

AKBP II

A Programmer's Introduction to the JX System

Michael Golm



University of Erlangen
Department of Computer Science
(Operating Systems and Distributed Systems)
Martensstrasse 1
91058 Erlangen, Germany
golm@cs.fau.de

1

CVS

- Initial
 - `tsh: setenv CVSROOT /proj/akbp/repository`
 - `cvscv co jx`
- During the development cycle
 - `cvscv (-n) update -d -P -R`
 - `cvscv commit`

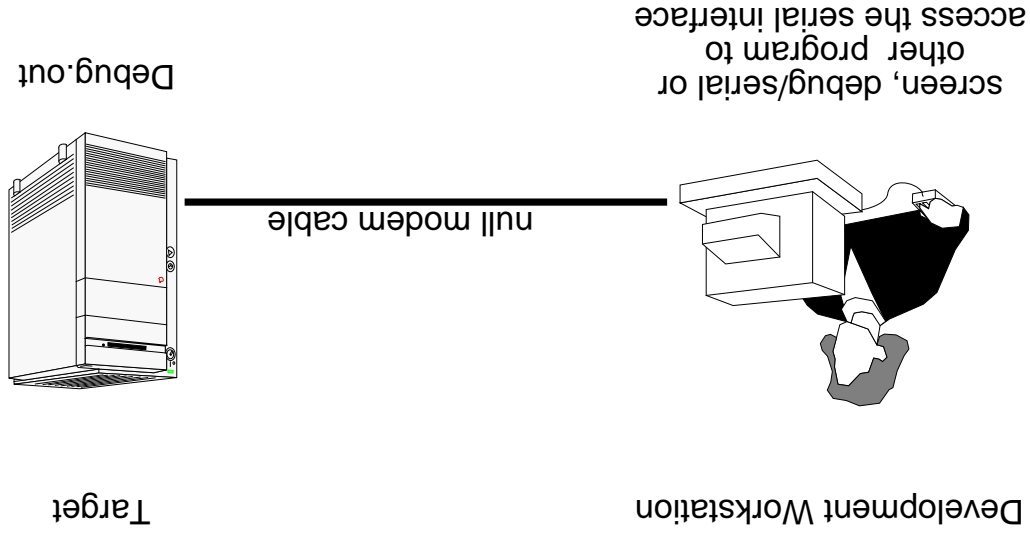
2

JX Directory Tree



3

Hardware Configuration



4

Boot Configuration

- Edit configuration file `boot.rc`
 - Select configuration
 - Specify configurations:
domain name, initial component, initial class, size of heap, args
- Example:

```
# select initial component
[My sysconf]
# configurationen
[My sysconf]
"Bar", "test.j11", "jx/test/Test", 8000000, "blubber"
"Foo", "murks.j11", "foo/bar/Main", 5000000, "bla", "blubb"
[Other configuration]
"FS", "fs.j11", "fs/filesystem", 8000000
"DB", "db.j11", "db/database", 5000000
```

5

Boot Procedure

- Build `µkernel` `jaxcore` and Java part `code.zip`:
 - Setup build environment: `source /proj/akbp/setUpenv.tcsh`
 - list needed components in the `COMPONENTS` file
- Boot using GRUB
 - invoke `make` in the top-level directory and wait

```
COMPONENTS = init
COMPONENTS += test
COMPONENTS += murks
```

6

- Used to explore a running system

- System is halted when monitor is activated (interrupts are disabled)

