Linux Port Sharding

@joewalnes

Wat?

Linux Kernel **3.9** introduced new socket option

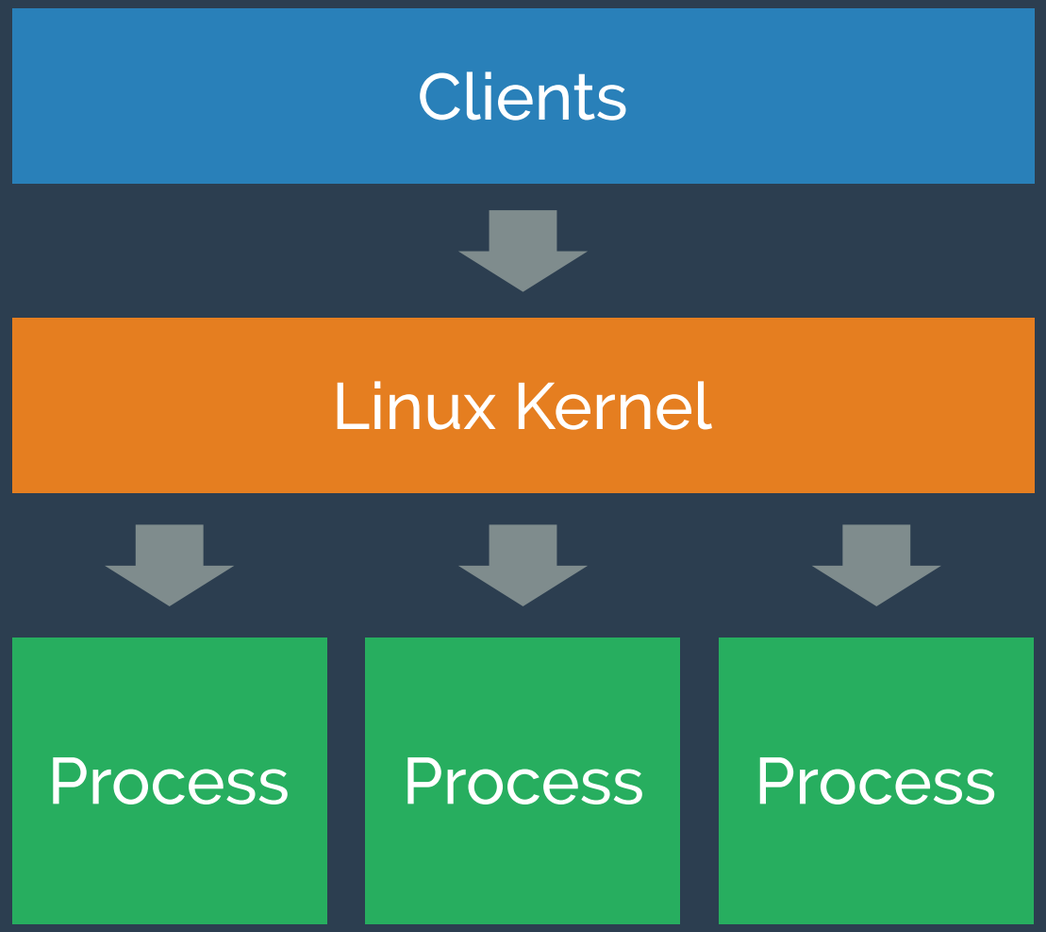
SO_REUSEPORT

```
index 9a6115e..37b699f 100644
--- a/include/asm-generic/socket.h
+++ b/include/asm-generic/socket.h
@@ -22,7 +22,7 @@
 #define SO_PRIORITY     12
 #define SO_LINGER       13
 #define SO_BSDCOMPAT    14
-/* To add :#define SO_REUSEPORT 15 */
+#define SO_REUSEPORT    15

 #ifndef SO_PASSCRED /* powerpc only differs
 #define SO_PASSCRED     16
diff --git a/include/linux/inet.h b/include/
index 4cca05c..bd8f0b6 100644
--- a/include/linux/inet.h
+++ b/include/linux/inet.h
@@ -51,6 +51,12 @@
 #define INET_ADDRSTRLEN                 (16)
 #define INET6_ADDRSTRLEN        (48)

+static inline u32 inet_next_pseudo_random32
+{
+       /* Pseudo random number generator fr
+       return seed * 1664525 + 1013904223;
+}
+
 extern __be32 in_aton(const char *str);
 extern int in4_pton(const char *src, int sr
 extern int in6_pton(const char *src, int sr
diff --git a/include/net/inet_hashtables.h b
index 74358d1..0887675 100644
--- a/include/net/inet_hashtables.h
+++ b/include/net/inet_hashtables.h
@@ -81,7 +81,9 @@ struct inet_bind_bucket {
        struct net              *ib_net;
 #endif
        unsigned short          port;
-       signed short            fastreuse;
```

