ez-clang

experimental C++ REPL for bare metal

CaaS Monthly Meeting, Stefan Gränitz, 10 March 2022

ez-clang

In a nutshell

- Cling-based REPL prompt for C++ and meta commands
- Code runs on the connected development board
- Only few Cling features work yet: no transaction rollback, some error recovery
- Linux only: works with Ubuntu 20.04 LTS
- Firmware built with PlatformIO and GCC
- Current development state of mind: go fast and break things

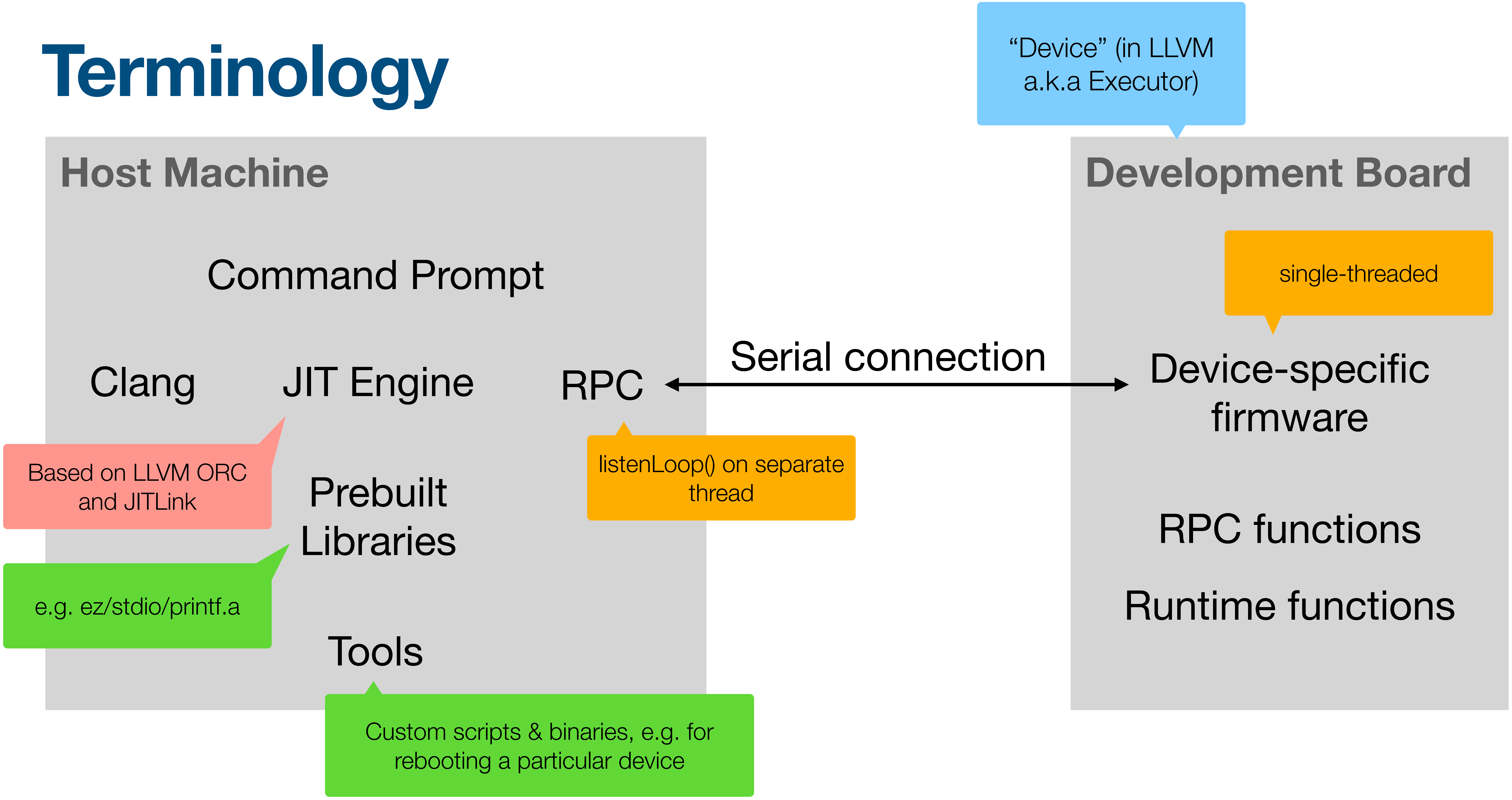
Demo time!

