ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Getting Started with Barrelfish

Ihor Kuz

Outline

Getting Barrelfish

Publicly available tar-ball:

- http://www.barrelfish.org/release_20091219.html
- Outdated

Mercurial repository

- Not-public
- Possibly available
- What this talk targets

Toolchain

- GCC: > 4.1.3
- Haskell
  - GHC: 6.10 or 6.12.2 (Note: Ubuntu 10.04 has 6.12.1)
  - ghc-paths, parsec

Building and running Barrelfish

Default

```
$ ls
barrelfish
$ mkdir build
$ cd build
$ ../barrelfish/hake/hake.sh ../barrelfish/
OK, source directory is ../barrelfish/
...
$ make
...
$ vi menu.lst
...
$ make sim
...
spawnd.0: spawning /x86_64/sbin/serial on core 0
spawnd.0: spawning /x86_64/sbin/fish on core 0
fish v0.2 -- pleased to meet you!
Running help
available commands:
help          print_cspace   quit          ps            demo
percore       pixels         mnfs          oncore        reset
poweroff      skb            mount         ls            cd
pwd           touch          cat           cp            rm
mkdir         rmdir          setenv        printenv
>
```

Directory Structure

Source

- kernel
- include
- libraries
- usr
- if

Build

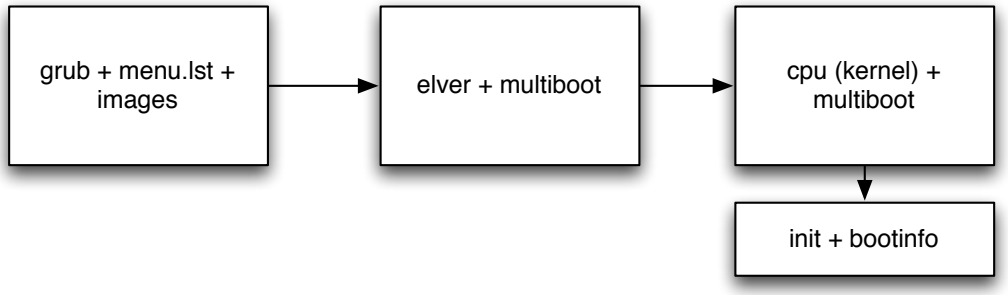
- Makefile
- symbolic_targets.mk
- menu.lst
- x86_64
  - include/if
  - usr/
  - sbin/

