Node.js: Asynchronous I/O for Fun and Profit

Stefan Tilkov @ QCon London 2011

Stefan Tilkov

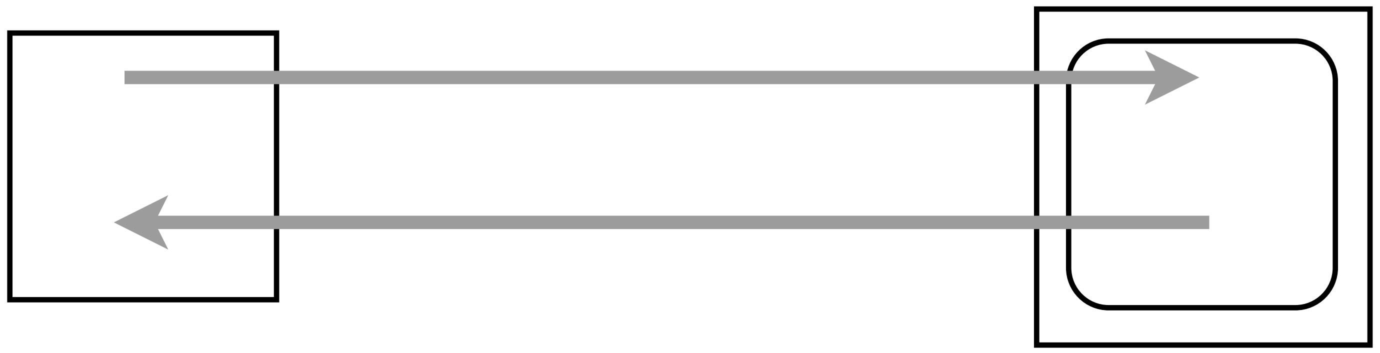
@stilkov

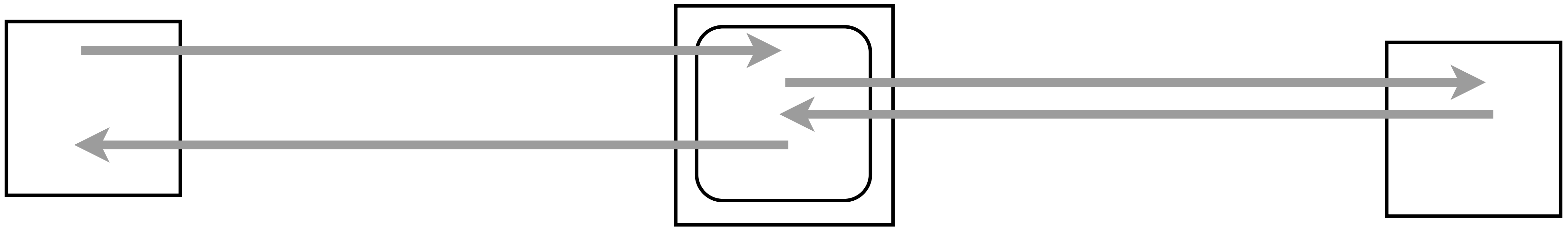
stefan.tilkov@innoq.com

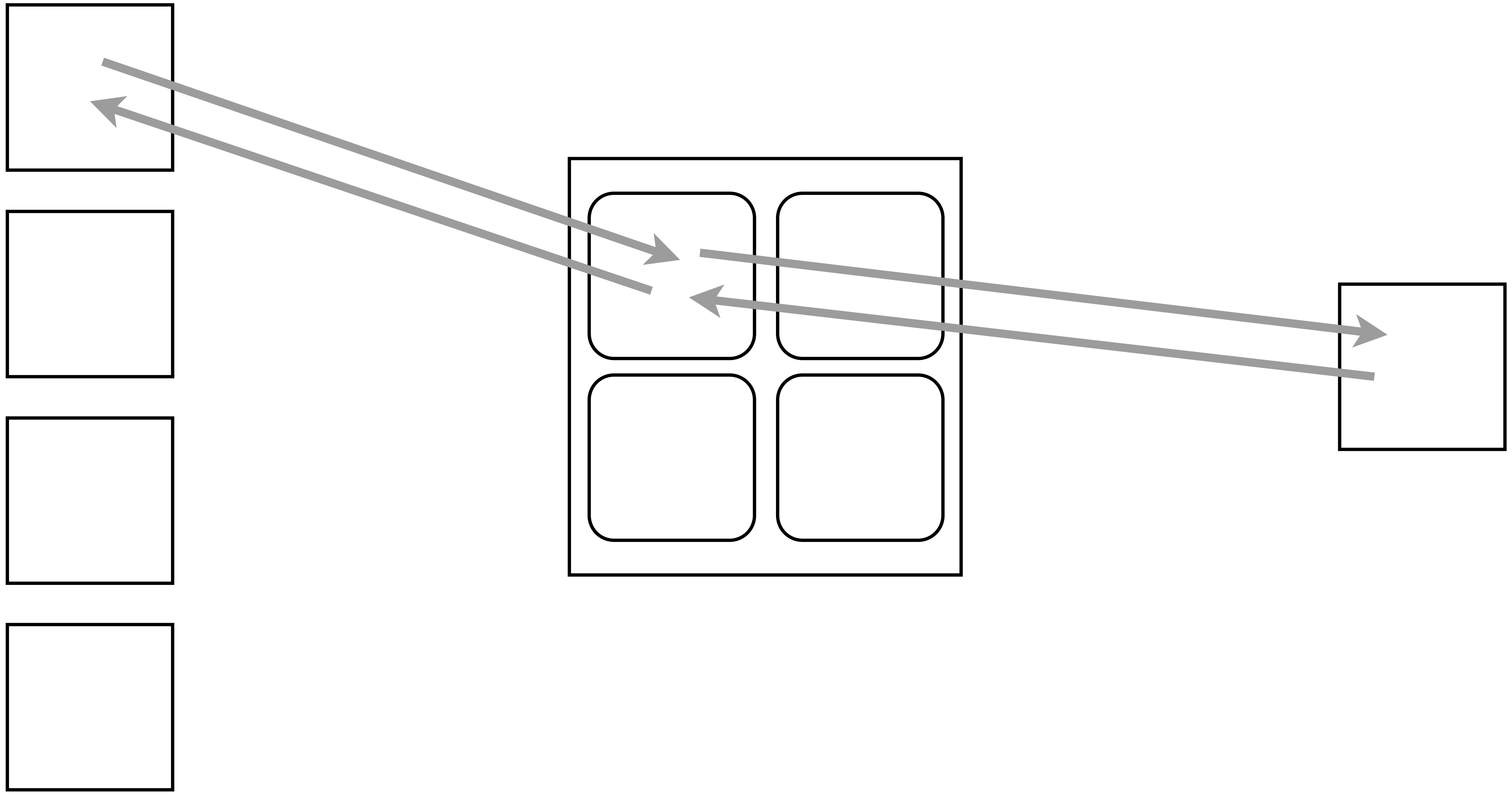
innoQ

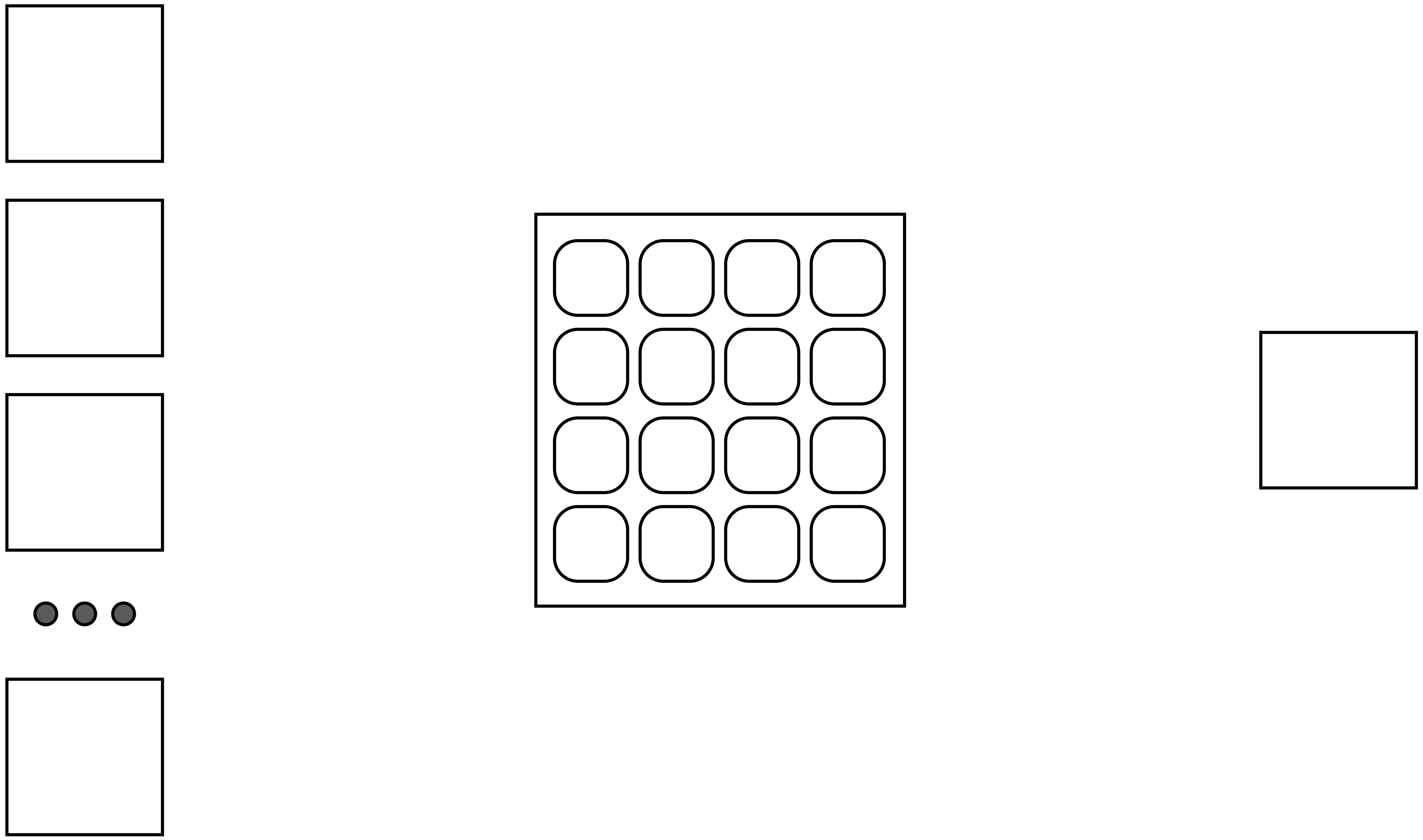
http://www.innoq.com

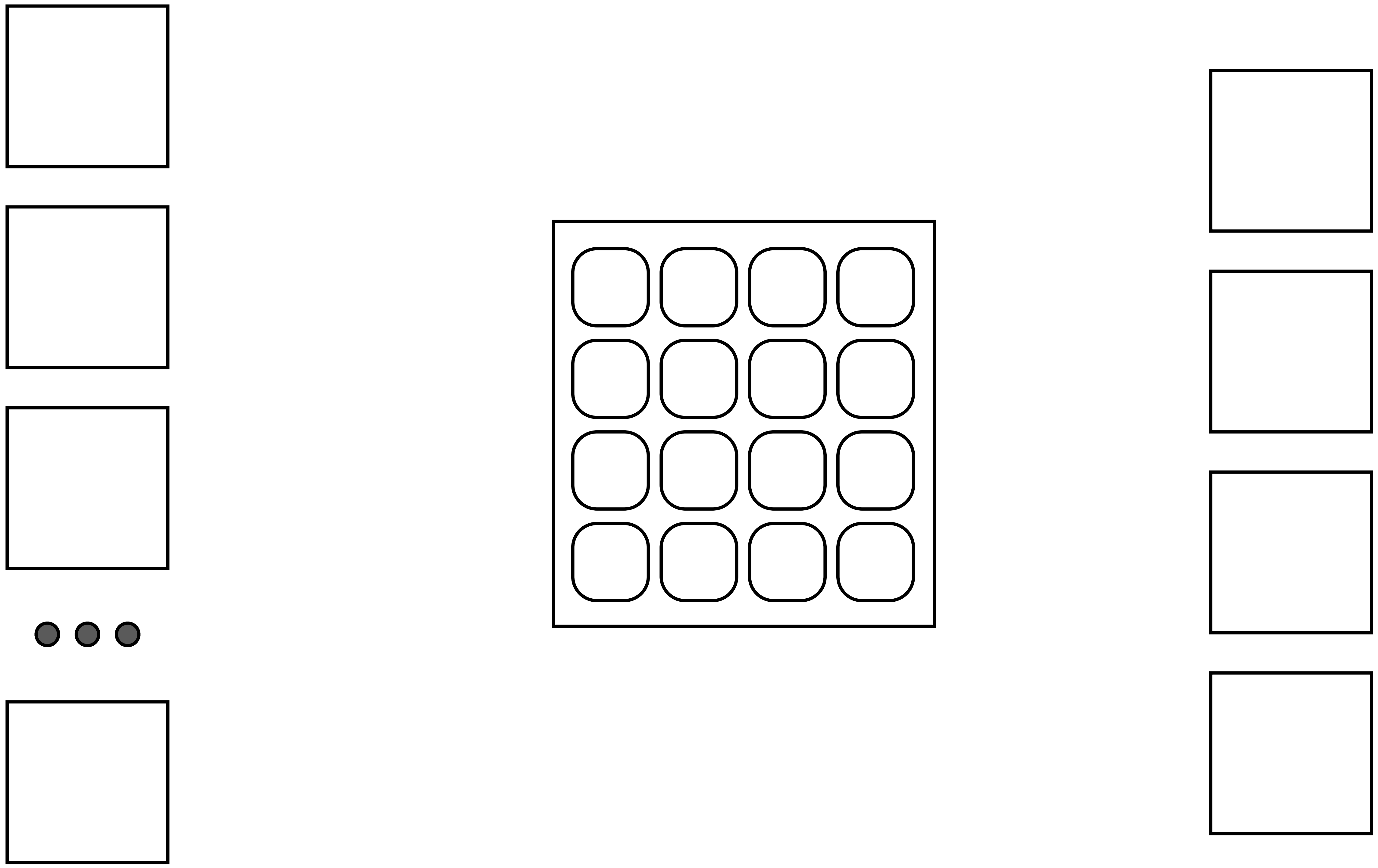
Concurrent Request Processing

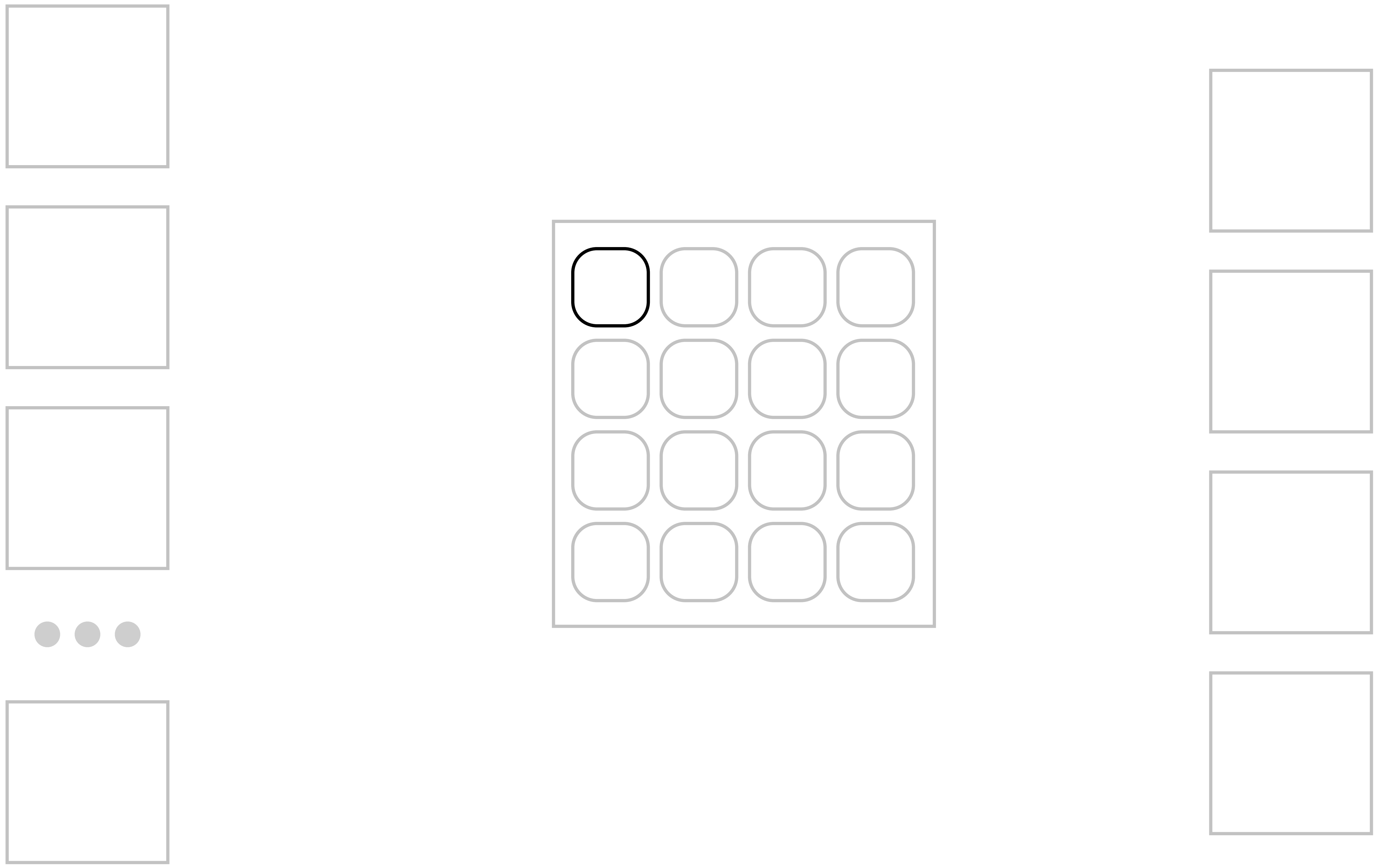












read request

read request

parse request

read request

parse request

process

read request

parse request

process

send backend request

read request

parse request

process

send backend request

read backend answer

read request

parse request

process

send backend request

read backend answer

process

read request

parse request

process

send backend request

read backend answer

process

format response

read request
parse request
process
send backend request
read backend answer
process
format response
send response

read request

parse request

process

send backend request

read backend answer

process

format response

send response

Blocking I/O Problems

Blocking I/O Problems

Thread starvation

Blocking I/O Problems

Thread starvation

Memory utilization

Blocking I/O Problems

Thread starvation

Memory utilization

External dependencies

Blocking I/O Problems

Thread starvation

Memory utilization

External dependencies

Cascading problems

Blocking I/O Problems

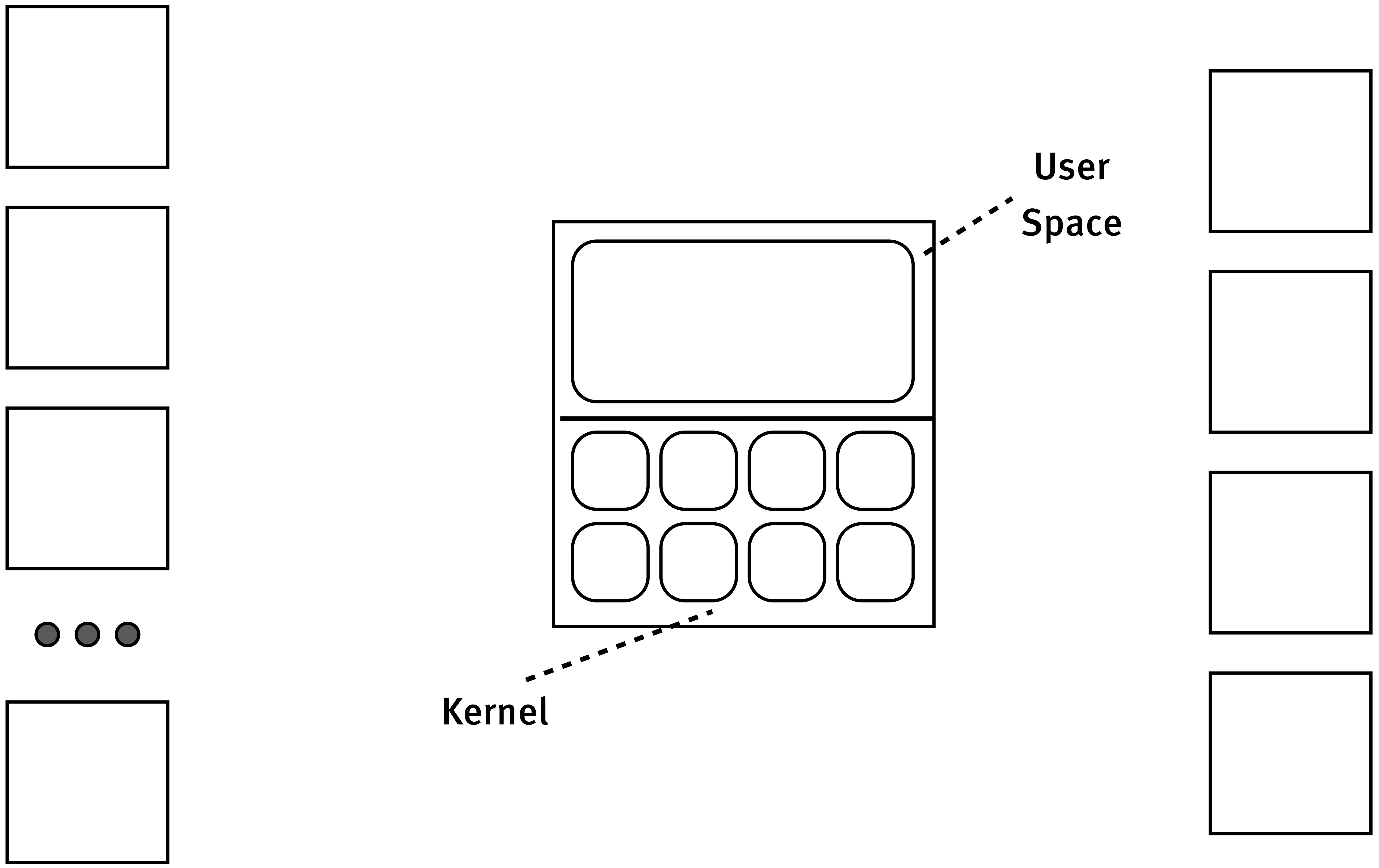
Thread starvation

Memory utilization

External dependencies

Cascading problems

Non-streaming approach


User
Space
Kernel

Event Loop

```
while (true)
  ready_channels = select(io_channels)
  for (channel in ready_channels)
    performIO(channel)
```