experimental C++ REPL for bare metal

Schedule

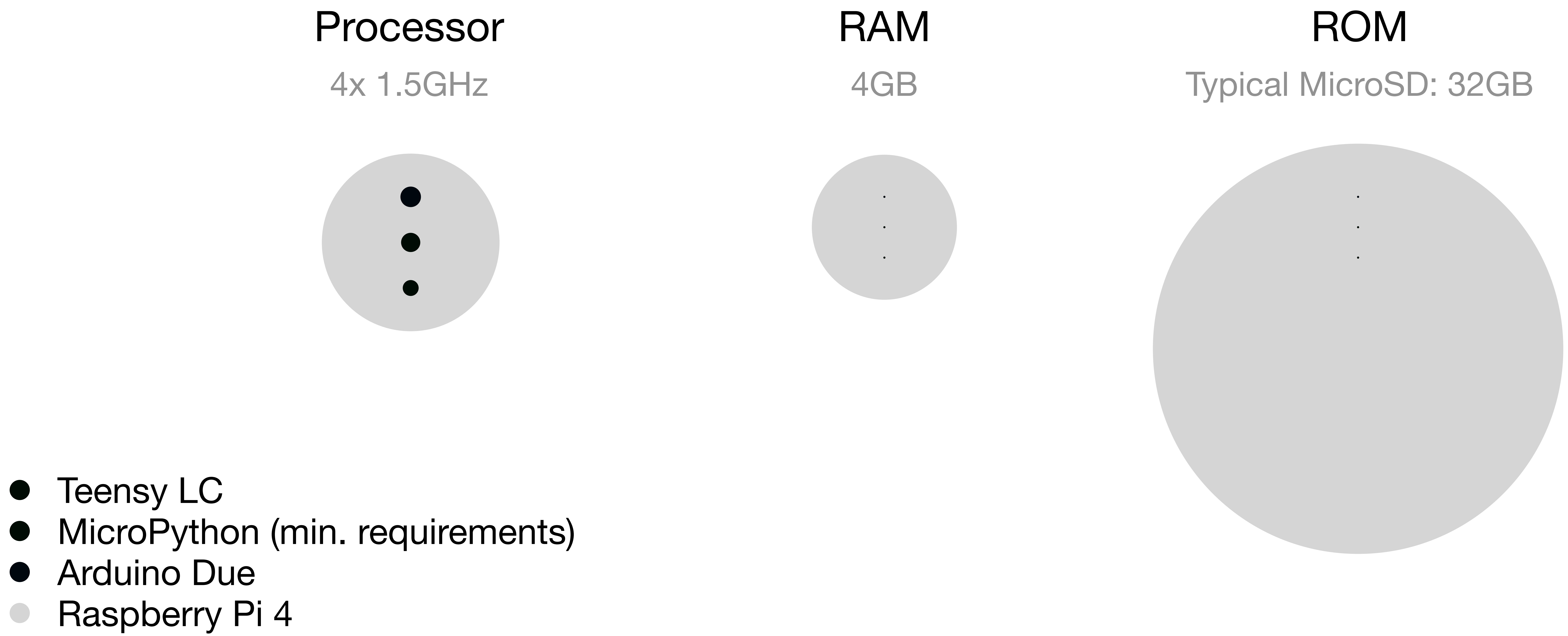
- Terminology
- Hardware Dimensions
- Comparable Projects
- REPL Pipeline
- Device Firmware
- RPC Pipeline
- In- vs. out-of-process example
- Challenges
- Feedback / Outlook

