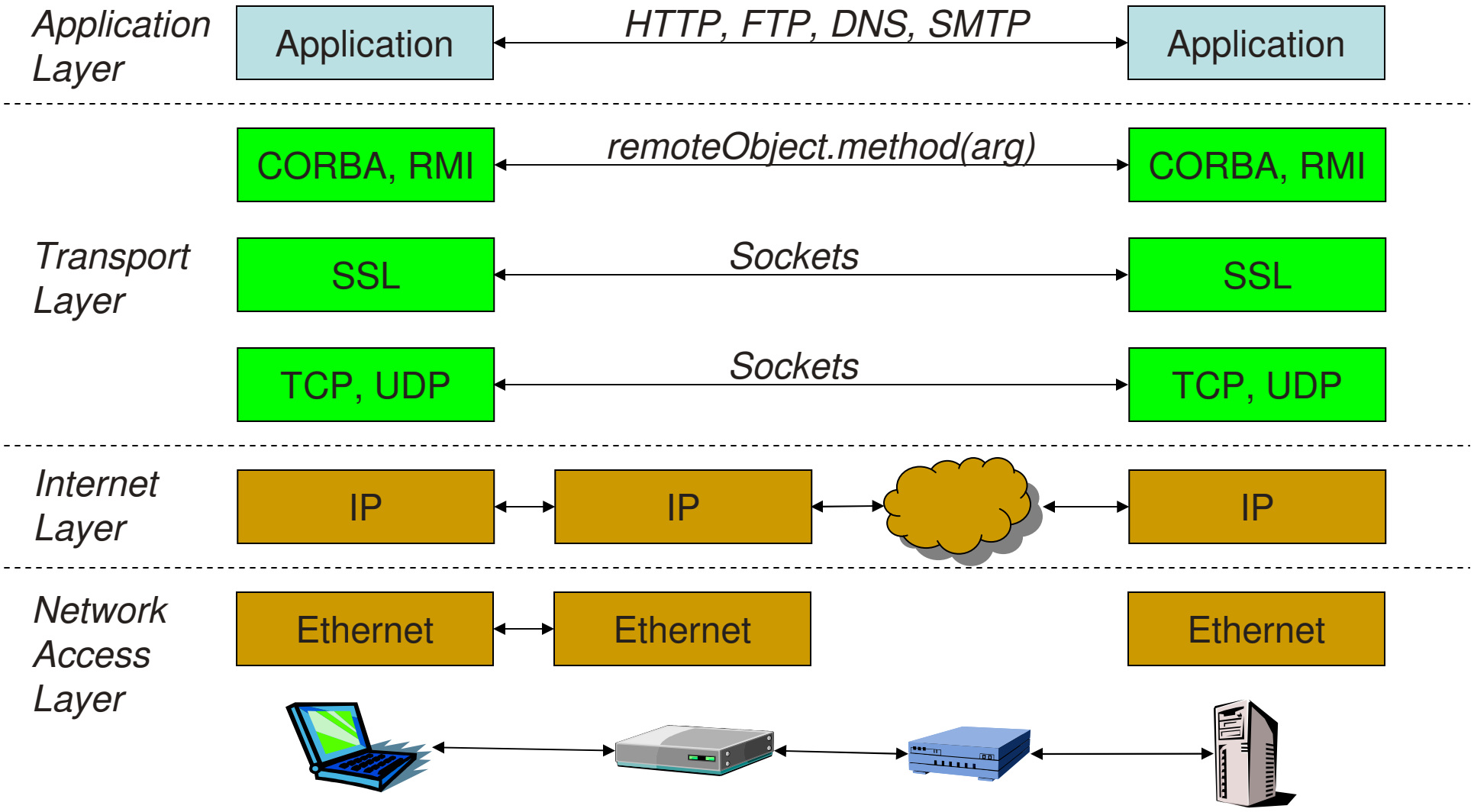
Java Sockets

Alexander V. Konstantinou

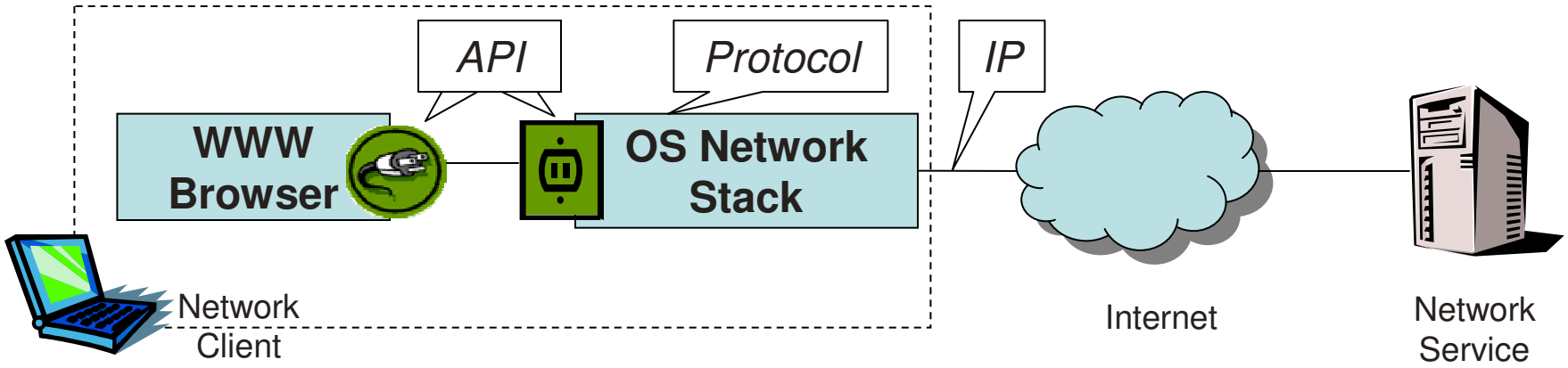
CS4119 – Computer Networks
Columbia University
Spring 2003

Coordinates



What Are Sockets?

- Programming abstraction (API)
  - Equivalent: File I/O API
- Network service end-point
- Originated as BSD UNIX concept
  - Now part of modern OS & languages



