IMPLEMENTATION OF CHERI CAPABILITIES IN A SAFETY-CRITICAL REAL-TIME OPERATING SYSTEM AND TYPE-1 HYPERVISOR FOR INTELLIGENT EDGE SYSTEMS

Dmytro Yeliseyev, Software Architect, Wind River

November 13, 2025

Agenda

WNDRVR

Intro

1.1

Security/safety problem

1.2

Looking for a solution: CHERI

SECURITY/SAFETY PROBLEM

Around 70% of all CVEs exploit the lack of memory safety or potential weaknesses in the implementation of software runtime environments based on C/C++.

```
#include <stdio.h>
int x=1;
int secret_key = 4091;
int main() {
  int *p = &x;
  p = p+1; {
  int leak = *p;
  printf("leak: %d\n",leak); }
}
```

© Peter Sewell: *CHERI and Arm Morello*
https://media.ccc.de/v/emf2024-87-cheri-and-arm-morello

Lines of open-source code

C	6,300,000,000
C++	2,100,000,000
JavaScript	1,800,000,000
Java	1,700,000,000
...	...
Rust	47,000,000

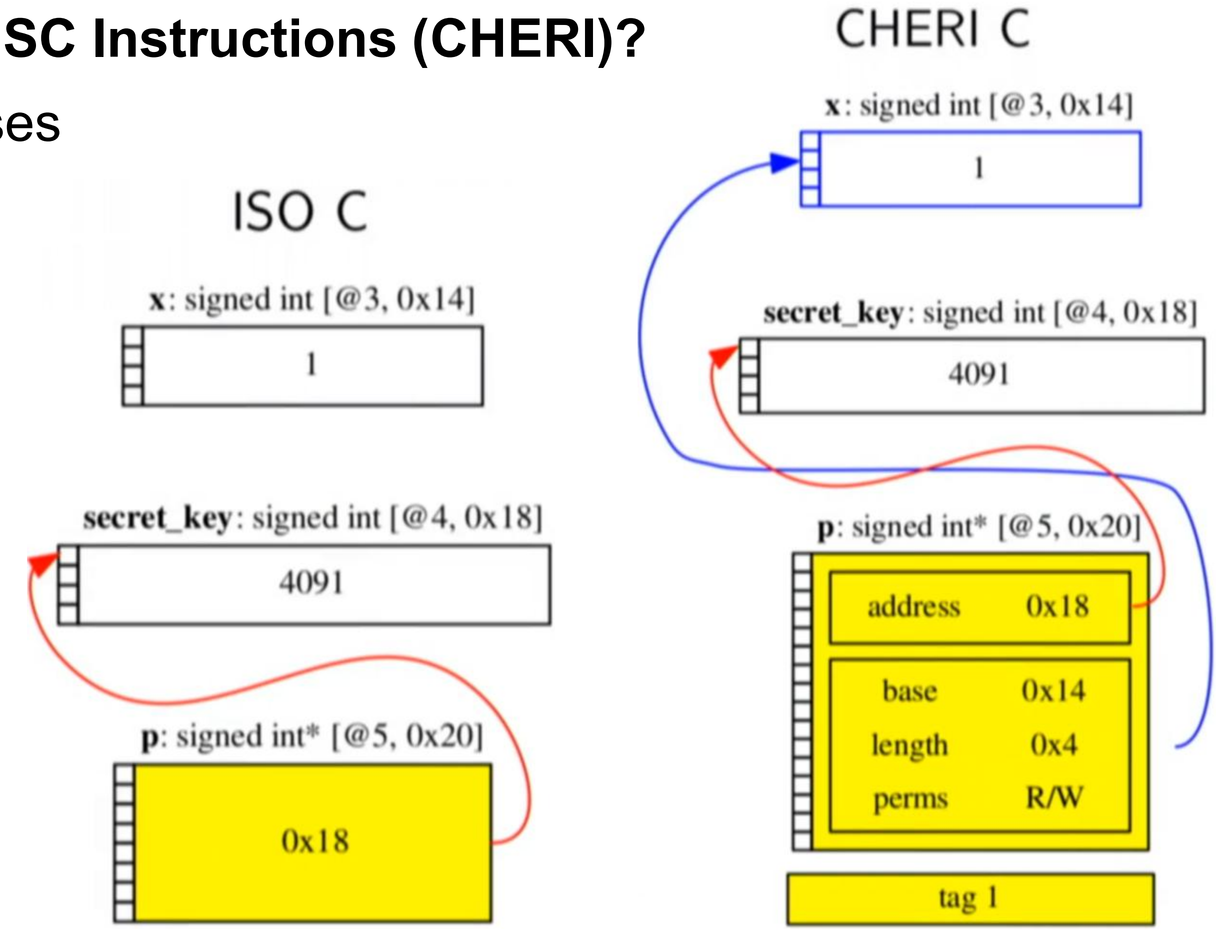
(Synopsys Black Duck Open Hub https://www.openhub.net/languages, May 2024)

LOOKING FOR A SOLUTION: CHERI

What is Capability Hardware Enhanced RISC Instructions (CHERI)?

