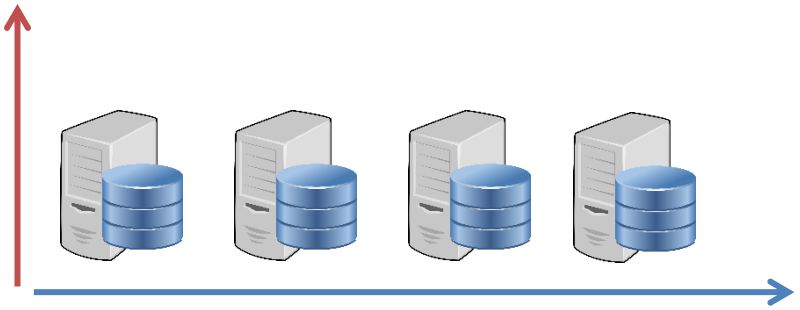
Scalability

- Scalability
- Horizontal vs. vertical scaling
- Scaling node.js

Scalability?

Vertical scaling makes your machine(s) bigger and stronger.
Think more cores, RAM.



Horizontal scaling adds more machines.
Think of them standing side-by-side.

- **Vertical scaling** is easy, rent more cores / RAM
  - *Instant scalability, but cores may sit idle most of the time (wasting $)*
- **Horizontal scaling** supports any # of users
  - *May take a few minutes for machines to startup (some users wait)*

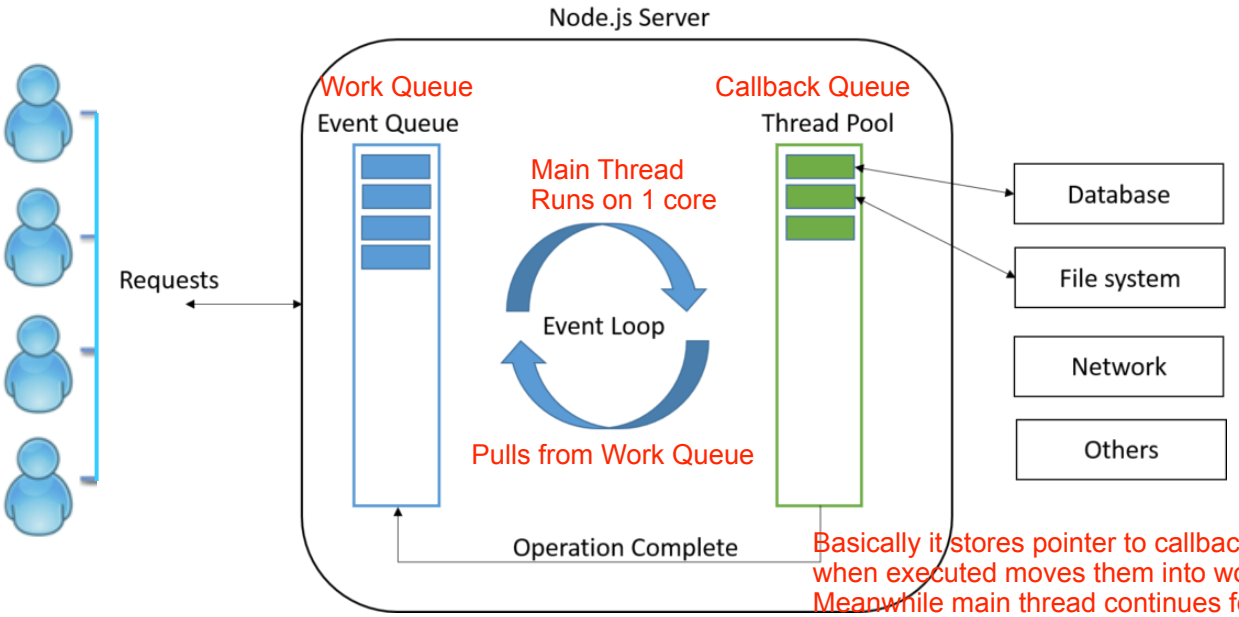
It is possible to rent out many machines from the beginning and wait for the load to scale up. But this is cost wise un-effective. Instead, start with 1 or 2 servers, as the load increases, then scale up the resources on the go, but yes this has one disadvantage of being slow as machines can take a while to startup

