Contrasting Java 8 Streams with Java I/O Streams and Collections

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

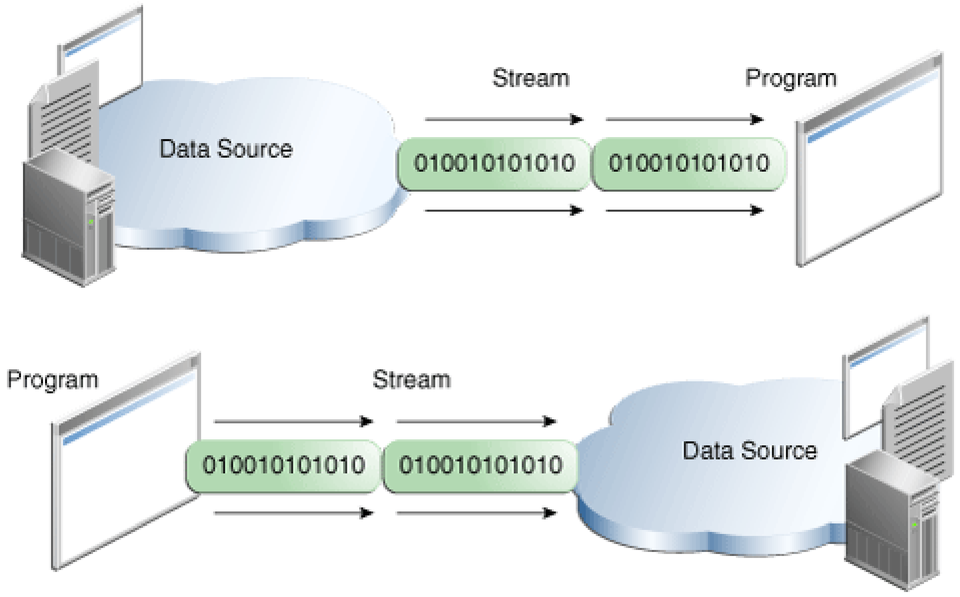
Professor of Computer Science

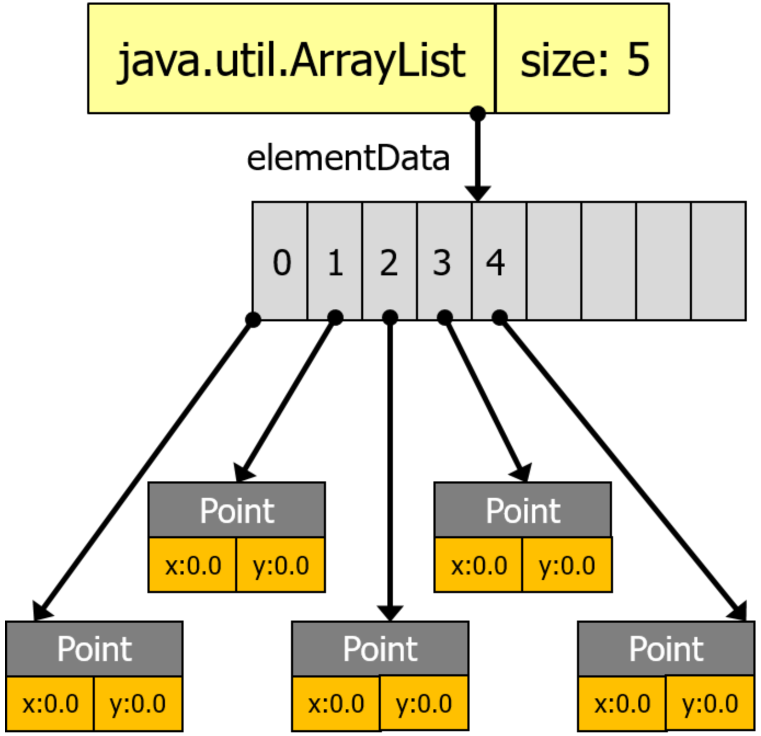
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand how Java 8 streams compare with other Java libraries