- Distinguish between numbers and addresses
- Add metadata to addresses to limit the scope of their usage
- Differentiate data and code addressing
- Address protection mechanism: sealing

```
#include <stdio.h>
int x=1;
int secret_key = 4091;
int main() {
   int *p = &x;
   p = p+1; {
   int leak = *p;
   printf("leak: %d\n",leak); }
}
```



Technology overview

2.1

~~Pointer~~ … capability!

2.2

Capability lifecycle

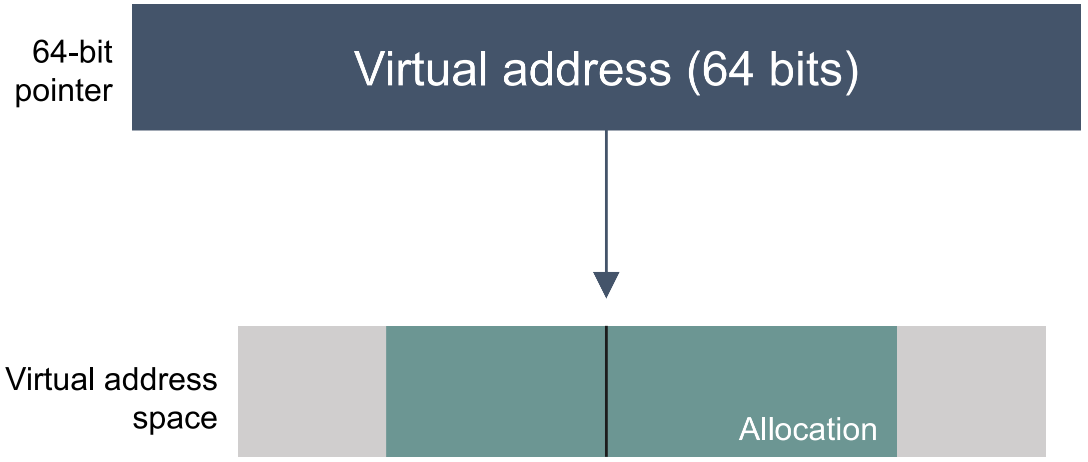
2.3

Deploying CHERI: Pure vs. hybrid capability modes

WNDRVR

~~POINTER~~ … CAPABILITY!

