Designing High-Confidence Embedded Systems

Operating Systems and Real-Time

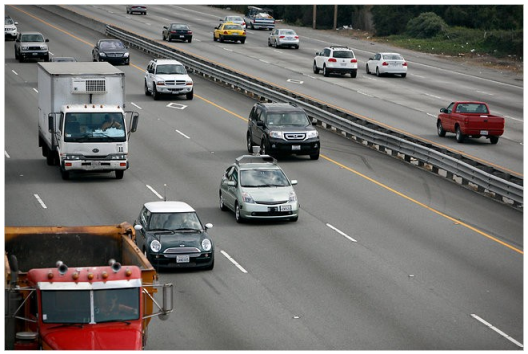
Gabe Parmer

Embedded Systems

Computers *embedded* in the physical world

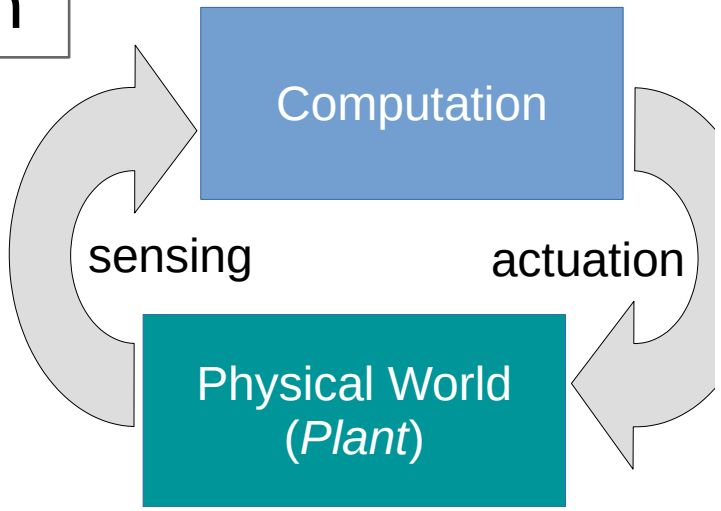
Computations **responsible** for physical interaction

Errors/Attacks → potential physical catastrophe



Embedded Systems

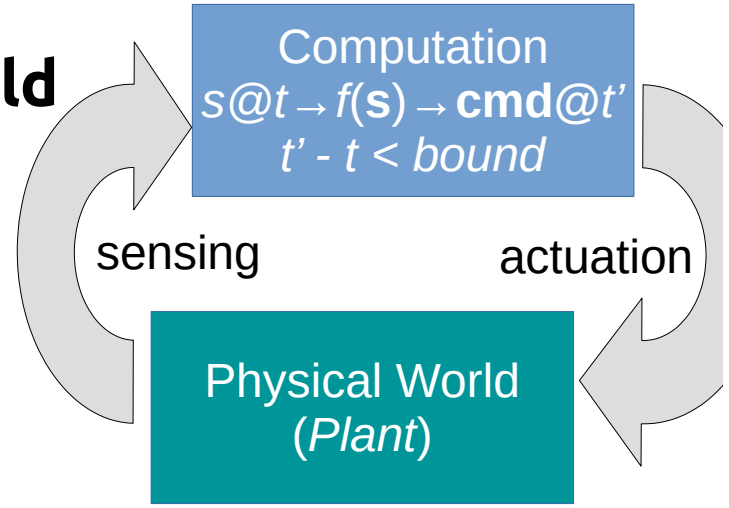
- Computers that interact with physical world
  - Sensors + actuators + computation
- Replace mechanical w/ digital
  - Microwaves, toys
  - Cars, airplanes



Real-Time/Cyber-Physical Systems

Embedded systems + **deadlines**

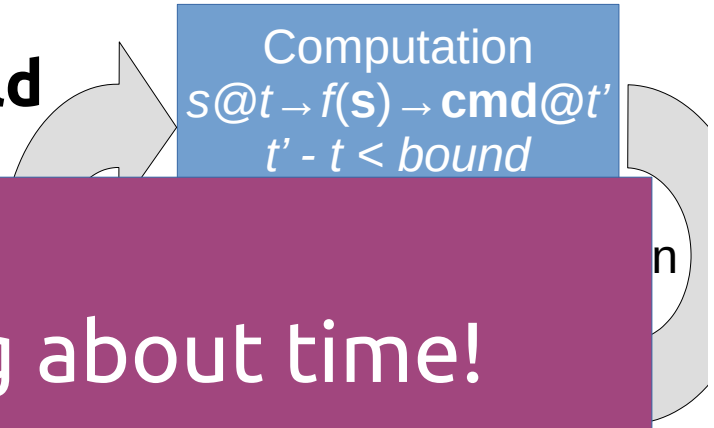
- The computation must adhere to the **timing constraints of the physical world**
- Send command to actuator within a **bounded time** from reading sensor



Real-Time/Cyber-Physical Systems

Embedded systems + **deadlines**

- The computation must adhere to the **timing constraints of the physical world**

