Computation

6.900 EFI

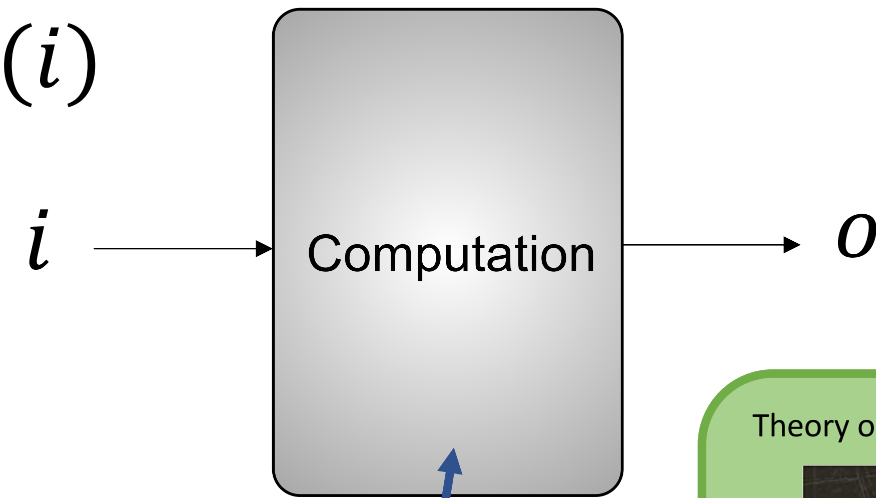
Spring 2023

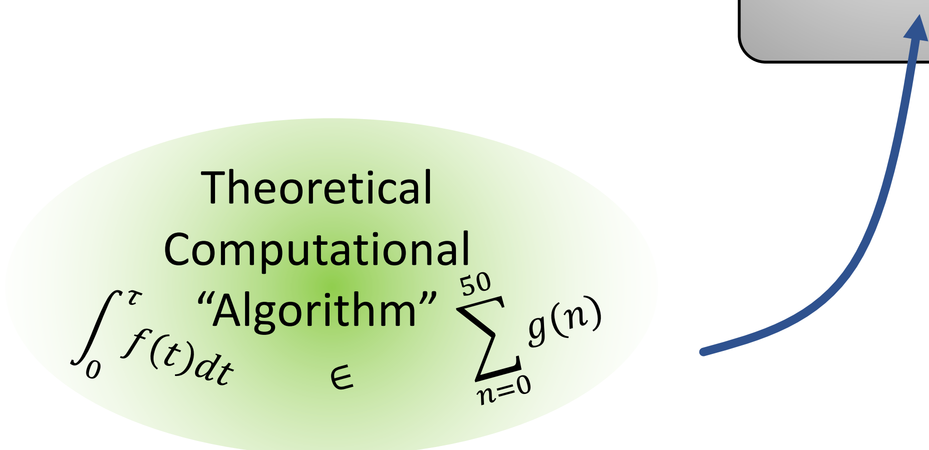
Flow of Information

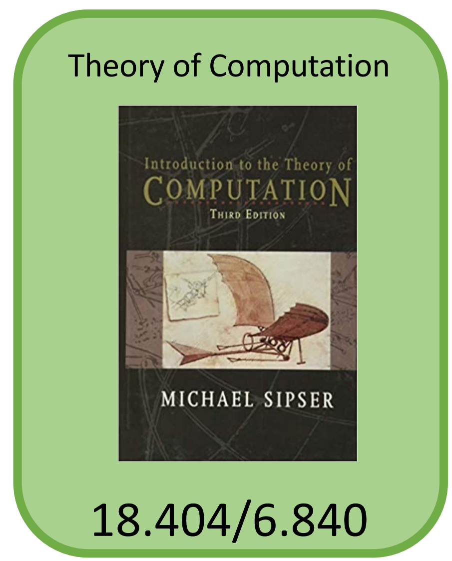
- All computation is really comprised of functions that produce **outputs** given **inputs**

o = outputs
i = inputs
f = your logic

$$o = f(i)$$







Computation

- Is an abstract concept
- But it can be implemented:
  - Mechanically (for example, gears)
  - Chemically (for example, DNA computer, cells,)
  - In Quantum realm (for example qbits)
  - Electrically (What we do now)

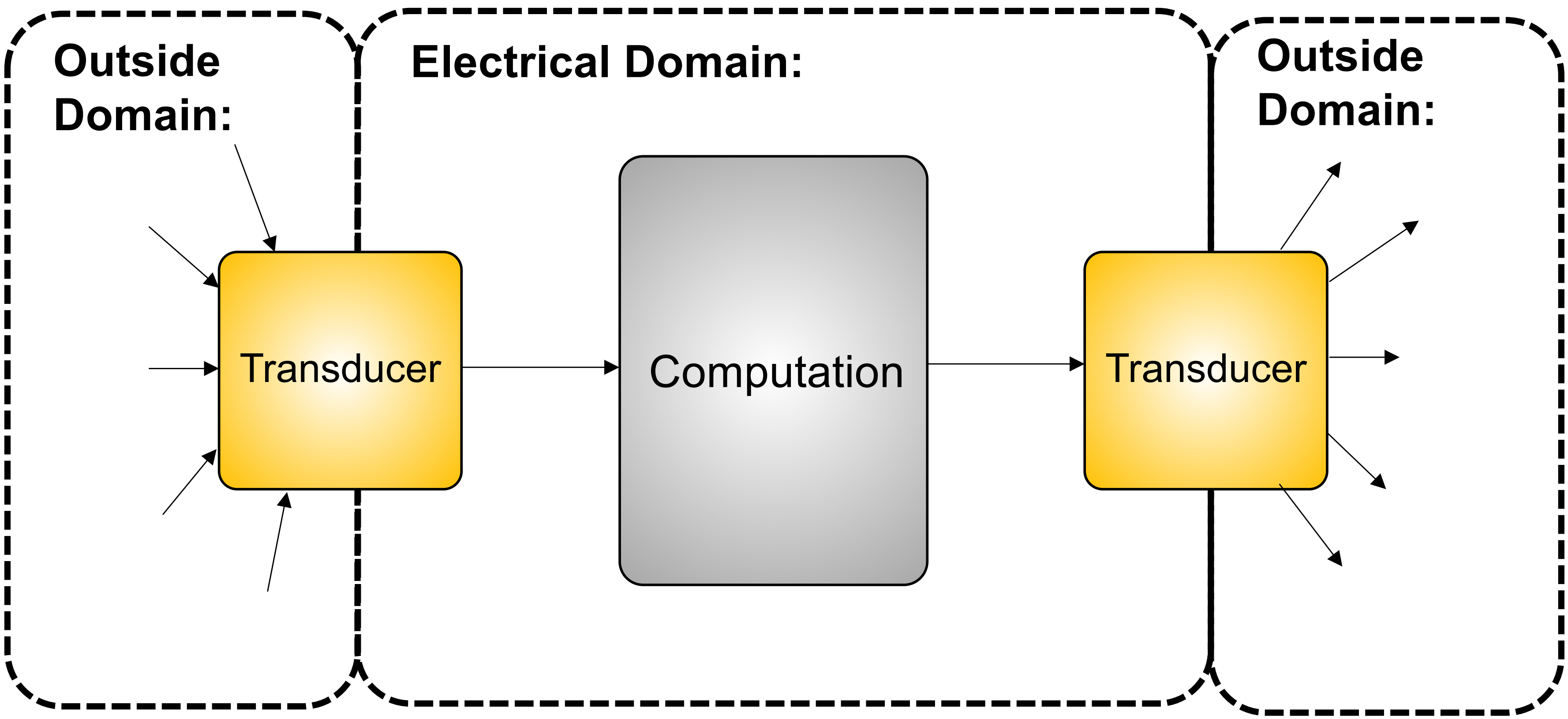
Electrical computation has proven extremely scalable, allowing an exponential rise in the amount of computation that can be deployed, in charge of much of the 20th and 21st century information revolutions

The Big Idea

- Computation takes place in the electrical domain because it is the easiest to scale:
- We design circuits to manipulate electrical energy to give us computation:
  - Circuits can be hardwired to perform a particular computation
  - Or can be generalized to be a “computer”
- Computers are circuits that are designed to be flexible in the logic they implement.
- We change that logic with programming languages

The Big Idea

- Most modern computational devices take on this form*:



*There's always exceptions, so this isn't a 100% true statement

Transducer

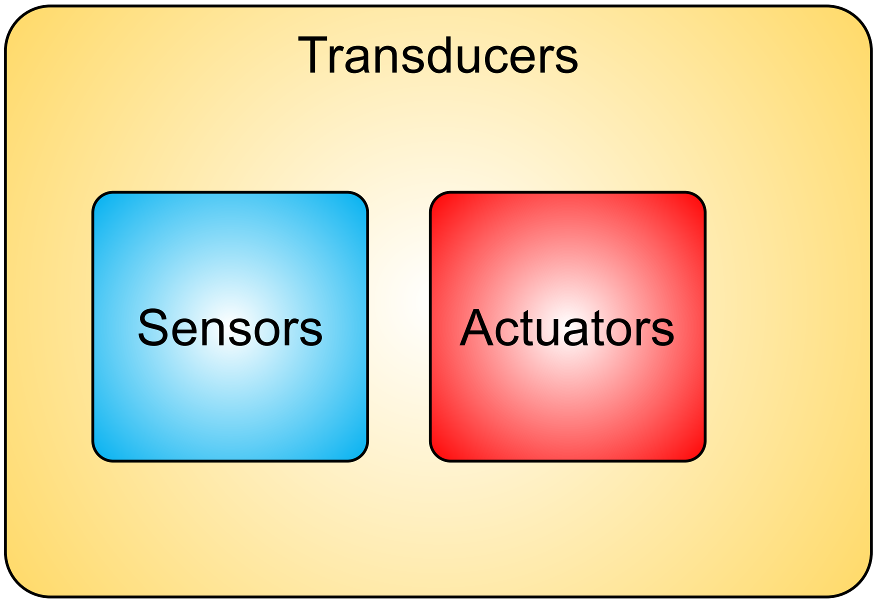
- Any device which converts energy/information from one form into another
  - IMU*: acceleration/velocity → electrical
  - Pressure sensor: pressure → electrical
  - Microphone: pressure waves → electrical
  - Buttons/Switches: mechanical force → electrical
  - LED: electrical → Light
  - Buzzer: electrical →pressure waves
  - LCD: electrical → Light/position

6.200, 6.012,
Prof. Farnaz Niroui's class

*Inertial Measurement Unit

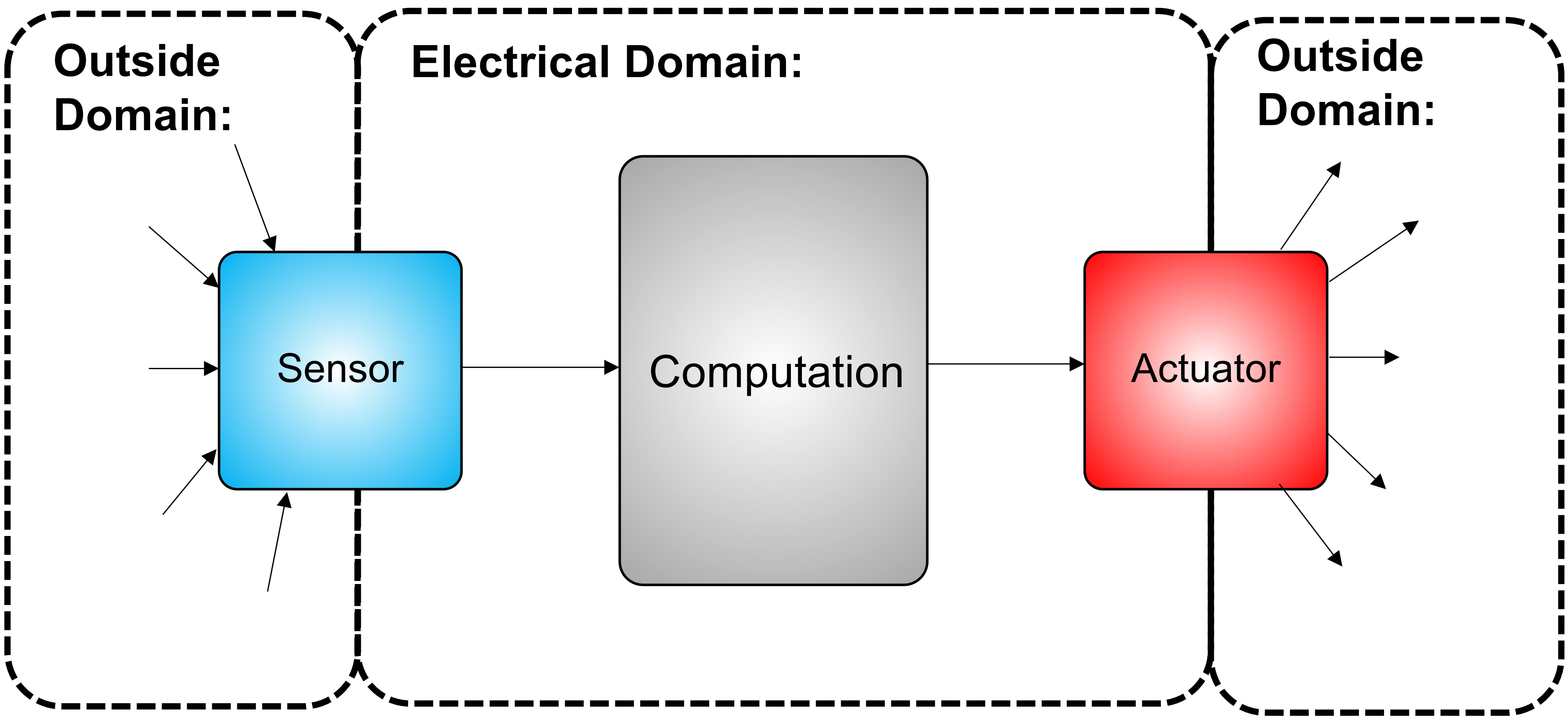
Break Transducers into Two Classes

- **Sensors:** Convert outside energy/info into electrical form
- **Actuators:** Convert electrical energy/info into outside form



Update our Big Idea

- Most modern computational devices take on this form*:



*There's always exceptions, so this isn't a 100% true statement

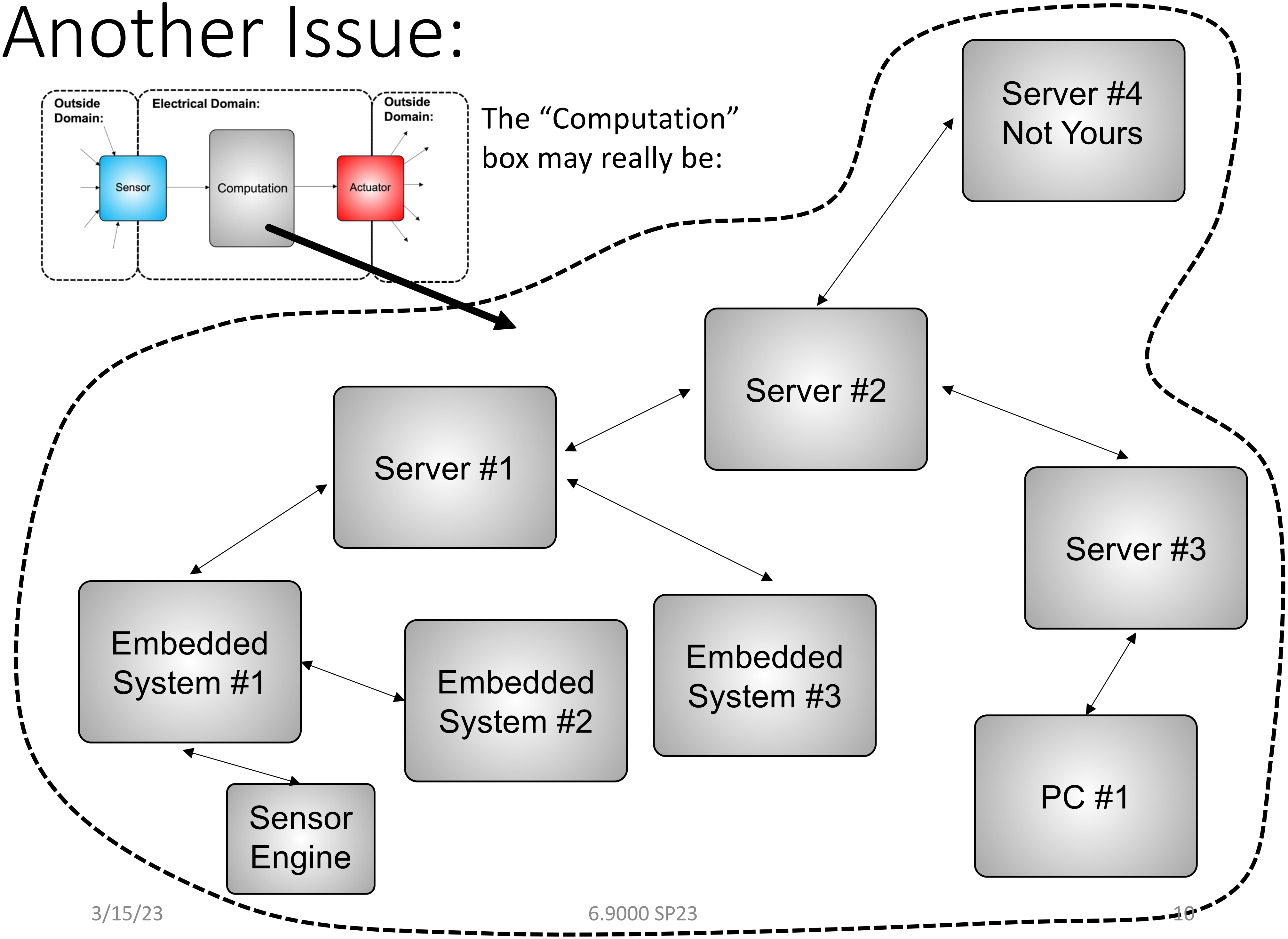
Update our Big Idea

- While the general pattern has remained constant since the 1900's, the scope/breadth of the inputs and outputs has scaled tremendously…

- In the 1940's, a computation block would take in several hand-entered numbers and maybe solve some third/fourth order differential equation and return the coefficients as a result

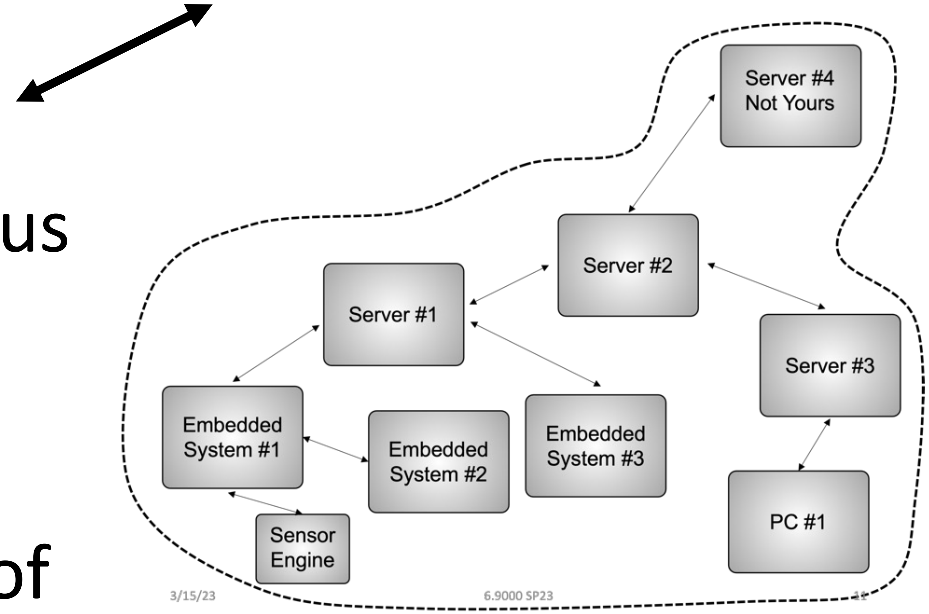
- In the 2020's, a computation block may take in Gbps of video, audio, environmental, meta data and control entire fleets of drones and direct vehicles and keep you entertained/engaged with random stuff

Another Issue:



Even Deeper

- The “arrow”s on the previous page represent communication channels, many of which have their own computation in them of varying complexities!



- Conveying the push/unpush of a button to a nearby embedded system needs maybe two wires that can be loopy messes…maybe bandwidth of 20bits per second (bps)
- Conveying 24 bit color 1080p video at 60fps is 2.98Gbps requires a little more than two wires

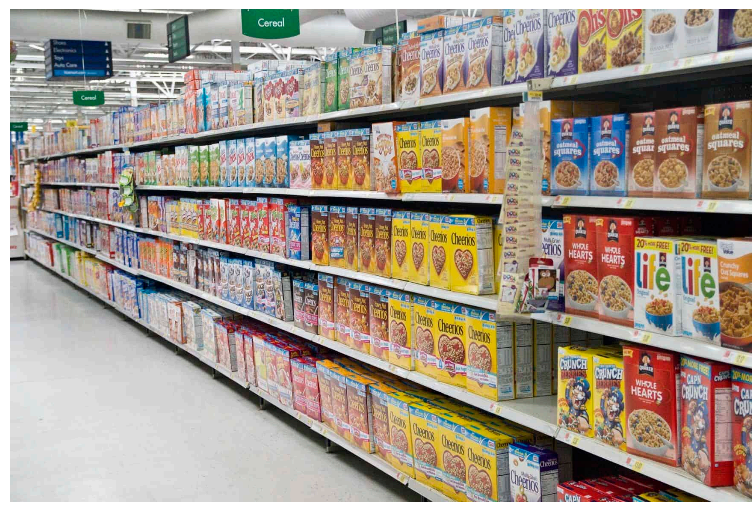
The “Computation”

- Computation is still developed and deployed…but we no longer deploy it on one machine
- Dozens up to millions of separate devices, all capable of computation and all capable of sharing data with one another are now the canvas
- As an EECS person, it is a lot to take in

Used to Be:

“I guess it’ll be Cheerios.”

How it is Now:

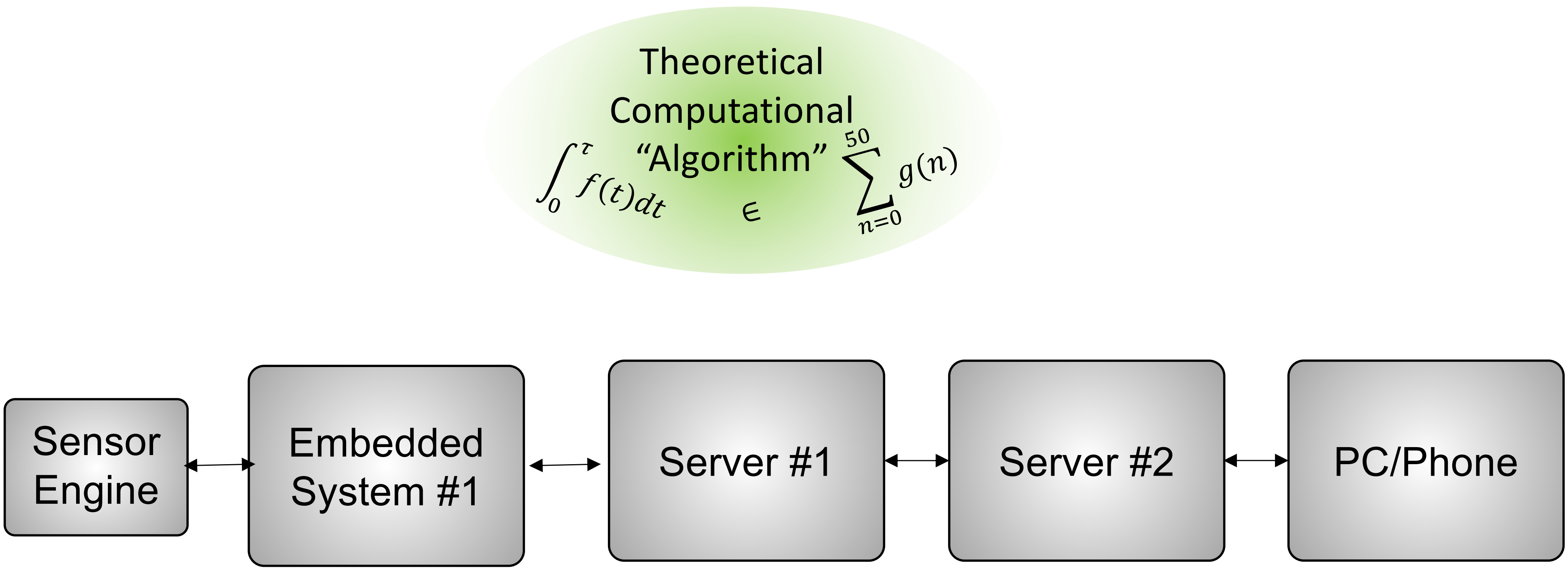


“I’m having an existential crisis. What does it all mean?”


100 SECONDS OF
2:34
React for the Haters in 100 Seconds
1M views • 10 months ago
Fireship
React is the most popular JS framework ever, but some
4K
Introduction to Embedded Linux
Part 1 - Buildroot
Part 1 - Buildroot | Digi-Key...
DON'T WASTE YOUR MONEY!
I Can Save You Money! – Raspberry Pi Alternatives
2M views • 1 month ago
PHP isn't used by REAL websites
9.3K views • 4 years ago
John Morris
I've been getting these comments more and more lately: "...le
WIKIPEDIA
The Free Encyclopedia
slack
8:23
11:07

Distributing Your Algorithm Over the System

- You have a computational task that must be done. How do you distribute it among the chain of computation that exists within your system?



Choices, Choices, Choices

- Where you choose to do your compute impacts:
  - What hardware
  - What communication protocols
  - Costs (startup, runtime)
  - Security
  - Robustness/Reliability/Flexibility
  - Energy Usage
  - Where/How to deploy (geographically)

Truly Socio-technical Systems

- The physical and conceptual range of scale in modern EECS systems is ridiculous:
  - Concerned with scales from nm up to 10,000 km
  - Concerned with data from bits to terabytes
- The system is embedded in society and it is linked to all of us.
  - Humans now carry phones around 24/7 of which we are constantly aware and monitoring.
  - It isn't exactly the Matrix™ but it isn't too far off

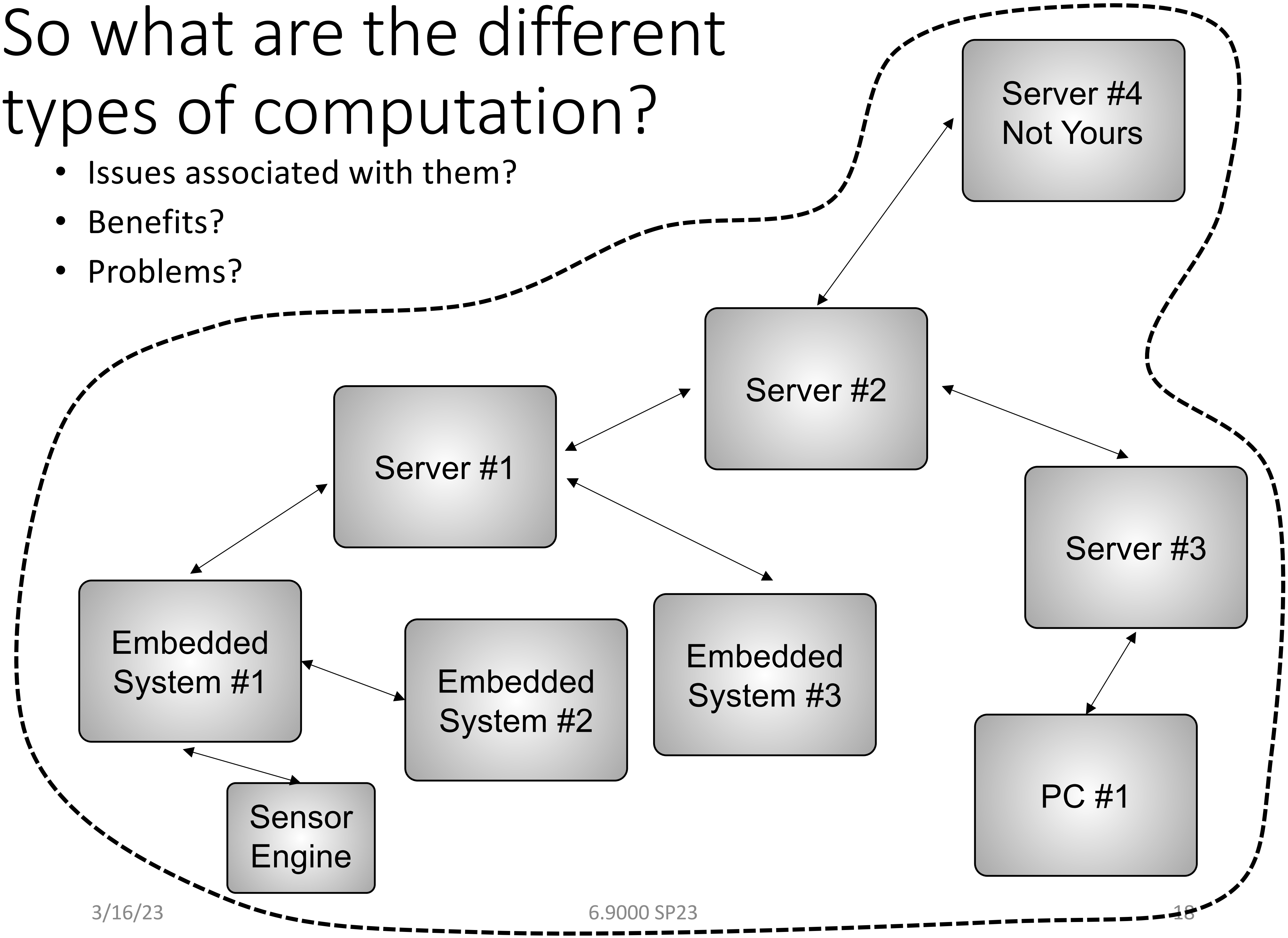
Reminds me of a Poem

No device is an island,
Entire of itself.
Each is a piece of the continent,
A part of the main.
If a clod be washed away by the sea,
Europe is the less.
As well as if a promontory were.
As well as if a manor of thine own
Or of thine friend's were.
Each device's failure diminishes me,
For I am involved in the ***Distributed, Interconnected Embedded Systems/IOT****.*
Therefore, send not to know
For whom the bell tolls,
It tolls for thee.

