Open Source @ SIEMENS, 2025 Wuxi

Open source system for telemetry

Jiahong Luo <luojh@mail.ustc.edu.cn>

Undergraduate, Dept. of Chemical Physics, USTC

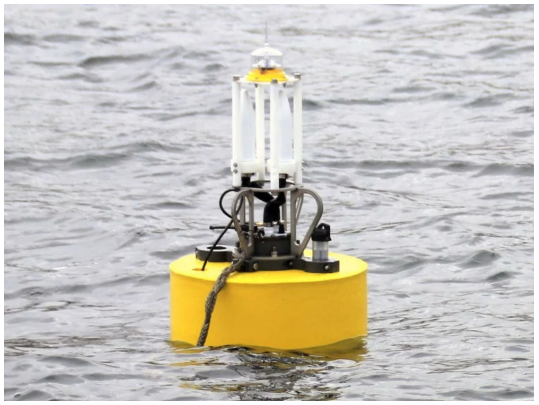
Introduction

Introduction

"**tele**/**metry**"

Remote Measure

"**tele**type": call it `tty`



Telemetry station for ocean observation

- **Why do we need telemetry? In labs?**
  - Monitoring the running status of infrastructure
  - Subtle change and abnormity could be identified early
  - Collect field data for further analysis and adjustment
  - Enabling the remote deployment of facilities

Introduction

Telemetry systems bridges computer system and the real world

Introduction

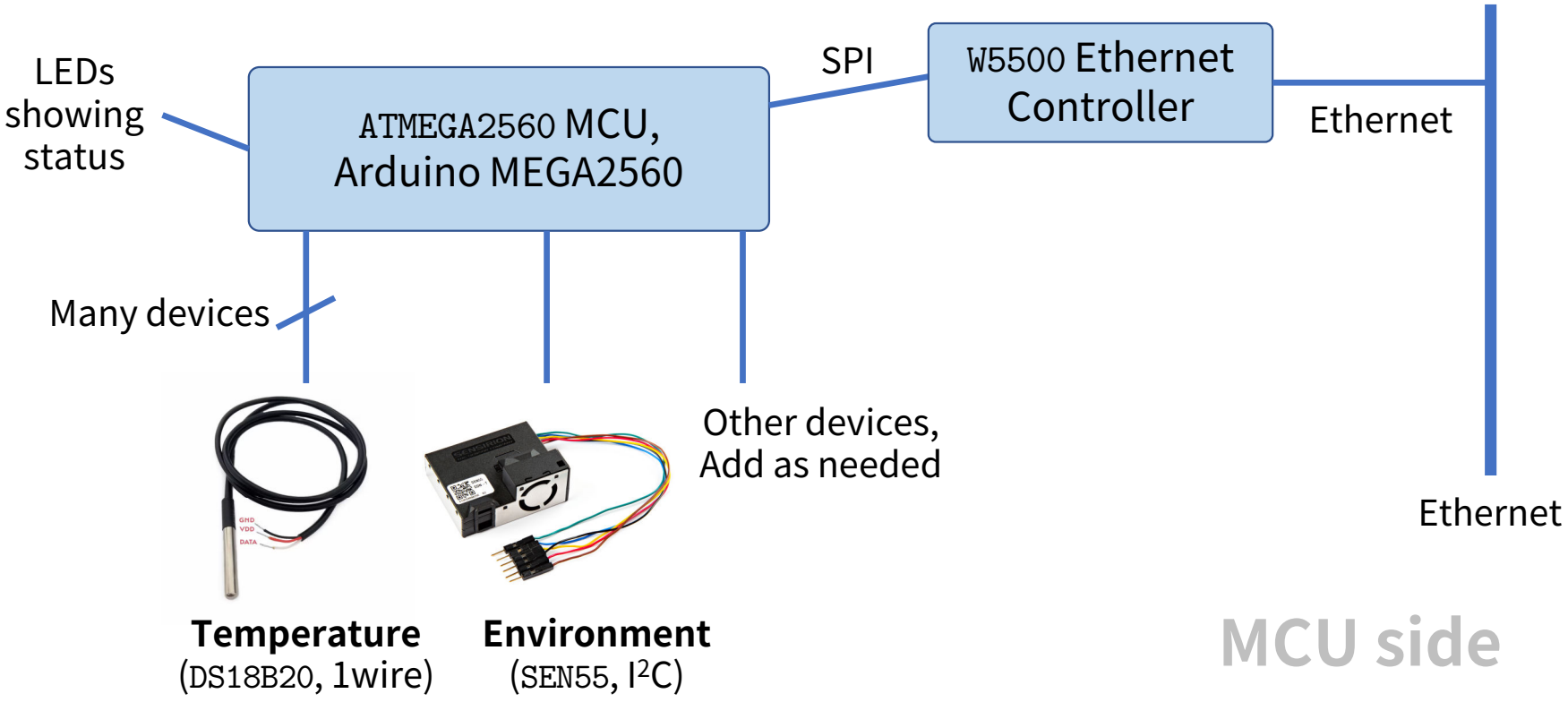
- A system used in my lab at USTC, for **monitoring the lab facilities**.
  - **Environment**: Temperature (multiple points), Humidity, Dust, ⋯
  - **Instrument**: Voltage/Current etc.
  - **Servers**: Power / HDD LEDs, Remote power ON or OFF
  - Run over lab internal network (Ethernet)
  - Prometheus + Grafana for storage, visualization and alerting

