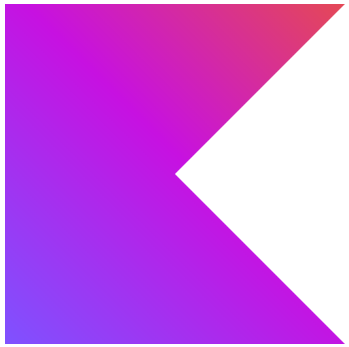
05.05.2025

Was ist Kotlin?

- **Statisch typisierte** und **objektorientierte** Programmiersprache.
- **Basierend auf Java und der JVM** mit vollständiger **Interoperabilität** zu beiden.

Was ist Kotlin?

- **Statisch typisierte** und **objektorientierte** Programmiersprache.
- **Basierend auf Java und der JVM** mit vollständiger **Interoperabilität** zu beiden.



- **Wichtigste Vorteile gegenüber Java:**
  - Klare und präzise Syntax.
  - Erweiterte Funktionen wie Null-Sicherheit.
  - Umfassende Multiplattform-Entwicklungsmöglichkeiten.

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

- Keine explizite Klassendeklaration erforderlich.

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

- Keine explizite Klassendeklaration erforderlich.
- Verwendung des Schlüsselworts `fun` zur Funktionsdeklaration.

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

- Keine explizite Klassendeklaration erforderlich.
- Verwendung des Schlüsselworts `fun` zur Funktionsdeklaration.
- Standardzugriffsmodifikator ist `public`.

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

- Keine explizite Klassendeklaration erforderlich.
- Verwendung des Schlüsselworts `fun` zur Funktionsdeklaration.
- Standardzugriffsmodifikator ist `public`.
- `args`-Parameter ist optional.

Main-Methode

Java Main-Methode

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

Kotlin Main-Methode

```kotlin
fun main() {
    println("Hello, World!")
}
```

- Keine explizite Klassendeklaration erforderlich.
- Verwendung des Schlüsselworts `fun` zur Funktionsdeklaration.
- Standardzugriffsmodifikator ist `public`.
- `args`-Parameter ist optional.
- Semikolons sind nicht erforderlich.

Variablen-Deklaration

Java

```java
int a = 5;
final String b = "Hallo";
```

Kotlin

```kotlin
var a: Int = 5
val b: String = "Hallo"
```

- `var` für veränderliche Variablen, `val` für unveränderliche Variablen.

Variablen-Deklaration

Java

```java
int a = 5;
final String b = "Hallo";
```

Kotlin

```kotlin
var a: Int = 5
val b: String = "Hallo"
```

- `var` für veränderliche Variablen, `val` für unveränderliche Variablen.
- Typangabe nach dem Variablennamen mit Doppelpunkt.

Variablen-Deklaration

Java

```java
int a = 5;
final String b = "Hallo";
```

Kotlin

```kotlin
var a: Int = 5
val b: String = "Hallo"
```

- `var` für veränderliche Variablen, `val` für unveränderliche Variablen.
- Typangabe nach dem Variablennamen mit Doppelpunkt.
- In Kotlin gibt es **keine** primitiven Typen.

Variablen-Deklaration

Java

```java
int a = 5;
final String b = "Hallo";
```

Kotlin

```kotlin
var a: Int = 5
val b: String = "Hallo"
```

- `var` für veränderliche Variablen, `val` für unveränderliche Variablen.
- Typangabe nach dem Variablennamen mit Doppelpunkt.
- In Kotlin gibt es **keine** primitiven Typen.

Variablen-Deklaration

Java

```java
int a = 5;
final String b = "Hallo";
```

Kotlin

```kotlin
var a: Int = 5
val b: String = "Hallo"
```

- `var` für veränderliche Variablen, `val` für unveränderliche Variablen.
- Typangabe nach dem Variablennamen mit Doppelpunkt.
- In Kotlin gibt es **keine** primitiven Typen.

Kotlin unterstützt **Typinferenz**, d.h. der Typ kann weggelassen werden.

- Der Compiler leitet den Typ aus dem initialisierten Wert ab.
- Beispiel: `var a = 5` ist auch möglich.

Klassen

Java

```java
public class Verkaufsperson {
  public final String name;
  private Double provision;

  public Verkaufsperson (String name, Double
      provision) {...}
}
```

Klassen

Java

```java
public class Verkaufsperson {
  public final String name;
  private Double provision;

  public Verkaufsperson (String name, Double
      provision) {...}
}
```

Kotlin

```kotlin
class Verkaufsperson() {



  val name: String
  private var provision: Double
}
```

Klassen

Java

```java
public class Verkaufsperson {
  public final String name;
  private Double provision;

  public Verkaufsperson (String name, Double
      provision) {...}
}
```

Kotlin

```kotlin
class Verkaufsperson(
  name: String,
  provision: Double = 0.2
) {
  val name: String = name
  private var provision: Double = provision
}
```

Klassen

Java

```java
public class Verkaufsperson {
  public final String name;
  private Double provision;

  public Verkaufsperson (String name, Double
      provision) {...}
}
```

Kotlin

```kotlin
class Verkaufsperson(
  val name: String,
  private var provision: Double = 0.2
) {


}
```

Klassen

Java

```java
public class Verkaufsperson {
  public final String name;
  private Double provision;

  public Verkaufsperson (String name, Double
      provision) {...}
}
```

Kotlin

```kotlin
class Verkaufsperson(
  val name: String,
  private var provision: Double = 0.2
) {}
```

- Ähnlich wie Java-Records, aber flexibler.
- Nur vererbbar, wenn als open deklariert.

Properties

Kotlin: Properties

```
1  class Verkaufsperson(val name: String,
2      private var provision: Double = 0.2) {
3
4    var umsatz : Int = 0
5
6
7
8
9
10
11 }
```

Properties

Kotlin: Properties Zugriffsmodifikator

```
1  class Verkaufsperson(val name: String,
2      private var provision: Double = 0.2) {
3
4    var umsatz : Int = 0
5      private set
6
7
8
9
10
11 }
```

Properties

Kotlin: Benutzerdefinierte Zugriffsmethoden

```
1  class Verkaufsperson(val name: String,
2      private var provision: Double = 0.2) {
3
4    var umsatz : Int = 0
5      private set(value) {
6        if (value < 0)
7          throw IllegalArgumentException("Umsatz␣muss␣
               positiv␣sein")
8        field = value
9      }
10 }
```

Properties

Kotlin: Benutzerdefinierte Zugriffsmethoden

```
1  class Verkaufsperson(val name: String,
2      private var provision: Double = 0.2) {
3
4    var umsatz : Int = 0
5      private set(value) {
6        if (value < 0)
7          throw IllegalArgumentException("Umsatz muss
               positiv sein")
8        field = value
9      }
10 }
```

- Punkt-Notation ruft automatisch Setter/Getter auf.
- Beispiel: `verkaufsperson.umsatz = -1` wirft eine `IllegalArgumentException`.

Null safety

Motiviation: Null safety

Java example

```
1 Verkaufsperson person = null;
2 System.out.println(person.name);
```

- Exception in thread "main" java.lang.NullPointerException
- Kann zu Programmabbruch führen oder weitere Fehler nach sich ziehen

Null safety

- unterscheidung zwischen nullable types und non-nullable types
- Programmierer muss Null safety gewährleisten

```
var a : String = "a␣is␣non-nullable"
var b : String? = "b␣is␣nullable"
```

Null safety: Safe call operator

Ziel: sicherer Zugriff auf Datenfeld

in Java

```java
private final SalesPerson supervisor;

public void printSupervisor() {
  if (supervisor == null)
    System.out.println("null");
  else System.out.println(supervisor.name);
}
```

in Kotlin

```kotlin
val supervisor: SalesPerson? = null

fun printSupervisor() {
  println(supervisor?.name)
}
```

Null safety: Safe call operator

```
val name: String? = supervisor?.supervisor?.name
```

```
supervisor?.supervisor?.salesVolume = 0.0
```

Null safety: Elvis operator

```java
public void printSupervisor() {
  if (supervisor == null)
    System.out.println("No supervisor");
  else System.out.println(supervisor.name);
}
```

```
fun printSupervisor() {
  println(supervisor?.name ?: "No supervisor")
}
```

Null safety: Elvis operator

```
public void printSupervisor() {
  if (supervisor == null)
    System.out.println("No supervisor");
  else System.out.println(supervisor.name);
}
```

```
fun printSupervisor() {
  println(supervisor?.name ?: "No supervisor")
}
```

Null safety: Not-null assertion

```
val couldBeNull: String? = null
var b: String = possiblyNull!!
```

- Kann zu NullPointerExceptions führen

Null safety: Nullable Receiver

- Baut auf Extension functions
- erlauben Methodenaufruf auf nullable types
- Null Werte werden innerhalb der Methode behandelt

```
fun SalesPerson?.print() {
    if (this == null) return println("This␣person␣does␣
        not␣exist.")
    return println("$name:␣$salesVolume␣sold")
}
```

```
var sales: SalesPerson? = null
sales.print() // This person does not exist
```

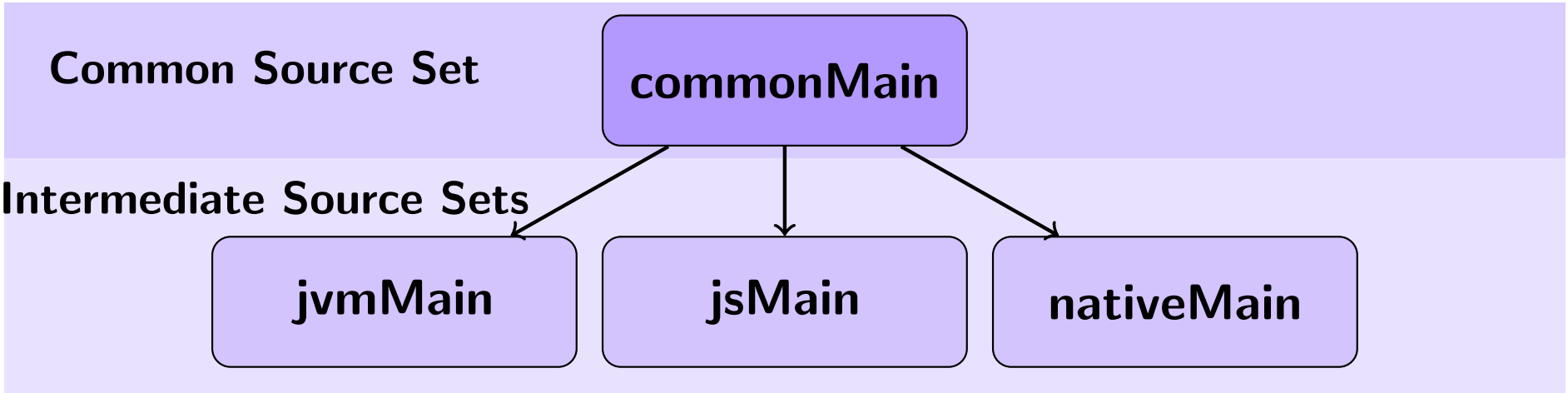
Multiplatform

- native binarie
- verschiedene Targets
- hierarchische Projektstruktur

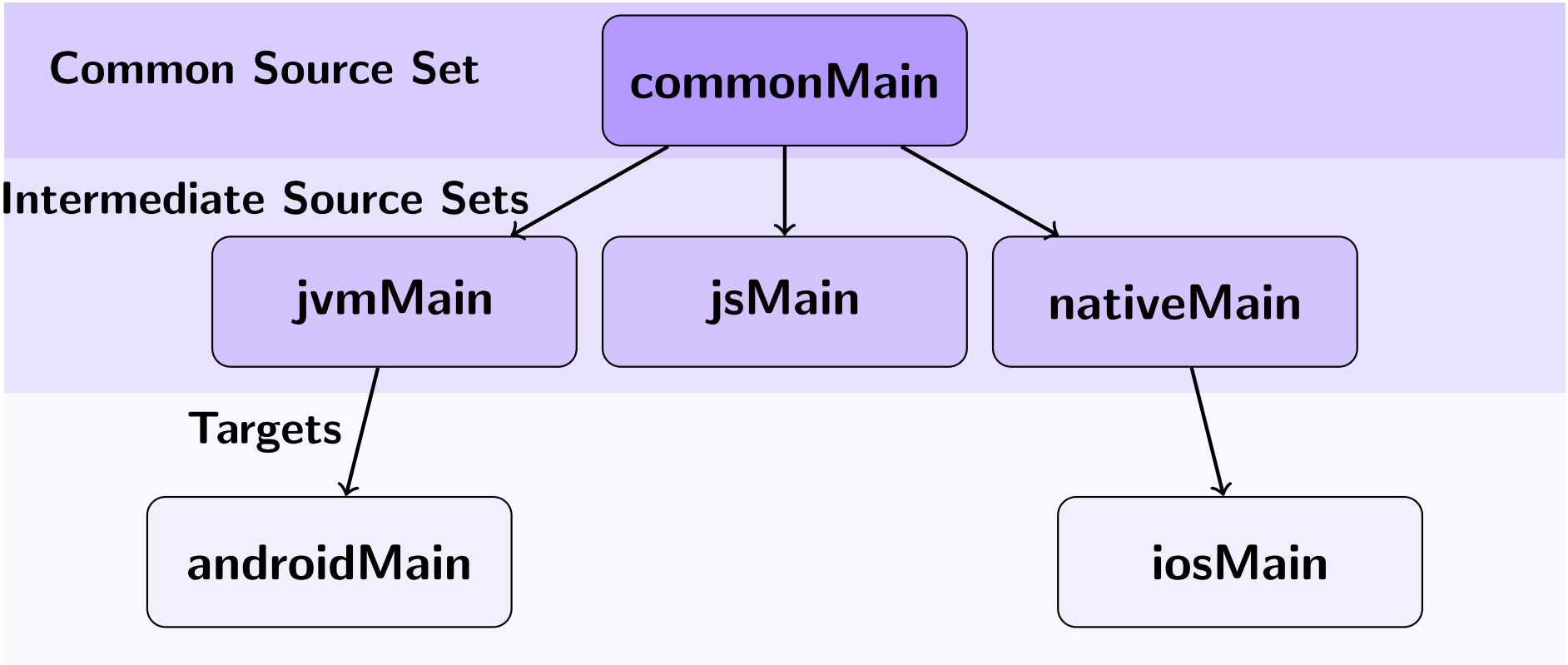
Multiplatform: hierarchische Projektstruktur

Common Source Set **commonMain**

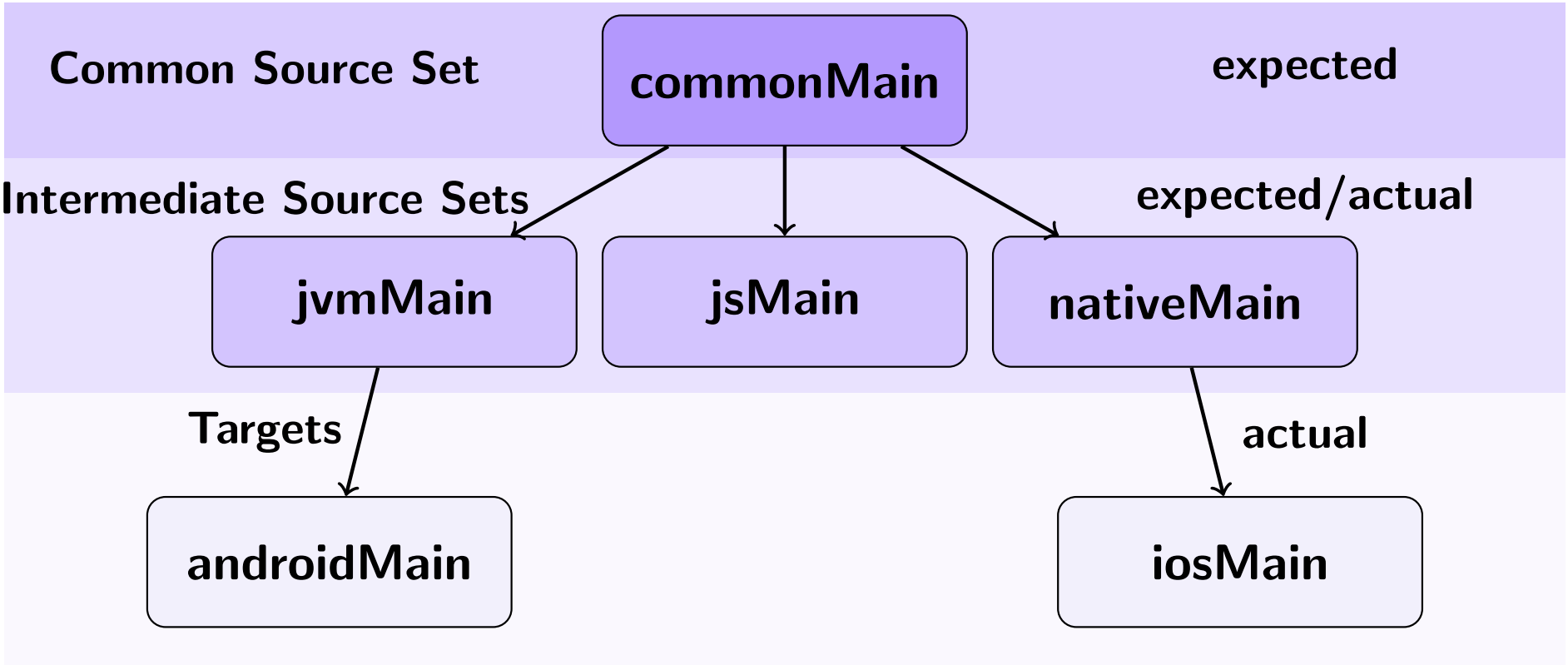
Multiplatform: hierarchische Projektstruktur