File vs. Socket API

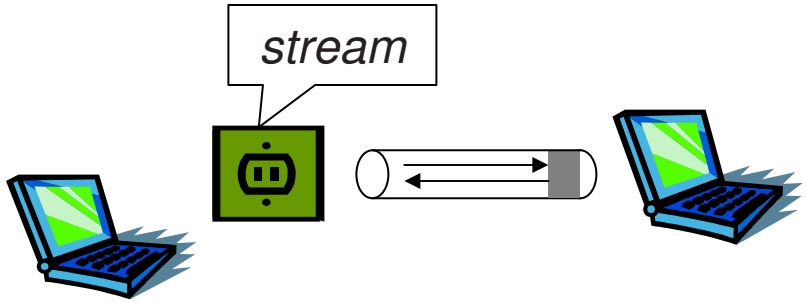
File	Network
open(filename) → stream	open(destination) → stream
read(bytes)	read(bytes)
write(bytes)	write(bytes)
close()	close()

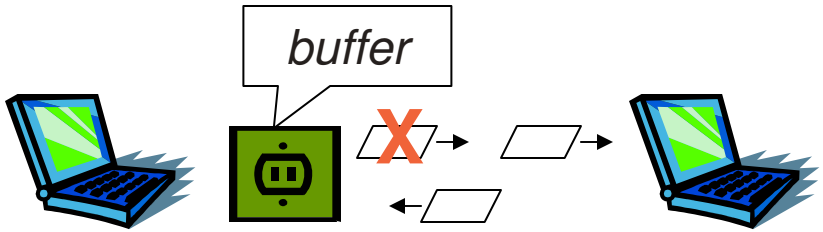
```
File f = new File("test");
FileInputStream fis = new FileInputStream(f);
char c = fis.read();
fis.close();
```