1st generation

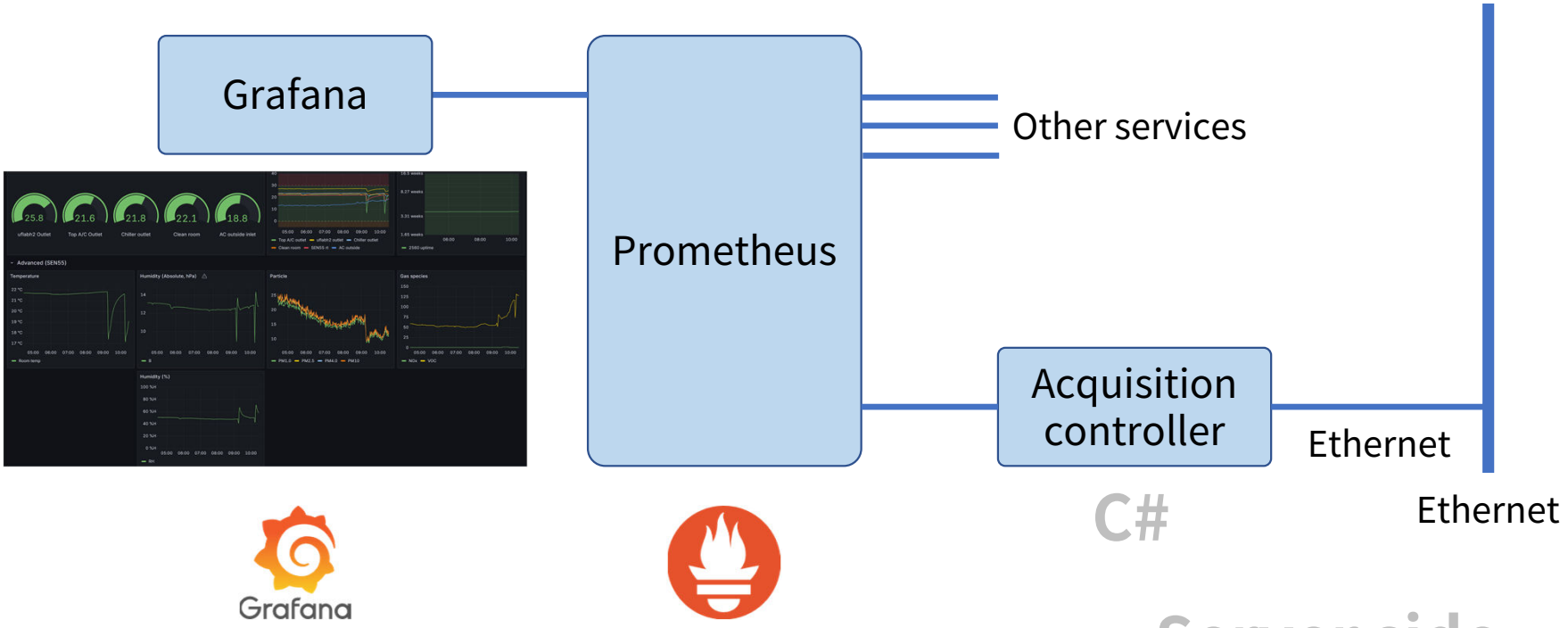
1st generation

- First built in September, 2024
- **Hardware**
  - AVR MCU running C++ code ≈ Arduino MEGA2560
  - Many I/O pins (60+) for connecting to the peripherals
  - W5500 (via SPI) for Ethernet access, self-bootstrap
- **Software**
  - Homebuilt embedded code + protocol adapter
  - Prometheus (TSDB) and Grafana (Visualization)