Pointers today

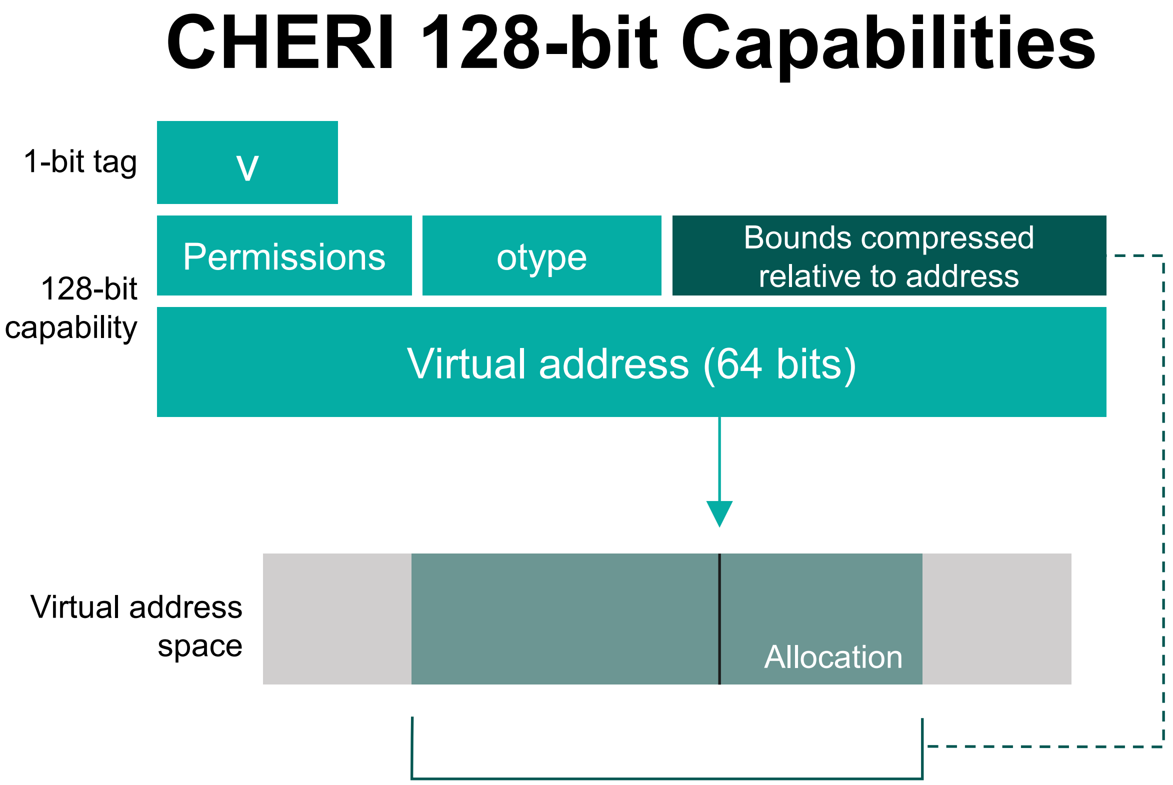


- Implemented as **integer virtual addresses (VAs)**
- (Usually) point into **allocations and mappings**
  - **Derived** from other pointers via integer arithmetic
  - **Dereferenced** via jump, load, and store
- **No integrity protection**—can be injected/corrupted
- **Arithmetic errors**—out-of-bounds leaks/overwrites
- **Inappropriate use**—executable data and format strings
- **Attacks on data and code pointers are highly effective, often achieving arbitrary code execution**

WNDRVR

UNIVERSITY OF CAMBRIDGE

~~POINTER~~ … CAPABILITY!



CHERI capabilities extend pointers with:

- **Tags**—Protect capabilities in registers and memory (tag per 128-bit)
  - Dereferencing an untagged capability throws an exception
  - In-memory overwrite automatically clears the capability tag
- **Permissions**—Prevent unintended use of pointers
- **Bounds**—Limit the range of address space accessible via a pointer
  - Floating-point compressed 64-bit lower and upper bounds (may be imprecise for large arrays)
  - Prevents pointers being used to access the wrong object
- **Sealing**: **immutable**, **non-dereferenceable capabilities**—Used for non-monotonic transitions; the basis for building fine-grained compartments

CAPABILITY LIFECYCLE

Conceptual questions:

Provenance validity:	Q: Where do pointers come from?
Integrity:	Q: How do pointers move in practice?
Bounds and permissions:	Q: What rights should pointers carry?
Monotonicity:	Q: How do you control rights escalation?
Controlled non-monotonicity:	Q: How do you handle exceptions?

Integrity and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations

Monotonicity prevents pointer privilege escalation; for example, broadening bounds

DEPLOYING CHERI: PURE VS. HYBRID CAPABILITY MODES

Pure: All, explicit, and implicit pointers are capabilities only.

Hybrid: By default, pointers are unsigned integers, but capabilities can be used as well.

Pure capability mode:

- **One of the main goals:**
  minimal changes to the existing C/C++ code!
- **Challenges:**
  - Assembler code in many places requires a special new instructions usage
  - C/C++ code:
    - Assumptions about the pointer size in data structures
    - Capability corruption; for example, in msgQSend/DMA
    - Handle the pointers used as the ID of the system objects
    - Source of provenance—provenance is inherited from the left-hand side

Hybrid mode:

- **Why:**
  - Allows the selective use of CHERI capabilities from within an otherwise unmodified and ABI-compatible, C/C++-language code base.
  - Its aim is to allow management of, and interoperation with, capability-enabled code while using an integer rather than capability pointer implementation internally.
- **How:**
  - Capabilities **specifically annotated** in the sources.
    - `int* __capability myPtr1, myPtr2;`
    - `printf (“Res is: %d", *(__cheri_fromcap int*) myPtr1);`

WNDRVR

Implementation

WNDRVR

APPROACH

Due to the **complexity** of the overall system architecture and dependencies of system components, it was decided to take an incremental development approach involving **smaller steps** that would enable progress to be assessed and validated, which would reduce overall technical risk compared to attempting to integrate modifications of multiple system architecture components in a single step.

- Get VxWorks RTOS running on Morello silicon but without enabling support for CHERI capabilities.
  - Get the VxWorks RTOS kernel running in hybrid mode.
    - Enable the pure capability mode support only in VxWorks user space.

      While estimating the changes needed in the kernel running in the hybrid mode to support pure capability mode in the user space, it was found that this effort is comparable to the effort needed to run the entire kernel in pure capability mode. It was therefore decided to skip this step.
    - Enable pure capability mode support in the VxWorks kernel.

SOURCE OF INSPIRATION

UNIVERSITY OF CAMBRIDGE https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/

A script to build and run CHERI-related software—one build tool to rule them all:
cheribuild https://github.com/CTSRD-CHERI/cheribuild

Supported operating systems include Ubuntu.

For example, the following command:

```
./cheribuild.py --include-dependencies run-morello-purecap
```

- Clones and builds **CHERI LLVM**
- Clones and builds **CHERI QEMU**
- Clones, builds, and runs **CheriBSD** on CHERI QEMU

CheriBSD: A complete memory- and pointer-safe FreeBSD C/C++ kernel + user space, which is very useful to get examples of how to use the CHERI software and tools existing so far.

