

o2

OPEN ORIENTED

凹凸实验室

node stream

陈嘉健

1

什么是流

2

可读流

3

可写流

4

管道



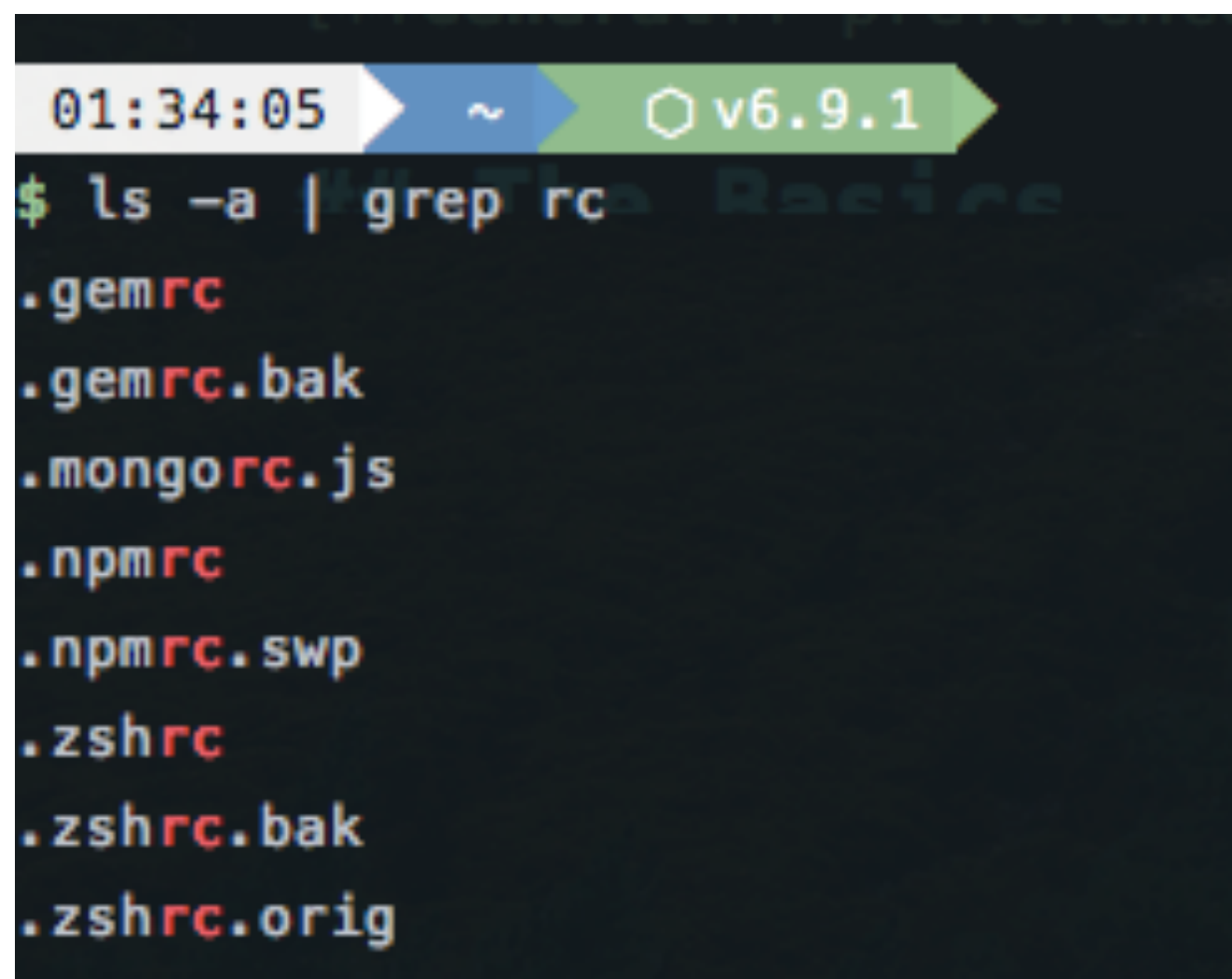
1

什么是流

“Make each program do one thing well.”

