Composable Custom Extensions and Custom Function Units for RISC-V

Jan Gray (Gray Research) , Tim Vogt (Lattice Semiconductor), Tim Callahan (Google), Charles Papon (SpinalHDL),
Guy Lemieux (University of British Columbia), Maciej Kurc (Antmicro), Karol Gugala (Antmicro) *draft #2, 2022-04-26*

RISC-V extensions allow diverse uses. But fragmentation!

- Standard extensions layer and compose, by design
  - But consume finite encoding space, take years to ratify
- Custom extensions enable rapid in-house accelerator+library solutions
  - FPGA RISC-V SoCs: *insight* → *accelerated SoC* in an hour
  - But custom extensions may not work *together* in a system, due to conflicting encodings, CPU pipelines, logic interfaces, models of discovery, computation, state, error handling, versioning, tooling
  - Siloed solutions hurt HW & SW reuse and fragment the ecosystem

A mix-and-match custom extensions ecosystem

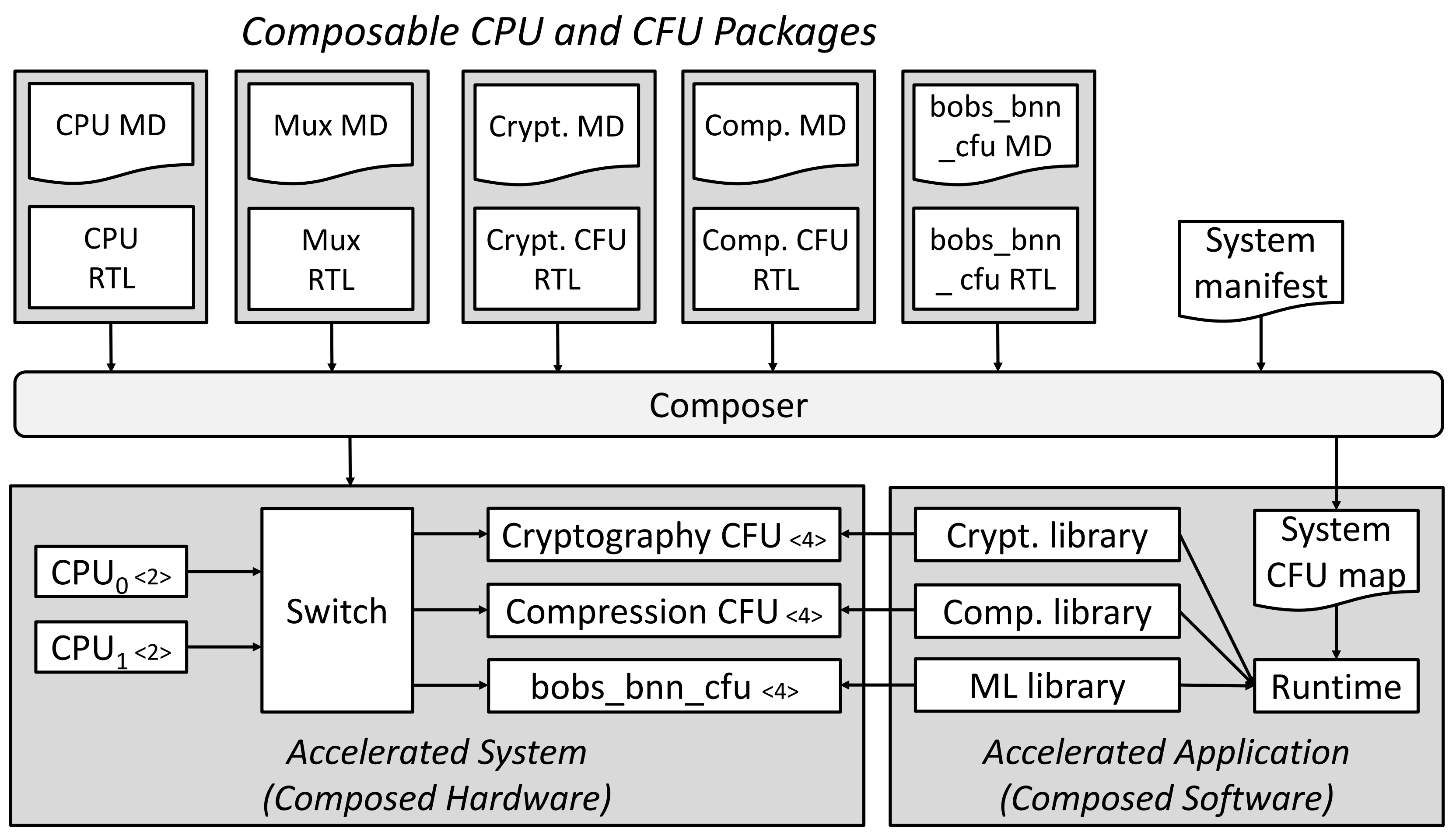
- *Agility* of custom extensions with *composability* of standard extensions
- Proposed hardware and software interop interfaces enable a new ecosystem of reusable accelerators & libraries that *just work* together

