Hadoop

Input / Output

ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Hadoop Distributed File System

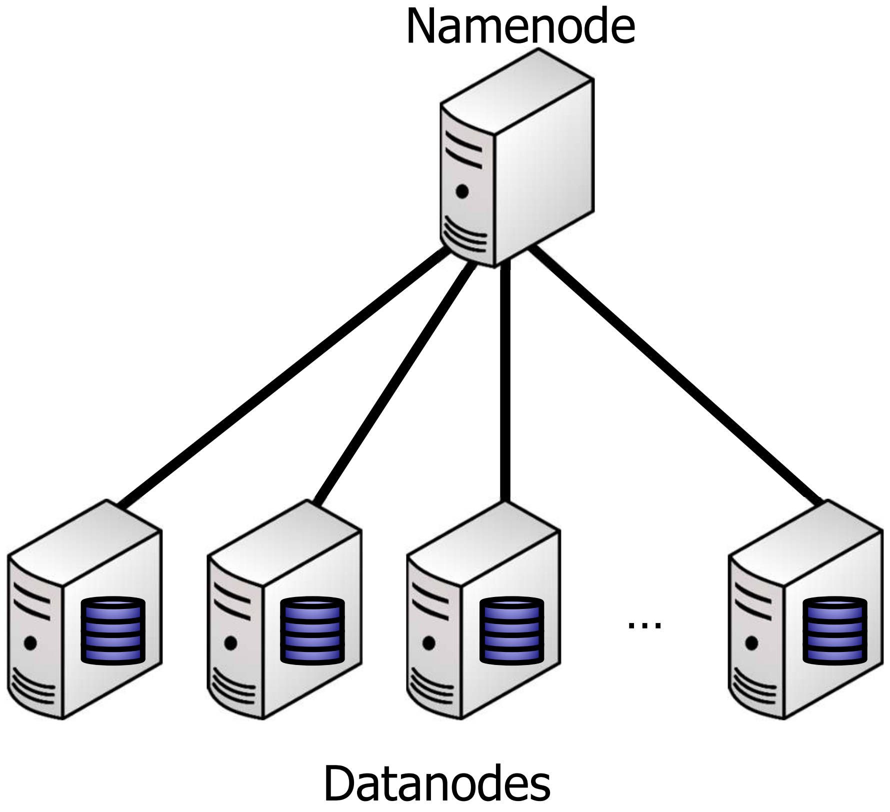
HDFS

HDFS design goal

- Handle files to support Big Data (Mbyte, Gbyte and Tbyte)
- Access to data follows the pattern write-once read-many
- To be executed on top of commodity hardware (with failures)

HDFS performance is not goof if…

- Applications required low latency access
- Lot of small files
- Multiple writes or random modifications



HDFS main concepts

- Blocks – File contents are placed inside blocks.
- HDFS block size is configured in such a way that the time taken to transfer data is very large when compared to the seek time
- Master/Slave (Namenode/Datanodes) architecture
- Block caching to increase read performance

HDFS main concepts

- HDFS Federation
  - Namenode keeps all the information in memory (for better performance) very large cluster with a large number of files require lots of memory
  - After version 2.x its is possible to use more than one Namenode (one for each branch of the file system tree, e.g. `/user` and `/tmp`)
- High Availability
  - Namenode is a single point of failure
  - How can we overcome this situations?

Interacting with HDFS Command Line Interface (CLI)

- Basic syntax
  - `hadoop <command> <sub command> <args>`
  - where:
    - `<command>` the file system command (`fs`)
    - `<sub command>` represents a sub command
    - `<args>` are the arguments

Hadoop CLI

https://hadoop.apache.org/docs/r3.2.0/hadoop-project-dist/hadoop-common/FileSystemShell.html

- Sub commands
  - `-copyFromLocal`
  - `-copyToLocal`
  - `-mkdir`
  - `-rmdir`
  - `-chmod`
  - `-chown`
  - `-ls`
  - `-cat`
  - `-appendToFile`
  - `-cp`
  - `-moveFromLocal`
  - `-moveToLocal`
  - …

POSIX
like interface

Examples

```
f1=/home/usermr/myFile1
f2a=hdfs://localhost/user/usermr/myFile2
f2b=hdfs://localhost:8020/user/usermr/myFile2
f3=/home/usermr/myFile3

f2a <=> f2b


hadoop fs -copyFromLocal ${f1} ${f2a}
hadoop fs -copyToLocal ${f2b} file://${f3}

md5sum ${f1}
e7891a2627cf263a079fb0f18256ffb2
md5sum ${f3}
e7891a2627cf263a079fb0f18256ffb2
```

Hadoop file systems

File system	Scheme	Java implementation
Local	file://	fs.LocalFileSystem
HDFS	hdfs://	hdfs.DistributedFileSystem
WebHDFS	webhdfs://	hdfs.web.WebHdfsFileSystem
Secure WebHDFS	swebhdfs://	hdfs.web.SWebHdfsFileSystem
HAR	har://	fs.HarFileSystem
View	viewfs://	viewfs.ViewFileSystem
FTP	ftp://	fs.ftp.FTPFileSystem
S3	s3a://	fs.s3a.S3AFileSystem
Azure	wasb://	fs.azure.NativeAzureFileSystem
Swift	swift://	fs.swift.snative.SwiftNativeFilesystem

Hadoop – Java interface

Ex13

```
import ...;

public class MyCat {
  static {
    URL.setURLStreamHandlerFactory( new FsUrlStreamHandlerFactory() );
  }

  public static void main(String args[]) throws Exception {
    System.out.println( "Displaying files using a URLStreamHandler" );
    InputStream in = null;
    try {
      in = new URL(args[0]).openStream();
      IOUtils.copyBytes(in, System.out, 4096, false);
    }
    finally {
      IOUtils.closeStream( in );
    }
  }
}
```

Close streams
(yes/no)

Hadoop – Java interface

Ex14

```
import ...;

public class MyCatFileSystem {

  public static void main(String[] args) throws Exception {
    System.out.println( "Displaying files using the FileSystem " );
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get( URI.create(uri), conf);
    InputStream in = null;
    try {
      in = fs.open( new Path(uri) );
      IOUtils.copyBytes(in, System.out, 4096, false);
    }
    finally {
      IOUtils.closeStream( in );
    }
  }
}
```

Hadoop – Java interface

Ex15

```
import ...;

public class MyCatFileSystemTwice {
  public static void main(String[] args) throws Exception {
    System.out.println( "Displaying files using seek()" );
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    FSDataInputStream in = null;
    try {
      in = fs.open(new Path(uri));
      IOUtils.copyBytes(in, System.out, 4096, false);
      // go back to the start of the file
      in.seek(0);
      IOUtils.copyBytes(in, System.out, 4096, false);
    }
    finally {
      IOUtils.closeStream( in );
    }
  }
}
```

Hadoop – Java interface

Ex16

```
import ...;

class MyProgressable implements Progressable {
  @Override
  public void progress() { System.out.print(".");    }
}

public class MyFileCopyWithProgress {
  public static void main(String[] args) throws Exception {
    String localSrc = args[0];
    String dst = args[1];
    InputStream in;
    in = new BufferedInputStream( new FileInputStream( localSrc ) );
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get( URI.create(dst), conf );
    Progressable p = new MyProgressable();
    OutputStream out = fs.create( new Path( dst ), p );
    IOUtils.copyBytes(in, out, 4096, true);
  }
}
```

Close both

Hadoop – Java interface

Ex17

```
import ...;

public class MyListStatus {
  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    Path[] paths = new Path[args.length];
    for (int i = 0; i < paths.length; i++) {
      paths[i] = new Path(args[i]);
    }
    FileStatus[] status = fs.listStatus( paths );
    Path[] listedPaths = FileUtil.stat2Paths( status );
    for (Path p : listedPaths) {
      System.out.println(p);
    }
  }
}
```

Data Integrity

HDFS

Data integrity

- Every transmission of data over a communication channel is associated to a small chance of introducing errors, and this probability increases with the volume of data transmission
- An usual approach to detected this errors is to generated a check sum of the data, in the beginning of the communication, and compare the check sum of the received data with the original check sum
- The transmission of the check sums over the communication channel can also be corrupted, but because the size of the check sum is very small (when compared to the original data) the probability of the check sum being corrupted is very small

Data integrity

- Hadoop uses check sums based on the `CRC32` / `CRC32C` algorithms

- A cyclic redundancy check (`CRC`) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents

- `CRC-32` and `CRC32C` (Castagnoli `CRC32`) differ in the polynomial used

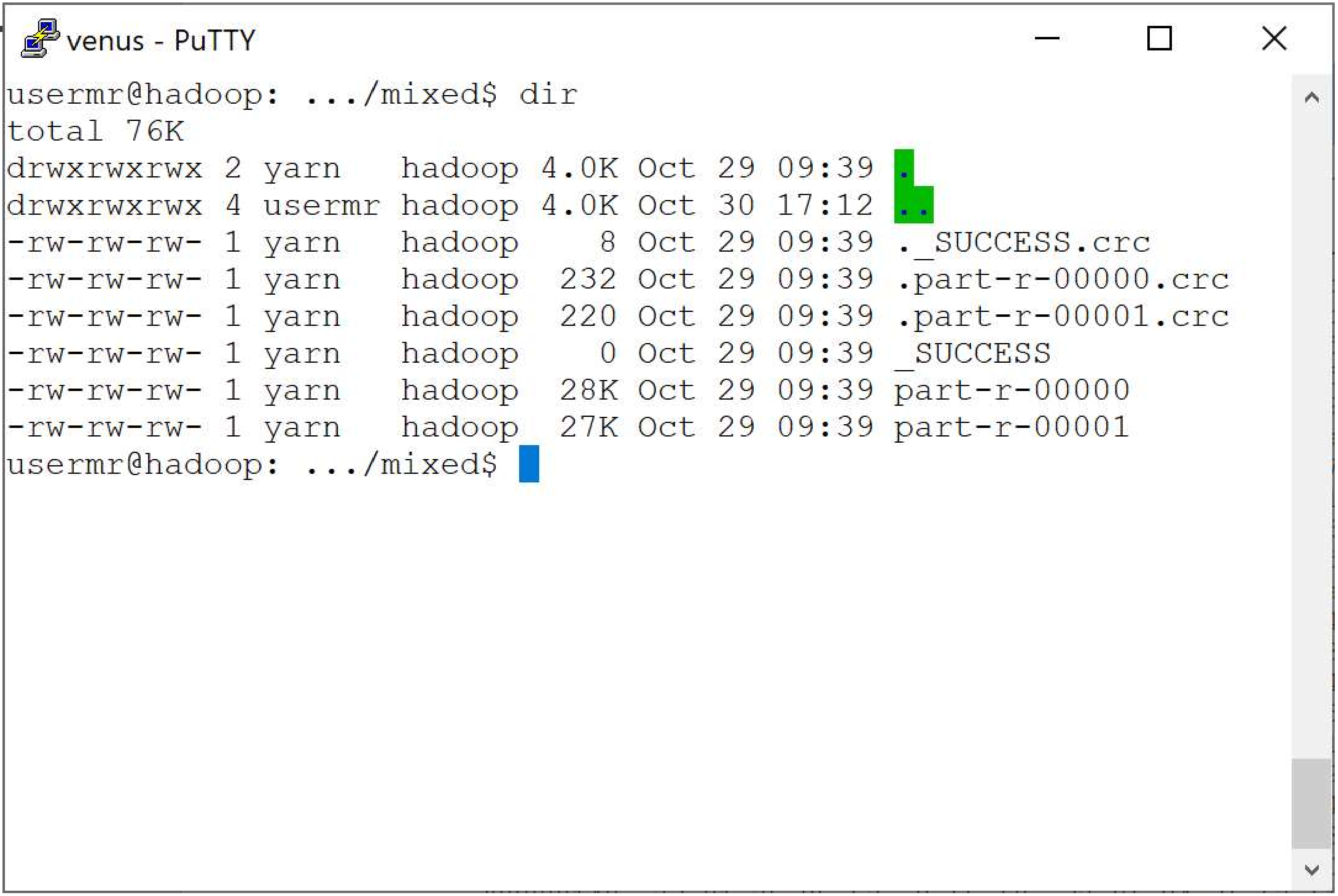
Data integrity HDFS

- To each block of 512 bytes (`dfs.bytes-perchecksum`) Hadoop calculates the corresponding CRC (4 bytes) leading to an overhead less than 1% (4/512*100=0.78%)
- Datanodes have the responsibility of verifying the CRC
- This verification is made on the receiving of data and in the replication process
- The last Datanode is responsible for comparing the original CRC with the CRC of the received data. If data is corrupted an exception is generated and is sent back to the client

Data integrity HDFS

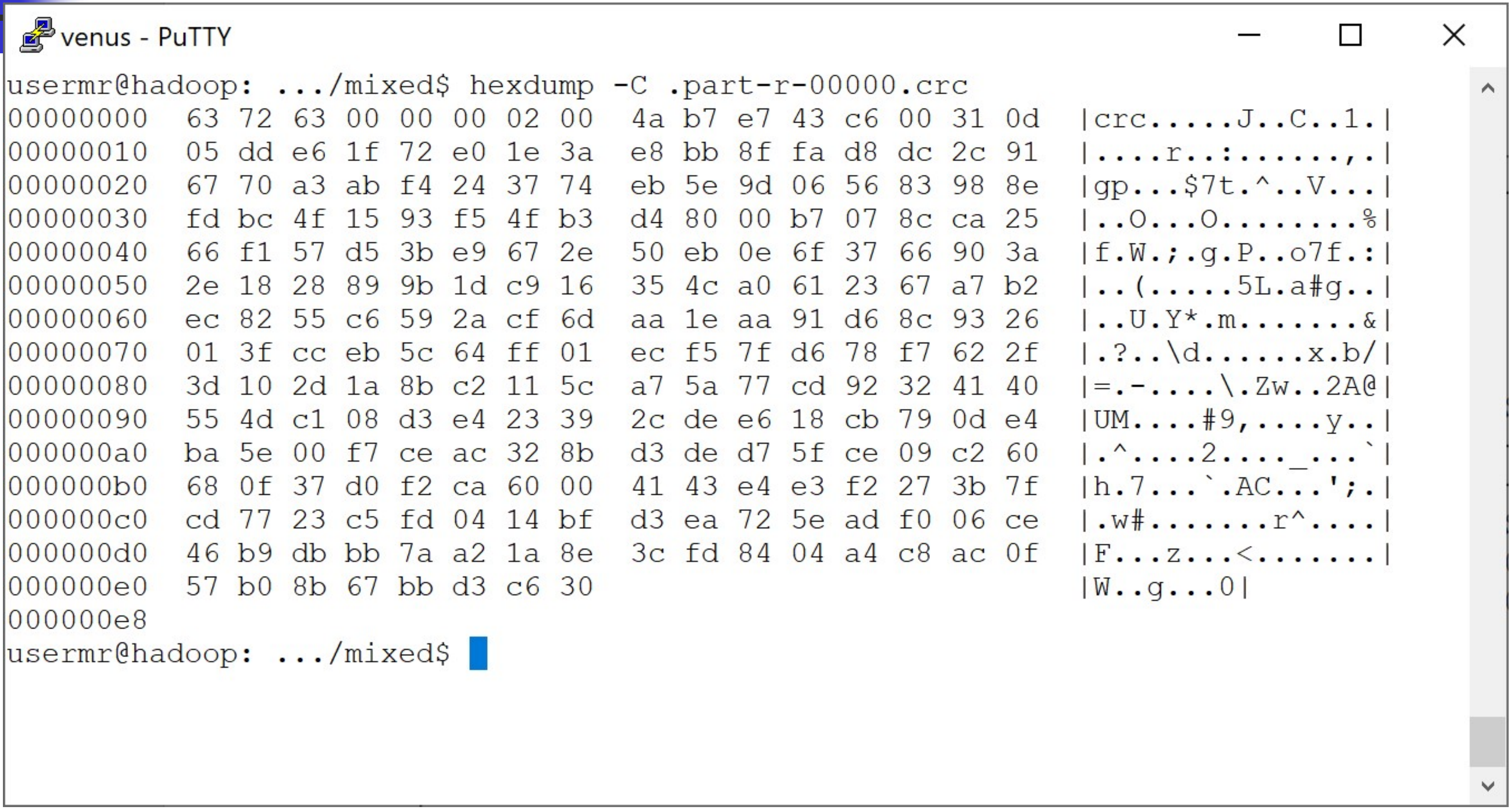
- On each Datanode there is a background task that is responsible for verifying the `CRC` of each block of data
- Corrupted blocks can be replaced by taking advantages of the replicas of that block
- In the local file system (`file://`), to each file (of name `filename`) Hadoop automatically generates its `CRC` that is kept in a file with the name `.filename.crc`

Data integrity HDFS



```
usermr@hadoop: .../mixed$ dir
total 76K
drwxrwxrwx 2 yarn    hadoop 4.0K Oct 29 09:39 .
drwxrwxrwx 4 usermr hadoop 4.0K Oct 30 17:12 ..
-rw-rw-rw- 1 yarn    hadoop    8 Oct 29 09:39 ._SUCCESS.crc
-rw-rw-rw- 1 yarn    hadoop  232 Oct 29 09:39 .part-r-00000.crc
-rw-rw-rw- 1 yarn    hadoop  220 Oct 29 09:39 .part-r-00001.crc
-rw-rw-rw- 1 yarn    hadoop    0 Oct 29 09:39 _SUCCESS
-rw-rw-rw- 1 yarn    hadoop  28K Oct 29 09:39 part-r-00000
-rw-rw-rw- 1 yarn    hadoop  27K Oct 29 09:39 part-r-00001
usermr@hadoop: .../mixed$
```