Adversarial CHERI exercises and missions: https://ctsrd-cheri.github.io/cheri-exercises

CHERI QEMU

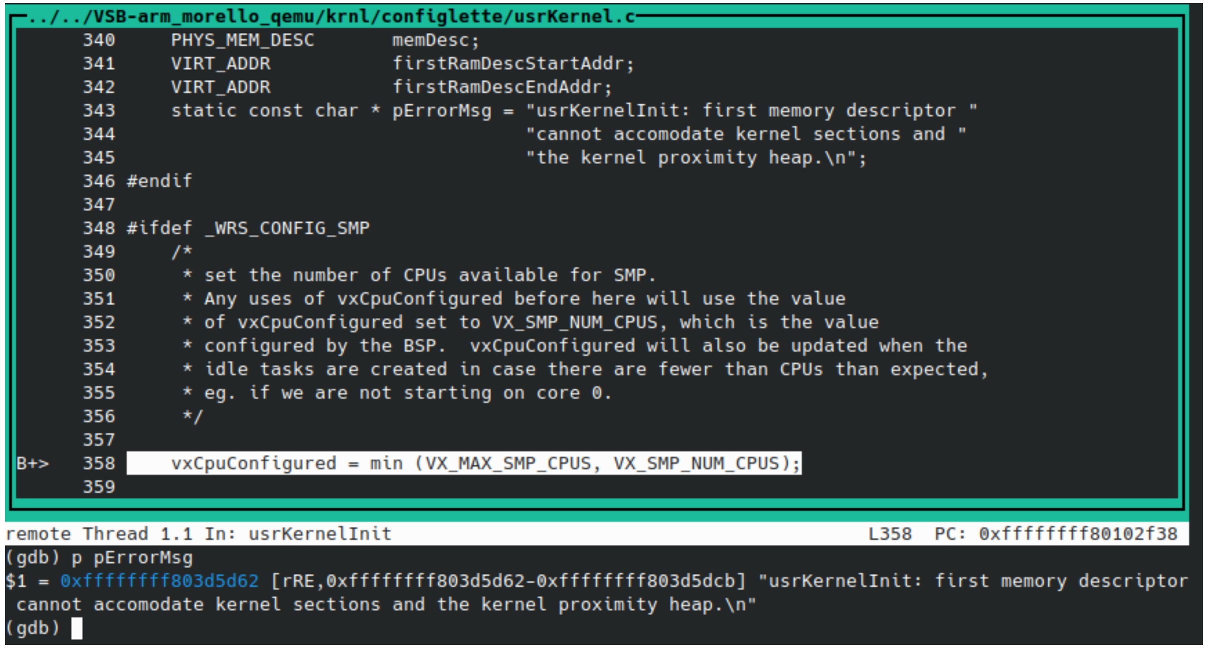
`/cheribuild.py --include-dependencies qemu`

- The QEMU machine that is used to run CheriBSD is taken as a base.
    - Machine: **virt** CPU: **morello**

*“The virt board is a platform which does not correspond to any real hardware;
it is designed for use in virtual machines.”*

Correspondingly, CHERI QEMU required its own BSP and FDT in VxWorks.

- GDB:
    - `./cheribuild.py gdb-native`
    - Simply brilliant.