Custom interface, custom function unit, accelerated library

- *Custom interface (CI):* a composable custom extension: a named, immutable set of composition-safe custom instructions
- *Custom function unit (CFU)*: a core that implements a custom interface
- *Accelerated library:* issues custom instructions to a custom interface
- Different CFUs may implement an interface; different libs may use one
- Anybody can create a custom interface, CFU, or library, a CPU that composes CFUs, and tools for same, such that all works with all, always

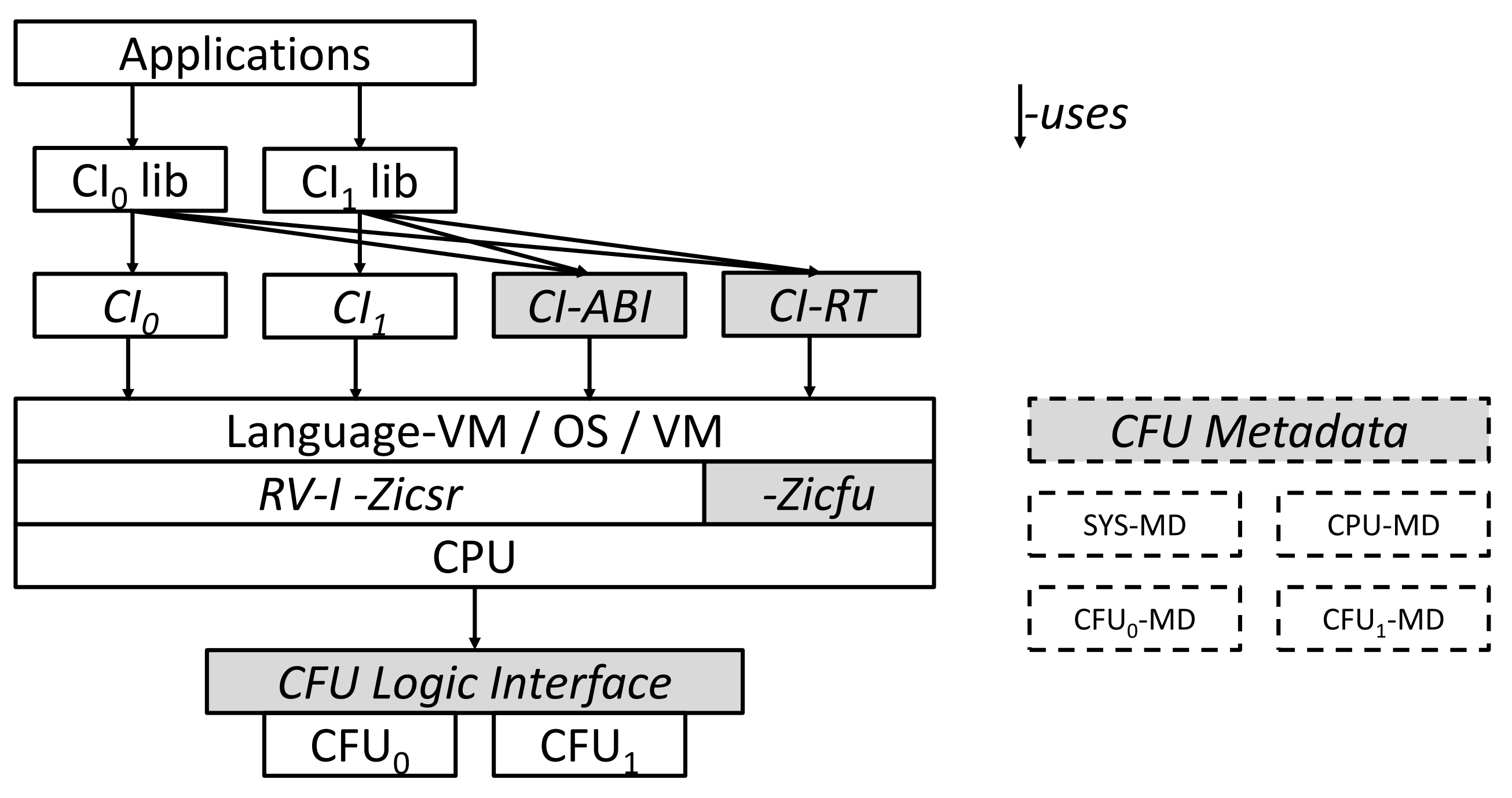
Example

- Alice's MPSoC uses a custom cryptography library and its CFU
- The compression lib can use custom instructions too. Alice adds its CFU
- Bob adds his new binary neural net custom interface, ML lib, and CFU
- Using Runtime, each lib *selects* its interface, then issues its instructions



