Capability based IPC on Linux

```
String path = “/sys/...”;
Led myled = new Led(path);
...
myled.SetColor(Led::Green);
myled.Enable();
...
myled.Disable();
```

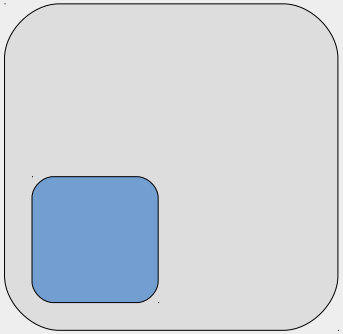
```
int Led::Enable() {
    int fd, r;

    fd = open(this.path, ...);
    if (fd < 0)
        return -errno;

    r = write(fd, “1\n”, 2);
    close(fd);
    return r < 0 ? -errno : 0;
}
```

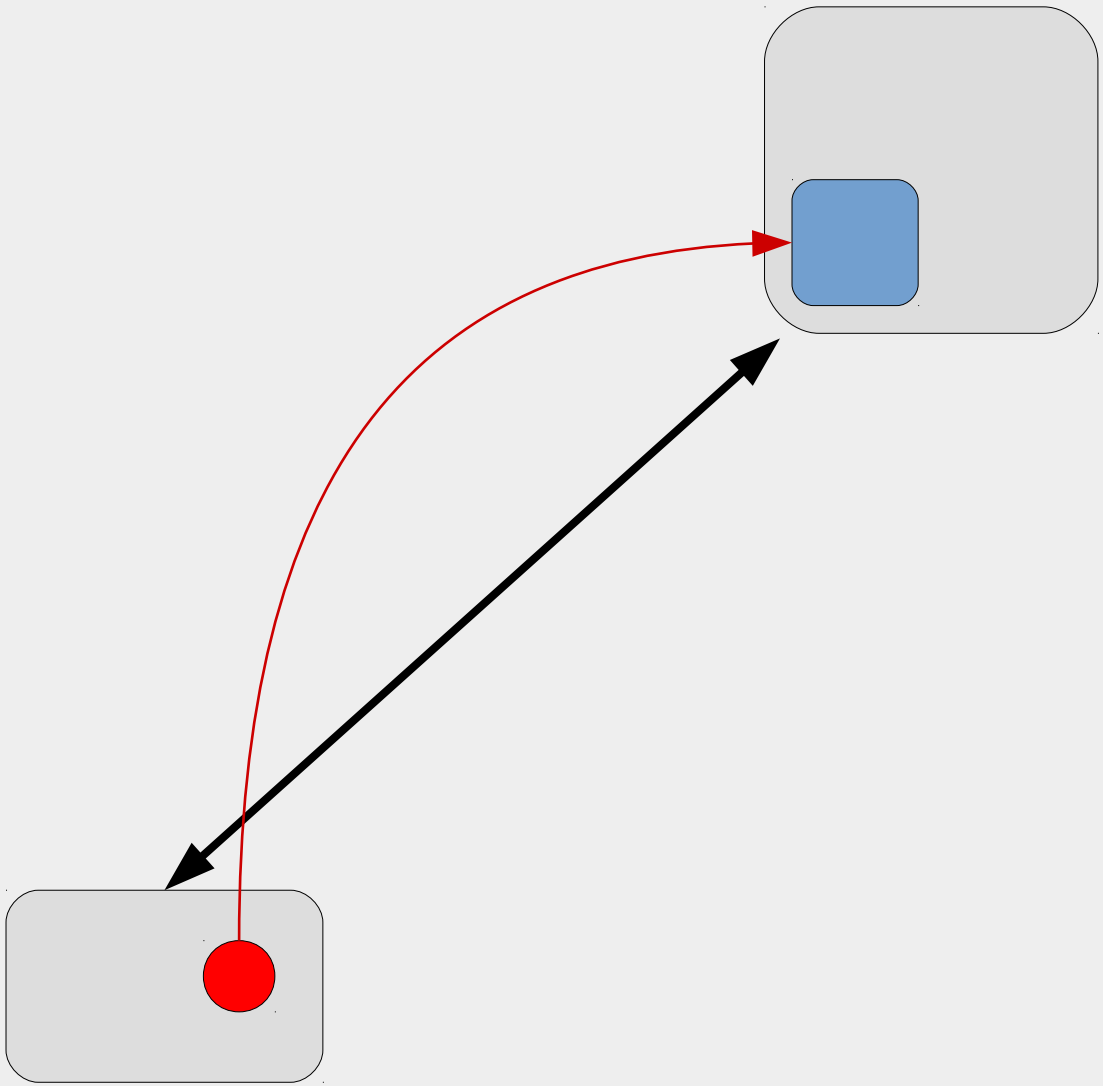
```
Handle h = app.find(“Led1”);
Led myled = new Led(h);
...
myled.SetColor(Led::Green);
myled.Enable();
...
myled.Disable();
```

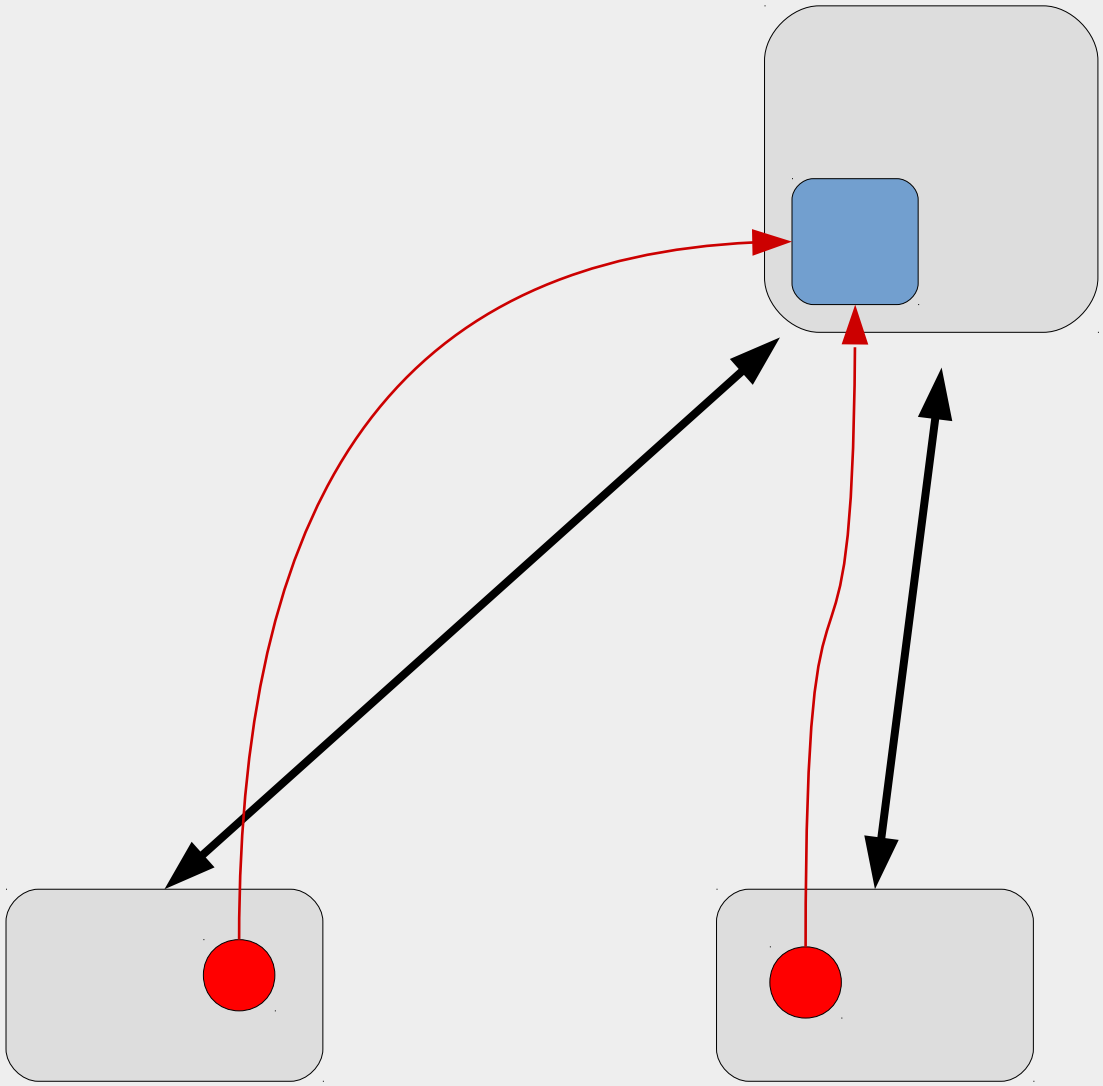
```
int Led::Enable() {
    int r, res = -EAGAIN;

    r = write(this.fd, “Enable\n”,
                    7);
    if (r < 0)
        return -errno;

    r = read(this.fd, res, 4);
    if (r < 0)
        return -errno;

    return res;
}
```

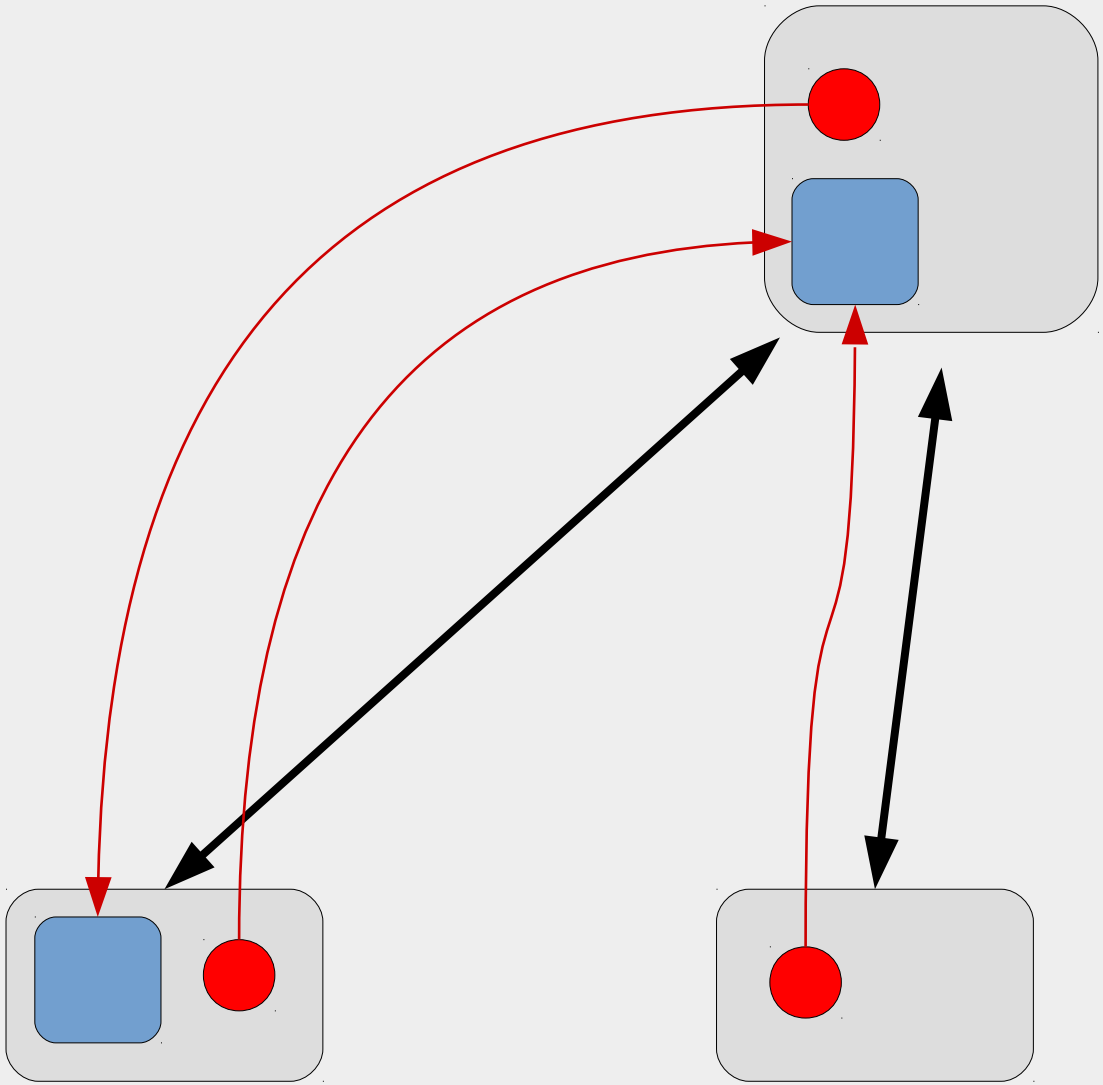
```
Handle h = app.find(“Led1”);
Led myled = new Led(h);
...
myled.SetColor(Led::Green);
myled.Enable();
...
myled.Disable();
```

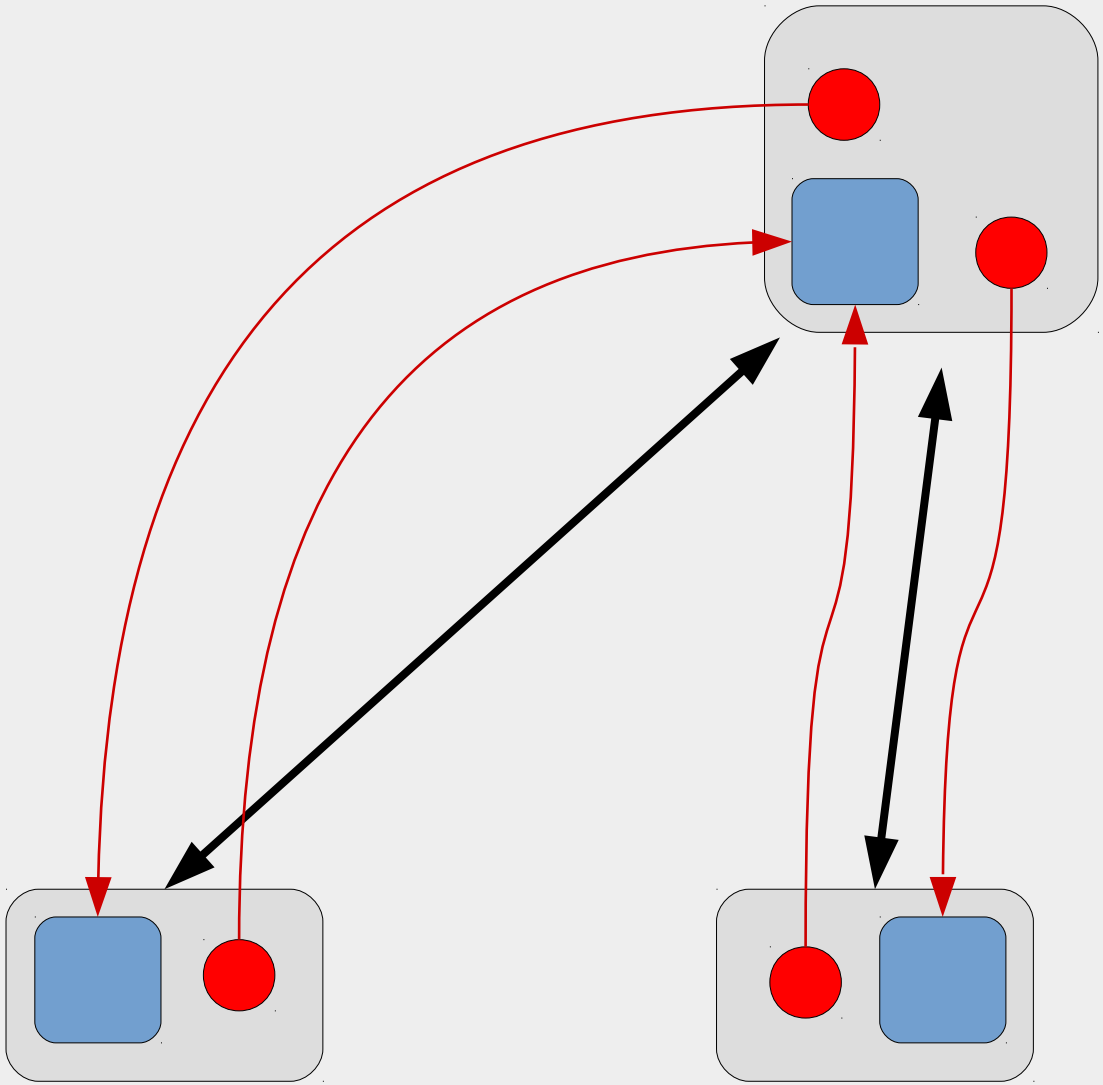
```
int Led::Enable() {
    int fd, r;

    fd = open(this.path, ...);
    if (fd < 0)
        return -errno;

    r = write(fd, “1\n”, 2);
    close(fd);
    return r < 0 ? -errno : 0;
}
```