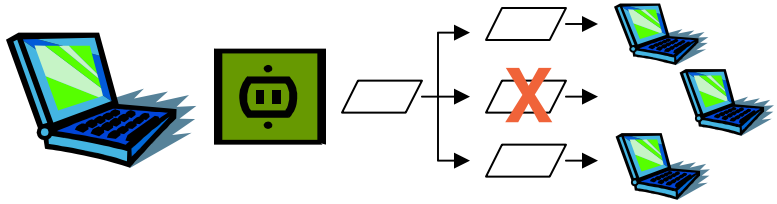
```
Socket s = new Socket("www.cs.columbia.edu", 80);
InputStream is = s.getInputStream();
char c = is.read();
s.close();
```

Socket Types

- Streaming sockets
  - Point-to-point
  - Connection based
    - Bi-directional
  - Reliable, in-order (FIFO)
- Datagram sockets
  - Point-to-{point | multipoint}
  - Connectionless
  - Non-reliable
  - Out-of-order delivery
  - Limited payload

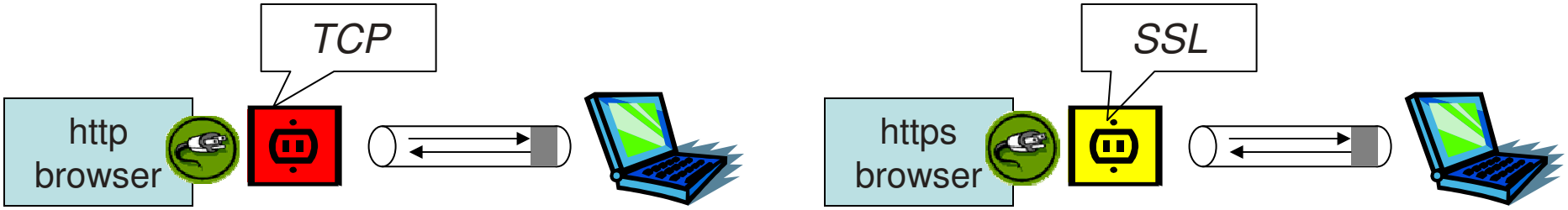






Sockets Advantages

