Introduction to Node.js

Mike Tunnicliffe

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

Who am I?

Mike Tunnicliffe

Work in virtual machine development
at IBM in Hursley

Currently working in the IBM SDK for Node.js development team and the IBM Java support.

What is Node.js?

The “theory”

Node.js is...

An **event-based**, **JavaScript** framework

for building **scalable** network applications;

well-suiting to **“real-time”** delivery

of data to distributed clients.

Scalable “real-time” what?

For example: A **chat application**

Real-time in that updates published as they happen

Scalable to large numbers of concurrent users

What do you mean "published as they happen"?

If one user submits a chat message

Other users should (ideally) see that message immediately

Achieving "real-time"

Assuming a client-server application, the **clients**:

poll the server very frequently to check for updates

or

keep a connection open and server pushes updates

Frequent polling wasteful

If **polling**, then the more **frequently** it happens, then the more **likely** there will be **no** useful **updates** to observe

Generates network **traffic** and **load** on server to re-establish connection, prepare and send responses

Keeping a connection open

would be more **efficient**

But at what **cost**?

Generic server

Socket **bind** to a port (eg 80)

Repeatedly:

Socket **accept** to get new connection

Generate and send **response**

Typical approach for server

Use a **one thread (or process) per connection** model

This **does not scale** well to many concurrent users
if clients keep connections open

Each thread **consumes** relatively large amount of **memory**,
and spend most of their time **idle**

A different approach

Single thread handles connections

using **event-based** approach plus **async IO**

to allow processing of concurrent requests

Analogies

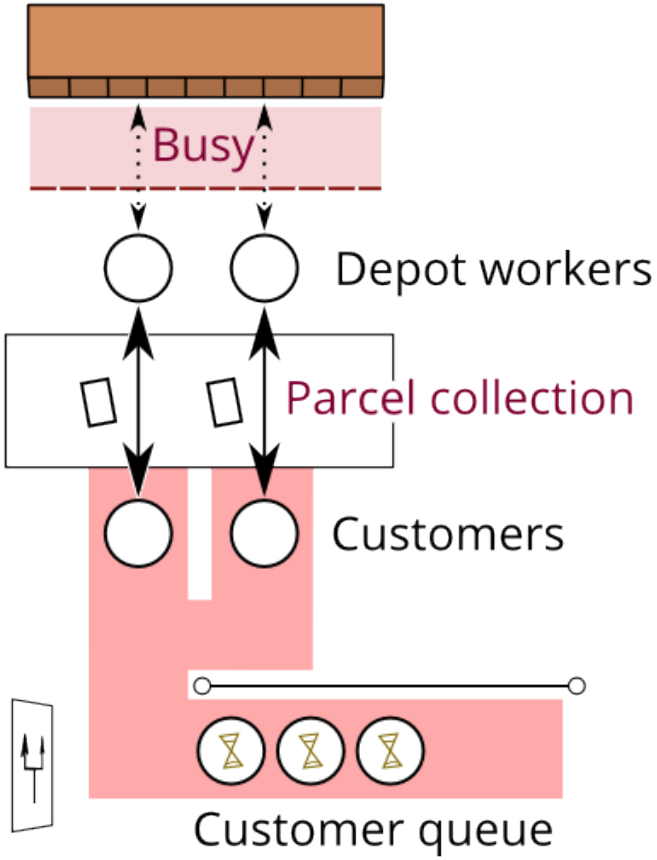
Parcel collection depot
– One thread per connection

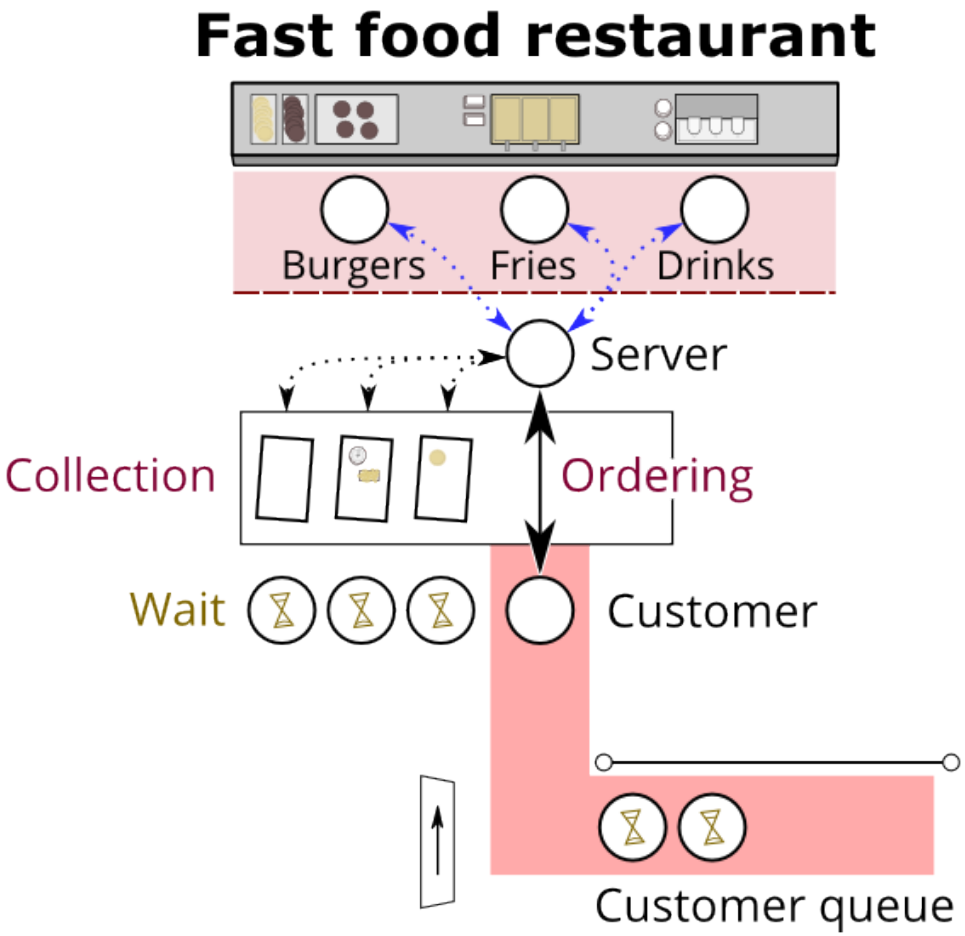
Depot worker **completes all tickets** for a customer before serving the next

Fast food restaurant
– Events and async IO

Server **takes** and dispatches **orders** moving through the queue without filling them; server **interleaves** this order taking with **filling orders** as food arrives

Parcel collection depot





Benefits

Cooperative multitasking provides concurrency
at **low overhead**

Drawbacks

Single thread only utilizes one CPU/core/SMT

Tasks **must complete quickly** so that other events are not delayed

Mitigations

Stick to right jobs for the tool (**IO-bound**)

Split CPU-bound tasks to a separate **back-end processes**

Use **multiple** Node **instances** – max 1 per CPU/core/SMT

Result

Scalable, responsive and **efficient**

provided application tasks are **IO-bound**

So...

An **event-based**, **JavaScript** framework

for building **scalable** network applications;

well-suiting to **“real-time”** delivery

of data to distributed clients.

But, why JavaScript?

Dynamic/higher level languages appealing

First class functions and **closures** fit well with events

Good **competition** in JavaScript runtimes

Google Chrome **V8** JavaScript engine is **fast**

JavaScript is in the **browser** and is **event-based**

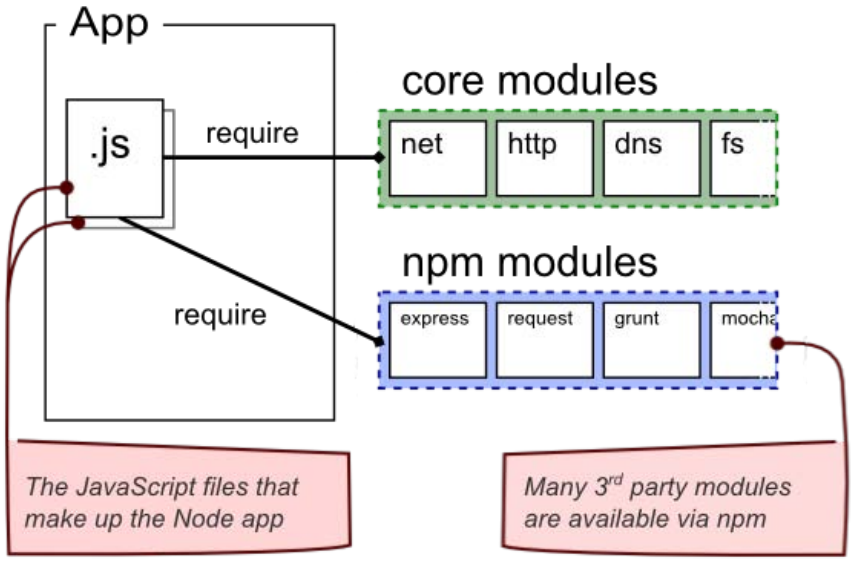
JavaScript has no **IO** library **baggage**

How does Node.js work?

The “practical”

A Node application is…

One or more JavaScript files



Hello HTTP World!

helloworld_http.js:

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

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World!\n");
  response.end();
}).listen(8080);
```

(QUICK DEMO)

Hello TCP World!

helloworld_tcp.js:

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

net.createServer(function(socket) {
  socket.write("Hello World!\n");
  socket.end();
}).listen(8080);
```

(QUICK DEMO)

Hello World!

helloworld_console.js:

```
console.log('Hello World!');
```

helloworld_console_delayed.js:

```
setTimeout(function() { console.log('World!'); }, 2000);
console.log('Hello');
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat server

```
var fs = require('fs'), http = require('http');                    // core mods
var sio = require('socket.io'), validator = require('validator'); // npm mods

fs.readFile('client.html', function(err, client_html) {
  if (err) { console.log('Failed: ' + err); return; }
  var http_server = http.createServer(function(request, response) {
    response.writeHead(200, {'Content-Type': 'text/html'});
    response.end(client_html);
  });
  var sio_server = sio.listen(http_server);
  http_server.listen(8080);

  sio_server.sockets.on('connection', function(socket) {
    socket.on('message', function(msg) {
      sio_server.sockets.emit('message', validator.escape(msg));
    });
  });
});
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat client

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example chat</title>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io.connect('localhost:8080');
      socket.on('message', function(msg) {
        var log = document.getElementById('log');
        log.innerHTML = msg + '<br>' + log.innerHTML;
      });

      function send() {
        var msg = document.getElementById('message').value;
        socket.emit('message', msg);
        document.getElementById('message').value = '';
      }
    </script>
  </head>

  <body>
    <input type="text" id="message" onkeypress="if (event.which==13) { send(); }">
    <button type="button" onclick="send()">Send</button>
    <div id="log"></div>
  </body>
</html>
```

Basic chat demo

Counting lines

count_lines.js:

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

function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ':' + count);
  }
}

fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

Counting lines

count_lines.js:

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

function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ':' + count);
  }
}

fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

Counting lines

count_lines.js:

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

function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ':' + count);
  }
}

fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

Counting lines

count_lines.js:

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

function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ':' + count);
  }
}

fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

Event loop example

Hypothetical internal implementation of readFile():