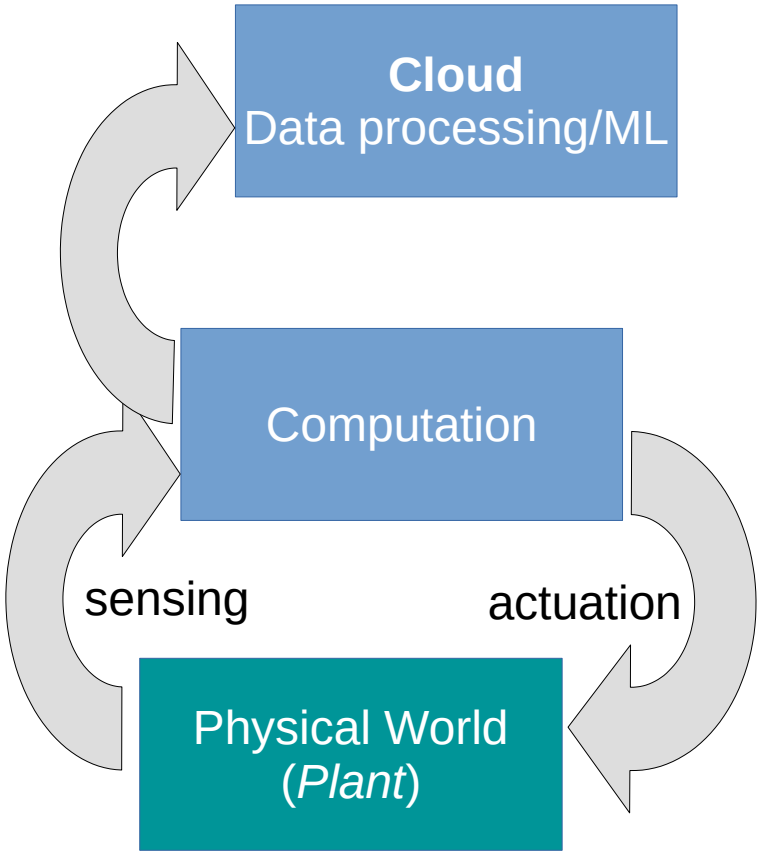


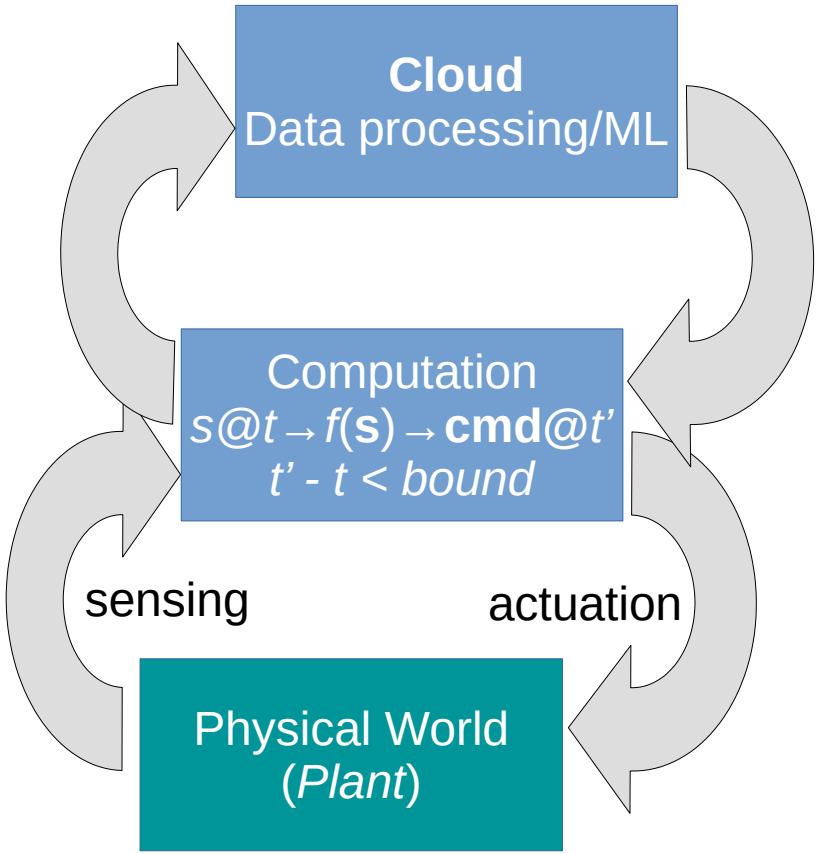
We don't write code thinking about time!

How do engineers program real-time systems???

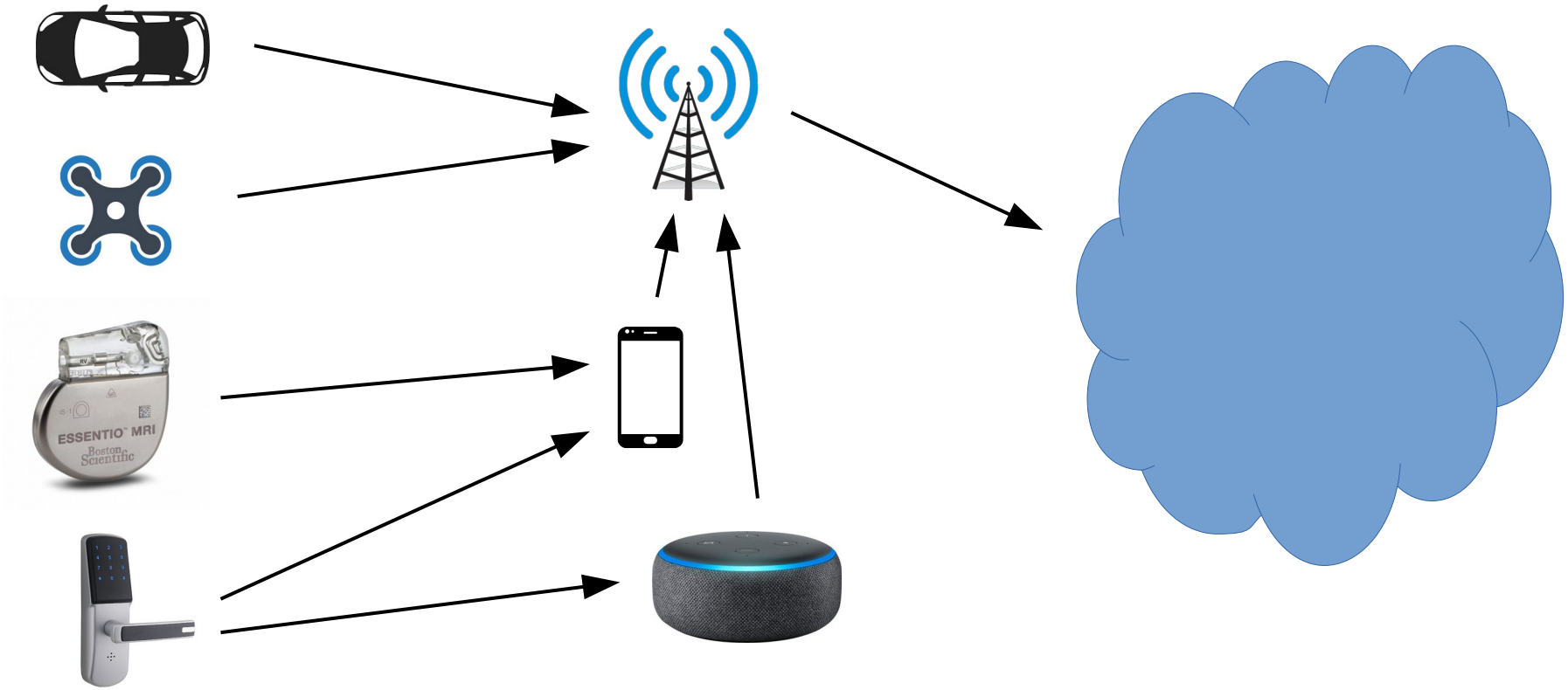
IoT Systems

Embedded/RT systems + **network**





Vectors for IoT/CPS Attacks?



Building High-Confidence Embedded Systems

- How can we **design** for time?
- How can we **design** to handle malicious attacks/failures?

Perspective on Liability

Who is to blame if it goes wrong?

- Computer Science
  - Users = Lusers
  - “Move fast and **break things**”

Perspective on Liability

Who is to blame if it goes wrong?

- Computer Science
  - Users = Lusers
- Mechanical/civil/bio:
  - You build it,
    you can be sued

Perspective on Liability

Who is to blame if it goes wrong?

