Comparing Java Sequential Streams with Parallel Streams

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

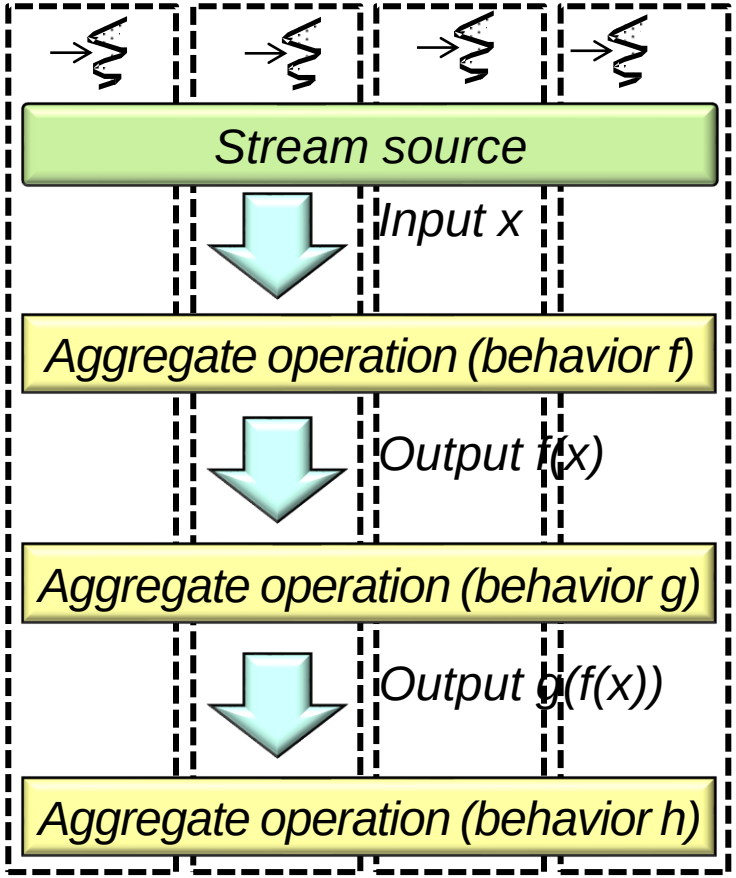
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

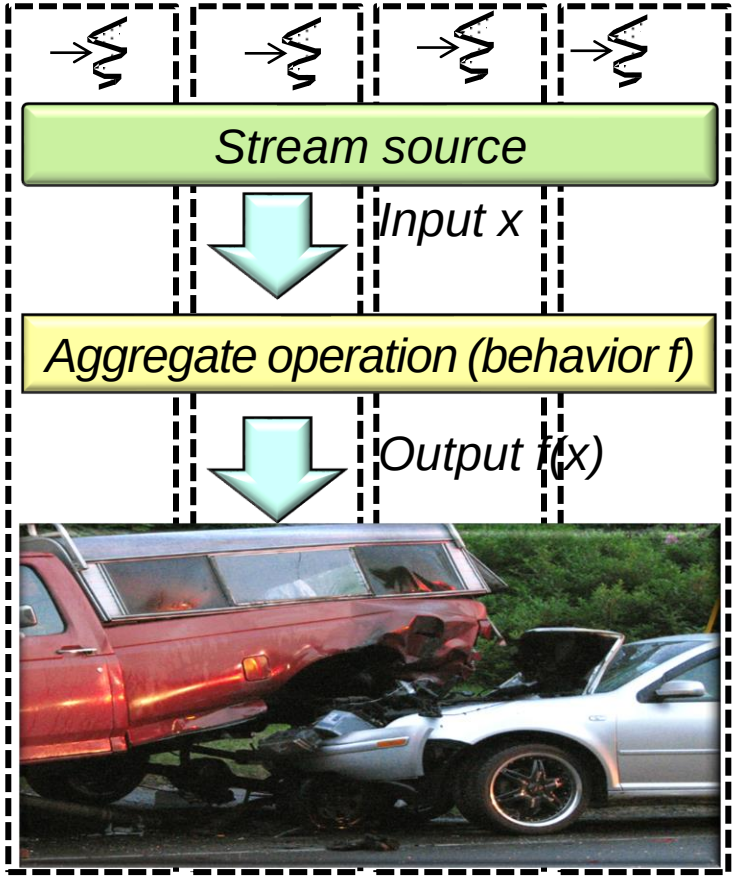
- Understand the structure & functionality of Java streams, e.g.,
  - Fundamentals of streams
  - Benefits of streams
  - Creating a stream
  - Aggregate operations in a stream
  - Applying streams in practice
  - Sequential vs. parallel streams