```
function readFile(filename, cb) {
  nb_open(filename, function(err, fd) {
    if (err) cb(err, null);
    nb_read(fd, function(err, data) {
      cb(err, data);
      nb_close(fd);
    });
  });
}
```

Oops: Diagram in following animation contains a mistake

NOT STARTED

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

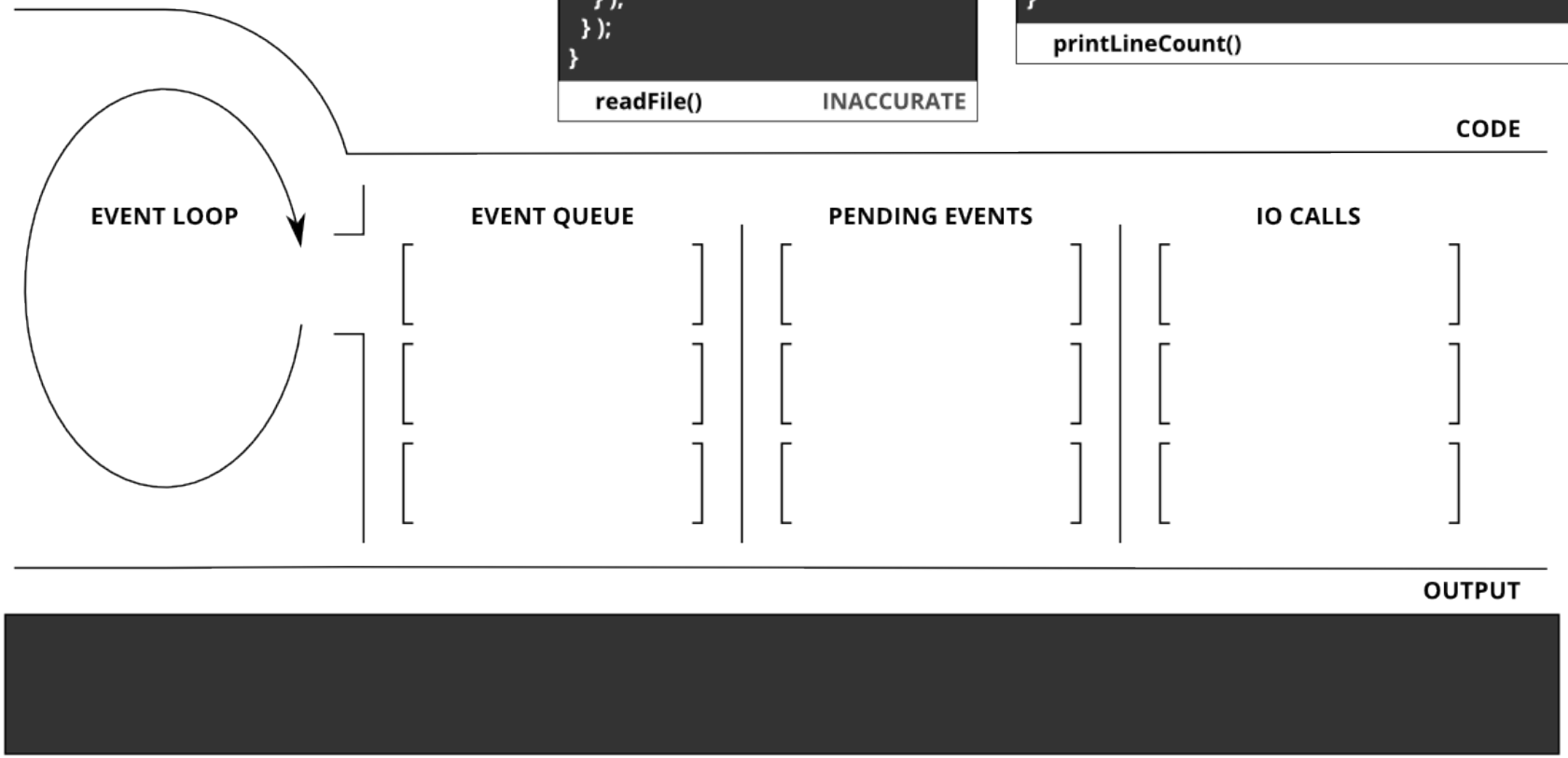
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

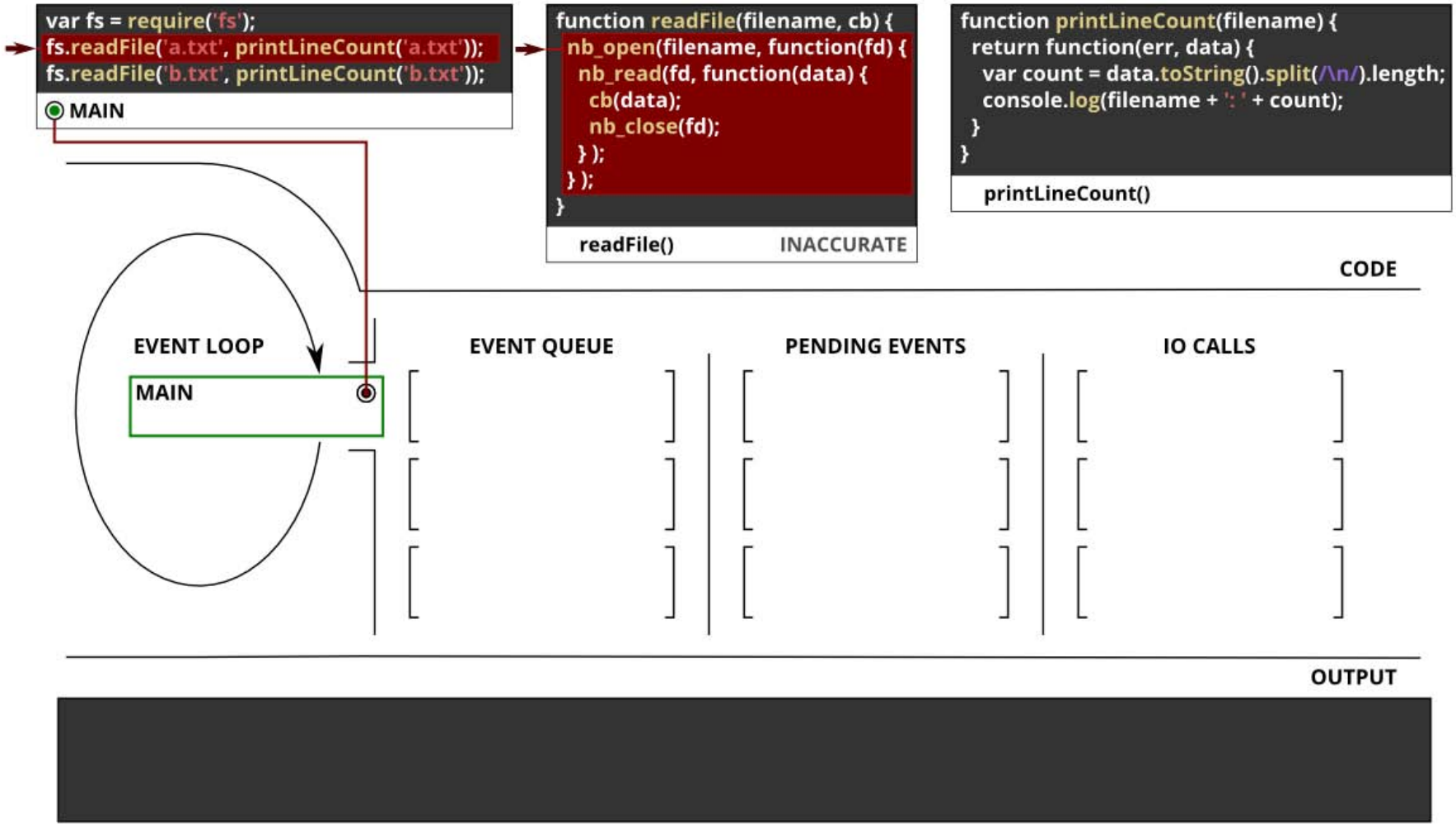
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

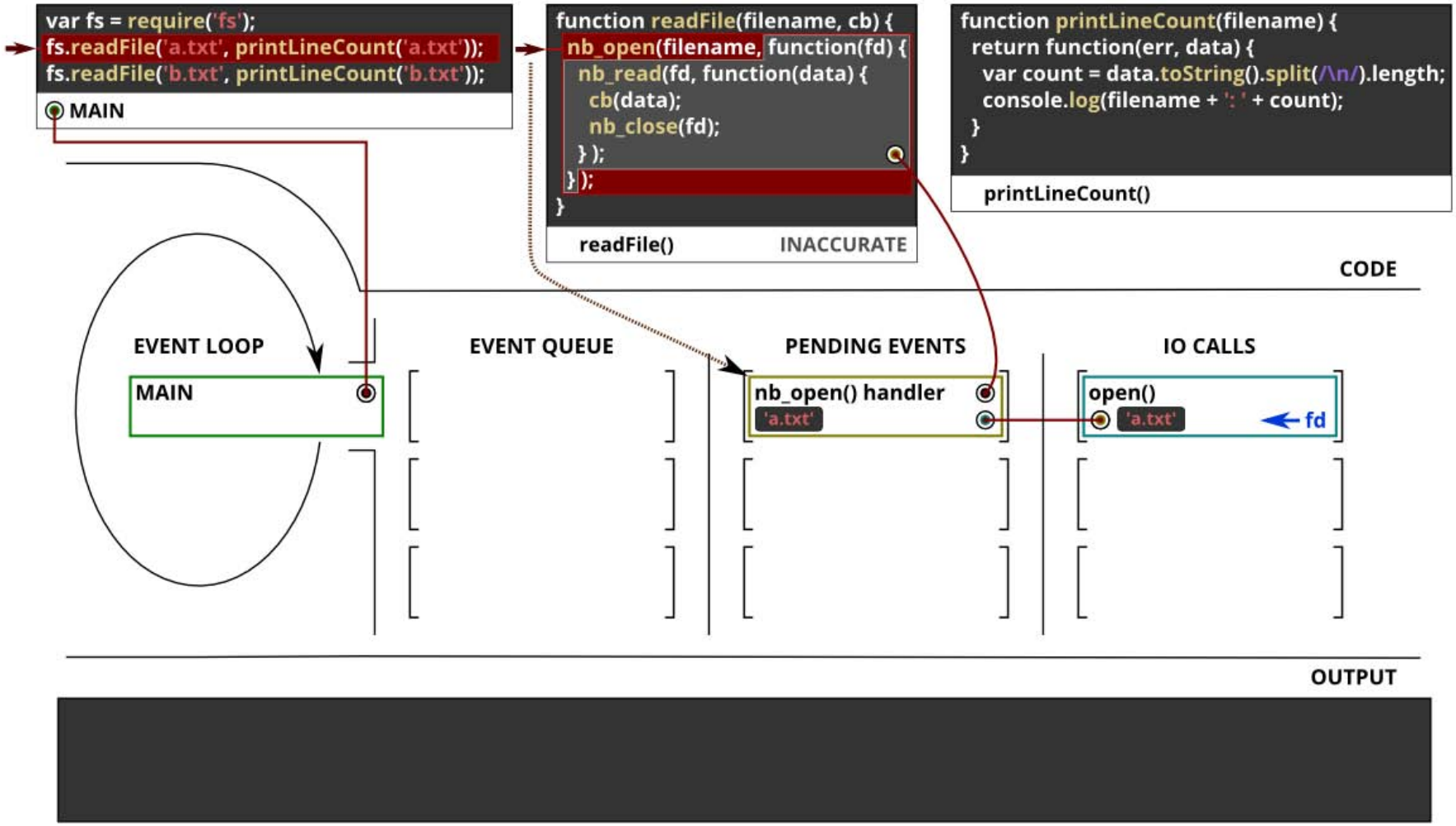
printLineCount()



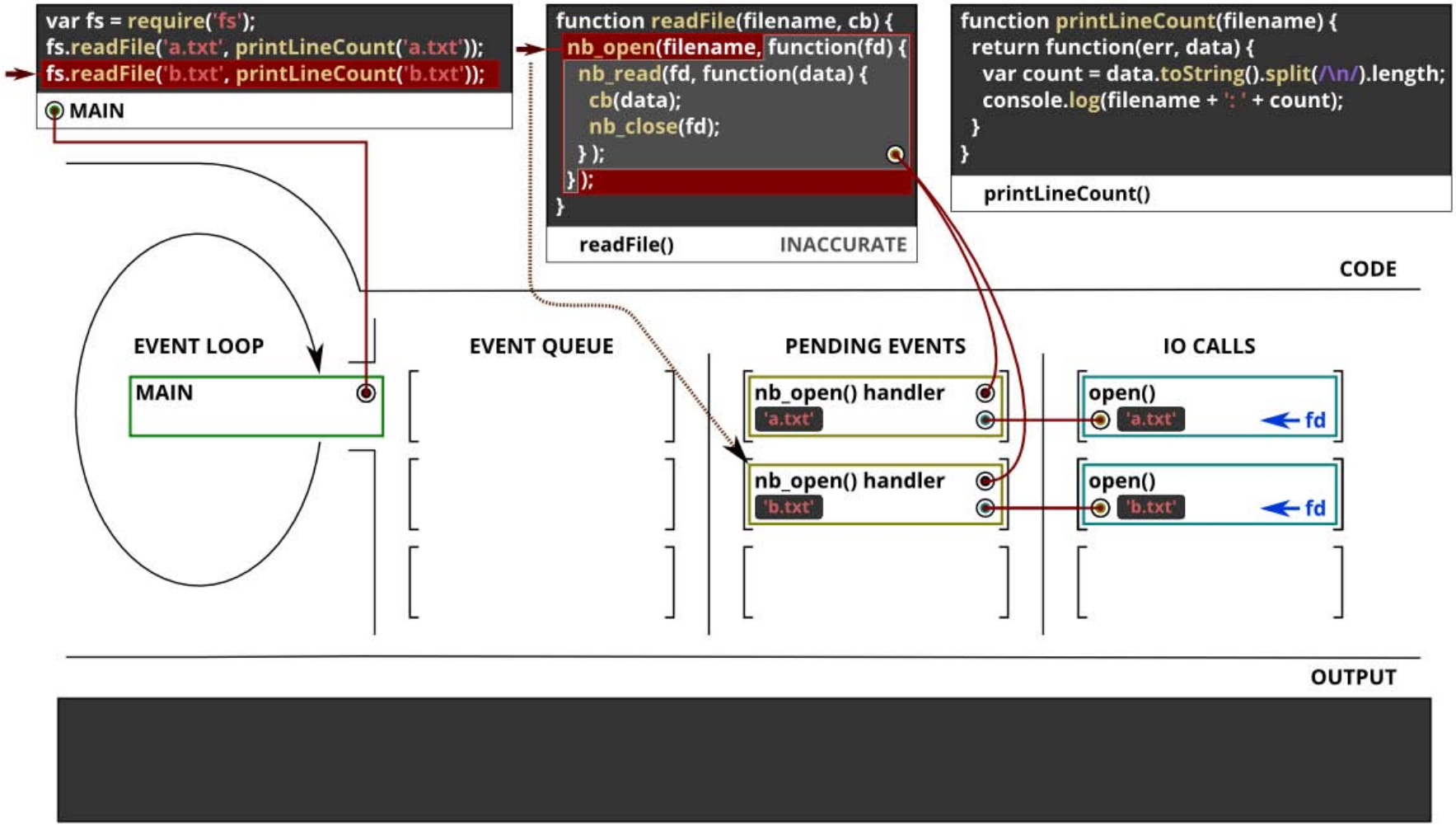
EXECUTING MAIN



INITIATING FILE OPEN OPERATION ('a.txt')



INITIATING 2nd FILE OPEN OPERATION ('b.txt')



WAITING FOR IO TO COMPLETE

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

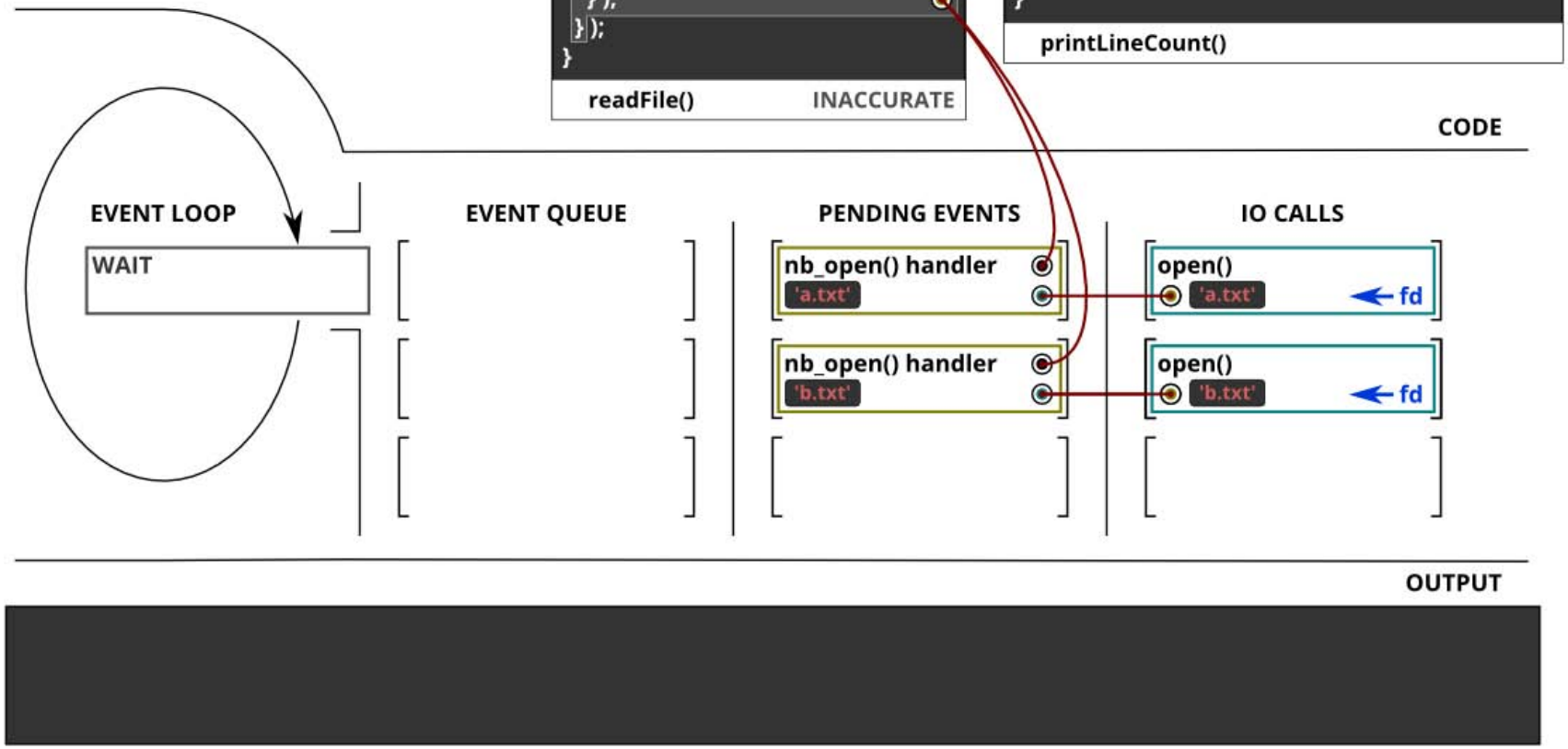
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



FILE OPEN OPERATION COMPLETES ('a.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

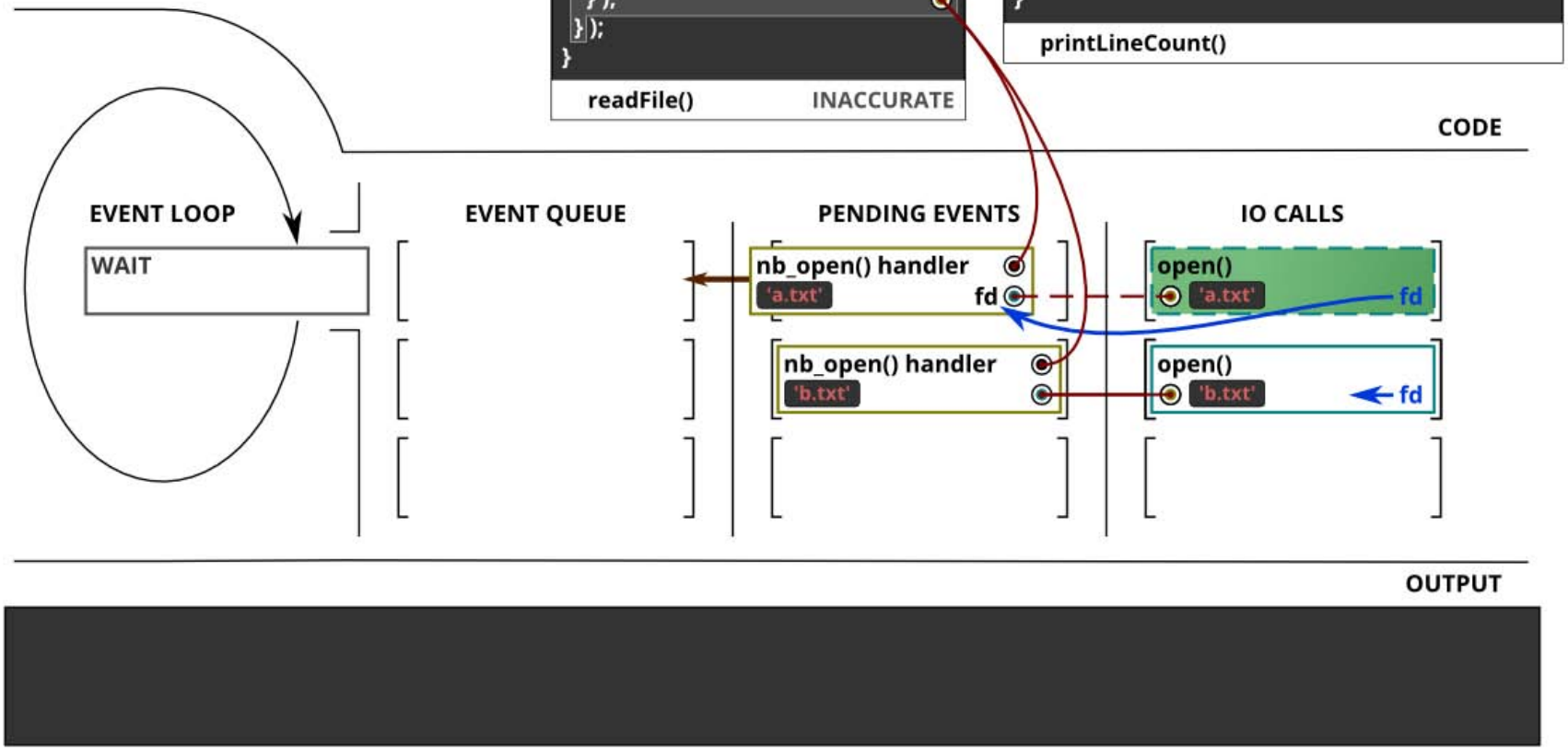
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



ADDING OPEN HANDLER ('a.txt') TO EVENT QUEUE

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

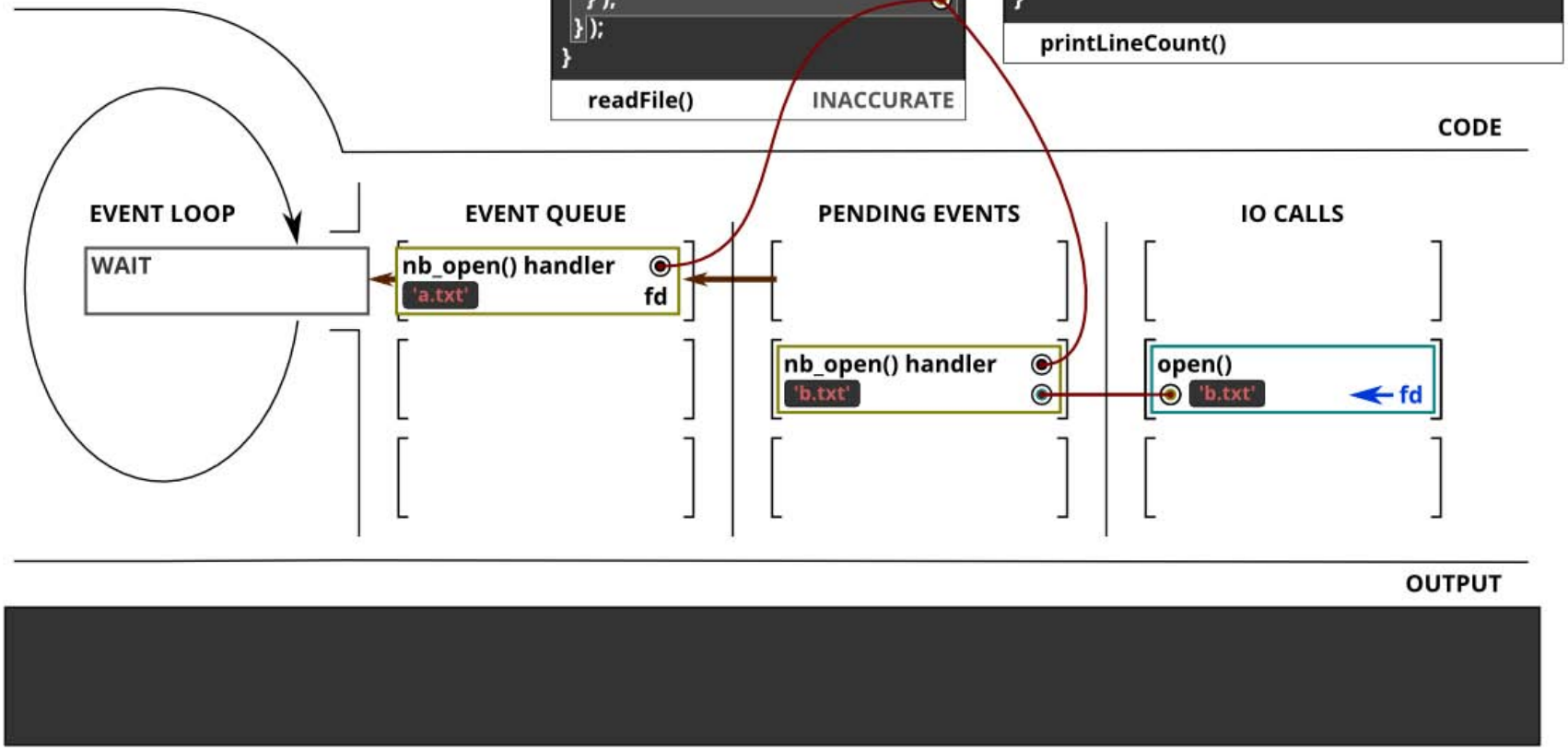
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

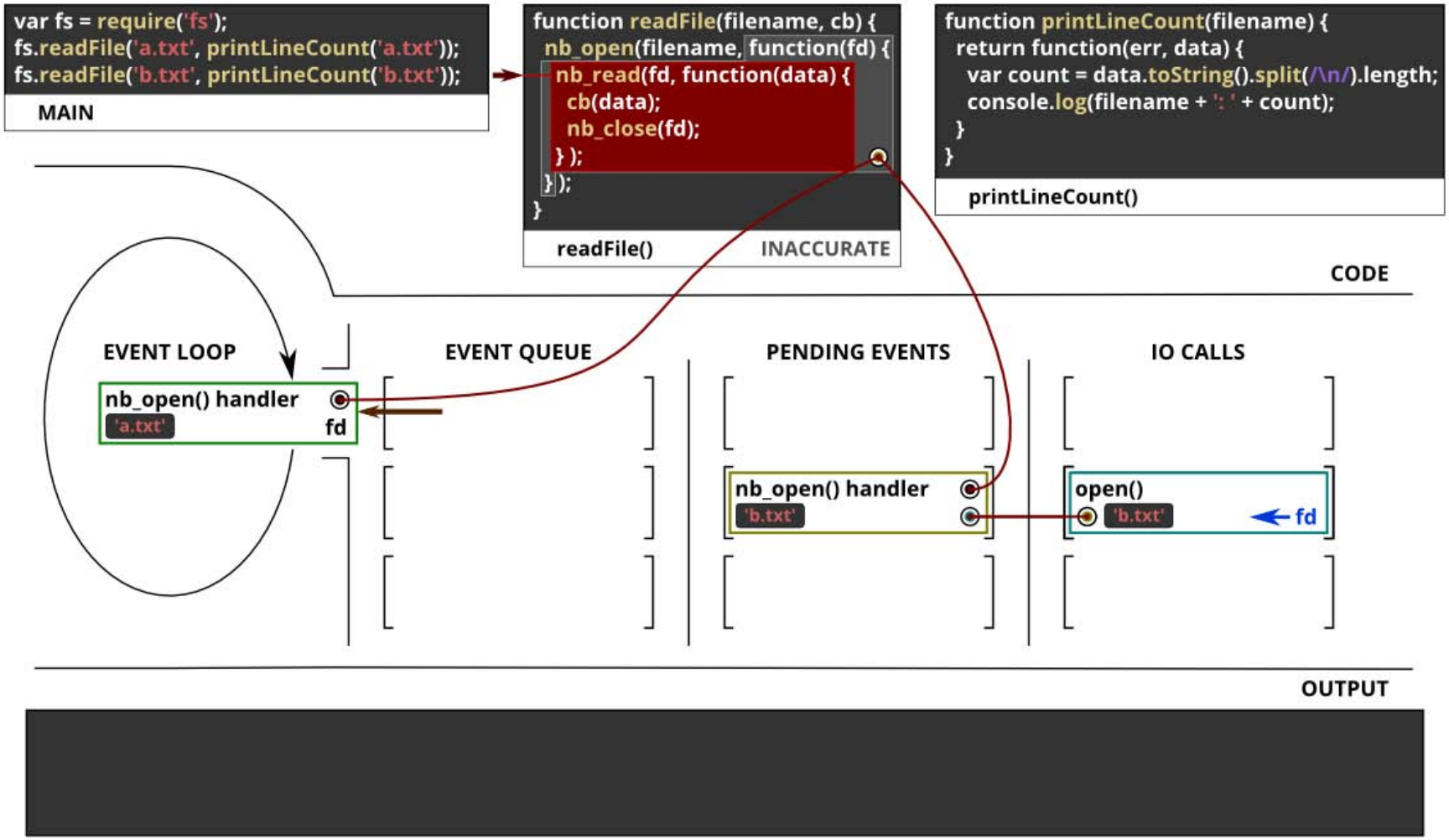
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

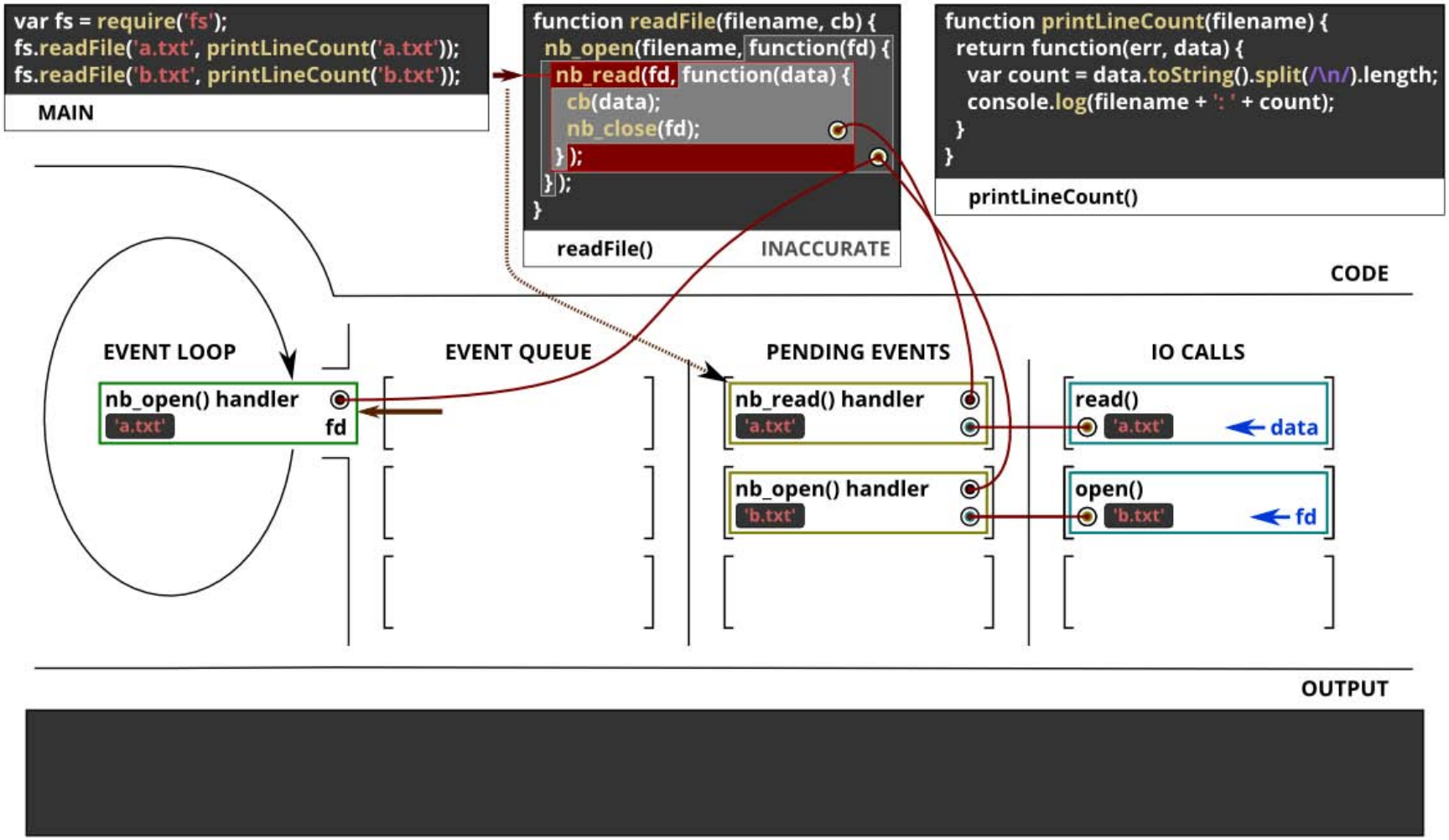
printLineCount()



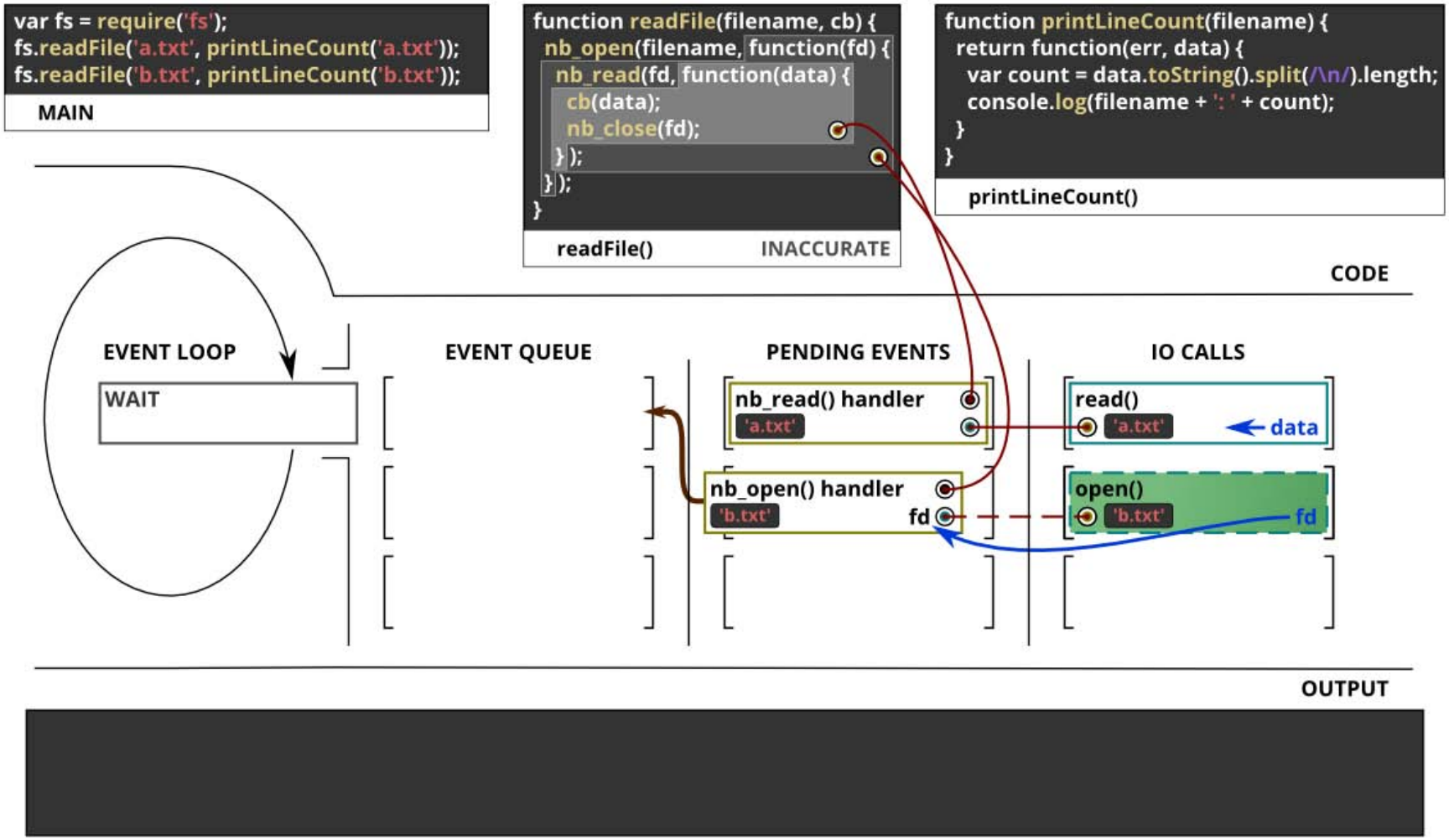
EXECUTING OPEN HANDLER ('a.txt')



INITIATING FILE READ OPERATION ('a.txt')



FILE OPEN OPERATION COMPLETES ('b.txt')