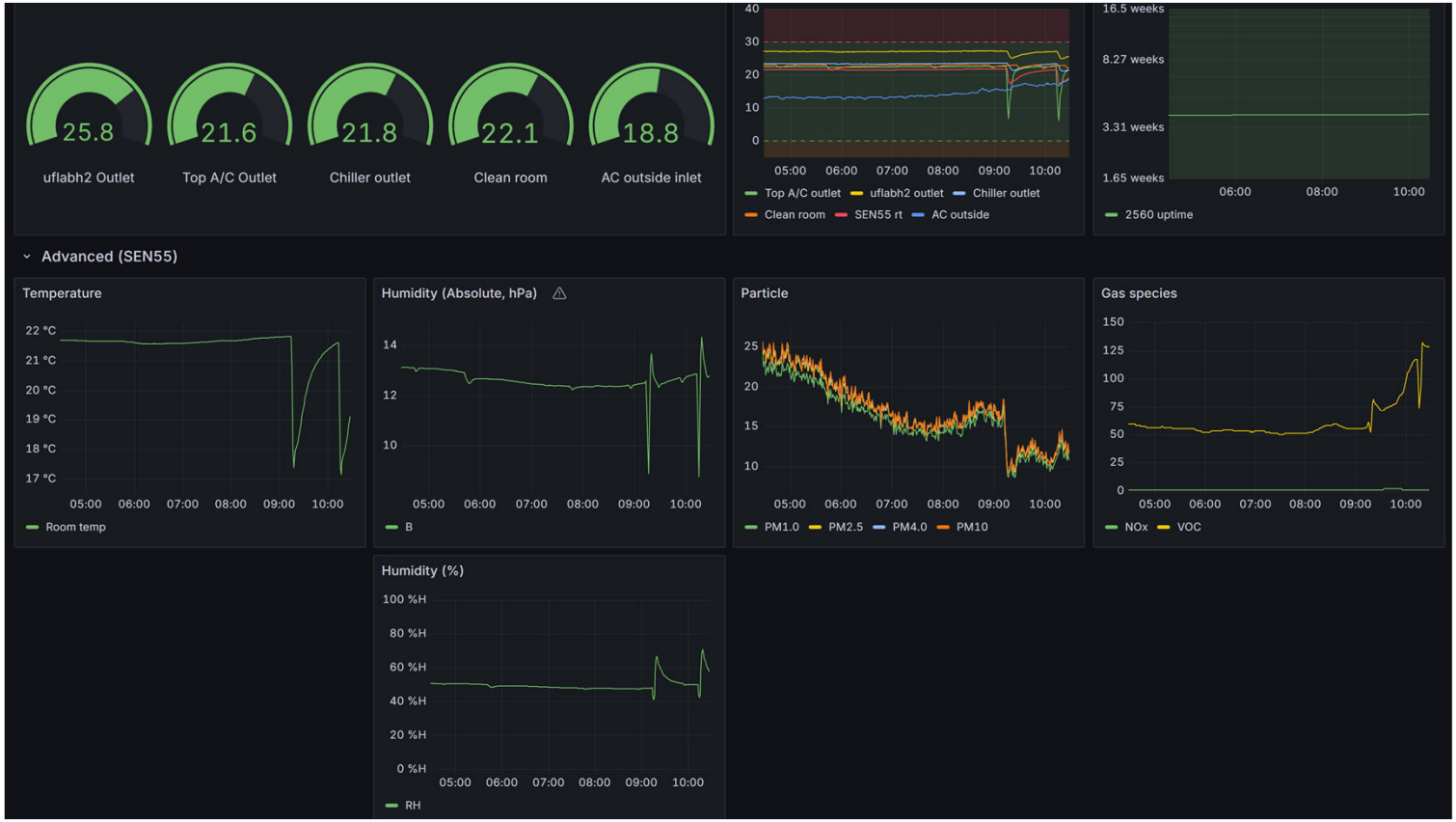
1st generation



1st generation

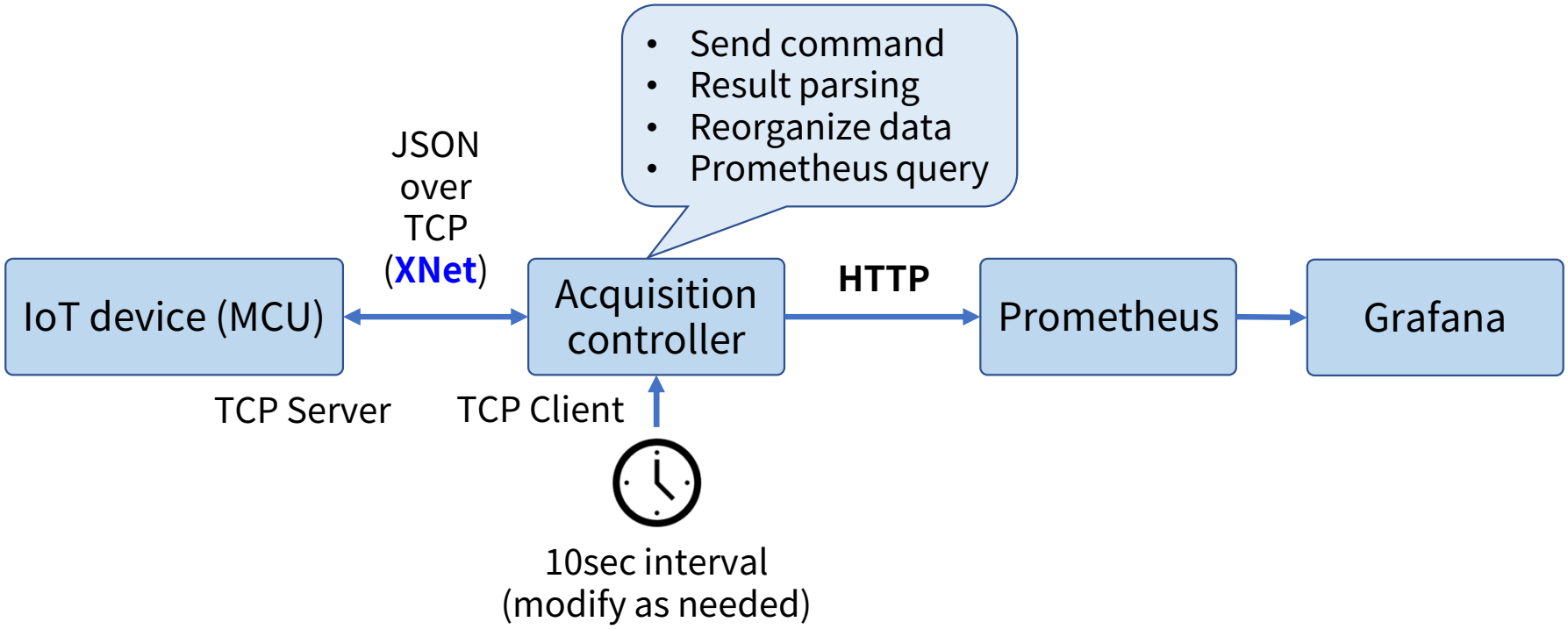


1st generation



1st generation

- Protocol



1st generation

- The "**XNet**": Simple JSON transfer protocol
  - Request and response are all JSONs, separated by an empty line
  - Works well on resource-limited processors
- Use **entirely clear text stream**
  - Typically XNet runs over **TCP/IP Ethernet**
  - But also available on anything that could transfer text:

  RS232/485, **Optical fiber**, **IR**, **WLAN**, **4G network**,…

1^{st} generation

- **Self-bootstrapping** configuration
  - **The MCU need the IP address and the TCP port for listening.**
  - (*optionally*) **Configuration for peripherals could be retrieved instead of being hard-coded in MCU.**

  How?