- On the development workstation: start the terminal program

- screen
- autodetach off
- screen -tjx 9 /dev/ttyS0 38400

- debug/serial

- kermi

- Any key activates the monitor

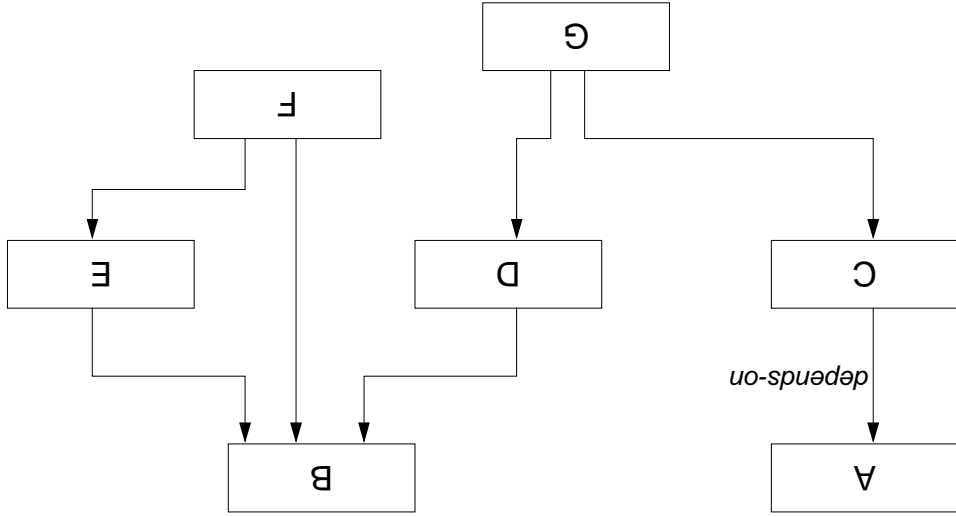
Monitor: Commands

- help: display help
- domains: Show list of all domains
- domain <domainid>: display information about a specific domain
- threads: show all threads
- heap <domainid>: show heap usage of a specific domain
- runq: display runqueue
- ns: display names of kernel name server
- c: continue normal system operation
- q: quit system and reboot
- ...

Components

- components depend on other components

- dependency relation creates a non-cyclic dependency graph



9

Components

- Each component has its own directory in `libs`

- `libs/<componentname>/META-File`

- NEEDLIBS = Lists all required components

- SUBDIRS = Lists all sub-directories

- LIBNAME = Name of this component

- NUMBERNV = int

- Sub-directories = package name; contain Java source files

- Add component to `libs/ALLCOMPONENTS-File` and `libs/COMPONENTS-File`, for example `COMPONENTS += my_com`

- Create sub-directories and invoke `make makefiles` in the top-level dir!

```

import jx.zero.debug.*;
...
public static void init(Naming naming, String[] args) {
    DebugChannel dbg = (DebugChannel) naming.lookup("DebugChannel0");
    Debug.out = new DebugPrintStream(new DebugOutputStream(dbg));
}
...
}

```

- New domain needs to initialize Debug.out

Initialize domain

- jx.zero contains interfaces to access DomainZero (µkernel)
- jx.zero.Naming is the name server portal
 - Portal lookup(string name)
 - void registerPortal(Portal portal, string name)

```

package xxx;
import jx.zero.*;
public class YY {
    public static void init(Naming naming, String[] args) {
    }
}

```

Entry point into new domain

Build a component

- invoke `build` in any sub-directory
 - compiles Java source into class files
 - creates a zip file of all class files (`libs/componentname.zip`)
 - translates the class files into machine code (x86) (`libs/componentname.jll`)
 - adds the `jll` file to `code.zip`

Portals

- Mechanism similar to Java-RMI

- 1. Portal interface

```
public interface MyService extends Portal {  
    void myMethod();  
}
```

- 2. Service implementation

```
public class MyServiceImpl implements MyService, Service {  
    public void myMethod() {  
        ...  
    }  
}
```

Naming

- Portals can be registered at a name service

- Every domain has at least one `Naming` portal

```
MyService svc = new MyServiceImpl();  
naming.registerPortal(svc, "Fooservice");
```

- Another (or the same) domain can lookup the portal using the name service

```
MyService svc = (MyService) naming.lookup("Foo");
```

- Wait until portal appears at the name service

```
MyService svc = (MyService) LookupHelper.waitForPortalAvailable(naming, "Foo");
```

- The initial naming portal of a domain can also be obtained using:

```
Naming naming = InitialNaming.getInitialNaming();
```

Naming

- Threads can be created as usual in Java (Subclass of `Java.lang.Thread` or implement `Java.lang.Runnable`); requires component `Jdk1.5`
- `DomainZero` (component zero) contains interface for thread creation

```
CPUSState thread = cpuManager.createCPUSState(new ThreadEntry() {  
    public void run() {  
        for(;;) {  
            Debug.out.println(""+mem.get8());  
            cpuManager.yield();  
        }  
    }  
});  
cpuManager.start(thread);
```

17

Memory: allocation

- Management of large memory regions and sharing of data between domains is done using `Memory` objects
- Get `Memory` object from `MemoryManager`

```
MemoryManager memoryManager = (MemoryManager) naming.Lookup("MemoryManager");  
Memory mem = memoryManager.alloc(4096);
```

Memory: revocation

- Revoke access to memory using `revoke()`. In the following example `mem2` is the only remaining reference to the memory.

```
Memory mem2 = mem.revoke();
```