- Computer Science
  - Autonomous Vehicles
  - Pervasive surveillance
  - City-level control

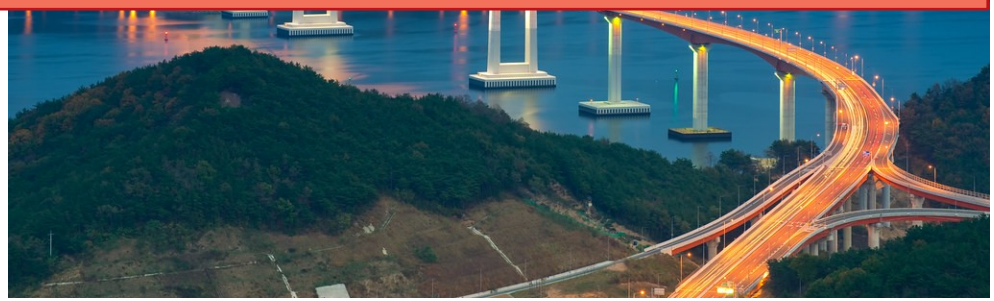
Now who's to blame? Regulation → Slower dev

Perspective on Liability

Who is to blame
goes wrong?

- Computer Sci
  - Autonomous
  - Pervasive surveillance
  - City-level control

Goal: Can we *understand* the risk of the system, to explicitly *analyze* it?



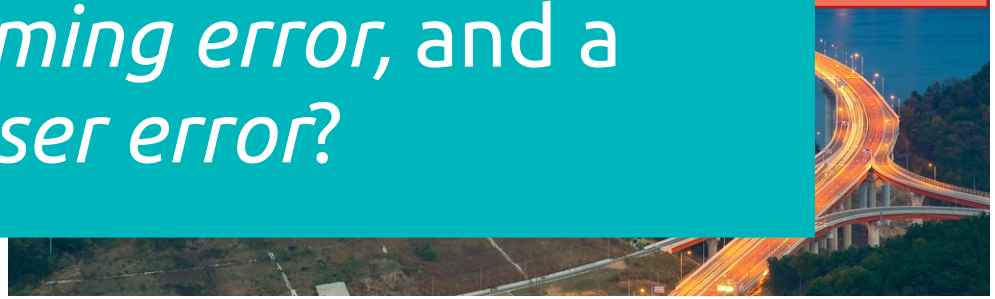
Perspective on Liability

Who is to blame
goes wrong

- Compute
  - Autono
  - Pervasi
  - City-lev

Goal: Can we *understand* the risk

Really hard: Can we differentiate between a *security breach*, a *programming error*, and a *user error*?



Perspective on Liability

Who is to blame
goes wrong

- Compute
  - A
  - P
  - C

Goal: Can we *understand* the risk

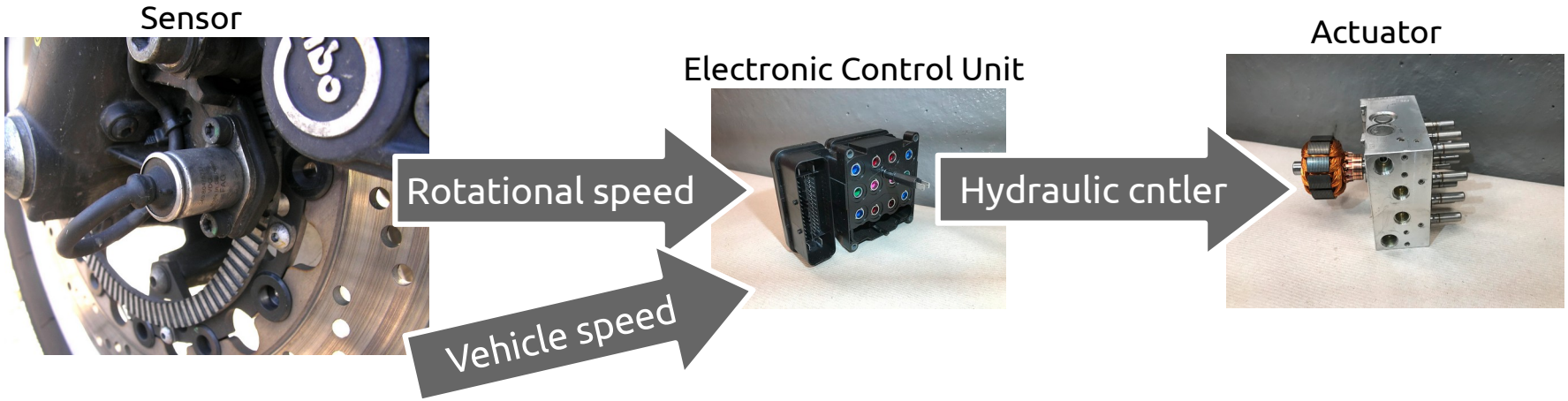
Really hard: Can we differentiate

, a

Really really hard: Who is *legally culpable* for the damages (see Toyota uncontrolled acceleration case)

“Real-time” = time-awareness

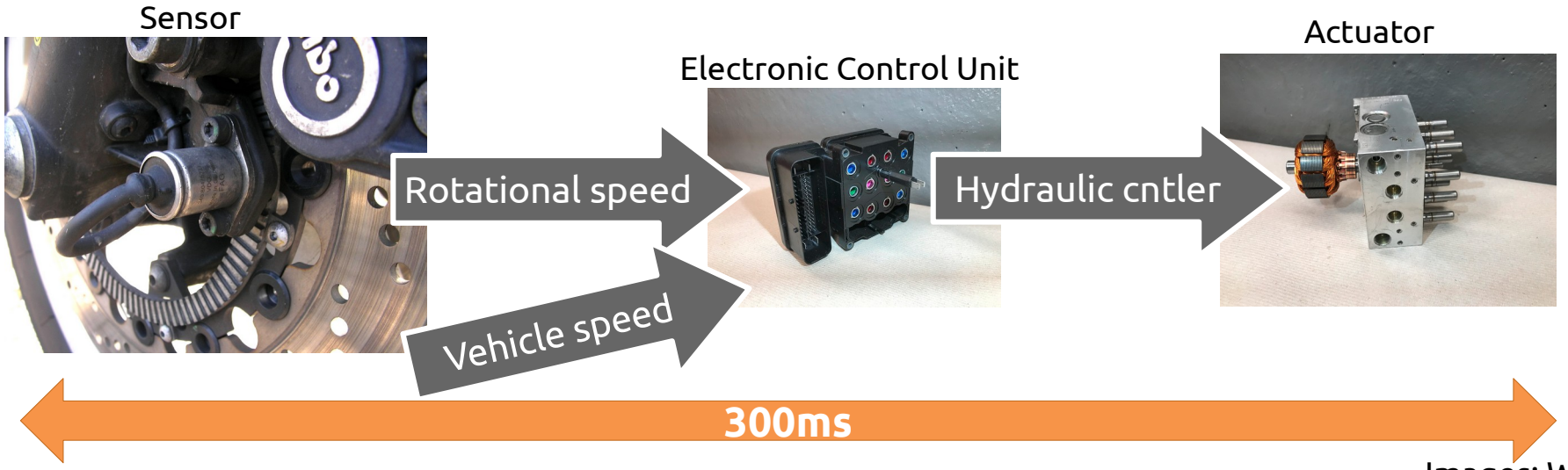
- Controlling physical world →
  Adhering to the world’s timing constraints
- Example: Anti-lock brakes



Images: Wikiped

“Real-time” = time-awareness

- Controlling physical world →
  Adhering to the world’s timing constraints
- Example: Anti-lock brakes



Images: Wikiped

“Real-time” = time-awareness

- Controlling physical world →
  Adhering to the world’s timing constraints
- Example: Anti-lock brakes

Images: Wikiped

RTSes: Deadline Examples

- Augmented Reality:
  - Human control → ~300ms (~150ms for *pros*)
  - human perception → ~10ms
- Robotics and AVs:
  - *1ms* a robot needs to regulate force to the environment
  - *<4ms* walking robot applications
  - *<10ms* manipulating robots
  - <100ms driving robots
  - *10-100ms* Vehicles: ATC, ABS

Real-Time Scheduling Goals

Goal: All tasks have **responsiveness ≤ deadline**

Assumptions:

1. All threads execute **recurring code** (p).
2. We know apriori **how much computation** they do each activation (WCET).