Terminology



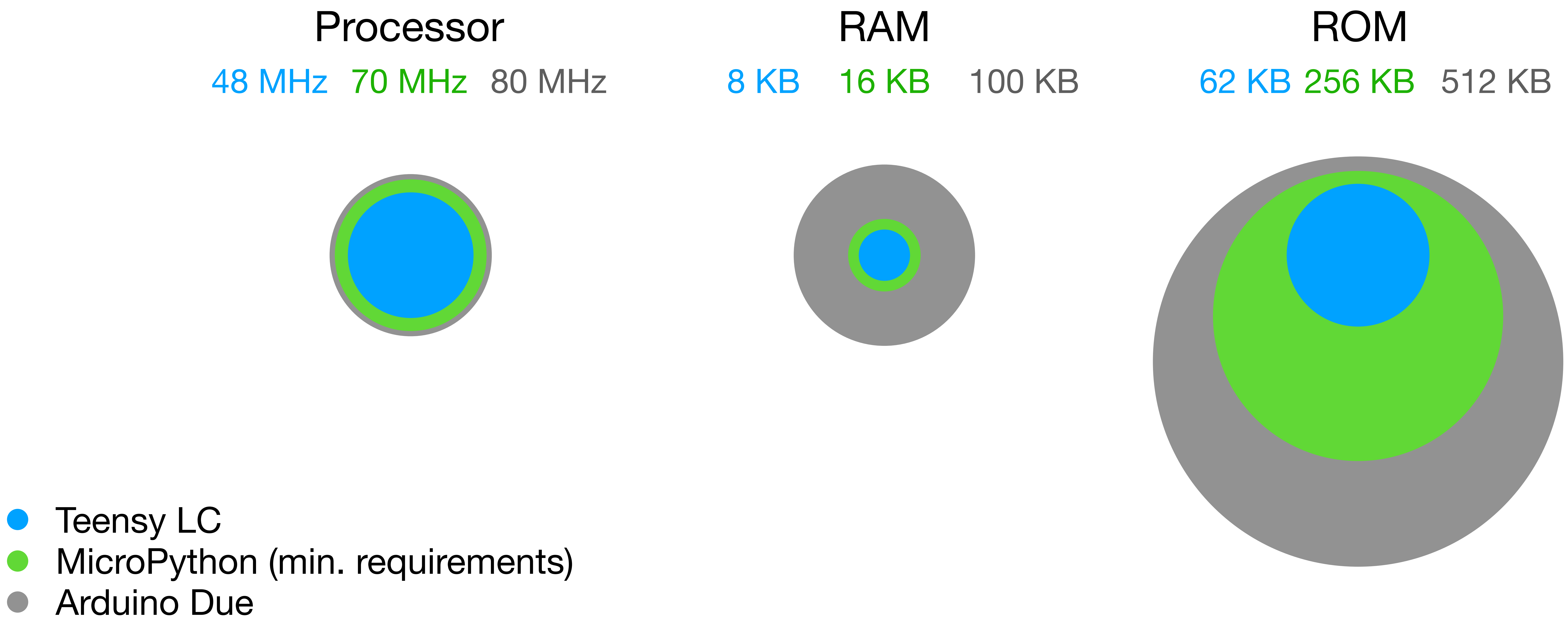
Hardware Dimensions

Raspberry Pi 4 vs. Bare Metal Microcontrollers



Hardware Dimensions

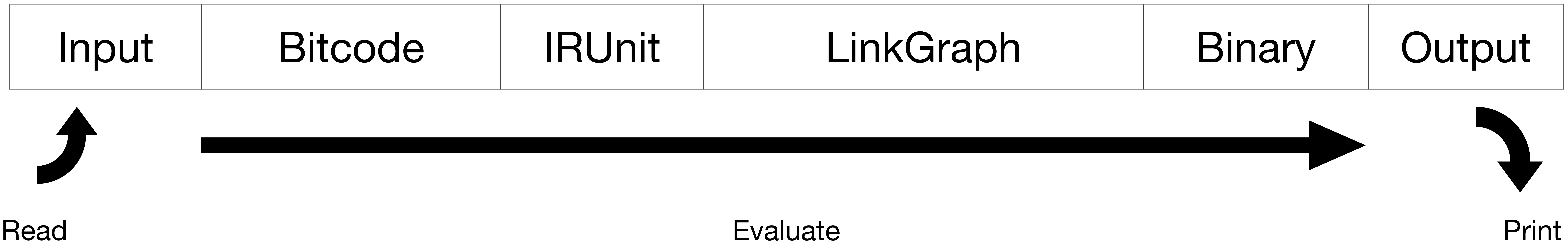
Bare Metal Microcontrollers



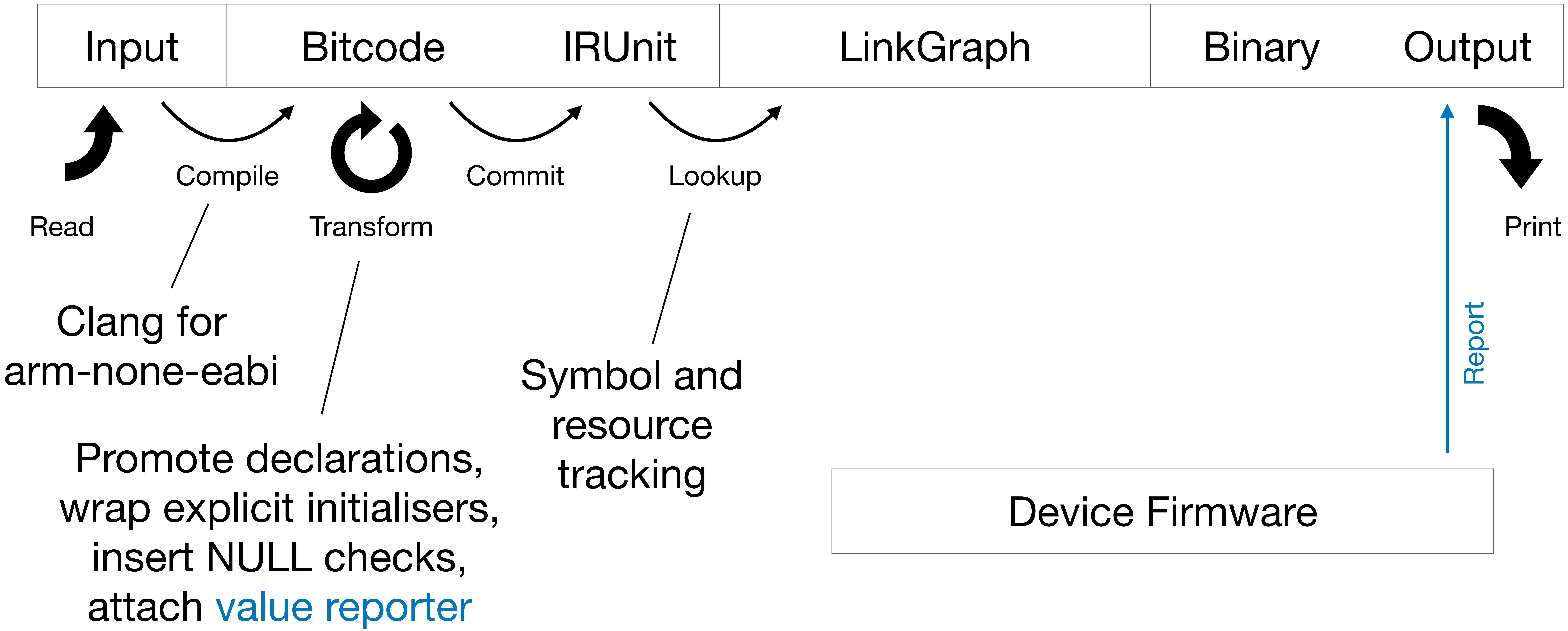
Comparable Projects

	MicroPython	ez-clang
Language	(Reduced) Python Dialect	Standard C++
Standard Libraries	Subset of Python Stdlib Feature-set depends on device capacity	No complete C Stdlib GCC has Newlib instead of glibc No complete C++ STL Partial adaptations like ETL
Execution Model	Interpreted Interpreter on device	Compiled, Toolchain on host Minimal stub on device