Memory: access

- Access using getter/setter methods
- ```
byte b = mem.get8(pos);
int i = mem.get32(pos);
mem.set8(pos, value);
```
- `get32/set32` use little-endian encoding
  - `get32/set32` uses 32 bit index (`get32(2)` reads bytes 8..11 as little endian)
  - `getBigEndian32/setBigEndian32` use big-endian encoding

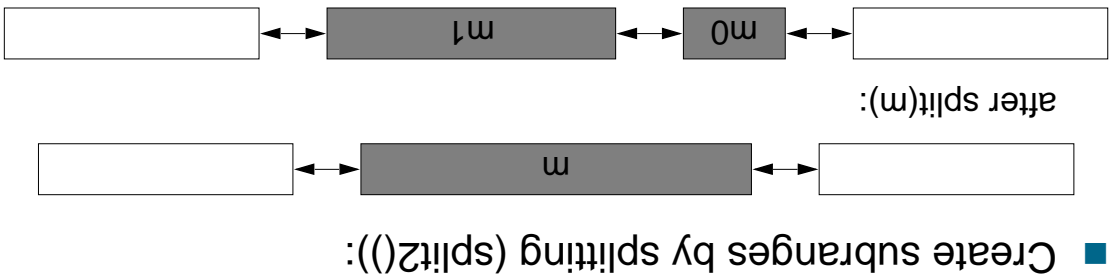
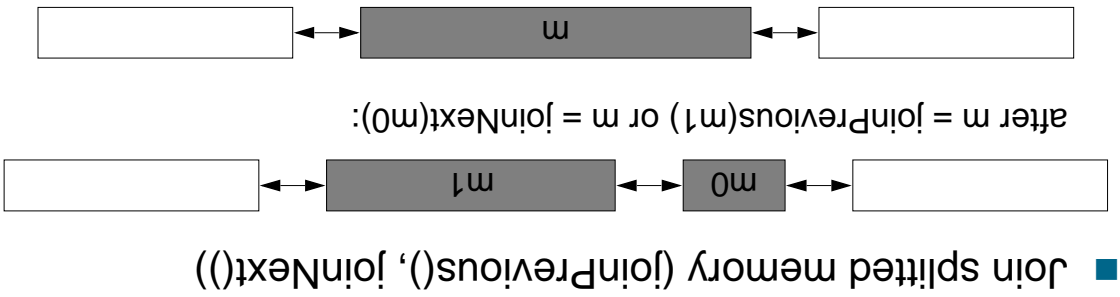
```

Memory mem = memoryManager.alloc(100+s);
Memory[] arr = new Memory[2];
mem.split2(100, arr);
// arr[0] contains reference to first half (size == 100)
// arr[1] contains reference to second half (size == s)

```

■ Example:

## Memory: splitting



## Memory: splitting

# ApplicationStarter

- Start normal java applications using the `main()` method

■ Example:

```
select initial component
[My sysconf]
Konfigurationen
[My sysconf]
"Bar", "Legacy_starter.j11", "jx/start/ApplicationStarter", 8000000, "jx/test/
Test", "test.j11", "blubb"
"Foo", "Legacy_starter.j11", "jx/start/ApplicationStarter", 5000000, "murks.j11",
"foo/bar/Main", "bla", "blubb"
```

23

# MonolithicApplicationStarter

- Run all applications in one domain
- End argument list by `null`

■ Example:

```
select initial component
[My sysconf]
Konfigurationen
[My sysconf]
"FooBar", "Legacy_starter.j11", "jx/start/MonolithicApplicationStarter", 8000000,
"jx/test/Test", "test.j11", "blubb", "blubb", "foo/bar/Main", "bla",
"blubb", null
```

24

---

## Visualization Tools: Thread Activity

### ■ steps to create a thread activity diagram:

- enable PROFILE\_EVENT\_THREADSWITCH in settings.makefile
- compile & boot
- use the debug/serial tool to drive the serial line
- activate the monitor and type the command "rswtchesonly"
- this starts a binary transmission of the thread switch log

### ■ use the ta script to create a diagram.mif file from the binary log file rswtchesonly

- `ta rswtchesb rswtchesonly diagram.mif - - - 1`

---

25

---

## Synchronization

- Java has a built-in recursive mutual exclusion lock (synchronized methods and synchronized blocks)

### ■ Problems:

- Is either slow or causes memory overhead
- Can not be used in interrupt handlers (they are not allowed to block)
- Is prone to deadlock

