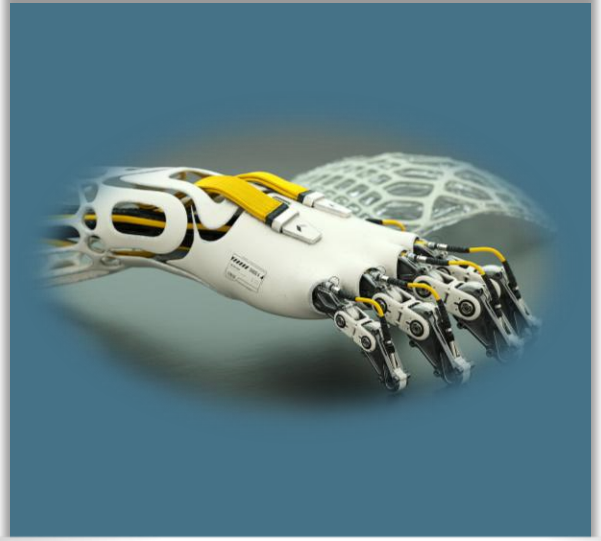
智能感知与物联网

授课人：王闻博

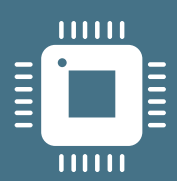
Email: wenbo_wang@kust.edu.cn

昆明理工大学 机电工程学院

2026年4月27日

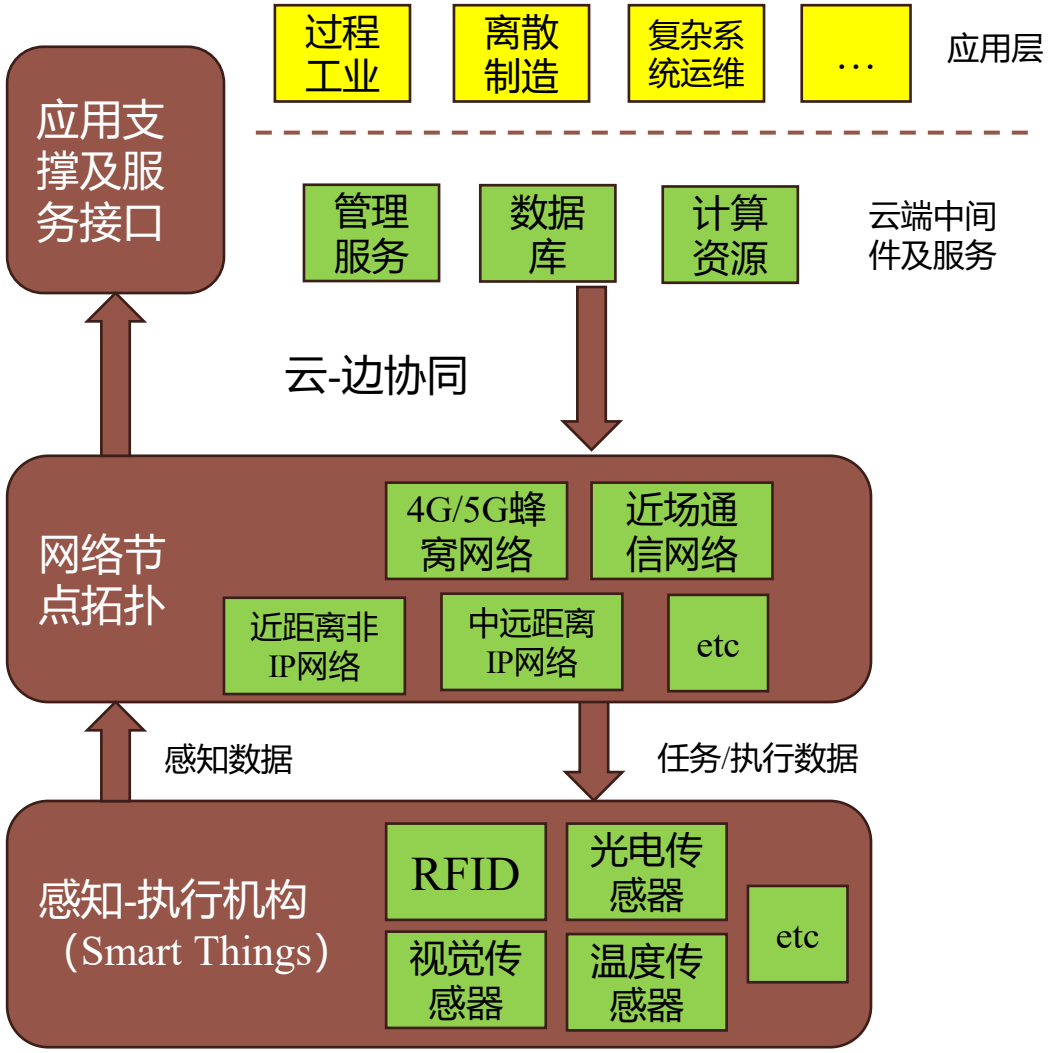


第四章
计算与决策的边缘化

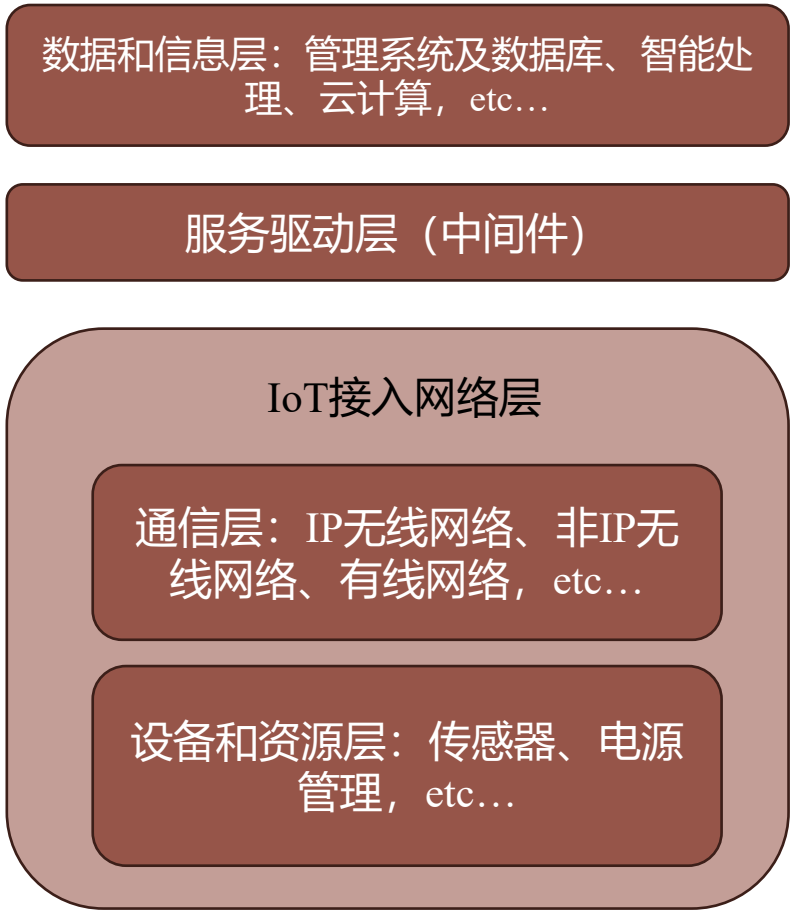



前章回顾：物联网的无线接入与组网技术

物理-软件实体



协议层抽象



前章回顾：物联网的无线接入与组网技术

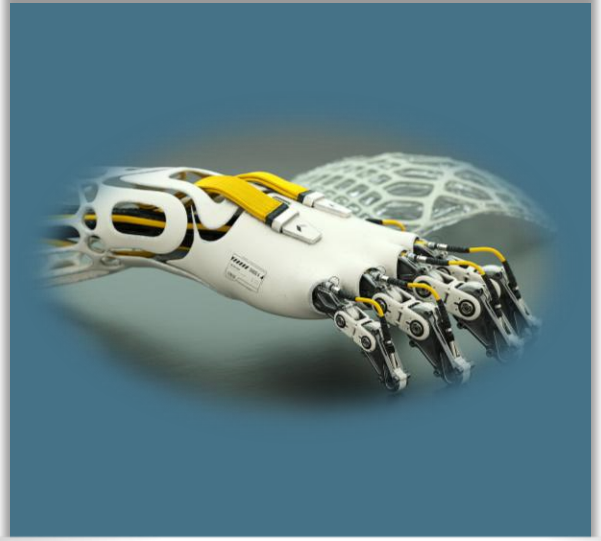
- **物联网组网的各层技术和协议**
  - **物理层：无线信号功率损耗模型 + 衰落 + 信道容量估计 + 物理层编码；**
  - **数据链路层：媒体访问协议（FDMA + CDMA + TDMA + ALOHA + CSMA）；**
  - **网络层与传输层：TCP/IP协议族+IP子网配置 + TCP报文自动重传控制 + Socket编程；**
  - **垂直协议族：802.15协议族（802.15.1/2/3蓝牙 + 802.15.4 Zigbee）；**
  - **蓝牙和Zigbee中的多跳组网方法：Flooding vs. AODV （Ad-hoc On-demand Distance Vector）。**

前章回顾：传感器与电源管理

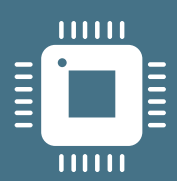
- 传感器的
  - 分类法则和发展趋势；
  - 误差（绝对误差和相对误差）+ 精度 + 灵敏度；
  - 网络化、智能化；
  - 多模态传感数据融合。
- 电源管理
  - 无线传感网络链路（设备对）的能耗模型。

未深入涉及但宜知晓的组网协议与技术

- **中远距离无线组网技术**
  - **基于IP的WiFi（802.11）协议：OFDMA多址技术 + MIMO技术；**
  - **低功耗广域网技术：LoRa（Long Range Radio）。**
- **基于蜂窝网络的物联网组网协议**
  - **超可靠性延时通信（uRLLC）+ 海量机器间通信（mMTC）技术；**
  - **窄带IoT通信技术（NB-IoT）。**
- **传感器与网络终端的数据连接**
  - **串口协议：UART（Universal Asynchronous Rx/Tx）和US（synchronous）ART协议；**
  - **SPI和I^2C串行总线，以及其他工业总线协议（Profibus和Modbus）等。**