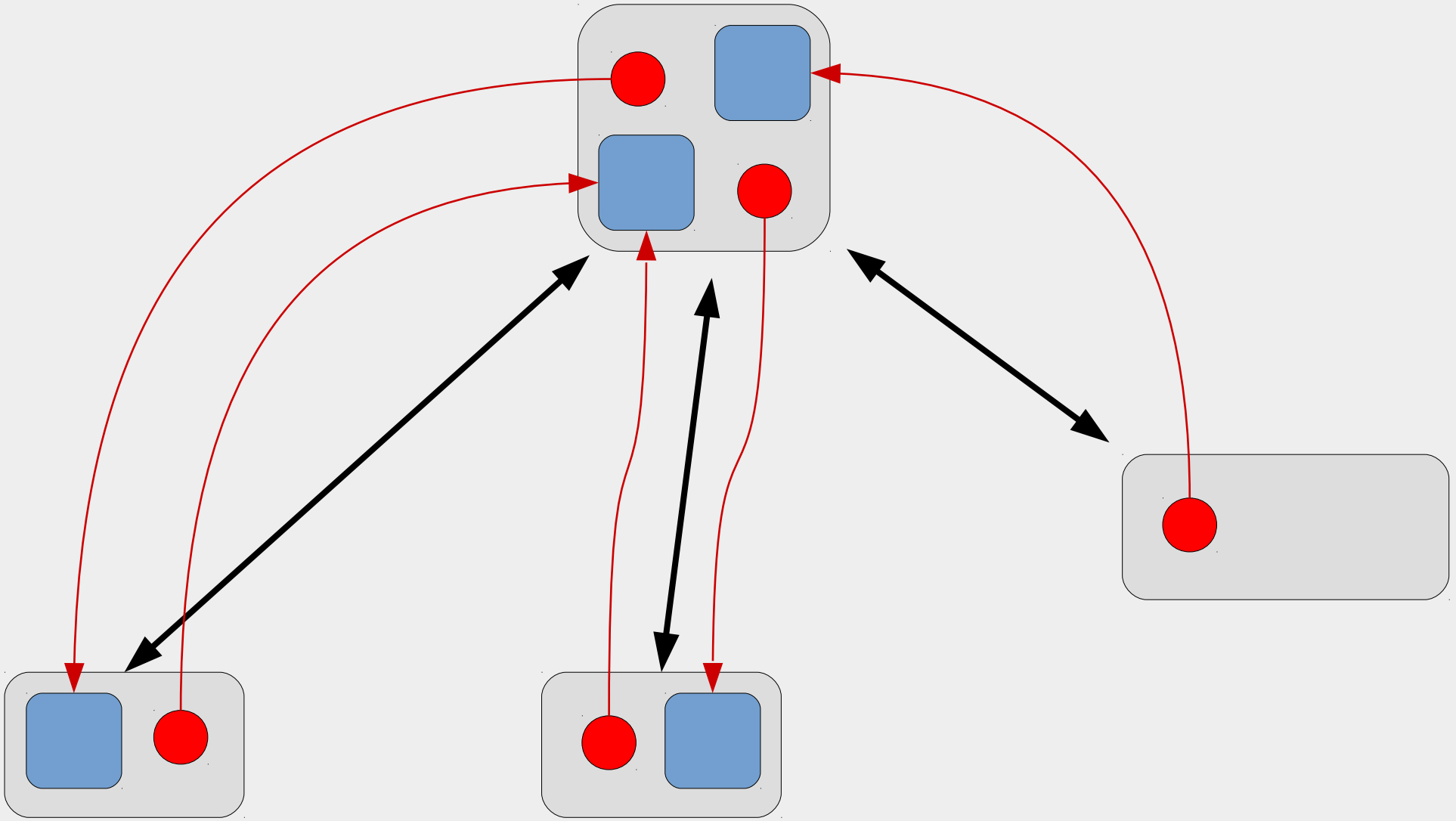


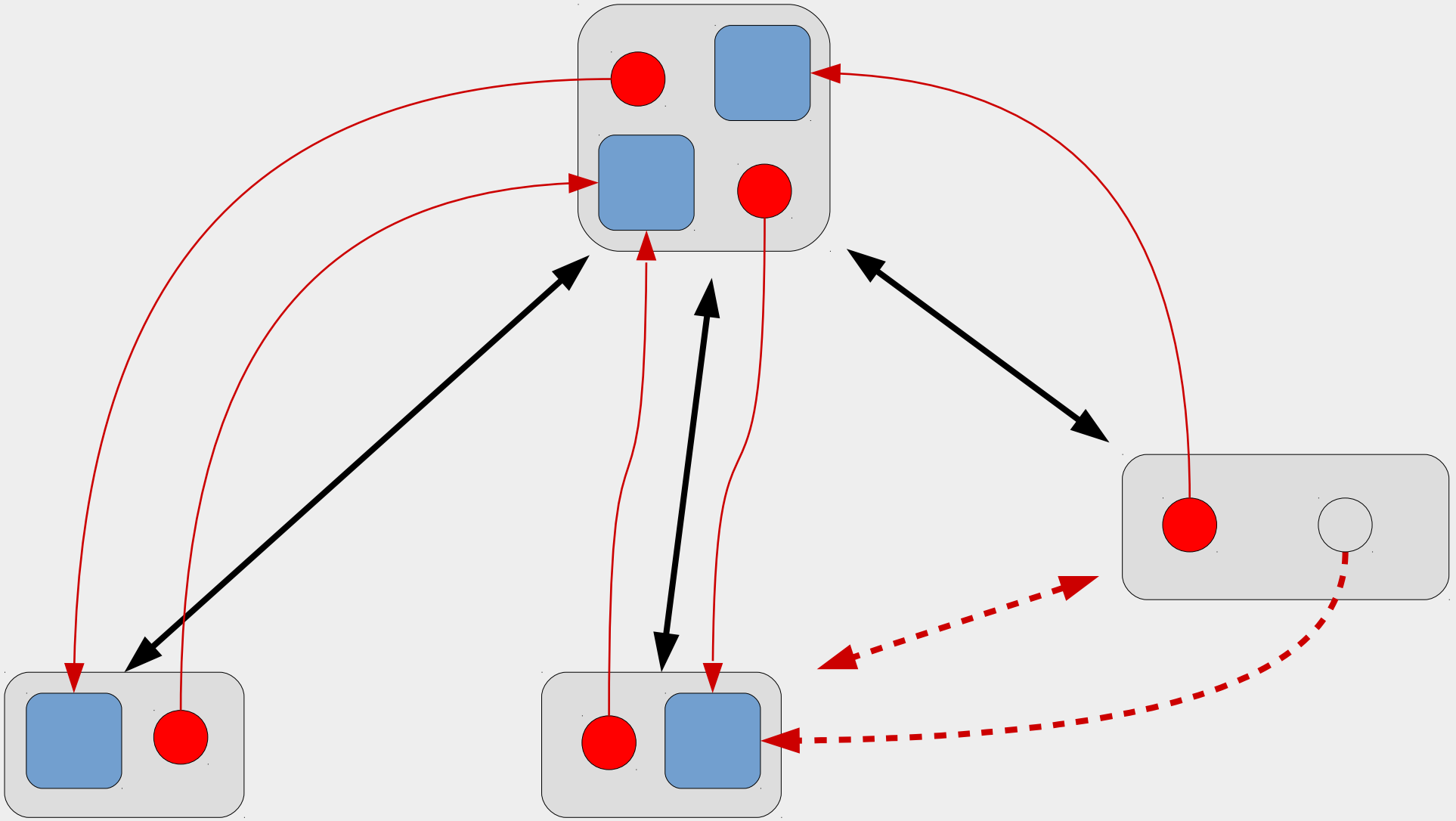


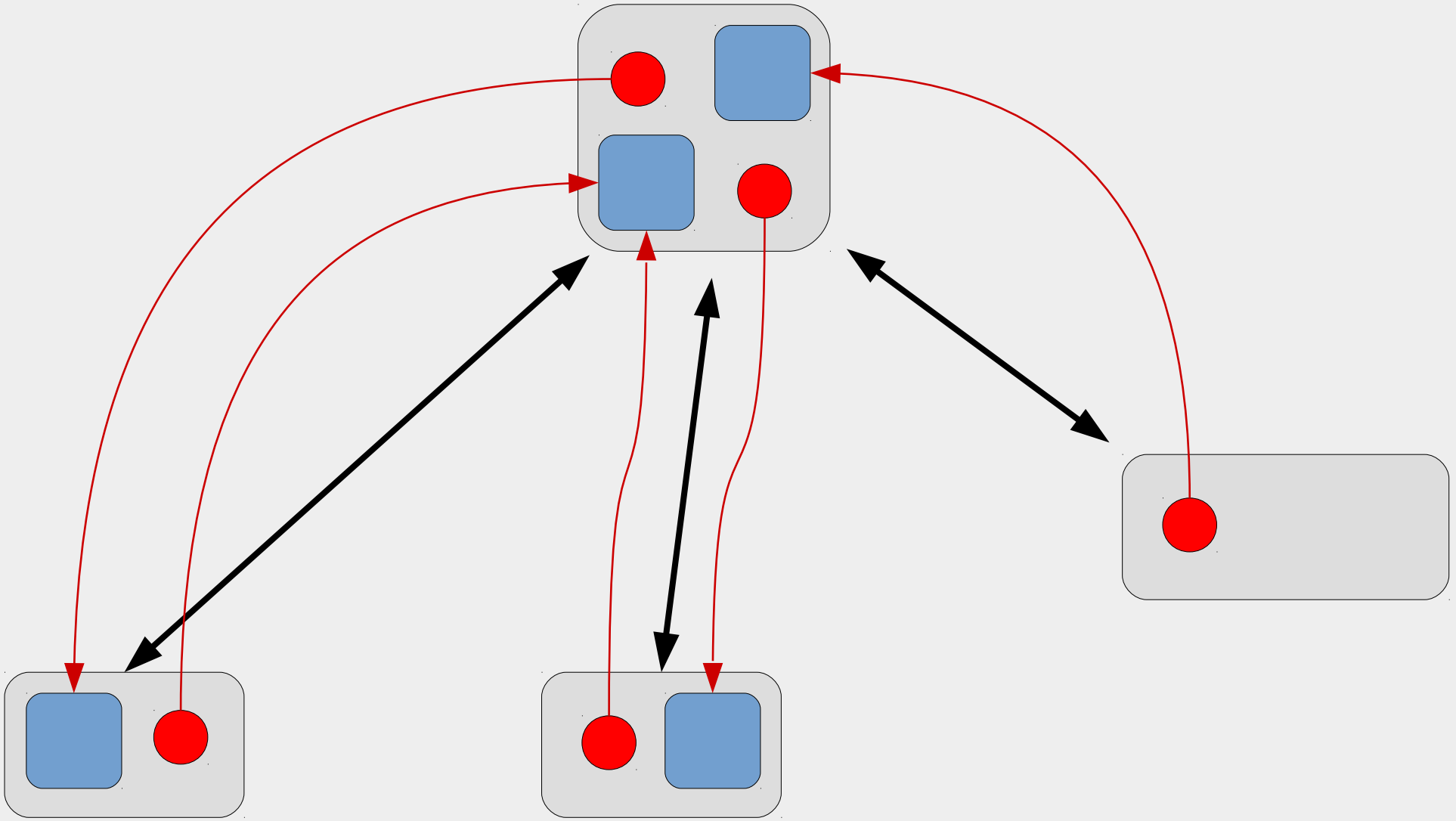


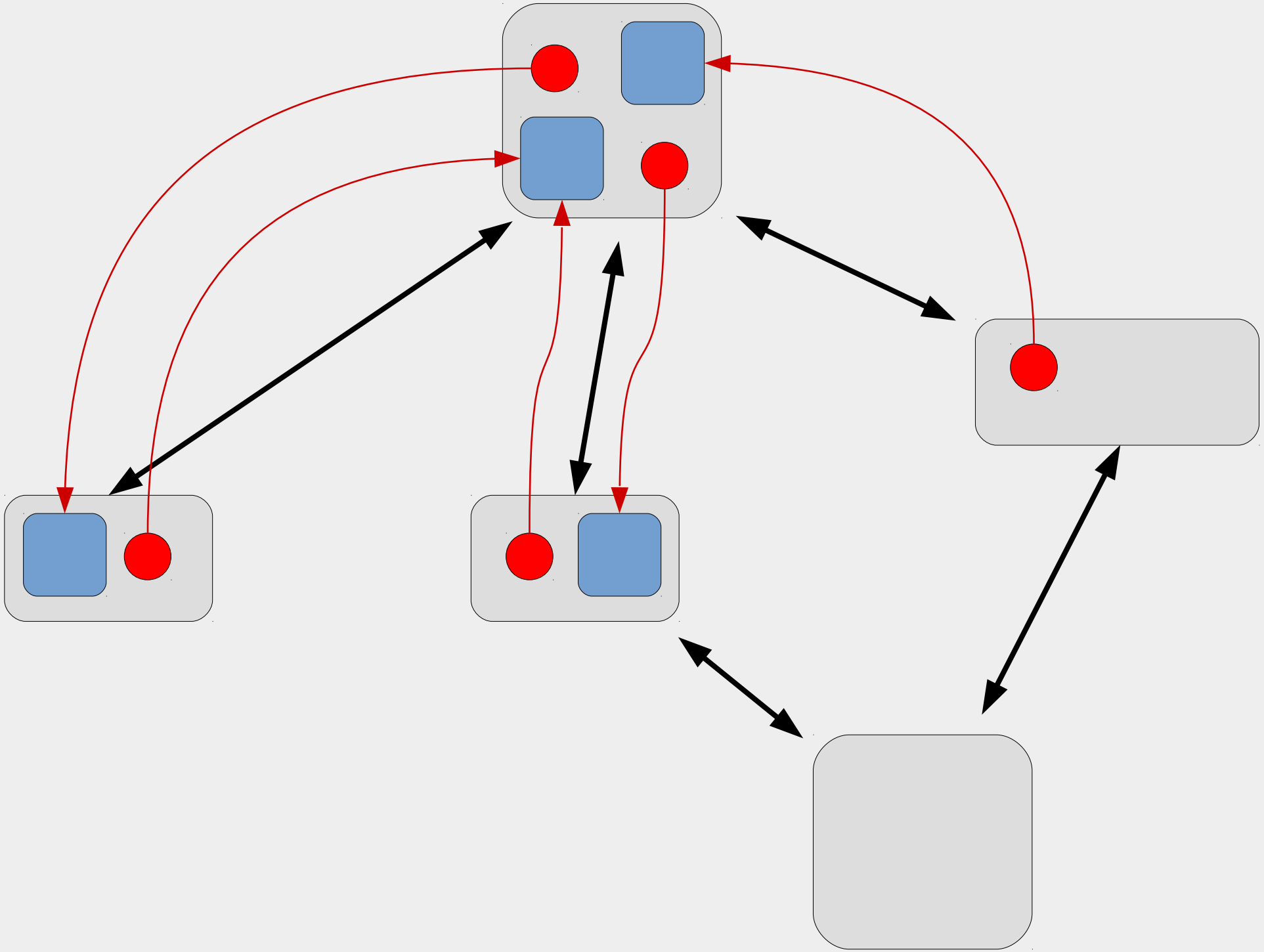


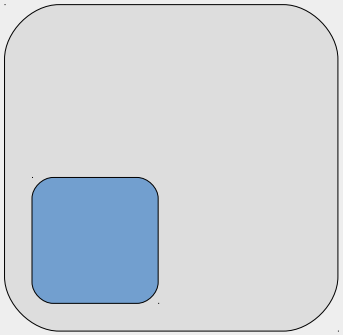


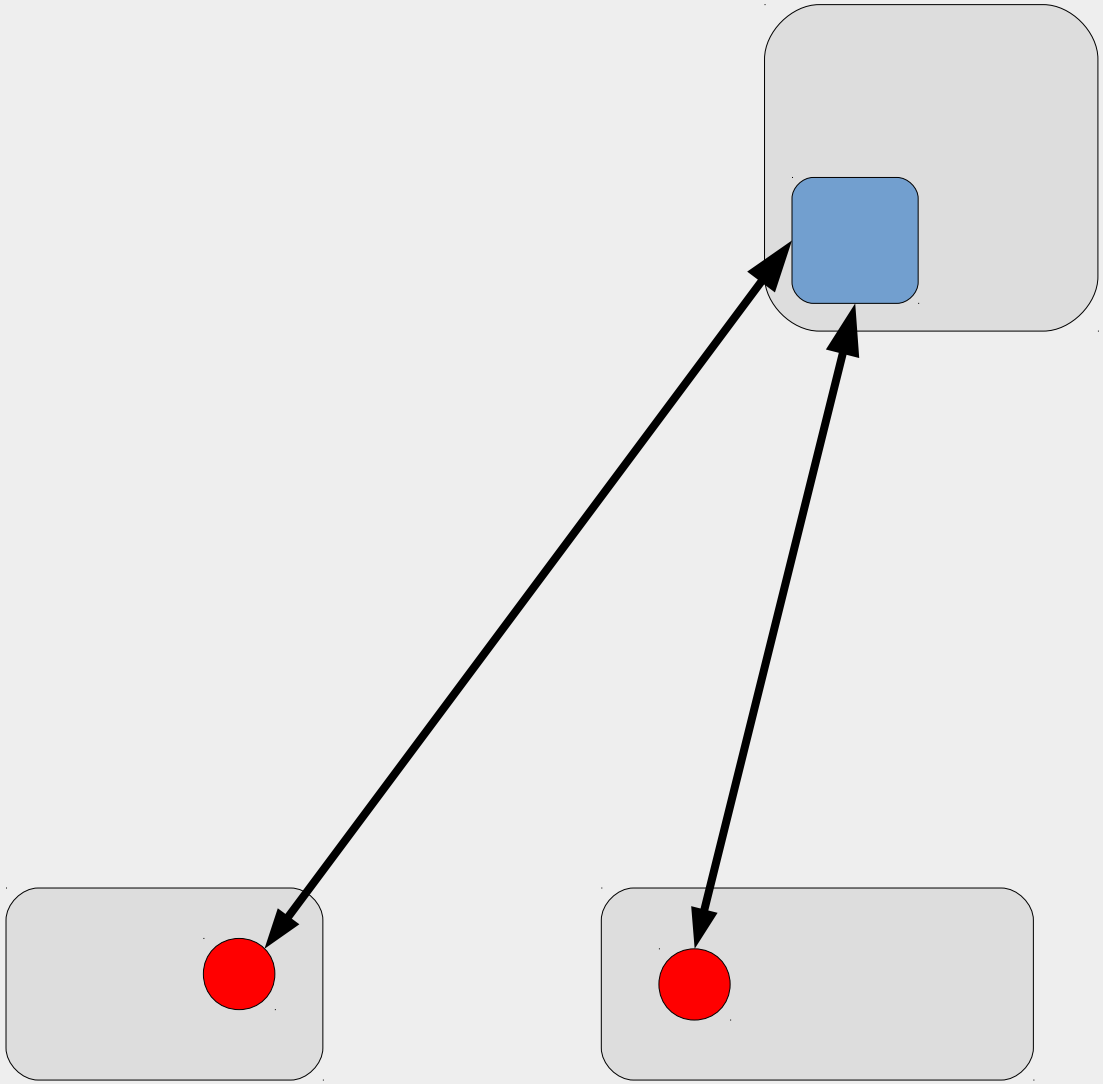


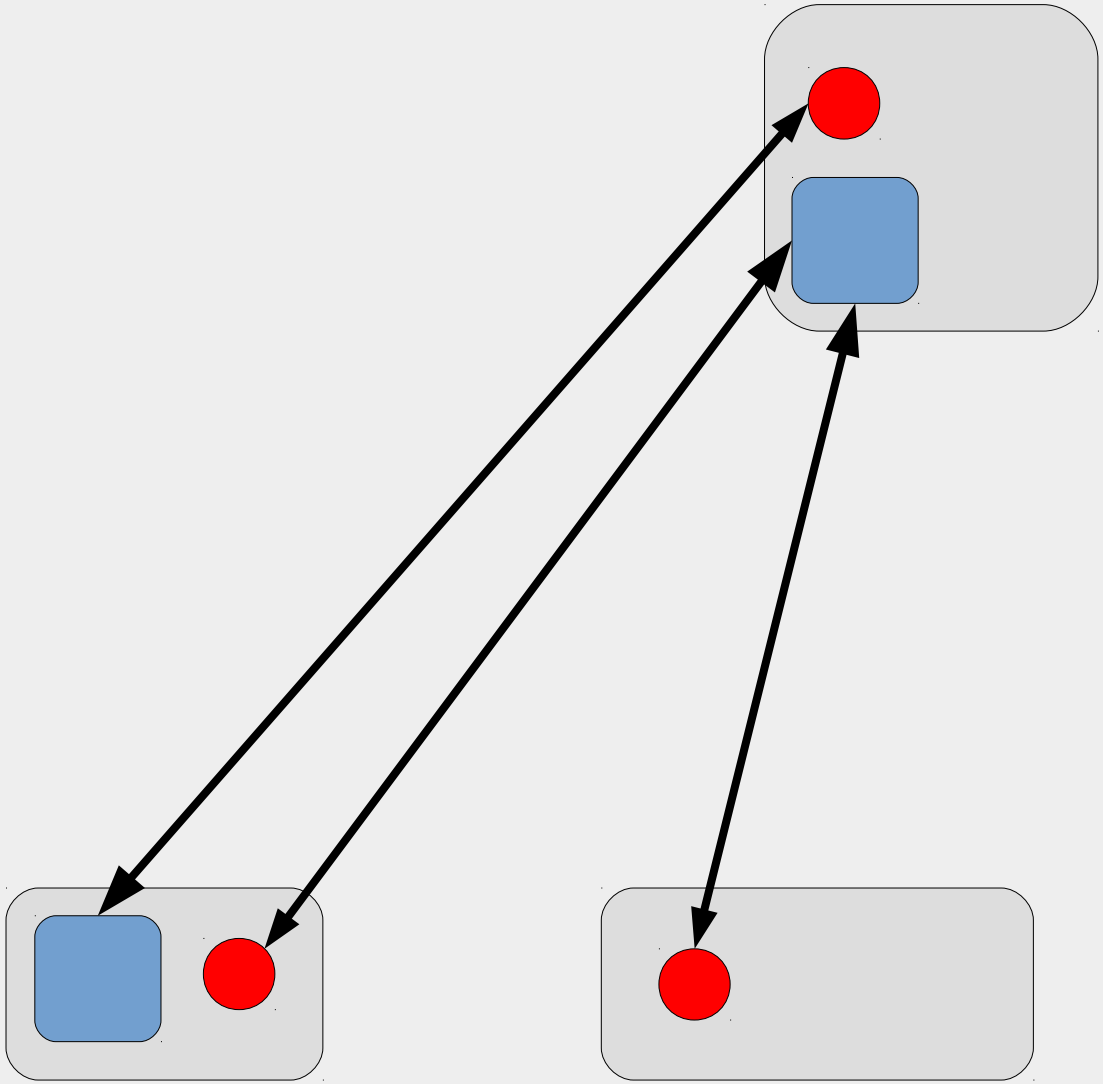


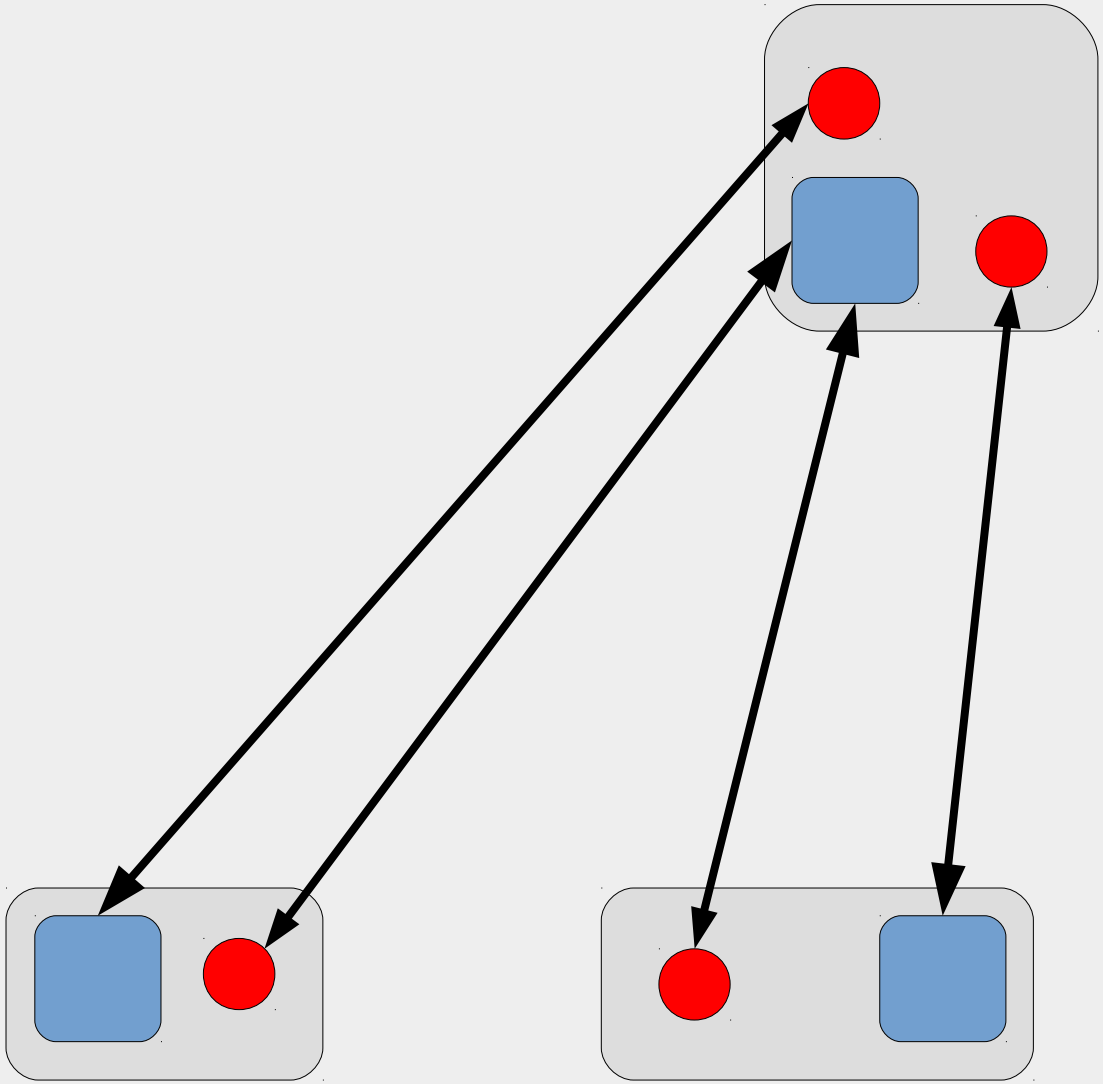


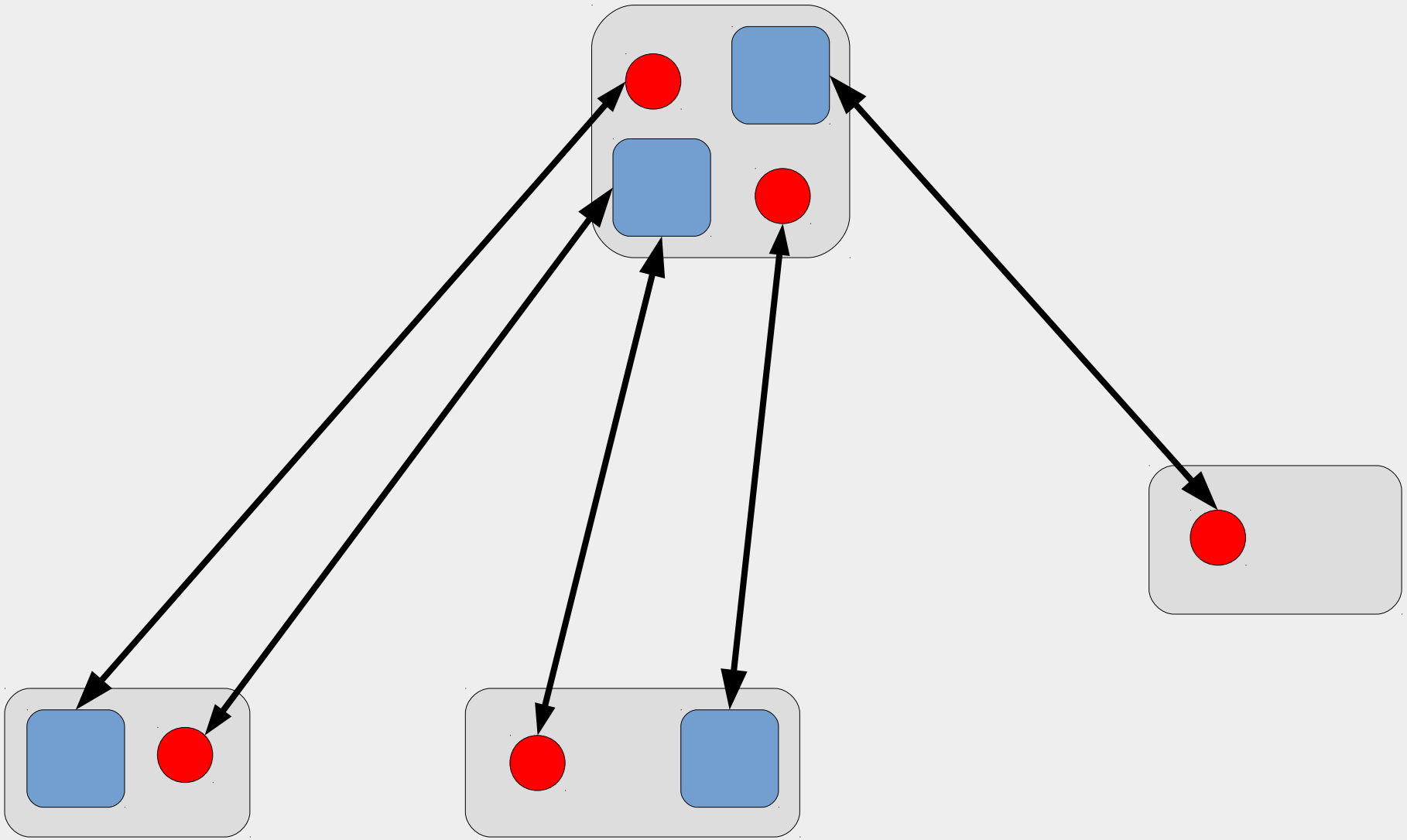