Real-Time Scheduling Goals

```
while(1) {
   // wakeup at next p
   block_till_activation();
   s = read_sensor();
   o = computation(s);
   update_actuator(o); // by deadline !
}
```

deadline

1. All threads execute **recurring code** (p).
2. We know apriori **how much computation** they do each activation (WCET).

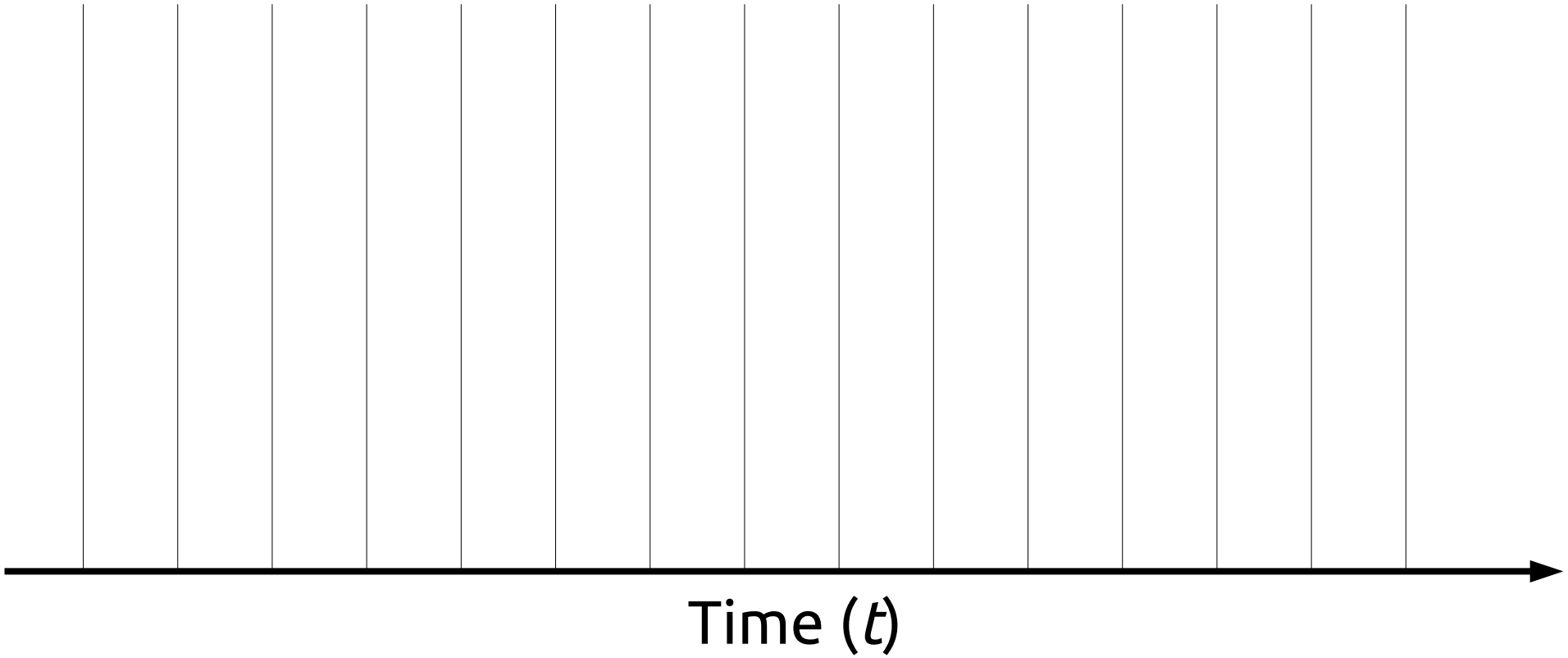
Periodic Task Model

Task (τ_i)– abstraction of computation, each has a

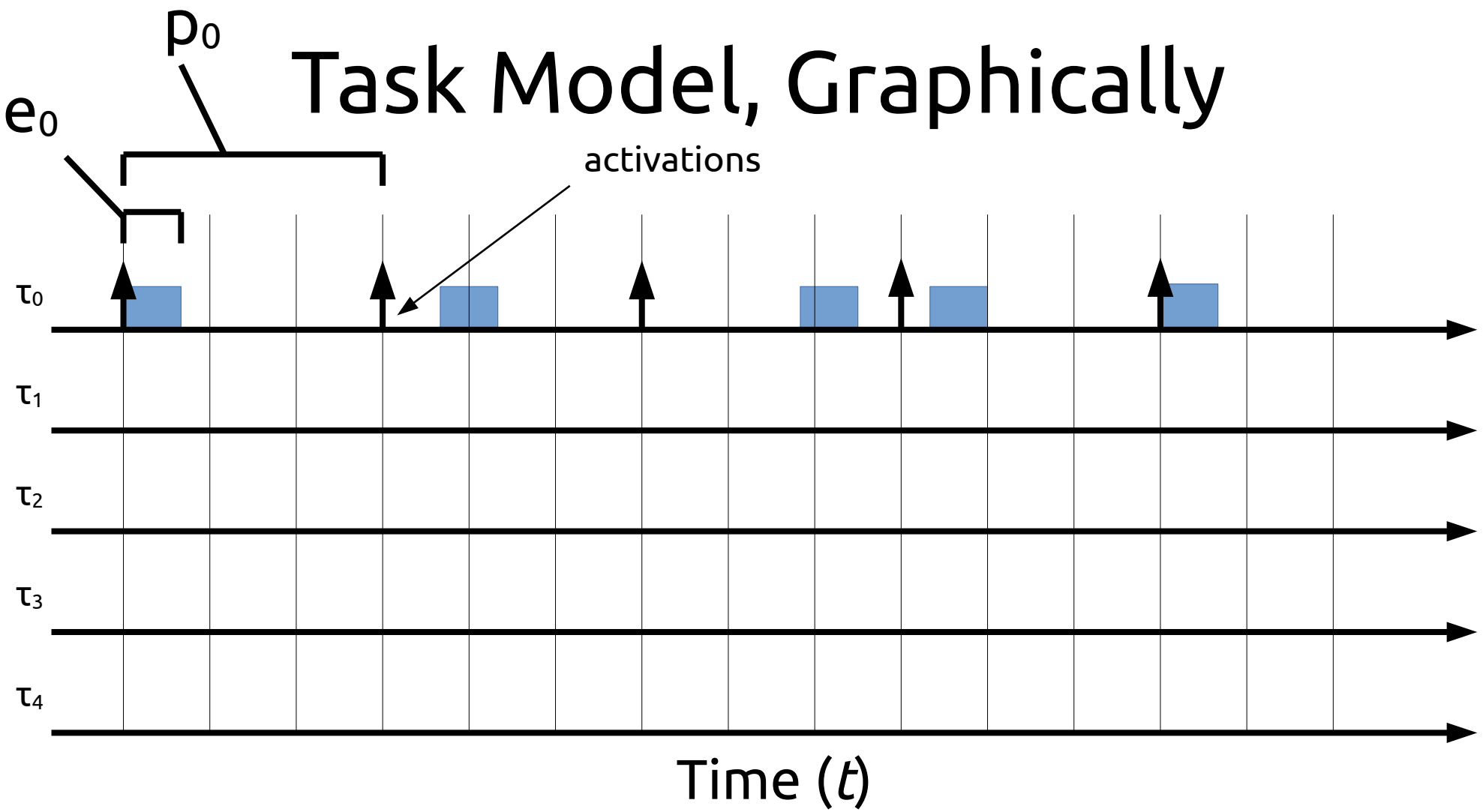
- **Period** (p_i) – period, span of repetitive exec.
  A task *activates* (wakes up) every p_i time units
- **WCET** (e_i) – worst-case execution time
  Every activation, task τ_i executes for max e_i
- **Deadline** of τ_i is p_i after its activation