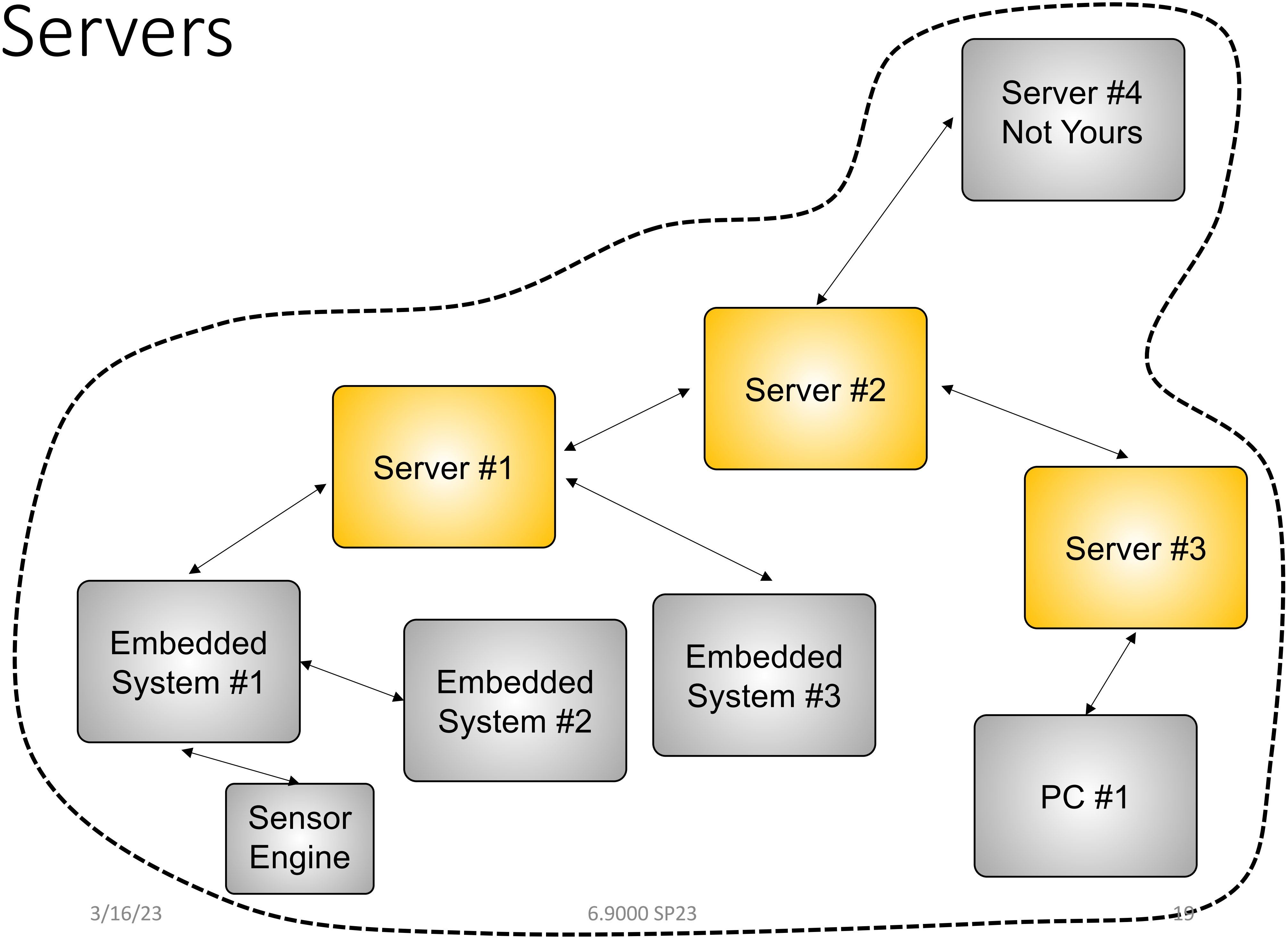
-John Donne (1572 - 1631)

So what are the different types of computation?

- Issues associated with them?
- Benefits?
- Problems?



Servers

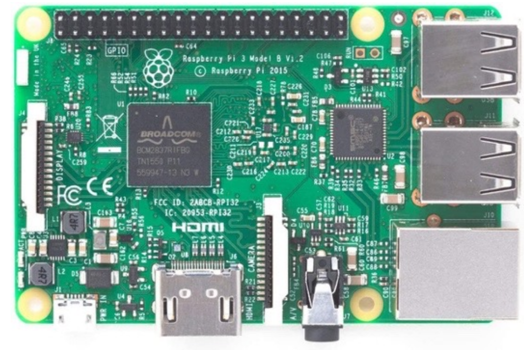

Server #4
Not Yours
Server #2
Server #1
Server #3
Embedded
System #1
Embedded
System #2
Embedded
System #3
PC #1
Sensor
Engine

Server?

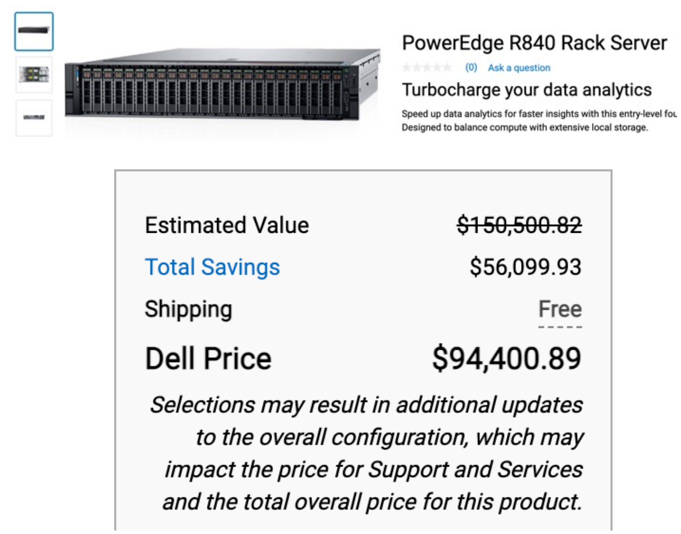
- Generally a standalone computer running a full operating system in an always-on mode
- Name comes from it “serving” files or resources
- Usually a fixed-in-position piece of computation
- Priority placed on:
  - up-time/reliability
  - Redundancy
  - Threading, parallelization where possible
  - Raw compute power (GPUs, specialty hardware/cards)
- Less priority placed on:
  - Power consumption (to a point)
  - Cost (to a point)
  - Size (to a point)

Any “Computer” Can Be a Server

- You’re using $35* Raspberry Pi’s in our class



- You can easily spend >$100K on a rack server, though
  - These can be specialized with:
    - Redundant power supplies
    - Huge numbers of cores/threads/memory
    - Expensive GPUs or other accelerators
  - Depends on the purpose of the Server



*now $200 because of supply chain issues

Use Servers to…

- Act as long-term, large-scale data storage for system (file management or databases)
- Act as relay/interconnect for many client devices
- Provide "callable" access to proprietary/expensive/power-consuming processing

- Often form the backbone/core of your modern distributed system

In Real-Life Do You…

- Buy server(s)? If so have to:
  - Pay for them
  - House them
  - Feed them (with power)
  - Maintain them (24/7)…pay for staff
  - Deal with ISPs directly
- Rent server(s)? If so:
  - Pay a premium (but depending on scale might save money)
  - Don’t have to house, feed, or maintain
  - Can complain to somebody when they break, up to a point
  - You lose autonomy

State of the Field

- Server space can be rented at varying levels of support
- Some companies offer raw servers (sort of like what we did)
- Some companies offer in-depth ecosystems

And Even Then…

- Do you rent real machines and get full claim to their hardware? (more expensive)
- Do you rent “virtual machines” which generally will not run as fast and aren’t the best for performance, but will be cheaper and might be fine.
- You can also rent “elastic compute” now
  - the resources allocated to your server(s) can be increased/decreased over time as needed
  - Billed to the second
- So lots of choices

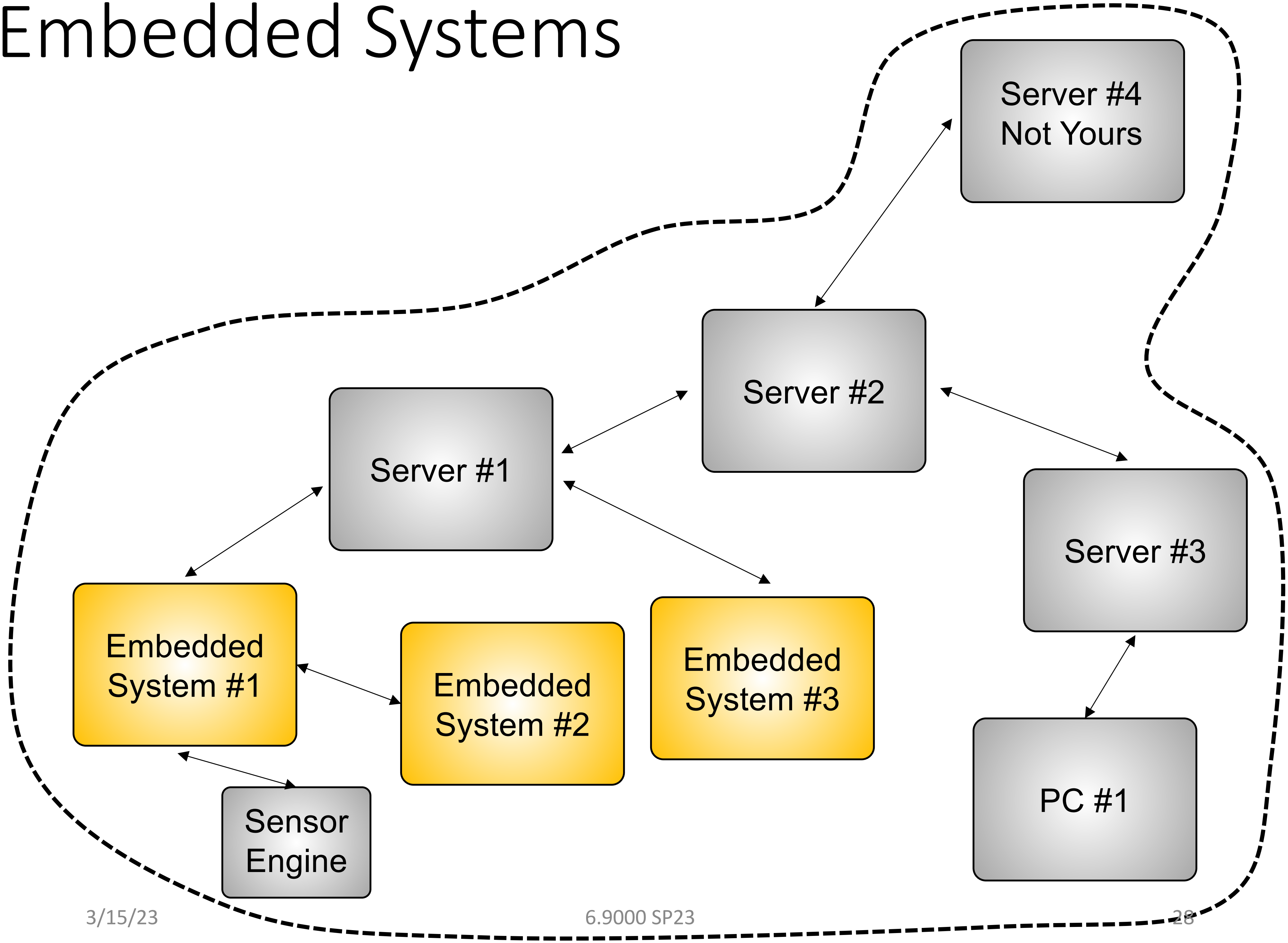
Server Languages?

- Massive Variety and Options when it comes to “server” programming
- While the highest performance is still largely accomplished with C/shell scripting…
- Most favorite languages have server frameworks:
    - Python (Django, Flask, FastAPI)
    - Javascript (NodeJS…React Angular, Vue)
    - Rust (Rocket)
    - Golang (gin)
    - PHP
    - Ruby (on Rails)
    - Java

Servers are Always Around

- Restarts and updates are relatively easy* to implement on a server
- Servers are generally easier to work on since they are by their nature remotely accessible
- Their central nature makes them very delicate.
  - A piece of edge computing (embedded sensor) going down affects only that sensor/area
  - A server going down takes the whole system out.

