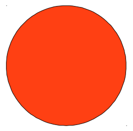
Sample Problem Solution

- Let's show each process as a circle...

```
int main (int argc, char *arg[])
{
    fork ();
    if (fork ()) {
        fork ();
    }
    fork ();
}
```

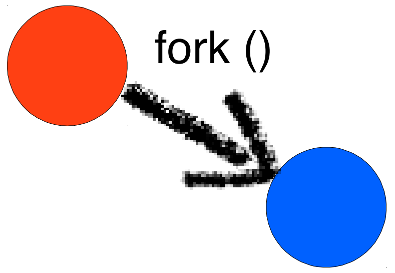


Red: original process right when main begins

Sample Problem Solution

```
int main (int argc, char *arg[])
{
    fork ();
    if (fork ()) {
        fork ();
    }
    fork ();
}
```

Call to fork() creates a copy of the original process: blue



Sample Problem Solution

```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    if (fork ()) {
        fork ();
    }
    fork ();
}
```

We now have two independent processes, red and blue, each about to execute the same code

Note:

```
 if (value) {
   Executed if value != 0
 } else {
   Executed if value == 0
 }
```

Sample Problem Solution

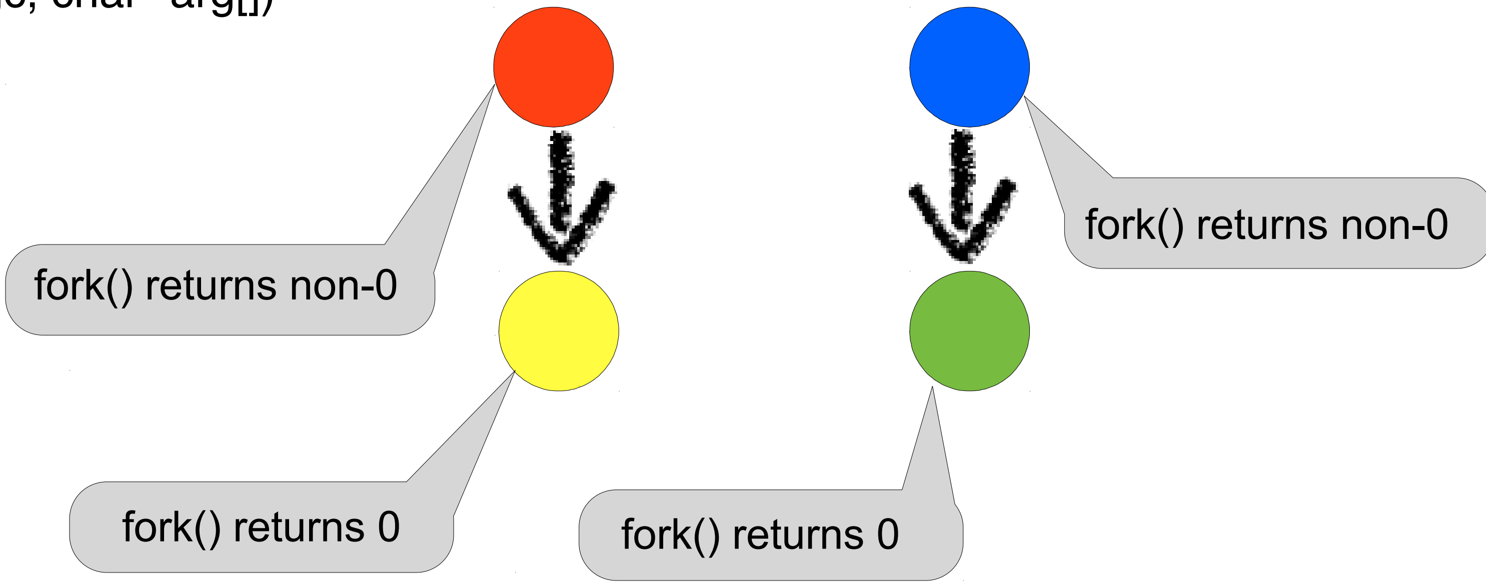
```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    if (fork ()) {
        fork ();
    }
    fork ();
}
```