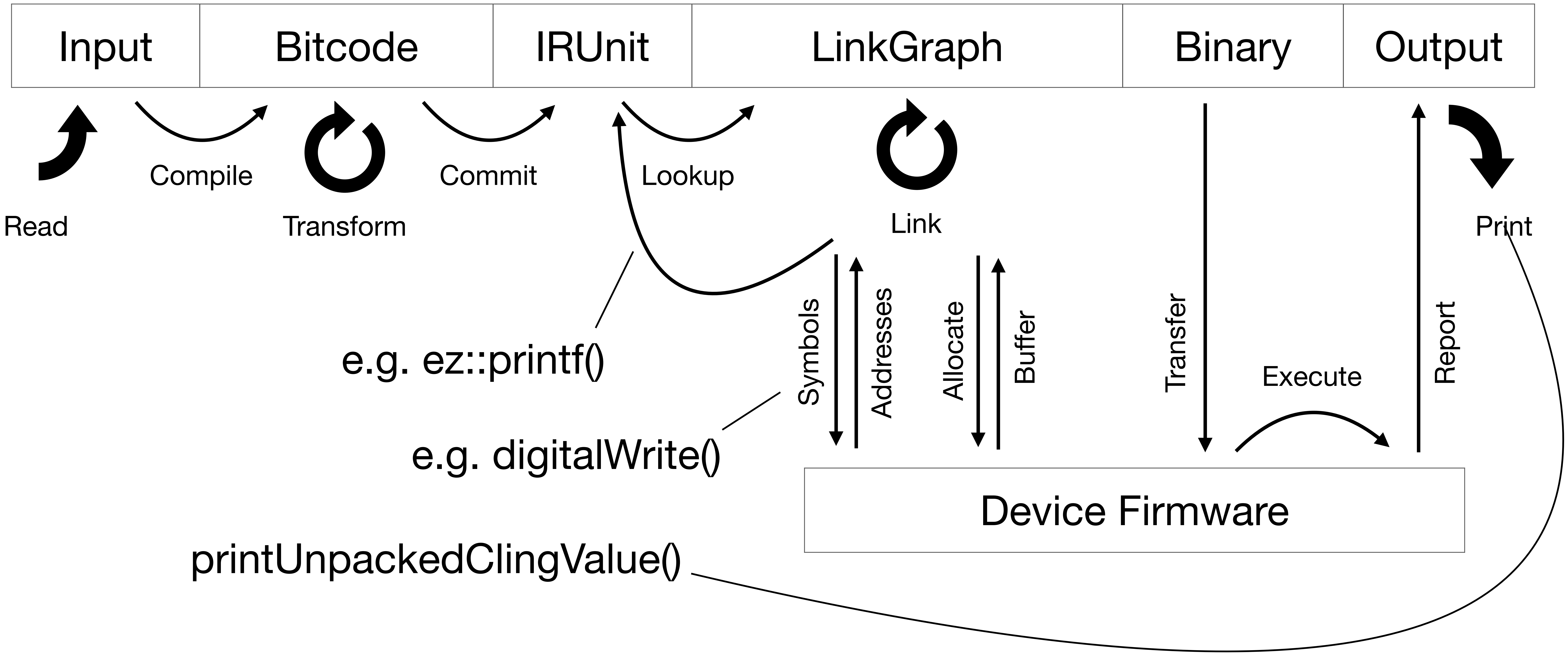
REPL Pipeline



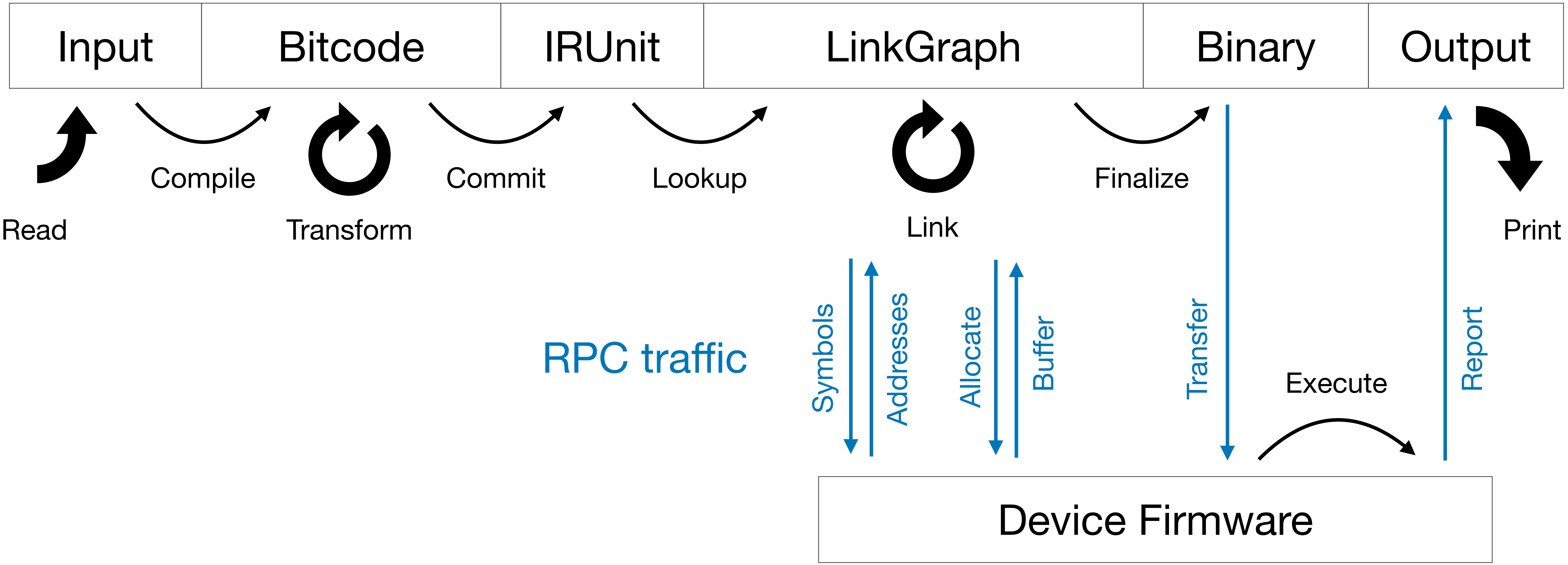
REPL Pipeline



REPL Pipeline



REPL Pipeline



Device firmware

RPC functions

```
char *__ez_clang_rpc_mem_read_cstring(const char *InputBegin, size_t InputSize) {
  uint32_t Addr;
  uint32_t Size = readAddr(InputBegin, Addr);
  assert(Size == InputSize, "We expect a single address parameter");

  const char *Str = addrToDataPointer(Addr);
  char *Resp = responseAcquire(8 + strlen(Str));
  Resp += writeString(Resp, Str);
  return responseFinalize(Resp);
}
```

- Invoked from the host side via RPC
- Serialized parameters and return value
- In ez-clang synchronous (host blocks), because device single-threaded

Device firmware

Runtime functions

```
void __ez_clang_report_value(uint32_t SeqNo, const char *Blob, size_t Size) {
  // The controller uses this function to print expression values. It knows the
  // QualType for the data in this blob.
  sendMessage(ReportValue, SeqNo, Blob, Size);
}
```

- Directly invoked from other functions on the device side
- Entrypoints: `__ez_clang_rpc_execute` or interrupt handler
- Bundled in firmware, defined in REPL or loaded from precompiled archives like `ez/stdio/printf.a`
- Can send asynchronous messages to host (if required functions are exposed)