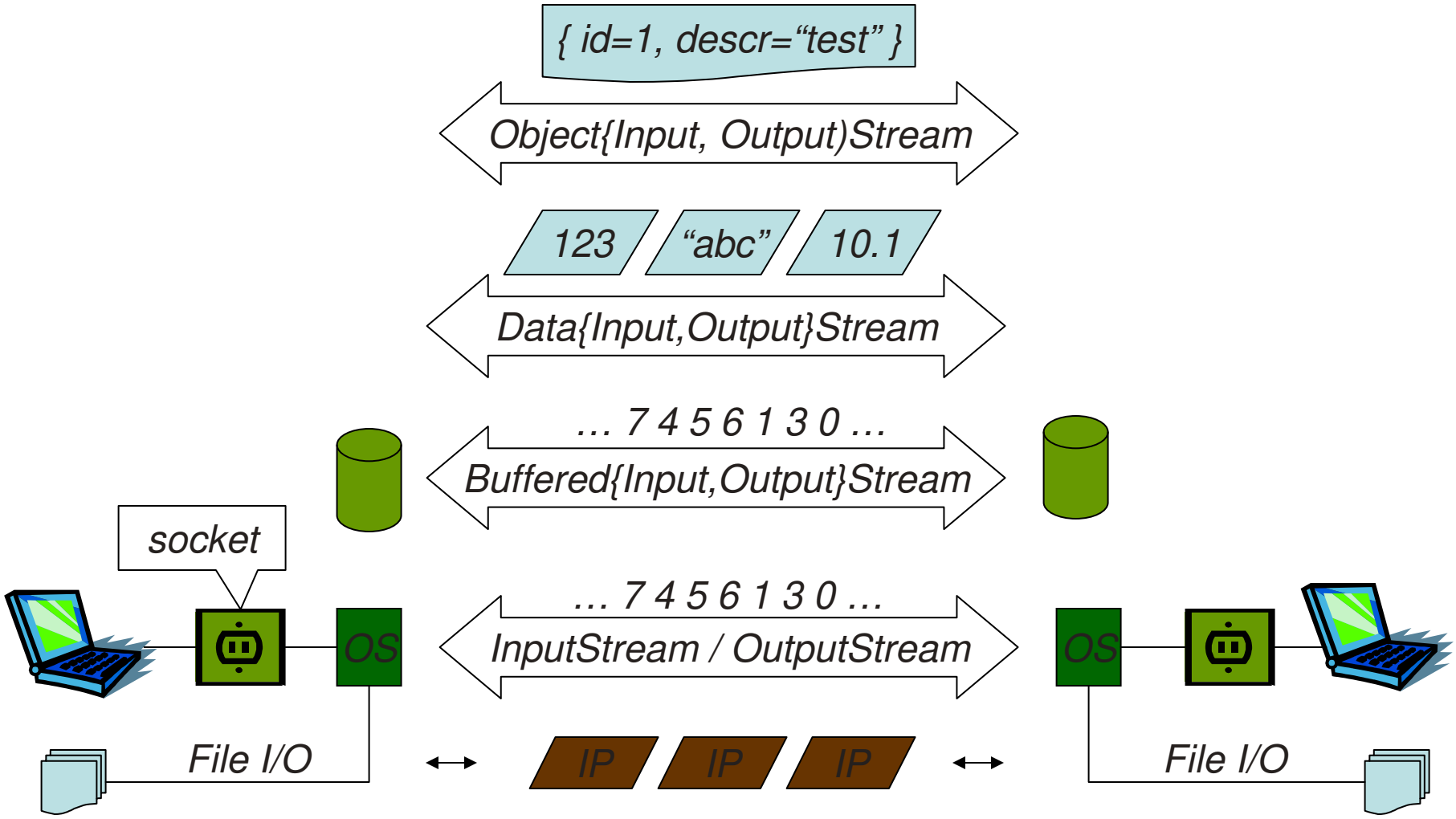
- Common programming abstraction
  - Separate the networking stack
- Application flexibility
  - SSL instead of vanilla TCP
  - Logging & performance measurements
  - Tunneling



Java Socket API Overview

Class	Type	Protocol
java.net.Socket	Stream	TCP client
java.net.ServerSocket	Stream	TCP server
java.net.DatagramSocket	Datagram	UDP client/server
javax.net.ssl.SSLSocket	Stream	SSL client
javax.net.ssl.SSLServerSocket	Stream	SSL server

