# Embedded System Design

## Security Considerations
## &
## Low-Power Design

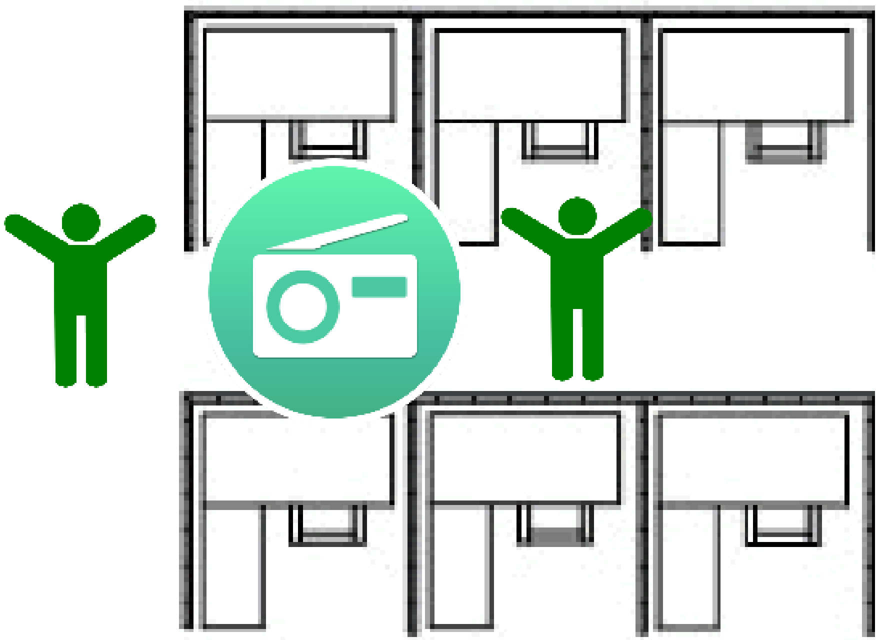
Jem Berkes
ECE, University of Manitoba

# Security Considerations

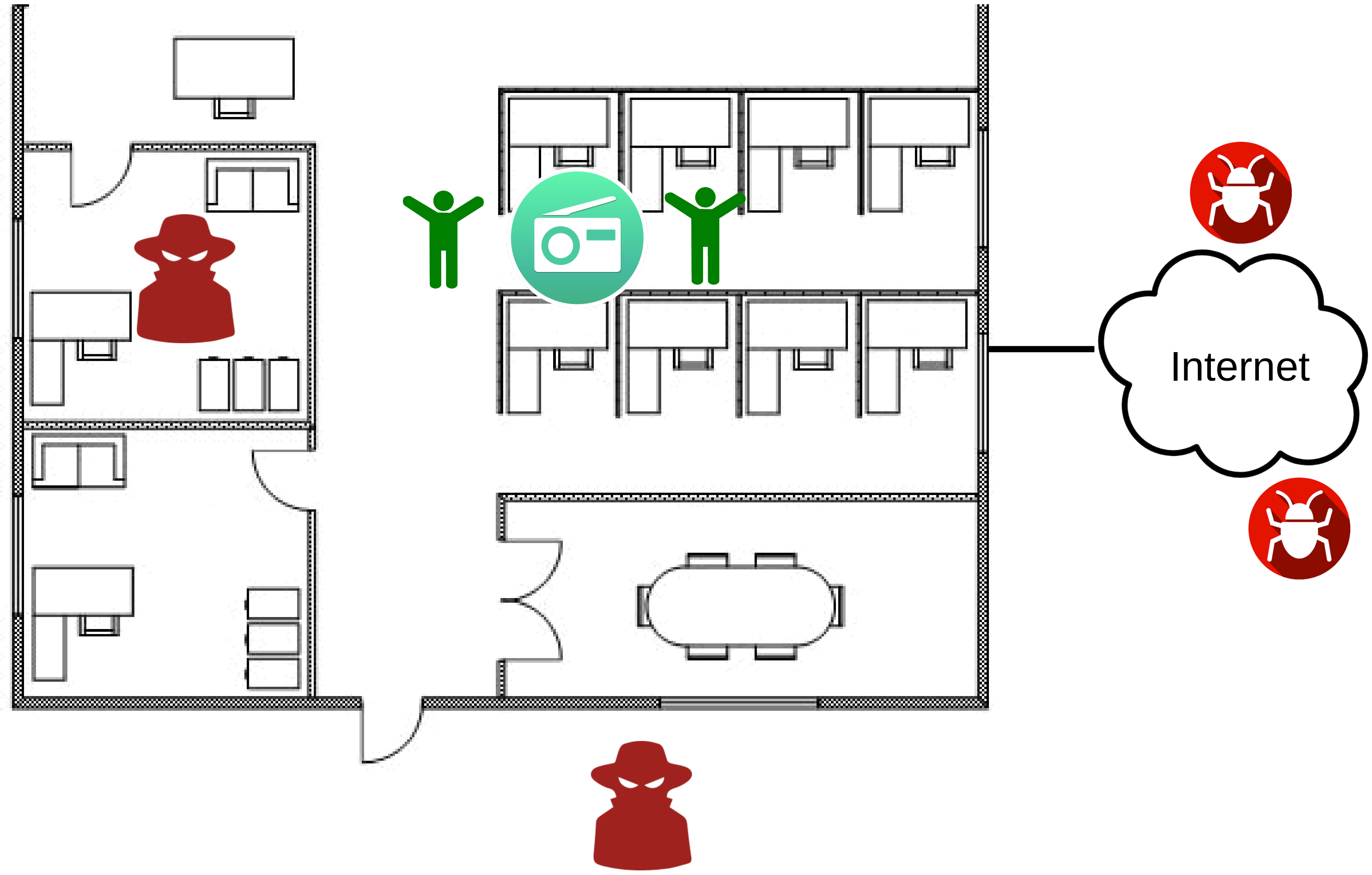
# Basics

- Embedded/IoT are feature-rich computers
- Sit in the physical world