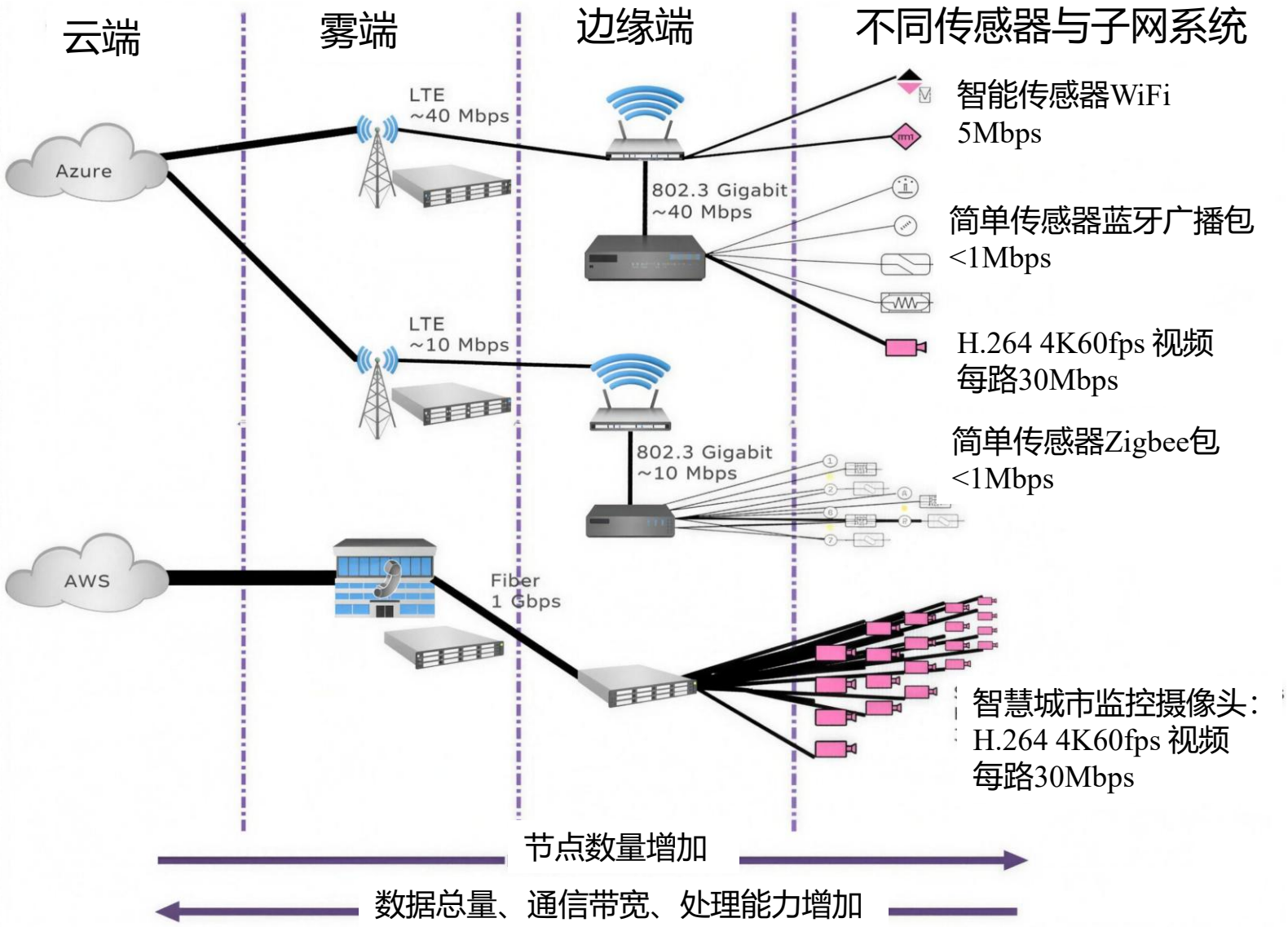
第四章
计算与决策的边缘化




物联网的计算（数据处理）资源分布设计概念

- **云（Cloud）**
  - 一个具有全局（Global，经由网络接入随时随地访问可配置资源的能力）适用性和覆盖面的数据中心，其计算资源对用户以动态扩展的虚拟资源的形式呈现。
- **分布式云（Distributed Cloud）**
  - 基于分布式架构的云计算模式，将云服务供应商的计算和资源分布到不同地理位置（边缘、本地、特定城市）以提高服务的性能与可靠性。分布式云中的不同节点可以进行连接和协作，用来处理负载均衡和环节网络拥塞。一个典型的分布式云的例子是CDN（Content Delivery Network，内容分发网络）。
- **雾（Fog）**
  - 雾计算仍是一种虚拟资源的云计算范式，“雾”表示在地理上将计算资源置于智能终端设备和中心云之间。雾节点间可以用复杂拓扑实现分层、聚合等协作形式。雾计算强调网络化协作。
- **边缘（Edge）**
  - 边缘是包含智能终端及其用户的网络部分（网络边缘），用于在靠近数据产生的位置提供本地计算能力。与雾计算相比，边缘计算的主体更强调单一的、离散的设备或网关。

云-雾-边缘端示意图



- 边缘端的“边缘”指示的是设备在网络中相对于“中心”云端的位置。
- 终端设备只在有中心的联网模式中运行，才能称为“边缘”设备，换言之，这些设备必须和云服务协同工作。
- 离云端越远，计算资源越紧张，设备的分布性（设备数量）也更强。

边缘计算设备实例：常见板上系统（BoS，Board-level System）片上系统（SoC，System on Chip）

X86-Intel赛扬H410工控板

ARM-瑞芯微RK3588核心板

Arduino核心板

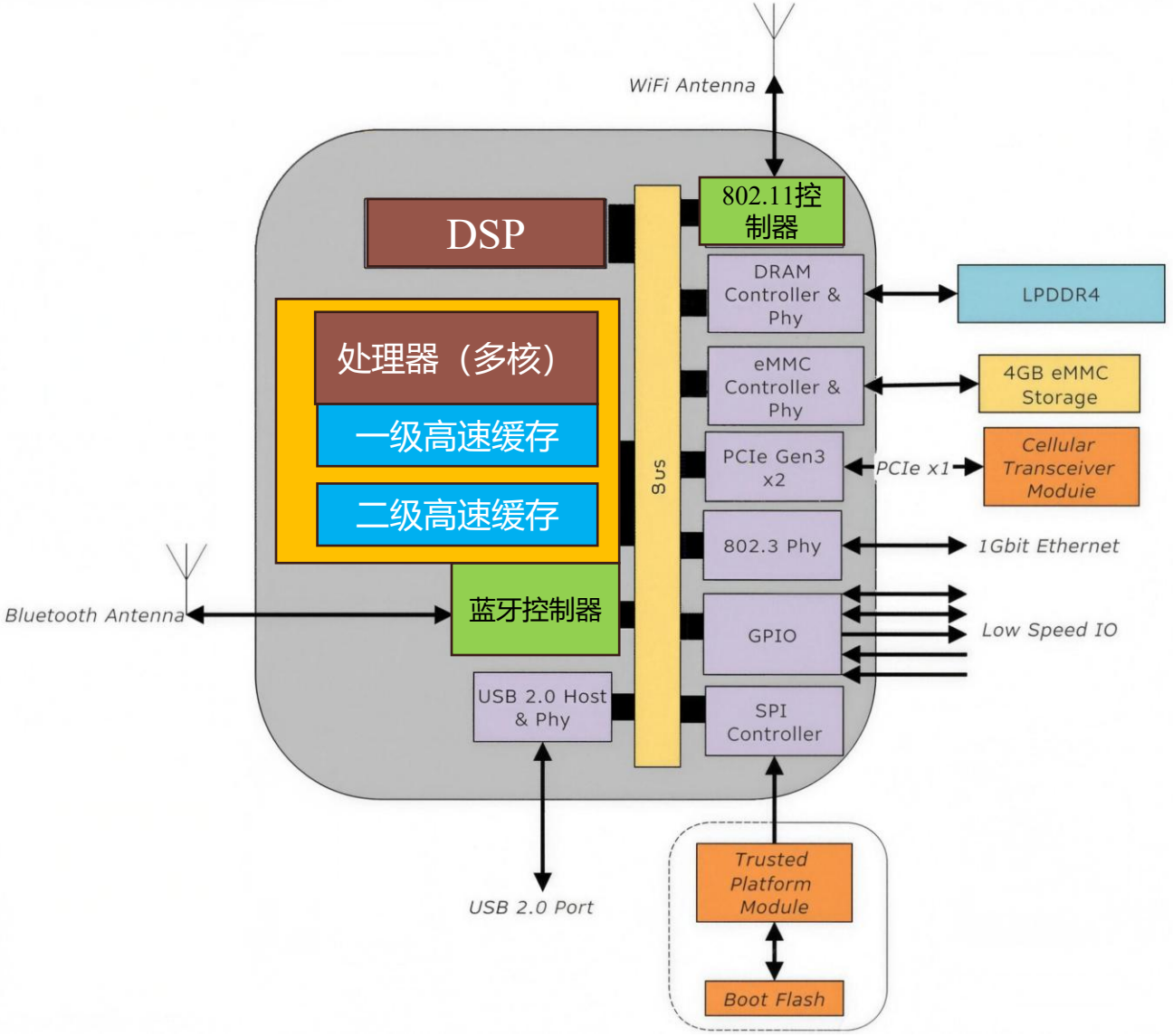
意法半导体STM32微控制器

雾计算/边缘服务器层：复杂数据分析、协议转换等

智能边缘网关/设备：嵌入式推理、多媒体处理等

终端感知与控制：数据采集、实时控制、设备驱动等

典型边缘计算系统结构



一个典型的边缘计算平台硬件组成有：

- 片上系统（SoC）：
  - 一个或多个核心（处理器+缓存）；
  - 一个或多个嵌入式数字信号处理器（根据不同厂商可能实现不同结构，如华为昇腾达芬奇架构还包含AI计算单元-TPU和数字视觉与处理单元-DVPP等）；
- 内、外存模块
  - DRAM（动态随机存取内存）；
  - 非易失性内存（外存）：SATA硬盘、eMMC多媒体卡，NVMe、USB闪存等；
- 无线传输控制器或端口
- 高速I/O
  - USB和以太网接口；
  - PCI Express总线接口；
- 低速I/O
  - 串行总线：SPI、 I^2C；
  - 通用输入输出：串口UART、GPIO；
  - CAN总线，等。