ADDING OPEN HANDLER ('b.txt') TO EVENT QUEUE

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

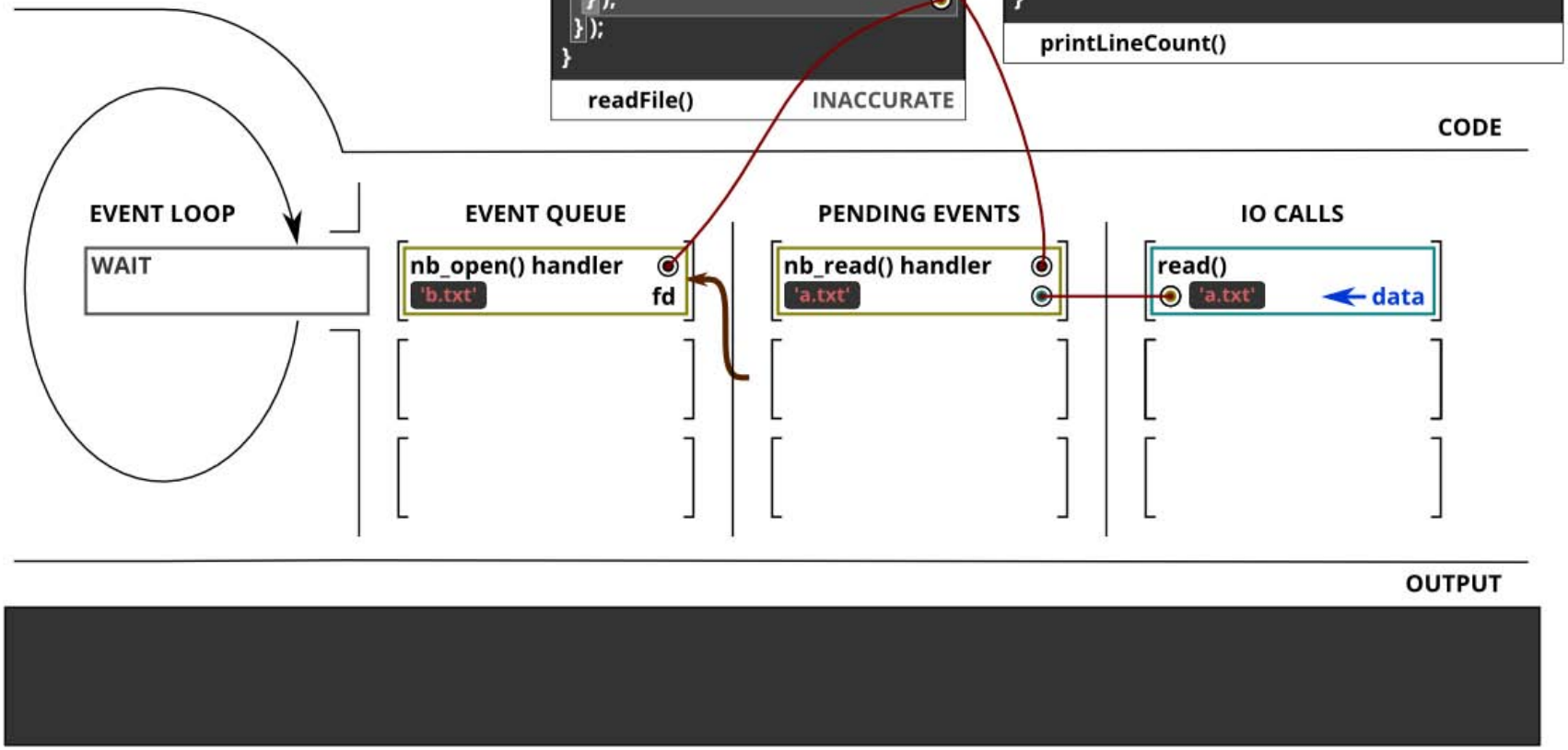
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



FILE READ OPERATION COMPLETES ('a.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

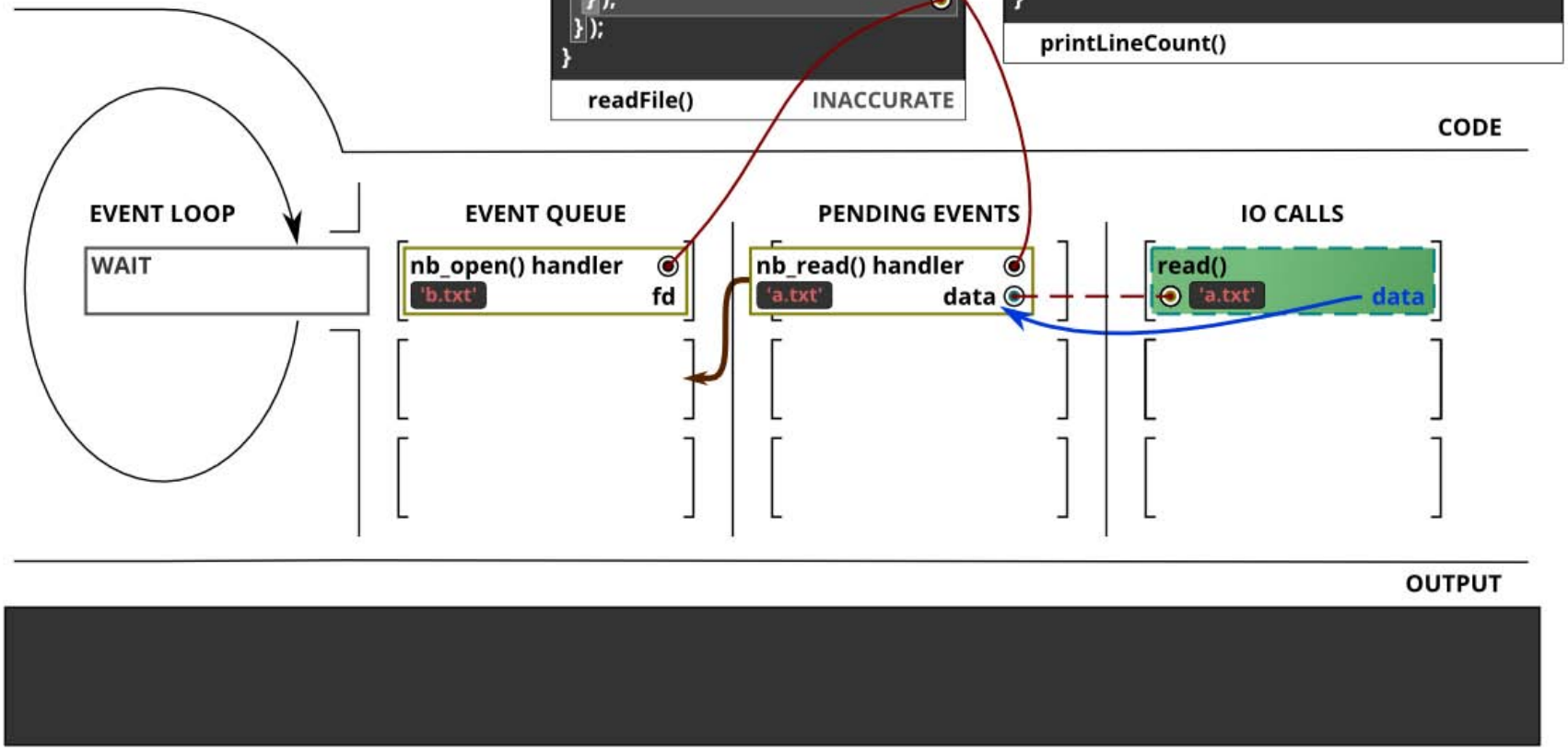
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

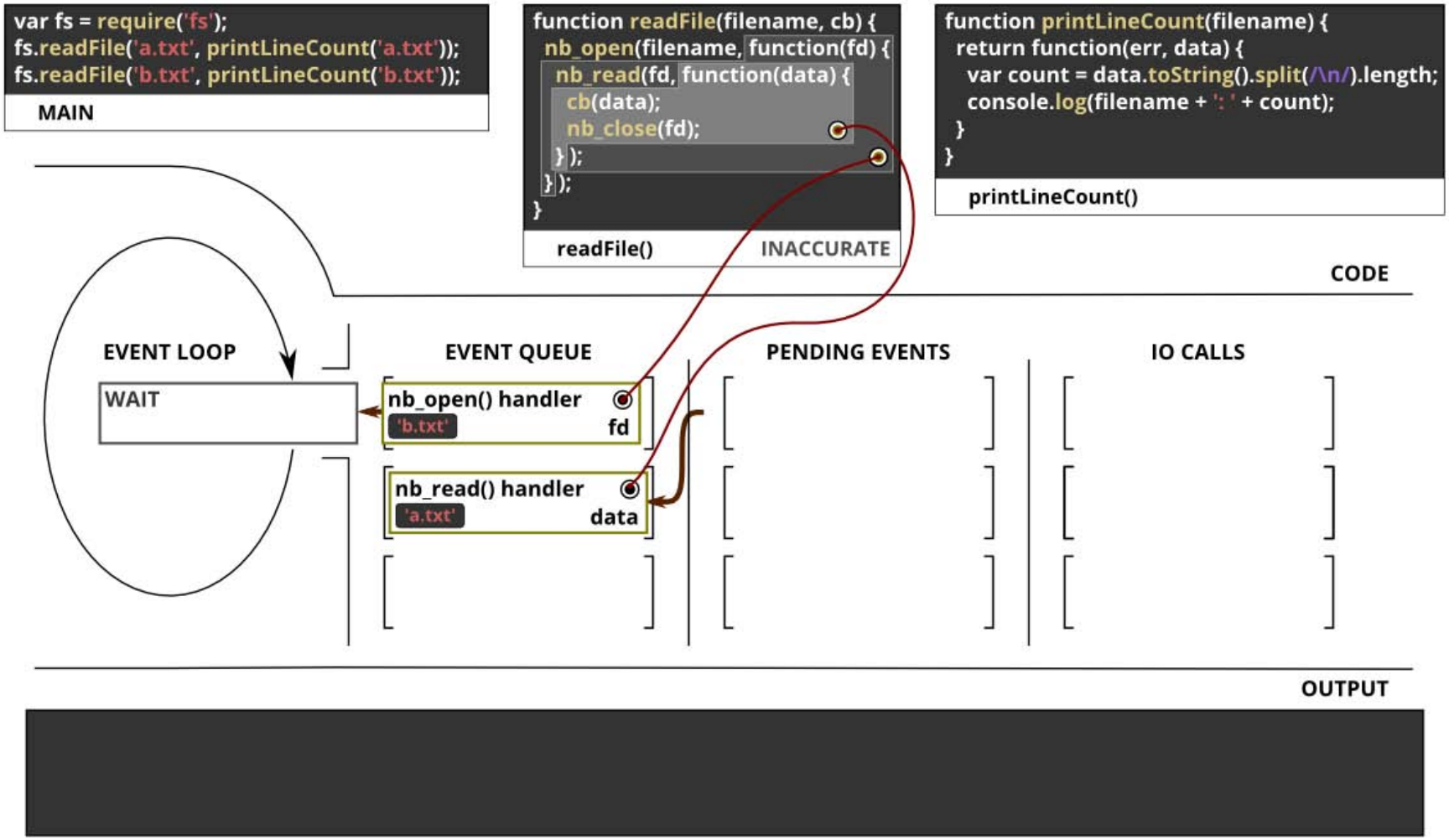
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

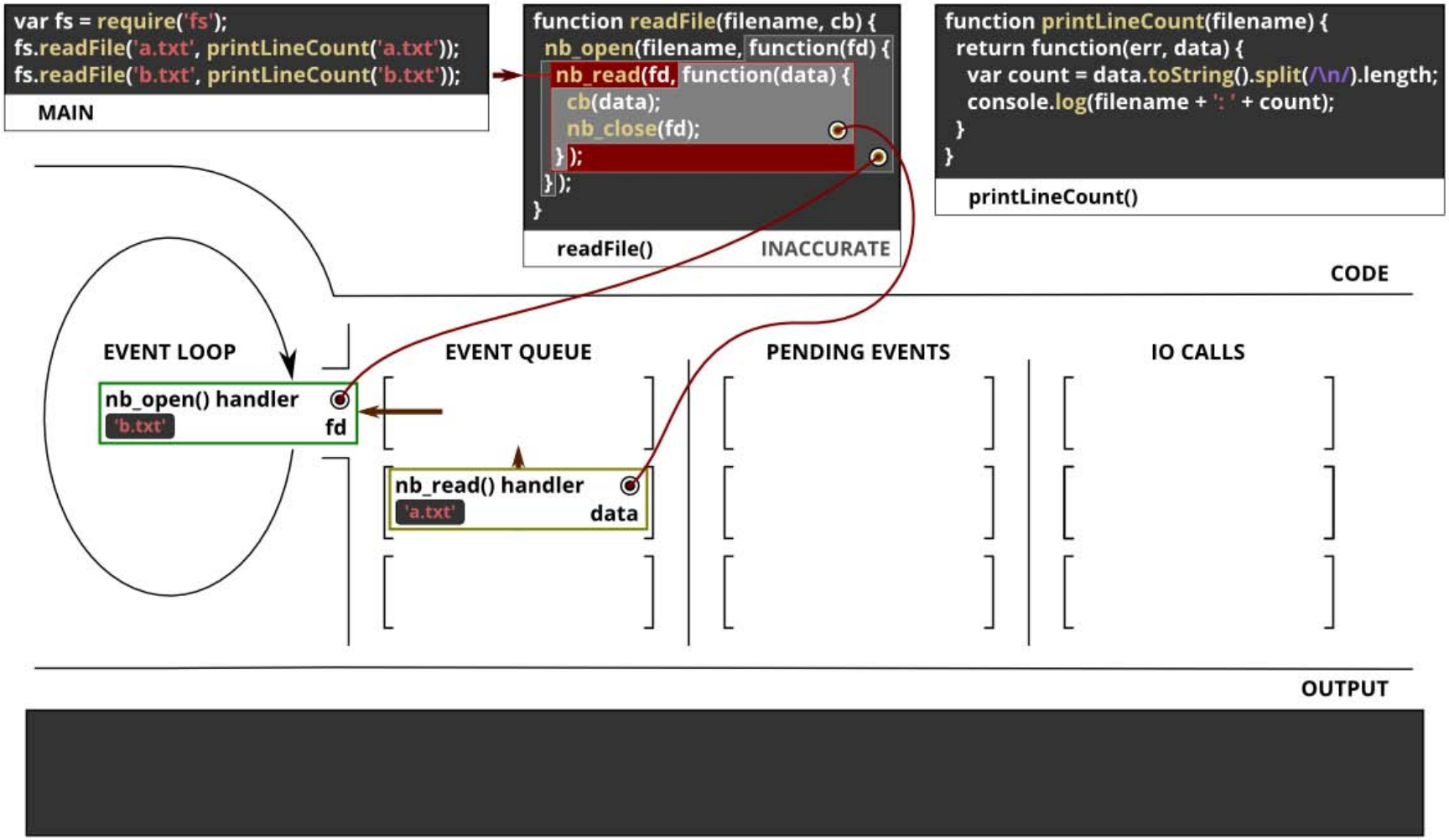
printLineCount()



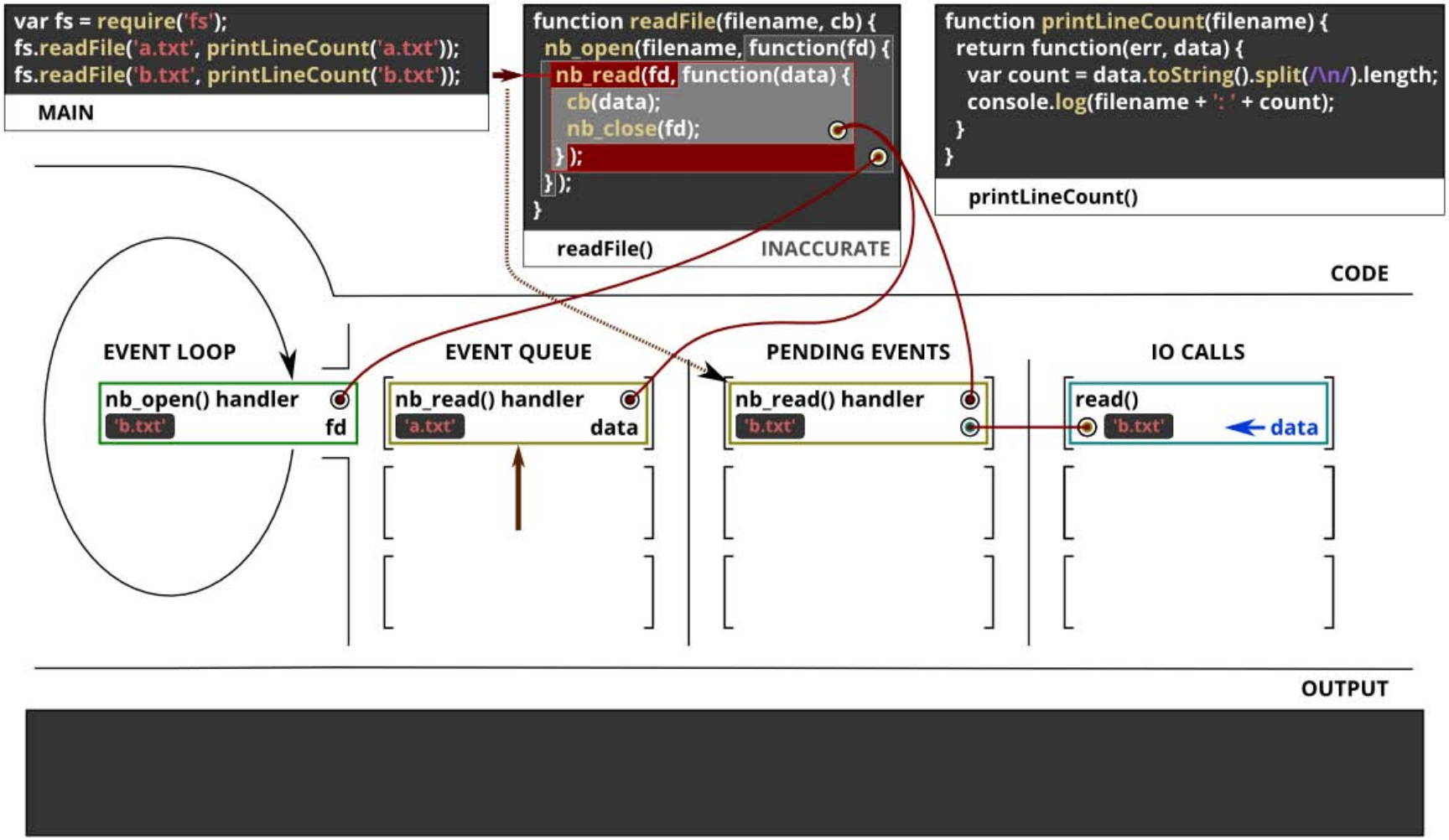
ADDING READ HANDLER ('a.txt') TO EVENT QUEUE



EXECUTING OPEN HANDLER ('b.txt')



INITIATING FILE READ OPERATION ('b.txt')



ABOUT TO EXECUTE READ HANDLER ('a.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

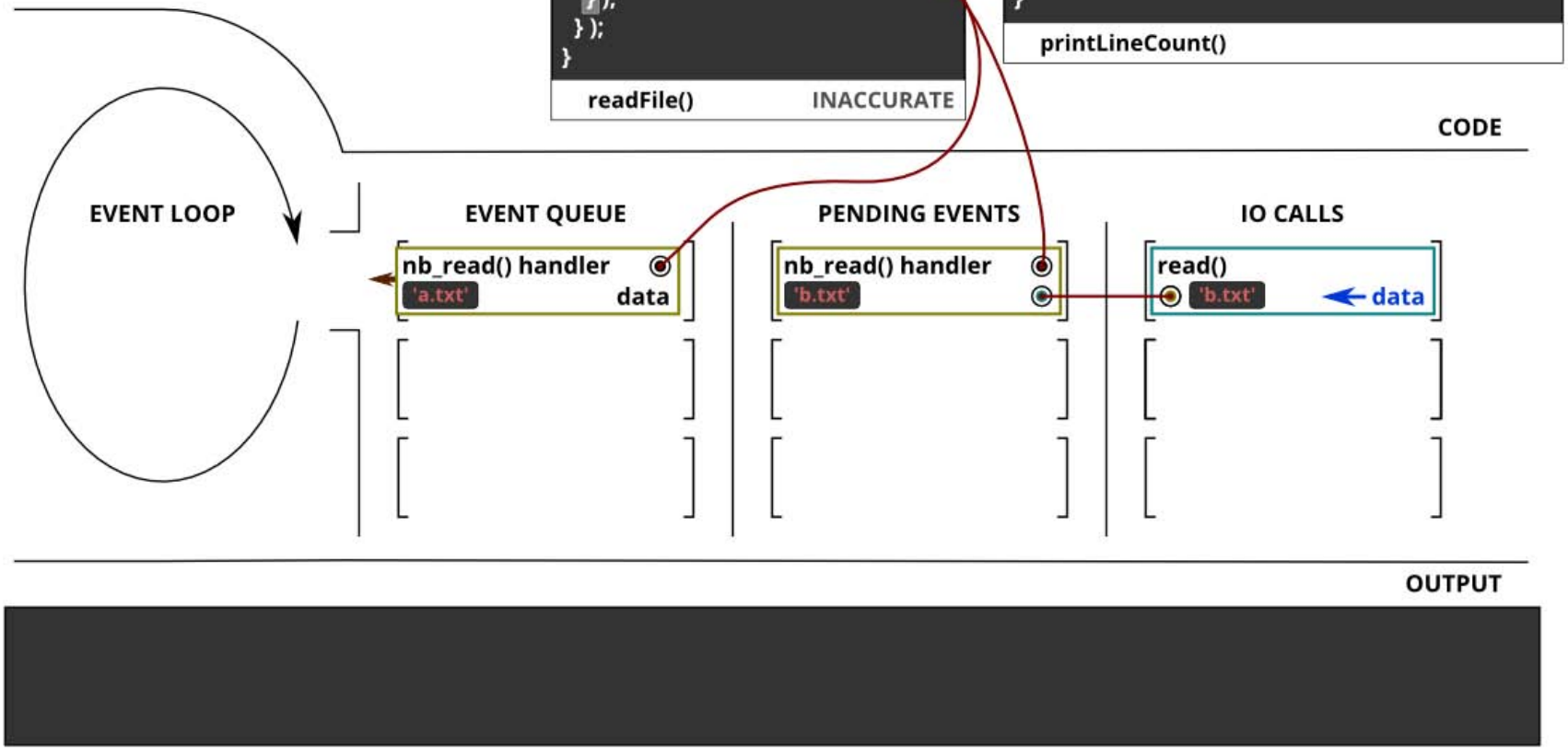
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

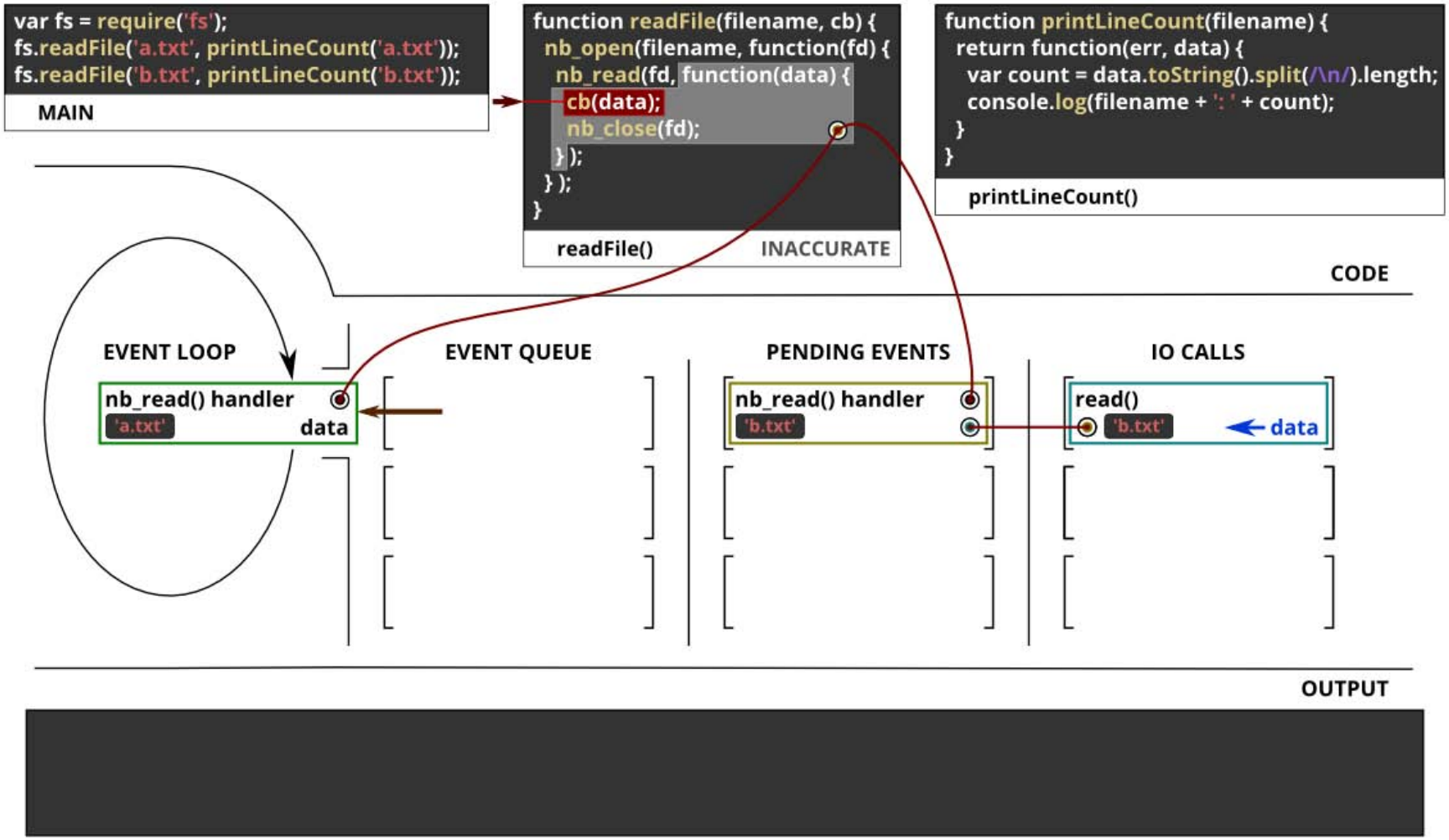
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

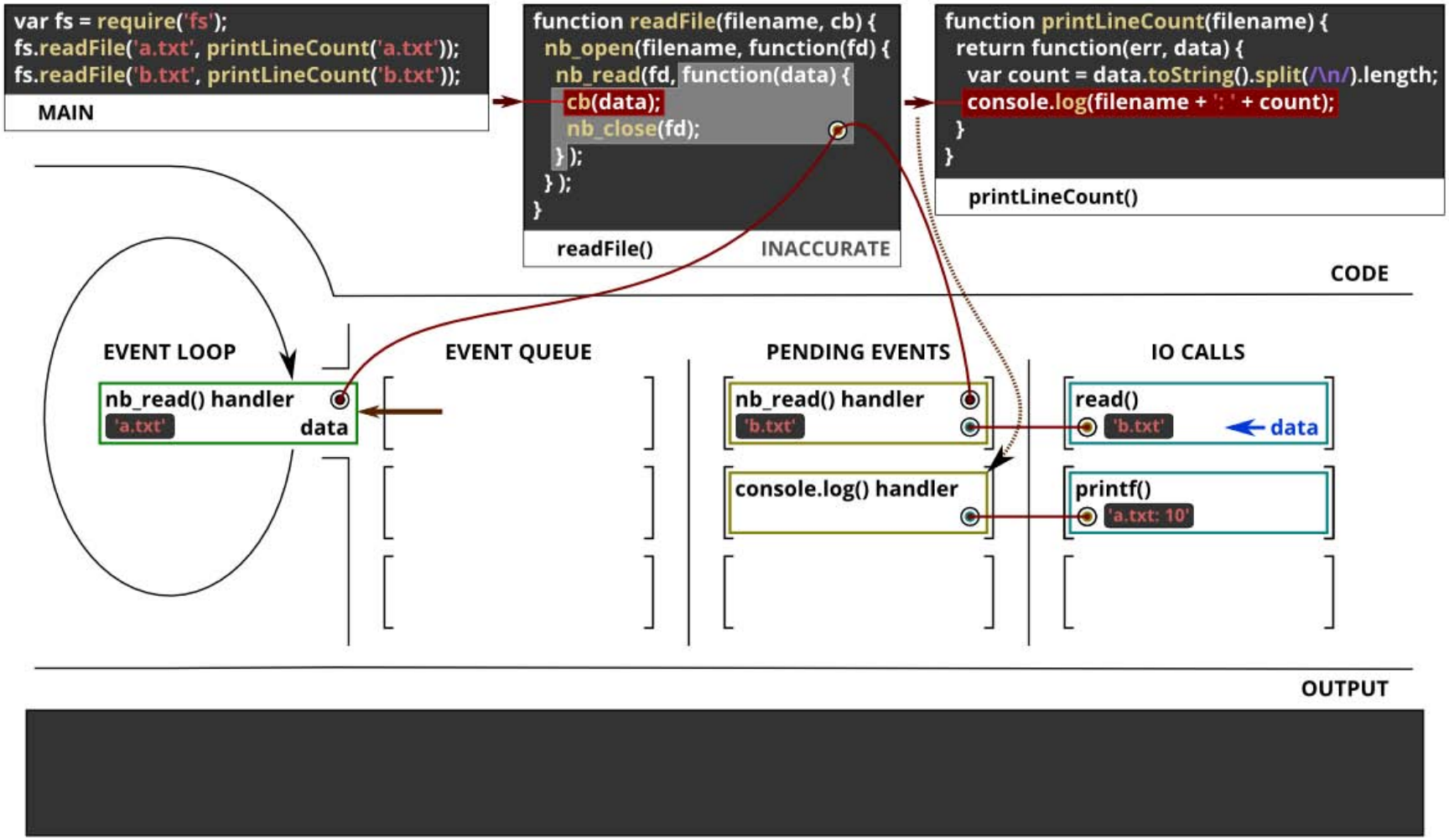
printLineCount()



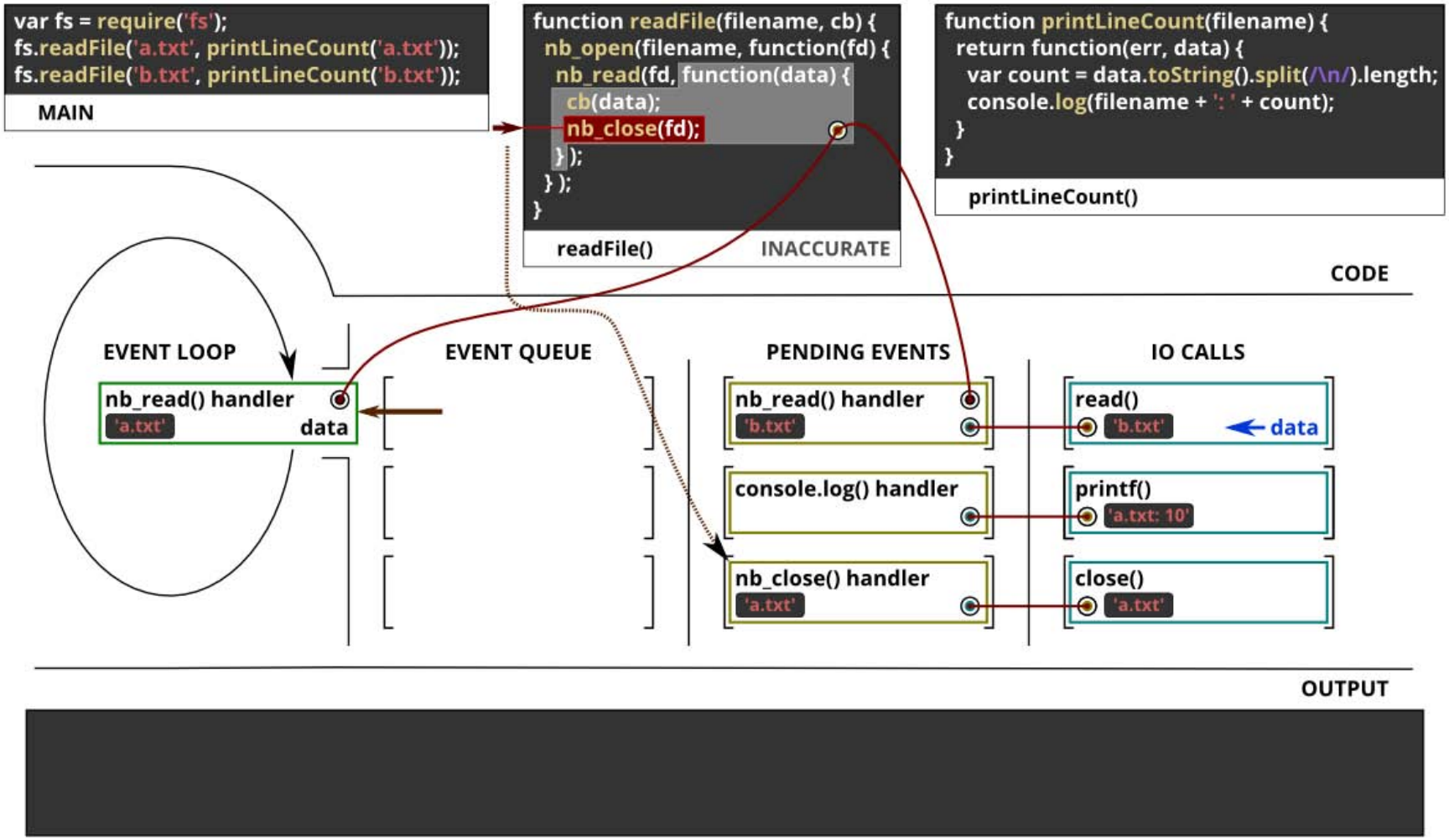
EXECUTING READ HANDLER ('a.txt')



INITIATING PRINT OPERATION ('a.txt: 10')



INITIATING FILE CLOSE OPERATION ('a.txt')



FILE CLOSE OPERATION COMPLETES ('a.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

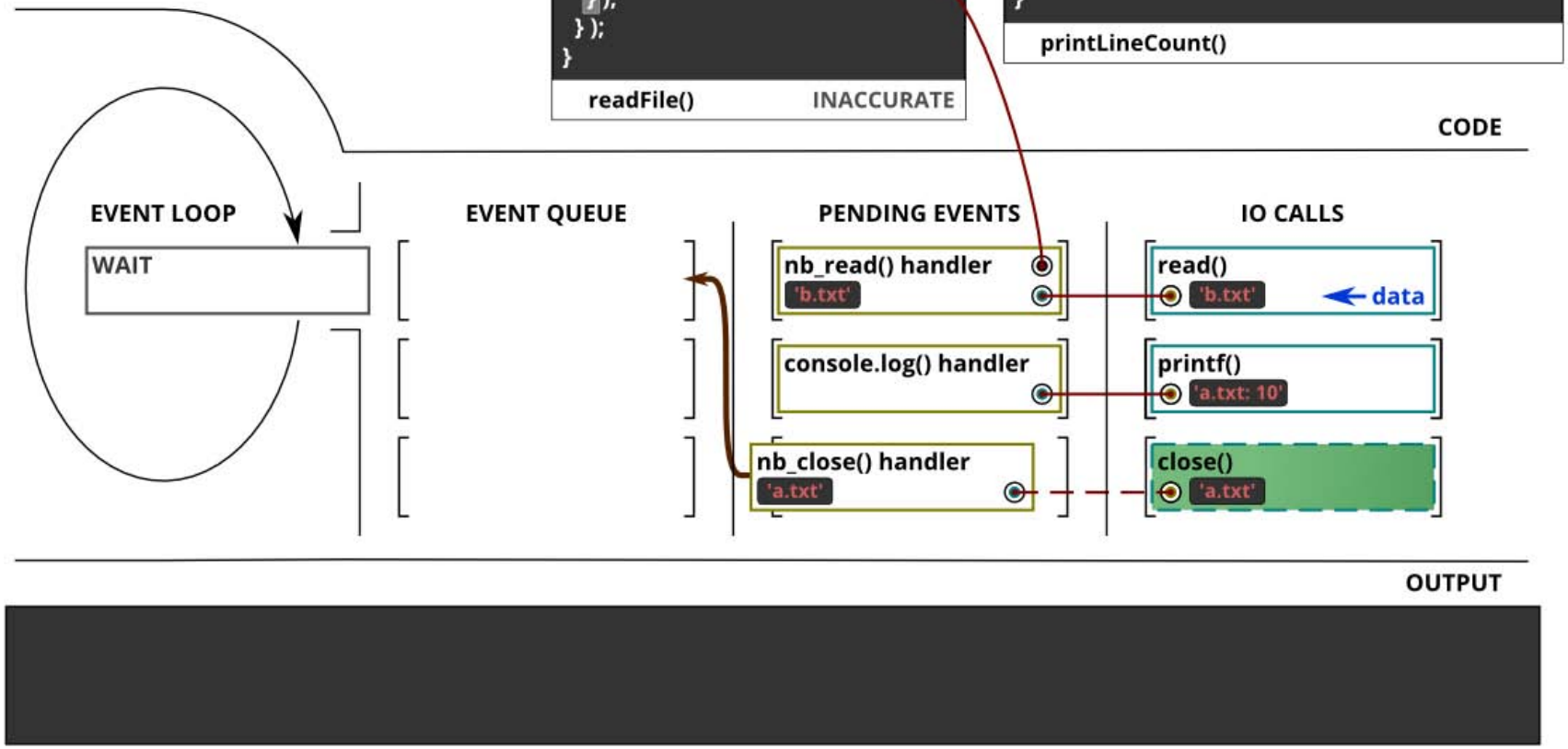
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



PRINT OPERATION COMPLETES ('a.txt: 10')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

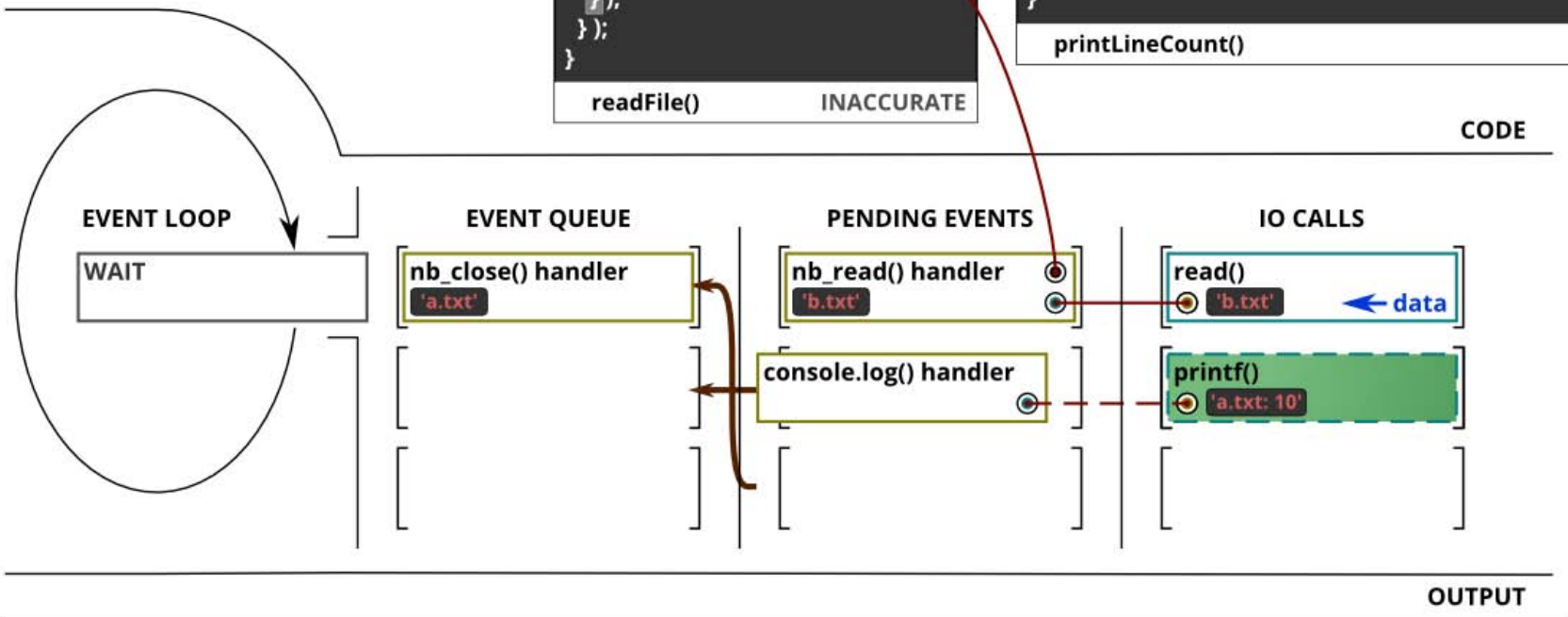
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
```

