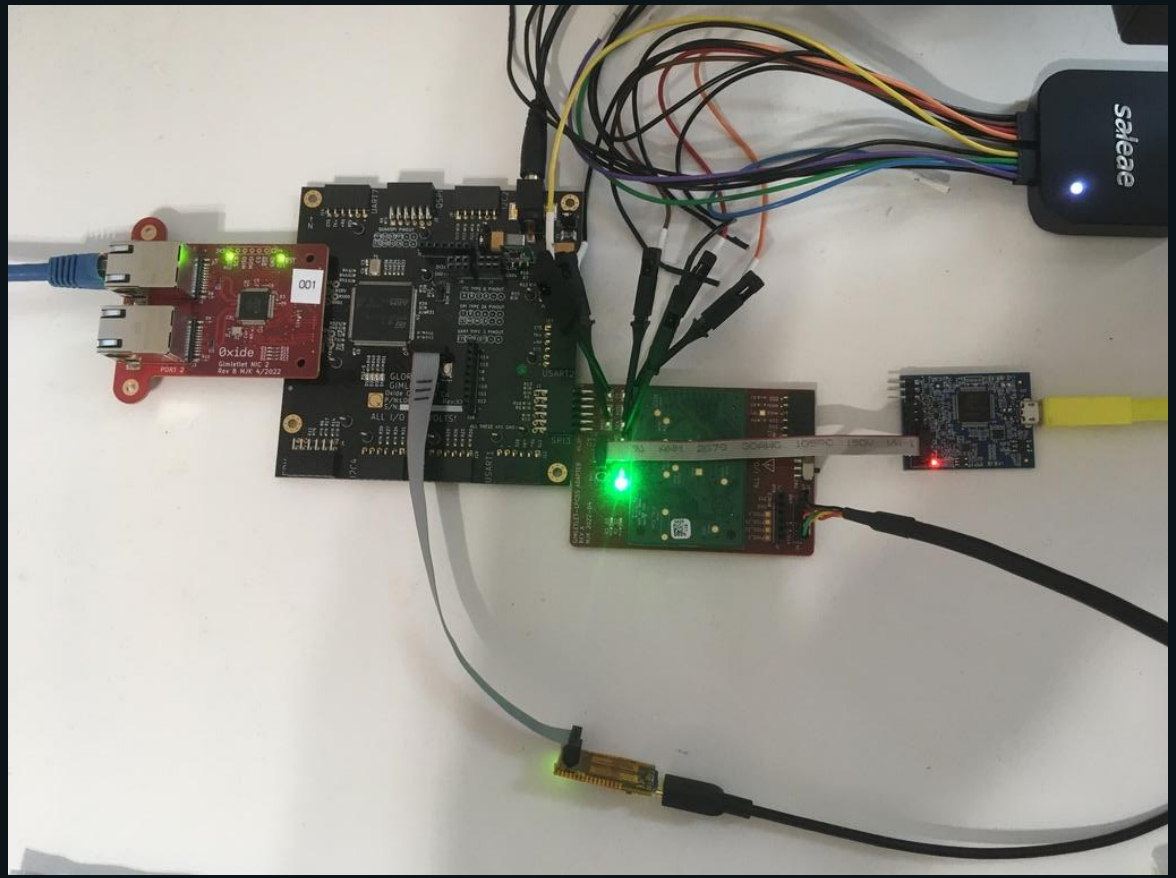
Unplugging the Debugger: Live and post-mortem debugging in a remote system

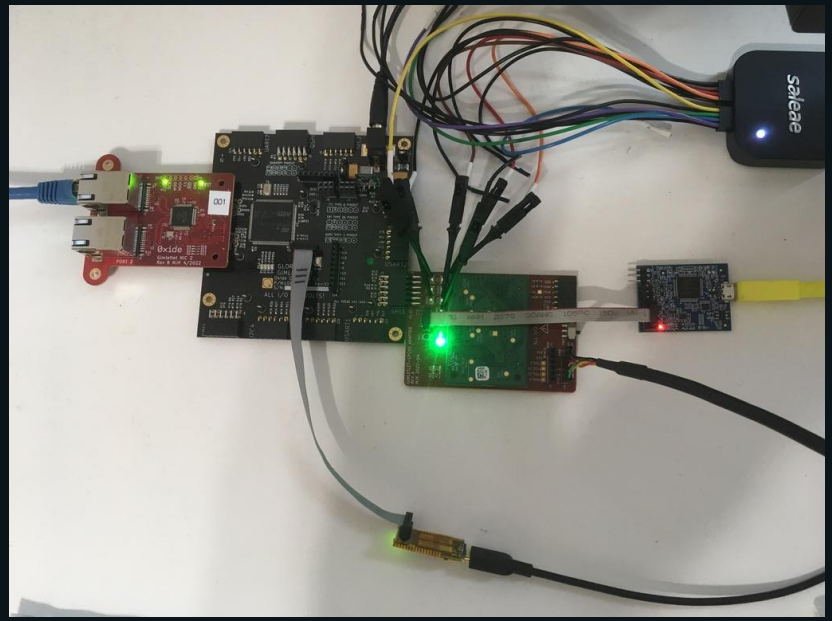
Matt Keeter
mattkeeter.com

Oxide Computer Company
matt@oxide.computer


saleae
0xide
001


0xide

Problem statement



Acknowledgements

This is the work of many people

- Laura Abbott
- Rick Altherr
- Cliff Biffle
- Bryan Cantrill
- John Gallagher
- Steve Klabnik
- Ben Stoltz
- Philip Tricca
- …and more!

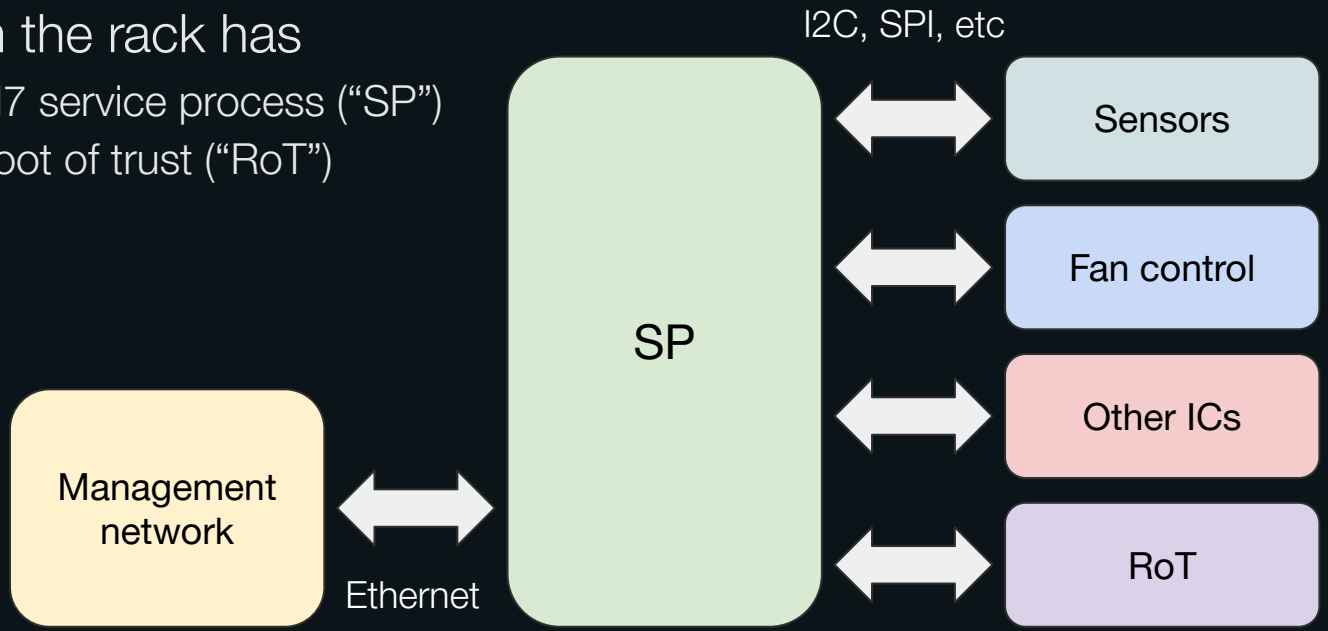
Acknowledgements

This is the work of many people

- Laura Abbott
- Rick Altherr
- Cliff Biffle
- Bryan Cantrill
- John Gallagher
- Steve Klabnik
- Ben Stoltz
- Philip Tricca
- …and more!

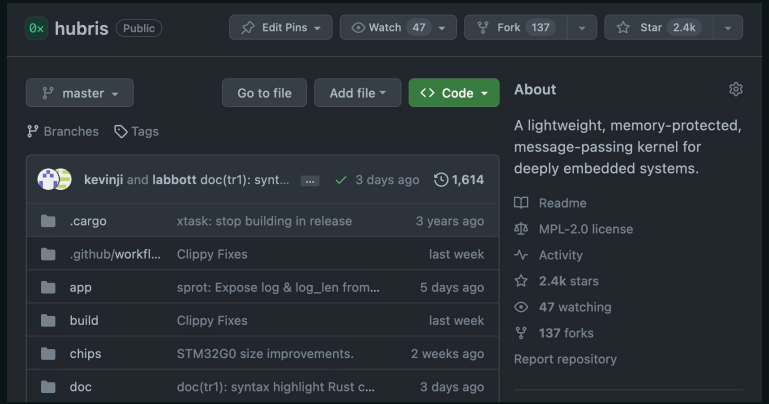
System overview

- Each server in the rack has
  - An STM32H7 service process (“SP”)
  - An LPC55 root of trust (“RoT”)