Bootstrap

core 0

core 1

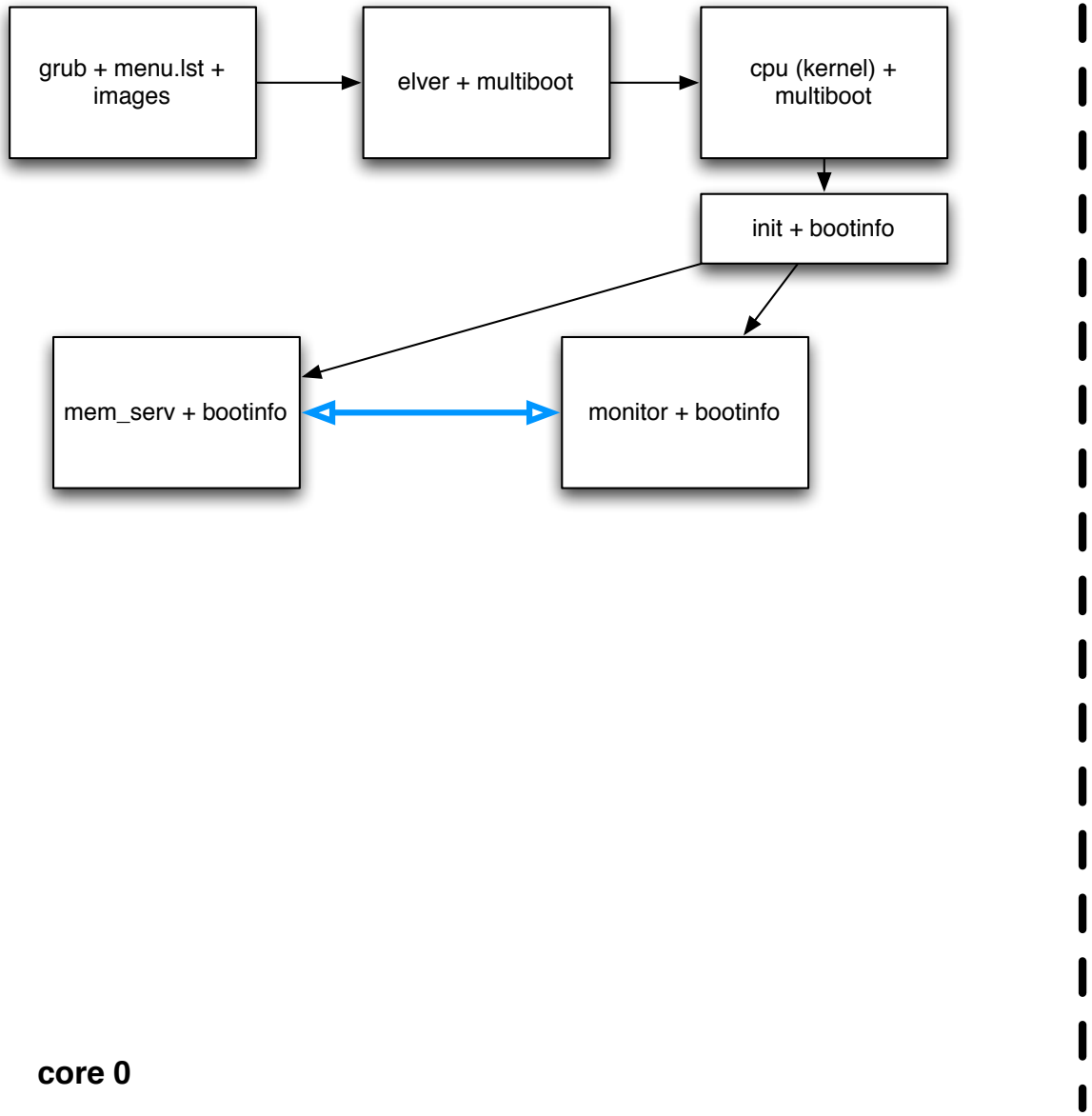
Bootstrap



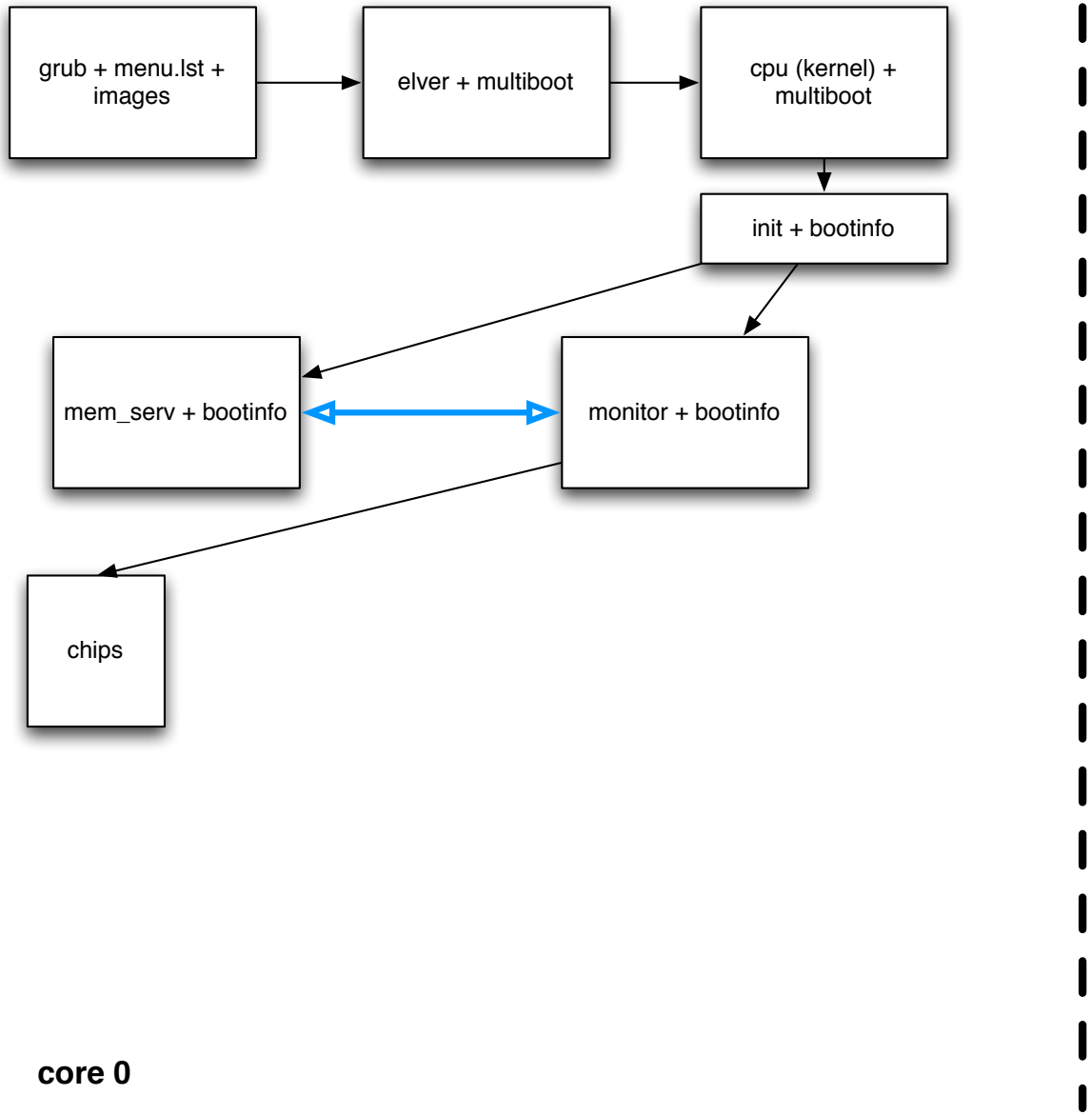
core 0

core 1

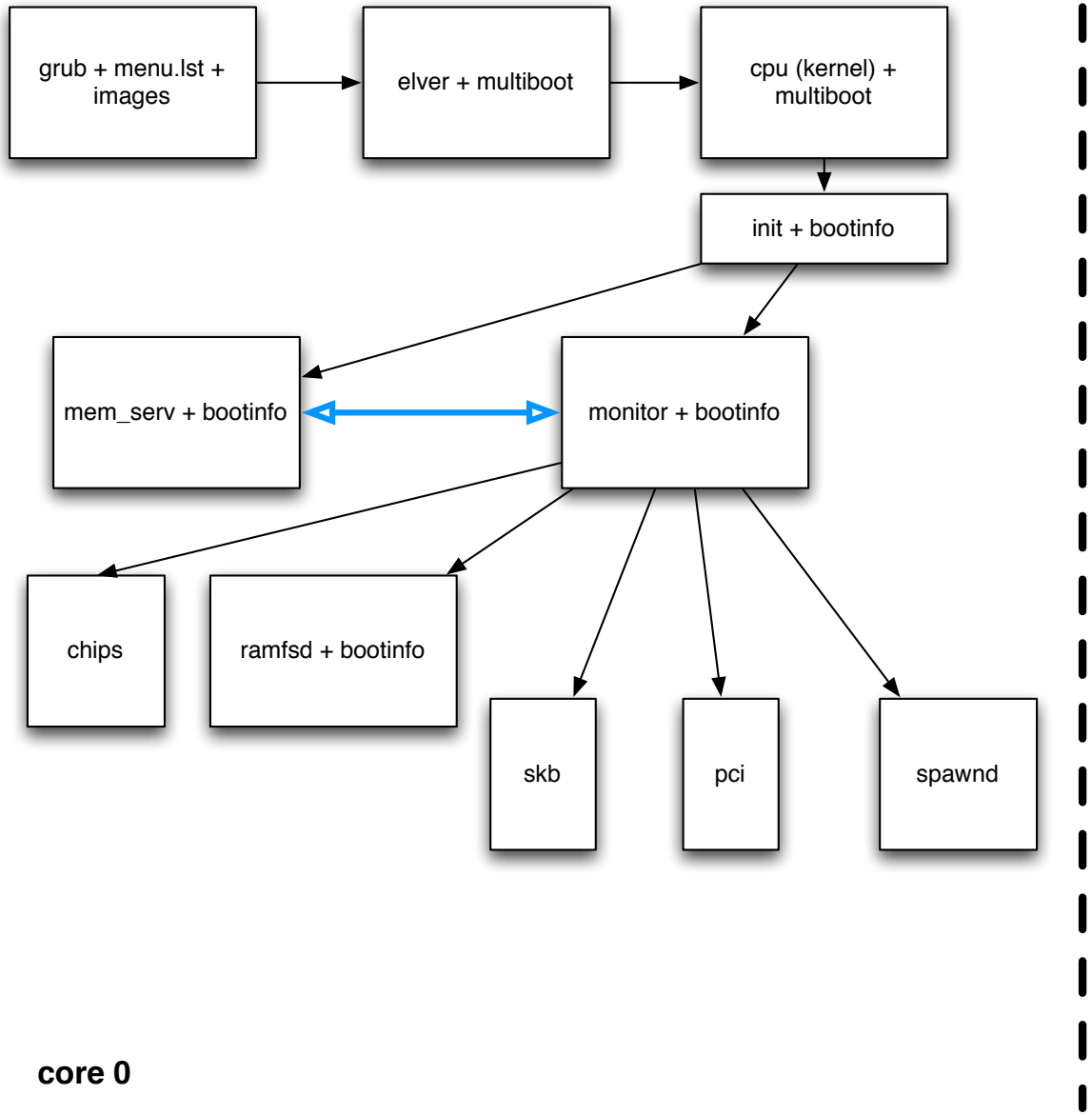
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
core 0
core 1