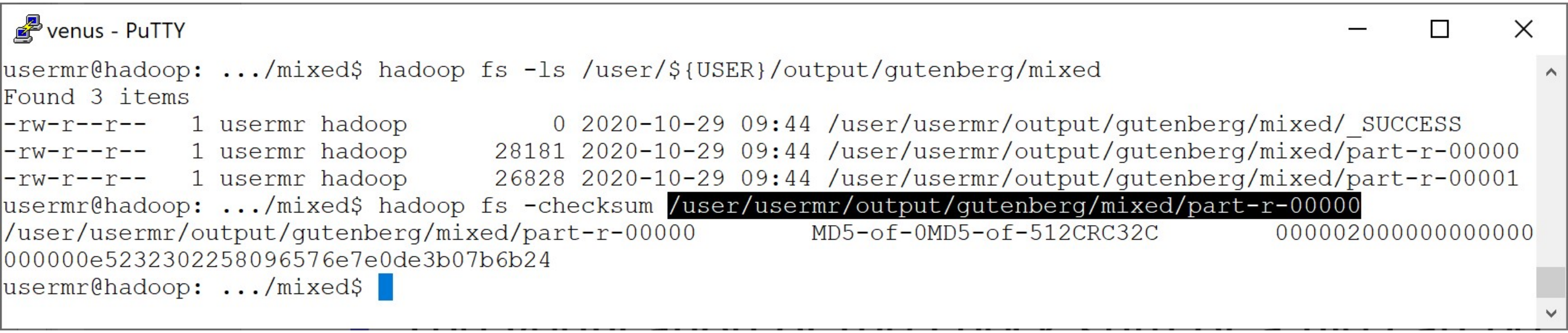
Data integrity HDFS



```
usermr@hadoop: .../mixed$ hexdump -C .part-r-00000.crc
00000000  63 72 63 00 00 00 02 00  4a b7 e7 43 c6 00 31 0d  |crc.....J..C..1.|
00000010  05 dd e6 1f 72 e0 1e 3a  e8 bb 8f fa d8 dc 2c 91  |....r..:......,.|
00000020  67 70 a3 ab f4 24 37 74  eb 5e 9d 06 56 83 98 8e  |gp...$7t.^..V...|
00000030  fd bc 4f 15 93 f5 4f b3  d4 80 00 b7 07 8c ca 25  |..O...O........%|
00000040  66 f1 57 d5 3b e9 67 2e  50 eb 0e 6f 37 66 90 3a  |f.W.;.g.P..o7f.:|
00000050  2e 18 28 89 9b 1d c9 16  35 4c a0 61 23 67 a7 b2  |..(.....5L.a#g..|
00000060  ec 82 55 c6 59 2a cf 6d  aa 1e aa 91 d6 8c 93 26  |..U.Y*.m.......&|
00000070  01 3f cc eb 5c 64 ff 01  ec f5 7f d6 78 f7 62 2f  |.?..\d......x.b/|
00000080  3d 10 2d 1a 8b c2 11 5c  a7 5a 77 cd 92 32 41 40  |=.-....\.Zw..2A@|
00000090  55 4d c1 08 d3 e4 23 39  2c de e6 18 cb 79 0d e4  |UM....#9,....y..|
000000a0  ba 5e 00 f7 ce ac 32 8b  d3 de d7 5f ce 09 c2 60  |.^....2...._...`|
000000b0  68 0f 37 d0 f2 ca 60 00  41 43 e4 e3 f2 27 3b 7f  |h.7...`.AC...';.|
000000c0  cd 77 23 c5 fd 04 14 bf  d3 ea 72 5e ad f0 06 ce  |.w#.......r^....|
000000d0  46 b9 db bb 7a a2 1a 8e  3c fd 84 04 a4 c8 ac 0f  |F...z...<.......|
000000e0  57 b0 8b 67 bb d3 c6 30                           |W..g...0|
000000e8
usermr@hadoop: .../mixed$
```

Data integrity HDFS

- The verification of the check sum of a file can be made using the command
  - `hadoop fs -checksum <file>`



venus - PuTTY

```
usermr@hadoop: .../mixed$ hadoop fs -ls /user/${USER}/output/gutenberg/mixed
Found 3 items
-rw-r--r--   1 usermr hadoop          0 2020-10-29 09:44 /user/usermr/output/gutenberg/mixed/_SUCCESS
-rw-r--r--   1 usermr hadoop      28181 2020-10-29 09:44 /user/usermr/output/gutenberg/mixed/part-r-00000
-rw-r--r--   1 usermr hadoop      26828 2020-10-29 09:44 /user/usermr/output/gutenberg/mixed/part-r-00001
usermr@hadoop: .../mixed$ hadoop fs -checksum /user/usermr/output/gutenberg/mixed/part-r-00000
/user/usermr/output/gutenberg/mixed/part-r-00000        MD5-of-0MD5-of-512CRC32C        000002000000000000
000000e5232302258096576e7e0de3b07b6b24
usermr@hadoop: .../mixed$
```

Data Compression

HDFS

Compression

- File compression benefits:
  - Reduces the space needed to store files
  - Speeds up data transfer across the network or to or from disk
- However, compression algorithms exhibit a space/time trade-off:
  - **Faster** compression and decompression **speeds** usually come at the **expense** of **smaller** space savings

Compression in HDFS

Format	Tool	Algorithm	Filename Extension	Splittable
DEFLATE [a]	N/A	DEFLATE	`.deflate`	No
gzip	gzip	DEFLATE	`.gz`	No
bzip2	bzip2	bzip2	`.bz2`	Yes
LZ0	lzop	LZ0	`.lzo`	No [b]
LZ4	N/A	LZ4	`.lz4`	No
Snappy	N/A	Snappy	`.snappy`	No

[a] `DEFLATE` is a compression algorithm whose standard implementation is `zlib`. The `.deflate` filename extension is a Hadoop convention.

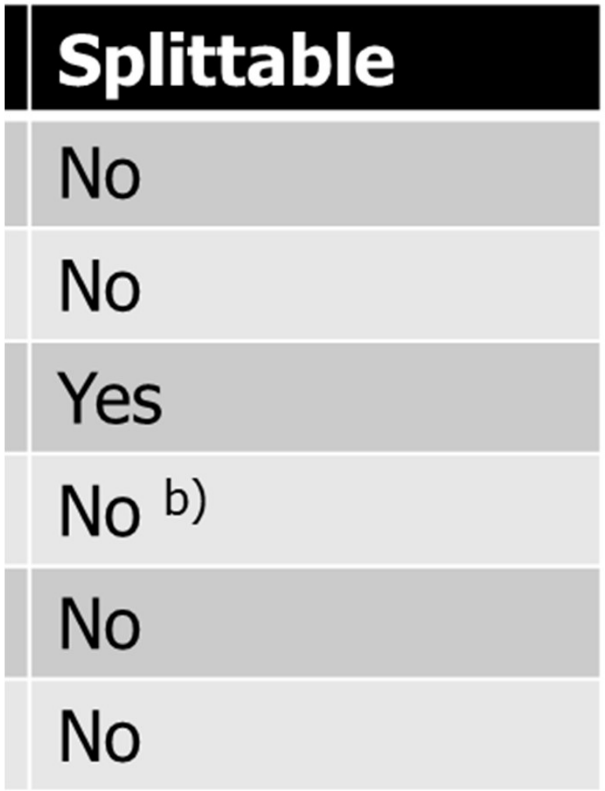
[b] `LZO` files are splittable if they have been indexed in a preprocessing step.

Compression in HDFS

- `gzip` is a general-purpose compressor and sits in the middle of the space/time trade-off
- `bzip2` compresses more effectively than gzip but is slower.
- `bzip2`'s decompression speed is faster than its compression speed, but it is still slower than the other formats
- `LZO`, `LZ4`, and `Snappy` on the other hand, all optimize for speed and are around an order of magnitude faster than `gzip`, but compress less effectively
- Snappy and `LZ4` are also significantly faster than `LZO` for decompression

Compression in HDFS

- The “Splittable” column indicates whether the compression format supports splitting (that is, whether we can seek to any point in the stream and start reading from some point further on)
- Splittable compression formats are especially suitable for Map-Reduce



Splittable
No
No
Yes
No b)
No
No

Compression in HDFS – Codecs

Compression Format	Hadoop Compression Codec Class Name
DEFLATE	`org.apache.hadoop.io.compress.DefaultCodec`
gzip	`org.apache.hadoop.io.compress.GzipCodec`
bzip2	`org.apache.hadoop.io.compress.BZip2Codec`
LZO a)	`com.hadoop.compression.lzo.LzopCodec`
LZ4	`org.apache.hadoop.io.compress.Lz4Codec`
Snappy	`org.apache.hadoop.io.compress.SnappyCodec`

The LZO libraries are GPL licensed and may not be included in Apache distributions, so, for this reason the Hadoop codecs must be downloaded separately from Google (or GitHub, which includes bug fixes and more tools)