System is $T = \{\tau_0, \tau_1, \ldots, \tau_N\}$

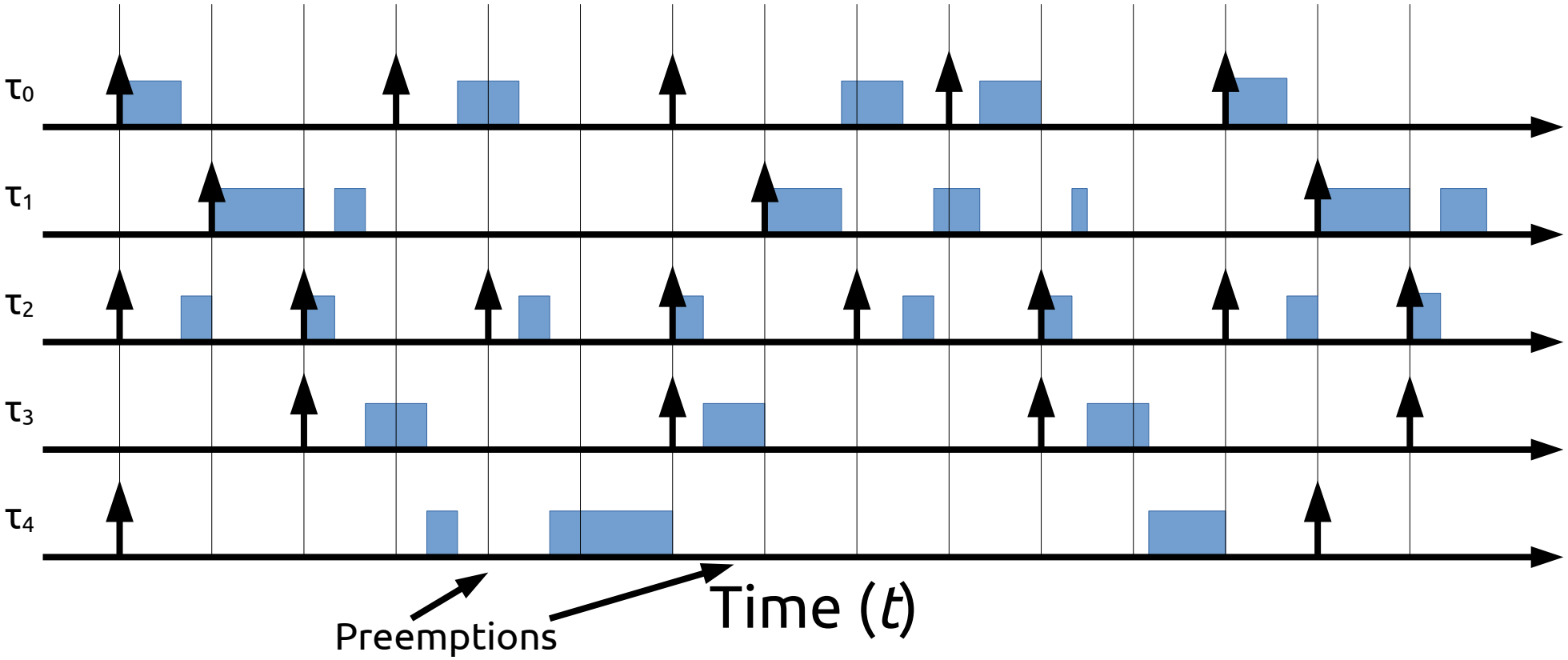
Task Model, Graphically

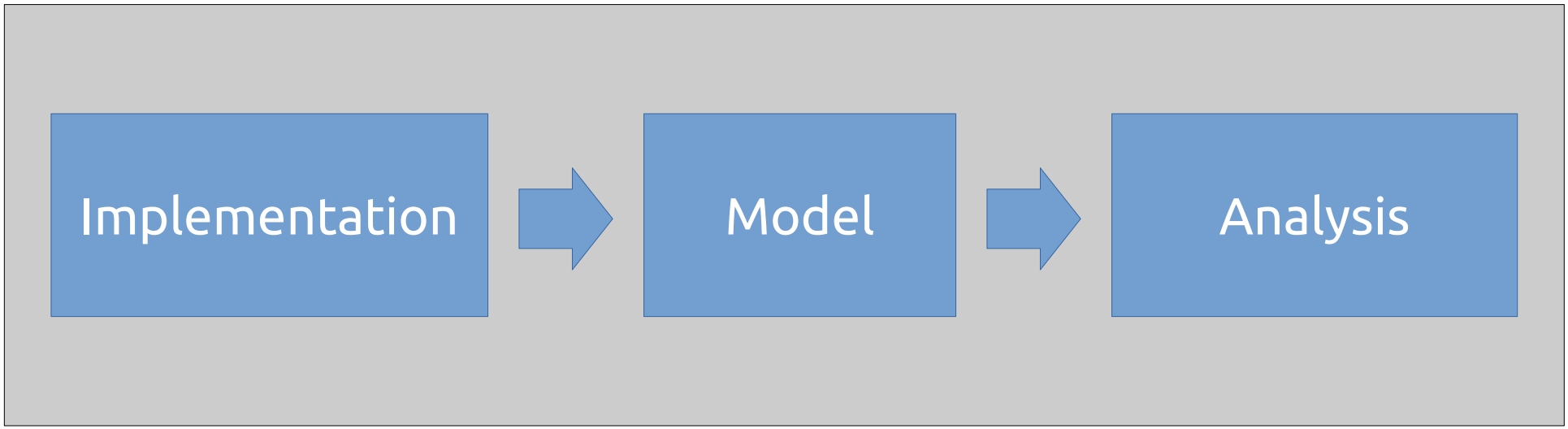


Task Model, Graphically



Task Model, Graphically





- *Goal*: real-time systems
  - **Implementation** → policies to instantiate a model
  - **Model** → mathematical description of impl.
  - **Analysis** → derive properties from model