Async I/O Characteristics

Async I/O Characteristics

Program always running

Async I/O Characteristics

Program always running

I/O-bound calls never block

Async I/O Characteristics

Program always running

I/O-bound calls never block

Kernel handles I/O

Async I/O Characteristics

Program always running

I/O-bound calls never block

Kernel handles I/O

Notification via events

Async I/O Characteristics

Program always running

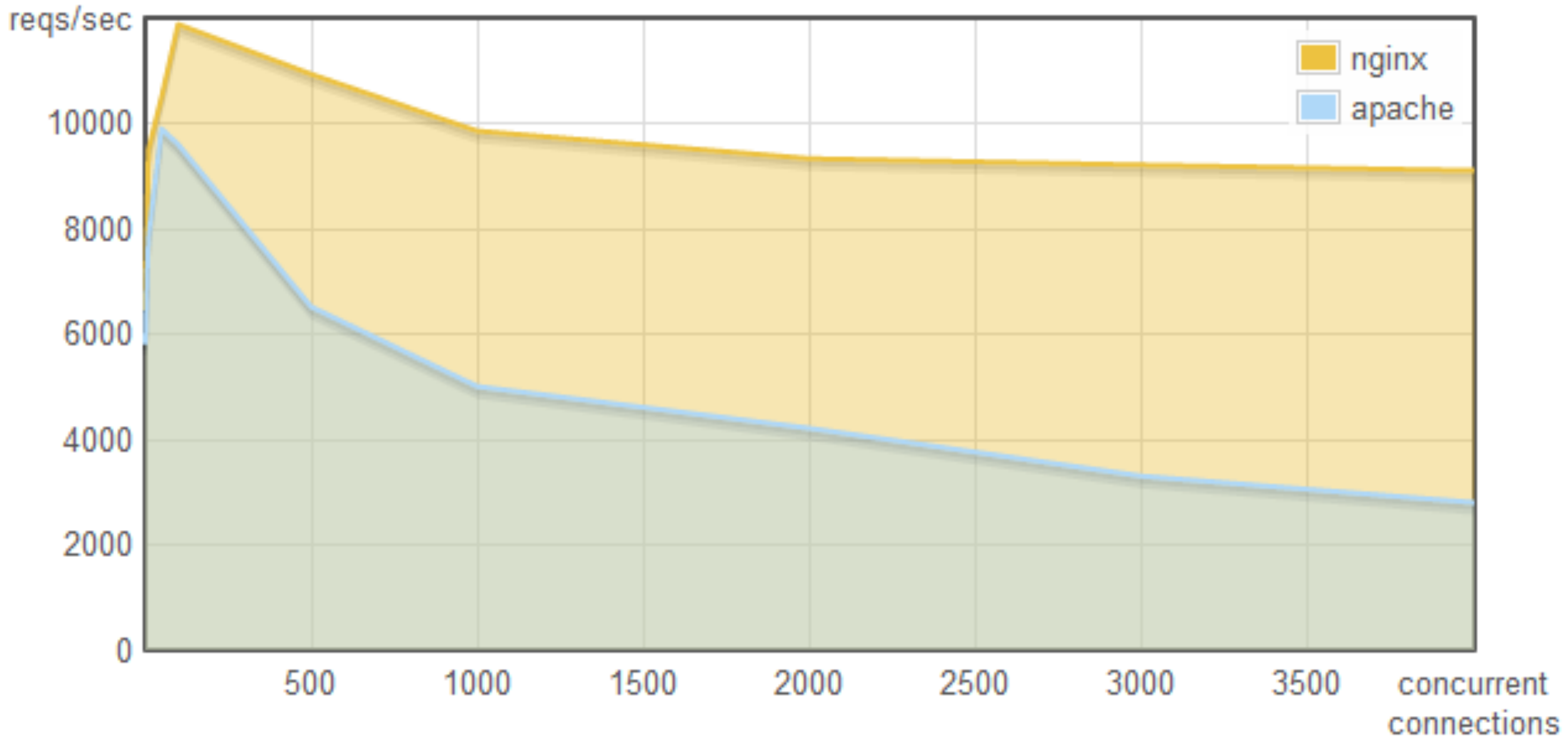
I/O-bound calls never block

Kernel handles I/O

Notification via events

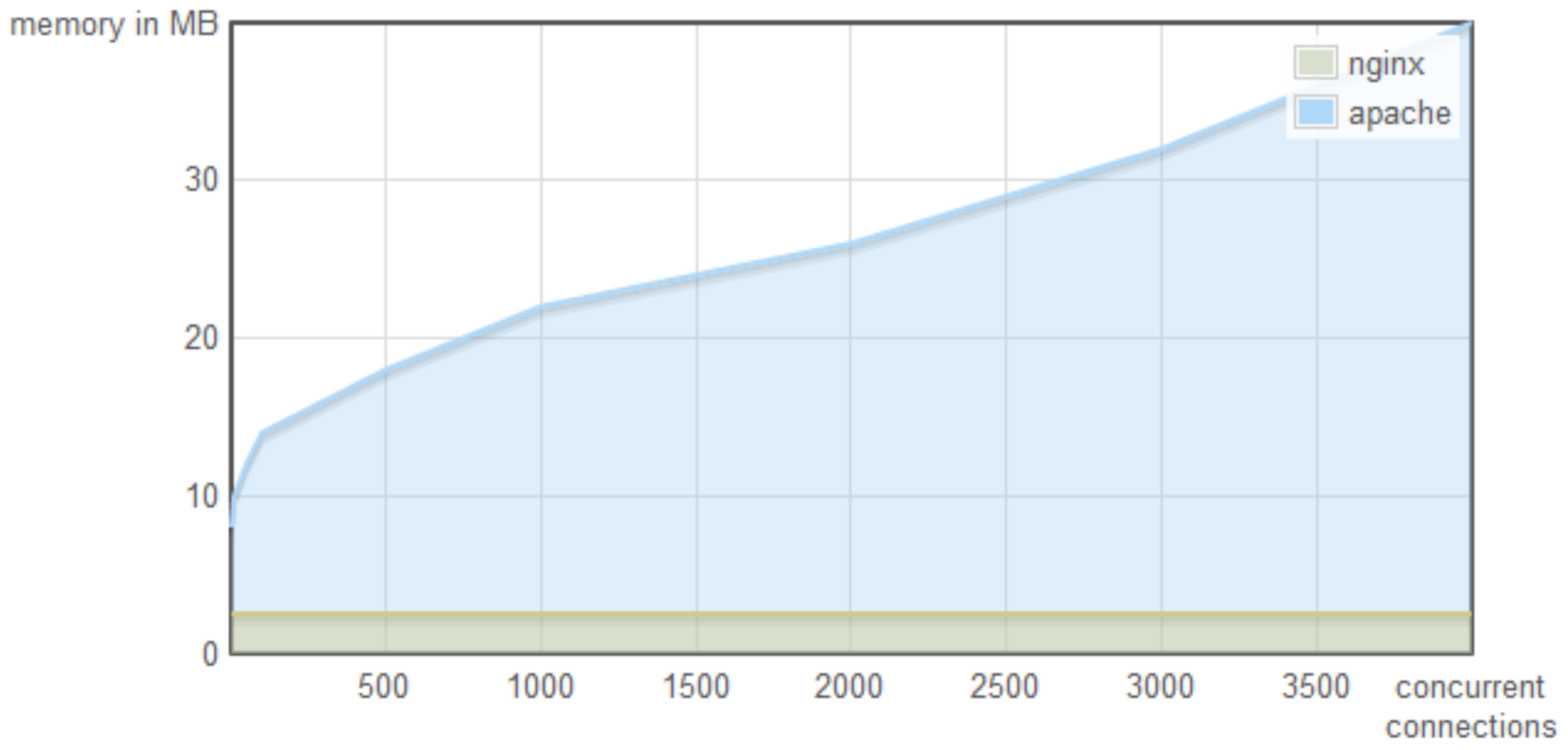
Used for timers, file I/O, net I/O, ...

requests/second



http://blog.webfaction.com/a-little-holiday-present

memory



http://blog.webfaction.com/a-little-holiday-present

/dev/poll

select()

aio_*()

poll()

kqueue()

epoll()

java.nio

.NET I/O Completion Ports

Async I/O Perception

innoQ

Async I/O Perception

Not widely known

Async I/O Perception

Not widely known

Low level

Async I/O Perception

Not widely known

Low level

Hard to use

Async I/O Perception

Not widely known

Low level

Hard to use

Exception rather than rule

JavaScript

JavaScript Perception

innoQ

JavaScript Perception

“Toy language”

JavaScript Perception

“Toy language”

Incompatible

JavaScript Perception

“Toy language”

Incompatible

Inherent design problems

JavaScript Perception

“Toy language”

Incompatible

Inherent design problems

Low Performance

http://commons.wikimedia.org/wiki/File:Audi_S5_V8_FSI_engine.jpg

http://commons.wikimedia.org/wiki/File:Ateles_paniscus_-Brazil-8.jpg

innoQ

The JavaScript Arms Race

jQuery
write less, do more.

dōjō toolkit

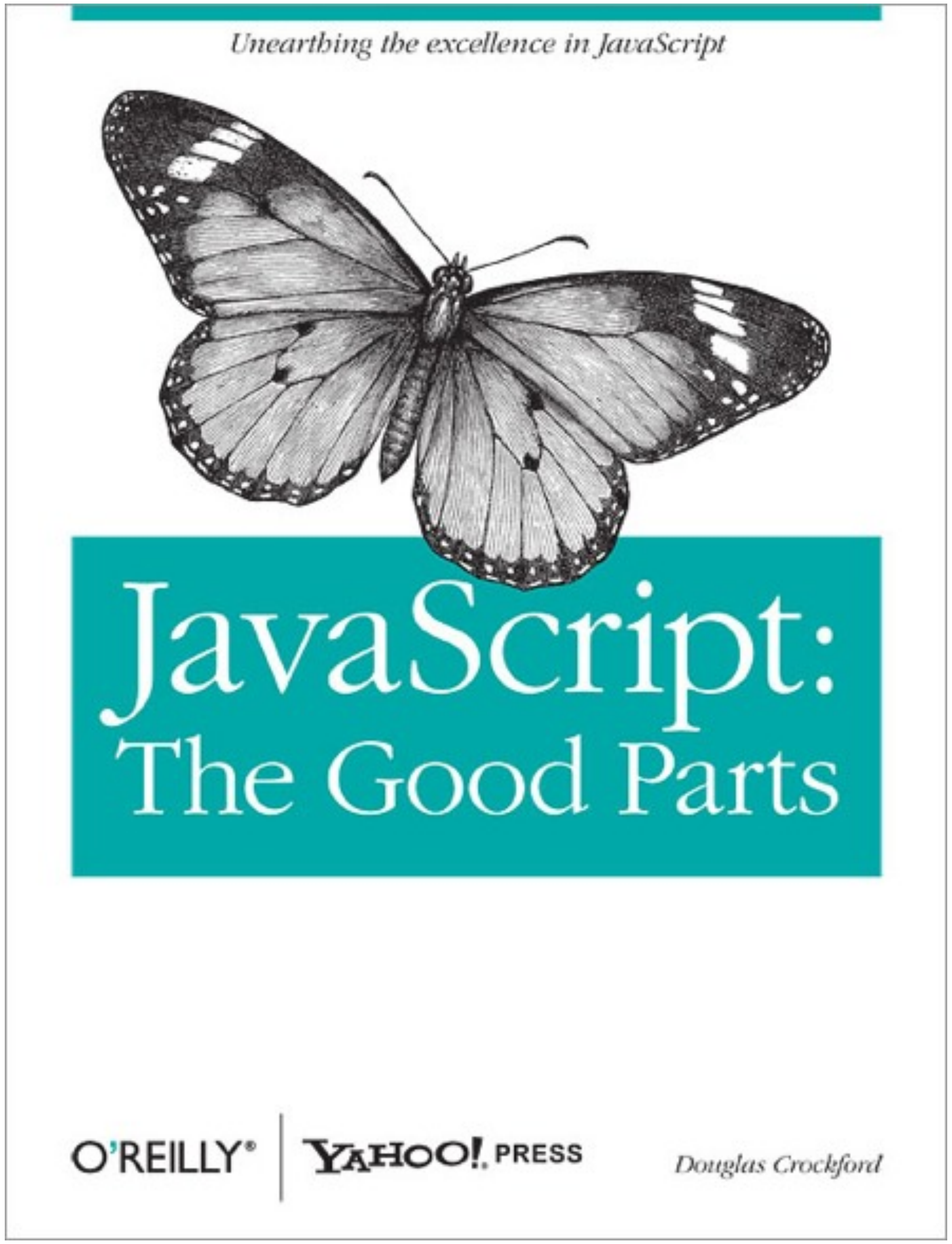
prototype

YUI LIBRARY

Ext JS

innoQ

CommonJS


Unearthing the excellence in JavaScript

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS

Douglas Crockford

http://oreilly.com/catalog/9780596517748

HTML

5

JavaScript Today

JavaScript Today

Popular & widely used

JavaScript Today

Popular & widely used

Often mandatory

JavaScript Today

Popular & widely used

Often mandatory

Fast

JavaScript Today

Popular & widely used

Often mandatory

Fast

Compatible

JavaScript Today

Popular & widely used

Often mandatory

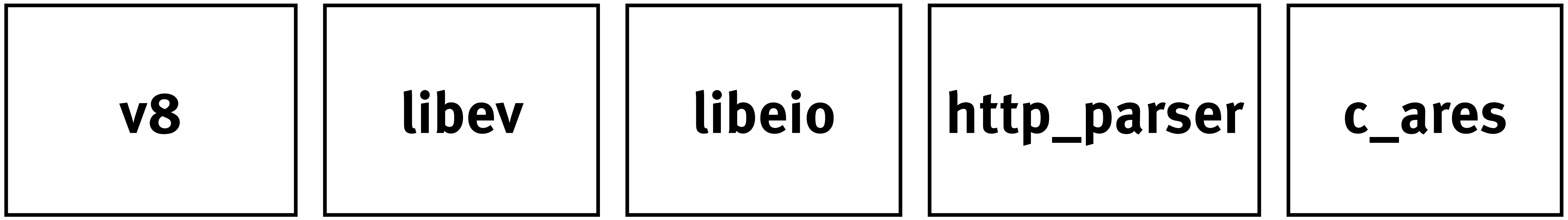
Fast

Compatible

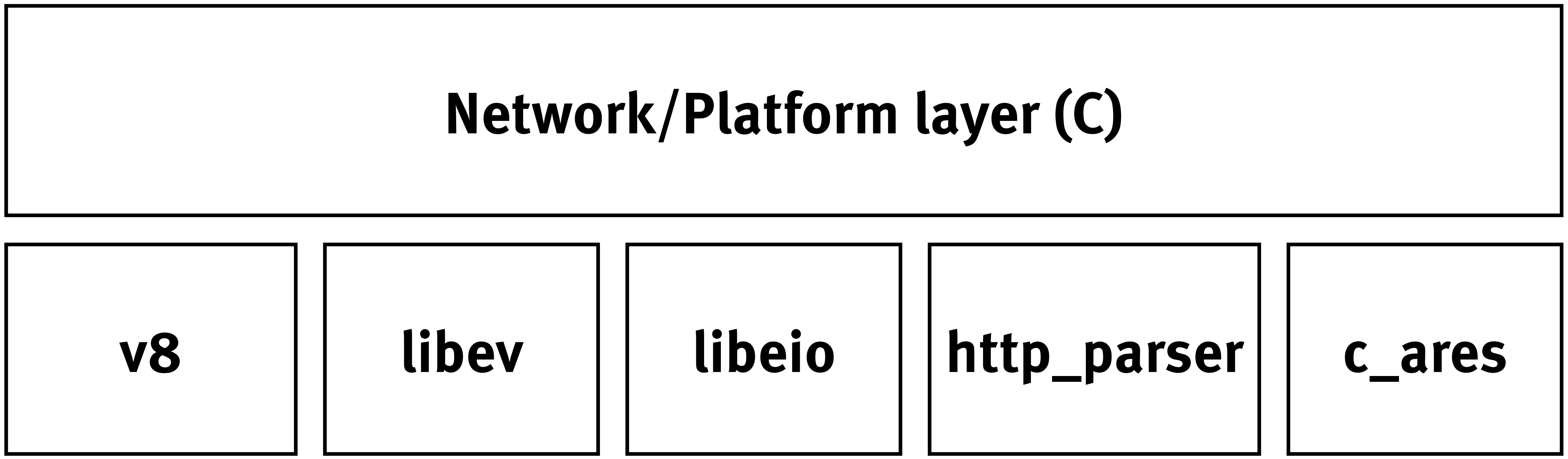
Best practices

Node.js

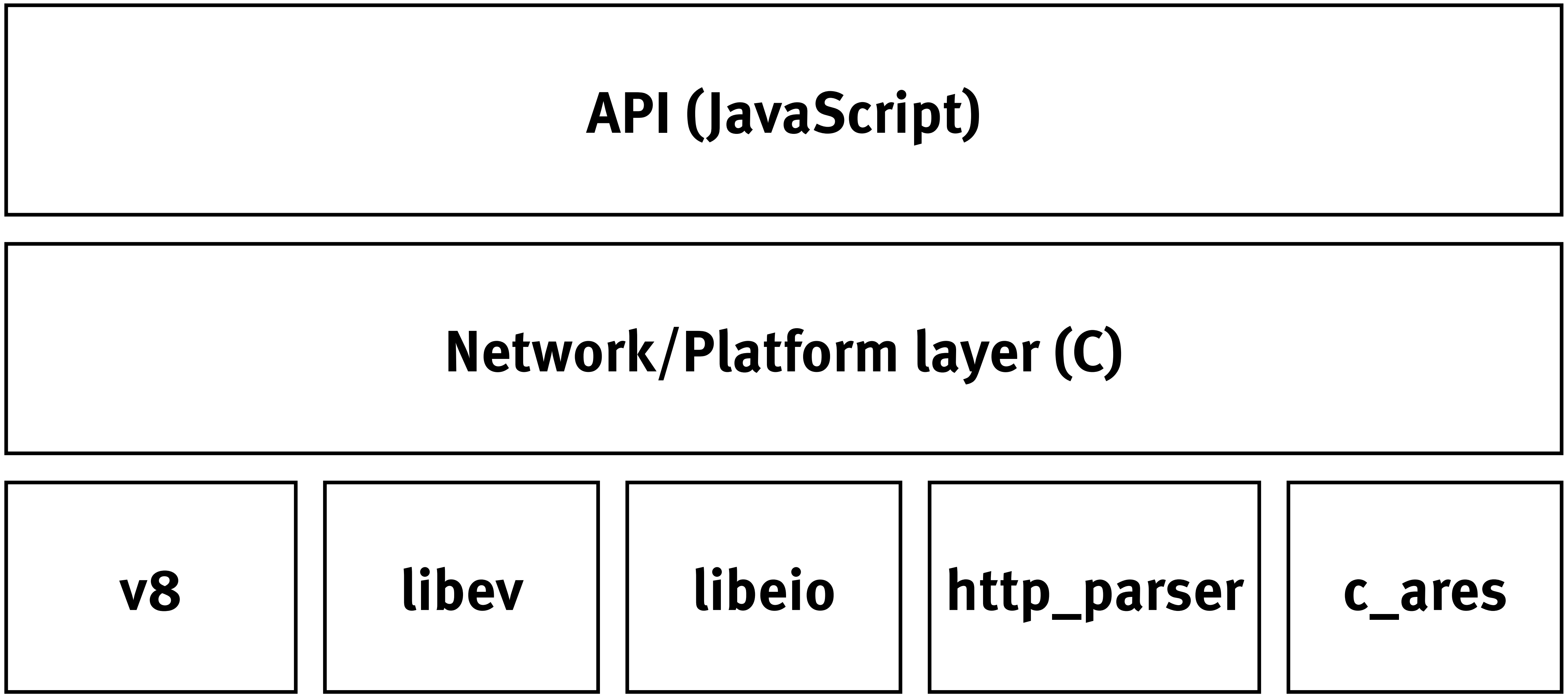
Node.js Architecture



Node.js Architecture



Node.js Architecture



High-performance network runtime, using JavaScript as a high-level DSL

```
var net = require('net');

var server = net.createServer(function (socket) {
  socket.write("Echo server\r\n");
  socket.pipe(socket);
})

server.listen(8124, "127.0.0.1");
```

