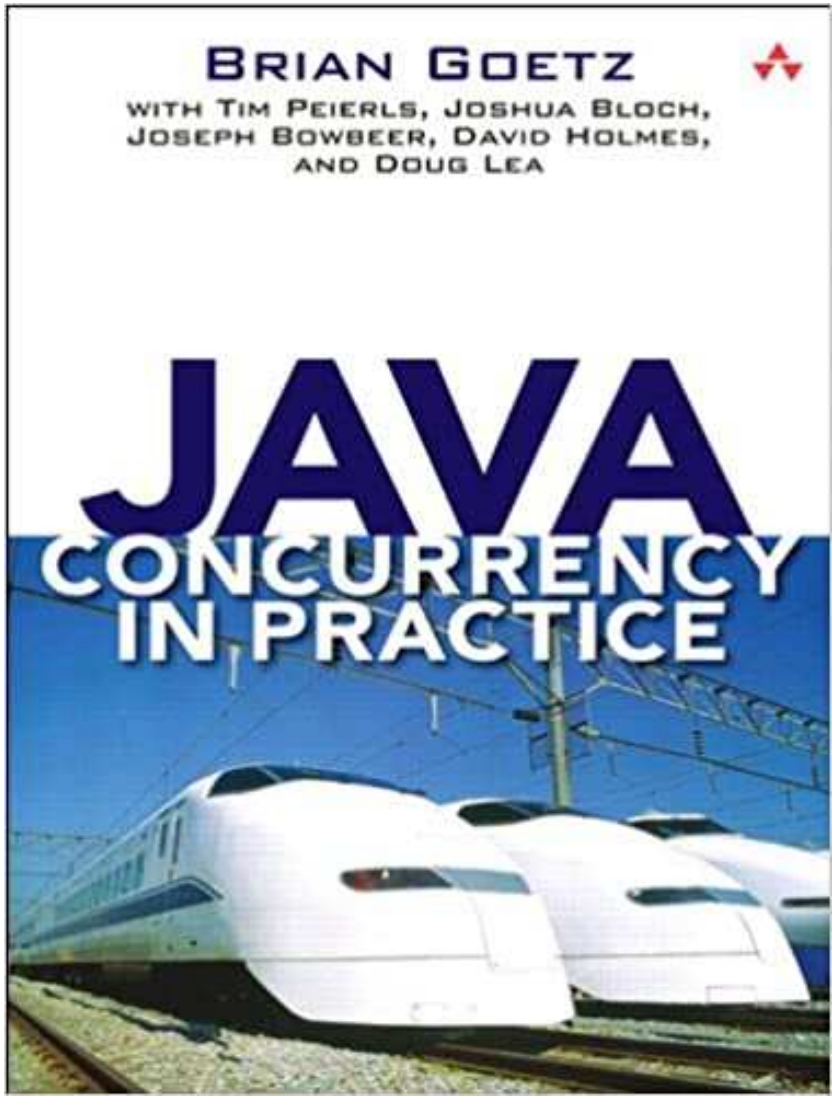
OpenJDK project Loom

Vojtech Juranek

oVirt storage team

2. 3. 2021

Java concurrency



Java concurrency

Low-level thread: `java.lang.Threads` class and `java.lang.Runnable`.

```
1 Runnable task = new MyRunnable();
2 Thread thread = new Thread(task);
3 thread.start();
```

Java concurrency

High-level Executor framework:

```
1 Runnable task = new MyRunnable();
2 ExecutorService executor = Executors.
     newFixedThreadPool(NUM_THREADS);
3 executor.execute(task);
```

```
1 Runnable task = new MyRunnable();
2 ExecutorService executor = Executors.
     newFixedThreadPool(NUM_THREADS);
3 Future<?> f = executor.submit(task);
```

Java threads

- 1:1 mapping to OS threads.
- Memory heavy (MBs).
- Context switching is time consuming.
- Creation is expensive.