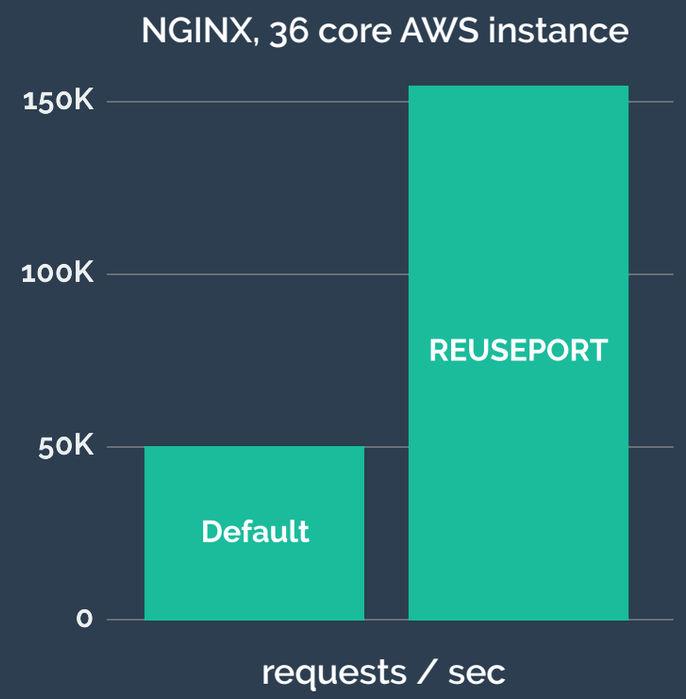
Allows **multiple processes** to listen on **same port**



Kernel distributes incoming connections

Why?