Multiplatform: hierarchische Projektstruktur



Multiplatform: hierarchische Projektstruktur



Android

- Jetpack compose
- Coroutines
- Beispiel

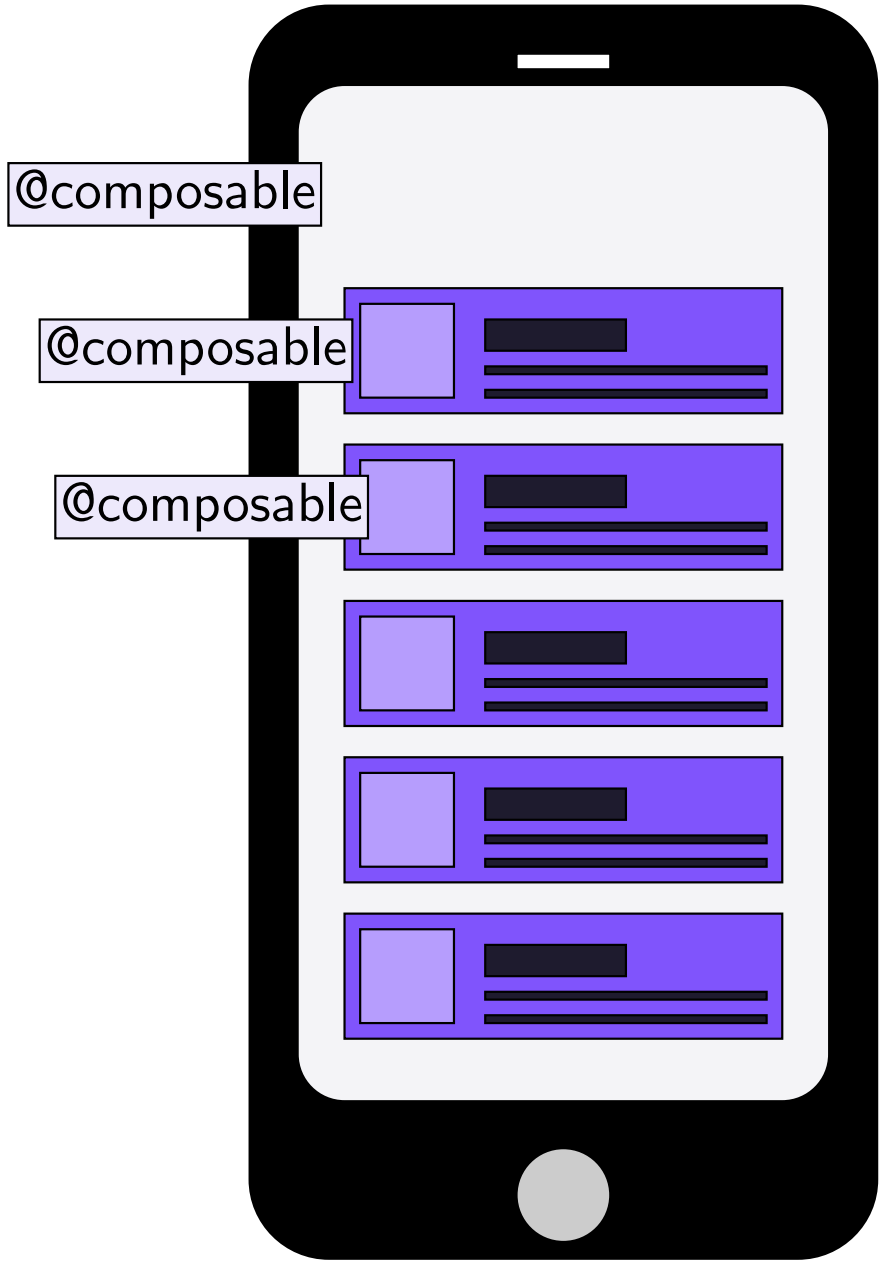
Android: Jetpack compose

- Ui Tool
- @composables
- Kotlin basiert

Android: Composable: Beispiel

States

Home

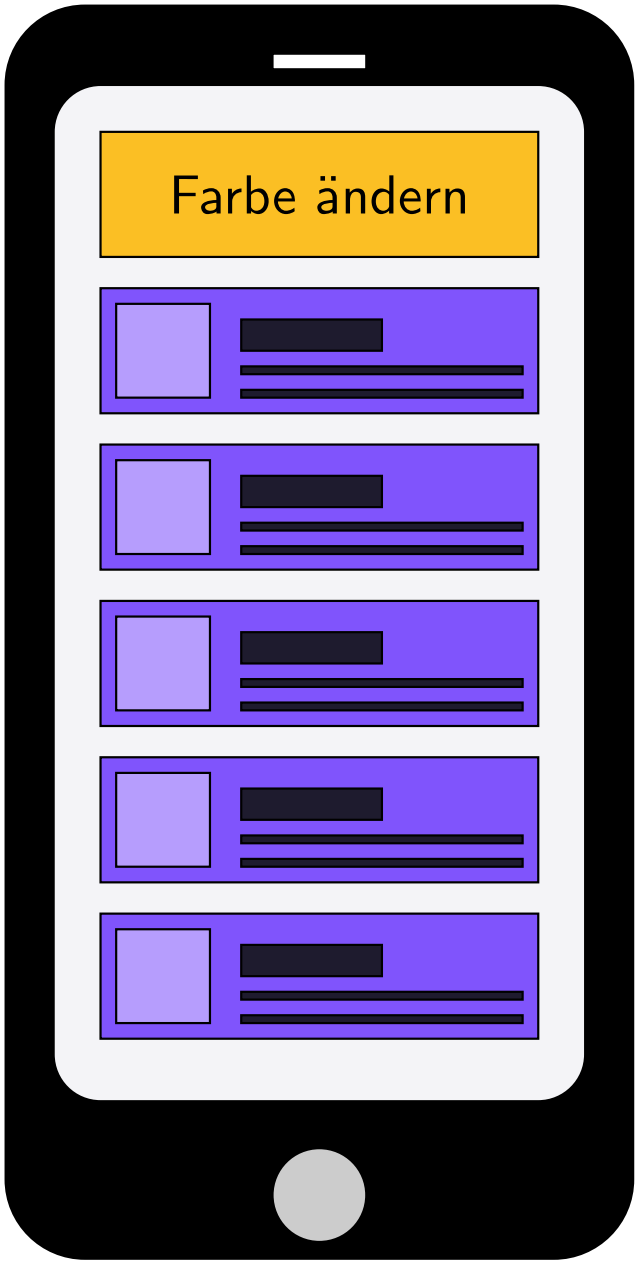


Android: Composable: Beispiel

States

Home

Farbe: Grau

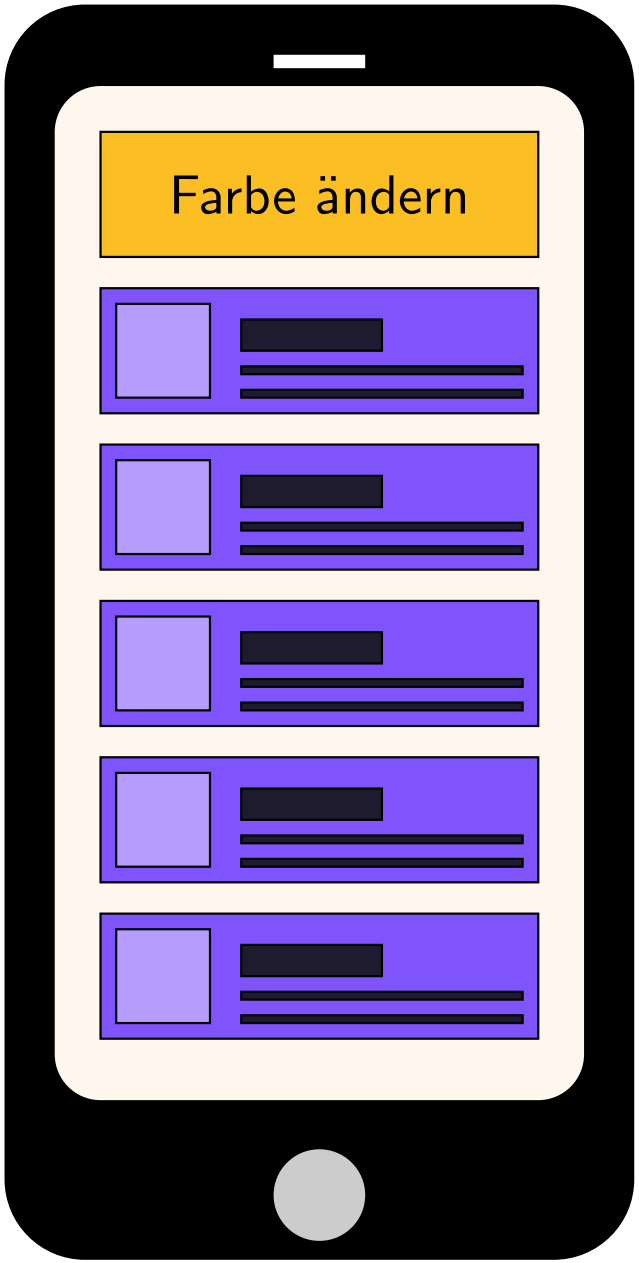


Android: Composable: Beispiel

States

Home

Farbe:
Grau Gelb

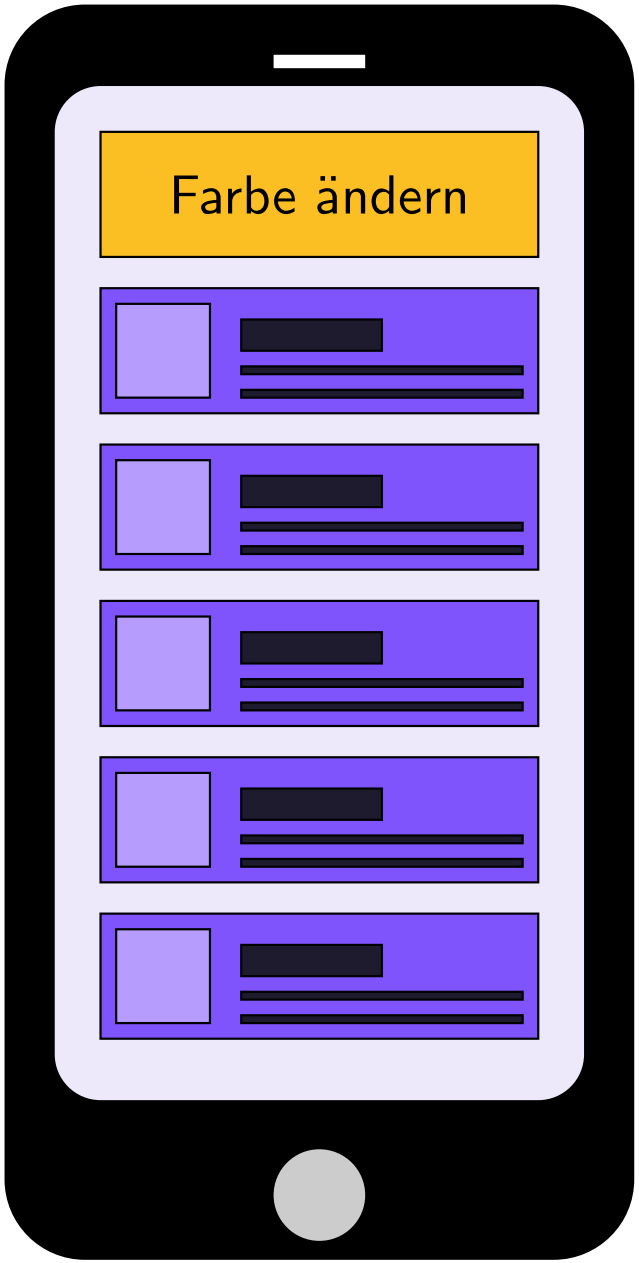


Android: Composable: Beispiel

States

Home

Farbe:
Grau Lila

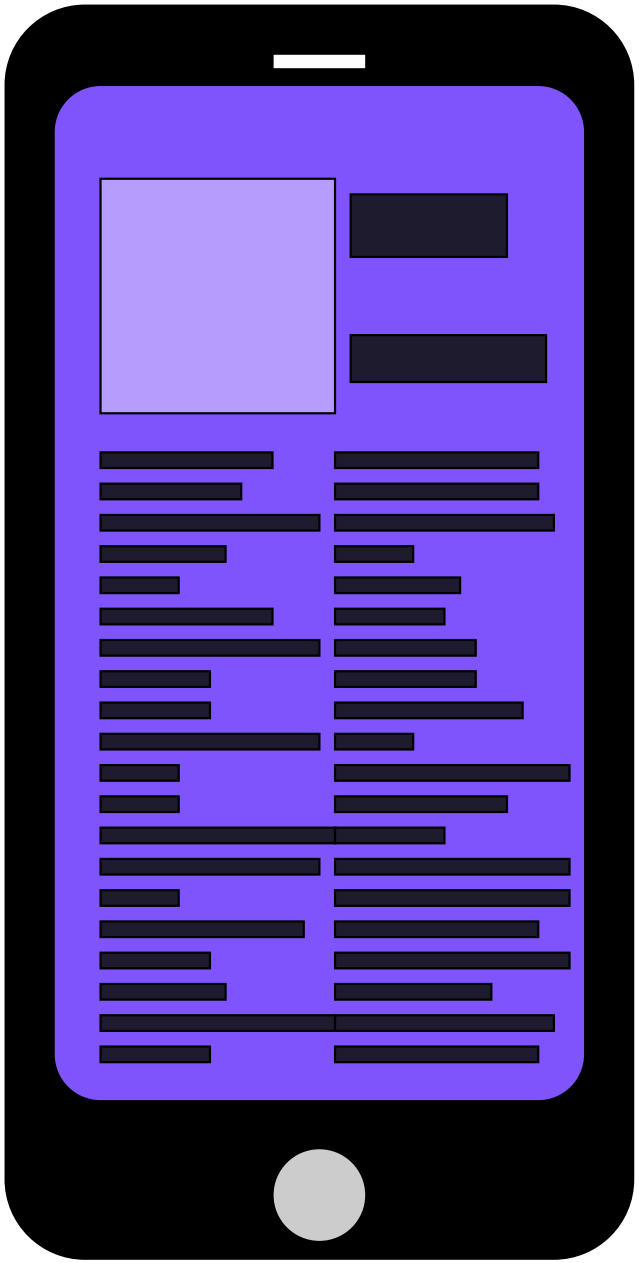


Android: Composable: Beispiel

States

Profil

Farbe:
Lila

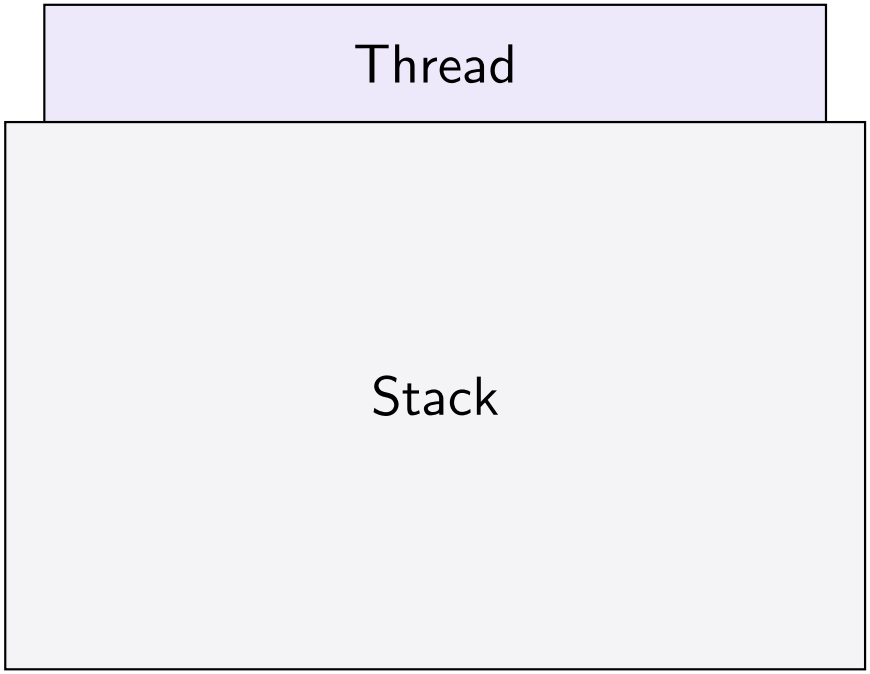


Android: Coroutines

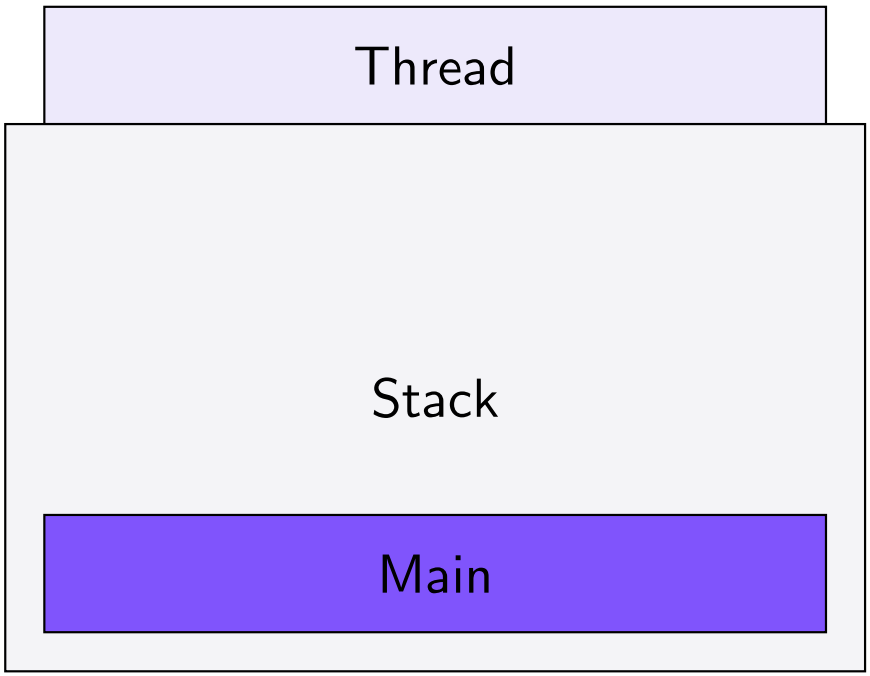
- Threads unterschiede
- Threads sind:
- subprocess
- Os heavy
- Build from(stack...)
- besseres organisier System
- Thread pools:

Pool name	(Max. Threads)	(Min. Threads)
Main	(1)	(1)
Default	(Cpu Cores)	(2)
I/O	(64)	(0)

Android: Coroutines



Android: Coroutines



-Subprocese

-Besteht aus:

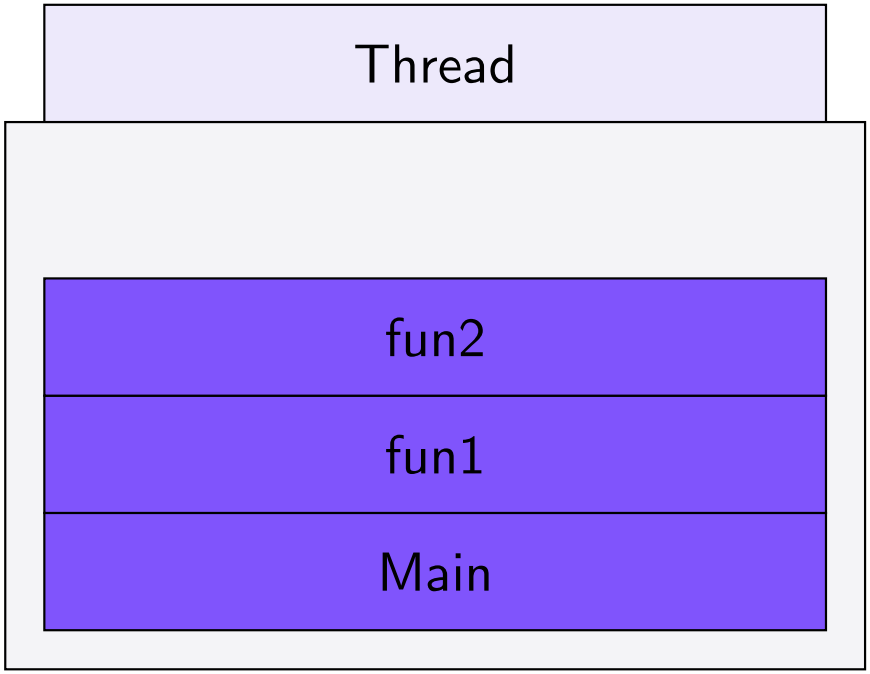
Stack

Thread Context

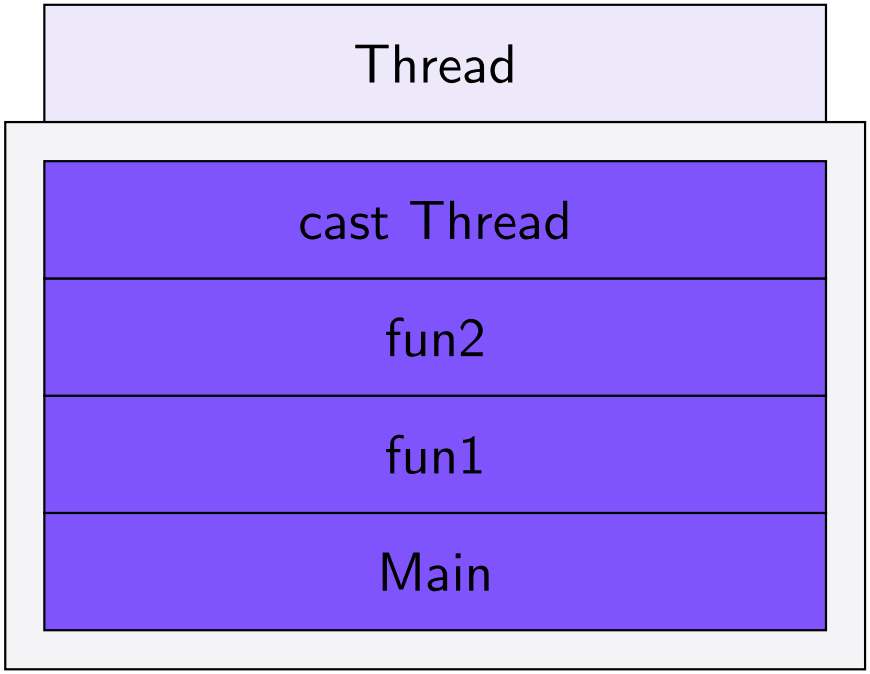
TCB

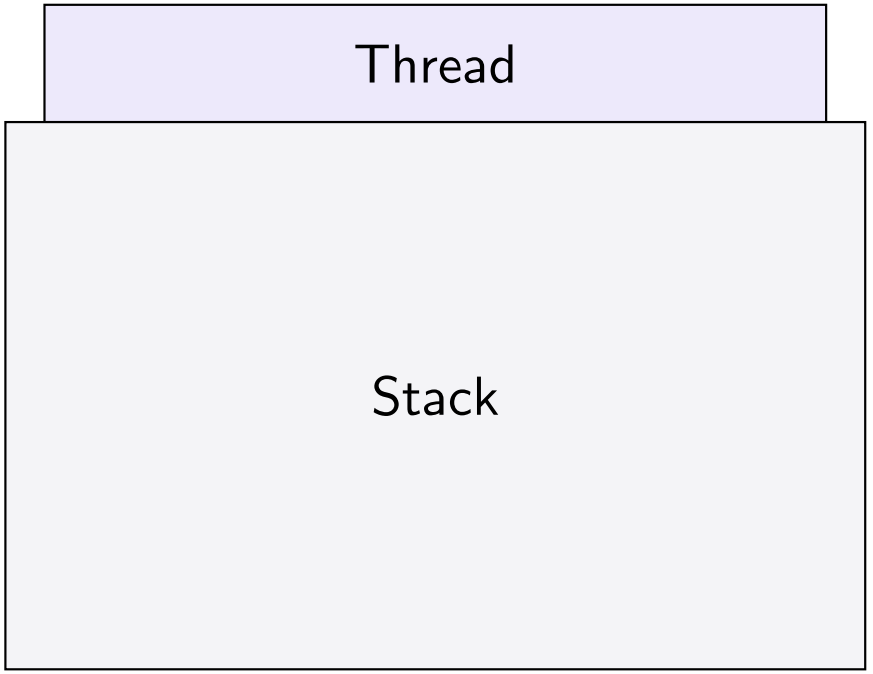
-OS Lastig

Android: Coroutines

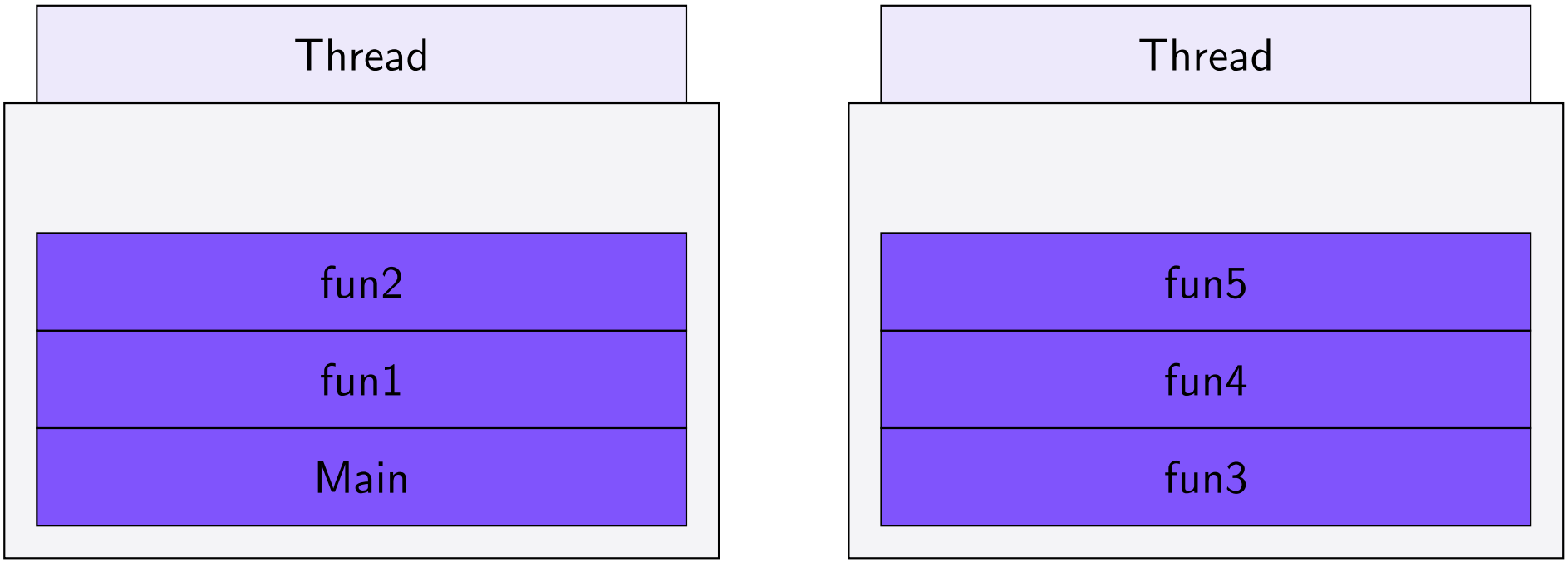


Android: Coroutines

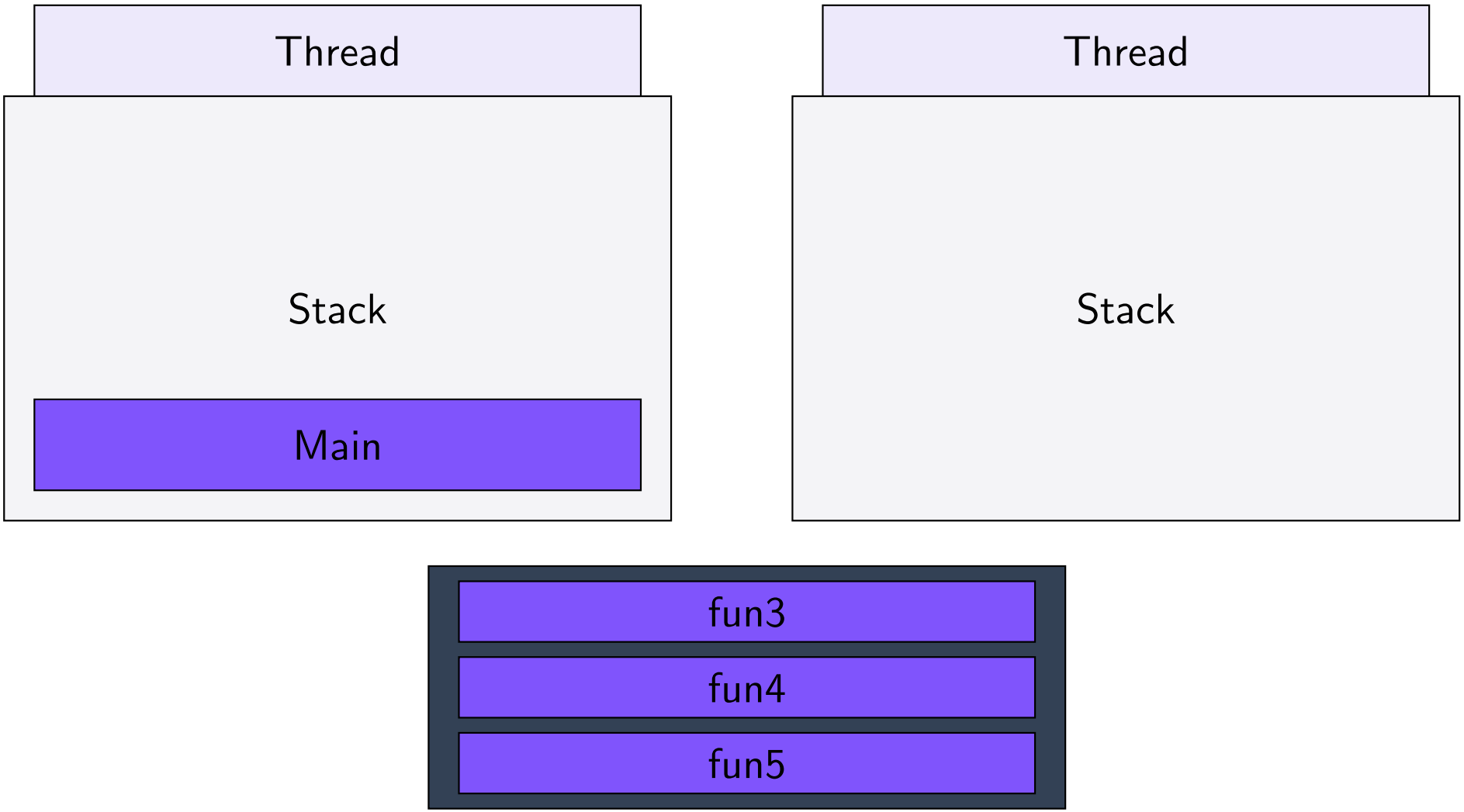




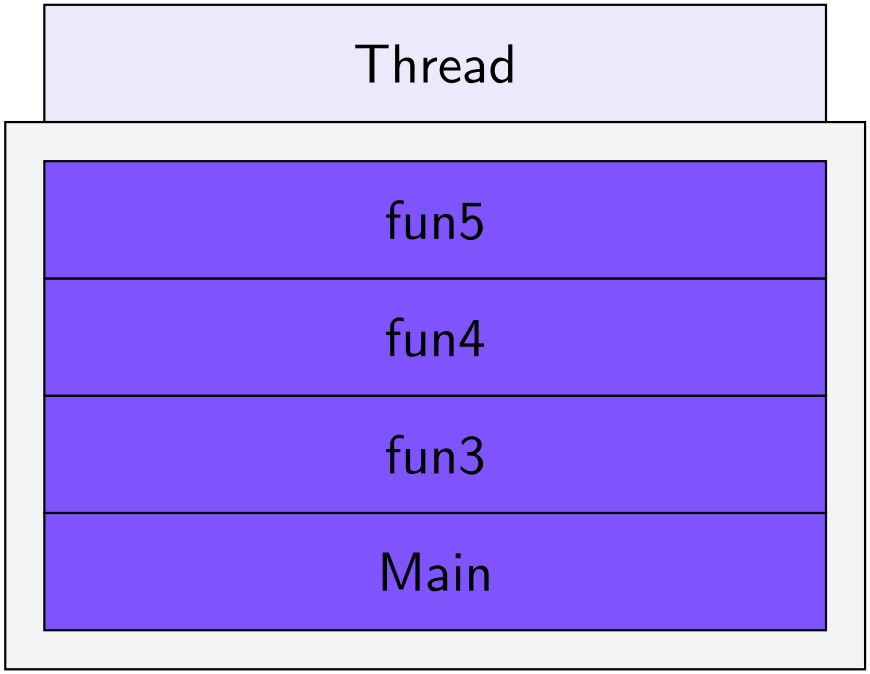
Android: Coroutines

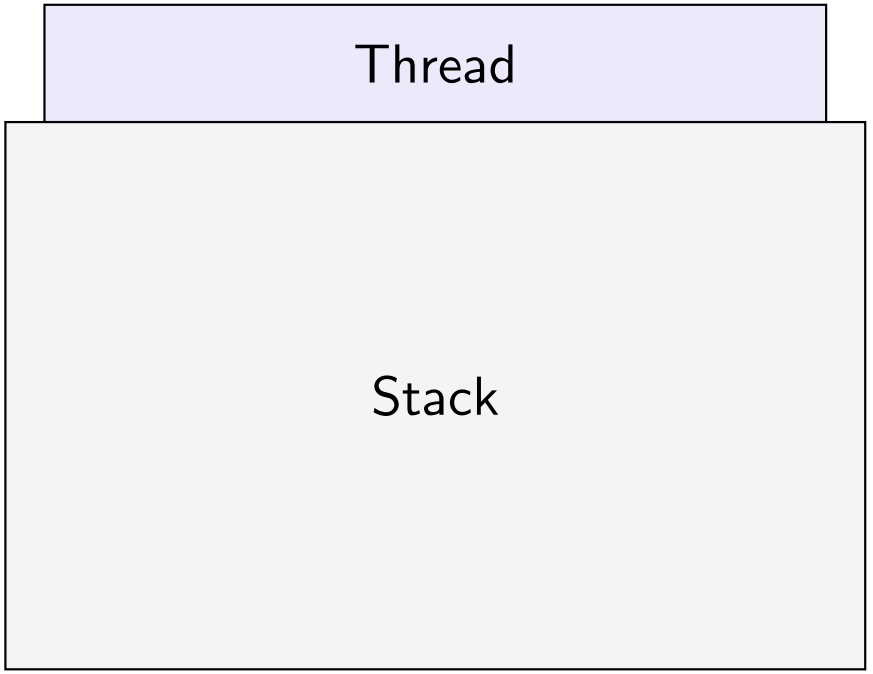


Android: Coroutines

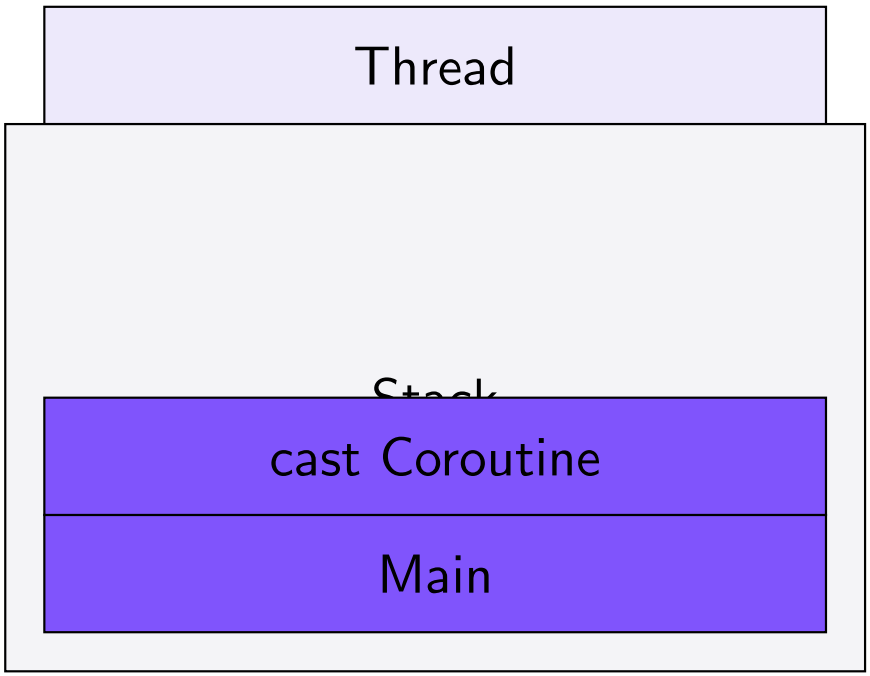


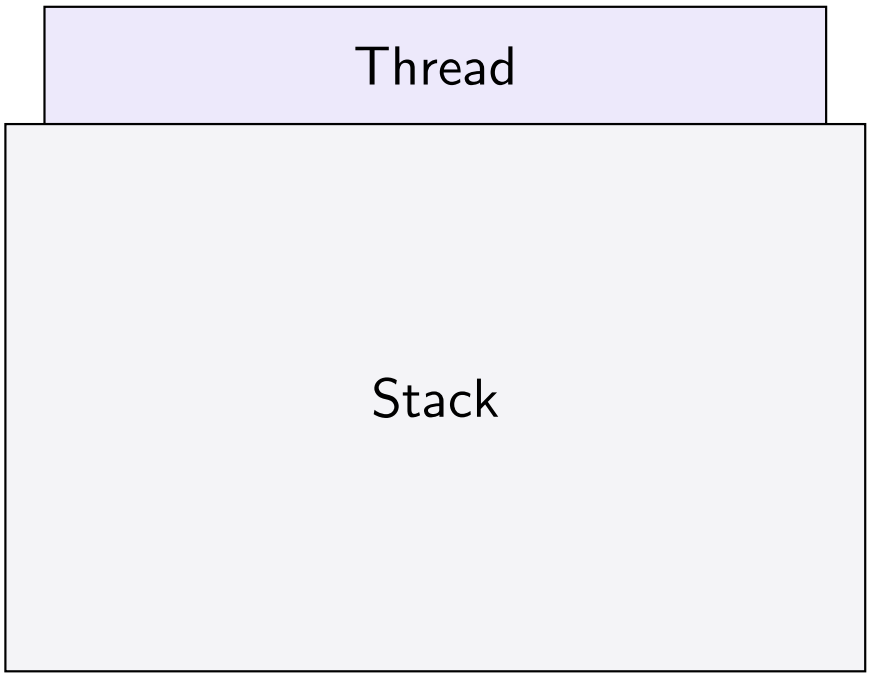
Android: Coroutines



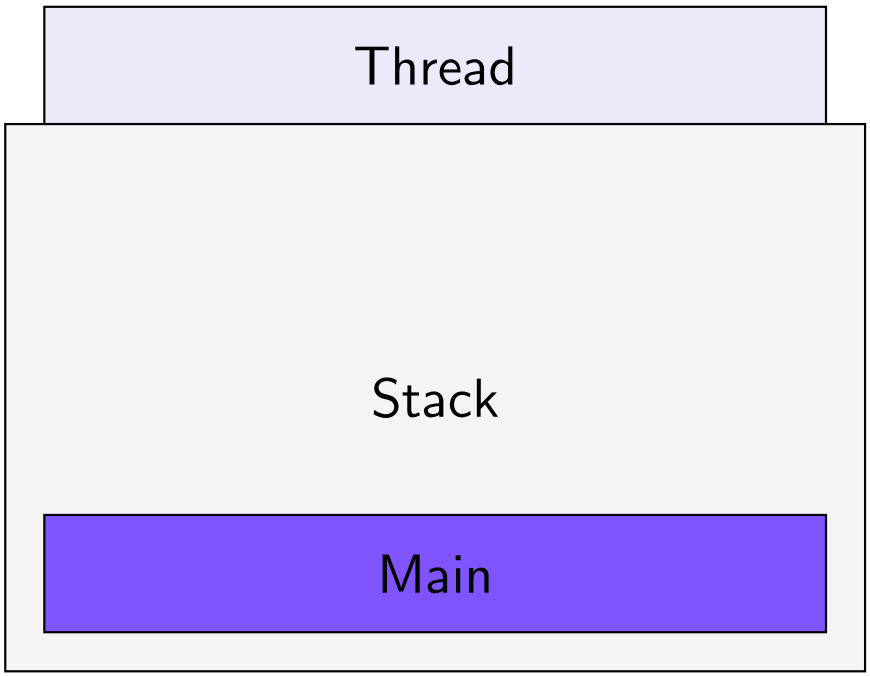


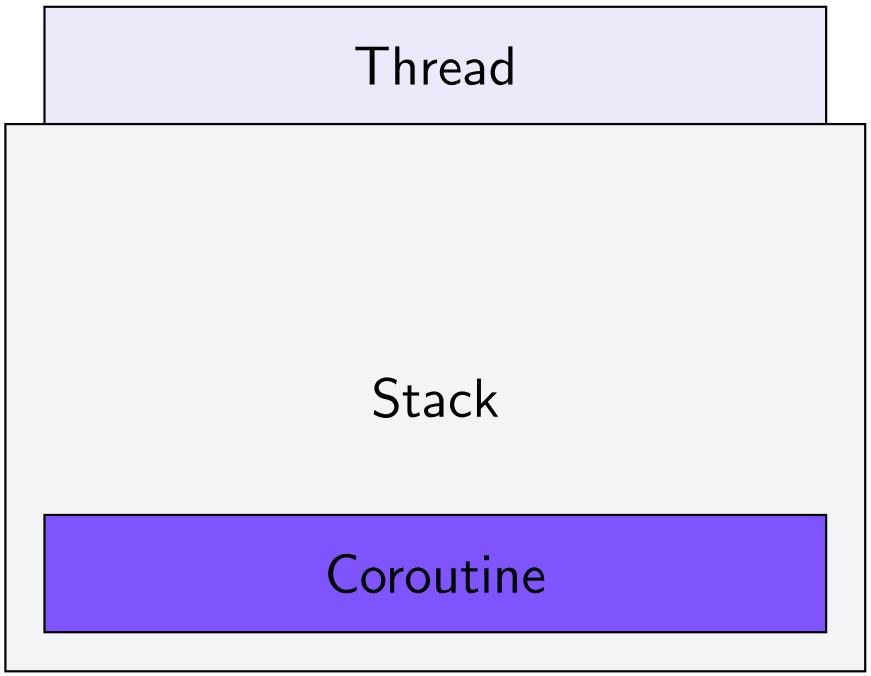
Android: Coroutines



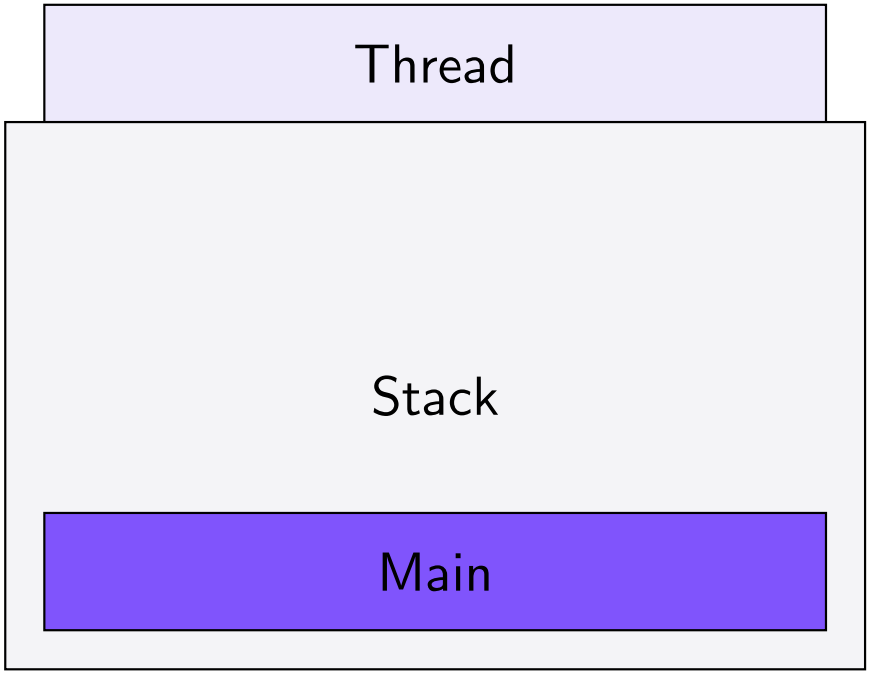


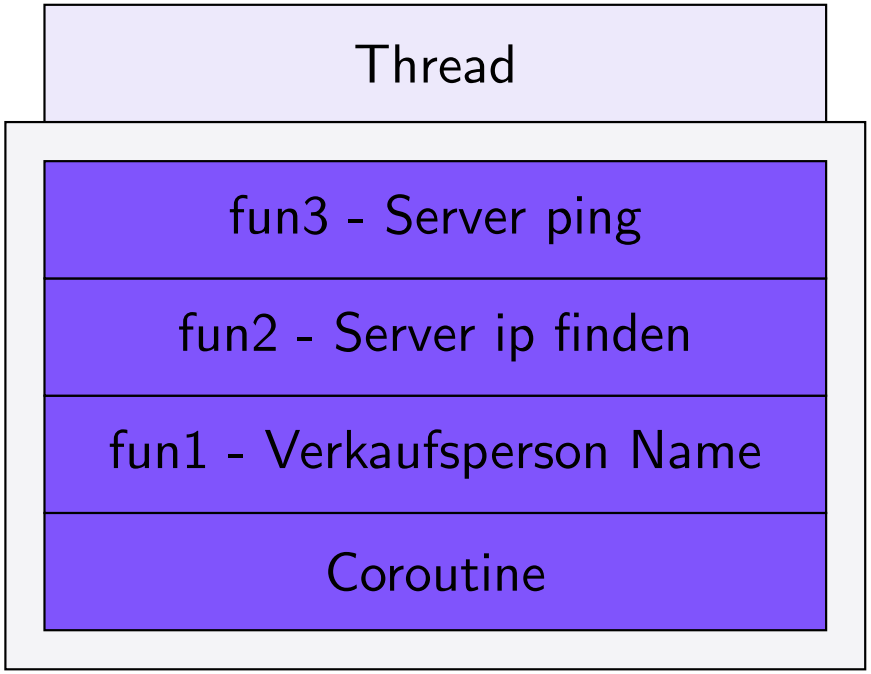
Android: Coroutines





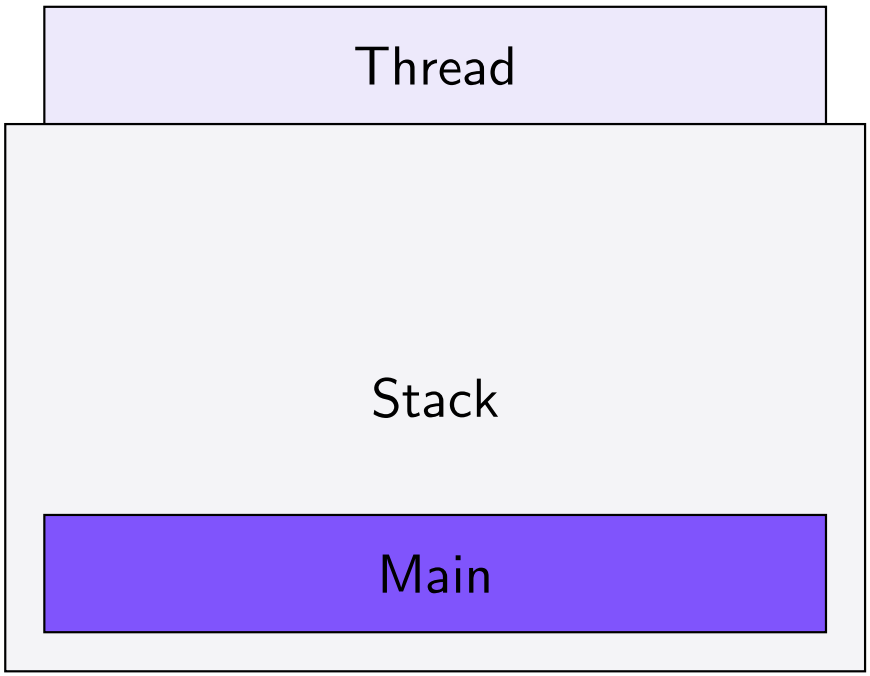
Android: Coroutines

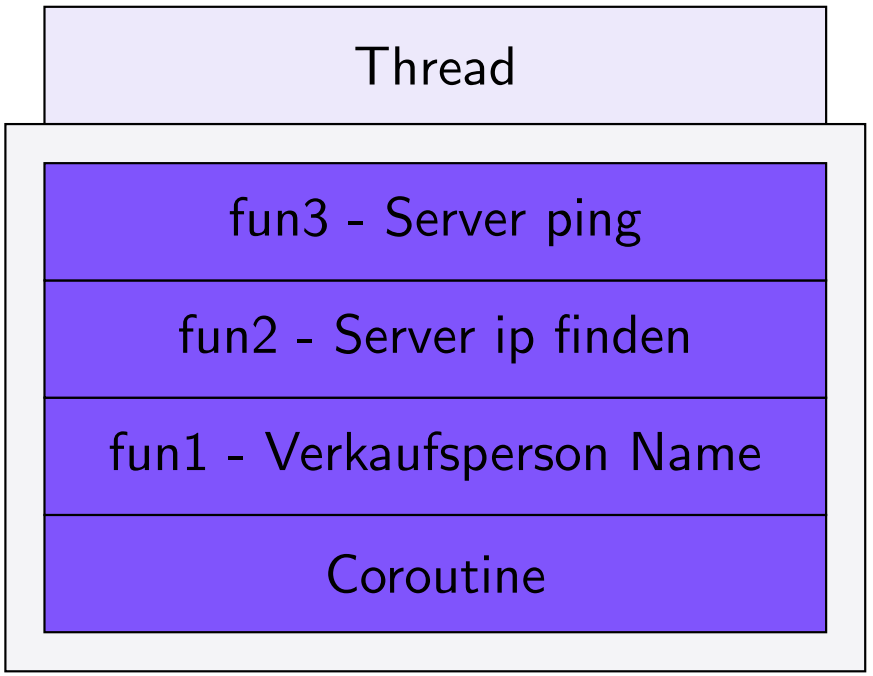




Android: Coroutines

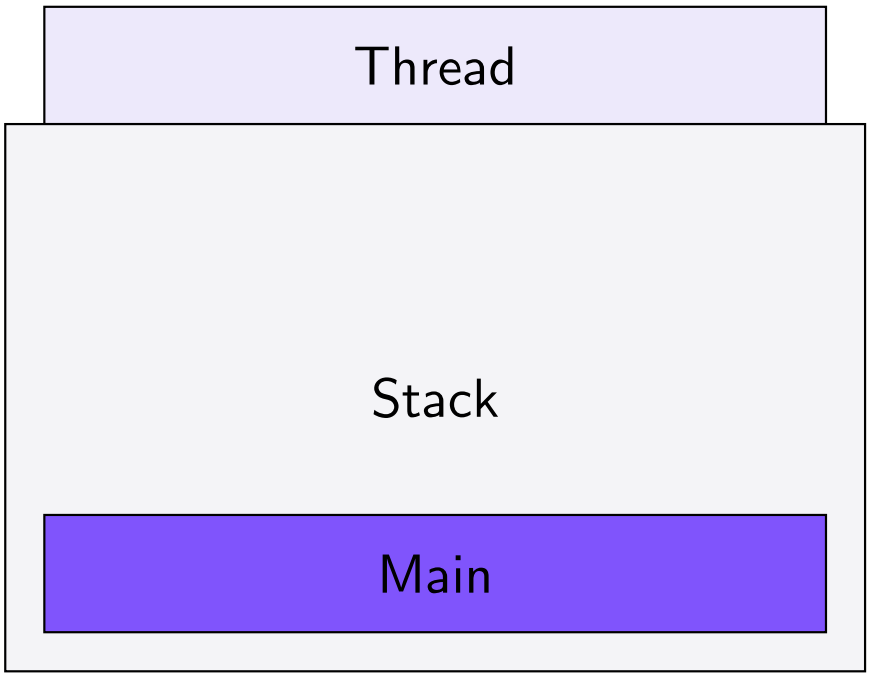
Suspends: delay(),yield(),withContext(),...

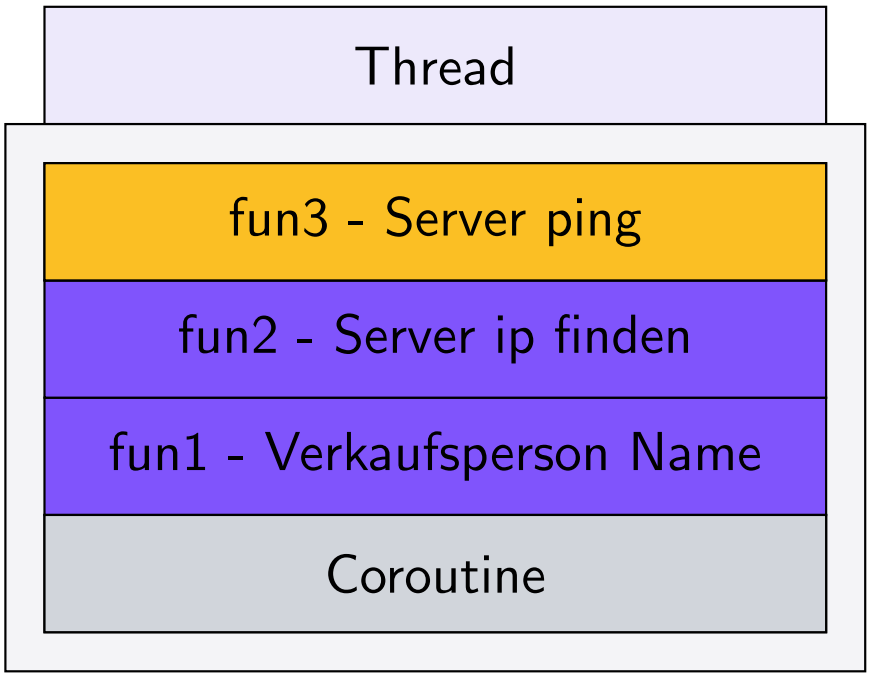




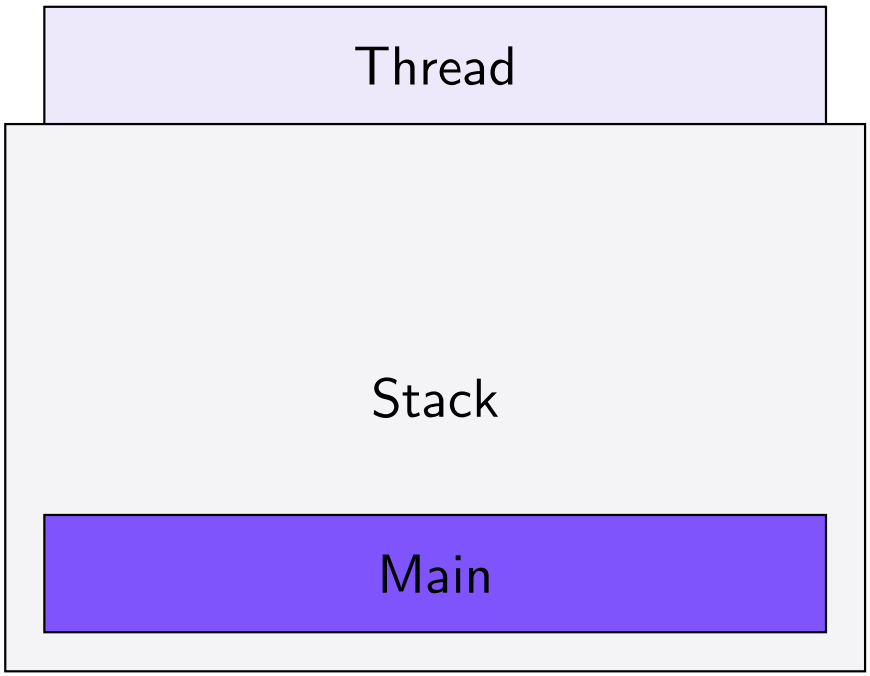
Android: Coroutines

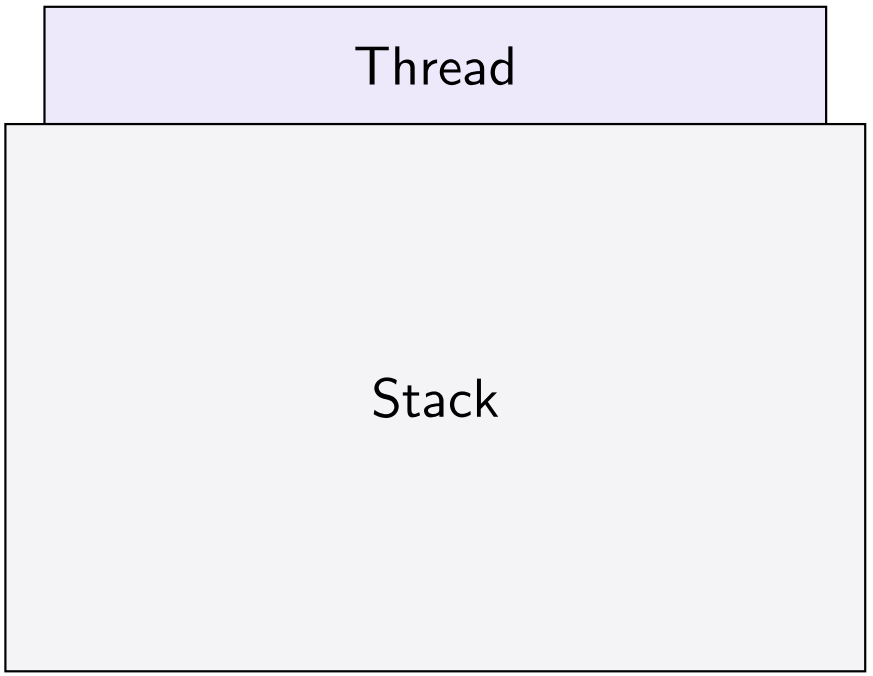
Suspends: delay(),yield(),withContext(),...





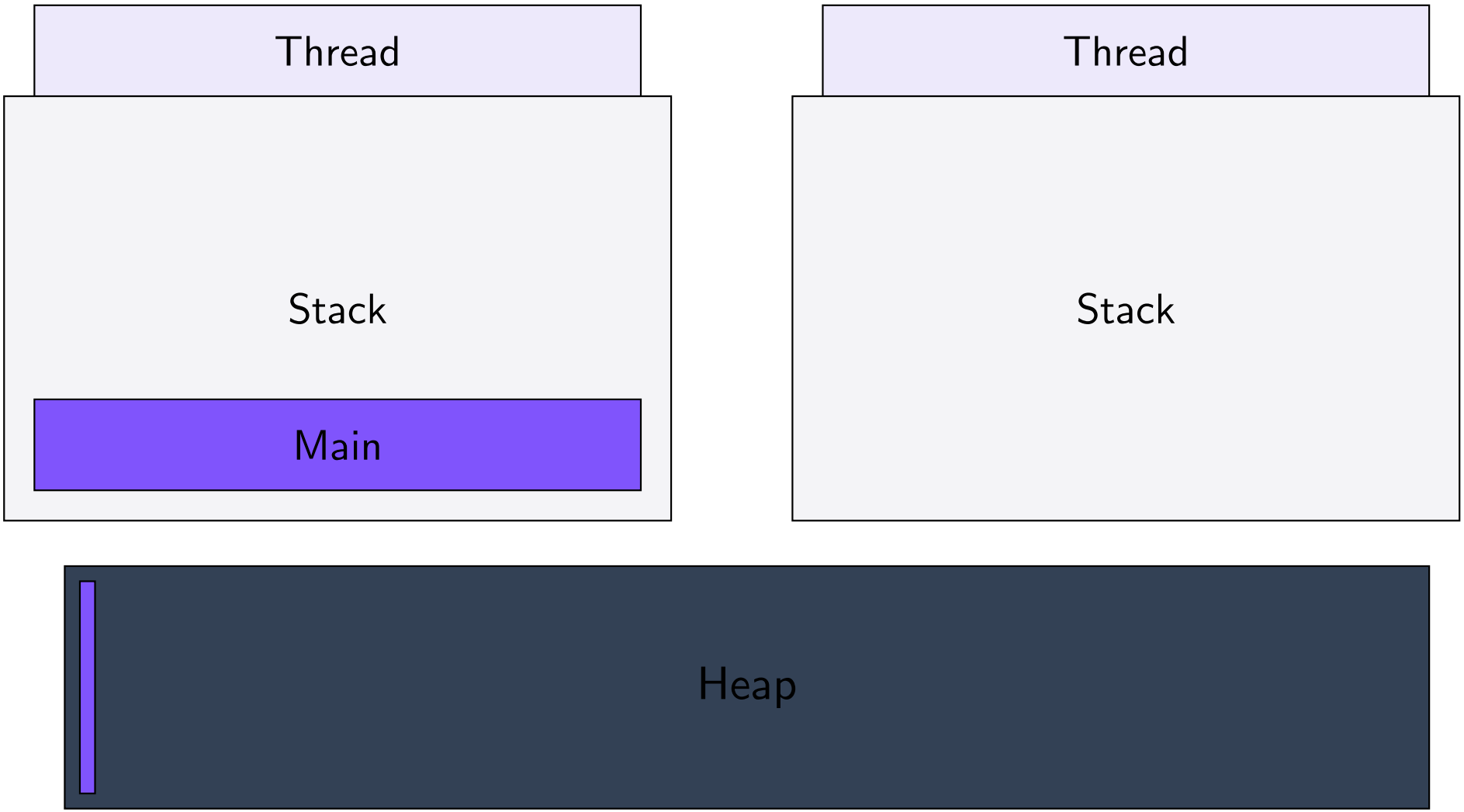
Android: Coroutines



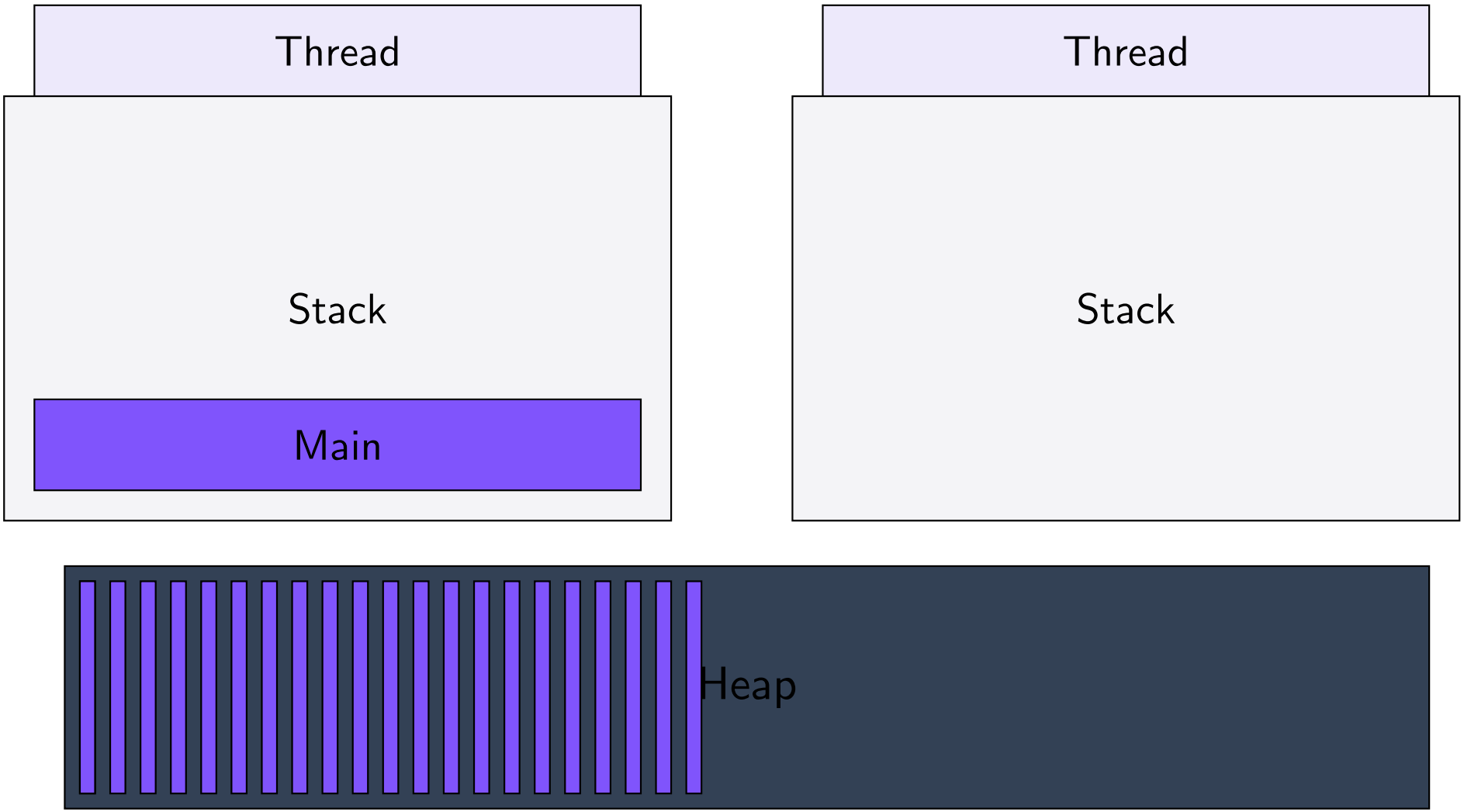



```
Coroutine:
 Variablen:        Name,Ip
   State:             1
    Path:     fun1-fun2-fun3
```