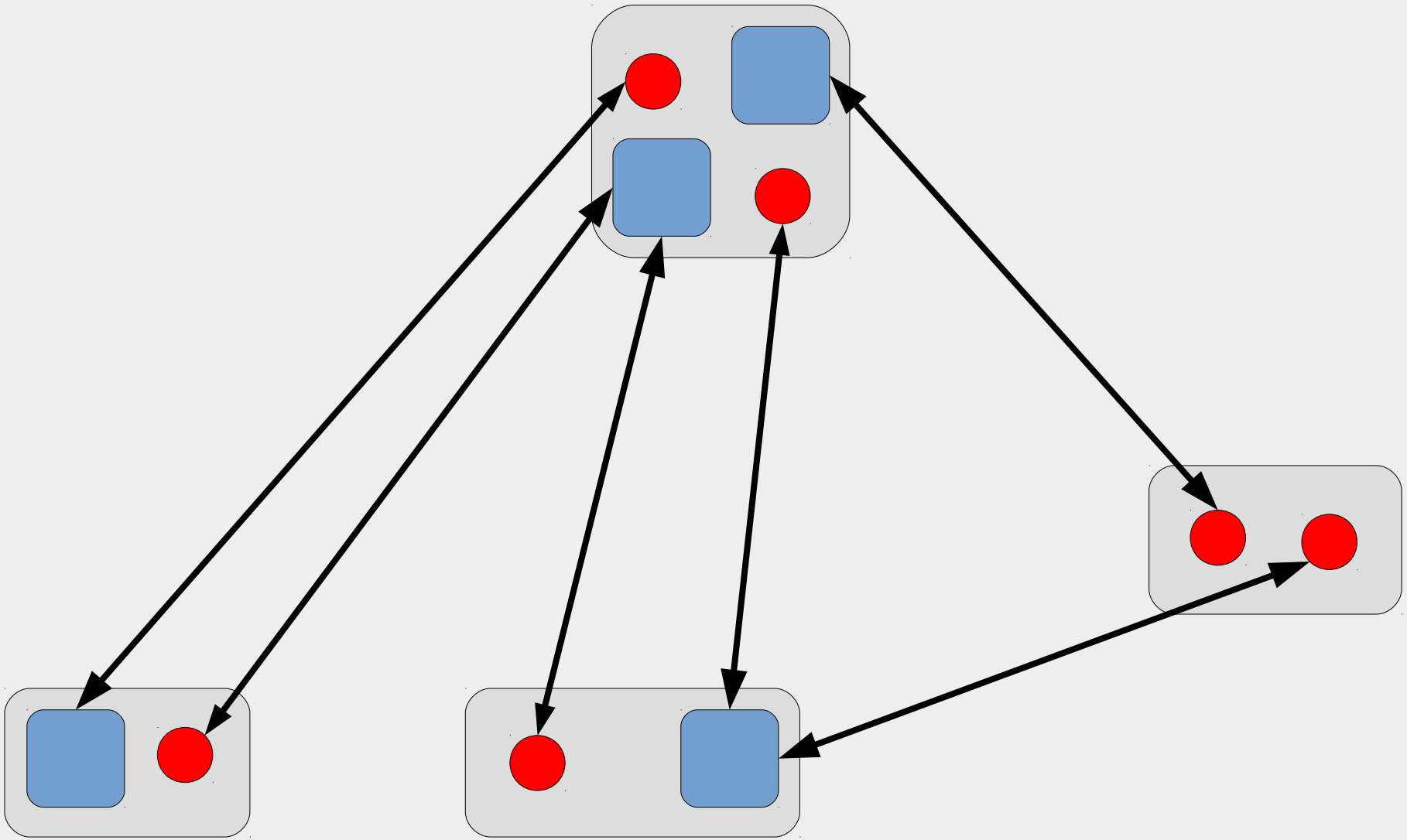


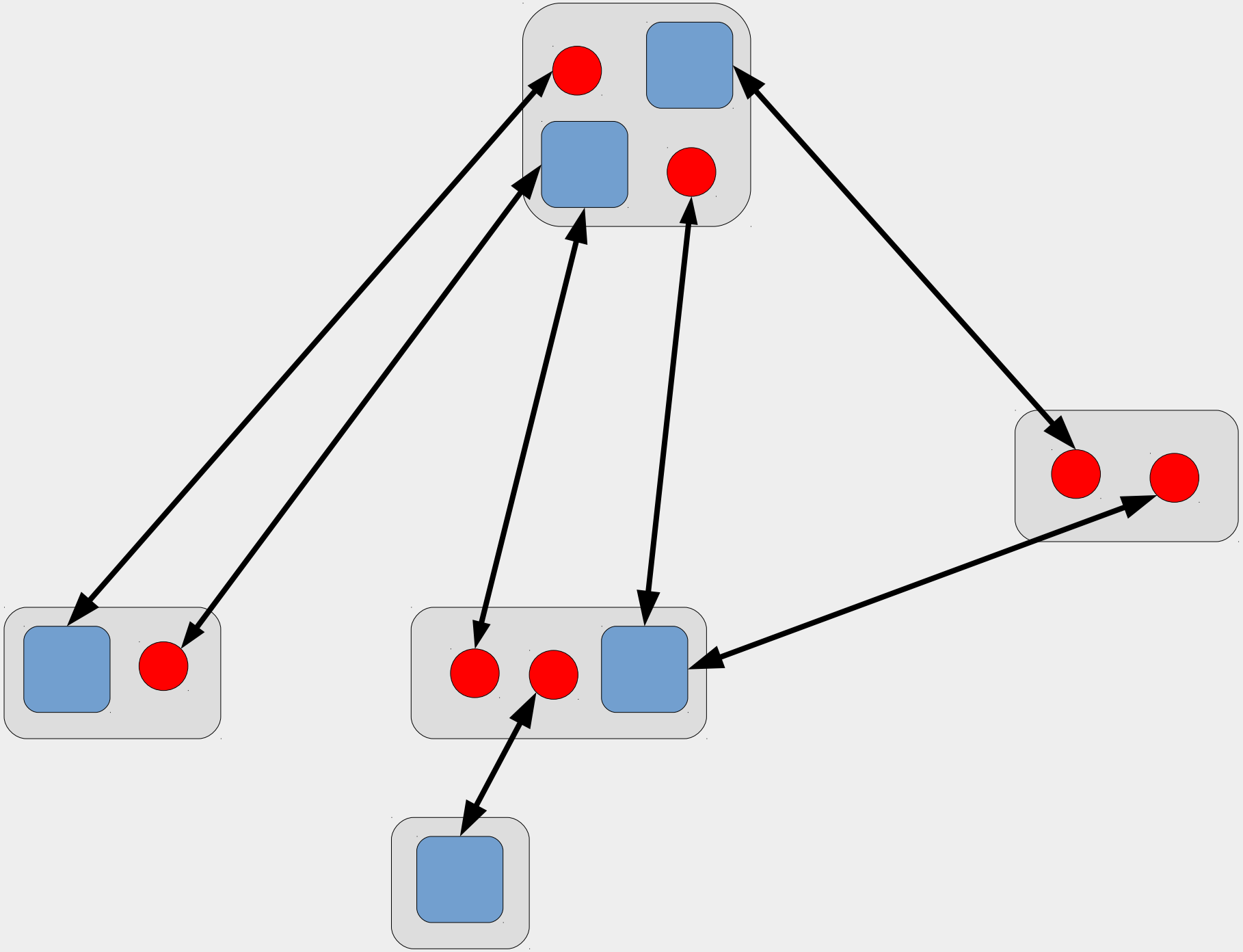


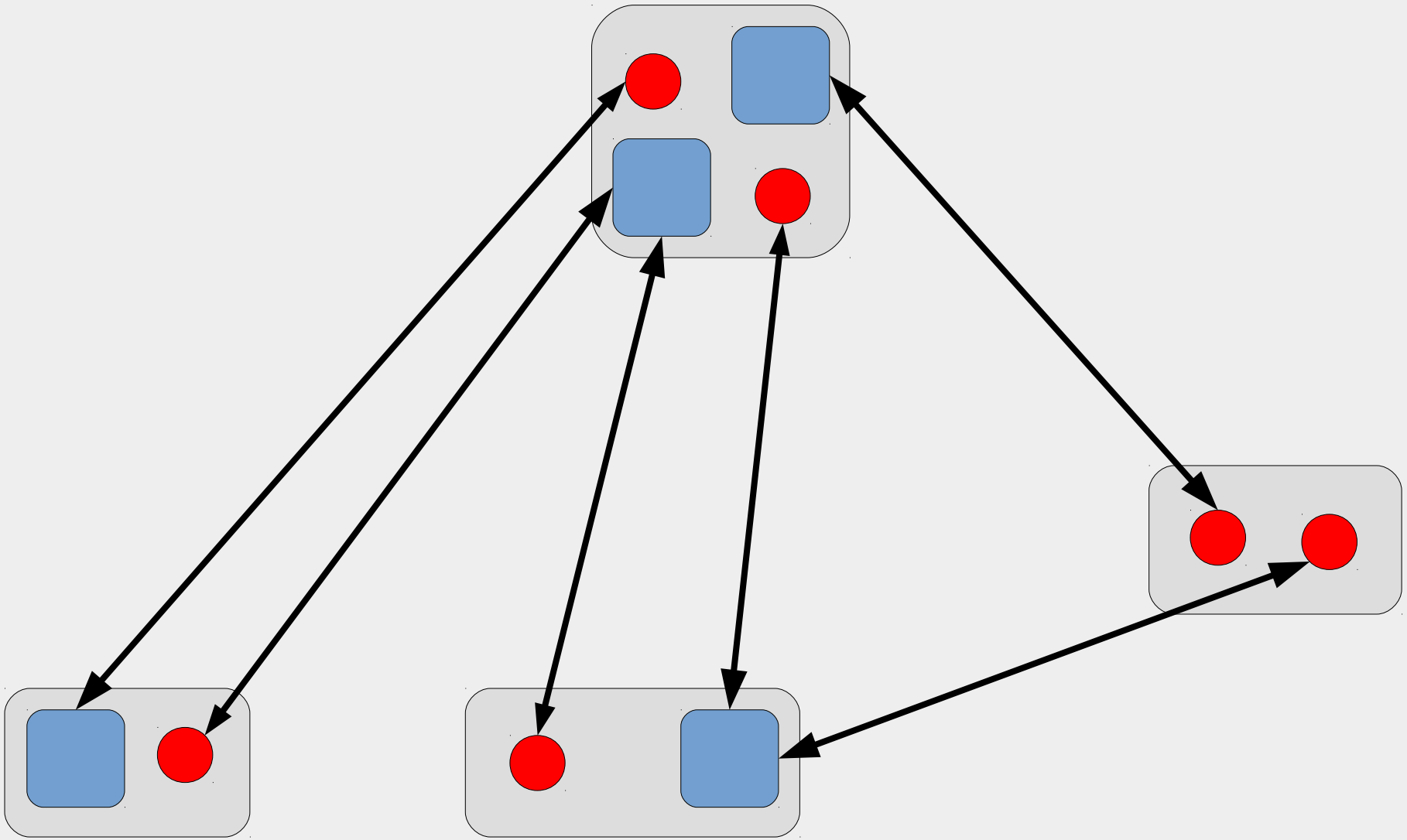


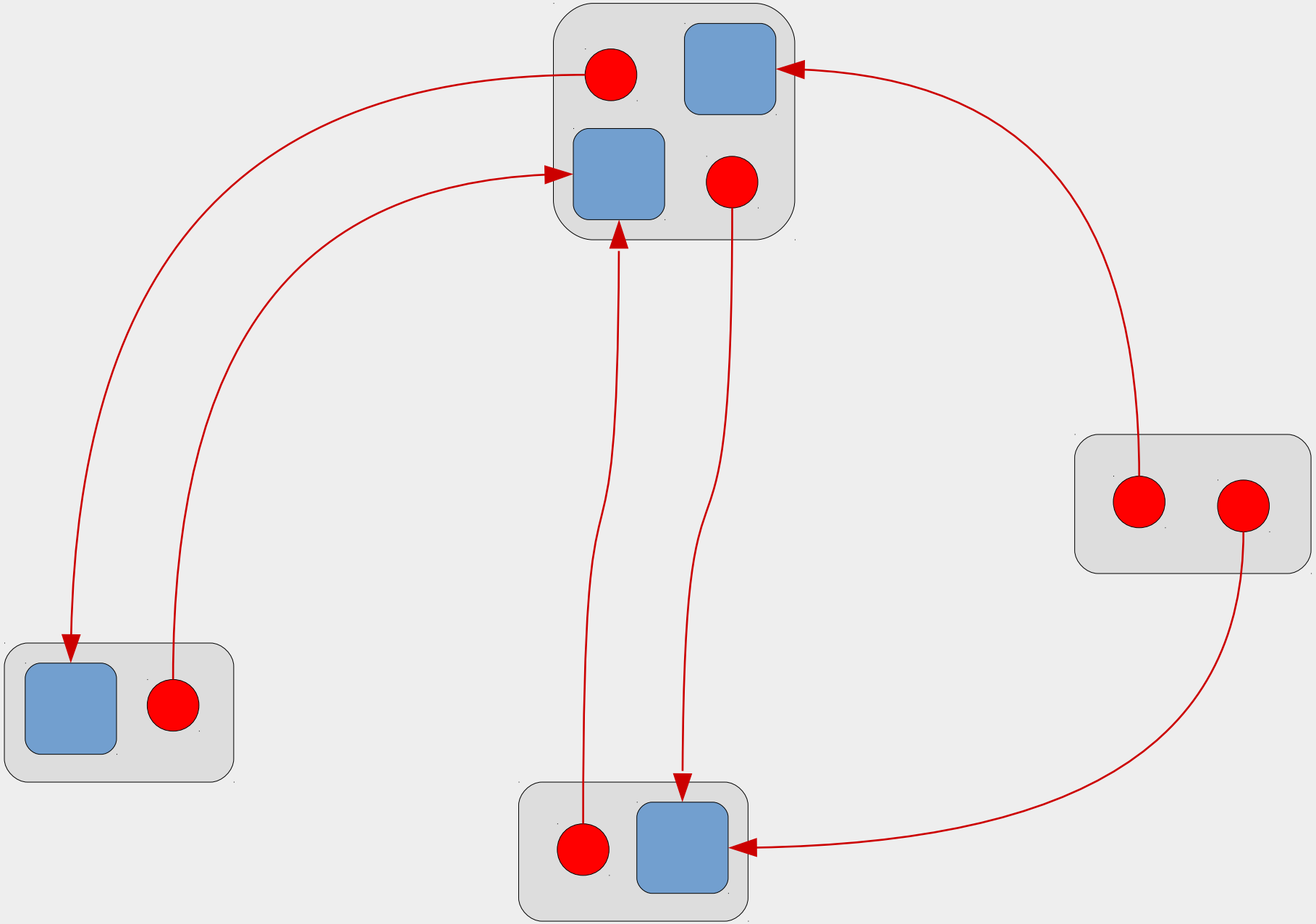


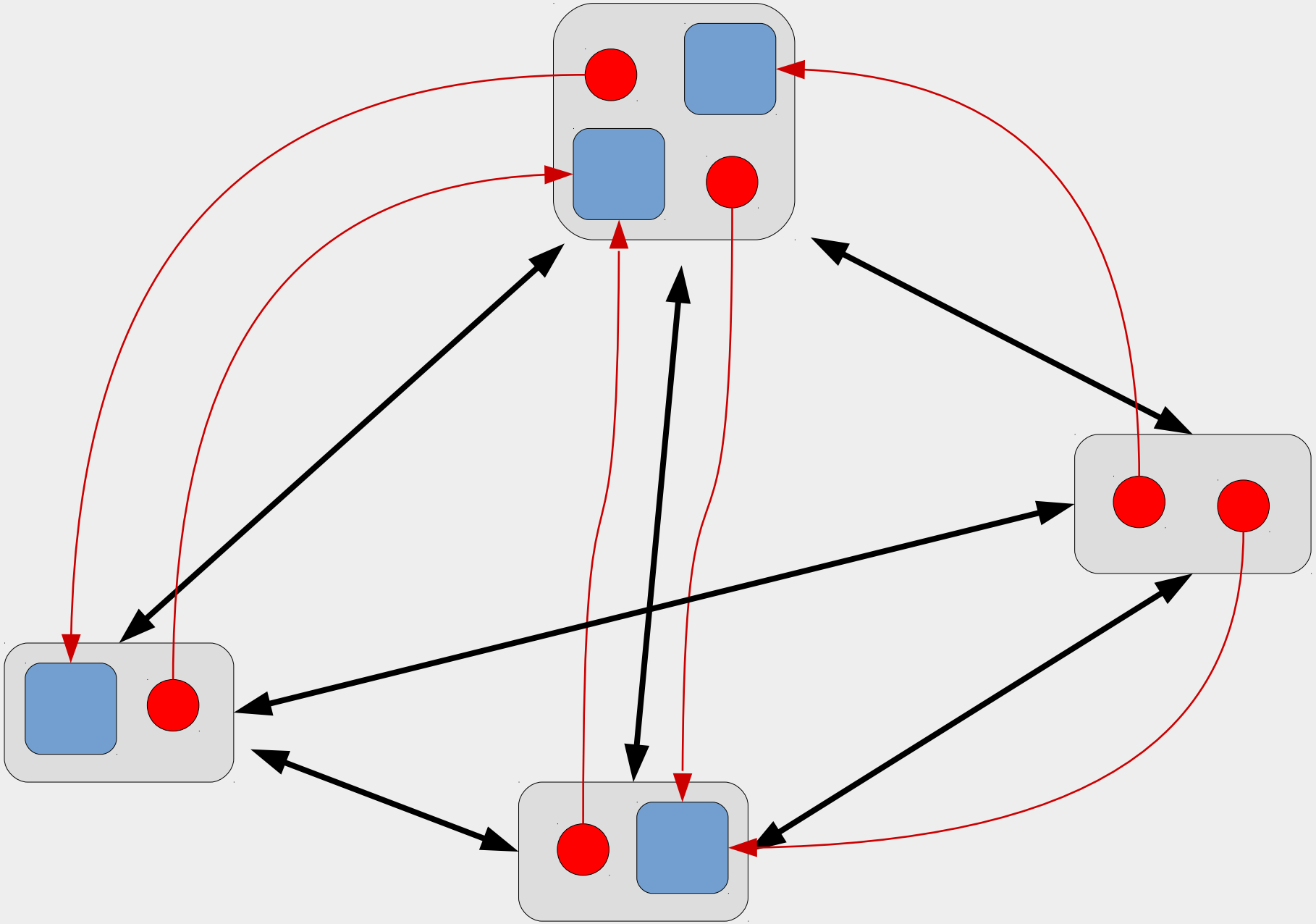


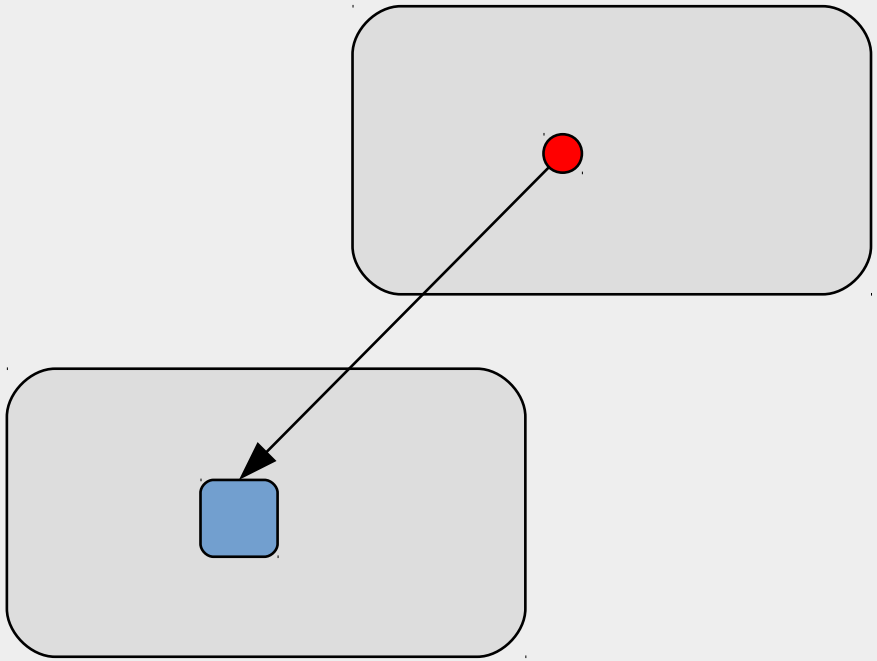


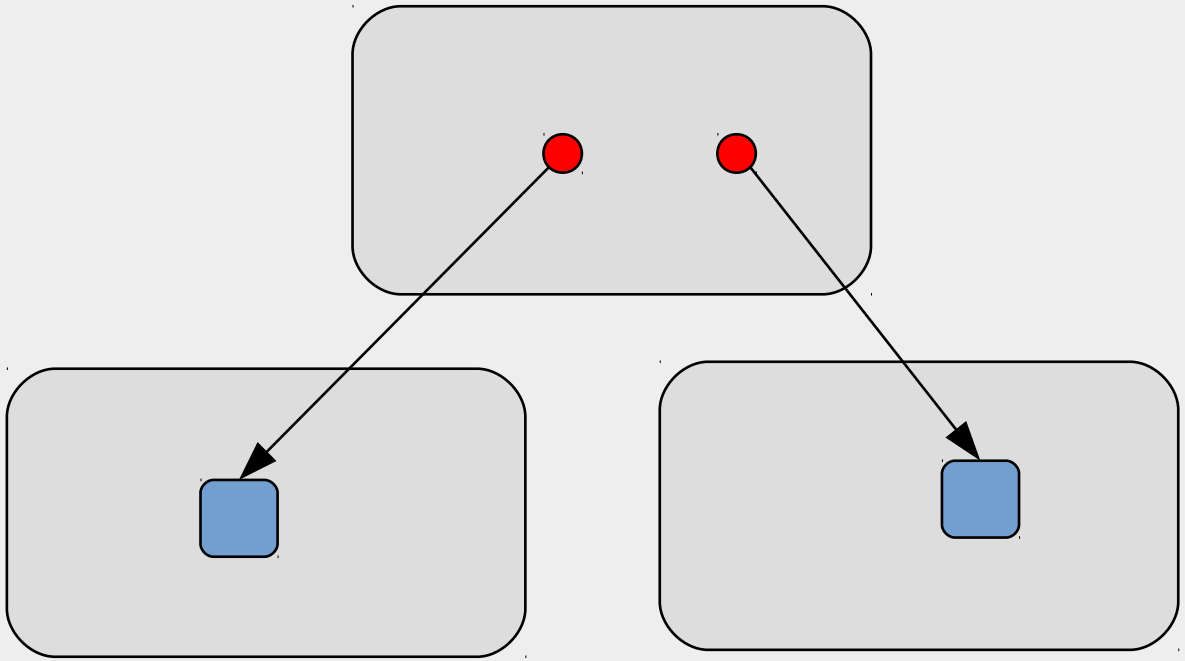


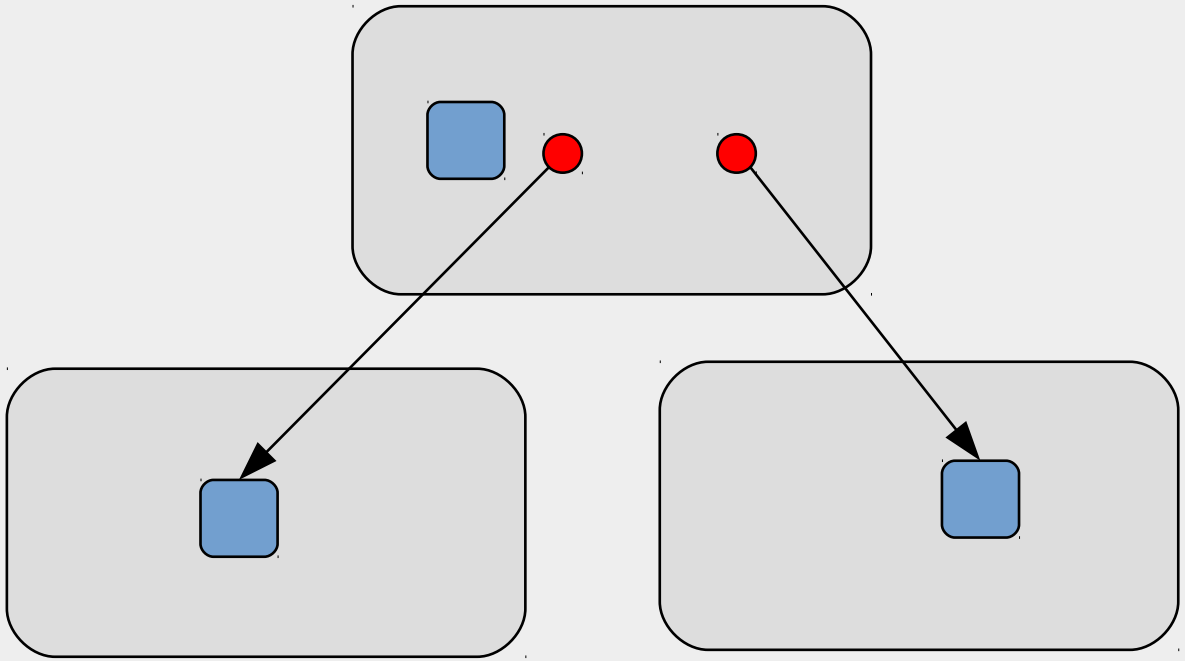


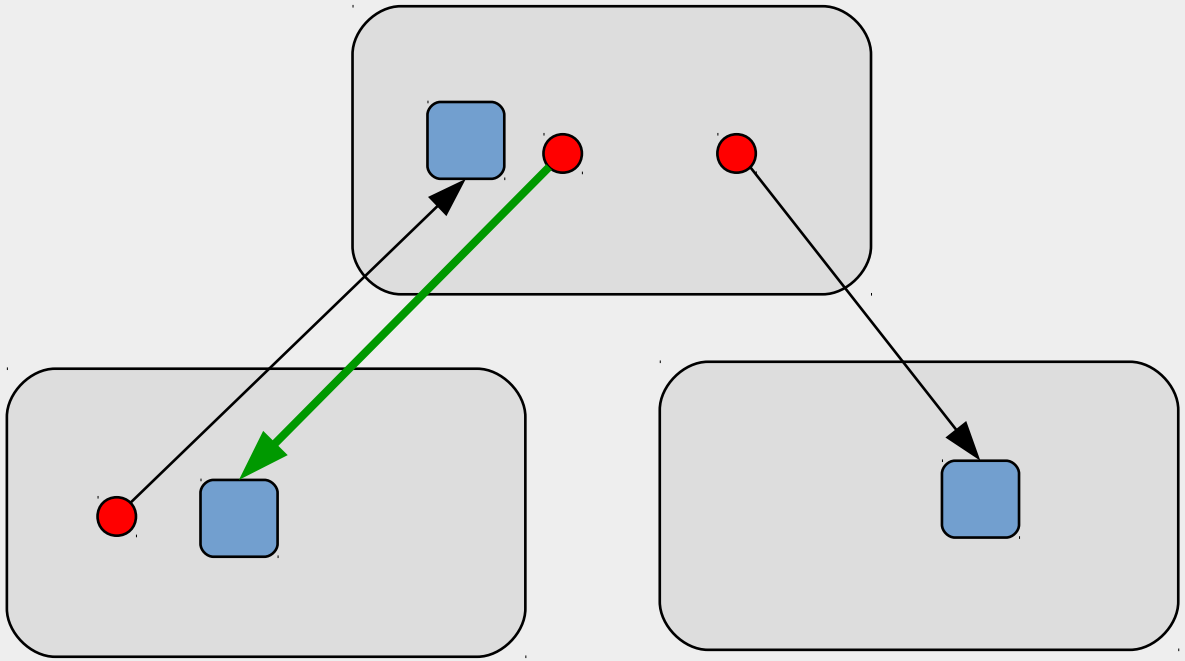


