

# Having fun with Kotlin coroutines

A first tour of concurrency models in Kotlin

---

Alessandro Candolini

June 27, 2018

# Agenda

1. We live in a concurrent world
2. Blocking vs non-blocking
3. Demystifying coroutines
4. Coroutines-powered concurrency models

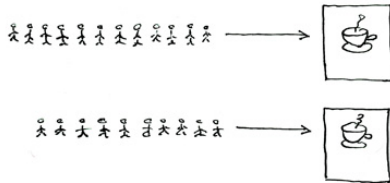
**We live in a concurrent world**

---

Concurrent = Two Queues One Coffee Machine



Parallel = Two Queues Two Coffee Machines



© Joe Armstrong 2013

**Figure 1:** <https://joearms.github.io/published/2013-04-05-concurrent-and-parallel-programming.html>

Is concurrency relevant for mobile development?

- IO (e. g., network, etc)
- sensors (e. g., gps, etc)
- UI events
- platform lifecycle

Is concurrency relevant for mobile development?

- IO (e. g., network, etc)
- sensors (e. g., gps, etc)
- UI events
- platform lifecycle

Is concurrency relevant for mobile development?

- IO (e. g., network, etc)
- sensors (e. g., gps, etc)
- UI events
- platform lifecycle

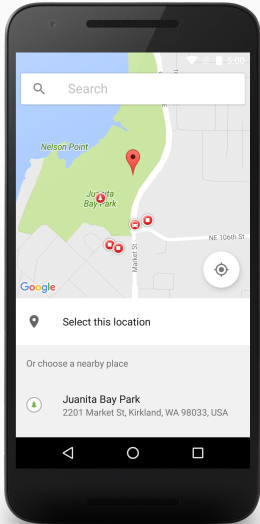
Is concurrency relevant for mobile development?

- IO (e. g., network, etc)
- sensors (e. g., gps, etc)
- UI events
- platform lifecycle



Is concurrency relevant for mobile development?

- IO (e. g., network, etc)
- sensors (e. g., gps, etc)
- UI events
- platform lifecycle



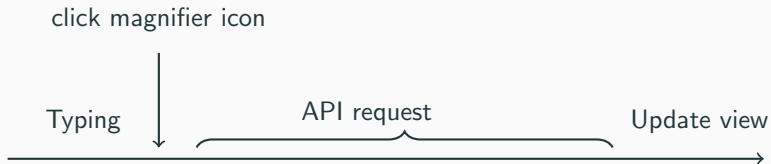
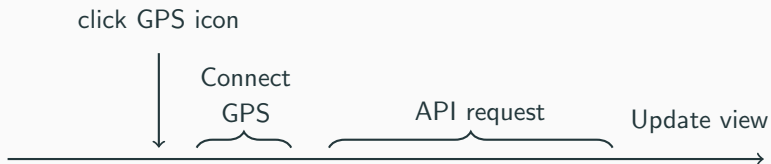
acceptance criteria:

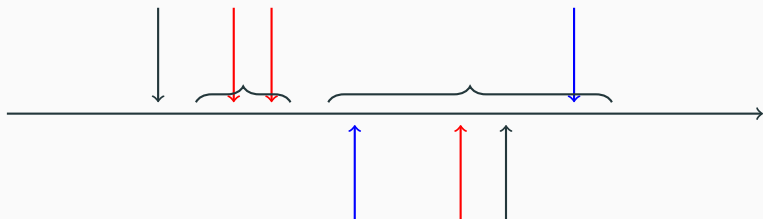
- search by current location
- search by location name

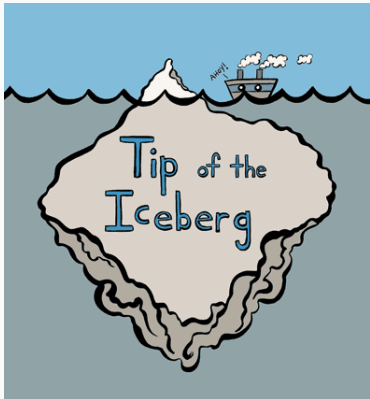
advanced

- search suggestions when typing

Translate ACs into code: simple *sequential* state machine (simplified)







- Delays
- User inputs
- Failures
- Ordering of api response
- retry logic
- debouncing
- timeouts
- caching
- *android lifecycle*, etc

First approach: put constraints in place to restrict the range of possible options

- Conditionally forbid user events (disable buttons, loading spinners, etc)
- Boolean flags
- Be defensive (if/else)
- Bind/unbind from lifecycle , etc

(Or more technical constraints like single thread executors, queues, synchronization, etc)

“Concurrency is the composition of independently executing processes, typically functions, but they don’t have to be.” “Parallelism is the simultaneous execution of multiple things, possibly related, possibly not.”

Rob Pike



Rob Pike - 'Concurrency Is Not Parallelism'

**Figure 2:** [https://www.youtube.com/watch?v=cN\\_DpYBzKso&t=1061s](https://www.youtube.com/watch?v=cN_DpYBzKso&t=1061s)



## Communicating Sequential Processes

C.A.R. Hoare  
The Queen's University  
Belfast, Northern Ireland

This paper suggests that input and output are basic primitives of programming and that parallel composition of communicating sequential processes is a fundamental program structuring method. When combined with a development of Dijkstra's guarded command, these concepts are surprisingly versatile. Their use is illustrated by sample solutions of a variety of familiar programming exercises.

