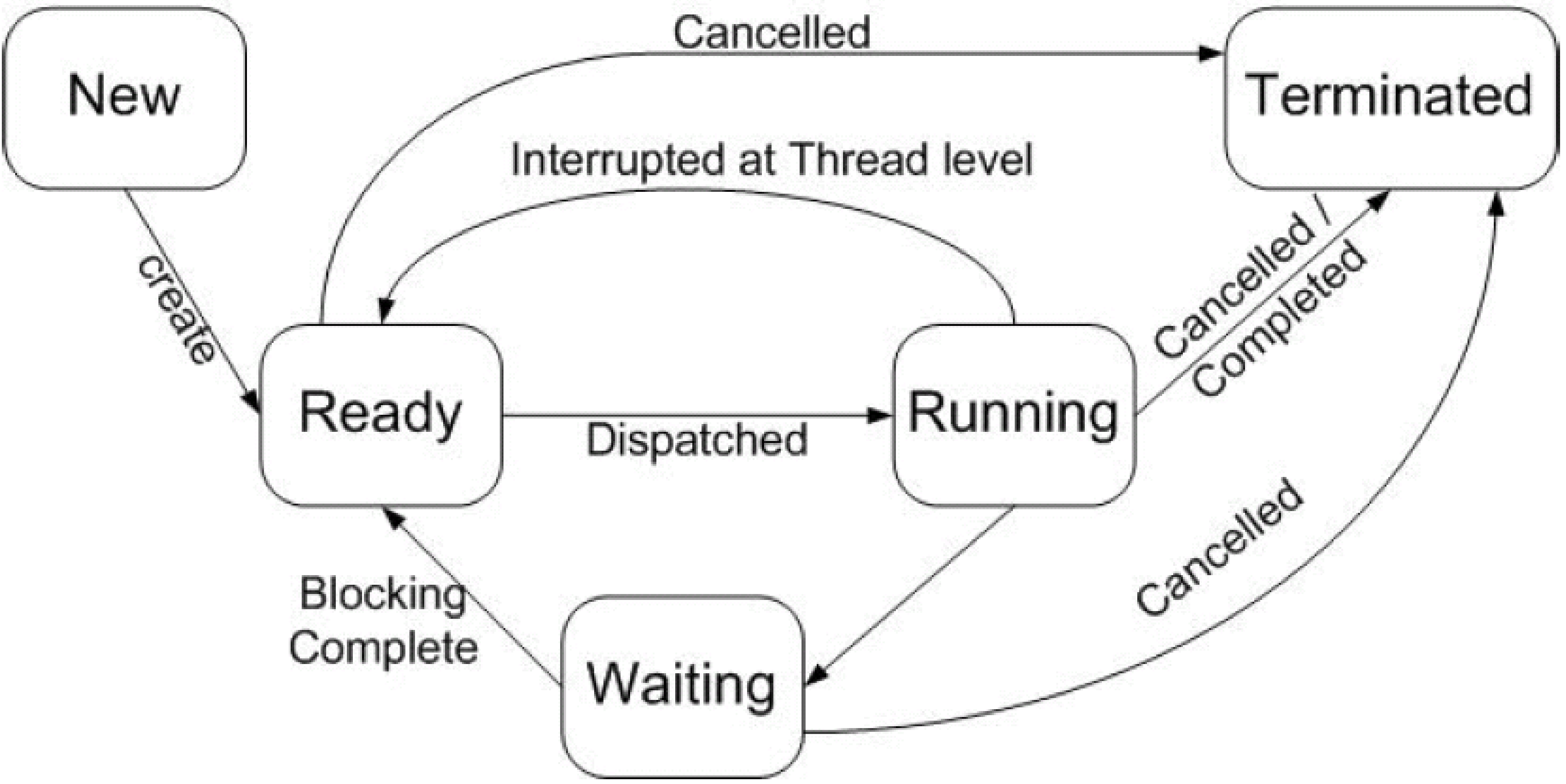
Thread States



NodeJS

NodeJS

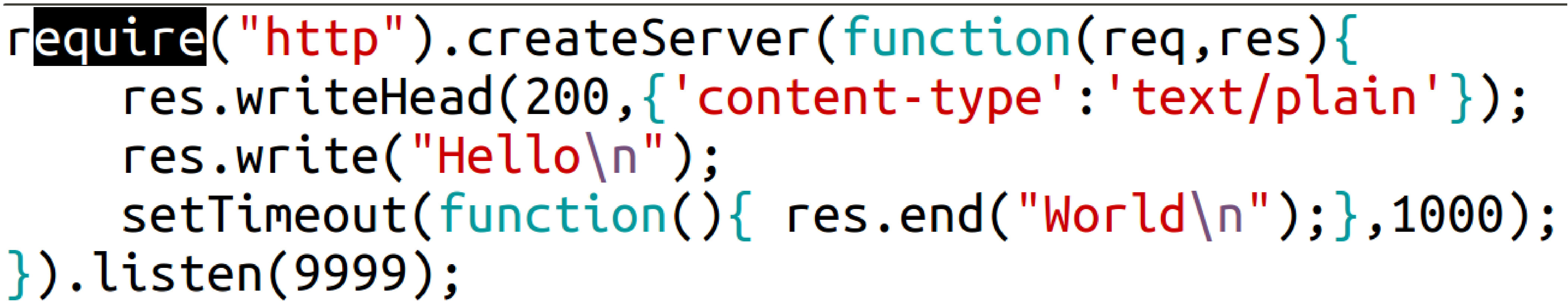
- Event Driven, Non Blocking IO.
- Lightweight
- Uses Google's Javascript V8 Engine
- perfect for data-intensive real-time applications that run across distributed devices.

NodeJS in The Industry

Y!
in
Link
Microsof
The New York Times
ebay
9
Cloud9 IDE
code anywhere, anytime

HTTP Server

- Using NodeJS



