Understanding the Pros & Cons of Synchrony

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Motivate the need for Java Future & CompletableFuture mechanisms by understanding the pros & cons of synchrony
Overview of Synchrony & Synchronous Operations
Method calls in typical Java programs are largely *synchronous*.

Example: 
- `searchForWord_1`
- `return result_1`
- `searchForWord_2`
- `return result_2`
- `searchForWord_3`
- `return return_3`

Example calls on Java collections & behaviors in Java stream aggregate operations
Overview of Synchrony & Synchronous Operations

- Method calls in typical Java programs are largely *synchronous*
- i.e., a callee borrows the thread of its caller until its computation(s) finish
Overview of Synchrony & Synchronous Operations

- Method calls in typical Java programs are largely *synchronous*
  - i.e., a callee borrows the thread of its caller until its computation(s) finish & a result is returned

![Diagram showing method calls and synchronization](Diagram)

*Note “request/response” nature of these calls*

The Pros of Synchrony
The Pros of Synchrony

• Pros of synchronous calls
The Pros of Synchrony

- Pros of synchronous calls
- “Intuitive” to program & debug

CALLER

searchForWord\textsubscript{1}  
return result\textsubscript{1}

searchForWord\textsubscript{2}  
return result\textsubscript{2}

searchForWord\textsubscript{3}  
return return\textsubscript{3}

CALLEE
The Pros of Synchrony

- Pros of synchronous calls
  - “Intuitive” to program & debug, e.g.
  - Maps onto common two-way method patterns

![TWO WAY](image)

CALLER

searchForWord$_1$

return result$_1$

searchForWord$_2$

return result$_2$

searchForWord$_3$

return return$_3$

CALLEE

See [www.iro.umontreal.ca/~keller/Layla/remote.pdf](www.iro.umontreal.ca/~keller/Layla/remote.pdf)
The Pros of Synchrony

- Pros of synchronous calls
  - "Intuitive" to program & debug, e.g.
  - Maps onto common two-way method patterns
  - Local caller state retained when callee returns

See wiki.c2.com/?ActivationRecord
The Pros of Synchrony

- Pros of synchronous calls
- “Intuitive” to program & debug, e.g.
  - Maps onto common two-way method patterns
- Local caller state retained when callee returns

```java
byte[] downloadContent(URL url) {
    byte[] buf = new byte[BUFSIZ];
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    try (InputStream is = url.openStream()) {
        for (int bytes;
             (bytes = is.read(buf)) > 0;)
            os.write(buf, 0, bytes);
    ...}
```

See Java8/ex20/src/main/java/utils/FileAndNetUtils.java
The Cons of Synchrony
The Cons of Synchrony

• Cons of synchronous calls
The Cons of Synchrony

- Cons of synchronous calls
- May not leverage all parallelism available in multi-core systems

See mincong.io/2020/06/26/completable-future
The Cons of Synchrony

- Cons of synchronous calls
  - May not leverage all parallelism available in multi-core systems
  - Blocking threads incur overhead
    - e.g., synchronization, context switching, data movement, & memory management costs
The Cons of Synchrony

• Cons of synchronous calls
• May not leverage all parallelism available in multi-core systems
  • Blocking threads incur overhead
• Selecting right # of threads is hard

```java
List<Image> filteredImages = urls
    .parallelStream()
    .filter(not(this::urlCached))
    .map(this::downloadImage)
    .map(this::applyFilters)
    .reduce(Stream::concat)
    .orElse(Stream.empty())
    .collect(toList());
```

```
Image downloadImage(URL url){
    return new Image
        (url,
         downloadContent
         (url));
}
```

The Cons of Synchrony

- Cons of synchronous calls
- May not leverage all parallelism available in multi-core systems
  - Blocking threads incur overhead
  - Selecting right # of threads is hard

List<Image> filteredImages = urls
  .parallelStream()
  .filter(not(this::urlCached))
  .map(this::downloadImage)
  .map(this::applyFilters)
  .reduce(Stream::concat)
  .orElse(Stream.empty())
  .collect(toList());

A large # of threads may help to improve performance, but can also waste resources
The Cons of Synchrony

- Cons of synchronous calls
- May not leverage all parallelism available in multi-core systems
  - Blocking threads incur overhead
- Selecting right # of threads is hard

```java
List<Image> filteredImages = urls
    .parallelStream()
    .filter(not(this::urlCached))
    .map(this::downloadImage)
    .map(this::applyFilters)
    .reduce(Stream::concat)
    .orElse(Stream.empty())
    .collect(toList());
```

A small # of threads may conserve resources at the cost of performance
The Cons of Synchrony

• Cons of synchronous calls
• May not leverage all parallelism available in multi-core systems
  • Blocking threads incur overhead
  • Selecting right # of threads is hard

Efficient Performance
Efficient Resource Utilization

Particularly tricky for I/O-bound programs that need more threads to run efficiently
The Cons of Synchrony

• Cons of synchronous calls
  • May not leverage all parallelism available in multi-core systems
  • May need to change the size of the common fork-join pool

See lesson on “The Java Fork-Join Pool: Maximizing Core Utilization w/the Common Fork-Join Pool”
The Cons of Synchrony

- Cons of synchronous calls
  - May not leverage all parallelism available in multi-core systems
  - May need to change the size of the common fork-join pool, e.g.
  - Set a system property

```java
String desiredThreads = "10";
System.setProperty(
  "java.util.concurrent.ForkJoinPool.common.parallelism",
  desiredThreads);
```

It’s hard to estimate the total # of threads to set in the common fork-join pool.
The Cons of Synchrony

- Cons of synchronous calls
  - May not leverage all parallelism available in multi-core systems
  - May need to change the size of the common fork-join pool, e.g.
    - Set a system property
    - Or use the ManagedBlocker to increase common pool size automatically/temporarily

ManageBlockers can only be used with the common fork-join pool..

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.ManagedBlocker.html
End of Understanding the Pros & Cons of Synchrony