

领略 Kotlin 协程的力量



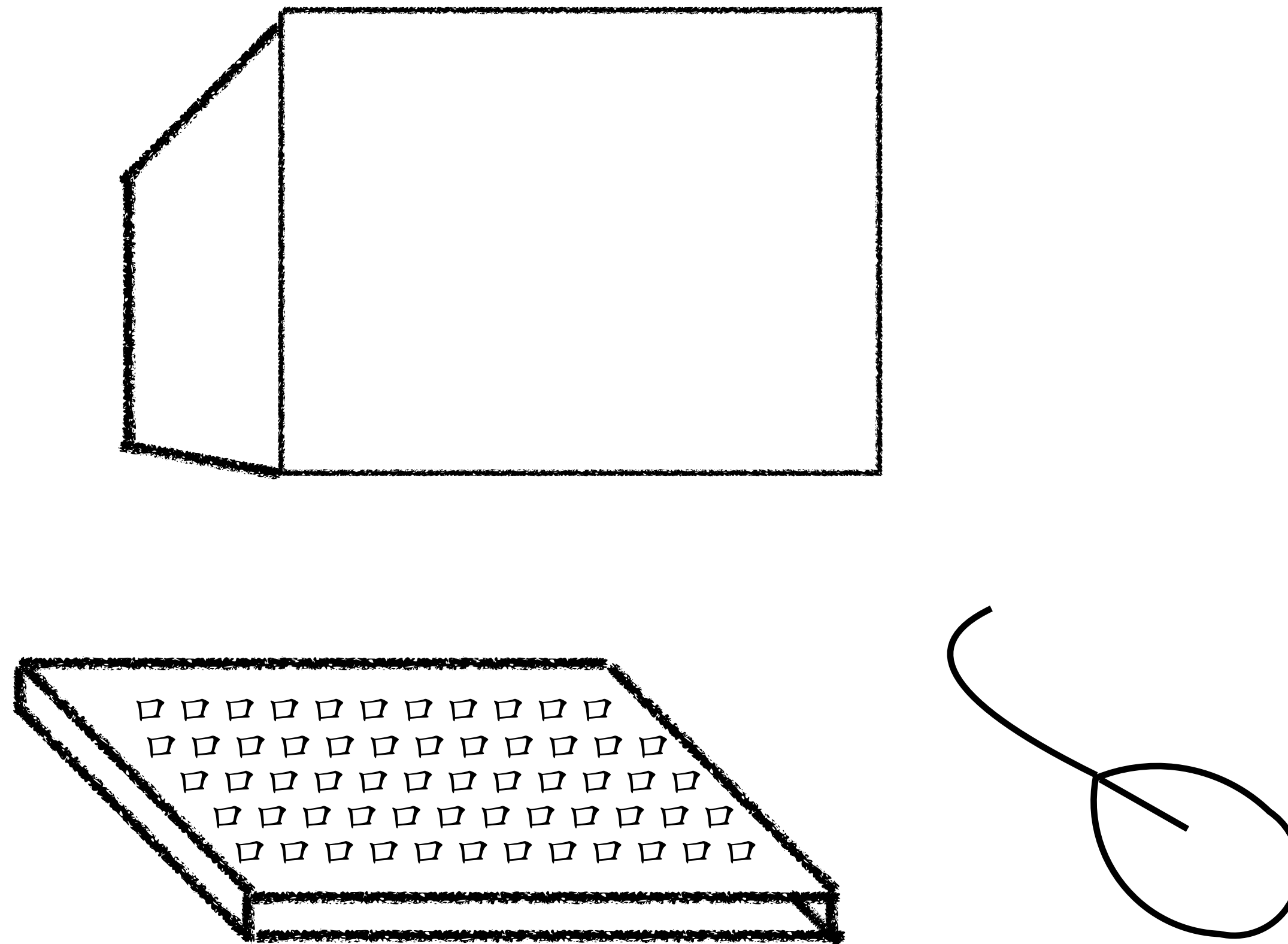
张涛

饿了么 Android

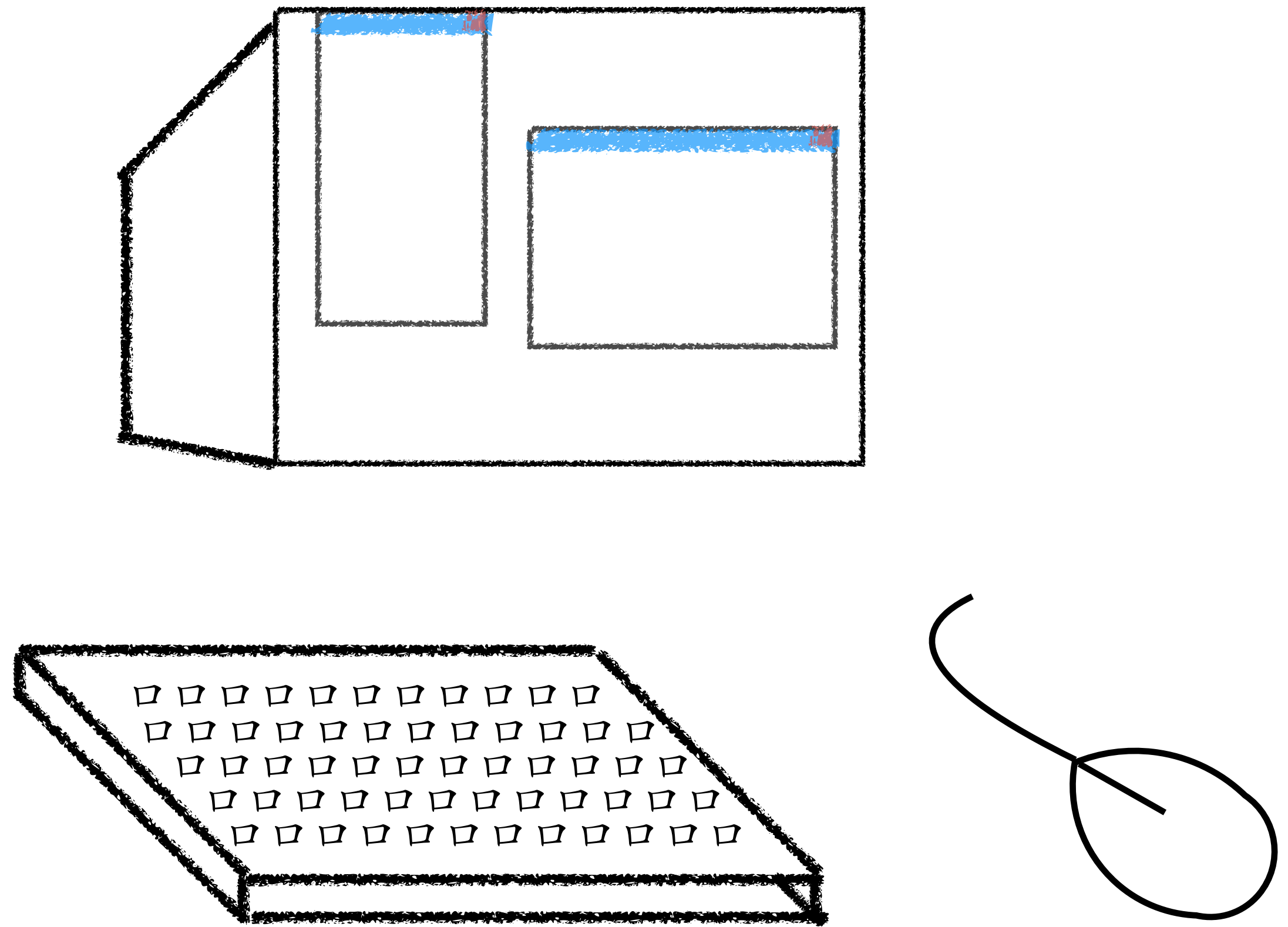