See radar.oreilly.com/2015/02/java-8-streams-api-and-parallelism.html

Learning Objectives in this Part of the Lesson

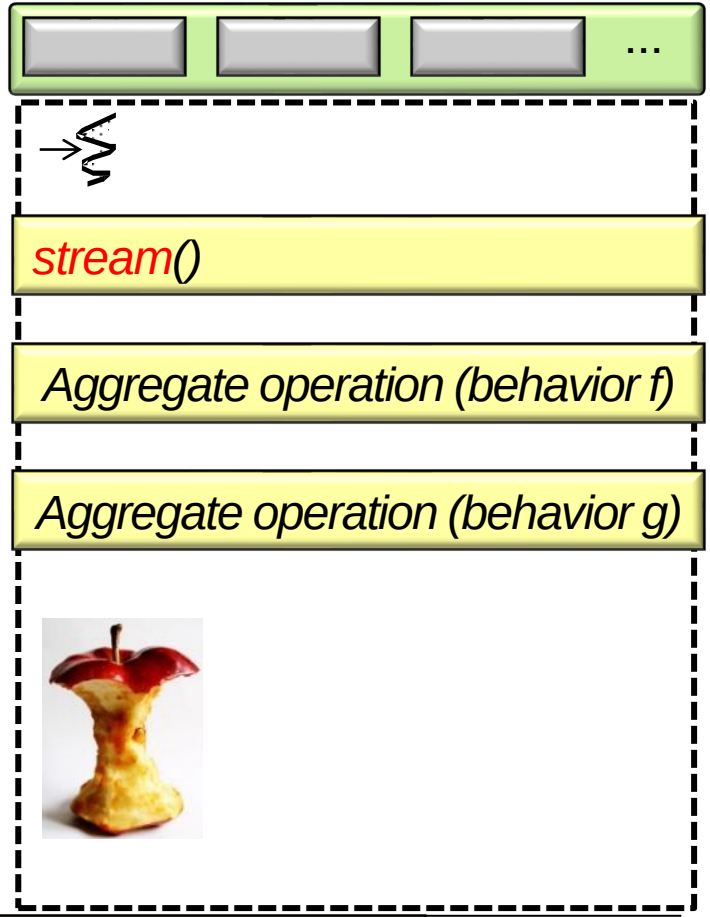
- Understand the structure & functionality of Java streams, e.g.,
  - Fundamentals of streams
  - Benefits of streams
  - Creating a stream
  - Aggregate operations in a stream
  - Applying streams in practice
  - Sequential vs. parallel streams
    - Common programming hazards of parallel streams