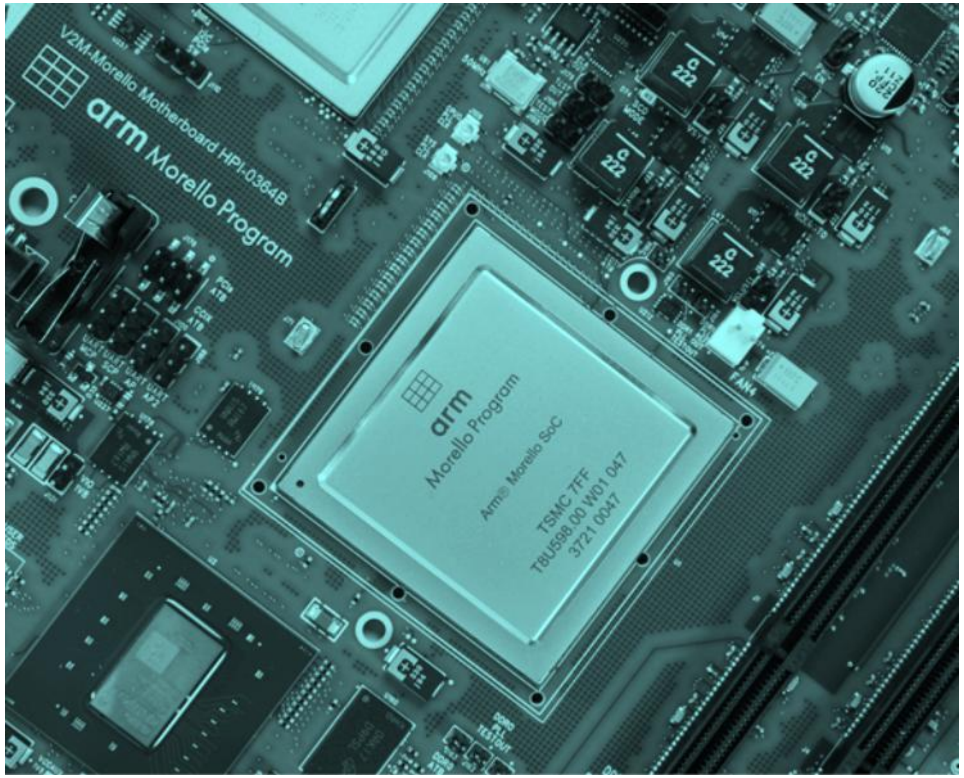


```
../../VSB-arm_morello_qemu/krnl/configlette/usrKernel.c
       340     PHYS_MEM_DESC       memDesc;
       341     VIRT_ADDR           firstRamDescStartAddr;
       342     VIRT_ADDR           firstRamDescEndAddr;
       343     static const char * pErrorMsg = "usrKernelInit: first memory descriptor "
       344                                     "cannot accomodate kernel sections and "
       345                                     "the kernel proximity heap.\n";
       346 #endif
       347
       348 #ifdef _WRS_CONFIG_SMP
       349     /*
       350      * set the number of CPUs available for SMP.
       351      * Any uses of vxCpuConfigured before here will use the value
       352      * of vxCpuConfigured set to VX_SMP_NUM_CPUS, which is the value
       353      * configured by the BSP.  vxCpuConfigured will also be updated when the
       354      * idle tasks are created in case there are fewer than CPUs than expected,
       355      * eg. if we are not starting on core 0.
       356      */
       357
B+>    358     vxCpuConfigured = min (VX_MAX_SMP_CPUS, VX_SMP_NUM_CPUS);
       359
remote Thread 1.1 In: usrKernelInit                                   L358   PC: 0xffffffff80102f38
(gdb) p pErrorMsg
$1 = 0xffffffff803d5d62 [rRE,0xffffffff803d5d62-0xffffffff803d5dcb] "usrKernelInit: first memory descriptor
 cannot accomodate kernel sections and the kernel proximity heap.\n"
(gdb)
```

MORELLO SYSTEM DEVELOPMENT PLATFORM (SDP)

Morello is a research program led by the University of Cambridge and Arm in association with partners and funded by the UKRI as part of the UK government Digital Security by Design (DSbD) program.

- The **Morello SoC** is a prototype silicon implementation of a capability hardware CPU instruction set architecture (ISA): an **experimental** application of CHERI ISAv8 to ARMv8-A. The Morello SoC is based on the **Arm Neoverse N1** core with tagged memory support.
- **ARM Development Studio (Morello Edition)** can be configured to use the embedded JTAG probe on the ARM Morello SDP.
- The VxWorks Morello BSP and PSL are based on the BSP and PSL of the already supported Amazon Web Services (AWS) Graviton2 CPU, based on the same Arm Neoverse N1 core, but without CHERI support.