博客：开源实验室

微信：kymjs123

协程是什么

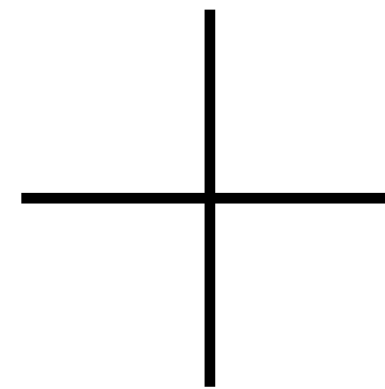
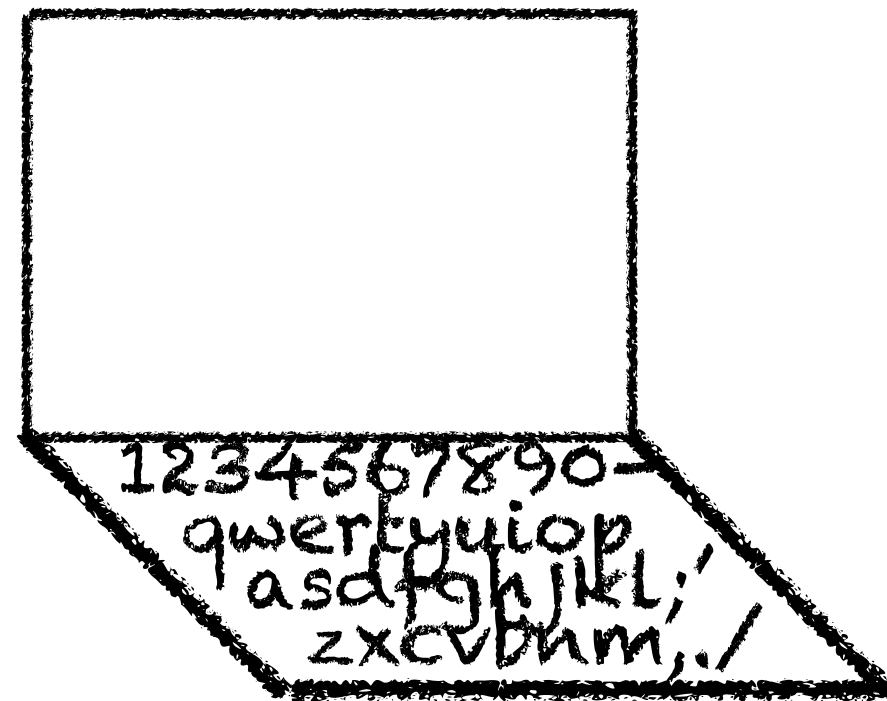