–Doug McIlroy

Unix 中通过管道（ Pipeline ） “|” 让一个程序的输出成为另一个程序的输入。

A terminal window with a dark background. The top status bar shows the time 01:34:05, a tilde symbol, and the version v6.9.1. The command prompt is \$, followed by the command 'ls -a | grep rc'. The output of the command is listed below: .gemrc, .gemrc.bak, .mongorc.js, .npmrc, .npmrc.swp, .zshrc, .zshrc.bak, and .zshrc.orig.

```
01:34:05 ~ v6.9.1
$ ls -a | grep rc
.gemrc
.gemrc.bak
.mongorc.js
.npmrc
.npmrc.swp
.zshrc
.zshrc.bak
.zshrc.orig
```

通过管道机制，数据在各个程序间流动，这就是抽象数据流的含义。

流 (stream) 在 Node.js 中是处理流数据的抽象接口。

stream 模块提供了四种基本流类型对象：

- Readable
- Writable
- Duplex
- Transform

通过继承相应的基本流类型对象，并补全底层数据处理接口，即可实现相应流类型的对象。

为什么要使用流

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

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

```
var server = http.createServer(function (req, res) {  
  fs.readFile(__dirname + '/data.txt', function (err, data) {  
    res.end(data);  
  });  
});  
  
server.listen(8000);
```

为什么要使用流

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

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

```
var server = http.createServer(function (req, res) {
```

```
    var stream = fs.createReadStream(__dirname + '/data.txt');
```

```
    stream.pipe(res);
```

```
});
```

```
server.listen(8000);
```


为什么要使用流

```
stream.pipe(oppressor(req)).pipe(res);
```

2

可读流

API :

- push
- read
- pipe
- unpipe
- pause
- resume
- unshift

```
var Readable = require('stream').Readable;
```

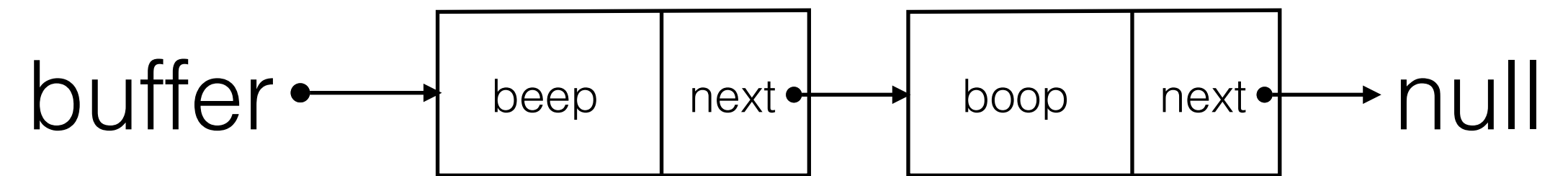
```
var rs = new Readable;
```

```
rs.push('beep ');
```

```
rs.push('boop\n');
```

```
rs.push(null);
```

```
rs.pipe(process.stdout);
```



```
var Readable = require('stream').Readable;
```

```
var rs = Readable();
```

```
var c = 97;
```

```
rs._read = function () {
```

```
    rs.push(String.fromCharCode(c++));
```

```
    if (c > 'z'.charCodeAt(0)) rs.push(null);
```

```
};
```