FILE READ OPERATION COMPLETES ('b.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

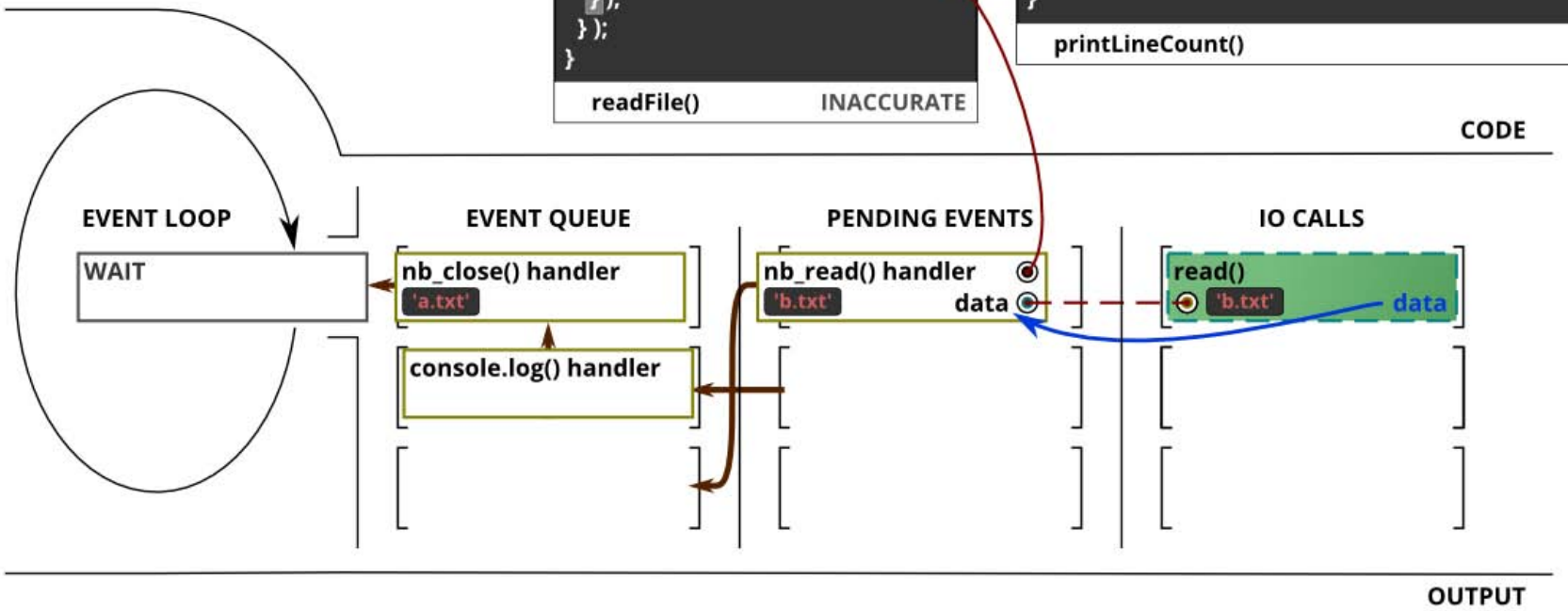
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
```

EXECUTING CLOSE HANDLER ('a.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

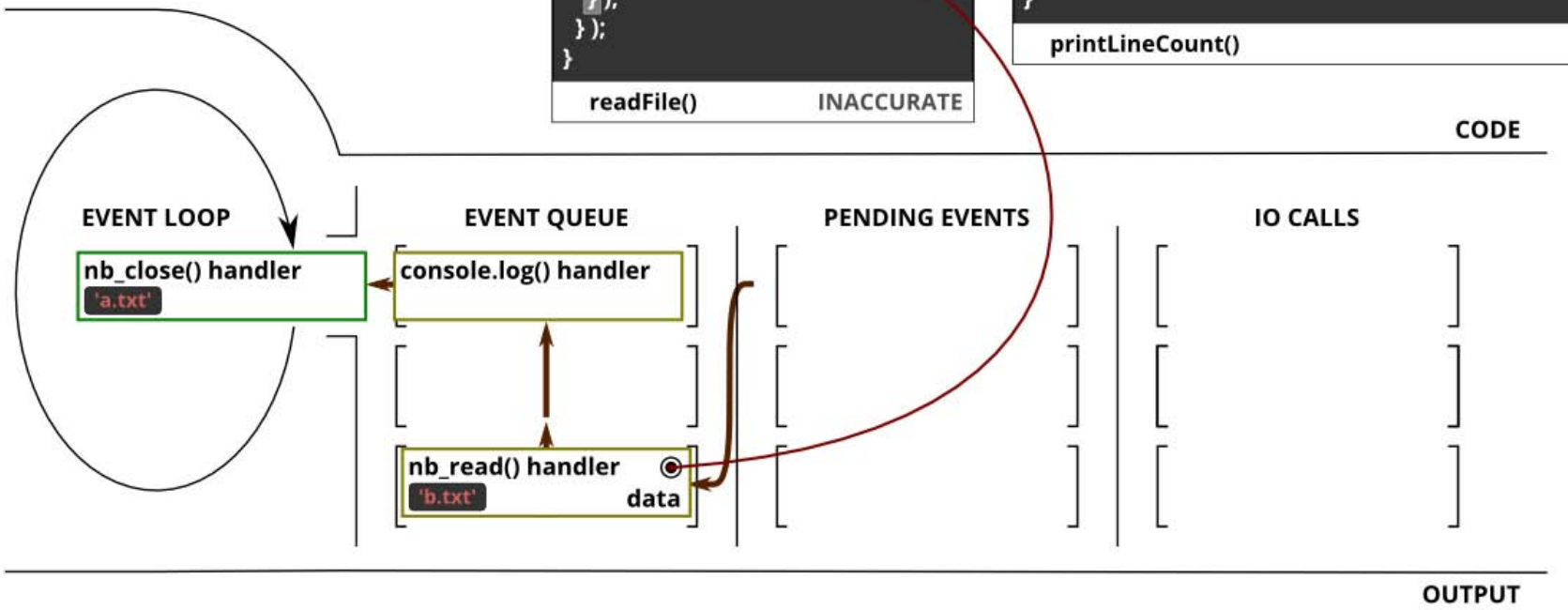
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

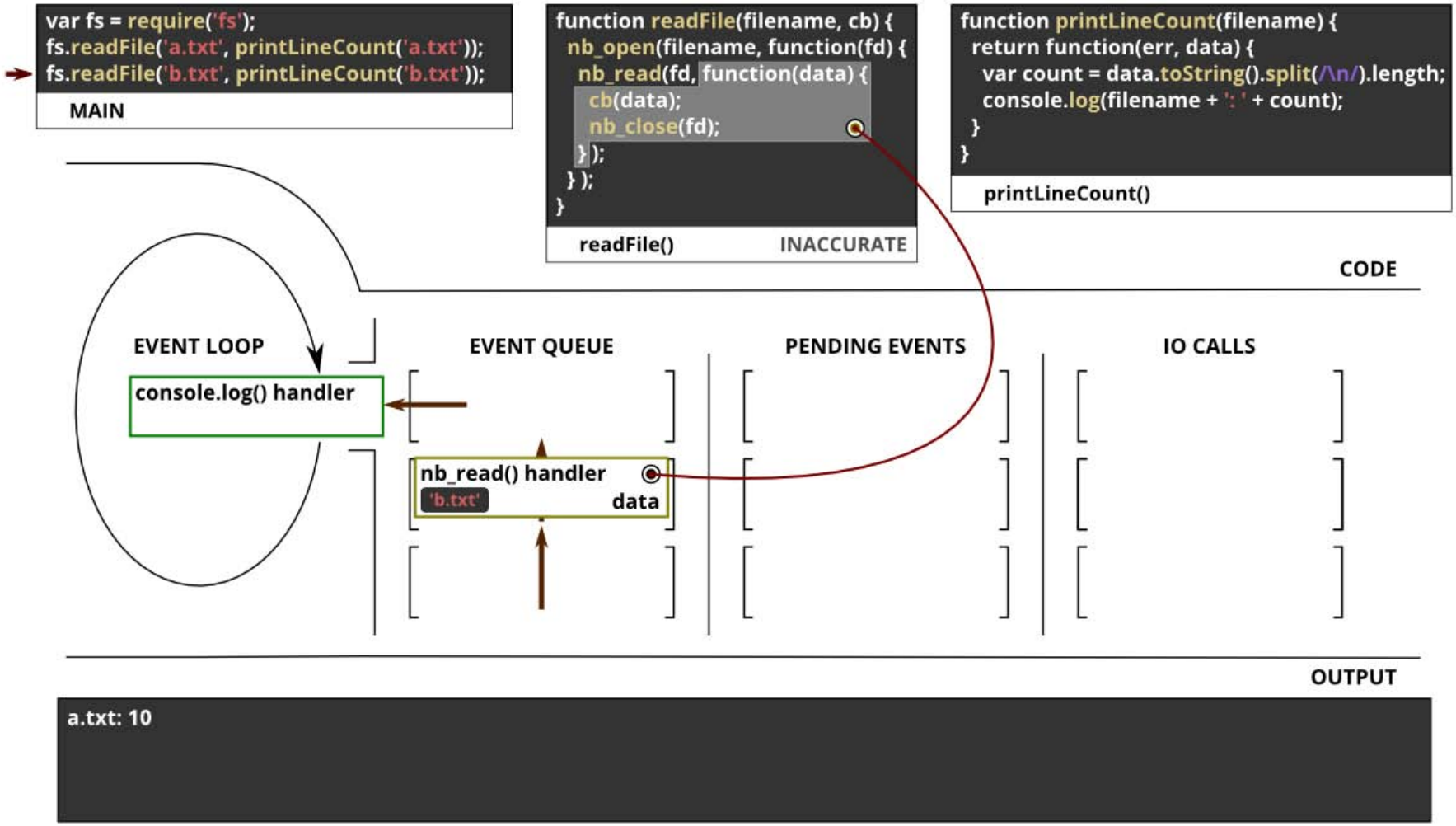
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



EXECUTING LOG HANDLER



EXECUTING LOG HANDLER

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

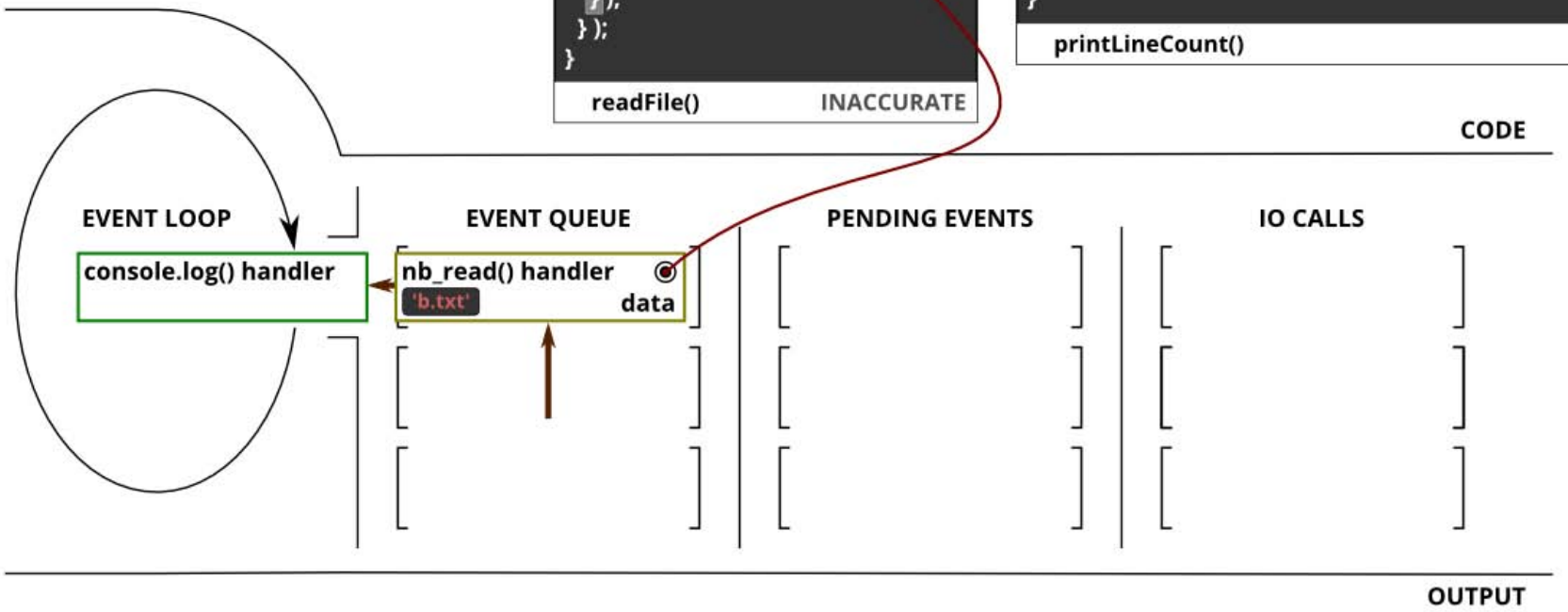
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()



EXECUTING READ HANDLER ('b.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

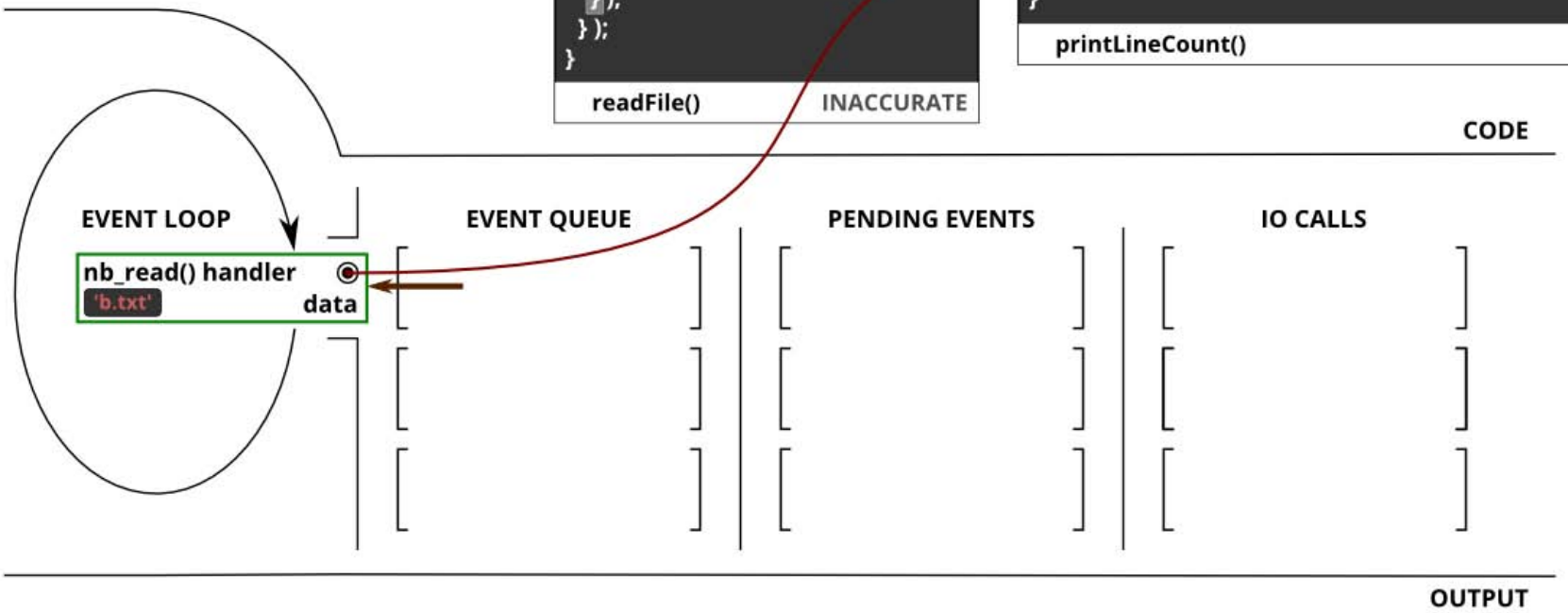
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    });
  });
}
```