Scalability of node.js

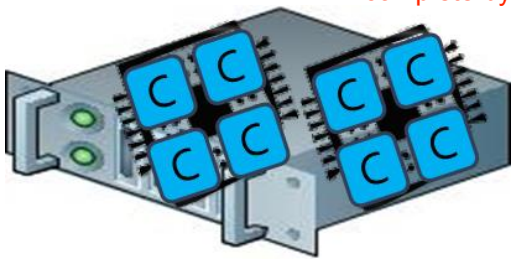
- **Node.js is single-threaded…**
  - *Even so, it can support 10-100 clients*
- **Netflix uses Node.js to serve 1M+ users**
- **How?**
  - *Let's talk about various strategies…*
  - *You can read about Netflix software architecture here…*

node.js

- **Single-threaded with large software library (MySQL, S3, …)**
  - *https://nodejs.org/api/documentation.html*



Basically it stores pointer to callbacks, when executed moves them into work Queue. Meanwhile main thread continues fetching tasks from work queue, executing things and putting things into callback Queues, while it waits for these tasks to complete by other systems like Database, S3, etc

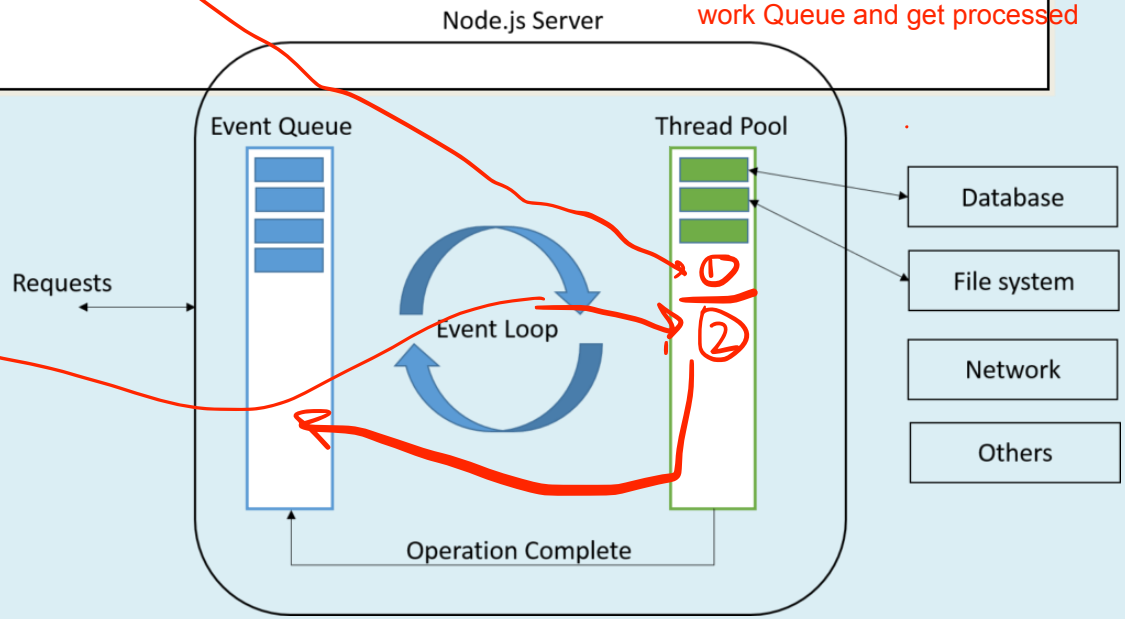


Example

```
app.get('/stats', (req, res) => {

  s3_response = s3.send("HeadBucketCommand"));

  db.query(`Select count(*) from users;
            Select count(*) from assets;`, async (err, results, _) => {

    if (err) {
      res.status(400).json(…);
      return;
    }

    var s3_result = await s3_response;  // wait for it…
    .
    .  // we have all the results, extract values and respond:
    .
    res.json(…);
  });
});
```

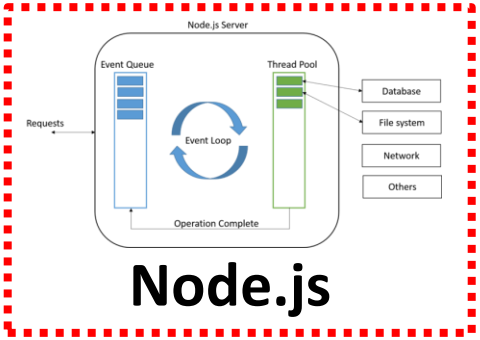
- Whenever stats command is triggered, node.js starts executing from the top. It will fetch (1) from the work queue,
sends command to s3 and move (1) to callback queue.
- Then it gives a call to DB with Query (2) and meanwhile it will fetch the data from DB, it moves (2) to callback Queue.
- After this main thread exits, and start processing other tasks from the work Queue.
- When (1) finishes, it gets moved back to work Queue, as it pops up the work Queue, main thread again start executing it.