WNDRVR

VxWorks: BUILD SYSTEM

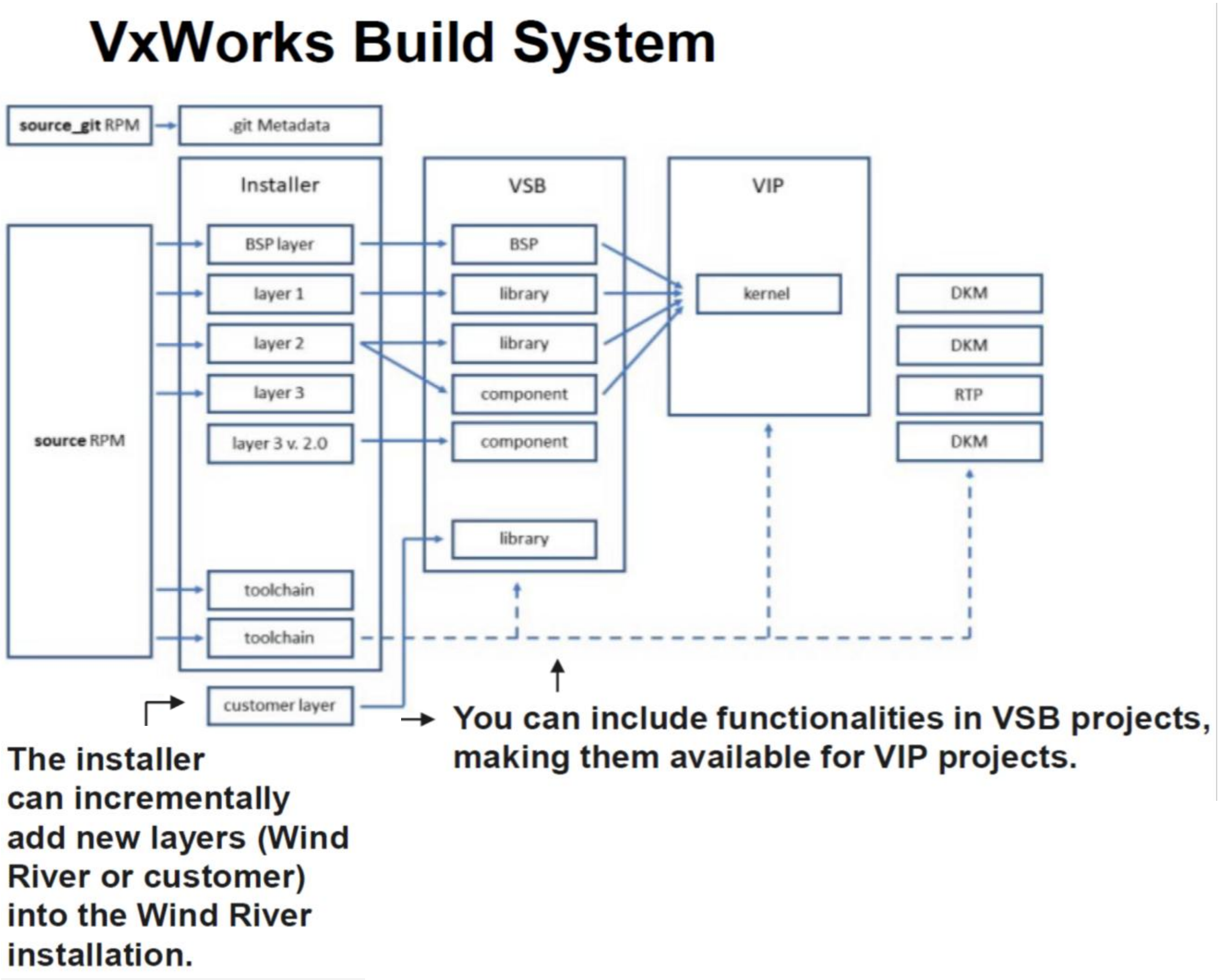
wr-llvm-morello = *wr-llvm + morello-llvm*