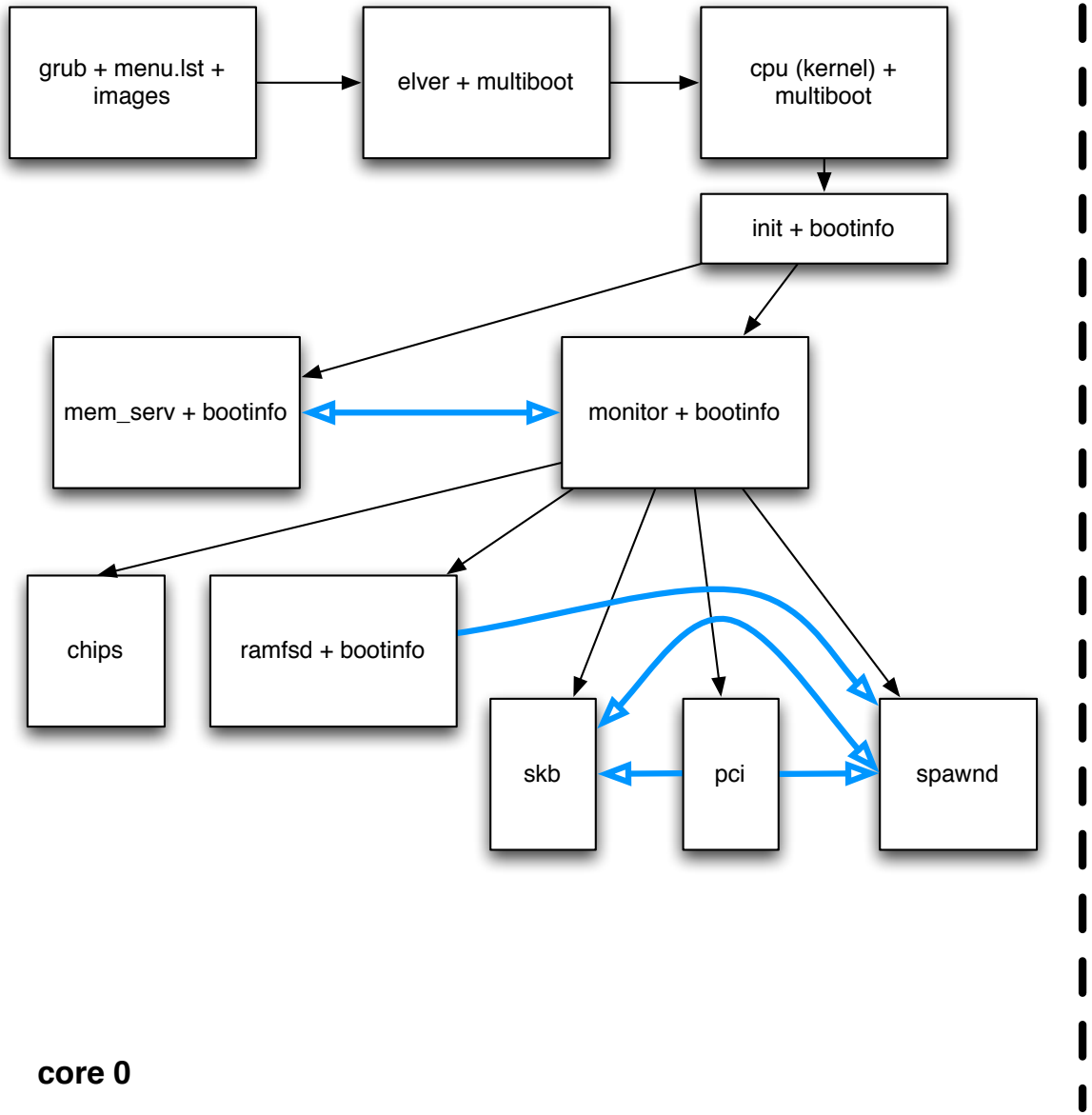
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
core 0
core 1

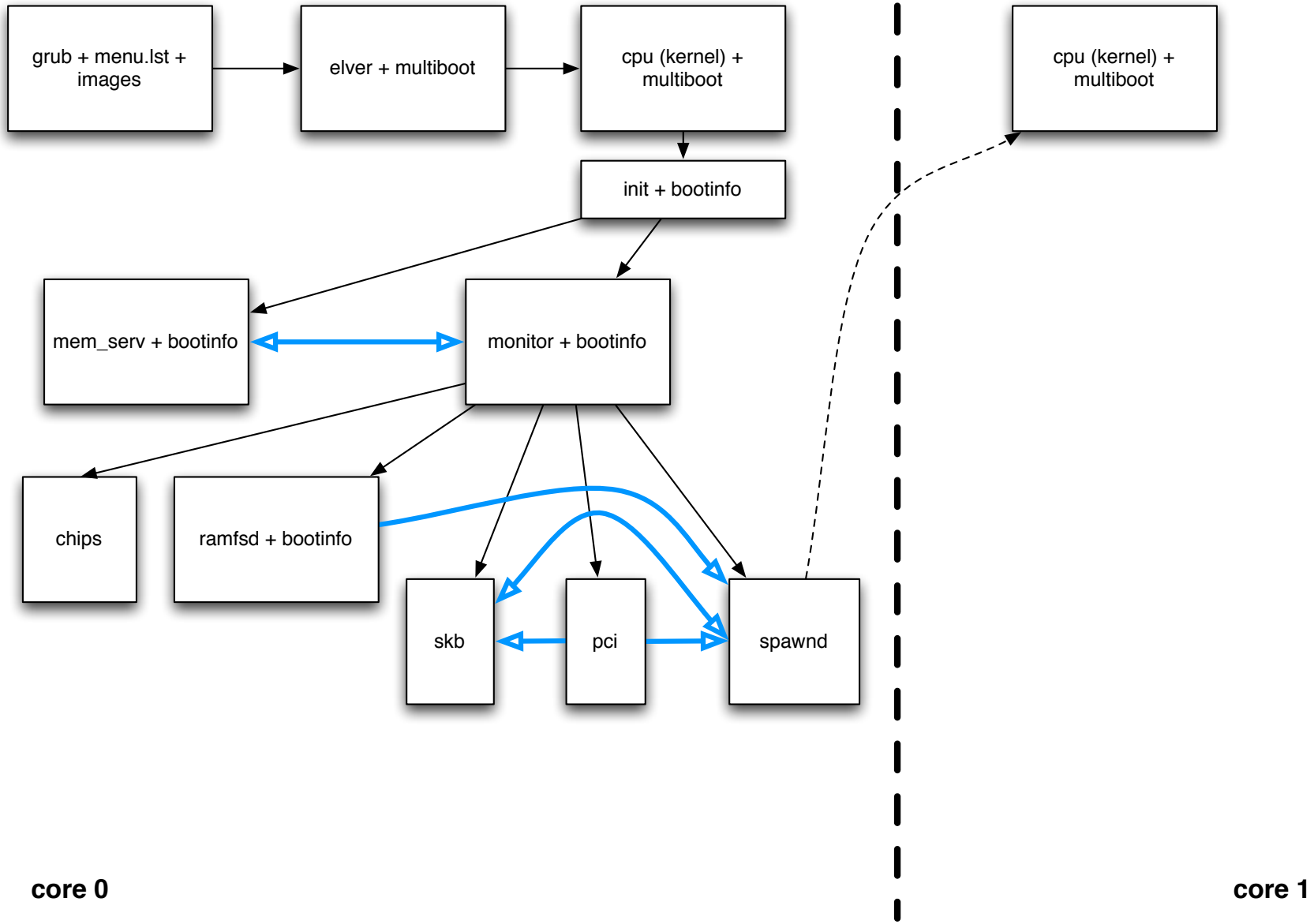
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
core 0
core 1

