

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 #4, 2022-04-27a**



RISC-V custom extensions’ interop problem

- Standard extensions layer and compose well. Each takes years to ratify
- Custom extensions allow rapid in-house accelerator & library solutions
 - Solutions may not work *together* – conflicting encodings, different means of discovery, computation, state, error handling, versioning
 - Incompatible solution silos limit reuse and fragment the ecosystem

Let us build a mix-and-match custom extensions future

- *Agility* of custom extensions with *composability* of standard extensions
- Proposed HW-SW and HW-HW interop interfaces enable reusable accelerators that *just work together* – a *marketplace* of accelerators

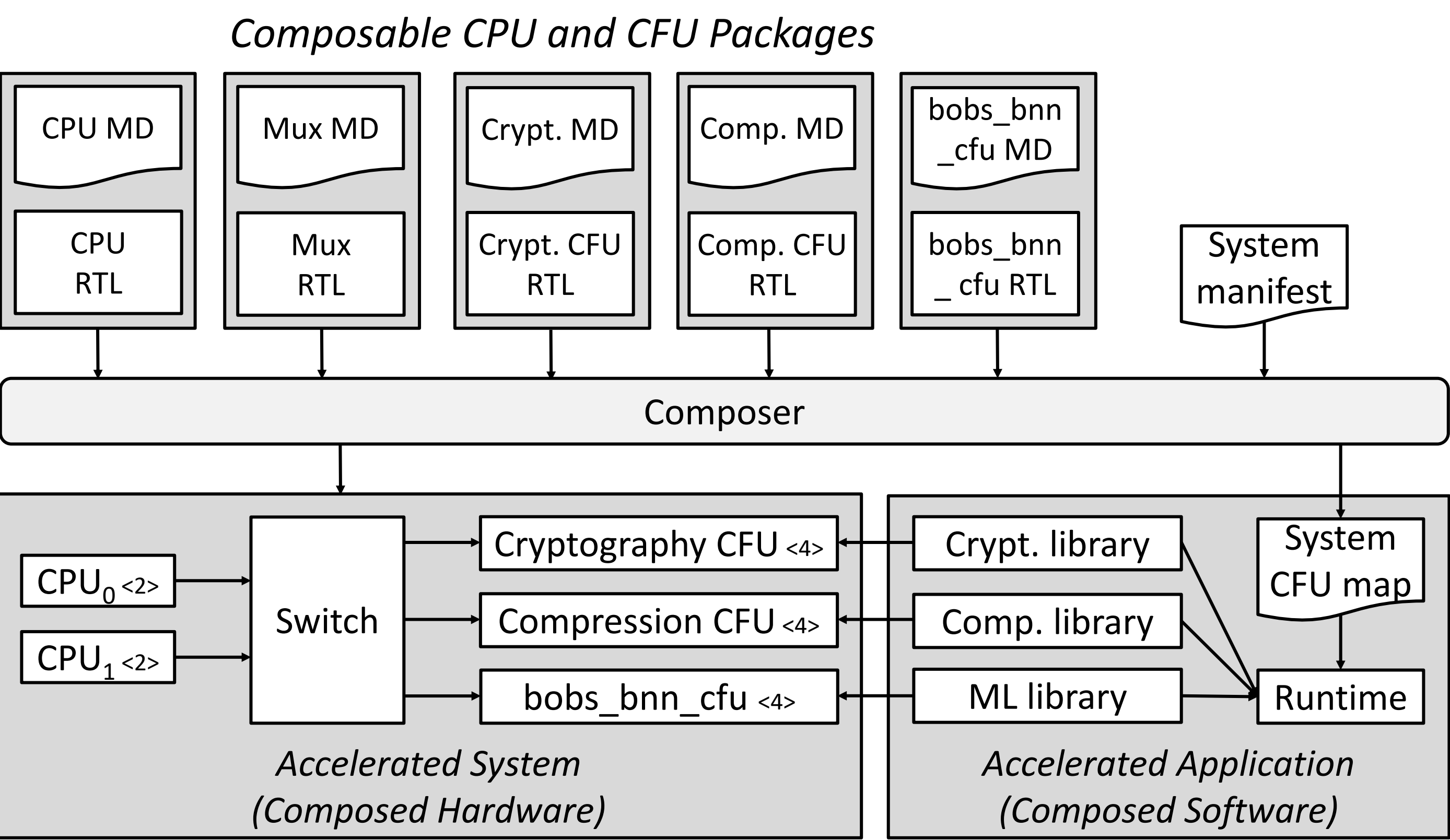
Key ideas

- **Custom interface (CI):** abstracts a *composable* custom extension
- **Custom function unit (CFU):** core that implements a custom interface
- **Accelerated library:** issues custom instructions of a custom interface