**Key Words and Phrases:** programming, programming languages, programming primitives, program structures, parallel programming, concurrency, input, output, guarded commands, nondeterminacy, coroutines, procedures, multiple entries, multiple exits, classes, data representations, recursion, conditional critical regions, monitors, iterative arrays  
CR Categories: 4.28, 4.22, 4.32

### 1. Introduction

Among the primitive concepts of computer programming, and of the high level languages in which programs are expressed, the action of assignment is familiar and well understood. In fact, any change of the internal state of a machine executing a program can be modeled as an assignment of a new value to some variable part of that machine. However, the operations of input and output, which affect the external environment of a machine, are not nearly so well understood. They are often added to a programming language only as an afterthought.

Among the structuring methods for computer programs, general permission to make fair use in teaching or research of all or part of the material is granted to individual readers and to nonprofit libraries acting for them provided that ACM's copyright notice is given and that reference is made to the publication, or, in the case of time, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery. To otherwise reprint a figure, table, other substantial passage, or the entire work requires specific permission as does republication, or systematic or multiple reproduction.

This research was supported by a Senior Fellowship of the Science Research Council.

Author's present address: Programming Research Group, 45, Banbury Road, Oxford, England.  
© 1978 ACM 0001-0782/78/0000-0066 \$00.75

066

grams, three basic constructs have received widespread recognition and use: A repetitive construct (e.g. the **while** loop), an alternative construct (e.g. the conditional **if-then-else**), and normal sequential program composition (often denoted by a semicolon). Less agreement has been reached about the design of other important program structures, and many suggestions have been made: Subroutines (FORTRAN), procedures (ALGOL 60 [15]), entries (PL/I), coroutines (UNIX [17]), classes (SIMULA 67 [5]), processes and monitors (Concurrent Pascal [2]), clusters (CLU [13]), forms (ALPACADO [19]), actors (Hewitt [1]).

The traditional stored program digital computer has been designed primarily for deterministic execution of a single sequential program. Where the desire for greater speed has led to the introduction of parallelism, every attempt has been made to disguise this fact from the programmer, either by hardware itself (as in the multiple function units of the CDC 6600) or by the software (as in an I/O control package, or a multiprogrammed operating system). However, developments of processor technology suggest that a multiprocessor machine, constructed from a number of similar self-contained processors (each with its own store), may become more powerful, capacious, reliable, and economical than a machine which is disguised as a monoprocessor.

In order to use such a machine effectively on a single task, the component processors must be able to communicate and to synchronize with each other. Many methods of achieving this have been proposed. A widely adopted method of communication is by inspection and updating of a common store (as in ALGOL 68 [18], PL/I, and many machine codes). However, this can create severe problems in the construction of correct programs and it may lead to expense (e.g. crossbar switches) and unreliability (e.g. glitches) in some technologies of hardware implementation. A greater variety of methods has been proposed for synchronization: semaphores [6], events (PL/I), conditional critical regions [16], monitors and queues (Concurrent Pascal [2]), and path expressions [3]. Most of these are demonstrably adequate for their purpose, but there is no widely recognized criterion for choosing between them.

This paper makes an ambitious attempt to find a single simple solution to all these problems. The essential proposals are:

- (1) Dijkstra's guarded commands [8] are adopted (with a slight change of notation) as sequential control structures, and as the sole means of introducing and controlling nondeterminism.
- (2) A parallel command, based on Dijkstra's *parbegin* [6], specifies concurrent execution of its constituent sequential commands (processes). All the processes start simultaneously, and the parallel command ends only when they are all finished. They may not communicate with each other by updating global variables.
- (3) Simple forms of input and output command are introduced. They are used for communication between concurrent processes.

Communications  
of  
the ACM

August 1978  
Volume 21  
Number 4

Figure 3: Tony Hoare's seminal paper

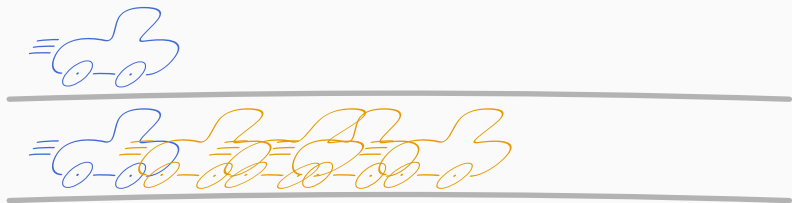
“The most obvious application of the new ideas is to the specification, design, and implementation of computer systems which continuously act and interact with their environment. The basic idea is that these systems can be readily decomposed into subsystems which operate concurrently and interact with each other as well as with their common environment. The parallel composition of subsystems is as simple as the sequential composition of lines or statements in a conventional programming language.”

Tony Hoare (CSP book, 2015)

Java definition of concurrency and Leslie Lamport seminal paper

Traditional picture of concurrency vs parallelism: coffe machine

Better representation: driving in the desert vs driving in London



The deep problem: *communication*

Communicating, orchestrating independent processing. Software development as a *dialog* between parts.

We will just sketch the surface of this deeper problem by focusing only on a smaller technical problem in this talk: *blocking* and *non wasting unnecessary resources when waiting*

# Blocking vs non-blocking

---

Test



# Demystifying coroutines

---

What rae

## Coroutines-powered concurrency models

---

- CSP (aka, channels)
- actors

**Questions?**