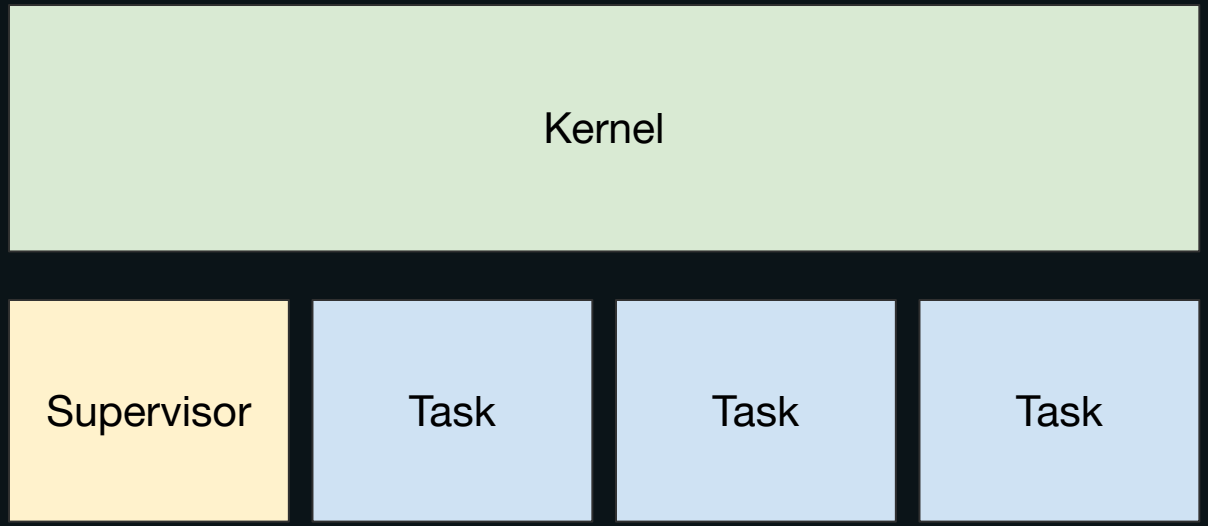


Hubris and Humility

- We wrote our own embedded OS
  - Multitasking
  - Using the MPU for memory protection
  - Message-passing between tasks
  - Written in (`no_std`) Rust
- …and a debugger to go with it
  - “Humility”
- This is all open-source!
  - github.com/oxidecomputer/hubris
  - github.com/oxidecomputer/humility



Hubris in 30 seconds



Message passing
Timers
Interrupts
Task switching

Business logic
HW peripherals
Independent & isolated

Debugging taxonomy

	Live system	Offline system
Without debug info	*Normal use* Calling well-known APIs Observing user-visible state	*Reverse engineering*
With debug info	*Typical debugging* Reading system state Tracing execution Modifying system state	*Post-mortem debugging* Core files, etc

```
humility

USAGE:
    humility [OPTIONS] [SUBCOMMAND]
```

SUBCOMMANDS:

auxflash	manipulate auxiliary flash
bankerase	Erase a bank
console-proxy	SP/host console uart proxy
dashboard	dashboard for Hubris sensor data
debugmailbox	interact with the debug mailbox on the LPC55
diagnose	analyze a system to detect common problems
doc	print command documentation
dump	generate Hubris dump
etm	commands for ARM's Embedded Trace Macrocell
exec	execute command within context of an environment
extract	extract all or part of a Hubris archive
flash	flash archive onto attached device
gdb	Attach to a running system using GDB
gpio	GPIO pin manipulation
hash	Access to the HASH block
help	Print this message or the help of the given subcommand
hiffy	manipulate HIF execution
i2c	scan for and read I2C devices
ibc	interface to the BMR491 power regulator
itm	commands for ARM's Instrumentation Trace Macrocell
jefe	influence jefe externally
lpc55gpio	GPIO pin manipulation (lpc55 variant)
manifest	print archive manifest
map	print memory map, with association of regions to tasks
monorail	Management network control and debugging
net	Management network device-side control and debugging
openocd	Run OpenOCD for the given archive
pmbus	scan for and read PMBus devices
power	show power-related information
powershelf	inspect powershelf over the management network
probe	probe for any attached devices
qspi	QSPI status, reading and writing
readmem	read and display memory region
readvar	read and display a specified Hubris variable
registers	print Hubris registers
rencm	query Renesas 8A3400X ClockMatrix parts
rendmp	Renesas digital muliphase controller operations
repl	read, eval, print, loop
reset	Reset the chip using external pins
ringbuf	read and display a specified ring buffer
rpc	execute Idol calls over a network
sbrmi	Sideband Remote Management Interface commands
sensors	query sensors and sensor data
spctrl	RoT -> SP control
spd	scan for and read SPD devices
spi	SPI reading and writing
stackmargin	calculate and print stack margins by task
stmsecure	change secure region settings on the stm32h7
tasks	list Hubris tasks
test	run Hubristest suite and parse results
tofino-eeprom	read and write to the Tofino SPI EEPROM
update	Write a software update
validate	validate presence and operation of devices
vpd	read or write vital product data (VPD)
writeword	writes one or more memory words

Design principles

Build functionality on top of small, composable primitives

```
trait Core {
    fn read_word_32(&mut self, addr: u32) -> Result<u32>;
    // ... etc
}
```

Make those primitives generic across backends

```
impl Core for ProbeCore {
    // ... implementation goes here
}
impl Core for OpenOCDCore {
    // ... implementation goes here
}
```

Reading and tracing execution

	Live system	Offline system
Without debug info	*Normal use* Calling well-known APIs Observing user-visible state	*Reverse engineering*
With debug info	*Typical debugging* Reading system state Tracing execution Modifying system state	*Post-mortem debugging* Core files, etc

`readmem` is all you need

- DWARF debug info contains
  - Data locations in RAM
  - Type information about variables
- This is all you need to implement a pretty-printer!
- Off-the-shelf debuggers do this for you
- We can do better by knowing Hubris idioms
  - Ringbufs for structured logging
  - Task tables for RTOS state
  - Structured tables of sensor data

readmem is all you need: readvar

```
$ humility readvar MAC_ADDRESS_BLOCK
humility: attached to 0483:374f:000C001F4D46500F20373033 via ST-Link V3
task_packrat::main::MAC_ADDRESS_BLOCK (0x240066a0) =
MaybeUninit<[core::option::Option<task_packrat_api::MacAddressBlock>; 1]> {
  value: [
    Some(MacAddressBlock {
      base_mac: [
        0xa8,
        0x40,
        0x25,
        0x1,
        0x1,
        0x41
      ],
      count: U16<byteorder::LittleEndian>([
          0x8,
          0x0
        ], PhantomData<byteorder::LittleEndian>),
      stride: 0x8
    })
  ]
}
```

readmem is all you need: ringbuf

```
$ humility ringbuf gimlet_seq
humility: attached to 0483:374f:000C001F4D46500F20373033 via ST-Link V3
humility: ring buffer drv_gimlet_seq_server::__RINGBUF in gimlet_seq:
 NDX LINE       GEN   COUNT   PAYLOAD
   0  173       1     1       Ice40Rails(true, true)
   1  202       1     1       Ice40PowerGoodV1P2(true)
   2  223       1     1       Ice40PowerGoodV3P3(true)
   3  265       1     1       IdentValid(false)
   4  268       1     1       ChecksumValid(false)
   5  271       1     1       Reprogram(true)
   6  285       1     1       Programming
   7  314       1     1       Programmed
   8  317       1     1       RailsOff
   9  320       1     1       Ident(0xde01)
  10  326       1     1       A2Status(0x0)
  11  349       1     446     ClockConfigWrite
  12  361       1     1       ClockConfigSuccess
  13 1190       1     1       V3P3SysA0VOut(Volts(0.0625))
  14  363       1     1       A2
  15  521       1     1       SpdDimmsFound(0x10)
  16  630       1     1       SetState(A2, A0, 0x710)
  17 1190       1     1       V3P3SysA0VOut(Volts(0.072265625))
```

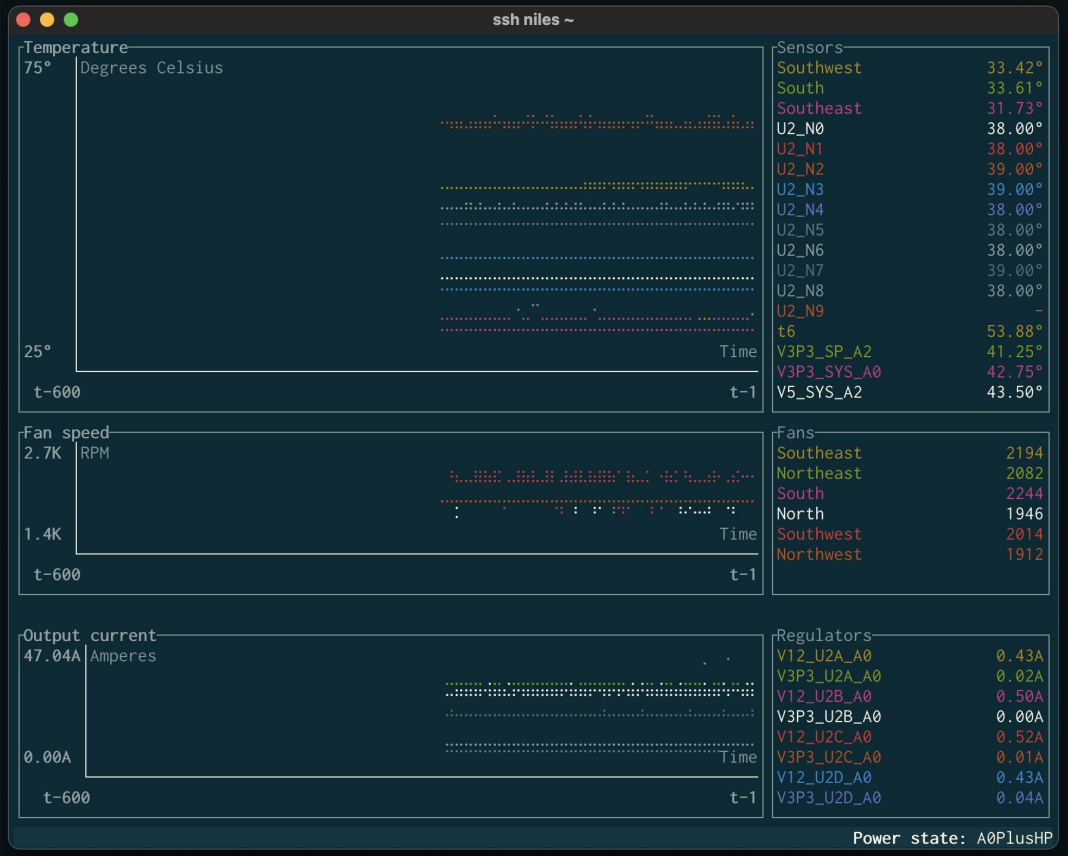
readmem is all you need: tasks list

```
$ humility tasks
humility: attached to 0483:374f:000C001F4D46500F20373033 via ST-Link V3
system time = 59929666
ID TASK                 GEN  PRI STATE
 0 jefe                 0    0   recv, notif: fault timer(T+34)
 1 net                  0    5   recv, notif: eth-irq(irq61) wake-timer(T+91)
 2 sys                  0    1   recv
 3 spi2_driver          0    3   recv
 4 i2c_driver           0    3   recv
 5 spd                  0    2   notif: i2c1-irq(irq31/irq32)
 6 packrat              0    1   recv
 7 thermal              0    5   recv, notif: timer(T+112)
 8 power                0    6   recv, notif: timer(T+459)
 9 hiffy                0    5   notif: bit31(T+154)
10 gimlet_seq           0    4   recv, notif: timer(T+98)
11 hash_driver          0    2   recv
12 hf                   0    3   recv
13 update_server        0    3   recv
14 sensor               0    4   recv, notif: timer(T+341)
15 idle                 0    8   RUNNING
```