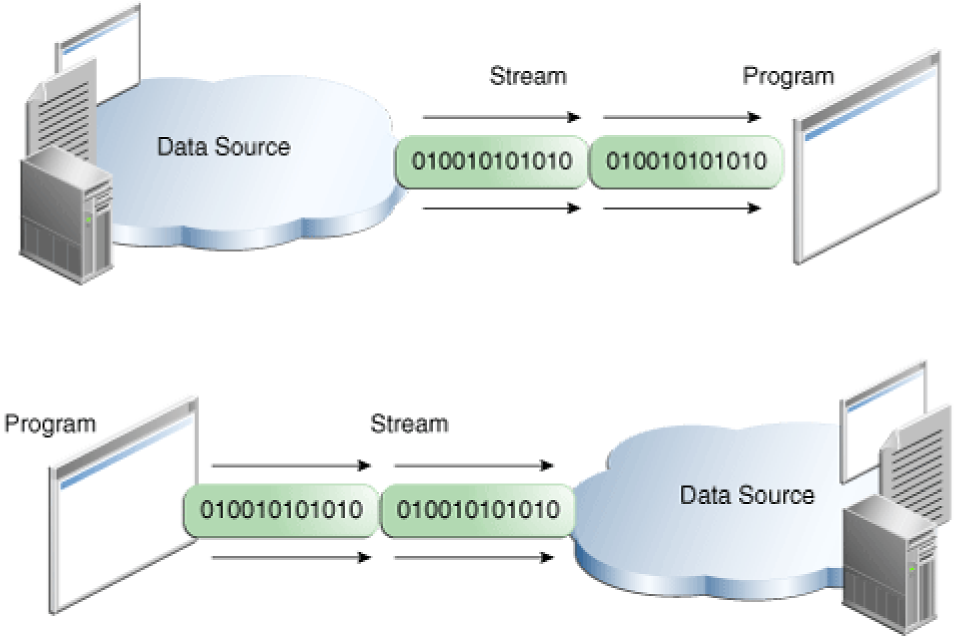




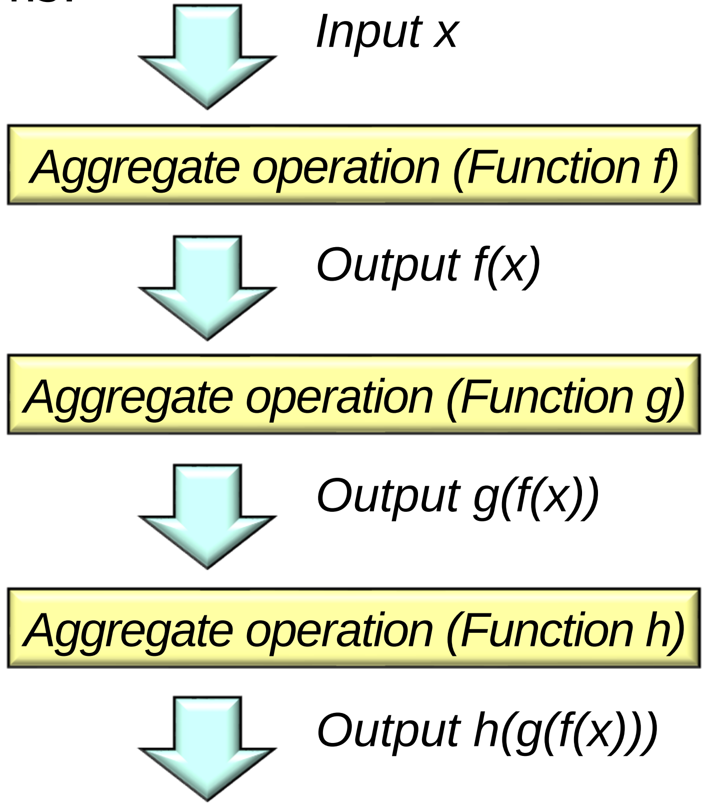
Contrasting Java 8 Streams with Other Java Libraries

Contrasting Java I/O Streams & Java 8 Streams

- Java I/O streams are different from Java 8 streams!