Compression in HDFS – Codecs

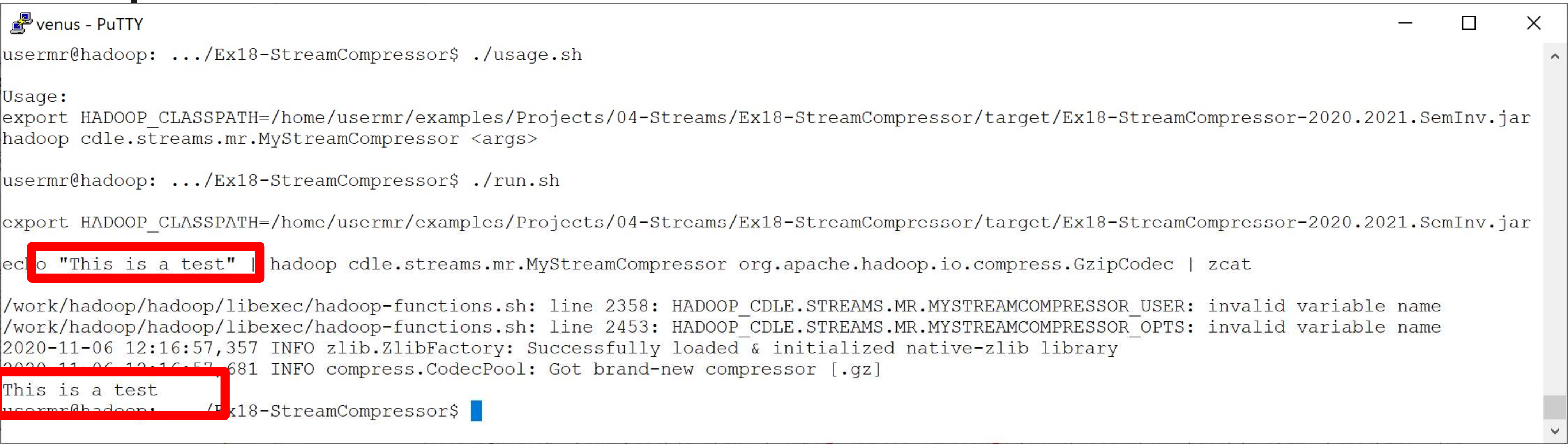
Compressing and decompressing streams

Ex18

```
import ...;

public class MyStreamCompressor {
  public static void main(String[] args) throws Exception {
    String codecClassname = args[0];
    Class<?> codecClass = Class.forName( codecClassname );
    Configuration conf = new Configuration();
    CompressionCodec codec = ( CompressionCodec )
    ReflectionUtils.newInstance(codecClass, conf);

    CompressionOutputStream out;
    out = codec.createOutputStream( System.out );
    IOUtils.copyBytes(System.in, out, 4096, false);
    out.finish();
  }
}
```

Compression in HDFS – Codecs

Compressing and decompressing streams



```
usermr@hadoop: .../Ex18-StreamCompressor$ ./usage.sh

Usage:
export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex18-StreamCompressor/target/Ex18-StreamCompressor-2020.2021.SemInv.jar
hadoop cdle.streams.mr.MyStreamCompressor <args>

usermr@hadoop: .../Ex18-StreamCompressor$ ./run.sh

export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex18-StreamCompressor/target/Ex18-StreamCompressor-2020.2021.SemInv.jar

echo "This is a test" | hadoop cdle.streams.mr.MyStreamCompressor org.apache.hadoop.io.compress.GzipCodec | zcat

/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYSTREAMCOMPRESSOR_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYSTREAMCOMPRESSOR_OPTS: invalid variable name
2020-11-06 12:16:57,357 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
2020-11-06 12:16:57,681 INFO compress.CodecPool: Got brand-new compressor [.gz]
This is a test
usermr@hadoop: .../Ex18-StreamCompressor$
```

Compression in HDFS – Codecs

Compressing and decompressing streams

Ex19

```
import ...;

public class MyFileDecompressorVer01 {
  public static void main(String[] args) throws Exception {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);

    Path inputPath = new Path(uri);
    CompressionCodecFactory factory;
    factory = new CompressionCodecFactory(conf);
    CompressionCodec codec = factory.getCodec(inputPath);
    if ( codec==null ) {
      System.err.println("No codec found for " + uri);
      System.exit(1);
    }
    ...
```

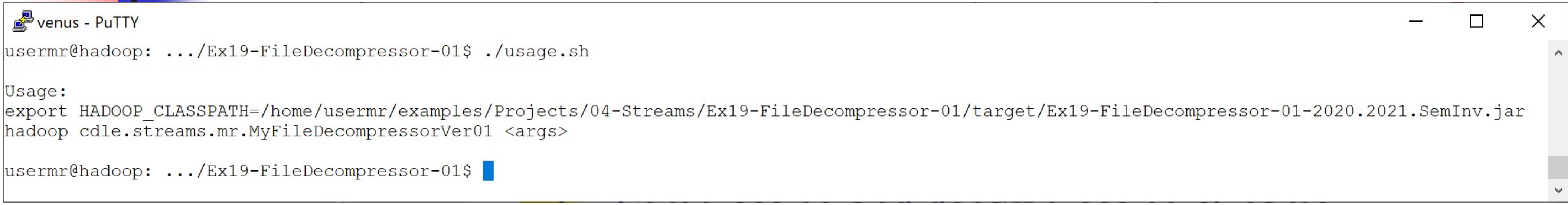
Compression in HDFS – Codecs

Compressing and decompressing streams

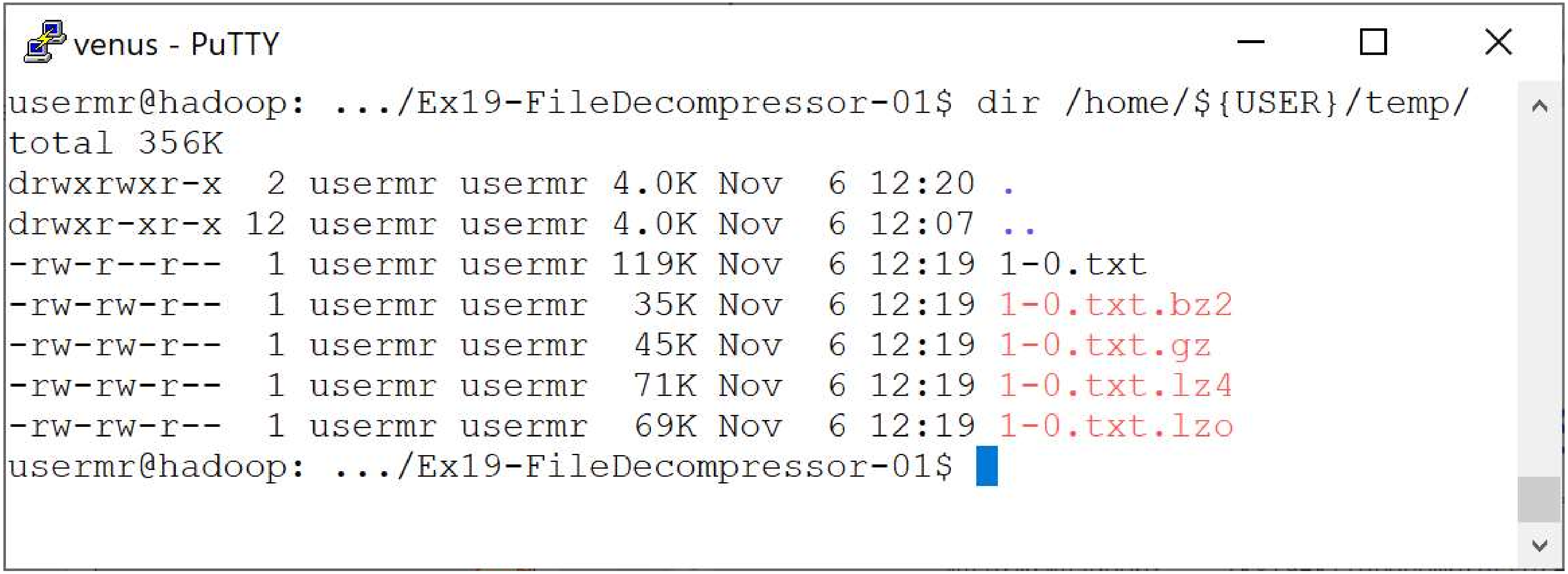
```
    ...
    String extension;
    extension = codec.getDefaultExtension();
    String outputUri;
    outputUri = CompressionCodecFactory.removeSuffix(uri, extension);
    InputStream in = null;
    OutputStream out = null;
    try {
      in = codec.createInputStream(fs.open(inputPath));
      out = fs.create(new Path(outputUri));
      IOUtils.copyBytes(in, out, conf);
    }
    finally {
      IOUtils.closeStream( in );      IOUtils.closeStream( out );
    }
  }
}
```