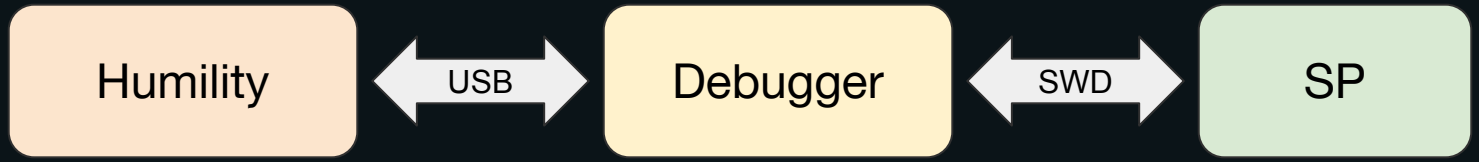
readmem is all you need: tasks backtrace

```
$ humility tasks -sl thermal
humility: attached to 0483:374f:000C001F4D46500F20373033 via ST-Link V3
system time = 60018824
ID TASK                        GEN PRI STATE
 7 thermal                     0   5 wait: reply from i2c_driver/gen0
   |
   +--->  0x24002448 0x0800c286 userlib::sys_send_stub
                    @ /hubris/sys/userlib/src/lib.rs:154
          0x240024c0 0x0800861c drv_i2c_api::I2cDevice::response_code
                    @ /hubris/drv/i2c-api/src/lib.rs:128
          0x240024c0 0x080085cc drv_i2c_api::I2cDevice::read_reg
                    @ /hubris/drv/i2c-api/src/lib.rs:158
          0x240024c0 0x080085cc drv_i2c_devices::nvme_bmc::NvmeBmc::read_temperature
                    @ /hubris/drv/i2c-devices/src/nvme_bmc.rs:56
          0x240024c0 0x0800861c task_thermal::control::TemperatureSensor::read_temp
                    @ /hubris/task/thermal/src/control.rs:67
          0x24003770 0x080095f6 task_thermal::control::ThermalControl::read_sensors
                    @ /hubris/task/thermal/src/control.rs:652
          0x24003770 0x08009810 main
                    @ /hubris/task/thermal/src/main.rs:336
```