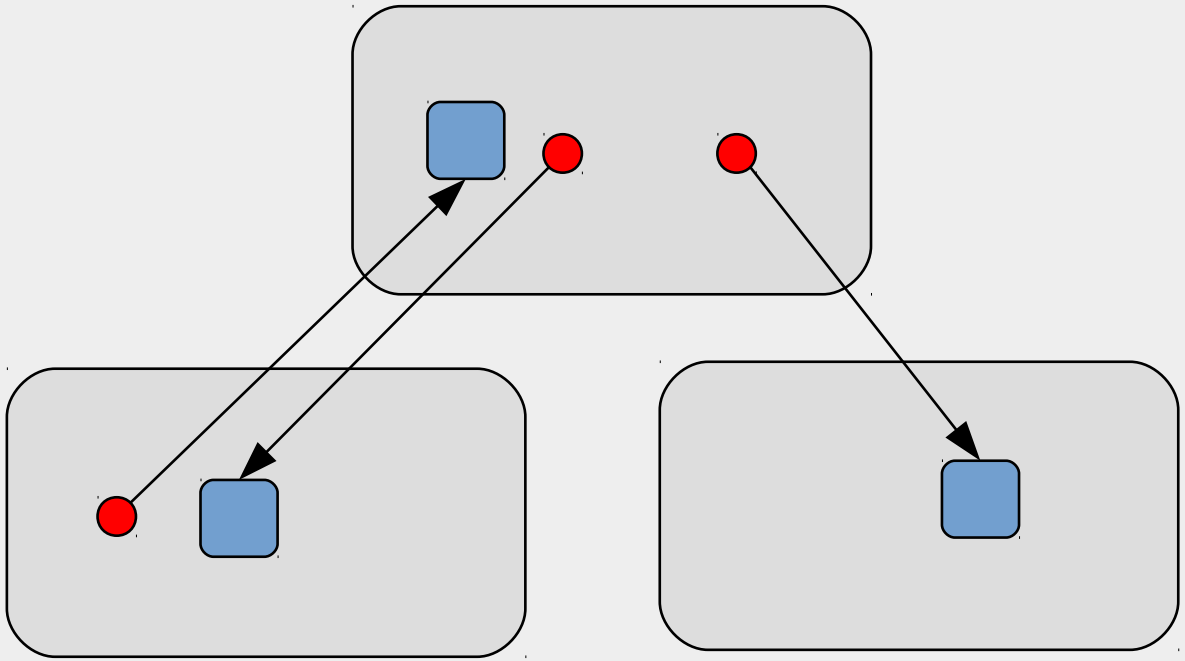


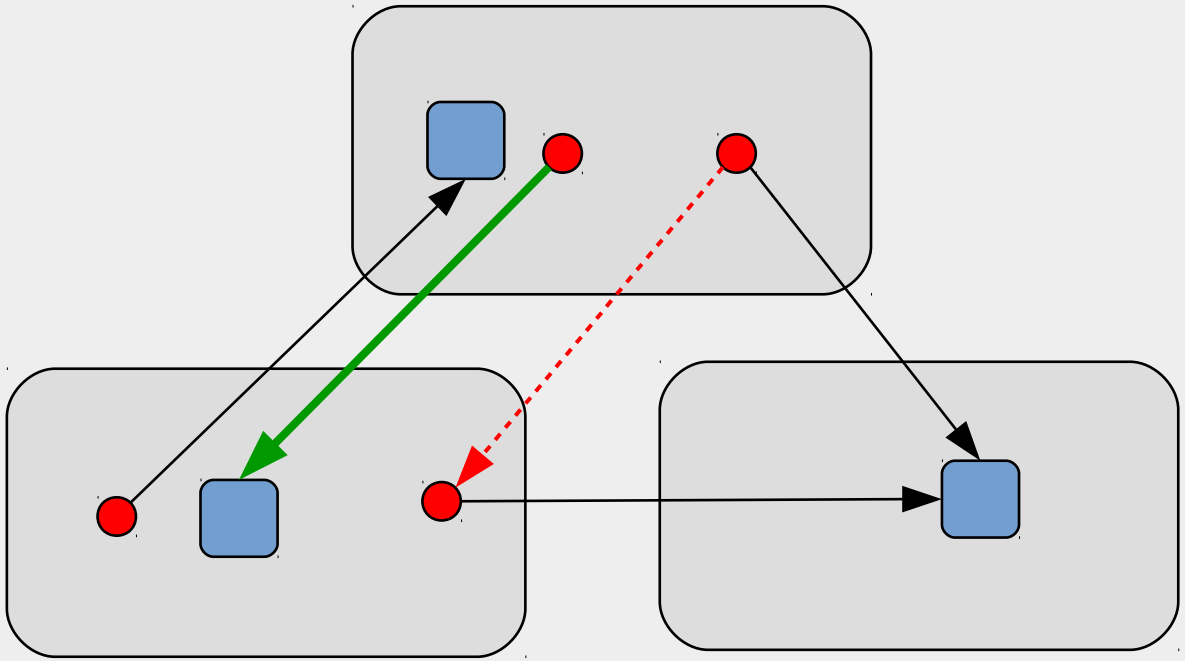


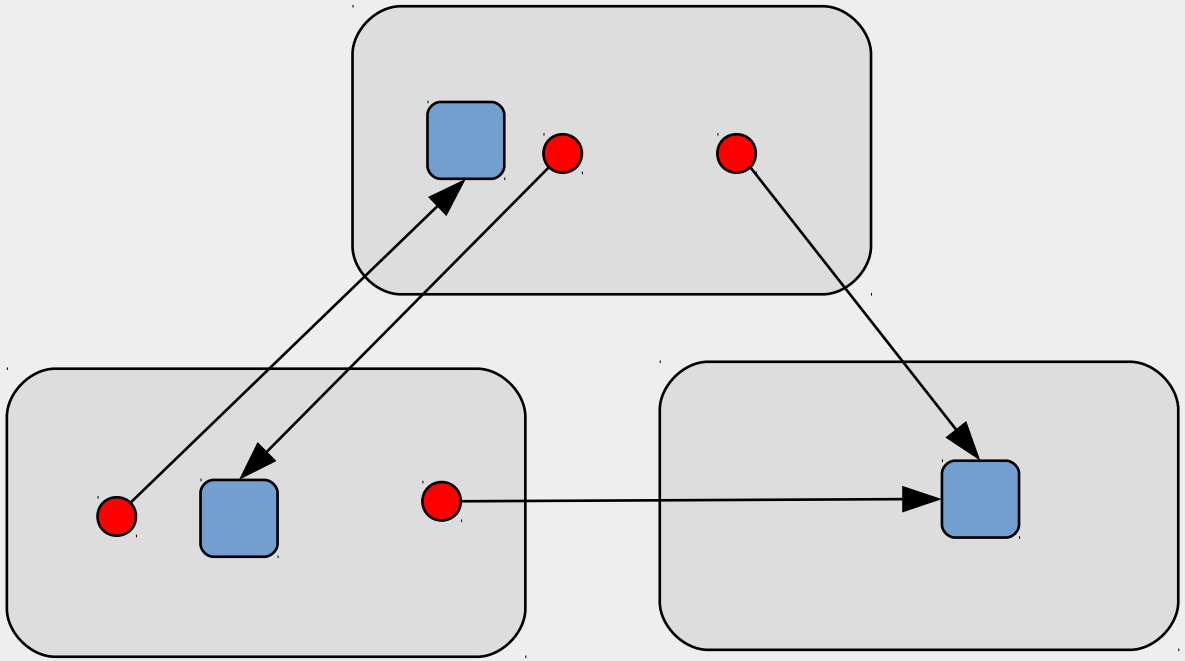


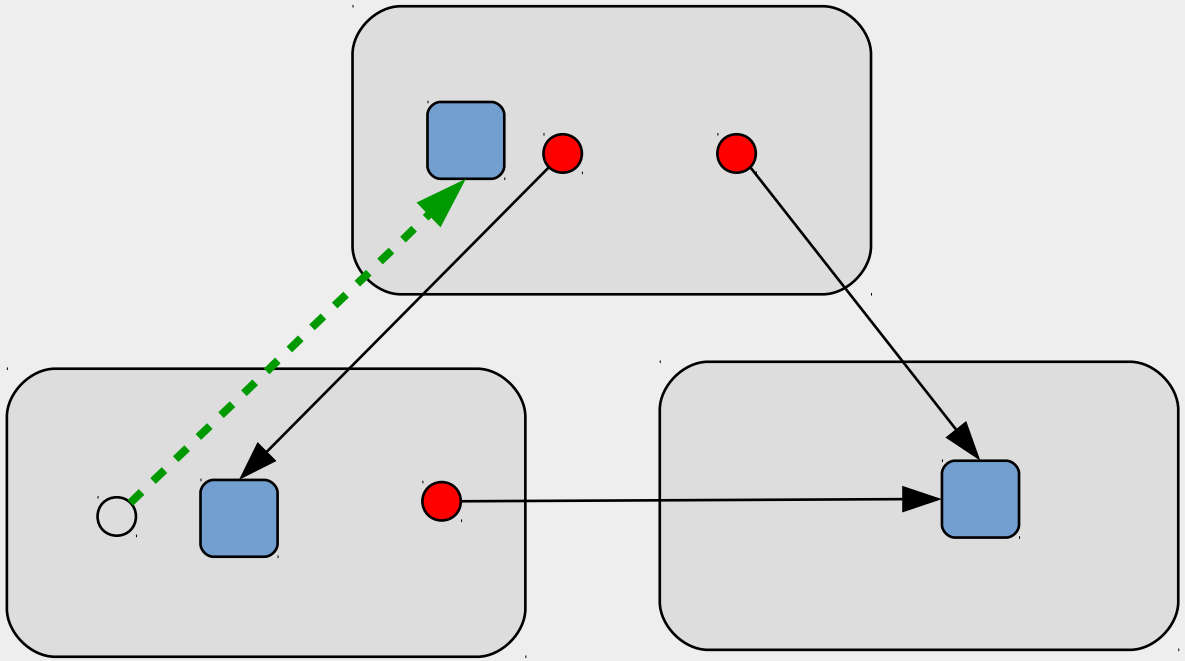


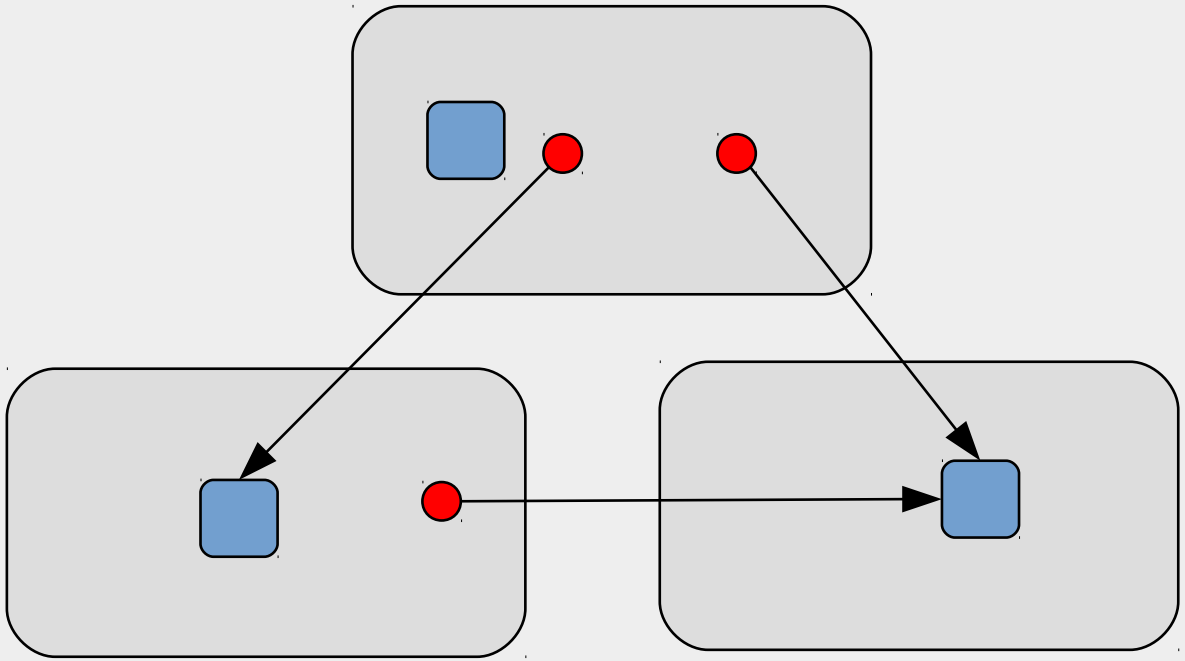


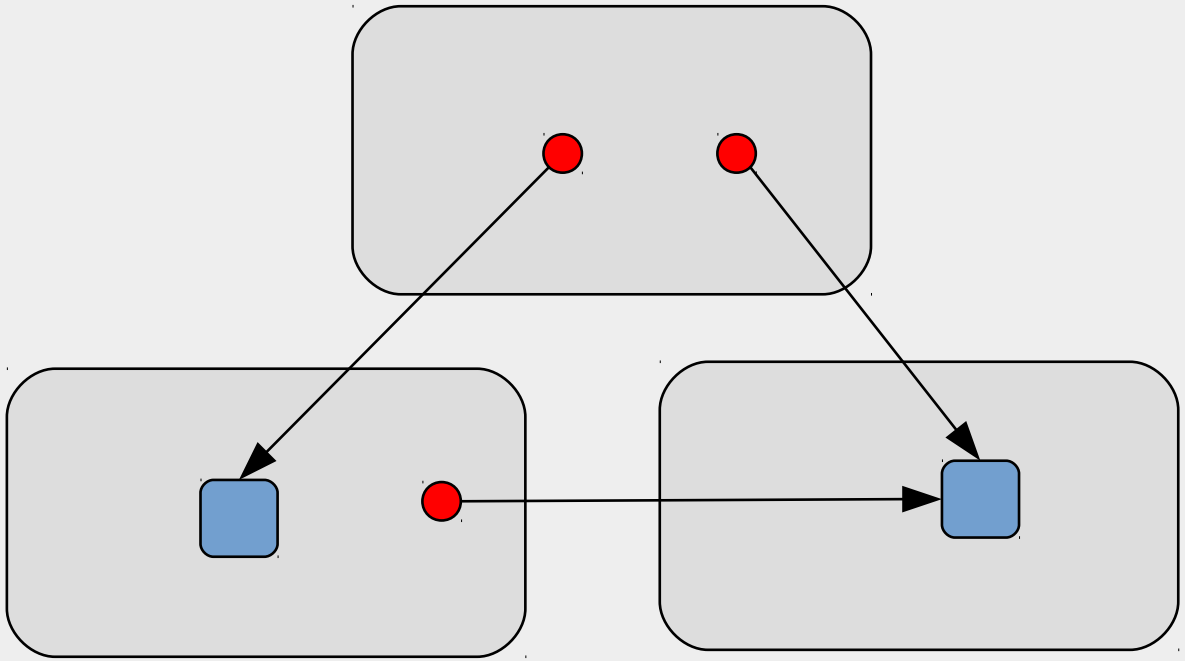


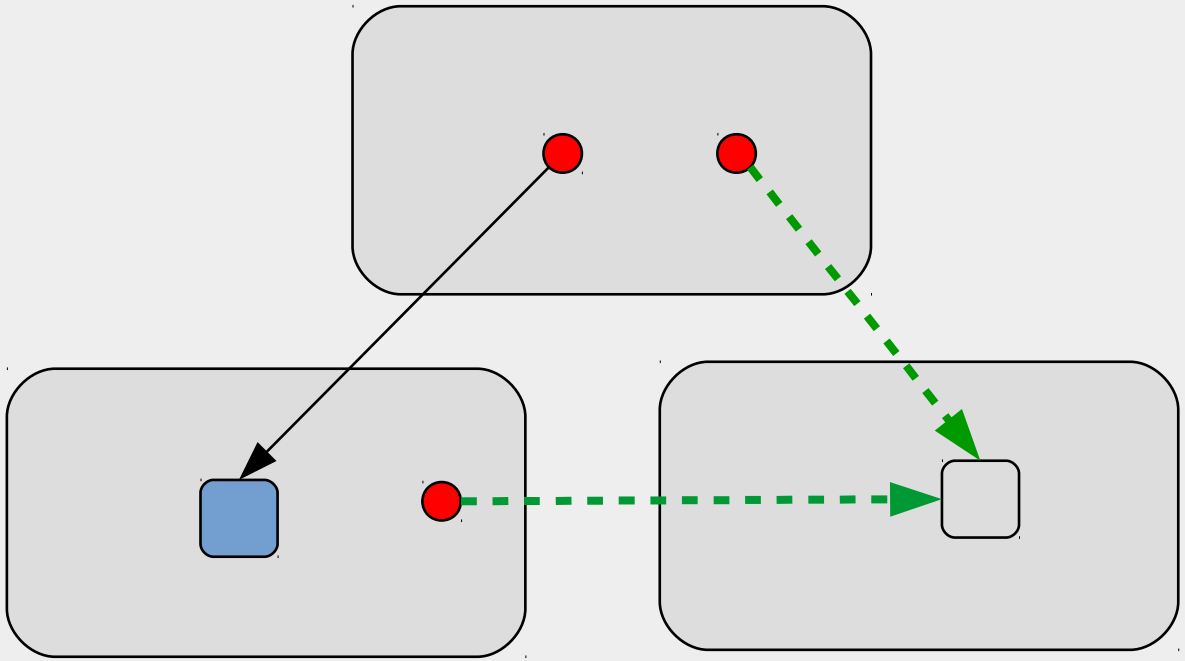


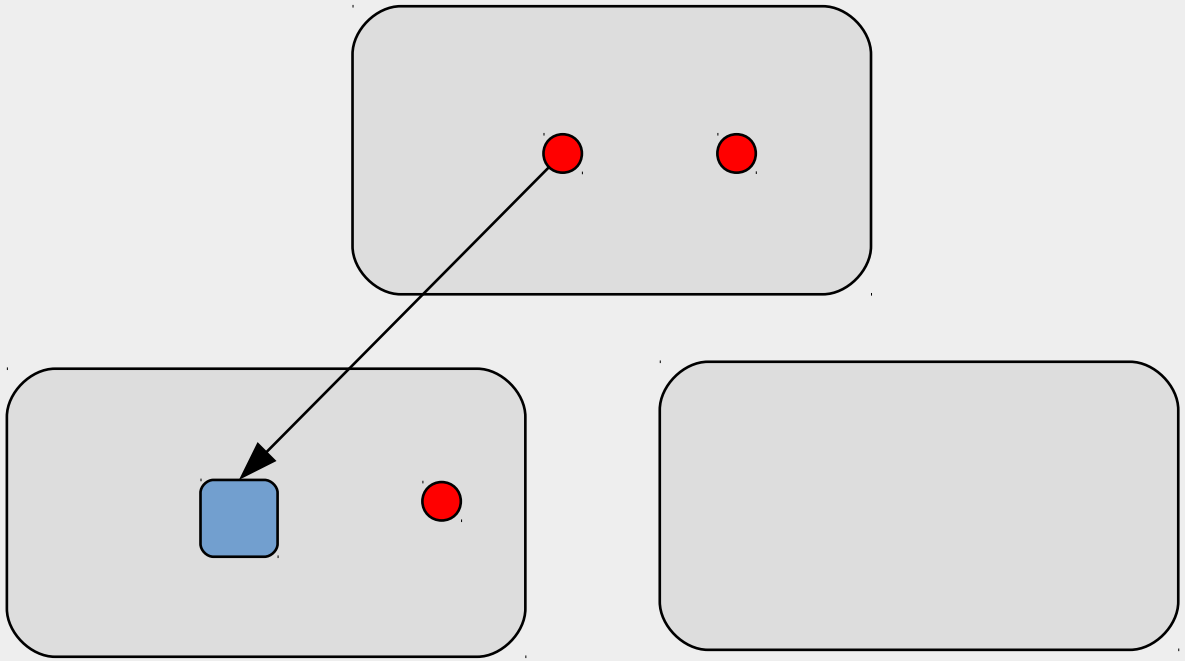