Embedded Systems

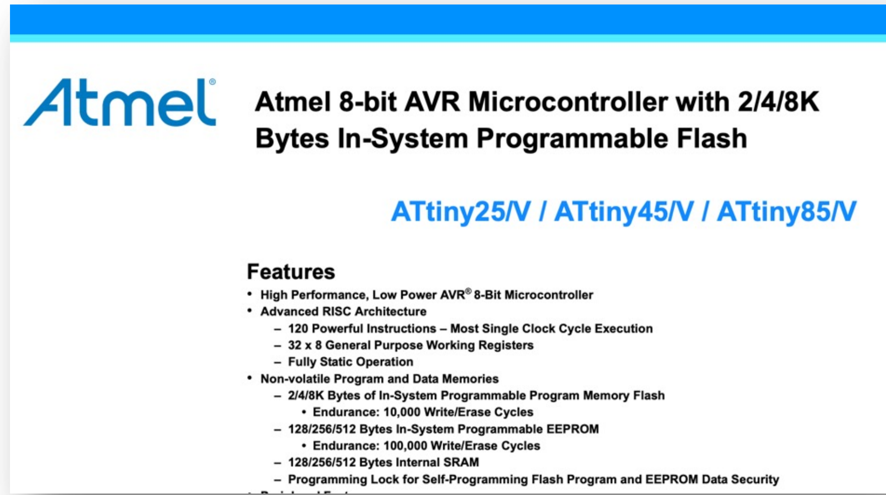


Embedded System

- Generally cheap and small enough to be embedded "in the field"
- Usually a "smaller" computational device, though that is a historically relative term:
  - 1980/90s: "embedded" meant 2MHz 8 bit microcontroller with 2K RAM
  - In 2023, "embedded" can mean full computer running an operating system with 16 GB of RAM

Wide Range of Capabilities

- From Very Cheap and very simple…
- Atmel and other companies have pursued a strategy of making very small, very simple (8 bit), very cheap microcontrollers
- Low power consumption, but not very computationally powerful



Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash

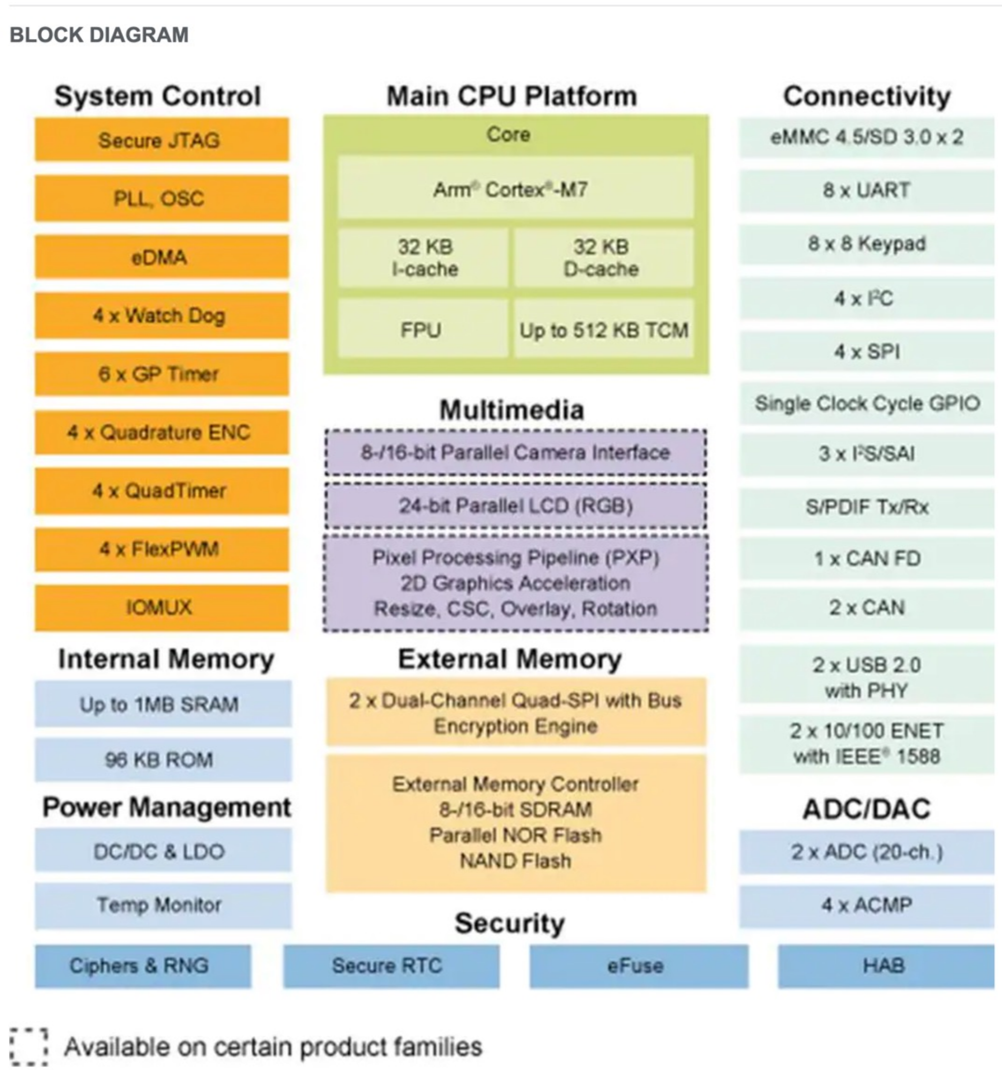
ATtiny25/V / ATtiny45/V / ATtiny85/V

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 120 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
- Non-volatile Program and Data Memories
  - 2/4/8K Bytes of In-System Programmable Program Memory Flash
    - Endurance: 10,000 Write/Erase Cycles
  - 128/256/512 Bytes In-System Programmable EEPROM
    - Endurance: 100,000 Write/Erase Cycles
  - 128/256/512 Bytes Internal SRAM
  - Programming Lock for Self-Programming Flash Program and EEPROM Data Security

Wide Range of Capabilities

- To VERY Capable…
- NXP has the i.MX RT1060 microcontroller, considered fastest in the world right now:
    - 600 MHz
    - All this stuff →
- Obviously more expensive but far far more performant

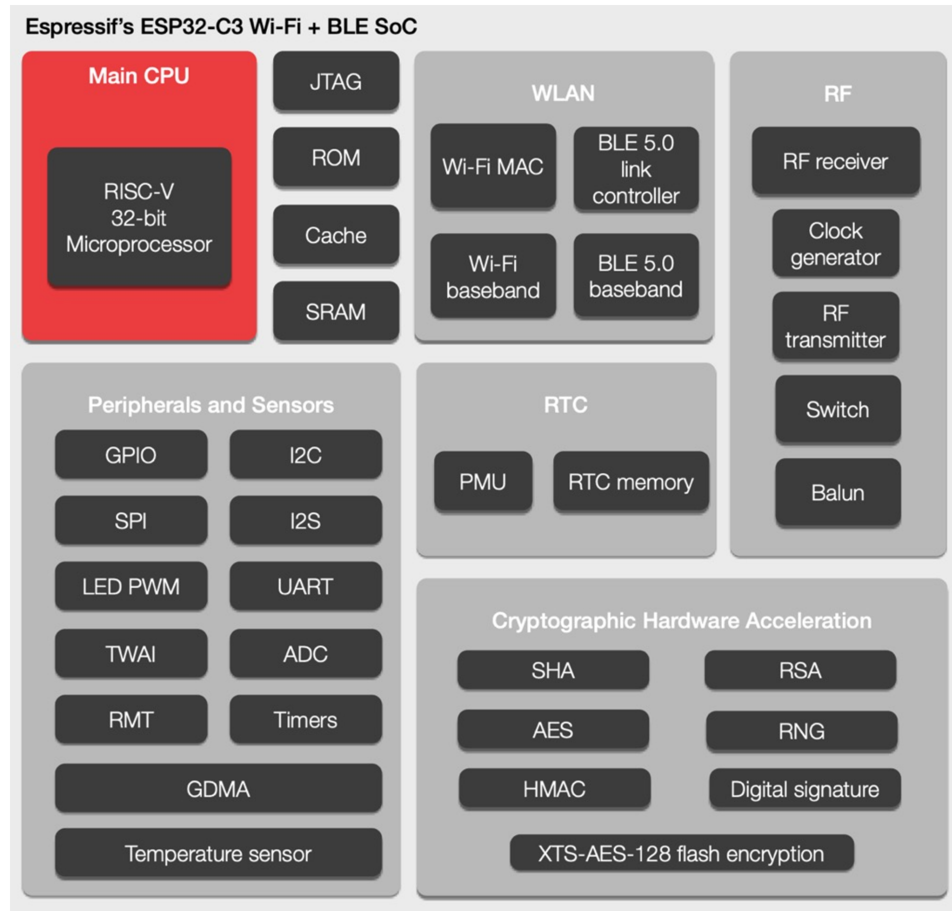


General Trend in Embedded Field

- Power consumption is a huge driving factor in the embedded field, in a far different way than with servers
- General move towards more computationally powerful platforms in conjunction with deep sleep modes of operation
- It is often better to be full-on hardcore for a short period of time and then deep sleeping than running all the time at a slower clock

System on Chips are also Becoming Popular

- Our ESP32-C3 by Espressif isn't just a programmable core with memory
- Also has additional systems on board for:
  - RF management
  - WiFi
  - Cryptographic tasks
  - All in one single piece of silicon

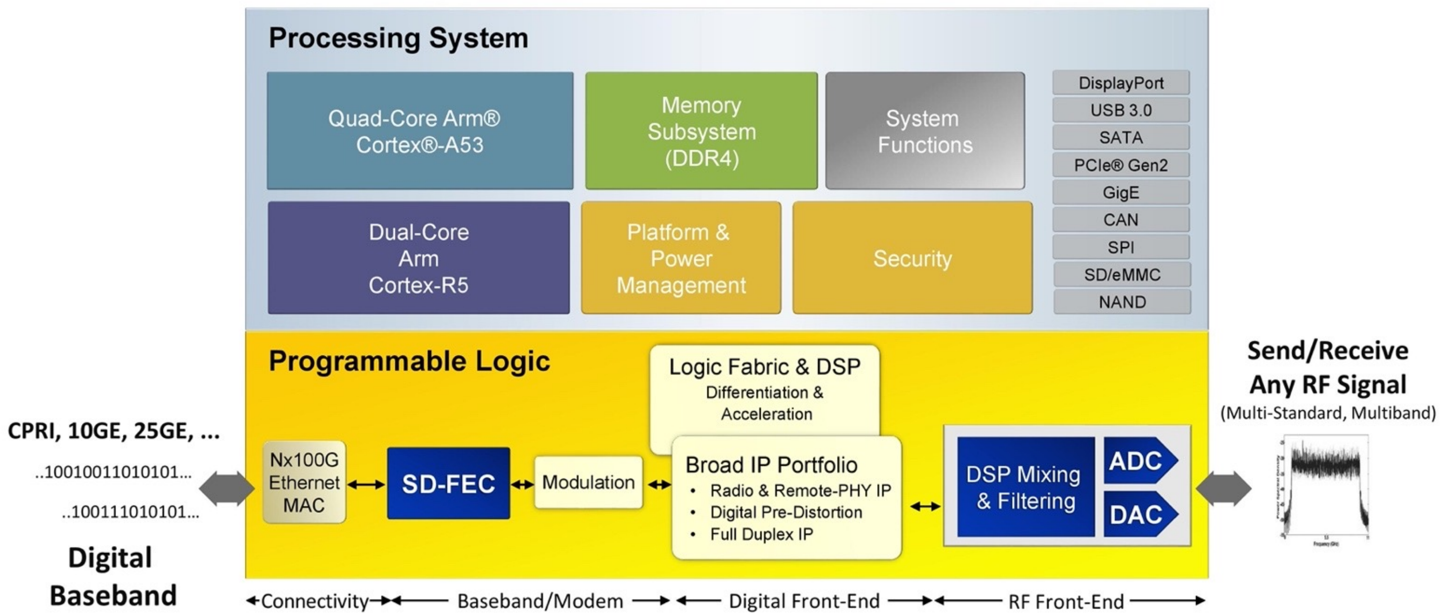


Block Diagram of ESP32-C3

FPGA/SOCs, RFSOCs

- A quad-core ARM
- A dual-core ARM
- An FPGA
- 10 Gsps ADC/DACs

Single-Chip Adaptable Radio Platform



Integrating the RF Signal Chain

Embedded Programming

- As embedded-level computational devices have become more and more capable, there has been some appearance of higher languages (python)
- For the most part, however, embedded programming is still heavily dominated by C and/or related languages.
  - Performance is very important
  - Stability is very important
  - C is lower level and therefore more transparent…if done correctly fewer things can go wrong