readmem is all you need: dashboard

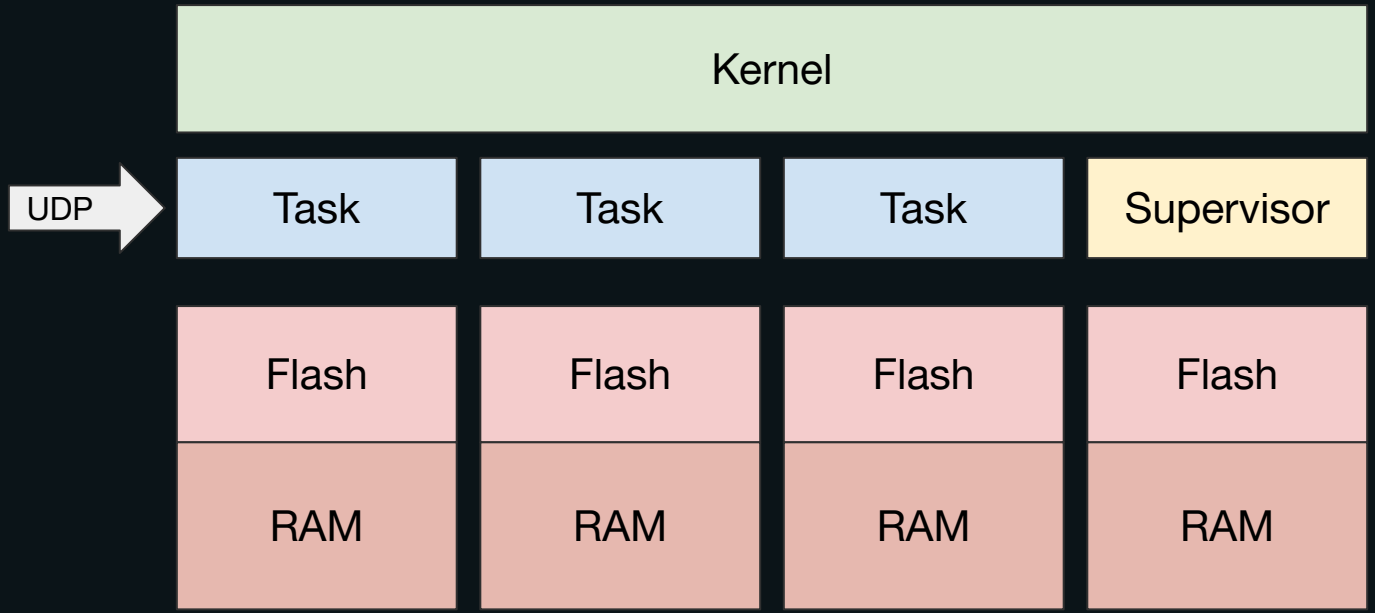


`readmem` is easy with a debugger

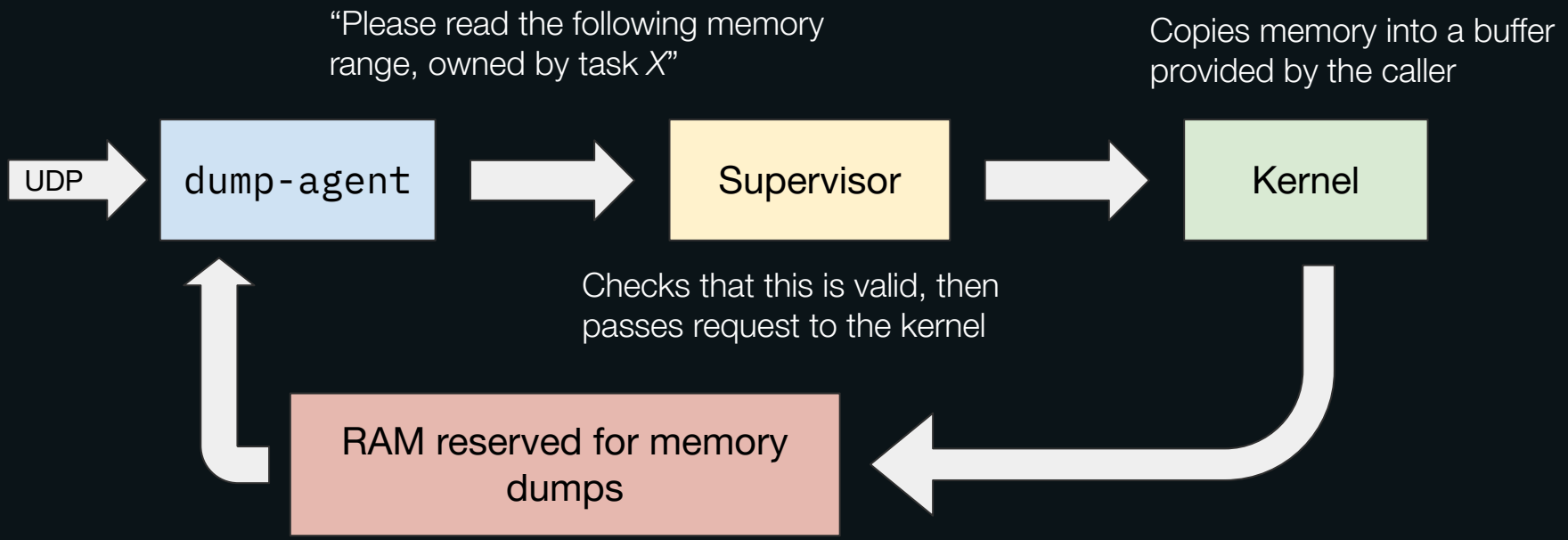


`readmem` is harder over the network

Safely reading memory over the network



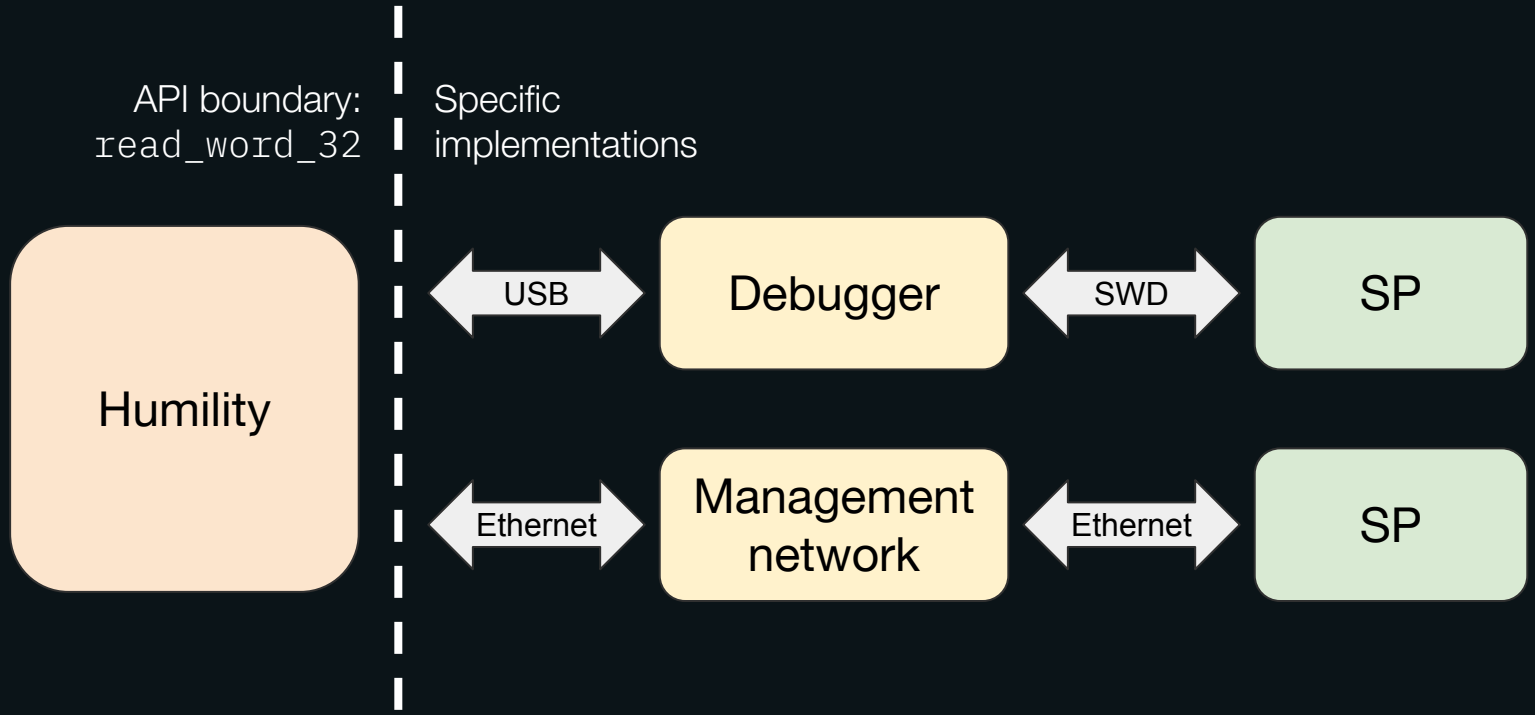
Safely reading memory over the network



Safely reading memory over the network

- There are some limitations compared to SWD!
- Can’t read supervisor memory
- Can’t read arbitrary kernel memory
  - Reading task tables is a special exception
- The memory read is not coherent across tasks
- Don’t do this on systems that contain secrets!
  - One could disable this on a per-task basis
  - Safer to not include it at all in the RoT image

This is another implementation of the `Core` trait!

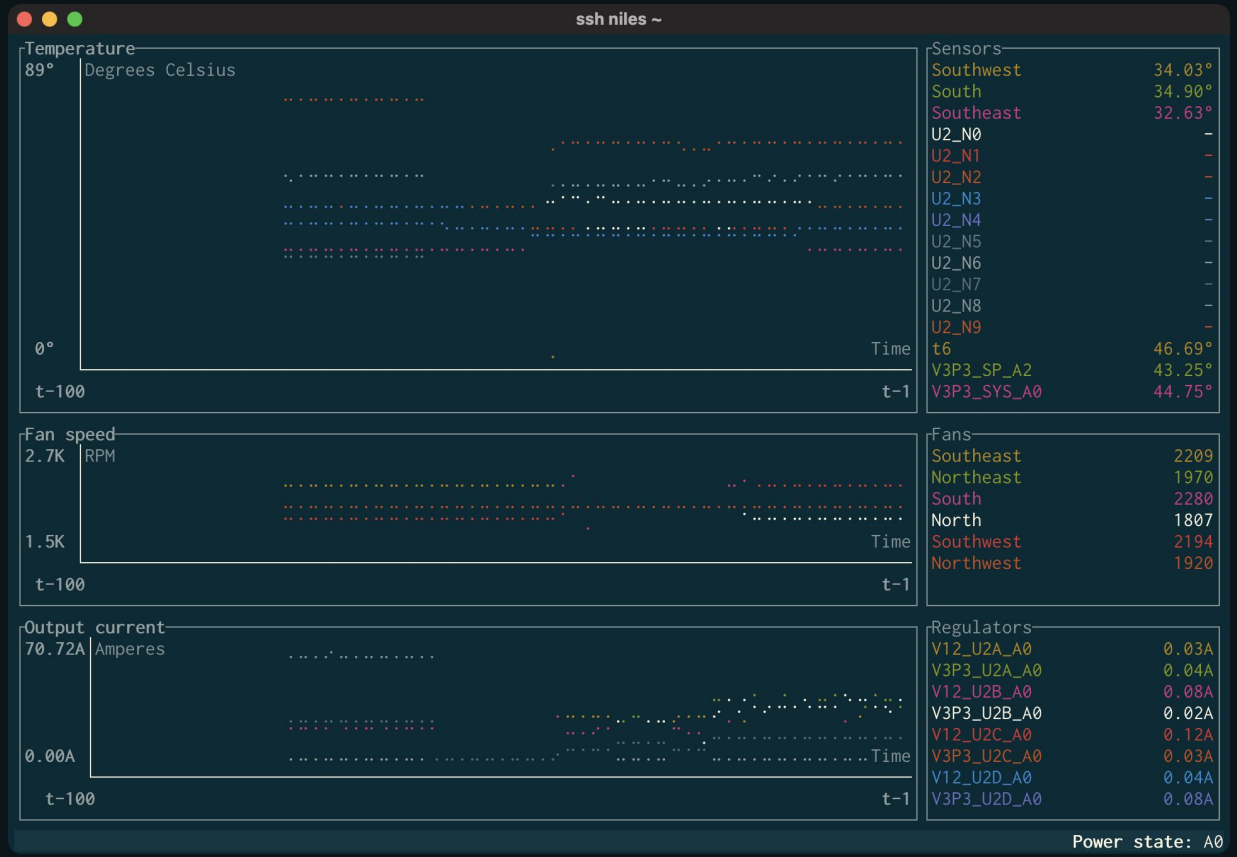


Remotely reading a `ringbuf`

```
$ humility --ip=fe80::aa40:25ff:fe01:0141%axf6 ringbuf gimlet_seq
humility: connecting to fe80::aa40:25ff:fe01:0141%axf6
humility: ring buffer drv_gimlet_seq_server::__RINGBUF in gimlet_seq:
 NDX LINE      GEN   COUNT  PAYLOAD
   0  173       1     1     Ice40Rails(true, true)
   1  202       1     1     Ice40PowerGoodV1P2(true)
   2  223       1     1     Ice40PowerGoodV3P3(true)
   3  265       1     1     IdentValid(false)
   4  268       1     1     ChecksumValid(false)
   5  271       1     1     Reprogram(true)
   6  285       1     1     Programming
   7  314       1     1     Programmed
   8  317       1     1     RailsOff
   9  320       1     1     Ident(0xde01)
  10  326       1     1     A2Status(0x0)
  11  349       1     446   ClockConfigWrite
  12  361       1     1     ClockConfigSuccess
  13 1190       1     1     V3P3SysA0VOut(Volts(0.0625))
  14  363       1     1     A2
  15  521       1     1     SpdDimmsFound(0x10)
  16  630       1     1     SetState(A2, A0, 0x710)
  17 1190       1     1     V3P3SysA0VOut(Volts(0.072265625))
```

Live demo!

Live demo


ssh niles ~
Temperature
89°
Degrees Celsius
0°
Time
t-100
t-1
Sensors
Southwest 34.03°
South 34.90°
Southeast 32.63°
U2_N0 –
U2_N1 –
U2_N2 –
U2_N3 –
U2_N4 –
U2_N5 –
U2_N6 –
U2_N7 –
U2_N8 –
U2_N9 –
t6 46.69°
V3P3_SP_A2 43.25°
V3P3_SYS_A0 44.75°
Fan speed
2.7K
RPM
1.5K
Time
t-100
t-1
Fans
Southeast 2209
Northeast 1970
South 2280
North 1807
Southwest 2194
Northwest 1920
Output current
70.72A
Amperes
0.00A
Time
t-100
t-1
Regulators
V12_U2A_A0 0.03A
V3P3_U2A_A0 0.04A
V12_U2B_A0 0.08A
V3P3_U2B_A0 0.02A
V12_U2C_A0 0.12A
V3P3_U2C_A0 0.03A
V12_U2D_A0 0.04A
V3P3_U2D_A0 0.08A
Power state: A0

Modifying system state

	Live system	Offline system
Without debug info	*Normal use* Calling well-known APIs Observing user-visible state	*Reverse engineering*
With debug info	*Typical debugging* Reading system state Tracing execution Modifying system state	*Post-mortem debugging* Core files, etc

`readmem` isn't all you need

Sometimes you want unstructured *live interaction* with a running system, beyond memory reads

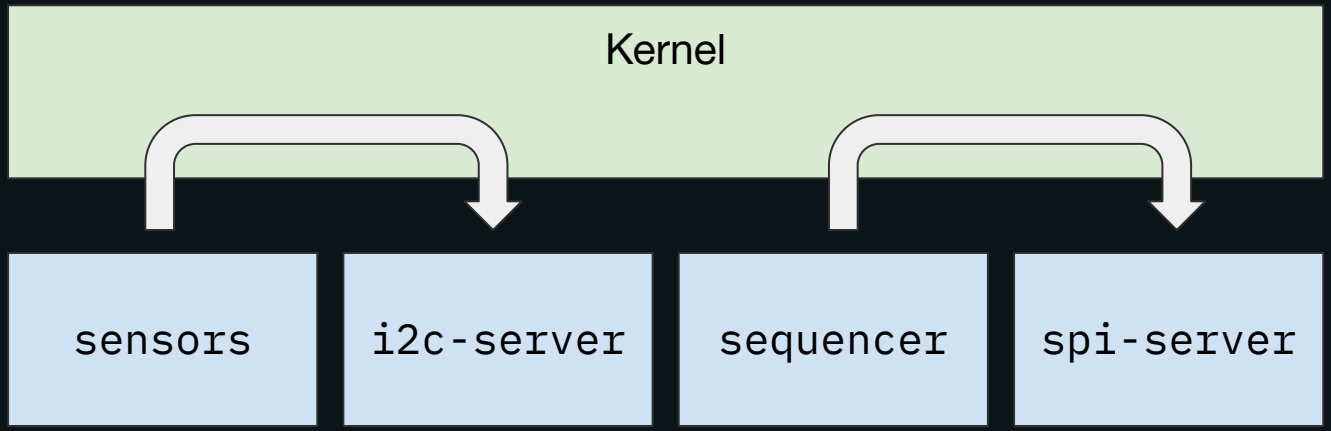
`writemem` is too aggressive

What kind of (relatively) safe interactions can we provide for a running system?