---

26

# JX synchronization mechanisms

## ■ JX.zero.AtomicVariable

```
public interface AtomicVariable extends Portal {
 void set(Object value);
 Object get();
 void atomicUpdateUnlock(Object value, CPUSState state);
 void blockIfEqual(Object testValue);
 void blockIfNotEqual(Object testValue);
}
```

## ■ Non-preemptive scheduling

## ■ JX.zero.CAS

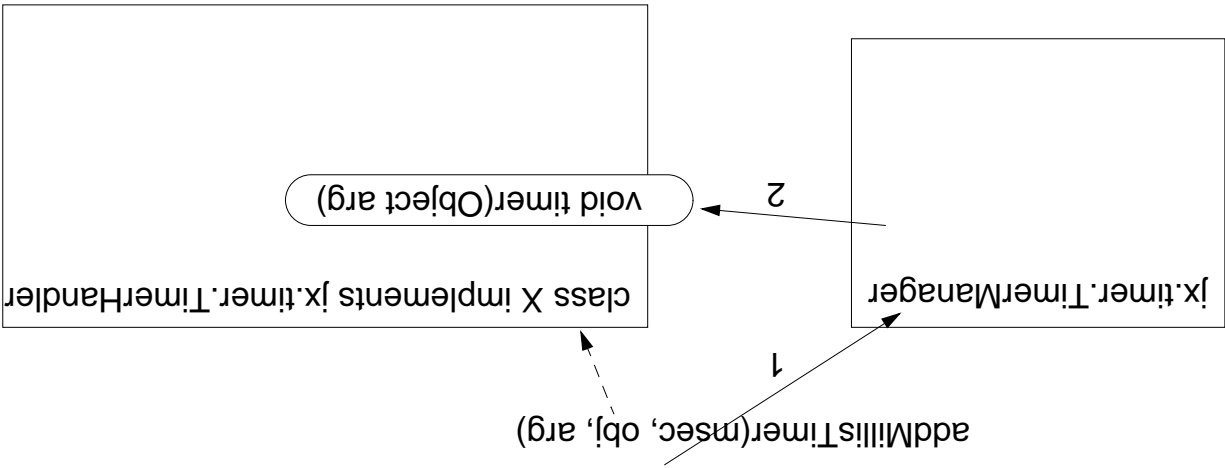
```
public interface CAS extends Portal {
 boolean casObject(Object obj, Object compareValue, Object setValue);
 boolean casInt(Object obj, int compareValue, int setValue);
 boolean casBoolean(Object obj, boolean compareValue, boolean setValue);
 boolean casShort(Object obj, short compareValue, short setValue);
 boolean casByte(Object obj, byte compareValue, byte setValue);
}
```

# Timer

## ■ TimerManager:

```
public interface TimerManager extends Portal {
 Timer addMillisTimer(int expiresFromNow, TimerHandler handler, Object argument);
 Timer addMillisIntervalTimer(int expiresFromNow, TimerHandler handler, Object argument);
 Timer addTimer(int expiresFromNow, int interval, TimerHandler handler, Object argument);
 boolean deleteTimer(TimerHandler t);
 int getTimerMillis();
 int getTimerBaseInMicros();
 int getCurrentTime();
}
```

## ■ components: `timer` (interfaces) and `timer_pc` (implementation)



## ■ Unix (Linux):

- List all network interfaces: `/sbin/ifconfig -a`
- Dump network packets: `/usr/sbin/tcpdump -i eth0`
  - `-e`: print link-level header (ethernet)
  - `-v & -vv`: verbose and more verbose
  - `-x`: print packet in hex
  - use filter expressions
- `tcpdump -x -v -i eth0 ether src or dst 8:67:42:24:00:07`
- `tcpdump -vv -R -e -x -p -nn -N -i eth0`

- "PCIDomain", "pci\_pc.jll", "jx/devices/pci/PCIGod", "jx/schedulier/HLRRobin", 300000
- "NicNetDomain", "legacy\_starter.jll", "jx/start/MonoIthicApplicationStarter", 20000000, "jx/net/StartNetDevice", "net\_devices.jll", "NIC", "eth0", null, "jx/net/protocols/StartNetworkProtocols", "net\_protocols.jll", "NIC", "NET", null

## ■ Example use:

name: name of the network portal ("NET")  
srcPort: local port number  
dstPort: remote port number

```
NetInit net = (NetInit)LookupHelper.waitForPortalAvailable(naming, name);
UDPSEnder u = net.getUDPSEnder(srcPort, new IPAddress("192.168.34.2"), dstPort);
Memory buf = net.getUDPBuffer(50);
u.send(buf);
```