Code samples: http://github.com/stilkov/node-samples

echo.js

```
var net = require('net');

var server = net.createServer(function (socket) {
  socket.write("Echo server\r\n");
  socket.setEncoding('ascii');
  socket.on('data', function(data) {
    socket.write(data.toUpperCase());
  });
});

server.listen(8124, "127.0.0.1");
```

echo-upcase.js

innoQ

```
var sys = require("sys"), http = require("http"), url = require("url"),
    path = require("path"),  fs = require("fs");

var dir = process.argv[2] || './public';
var port = parseFloat(process.argv[3]) || 8080;
sys.log('Serving files from ' + dir + ', port is ' + port);

http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), dir, uri);
    path.exists(filename, function(exists) {
        if(exists) {
            fs.readFile(filename, function(err, data) {
                response.writeHead(200);
                response.end(data);
            });
        } else {
            sys.log('File not found: ' + filename);
            response.writeHead(404);
            response.end();
        }
    });
}).listen(port);
```

file-server.js

```
Concurrency Level:      100
Time taken for tests:   6.000 seconds
Complete requests:      10000
Failed requests:        0
Write errors:           0
Keep-Alive requests:    0
Total transferred:      710781 bytes
HTML transferred:       150165 bytes
Requests per second:    1666.72 [#/sec] (mean)
Time per request:       59.998 [ms] (mean)
Time per request:       0.600 [ms] (mean, across all concurrent requests)
Transfer rate:          115.69 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    8   8.3      5      57
Processing:     1   51  44.4     40     307
Waiting:        0   43  43.5     30     302
Total:          1   59  44.8     50     316

Percentage of the requests served within a certain time (ms)
  50%     50
  66%     58
  75%     68
  80%     73
  90%    112
  95%    174
  98%    206
  99%    224
 100%    316 (longest request)
```

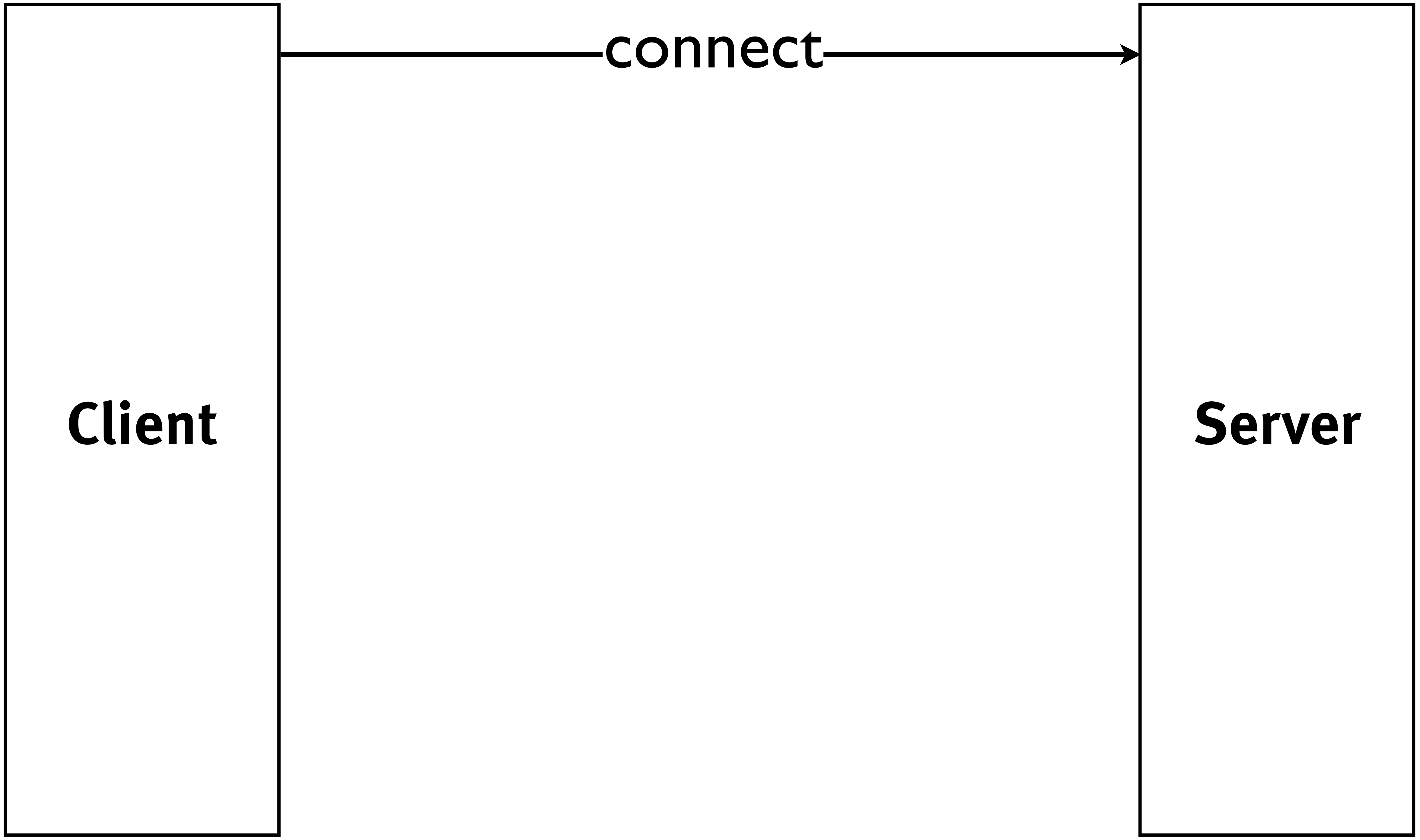
```
http.createServer(function(request, response) {
  var uri = url.parse(request.url).pathname;
  var filename = path.join(process.cwd(), dir, uri);
  sys.log('Serving file ' + filename);
  path.exists(filename, function(exists) {
    if(exists) {
      fs.readFile(filename, function(err, data) {
        var hash = crypto.createHash('md5');
        hash.update(data);
        response.writeHead(200,
             { 'Content-Type': 'text/plain',
               'Content-MD5': hash.digest('base64') }
        );
        response.end(data);
      });
    } else {
      response.writeHead(404);
      response.end();
    }
  });
}).listen(port);
```