Comparing Sequential vs. Parallel Streams

Comparing Sequential vs. Parallel Streams

- Stream operations run sequentially

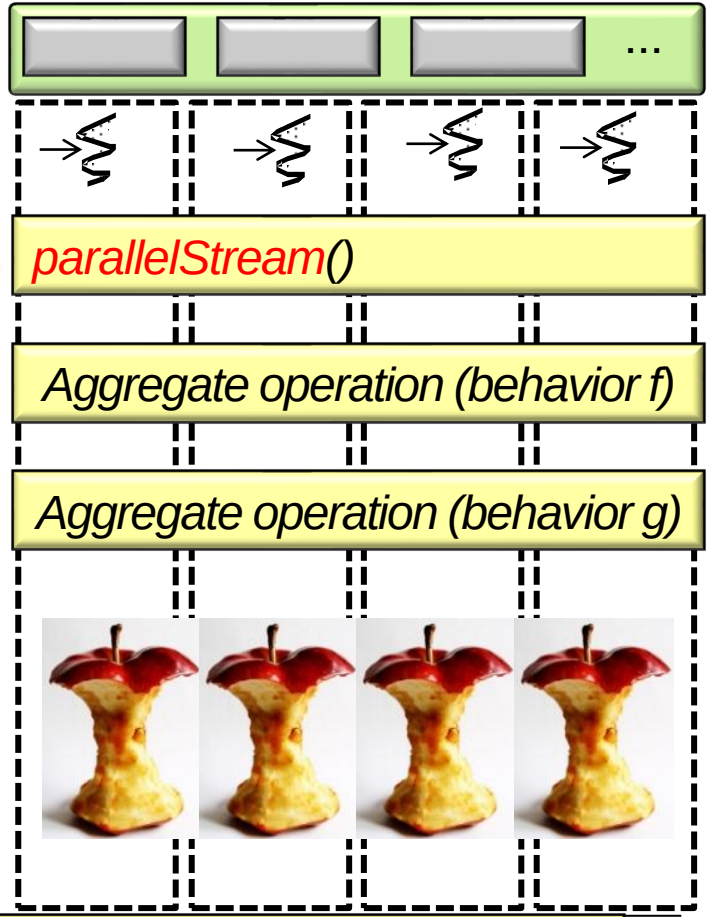


We'll cover sequential streams first

See docs.oracle.com/javase/tutorial/collections/streams

Comparing Sequential vs. Parallel Streams

- Stream operations run sequentially or in parallel

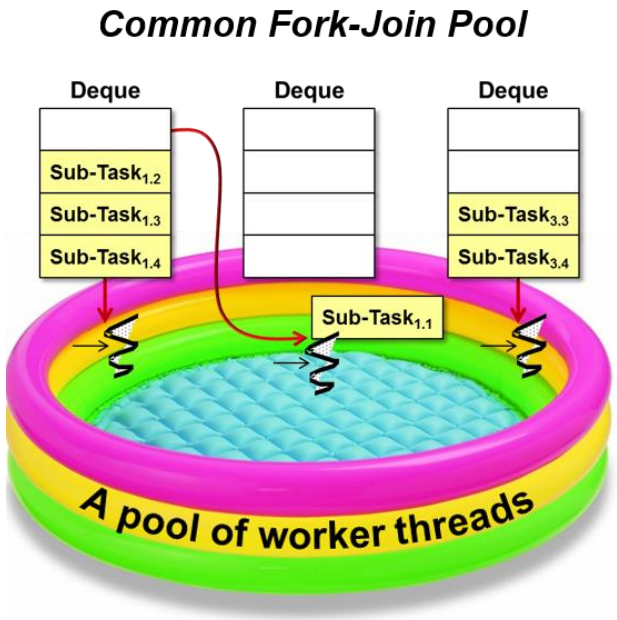


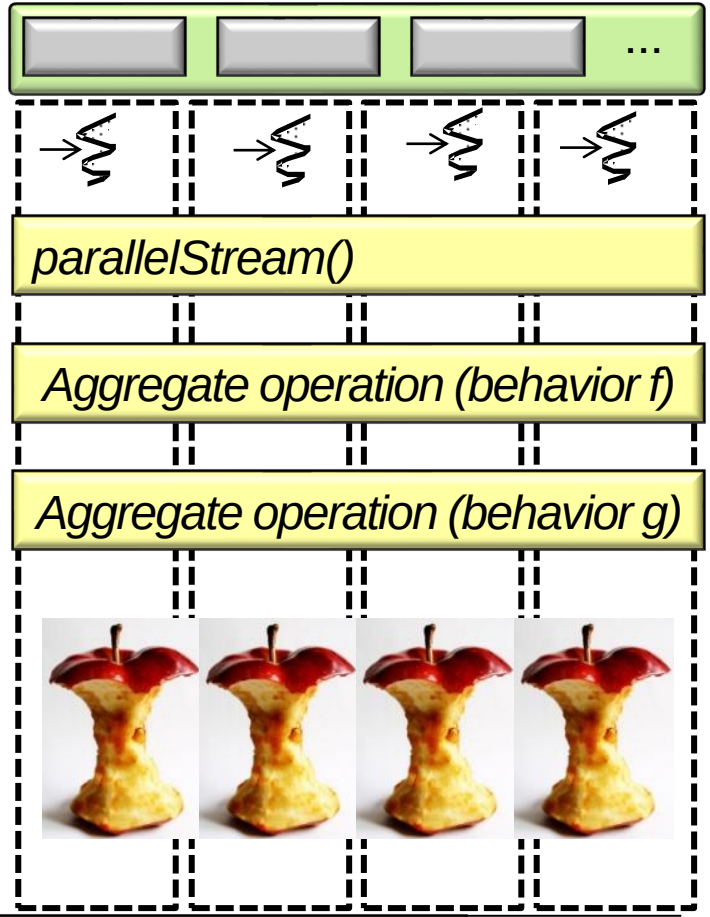
We'll cover parallel streams later

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

Comparing Sequential vs. Parallel Streams

- A parallel stream splits its elements into multiple chunks & uses the common fork-join pool to process these chunks independently