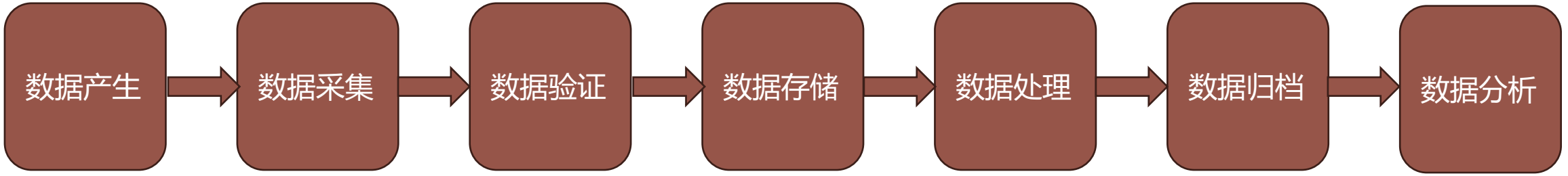
探究问题：为何需要引入边缘计算?

物联网（IoT）数据的一些关键特征

- **大**数据：大规模终端节点生成大量关于物联网设备感知、驱动物理过程的数据；
- **异构**数据：数据由大量不同的传感器产生，存在采样率、数据幅值、数据质量（如精度、噪音等）等方面的差异；
- **实时**数据：物联网的数据由传感器实时生成，终端执行器也可能需要实时控制输入。数据的业务价值往往取决于数据处理的实时性。
- **时空耦合**数据：绝大多数物联网数据不仅在时间上呈现顺序性（sequential），在空间上也存在耦合，数据的评估结果可能随时空关系变化而变化；
- **多模态（Multi-modal）**数据：物联网过程获取和使用的数据可能很复杂，涉及不同的维度，如图像、声音、文本等数据可能被同时处理和分析，典型应用场景如基于车联网的自动驾驶。
- **易失性**数据：物联网数据往往不能批量加载，而是按流式（stream）生成/传入的，在本地寿命有限。

探究问题：为何需要引入边缘计算?

物联网（IoT）数据的一般管理流程



由传感器或智能终端主动或被动生成数据，待用于传输的过程

通过有线或无线链路与分布式设备通信以获取数据的过程

在特定的操作环境（上下文/Context）中基于规则、语义或其他逻辑验证所获得数据的正确性

将大规模物联网的流数据和信息捕获并存储，以供后续进一步分析和处理的过程

数据处理既包含对已存储的静态数据的处理，也包含对流数据的处理，这可以是简单的数据转换和预处理，也可以使复杂的数据挖掘（如基于机器学习的分类）

数据归档（或数据剩余）是对物联网数据的生命周期管理。在有限存储资源条件下，对服务关联数据做全域数据归档或删除的操作

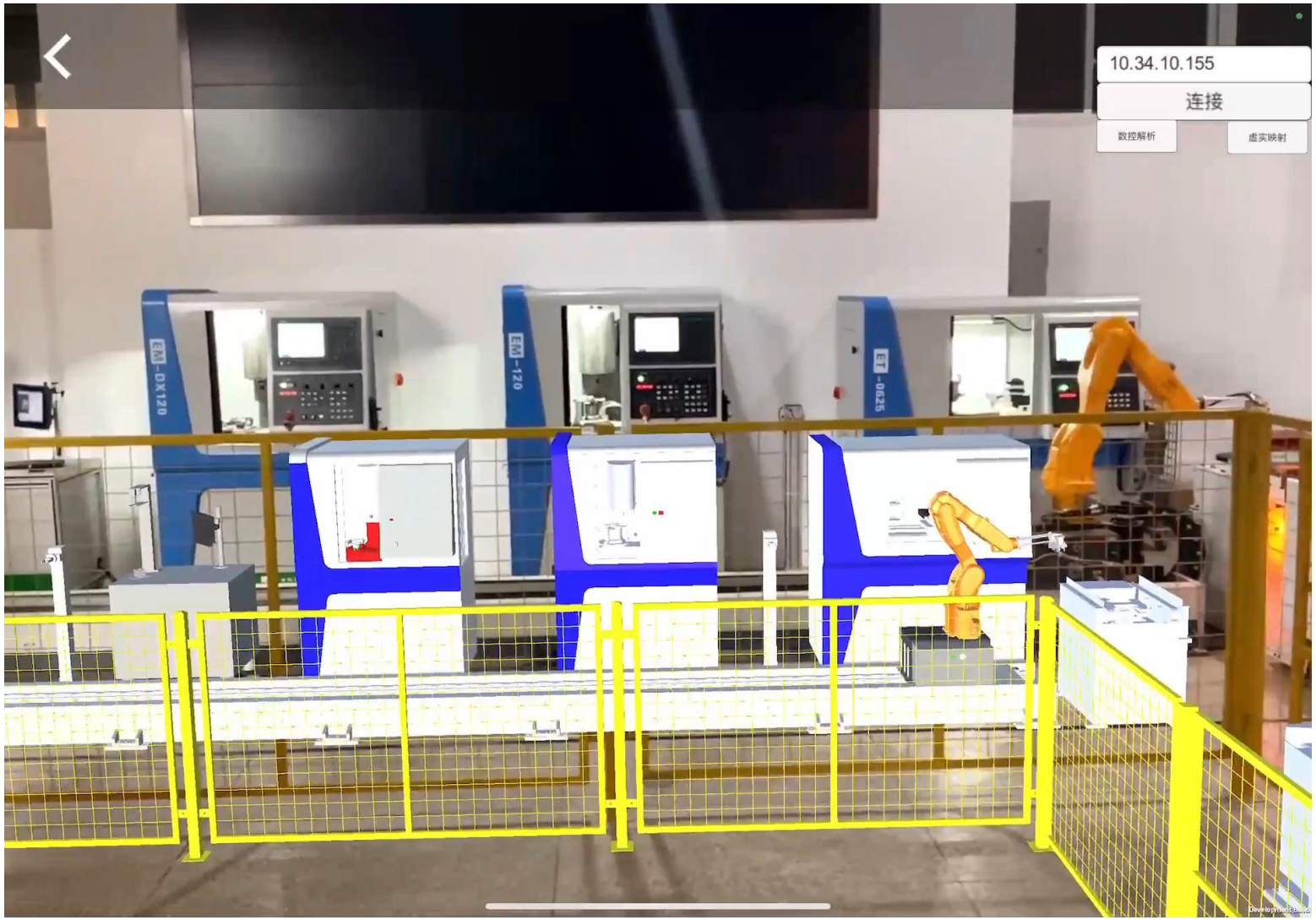
数据分析一般是指对数据库中的已存储数据进行分析，以获得隐藏信息的过程。分析结果用于支持服务（控制）决策过程。