```
require("http").createServer(function(req,res){
    res.writeHead(200,{'content-type':'text/plain'});
    res.write("Hello\n");
    setTimeout(function(){ res.end("World\n");},1000);
}).listen(9999);
```

- Using Unix C API
- Benchmark

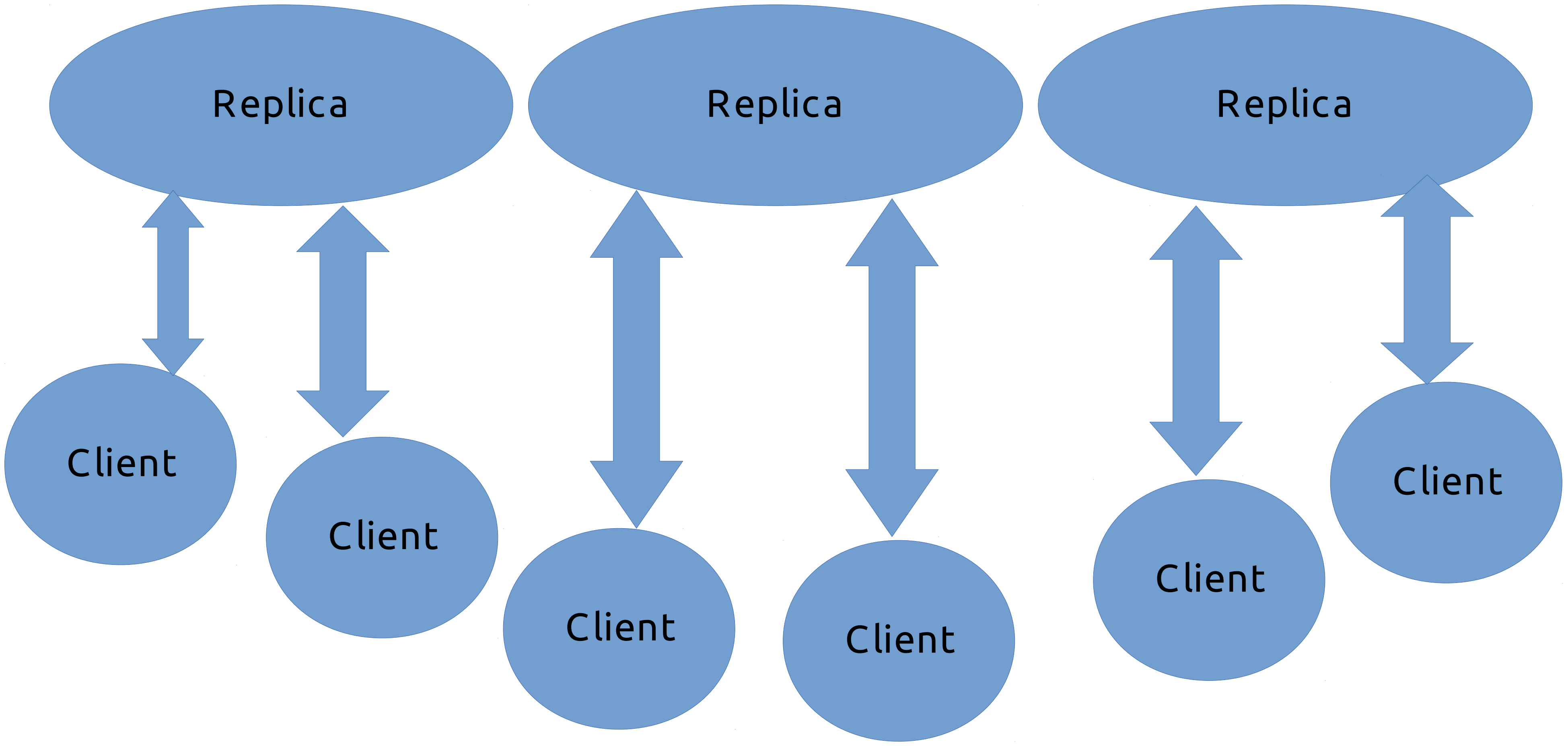
TCP Server

```
require("net").createServer(function(socket){
   socket.write("Hello\n");
   socket.end("world\n");
}).listen(9999);
```