Spread socket event loop across multiple CPU cores

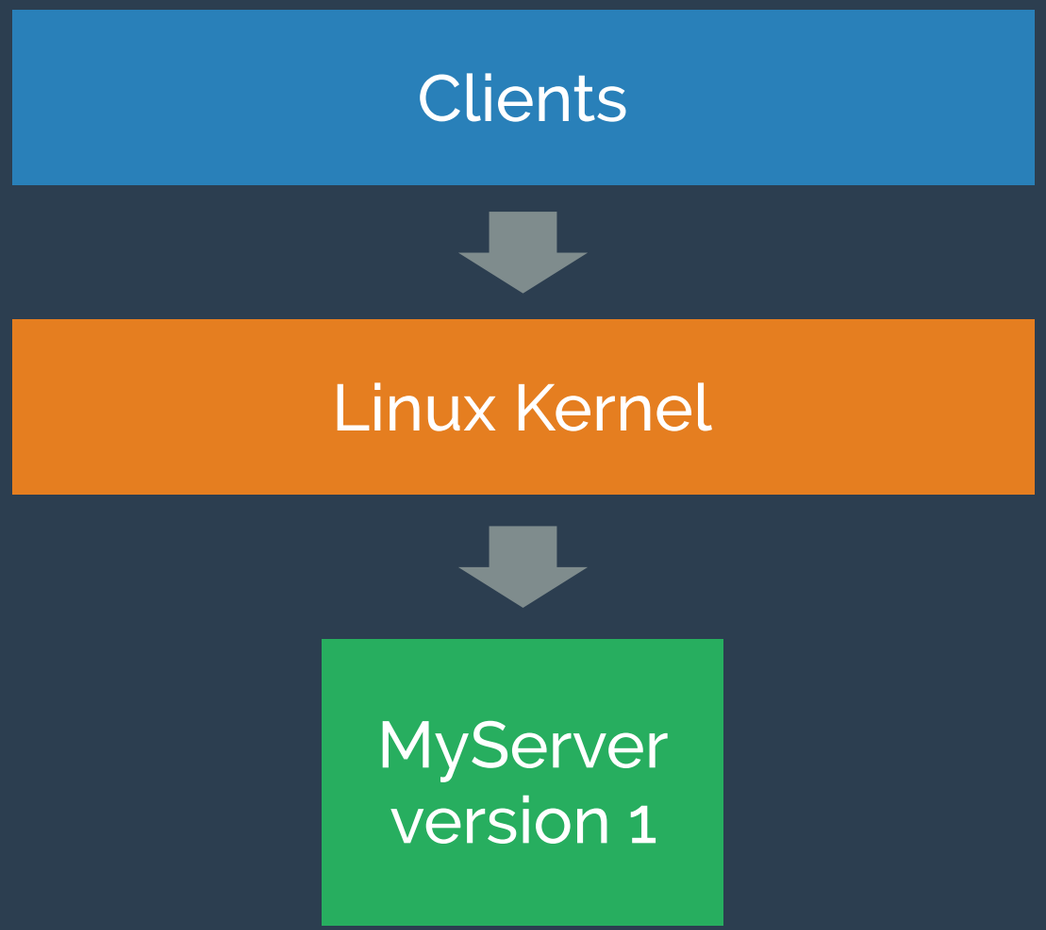


Source: https://www.nginx.com/blog/socket-sharding-nginx-release-1-9-1/

But it has another trick...