边缘计算资源的部署原则：将数据适配计算，或将计算适配数据

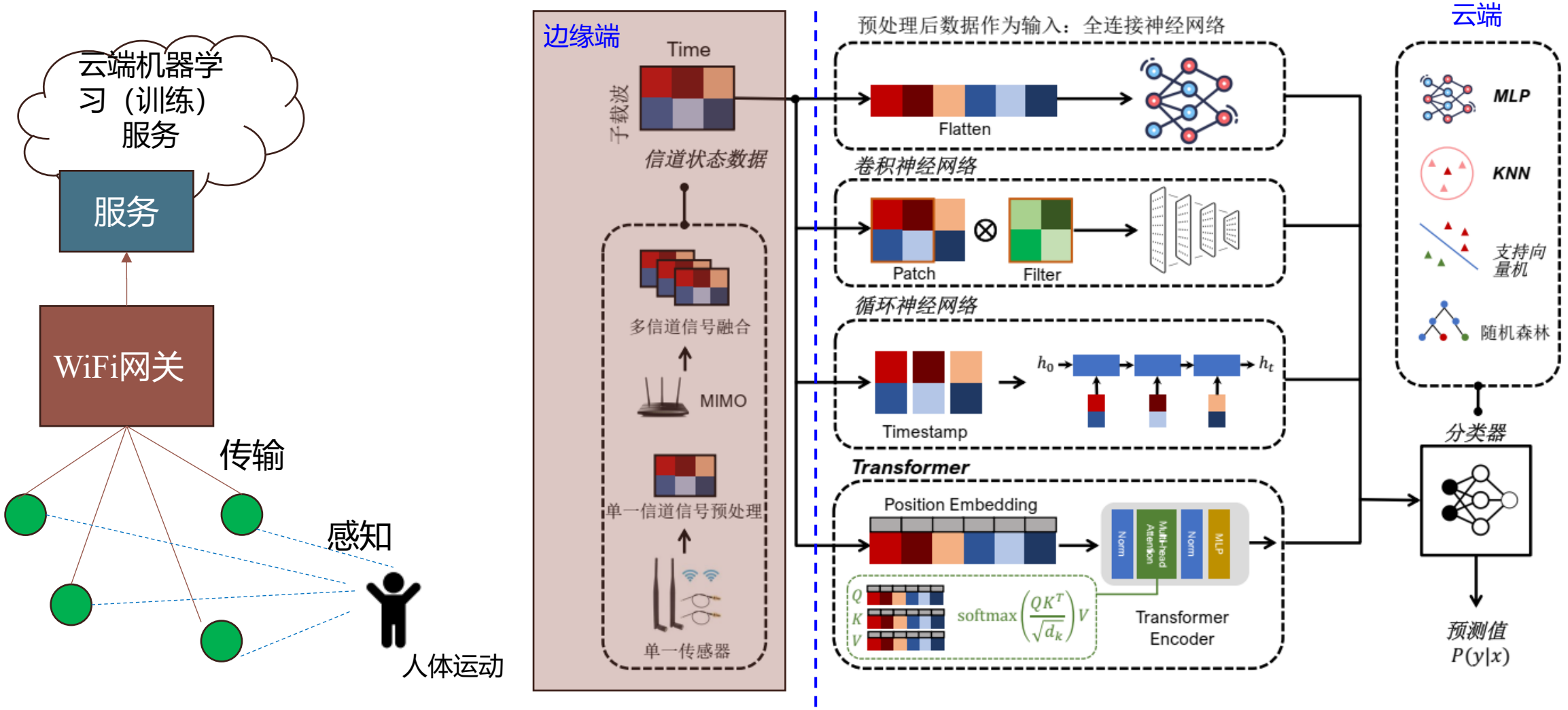
将（部分）计算和数据处理向靠近数据产生的位置部署

- 降低延迟
  - 通过将边缘系统部署在距离终端设备更近的地方，避免数据传输中因网络跳转和传输延时造成的数据处理时延。适用于某些对延迟敏感的应用，如基于物联网数据的数字孪生3D渲染等。
- 优化数据吞吐带宽
  - 在对进出边缘系统的带宽受限的情况下，或云端的数据处理成本随边缘节点增长而超过线性增长时，边缘节点可以通过预处理、压缩或降维等方式，节省带宽或提高给定带宽的效率。
- 安全和隐私
  - 在某些情况下，本地数据因用户隐私和数据安全原因，无法直传云端或其他边缘设备，典型场景如涉及健康监测的数据、或图像等涉及个人隐私监控的数据。这需要边缘侧的大量计算资源对数据进行脱敏处理等预处理操作。

边缘计算的典型场景I-延迟约束：数字孪生产线的3D渲染

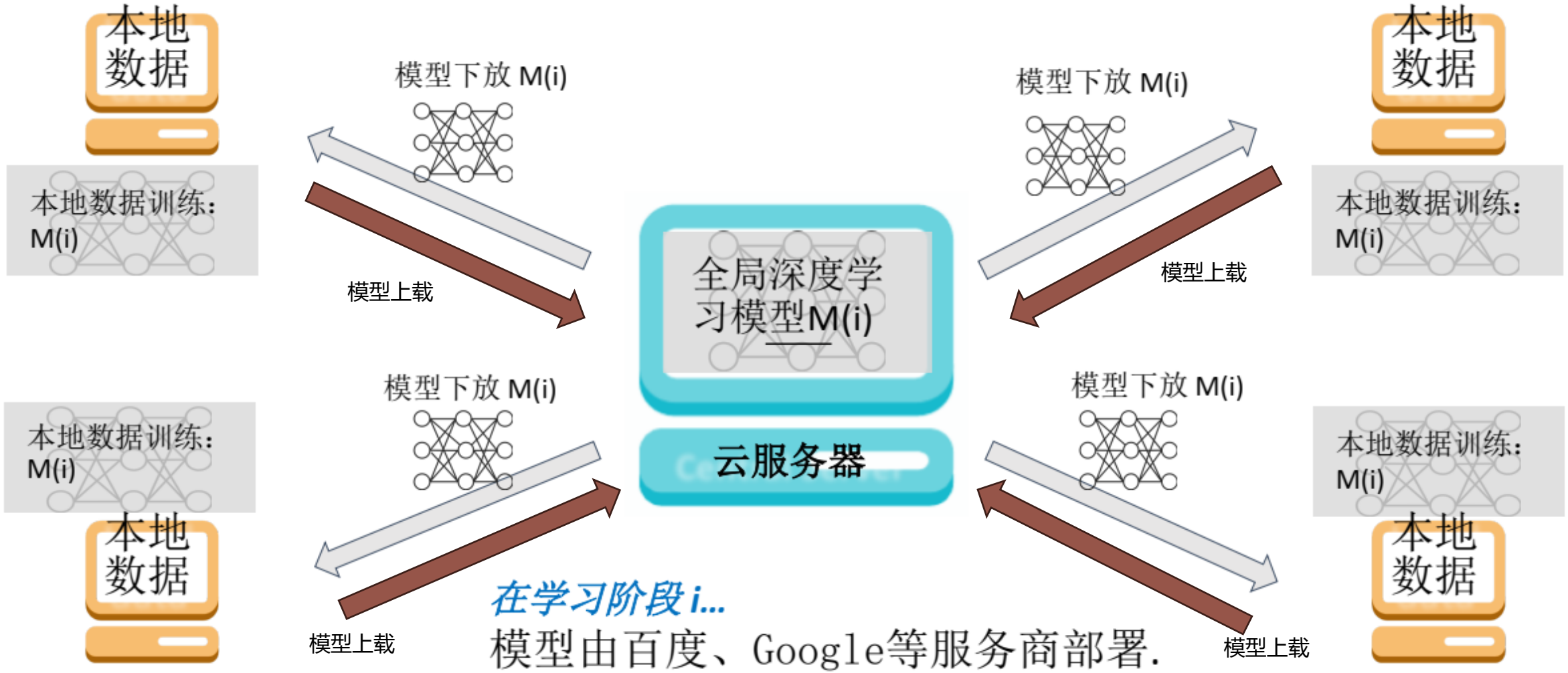


边缘计算的典型场景II-数据预处理：基于无线传感网络的人体姿态识别



图片来源：Jianfei Yang, et. al, “SenseFi: A Library and Benchmark on Deep-Learning-Empowered WiFi Human Sensing”, arXiv:2207.07859, 2022

边缘计算的典型场景III-隐私计算：联邦学习



在学习阶段 i...

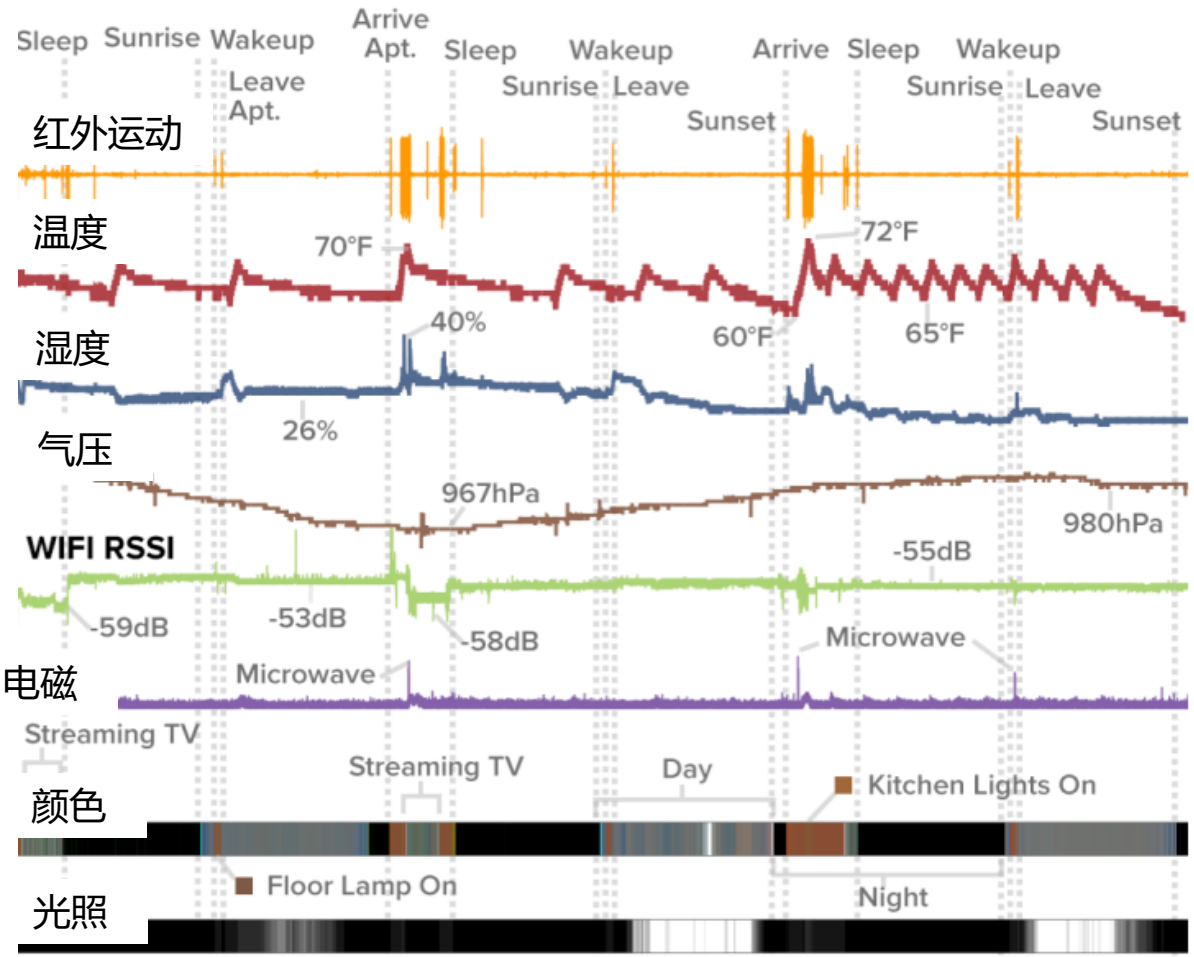
模型由百度、Google等服务商部署.