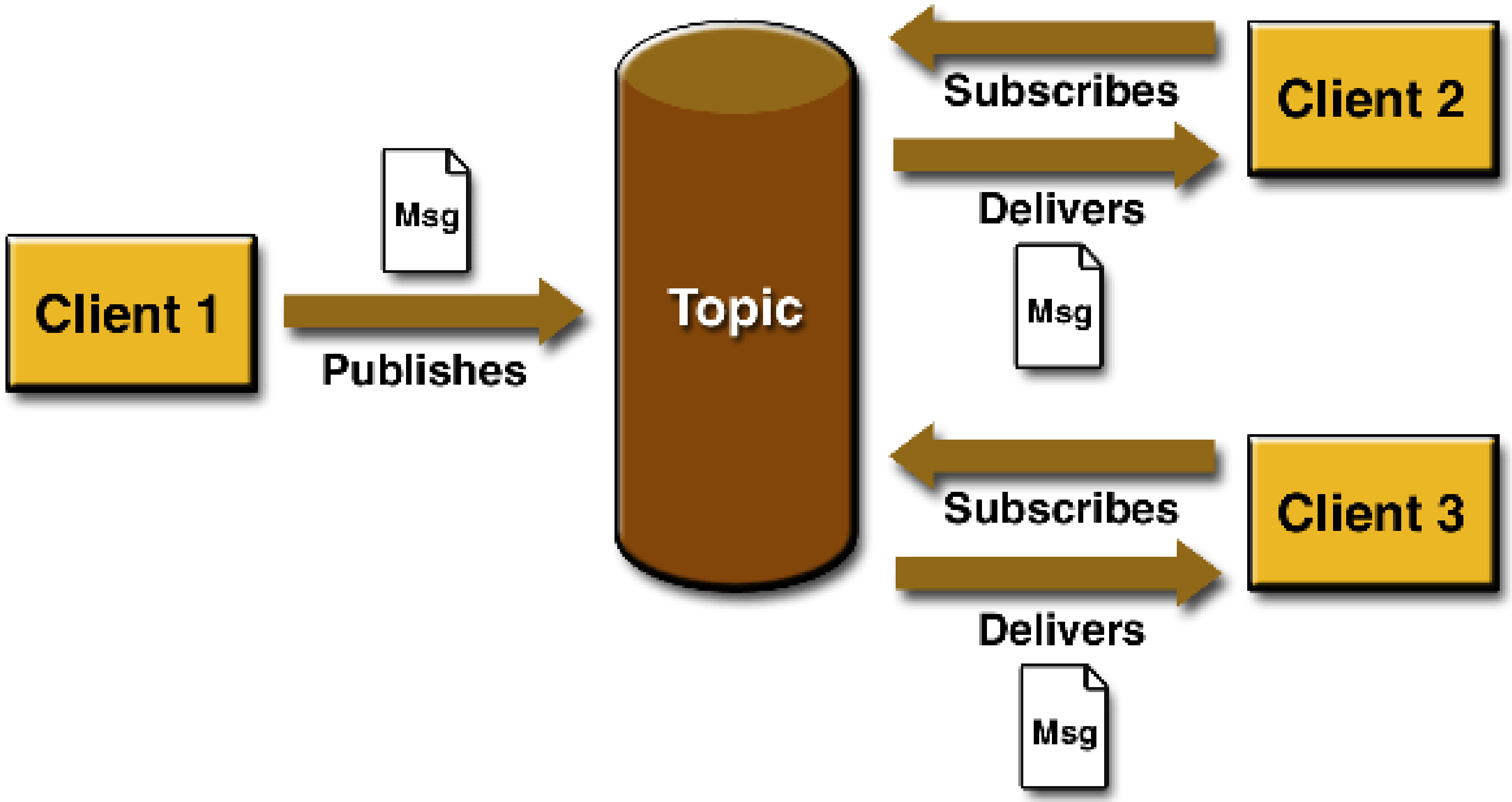
Single Node Chat Server

```
var sockets = [];
require("net").createServer(function(socket){
   sockets.push(socket);
   console.log(socket);
   socket.write("welcome to node's tcp chat server\n");
   socket.on('data',function(data){
       for(var i=0;i<sockets.length;i++){
           if(sockets[i] == socket) continue;
           sockets[i].write(data);
       }
   });
//   socket.on('close',function(){
//       sockets.splice(sockets.indexOf(socket),1);
//   });
}).listen(9999);
```