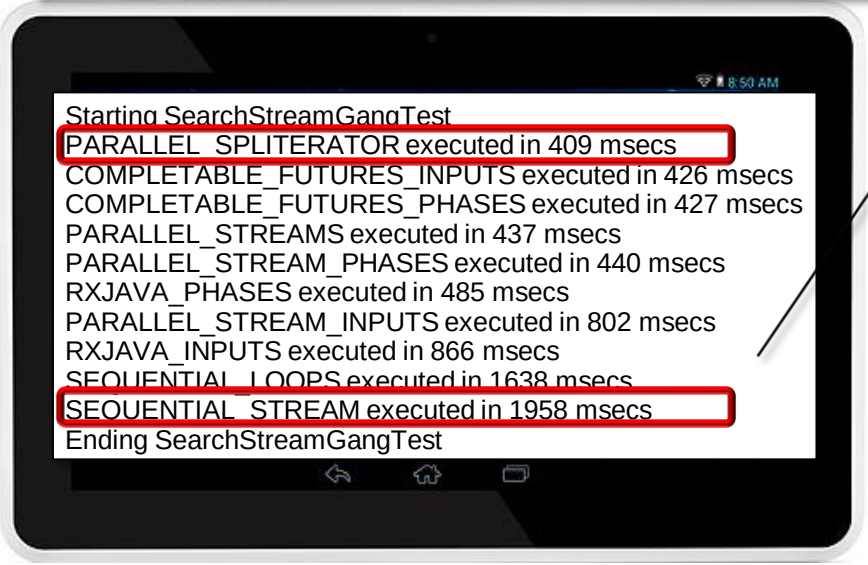


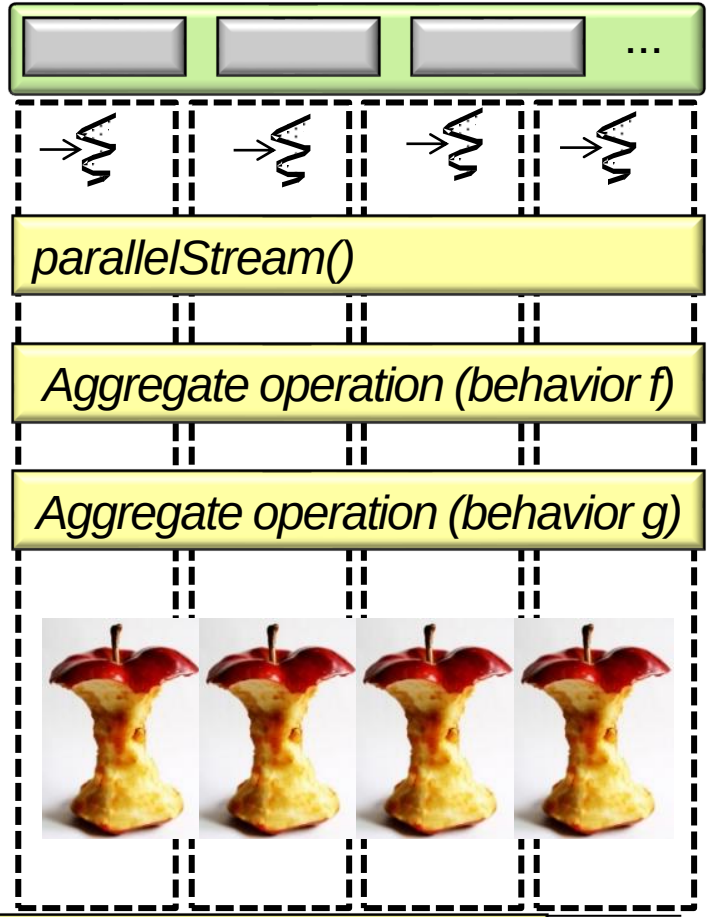
See dzone.com/articles/common-fork-join-pool-and-streams

Comparing Sequential vs. Parallel Streams

- A parallel stream splits its elements into multiple chunks & uses the common fork-join pool to process these chunks independently

A parallel stream can usually be much more efficient & scalable than a sequential stream.