Compression in HDFS – Codecs

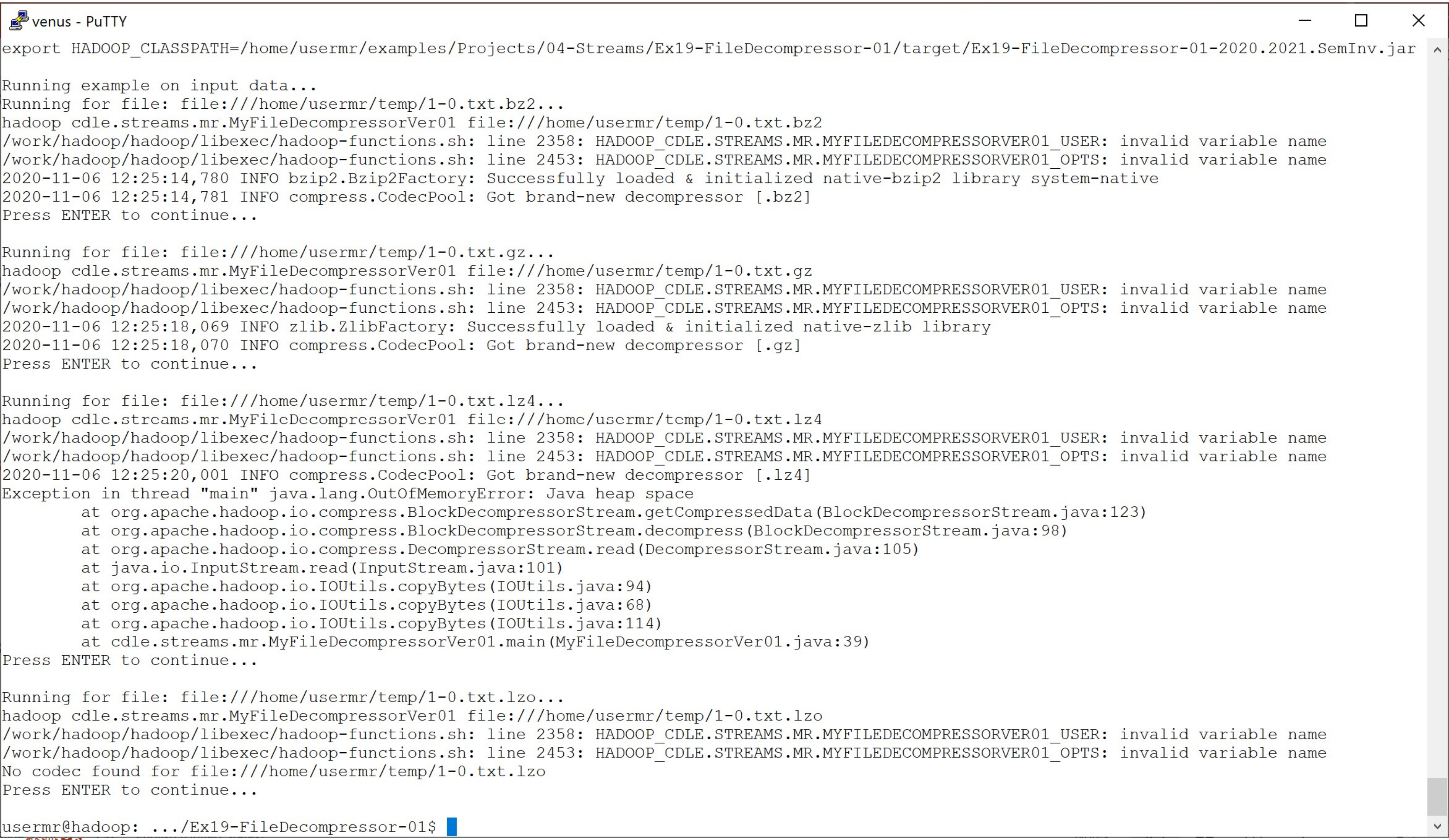
Compressing and decompressing streams



```
usermr@hadoop: .../Ex19-FileDecompressor-01$ ./usage.sh

Usage:
export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex19-FileDecompressor-01/target/Ex19-FileDecompressor-01-2020.2021.SemInv.jar
hadoop cdle.streams.mr.MyFileDecompressorVer01 <args>

usermr@hadoop: .../Ex19-FileDecompressor-01$
```



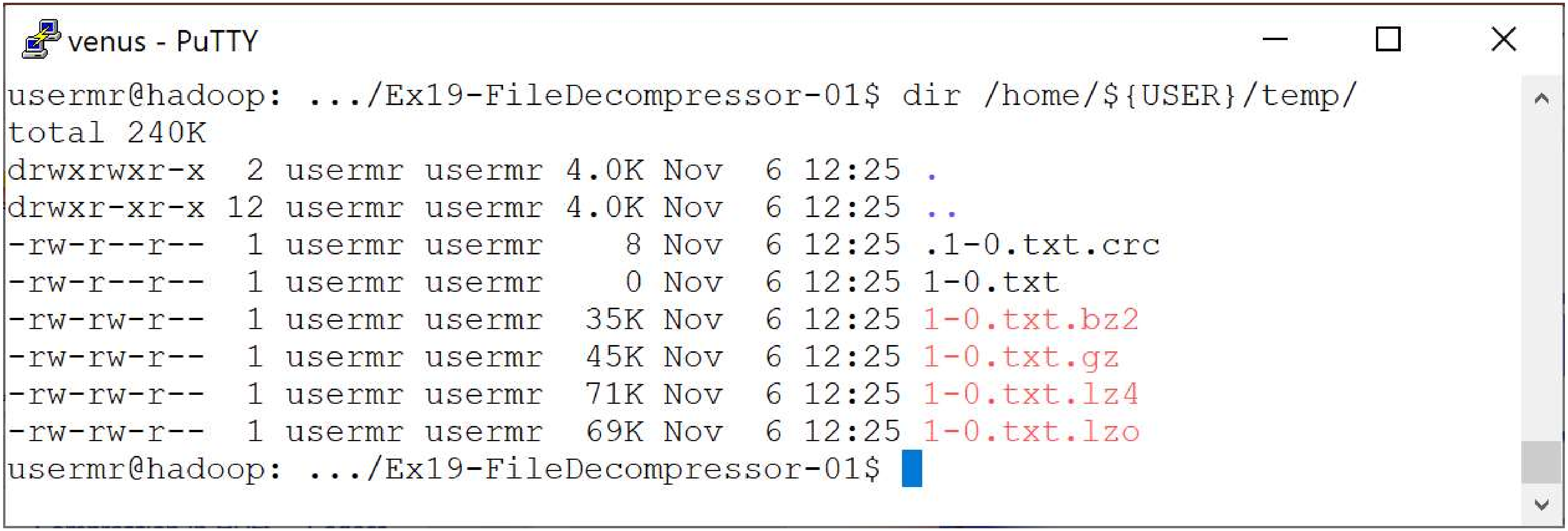
```
usermr@hadoop: .../Ex19-FileDecompressor-01$ dir /home/${USER}/temp/
total 356K
drwxrwxr-x  2 usermr usermr 4.0K Nov  6 12:20 .
drwxr-xr-x 12 usermr usermr 4.0K Nov  6 12:07 ..
-rw-r--r--  1 usermr usermr 119K Nov  6 12:19 1-0.txt
-rw-rw-r--  1 usermr usermr  35K Nov  6 12:19 1-0.txt.bz2
-rw-rw-r--  1 usermr usermr  45K Nov  6 12:19 1-0.txt.gz
-rw-rw-r--  1 usermr usermr  71K Nov  6 12:19 1-0.txt.lz4
-rw-rw-r--  1 usermr usermr  69K Nov  6 12:19 1-0.txt.lzo
usermr@hadoop: .../Ex19-FileDecompressor-01$
```



```
export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex19-FileDecompressor-01/target/Ex19-FileDecompressor-01-2020.2021.SemInv.jar

Running example on input data...
Running for file: file:///home/usermr/temp/1-0.txt.bz2...
hadoop cdle.streams.mr.MyFileDecompressorVer01 file:///home/usermr/temp/1-0.txt.bz2
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_OPTS: invalid variable name
2020-11-06 12:25:14,780 INFO bzip2.Bzip2Factory: Successfully loaded & initialized native-bzip2 library system-native
2020-11-06 12:25:14,781 INFO compress.CodecPool: Got brand-new decompressor [.bz2]
Press ENTER to continue...

Running for file: file:///home/usermr/temp/1-0.txt.gz...
hadoop cdle.streams.mr.MyFileDecompressorVer01 file:///home/usermr/temp/1-0.txt.gz
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_OPTS: invalid variable name
2020-11-06 12:25:18,069 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
2020-11-06 12:25:18,070 INFO compress.CodecPool: Got brand-new decompressor [.gz]
Press ENTER to continue...

Running for file: file:///home/usermr/temp/1-0.txt.lz4...
hadoop cdle.streams.mr.MyFileDecompressorVer01 file:///home/usermr/temp/1-0.txt.lz4
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_OPTS: invalid variable name
2020-11-06 12:25:20,001 INFO compress.CodecPool: Got brand-new decompressor [.lz4]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at org.apache.hadoop.io.compress.BlockDecompressorStream.getCompressedData(BlockDecompressorStream.java:123)
        at org.apache.hadoop.io.compress.BlockDecompressorStream.decompress(BlockDecompressorStream.java:98)
        at org.apache.hadoop.io.compress.DecompressorStream.read(DecompressorStream.java:105)
        at java.io.InputStream.read(InputStream.java:101)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:94)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:68)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:114)
        at cdle.streams.mr.MyFileDecompressorVer01.main(MyFileDecompressorVer01.java:39)
Press ENTER to continue...

Running for file: file:///home/usermr/temp/1-0.txt.lzo...
hadoop cdle.streams.mr.MyFileDecompressorVer01 file:///home/usermr/temp/1-0.txt.lzo
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER01_OPTS: invalid variable name
No codec found for file:///home/usermr/temp/1-0.txt.lzo
Press ENTER to continue...

usermr@hadoop: .../Ex19-FileDecompressor-01$
```