Device firmware

Pitfalls: Invoke runtime function from REPL

```
teensylc> auto *str = "abc";
(const char *) 0x20071100
teensylc> __ez_clang_report_string(str, 3);
input_line_6:2:2: error: use of undeclared identifier '__ez_clang_report_string'
 __ez_clang_report_string(str, 3);
 ^

teensylc> 
```

- Clang needs a function declaration to compile the expression!
- Include a header or declare the function manually:

```
#include <cstddef>
extern "C" void __ez_clang_report_string(const char *Data, size_t Size);
```

RPC Messages

Type	Device		Host
Setup	Send	→	Receive
Hangup	Send	⇄	Confirm
Call	Execute	←	Send
Result	Send	→	Dispatch
ReportValue	Send	→	Dispatch
ReportString	Send	→	Print

In principle, ORC allows bidirectional calls

ez-clang extensions

RPC Messages

Type	Device		Host
Setup	Send	→	Receive
Hangup	Send	⇄	Confirm
Call	Execute	←	Send
Result	Send	→	Dispatch
ReportValue	Send	→	Dispatch
ReportString	Send	→	Print

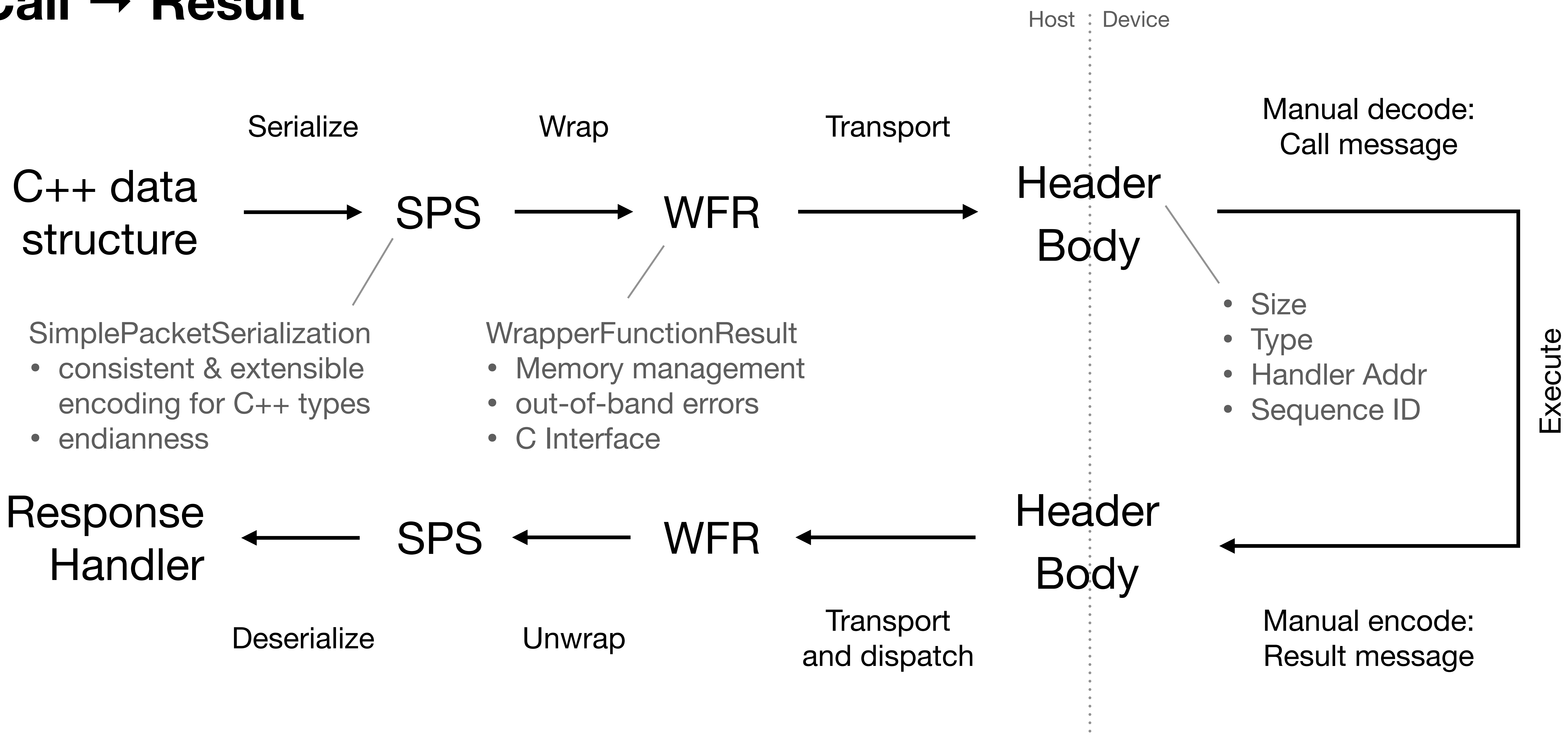
Result of e.g.:
`__ez_clang_rpc_execute()`
Payload: llvm::Error