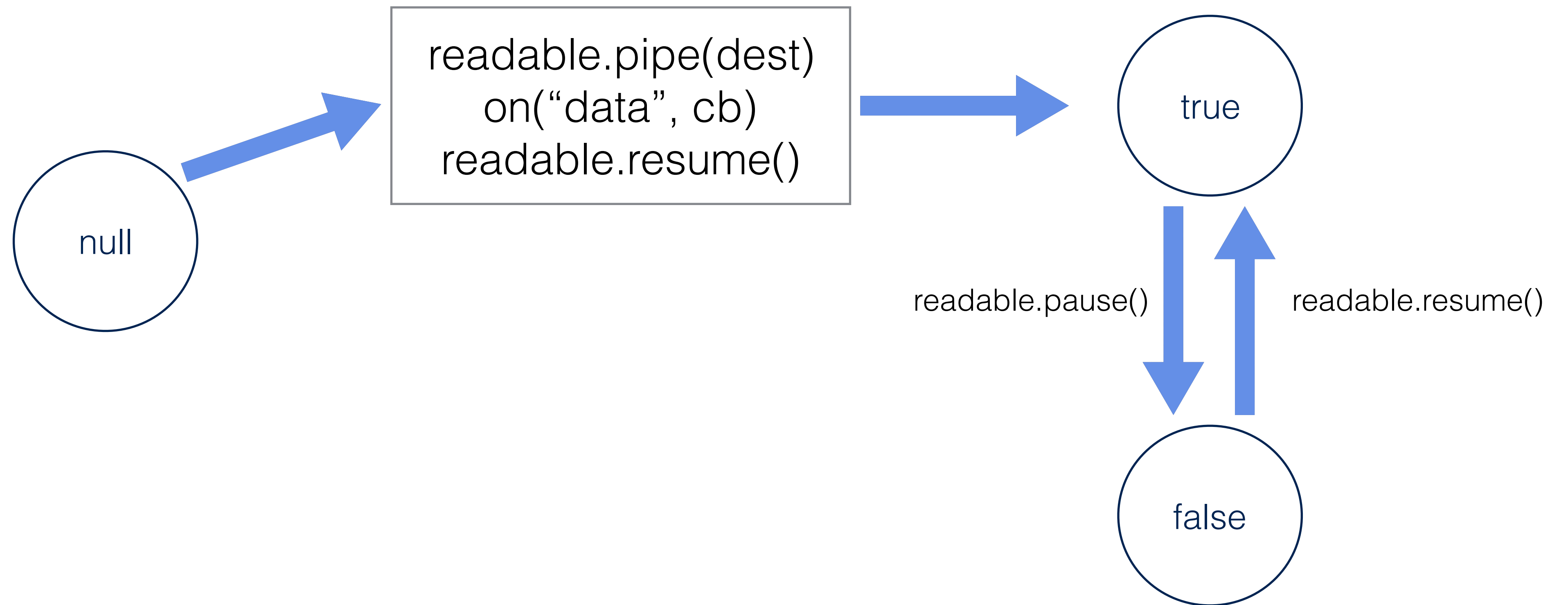
```
rs.pipe(process.stdout);
```

三种状态

`readable._readableState.flowing`

- `null`
- `true`
- `false`

`readable._readableState.flowing`



两种模式

- pause

默认模式。在该模式下，需要手动调用`stream.read()`来获取数据。

- flowing

在该模式下，会尽快获取数据向外输出。因此如果没有事件监听，也没有`pipe()`来引导数据流向，数据可能会丢失。

readable.resume

```
while (state.flowing && stream.read() !== null);
```

3

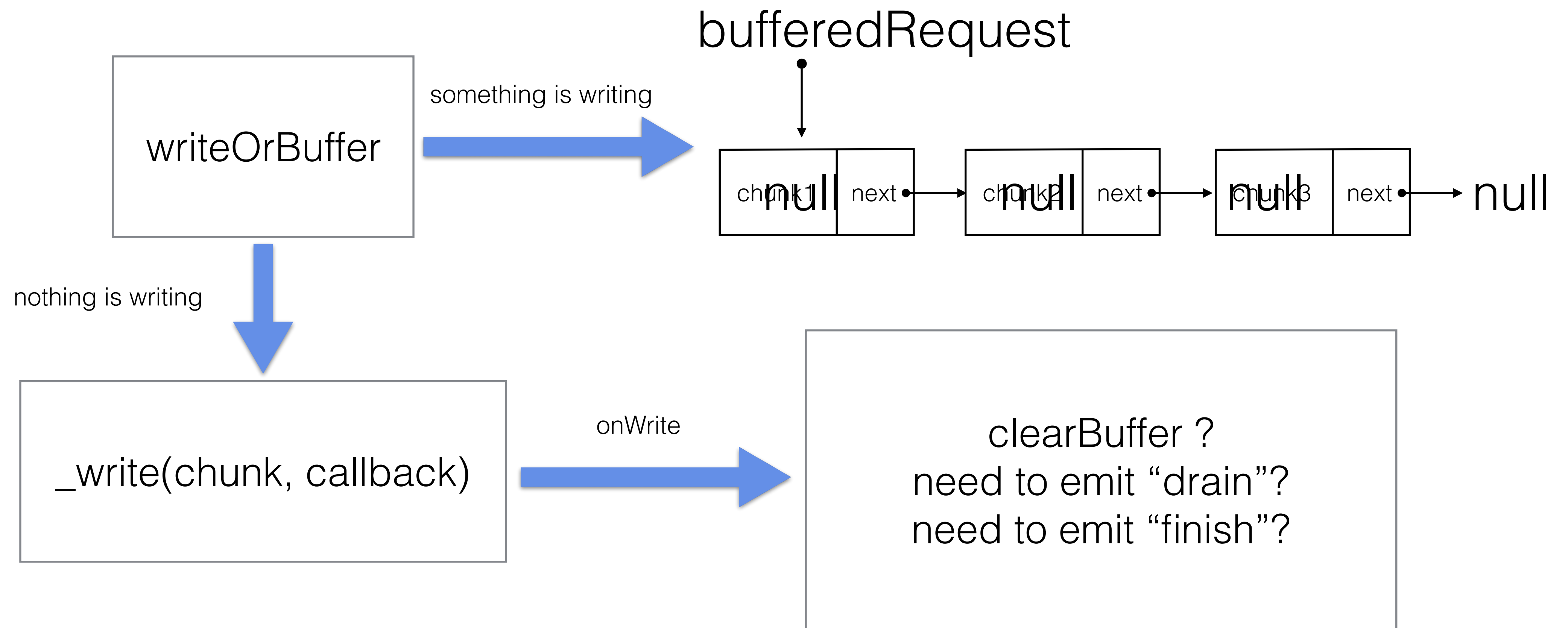
可
写
流

API :