HW-SW stack incorporating proposed interop interfaces

- Custom Interface ABI, Custom Interface Runtime
- "-Zicfu": CSRs: **`mcfu_select`**, **`cfu_status`**; instructions: **`custom-0/1/2`**
- CFU Logic Interface: for automatic CPUs + CFUs composition



Composition challenges → our proposed solutions

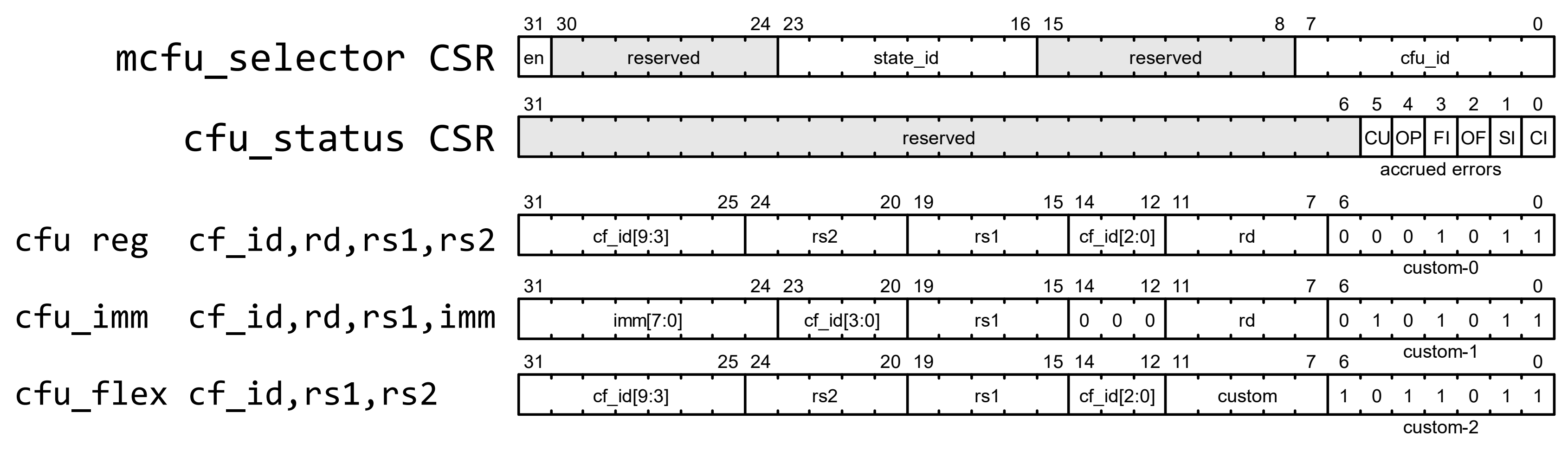
- Naming? Everything scoped to unique 128b-GUID custom interface IDs
- Opcode space? Inexhaustible with custom interface multiplexing
- Correctness under composition? Strict isolation of interfaces' states
- Error handling? **`mcfu_status`** CSR accrues errors
- Discovery? The Runtime consults a generated System CFU Map
- Context save/restore? Use standard *IStateContext* custom instructions
- Versioning? Immutable interfaces + newer-interface discovery
- Access control? Optional privileged access control of CFUs & state
- *Please see spec for all the details*