  Self-bootstrapping allows the device to **fetch configuration** over simple network protocols and be **configured at startup**

1^{st} generation

- **Self-bootstrapping** configuration: **How**?
- Each IoT device has its unique MAC address like `02:ab:cd:ef:01:23`

1. Try DHCP for an initial IP assignment + DNS server address
2. Request a DNS TXT record at `ef0123.iot.local`:

   ```
   { "ip": "12.34.56.78", "port": "12345", "config": "http://xxx" }
   ```

3. Set the IP to that specified in TXT record
4. (*optionally*) Load config from the specified URL
5. Listen to the specified port and then start working

1^{st} generation

- **Self-bootstrapping** configuration: **How**?
- The infrastructure required:
  - **DHCP server** (`dnsmasq`)
  - **DNS server** (`Bind9`)
  - (*optional*) **HTTP server** (`Apache2`)
  - **Acquisition controller** (homemade; C#)
  - **TSDB** (`Prometheus`) and **Visualization** (`Grafana`)

1st generation

- **Self-bootstrapping** configuration: **Why**?
- Instant **plug-and-play**
  - **Avoid complex configuration per device, reduce errors**
  - When a new device come in:
  1. Take note of its MAC address
  2. Configure the DNS server (bootstrap IP) and HTTP server (config)
  3. Power on the device anywhere

1st generation

- **Self-bootstrapping** configuration: **Why**?
- Updating the **config**
  - **Enabling remote update**
  1. Change the config file + DNS record
  2. Send command to reboot the device
  3. (Then it would automatically reload the new config at startup)

1st generation

- **Pros**
- Fully remote operation
  - No local operation needed – suitable for remote working
- All opensource tools and standard protocols
  - Easily migrate to anything that could transfer ASCII
  - Expand the system with confidence, as needed
  - Fully programmable: no technical lock-ins

1st generation

- **Pros**
- Additional features on physical layer possible:
  - PoE power: Just one cable could support operation
  - 10BASE-T1S and 10BASE-T1L: IEEE Std 802.3cg, cable power + one twisted pair Ethernet

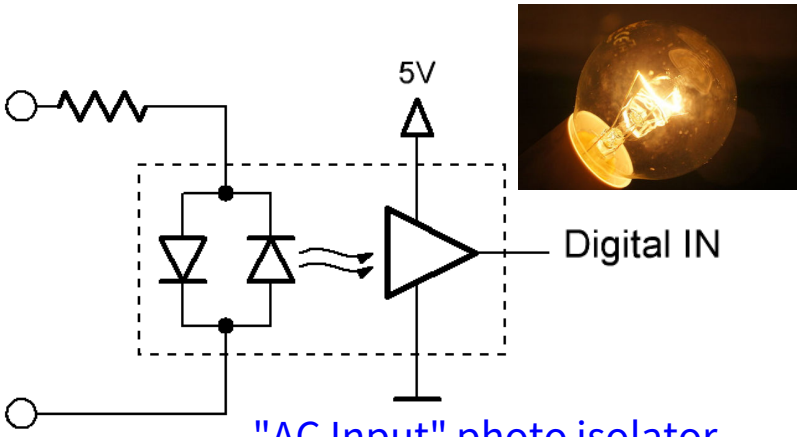
1st generation

- **Cons**
- Relatively complex structure
  - Endpoints → **Acquisition controller** → Prometheus ⋯
  - **Acquisition controller** is proved to be the most vulnerable part: an error in one channel (connection) could affect others
  - More layer means lower reliability
- Limited computing resource at endpoint
  - Only 8-bit MCUs, nearly no data processing

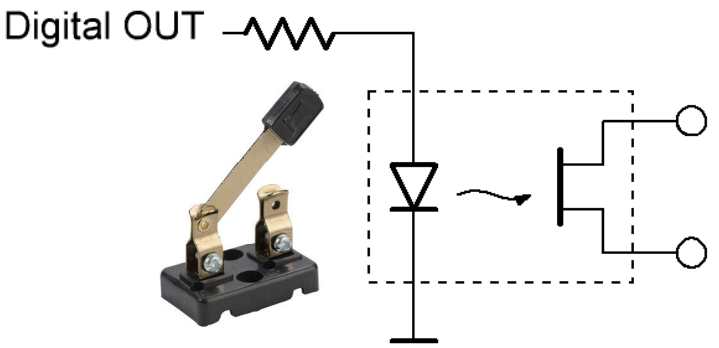
1st generation

- **More could be implemented**
- Using it as a **PLC**? Such devices *do exist*!