Thread pooling

- Doesn't solve memory overhead and context switching, just creating overhead.
- Issues with `ThreadLocal` and thread interrupts.
- Mitigate context switching $\rightarrow$ core-per-thread architecture.

Asynchronous programming

- Breaking tasks into smaller ones.
- Better CPU utilization, as blocking tasks (typically IO) are waiting in a queue instead blocking thread.
- Chaining usually via callbacks.

Java concurrency

Async. constructs added in Java 8 (e.g. CompletableFuture):

```
1 public String myFunction(int a) {...}
2 CompletableFuture<String> cf =
     CompletableFuture.supplyAsync(() ->
     myFunction(10));
3 cf.thenAccept(result -> System.out.println(
     result));
```

And many other ways how to run function asynchronously, e.g.:

```
1 public String myFunction(int a) {...}
2 Stream.of(1, 2).parallel().forEach(i ->
     myFunction(i));
```

Asynchronous frameworks

- vert.x
- Netty
- ...

Vert.x example:

```
1 vertx.createHttpServer().requestHandler(r -> {
2   r.response()
3     .putHeader("content-type", "text/plain")
4     .end("Hello from Vert.x!");
5 }).listen(8080);
```

Handling error in async. code

Netty example:

```
1 ChannelFuture f = ctx.writeAndFlush(b);
2 f.addListener(writeFailed);
3
4 private final ChannelFutureListener writeFailed
      = (ChannelFuture future) -> {
5     if (!future.isSuccess()) {
6         future.cause().printStackTrace();
7         future.channel().close();
8     }
9 };
```

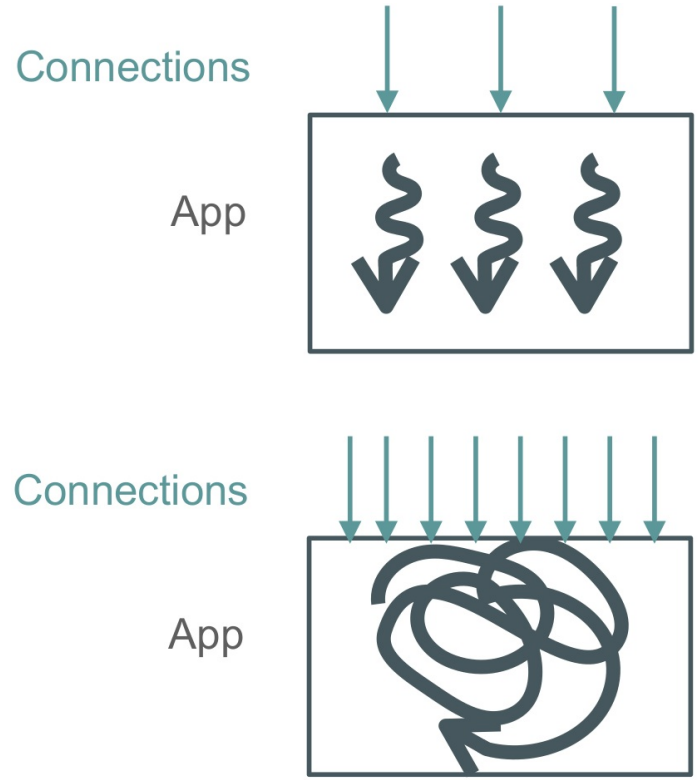
Synchronous vs. asynchronous

Synchronous:

- Simple to reason about it.
- Blocking.
- Less scalable.

Asynchronous:

- Better scales.
- Harder to reason about it.
- “Callback hell”.
- To get maximum benefit from it, all parts have to be asynchronous.
- Can be problematic to make it working with legacy synchronous code.