file-server-md5.js

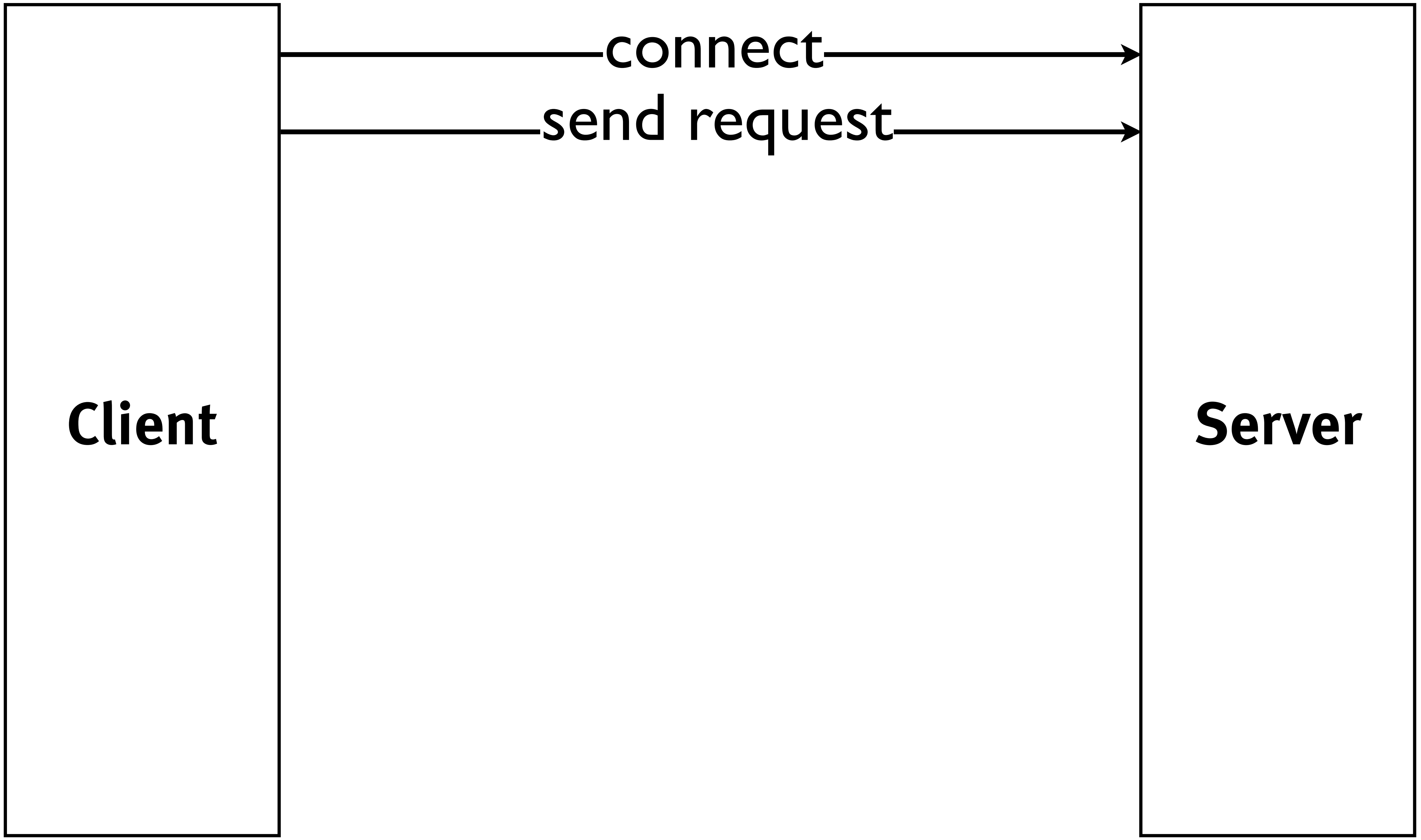
innoQ

HTTP Chunking

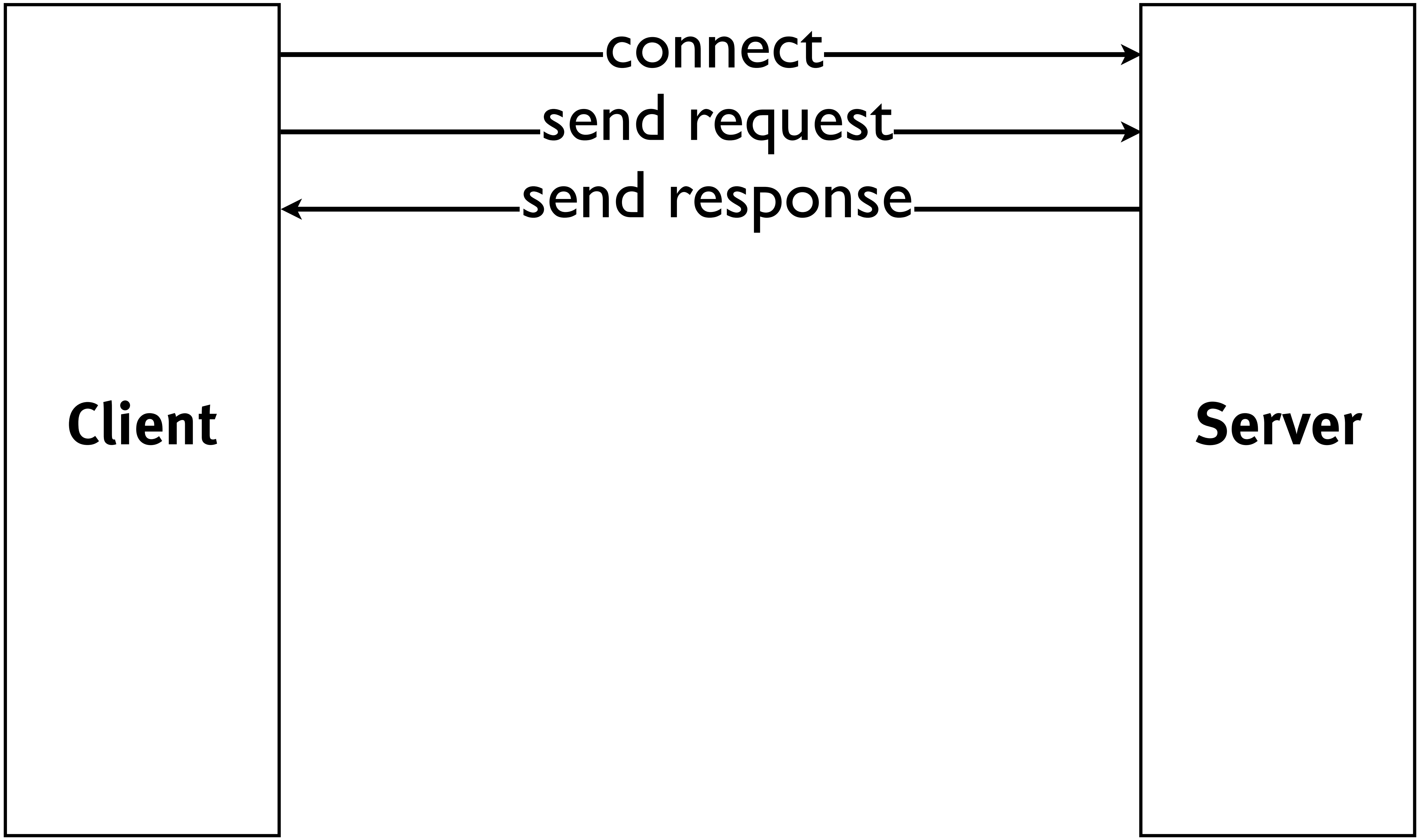
HTTP/1.0



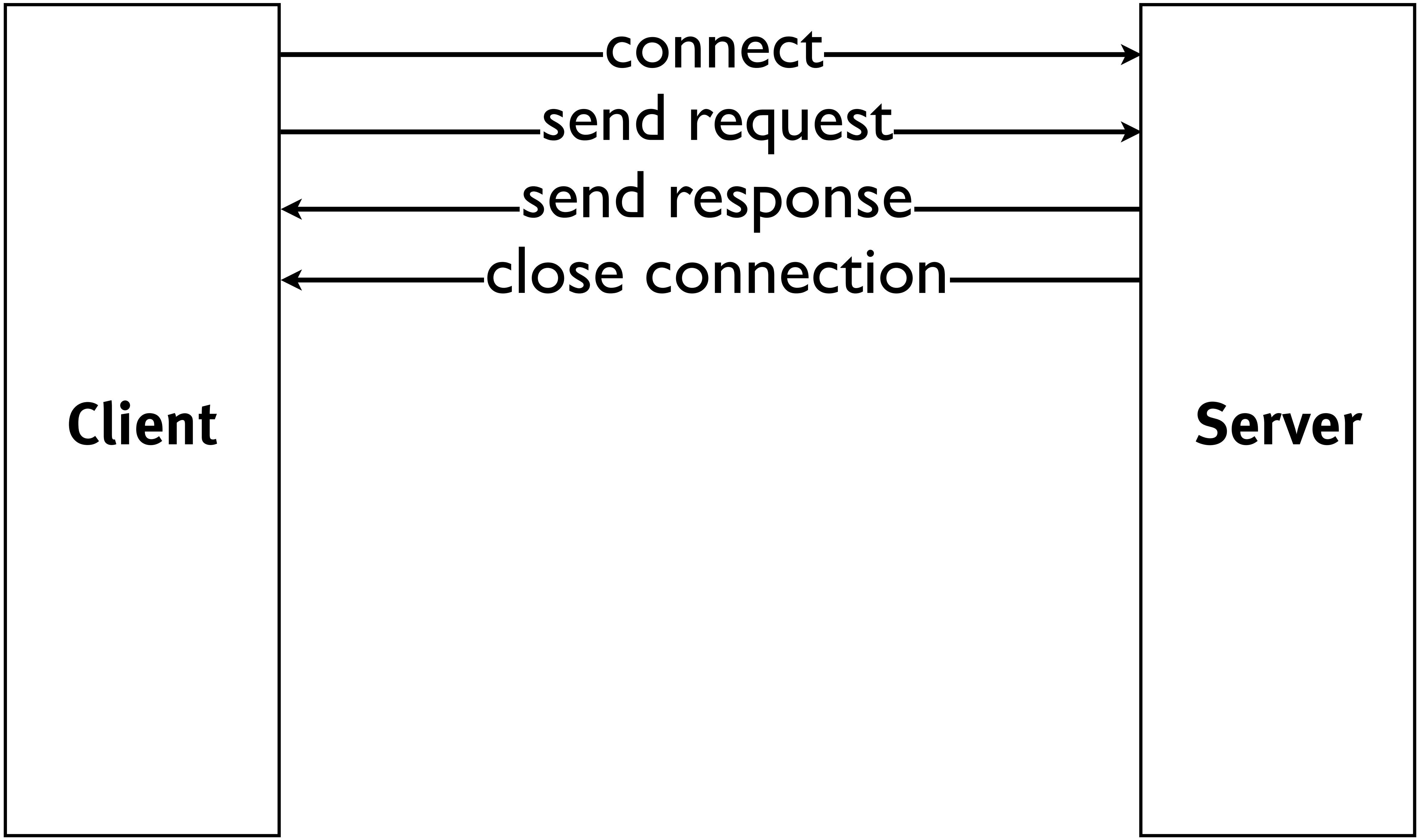
HTTP/1.0



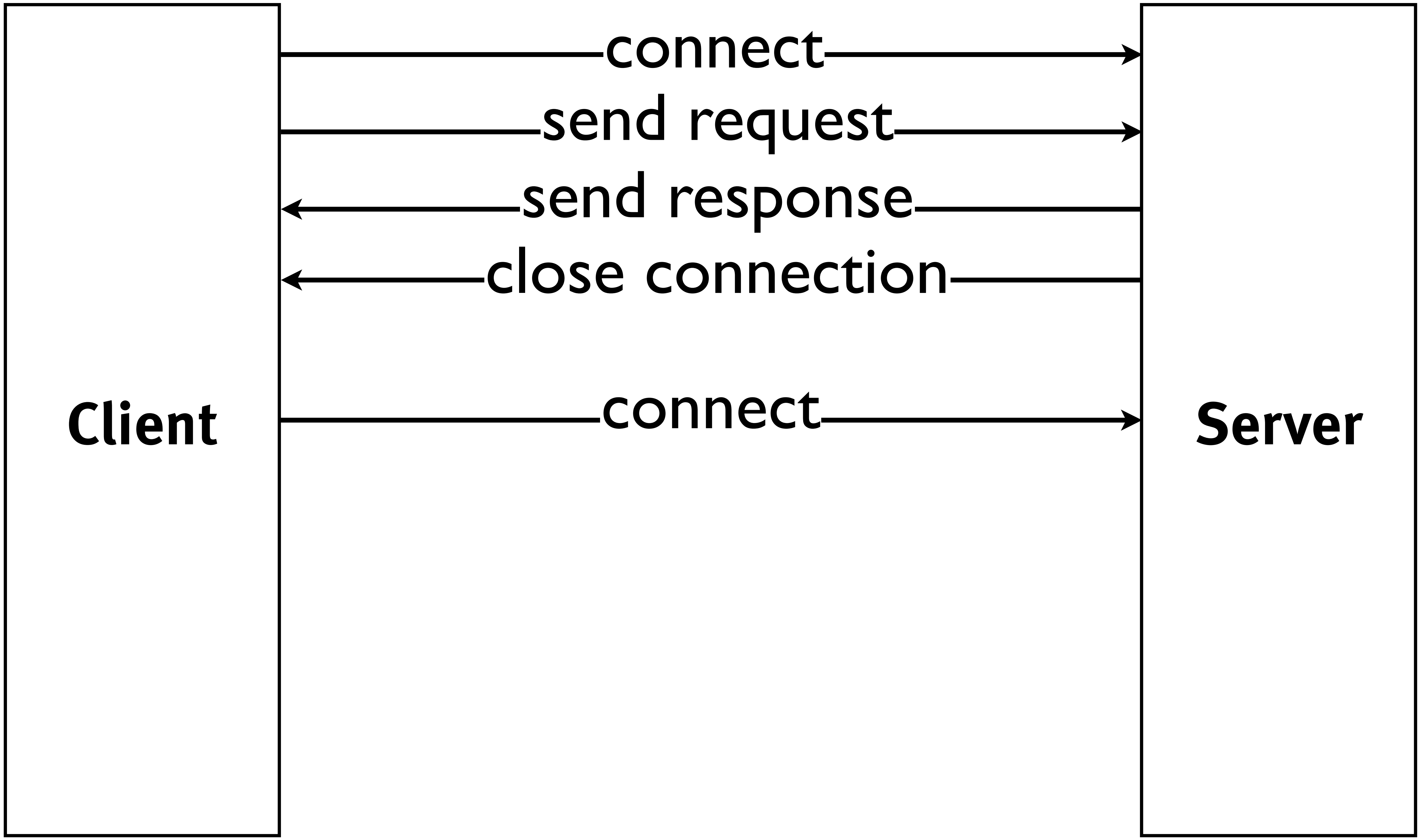
HTTP/1.0



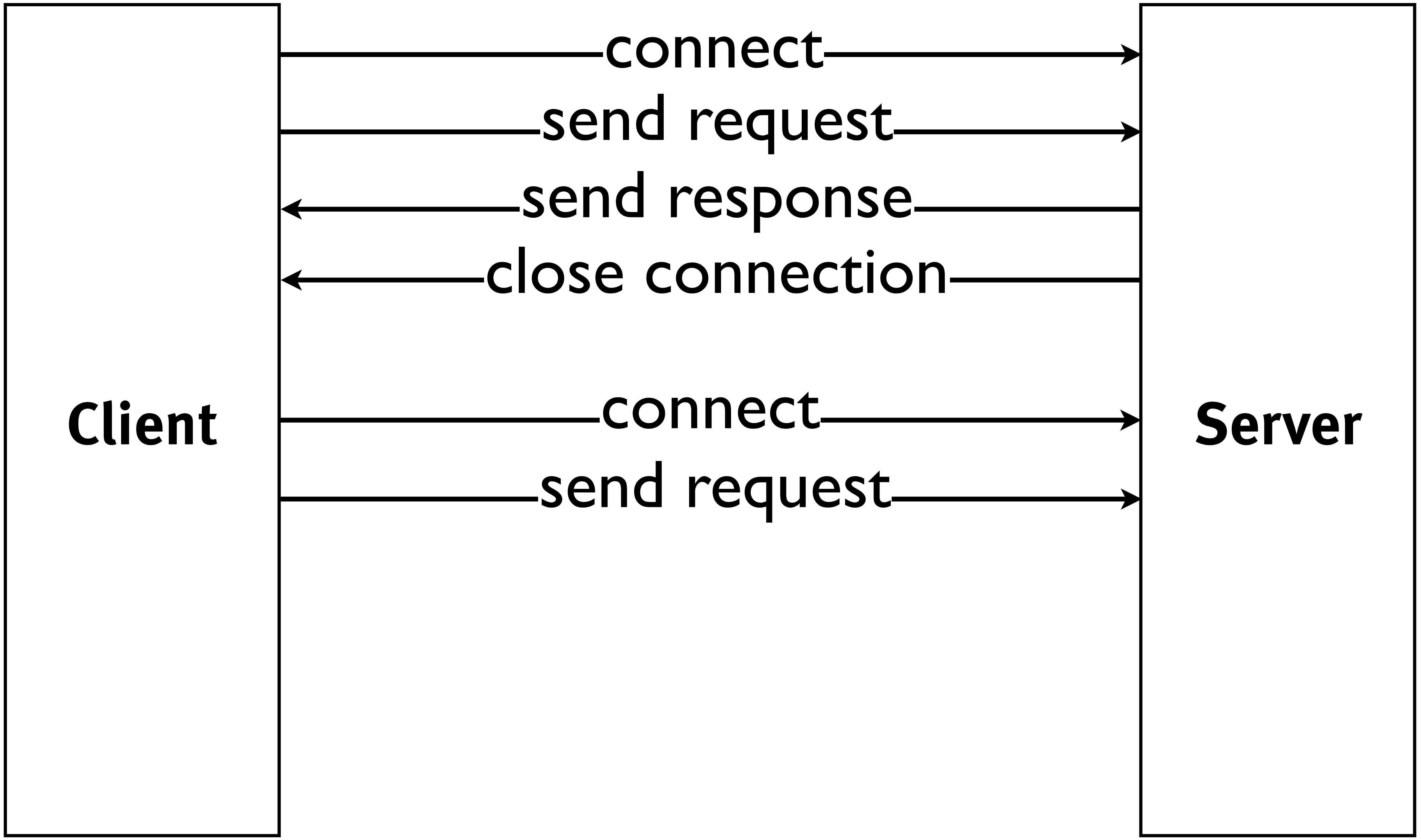
HTTP/1.0


connect
send request
send response
close connection
Client
Server

HTTP/1.0

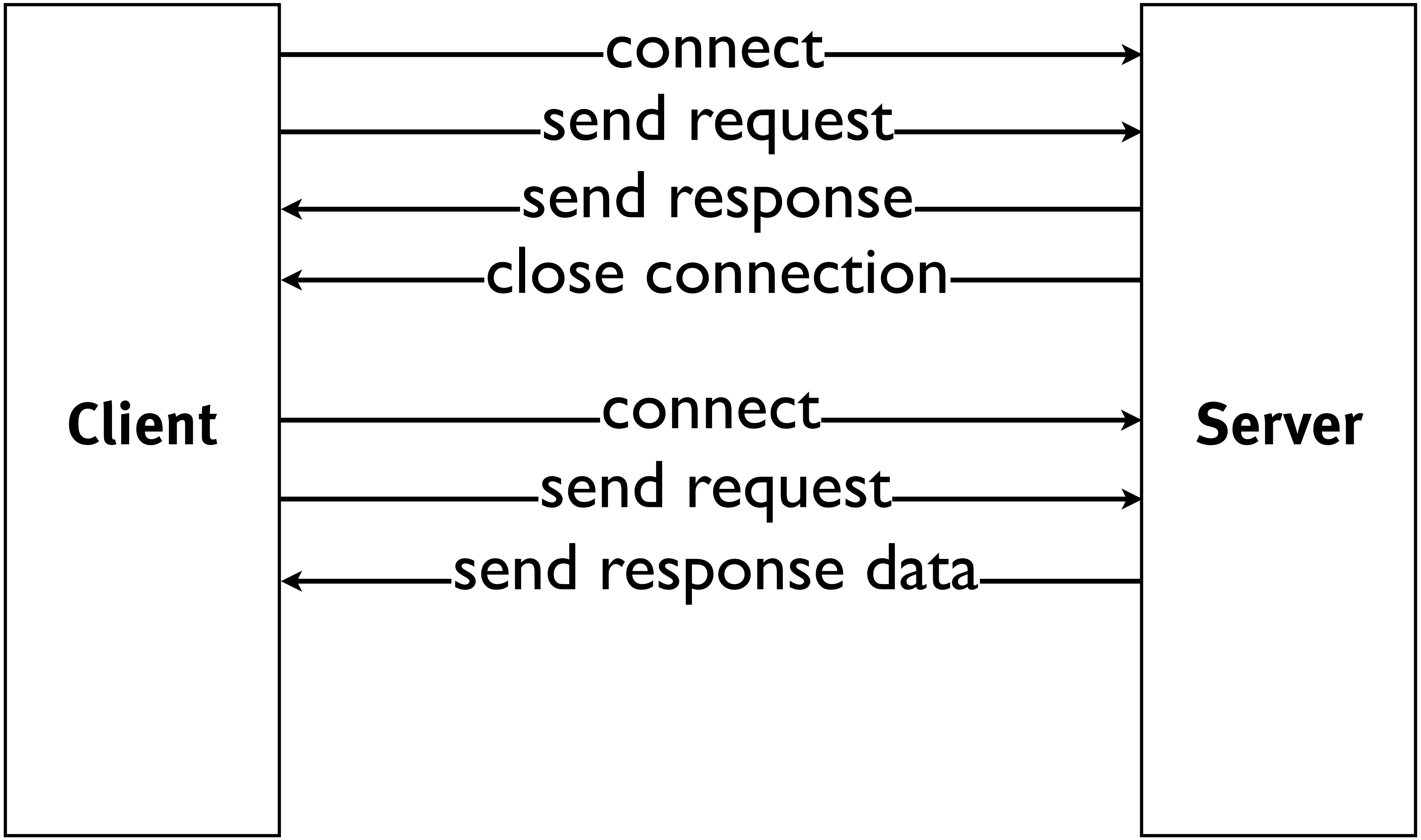

Client
connect
send request
send response
close connection
connect
Server

HTTP/1.0

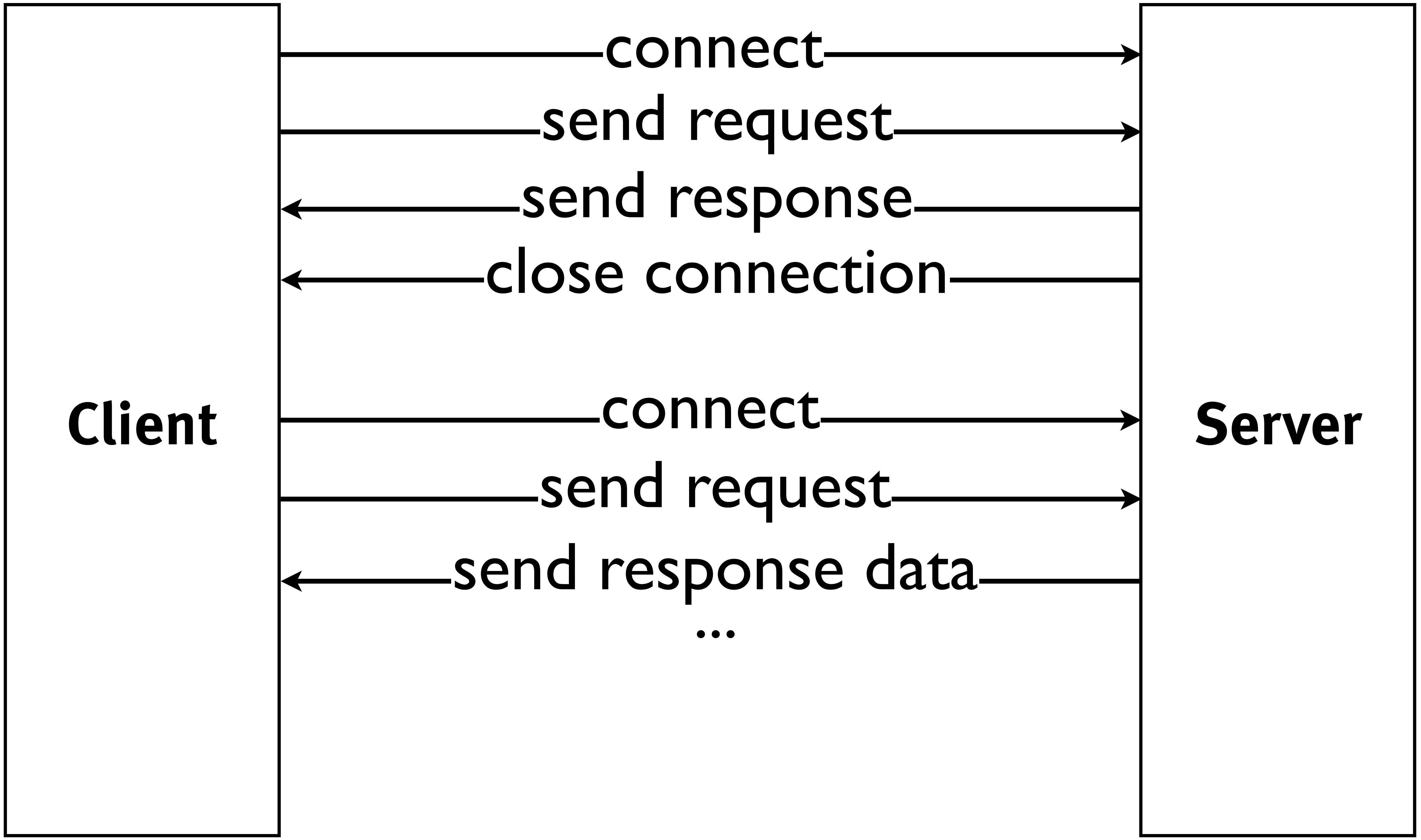


Client
Server
connect
send request
send response
close connection
connect
send request

HTTP/1.0

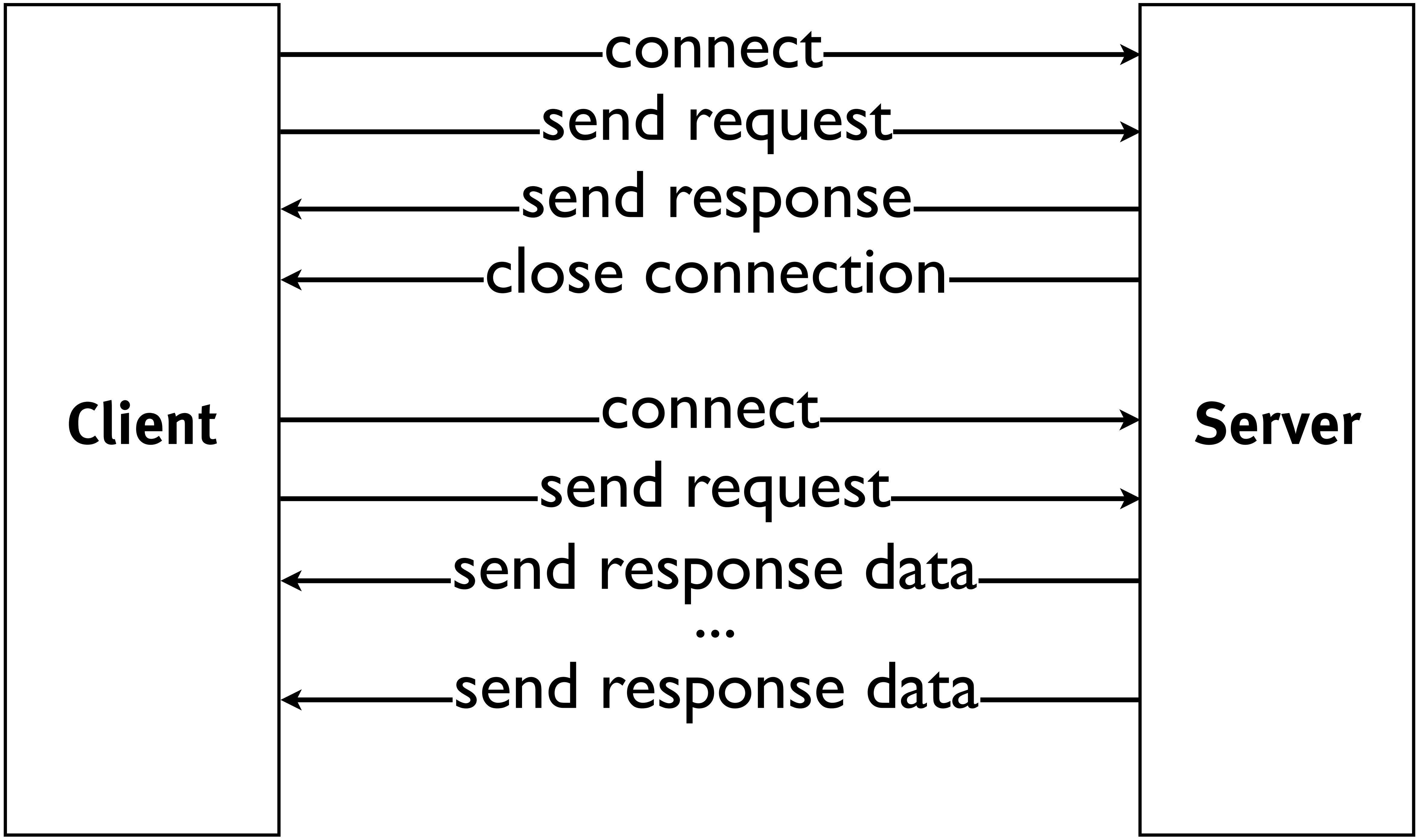

Client
Server
connect
send request
send response
close connection
connect
send request
send response data

HTTP/1.0



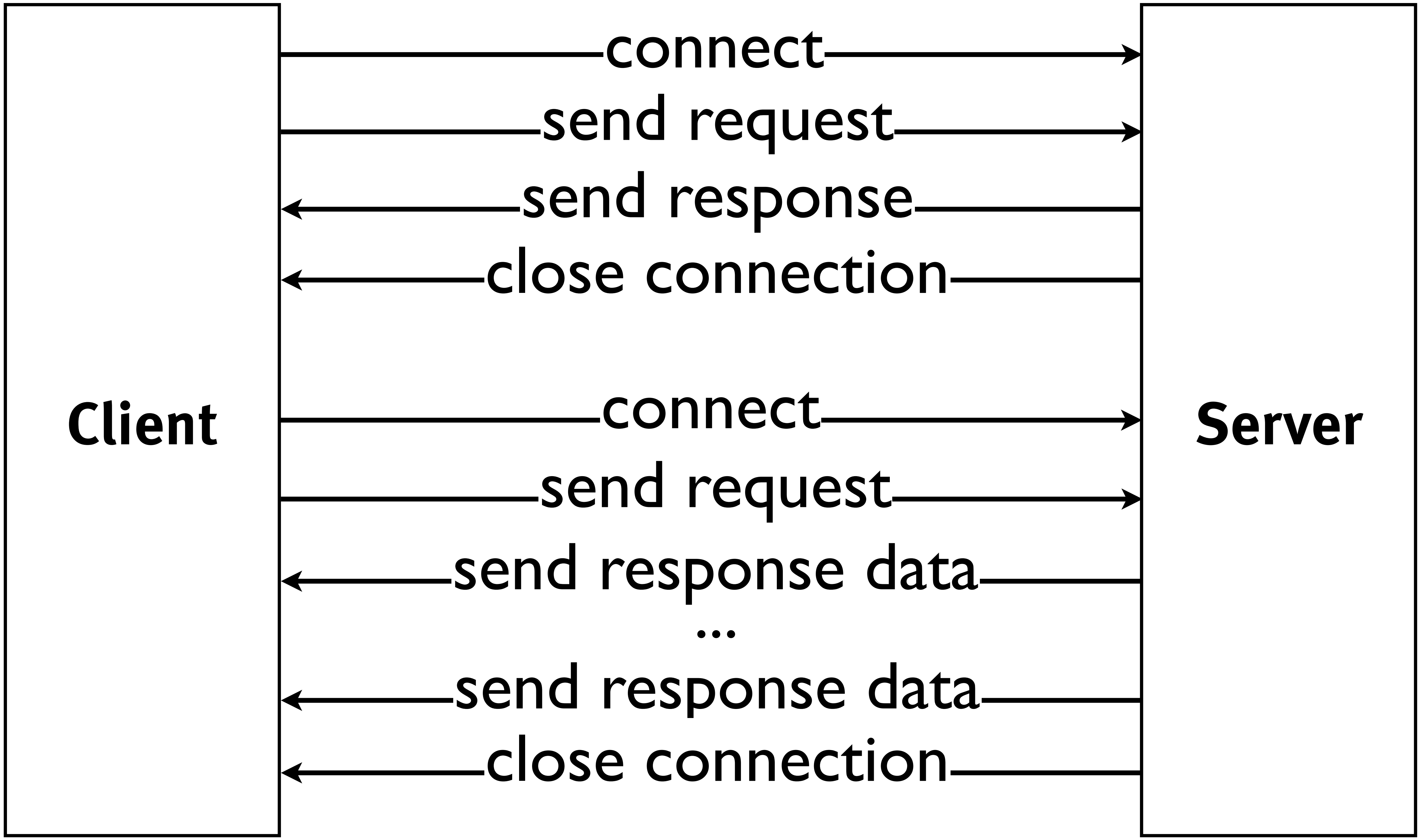
Client
Server
connect
send request
send response
close connection
connect
send request
send response data
...