Security/Confidence

- Network attached computers are vulnerable
  - No network → revert to ‘90s
  - **Plan for faults/compromises**

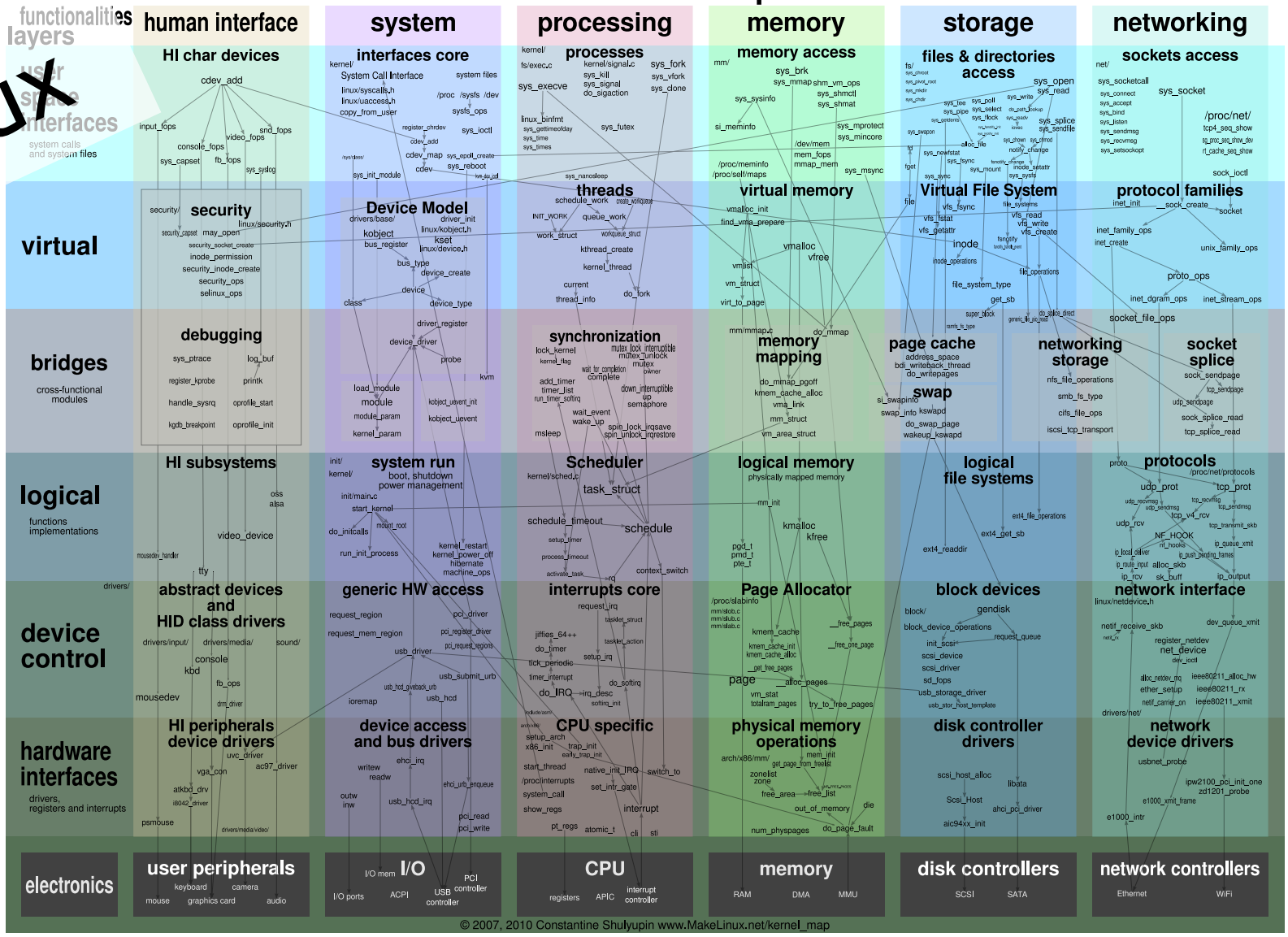
Security/Confidence

- Network attached computers are vulnerable
  - No network → revert to ‘90s
  - **Plan for faults/compromises**
- How?
  - Isolation → limit **scope** of effects of **faults**
  - Controlled degradation → faults lead to **lower functionality**, not **system failure**