Java I/O Streams

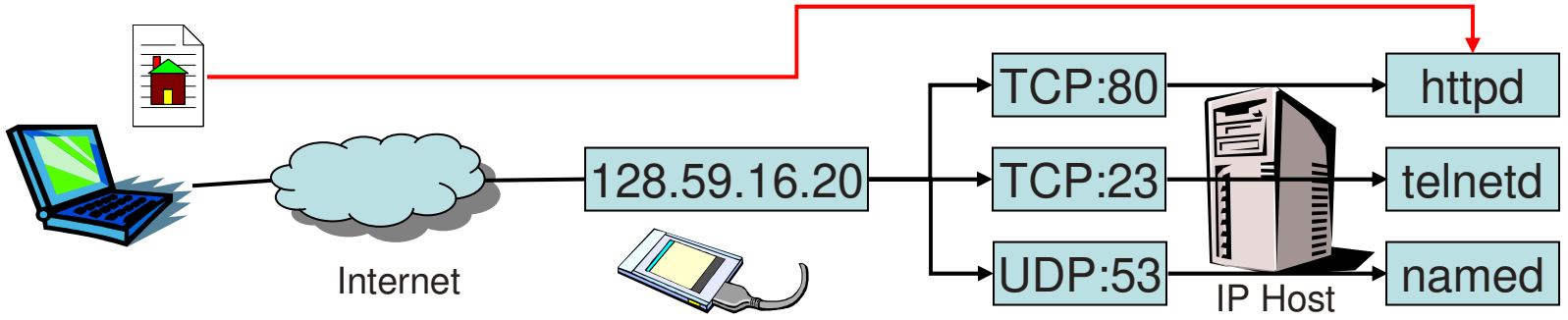


Network Communication

- Byte-based
  - Byte-stream or byte-array
- Protocol types
  - Text-based: HTTP, SMTP, SOAP, …
  - Bit-based: DNS, LDAP, SNMP …
- Issues
  - Text: encoding (Unicode 16bit → 8bit)
  - Bit: byte ordering (little/big endian)

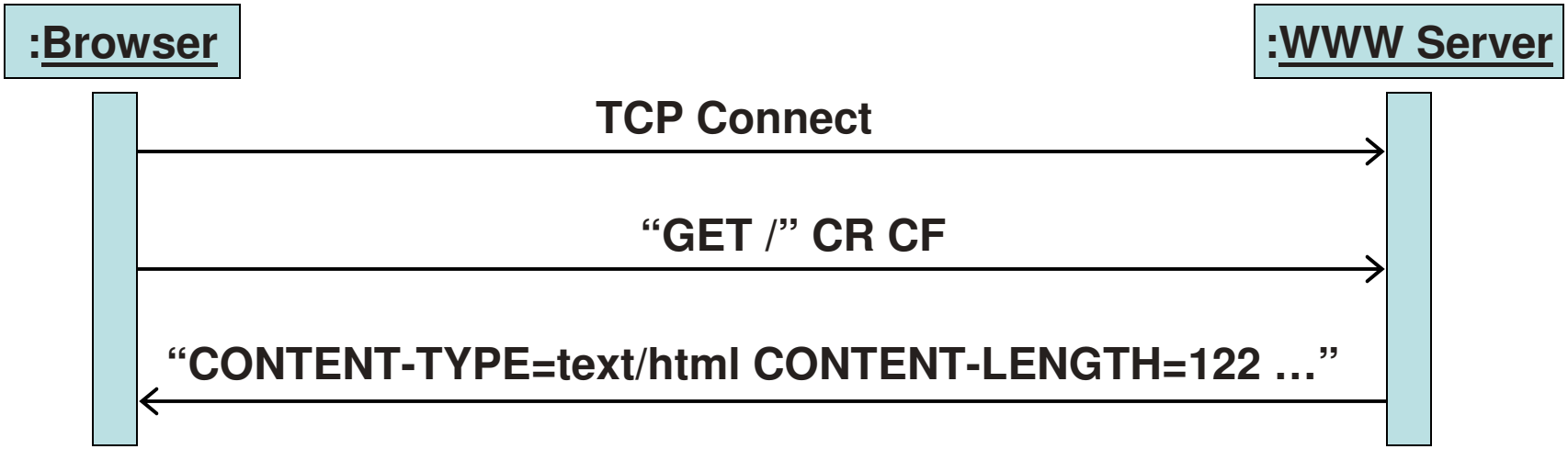
Network Ports

- Internet host address
  - Example: 128.59.16.1
- How to offer multiple services?
  - Mailbox approach
  - Port number addressing [1-65535] (2^{16})
  - Example: 128.59.16.1:TCP:80



Example: HTTP Client

- Trivial HTTP 1.0 Client
  - Application-layer protocol over TCP
  - Text-based protocol



HTTP Client (No Error Handling)

```
Socket socket = new Socket("www.cs.columbia.edu", 80);

BufferedOutputStream ostream =
  new BufferedOutputStream(socket.getOutputStream());

byte[] request = "GET /\r\n".getBytes("ISO-8859-1");
ostream.write(request);
ostream.flush();

BufferedInputStream istream =
  new BufferedInputStream(socket.getInputStream());

byte[] buffer = new byte[4096];
int count = istream.read(buffer);
while(count != -1) {
  System.out.print(new String(buffer, 0, count));
  count = istream.read(buffer);
}
socket.close();
```