HTTP/1.0



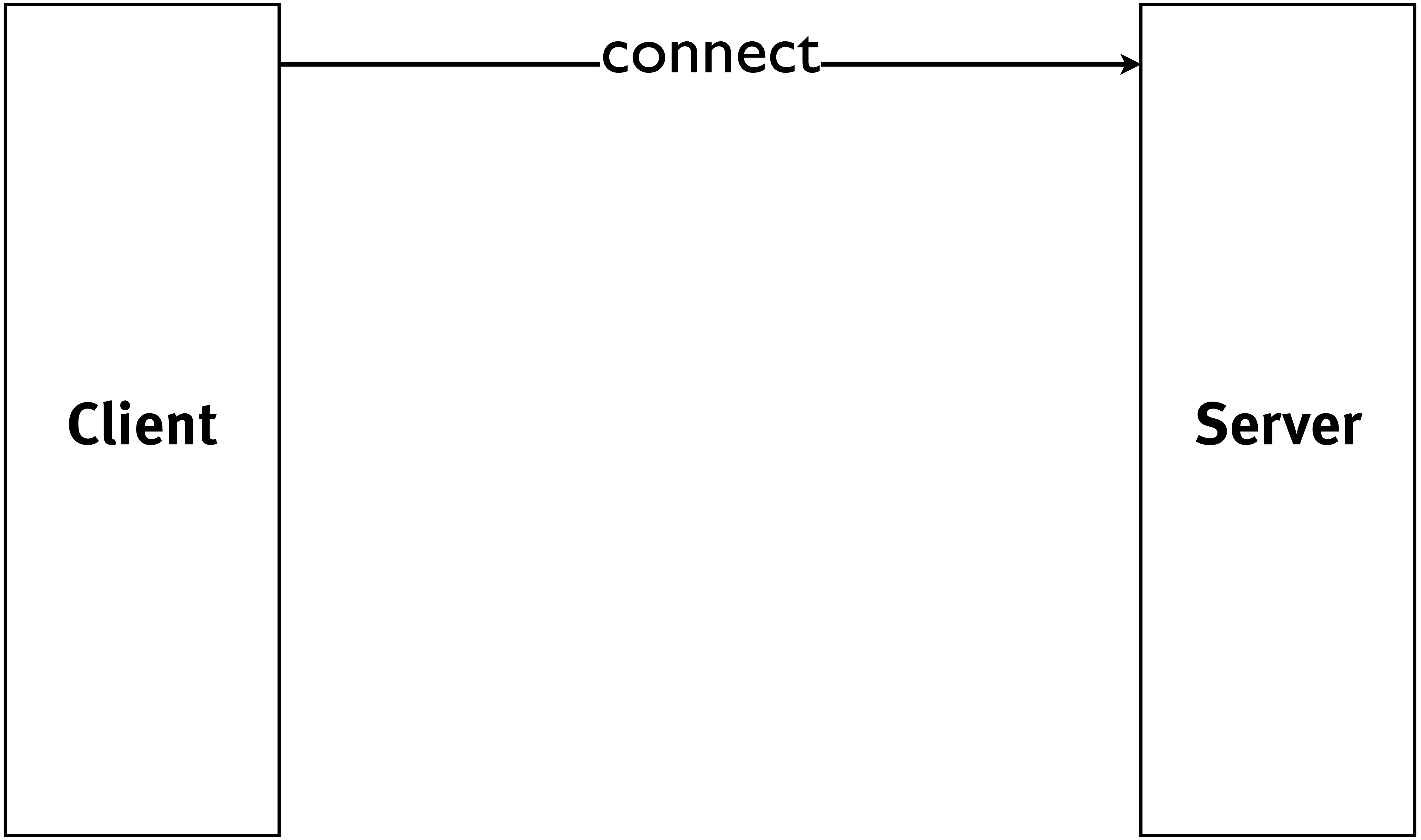
innoQ

HTTP/1.0

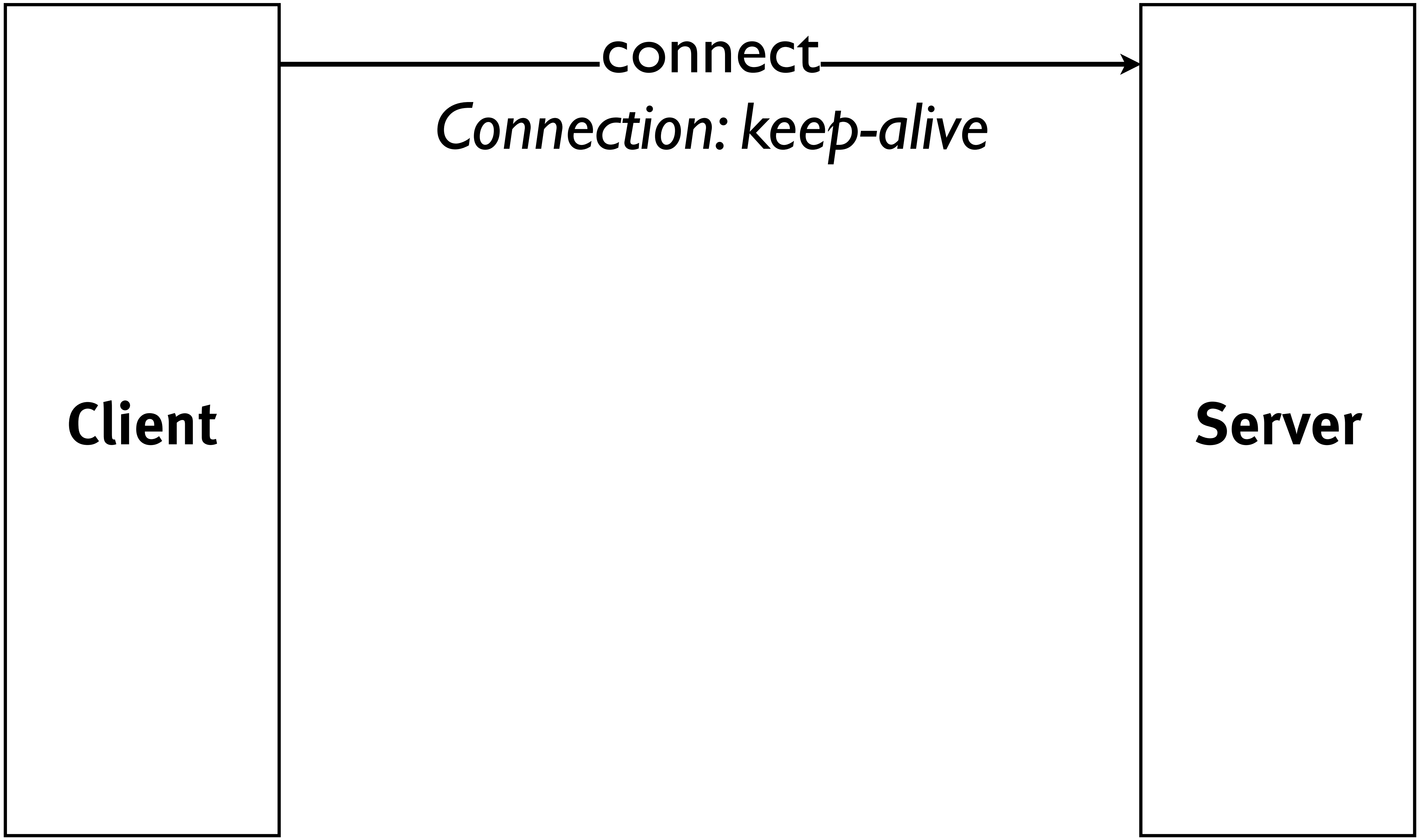

Client
Server
connect
send request
send response
close connection
connect
send request
send response data
...
send response data
close connection