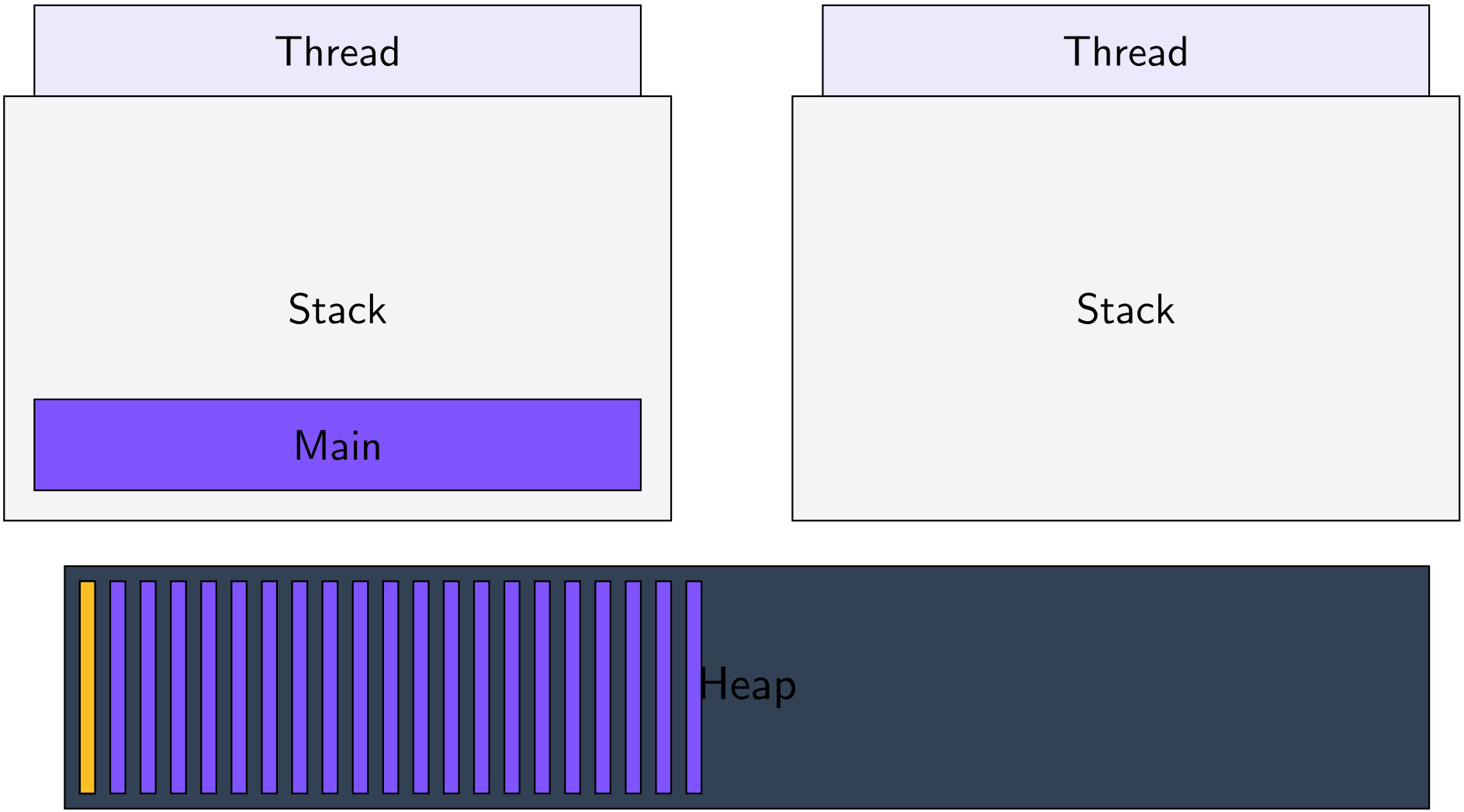
Android: Coroutines



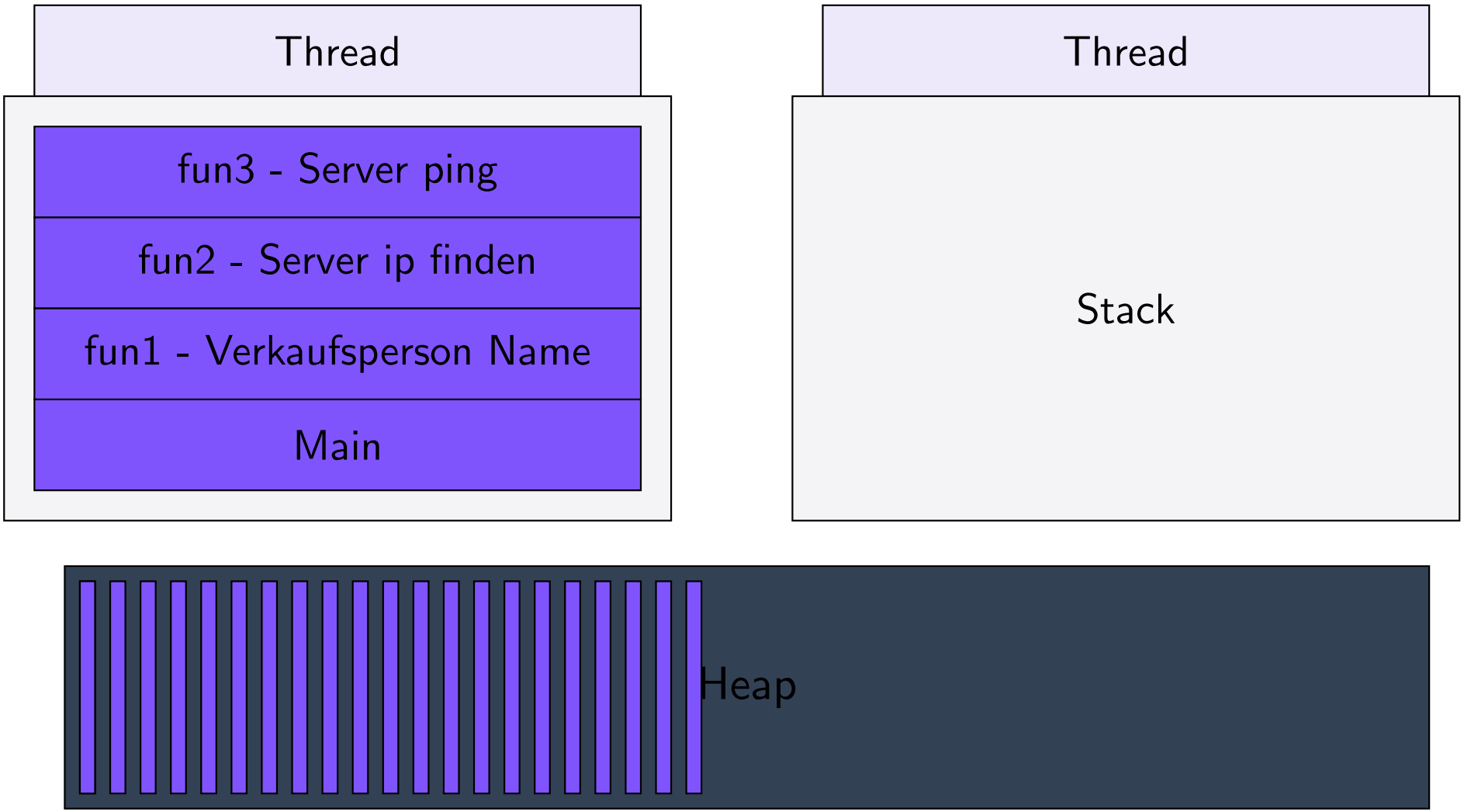
Android: Coroutines



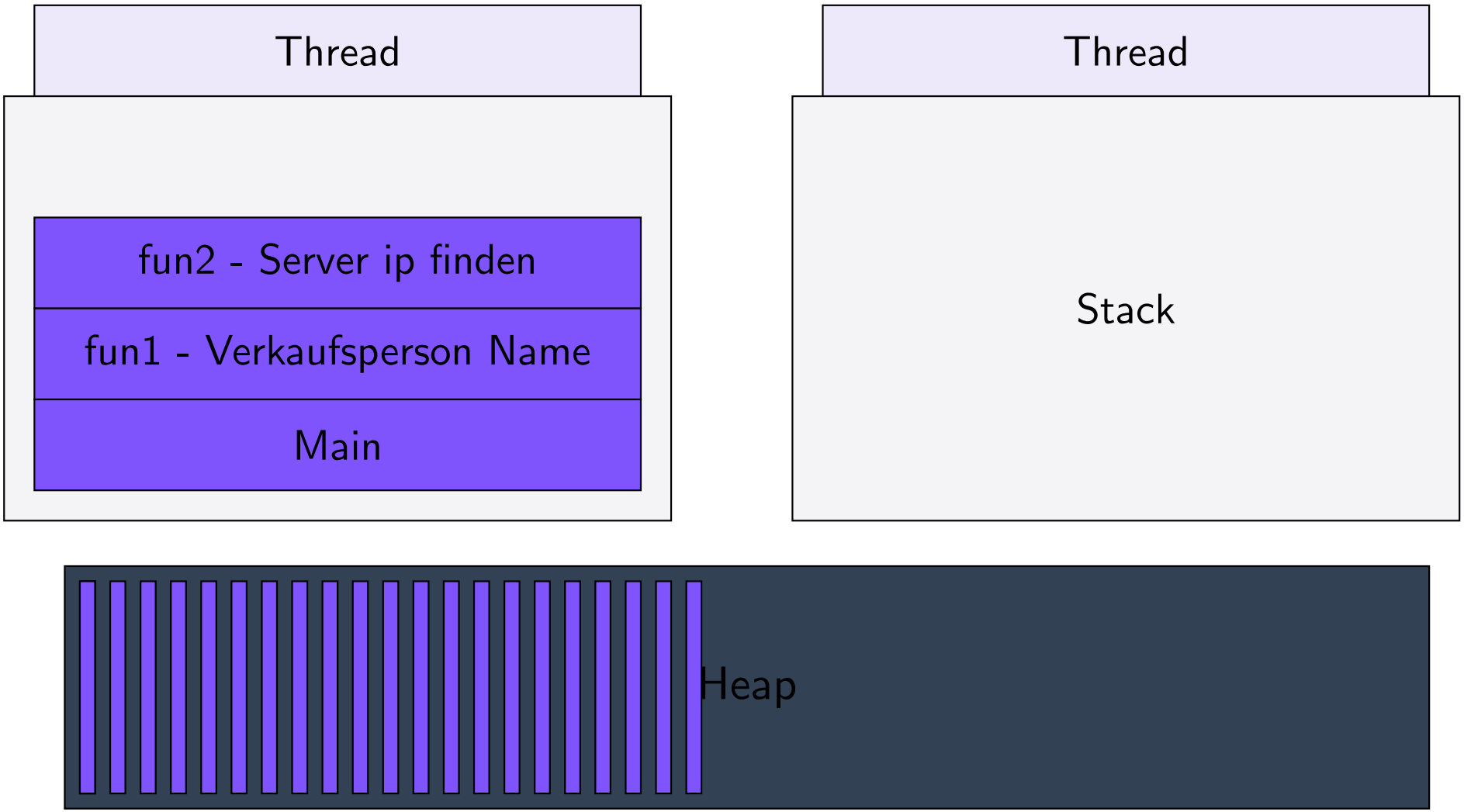
Android: Coroutines



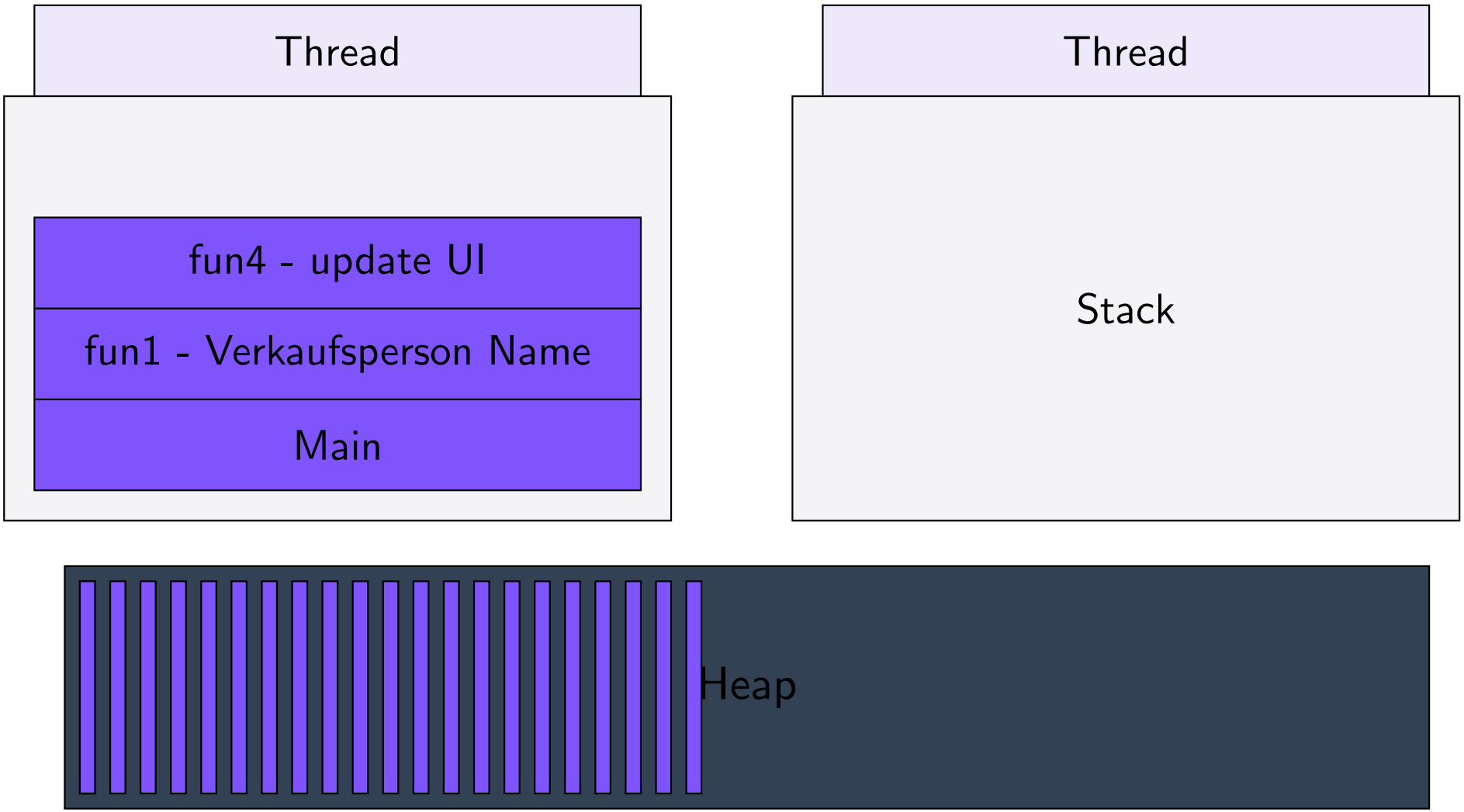
Android: Coroutines



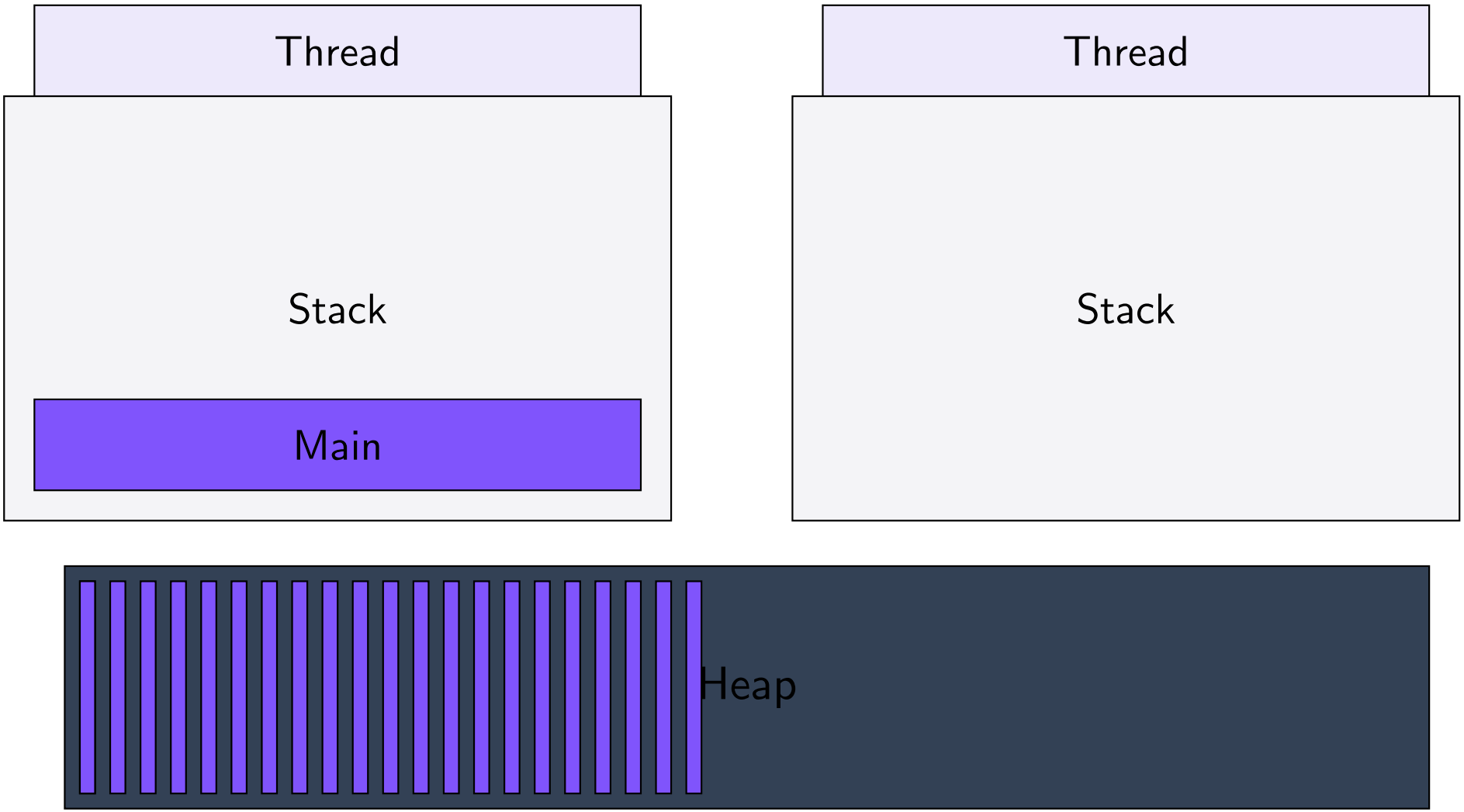
Android: Coroutines



Android: Coroutines



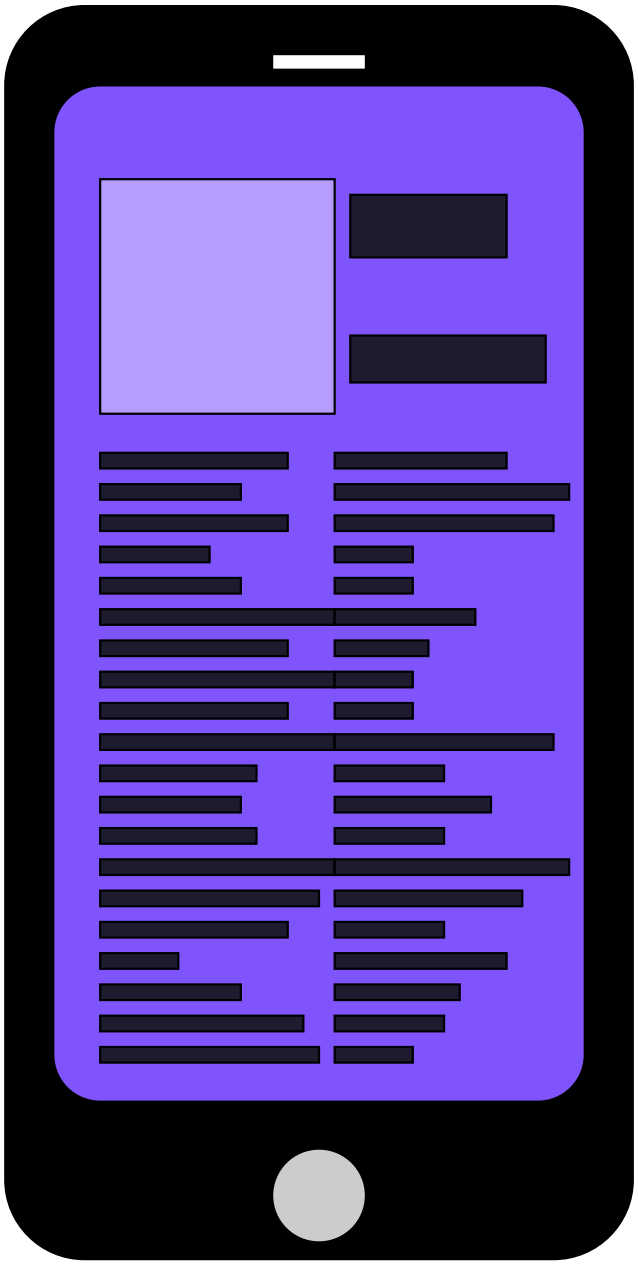
Android: Coroutines



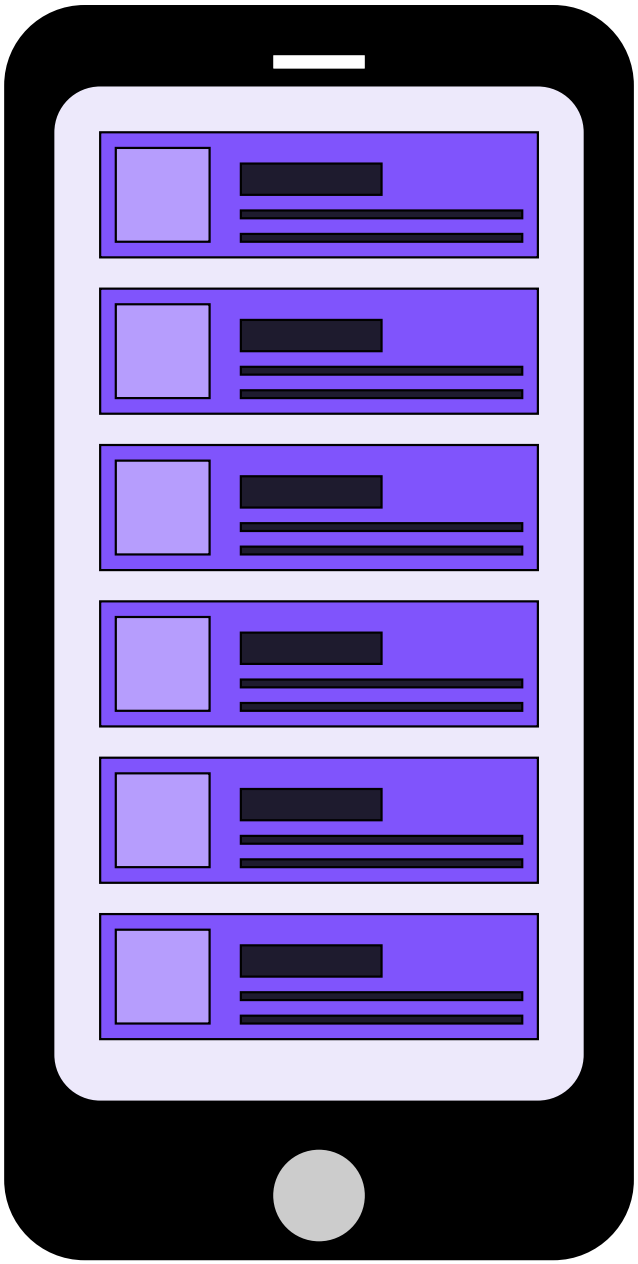
Android: Coroutines: Beispiel

Multithreading mit Coroutines
ein Thread
Multithreading
Multithreading mit Coroutines