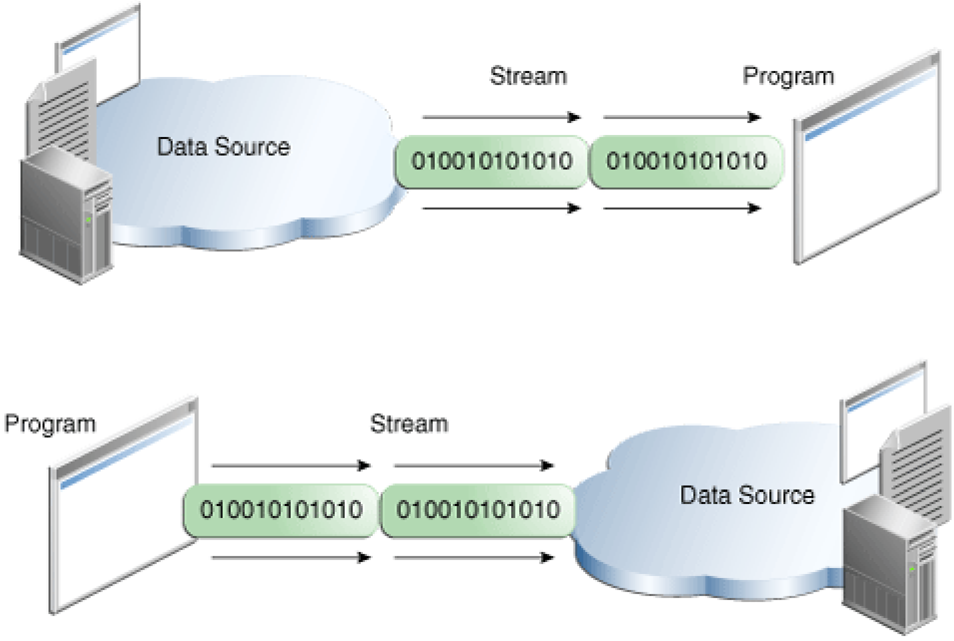
≠


Input x
Aggregate operation (Function f)
Output f(x)
Aggregate operation (Function g)
Output g(f(x))
Aggregate operation (Function h)
Output h(g(f(x)))

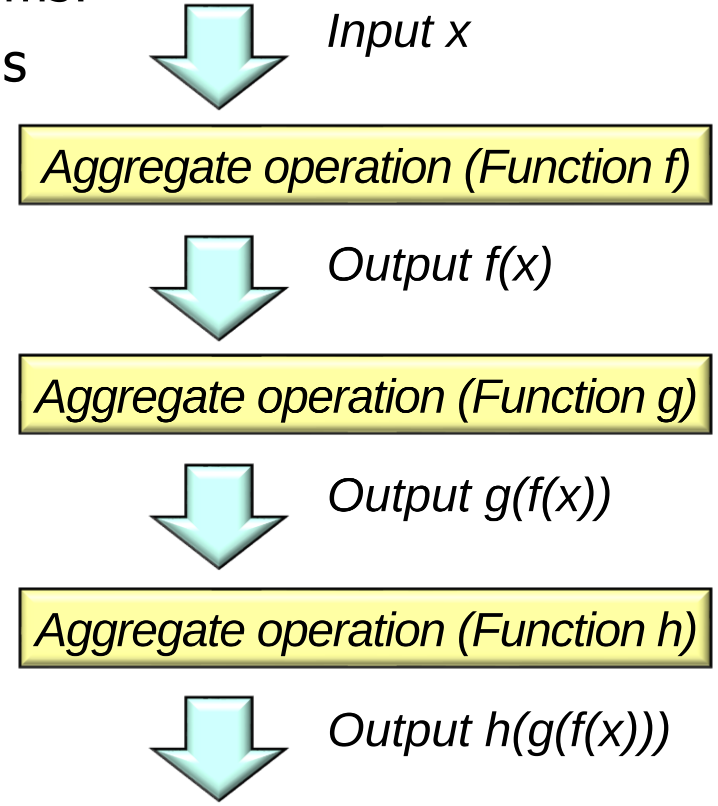
See docs.oracle.com/javase/tutorial/essential/io/streams.html

Contrasting Java I/O Streams & Java 8 Streams

- Java I/O streams are different from Java 8 streams!
  - They are often used together in Java programs