HTTP/1.1: Content-length

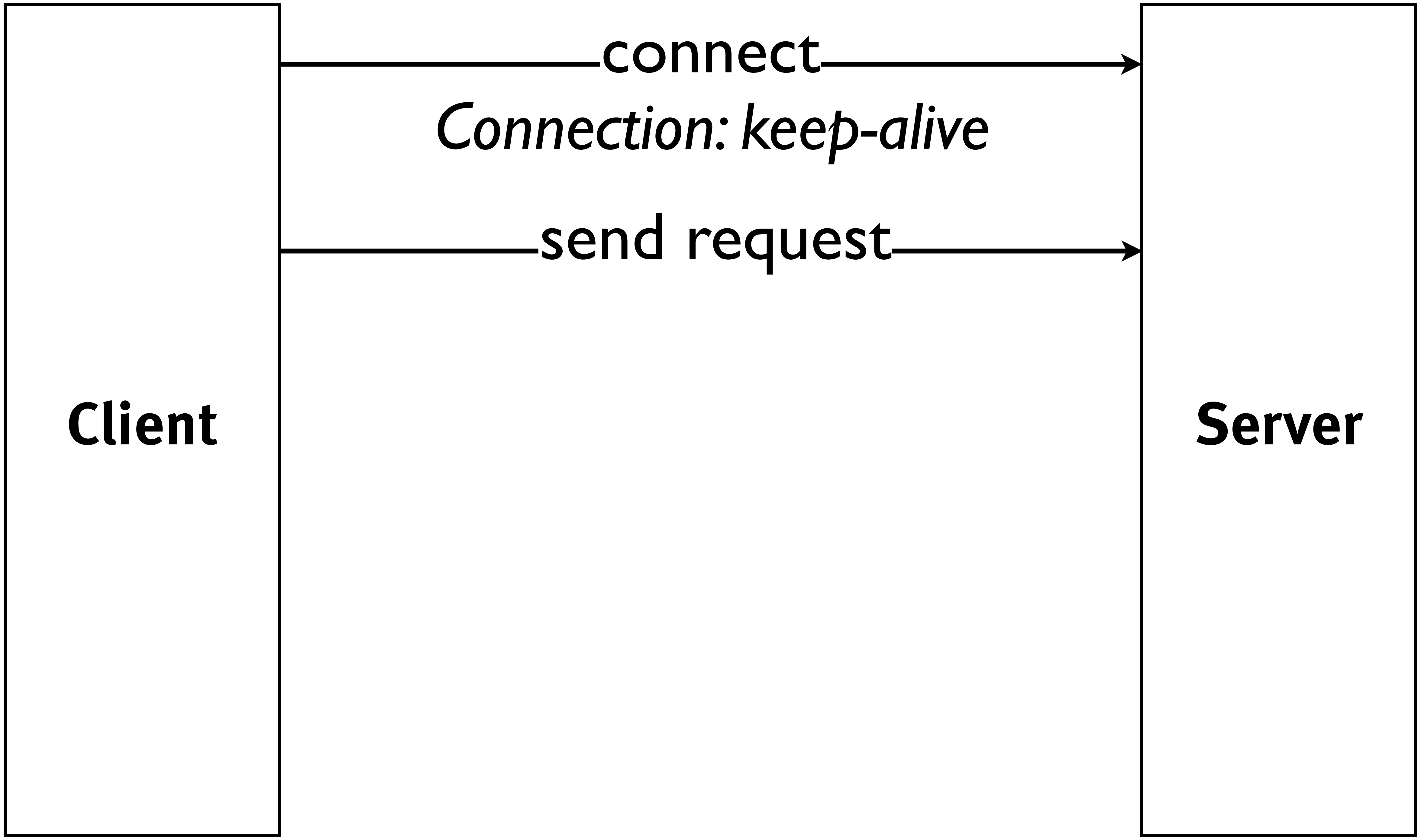
HTTP/1.1: Content-length



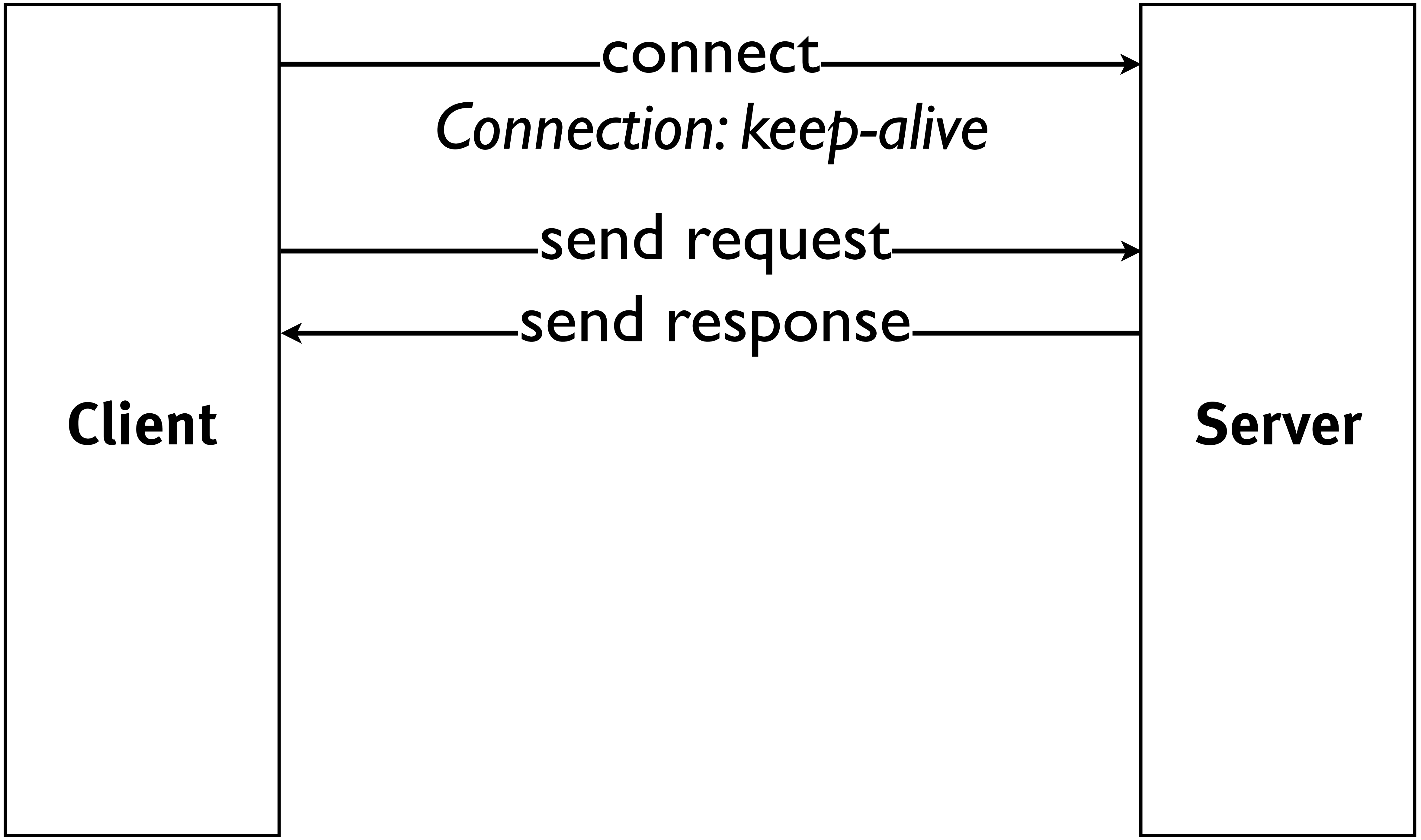
HTTP/1.1: Content-length



HTTP/1.1: Content-length

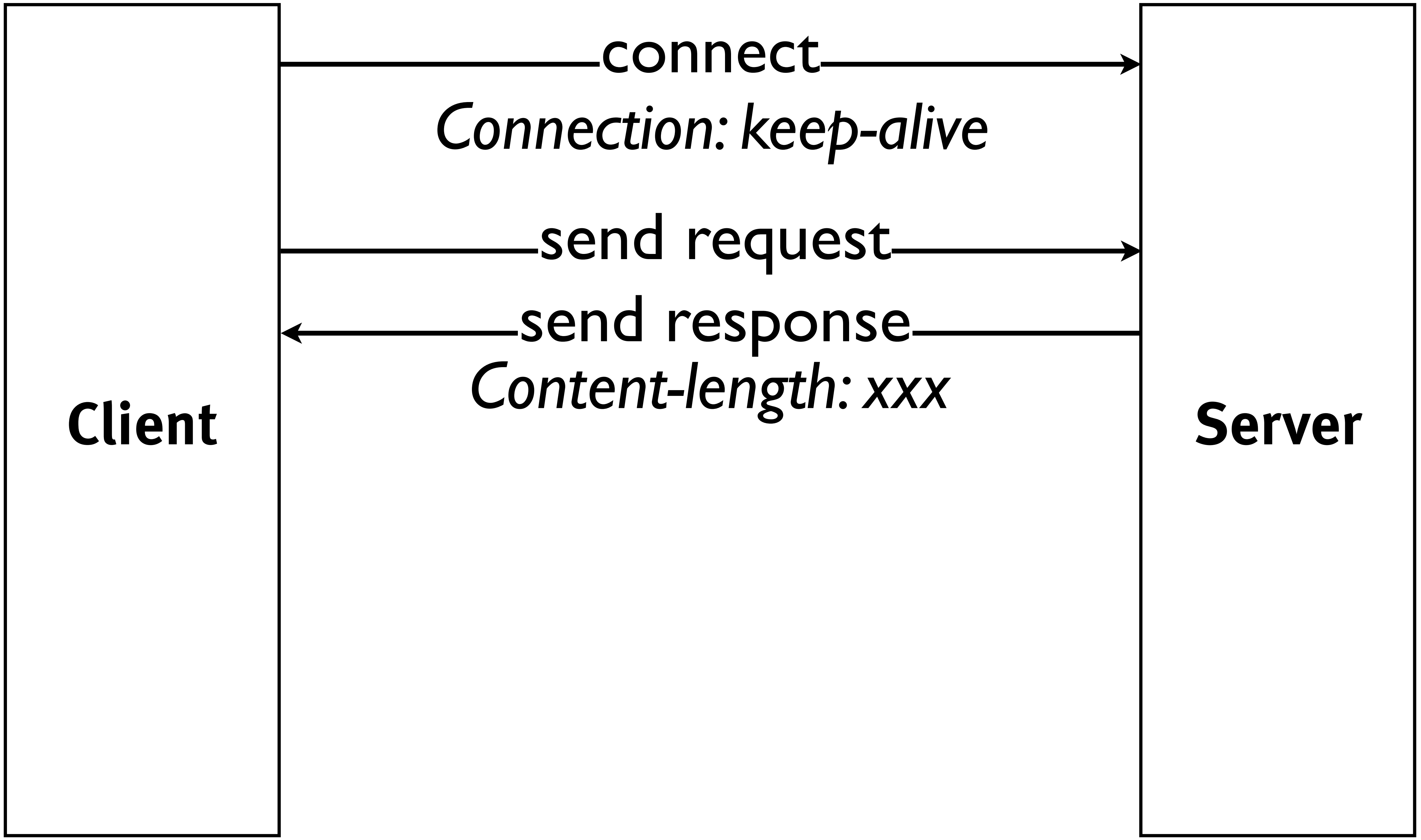


HTTP/1.1: Content-length

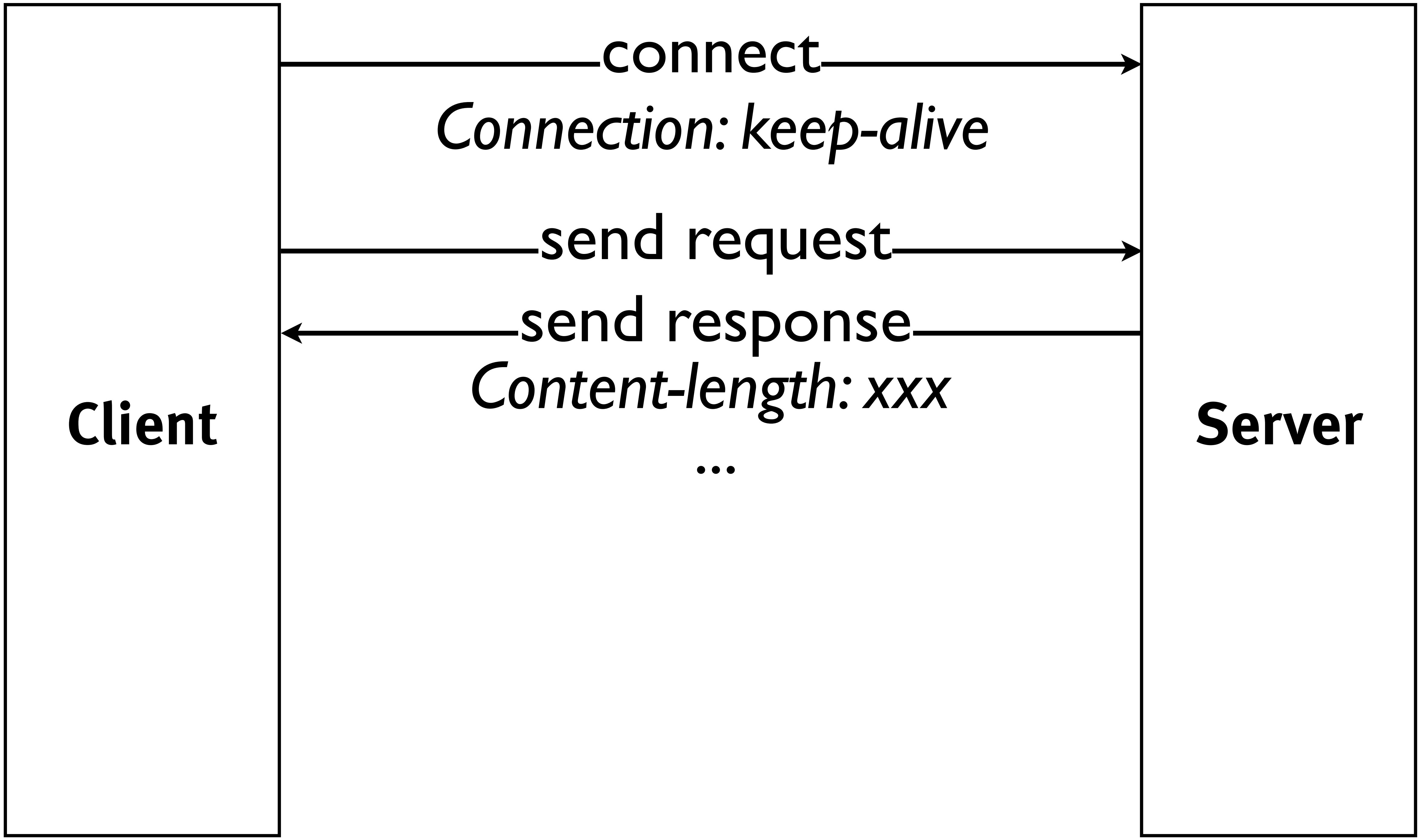


innoQ

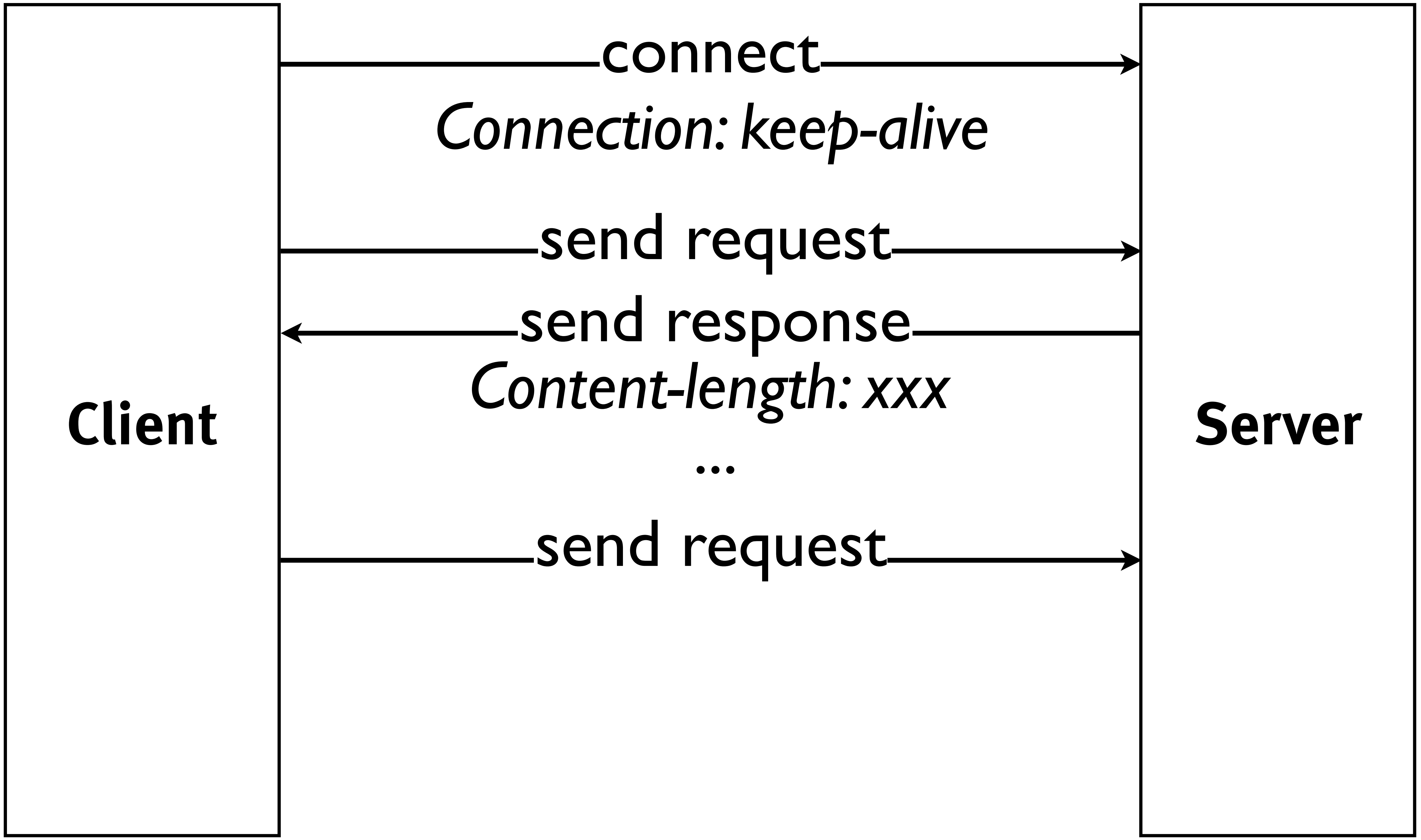
HTTP/1.1: Content-length



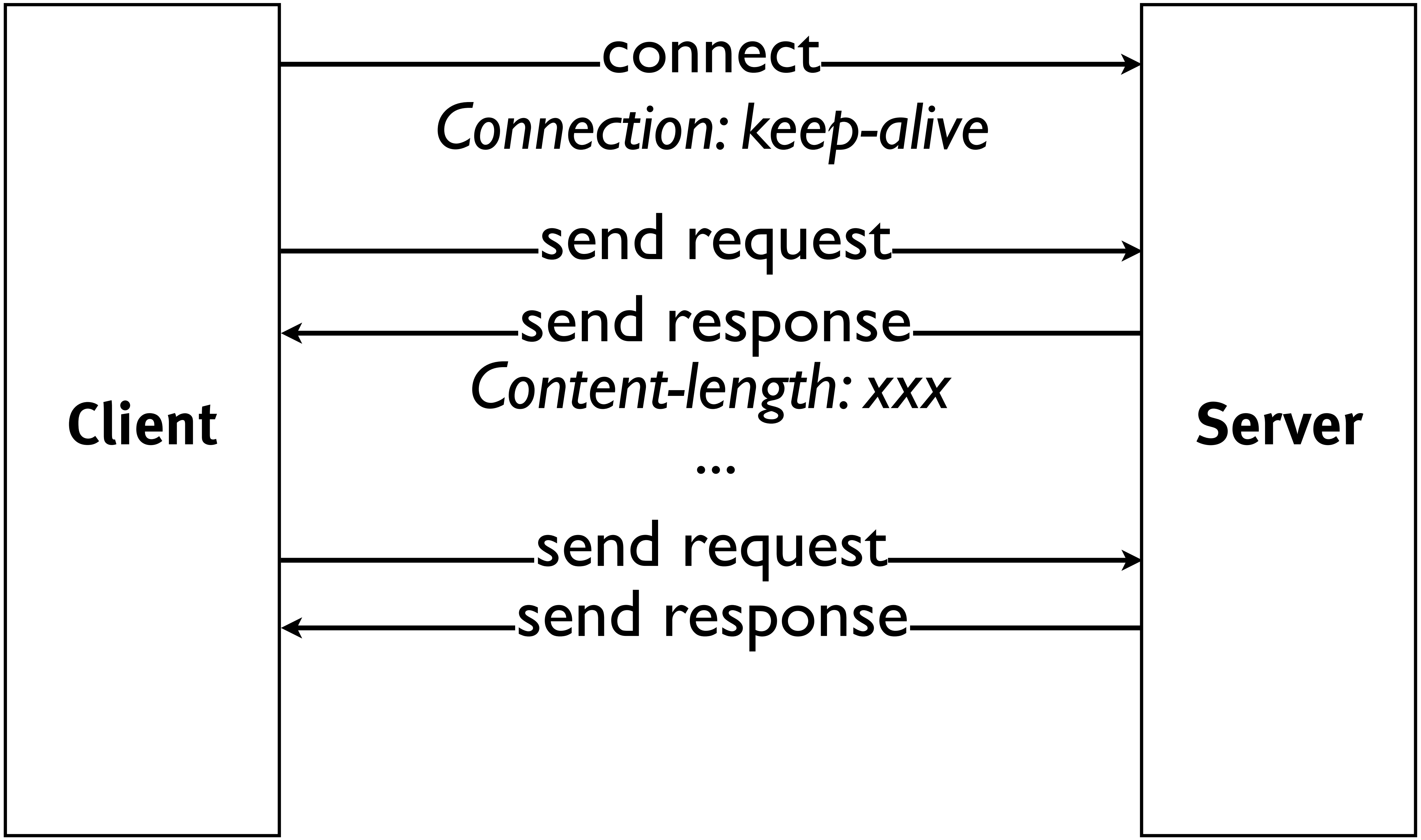
HTTP/1.1: Content-length



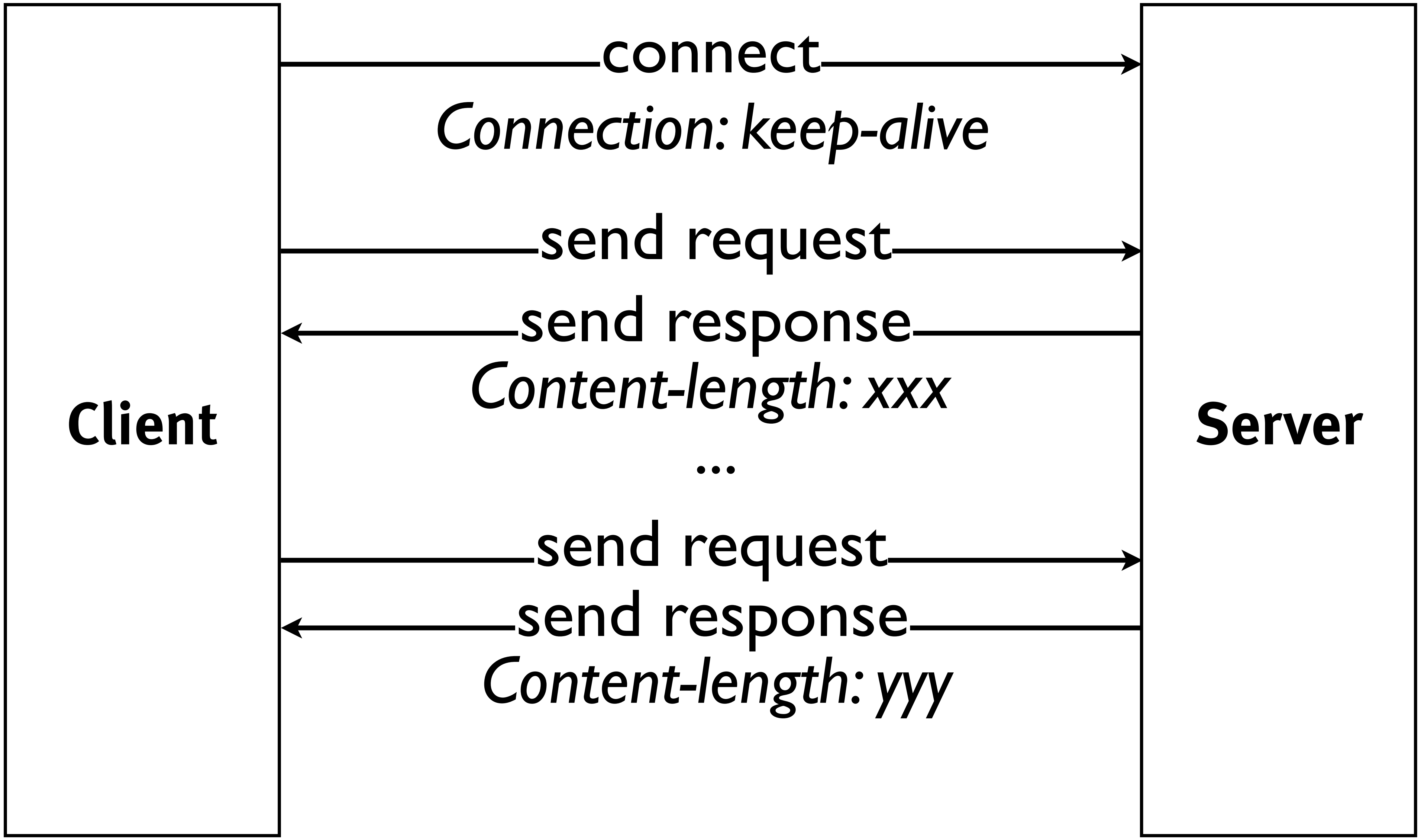
HTTP/1.1: Content-length



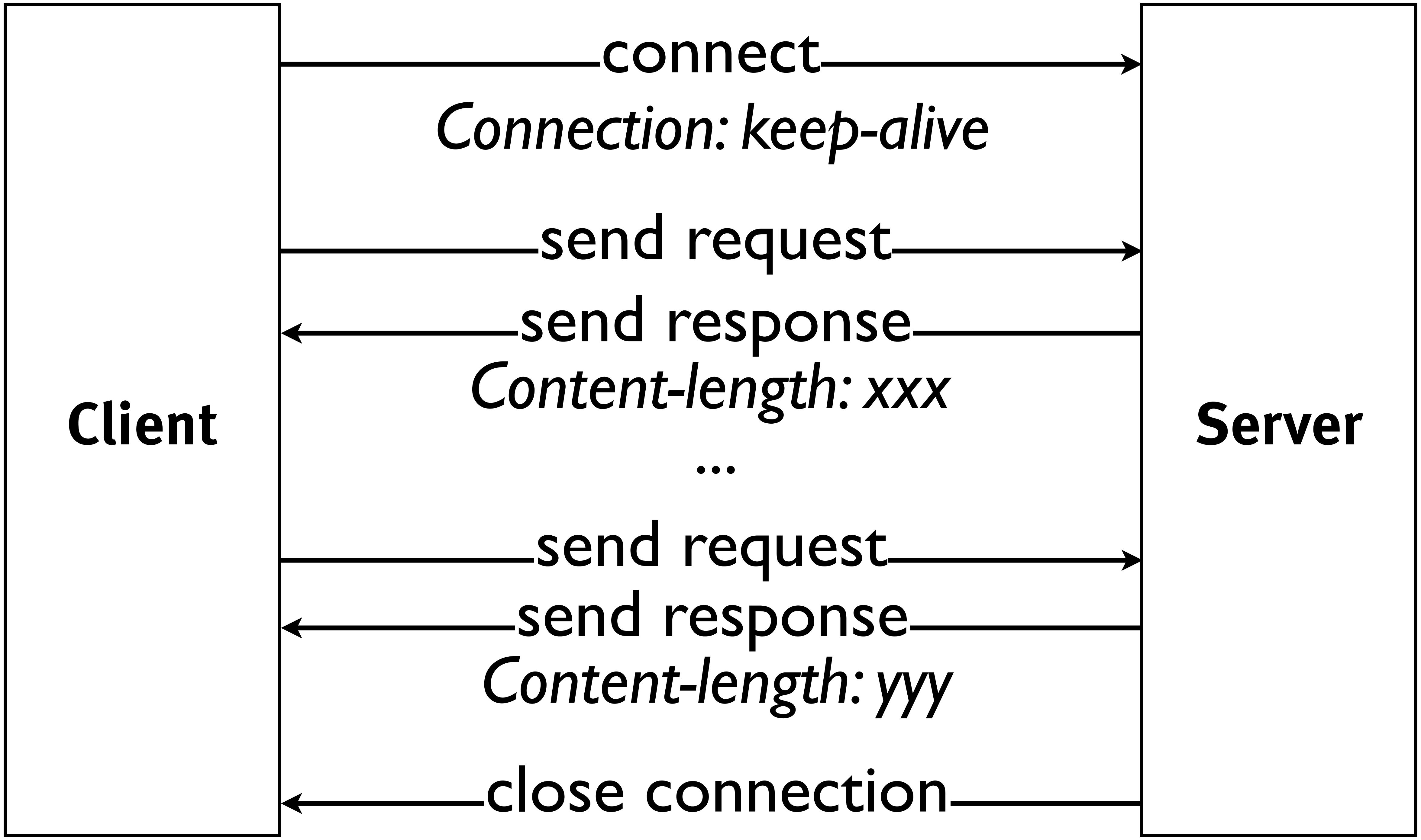
HTTP/1.1: Content-length



HTTP/1.1: Content-length

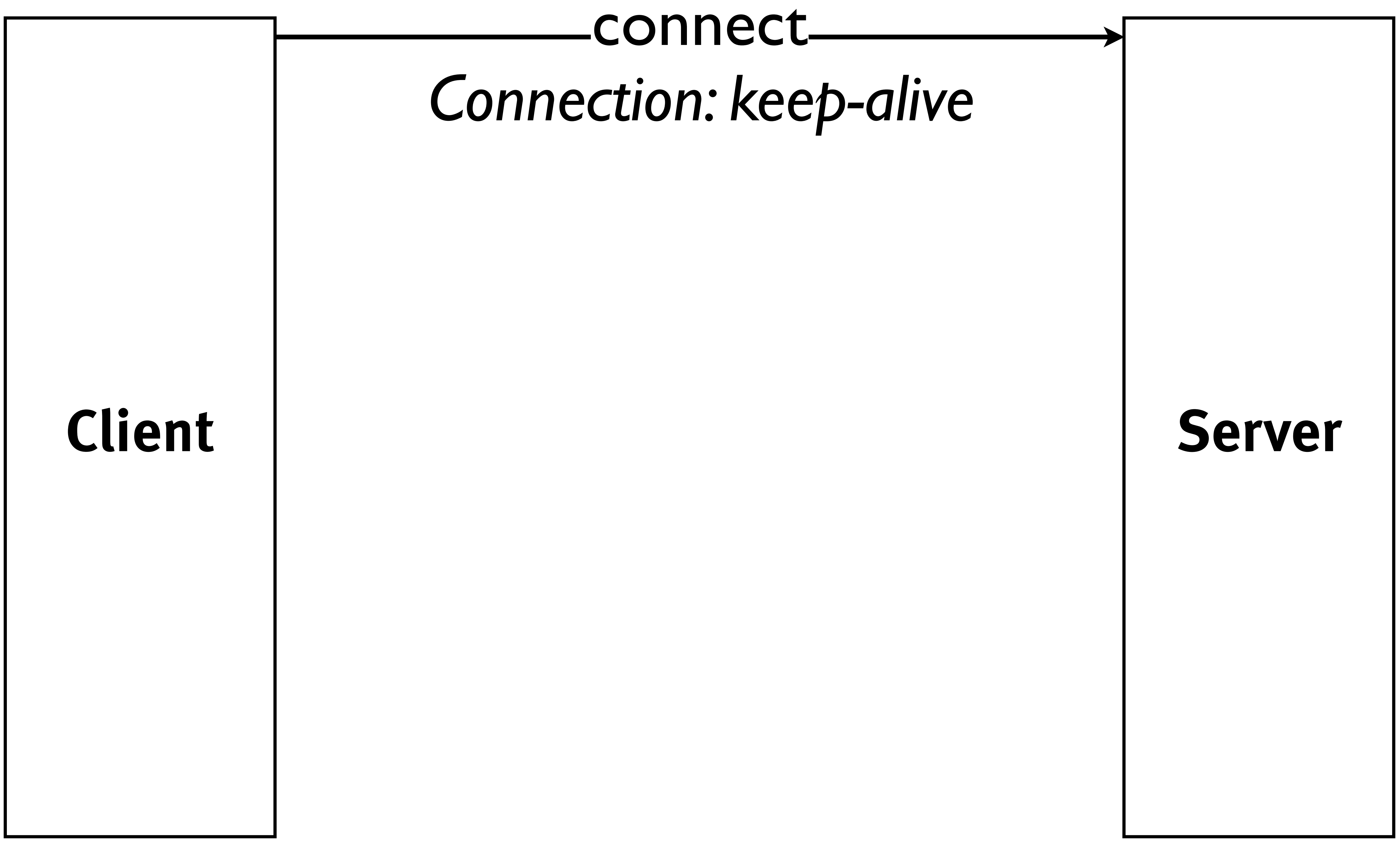


HTTP/1.1: Content-length

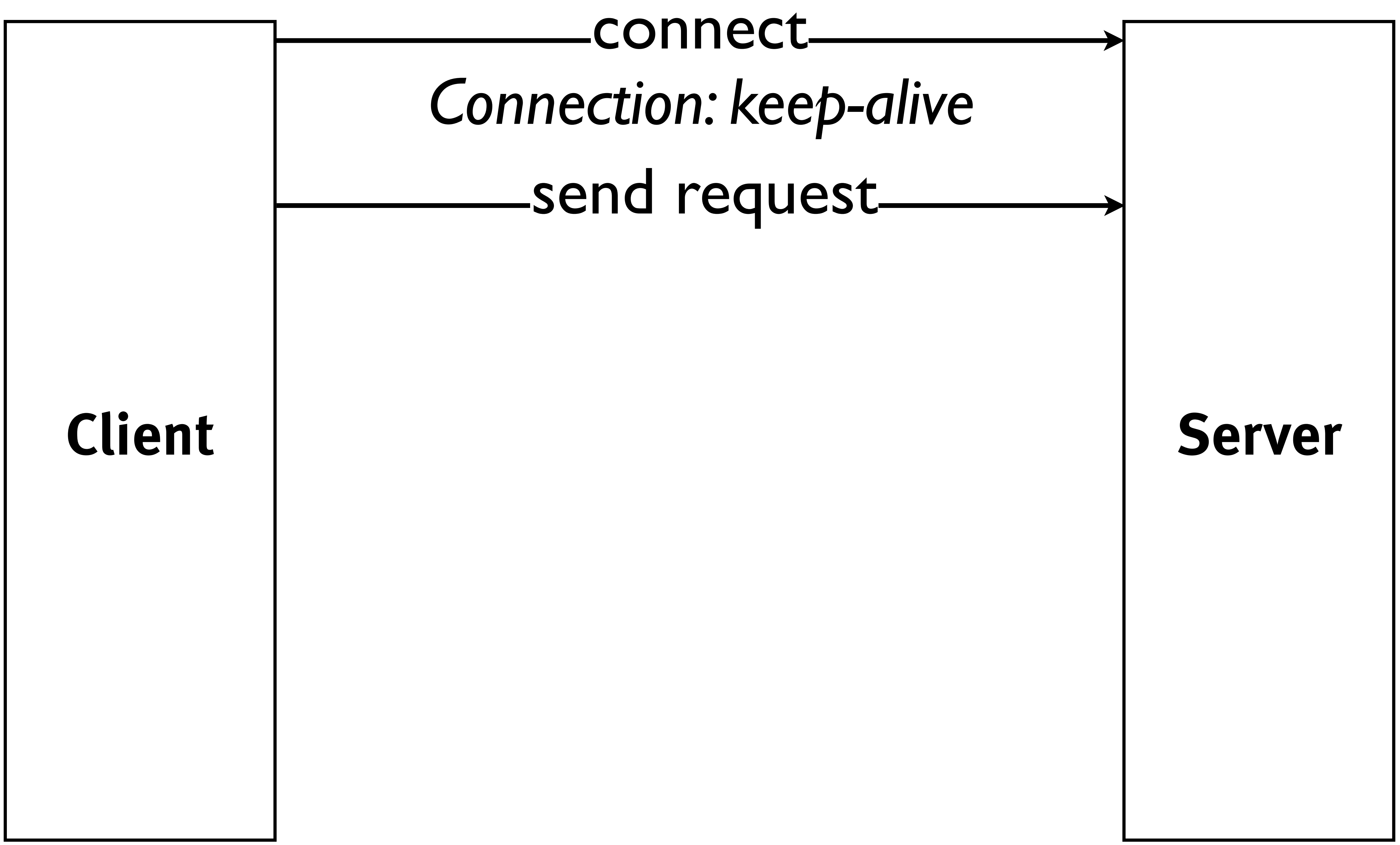


HTTP/1.1: Transfer-encoding: chunked

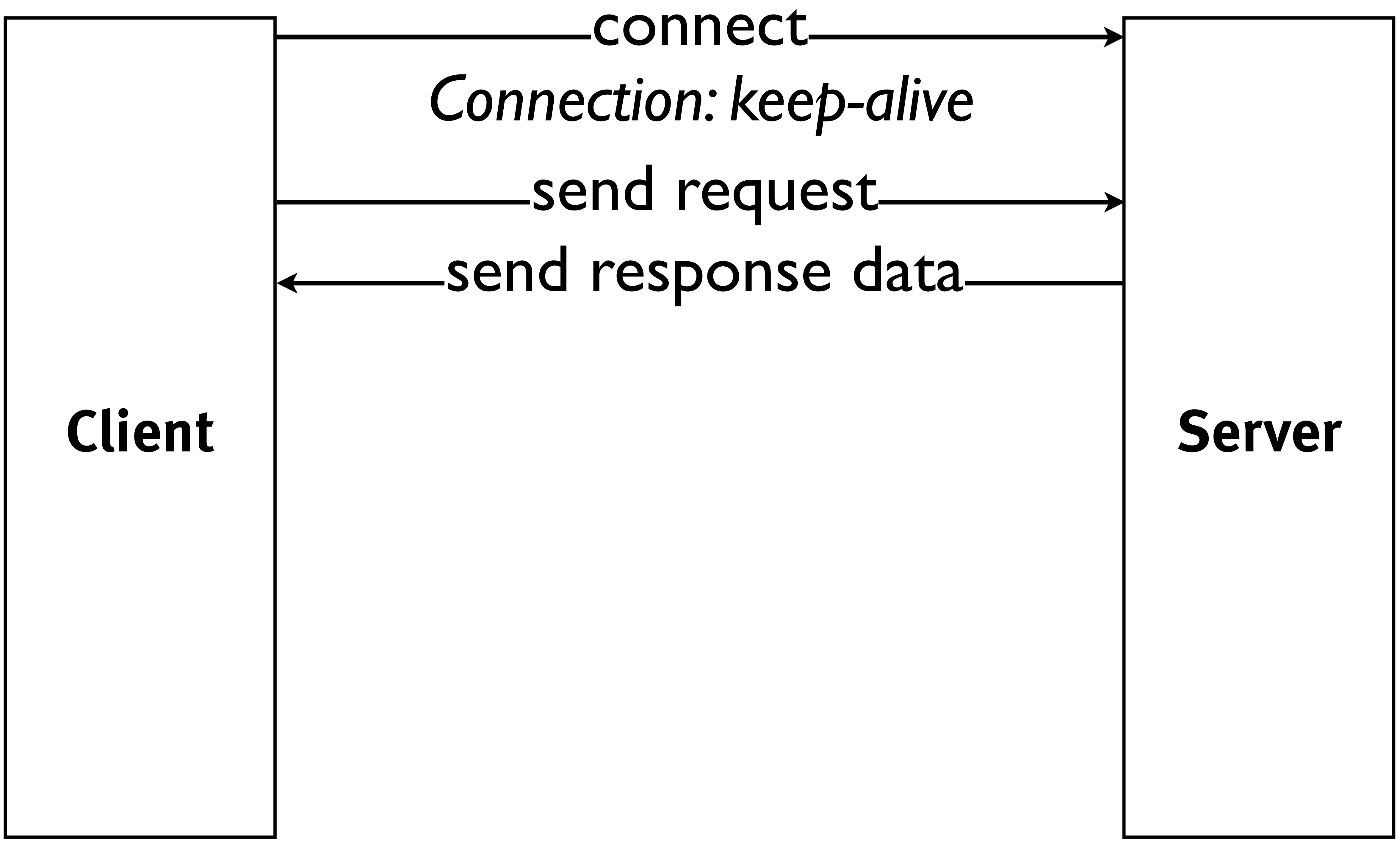
HTTP/1.1: Transfer-encoding: chunked



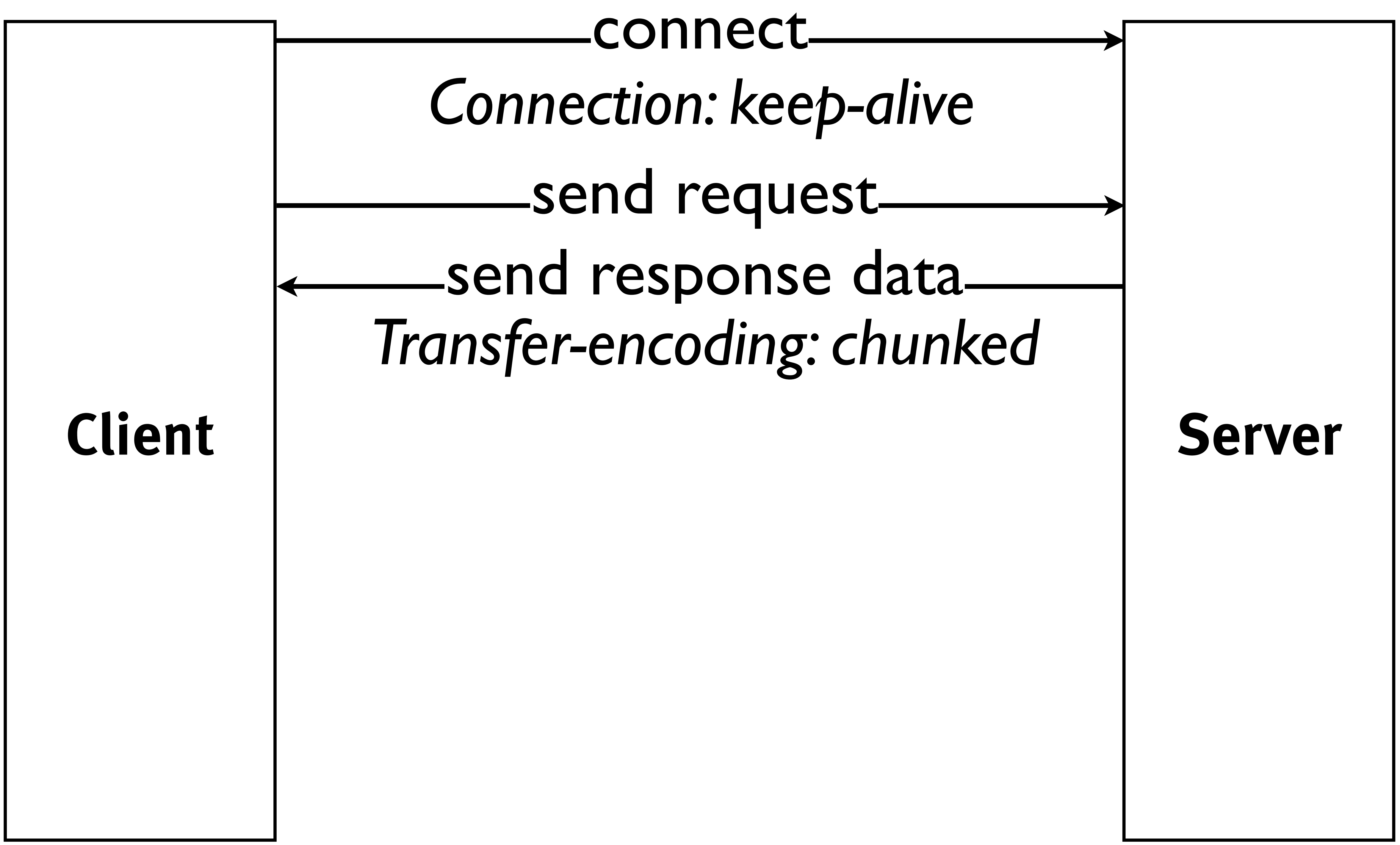
HTTP/1.1: Transfer-encoding: chunked



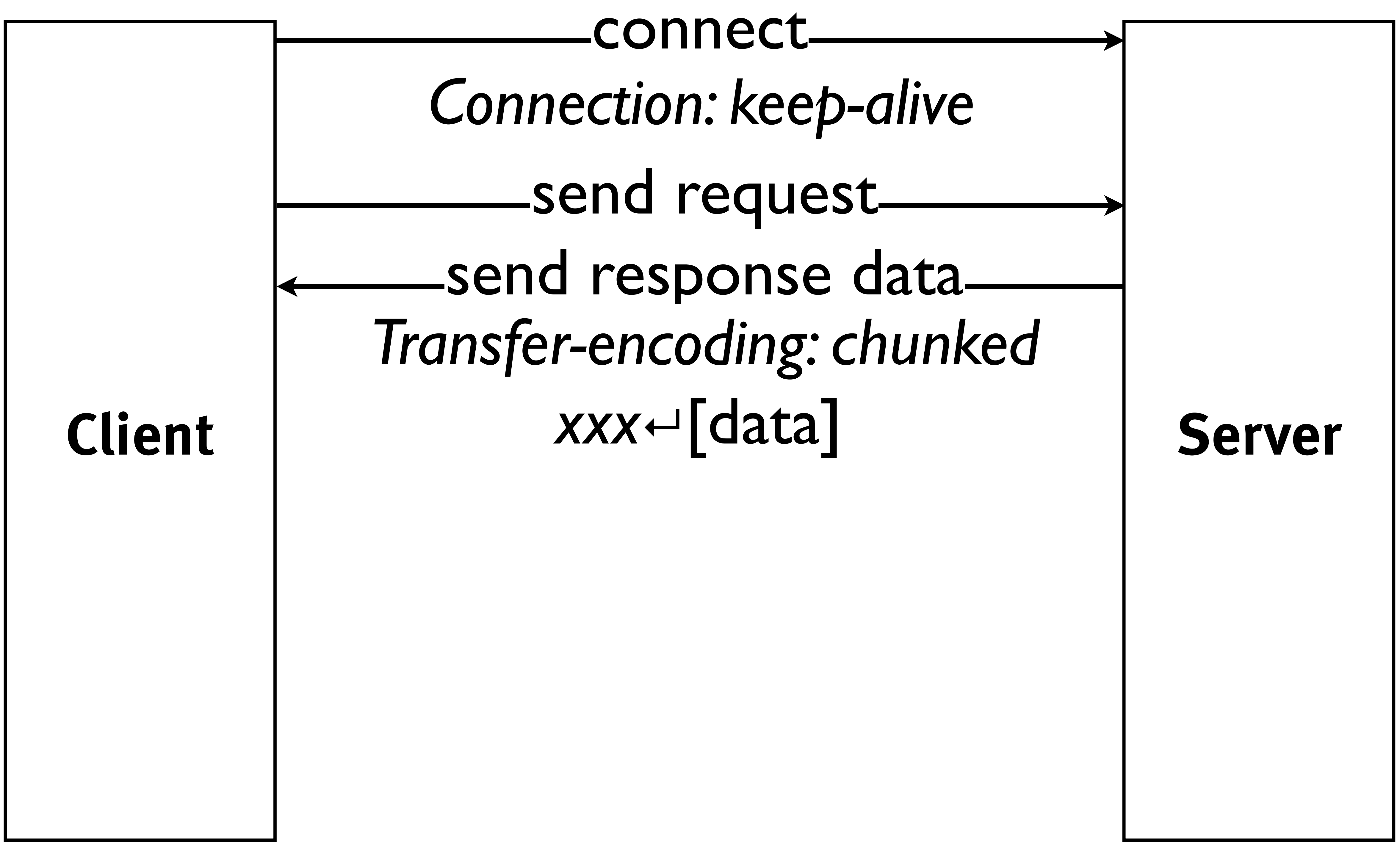
HTTP/1.1: Transfer-encoding: chunked



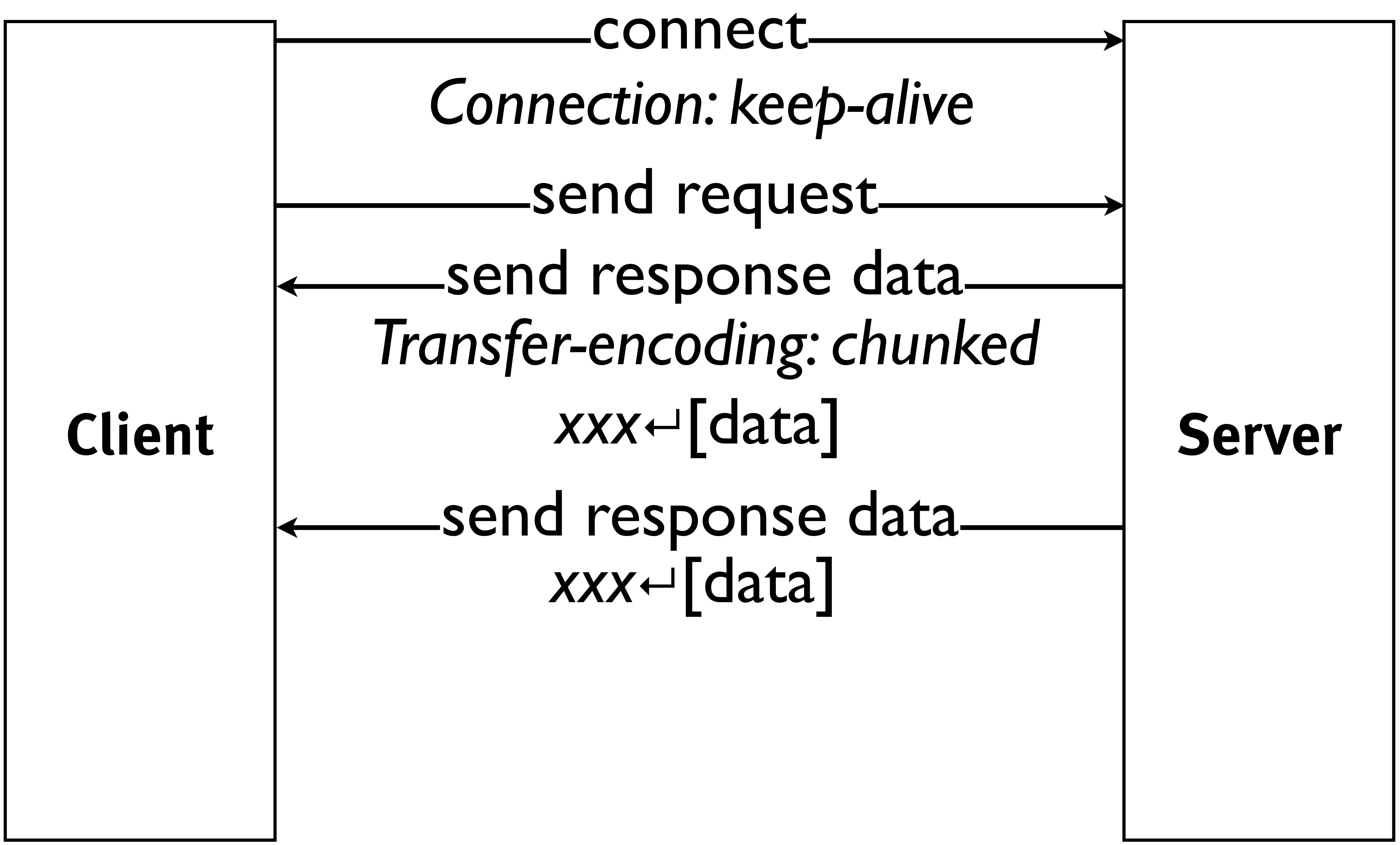
HTTP/1.1: Transfer-encoding: chunked



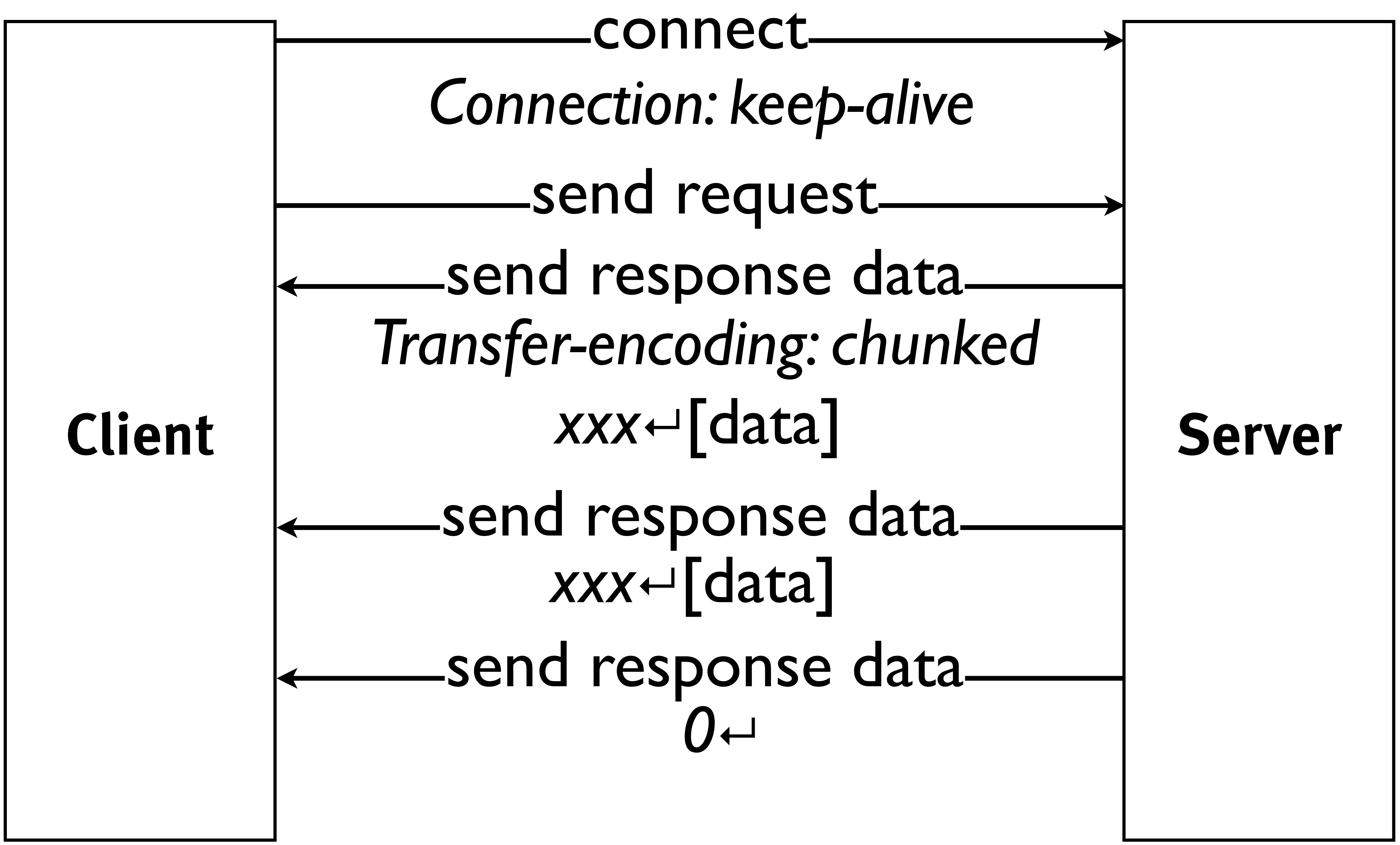
HTTP/1.1: Transfer-encoding: chunked



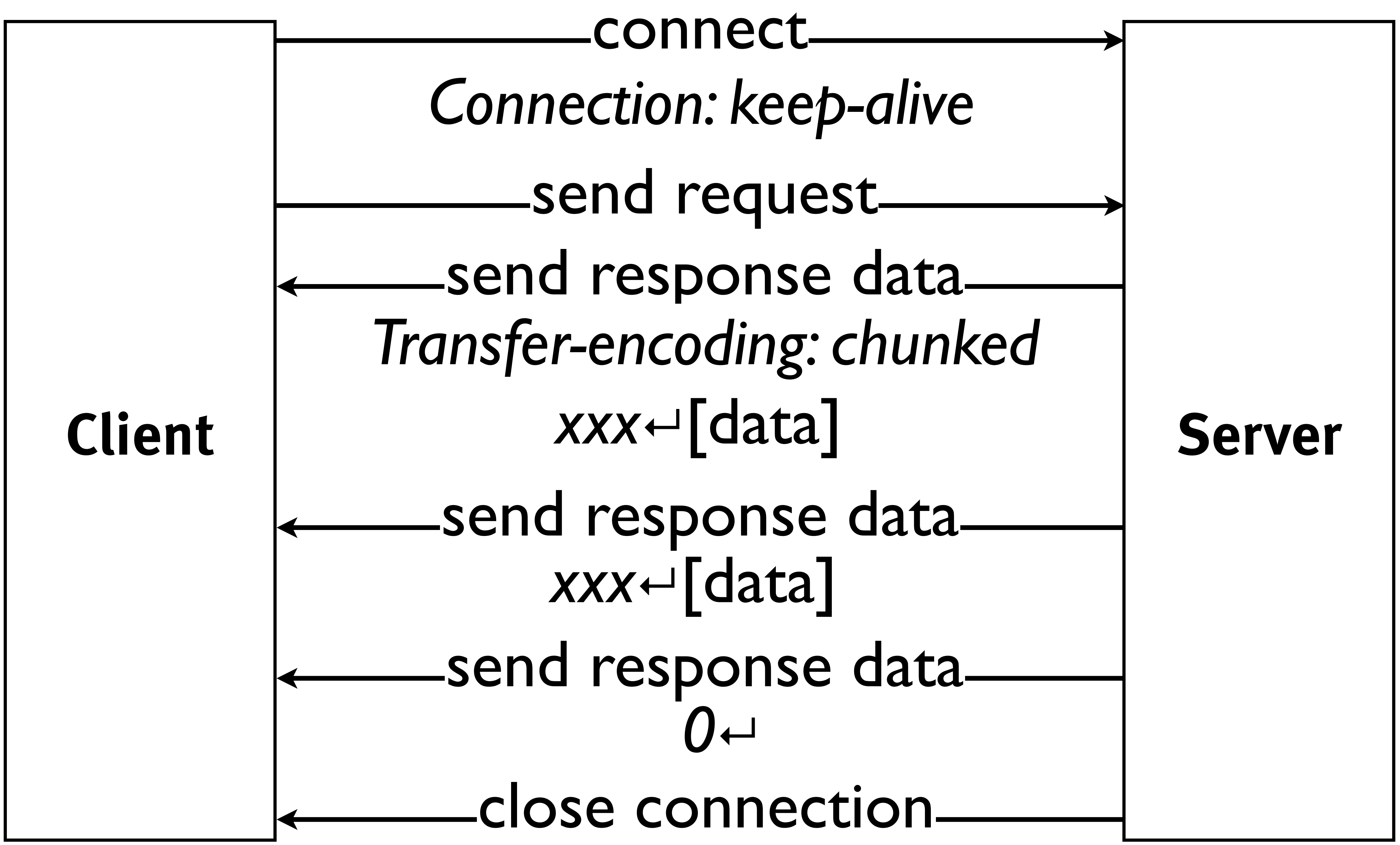
HTTP/1.1: Transfer-encoding: chunked



HTTP/1.1: Transfer-encoding: chunked



HTTP/1.1: Transfer-encoding: chunked



```
http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), dir, uri);
    path.exists(filename, function(exists) {
        if(exists) {
            f = fs.createReadStream(filename);
            f.on('open', function() {
                response.writeHead(200);
            });

            f.on('data', function(chunk) {
                response.write(chunk);
            });

            f.on('error', function(err) {
                // ...
            });

            f.on('end', function() {
                response.end();
            });
        } else {
            response.writeHead(404);
-           response.end();
        }
    });
}).listen(port);
```

stream-file-server.js

innoQ

```
var hashFile = function(filename, cb) {
  path.exists(filename, function(exists) {
    if(exists) {
      r = fs.createReadStream(filename);
      var hash = crypto.createHash('md5');
      r.on('data', function(data) {
        hash.update(data);
      });
      r.on('end', function() {
        cb(hash.digest('base64'));
      });
    } else {
      throw 'File ' + filename + ' does not exist or can not be read';
    }
  });
}

var filename = path.join(process.argv[2]);
hashFile(filename, function(hash) {
  console.log(filename + ': ' + hash);
});
```

hash-file-stream.js *(see stream-file-server-md5.js)*

innoQ

```
var options = function(request) {
  // ...
}

http.createServer(function(request, response) {
  sys.log("--> " + request.url);
  var remoteRequest = http.request(options(request), function (remoteResponse) {
    response.writeHead(remoteResponse.statusCode, remoteResponse.headers);
    remoteResponse.on('data', function (chunk) {
      response.write(chunk);
    });
    remoteResponse.on('end', function () {
      sys.log("<-- " + response.statusCode + " " +  request.url);
      response.end();
    });
  });
  request.on('data', function (chunk) {
    remoteRequest.write(chunk);
  });
  request.on('end', function () {
    remoteRequest.end();
  });
}).listen(port);
```

proxy.js

```
http.createServer(function(request, response) {