Result of user expression
Payload: raw data block

Asynchronous info
Payload: C-String

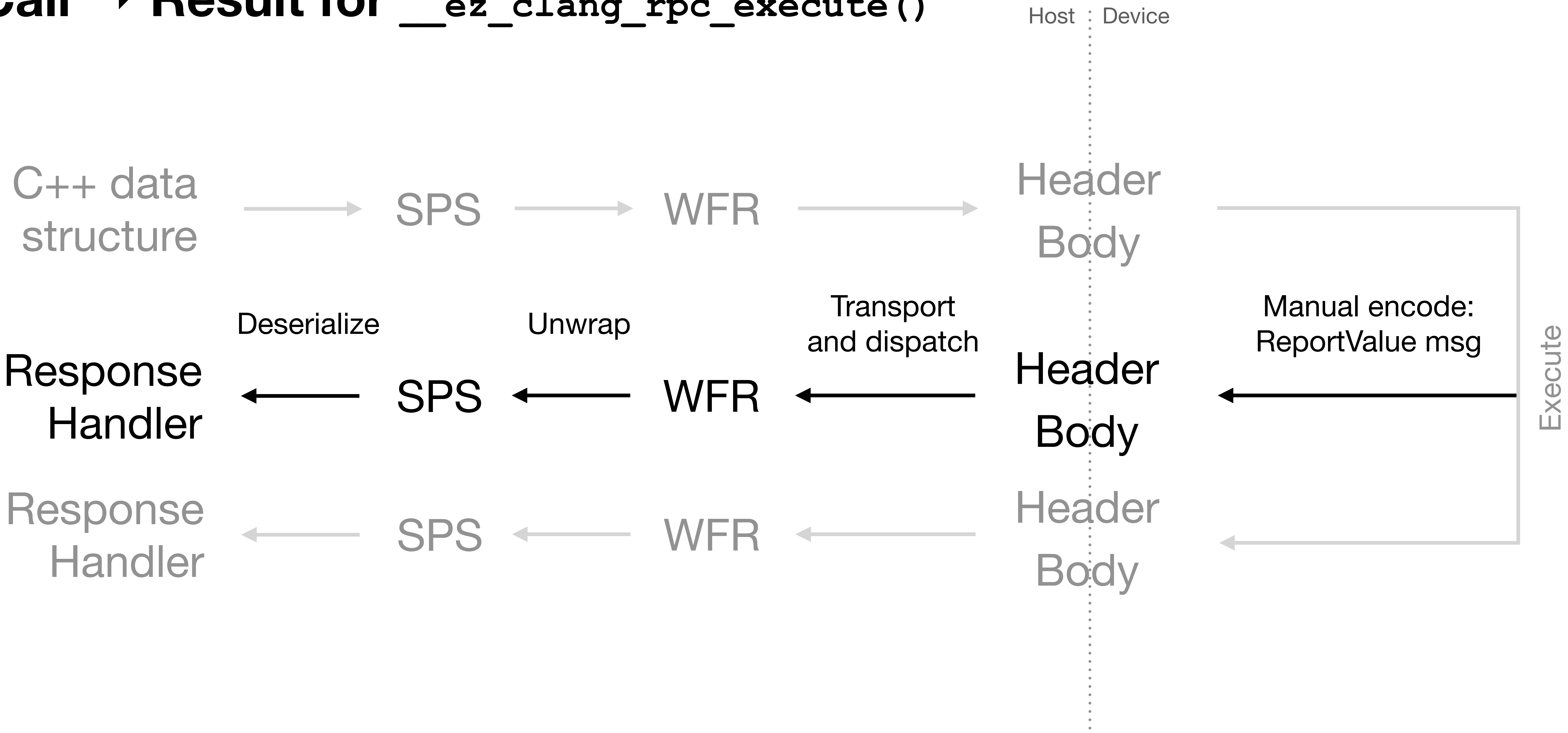
RPC Pipeline

Call → Result



RPC Pipeline

Call → Result for `__ez_clang_rpc_execute()`



Transform: Return value formatter

In-process implementation in Cling

1. `ValueExtractionSynthesizer` captures `clang::QualType` of return value
2. Synthesizes extra call to `setValueNoAlloc()` as last evaluation step and hardcodes it to pass on all relevant information, i.e.: `cling::Interpreter*, clang::QualType*, cling::Transaction*, cling::Value*`
3. `setValueNoAlloc()` is defined in Cling's `RuntimeUniverse` and delegates the request back to the Interpreter's `ValuePrinter` class