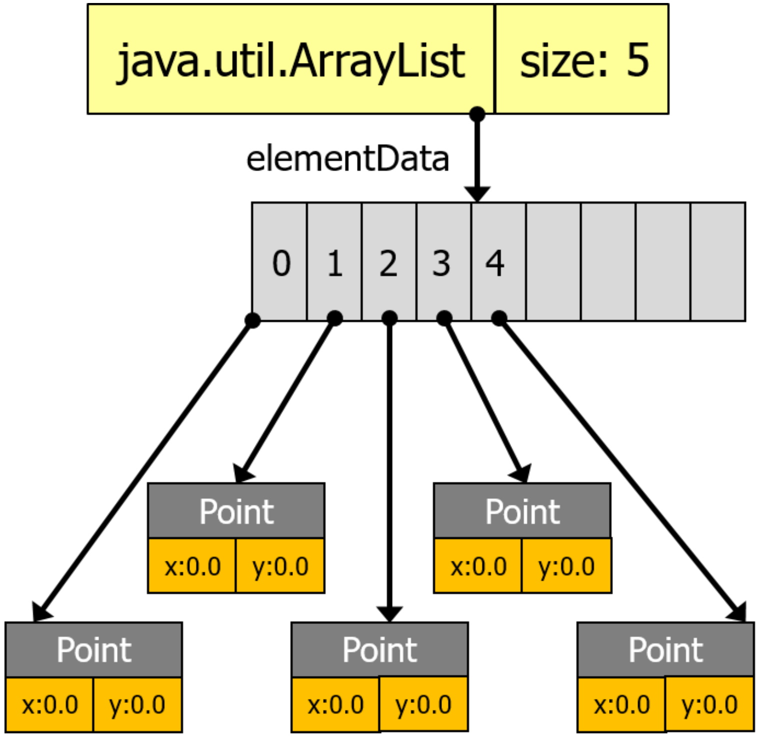


$\neq$

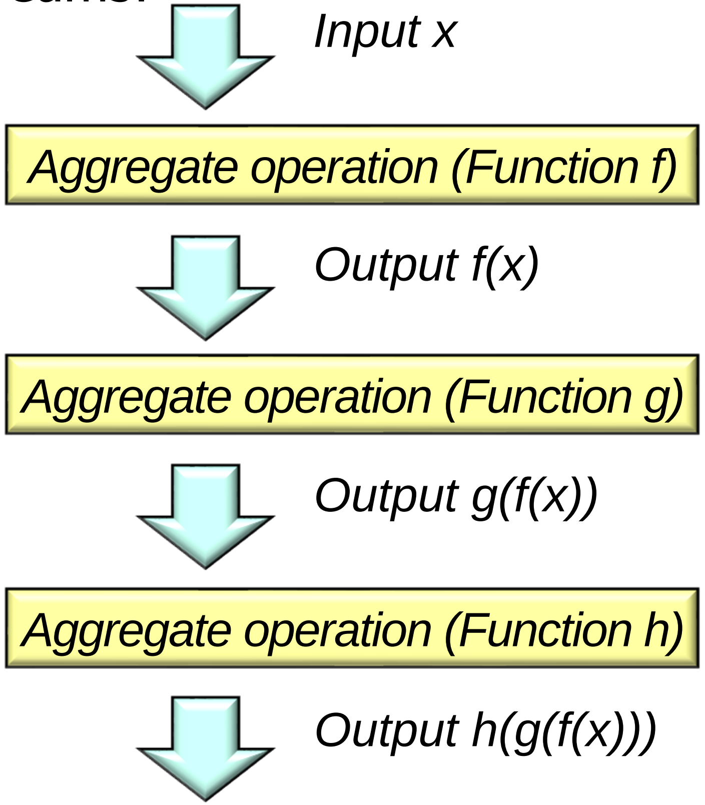

Input x
Aggregate operation (Function f)
Output f(x)
Aggregate operation (Function g)
Output g(f(x))
Aggregate operation (Function h)
Output h(g(f(x)))

Contrasting Collections & Streams

- Java collections are also different from Java 8 streams!