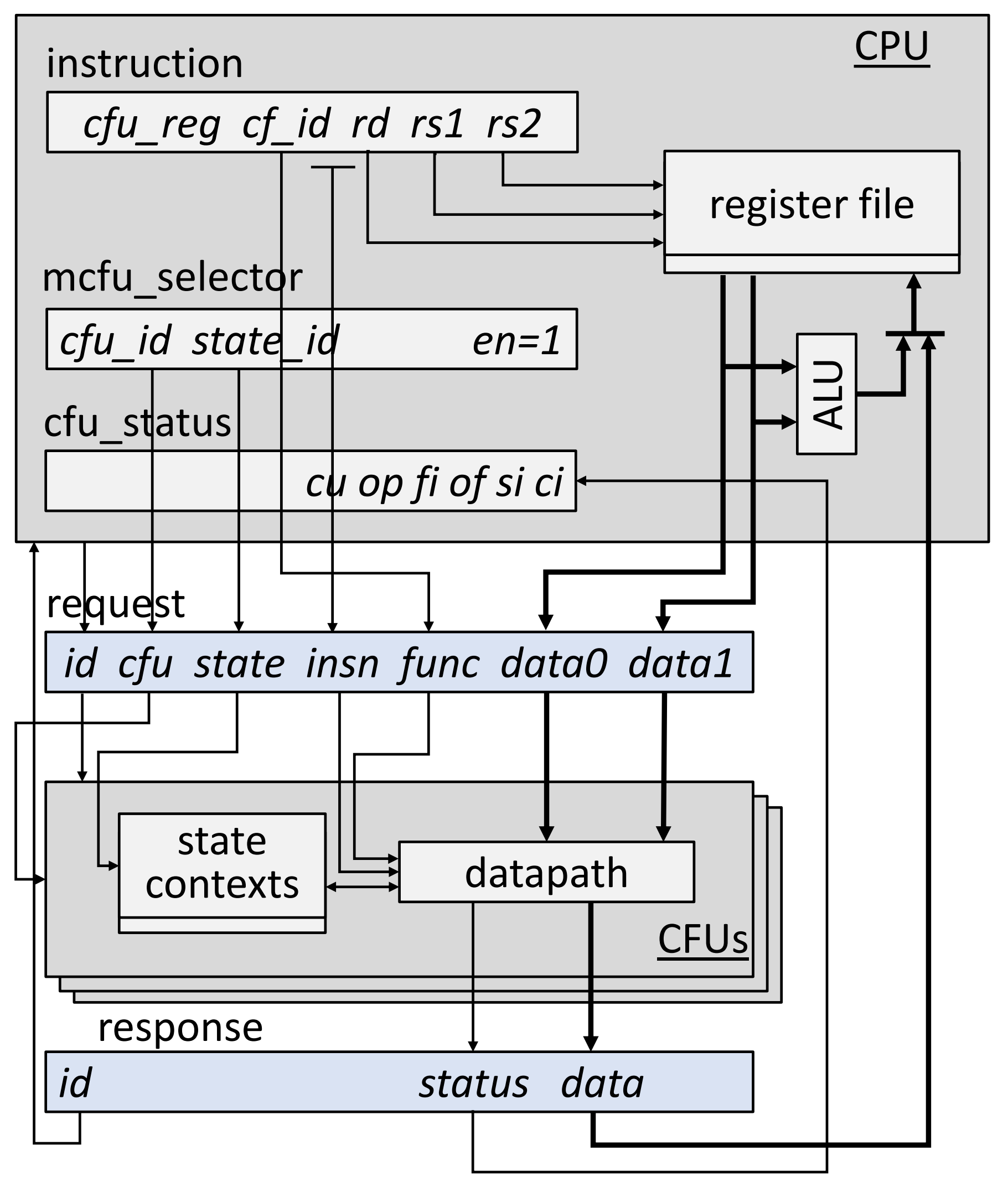
Correct composition via strict isolation of interface state

- Behavior of interfaces **must not** change when composed with others
- Custom instructions only access register file & the selected state context
- A CFU may be configured with 0, 1, #harts, or *n* isolated state contexts
- *V1:* no access to other shared resources (memory, CSRs, PC, exceptions)

HW-SW interface: custom interface multiplexing

- Inexhaustible, collision-free opcode space & no central opcode authority
- **`mcfu_selector`** CSR selects hart's *current* CFU and its state context
- **`custom-0/-1/-2`** instructions send a request to the current CFU
- CFU executes request, may read & write its state context
- CFU response updates destination register and **`cfu_status`** CSR