边缘智能：边缘节点的自主决策 + 边缘节点的自组织管理

- **基于单一边缘节点的合成感知和决策**
  - **由边缘节点的计算-存储资源和任务的QoS（Quality-of-Service）需求共同决定是否将数据处理-决策任务（如训练-推理-分类-决策）在边缘节点执行。**
  - **案例：一台刀片服务器+多个环境传感器（非视觉传感器）可以用来部署机器学习模型，在本地训练并推理实时传感信号和所在环境之间的关系。**

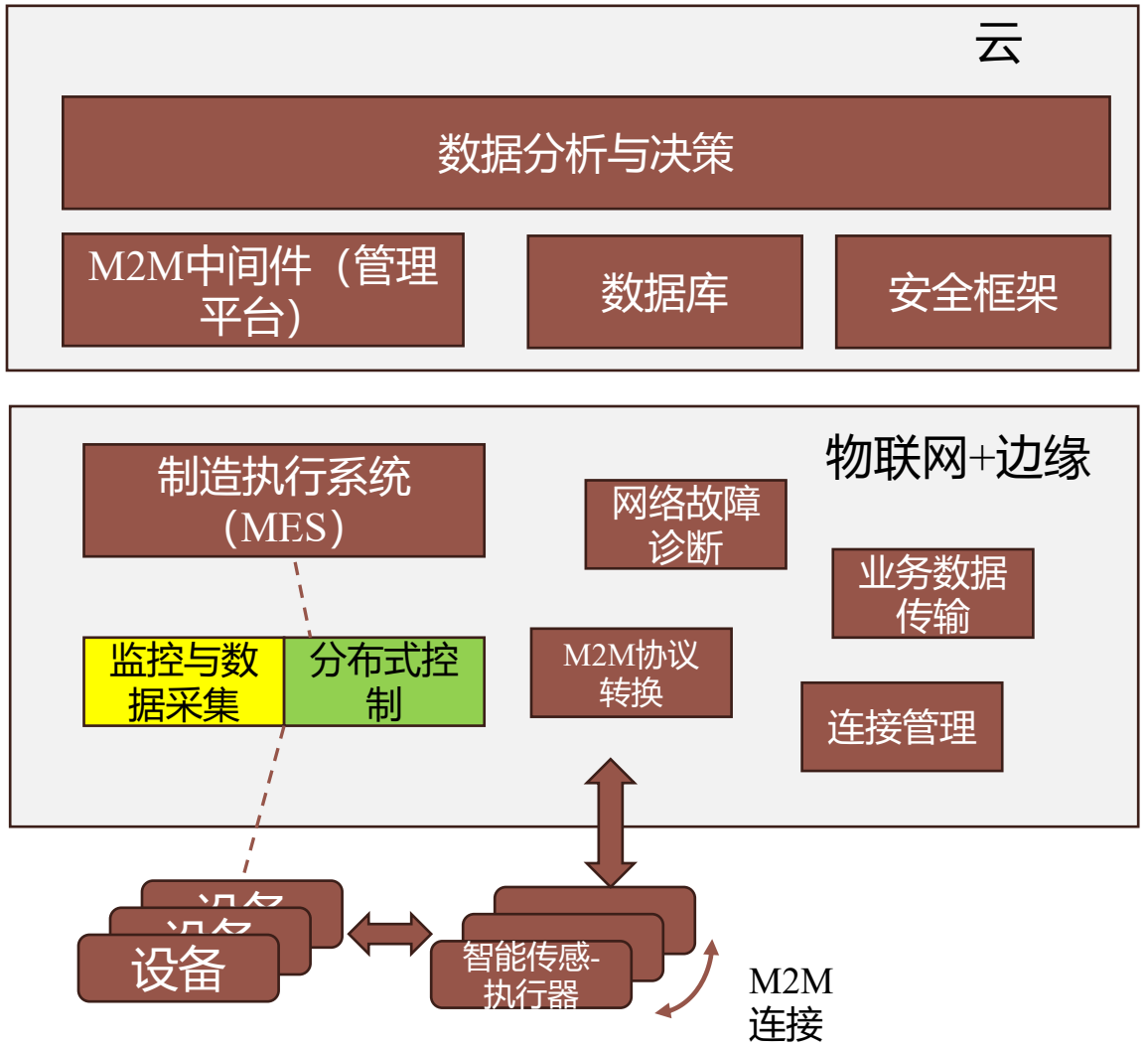
Adapted From [1] Gierad Laput, et. al, “Synthetic Sensors: Towards General-Purpose Sensing”, ACM-CHI ’17, 2017

多传感器（Y轴）在边缘端融合检测时序环境事件（X轴）



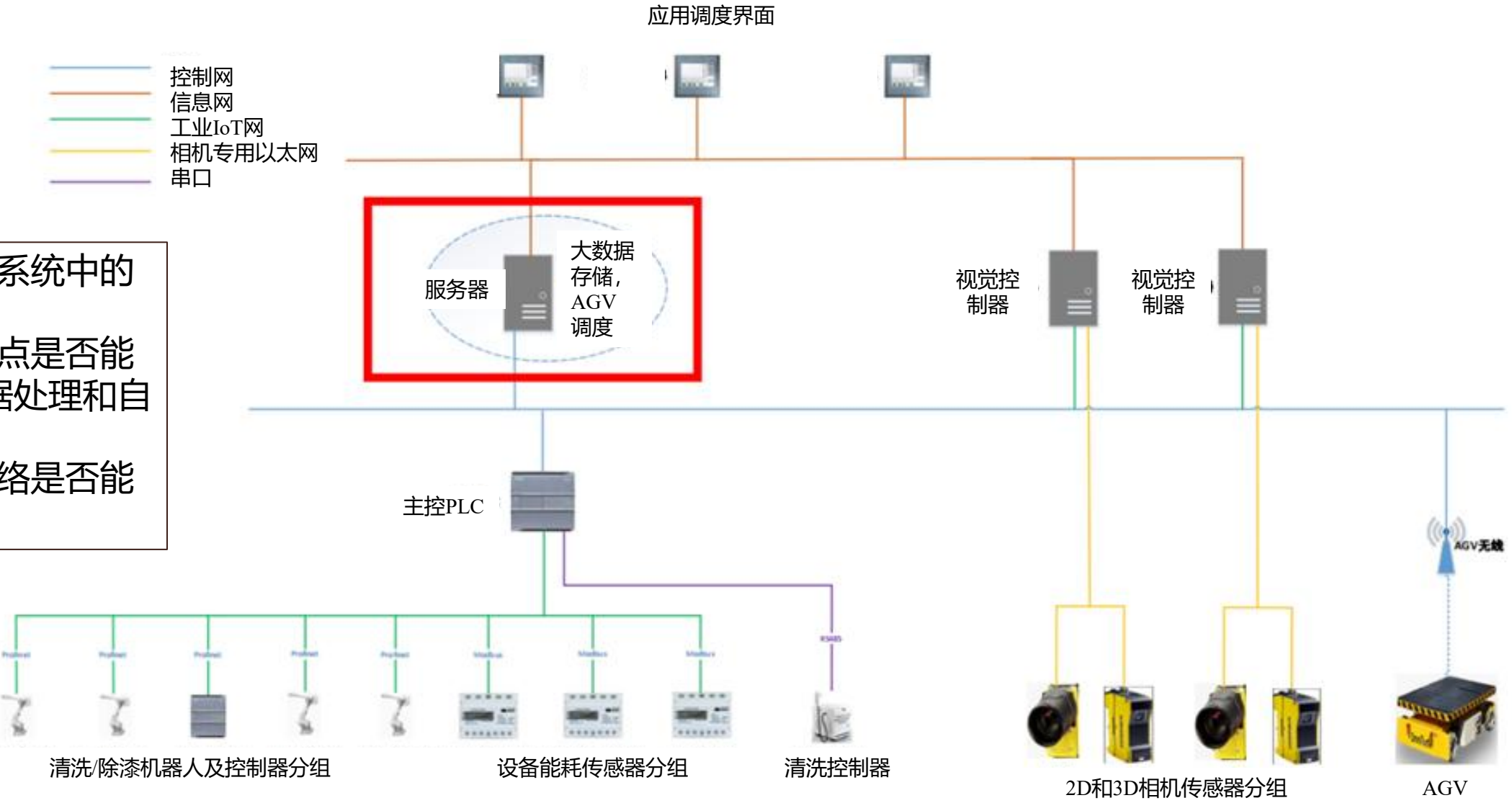
边缘智能：边缘节点的自主决策 + 边缘节点的自组织管理

- **基于M2M（机器间通信）架构的节点自组织管理**
  - **自主终端接入：由M2M设备和M2M网关实现网络的自动接入，自动身份认证和配置、设备/连接故障诊断等功能。**
  - **自主业务处理：根据M2M或应用发出的请求消息执行对应的逻辑处理，M2M管理平台（一般是中间件）对组网之外的业务消息进行自主解析、协议转换、转发/推送。**