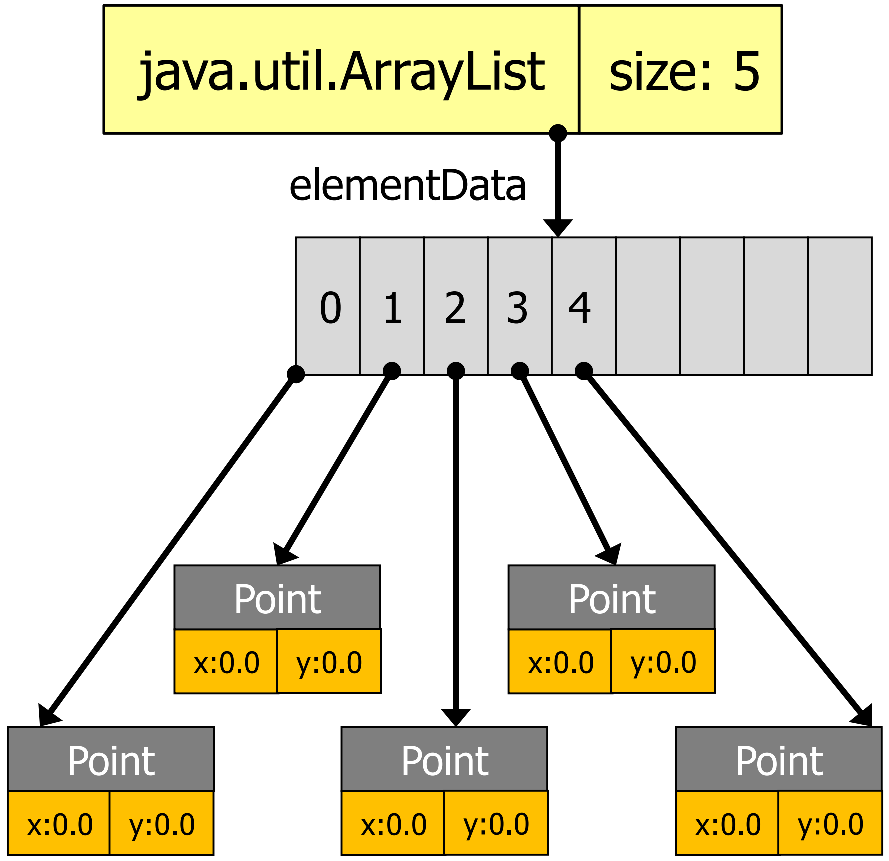
$\neq$


Input x
Aggregate operation (Function f)
Output f(x)
Aggregate operation (Function g)
Output g(f(x))
Aggregate operation (Function h)
Output h(g(f(x)))

See www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html

Contrasting Collections & Streams

- A collection is an in-memory data structure that can store, retrieve, & manipulate groups of elements



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

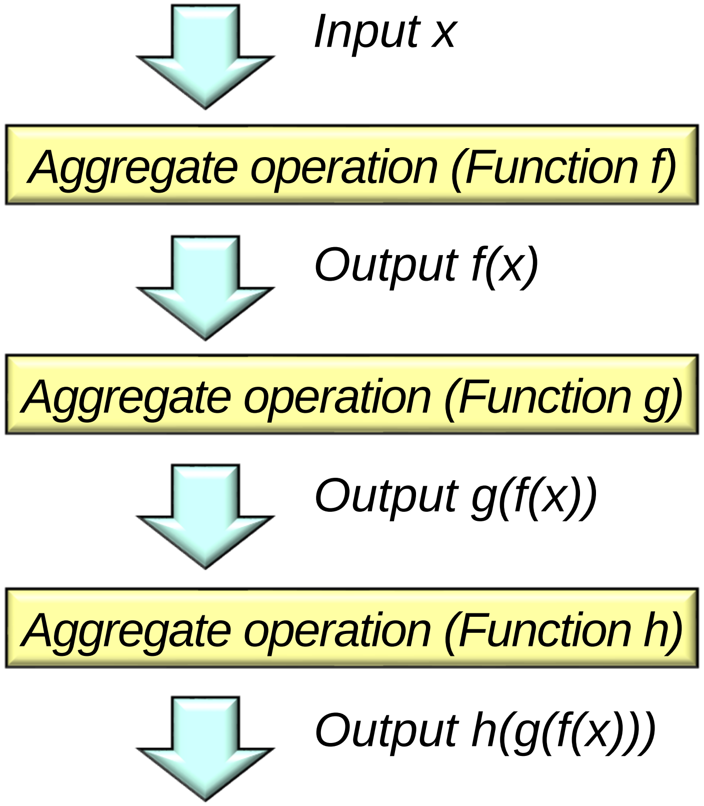
Contrasting Collections & Streams

- A collection is an in-memory data structure that can store, retrieve, & manipulate groups of elements
  - It is analogous to a DVD

All content exists statically (though not persistently)

Contrasting Collections & Streams

- A stream is a fixed data structure that processes elements on-demand



Input x
Aggregate operation (Function f)
Output f(x)
Aggregate operation (Function g)
Output g(f(x))
Aggregate operation (Function h)
Output h(g(f(x)))

A stream can manipulate elements obtained from a collection without explicitly iterating over them