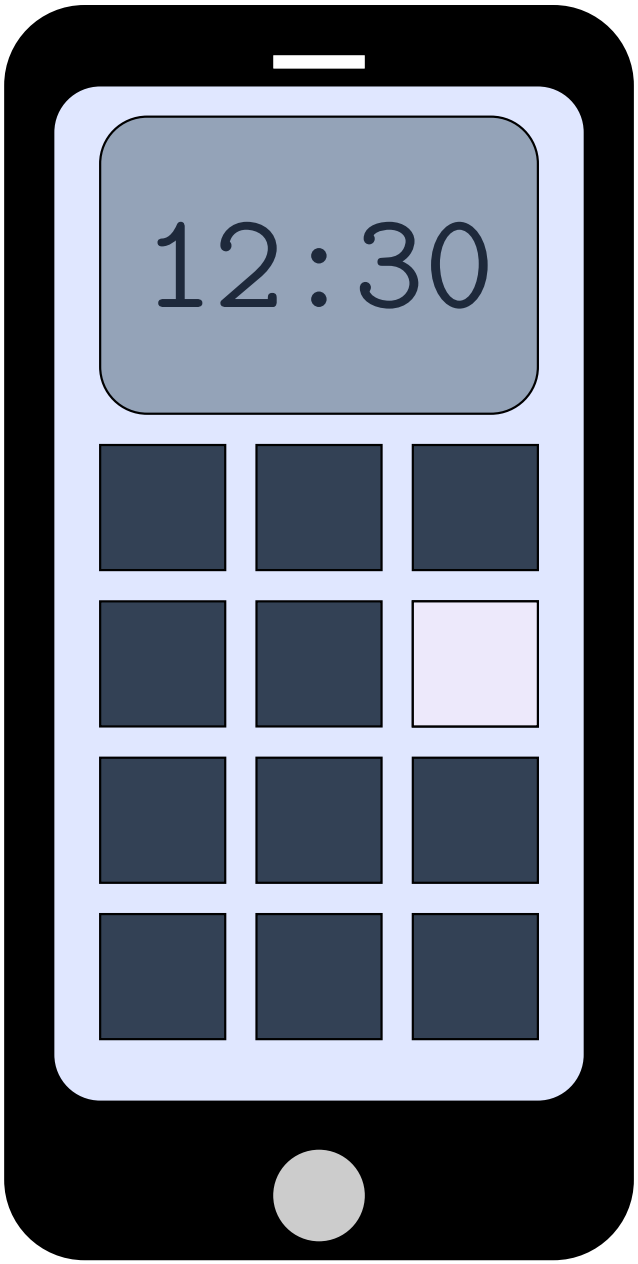
Android: Coroutines: Beispiel



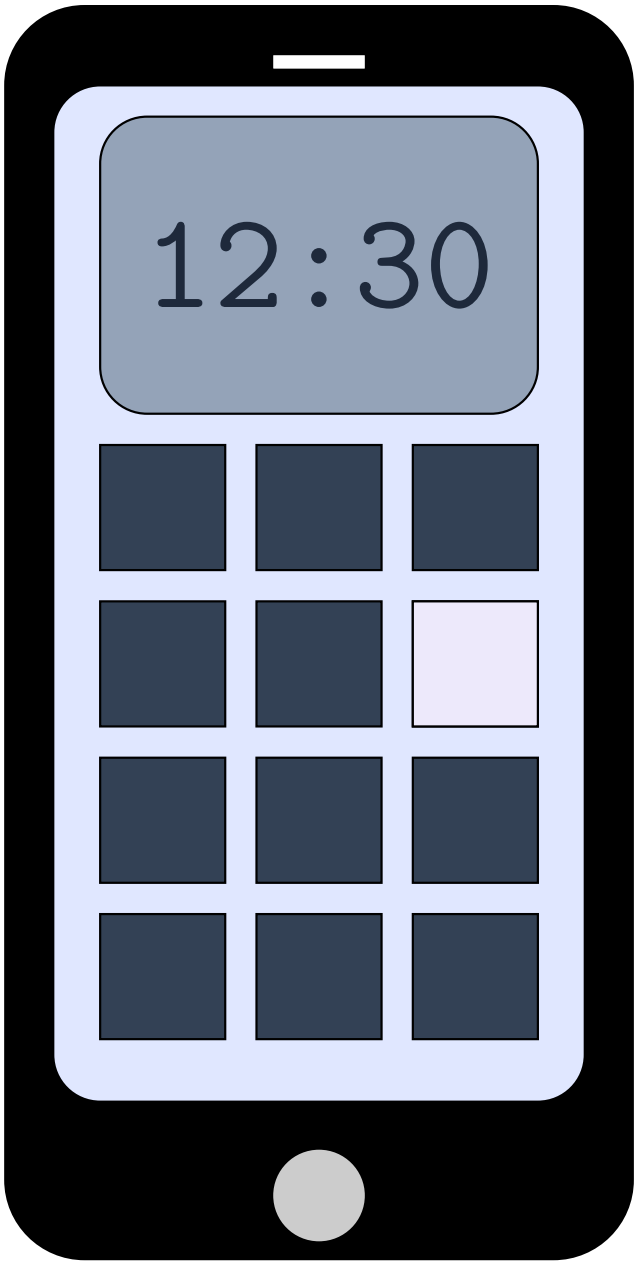
Android: Coroutines: Beispiel

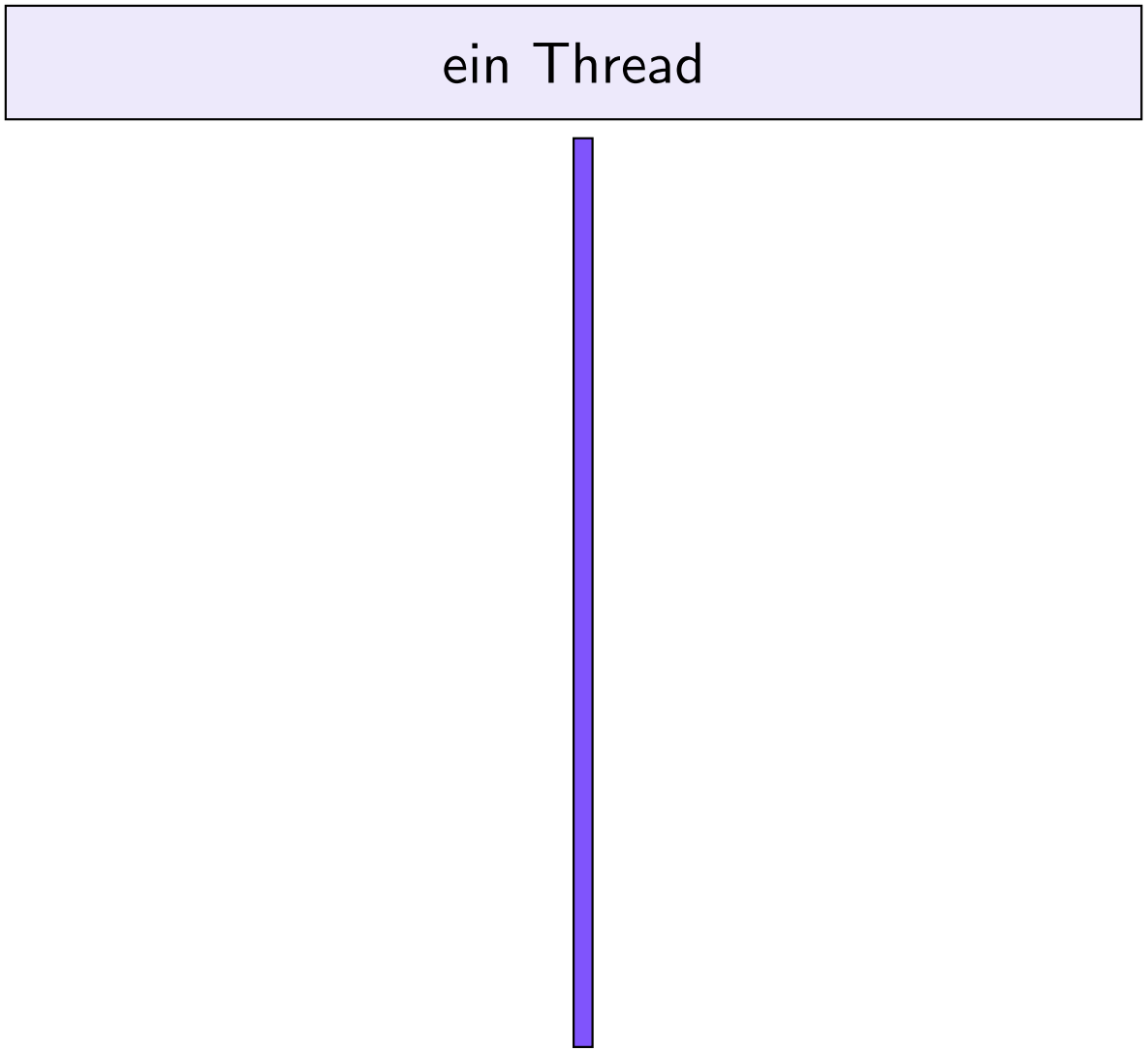


Android: Coroutines: Beispiel

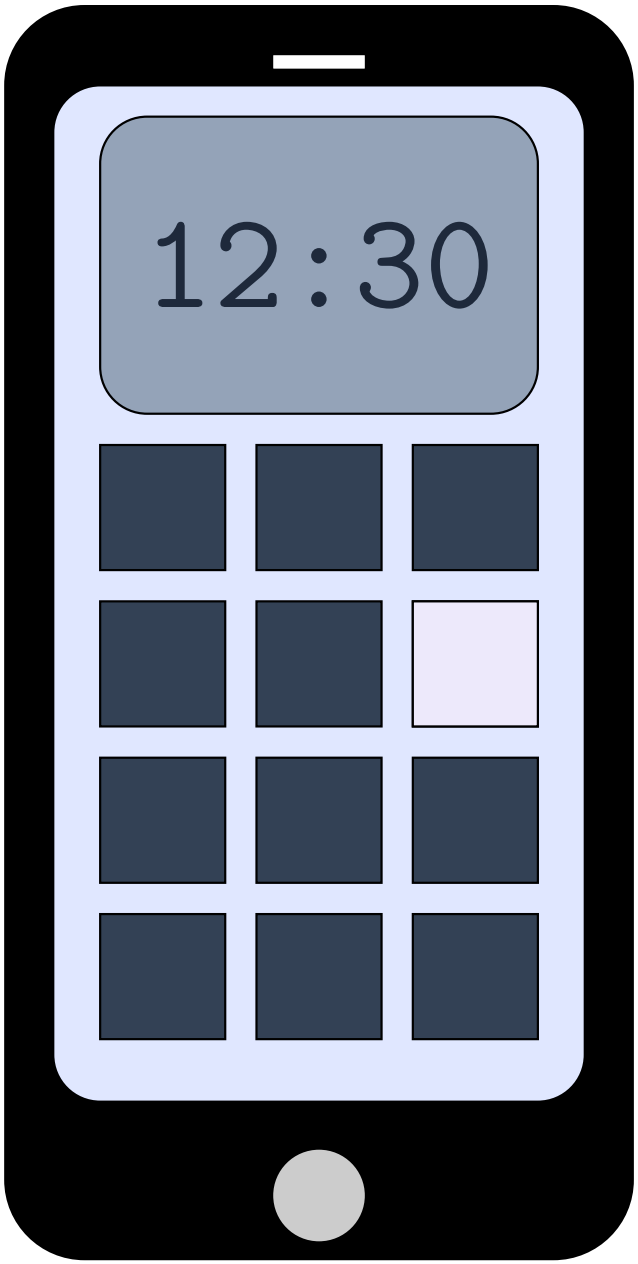


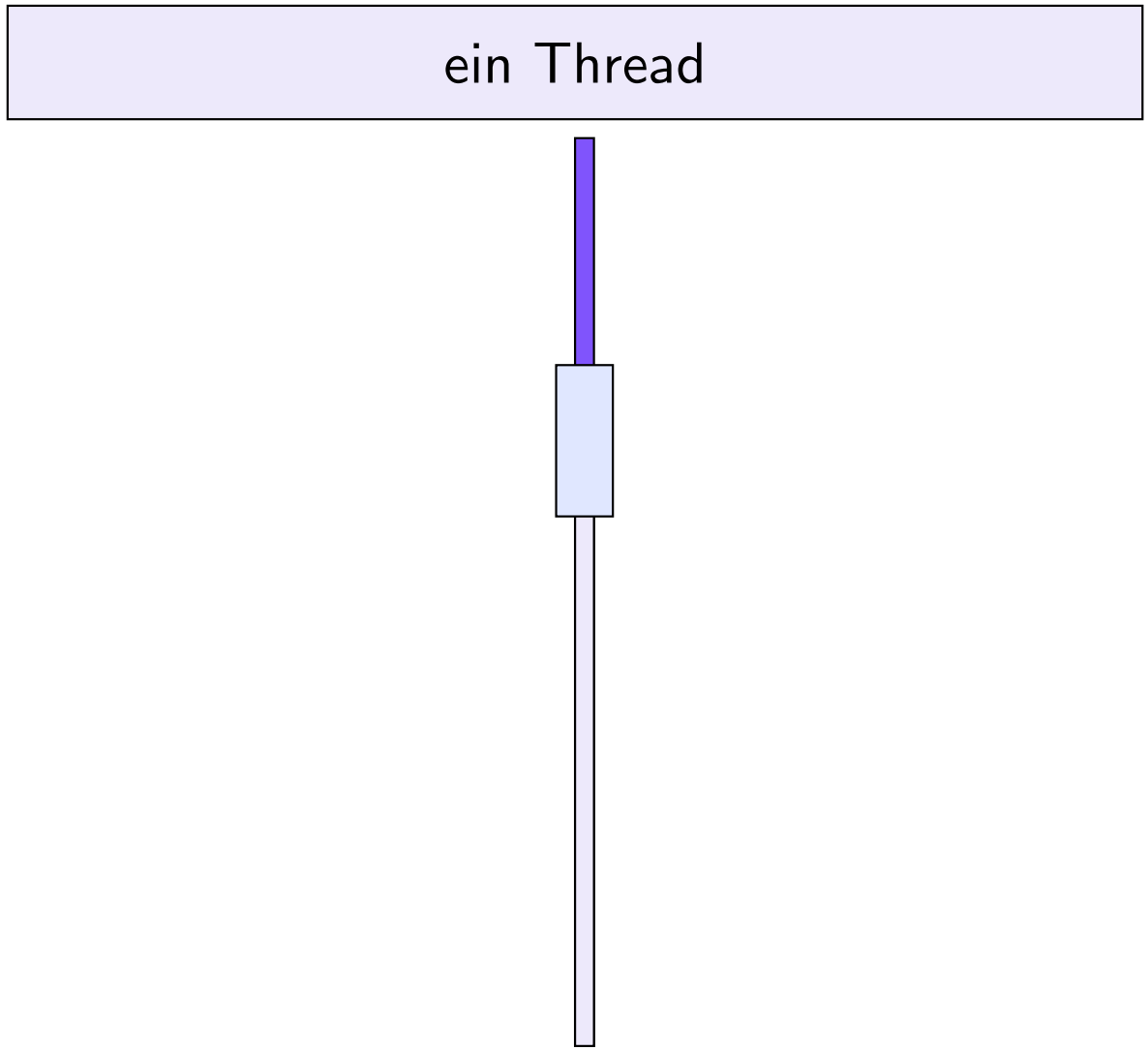
Android: Coroutines: Beispiel



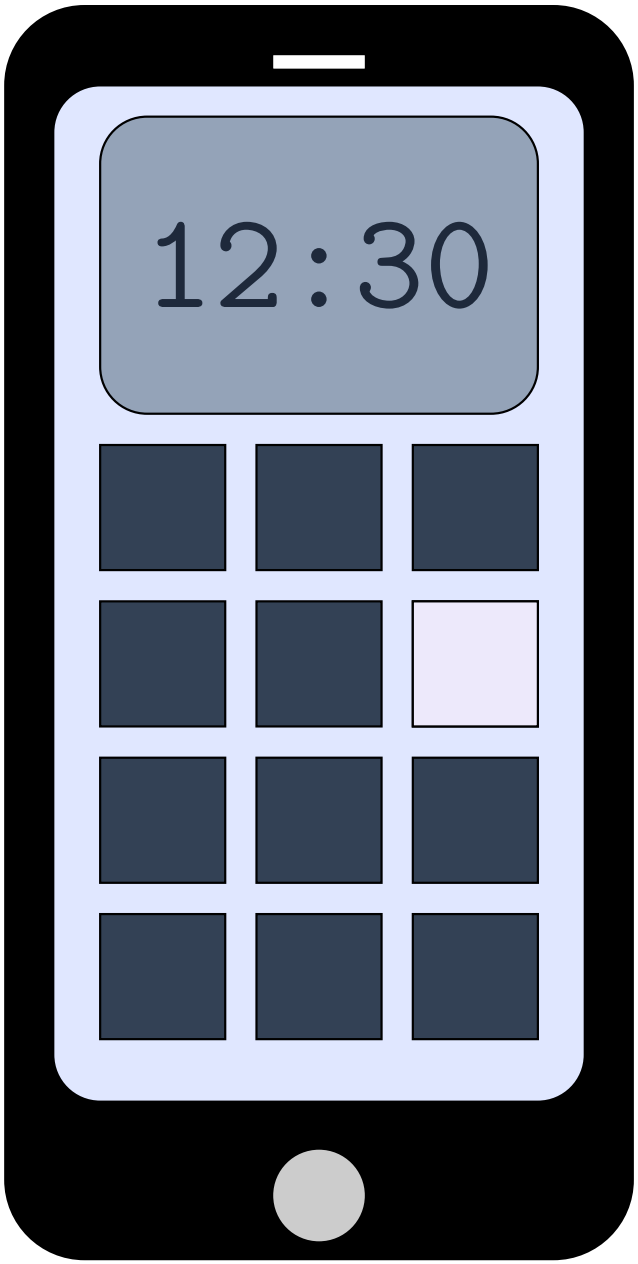


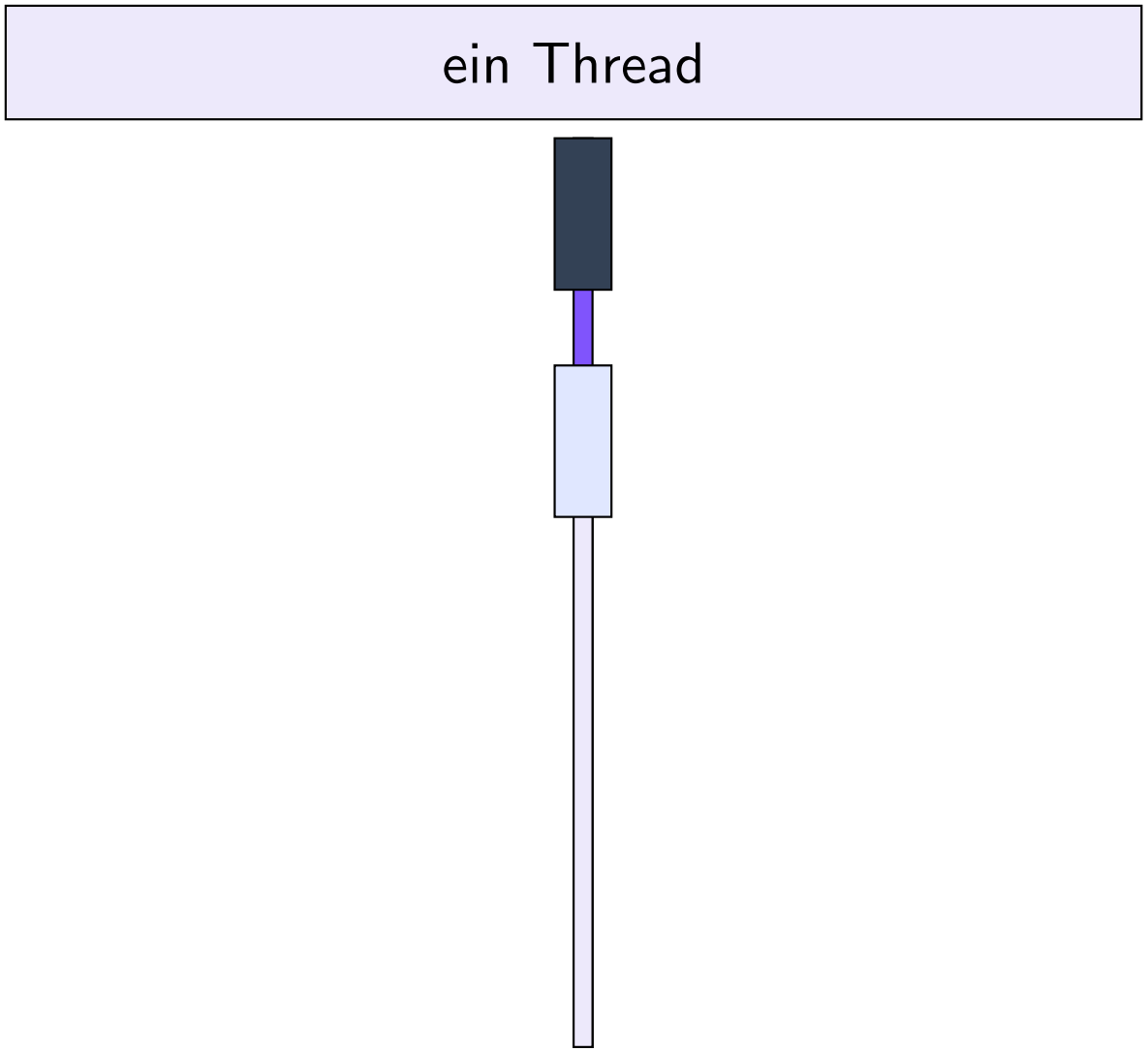
Android: Coroutines: Beispiel



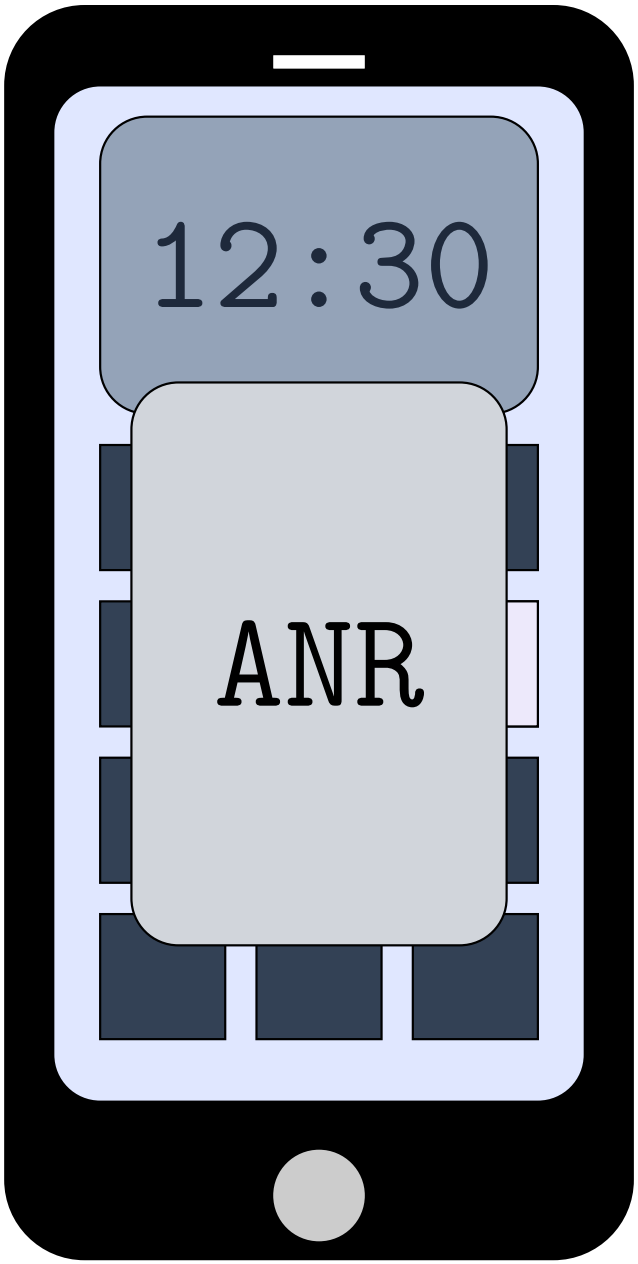


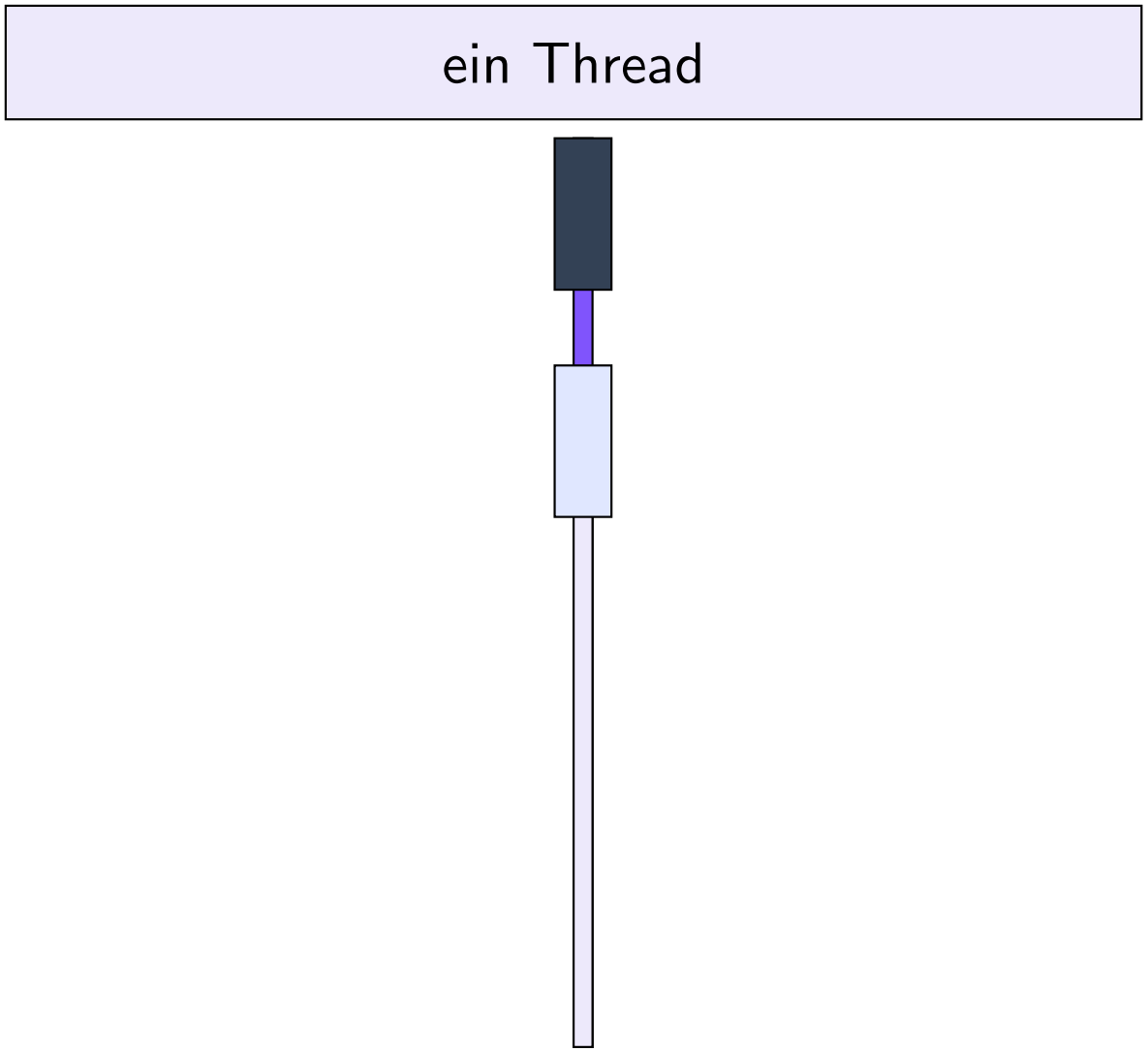
Android: Coroutines: Beispiel



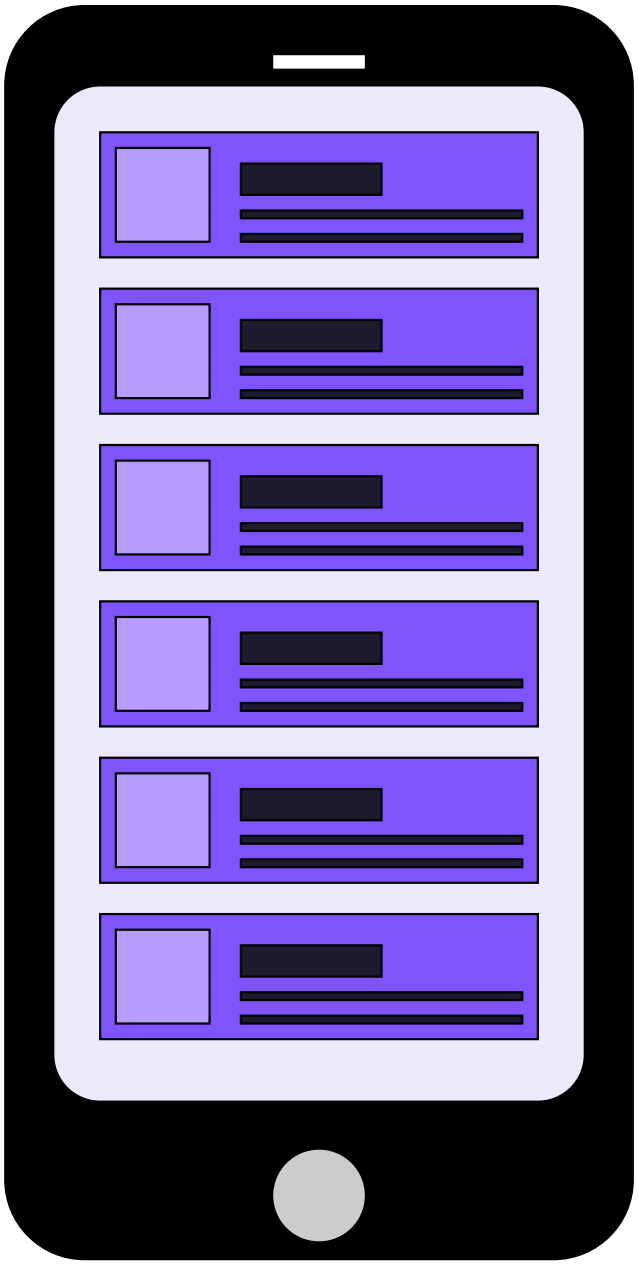


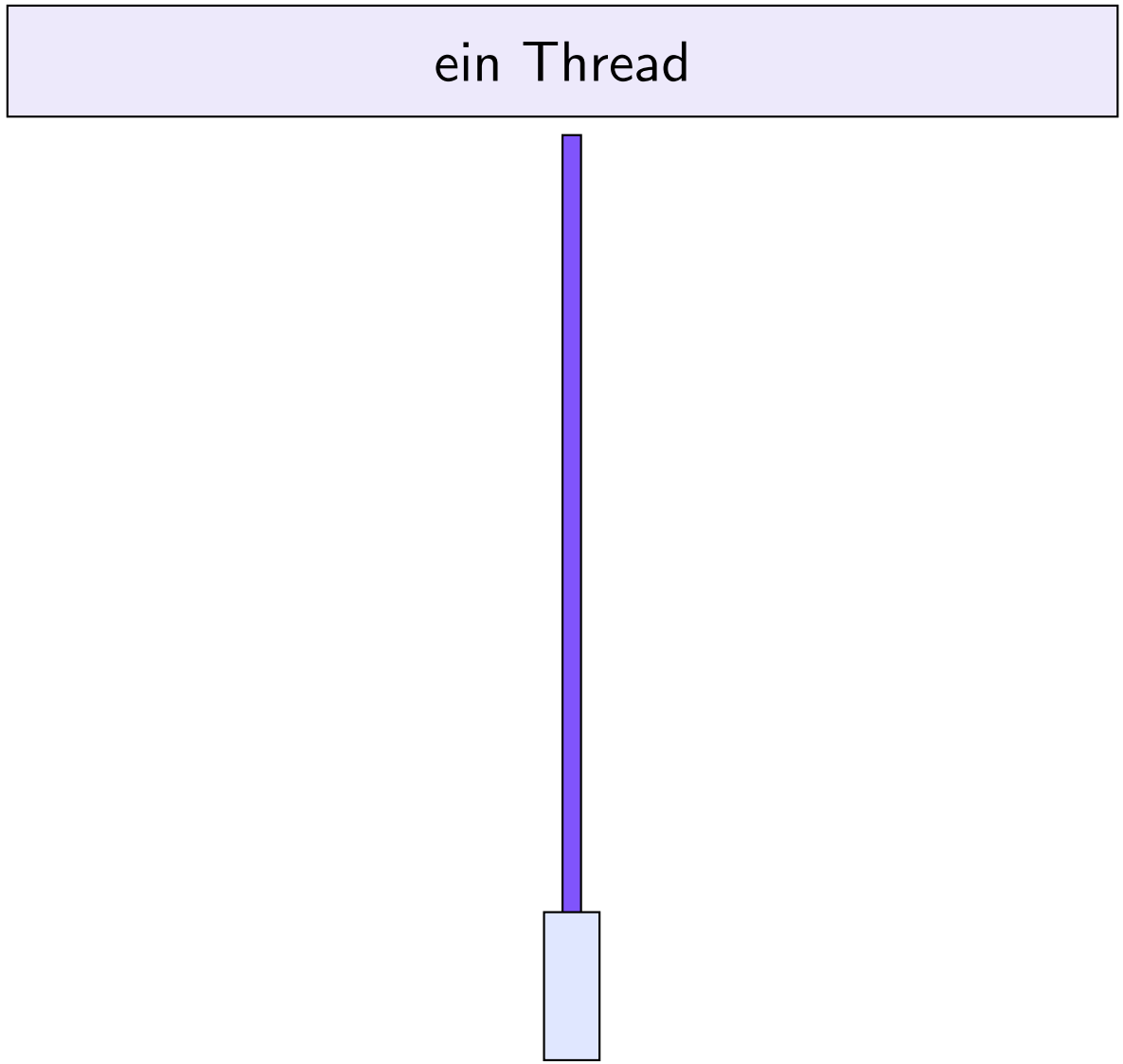