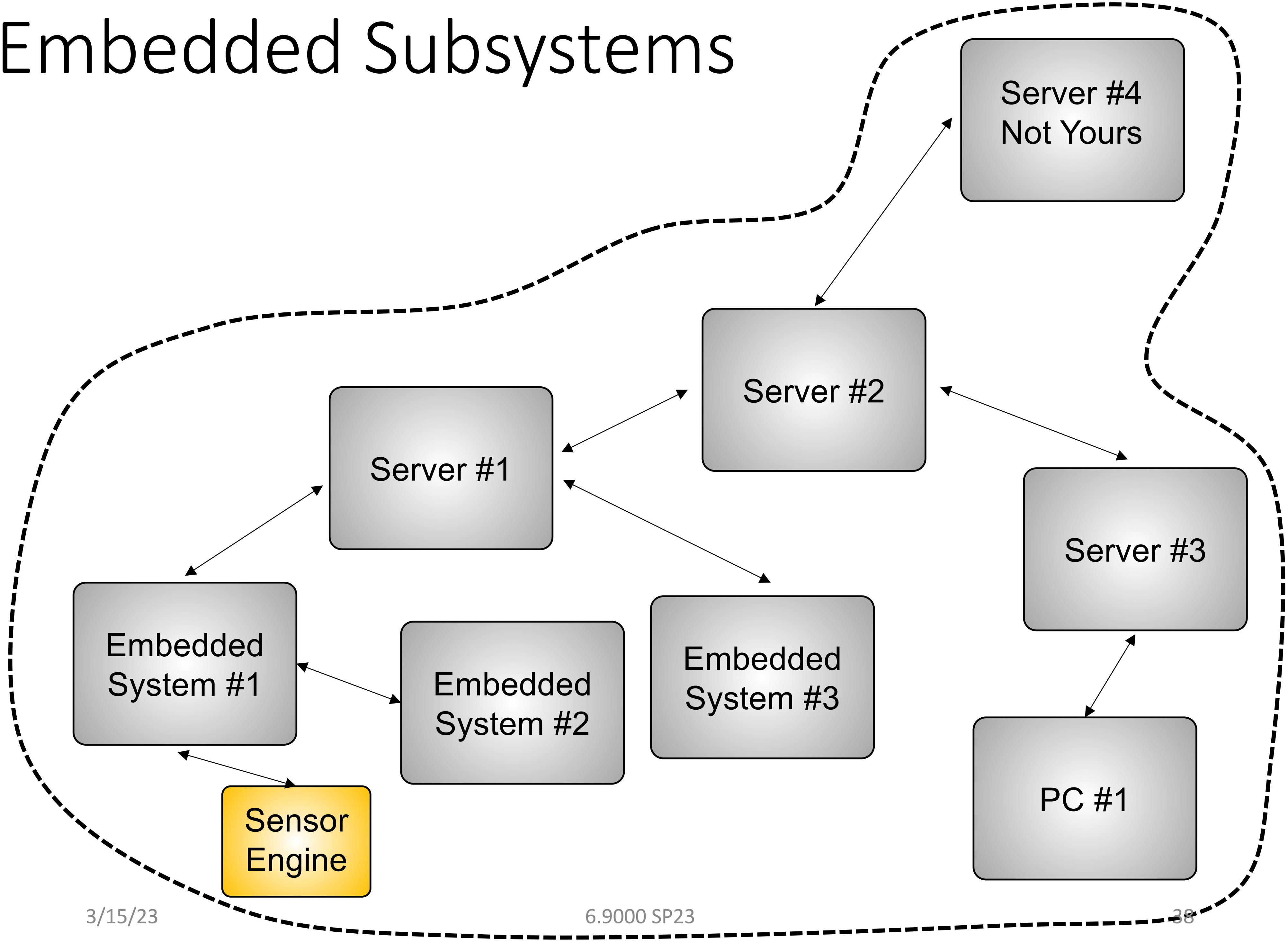
Embedded Programming Rules

- A lot of rules have to be in place to make sure devices running embedded code are super stable for long periods of time
- Updates/fixes are very, very costly! Basically have to do a recall.
- JPL's C coding standard is a great document to read and inform in regards to writing software for embedded:
  - https://yurichev.com/mirrors/C/JPL_Coding_Standard_C.pdf
  - Makes sense since JPL does stuff with satellites and other things where you can't really drive in and reboot the system

OTA Updates

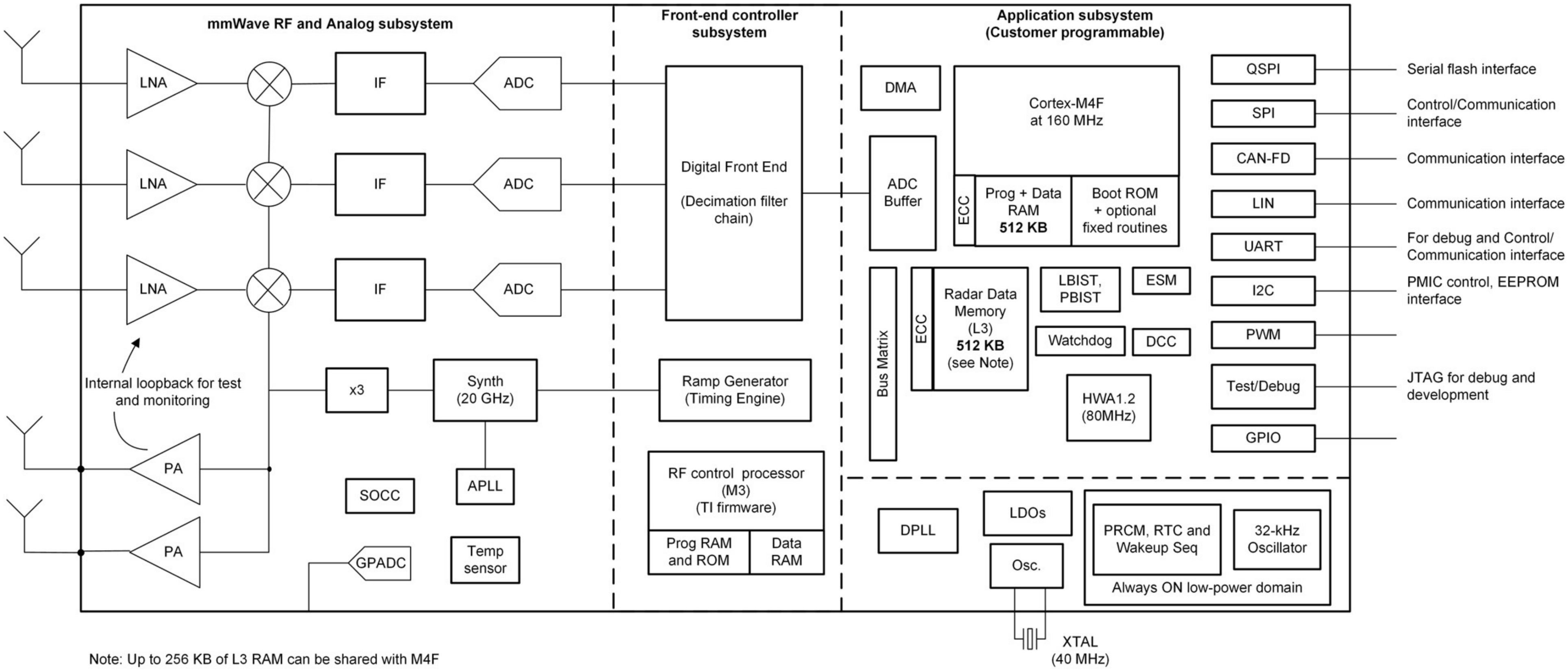
- Some new microcontrollers are starting to posess the ability to do “over-the-air” reprogramming which would enable updates, but this is still somewhat rare

Embedded Subsystems



Example: TI's AWRL6432

- Single-chip low-power 57-GHz to 64-GHz automotive mmWave radar sensor



Note: Up to 256 KB of L3 RAM can be shared with M4F

IMUs

- IMUs have been used for step counting for decades now.
- You can now buy cheap IMUs that deploy step-counting algorithms inside and report steps to you…
- Don't even need to spend your microcontroller's compute cycles on finding peaks/troughs of steps.

VOC sensor

Datasheet SGP41

Air Quality Sensor for VOC and NO_x Measurements

Disclaimer: all specifications are subject to change without further notice

- MOx based gas sensor for air quality applications
- Outstanding long-term stability and lifetime
- I²C interface with digital output signals
- Very small 6-pin DFN package: 2.44 x 2.44 x 0.85 mm^3
- Low power consumption: 3.0 mA at 3.3 V
- Tape and reel packaged, reflow solderable

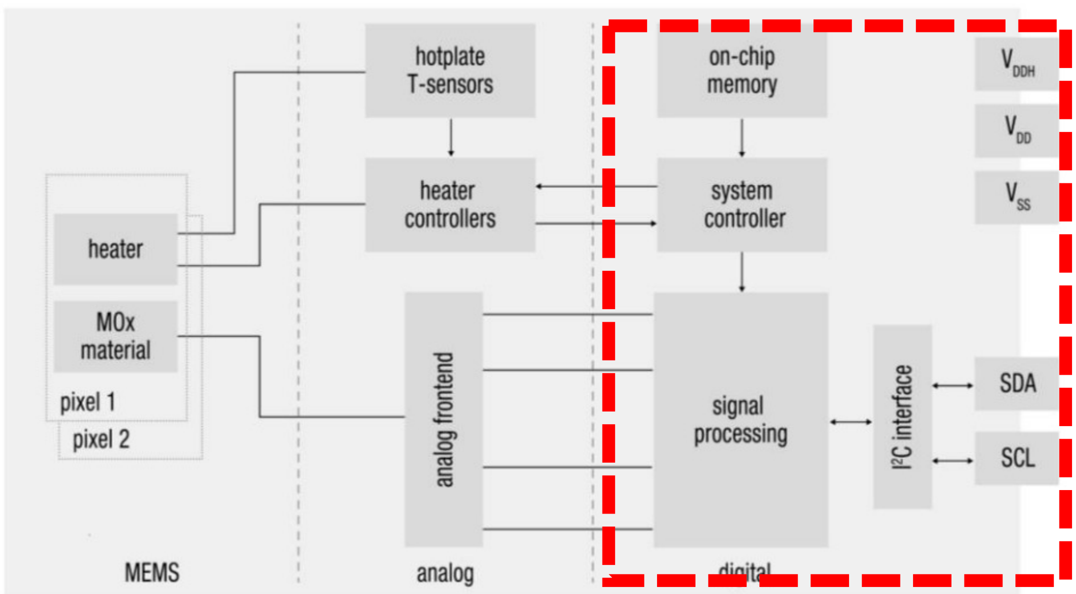
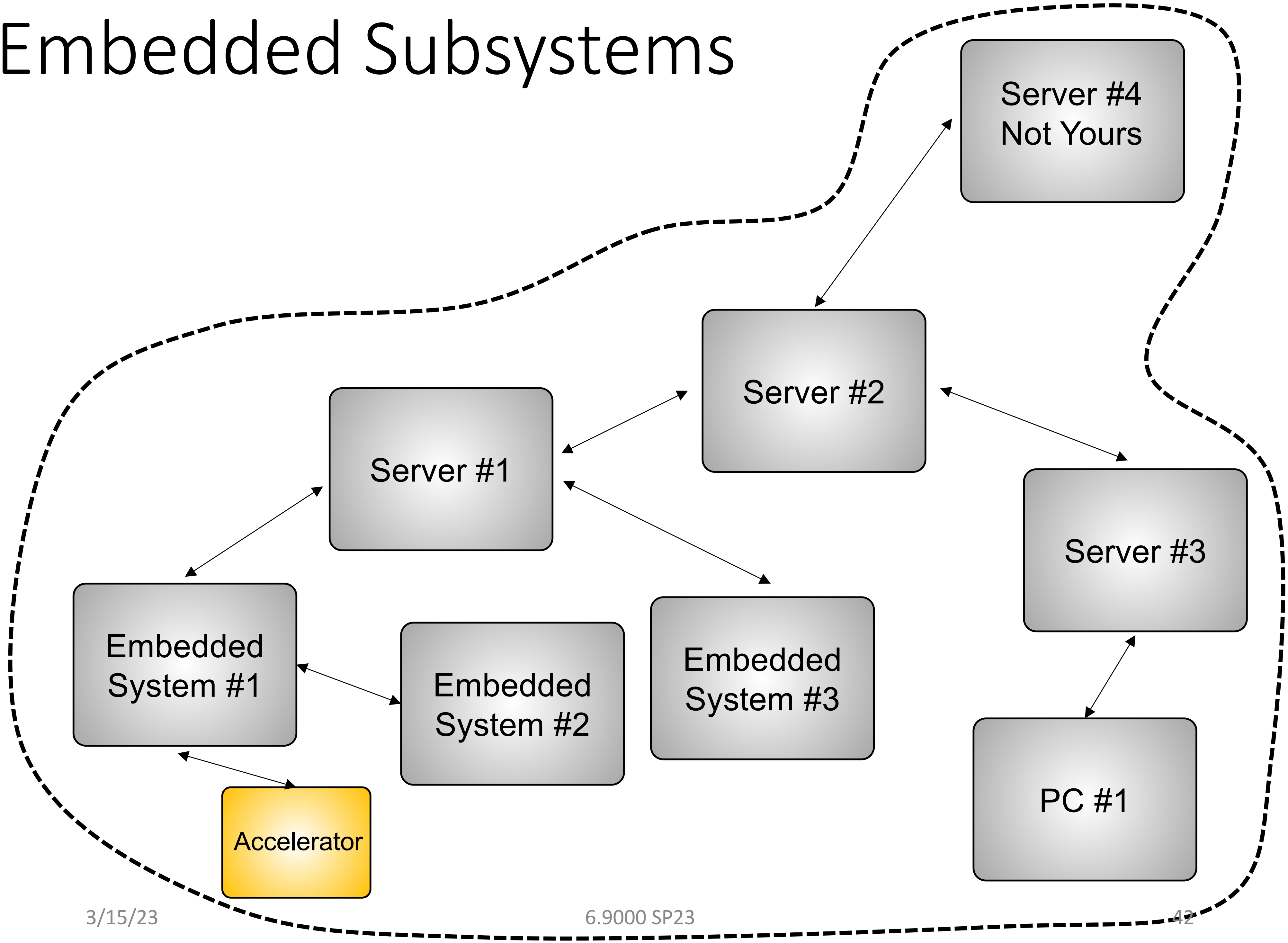