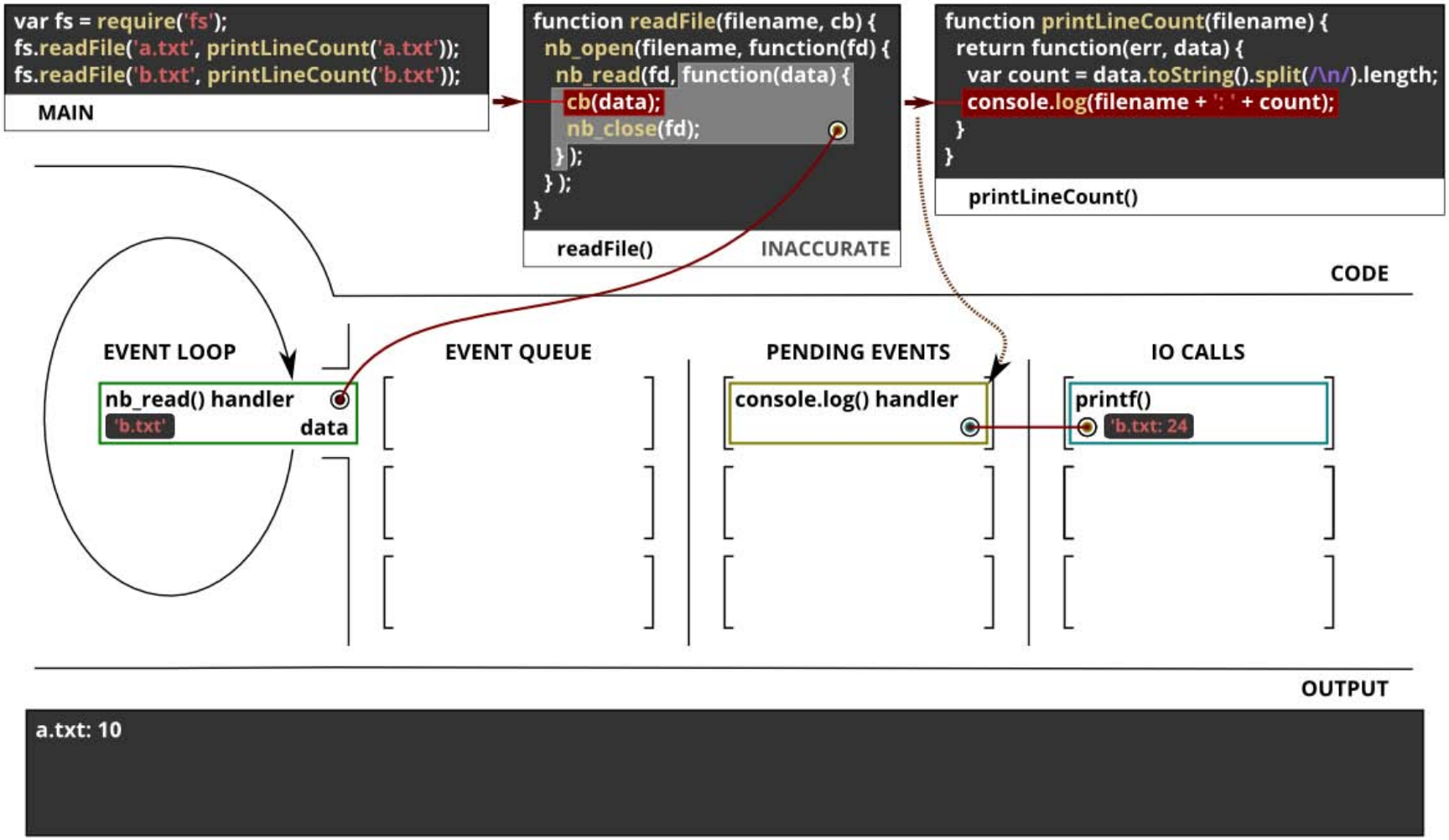
readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