Android: Coroutines: Beispiel



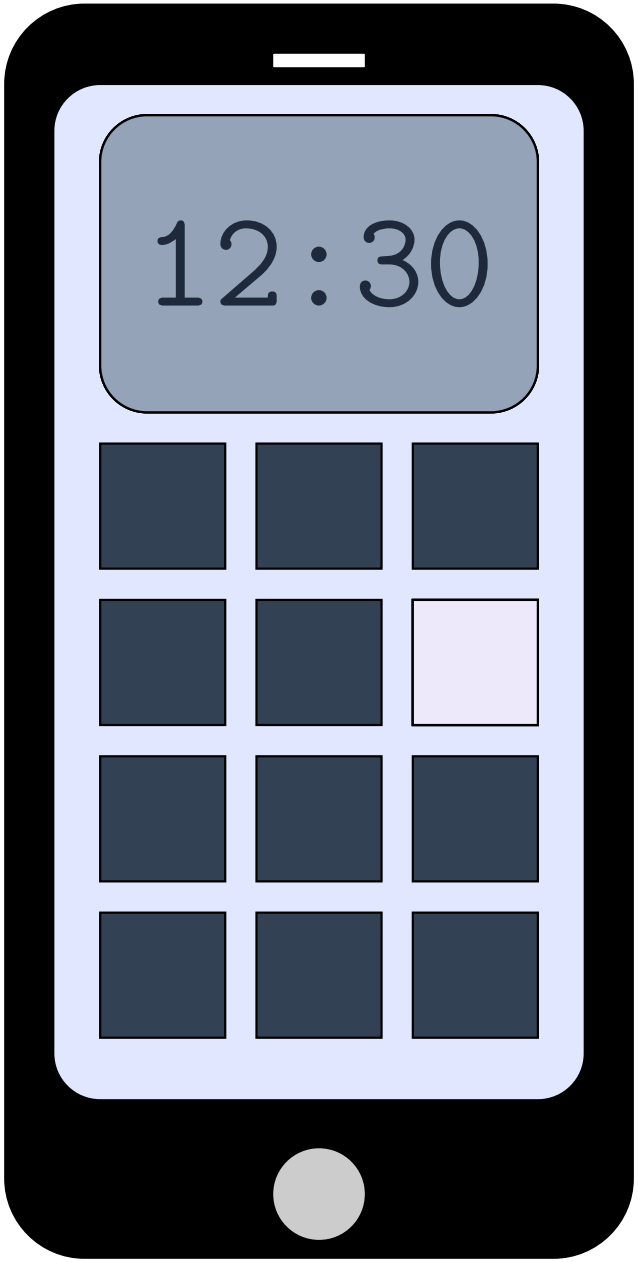


Android: Coroutines: Beispiel



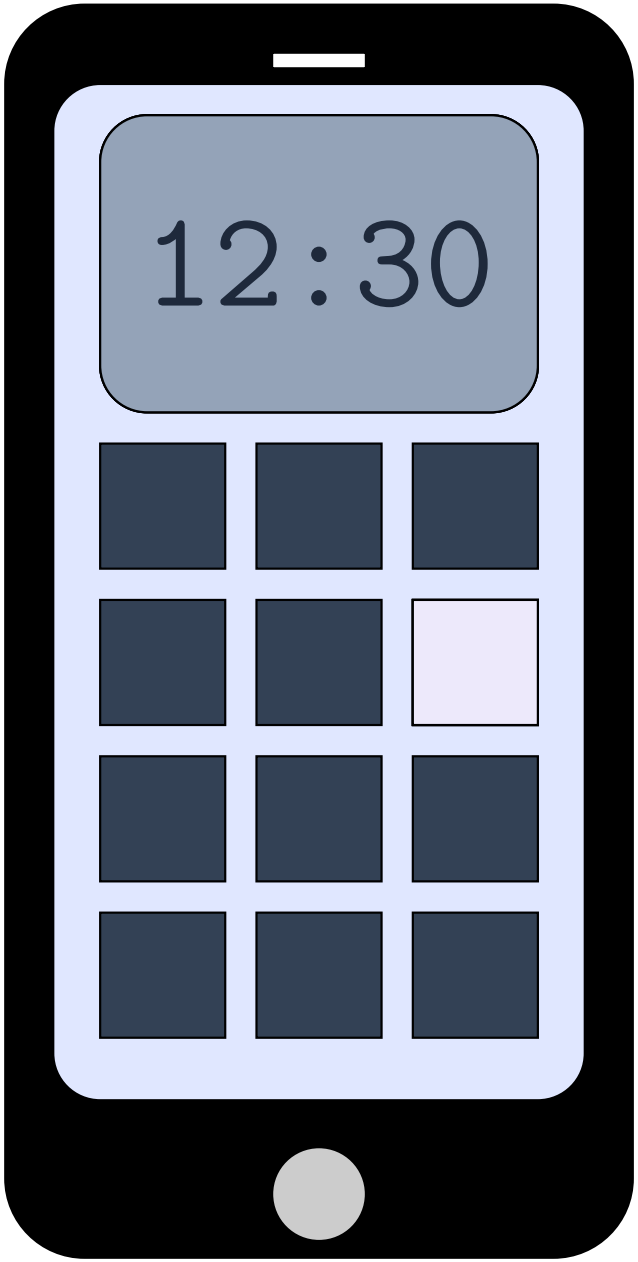


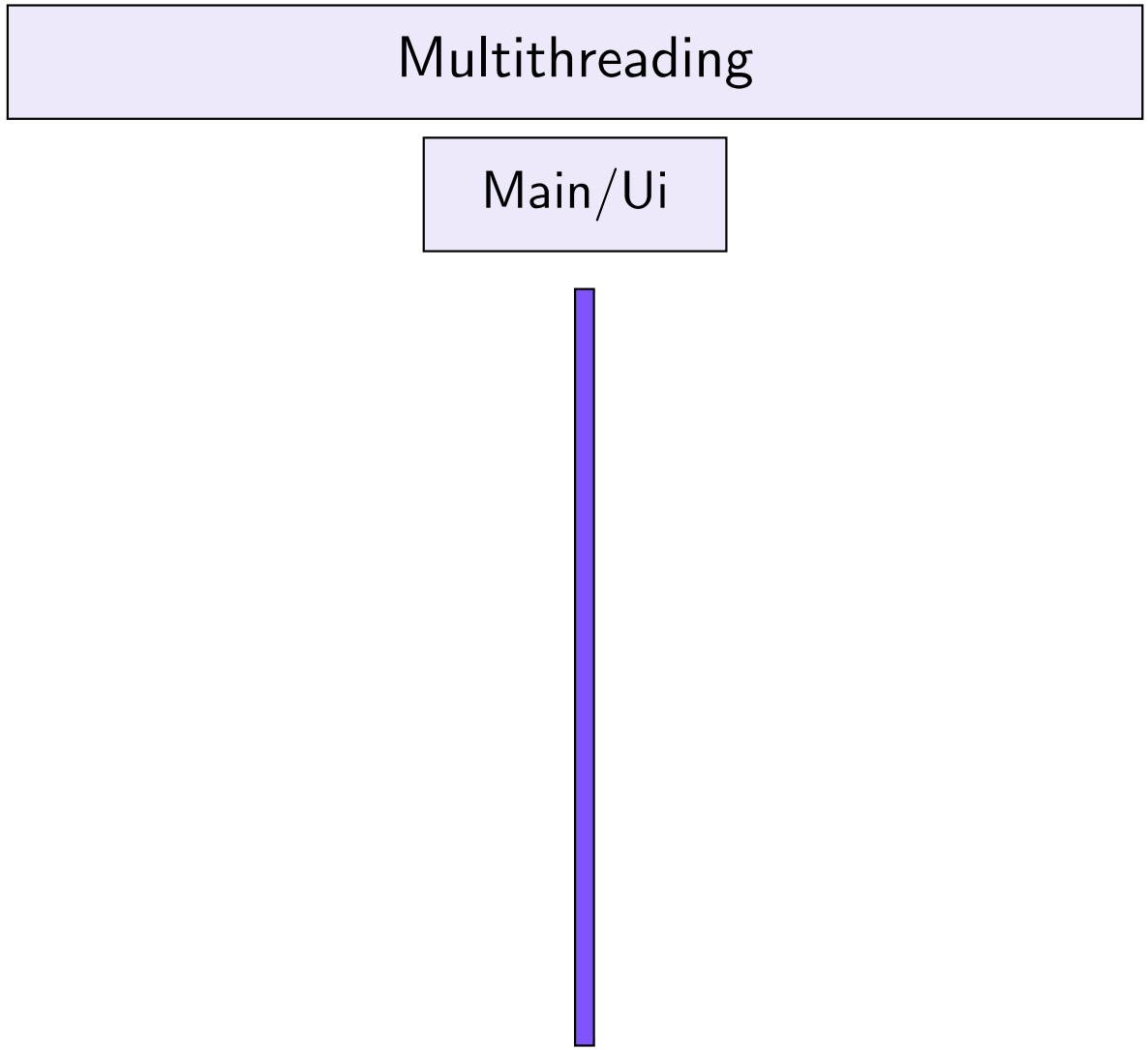
Android: Coroutines: Beispiel



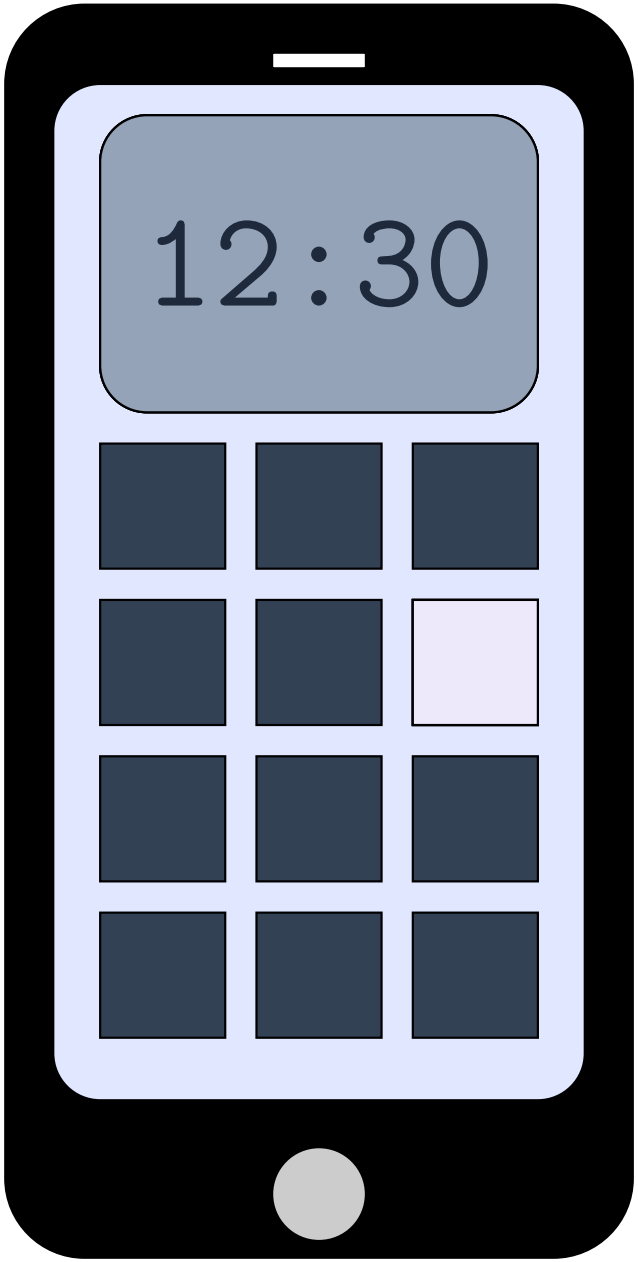
Multithreading

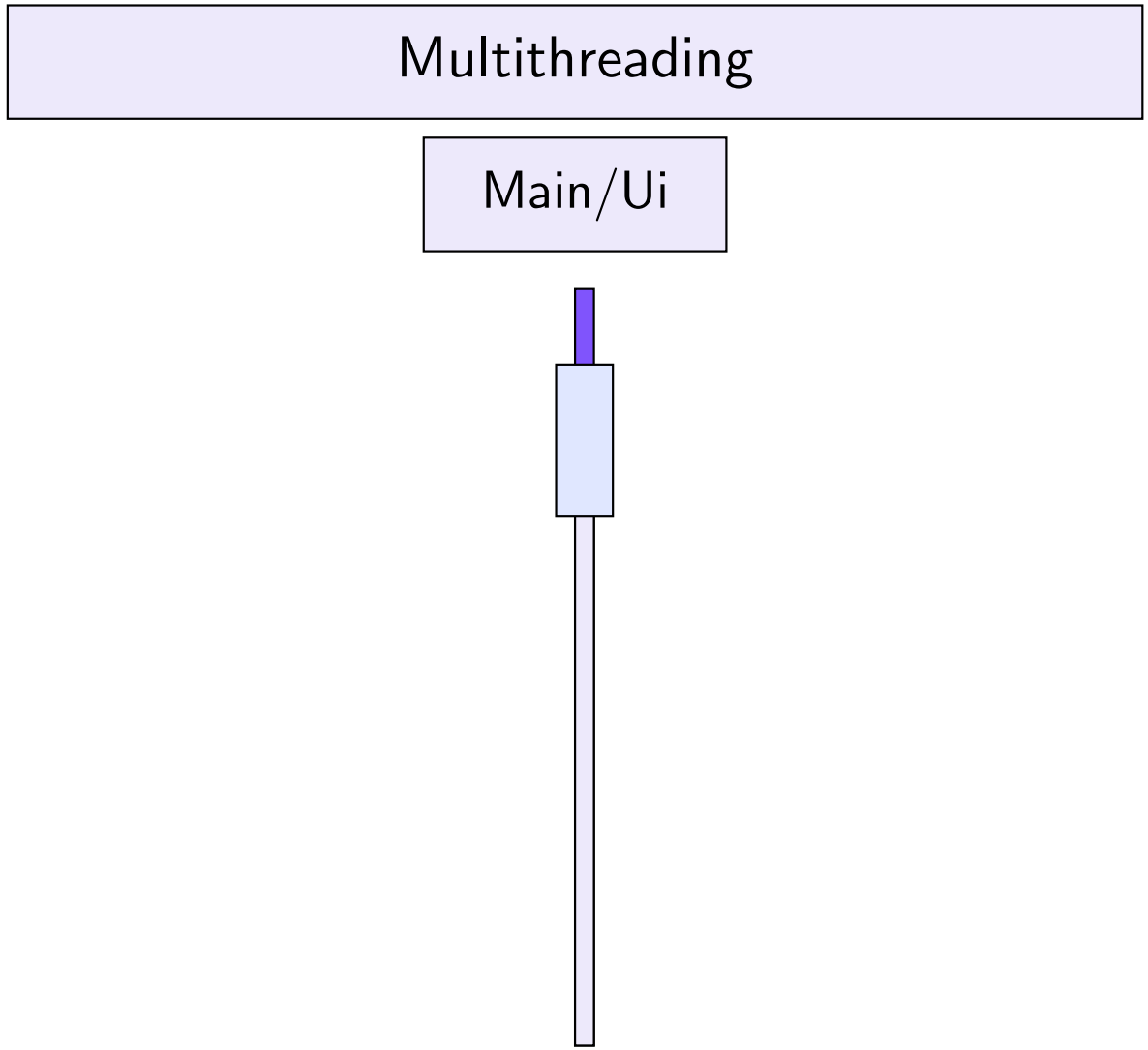
Android: Coroutines: Beispiel



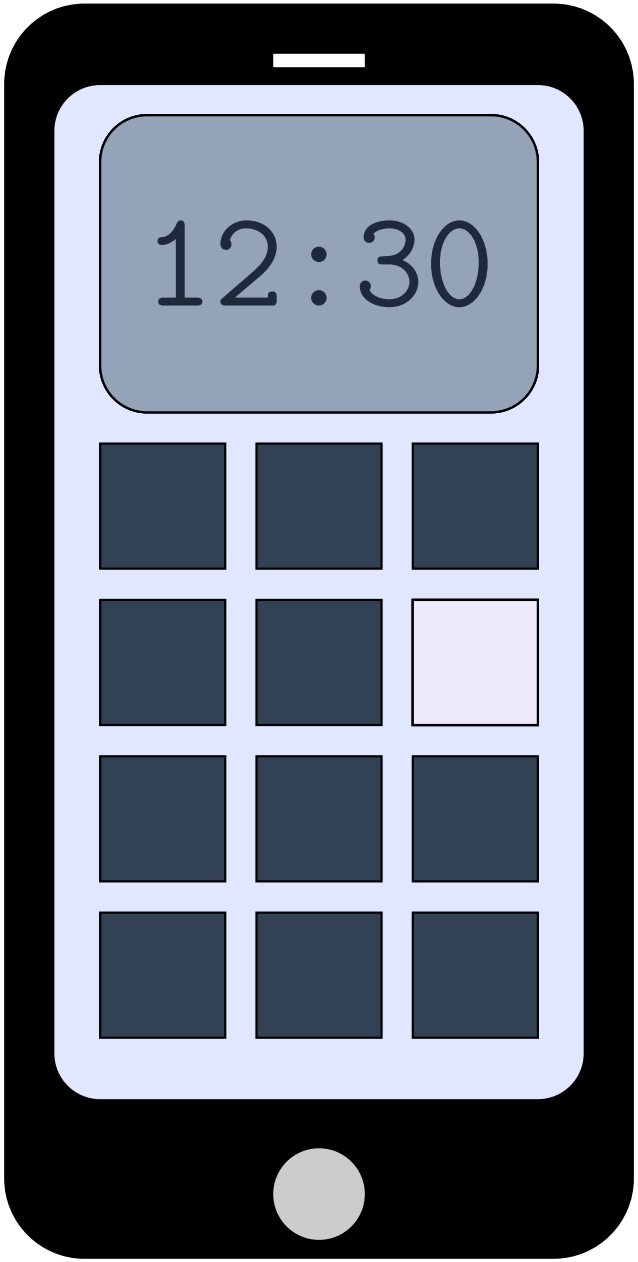


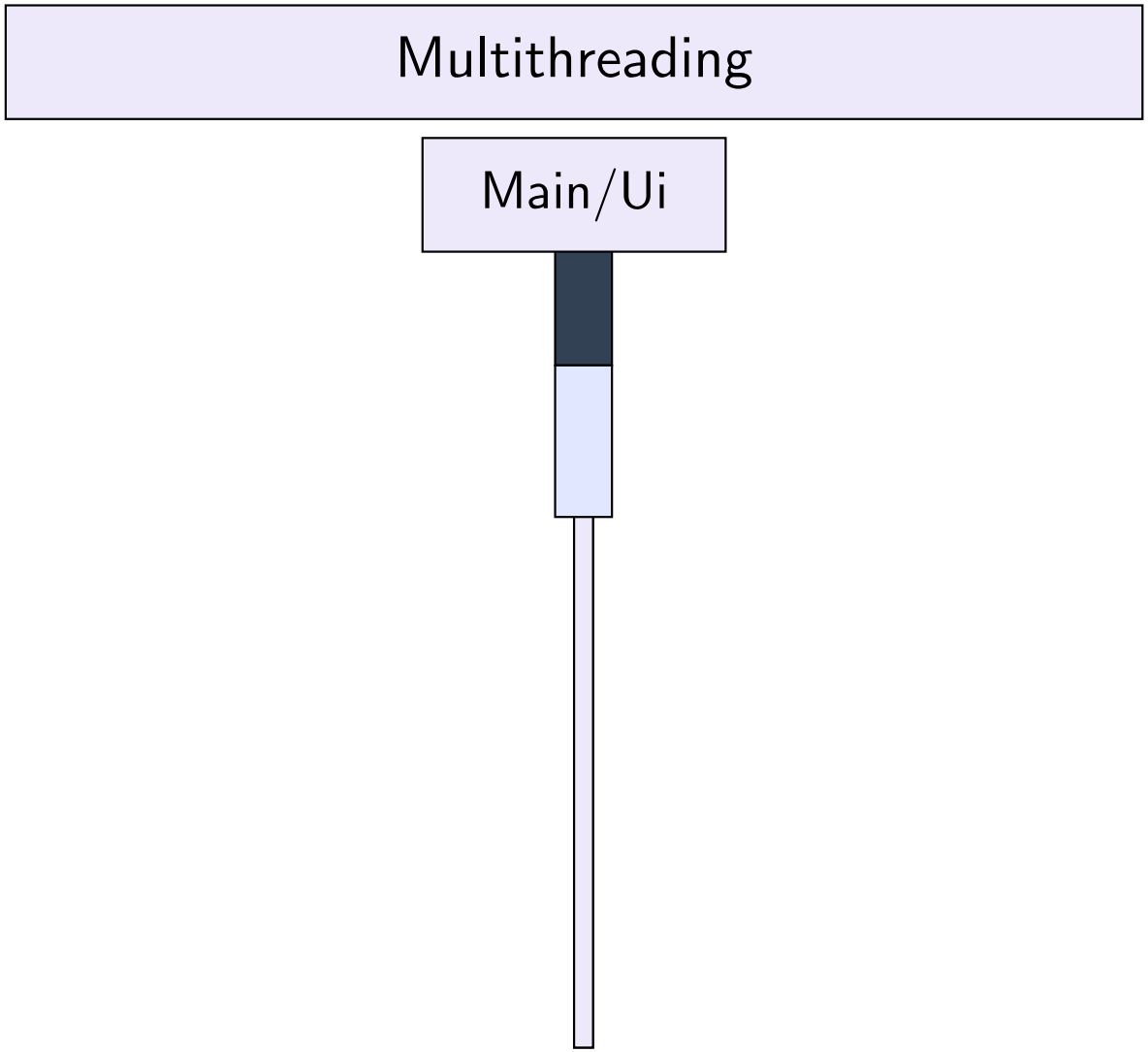
Android: Coroutines: Beispiel



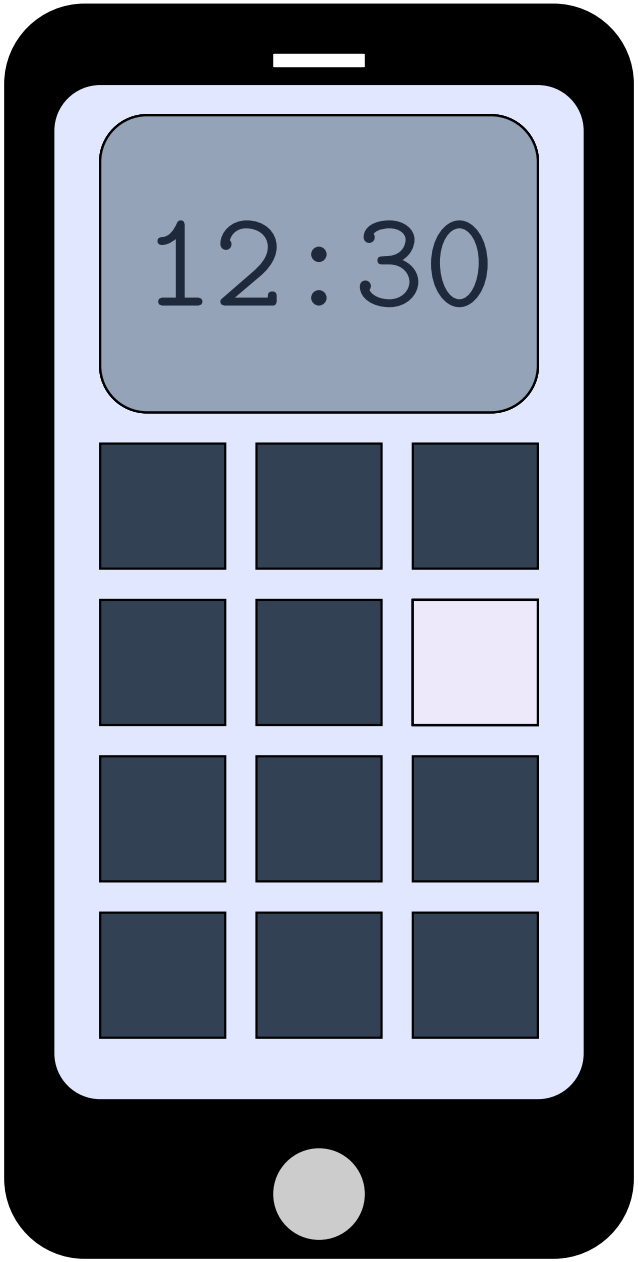


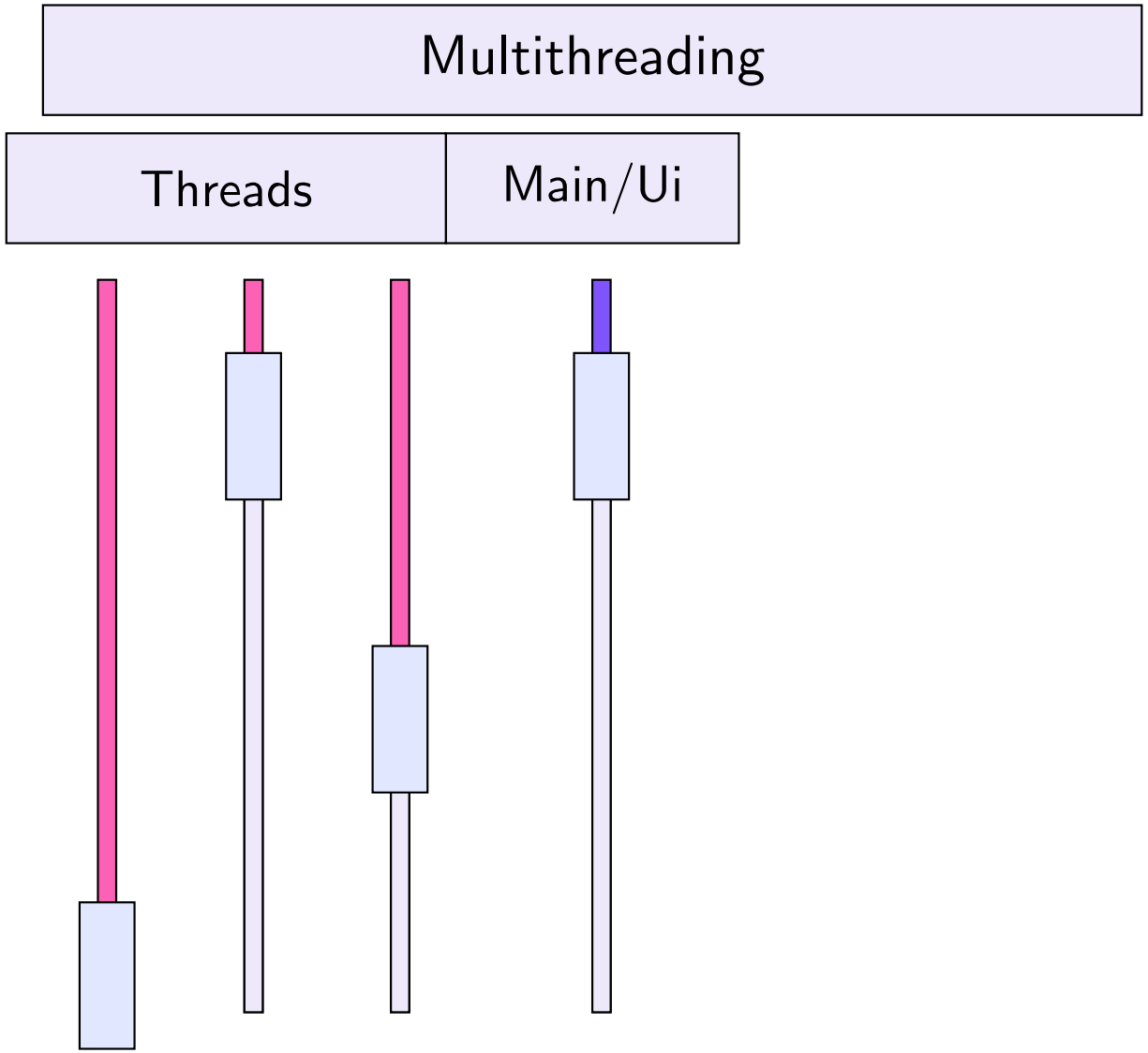
Android: Coroutines: Beispiel



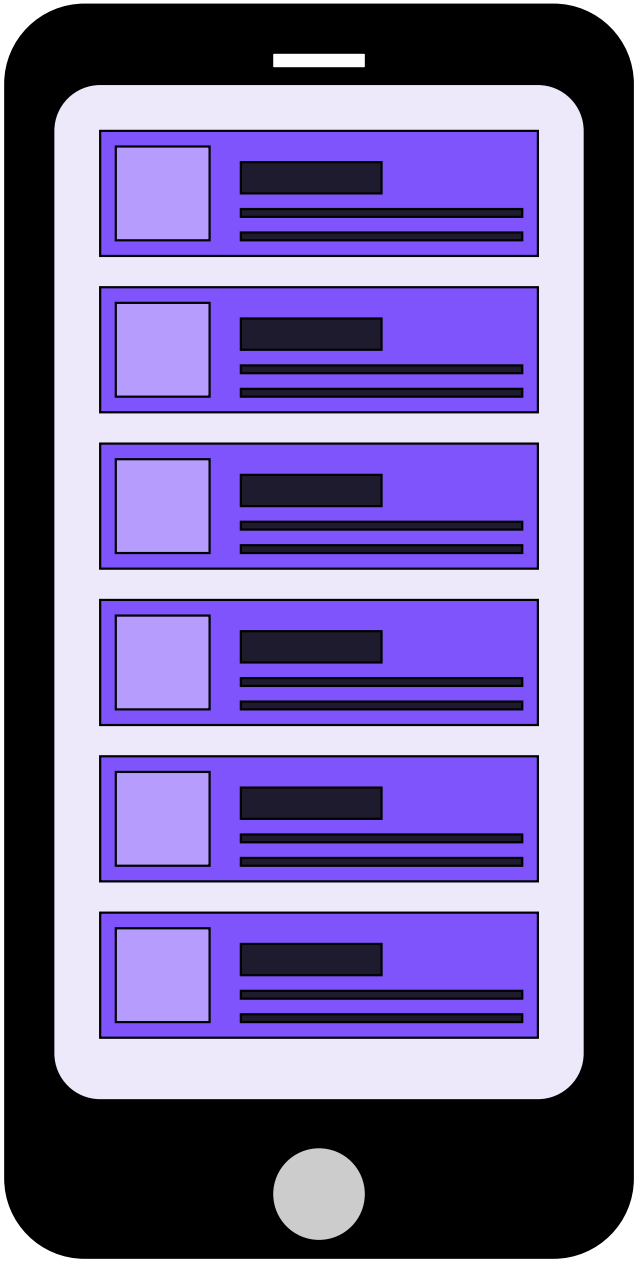


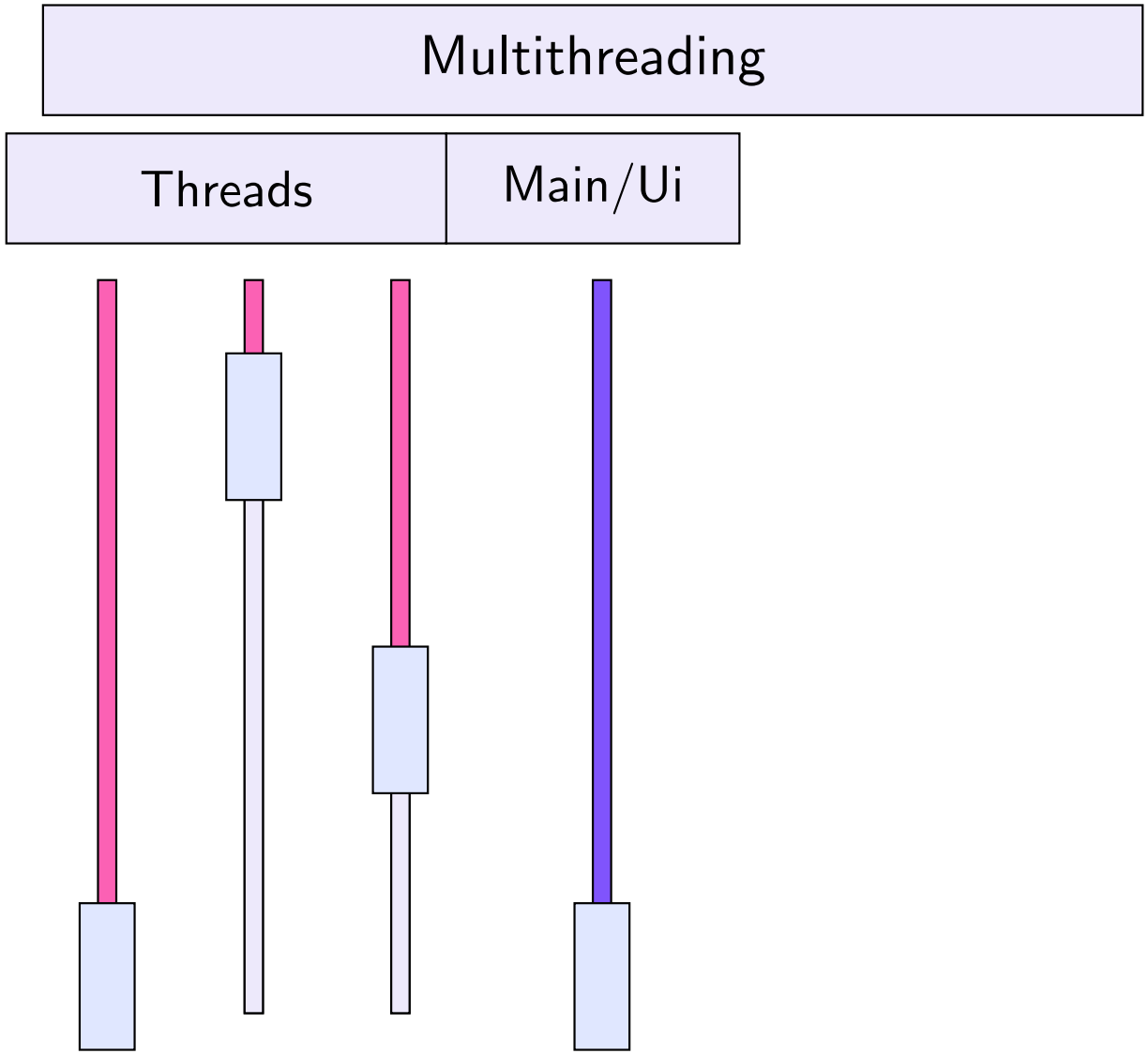