# Network: IP

## ■ Send IP packets:

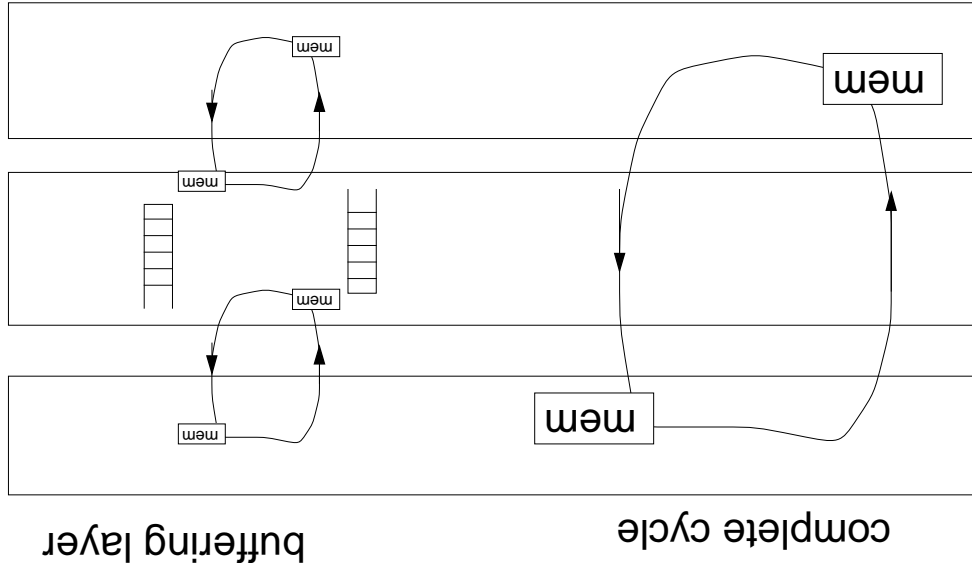
name: name of the network portal ("NET")

```
NetInit net = (NetInit)LookupHelper.waitForPortalAvailable(naming, name);
UDPSEnder u = net.getIPSEnder(new IPAddress("192.168.34.2"));
Memory buf = net.getIPBuffer(50);
u.send(buf);
```



## Buffer Management

- Buffers are cycled through the stack
- Each layer has an initial pool of buffers that are used to receive packets