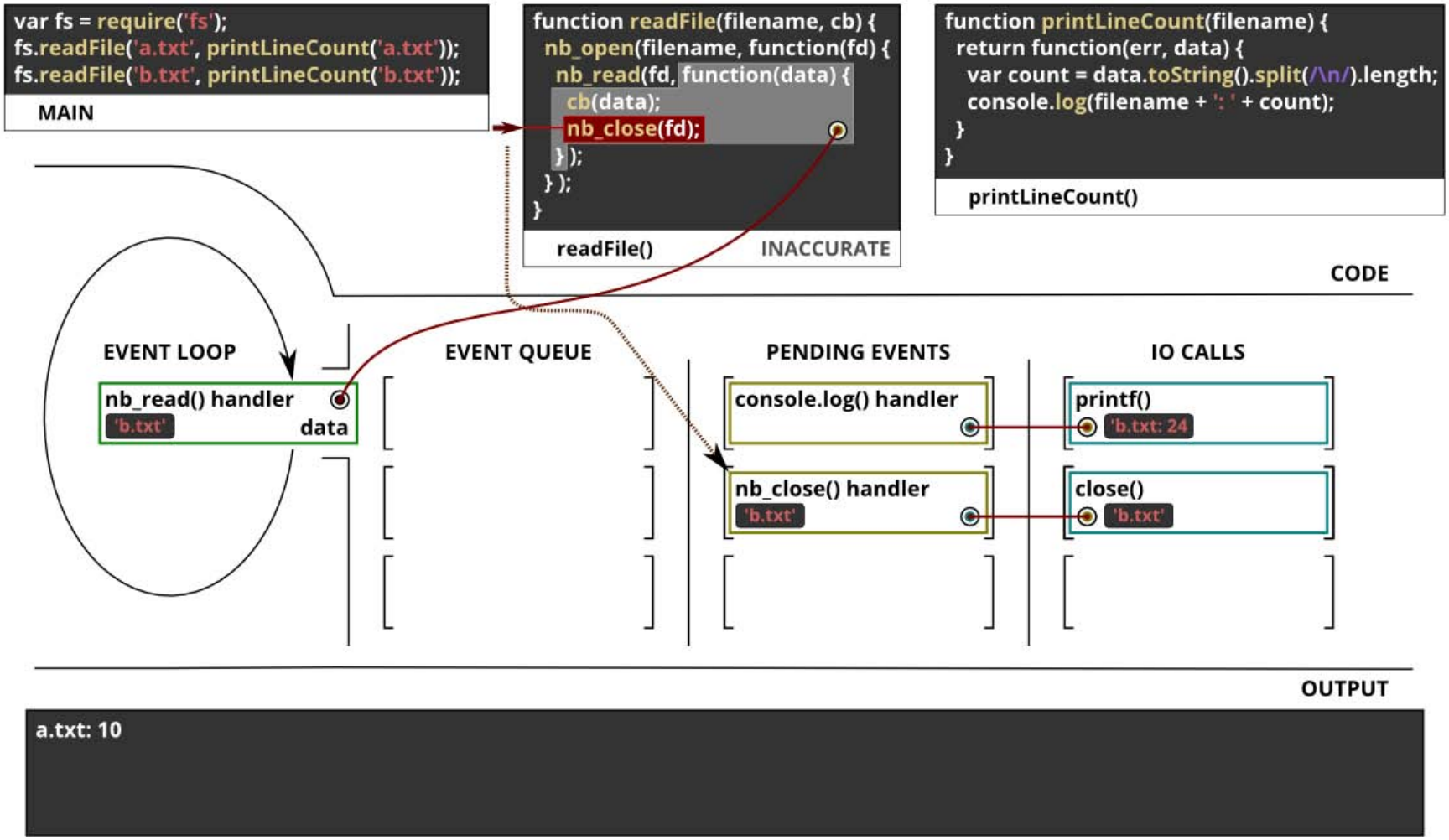
printLineCount()