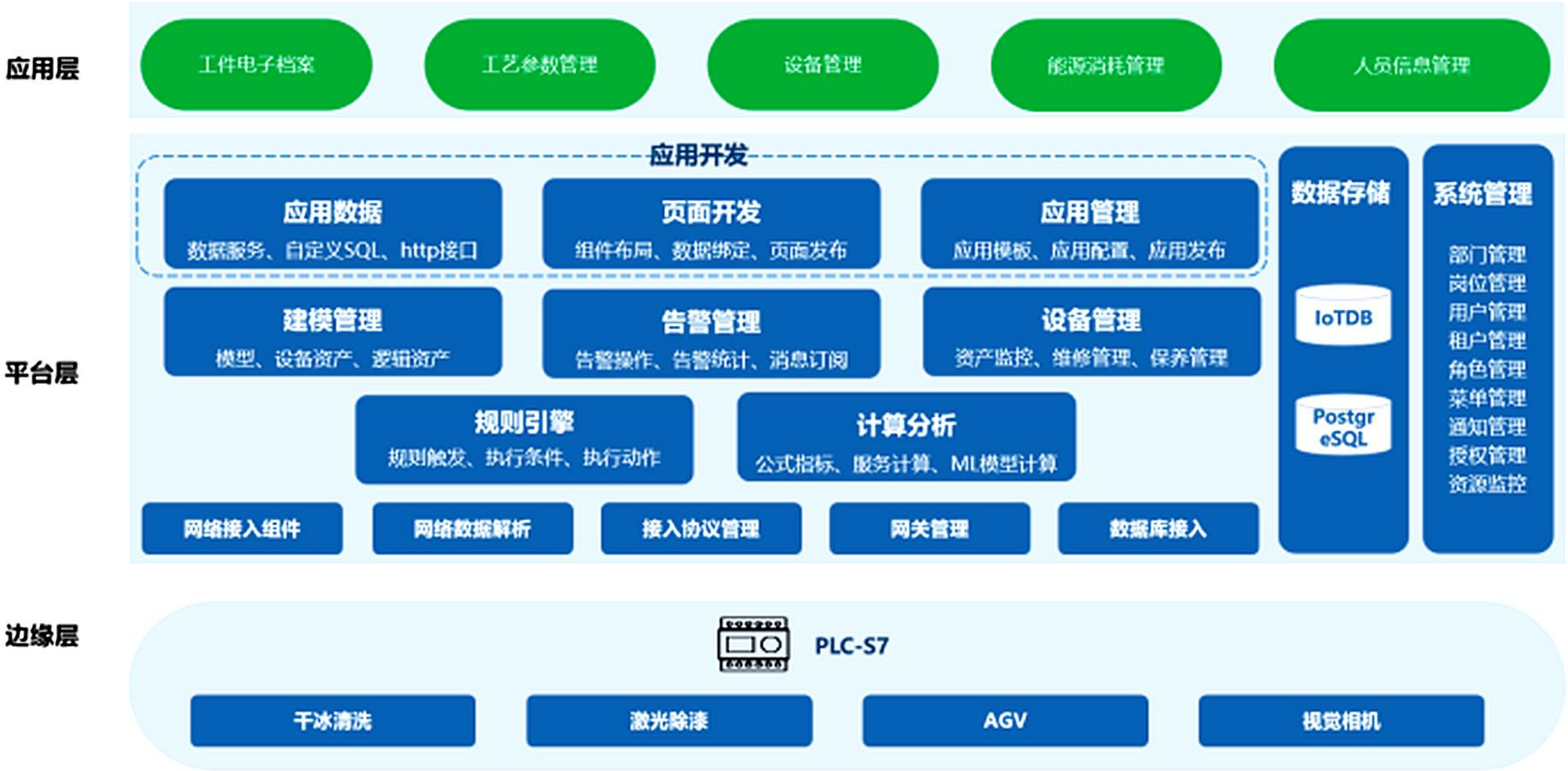


企业案例分析：基于IoT的某喷漆清洗车间管理系统

问题1：请识别系统中的边缘节点。
问题2：边缘节点是否能够独立完成数据处理和自主决策?
问题3：边缘网络是否能够实现自组织?

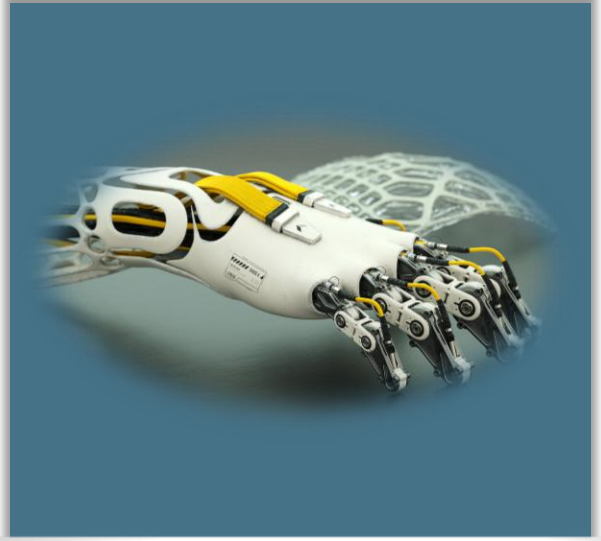


案例分析：基于IoT的某清洗车间管理系统（续：云-边协同系统设计）

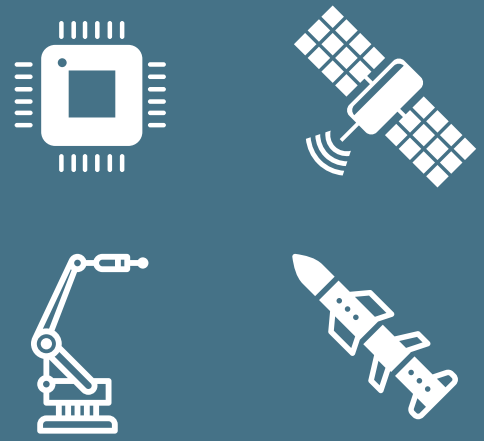


案例系统设计要点

- 基于M2M（机器间通信）架构的节点自组织管理
  - **系统硬件和网络架构模块化设计**：包括边缘智能系统的部署和配置，工业相机、AGV、操作屏、主控PLC等设备的接入和通讯。
  - **数据存储与备份、归档**：包括对采集的数据进行存储、备份和数据分析，以提供设备运营管理和生产管理的支持。
  - **多网融合协议支持**：支持多个工业总线协议和数据采集接口标准。支持子网间数据转发和集成。
  - **边缘智能管理**：实现边缘节点的自组织组网，包括边缘节点的注册、配置、监控、删除等完整生命周期管理。
  - **数据分析能力**：系统支持时序数据落库（如IoTDB）。系统支持基于预设机器学习模型的数据自动分析、故障预测告警、设备维护管理等。
  - 其他：系统访问控制、数据加密、日志、可扩展性设计等。

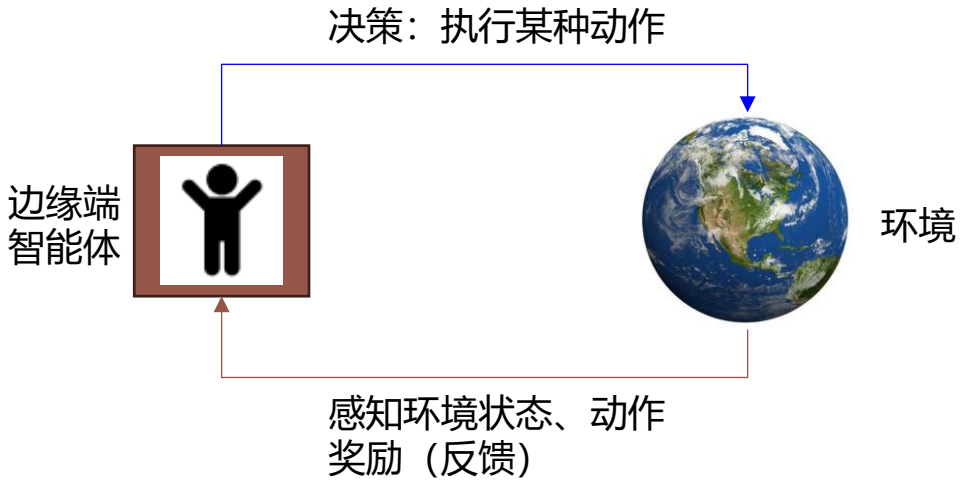


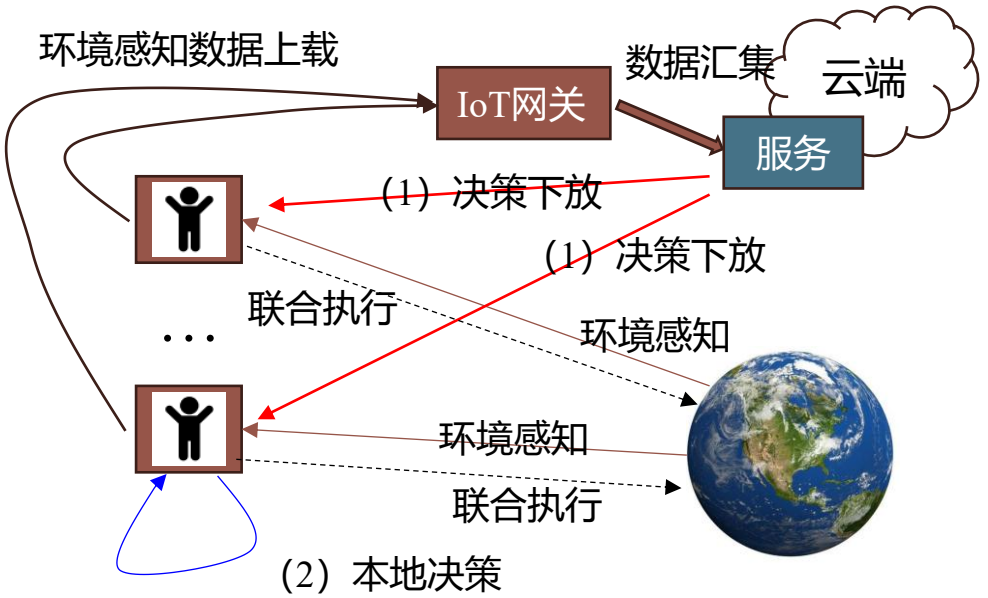
第四章
计算与决策的边缘化



感知-决策的信号流

- **传感-决策功能都下放到边缘端的情况**
  - **单一边缘端智能体：本地环境感知+行为决策**
- **传感-决策功能部分下放到边缘端的情况**
  - **联合感知：本地环境感知+中心（云端）感知数据融合+中心（云端）决策；**
  - **联合决策：本地环境感知+中心（云端）感知数据融合+边缘端决策**