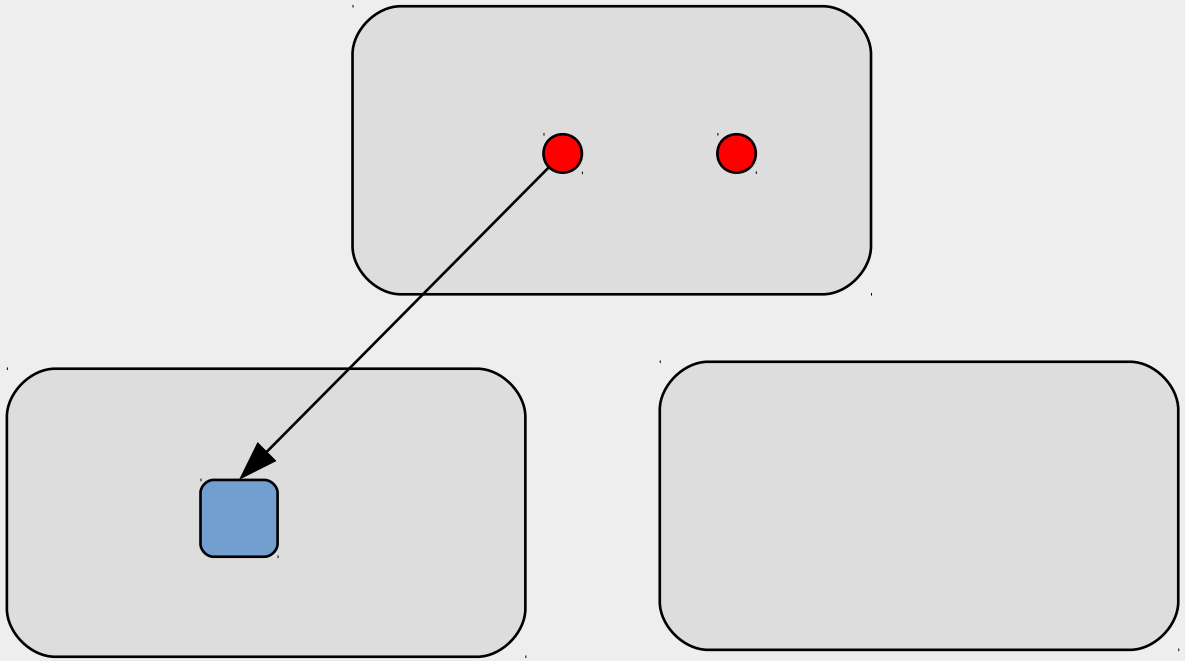


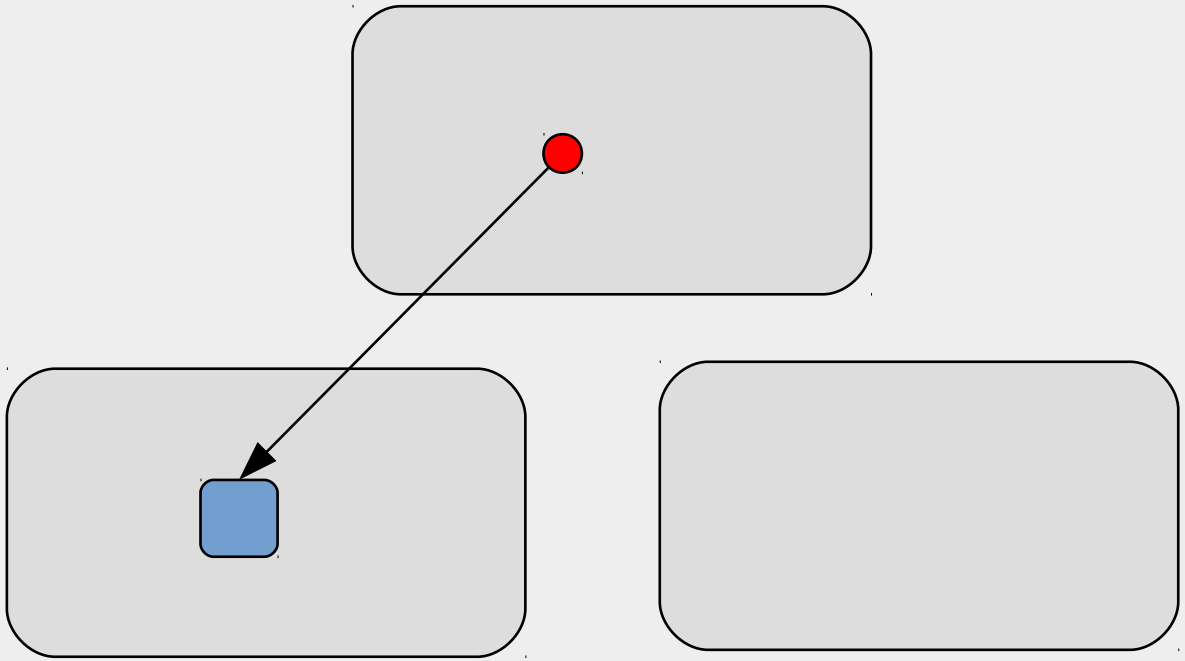








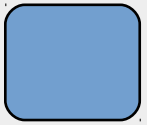




open(/dev/bus1)

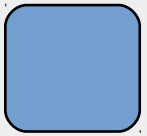
open(/dev/bus1)

...XXXXXXX0



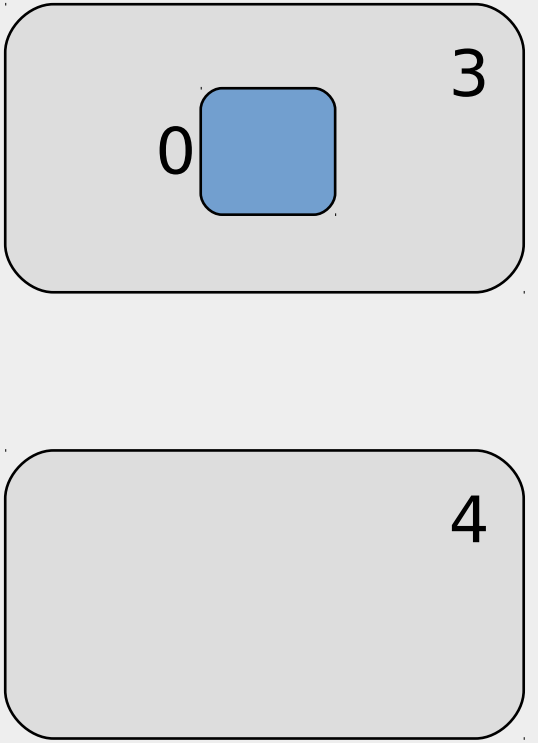
open(/dev/bus1)

...XXXXXXX0

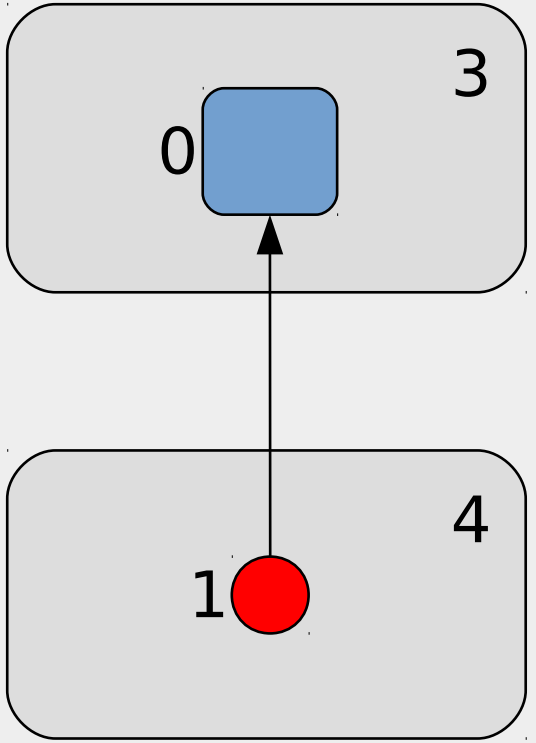


...XXXXXXX1

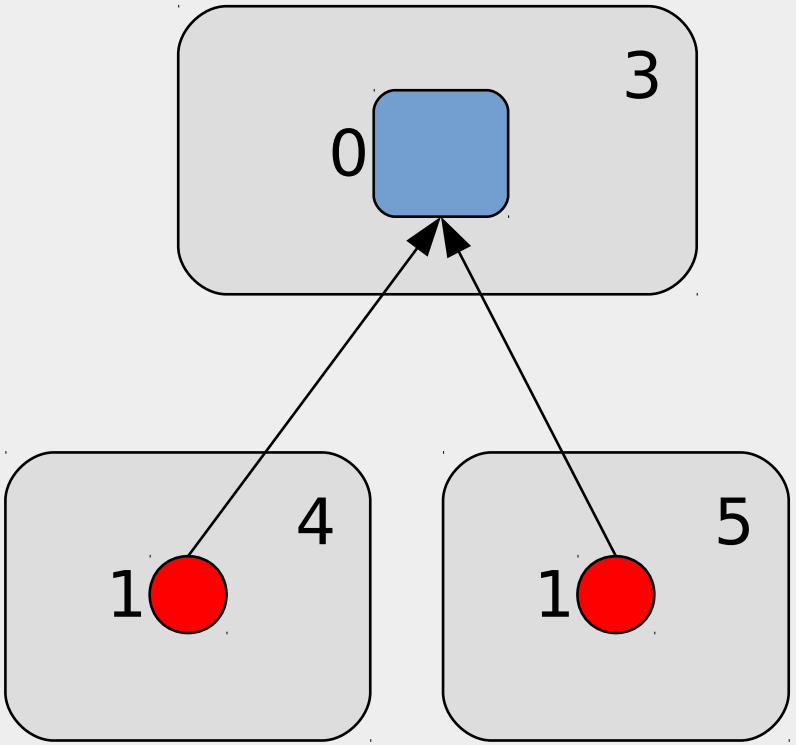
```