INITIATING PRINT OPERATION ('b.txt: 24')



INITIATING FILE CLOSE OPERATION ('b.txt')



PRINT OPERATION COMPLETES ('b.txt: 24')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

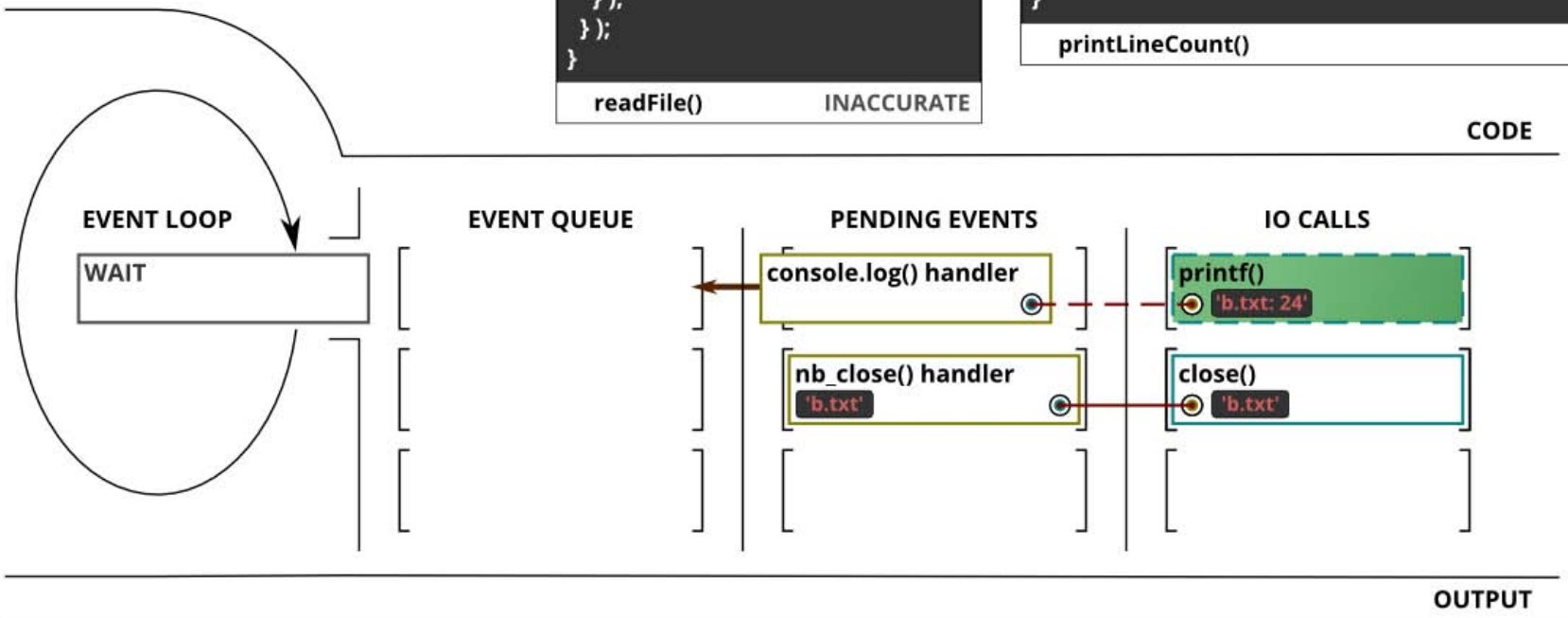
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

ADDING LOG HANDLER TO EVENT QUEUE

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

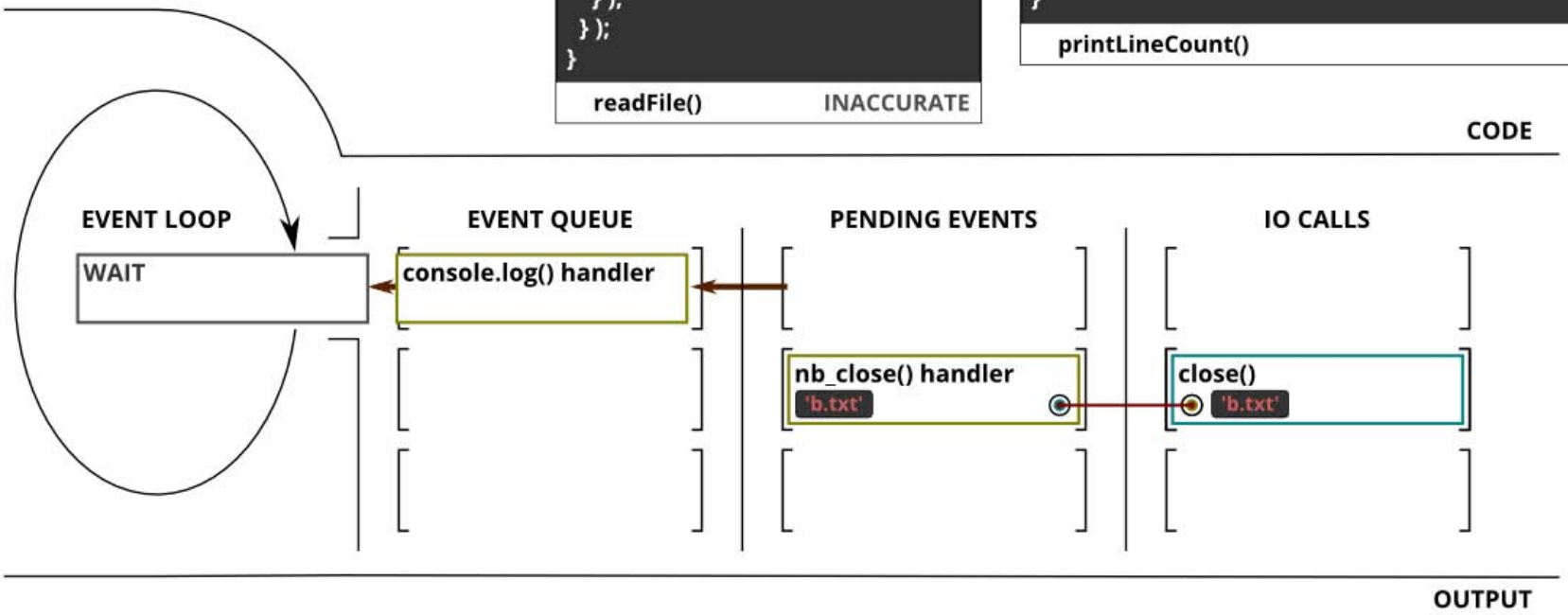
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