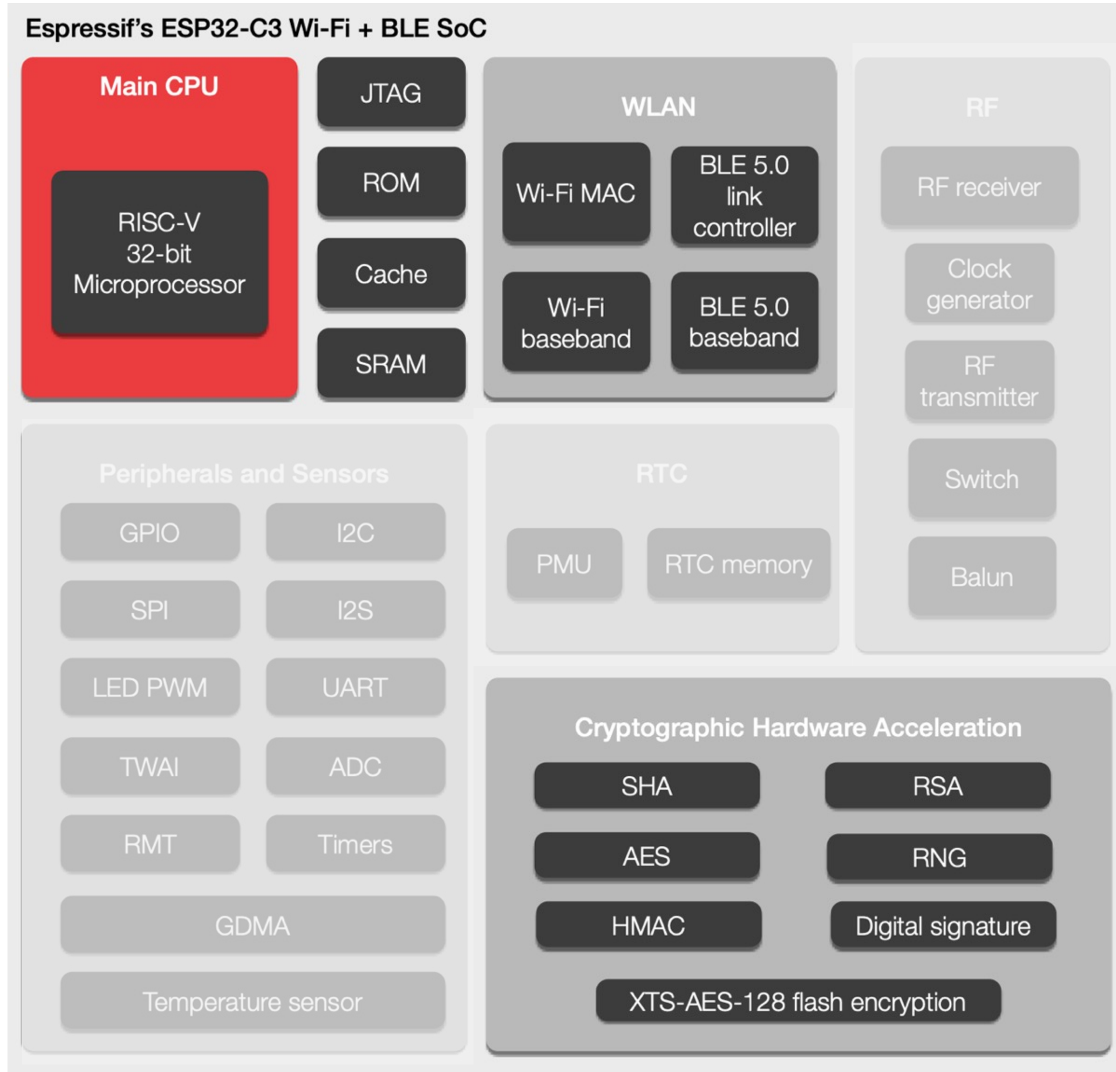
Figure 1 Functional block diagram of the SGP41.

Embedded Subsystems



The ESP32-C3

- Quite a few accelerators



Block Diagram of ESP32-C3

Other Types of Accelerators?

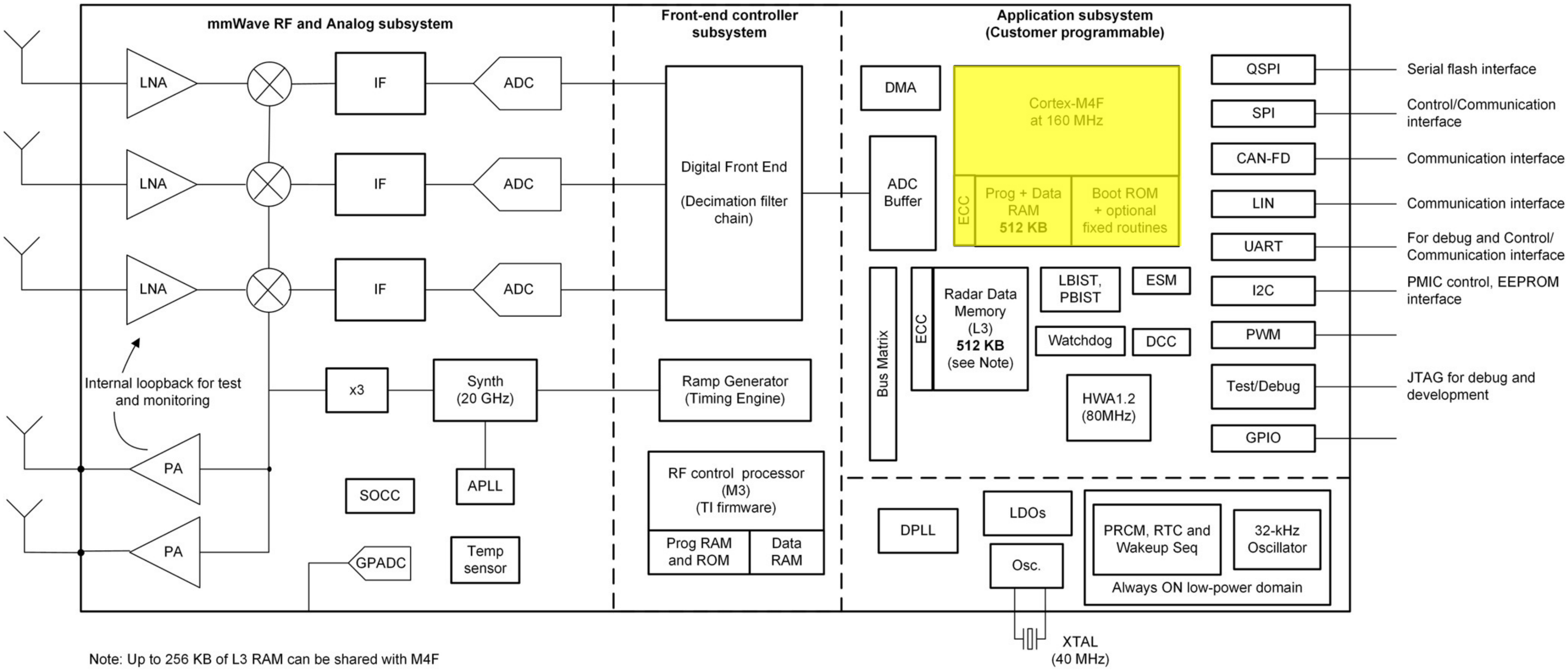
- Floating-Point Units (FPUs). You can do floating point representation using just `ints`, but it takes time and CPU cycles. Instead spend a bit more on a hardware FPU
- Compression algorithms
- Interface hardware (ethernet, memory management)
- Machine-Learning Circuits (TPU)

An Interesting Pattern

- You don’t see as many raw accelerators in their own chip
- What you are seeing are many accelerators accompanied by additional processor cores.
  - A lot of ARM cores
  - Starting to see more RISCV cores show up
  - Little in the way of x86 ones because of bloat
  - Other proprietary cores

TI's AWRL6432

- Single-chip low-power 57-GHz to 64-GHz automotive mmWave radar sensor



I don't want this. Just give me the digital front-end

uBlox SARA-R500 series

- The same cellular modem used in the Boron LTE board some of you requested:
- Has a full internal processor in it as well, with some degree of programmability

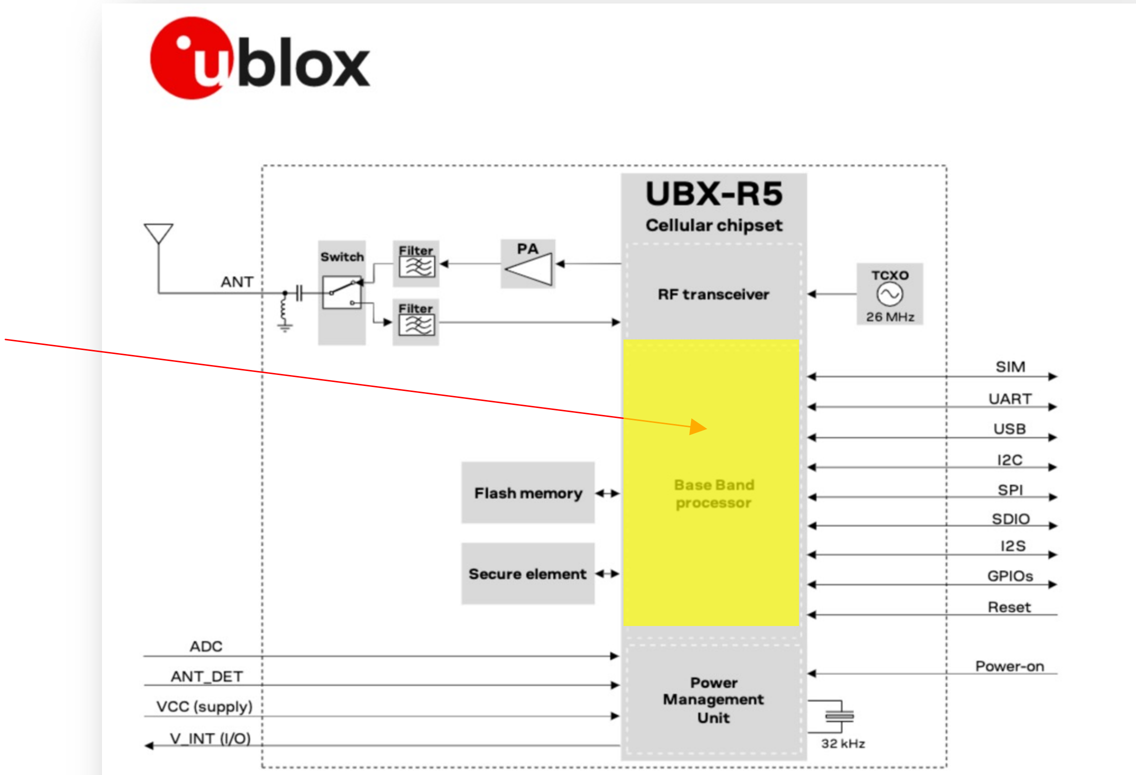
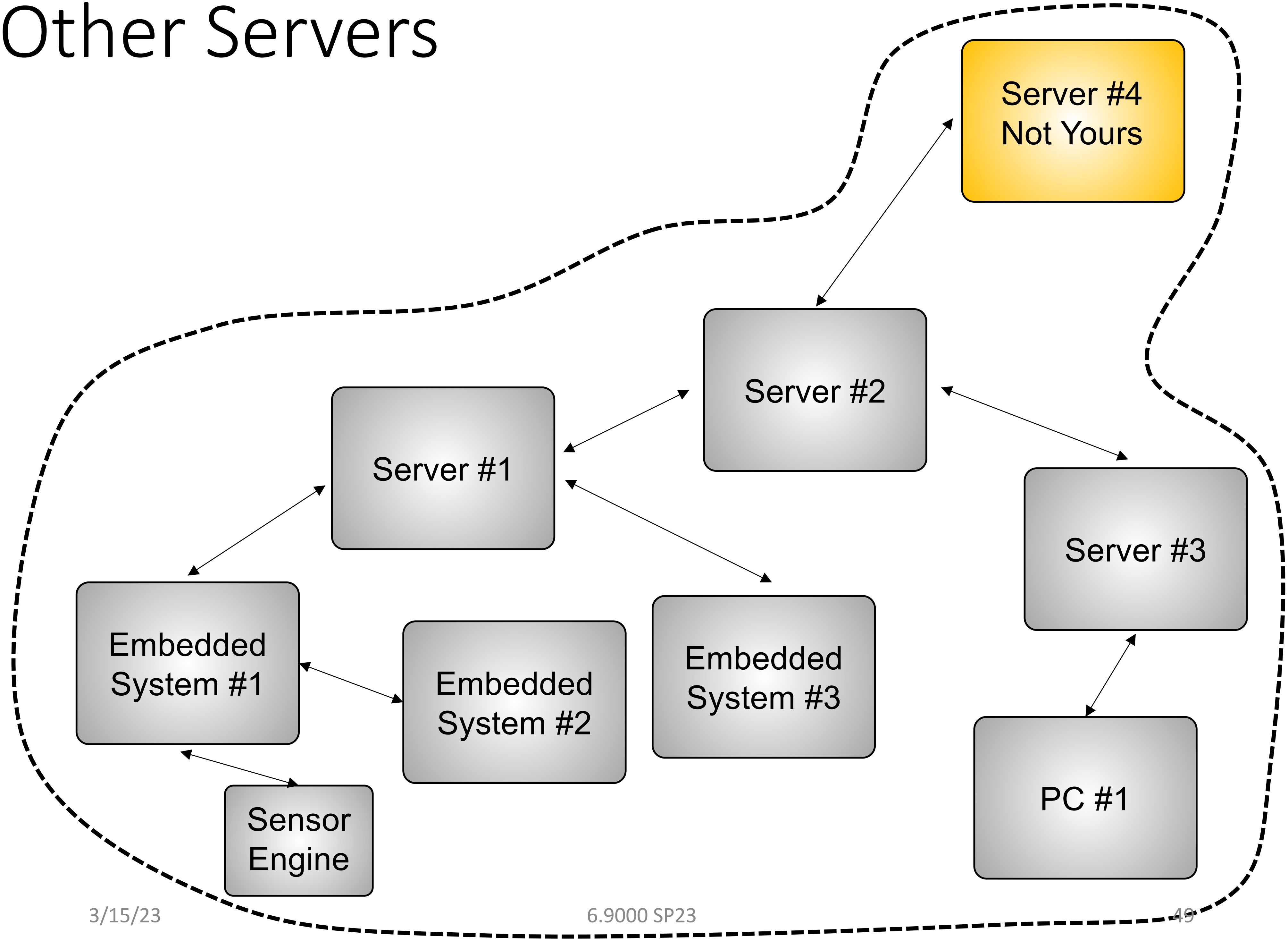


Figure 2: SARA-R500S block diagram

Even an Embedded System may have multiple programmable elements on it

- Do you use all the computation?
- Is it cost-effective to do so?
  - Not always so clear-cut

Other Servers

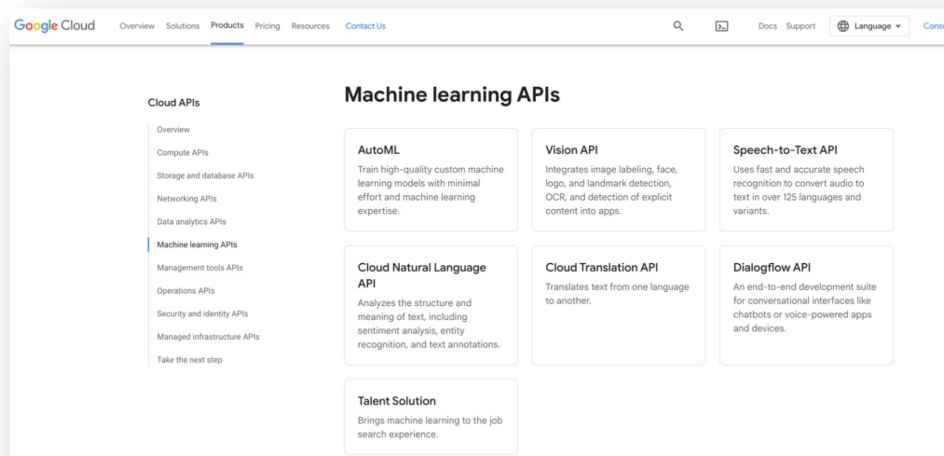


Other Servers

- You may also find yourself using (either for free or via paying) other servers
- Companies and organizations will make callable resources accessible to perform particular computation tasks