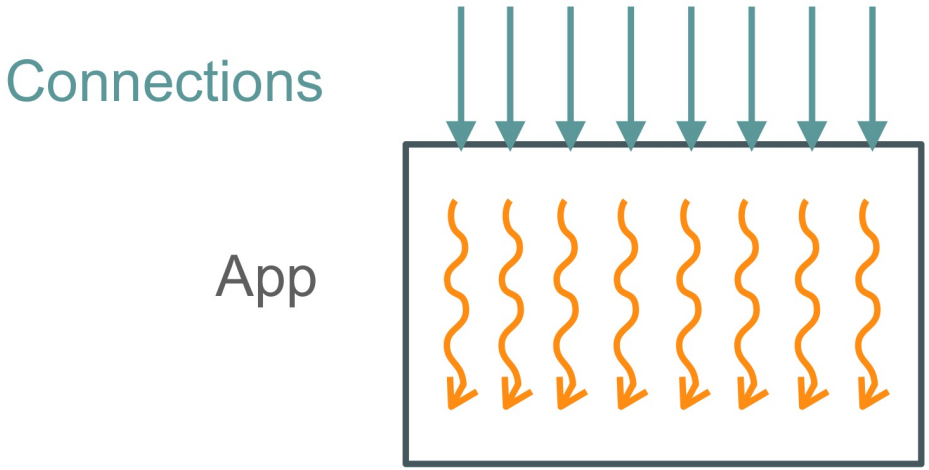


Source:
`http://cr.openjdk.java.net/~alanb/loom/Devoxx201`

Java threads

What about having positives from both approaches:

- Code like synchronous code.
- Work like asynchronous code.

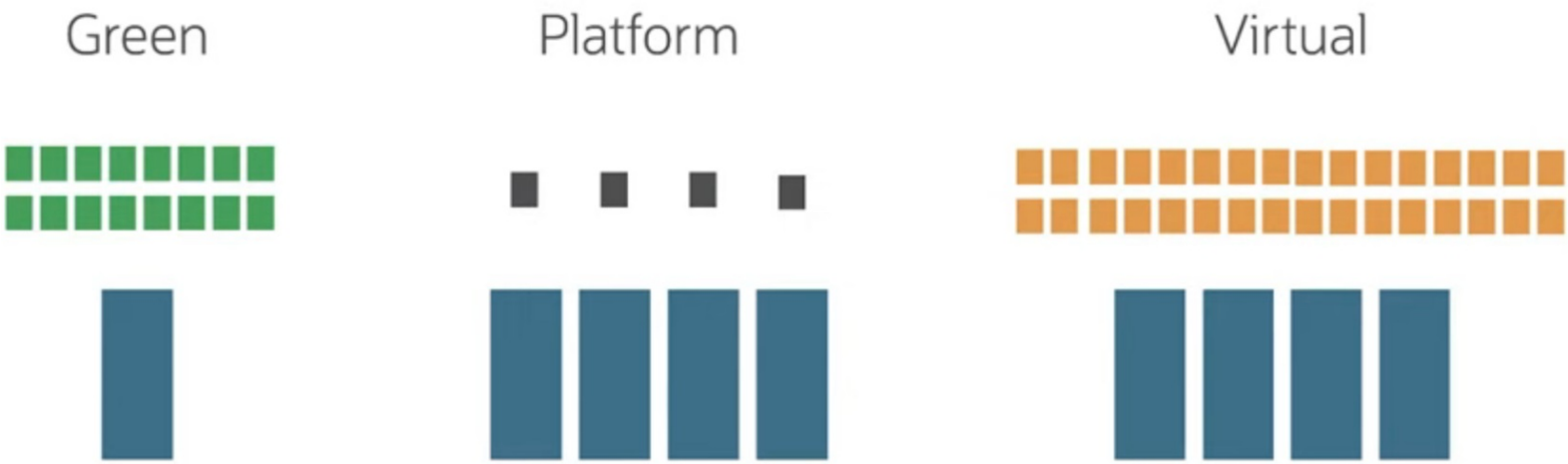


Source: `http://cr.openjdk.java.net/~alanb/loom/Devoxx2018.pdf`

Project Loom

- `https://wiki.openjdk.java.net/display/loom/Main`
- Virtual threads
- Structured concurrency