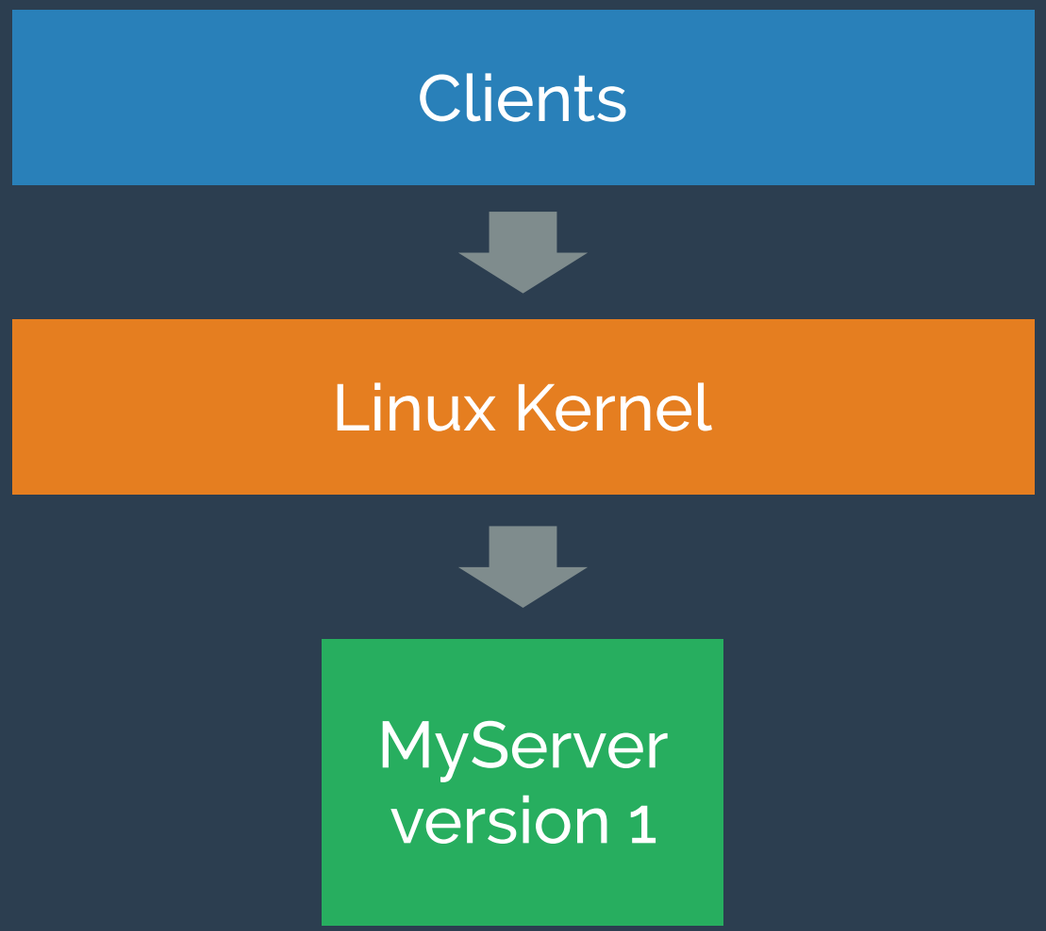
Rolling upgrades

Rolling upgrades



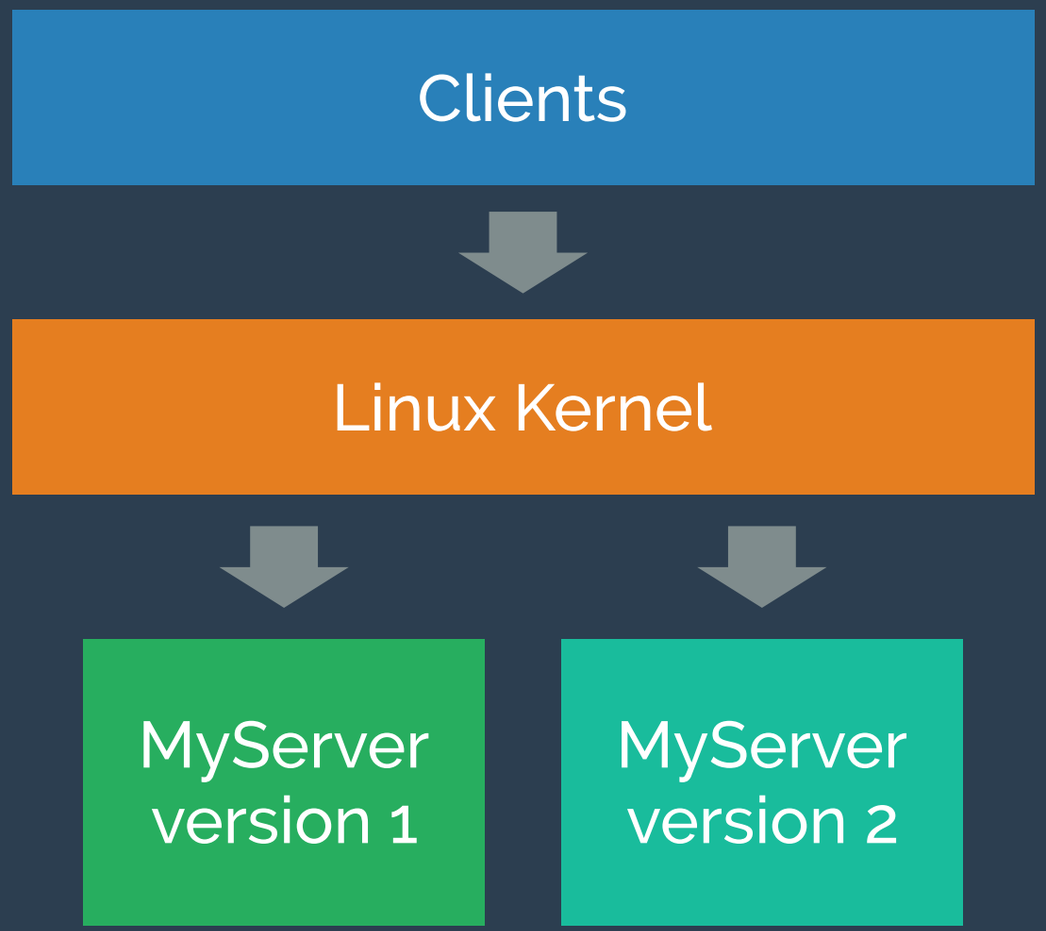
Rolling upgrades

Let's upgrade a server application



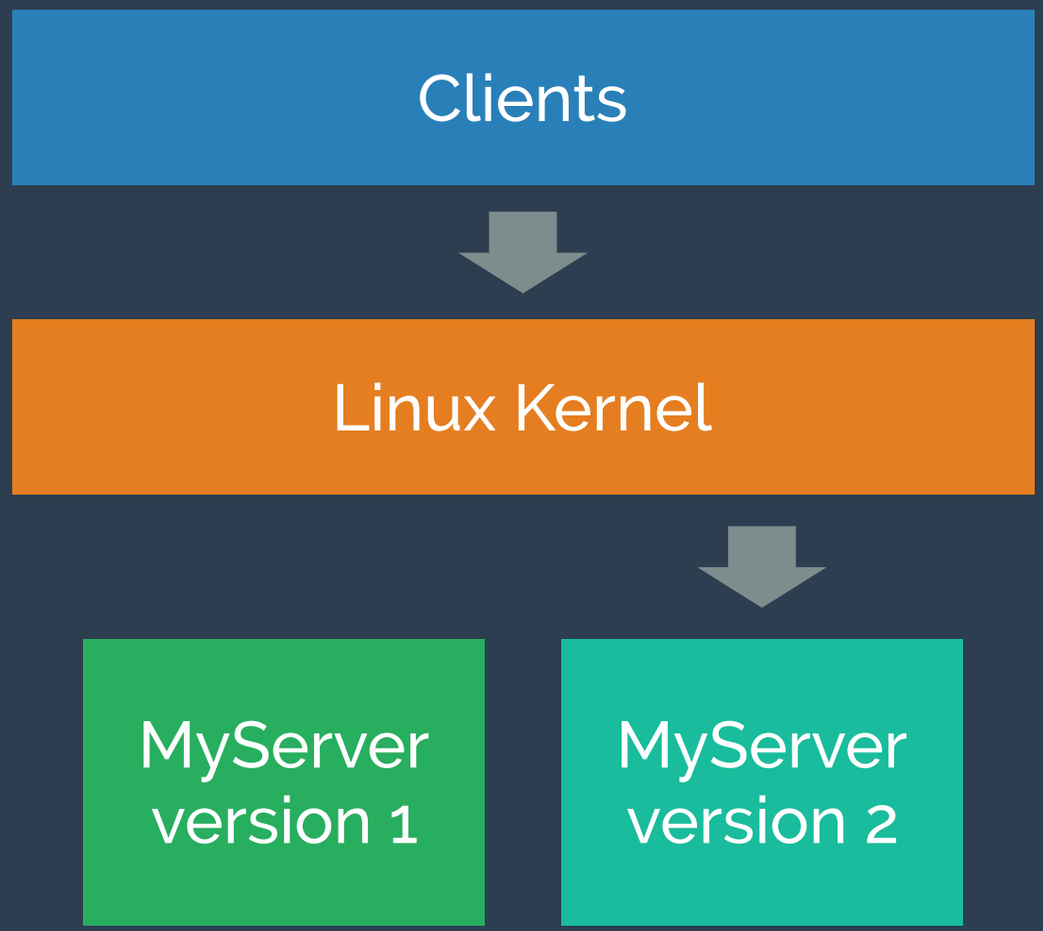
Rolling upgrades

Start new version: accepts request alongside old



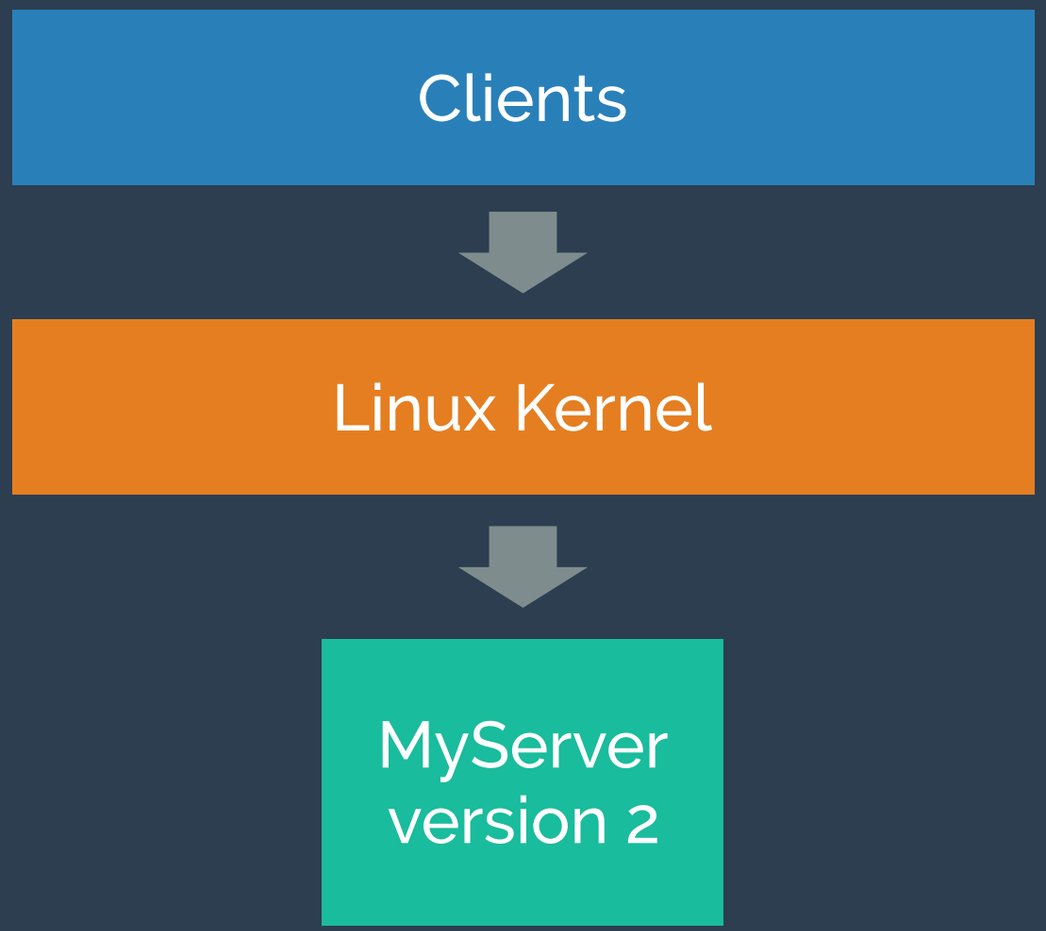
Rolling upgrades

Signal old version to stop accepting new requests



Rolling upgrades

Old version terminates when done with requests



Using it

Ruby edition

```ruby
require "socket"

port = 1234

server = Socket.new Socket::AF_INET,
                    Socket::SOCK_STREAM
server.setsockopt Socket::SOL_SOCKET,
                  Socket::SO_REUSEPORT, true
server.bind Addrinfo.tcp "", port
server.listen 0

loop do
    client, addr = server.accept
    data = client.recv 1024
    # echo data back
    client.send data, 0
    client.close
end
```

Python edition

```python
from socket import *

port = 1234

server = socket(AF_INET, SOCK_STREAM)
server.setsockopt(SOL_SOCKET, SO_REUSEPORT, 1)
server.bind(('', port))
server.listen(0)

while True:
    client, addr = server.accept()
    data = client.recv(1024)
    # echo data back
    client.send(data)
    client.close()
```

Caveats:

Don't confuse this with the BSD SO_REUSEPORT option of exactly the same name (it won't work on OSX)

Also it's not SO_REUSEADDR

Sharing is not very smart: kernel still passes to process even when slow at accepting connections

More on SO_REUSEPORT → https://lwn.net/Articles/542629/

Try it → https://github.com/joewalnes/port-sharding