- write
- setDefaultEncoding
- end
- cork
- uncork

writable.write

`write(chunk[, encoding][, callback])`



背压 backpressure

想象把一根水管的尾部弯曲，源头会感受到压力，这就是所谓的背压。

当

缓存里 $\text{chunk size} + \text{chunk size} > \text{highWaterMark}$ 时，`write` 函数会返回 `false`。

排水事件：“`drain`”，缓存里的数据都已经全部写入。



cork & uncork

- cork 是木塞的意思。writable.cork() 这个方法的作用是强行使 chunk 进行缓存，不会调用底层的写入方法 _write。
- 而 writable.uncork() 方法则会调用 clearBuffer 方法对缓存里的数据进行写入。

在向流中写入大量小块数据时，内部缓存可能失效，从而导致性能下降。writable.cork() 方法主要就是用来避免这种情况。

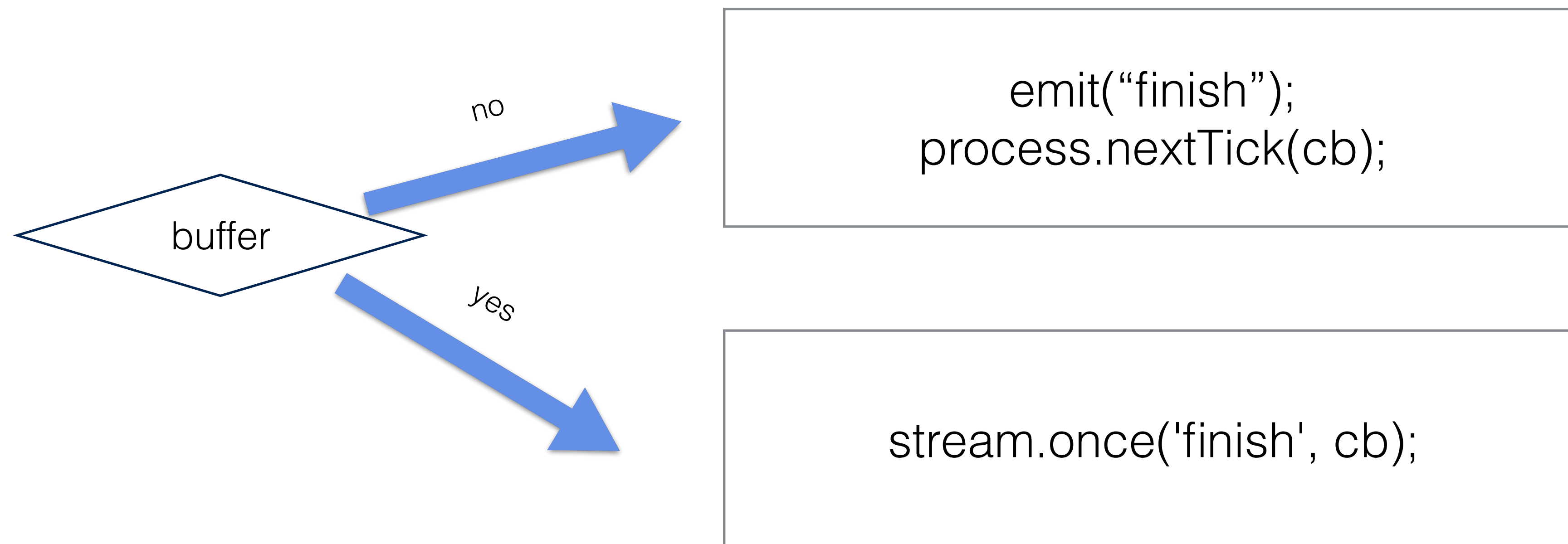


writable.end

`end(chunk[, encoding][, callback])`

可以传入一段chunk作为关闭流前最后一次写入。

若之前调用了cork方法，则会调用uncork方法，将缓存里的数据全部写入。



4

管道

- `src.on("data" , () => dest.write(data))`
- `src.on("end" , () => dest.end())`
- `dest.on("drain" , () => src.resume())`
- `dest.on("unpipe" , cleanUp)`
- `dest.on("error" , unpipe)`
- `dest.on("close" , unpipe)`
- `dest.on("finish" , unpipe)`

`src.resume()`

THANKS
FOR YOUR WATCHING



OPEN ORIENTED

凹凸实验室