  sys.log("--> " + request.url);
  var remoteRequest = http.request(options(request), function (remoteResponse) {
    response.writeHead(remoteResponse.statusCode, remoteResponse.headers);
    remoteResponse.on('end', function () {
      sys.log("<-- " + response.statusCode + " " +  request.url);
    });
    util.pump(remoteResponse, response);
  });
  util.pump(request, remoteRequest);
}).listen(port);
```

proxy-pump.js

innoQ

Asynchronous Programming Challenges

or:

Why Programming with Callbacks Sucks

```
var bold = function(text) {
  return text.bold();
};

var capitalize = function(text) {
  return text.toUpperCase();
};

console.log("Synchronous:");
var result1 = capitalize("Hello, synchronous world.");
var result2 = bold(result1);
console.log("Sync result is " + result2);
```

async1.js

innoQ

```javascript
var boldAsync = function(text, callback) {
  setTimeout(function (text) {
    callback(text.bold());
  }, 100, text);
};

var capitalizeAsync = function(text, callback) {
  setTimeout(function (text) {
    callback(text.toUpperCase());
  }, 100, text);
};
```

async1.js

innoQ

```
var boldAsync = function(text, callback) {
  setTimeout(function (text) {
    callback(text.bold());
  }, 100, text);
};

var capitalizeAsync = function(text, callback) {
  setTimeout(function (text) {
    callback(text.toUpperCase());
  }, 100, text);
};


console.log("Asynchronous:");
capitalizeAsync("Hello, asynchronous world.", function(result1) {
  boldAsync(result1, function(result2) {
    console.log("Async result is " + result2);
  });
});
```

async1.js

```
try {
  console.log("Synchronous:");
  var result1 = capitalize(null);
  var result2 = bold(result1);
  console.log("Sync result is " + result2);
} catch (exception) {
  console.log("Sync exception caught: " + exception);
}
```

async2.js

```
try {
  console.log("Asynchronous:");
  capitalizeAsync(text, function(result1) {
    boldAsync(result1, function(result2) {
      console.log("Async result is " + result2);
    });
  });
} catch (exception) {
  console.log("Async exception caught: " + exception);
}
```

async2.js

innoQ

```
// bad, don't do this

try {
  console.log("Asynchronous:");
  capitalizeAsync(text, function(result1) {
    boldAsync(result1, function(result2) {
      console.log("Async result is " + result2);
    });
  });
} catch (exception) {
  console.log("Async exception caught: " + exception);
}
```

async2.js

innoQ

```
var boldAsync = function(text, callback) {
  setTimeout(function (text) {
    try {
      callback(null, text.bold());
    } catch (exception) {
      callback(exception);
    }
  }, 100, text);
};

var capitalizeAsync = function(text, callback) {
  setTimeout(function (text) {
    try {
      callback(null, text.toUpperCase());
    } catch (exception) {
      callback(exception);
    }
  }, 100, text);
};
```

async3.js

innoQ

```
capitalizeAsync(text, function(err, result1) {
  if (!err) {
    boldAsync(result1, function(err, result2) {
      if (!err) {
        console.log("Async result is " + result2);
      } else {
        console.log("Handling async error: " + err);
      }
    });
  } else {
    console.log("Handling async error: " + err);
  }
});
```

async3.js

```
var handleError = function(err, fn) {
  if (err) {
    console.log("Handling async error: " + err);
  } else {
    fn();
  }
}

capitalizeAsync(text, function(err, result1) {
  handleError(err, function () {