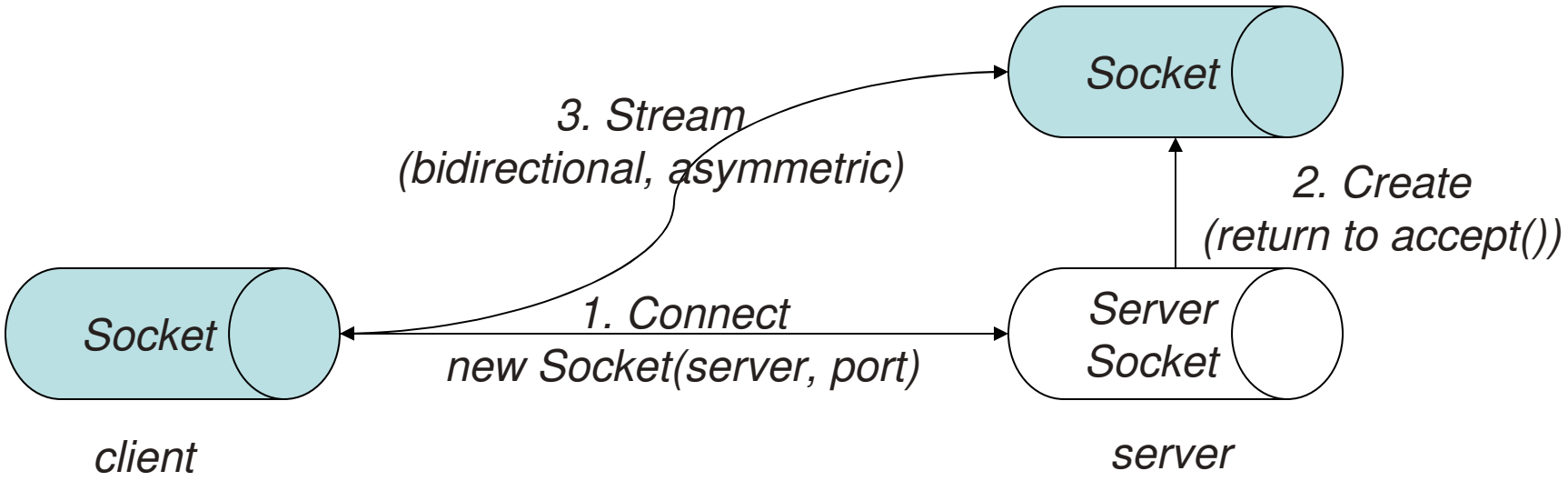
Stream Socket Error Handling

- Networks fail!
  - Applications must be aware of failures
- Sockets are scarce OS resources
  - Don't rely on garbage collection; close() when done!

```
Socket socket = null;
try {
  socket = new Socket("www.cs.columbia.edu", 80);
  // Perform I/O
} catch (Throwable e) {
  // Application-specific error recovery
} finally {
  socket.close();
}
```

Server Stream Sockets

- Bind to port (e.g. 80)
- Wait for incoming TCP connection
  - Encapsulate request as Socket object
  - Socket pair now communicate



HTTP Server (Trivial)

```
ServerSocket server = new ServerSocket(8080);

while(active) {
  Socket socket = server.accept();

  BufferedReader reader = new BufferedReader
   (new InputStreamReader
     (new BufferedInputStream(socket.getInputStream())));

  String request = reader.readLine();

  BufferedOutputStream ostream =
    new BufferedOutputStream(socket.getOutputStream());

  ostream.write("HTTP/1.0 404 HTTP NOT FOUND\r\n");
  ostream.flush();
  socket.close();
}
server.close();
```

Java Threads

- Why parallelize?
  - Prevent blocking from malformed requests
  - Deal with writing to slow clients
  - Interleave processing/file reading
- How?
  - Multiple threads
  - Shared memory
  - Interleaved execution