Compression in HDFS – Codecs

Compressing and decompressing streams



```
usermr@hadoop: .../Ex19-FileDecompressor-01$ dir /home/${USER}/temp/
total 240K
drwxrwxr-x  2 usermr usermr 4.0K Nov  6 12:25 .
drwxr-xr-x 12 usermr usermr 4.0K Nov  6 12:25 ..
-rw-r--r--  1 usermr usermr    8 Nov  6 12:25 .1-0.txt.crc
-rw-r--r--  1 usermr usermr    0 Nov  6 12:25 1-0.txt
-rw-rw-r--  1 usermr usermr  35K Nov  6 12:25 1-0.txt.bz2
-rw-rw-r--  1 usermr usermr  45K Nov  6 12:25 1-0.txt.gz
-rw-rw-r--  1 usermr usermr  71K Nov  6 12:25 1-0.txt.lz4
-rw-rw-r--  1 usermr usermr  69K Nov  6 12:25 1-0.txt.lzo
usermr@hadoop: .../Ex19-FileDecompressor-01$
```

Compression in HDFS – Codecs

Compressing and decompressing streams

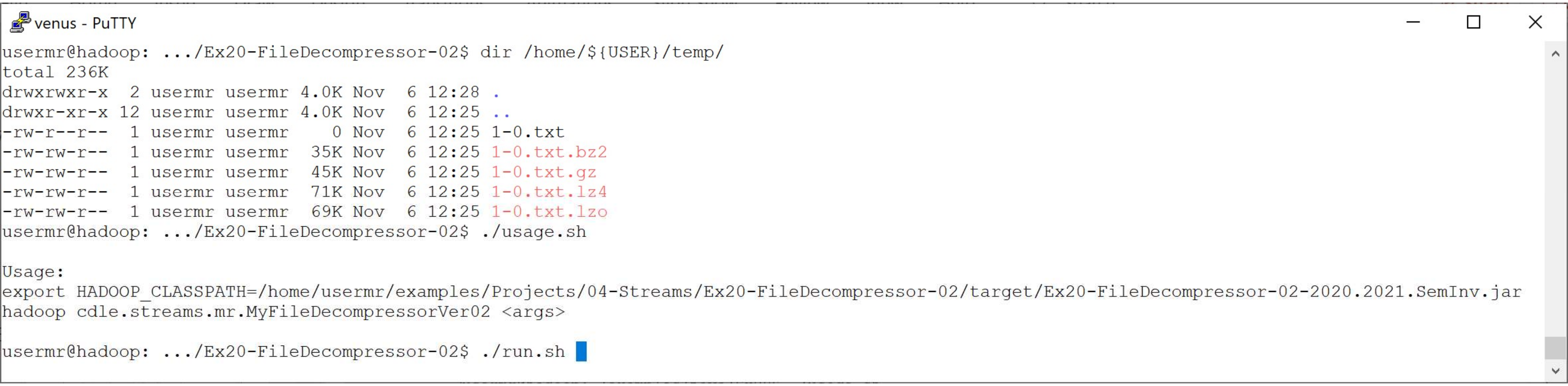
- Modify previous example in order to decompress all the files that exists in the directory passed as argument

Ex20

Challenge

Compression in HDFS – Codecs

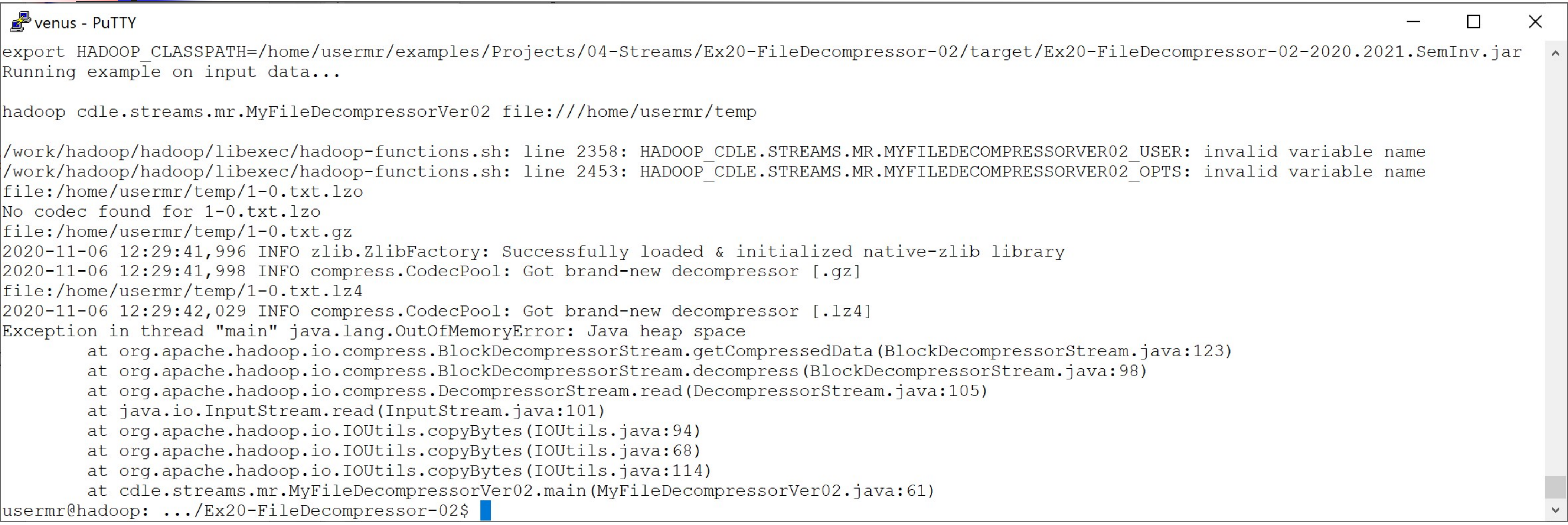
Compressing and decompressing streams



```
usermr@hadoop: .../Ex20-FileDecompressor-02$ dir /home/${USER}/temp/
total 236K
drwxrwxr-x  2 usermr usermr 4.0K Nov  6 12:28 .
drwxr-xr-x 12 usermr usermr 4.0K Nov  6 12:25 ..
-rw-r--r--  1 usermr usermr    0 Nov  6 12:25 1-0.txt
-rw-rw-r--  1 usermr usermr  35K Nov  6 12:25 1-0.txt.bz2
-rw-rw-r--  1 usermr usermr  45K Nov  6 12:25 1-0.txt.gz
-rw-rw-r--  1 usermr usermr  71K Nov  6 12:25 1-0.txt.lz4
-rw-rw-r--  1 usermr usermr  69K Nov  6 12:25 1-0.txt.lzo
usermr@hadoop: .../Ex20-FileDecompressor-02$ ./usage.sh

Usage:
export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex20-FileDecompressor-02/target/Ex20-FileDecompressor-02-2020.2021.SemInv.jar
hadoop cdle.streams.mr.MyFileDecompressorVer02 <args>

usermr@hadoop: .../Ex20-FileDecompressor-02$ ./run.sh
```

Compression in HDFS – Codecs

Compressing and decompressing streams



```
export HADOOP_CLASSPATH=/home/usermr/examples/Projects/04-Streams/Ex20-FileDecompressor-02/target/Ex20-FileDecompressor-02-2020.2021.SemInv.jar
Running example on input data...

hadoop cdle.streams.mr.MyFileDecompressorVer02 file:///home/usermr/temp

/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2358: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER02_USER: invalid variable name
/work/hadoop/hadoop/libexec/hadoop-functions.sh: line 2453: HADOOP_CDLE.STREAMS.MR.MYFILEDECOMPRESSORVER02_OPTS: invalid variable name
file:/home/usermr/temp/1-0.txt.lzo
No codec found for 1-0.txt.lzo
file:/home/usermr/temp/1-0.txt.gz
2020-11-06 12:29:41,996 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library
2020-11-06 12:29:41,998 INFO compress.CodecPool: Got brand-new decompressor [.gz]
file:/home/usermr/temp/1-0.txt.lz4
2020-11-06 12:29:42,029 INFO compress.CodecPool: Got brand-new decompressor [.lz4]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at org.apache.hadoop.io.compress.BlockDecompressorStream.getCompressedData(BlockDecompressorStream.java:123)
        at org.apache.hadoop.io.compress.BlockDecompressorStream.decompress(BlockDecompressorStream.java:98)
        at org.apache.hadoop.io.compress.DecompressorStream.read(DecompressorStream.java:105)
        at java.io.InputStream.read(InputStream.java:101)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:94)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:68)
        at org.apache.hadoop.io.IOUtils.copyBytes(IOUtils.java:114)
        at cdle.streams.mr.MyFileDecompressorVer02.main(MyFileDecompressorVer02.java:61)
usermr@hadoop: .../Ex20-FileDecompressor-02$
```

Compression in HDFS – Codecs

Native libraries

Compression Format	Java implementation?	Native implementation?
DEFLATE	Yes	Yes
gzip	Yes	Yes
bzip2	Yes	Yes
LZO [a)]	No	Yes
LZ4	No	Yes
Snappy	No	yes

- For performance, it is preferable to use a native library for compression and decompression.
- By default, Hadoop looks for native libraries for the platform it is running on, and loads them automatically if they are found