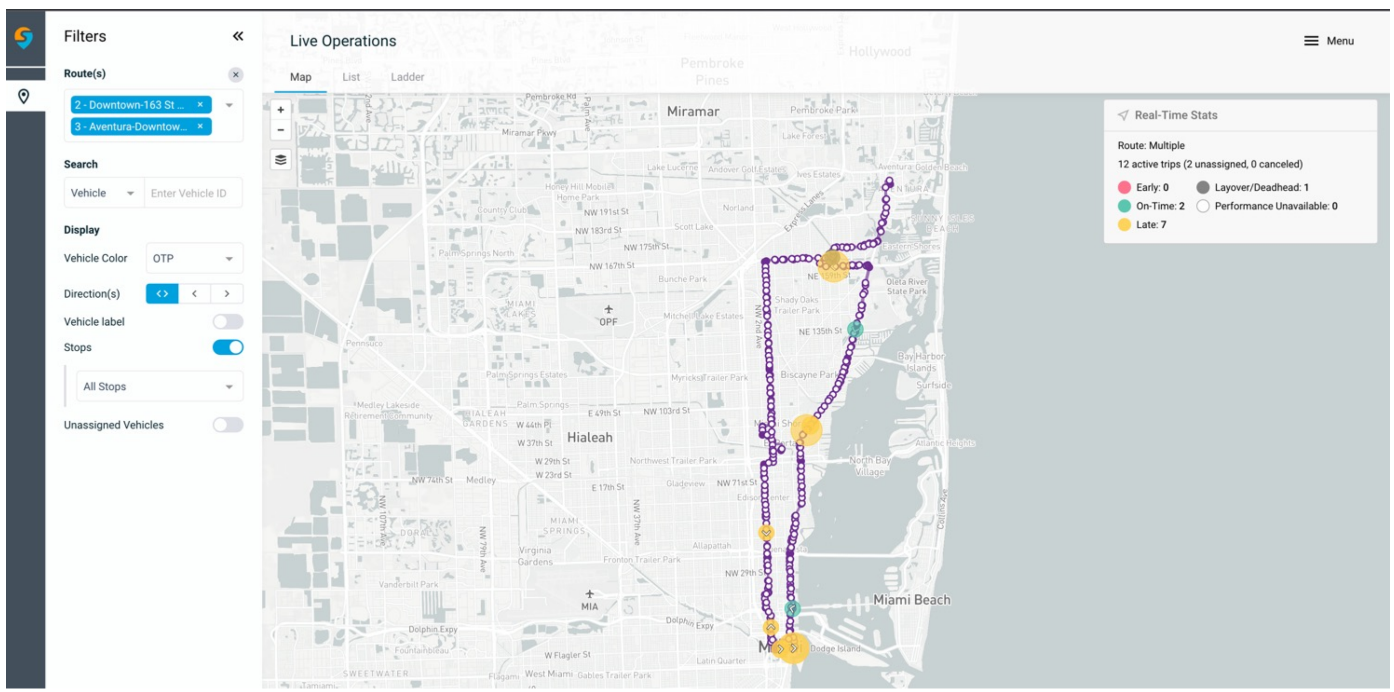
chatGPT

- Companies and organizations will make callable resources accessible to perform particular computation tasks

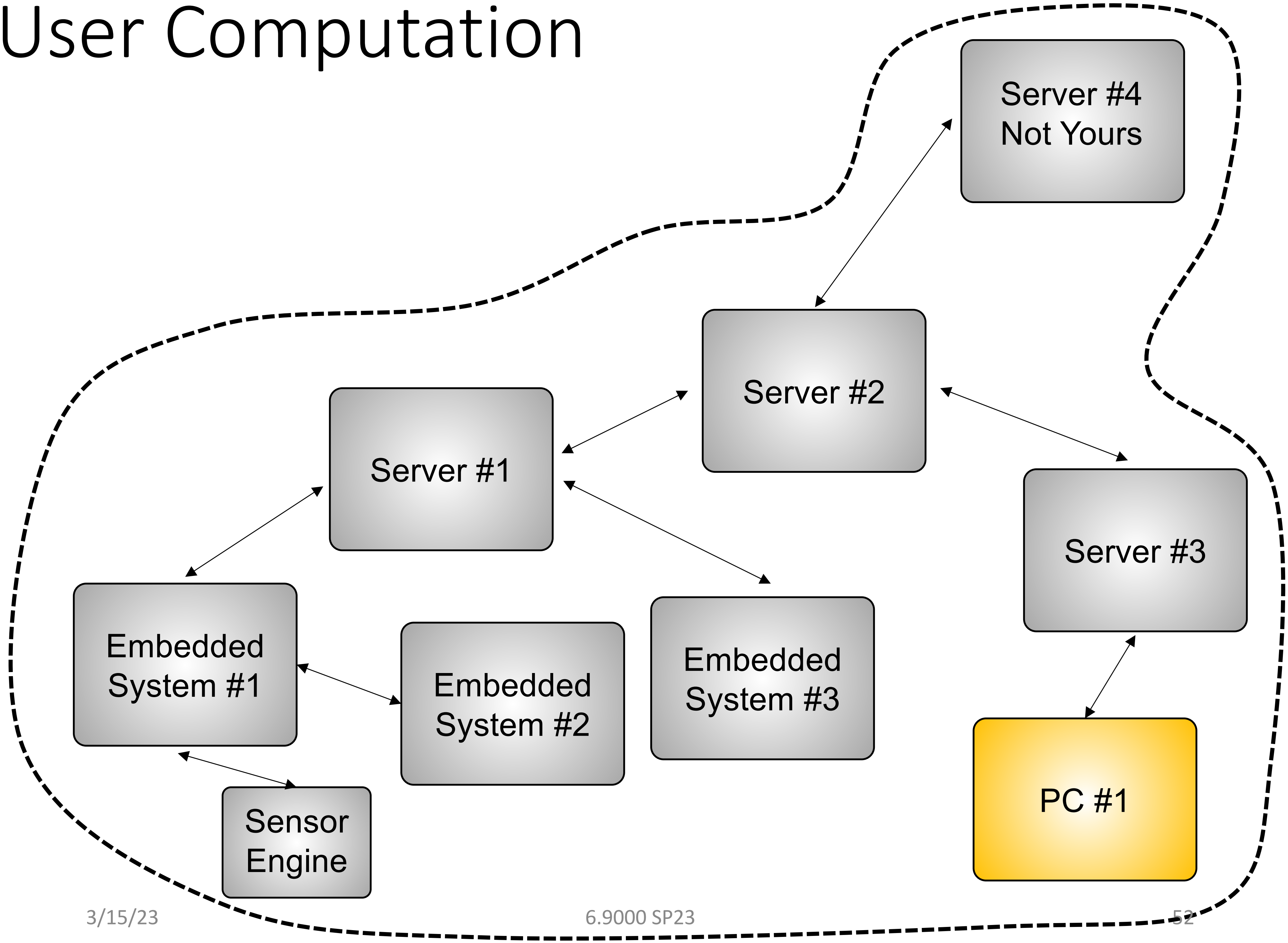


Swiftly

- Company we'll work with
- Have servers, provide digested/presented data that isn't really their data



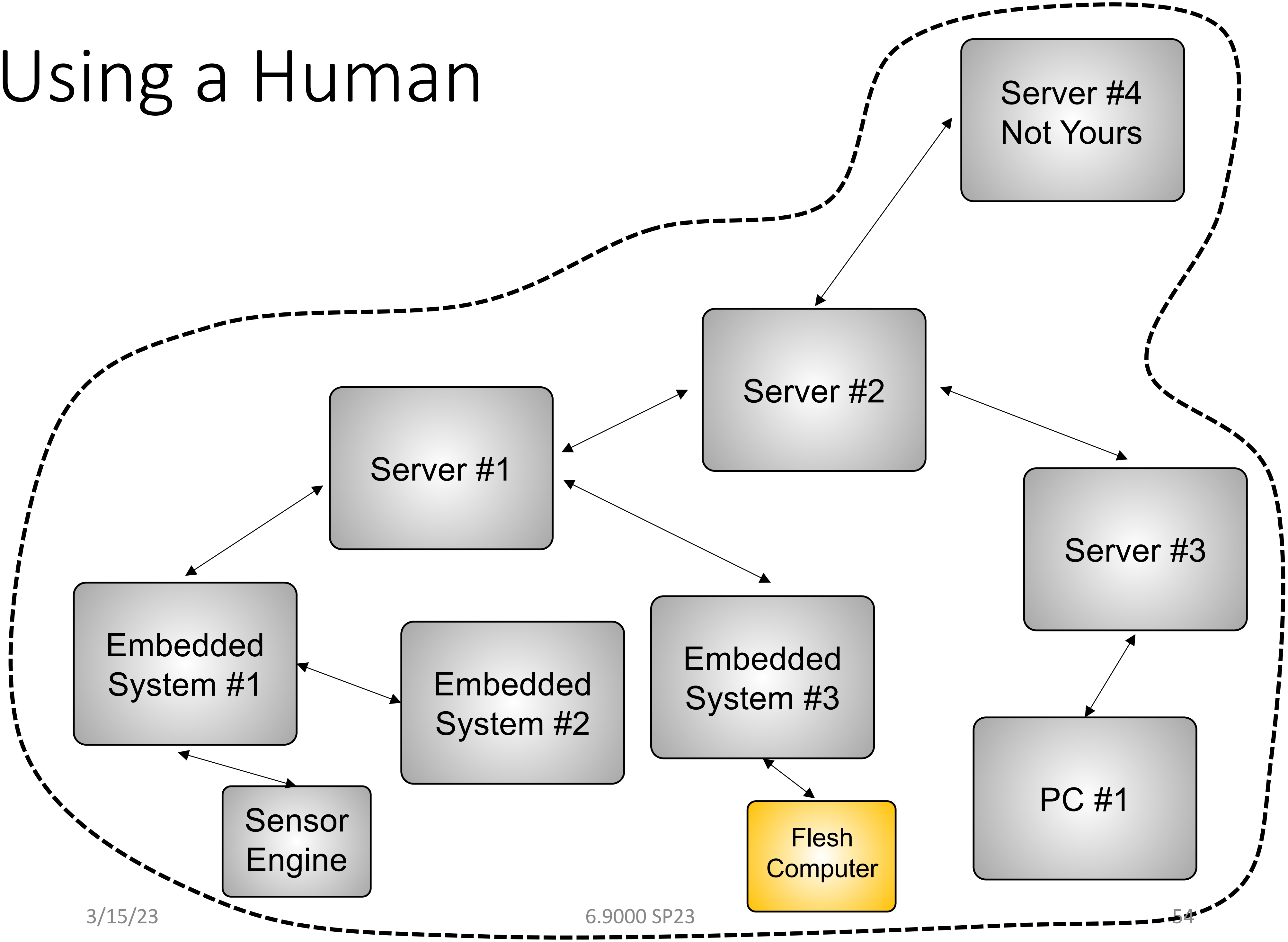
User Computation



Consumer/Client Computation

- You can also “farm out” computation to consumers
- Example:
  - Do you generate beautiful graphs on a server then send the user/consumer an SVG?... (Pros/Cons?)
  - Do you send the user raw data along with javascript/html and have their computer render it locally? (Pros/Cons?)
- Can you take advantage of consumer’s device computation to avoid needing to do additional computation on your own hardware? Examples? Examples that we have already seen???....

Using a Human



Using Humans

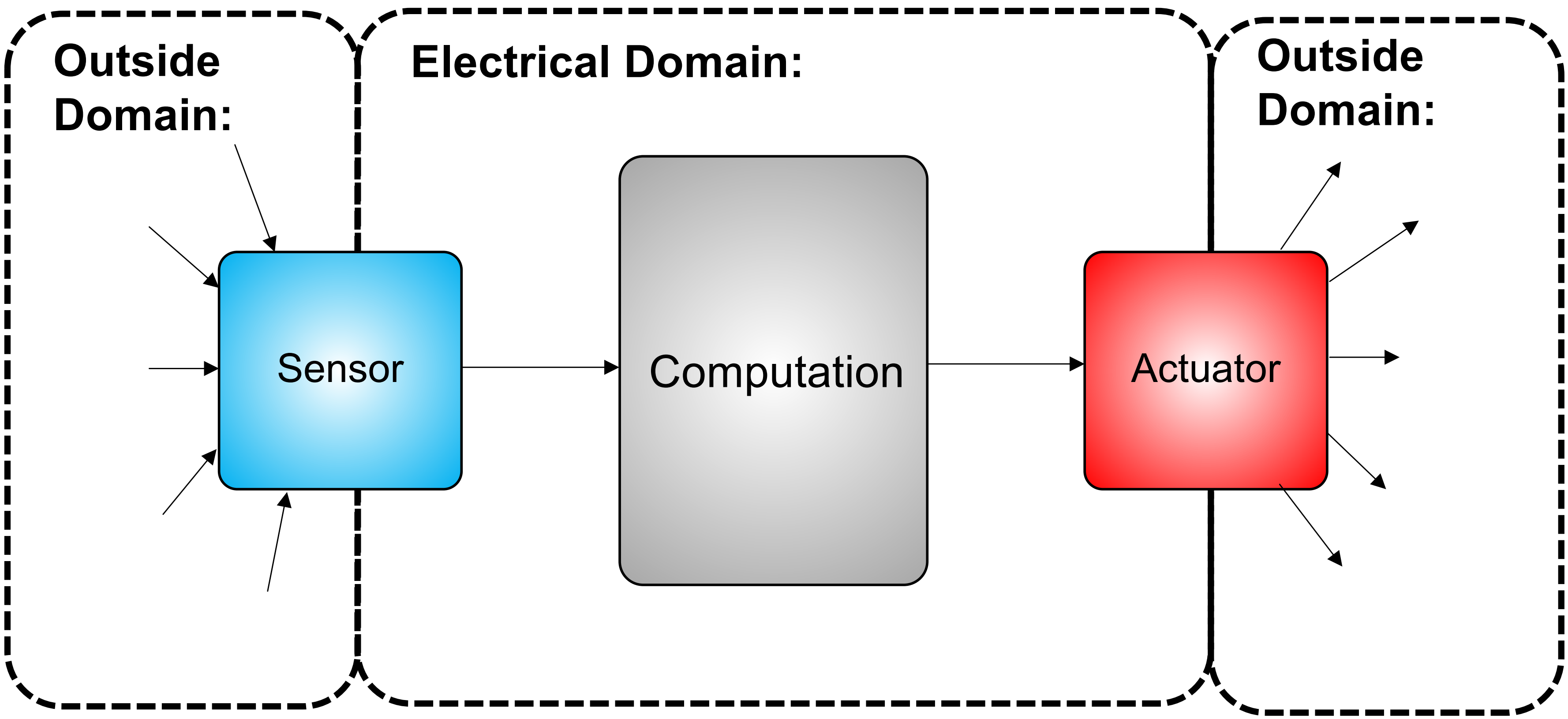
- Can you convince/entice/trick the human to do some computational tasks which would be otherwise hard to do given input sensor data?