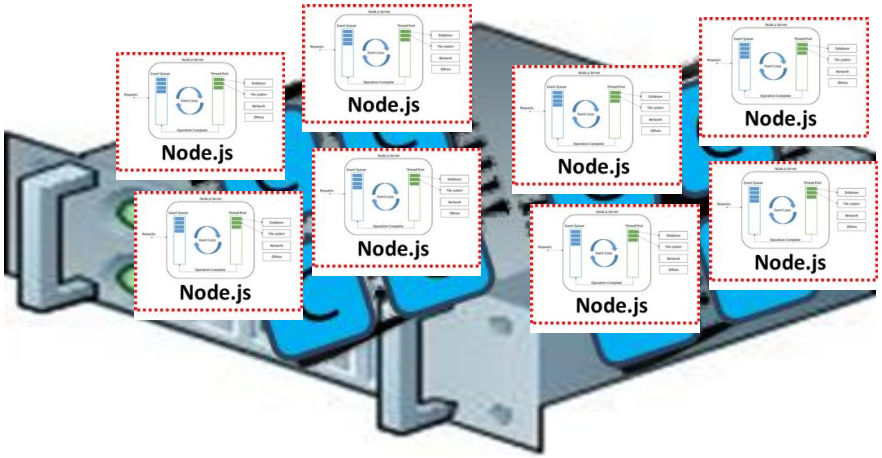
- When it sees the await, it itself converts the code after this into a callback (3). When (2) finishes, it moves back to work Queue, gets processed. This will trigger 3 being moved to work Queue and get processed



Vertical scaling of Node.js

- **Replicate node.js across the cores…**

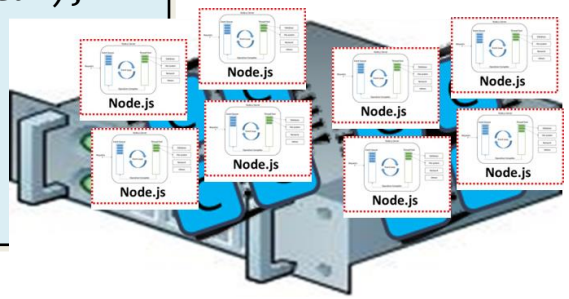




node.js cluster module

- **Cluster** offers scale-up functionality (ref, ref)
- **PM2** is another good option (ref)

```
const cluster = require("cluster");
const totalCPUs = require("os").cpus().length;

var app;

if (cluster.isMaster) {
  console.log(`Main process ${process.pid} has started`);
  console.log(`Number of CPUs is ${totalCPUs}`);

  // replicate across the cores:
  for (let i = 0; i < totalCPUs; i++) {
    cluster.fork();
  }
}
else {
  console.log(`Working process ${process.pid} started`);
  app = express();
  .
  .  // app.get() handlers
  .
}
```



Vertical Scaling pros and cons

✓Easy to write your programs to run on a single machine.

✓Most languages support multi threads / processes

✓Most "off the shelf" software is written to run on one machine.

Eg.: MySQL, node.js, etc

✓Modern servers can do a lot of work in parallel with ~96 cores.

✕ Cannot handle huge workloads (millions of requests)

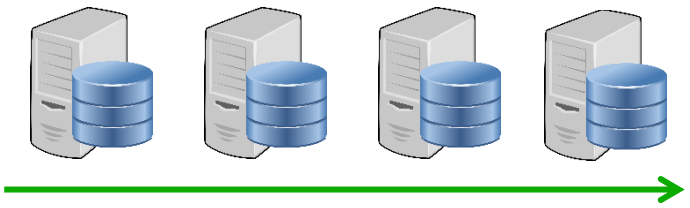
✕ Single point of failure.

✕ Price/performance ratio is poor for top-of-the-line machines.