struct bus1_cmd_handle_transfer {
  __u64 src_handle;
  __u64 dst_fd;
  __u64 dst_handle;
} cmd = {
  .src_handle = 0,
  .dst_fd = 4,
  .dst_handle = HANDLE_INVALID,
};
```

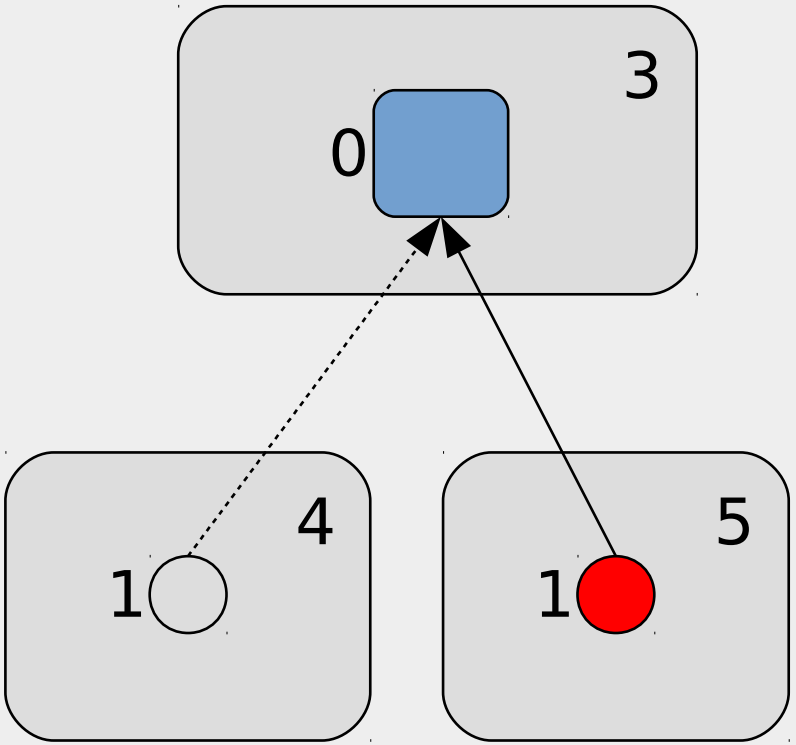


```
struct bus1_cmd_handle_transfer {
  __u64 src_handle;
  __u64 dst_fd;
  __u64 dst_handle;
} cmd = {
  .src_handle = 0,
  .dst_fd = 4,
  .dst_handle = 1,
};
```