See tutorials.jenkov.com/java-collections/streams.html

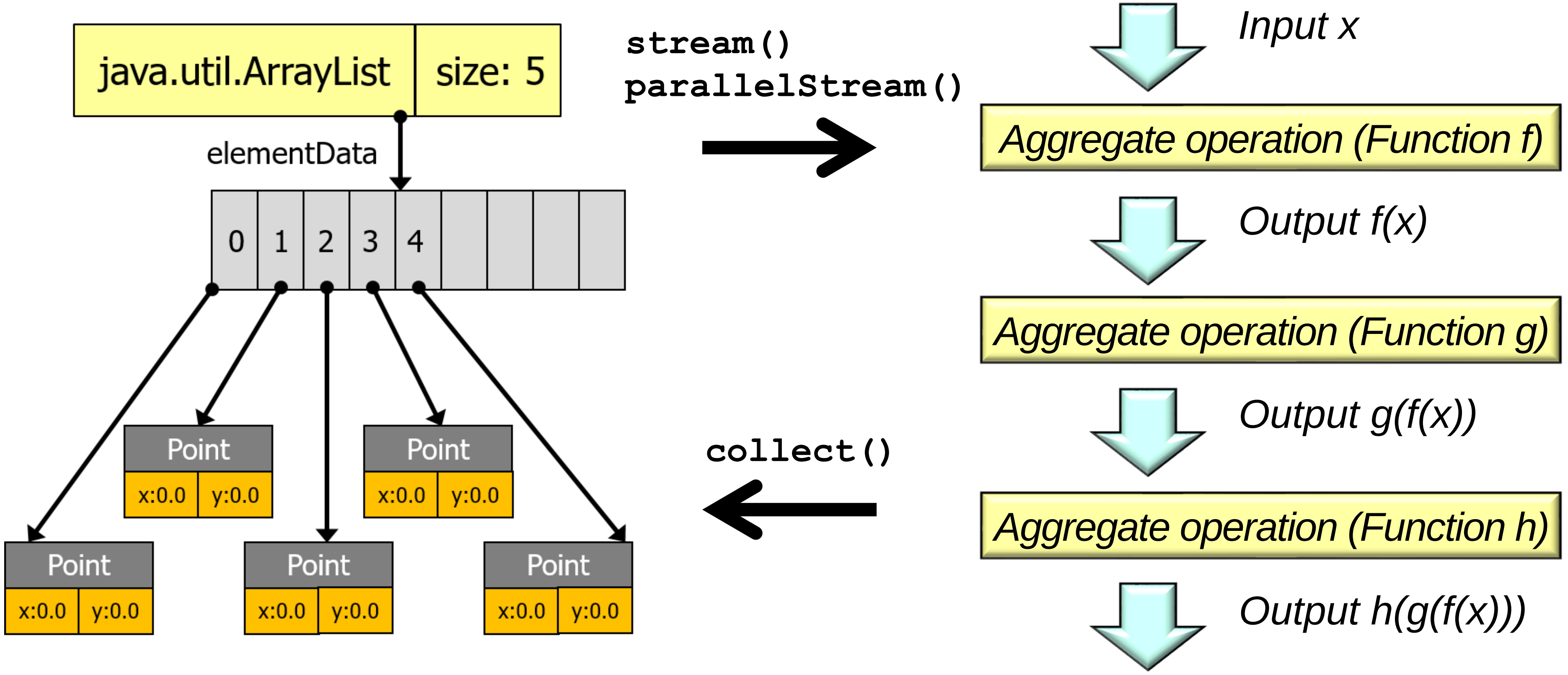
Contrasting Collections & Streams

- A stream is a fixed data structure that processes elements on-demand
  - A Java stream is analogous to a flow of bytes in a streaming video