Android: Coroutines: Beispiel



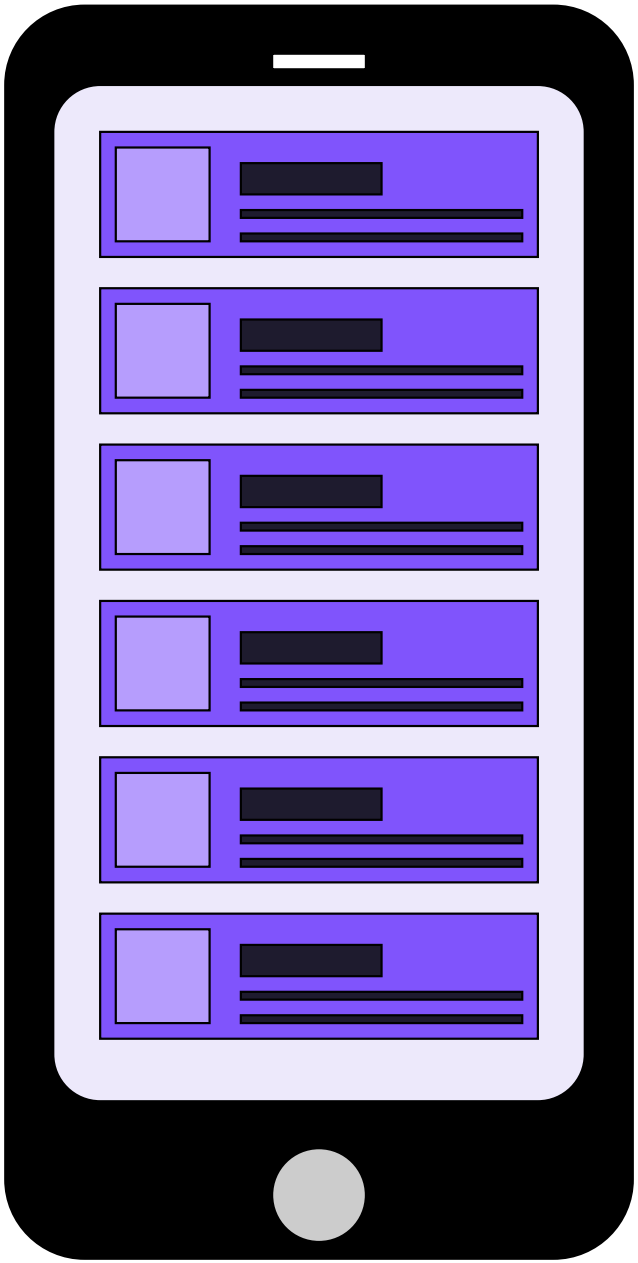


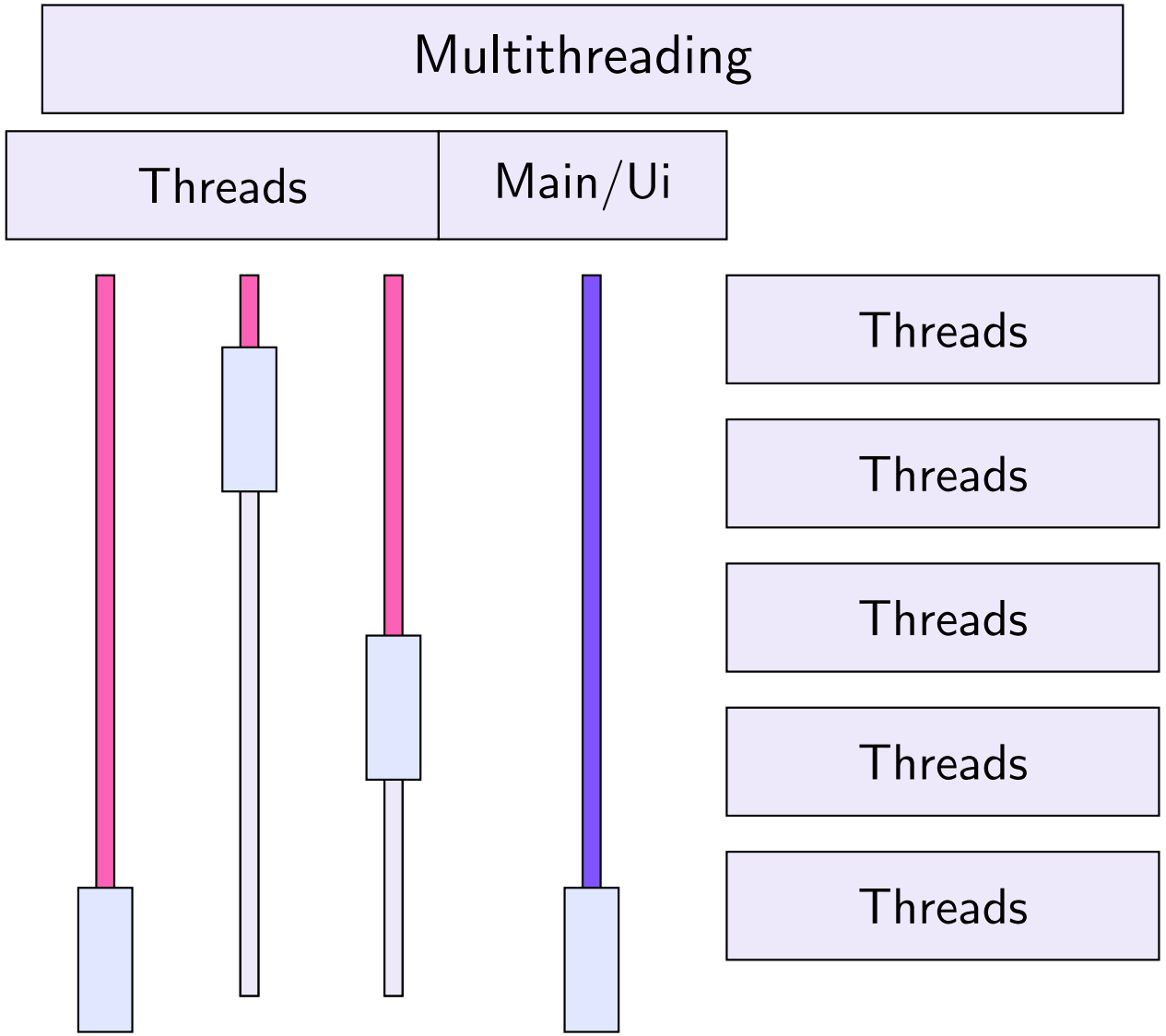
Android: Coroutines: Beispiel



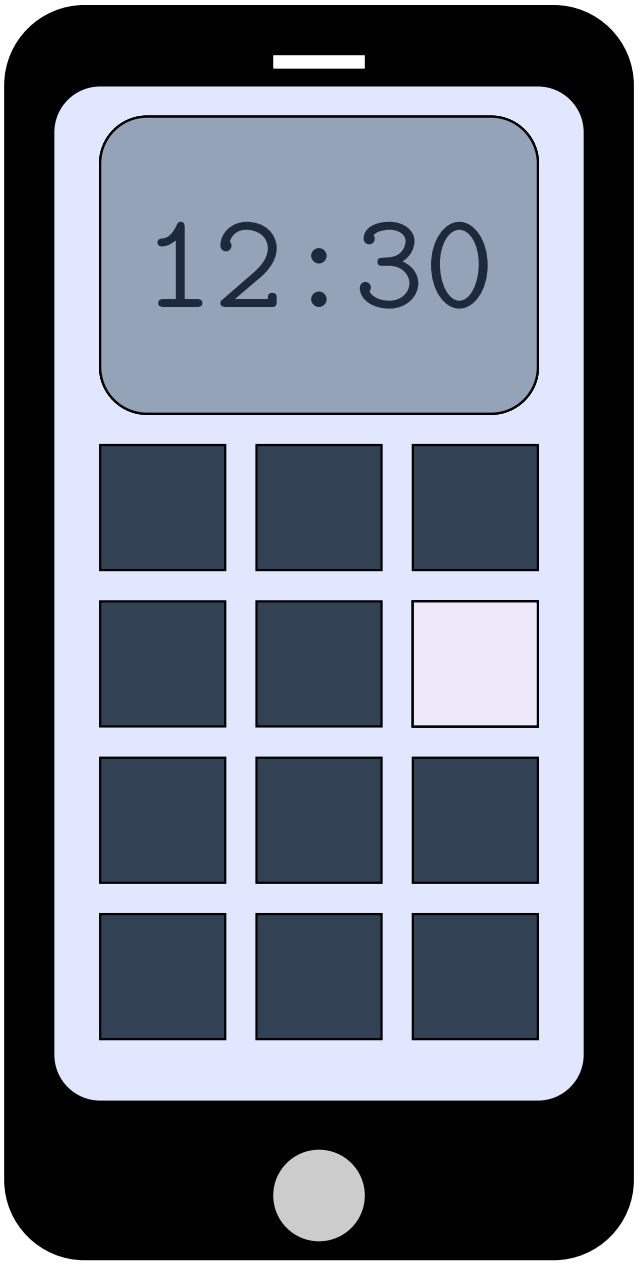


Android: Coroutines: Beispiel



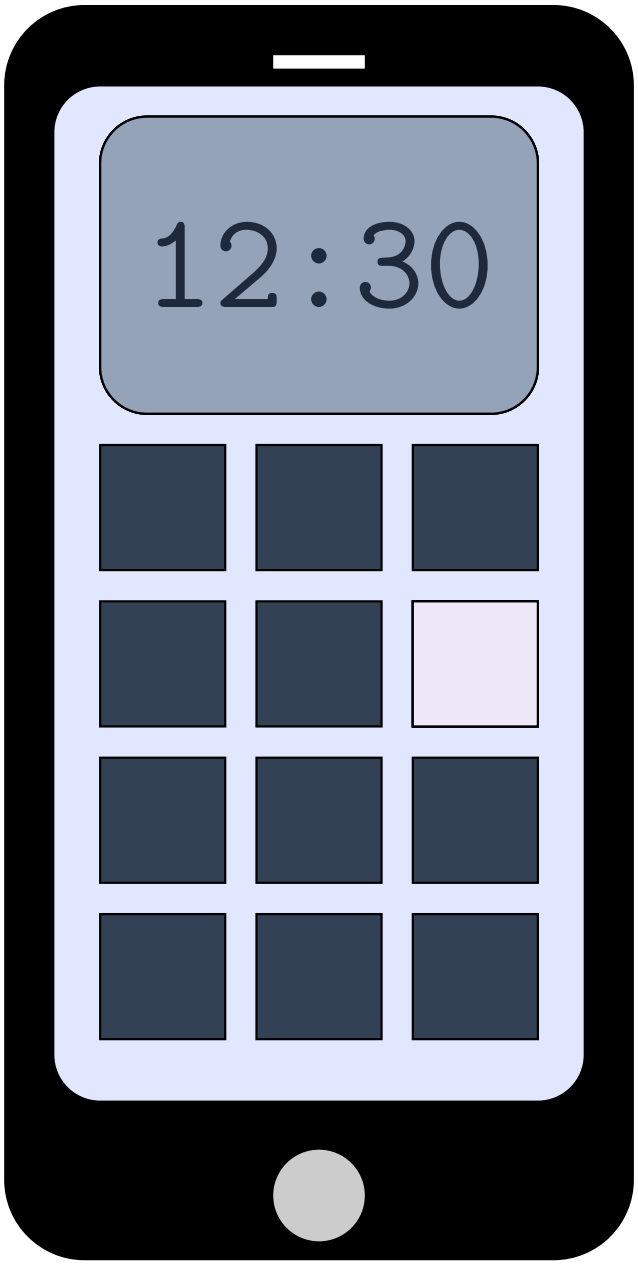


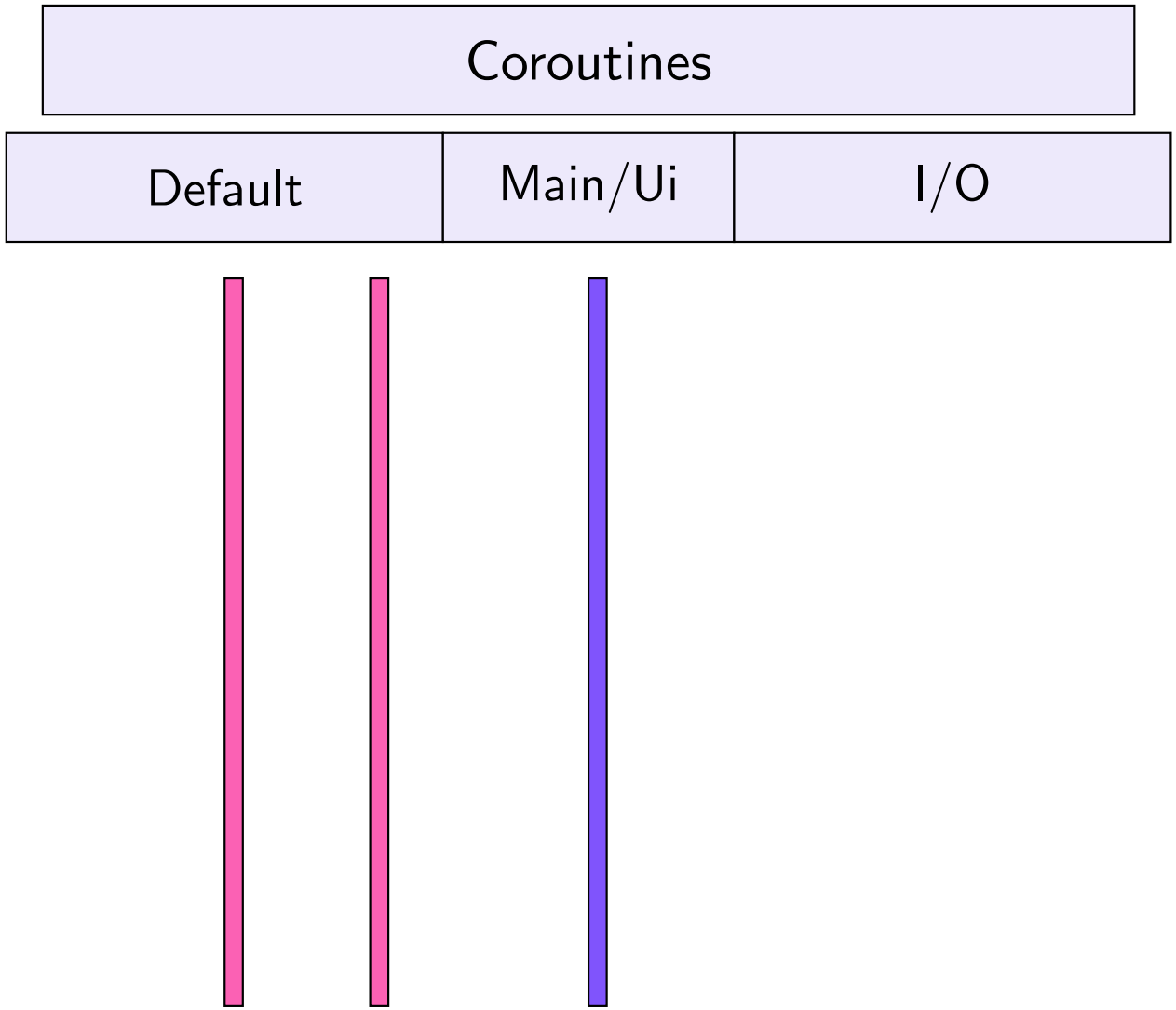
Android: Coroutines: Beispiel



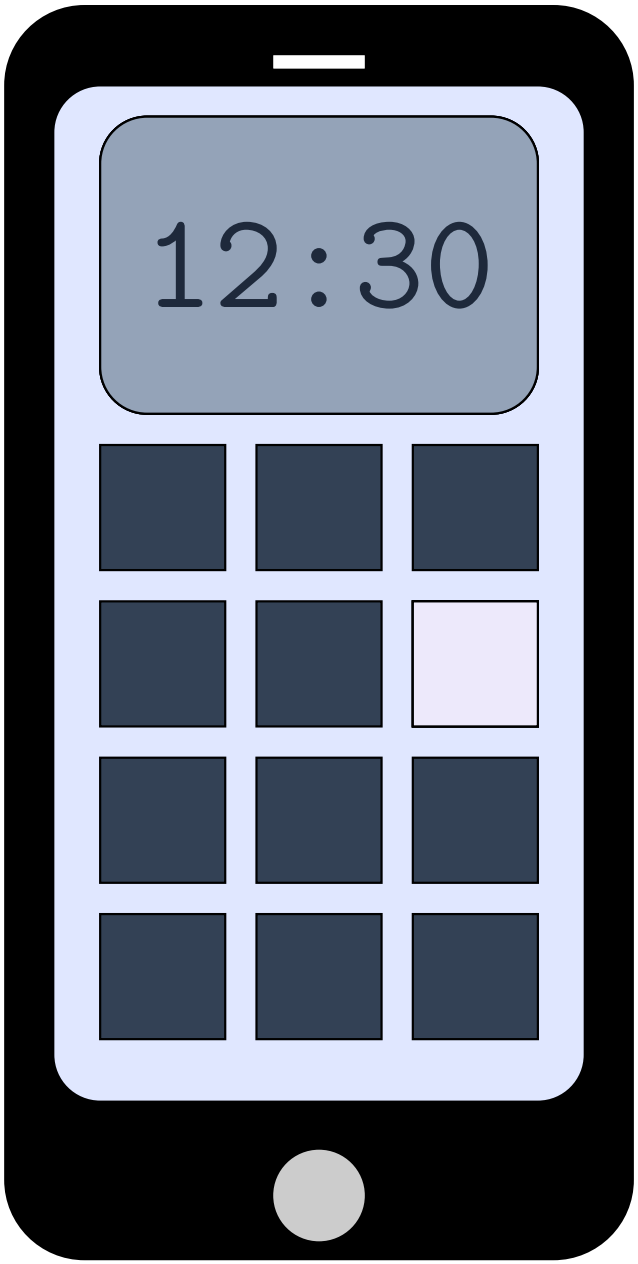
Coroutines

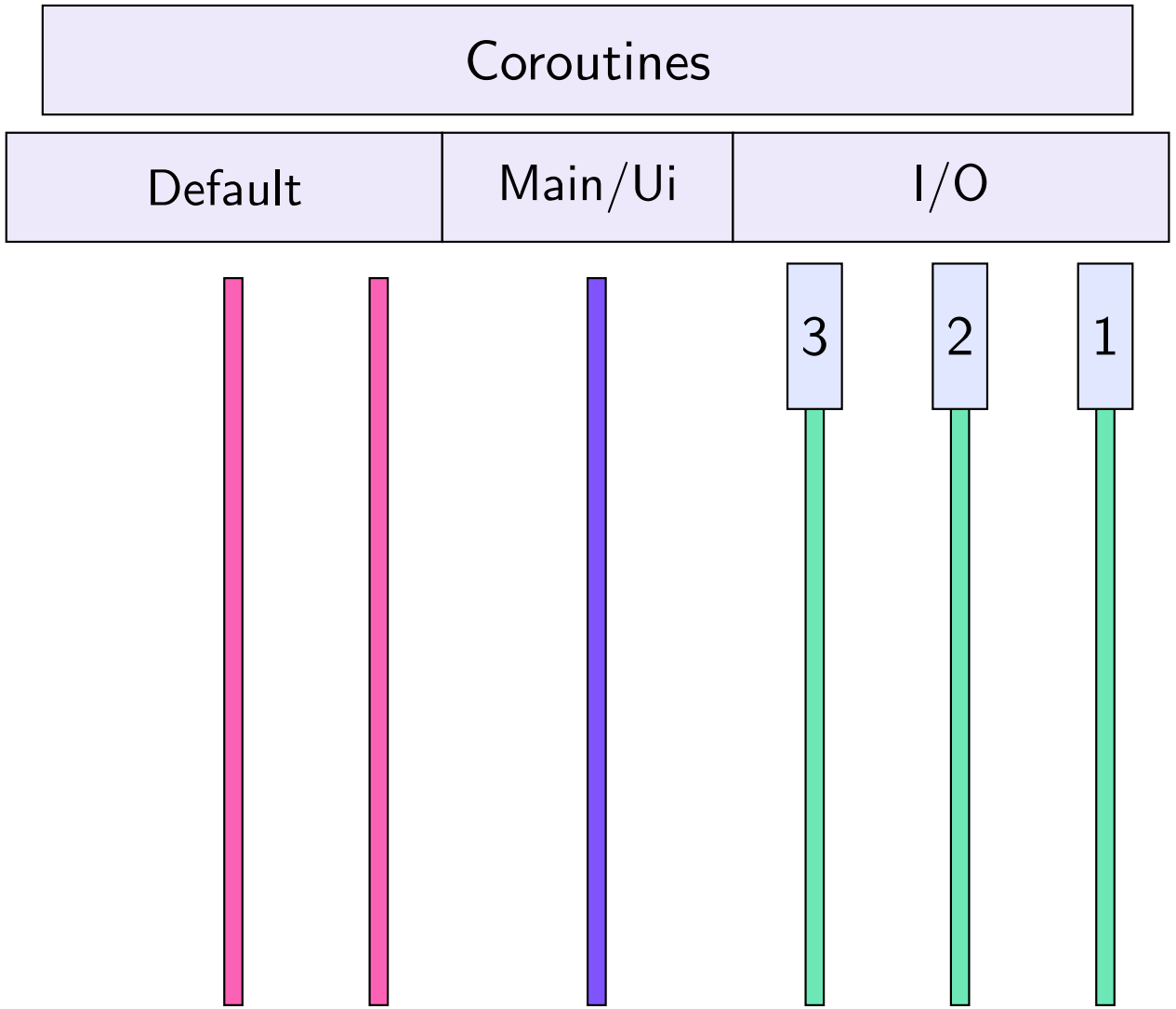
Android: Coroutines: Beispiel



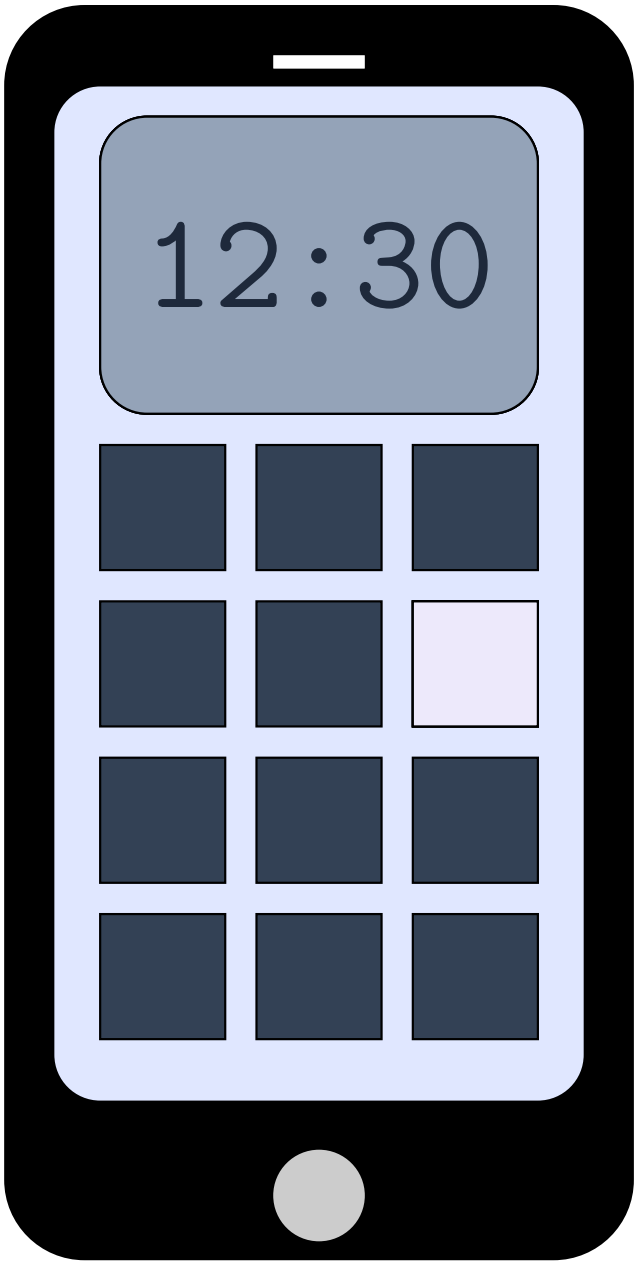


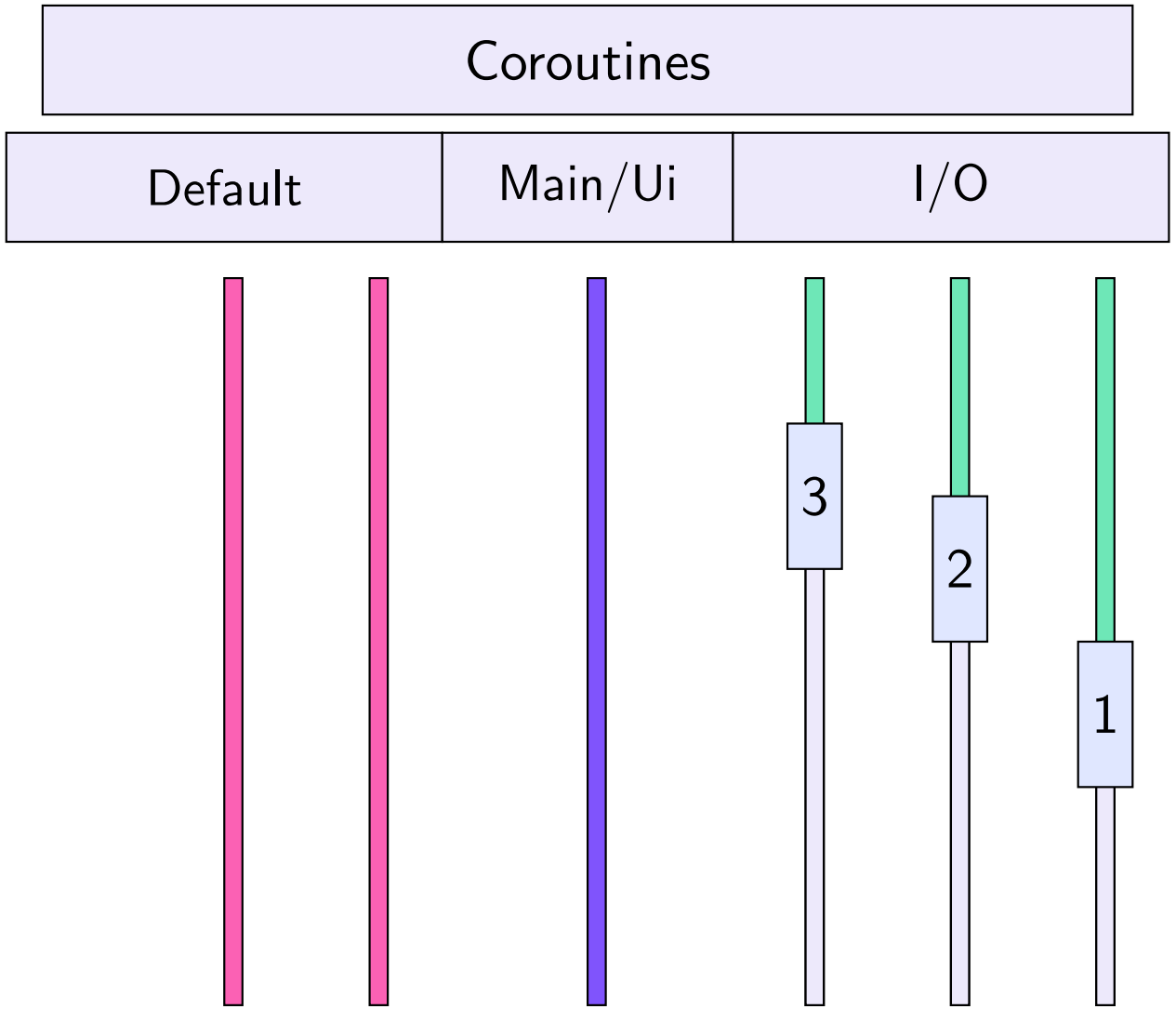
Android: Coroutines: Beispiel



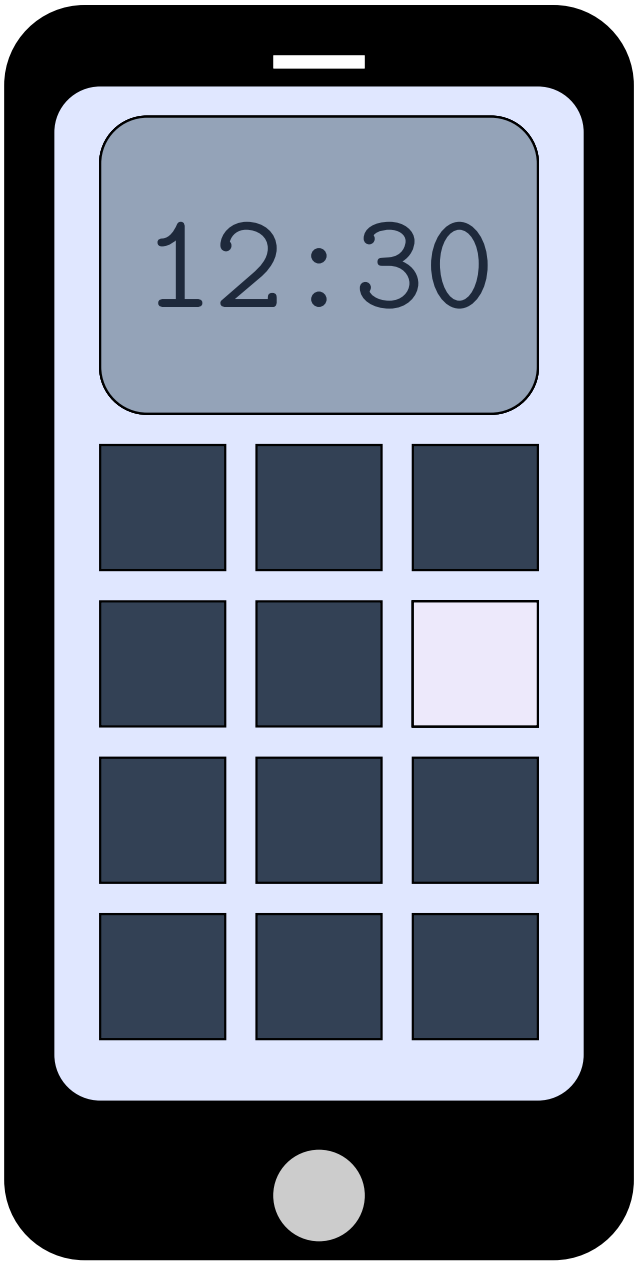


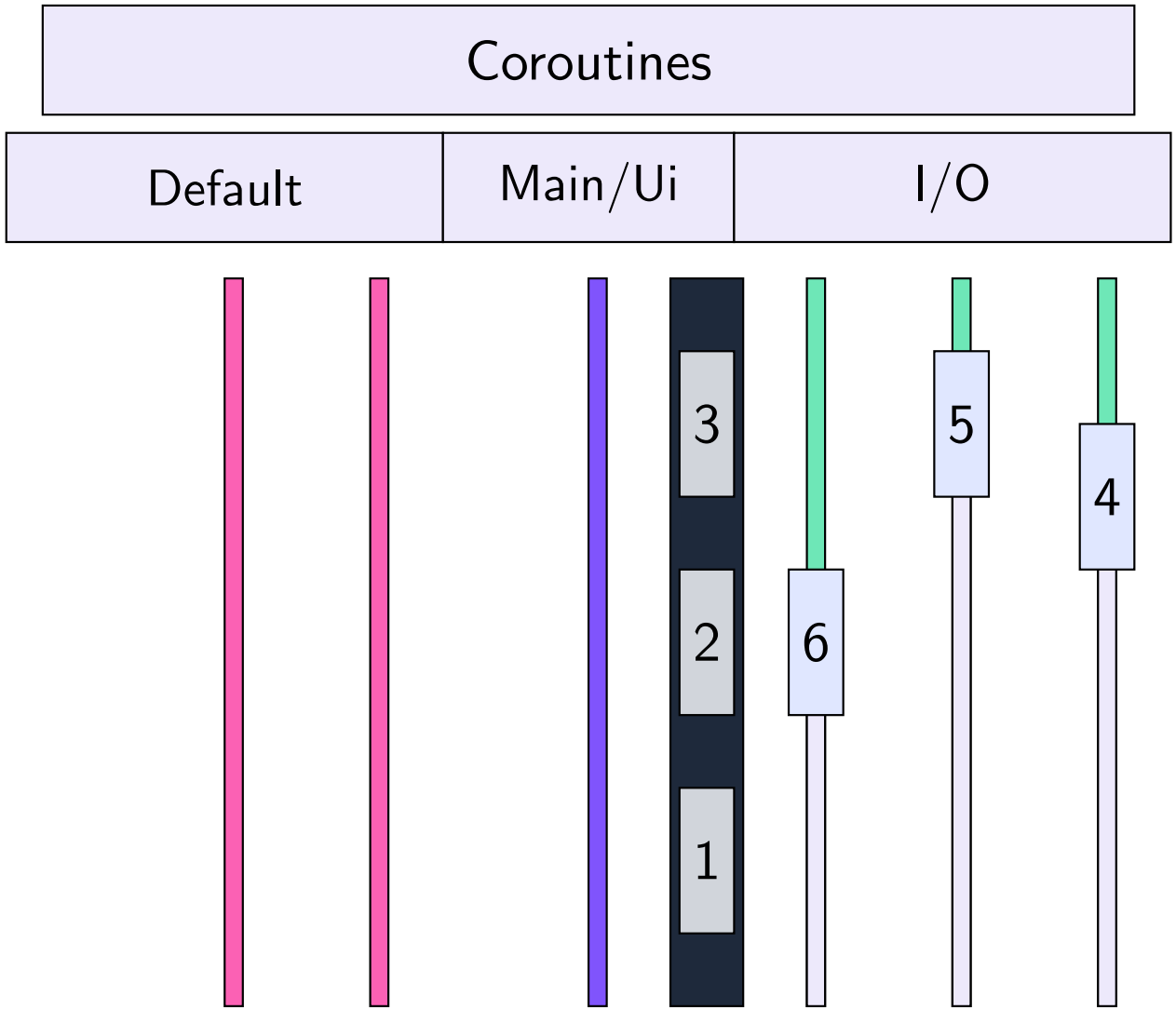