External and internal APIs

External	Internal
Formal	Informal
Stable / versioned	Unstable, no guarantees
Consistent over time	Only consistent within a single image
Hard to add	Easy to add
Almost always structured	May be structured

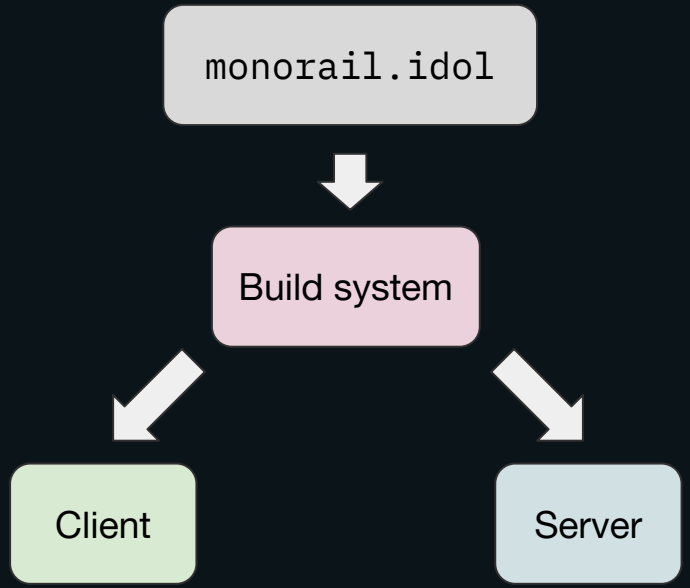
Internal APIs in Hubris



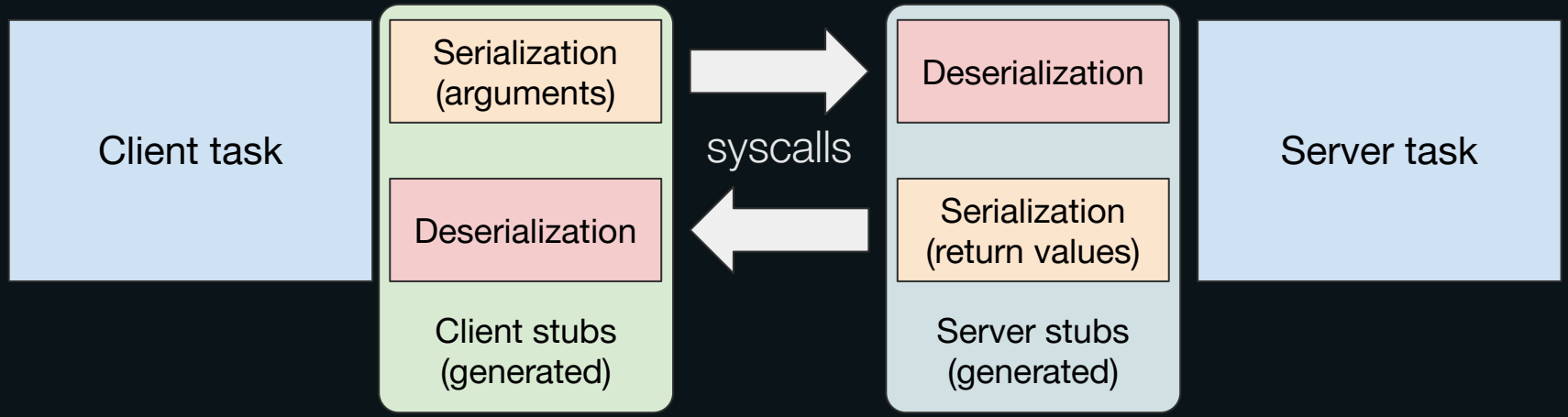
Structured internal APIs for IPC

monorail.idol

```
Interface(
 name: "Monorail",
 ops: {
   "get_port_status": (
     doc: "Read the state of a port",
     args: {
       “port": "u8",
     },
     reply: Result(
       ok: "PortStatus",
       err: CLike("MonorailError"),
     ),
   ),
```



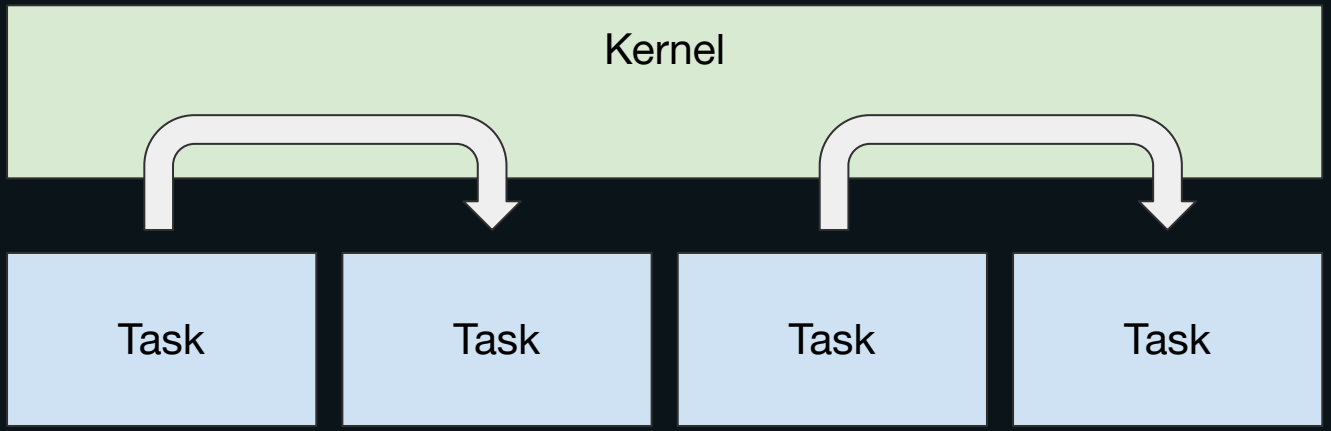
Structured internal APIs for IPC



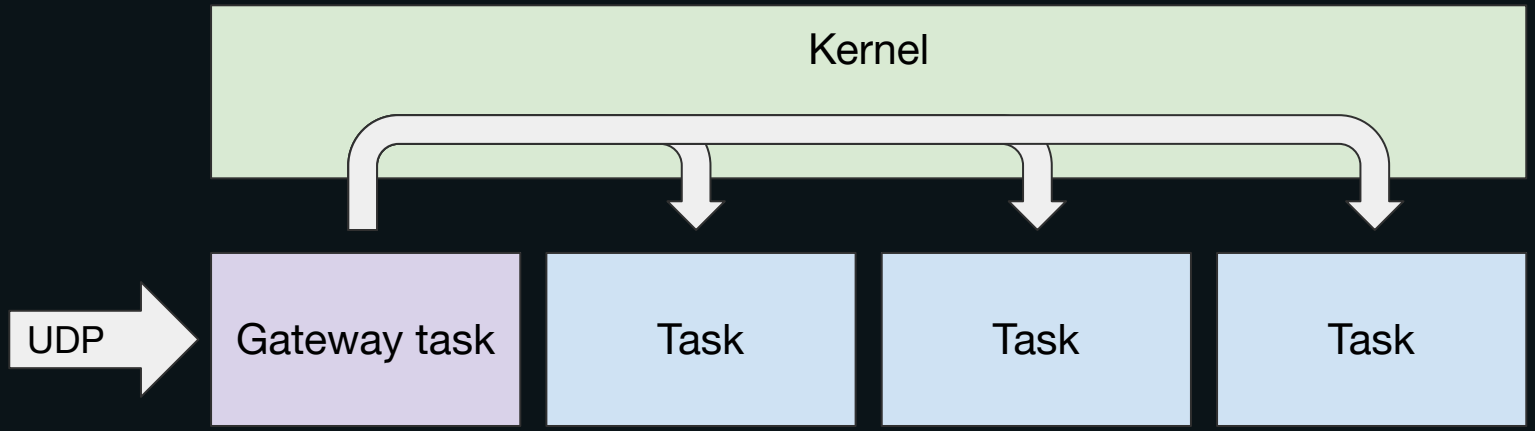
Hubris has *a lot* of structured internal APIs

```
attest.idol
auxflash.idol
caboose.idol
control-plane-agent.idol
dump-agent.idol
dumper.idol
eeprom.idol
fpga.idol
gimlet-hf.idol
gimlet-seq.idol
hash.idol
host-sp-comms.idol
ignition.idol
jefe.idol
lpc55-pins.idol
lpc55-update.idol
meanwell.idol
monorail.idol
net.idol
packrat.idol
power.idol
rng.idol
sbrmi.idol
sensor.idol
sidecar-seq.idol
sp-ctrl.idol
spi.idol
sprot.idol
stm32h7-rcc.idol
stm32h7-update.idol
stm32xx-sys.idol
syscon.idol
thermal.idol
transceivers.idol
user-leds.idol
validate.idol
vpd.idol
```

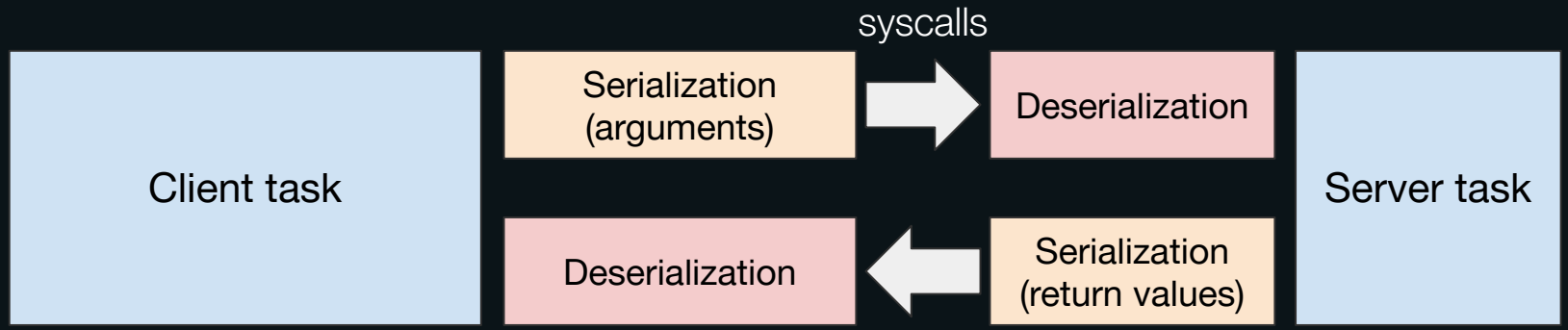
Structured internal APIs are interaction points!