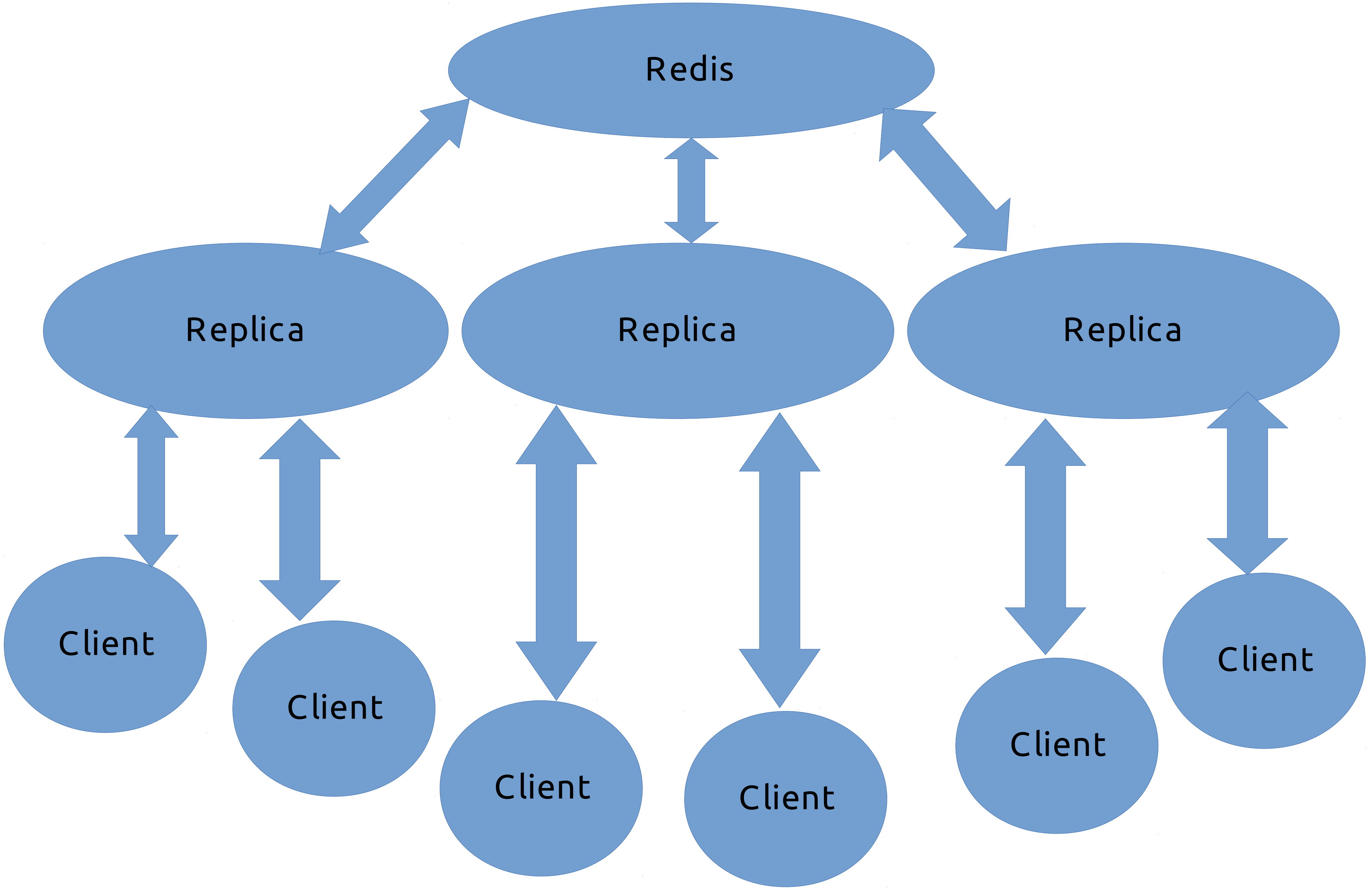
Replication



Publisher Subscriber



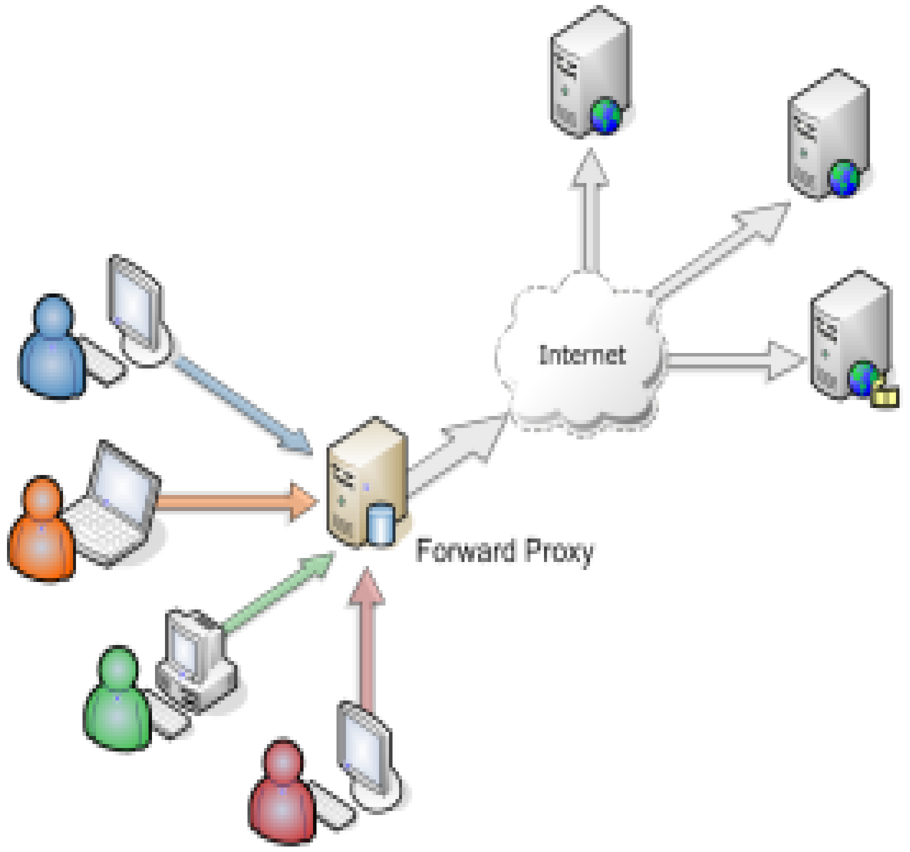
Message Queuing

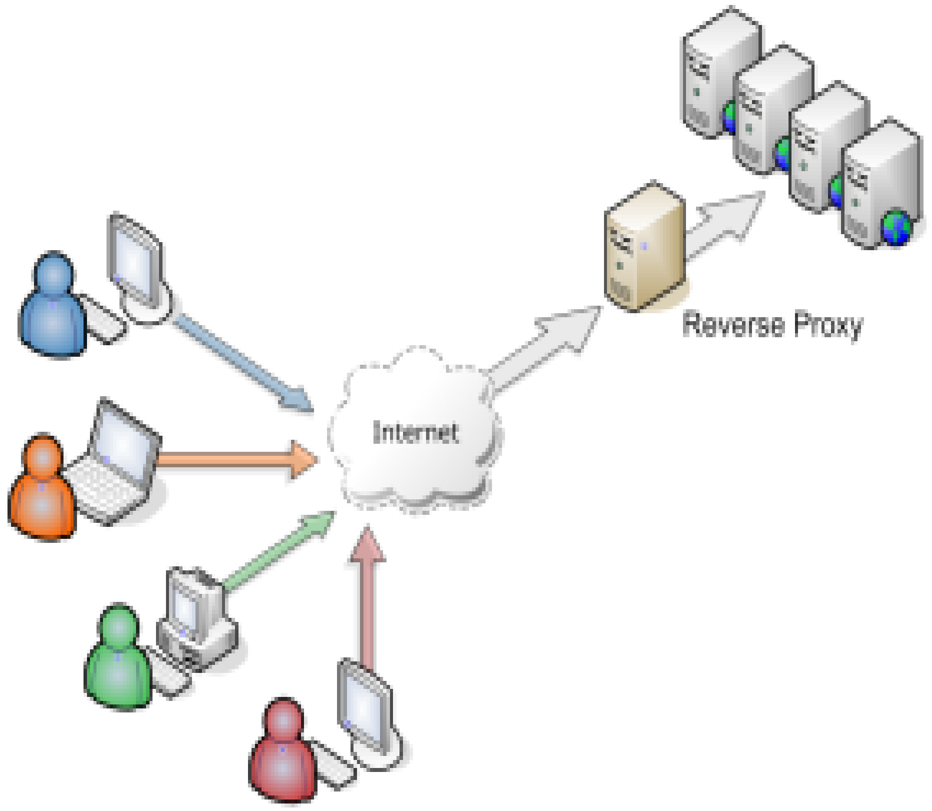


Distributed Chat System

```
var sockets = [];
var optimist = require("optimist");
var redis = require("redis");
var subClient = redis.createClient();
var globalIncr = 0;
subClient.subscribe("chat_channel");
subClient.on("message", function(channel, data) {
   data = JSON.parse(data);
   for(var i=0;i<sockets.length;i++){
       if(sockets[i].userId === data.userId) continue;
       sockets[i].write(data.message);
   }
});
require("net").createServer(function(socket){