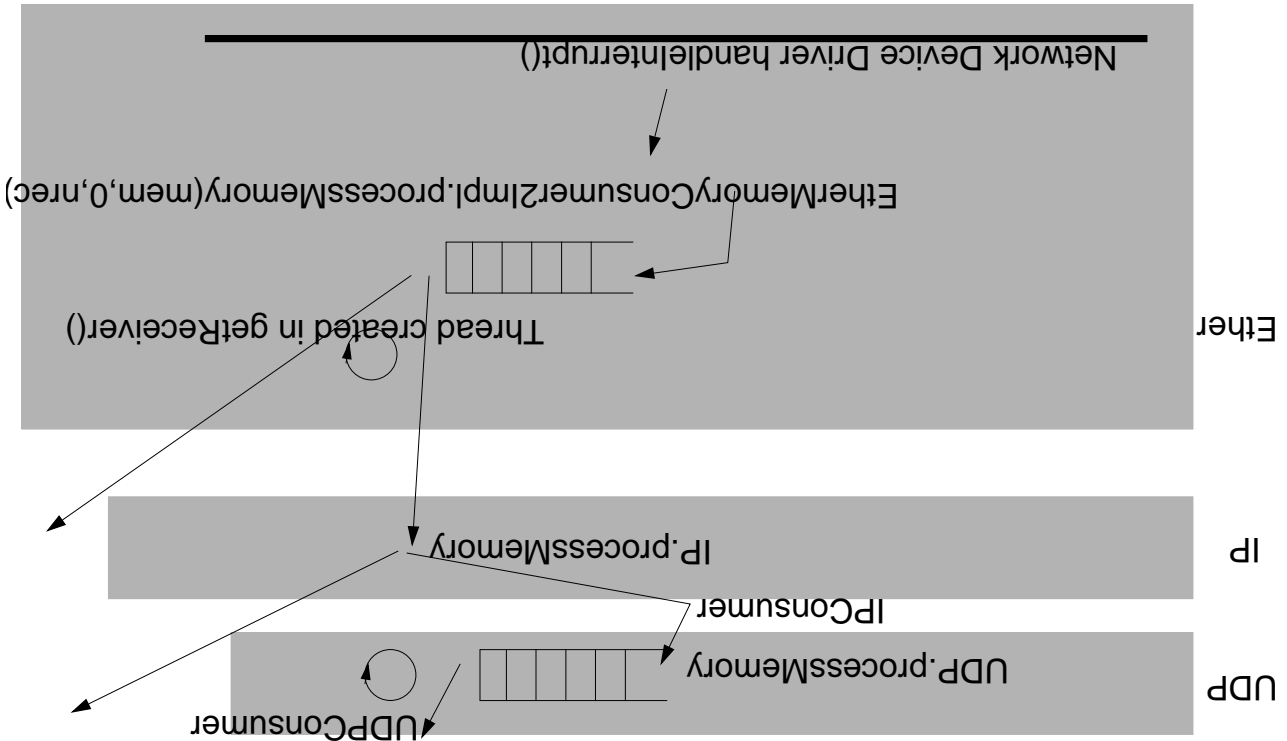
33

## Network

- Modern NICs have on-device buffer. They detect network collisions and automatically resend the packet.
- Packets can get lost within the network.
- Packets can be received in arbitrary order.

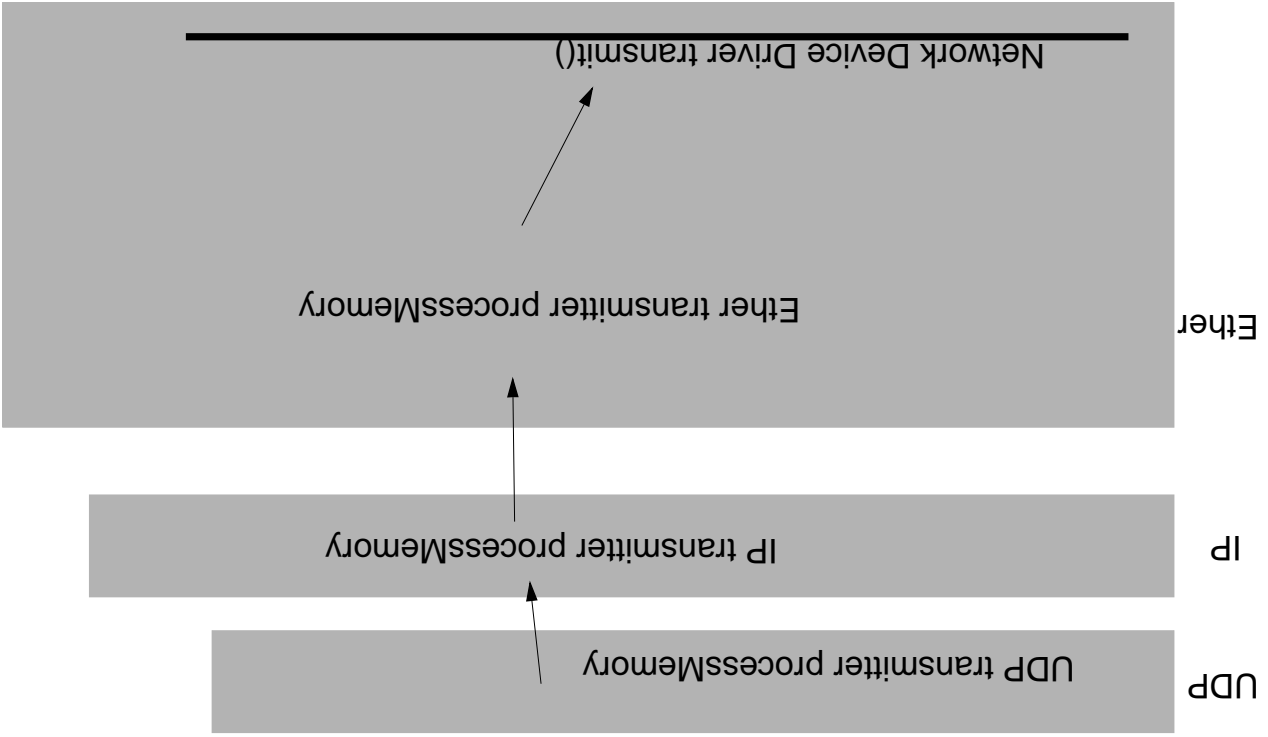
34

## Network Stack: Receive



35

## Network Stack: Transmit



36