一次执行一个任务

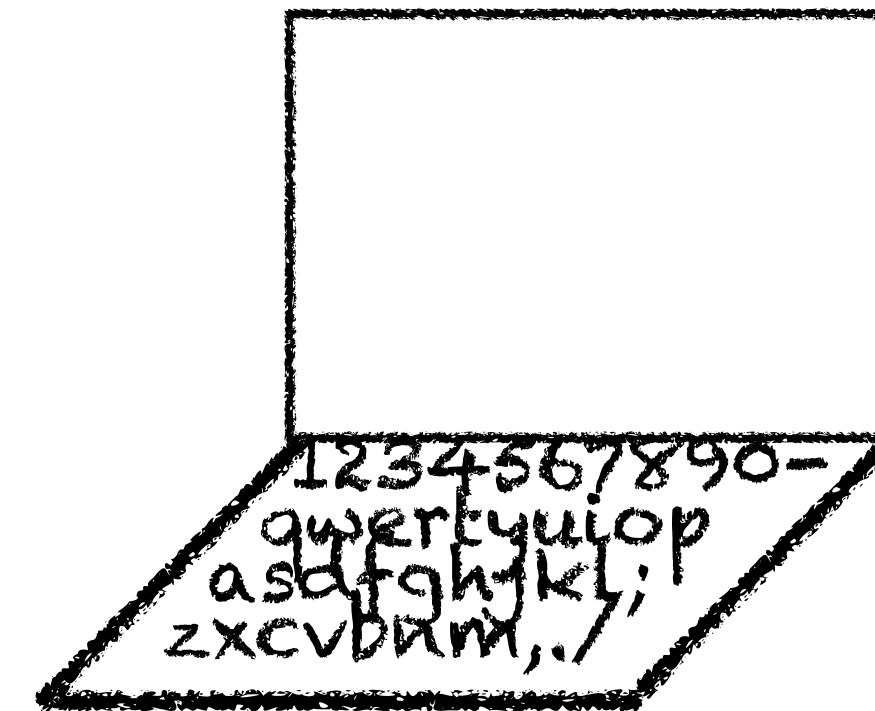


你执行一会，他执行一会

单CPU时间分片

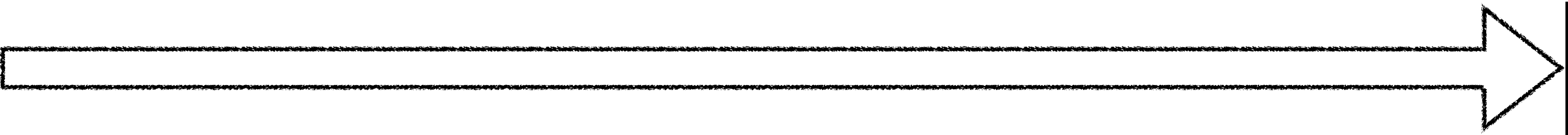


多CPU并行执行

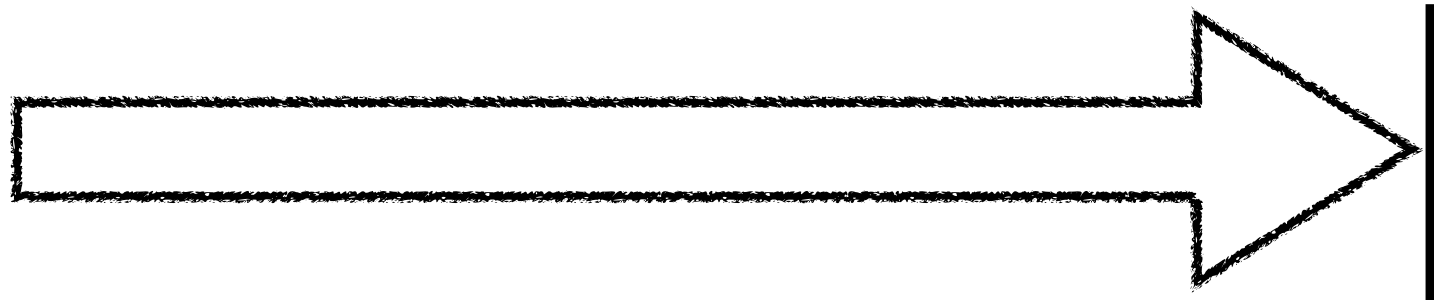


并发执行更多任务

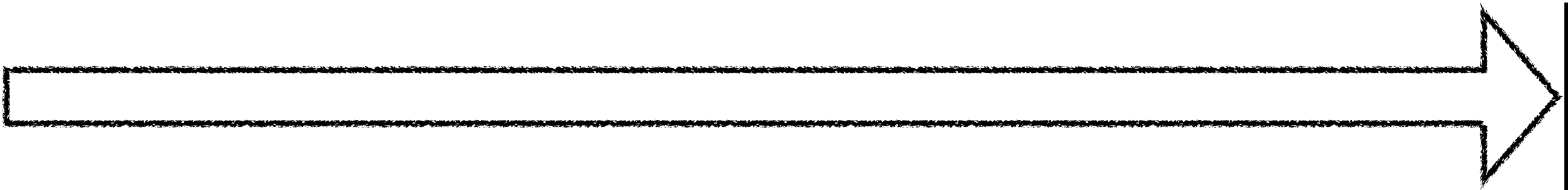
main



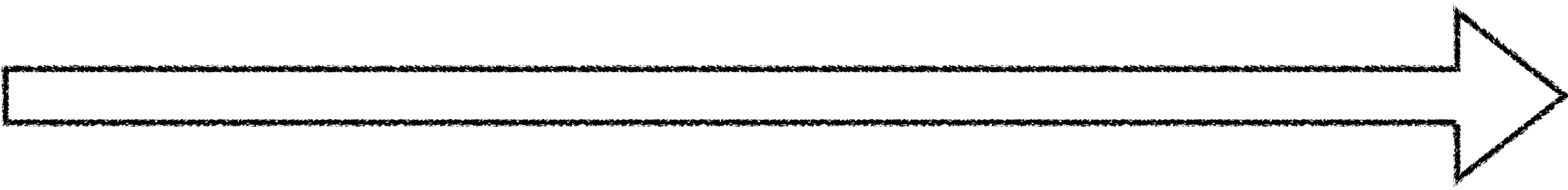
thread1

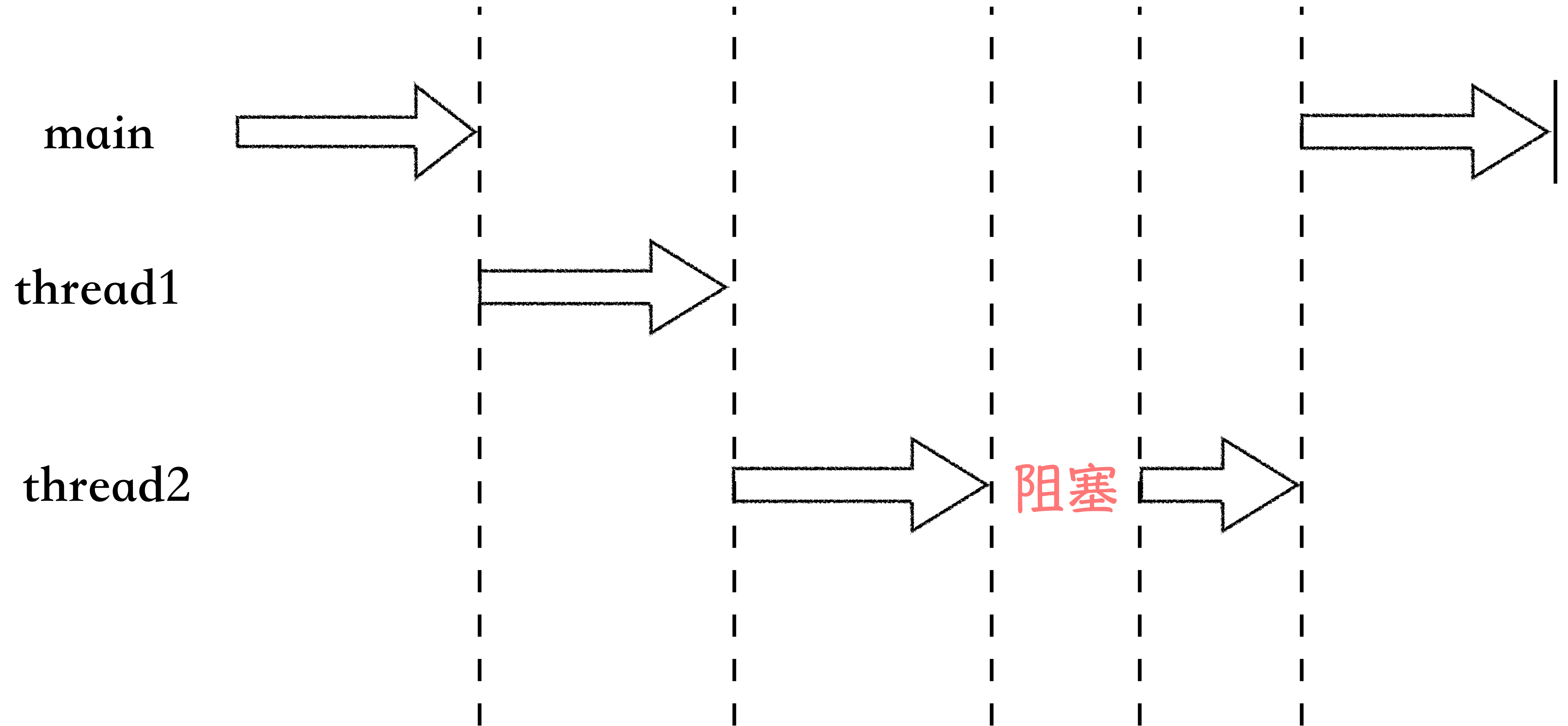


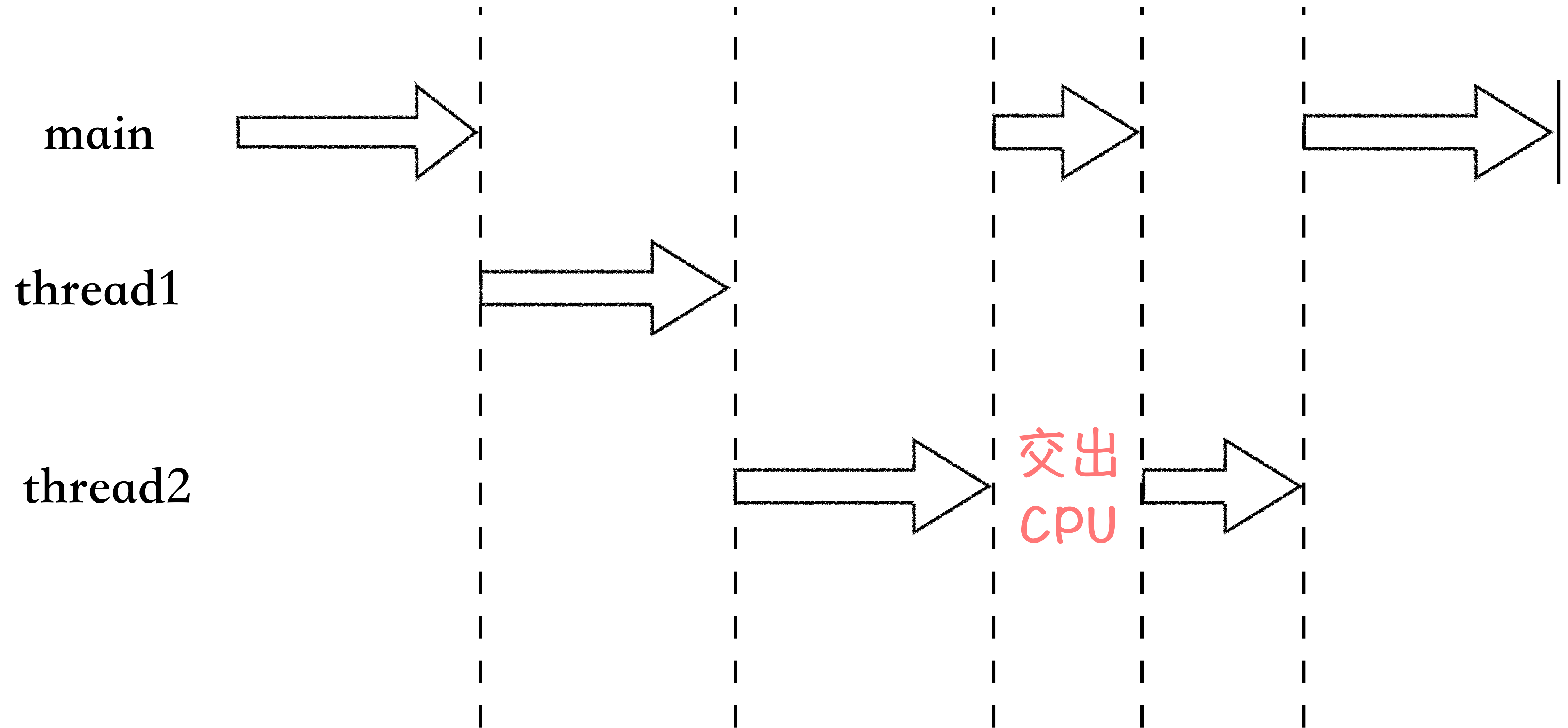
thread2



thread3







通过提升 CPU 利用率，减少线程切换