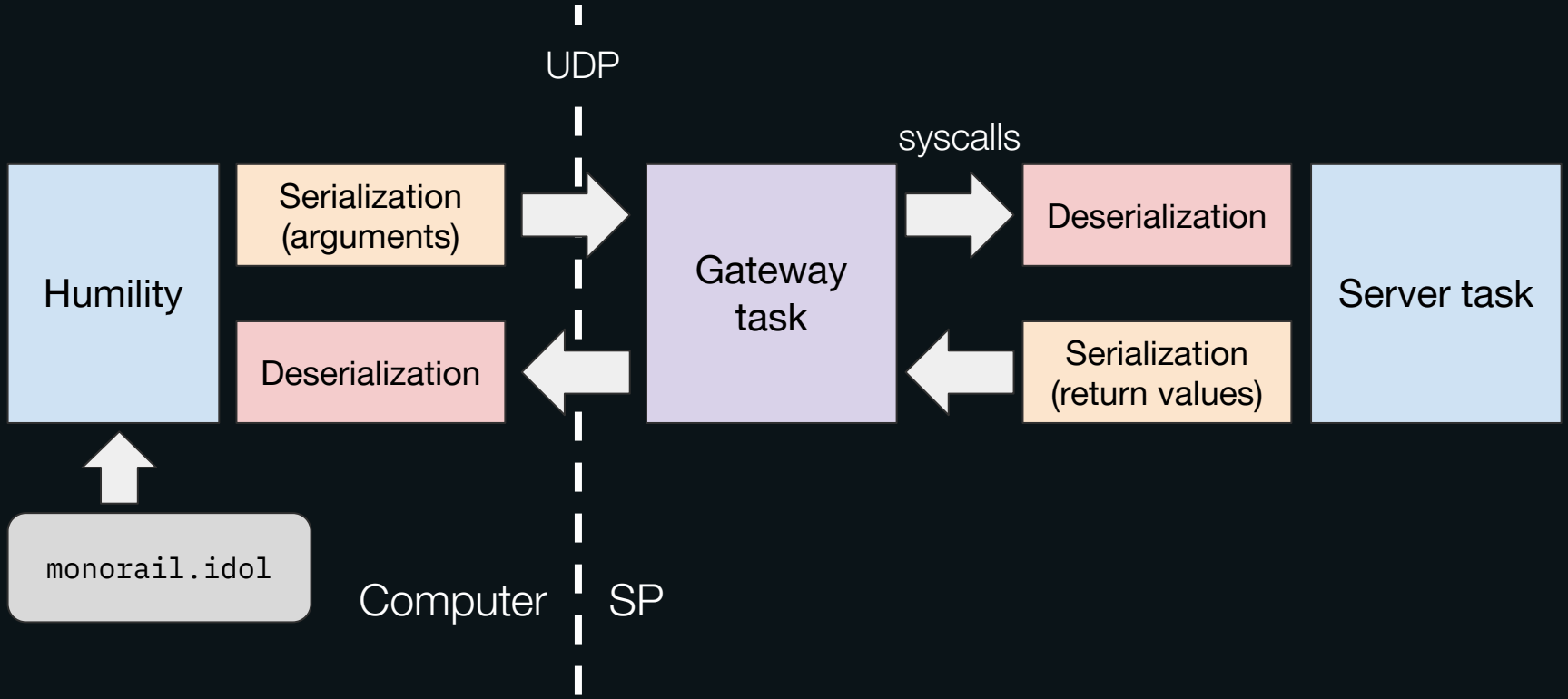
Structured internal APIs are interaction points!



Remotely calling structured internal APIs



Remotely calling structured internal APIs



Remotely calling structured internal APIs

```
$ humility hiffy -c Monorail.get_port_status -aport=40
```

Remotely calling structured internal APIs

```
$ humility hiffy -c Monorail.get_port_status -aport=40
humility: connecting to fe80::aa40:25ff:fe05:ff00%axf6
Monorail.get_port_status() => PortStatus {
    cfg: PortConfig {
        mode: Qsgmii(Speed100M),
        dev: (Dev1g, 0x10),
        serdes: (Serdes6g, 0xe)
    },
    link_up: Up
}
```

Higher-level debug UIs

```
$ humility monorail status -p0,40,41,42,43,48,49
humility: connecting to fe80::aa40:25ff:fe05:ff00%axf6
PORT | MODE    SPEED  DEV     SERDES  LINK |   PHY    MAC LINK  MEDIA LINK
-----|--------------------------------------|-------------------------------
 0   | SGMII   100M   1G_0    1G_1    down | --       --        --
 40  | QSGMII  100M   1G_16   6G_14   up   | VSC8504  up        up
 41  | QSGMII  100M   1G_17   6G_14   up   | VSC8504  up        down
 42  | QSGMII  100M   1G_18   6G_14   up   | VSC8504  up        down
 43  | QSGMII  100M   1G_19   6G_14   up   | VSC8504  up        down
 48  | SGMII   100M   2G5_24  1G_0    up   | --       --        --
 49  | BASEKR  --     10G_0   10G_0   up   | --       --        --
```

From live to post-mortem debugging

	Live system	Offline system
Without debug info	*Normal use* Calling well-known APIs Observing user-visible state	*Reverse engineering*
With debug info	*Typical debugging* Reading system state Tracing execution Modifying system state	*Post-mortem debugging* Core files, etc

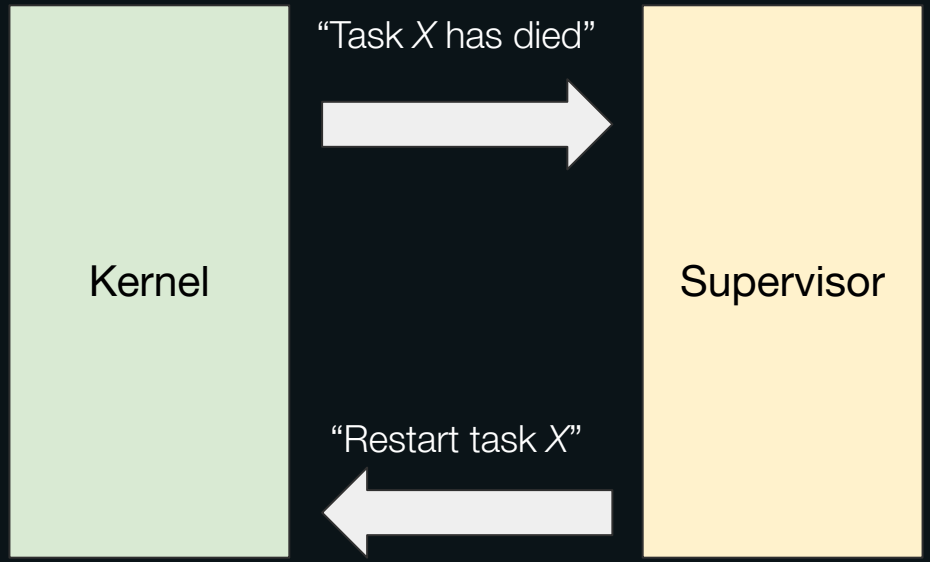
From live to post-mortem debugging

	Live system	Offline system
Without debug info	*Normal use* Calling well-known APIs Observing user-visible state	*Reverse engineering*
With debug info	*Typical debugging* Reading system state Tracing execution Modifying system state	*Post-mortem debugging* Core files, etc

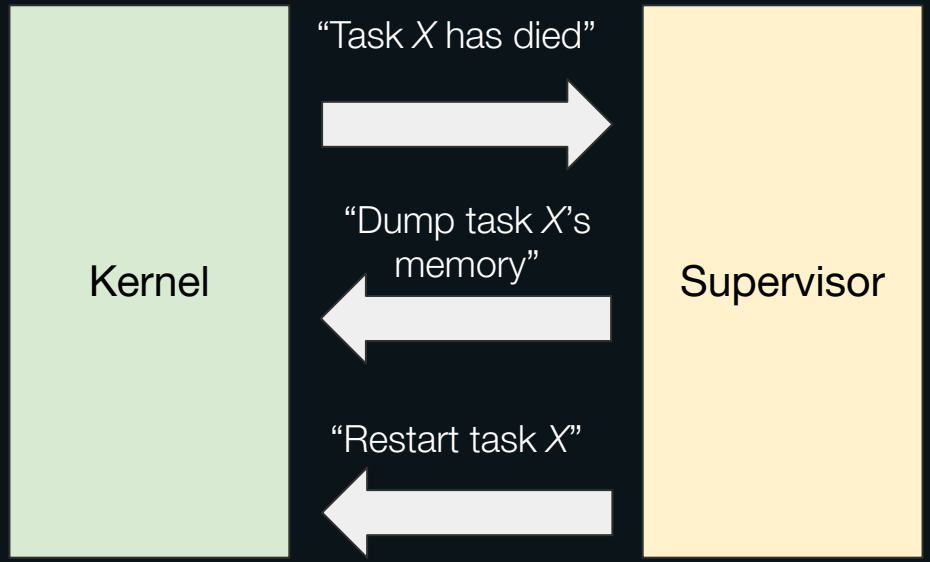
Individual tasks can crash

- Despite memory safety, crashes are possible
  - Stack overflow
  - Invalid `unwrap()`
  - Explicit `panic!()`
- The supervisor task is notified of crashes
  - …and can ask the kernel to restart a task
  - All tasks are restarted automatically by default
  - This can be changed on a per-task basis