New interop interfaces

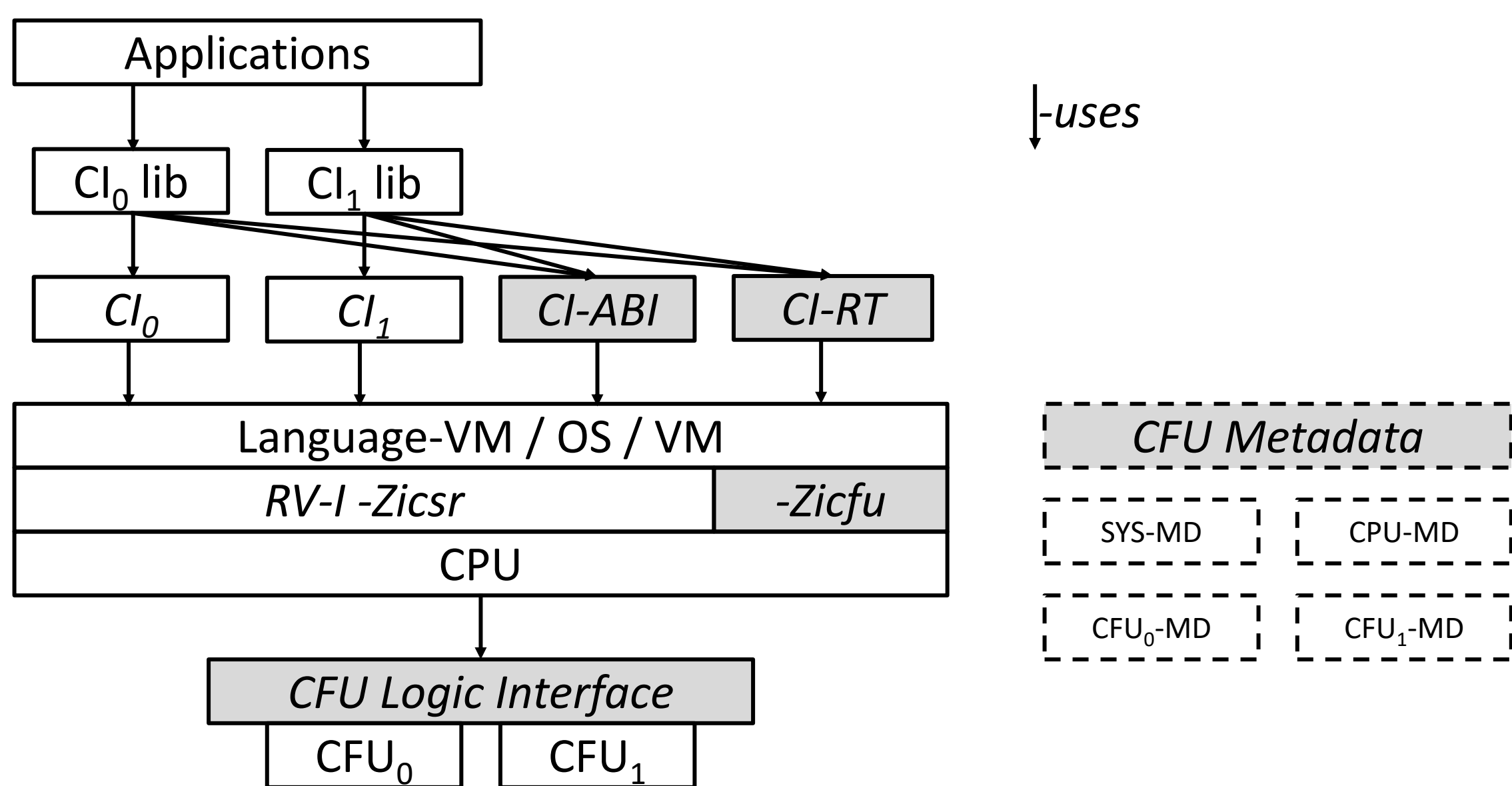
- **HW-SW: interface multiplexing:** libs select hart’s *current* CI & CI-state
 - each custom interface enjoys full custom instruction encoding space
- **HW-HW: CFU Logic Interface (CFU-LI):** CFU signaling standard
 - automatic composition of CPU+CFU complexes