   var pubClient = redis.createClient();
   socket.userId = process.pid.toString()+(globalIncr++)+Math.floor(Math.random()*100000);
   sockets.push(socket);
   socket.write(process.pid + " : welcome to node's tcp chat server\n");
   socket.on('data',function(data){
     var stringifiedMessage = JSON.stringify({"message":data.toString("utf8"),"userId":socket.userId});
     pubClient.publish("chat_channel",stringifiedMessage);
   });
   socket.on('close',function(){
       sockets.splice(sockets.indexOf(socket),1);
   });
}).listen(optimist.argv.port);
```

Reverse Vs Forward Proxy





NodeJS Reverse Proxy

```
var net = require("net");
var hosts = [
    { port:9999, host:'127.0.0.1'},
    { port:9998, host:'127.0.0.1'},
    { port:9997, host:'127.0.0.1'},
    { port:9996, host:'127.0.0.1'},
    { port:9995, host:'127.0.0.1'},
    { port:9994, host:'127.0.0.1'},
    { port:9993, host:'127.0.0.1'},
    { port:9992, host:'127.0.0.1'}
];
var counter = 0;
net.createServer(function(socket){
    counter = (counter+1)%hosts.length;
    console.log("forwarding connection to : ", hosts[counter]);
    var hostSocket = net.connect(hosts[counter],function(){
        socket.pipe(hostSocket).pipe(socket);
    });
    socket.on('end',function(){
        console.log("destroying socket");
        hostSocket.destroy();
    });
}).listen(8000);
```