- Attacks are very likely
  - Automated worms & viruses
  - Curious or malicious people

# What we see



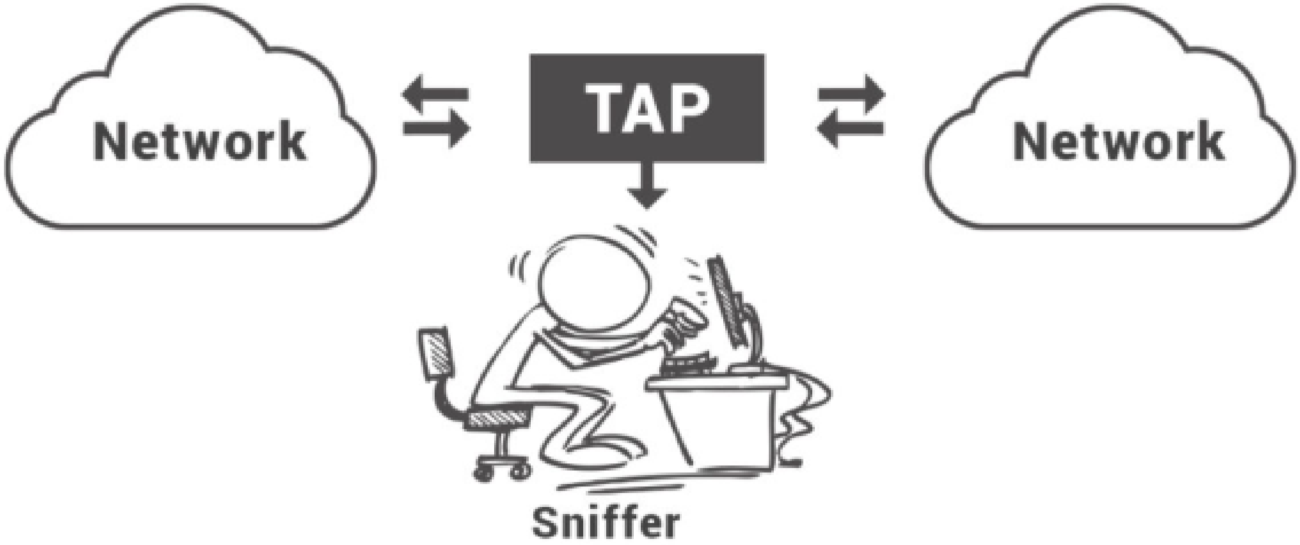
# There’s more around you!



# Types of Attacks

# Eavesdropping

- Someone intercepts or “sniffs” data packets
- Can expose or steal sensitive data