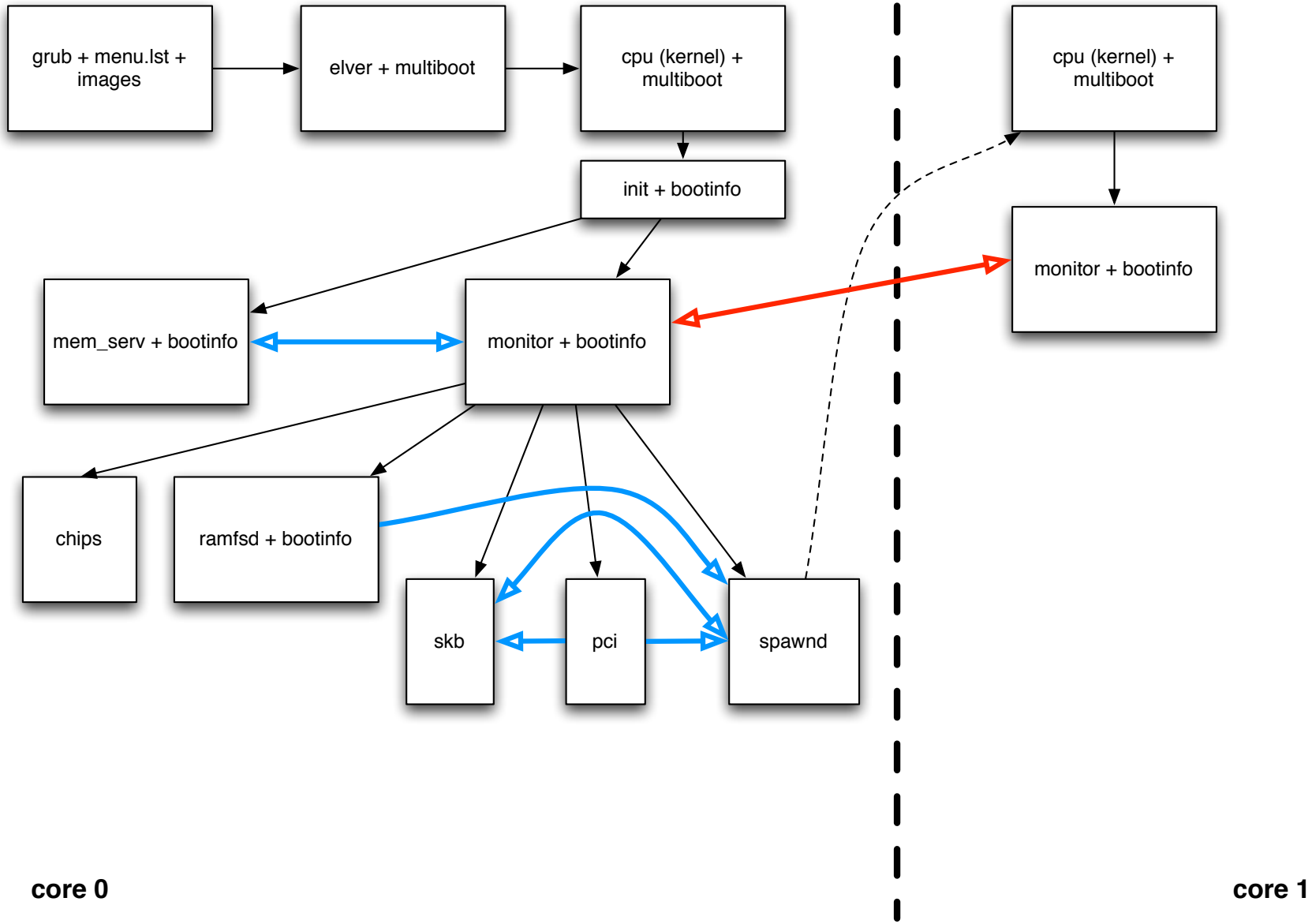
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
core 0
core 1

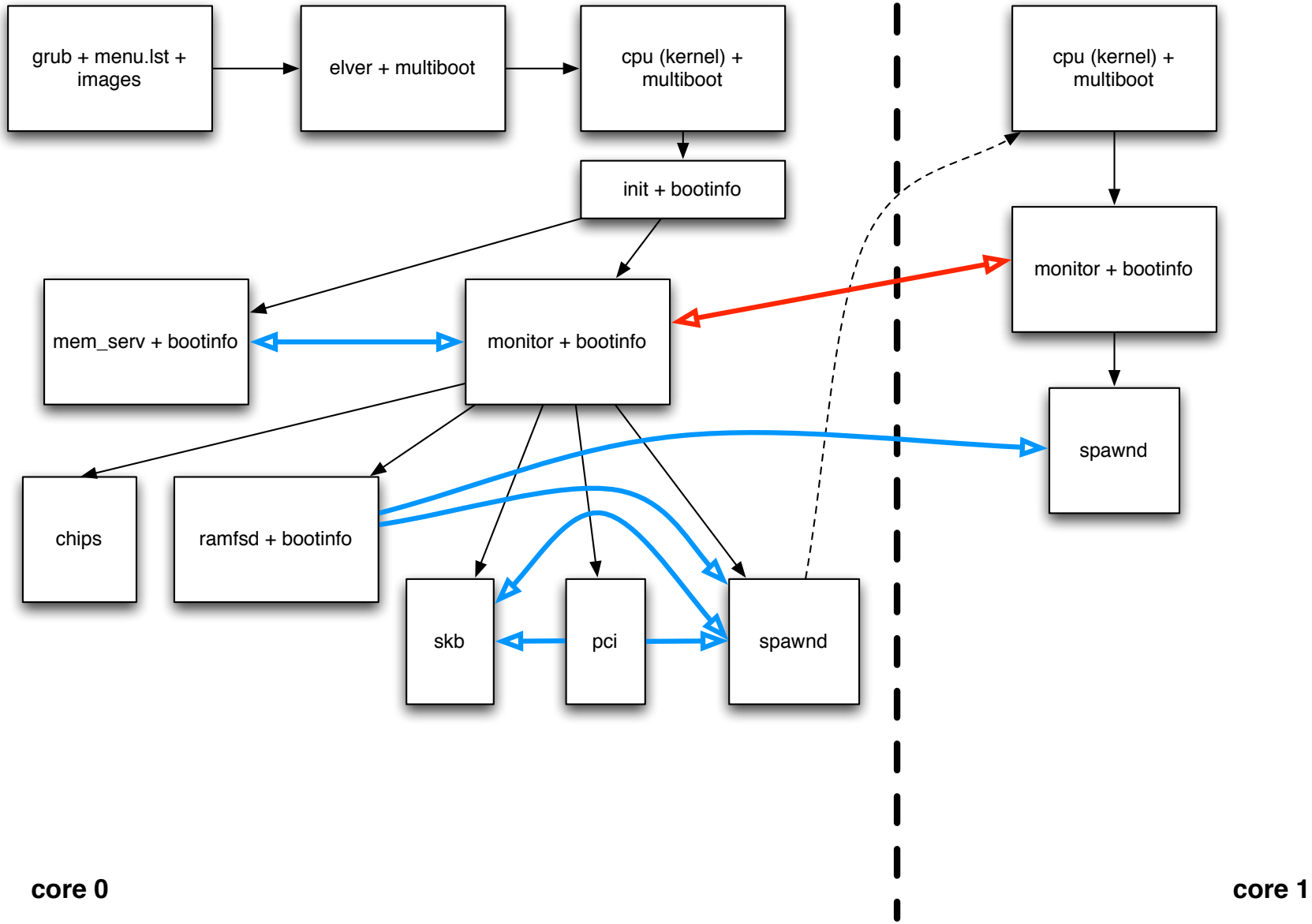
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
core 0
core 1