Compression in HDFS – Codecs

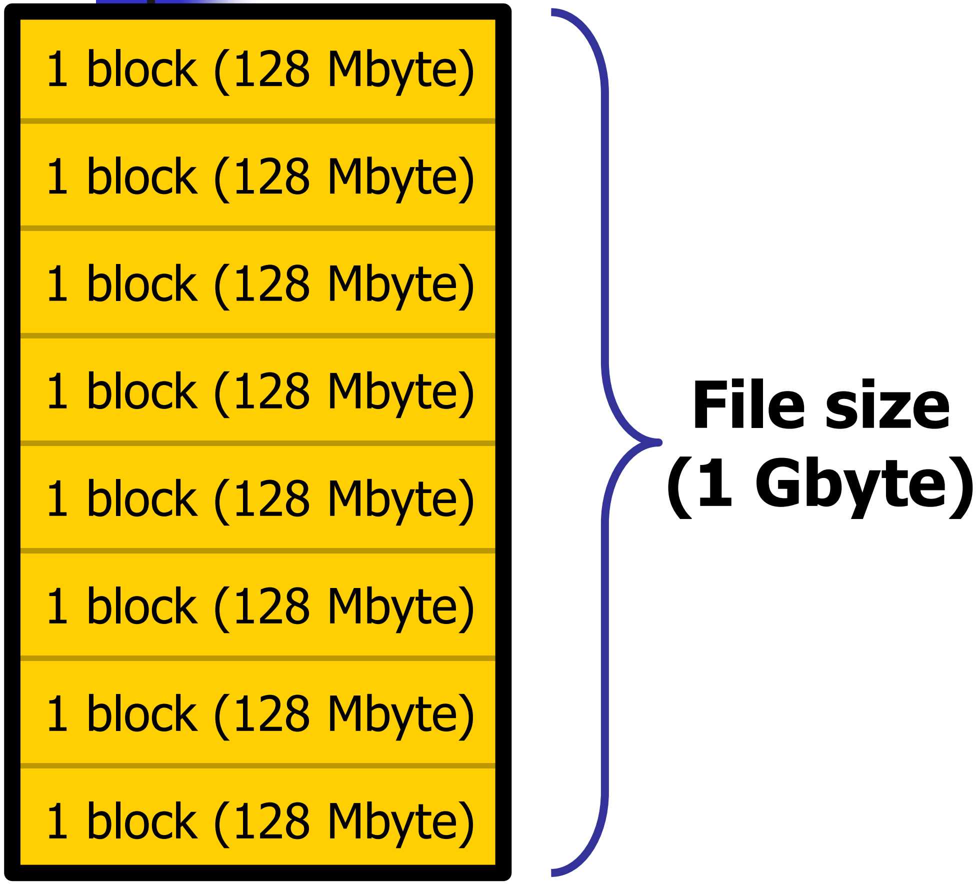
Codec pools

Ex21

```
import ...;

public class MyPooledStreamCompressor {
  public static void main(String[] args) throws Exception {
    Class<?> codecClass = Class.forName(args[0]);
    Configuration cfg = new Configuration();
    CompressionCodec cod;
    cod=(CompressionCodec)ReflectionUtils.newInstance(codecClass,cfg);
    Compressor compressor = null;
    try {
      compressor = CodecPool.getCompressor(codec);
      CompressionOutputStream out;
      out = codec.createOutputStream(System.out, compressor);
      IOUtils.copyBytes(System.in, out, 4096, false);
      out.finish();
    }
    finally { CodecPool.returnCompressor(compressor); }
  }
}
```

Compression in HDFS – Codecs

Compression and Input Splits


1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)
1 block (128 Mbyte)

File size (1 Gbyte)

- How many splits "<=>" how many maps
- What happens if file is compressed?
  - How many blocks?
  - How many splits?

Compression in HDFS – Codecs

Compression and Input Splits

- "In this case, MapReduce will do the right thing and not try to split the gzipped file, since it knows that the input is gzip-compressed (by looking at the filename extension) and that `gzip` does not support splitting. This will work, but at the expense of locality: a single map will process the eight `HDFS` blocks, most of which will not be local to the map. Also, with fewer maps, the job is less granular and so may take longer to run." [1]

Compression in HDFS – Codecs

Which Compression Format Should I Use?

- Try to use compression formats that support splitting. `Avro` datafiles, `ORCFiles` and `Parquet` files both support compression and splitting. Fast compressors such as `LZO`, `LZ4`, or `Snappy` are generally a good choice;
- `bzip2` supports splitting but is fairly slow (when compared to the above mentioned);
- Split input files in chunks and compressed them such that the size of the compressed chunks are approximately equal to the size of an `HDFS` block.

Compression in HDFS – Codecs

Using Compression in MapReduce

Ex02

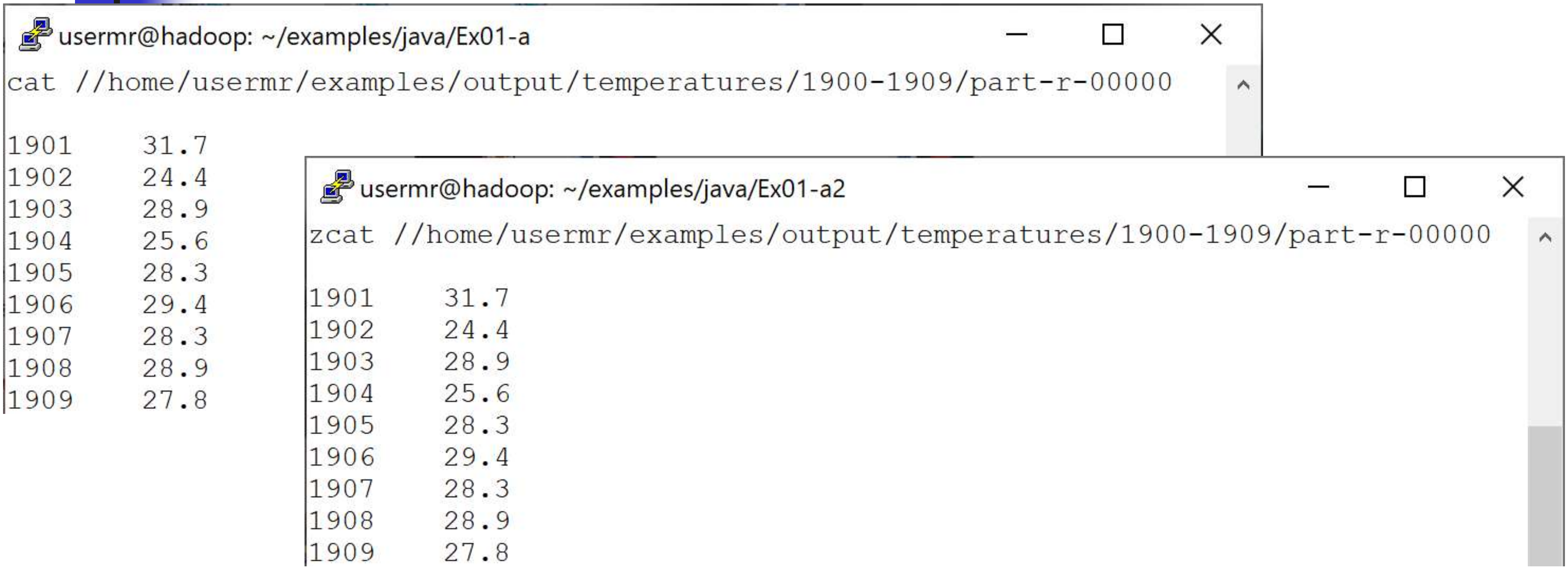
```
import ...;

public class MaxTemperatureApplicationWithCompression {
  public static void main(String[
    job.setJarByClass( MaxTempera                              class );
    job.setJobName( "Max
    FileInputFormat.addI
    FileOutputFormat.set
    job.setMapperClass(
    job.setReducerClass(
    job.setMapOutputKeyC
    job.setMapOutputValu
    job.setOutputKeyClas
    job.setOutputValueClass( FloatWritable.class );
    FileOutputFormat.setCompressOutput(job, true);
    FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);
    System.exit( job.waitForCompletion(true) ? 0 : 1 );
  }
}
```

With the new API

```
Configuration conf = new Configuration();
conf.setBoolean(Job.MAP_OUTPUT_COMPRESS, true);
conf.setClass(
    Job.MAP_OUTPUT_COMPRESS_CODEC,
    GzipCodec.class,
    CompressionCodec.class);
Job job = new Job(conf);
```

Compression in HDFS – Codecs

Using Compression in MapReduce



usermr@hadoop: ~/examples/java/Ex01-a

```
cat //home/usermr/examples/output/temperatures/1900-1909/part-r-00000

1901    31.7
1902    24.4
1903    28.9
1904    25.6
1905    28.3
1906    29.4
1907    28.3
1908    28.9
1909    27.8
```

usermr@hadoop: ~/examples/java/Ex01-a2

```
zcat //home/usermr/examples/output/temperatures/1900-1909/part-r-00000

1901    31.7
1902    24.4
1903    28.9
1904    25.6
1905    28.3
1906    29.4
1907    28.3
1908    28.9
1909    27.8
```

Compression in HDFS – Codecs

Using Compression in MapReduce

- If the MapReduce application use the `Tool` interface it is possible to pass properties to the application using the command line, which can be more convenient than modifying the program every time we must change the compression properties (or others)

Property name	Type	Default value
mapreduce.output.fileoutputformat.compress	boolean	false
mapreduce.output.fileoutputformat.compress.codec	Class name	org.apache.hadoop.io.compress.DefaultCodec
mapreduce.output.fileoutputformat.compress.type	String	RECORD