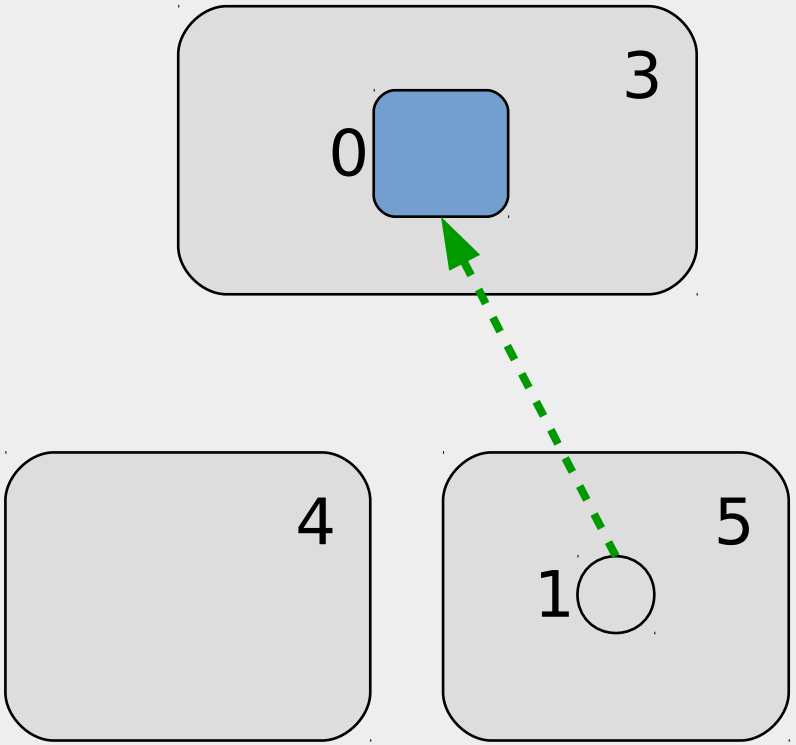


ioctl(3, BUS1_CMD_HANDLE_TRANSFER, &cmd);

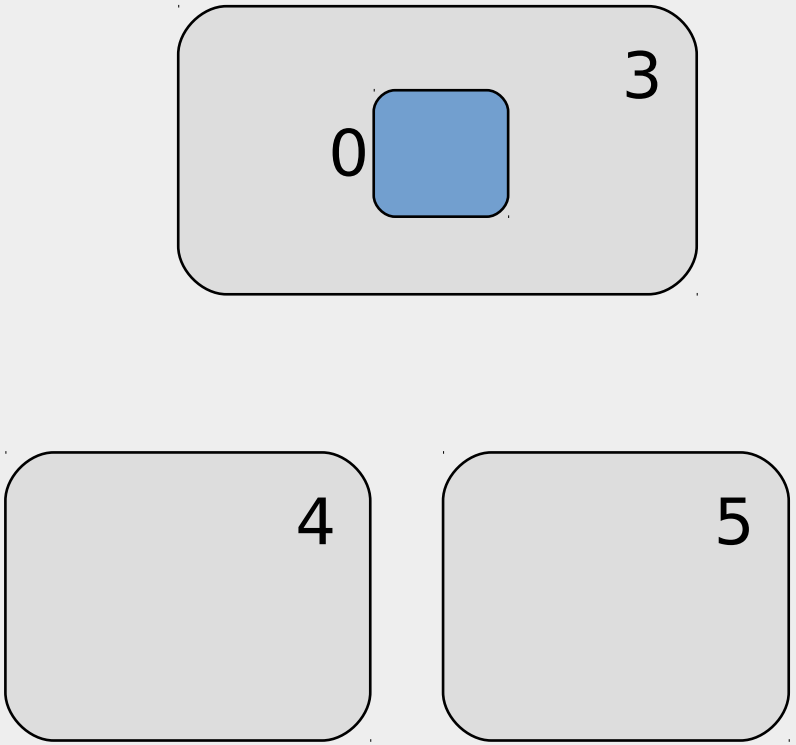

3
0
4
1
5
1



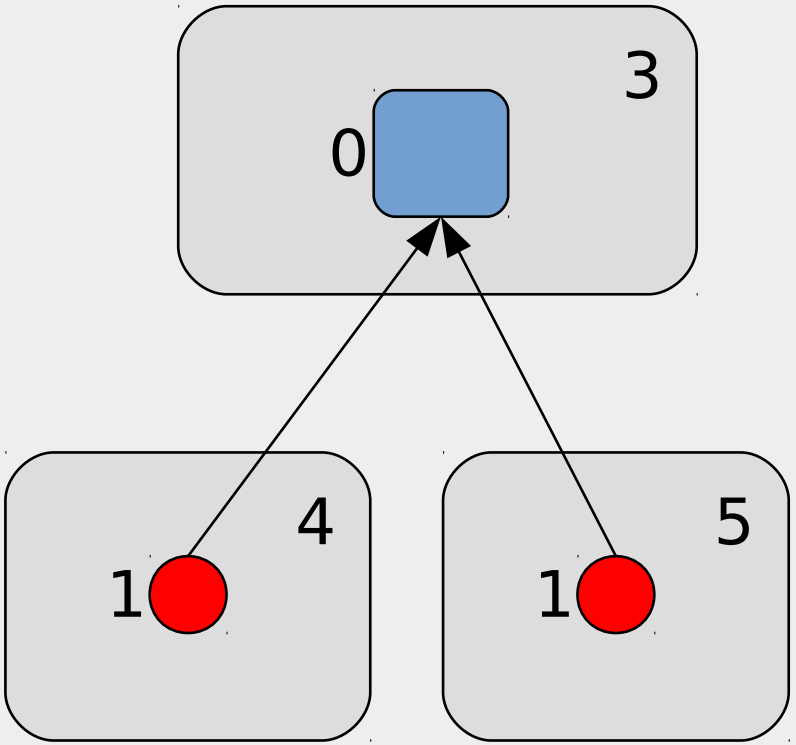
ioctl(4, BUS1_CMD_HANDLE_RELEASE, 1);

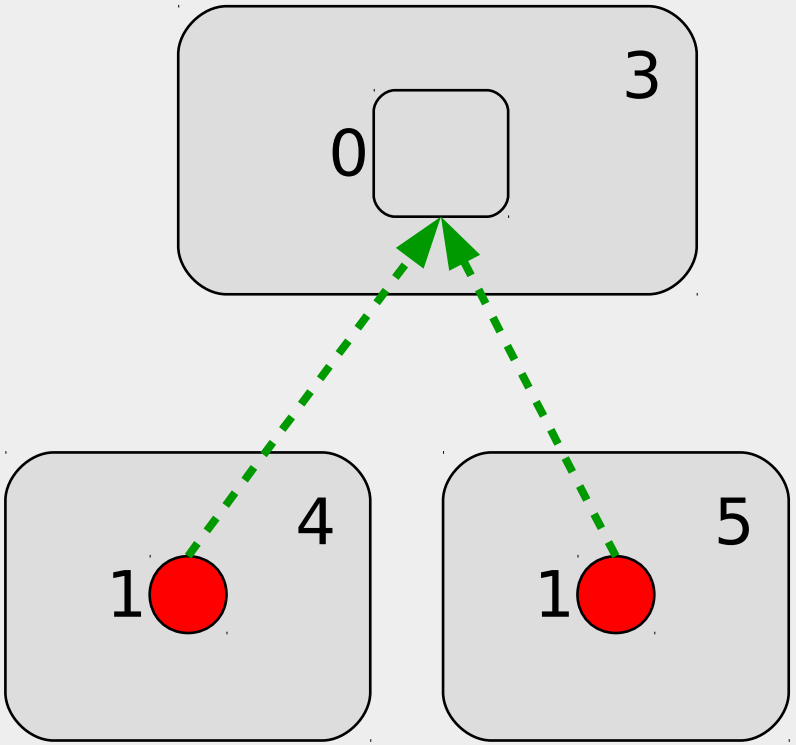


ioctl(5, BUS1_CMD_HANDLE_RELEASE, 1);

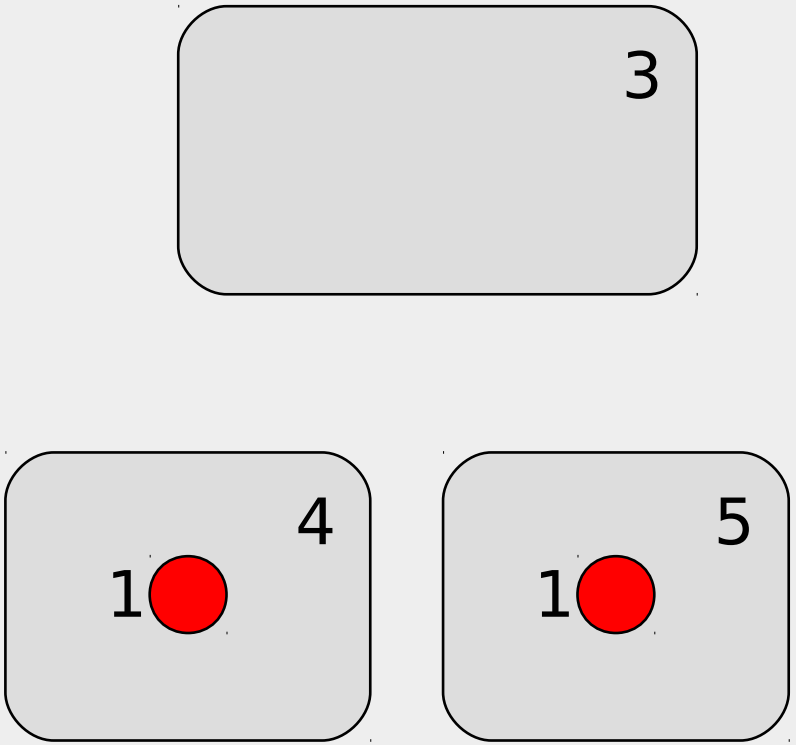


ioctl(5, BUS1_CMD_HANDLE_RELEASE, 1);


3
0
4
1
5
1

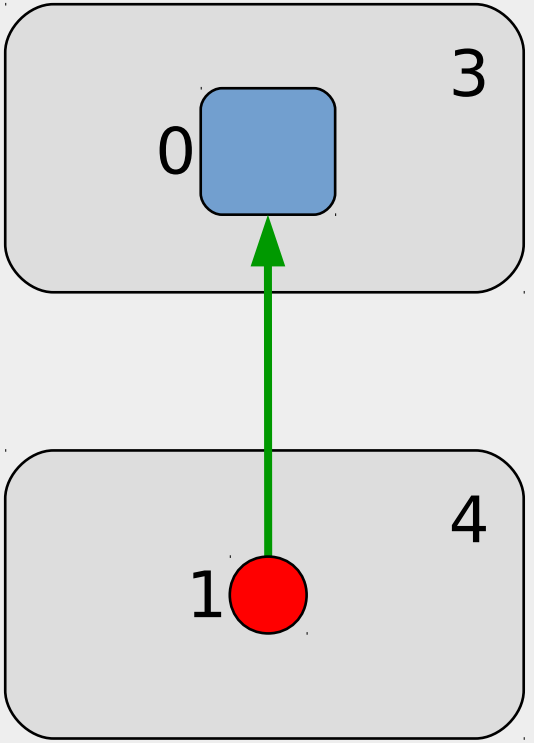


ioctl(3, BUS1_CMD_NODE_DESTROY, 0);



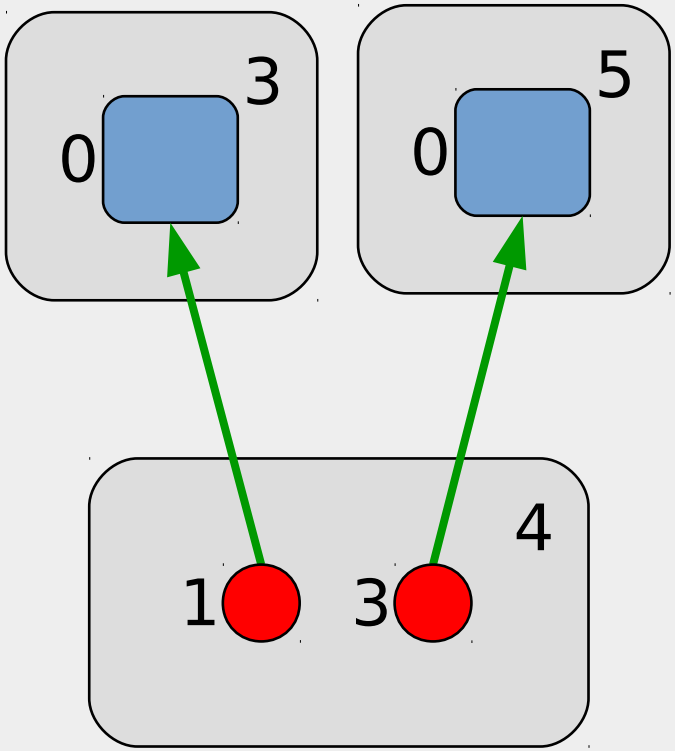
ioctl(3, BUS1_CMD_NODE_DESTROY, 0);

```
struct bus1_cmd_send {
  __u64 destination;
  __u64 ptr_data;
  __u64 n_bytes;
} cmd = {
  .destination = 1,
  .ptr_data = “foobar”,
  .n_bytes = strlen(“foobar”),
};
```



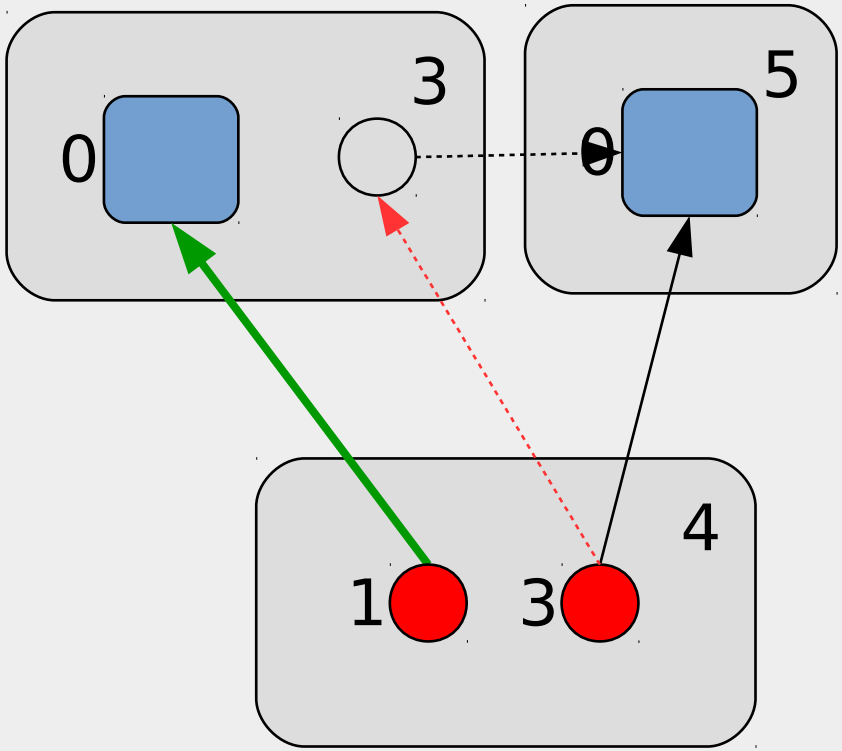
ioctl(4, BUS1_CMD_SEND, &cmd);

```
struct bus1_cmd_send {
  __u64 ptr_destinations;
  __u64 n_destinations;
  __u64 ptr_data;
  __u64 n_bytes;
} cmd = {
  .ptr_destinations = { 1, 3},
  .n_destinations = 2,
  .ptr_data = “foobar”,
  .n_bytes = strlen(“foobar”),
};
```



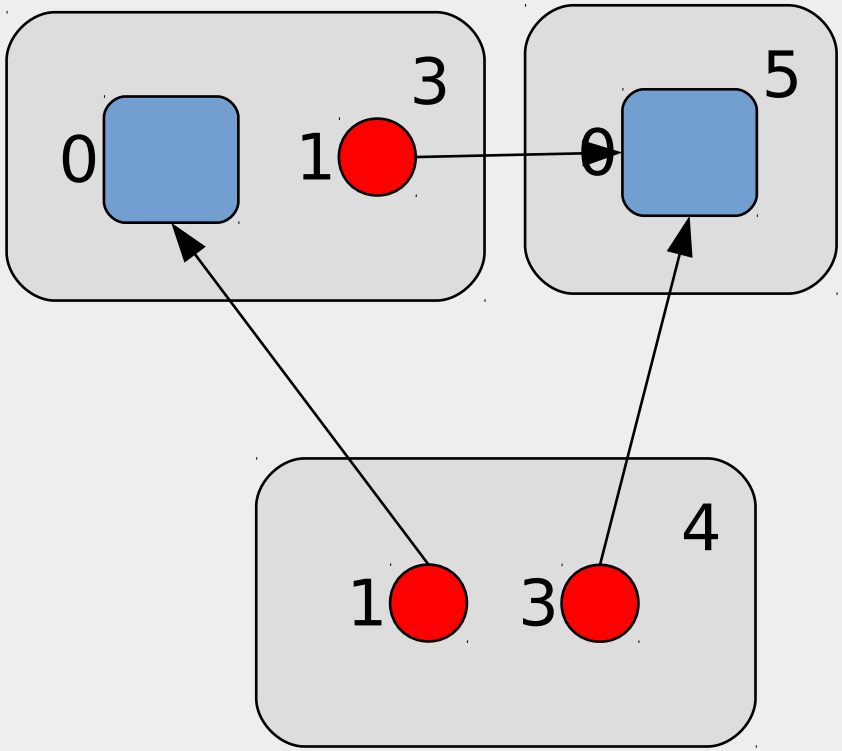
ioctl(4, BUS1_CMD_SEND, &cmd);

```
struct bus1_cmd_send {
  __u64 ptr_destinations;
  __u64 n_destinations;
  __u64 ptr_handles;
  __u64 n_handles;
} cmd = {
  .ptr_destinations = { 1 },
  .n_destinations = 1,
  .ptr_handles = { 3 },
  .n_handles = 1,
};
```



ioctl(4, BUS1_CMD_SEND, &cmd);