板载操作系统：执行-决策的底层支撑和软件载体

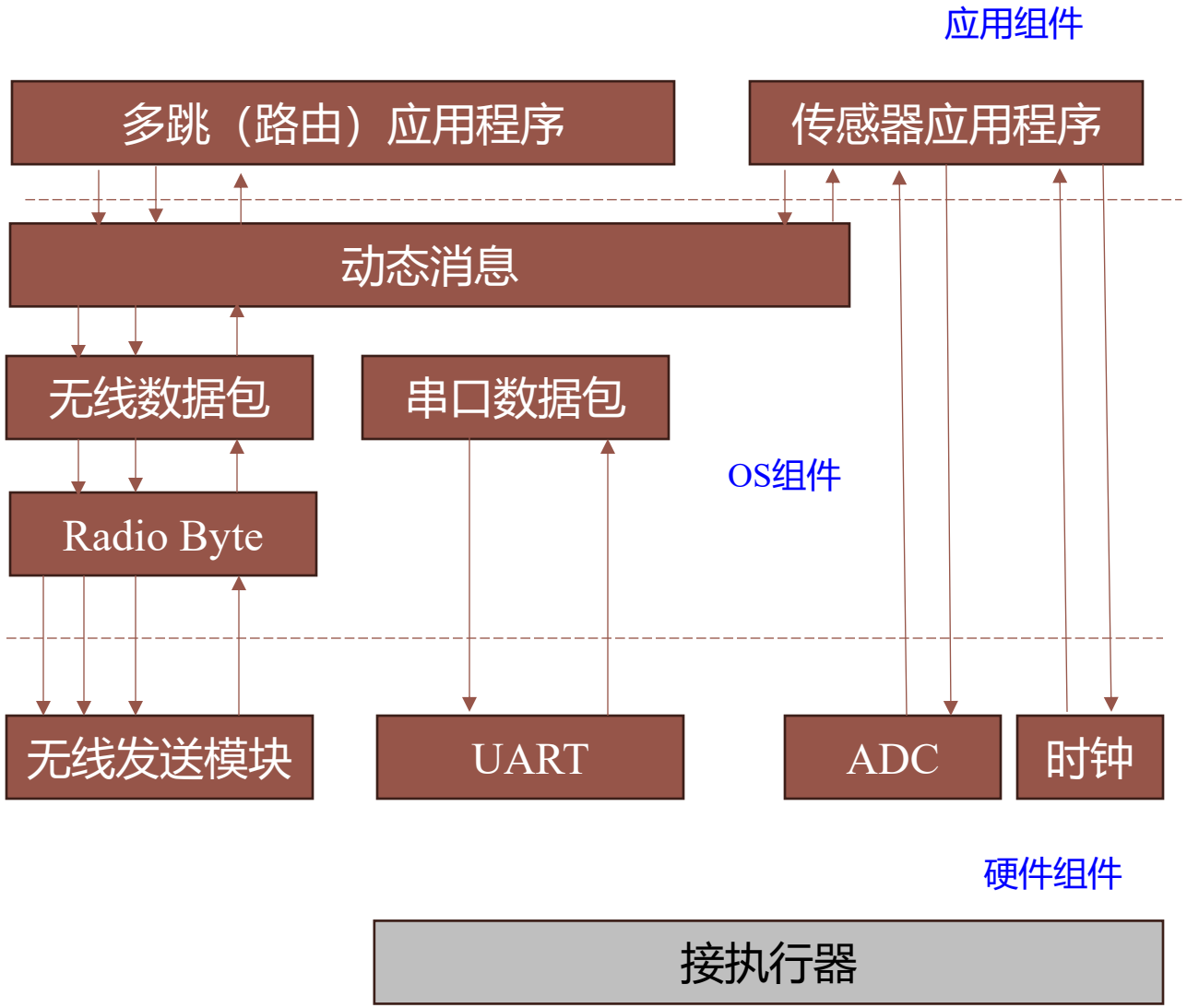
- **边缘端设备操作系统的主要服务**
  - **硬件抽象与资源管理**
    - 内存分配、电源/功耗管理、设备驱动框架、时钟与中断控制；
  - **处理器与任务调度**：
    - 进程/线程管理、实时调度算法（如优先级抢占）、中断服务例程（ISR）管理。
  - **数据与存储服务**：
    - 文件系统、轻量级数据库、本地数据缓存与持久化。
  - **网络通信与协议栈**：
    - 多网卡管理（Wi-Fi/5G/BT）、TCP/IP协议栈、工业协议转换网关。**安全服务；**
  - **应用运行与容器服务：**
    - 边缘计算引擎、轻量级容器运行时（Docker/LXC）、流数据处理框架。
  - **安全与防护机制：**
    - 可信执行环境（TEE）、加解密API、身份认证、访问控制（贯穿上述所有层）。
  - **外围接口与开发者生态：**
    - CLI/GUI界面、SDK、编译器/调试器、远程日志与监控。

操作系统的选择要点

- 是否有实时需求，操作系统是否支持实时或扩展?
- 操作系统支持那种处理器（x86/ARM/RISC-V）？
- 操作系统是否支持所需的处理器特性：虚拟内存、SIMD、浮点仿真等。
- 操作系统是否支持深度精简：内存和存储的最小消耗是多少?
- 需要什么样的文件系统，是否支持本地数据库部署?
- 操作系统是否支持所需的通信与网络协议栈?
- 操作系统支持什么样的开发环境和软件包?
- 是否需要用户图形界面（GUI）？

常见的嵌入式操作系统

- **深度精简系统（微系统）：TinyOS**
  - **Berkeley U开发的开源嵌入式系统，内核很小（核心代码和数据大约400B）。**
  - **组件式（Component-based）开发：通过连接配置文件（configuration）将两个以上的组建相互连接（wiring）组成。组建是操作系统的基本构建模块。**
  - **多个组建配置成单一的可执行代码，操作系统即应用（没有明确区分点），静态内存分配，FIFO任务调度管理，无系统调用管理。**
  - **提供基于C语言扩展的nesC开发环境。**
  - **目前常用于高校教学。**



常见的嵌入式操作系统

- **开源精简系统：μC/OS-II / μC/OS-III**
  - **开源嵌入式系统，核心代码（任务调度、中断管理）通常只有 几 KB 到 几十 KB（如 μC/OS-II 核心约 6KB~12KB）。**
  - **硬实时操作系统（RTOS），直接工作在物理内存上，通过关闭中断和临界区保护来保证内存操作的原子性和系统的实时性。**
  - **提供的专属“内核对象”：信号量（Semaphore）、互斥锁（Mutex）、消息队列（Message Queue）和事件标志组（Event Flag）。但不兼容POSIX 标准。**
  - **95% 以上的代码由 ANSI C 编写。**
  - **广泛用于电机控制、无人机飞控、医疗传感器等领域。**

Linux衍生的操作系统

- **精简系统： ucLinux**
  - 开源嵌入式系统，内核大小约数百KB。
  - 针对无内存管理单元（MMU）的处理器设计，对MIPS、ARMv4指令集等都有良好兼容，采用是“平坦内存模型” （Flat Memory Model） 。
  - ucLinux没有fork()函数，因此不能实现多进程（任务）管理，但其支持POSIX标准下的pthread线程标准库。
  - ucLinux下的程序通常使用C语言开发。
  - 目前常用于高校教学。
- **实时系统： RTLinux (Real-Time Linux)**
  - RTLinux在Linux内核与硬件之间增加了一个虚拟层，构建了一个小的、时间上可预测的、与Linux内核分开的实时内核（双内核）。这使得在其中运行的实时进程能够满足硬实时性要求。
  - RTLinux采用优先级调度策略，将原始Linux内核作为空闲时运行的低优先级线程运行。
  - RTLinux使用硬中断管理任务，能够管理符合POSIX.1b，最小延迟是15μs 的实时进程。
  - 在工业机床、半导体设备、军工领域极其常见。

新崛起的嵌入式操作系统

- **鸿蒙（OpenHarmony）嵌入式内核（LiteOS）：LiteOS-M 和 LiteOS-A**
  - **支持“多进程”与“多线程”双模**
    - LiteOS-M（轻量系统，128KB RAM起）类似 uC/OS，采用单进程多线程模型；
    - LiteOS-A（小型系统，1MB RAM起）引入了真正的多进程模型。它利用 MMU（内存管理单元）实现了内核态与用户态的隔离。
  - **支持高级的虚拟内存管理**
    - LiteOS-M：采用物理内存直接操作，提供静态和动态内存算法。
    - LiteOS-A：支持虚拟内存管理（MMU）。内核和用户进程拥有独立的地址空间，支持按需分页和写时复制（COW）。
  - **高度兼容POSIX 标准。**
  - **分布式软总线（Distributed Soft Bus）特性**
    - 鸿蒙区别于所有传统 RTOS 的最核心特征。它可以将多个物理上分离的设备（如手环、手机、耳机）在软件层面虚拟成一个超级终端。

无完整操作系统下的本地感知-执行用例开发

- 基于Arduino系统的开发
  - Arduino开发板上的微控制器（如ATmega328P或ATmega2560等）运行的是直接写在微控制器芯片上的固件（管理输入/输出、处理中断等），通常是一个没有操作系统的裸机程序。
  - 开发者编写的Arduino程序（Sketch）成机器码，并通过Arduino IDE上传至上执行。
  - 每一个Arduino程序（Sketch）都有一文件，后缀为.ino。文件中需要包含和loop()函数，通过调用其他模块化代码实现复杂功能。

- 简单用例
  - 本地感知：通过microphone检测环境噪音量；
  - 本地执行：驱动LED灯发出报警信号；
  - 通常使用C/C++开发；
  - Setup()函数：

```
const int ledPin =  12;          // the number of the LED pin
const int thresholdvalue = 400; //The threshold to turn the led on

void setup() {
 pinMode(ledPin, OUTPUT);
 Serial.begin(9600);
}
```