    boldAsync(result1, function(err, result2) {
      handleError(err, function () {
        console.log("Async result is " + result2);
      });
    });
  });
});
```

async3.js

```
var step = require("step");
step(
  function () {
    capitalizeAsync(text, this);
  },
  function (err, result) {
    if (err) throw err;
    boldAsync(result, this);
  },
  function(err, result) {
    if (err) {
      console.log("Handling async error: " + err);
    } else {
      console.log("Async result is " + result);
    }
  }
);
```

async3.js

```
var words = ['one', 'two', 'three', 'four', 'five'];
var upcasedWords = [];


words.forEach(function (word) {
  capitalize(word, function(err, word) {
    upcasedWords.push(word);
  });
});
console.log('Done, upcased words: <'
            + upcasedWords.join(' ') + '>');
```

parallel1.js

innoQ

```
var words = ['one', 'two', 'three', 'four', 'five'];
var upcasedWords = [];

// bad, don't do this
words.forEach(function (word) {
  capitalize(word, function(err, word) {
    upcasedWords.push(word);
  });
});
console.log('Done, upcased words: <'
            + upcasedWords.join(' ') + '>');
```

parallel1.js

innoQ

```
var count = words.length;
words.forEach(function (word) {
  capitalize(word, function(err, word) {
    upcasedWords.push(word);
    if (--count === 0) {
      console.log('Done, upcased words: <'
                    + upcasedWords.join(' ') + '>');
    }
  });
});
```

parallel1.js

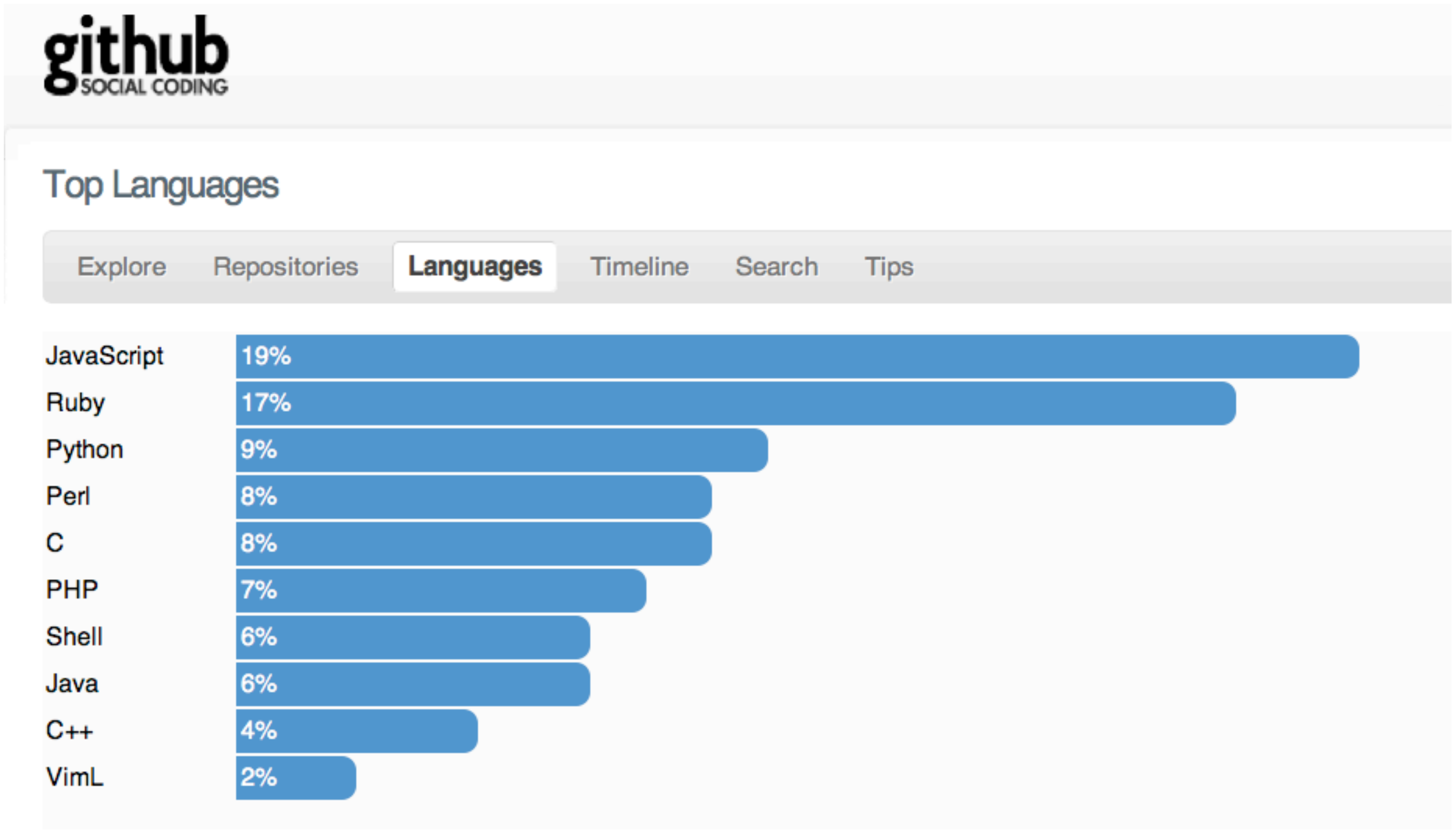
innoQ

```
var words = ['one', 'two', 'three', 'four', 'five'];

step(
  function () {
    var i, length;
    for (i = 0, length = words.length; i < length; i++) {
      capitalize(words[i], this.parallel());
    }
  },

  function (err) {
    if (err) throw err;
    var upcasedWords = Array.prototype.slice.call(arguments);
    upcasedWords.shift();
    console.log('Done, upcased words: <'
            + upcasedWords.join(' ') + '>');
  }
);
```

parallel2.js

innoQ

Tools & Ecosystem



JavaScript	19%
Ruby	17%
Python	9%
Perl	8%
C	8%
PHP	7%
Shell	6%
Java	6%
C++	4%
VimL	2%

npm	node package manager
Connect	Asynchronous, low-level HTTP handler framework inspired by Rack/WSGI
Express	Sinatra-inspired Web framework on top of Connect
multi-node	Spawns child processes sharing listeners
node-inspector	Visual debugger for Node.js
›700 more modules	see https://github.com/joyent/node/wiki/modules

```
var multi = require("multi-node");

var server = http.createServer(function(request, response) {
  var uri = url.parse(request.url).pathname;
  var filename = path.join(process.cwd(), dir, uri);
  path.exists(filename, function(exists) {
    if(exists) {
      fs.readFile(filename, function(err, data) {
        if (err) {
          sys.log('Error serving file ' + filename + ' ' + err);
          sys.log('request: ' + uri);
        }
        response.writeHead(200, {
          'X-Node-Id': process.pid
        });
        response.end(data);
      });
    } else {
      response.writeHead(404);
      response.end();
    }
  });
});


var nodes = multi.listen({ port: port, nodes: 10 }, server);
sys.log("Server " + process.pid + " running at http://localhost:" + port);
```

multi-file-server.js

Summary

Node.js popularizes the “right way” of network programming

JavaScript doesn't suck as much as you think

There’s a smart and active community

Node.js is fun to use!

Thank you!

Q&A

innoQ