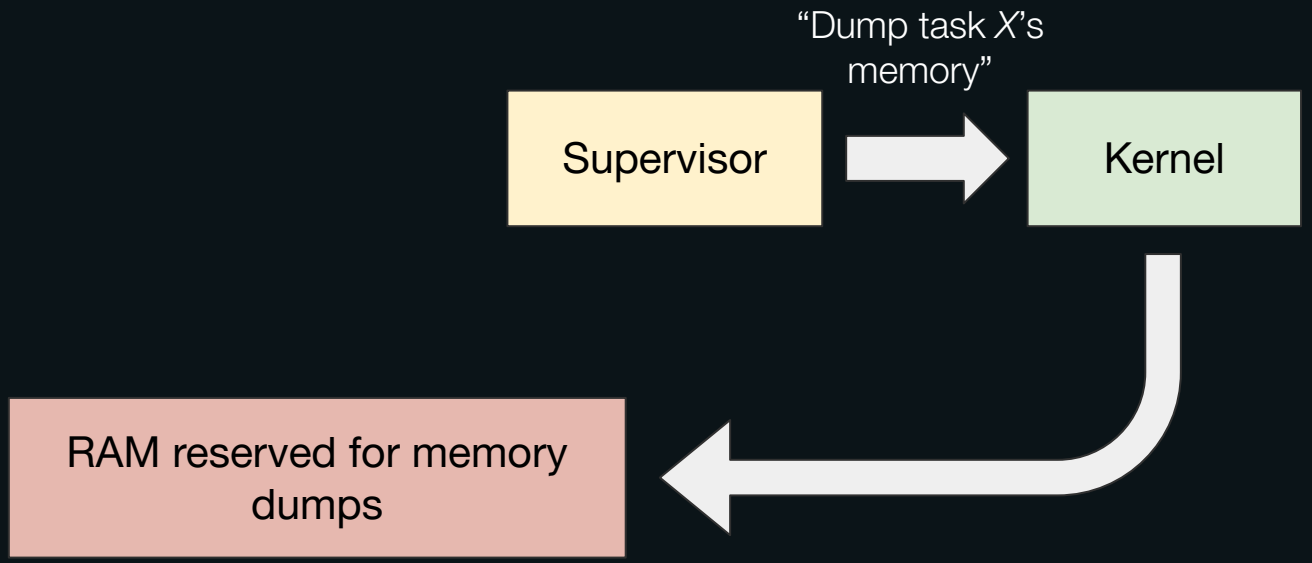
Task crashes and restarts



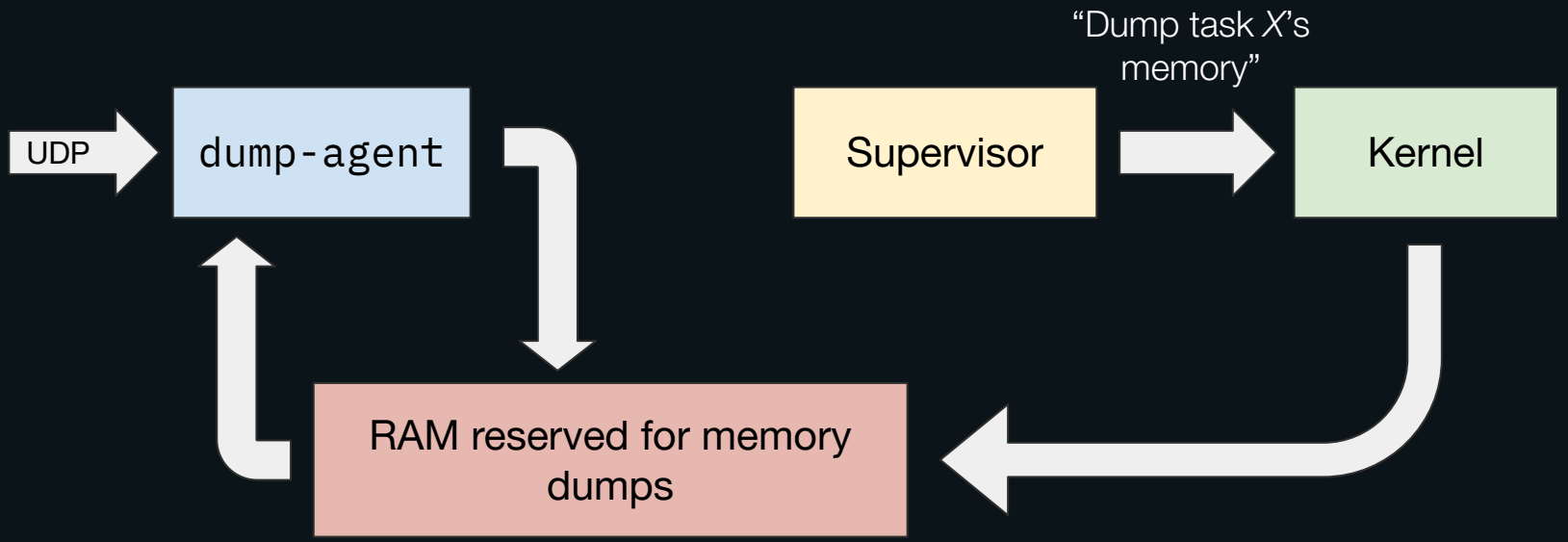
Automatic task crash dumps



Automatic task crash dumps



Automatic task crash dumps



Automatic crash dumps

```
$ humility dump -l
humility: connecting to fe80::aa40:25ff:fe01:0141%axf6
humility: using UDP dump agent
AREA TASK                TIME       SIZE
   0 net                 64295000   55080
  14 sensor              64399300   5720
```

Automatic crash dumps

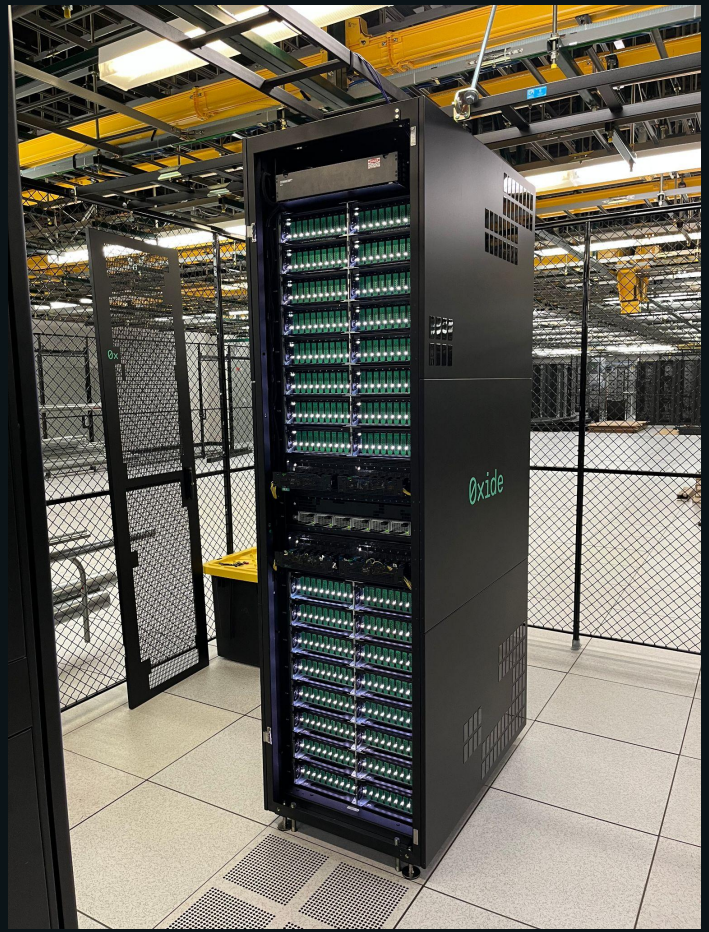
```
$ humility dump -l
humility: connecting to fe80::aa40:25ff:fe01:0141%axf6
humility: using UDP dump agent
AREA TASK              TIME       SIZE
   0 net               64295000   55080
  14 sensor            64399300   5720
$ humility dump --all
humility: connecting to fe80::aa40:25ff:fe01:0141%axf6
humility: using UDP dump agent
humility: dumping net (area 0)
humility: pulled 53.79KB in 0 seconds
humility: dumping to hubris.core.net.0
humility: dumped 947.99KB in 0 seconds
humility: dumping sensor (area 14)
humility: pulled 5.59KB in 0 seconds
humility: dumping to hubris.core.sensor.0
humility: dumped 875.99KB in 0 seconds
humility: resetting dump agent state
```

Crash dumps are another backend!

```
$ humility --dump=hubris.core.net.0 ringbuf
humility: attached to dump
humility: ring buffer ksz8463::__RINGBUF in net:
 NDX LINE     GEN  COUNT  PAYLOAD
   9  148     4    1      Write(P2VIDCR, 0x302)
  10  148     4    1      Write(P3VIDCR, 0x3ff)
  11  134     4    1      Read(P1CR1, 0x0)
  12  148     4    1      Write(P1CR1, 0x202)
  13  134     4    1      Read(P2CR1, 0x0)
  14  148     4    1      Write(P2CR1, 0x202)
  15  134     4    1      Read(P3CR1, 0x0)
   0  148     5    1      Write(P3CR1, 0x4)
   1  148     5    1      Write(SGCR9, 0xa)
   2  134     5    1      Read(P3CR2, 0x607)
   3  148     5    1      Write(P3CR2, 0x4607)
   4  134     5    1      Read(SGCR2, 0xf0)
   5  148     5    1      Write(SGCR2, 0x80f0)
   6  148     5    1      Write(CIDER, 0x1)
   7  134     5    1      Read(P1MBSR, 0x780c)
   8  134     5    1      Read(P2MBSR, 0x780c)
```

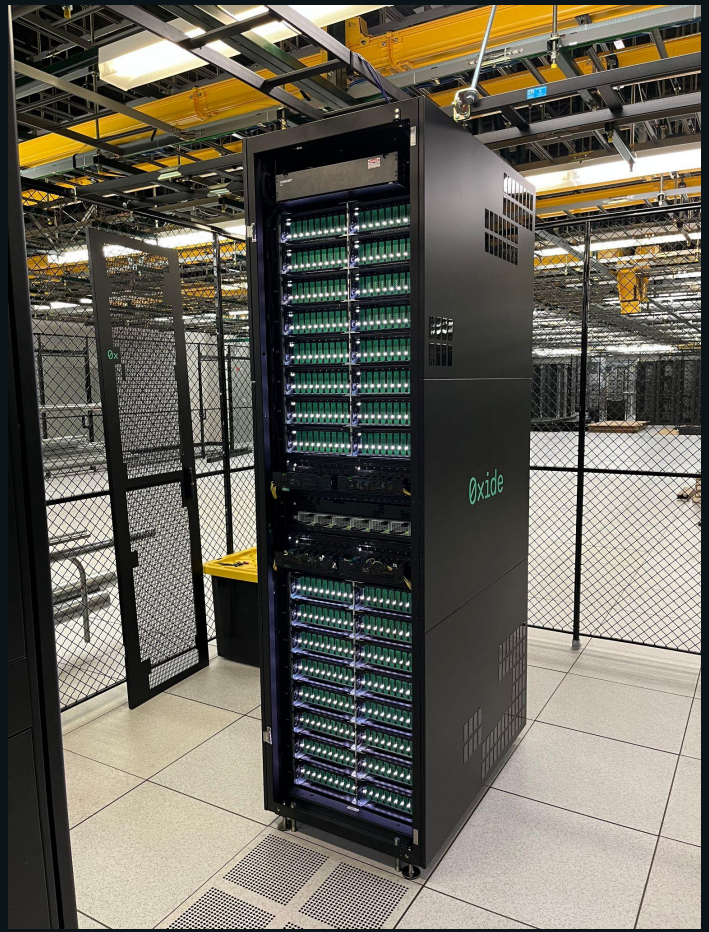
It’s too bad about those limitations…

- Can’t read supervisor memory
- Can’t read arbitrary kernel memory
- The memory read is not coherent across tasks