--target=arm64 -> --target=aarch64

- -march=morello+noa64c
- -march=morello+a64c
- -march=morello+c64 -mabi=purecap

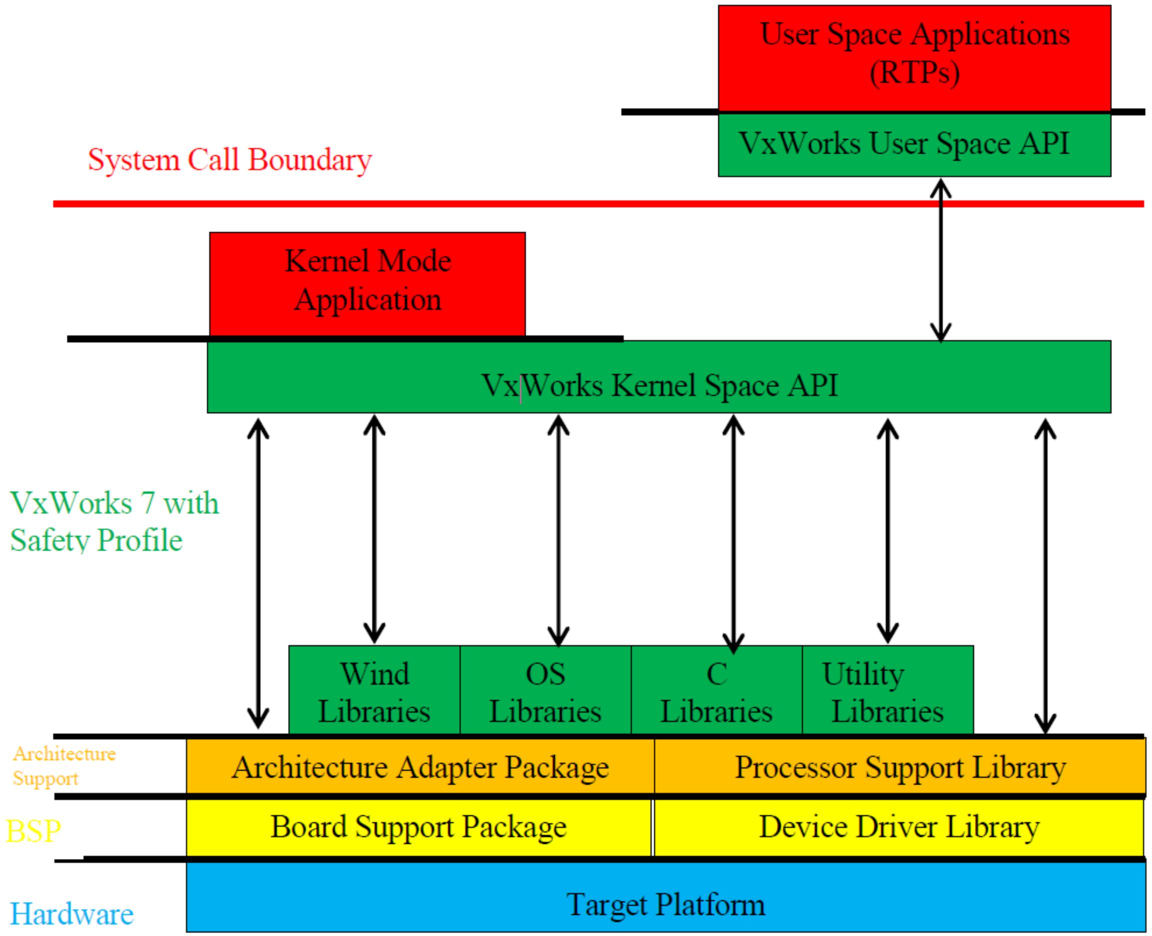
ldarm64 -> ld.lld

- .cpu_private (DSECT) **->** .cpu_private (COPY)
- __cap_reloc – split .text vs .rodata
- .size for asm symbols



VxWorks: RTOS COMPONENTS

- **Hardware support: Morello SDP + QEMU:**
  - Architecture support for the Neoverse N1 CPU
  - BSP and PSL (*FDT, boardLib, and std drivers*)
  - MMU (*> 512 GB mem address space and more*)
- **Startup**
  - Vectors
  - MMU enables RW of capabilities
  - Enable CHERI instructions
  - __cap_reloc runtime initialization
- **Scheduler**
  - Extend TCBs, 128-bit regs, and special regs
  - Align structures, system call APIs, and more
- **Exceptions**
  - For example, ERET required CELR instead of ELR
  - New exception types > handlers
- **Memory managers**
- **Kernel libraries API**
  - Tasks, signals, utils, shell, and user space
- **User space**
  - RTP DLL: TLS descriptor reloc types support



VxWorks: SOURCE CODE

Expected problems:

Alignment issues: Capabilities are always naturally aligned. This is a requirement of the hardware. (there is one **tag** bit per 128 bits/16 bytes)

```
/*
 * REG_SET - ARM Register set
 */

typedef struct          /* REG_SET - ARM register set */
    {
    _Vx_ULONG   r[_GREG_NUM];   /* general purpose registers */
    _Vx_ULONG   sp;             /* stack pointer             */
    _Vx_INSTR * pc;             /* program counter           */
```

```
#if __has_feature(capabilities)
          typedef uintcap_t ARM_REG_TYPE;
#else
          typedef uintptr_t ARM_REG_TYPE;
#endif

#define ARM_REG_ALIGN    _Alignas (sizeof (ARM_REG_TYPE))

#define ARM_REG_M        ARM_REG_ALIGN ARM_REG_TYPE

/*
 * REG_SET - ARM Register set
 */

typedef struct          /* REG_SET - ARM register set */
    {
    ARM_REG_M    r[_GREG_NUM];   /* general purpose registers */
    ARM_REG_M    sp;             /* stack pointer             */
    ARM_REG_M    pc;             /* program counter           */
```

bcopy: To be able to copy memory blocks with capabilities inside, you must use capability load and store instructions to propagate capability metadata and tags.

- The source address must be 16-byte aligned before whole 16-byte chunks are copied, so copy small chunks first until the address is aligned.
- Modify copy instructions:

```
ldp   x1, x2, [x0], #16          ldr   c1, [c0], #16
stp   x1, x2, [x0], #16   →      str   c1, [c0], #16
```

WNDRVR

VxWorks: SOURCE CODE

<u>Unexpected problems:</u>

Atomic op:
Non-morello: `LDAXR/STLXR`;

```
                   vxAtomic64Cas
38B8 D10103FF SUB     sp,sp,#0x40
38BC F9001FE0 STR     x0,[sp,#0x38]
38C0 F9001BE1 STR     x1,[sp,#0x30]
38C4 F90017E2 STR     x2,[sp,#0x28]
38C8 F9401FEB LDR     x11,[sp,#0x38]
38CC F94017E8 LDR     x8,[sp,#0x28]
38D0 F9000FE8 STR     x8,[sp,#0x18]
38D4 F9401BE9 LDR     x9,[sp,#0x30]
38D8 F9400FEC LDR     x12,[sp,#0x18]
38DC C85FFD?8 LDAXR   x8,[x11]
38E0 EB09011F CMP     x8,x9
38E4 54000061 B.NE    vxAtomic64Cas+56 ; 0xFFFFFFFF80
38E8 C80A?D6C STLXR   w10,x12,[x11]
38EC 35FFFF?A CBNZ    w10,vxAtomic64Cas+36 ; 0xFFFFFF
38F0 F90007E8 STR     x8,[sp,#8]
38F4 EB09010A SUBS    x10,x8,x9
38F8 1A9F17EA CSET    w10,EQ
38FC B90013EA STR     w10,[sp,#0x10]
3900 EB090108 SUBS    x8,x8,x9
3904 54000060 B.EQ    vxAtomic64Cas+88 ; 0xFFFFFFFF80
3908 F94007E8 LDR     x8,[sp,#8]
```