Content is dynamically received & processed

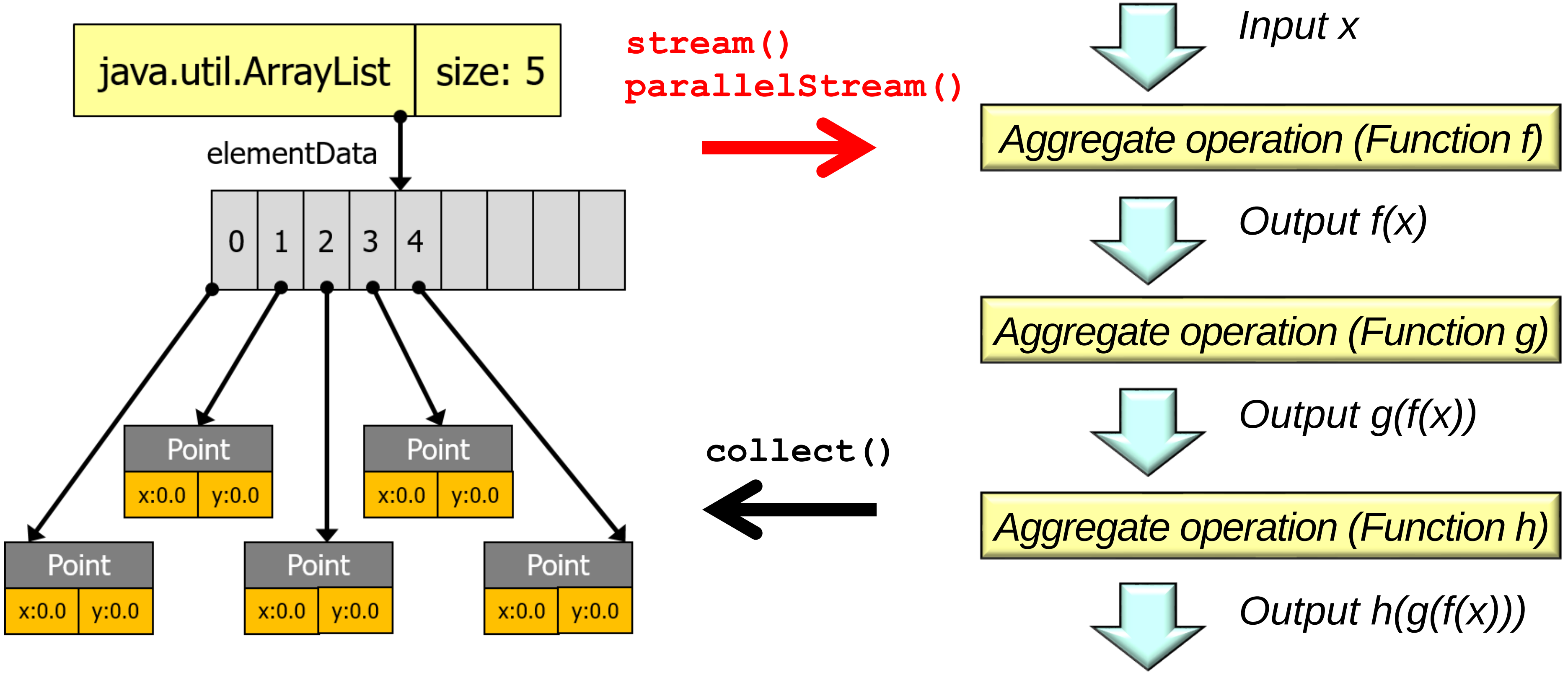
Contrasting Collections & Streams

- Various factory methods can convert collections to streams & vice versa



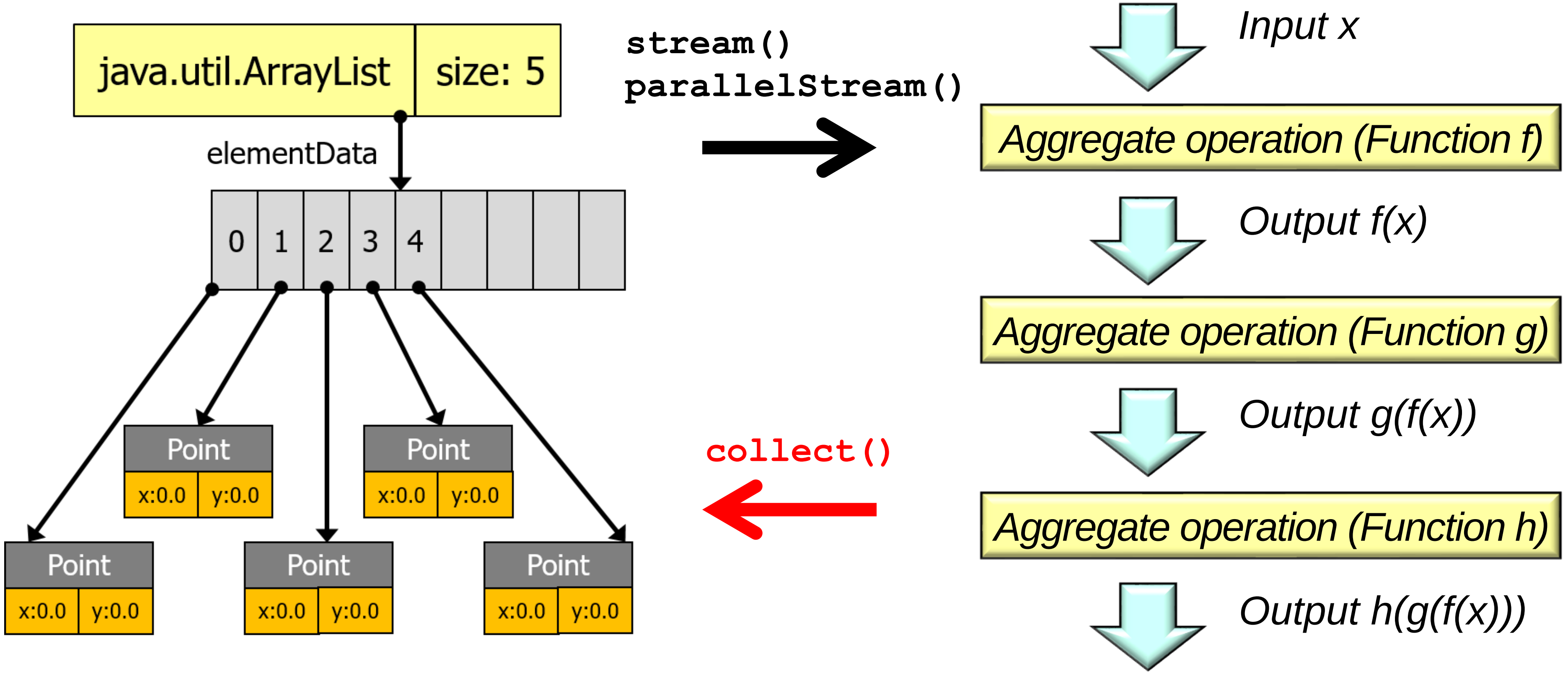
Contrasting Collections & Streams

- Various factory methods can convert collections to streams & vice versa



Contrasting Collections & Streams

- Various factory methods can convert collections to streams & vice versa



Contrasting Collections & Streams

- A simple example of manipulating a Java collection

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Arrays.asList(urlArray);

 for (int i = 0; i < urls.size(); ++i)
   if (!urls.get(i).contains("cse.wustl"))
     continue;
   urls.set(i,
            urls.get(i).replace("cse.wustl","dre.vanderbilt"));
```

This example demonstrates external iteration

Contrasting Collections & Streams

- A simple example of manipulating a Java collection

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Arrays.asList(urlArray);


 for (int i = 0; i < urls.size(); ++i)
   if (!urls.get(i).contains("cse.wustl"))
     continue;
   urls.set(i,
            urls.get(i).replace("cse.wustl","dre.vanderbilt"));
```

Create a list from an array

Contrasting Collections & Streams

- A simple example of manipulating a Java collection

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Arrays.asList(urlArray);
```

Explicitly iterate through a list & modify each matching value

```
 for (int i = 0; i < urls.size(); ++i)
   if (!urls.get(i).contains("cse.wustl"))
     continue;
   urls.set(i,
            urls.get(i).replace("cse.wustl","dre.vanderbilt"));
```

Contrasting Collections & Streams