Android: Coroutines: Beispiel



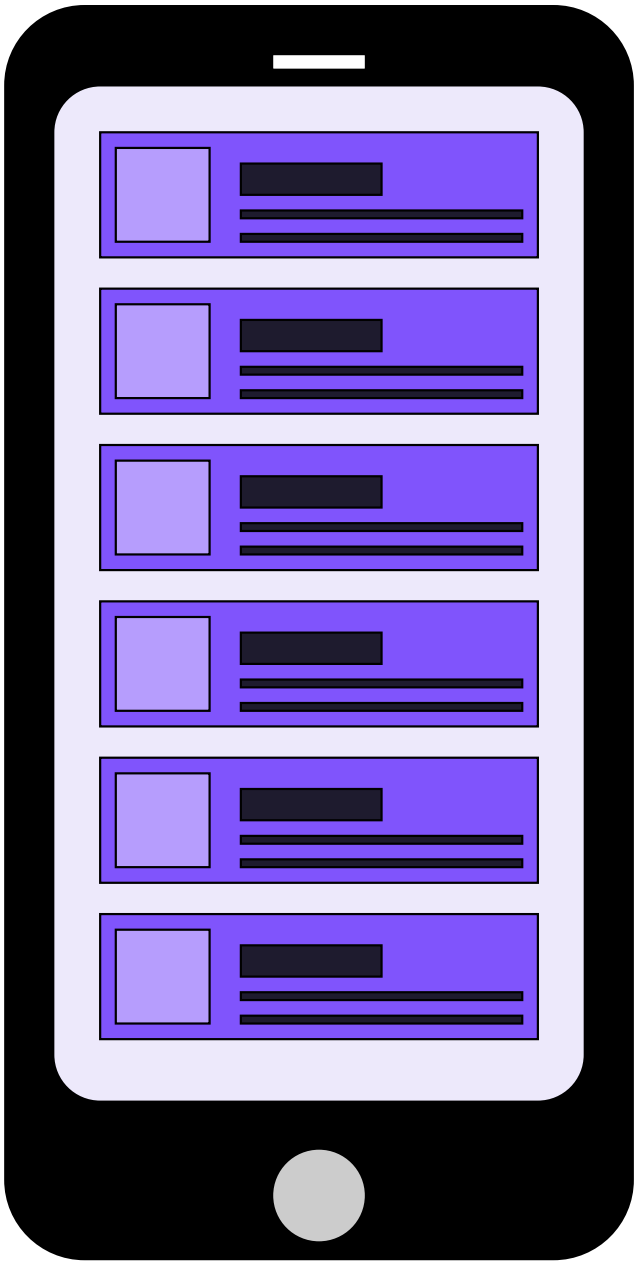


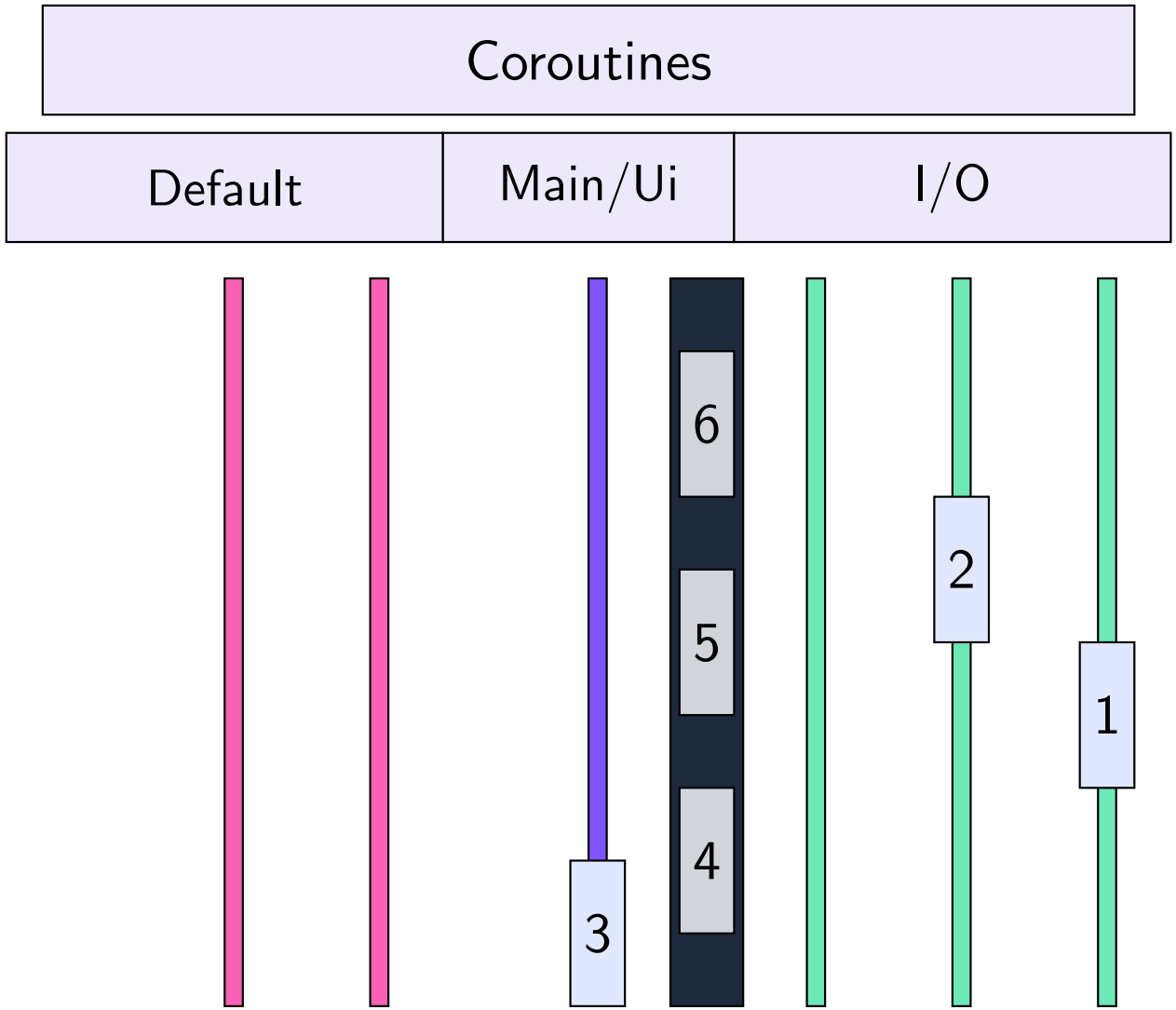
Android: Coroutines: Beispiel



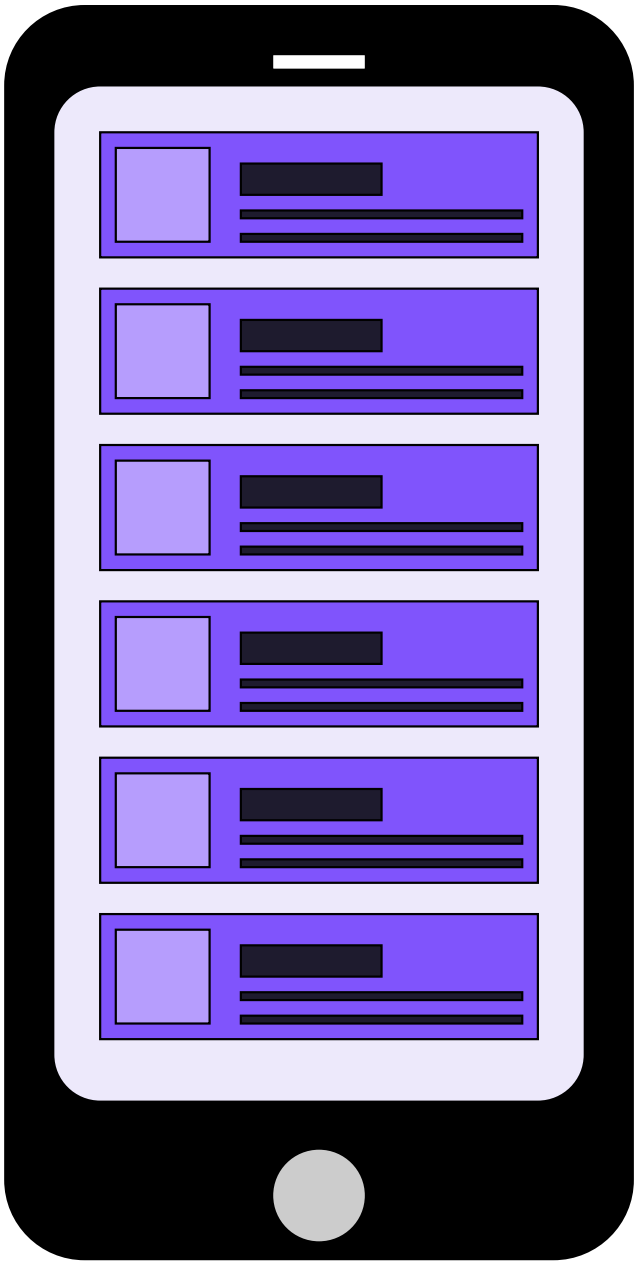


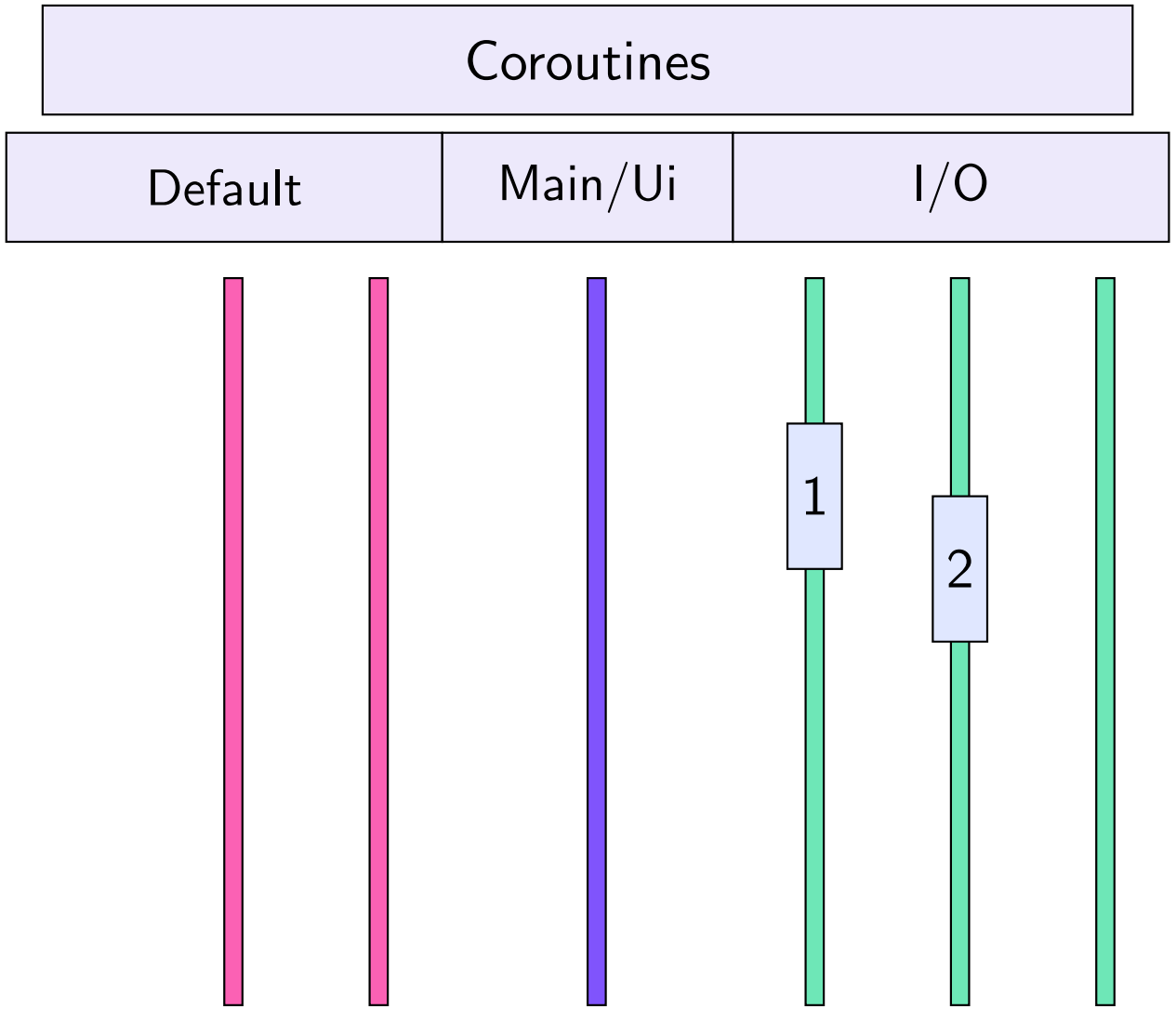
Android: Coroutines: Beispiel



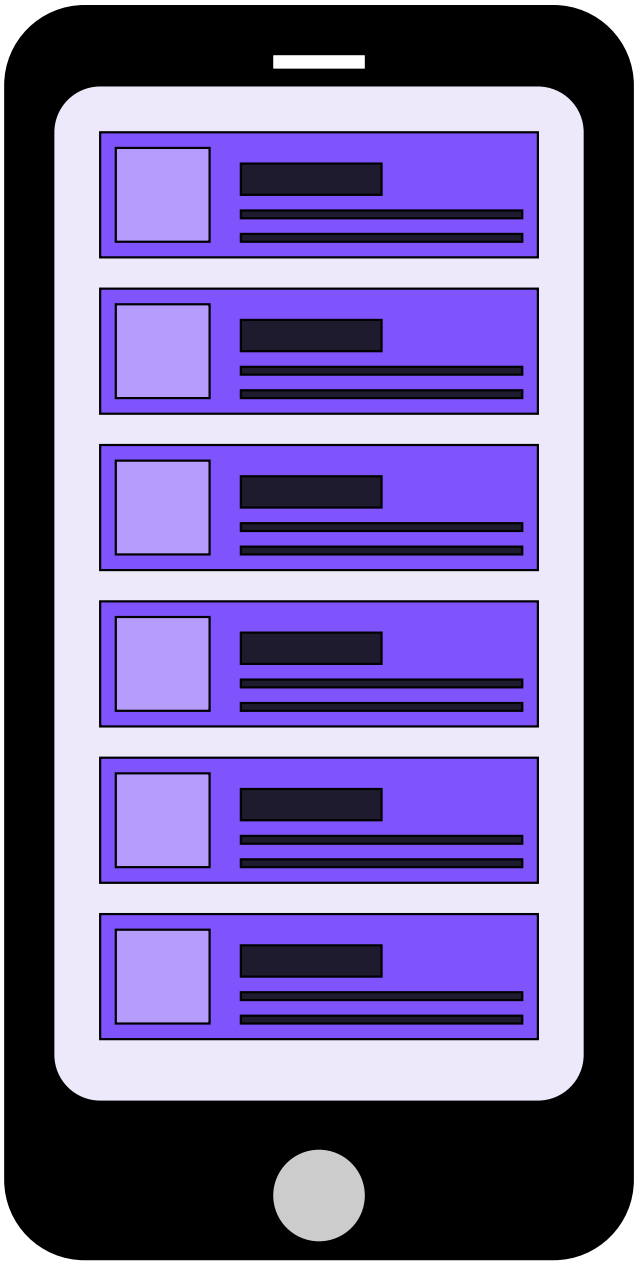


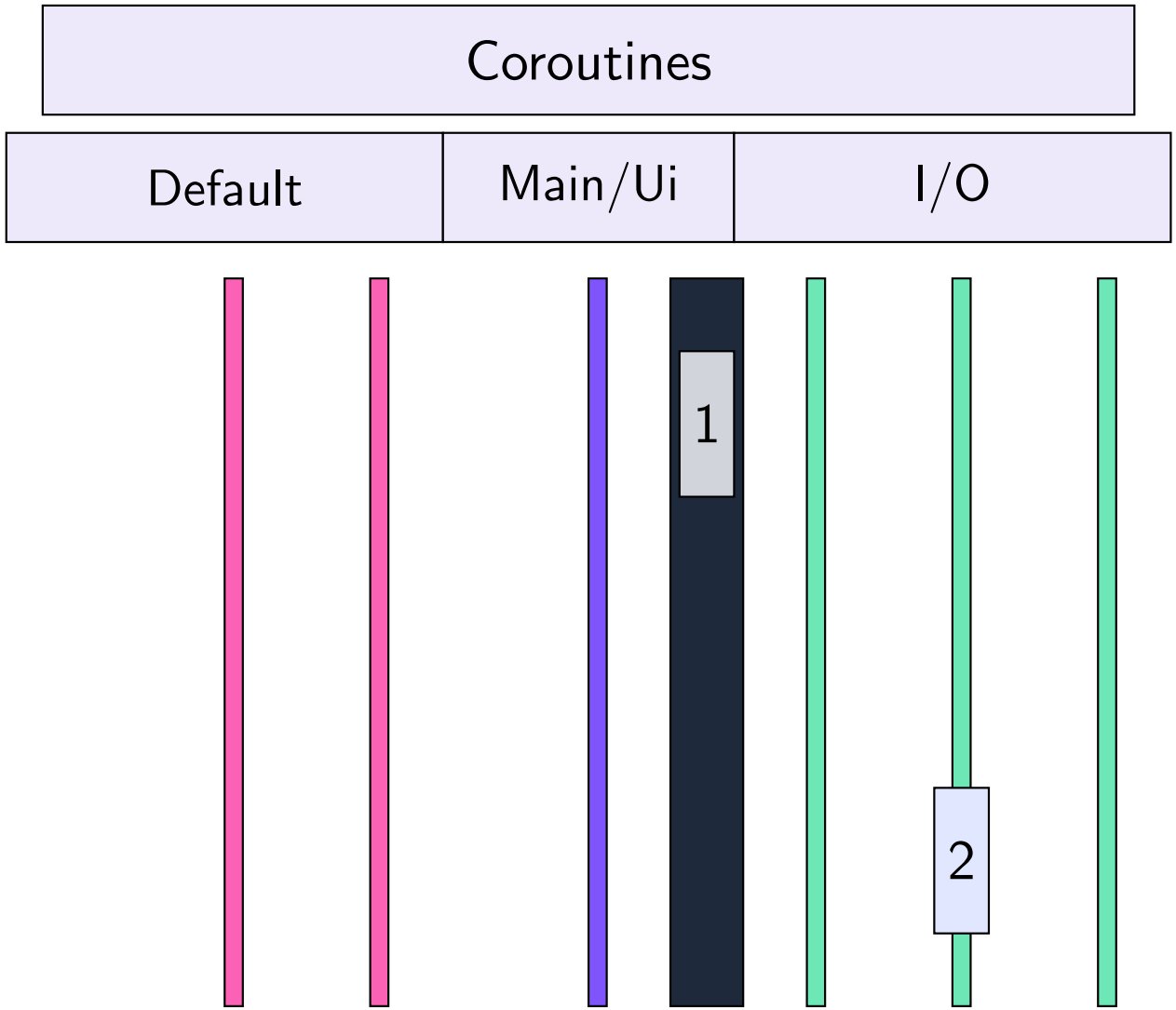
Android: Coroutines: Beispiel



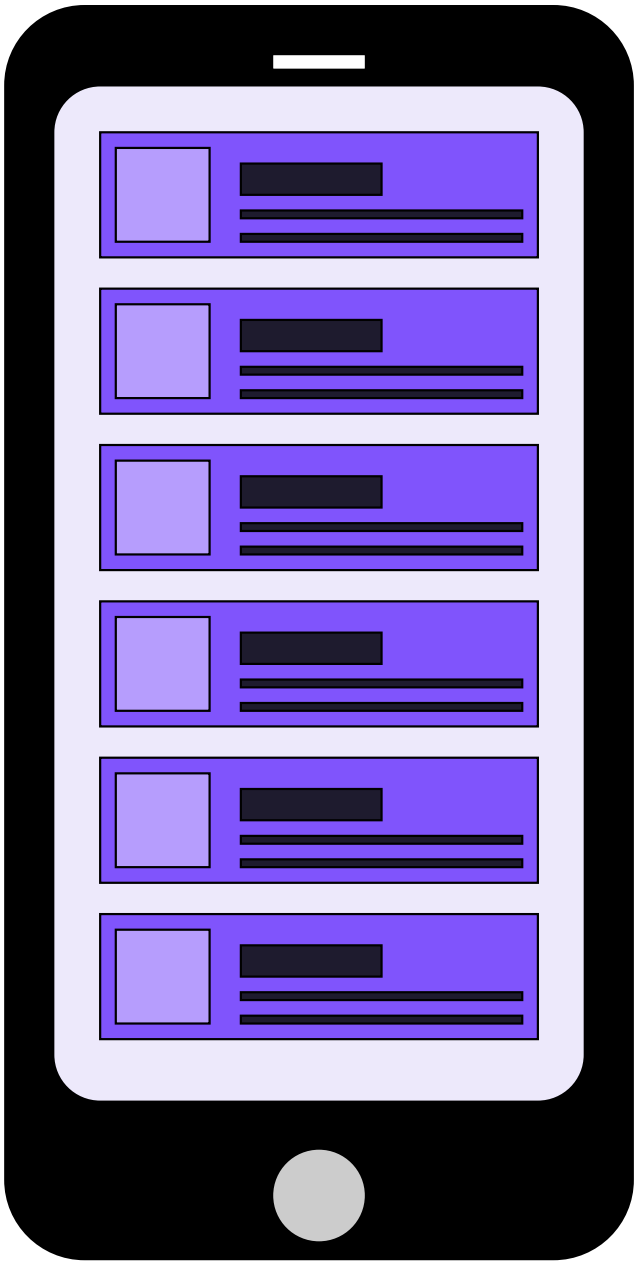


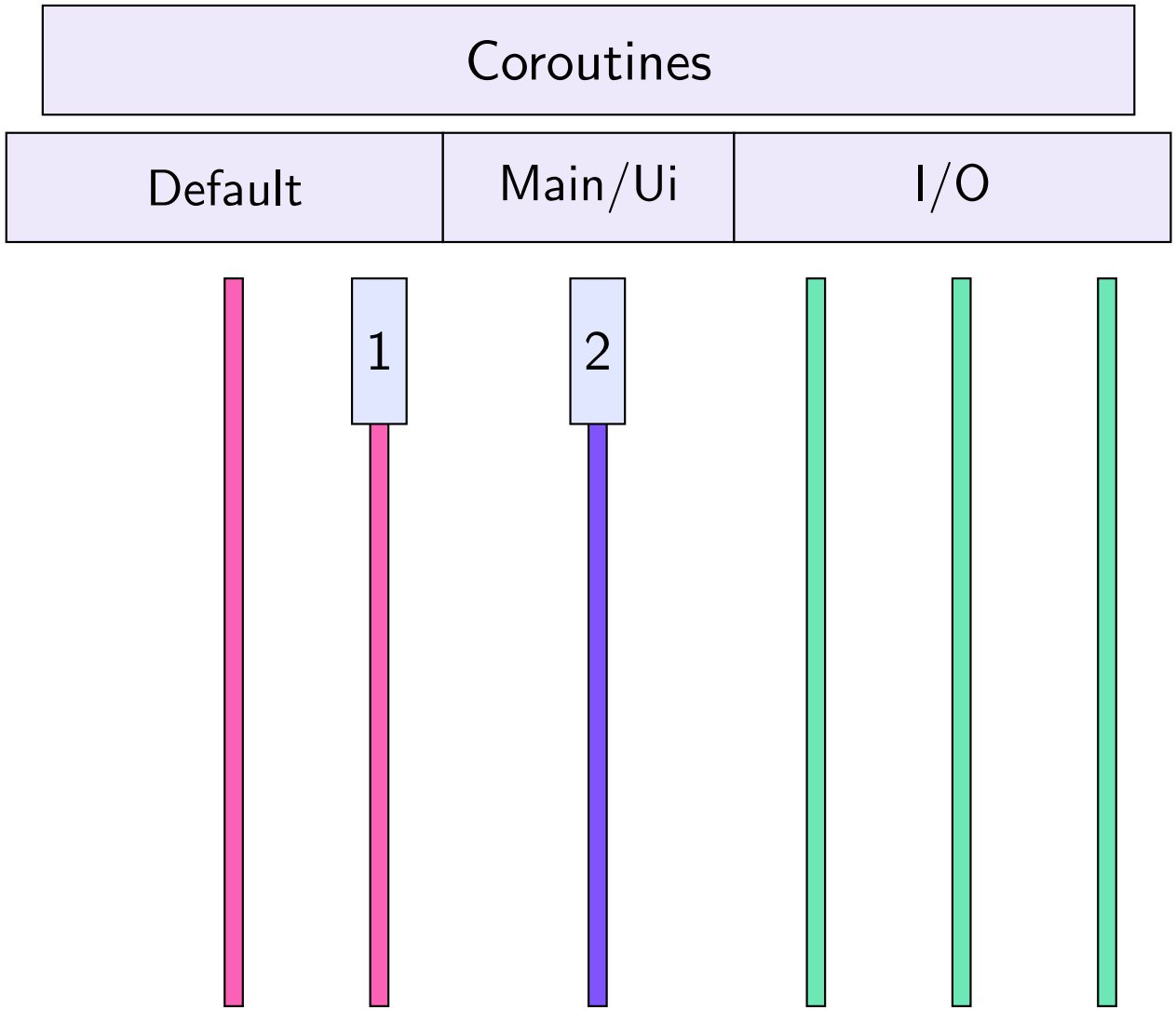
Android: Coroutines: Beispiel



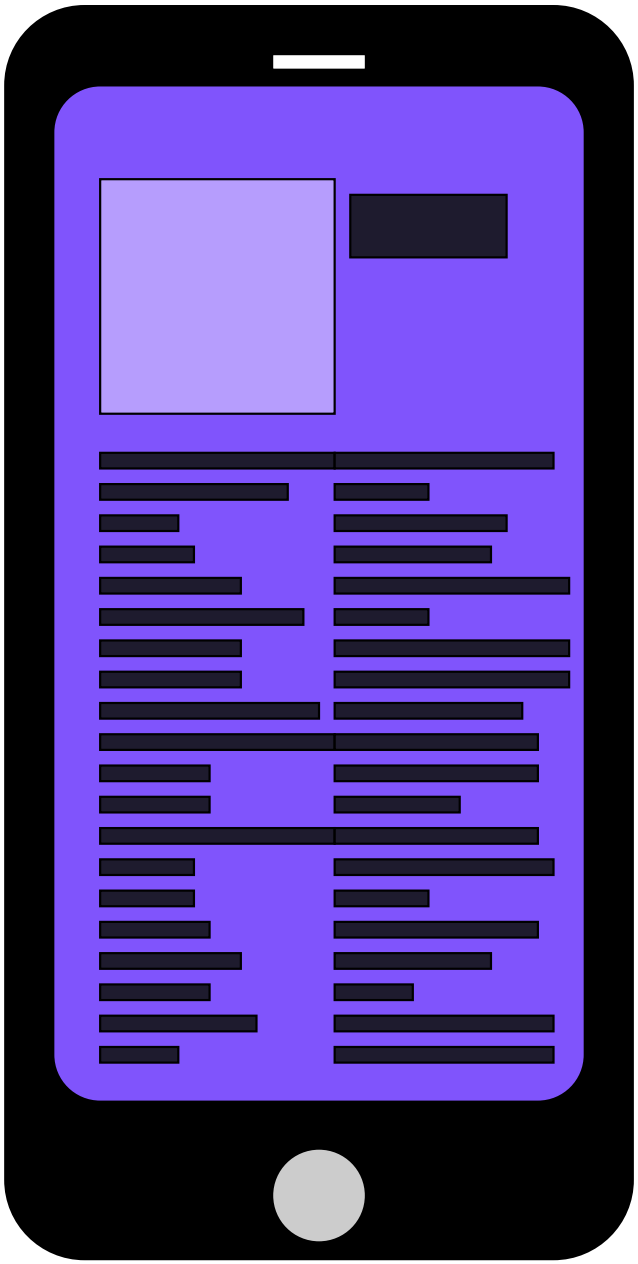


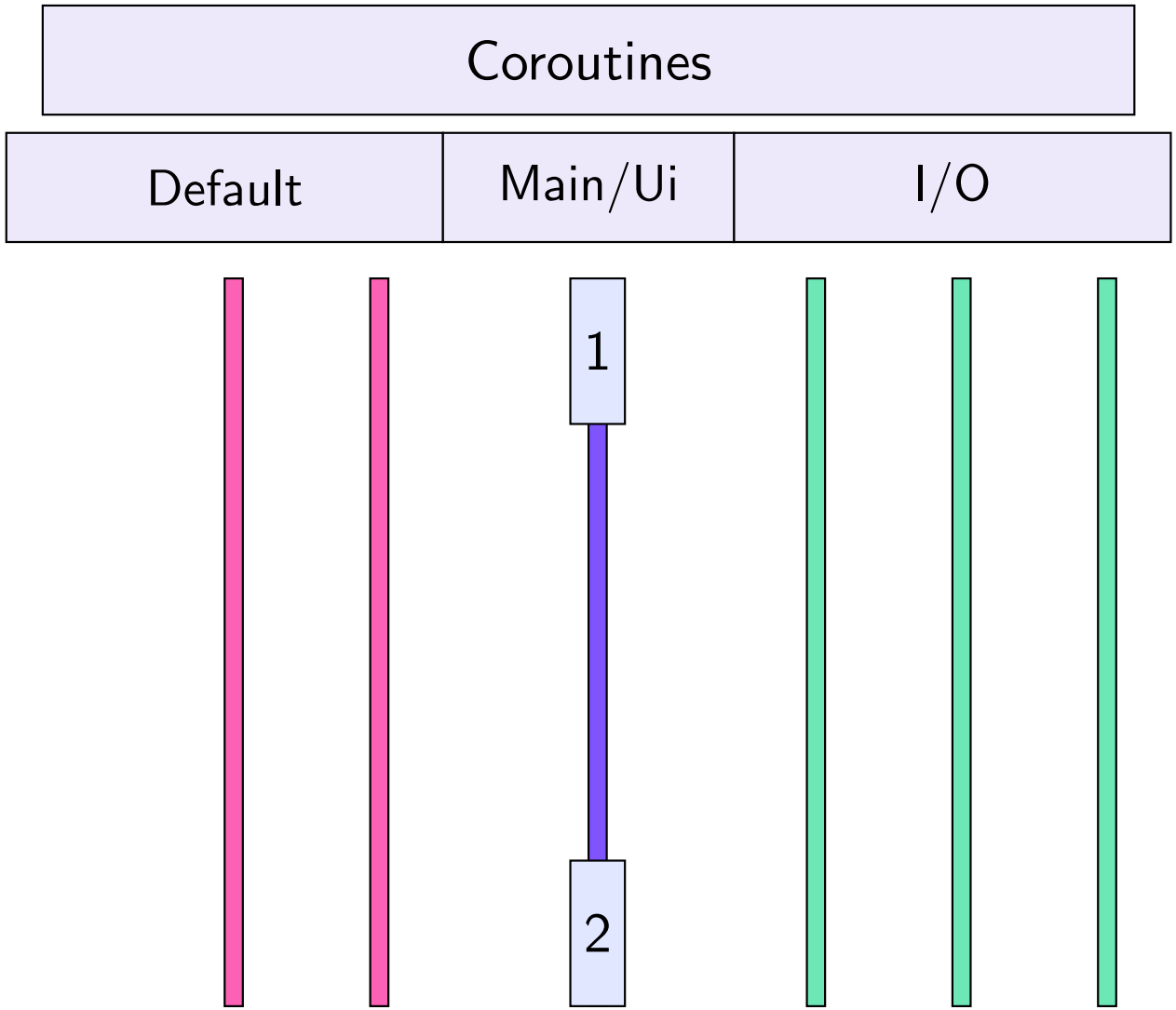
Android: Coroutines: Beispiel





Android: Coroutines: Beispiel





Android: Coroutines: Beispiel