进而提升程序运行效率

- ⑥ 可控制：协程能做到可被控制的发起子任务
- ⑥ 轻量级：协程非常小，占用资源比线程还少
- ⑥ 语法糖：使多任务或多线程切换不再使用回调语法

通过 Kotlin 在 JVM 平台使用协程

示例：第三方登录

```
fun signInWith() {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```

示例：第三方登录

```
fun signInWith() {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```


示例：第三方登录

```
fun signInWith() {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```

```
fun signInWith() {  
    requestToken { token ->  
        requestUserInfo(token) { user ->  
            setText(user.name)  
        }  
    }  
}
```




```
fun signInWith() {  
    requestToken{ token ->  
        requestUserInfo(token){user ->  
            setText(user.name)  
        }  
    }  
}
```

用协程实现

```
fun signInWith() = runBlocking {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```



```
fun signInWith() = runBlocking {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```

启动协程

- **runBlocking : T** // 用于执行协程任务，
通常只用于启动最外层协程
- **launch : Job** // 用于执行协程任务
- **async/await : Deferred** // 用于执行协程任务，并得到执行结果

```
fun setText(name: String) = launch(UI) {  
    textView.text = name  
}
```

```
suspend fun requestToken() = async(CommonPool) {  
    return@async "curl http://www.example.com"  