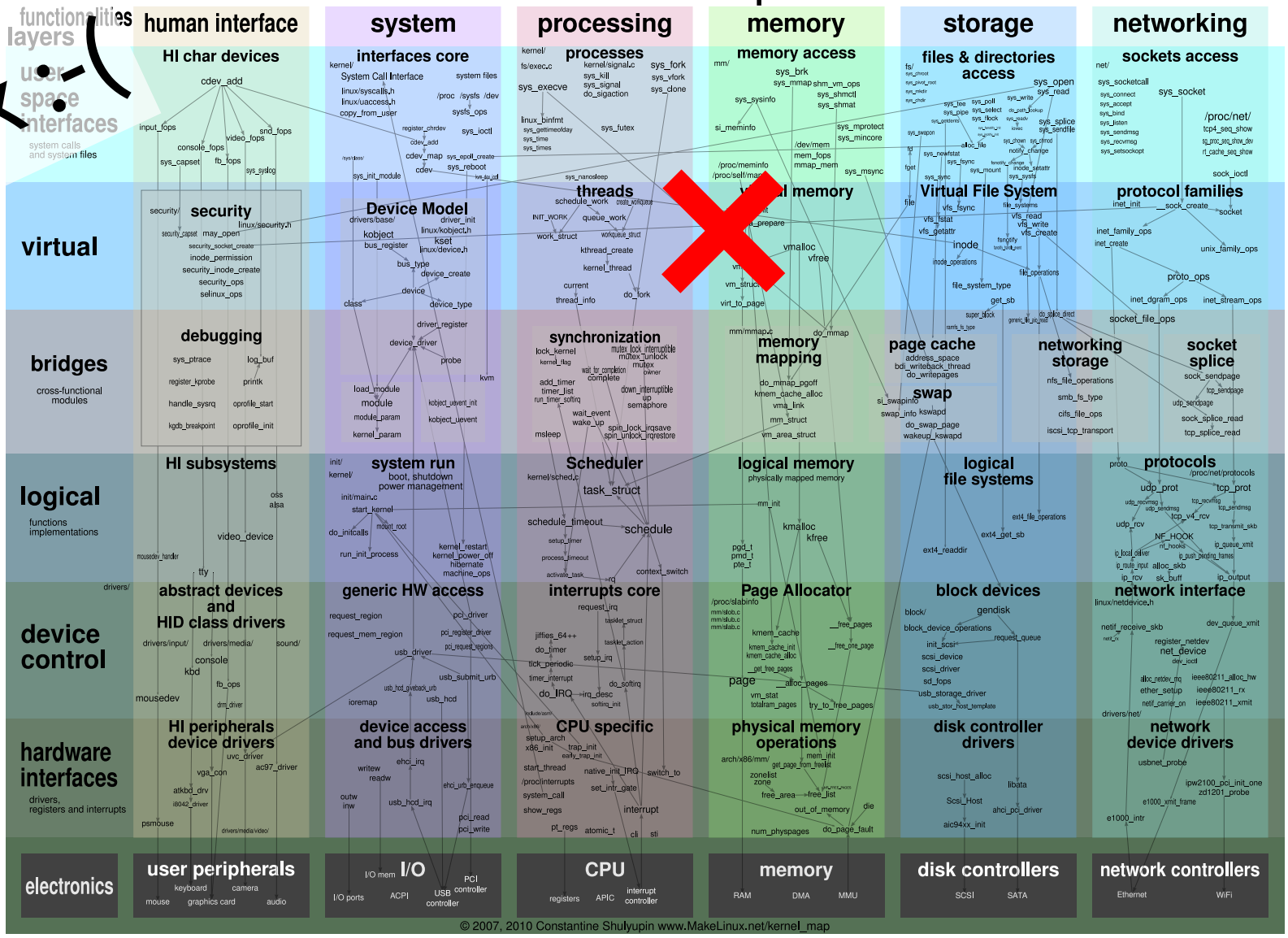
Operating Systems FTW

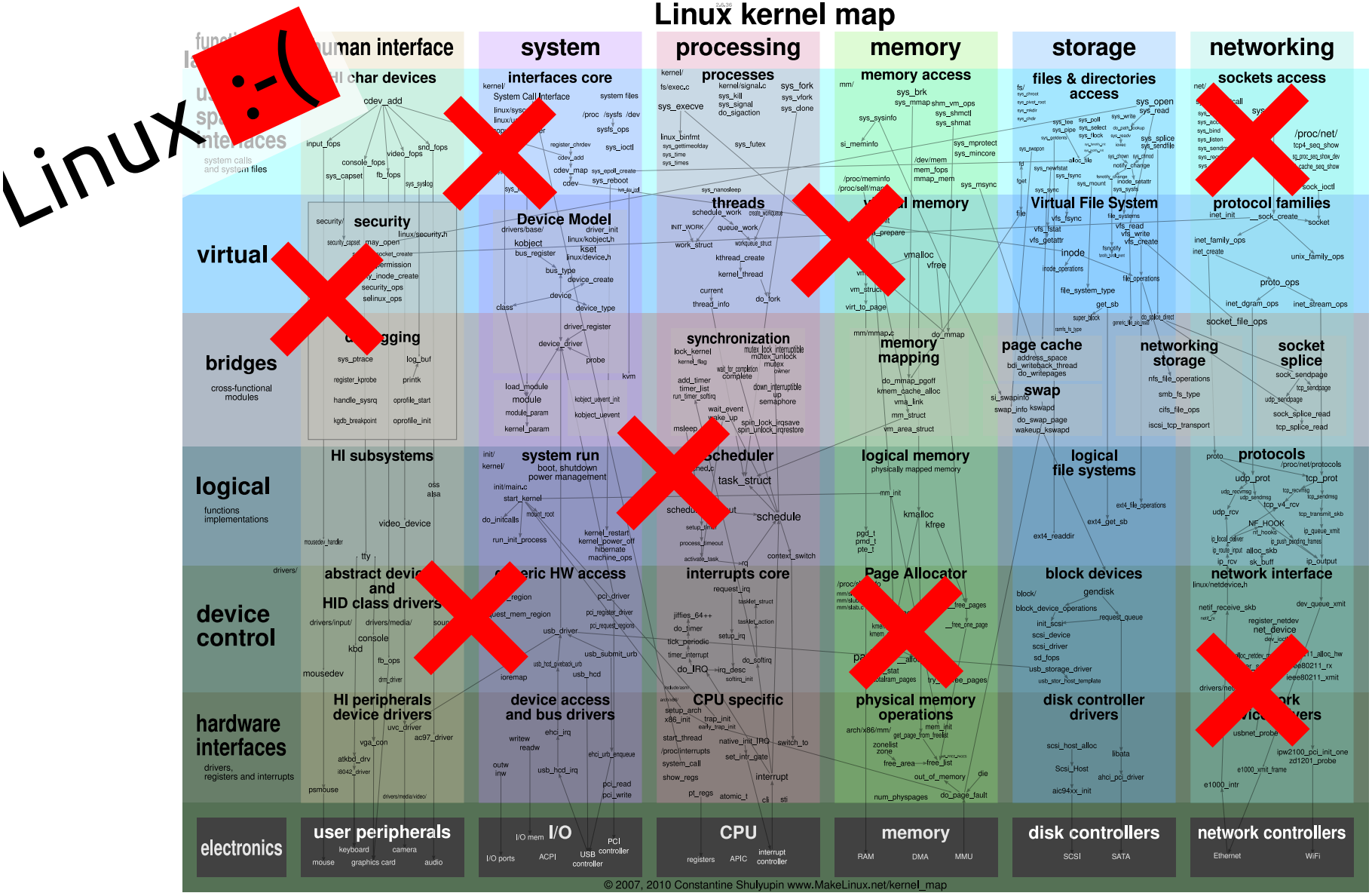
- Isolation in Linux
  - Processes can independently fail
  - No process can steal all CPU time