- Everything happens within the same call-stack → synchronous process
- Relies on pointers passed through JITed code layers → shared memory

Transform: Return value formatter

Out-of-process implementation in ez-clang

1. `ValueExtractionSynthesizer` captures `clang::QualType` of return value
2. Synthesizes extra call to built-in `__ez_clang_report_value()` runtime function as last evaluation step and hardcodes it to send back raw data
3. Installs an asynchronous response handler for the current RPC message ID, which stores the `clang::QualType` and constructs a `cling::Value` for the actual printing

- Asynchronous process → response handlers, memory management, timeouts
- Message-passing interface → fault isolation, no shared memory required

Challenges

- Tooling environment
- Memory constraints
- Serial connections
- Built-in symbol lookup
- Build on upstream libLLVMOrcJIT
- Debugging

Challenges

Tooling environment

Source: https://www.st.com/en/development-tools/stm32cubeide.html

Most hardware proprietary

- Software tooling as well (historically)
- Risk of vendor lock-in
- Open-source tooling slowly evolving, e.g.:
  Arduino (2005), ARM mbed (2009), RISC-V (2010), **PlatformIO (2014)**
- Proprietary tooling appears to remain dominant in industrial applications
- GCC appears to be the dominant OSS toolchain

Challenges

Memory constraints: minimize resource consumption

Much of libLLVMOrcJIT depends on libLLVMSupport (partially obsolete) and STL containers

➡ Reverse engineer SPS encoding to implement manual serialization

Serialized RPC messages must fit in message buffer

➡ Reduce overall message sizes: 64bit → 32bit fields, shorten names of bootstrap symbols, etc.

Modified some RPC details, e.g. remove fields like memory-manager ID

Challenges

Serial connections

Using termios TTY on host for serial connection with UART interface on device

Experience so far: device specific and at times unreliable

- ➡ Magic number to mark start of serial stream
- ➡ Custom per-device plans for error recovery

Serial port handling varies between operating systems

- ➡ For now: made for Linux, partially macOS, no Windows

Interrupts and JTAG/SWD debugger can corrupt serial streams

Challenges

Built-in symbol lookup

No operating system → no dynamic linking

- No `--export-dynamic` support in linkers → no `.dynsym` in binaries
- Workaround for built-in symbol lookup right now:

  ➡ Relink step + linker script magic to retain static symbol table info

Clang C++ ABI not fully compatible with GCC, e.g.:

- `uint32_t` is `unsigned long int` in GCC and `unsigned int` in Clang

  ➡ Recommend Clang toolchain for firmware builds?

Challenges

Build on upstream libLLVMOrcJIT

Overall: lots of well-designed extension points

A few downstream changes still appear necessary — will propose patches upstream soon — e.g.:

- Add extra RPC message type in downstream `EPCOpcode` type, but virtual `SimpleRemoteEPCTransport::sendMessage()` hardcodes `SimpleRemoteEPCOpcode` enum
- Sanity check before allocating memory in `FDSimpleRemoteEPCTransport::parseHeaderBuffer()`
- RPC message header customization not possible: e.g. 32bit fields

Challenges

Debugging

- Debug static firmware code with GDB via JTAG/SWD and openocd
- Some devices have no JTAG/SWD connector (e.g. Teensy LC)
- LLVM ORC implements GDB JIT interface, but no debug-server on device:
  - JITed code: no debug info, no callstacks
  - Dump relocated object buffers to disk and side-load in openocd?

  ➡ IPC protocol for JIT ⇄ openocd/GDB server?

  ➡ What if JIT and debugger run on different (virtual) machines?

Feedback

What's missing for you to try out ez-clang?

- QEMU device? Candidate: lm3s811evb, Cortex-M0, 8KB ROM
- Host-side transport interface: Is TTY sufficient? Priority for TCP? others?
- Device-side:
  - Is UART sufficient? Are CAN, SPI, I2C relevant/compatible?
  - Priority for other archs? RISC-V, AVR?
- Windows support?
- Specific Cling, C++ or meta features? Modules?

Outlook

Next few weeks

1. March: First binary distribution of current development state
2. March: First reference implementation for firmware
3. April: ABI documentation
4. April: Second binary distribution allows to configure custom devices
5. 2nd week of May: EuroLLVM presentation? (If it happens)
6. Some big features — when time permits

Thanks for attending

Can’t wait to hear your questions!

Slides

https://compiler-research.org/meetings/#caas_10Mar2022

Updates

https://github.com/echtzeit-dev/ez-clang

https://echtzeit.dev/ez-clang

https://twitter.com/weliveindetail