Plugins for the 1st gen controller:



"AC Input" photo isolator



"PhotoMOS" ≈ Solid State Relays

2nd generation

2nd generation

- 2nd generation is currently planned. Main differences include:
  - Using RISC-V 64 MPU (SG2000) instead of AVR 8-bit MCU
  - Network controller (with Ethernet & WLAN) integrated
  - Standard Linux installed, better tools
  - Direct Prometheus datasource (no conversion)

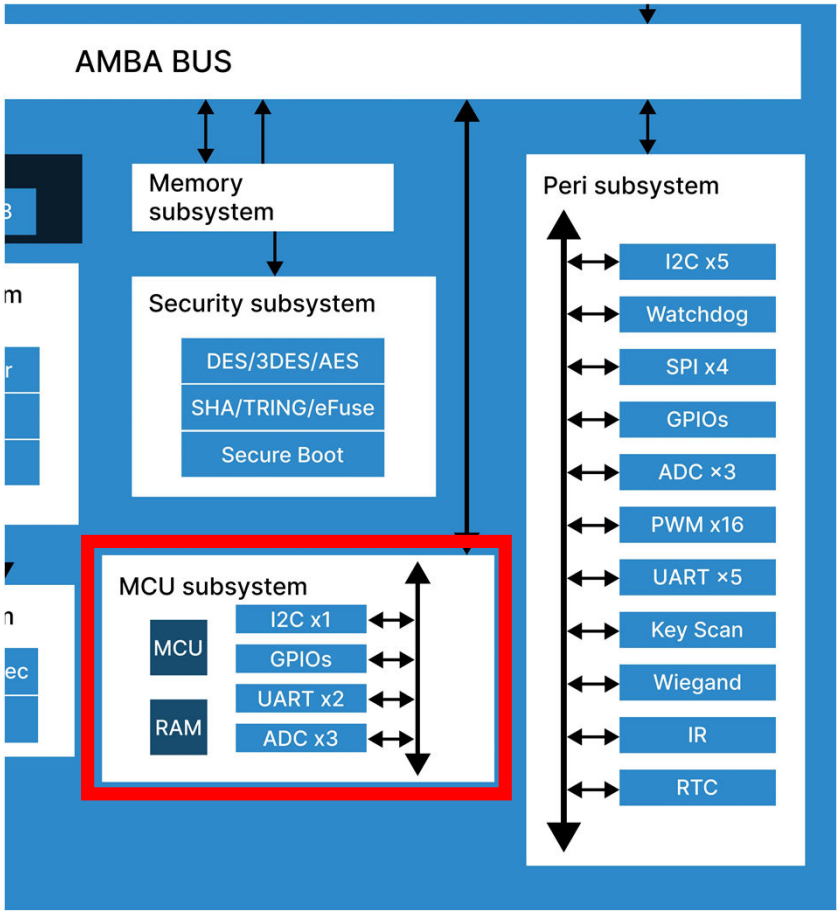
The RISC-V hardware →
Milk-V Duo S

2nd generation

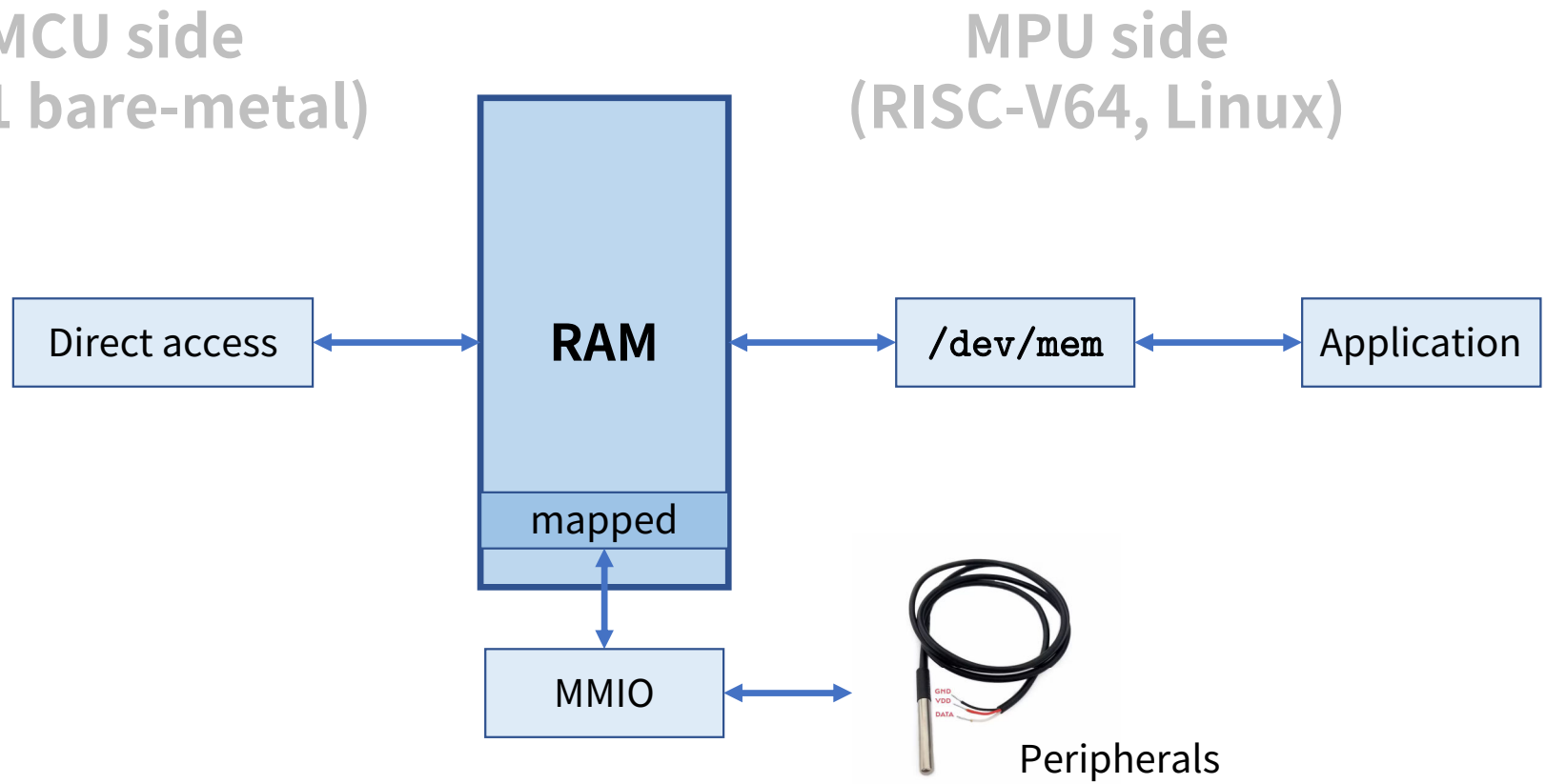
- Why MPU?
  - Much better processing ability – Local data processing possible
  - Directly accesses network – no W5500 controller needed, enabling HTTPS to improve security
  - File-based nonvolatile configuration, updated via HTTP/SSH…
- What is the *difference* on MPU?
  - Program runs on OS, not directly on processor core ("bare-metal")
  - IO via mapped file, not special registers / MMIO regions

2nd generation

- What about the IOs?
  - If directly using filesystem-mapped IO, the precise timing for some peripherals could not be satisfied.
  - **OS would add significant delay**
  - But SG2000 have a small **8051 kernel** embedded inside!
  - **truly MCU behavior**