Virtual threads



Source: `https://2020.accento.dev/talks/project-loom/`

Virtual threads

- Not bound to kernel thread (blocking in virtual thread doesn't block kernel thread).
- Cheap to create.
- Very low memory overhead.
- Future compatible (older code can benefit from new features).

```
1 var thread = Thread.startVirtualThread(() ->
    System.out.println("Virtual thread!"));
2 thread.join();
```

Virtual threads

```
1 Thread thread = Thread.startVirtualThread(
    runnable);
```

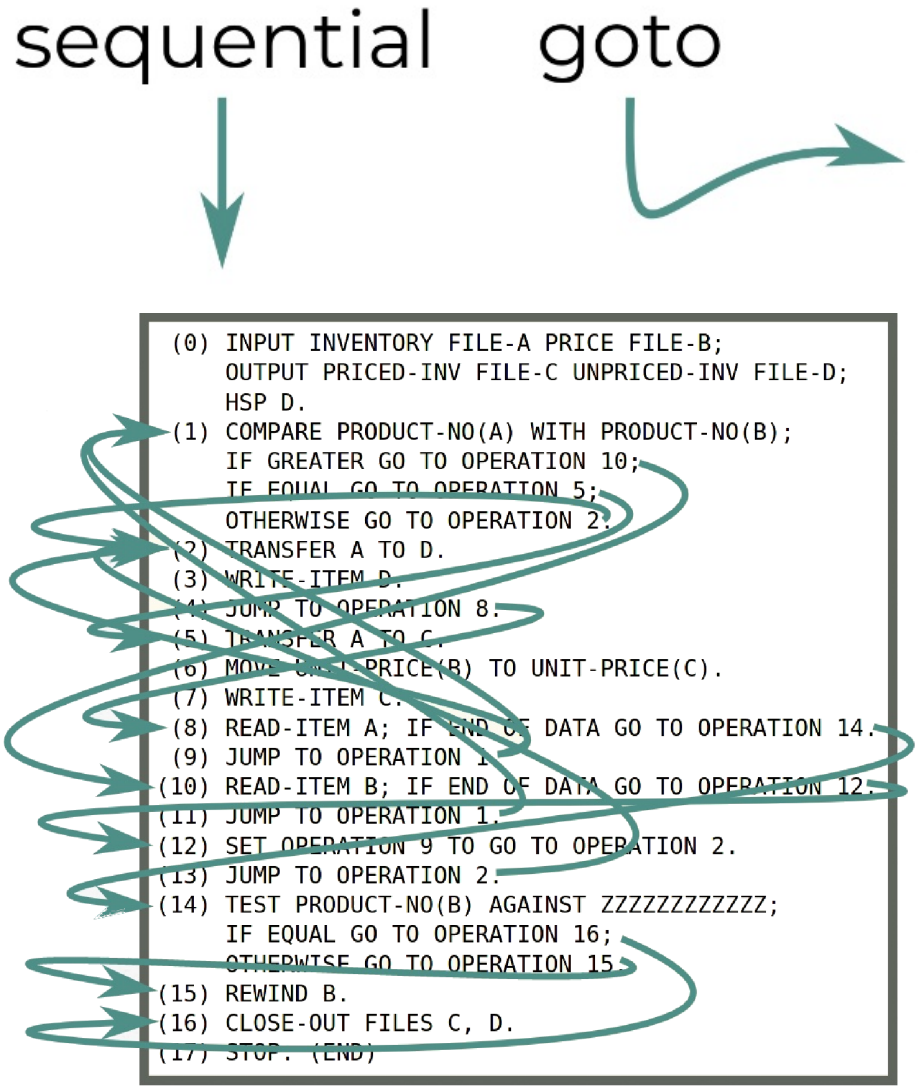
```
1 Thread thread = Thread.builder()
2     .virtual()
3     .name(taskname)
4     .task(runnable)
5     .build();
```

