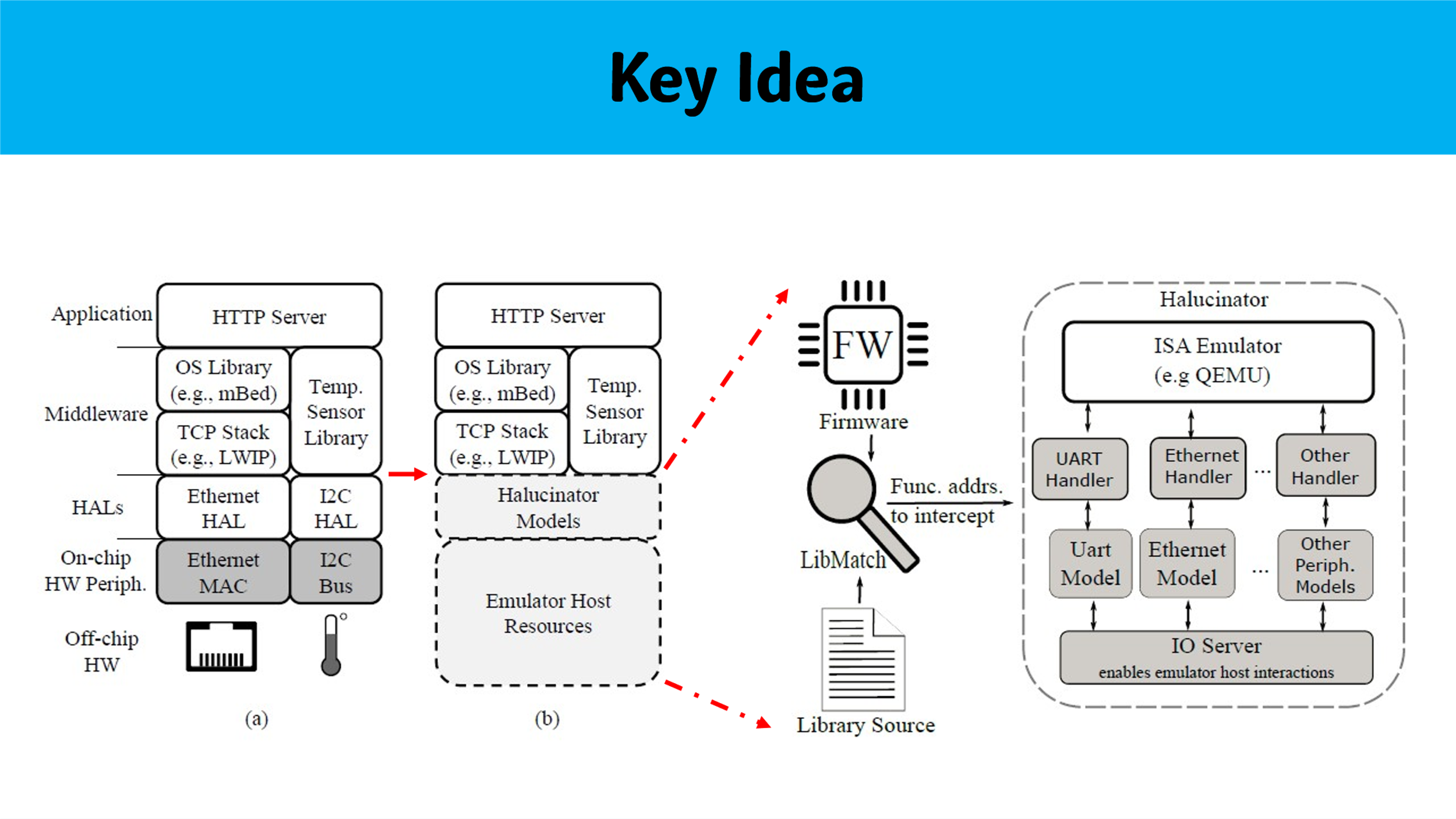
**HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation**

1. **Motivation**

* **For What?**
  + **Enable scalable firmware testing without requiring specialized hardware**
* **Why need to?**
  + Embedded systems are pervasive in modern life
  + developers create and test firmware almost entirely on physical testbeds
  + embedded hardware provides limited introspection capabilities, restricting the ability to perform dynamic analysis on firmware
  + firmware re-hosting, provides a means of addressing many of these challenges
* **How difficult?**
  + heterogeneity in embedded hardware poses a significant barrier to the useful emulation of firmware.
  + most embedded systems have other components on their circuit boards(off-chip)
  + Current solutions is not enough

1. **Idea**



* + To mitigate some of the challenges of developing firmware, chip vendors and various third parties provide Hardware Abstraction Layers (HALs)
  + Enable scalable emulation of embedded systems through the use of high-level abstraction layers and reusable replacement functionality, known as High-Level Emulation (HLE)

1. **Implementation**

* **Components(Design):**
  + LibMatch: based angr,with some extensions
  + HALucinator: in python,based on QEMU,avatar2 for control
  + \*Fuzz: AFL-Unicorn combines the ISA emulation features of QEMU with a flexible API, and provides the coverage instrumentation and fork-server capabilities used by AFL.
* **Prerequisites**
  + obtain the complete firmware for the device
  + obtain the libraries, such as HALs, OS library, middleware, or networking stacks they want to emulate, and the toolchain typically used by that chip vendor to compile them
* **LibMatch**
  + Statistical comparison: **number of basic blocks, control flow graph (CFG) edges, and function calls**
  + Basic Block Comparison: compare the content of their basic blocks, in terms of an intermediate representation
  + Contextual Matching-->to resolve collisions
    - Caller context :
    - Callee context
    - Both of these processes occur recursively, as resolving conflicts in one function may lead to additional matches
* **HALucinator（Manual Effort）**

1. **Handles**
   * Run binary in HALucinator
   * Stuck or misbehavior, which functions contain the I/O operations
   * The process repeats, and successive handlers produce greater coverage and more accurate functionality
2. **Peripheral models**
   * Reflect aspects common to a peripheral type
   * Contain little actual logic
3. **I/O server**
   * centralizes external communication with the emulated system, by facilitating multiple use cases without changing the emulator’s configuration
   * The I/O server uses a publish/subscribe design pattern, to which peripheral models publish and/or subscribe to specific topics that they handle
   * centralizing all I/O enables a program to coordinate all external interactions of an emulated firmware
   * This enables powerful multiple interface instrumentation completely in software, and enables dynamic analysis to explore complex internal states of the firmware
4. **Peripheral Accesses Outside a HAL**
   * report all I/O outside handlers to the user.
   * all read operations to these areas will return zero, and all writes will be ignored
   * many MMIO operations can be safely ignored

* **Fuzz**
  + AFL-Unicorn combines the ISA emulation features of QEMU with a flexible API, and provides the coverage instrumentation and fork-server capabilities used by AFL
  + Provides a special fuzz peripheral model, will dispense data from the fuzzer’s input stream to the handler.