Example



HW-SW stack changes

- Custom Interface Runtime: accelerated library services (discovery ...)
- “-Zicfu”: interface multiplexing CSRs: **mcfs_select**, **cfu_status**
- CFU Logic Interface & metadata: automatic CPUs+CFUs composition



Some composition challenges we address

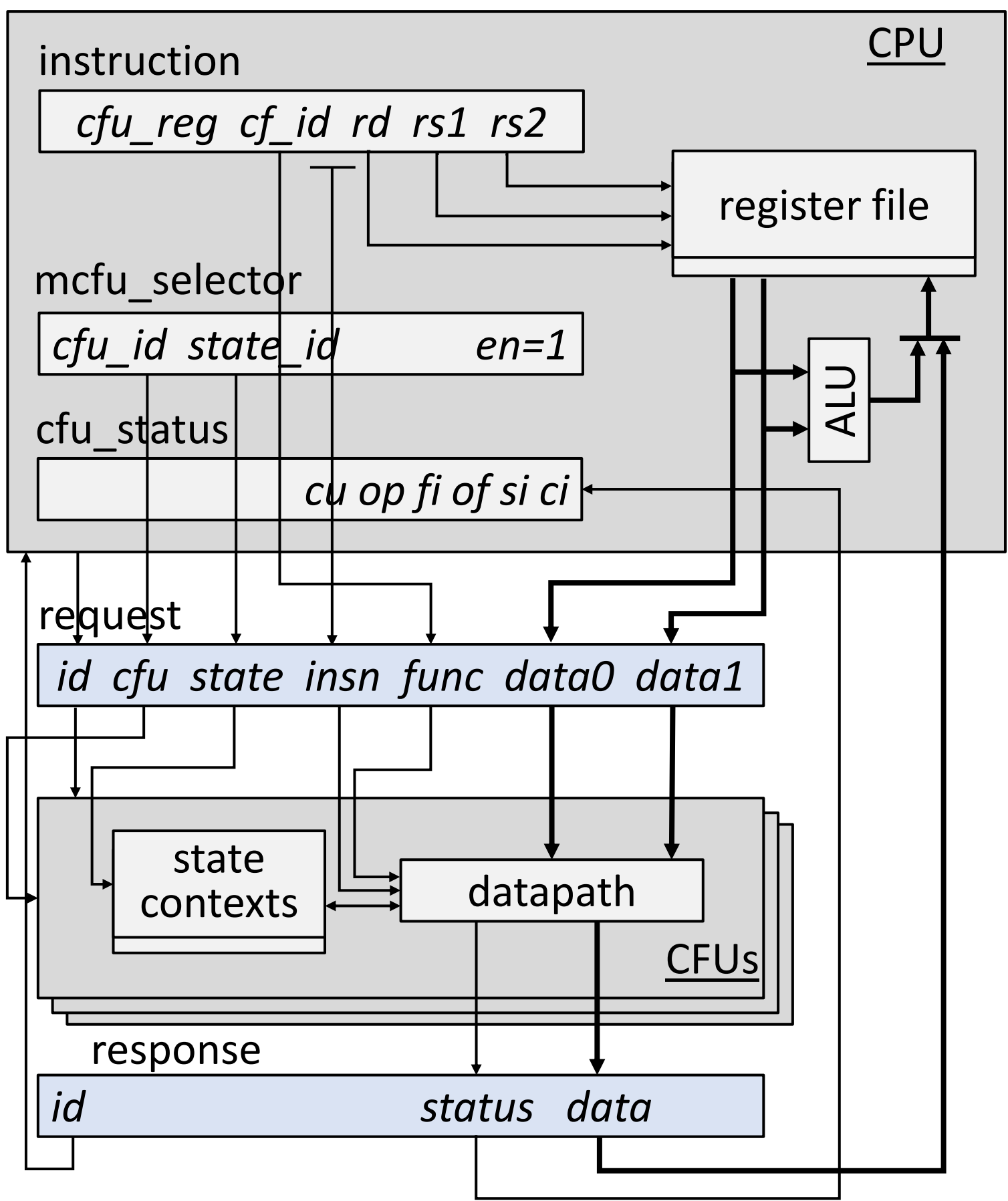
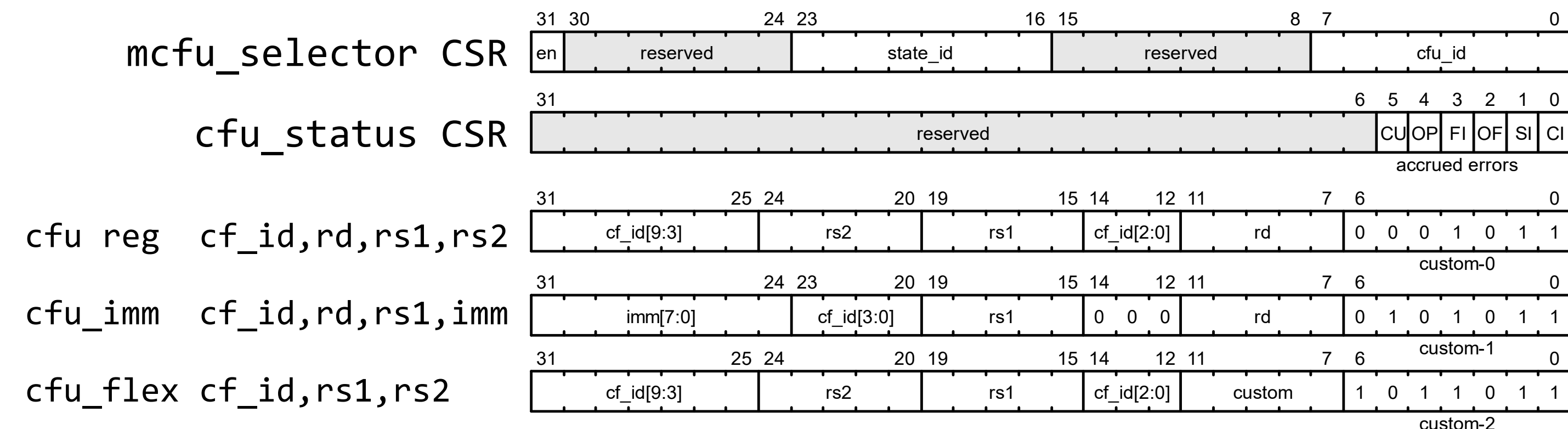
- Dynamic discovery of custom extensions
- Namespace / ID management with no central authority
- Collision free custom instruction encodings
- Correct composition of stateful custom extensions
- Uniform error handling
- Uniform context save/restore
- Versioning of custom extensions
- Privileged systems: access control to extensions and state
- *Please see spec for all the details*