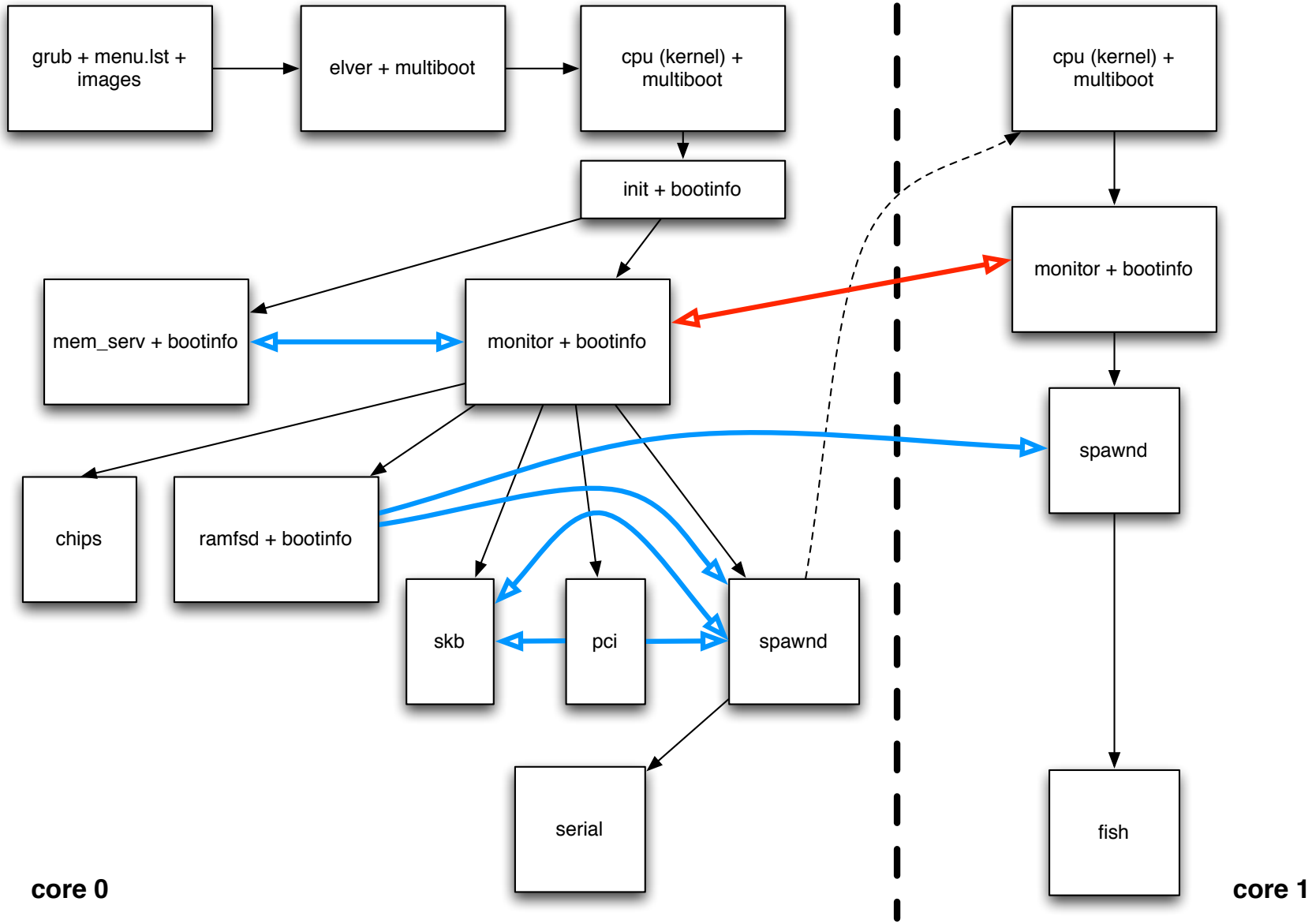
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
cpu (kernel) + multiboot
monitor + bootinfo
core 0
core 1

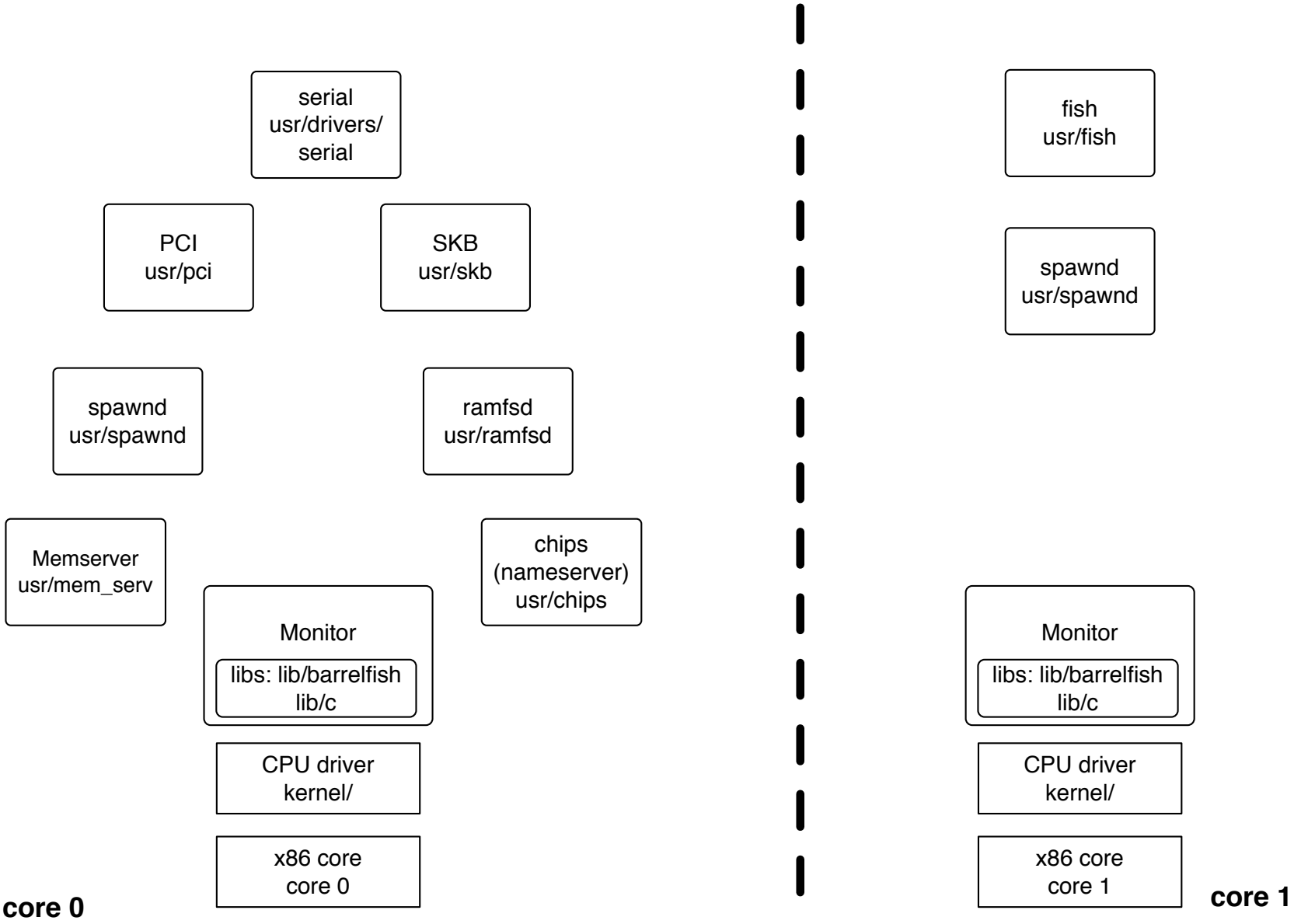
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
cpu (kernel) + multiboot
monitor + bootinfo
spawnd
core 0
core 1