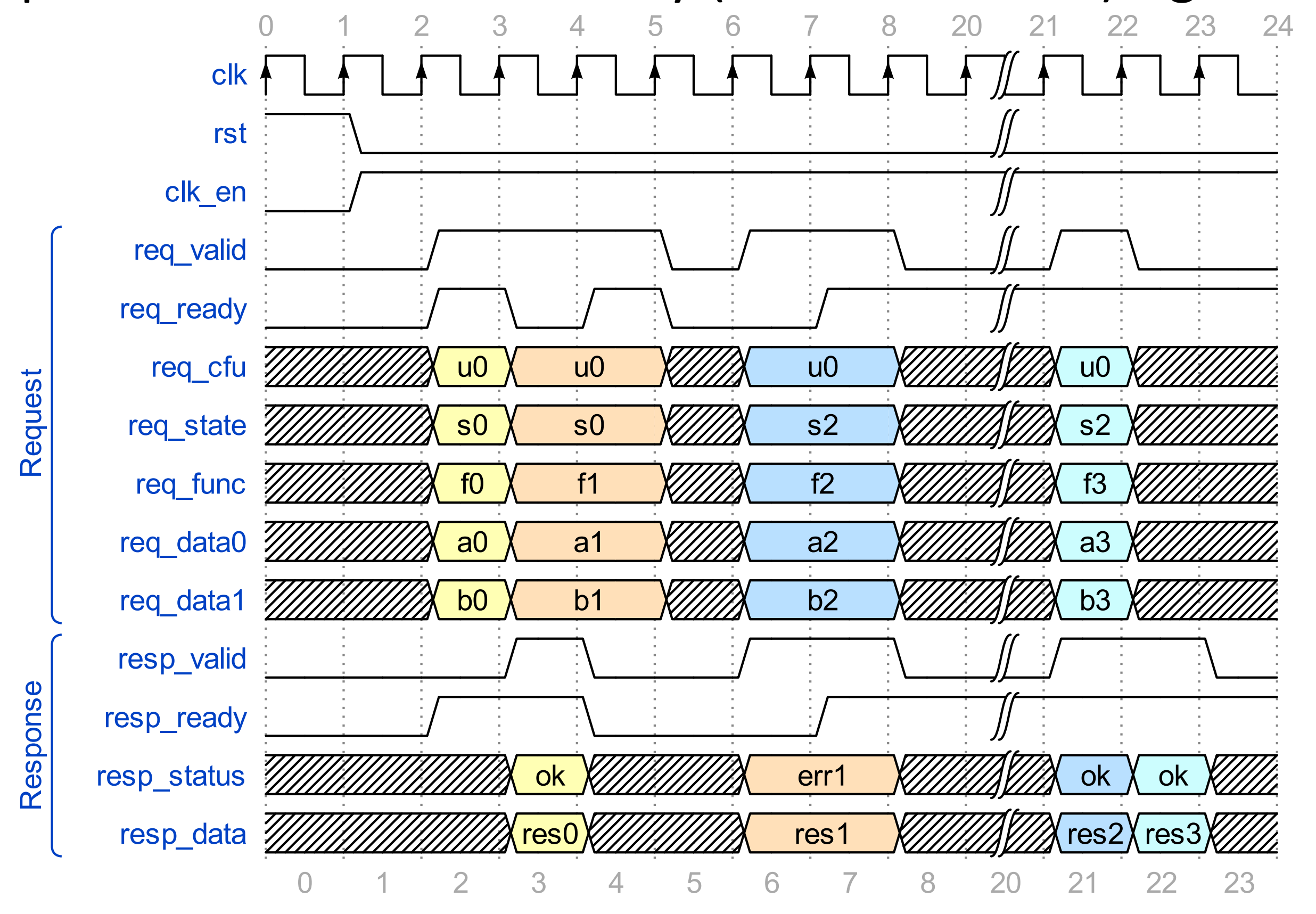
Exemplary accelerated library programming model

- Try to select a custom interface, issue custom instructions if CFU present

```
CI bm(CI_ID_IBitmanip);            // csrrw mcfu_selector …
if (bm)                            // accelerator present:
  count = cf(pcnt_cf, data, 0);    //   cfu_reg pcnt_cf,rd,data,x0
else                               // accelerator not present:
  count = popcount(data);          //   software version
```

HW-HW interface: CFU Logic Interface (CFU-LI)

- Interoperable CPUs-to-CFUs-to-CFUs signaling and metadata
- Flexible feature levels: combinational, fixed, variable latency, reordering
- Includes prebuilt switches & adapters for automatic composition
- Example: CFU-L2 variable latency (flow-controlled) signaling:



Status, more information, help us shape these interfaces

- *Draft Proposed RISC-V Composable Custom Extensions Specification,* https://github.com/grayresearch/CFU
- Status: refining spec, building consensus, writing RTL to demo mix-and-match composition across various CPUs and multiple CFUs
- Discussions and meetings: see RISC-V [sig-soft-cpu] list

Let us add just enough architecture that our custom extensions just work together