Correct composition via isolation

- Behavior of an extension **must not** change when composed with others
- Custom instructions only access registers & selected CFU’s state context
 - Each interface/CFU may have 0, 1, #harts, or *n* isolated state contexts

HW-SW interface: custom interface multiplexing

- Inexhaustible, collision-free instruction encodings
- **mcfs_selector** CSR selects hart’s *current* CFU and some state context
- **custom-0/-1/-2** functions routed to the selected CFU
- CFU performs function, may update its state context
- CFU response updates destination register and **cfu_status** CSR

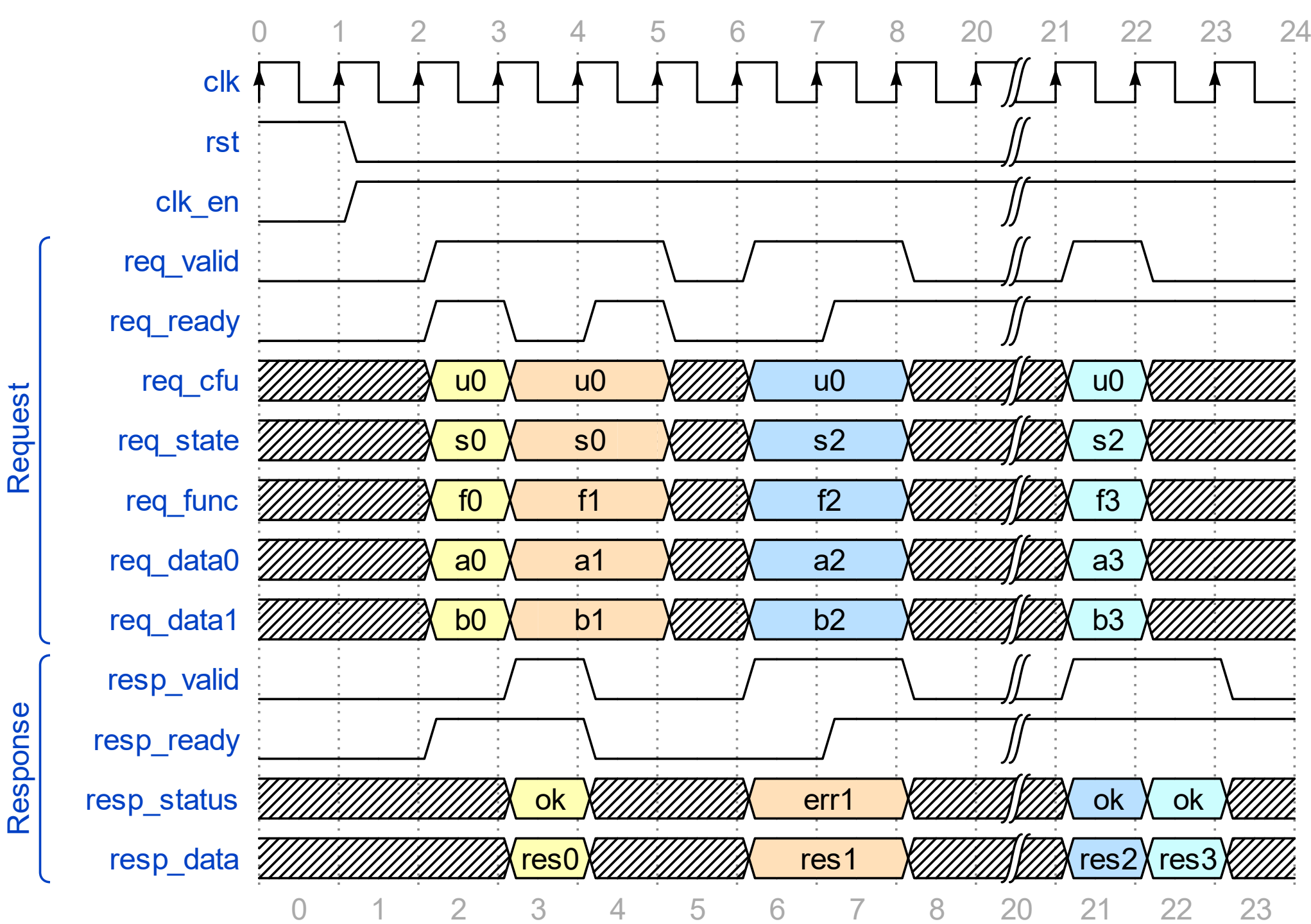


Example accelerated library programming model

- Try to select a custom interface, issue custom instructions if CFU present
- ```
CI bitmanip(CI_ID_IBitmanip); // csrrw mcfs_selector ...
if (bitmanip) // 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)

- Flexible feature levels: combinational, fixed, variable latency, reordering
- Prebuilt switches & adapters for glueless composition
- Example: CFU-L2 variable latency signaling:



## Learn more

- **Draft Proposed RISC-V Composable Custom Extensions Specification**, <https://github.com/grayresearch/CFU>

- Status: refining spec, building CPUs+CFUs composition demos
- Join us! Discuss, meet on RISC-V [sig-soft-cpu] list



**Let’s define a common custom extensions architecture  
so our custom extensions “just work” together**