设置串行比特率

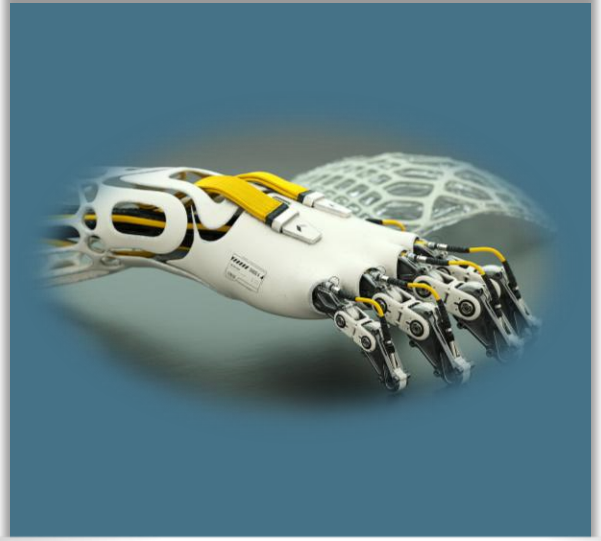
无完整操作系统下的本地感知-执行用例开发（续）

- **基于Arduino系统的简单开发实例：噪声报警**
  - **Loop()函数**

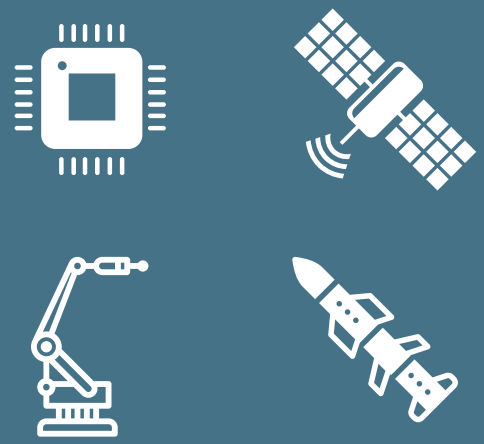
```
13  void loop() {
14      int sensorValue = analogRead(A0);   //use A0 to read the electrical signal
15      Serial.print("Noise detected=");
16      Serial.println(sensorValue);
17      delay(100);
18      if(sensorValue > thresholdvalue)
19      digitalWrite(ledPin,HIGH);//if the value read from A0 is larger than 400,then light the LED
20      delay(200);
21      digitalWrite(ledPin,LOW);
22
23  }
```

模拟输入量引脚

输入信号大于阈值（400）的情况下，先点亮LED，
延迟200毫秒后，
熄灭LED。



第四章
计算与决策的边缘化



4.0 前章回顾

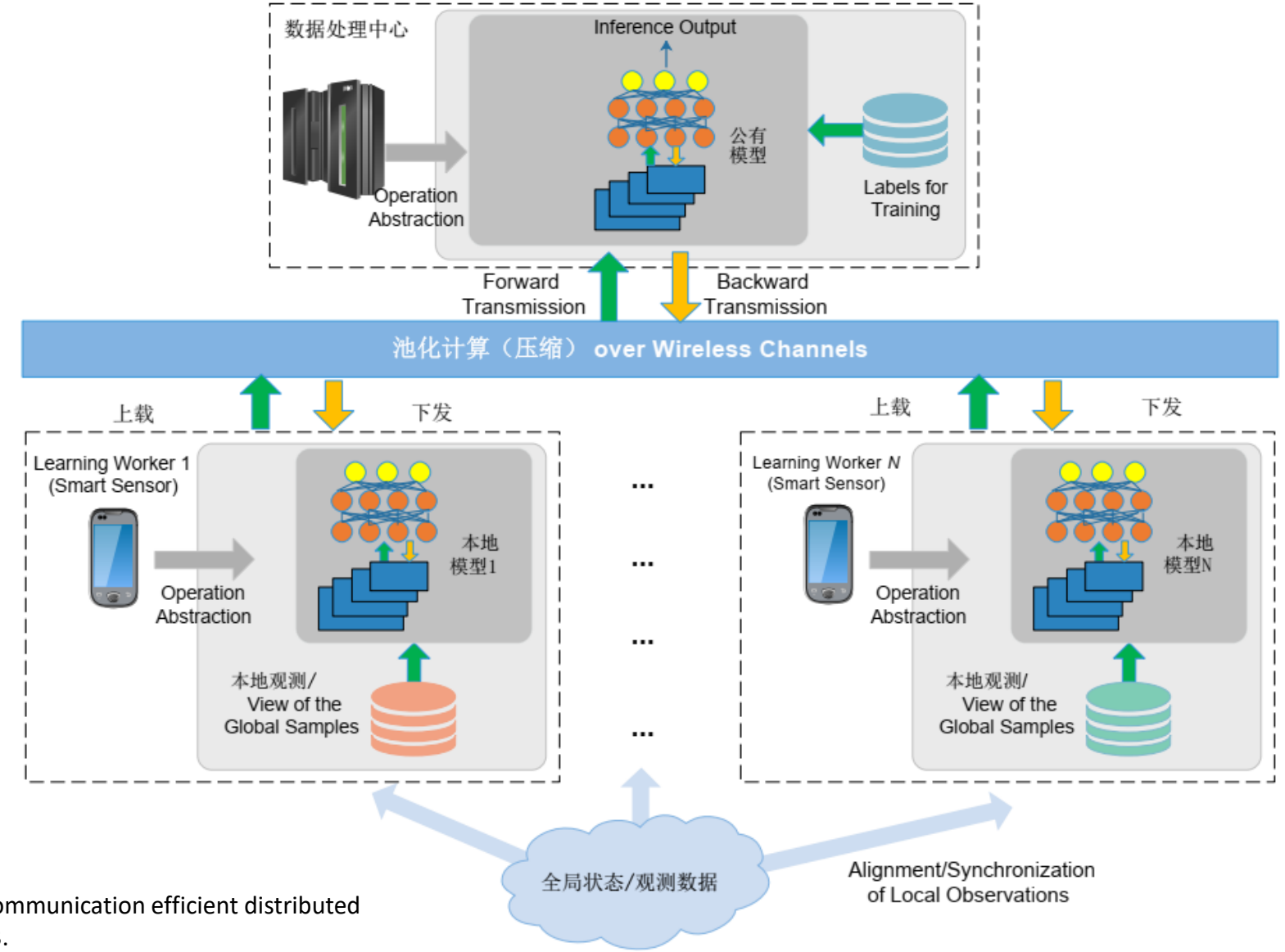
4.1 边缘计算与边缘智能

4.2 从传感器到执行器

4.3 感知与决策的边缘端融合：联邦学习概览

联邦学习（Federated Learning）

- **基本步骤（横向联邦学习）**
  - **（1）数据中心向智能终端（边缘端Learning Worker）下发同一结构的神经网络模型。**
  - **（2）边缘设备在本地持有数据（输入-输出结构一致），基于本地训练集训练所收到的神经网络模型。**
  - **（3）边缘设备将训练后的神经网络模型参数（权重）上载至数据中心。**
  - **（4）数据中心通过聚合算法更新服务器端/云端模型。**



图片来源：I. Achituve, Wenbo. Wang, E. Fetaya, and A. Leshem, “Communication efficient distributed learning over wireless channels,” IEEE Signal Processing Letters, 2023.

（横向）联邦学习的特点

- 各边缘设备持有不同的训练输入数据集，但共享相同的样本标签（Label）集。
- 基于全连接神经网络的示例：
  - 某边缘设备 n 持有训练集 $\mathcal{X}=\{(\mathbf{x}^{[m]},\mathbf{y}^{[m]})\}_{m=1}^{|\mathcal{X}|}$，其中所有设备的输入组成全局特征 $\mathbf{x}=[\mathbf{x}_1^{\mathrm{T}},\ldots,\mathbf{x}_N^{\mathrm{T}}]^{\mathrm{T}}$。
  - 假设边缘设备本地的神经网络输出为多层全连接层映射，对单个样本 $\mathbf{x}_n$ 而言，输出为：

$$f_n(\mathbf{x}_n;\boldsymbol{\theta}_n)=\left(\boldsymbol{\omega}_n^L\right)^{\mathrm{T}}\sigma_L\left(\left(\boldsymbol{\omega}_n^{L-1}\right)^{\mathrm{T}}\sigma_{L-1}\left(\ldots\sigma_1\left(\left(\boldsymbol{\omega}_n^1\right)^{\mathrm{T}}\mathbf{x}_n\right)\right)\right)$$

  其中，所有L层网络权重组成参数集 $\boldsymbol{\theta}_n=\{\boldsymbol{\omega}_n^\ell\}_{\ell=1}^L$

  - 理想状态下，对数据中心而言，希望最小化来自所有边缘设备的本地训练损失函数的期望值：

$$(\boldsymbol{\theta}_0^*,\ldots,\boldsymbol{\theta}_N^*)=\arg\min_{\boldsymbol{\theta}_0,\ldots,\boldsymbol{\theta}_N}\mathbb{E}_{\mathbf{x}^{[1]},\ldots,\mathbf{x}^{|\mathcal{X}|}}\frac{1}{|\mathcal{X}|}\sum_{m=1}^{|\mathcal{X}|}\mathcal{L}\left(\hat{\mathbf{y}}^{[m]},\mathbf{y}^{[m]}\right)$$

（横向）联邦学习的特点（续）

- 假设总共有N个边缘设备，其中第n个设备持有k_n个训练样本，则本地训练损失函数：

$$L_n(\theta_n) = \frac{1}{k_n}\sum_{j=1}^{k_n} L(\hat{y}(x_{nj}, \theta_n), y_j)$$

- 根据自动微分反向传播规则更新网络参数
  - 对第n个设备而言，通过SGD等梯度下降方法更新网络参数训练结果：

$$\theta_{t+1}^n \leftarrow \theta_t^n - \eta \nabla L_n(\theta_n)$$

- 根据著名的FedAvg算法，数据中心可以接受如下两类模型更新方式
  - 边缘设备直接提交权重 θ_{t+1}^n，数据中心对其进行加权：$\theta_{t+1} = \sum_{n=1}^{N} \frac{k_n}{\sum_j k_j} \theta_t^n$ ← 总样本数
  - 边缘设备提交梯度，数据中心据此进行加权梯度更新：

$$\theta_{t+1} \leftarrow \theta_t - \eta \sum_{n=1}^{N} \frac{k_n}{\sum_j k_j} \nabla L_n(\theta_n)$$

← 总样本数

结语（Q&A）

- 参考文献

- [1] Ruben Oliva Ramos，杨悦伦译，物联网系统开发——树莓派JavaScript编程指南，机械工业出版社，2018.
- [2] 王汝传等，无线传感器网络技术及其应用，人民邮电出版社，2011.
- [3] Vlasios Tsiatsis等，王慧娟等译，物联网架构、技术及应用，机械工业出版社，2021.
- [4] 郭斌等，智能物联网导论，机械工业出版社，2022.
- [5] Sebastian Raschka等，陈斌译，Python机器学习，机械工业出版社，2021.
- [6] Perry Lea, 中国移动设计院北京分院译，物联网系统架构设计与边缘计算，机械工业出版社，2021.