Linux kernel map



Linux kernel map

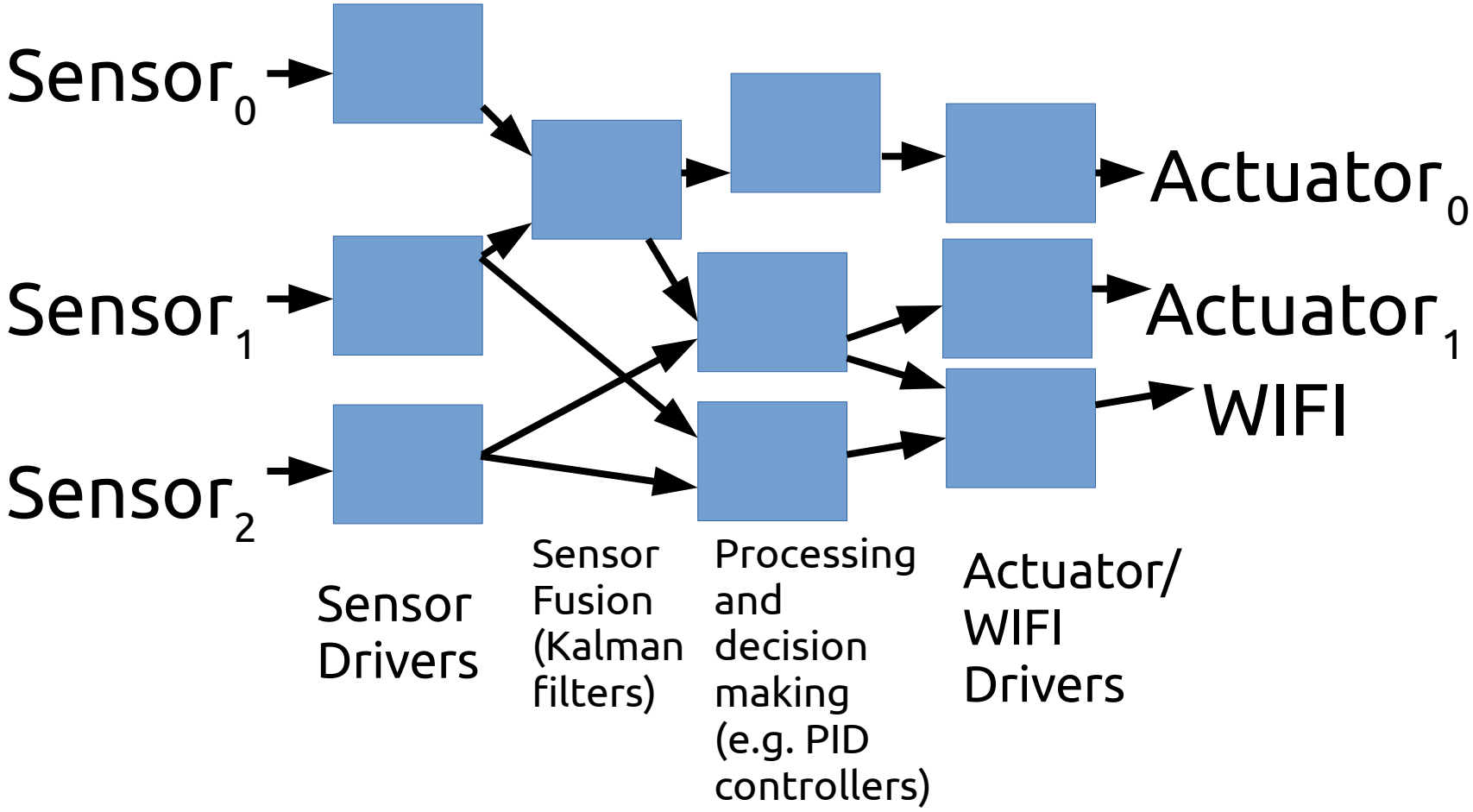




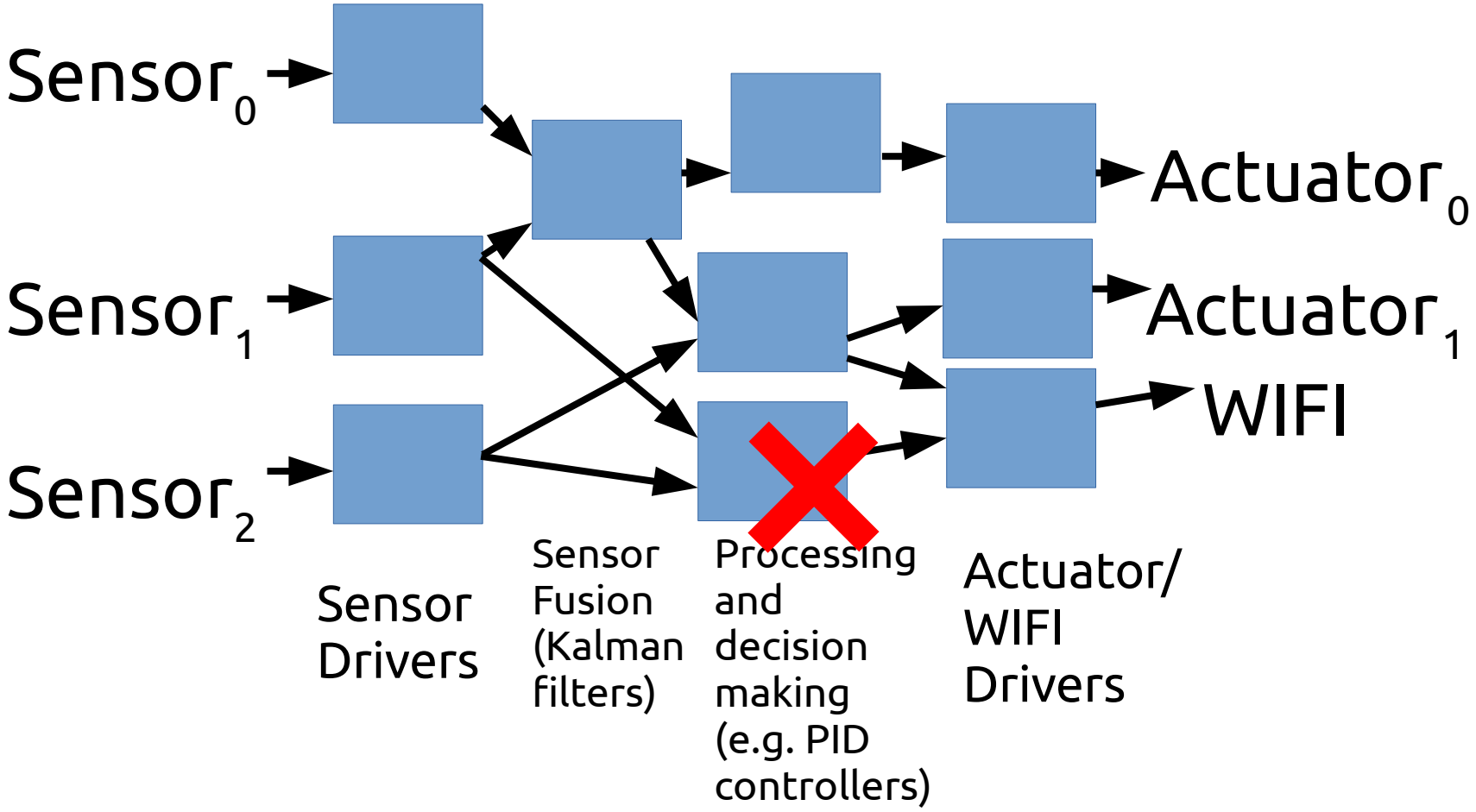
Secure Operating Systems

- **Break up** system software →
  minimize impact and scope of faults
- **The "core"** that provides isolation →
  must be **small/trustworthy**

Secure Operating Systems



Secure Operating Systems :-)



High-Confidence Embedded Sys

- Real-time processing
- Isolation as a primary design concern