“There are no debuggers plugged in”

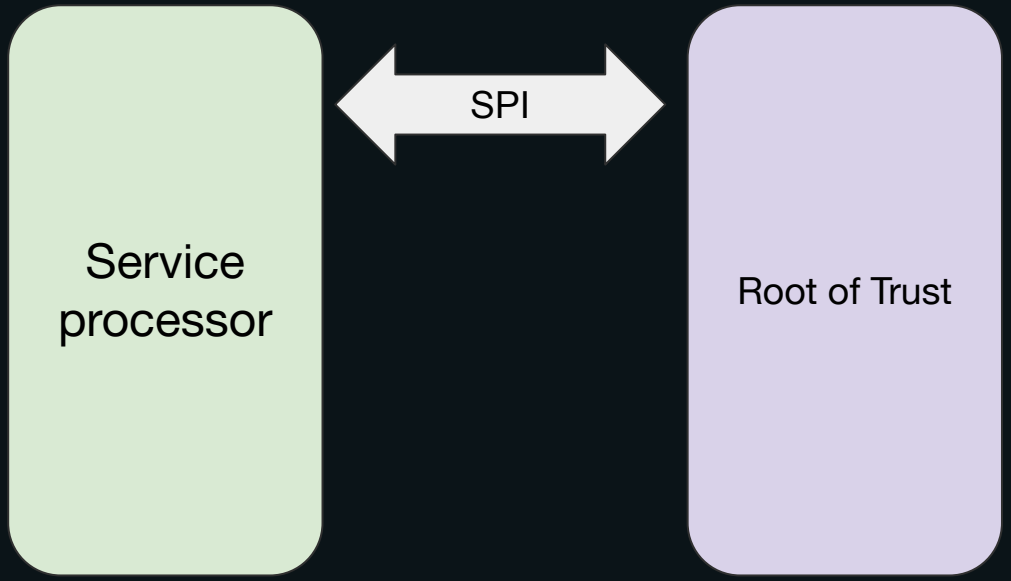
- Me, 20 minutes ago



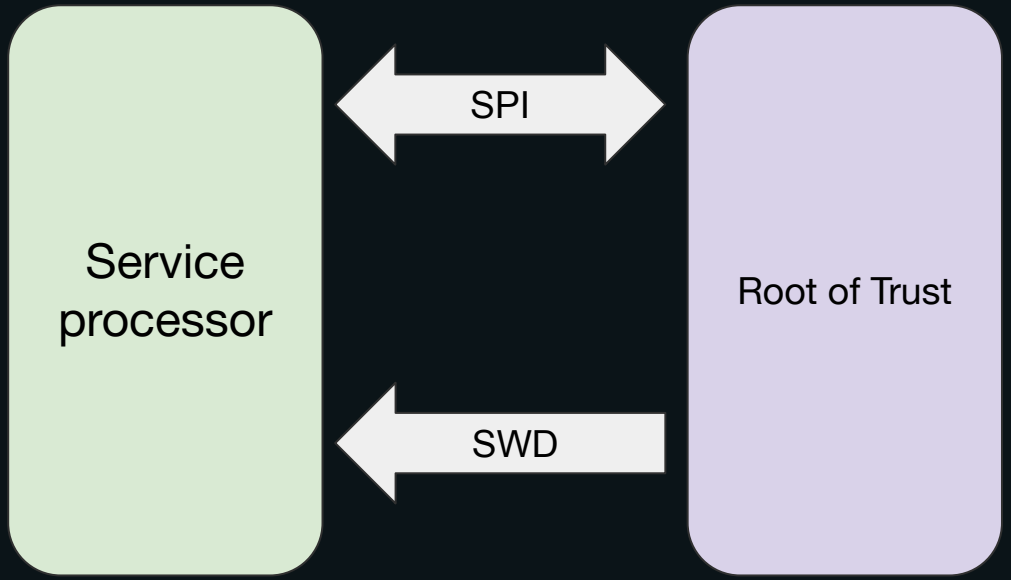
“There are no debuggers plugged in”

- Me, 20 minutes ago, **lying**

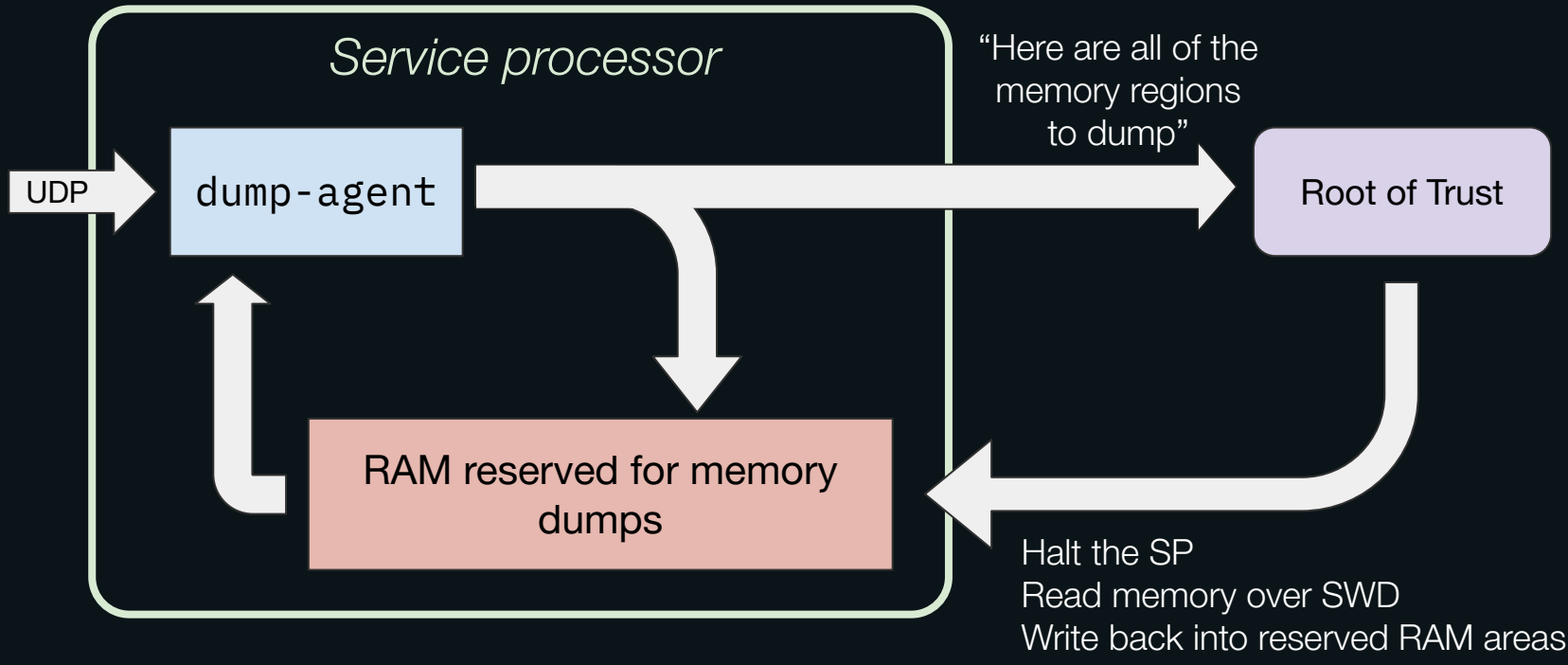
The Root of Trust



The Root of Trust



Full-system memory dumps



Full-system memory dumps

```
$ humility dump
humility: connecting to fe80::aa40:25ff:fe04:03c0%axf6
humility: using UDP dump agent
humility: initializing dump agent state
humility: initializing segments
humility: taking dump; target will be stopped for ~20 seconds
humility: pulled 193.82KB in 0 seconds
humility: resetting dump agent state
humility: dumping to hubris.core.0
humility: dumped 1.18MB in 18 seconds
```

Full-system memory dumps are another backend

```
$ humility --dump=hubris.core.0 tasks
humility: attached to dump
system time = 82074834
ID TASK                 GEN PRI STATE
 0 jefe                 0   0   recv, notif: fault timer(T+66)
 1 net                  0   5   recv, notif: eth-irq(irq61) wake-timer(T+421)
 2 sys                  0   1   recv
 3 spi2_driver          0   3   recv
 4 i2c_driver           0   3   recv
 5 spd                  0   2   notif: i2c1-irq(irq31/irq32)
 6 packrat              0   1   recv
 7 thermal              0   5   recv, notif: timer(T+900)
 8 power                0   6   recv, notif: timer(T+417)
 9 hiffy                0   5   notif: bit31(T+215)
10 gimlet_seq           0   4   recv, notif: timer(T+36)
11 hash_driver          0   2   recv
12 hf                   0   3   recv
13 update_server        0   3   recv
14 sensor               0   4   recv, notif: timer(T+173)
15 idle                 0   8   RUNNING
```

Full-system memory dumps are another backend

```
$ humility --dump=hubris.core.0 ringbuf jefe
humility: attached to dump
humility: ring buffer task_jefe::dump::__RINGBUF in jefe:
 NDX LINE       GEN  COUNT PAYLOAD
   3   96        83        1 GetDumpArea(0x0)
   4   97        83        1 Base(0x30020000)
   5   96        83        1 GetDumpArea(0x0)
   6   97        83        1 Base(0x30020000)
   7   96        83        1 GetDumpArea(0x0)
   0   97        84        1 Base(0x30020000)
   1   96        84        1 GetDumpArea(0x0)
   2   97        84        1 Base(0x30020000)

$ humility --dump=hubris.core.0 readvar KERNEL_HAS_FAILED
humility: attached to dump
kern::fail::KERNEL_HAS_FAILED (0x24000408) = false
```

Full-system memory dumps are another backend

- ✓ Can read supervisor memory
- ✓ Can read arbitrary kernel memory
- ✓ The memory read is coherent across tasks
- ■ Requires hardware support

Wrapping up

- You can write your own debug tools
  - No one will stop you
- `readmem` + domain knowledge = many useful tools!
- Structured internal APIs can be entry points for informal debugging
- Adding core dumps to an embedded system is surprisingly feasible
- Designing your debug tools for multiple backends pays dividends
  - Attached
  - Networked
  - Memory dumps

Unplugging the Debugger: Live and post-mortem debugging in a remote system

Matt Keeter
mattkeeter.com

Oxide Computer Company
matt@oxide.computer