Data Serialization

HDFS

Serialization

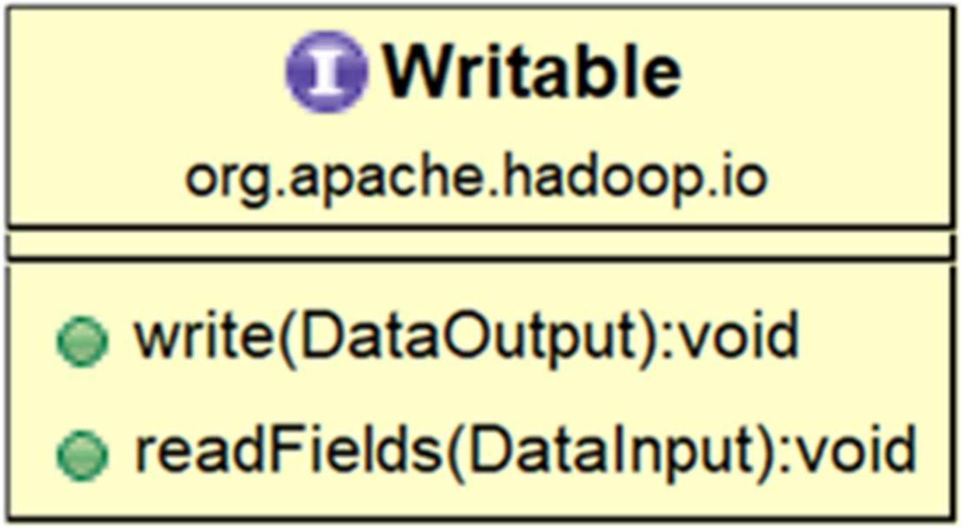
- "*Serialization* is the process of turning structured objects into a byte stream for transmission over a network or for writing to persistent storage" [1]
- "*Deserialization* is the reverse process of turning a byte stream back into a series of structured objects" [1]

Serialization

- Hadoop use RPC (Remote Procedure Calls) to transfer data between nodes
- A RPC protocol serializes data before send it and deserializes it on the receiving node
- In general the serialization/de serialization should be:
  - Compact
  - Fast
  - Extensible
  - Interoperable

Hadoop serialization

- Hadoop uses its own serialization format, `Writables`, which is certainly compact and fast, but not so easy to extend or use from languages other than Java



Writable

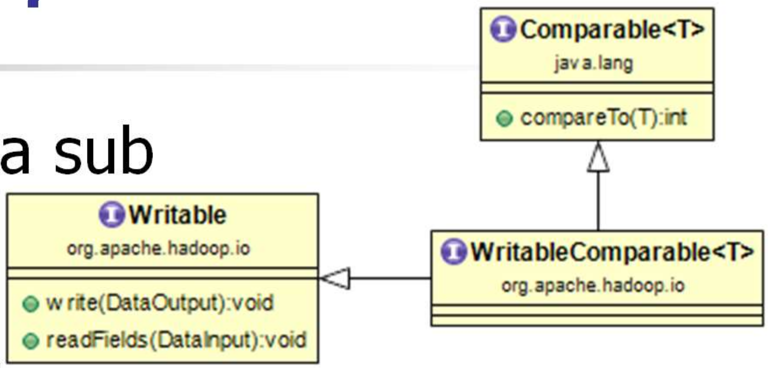
org.apache.hadoop.io

write(DataOutput):void

readFields(DataInput):void

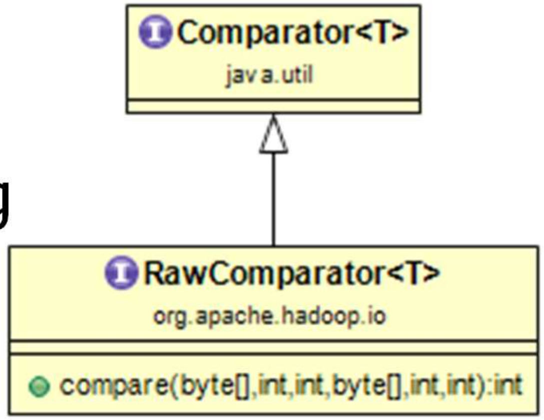
Hadoop serialization

- `WritableComparable` interface, is just a sub interface of the `Writable` and `java.lang.Comparable` interfaces



- Comparison of types is crucial for MapReduce. One optimization that Hadoop provides is the `RawComparator` extension of Java’s `Comparator`

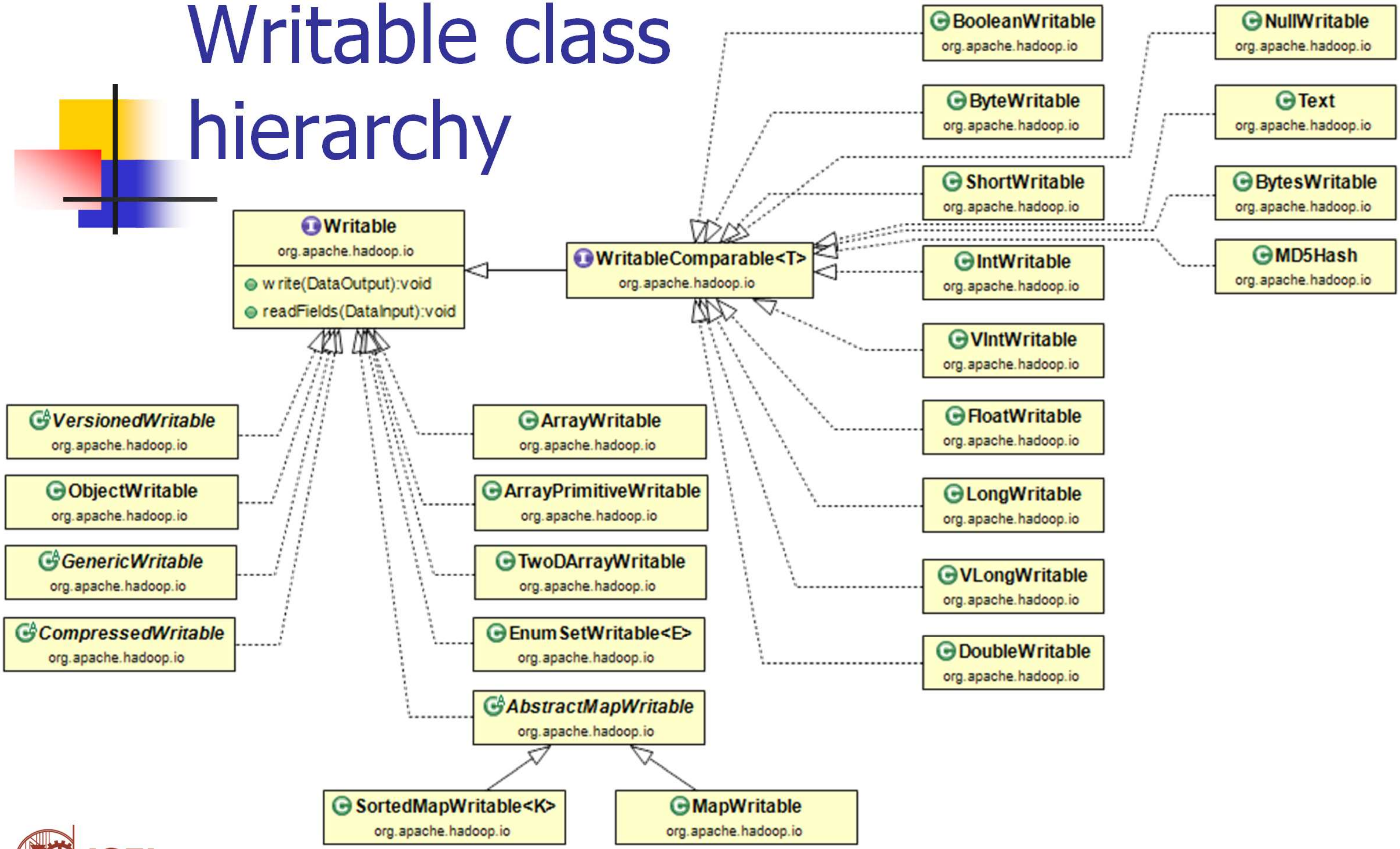
- This interface permits implementors to compare records read from a stream without deserializing them into objects, thereby avoiding any overhead of object creation



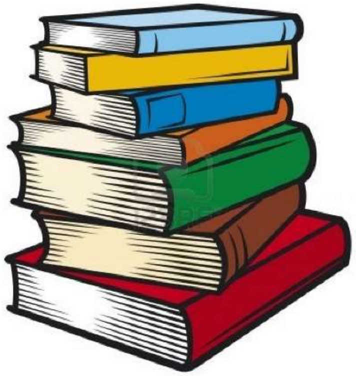
Hadoop serialization

Java primitive type	Writable implementation	Serialized size (bytes)
boolean	`BooleanWritable`	1
byte	`ByteWritable`	1
short	`ShortWritable`	2
int	`IntWritable`	4
	`VIntWritable`	1 → 5
float	`FloatWritable`	4
long	`LongWritable`	8
	`VLongWritable`	1 → 9
double	`DoubleWritable`	8

Writable class hierarchy



References



[1] T. White, "Hadoop - The Definitive Guide" 4th Edition", ISBN-13: 9781491901632, ISBN-10: 1491901632