Virtual threads with executor service

```
1 ExecutorService exec = Executors.
    newVirtualThreadExecutor();
2 exec.submit(runable1)
3 exec.submit(runable2)
4 ...
```

```
1 ThreadFactory factory = Thread.builder()
2     .virtual()
3     .name(taskname)
4     .task(runnable)
5     .factory();
6
7 ExecutorService exec = Executors.
    newThreadExecutor(factory);
```

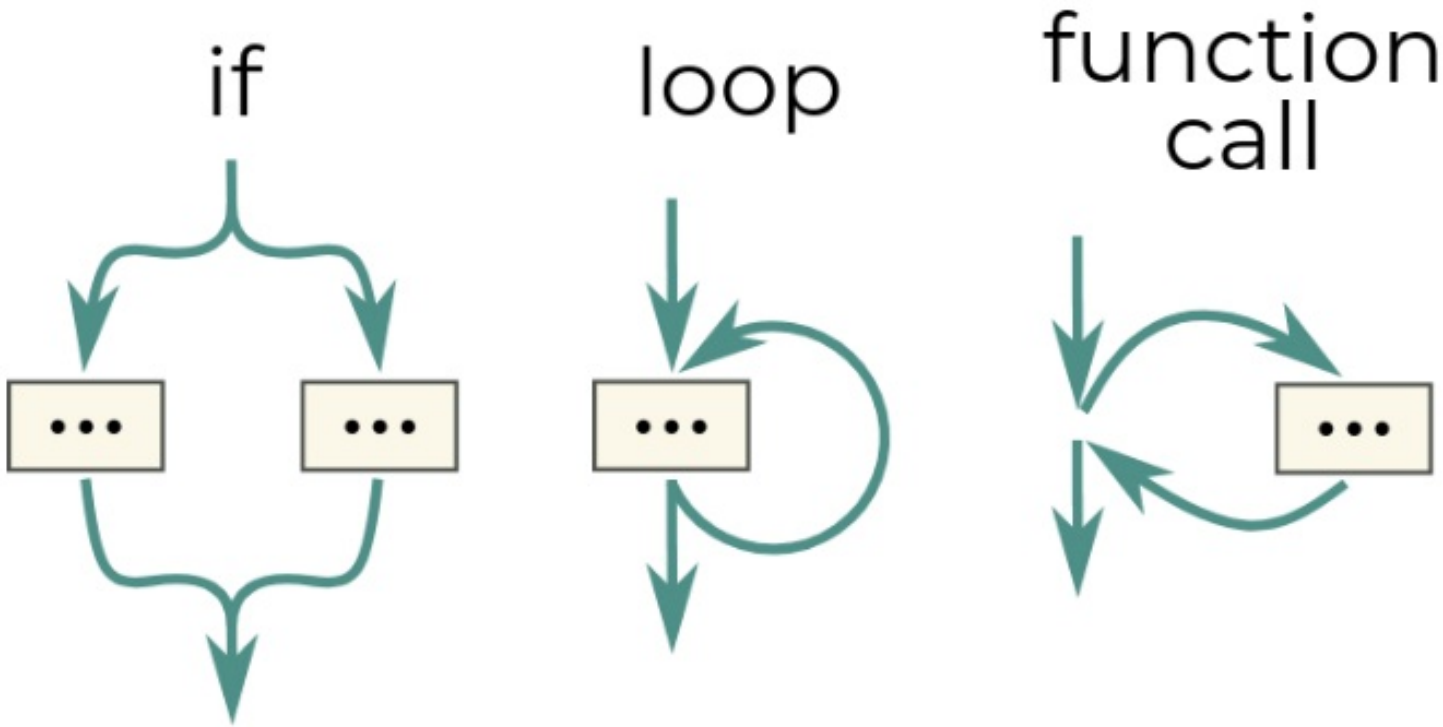
Structured programming



Source: `https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/`

Structured programming

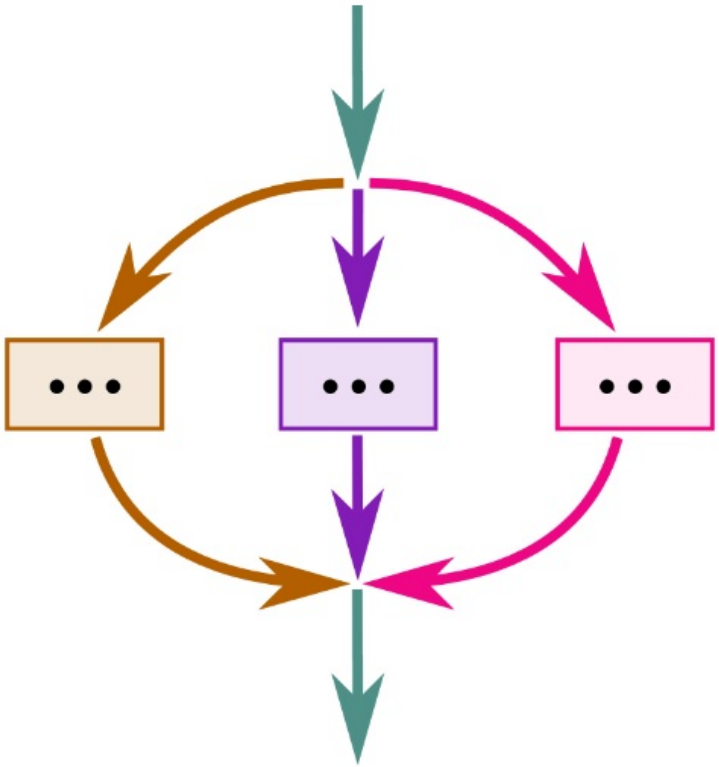
Replacing `goto` with `if-else`, `loop` statements and function calls in 60s:



Source: `https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/`

Structured concurrency

- Launching a task in a thread is like using `goto`.
- Structured concurrency is similar to using `if-else` and `loop` statements instead of `goto`.



Source: `https://vorpus.org/blog/notes-on-structured-concurrency-or-go-statement-considered-harmful/`

Structured concurrency

```
1 try (ExecutorService exec = Executor.
     newVirtualThreadExecutor()) {
2         exec.submit(runable1);
3         exec.submit(runable2);
4         exec.submit(runable3);
5 }
```