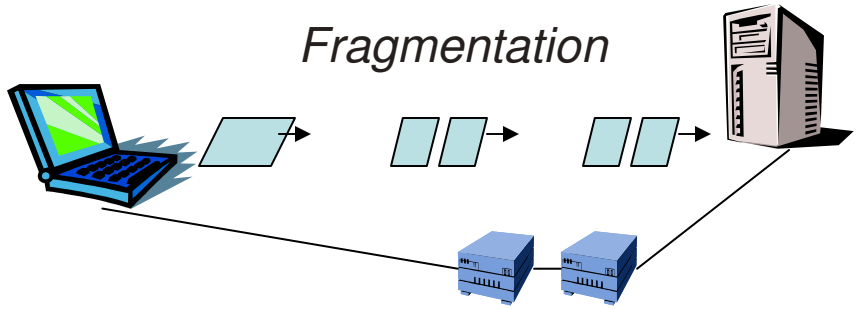
Threaded Server Pattern

```
ServerSocket server =
  new ServerSocket(80);

while(isActive) {
  Socket socket =
    server.accept();

  Handler handler =
    new Handler(socket);

  handler.start();
}
```

```
public class Handler
                    extends Thread {

  protected final Socket socket;

  public Handler(Socket socket) {
    this.socket = socket;
  }

  public void run() {
      try {
        // Protocol Impl.
      } finally {
        socket.close();
      }
  }
}
```

Datagram Sockets

- Connectionless
- Unreliable
- Limited payload
  - Theoretical 65508 bytes
  - Practical 512 bytes (IETF RFC)
- When to use
  - Simple request-response protocol
  - Small requests/replies (fit in datagram)
  - Stateless/re-entrant servers
  - Real-time streaming



UDP Client Example

```
DatagramSocket socket = new DatagramSocket();

byte[] data = "hello world".getBytes();

DatagramPacket packet = new DatagramPacket
       (data,
        data.length,
        InetAddress.getByName("localhost"),
        1234);

socket.send(packet);

data = "followup call".getBytes();
packet.setData(data);

socket.send(packet);
```

UDP Server Example

```
DatagramSocket socket = new DatagramSocket(1234);

byte[] buffer = new byte[512];

DatagramPacket packet =
  new DatagramPacket(buffer, buffer.length);

while(true) {
  socket.receive(packet);

  String msg =
    new String(packet.getData(), 0, packet.getLength()));

  System.out.println(new java.util.Date() + ": " + msg);

  packet.setLength(buffer.length); // !!! Necessary !!!
}
```

Multicast

- One-to-many transmission
  - Special range of IP addresses
    - 224.0.0.0 – 239.255.255.255
- Same socket API as UDP
- Not universally available

SSL Sockets

- Secure communications
  - Layered over TCP
- Drop-in replacement (subclasses)
  - Socket → SSLSocket
  - ServerSocket → SSLServerSocket
- Key management
  - `keytool` command-line utility
  - Creates public/private keys
    - Stored in “keystores”
  - Java VM invocation with keystore info

Non-Blocking Java I/O

- Thread scaling issues
  - E.g. HTTP server with 1000s connections
- JDK 1.4 introduced non-blocking I/O
  - Perform all processing in one thread
  - java.nio.* package

Socket FAQs

- Wrong host/port/protocol
  - E.g “localhost”, “127.0.0.1”, “www” vs. “www.foo.com”
- Ignoring I/O exceptions
- Streams:
  - flush() when done
  - use buffered reader/stream
  - close connections when done
- Datagrams
  - packet.setLength() before reuse
  - Test in lossy environment

Socket FAQs (2)

- Wait-forever
  - Use Socket.setSoTimeout()
  - Catch InterruptedIOException
- Byte-alignment issues
  - Network byte order
- Security restrictions (applets)
- Unix restricts non-root servers to ports >1024

Advanced Issues

- Thread pools
  - Resource management
- Binding to specific interfaces
  - Binding to a specific interface (security)
- Swing applications
  - Swing is single threaded
  - Network operations in non-Swing thread
  - Use SwingUtilities.invokeLater()