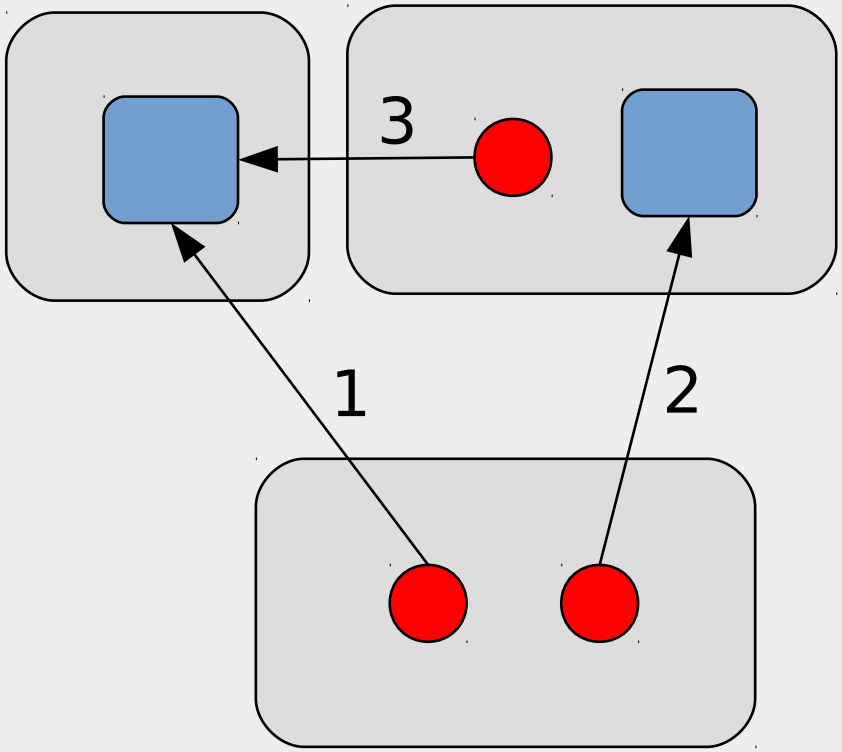
```
struct bus1_cmd_recv {
  __u64 type;
  __u64 destination;
  __u64 offset;
  __u64 n_bytes;
  __u64 n_handles;
} cmd = {
  .type = BUS1_MSG_DATA,
  .destination = 0,
  .offset = 0,
  .n_bytes = 0,
  .n_handles = 1,
};
```

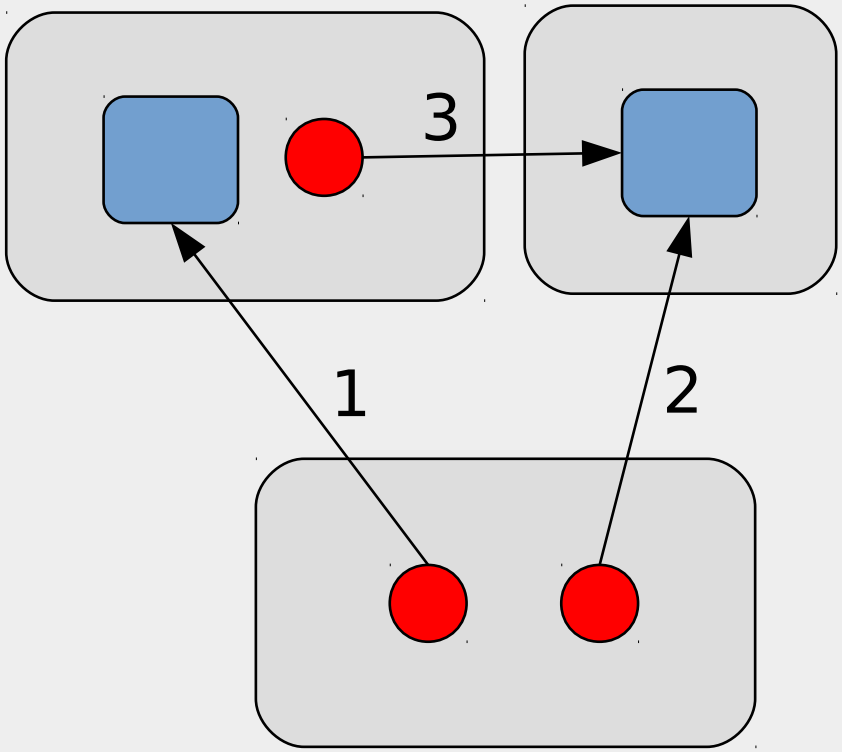


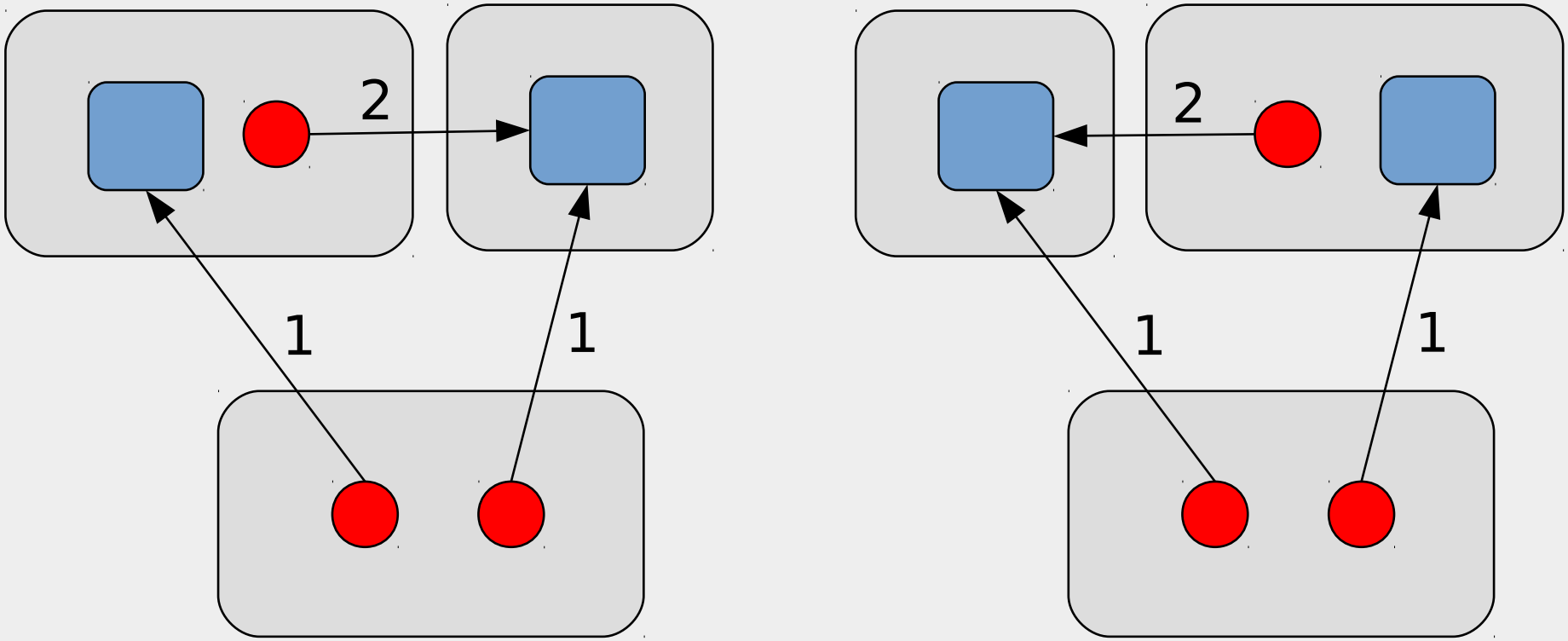
ioctl(3, BUS1_CMD_RECV, &cmd);

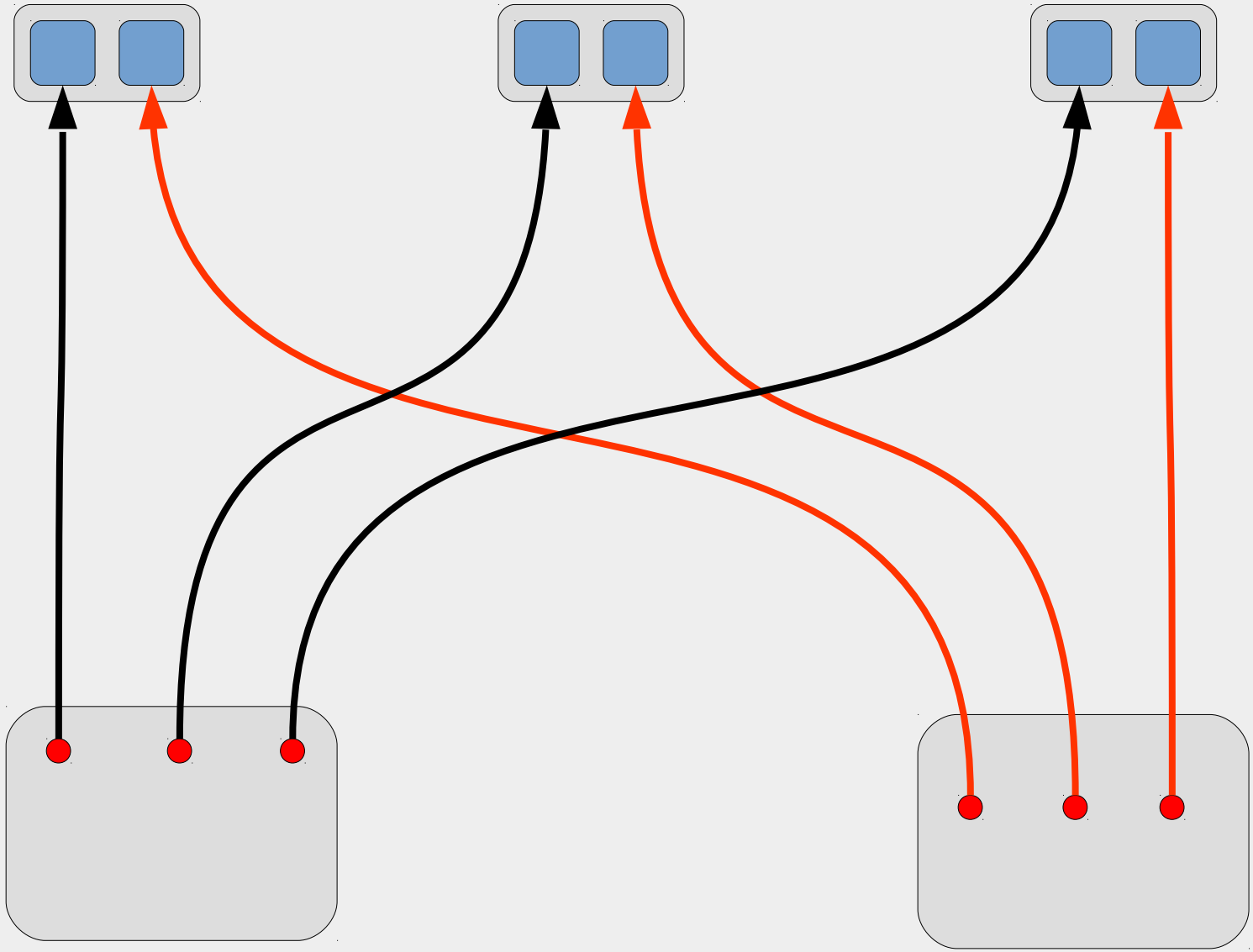
bus1 Features

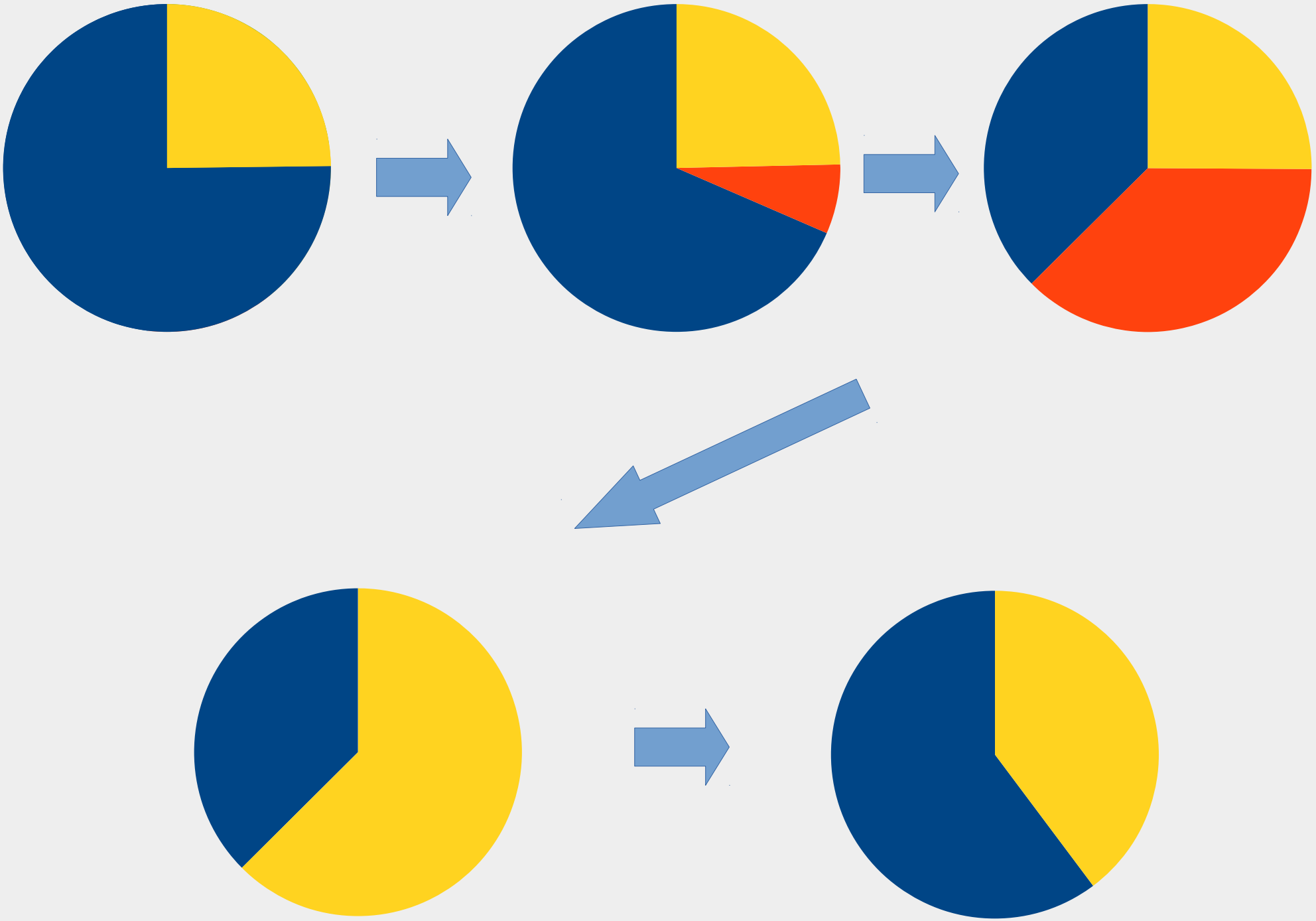
- Global Ordering
- Multicast
- Resource Accounting
- No Global Locking
- No Global Context

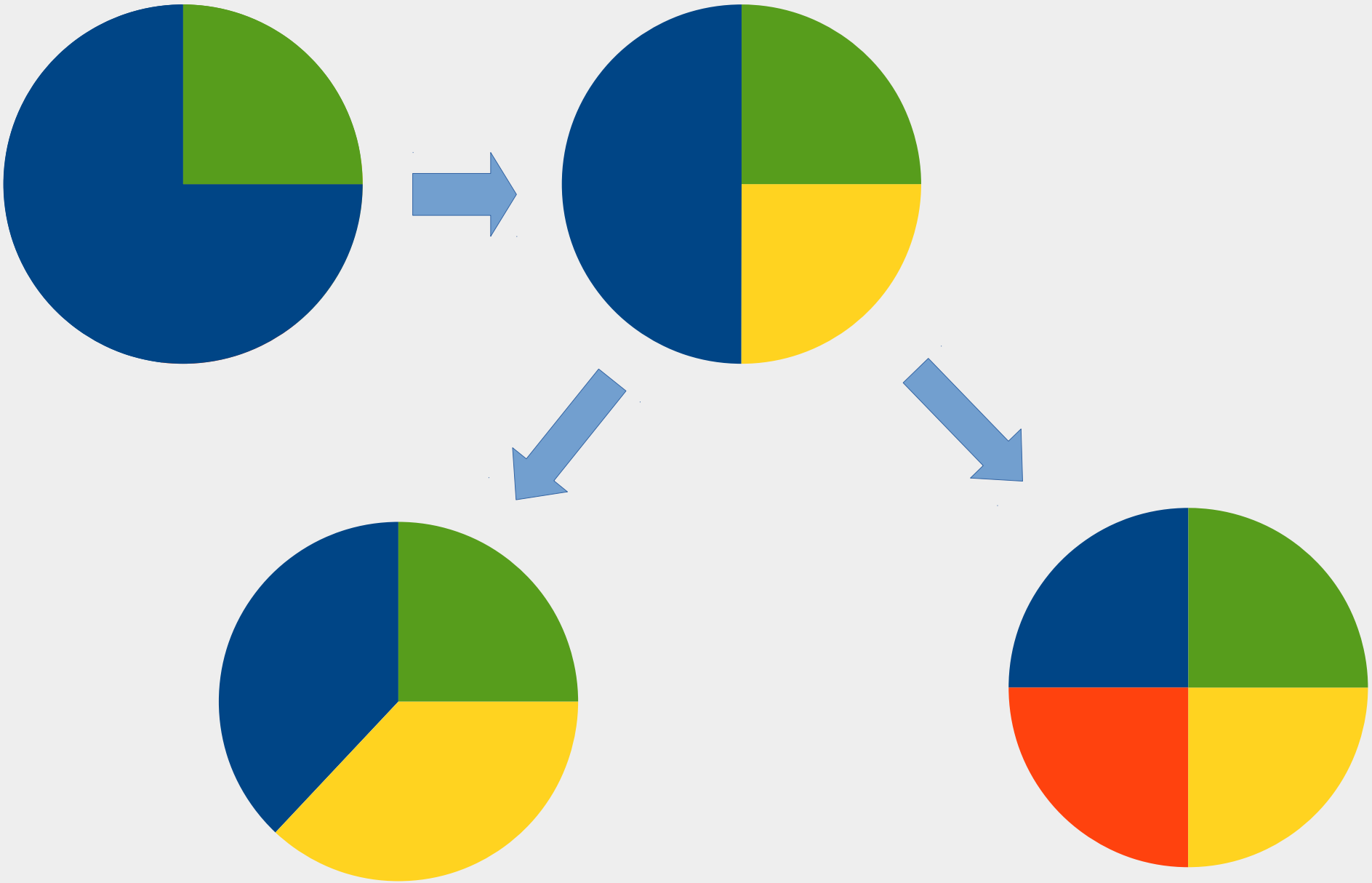

3
1
2


3
1
2


2
1
1
2
1
1







http://www.bus1.org

http://wiki.bus1.org