Structured concurrency: deadlines

```
1 ThreadFactory factory = Thread.builder().
     virtual().factory();
2 try (var executor =
3     Executors.newThreadExecutor(factory)
4     .withDeadline(Instant.now().plusSeconds(30)
        )) {
5     executor.submit(task1);
6     executor.submit(task2);
7 }
```

There is more

- Structured cancelations.
- Scope variables.
- Thread local.
- . . .

Beware: nothing is set in the store yet, still under development.

Links

- http://jdk.java.net/loom/
- https://wiki.openjdk.java.net/display/loom/Main
- https://cr.openjdk.java.net/~rpressler/loom/Loom-Proposal.html
- http://cr.openjdk.java.net/~rpressler/loom/loom/sol1_part1.html
- https://cr.openjdk.java.net/~rpressler/loom/loom/sol1_part2.html
- https://www.javaadvent.com/2020/12/project-loom-and-structured-concurrency.html
- https://blogs.oracle.com/javamagazine/going-inside-javas-project-loom-and-virtu
- https://blog.softwaremill.com/will-project-loom-obliterate-java-futures-fb1a285
- https://www.youtube.com/watch?v=fOEPEXTpbJA
- https://www.youtube.com/watch?v=23HjZBOIshY
- https://www.youtube.com/watch?v=_fFzyY_7UmA
- http://belaban.blogspot.com/2020/07/double-your-performance-virtual-threads.htm

Questions?

Thank you!

Questions?