EXECUTING LOG HANDLER

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

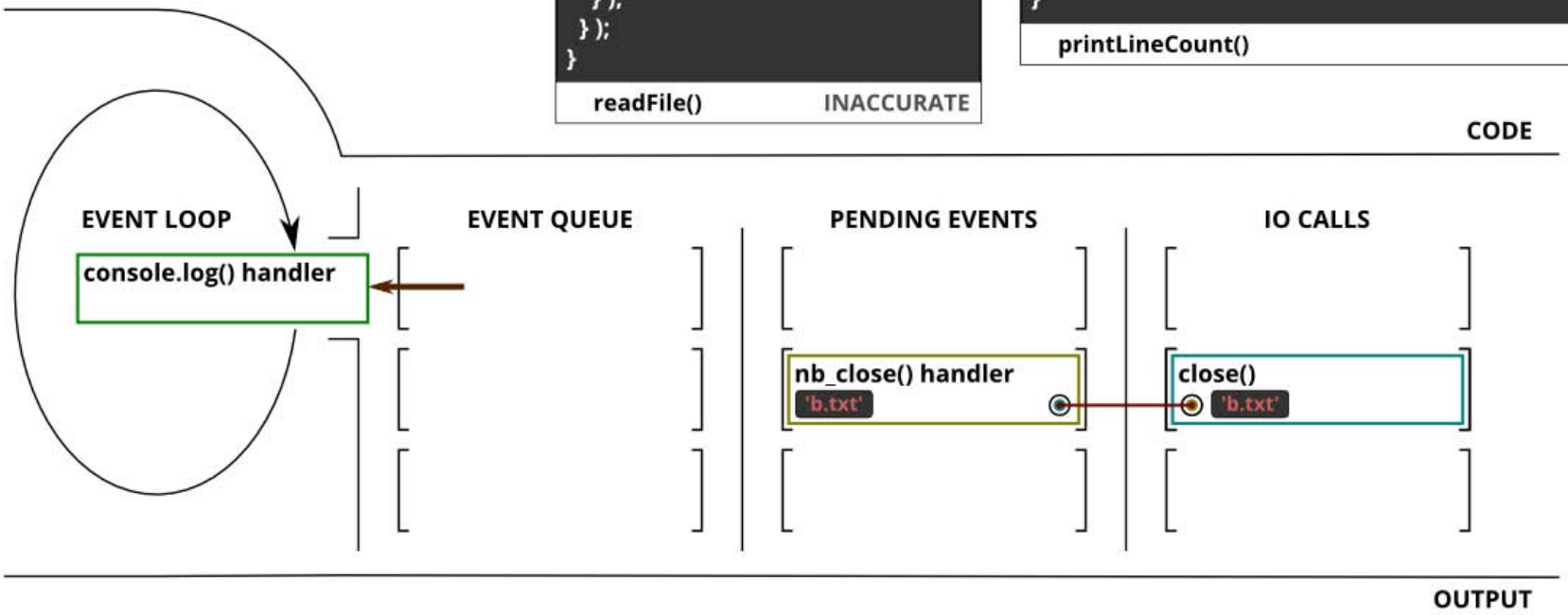
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

FILE CLOSE OPERATION COMPLETES ('b.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

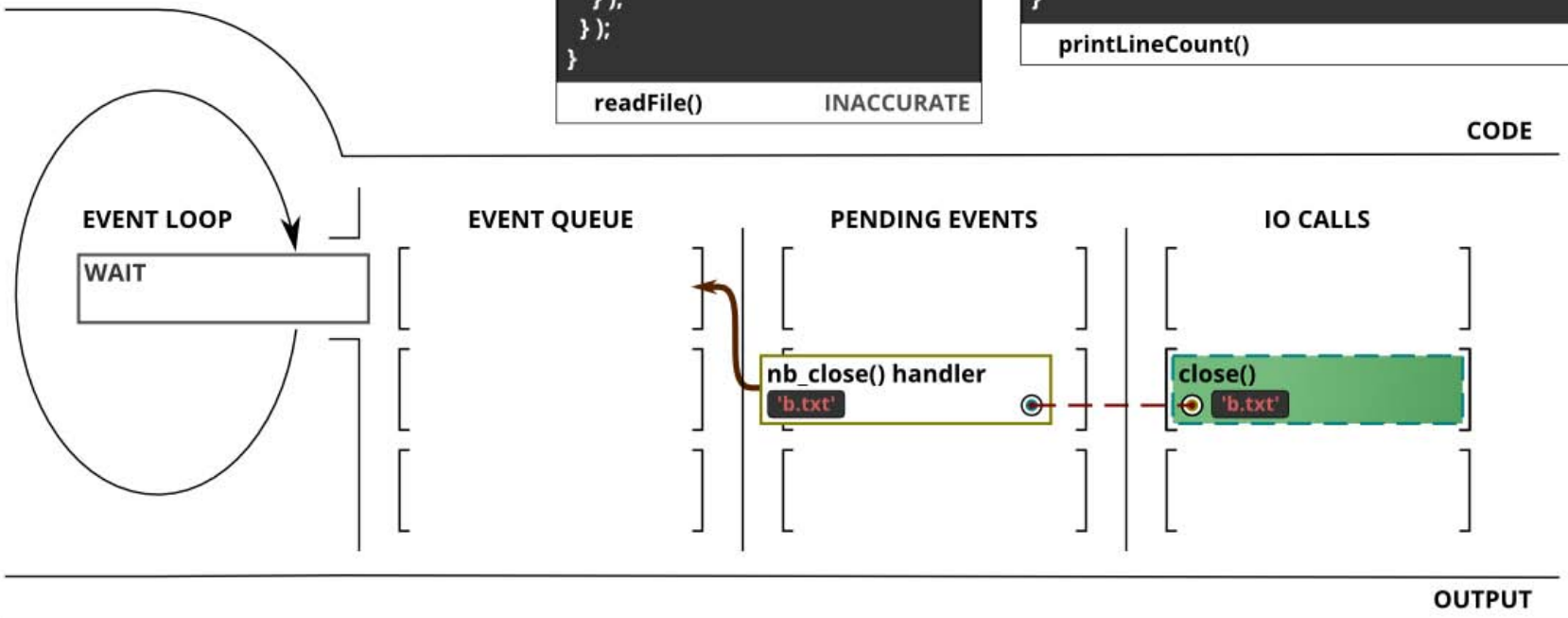
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

ADDING CLOSE HANDLER TO EVENT QUEUE ('b.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

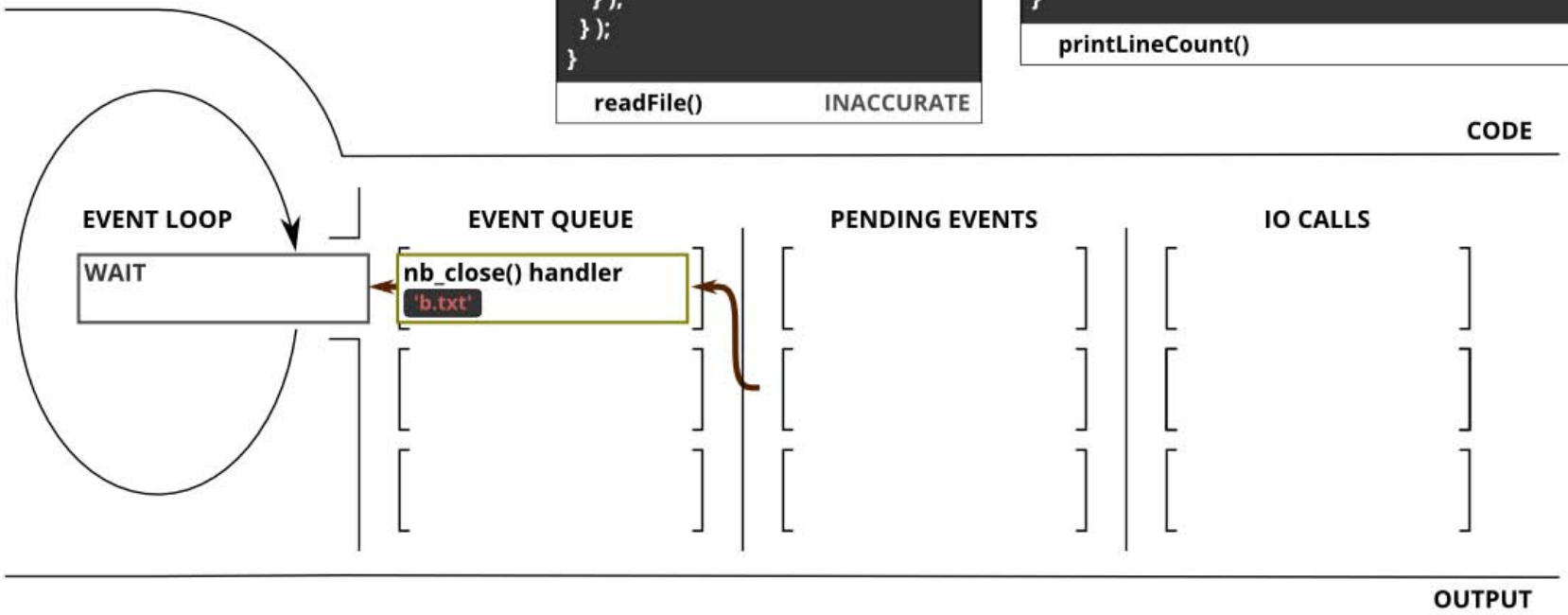
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

EXECUTING CLOSE HANDLER ('b.txt')

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

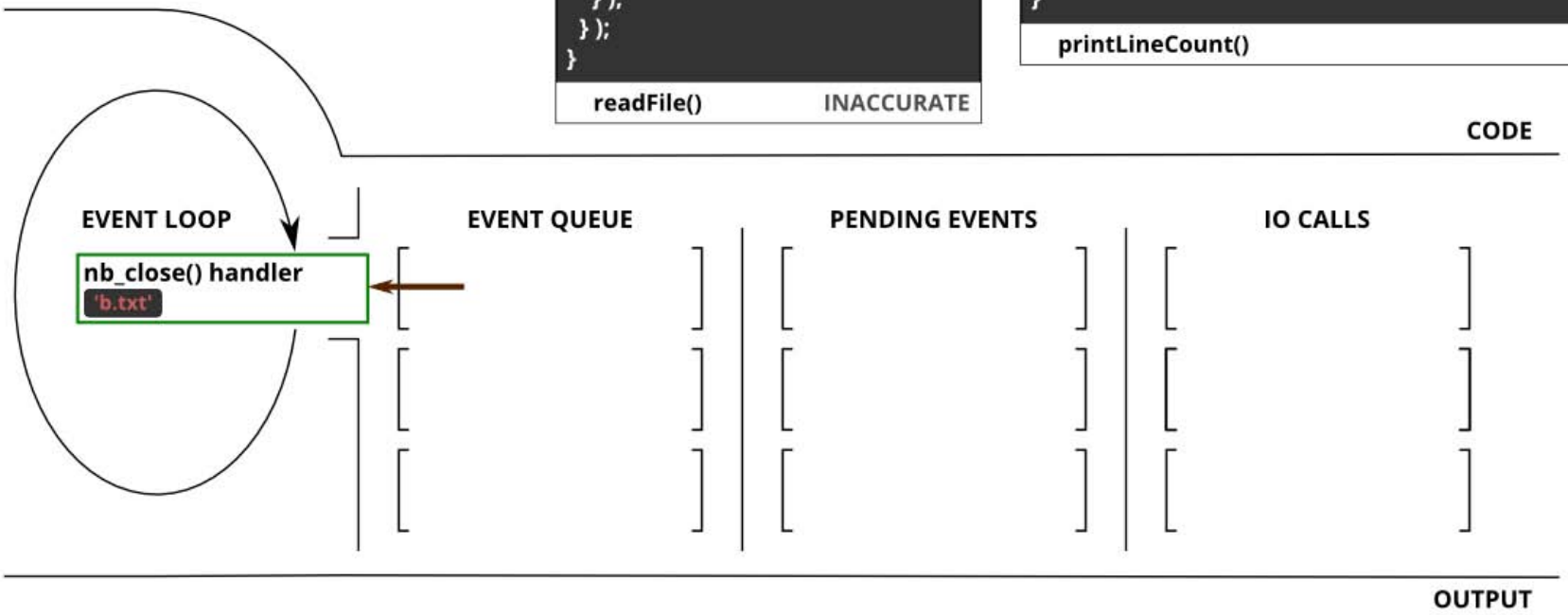
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

EXITING

```
var fs = require('fs');
fs.readFile('a.txt', printLineCount('a.txt'));
fs.readFile('b.txt', printLineCount('b.txt'));
```

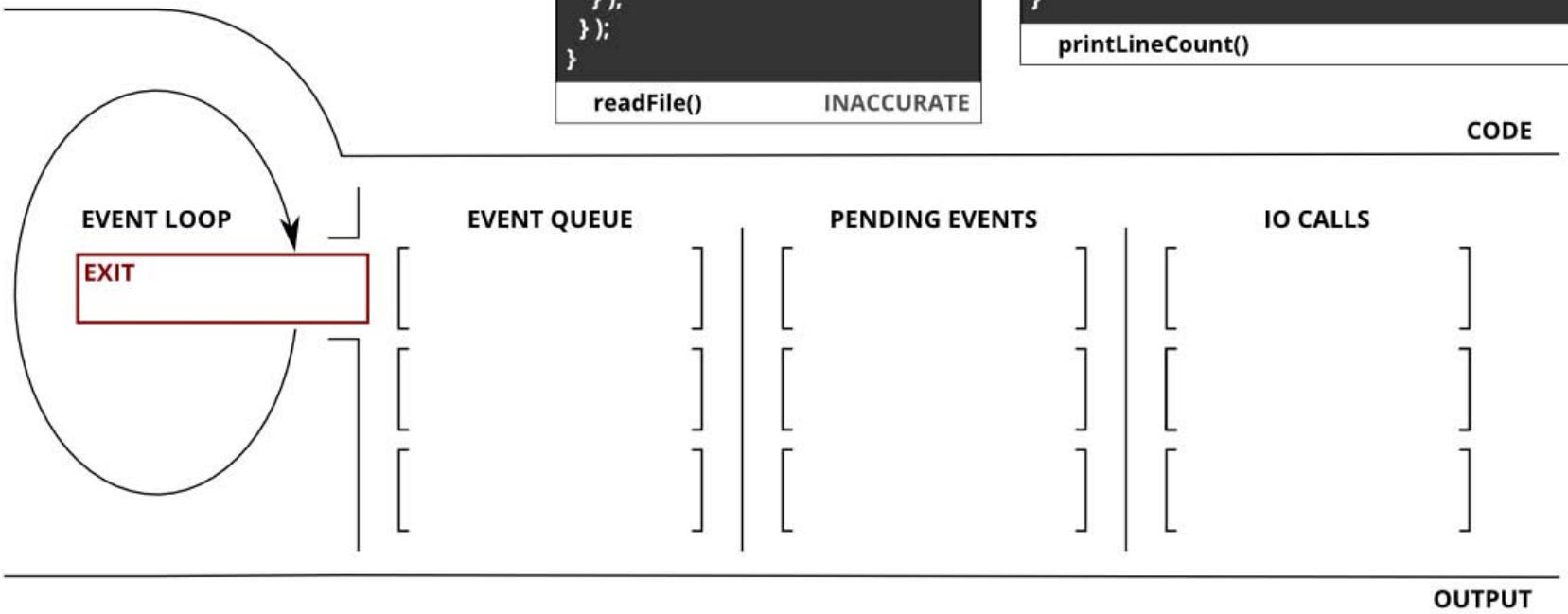
MAIN

```
function readFile(filename, cb) {
  nb_open(filename, function(fd) {
    nb_read(fd, function(data) {
      cb(data);
      nb_close(fd);
    } );
  } );
}
```

readFile() INACCURATE

```
function printLineCount(filename) {
  return function(err, data) {
    var count = data.toString().split(/\n/).length;
    console.log(filename + ': ' + count);
  }
}
```

printLineCount()




```
a.txt: 10
b.txt: 24
```

Questions?

Copyright and Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., and registered in many jurisdictions worldwide.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

A current list of IBM trademarks is available on the Web – see the IBM “Copyright and trademark information” page at URL: www.ibm.com/legal/copytrade.shtml