We now have two independent processes, each about to execute the same code

This code calls fork() and test its return value

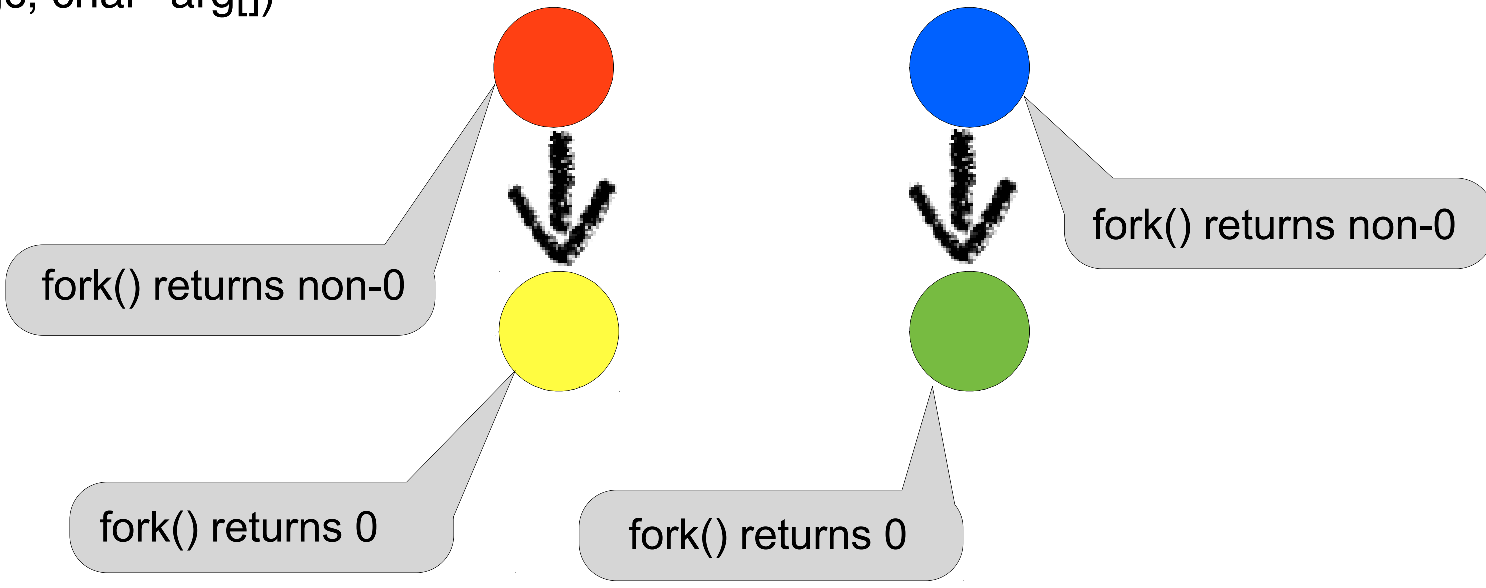
Sample Problem Solution

```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    **if (fork ()) {**
        fork ();
    }
    fork ();
}
```



Sample Problem Solution

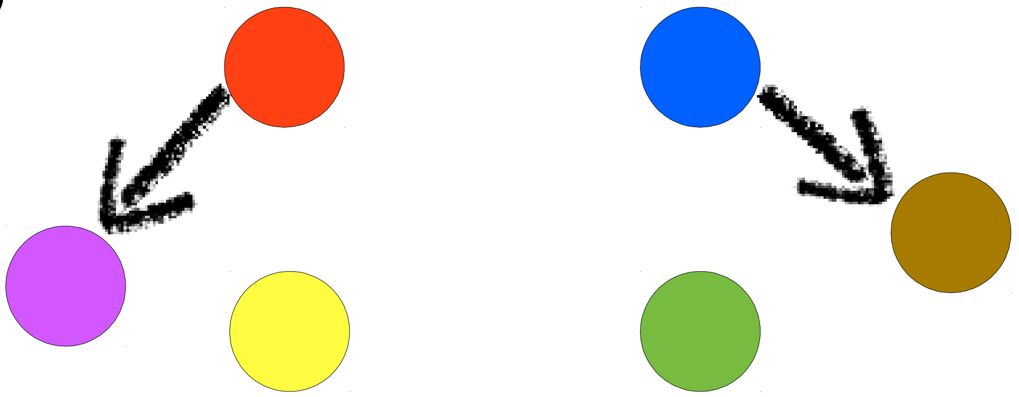
```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    if (fork ()) {
        fork ();
    }
    fork ();
}
```



yellow and green: don’t go into the if clause
red and blue: go!!

Sample Problem Solution

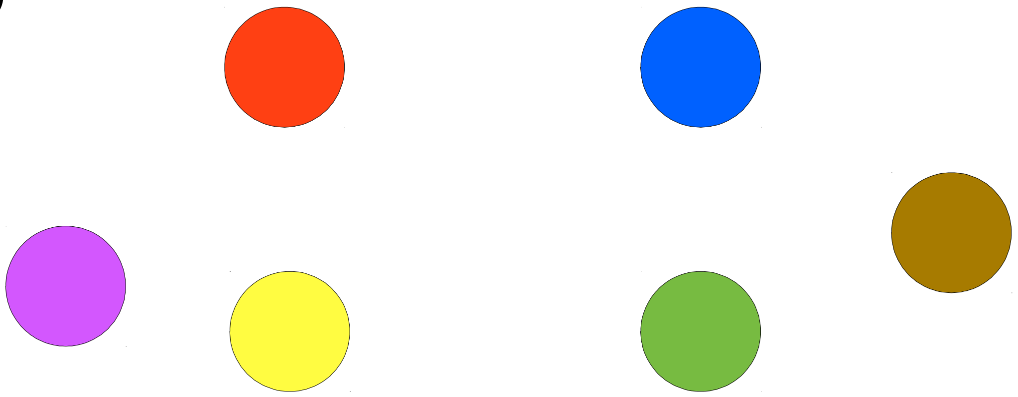
```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    if (fork ()) {
        fork ();
    }
        fork ();
}
```



red and blue each creates a new child process (purple an brown)

Sample Problem Solution

```
int main (int argc, char *arg[])
{
    ~~fork ();~~
    if (fork ()) {
        fork ();
    }
    fork ();
}
```



ALL processes execute the last call to fork()

- red, purple, blue and brown after they exit from the if clause
- yellow and green after they skip the if clause

We have 6 processes calling fork(), each creating a new process

So we have a total of **12 processes** at the end, one of which was the original process