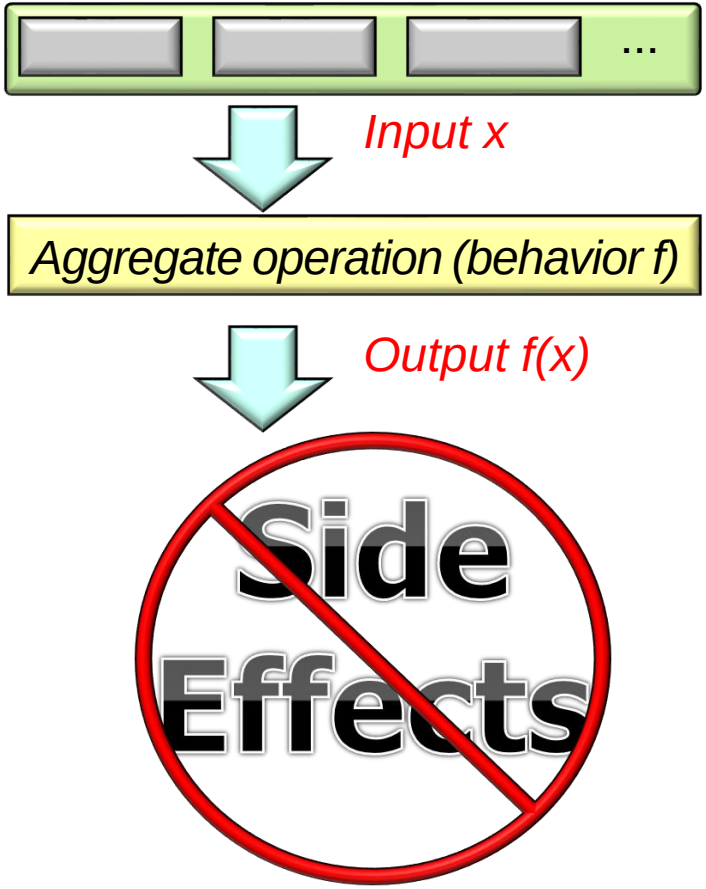


Tests conducted on a quad-core Lenovo P50 with 32 Gbytes of RAM

Common Programming Hazards for Parallel Streams

Common Programming Hazards for Parallel Streams

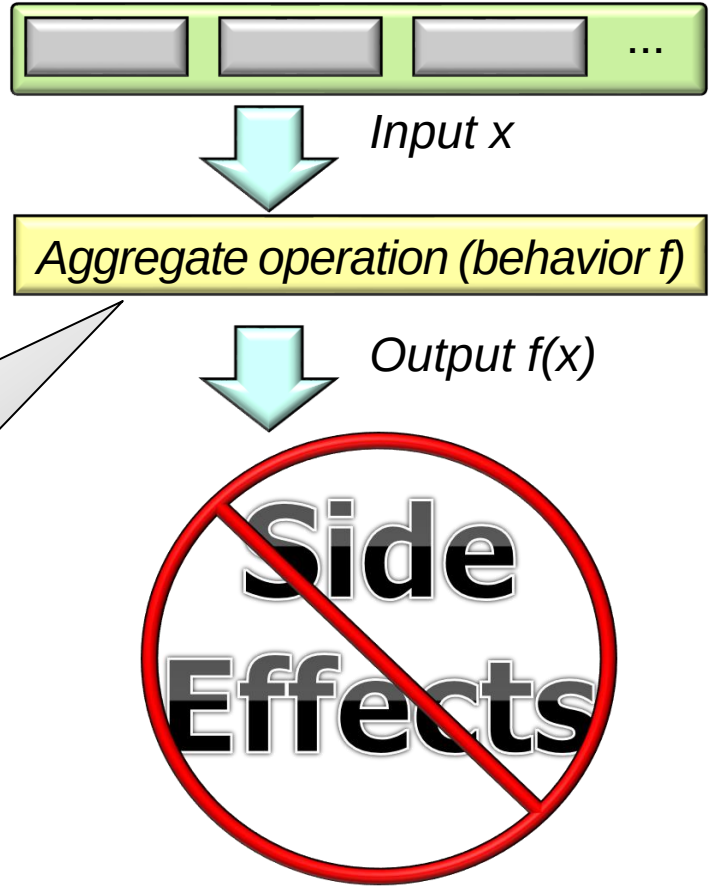
- Ideally, a behavior's output in a stream depends only on its input arguments



See en.wikipedia.org/wiki/Side_effect_(computer_science)

Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments



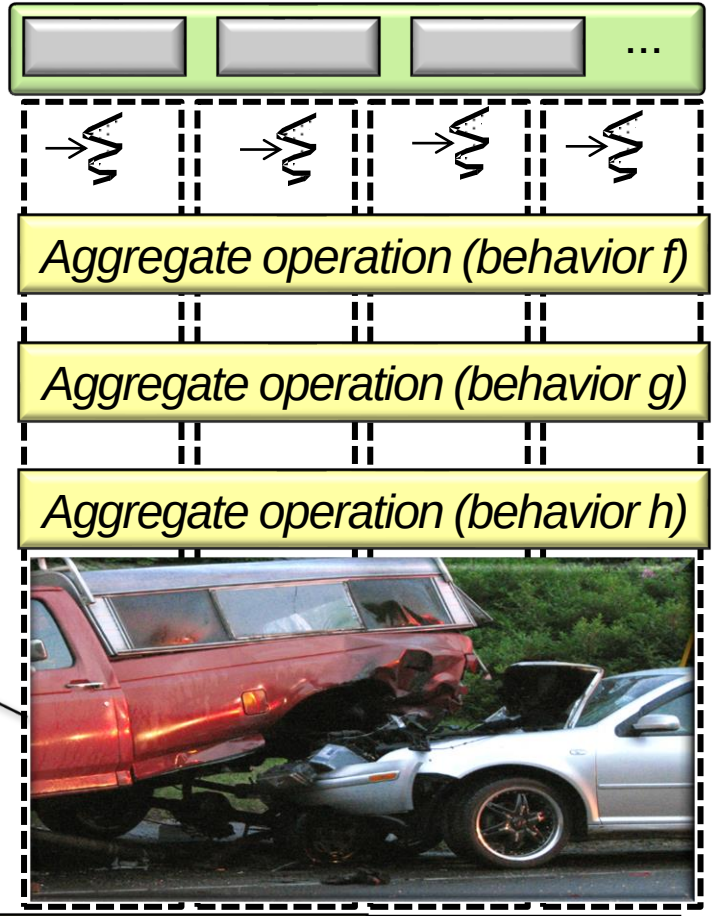

```
String capitalize(String s) {
  if (s.length() == 0)
    return s;
  return s.substring(0, 1)
          .toUpperCase()
        + s.substring(1)
           .toLowerCase();
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex12

Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams

Race conditions arise in software when an application depends on the sequence or timing of threads for it to operate properly



See en.wikipedia.org/wiki/Race_condition#Software

Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

A buggy attempt to compute the 'nth' factorial in parallel

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

Shared mutable state

See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html

Common Programming Hazards for Parallel Streams

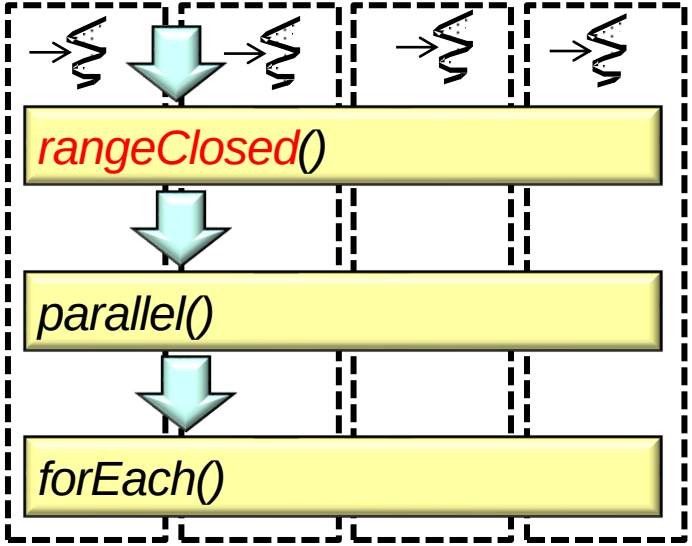
- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

Generate a range of values from 1..n

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```



Common Programming Hazards for Parallel Streams

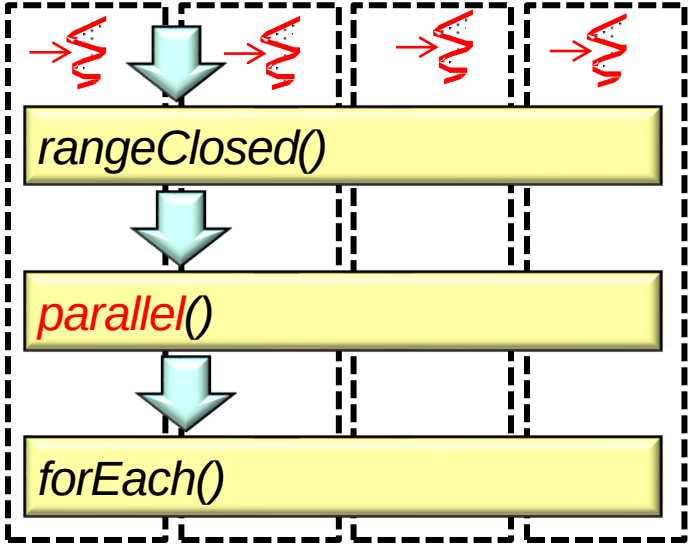
- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

Run in parallel

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```