2nd generation



2nd generation

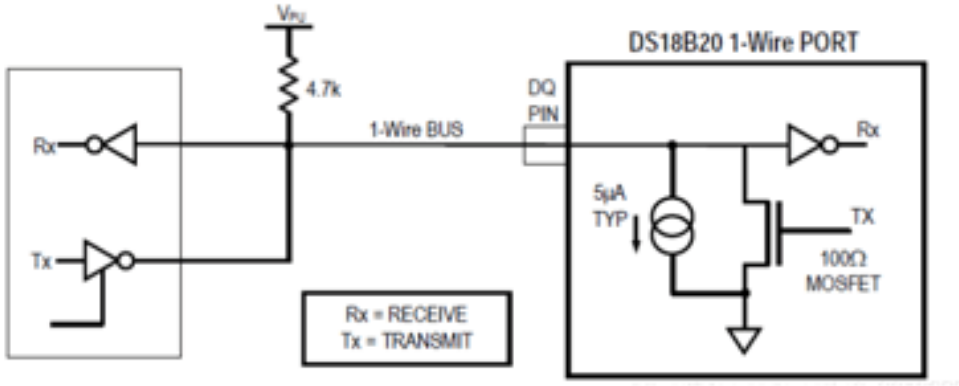
- Challenges:
  - Limited resource on 8051 & RISC-V architecture, cross compilers, interop.
  - 3.3V low voltage IO is incompatible with industry-standard sensors.
  - Higher power consumption and fragile construction, not so robust like simple MCUs.
  - Fewer IO on MPUs (20) compared to MCU (easily 50+).
    - Use shift registers (74HC595 like) to operate peripherals insensitive to timing

2nd generation

- Challenges:
  - Sensors with special IO interfaces like **DS18B20** needs special care:

"One wire" bus requires precise timing and truly bidirectional IO pins;
– A bit difficult for level shifters "emulated" 5V IOs.

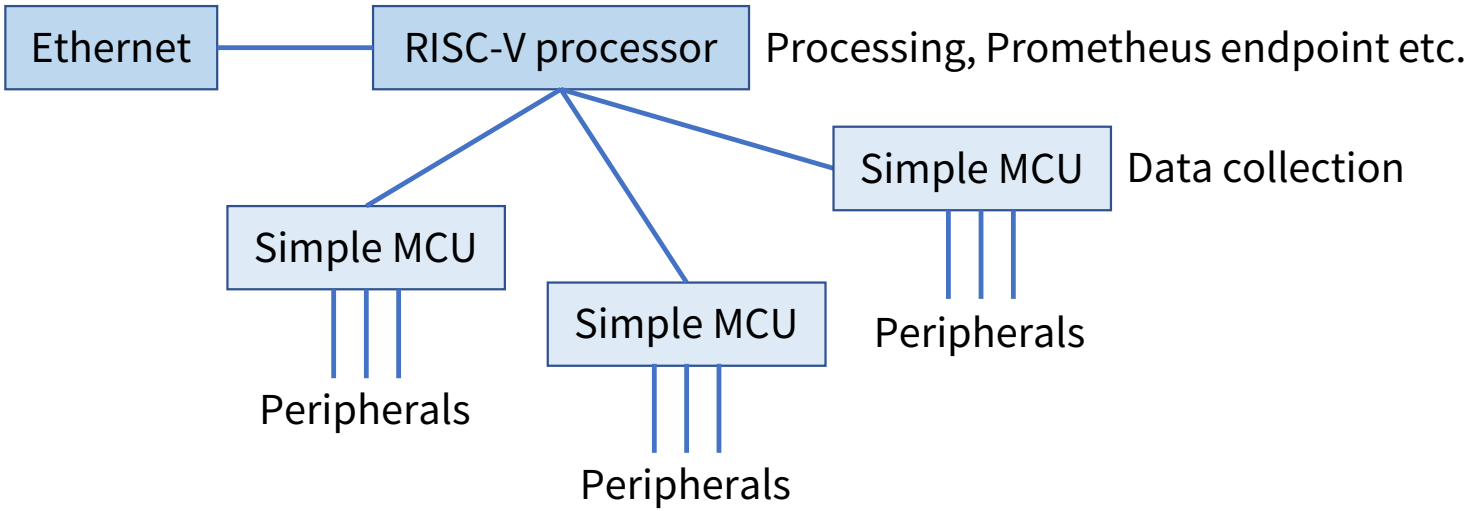
Bypass this:
Using SPI temperature sensors like **TC72**, **TMP125**, etc.



2nd generation

- Challenges:
  - Higher price compared to fully-MCU solutions

  but in fact RISC-V devices could work well with satellite MCUs:



Thanks