}.await()
```



```
suspend fun requestToken() = async(CommonPool) {  
    return@async "curl http://www.example.com"  
}.await()
```

suspend 关键字

suspend 修饰的函数(或 lambda),
只能被 suspend 修饰的函数(或 lambda)调用

suspend

```
suspend fun requestToken(): String { ... }
```

```
Object requestToken(Continuation<String> c) { ... }
```

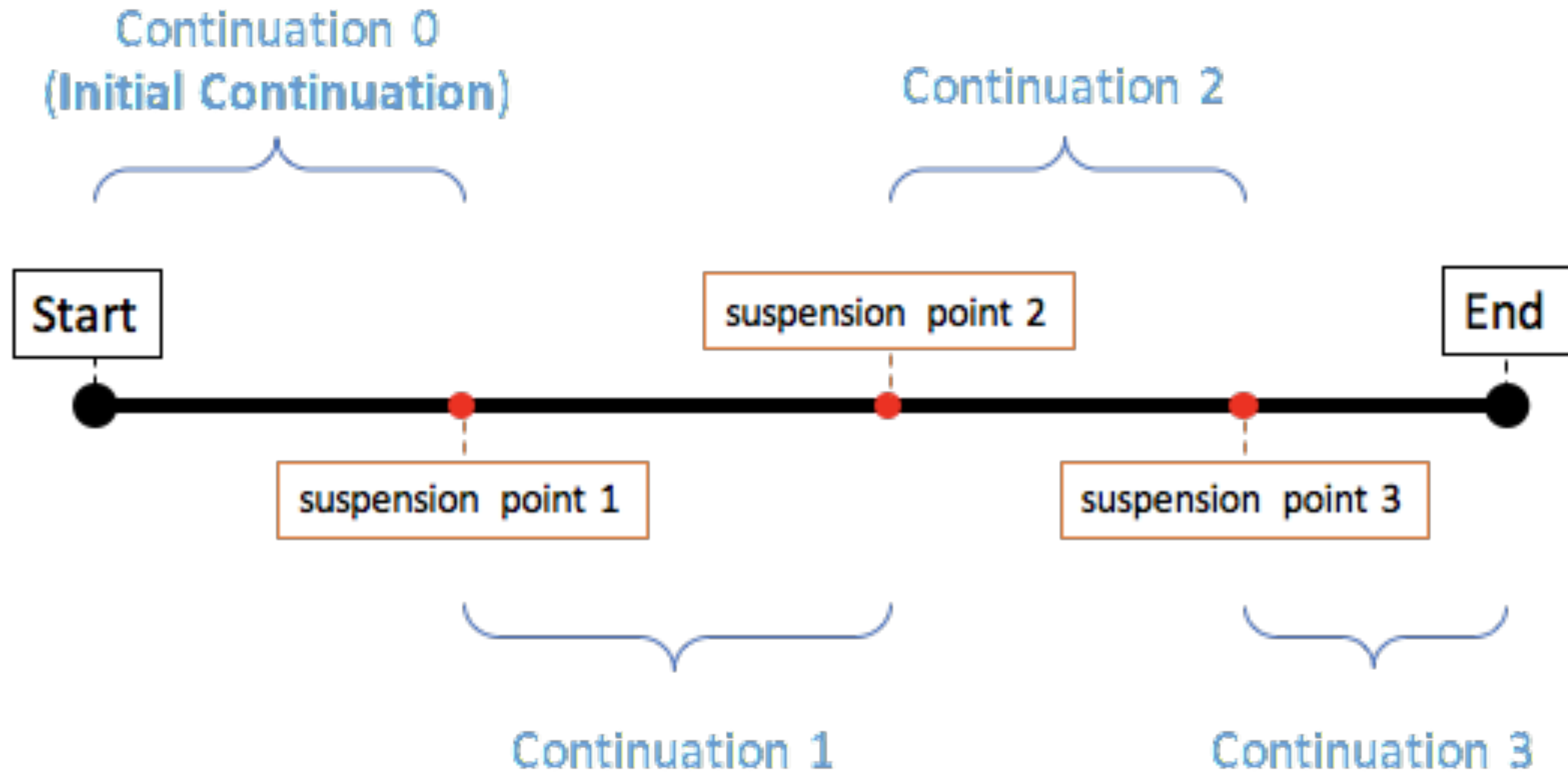
Continuation