Morello: `CAS`—crash without ISB in front of it

```
                   vxAtomic64Cas
01EC D10103FF SUB     sp,sp,#0x40
01F0 F9001FE0 STR     x0,[sp,#0x38]
01F4 F9001BE1 STR     x1,[sp,#0x30]
01F8 F90017E2 STR     x2,[sp,#0x28]
01FC F9401FEB LDR     x11,[sp,#0x38]
0200 F94017E8 LDR     x8,[sp,#0x28]
0204 F9000FE8 STR     x8,[sp,#0x18]
0208 F9401BE8 LDR     x8,[sp,#0x30]
020C F9400FE? LDR     x10,[sp,#0x18]
0210 AA080?E9 MOV     x9,x8
0214 C8A9?D6A CAS     x9,x10,[x11]
0218 EB080128 SUBS    x8,x9,x8
021C 1A9F17E? CSET    w8,EQ
0220 F90007E9 STR     x9,[sp,#8]
0224 2A0803E9 MOV     w9,w8
0228 B90013E9 STR     w9,[sp,#0x10]
022C 370000A8 TBNZ    w8,#0,vxAtomic64Cas+84
0230 14000001 B       vxAtomic64Cas+72 ; 0xFF
```

VxWorks: PROBLEMS DETECTED IN COMPILE-TIME

VxWorks Source Build (VSB) in pure capability mode:

- ~115 warnings not related to capabilities
- ***~2,345 warnings related to capabilities***
- Breakdown by type of warning:
  - 2,160 (~92%): Cast from provenance-free integer type to pointer type will give pointer that cannot be dereferenced
  - 110 (~5%): Alignment problems of various types; for example, structure members
  - 67 (~3%): Implicit conversion loses capability metadata
  - 8 (0.3%): Binary expression on capability types, not clear which is source of provenance

The vast majority of warnings are indicators of **traditionally-written code**, especially when assumptions are made about arbitrarily-sized integers (that is, long) being able to store pointer values.

VxWorks: PROBLEMS DETECTED IN COMPILE-TIME

Resolution:

- Apply VxWorks CHERI **coding rules!**
  - C/C++: Macros throughout to handle conversion between pointer and integer values
  - ASM: Macros for registers and common operations
- Modify kernel APIs using VIRT_ADDR, where a pointer (capability) is really intended/required.
- Add support for atomic operations on capabilities (128-bit values).
- Rework structure alignment as needed to be sympathetic to capabilities.
- GOT: **__cap_reloc** runtime initialization:

```
#define SYS_BOOT_LINE_LEN 256
char bootLine[SYS_BOOT_LINE_LEN];
void* addr = (void*) &bootLine;
```

```
cheri_init_globals_3();
```

```
void* addr = (void*) 0x1C090000;
```

```
size_t  len = SYS_BOOT_LINE_LEN;
VIRT_ADDR const tmp = ADR_FROM_PTR (*addr);
*addr = DATA_PTR_FROM_ADR_WITH_LEN (tmp, len);
```

WNDRVR

CONCLUSION

- **Team of four engineers -** two years
- **CHERI tool chain**
- **Morello hardware and QEMU support**
- **VxTest and CHERI tests**
  - Regression test suite
  - OS integration tests
  - CHERI core functionality tests
- **Hybrid capability mode**
- **Pure capability mode**
  - Kernel & user space support
  - VxWorks debugging facilities – still in progress…
  - CHERI compartmentalization – prototypes under construction
  - Bounds tightening for DDC-derived objects – planned…
  - Vulnerability analysis…

```
Target Name: vxTarget


 _________            _________
 \........\          /......../
  \........\        /......../
   \........\      /......../
    \........\    /......../
     \........\   \......./
      \........\   \...../              VxWorks Cert Edition SMP 64-bit
       \........\   \.../
        \........\   \./     Release version: 23.06
         \........\   -      Build date: Jan 16 2025 17:15:49
          \........\
           \......./         Copyright Wind River Systems, Inc.
            \...../   -                 1984-2025
             \.../   /.\
              \./   /...\
               -   -------

                    Board: Arm Morello (FDT)
                CPU Count: 1
           OS Memory Size: 14208MB
         ED&R Policy Mode: Deployed

 Adding 14983 symbols for standalone.

vxTestOptions:            -em -v 4
->
-> vxTest
```

THANK YOU!

www.windriver.com/contact

WNDRVR