✕ 48 2-core machines are much cheaper than 1 96-core machine

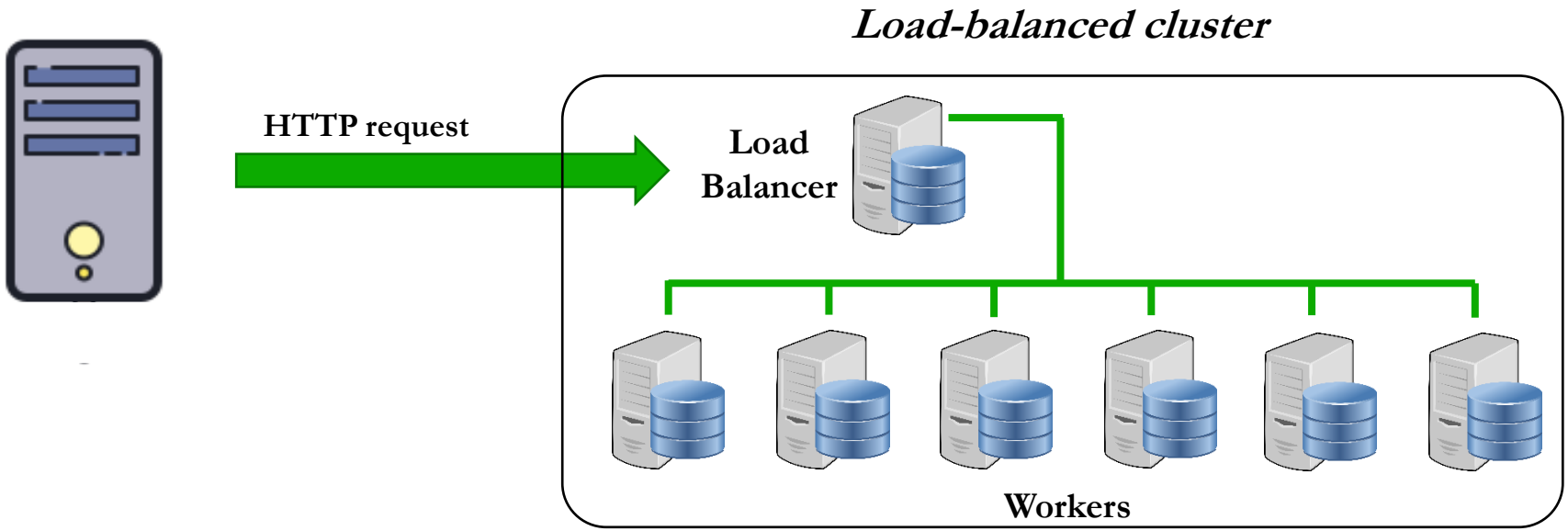
Vertical scaling is not scalable!

Horizontal scaling enables true scalability…



Horizontal scaling => load balancer

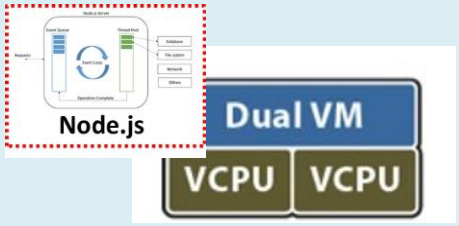
- You need a machine to distribute the incoming requests…
- **Load balancer** => makes a cluster of servers act like one *BIG* server
- The **load balancer** provides the same interface as a single server --- e.g. **acts** like a web server operating on a single network address



Virtual machines (VMs)

- **In the cloud we think in terms of virtual hardware**
  - *You don't rent a physical computer*
  - *You rent a "virtual" computer that can live / run anywhere*

```
app.get('/stats', (req, res) => {

  s3_response = s3.send("HeadBucketCommand"));

  db.query(`Select count(*) from users;
            Select count(*) from assets;`, async (err, results, _) => {

    if (err) {
      res.status(400).json(…);
      return;
    }

    var s3_result = await s3_response;  // wait for it…
    .
    .  // we have all the results, extract values and respond:
    .
    res.json(…);
  });
});
```



AWS EC2 instance types

- **AWS offers different VMs at different price points**
  - **t3.nano** (2 cores, 0.5GB, remote SSD disk) $.0052/hour
  - **m5d.24xlarge** (96 cores, 384GB, local SSD disk) $5.424/hour

Name	vCPUs	Memory (GiB)
t3.nano	2	0.5
t3.micro	2	1.0
t3.small	2	2.0
t3.medium	2	4.0
t3.large	2	8.0
t3.xlarge	4	16.0
t3.2xlarge	8	32.0

m5d.4xlarge	16	64
m5d.8xlarge	32	128
m5d.12xlarge	48	192
m5d.16xlarge	64	256
m5d.24xlarge	96	384

That's it, thank you!