```
suspend fun requestToken(): String { ... }
```

```
Object requestToken(Continuation<String> c) { ... }
```

```
interface Continuation<in T> {  
    val context: CoroutineContext  
    fun resume(value: T)  
    fun resumeWithException(e: Throwable)  
}
```

多协程组合: suspension & Continuation



示例：第三方登录

```
fun signInWith() = runBlocking {  
    val token = requestToken()  
    val user = requestUserInfo(token)  
    setText(user.name)  
}
```

协程的切换

```
class Main$signinWith$1 : CoroutineImpl(){  
    override fun doResume(any: Any){  
        switch(this.label){  
            case 0:  
                this.label = 1  
                requestToken(this)  
            case 1:  
                this.label = 2  
                requestUserInfo(token, this)  
            case 2:  
                setText(name)  
        }  
    }  
}
```

在现有项目引入协程

```
interface Service {  
    fun requestTokens(): Call<Token>  
}
```

```
suspend fun requestToken(): String =  
    serviceInterface.requestToken().await()
```


定义扩展方法

```
suspend fun <T> Call<T>.await(): T {  
    ...  
}
```

定义扩展方法

```
suspend fun <T> Call<T>.await() =  
    suspendCoroutine<T> { continuation ->  
  
        enqueue(object : Callback<T> {  
            override fun onResponse(c: Call<T>, r: Response<T>) {  
                continuation.resume(r.body()!!)  
            }  
  
            override fun onFailure(c: Call<T>?, t: Throwable) {  
                continuation.resumeWithException(t)  
            }  
        })  
    }  
}
```


改进 I/O 操作

```
fun save(textView: TextView, file: File) =  
    runBlocking {  
        val text = textView.text.toString()  
        val deferred = async(CommonPool) {  
            file.appendText(text)  
            file.readText()  
        }  
        deferred.await()  
    }  
}
```

目标方向

- ⑤ 可控制：协程能做到可被控制的发起子任务
- ⑤ 轻量级：协程非常小，占用资源比线程还少
- ⑤ 语法糖：使多线程切换不再使用回调语法

Thanks