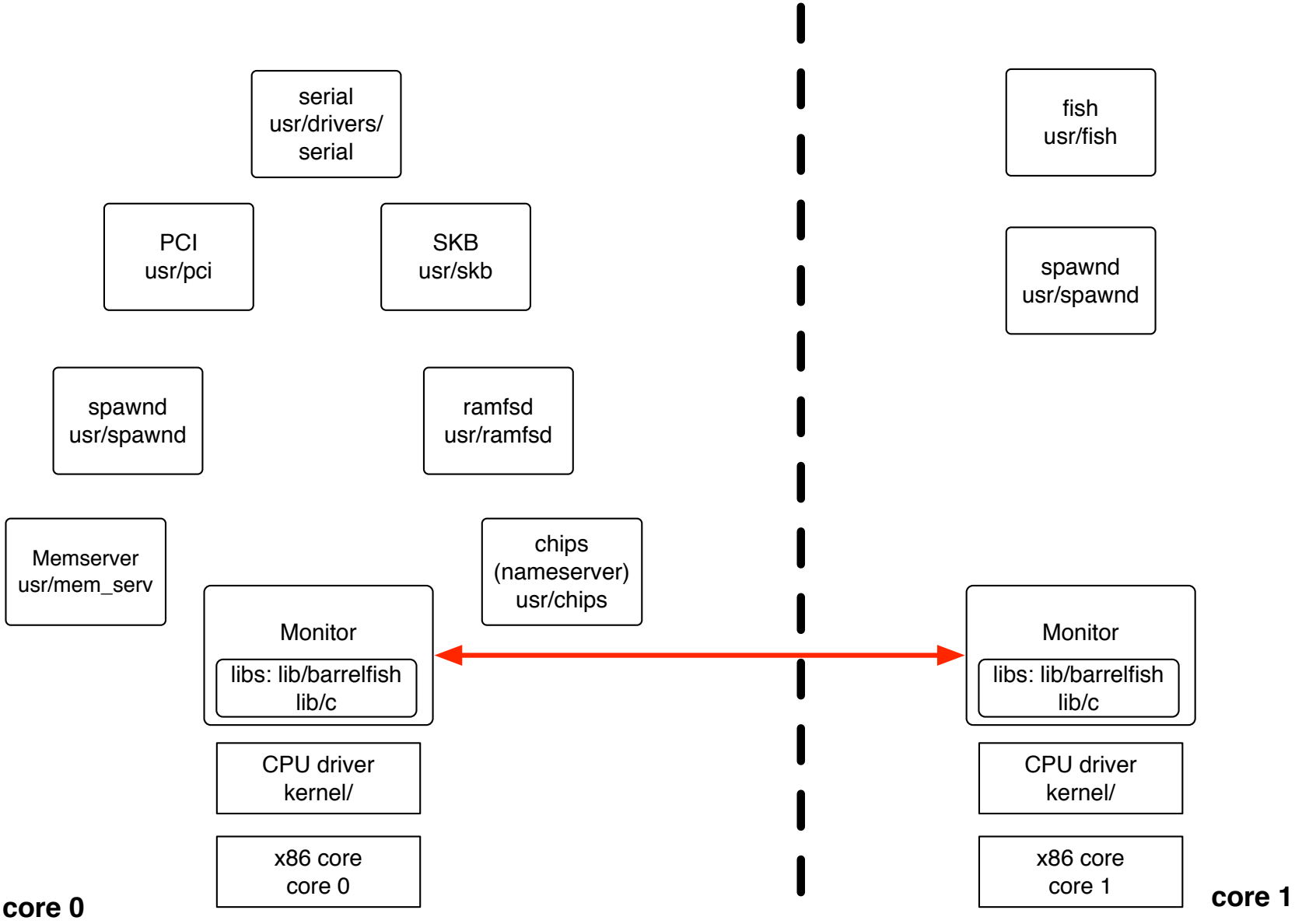
Bootstrap


grub + menu.lst + images
elver + multiboot
cpu (kernel) + multiboot
init + bootinfo
mem_serv + bootinfo
monitor + bootinfo
chips
ramfsd + bootinfo
skb
pci
spawnd
serial
cpu (kernel) + multiboot
monitor + bootinfo
spawnd
fish
core 0
core 1