- A simple example of manipulating a Java collection

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Arrays.asList(urlArray);
```

External iteration enables fine-grained control of loop behavior

```
 for (int i = 0; i < urls.size(); ++i)
   if (!urls.get(i).contains("cse.wustl"))
     continue;
   urls.set(i,
            urls.get(i).replace("cse.wustl","dre.vanderbilt"));
```

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

This example demonstrates "fluent interface" programming style, internal iteration, chaining of transformations

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<String> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .collect(toList());
```

Like iterators, elements in a stream can only be visited once during its lifetime

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<URL> urls = Stream
   .of(urlArray)
   .filter(s -> s.contains("cse.wustl"))
   .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
   .map(rethrowFunction(URL::new))
   .collect(toList());
```

Java 8 streams simplifies chaining of transformations

Contrasting Collections & Streams

- A simple example of manipulating a Java stream

```
String[] urlArray = {
  "http://www.cse.wustl.edu/~schmidt/ka.png",
  "http://www.cse.wustl.edu/~schmidt/robot.png",
  "http://www.cse.wustl.edu/~schmidt/kitten.png"};

 List<URL> urls = Stream
    .of(urlArray)
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
         s.replace("cse.wustl", "dre.vanderbilt"))
    .map(rethrowFunction(URL::new))
    .collect(toList());
```

rethrowFunction() converts checked exception into runtime exception

See stackoverflow.com/a/27661504/3312330

End of Contrasting Java 8 Streams with Java I/O Streams & Collections