Common Programming Hazards for Parallel Streams

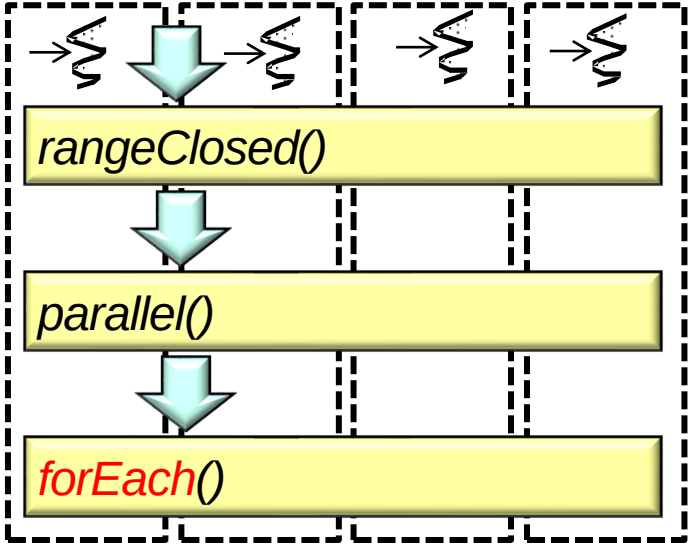
- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

Multiply the running total w/the latest value

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```



Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

Beware of race conditions!!!

See en.wikipedia.org/wiki/Race_condition#Software

Common Programming Hazards for Parallel Streams

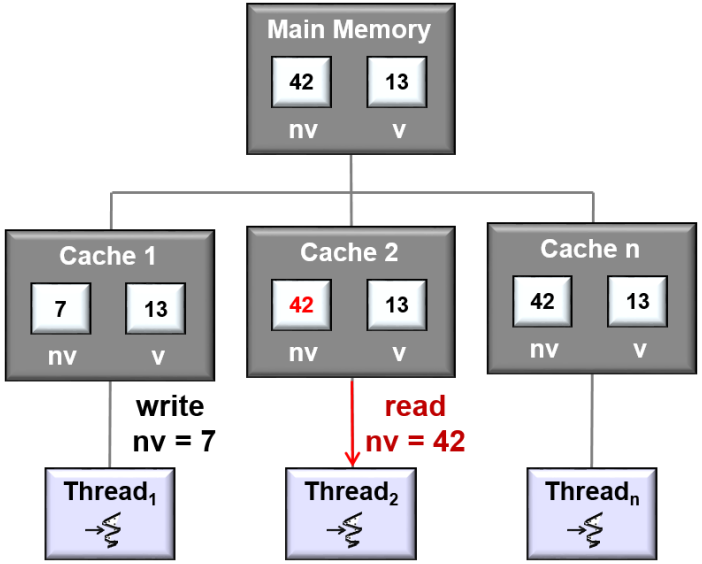
- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

Beware of inconsistent memory visibility

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```



See jeremymanson.blogspot.com/2007/08/atomicity-visibility-and-ordering.html

Common Programming Hazards for Parallel Streams

- Ideally, a behavior's output in a stream depends only on its input arguments
  - Behaviors with side-effects can incur race conditions in parallel streams, e.g.

```
long factorial(long n) {
  Total t = new Total();
  LongStream
    .rangeClosed(1, n)
    .parallel()
    .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

Only you can prevent concurrency hazards!

In Java *you* must avoid these hazards, i.e., the compiler & JVM won't save you..

End of Comparing Java Sequential Streams with Parallel Streams