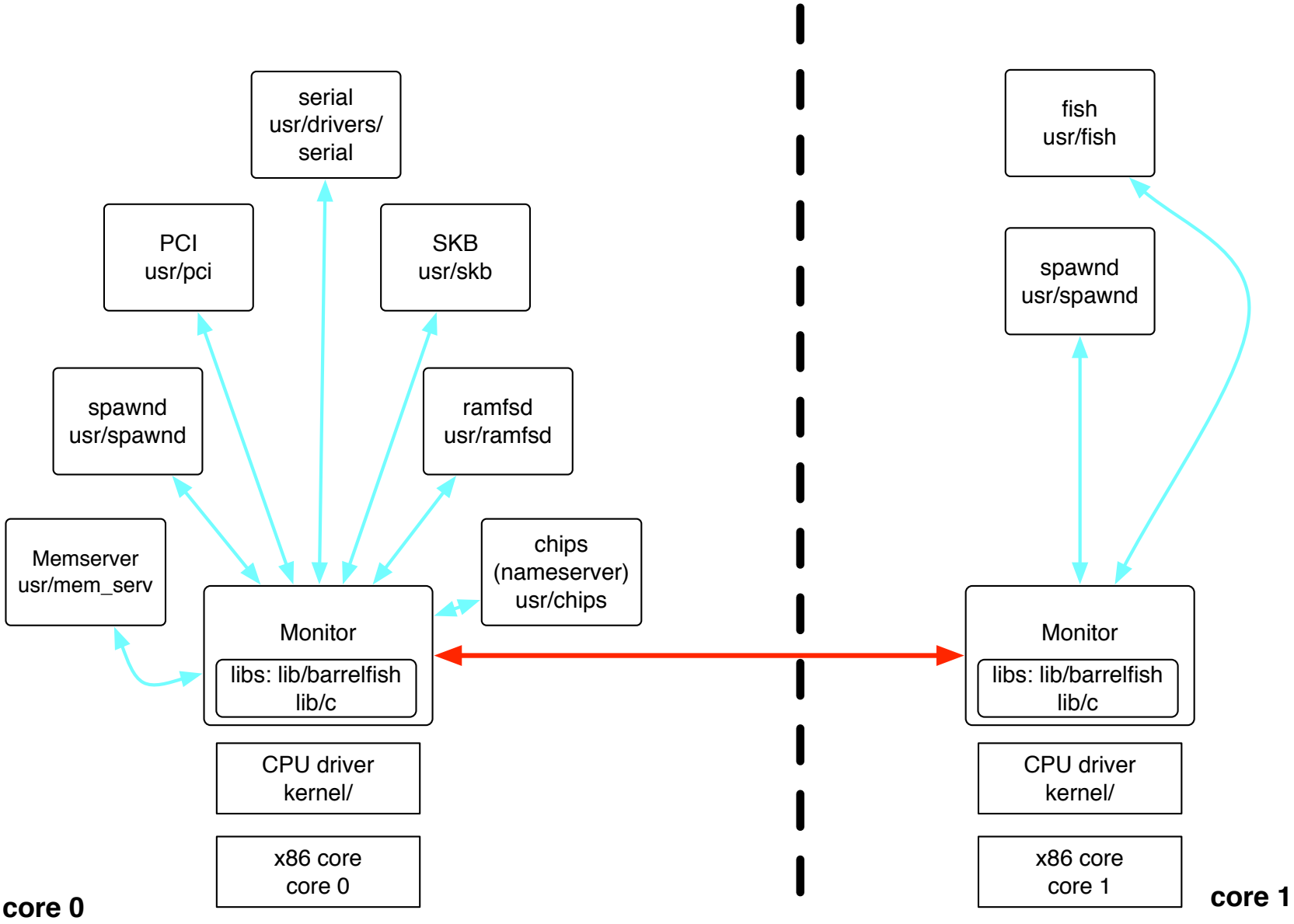
A Running System



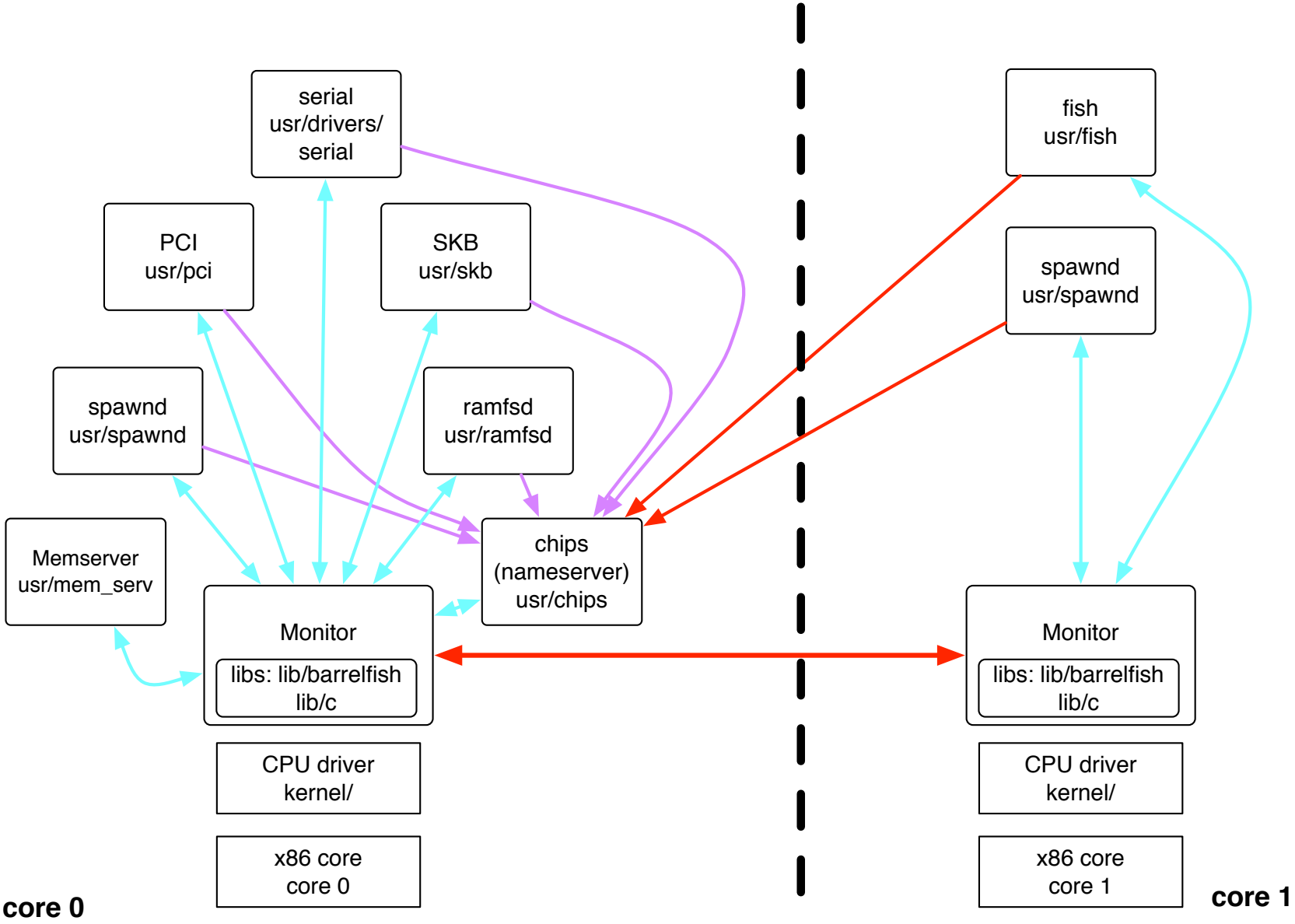
A Running System



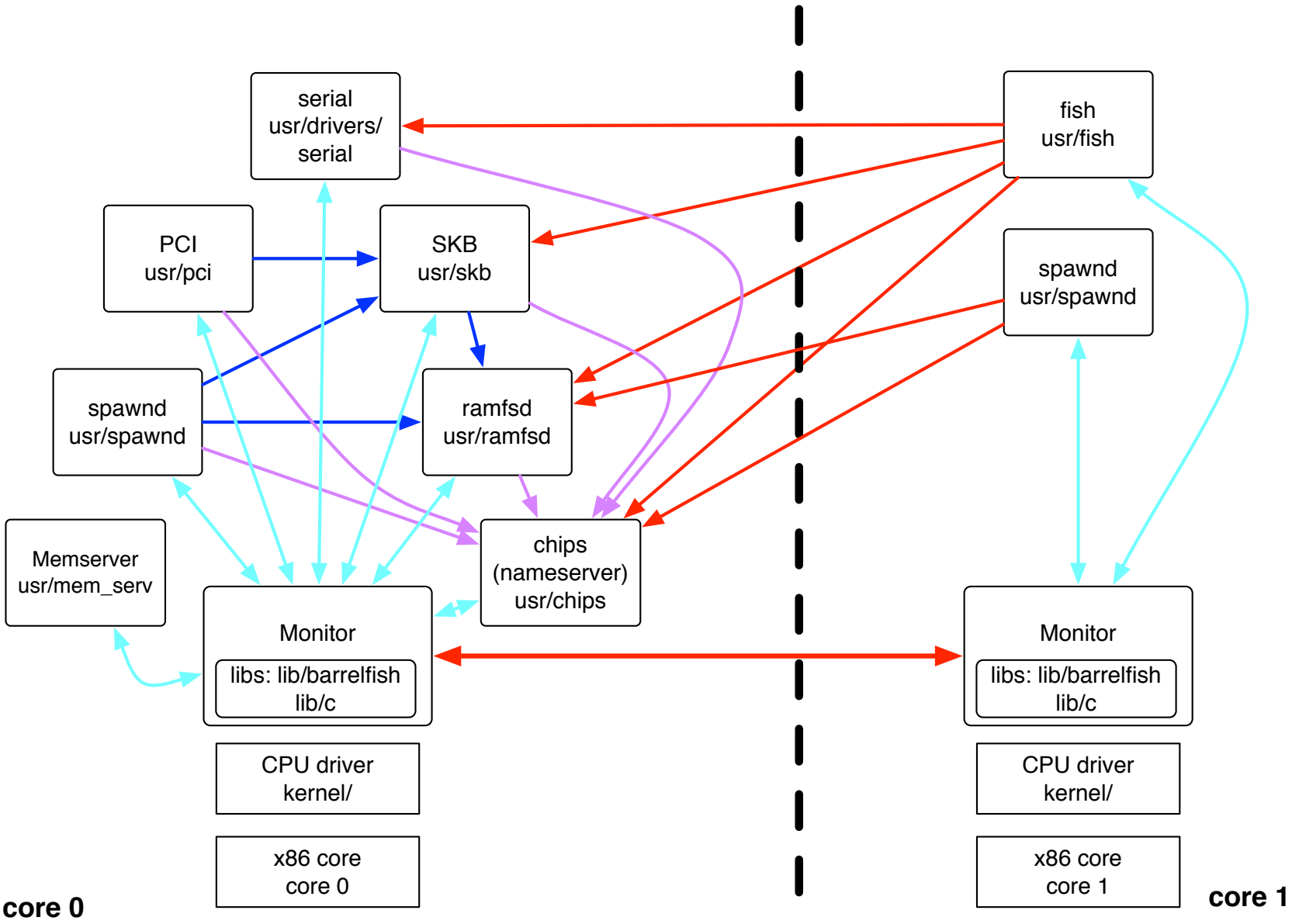
A Running System



A Running System



A Running System

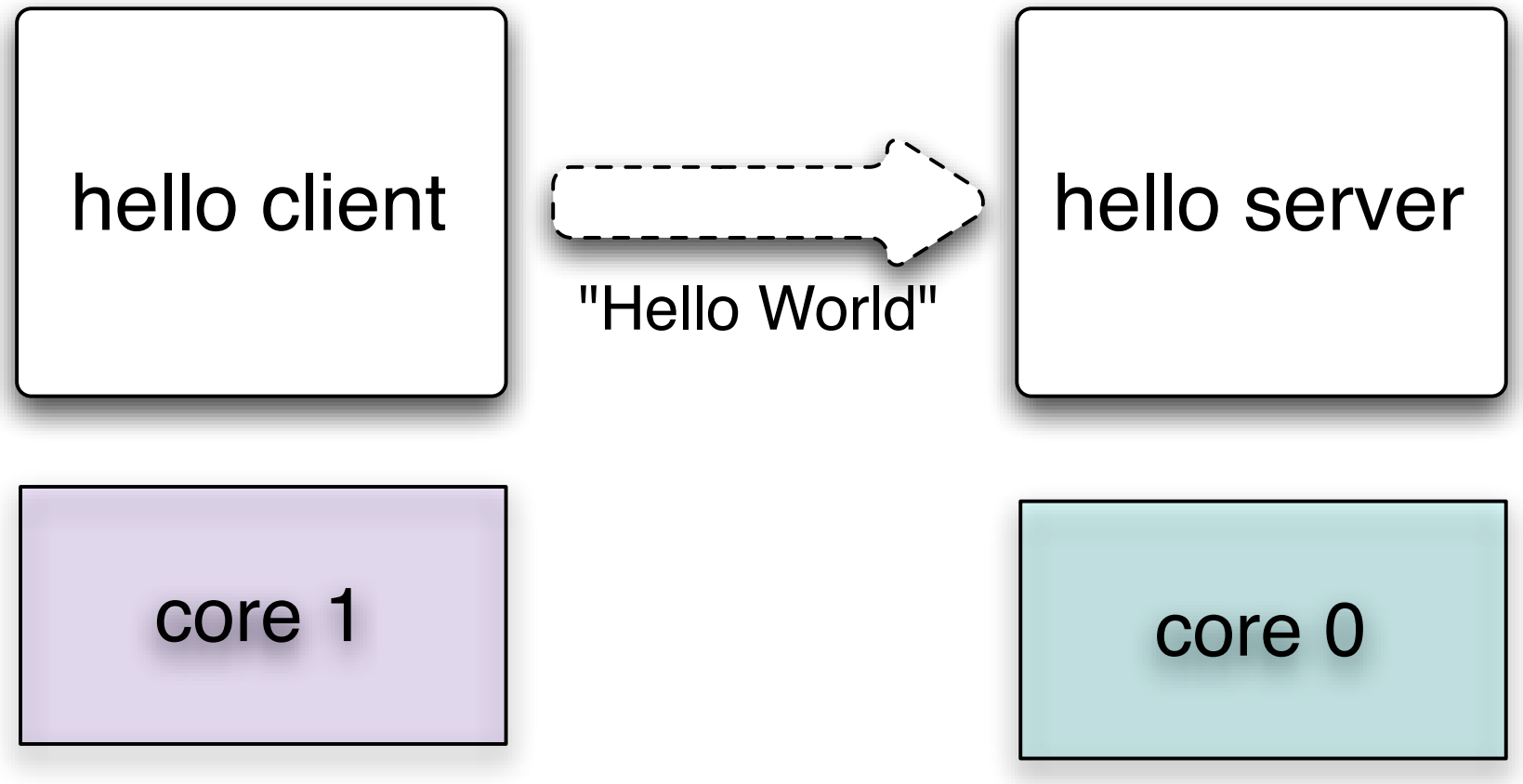


Barrelfish Hello World

Default

```
$ cd barrelfish/
$ mkdir usr/hello
$ cd usr/hello
$ vi Hakefile
...
$ vi hello.c
...
$ cd ../../../build/
$ vi symbolic_targets.mk
...
$ make rehake
...
$ make
...
$ vi menu.lst
...
$ make sim
...
spawnd.0: spawning /x86_64/sbin/hello on core 0
Hello World
$
```

Communicating Dispatchers



Interface

- in directory if/
- file: `hello.if`

```
interface hello "Hello World interface" {
  message hello_msg(string s);
};
```

- file: `Hakefile`
- add line for `hello` interface

```
[ flounderGenDefs (options arch) f
      | f <- [ "bcast",
...
       "hello" ],
             arch <- allArchitectures
] ++
```

Source Code

- directory: `usr\hello-cs`
- file: `hello.c`

What the code does:

- Determine whether client or server
- Server:
  - Export interface
  - Register with name service
  - Wait for connection
  - Wait for and handle messages
- Client:
  - Find interface
  - Connect
  - Send messages

Main

```
int main(int argc, char *argv[]) {
    errval_t err;

    if ((argc >= 2) && (strcmp(argv[1], "client") == 0)) {
        start_client();
    } else if ((argc >= 2) && (strcmp(argv[1], "server") == 0)) {
        start_server();
    } else {
        return EXIT_FAILURE;
    }
    struct waitset *ws = get_default_waitset();
    while (1) {
        err = event_dispatch(ws);
        if (err_is_fail(err)) {
            DEBUG_ERR(err, "in event_dispatch");
            break;
        }
    }
    return EXIT_FAILURE;
}
```

Server: Export

```
static void start_server(void)
{
    errval_t err;

    err = hello_export(NULL /* state for callbacks */,
                       export_cb, connect_cb,
                       get_default_waitset(),
                       IDC_EXPORT_FLAGS_DEFAULT);
    if (err_is_fail(err)) {
        USER_PANIC_ERR(err, "export failed");
    }
}
```

Server: Register

```
const char *service_name = "hello_service";

static void export_cb(void *st, errval_t err,
                      iref_t iref)
{
    if (err_is_fail(err)) {
        USER_PANIC_ERR(err, "export failed");
    }

    err = nameservice_register(service_name, iref);
    if (err_is_fail(err)) {
        USER_PANIC_ERR(err,
                      "nameservice_register failed");