"Uck. This bathroom is disgusting. What is wrong with people?…. I will form part of a transducer and do some computation."

- Don't need to worry about designing some complicated sensor network to deduce

Update our Big Idea

- Most modern computational devices take on this form*:



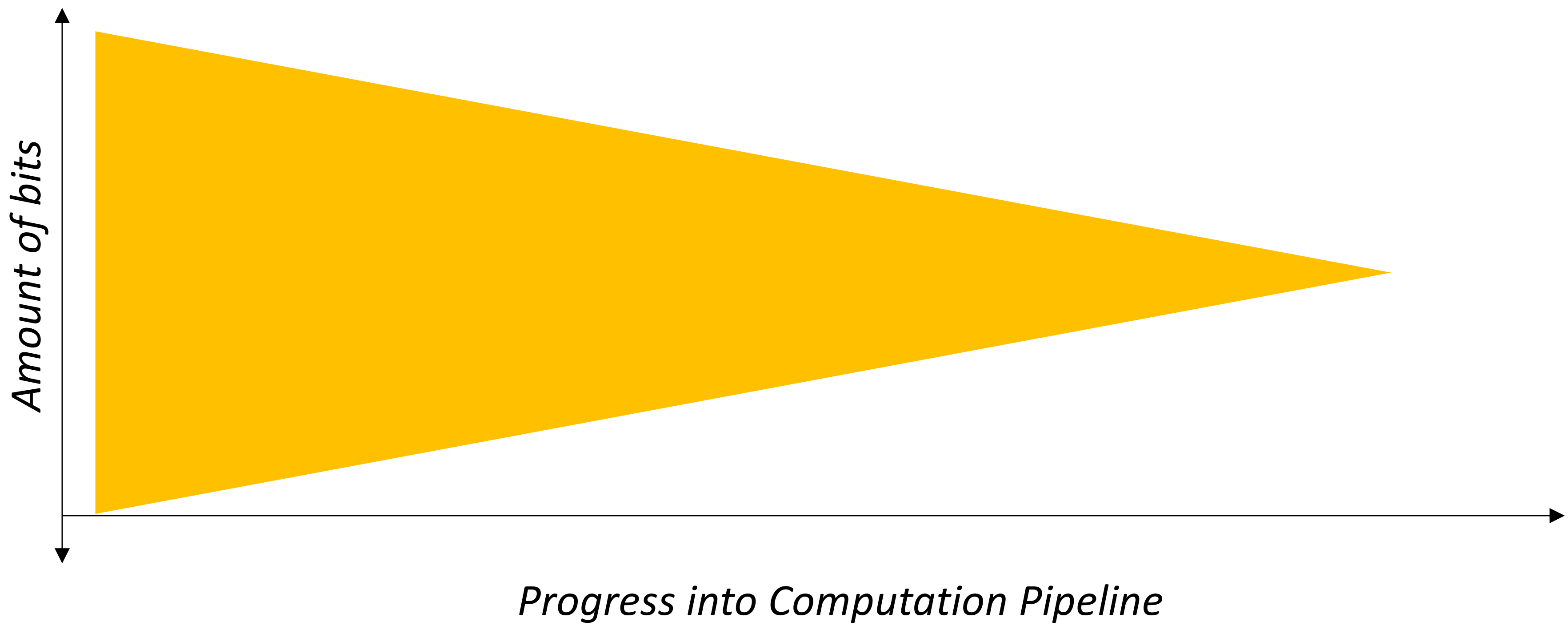
*There's always exceptions, so this isn't a 100% true statement

Update our Big Idea

- While the general pattern has remained constant since the 1900's, the scope/breadth of the inputs and outputs has scaled tremendously…
- In the 1940's, a computation block would take in several hand-entered numbers and maybe solve some third/fourth order differential equation and return the coefficients as a result
- In the 2020's, a computation block may take in Gbps of video, audio, environmental, meta data and control entire fleets of drones and direct vehicles and keep you entertained/engaged with random stuff

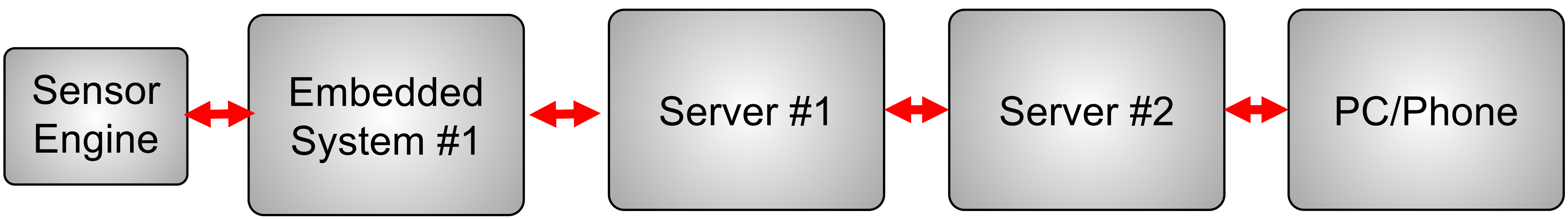
The Flow of Data

- Data being processed usually gets scaled/compressed/interpreted the further into your compute pipeline.

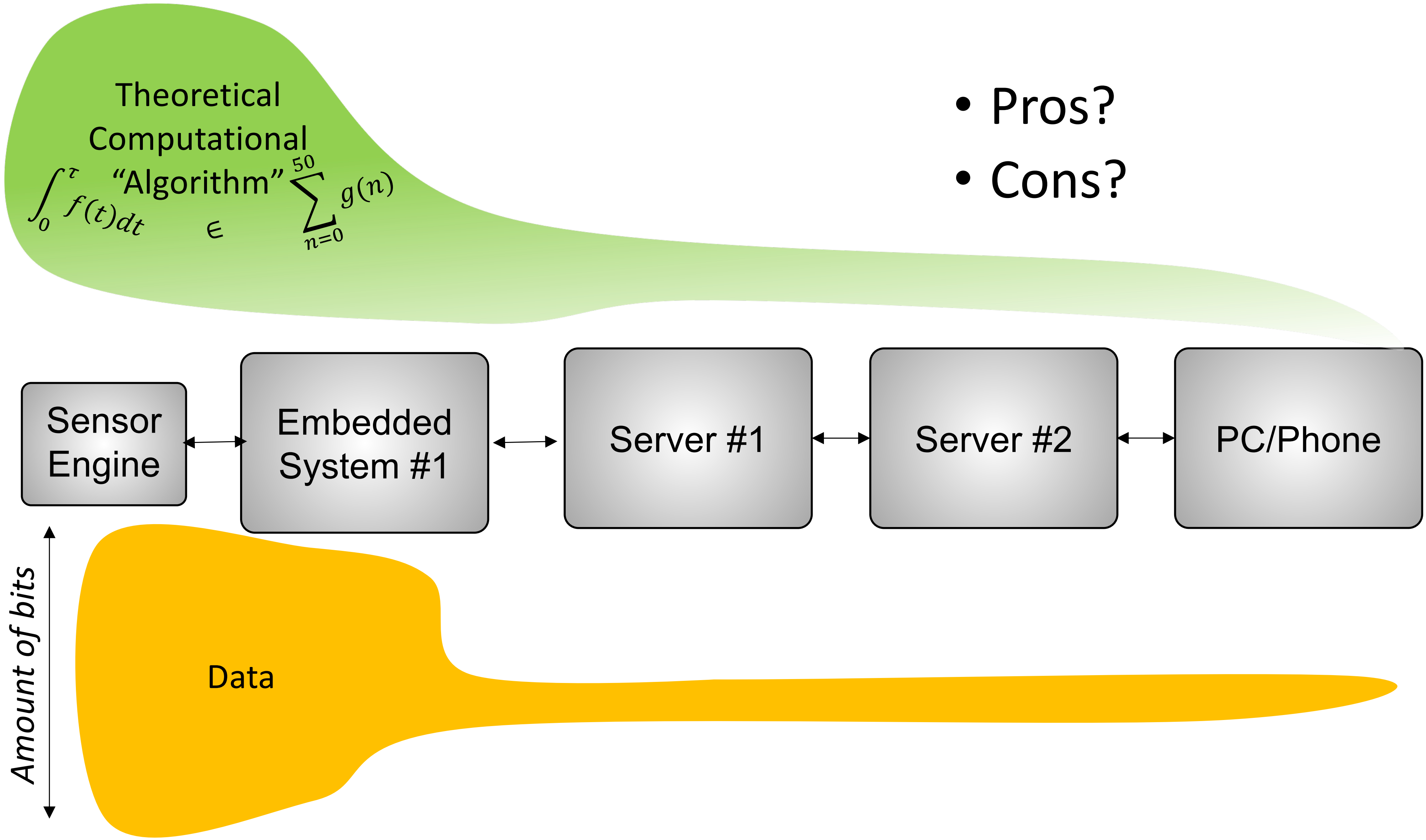


Distributing Your Algorithm Over the System

- How early you start compression impacts how much data needs to move between each link in your computation chain



Front-Load Computation



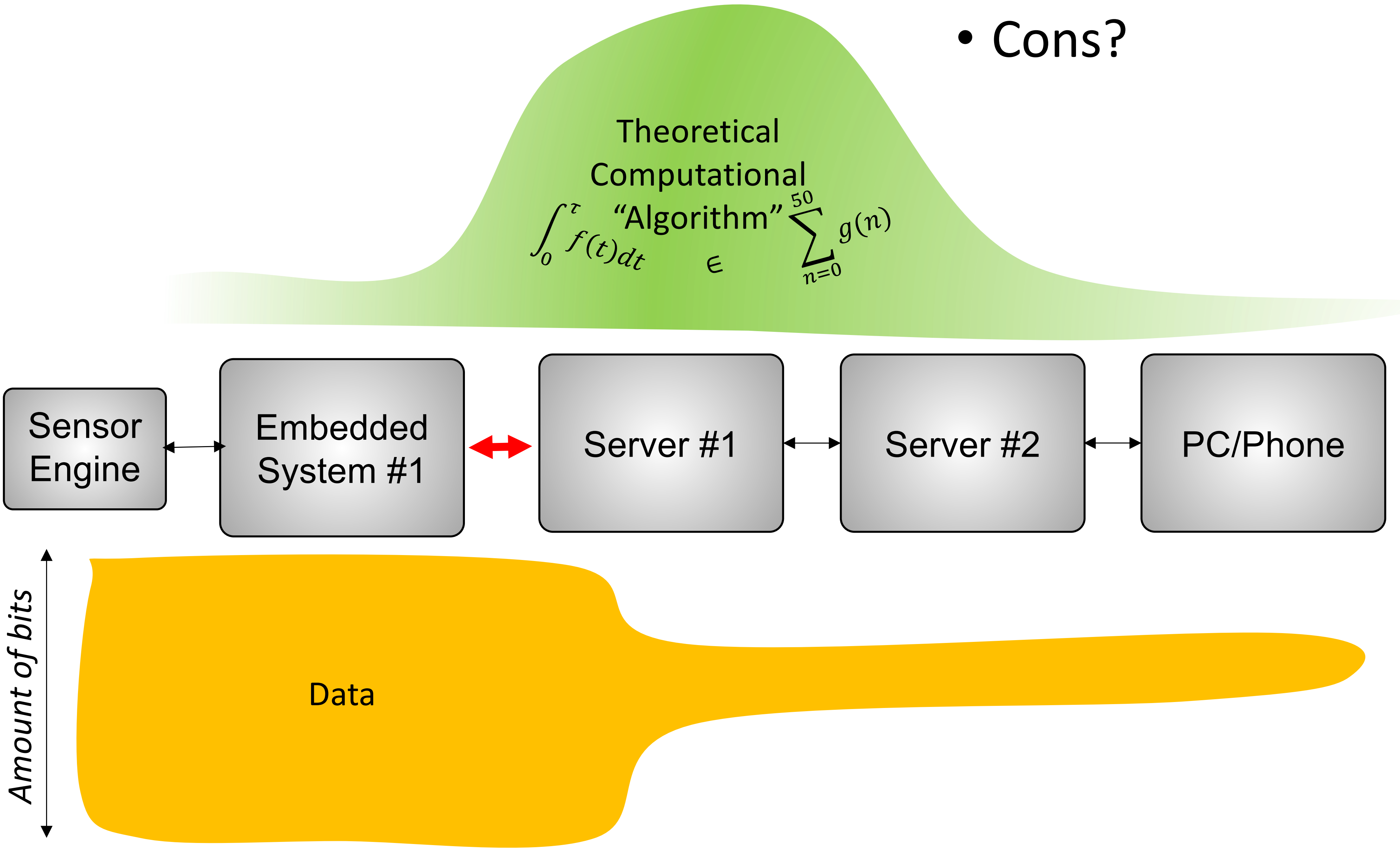
- Pros?
- Cons?

Tiny ML/Edge AI

- Traditionally ML etc. was deployed in servers, so a lot of raw data would need to get sent up over network
- TinyML and associated fields have looked into extracting meaning from large data sets in efficient manners using embedded-type hardware
- Result is you need to send much less data over the network:
  - Saves money
  - Saves energy

Back-Load Computation

- Pros?
- Cons?



Conclusion

- There are a lot of choices to be made