    }
}
```

Server: Connect and Handle Messages

```
static void rx_hello_msg(struct hello_binding *b,
                         char *str)
{
    printf("server: received hello_msg:\n\t%s\n", str);
    free(str);
}

static struct hello_rx_vtbl rx_vtbl = {
    .hello_msg = rx_hello_msg,
};

static errval_t connect_cb(void *st,
                           struct hello_binding *b)
{
    b->rx_vtbl = rx_vtbl;
    return SYS_ERR_OK;
}
```

Client: Find and Bind

```
static void start_client(void)
{
    errval_t err;
    iref_t iref;

    err = nameservice_blocking_lookup(service_name,
                                      &iref);
    if (err_is_fail(err)) {
        USER_PANIC_ERR(err,
               "nameservice_blocking_lookup failed");
    }
    err = hello_bind(iref, bind_cb, NULL/*for bind_cb*/,
                     get_default_waitset(),
                     IDC_BIND_FLAGS_DEFAULT);
    if (err_is_fail(err)) {
        USER_PANIC_ERR(err, "bind failed");
    }
}
```

Client: Bind Callback

```
struct client_state {
    struct hello_binding *binding;
    int count;
};

static void bind_cb(void *st, errval_t err,
                    struct hello_binding *b)
{
    struct client_state *myst =
                malloc(sizeof(struct client_state));
    assert(myst != NULL);
    myst->binding = b;
    myst->count = 0;

    run_client(myst);
}
```

Client: Send Messages

```
static void run_client(void *arg)
{
    errval_t err;

    struct client_state *myst = arg;
    struct hello_binding *b = myst->binding;
    struct event_closure txcont =
                        MKCONT(run_client, myst);
    if (myst->count < MAX_SEND) {
        (myst->count)++;
        err = hello_hello_msg__tx(b, txcont,
                                  "Hello World");
        if (err_is_fail(err)) {
            DEBUG_ERR(err, "error sending message %d\n",
                      myst->count);
        }
    }
}
```

Hakefile

```
[ build application { target = "hello-cs",
                      cFiles = [ "hello.c" ],
                      flounderBindings = [ "hello" ]
                    }
]
```

menu.lst

```
...
# General user domains
module /x86_64/sbin/hello-cs core=0 server
module /x86_64/sbin/hello-cs core=1 client
```

Running It

Default

```
spawnd.0: spawning /x86_64/sbin/hello-cs on core 0
spawnd.1: spawning /x86_64/sbin/hello-cs on core 1
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
server: received hello_msg:
        Hello World
```