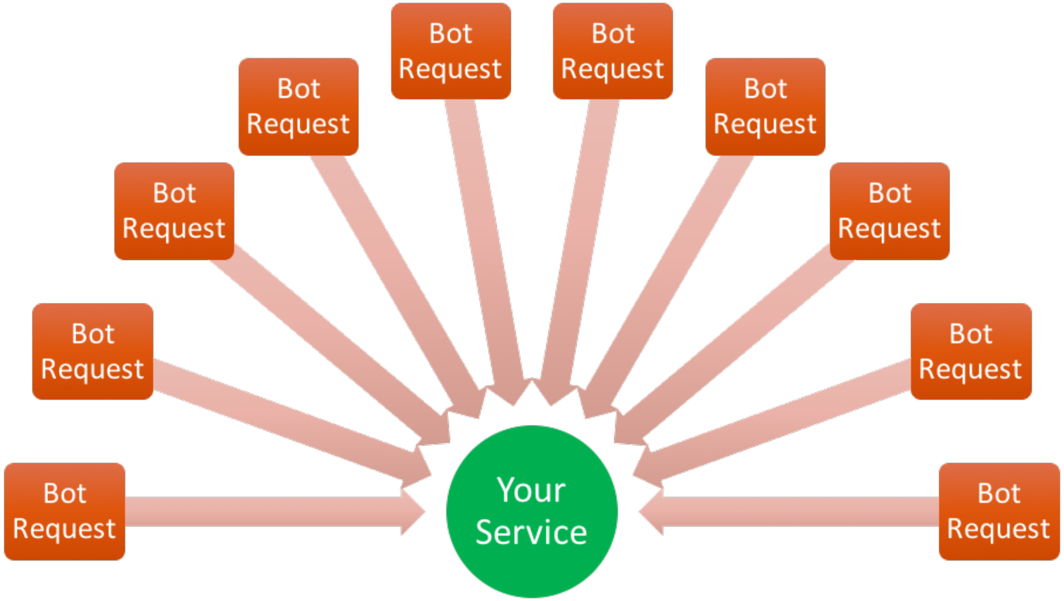


**Solution:**

Encrypt your traffic, use SSL/TLS

# Denial of Services (DoS)

- Someone floods your devices with requests
- Tries to slow down or disable the service

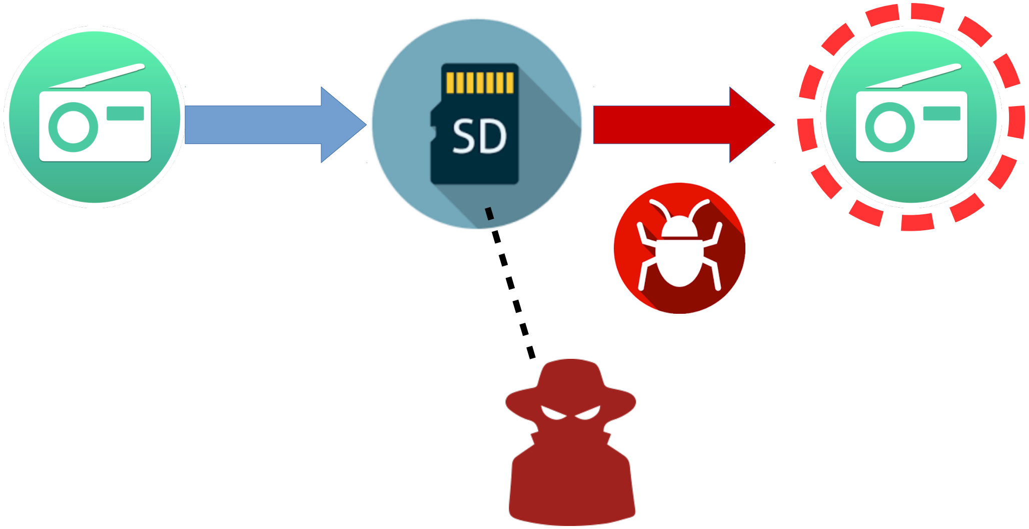


**Solution:**

Implement rate limiting, or auto-ban malfunctioning clients

# Device Tampering

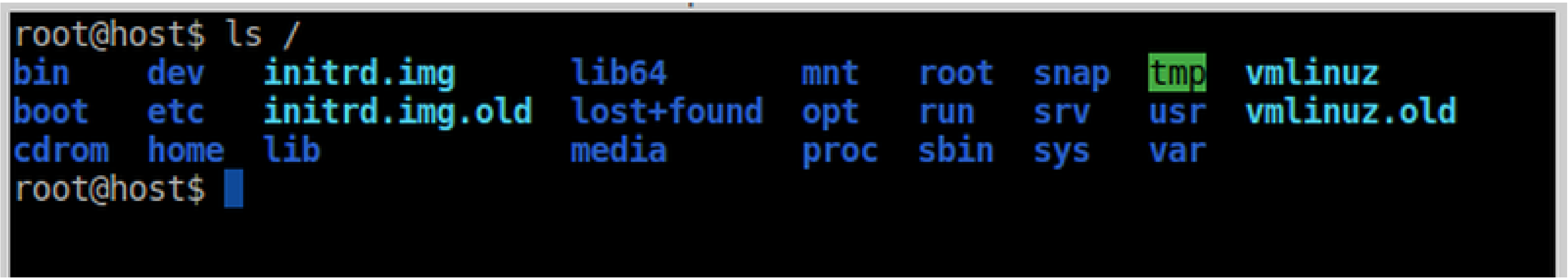
- Someone accesses the disk and reads the files
- Or modifies the embedded software



See “Industrial Grade Concerns”

# Compromise or Hack

- The device is infiltrated
- Someone (or software) takes control



```
root@host$ ls /
bin    dev   initrd.img      lib64       mnt   root  snap  tmp  vmlinuz
boot   etc   initrd.img.old  lost+found  opt   run   srv   usr  vmlinuz.old
cdrom  home  lib             media       proc  sbin  sys   var
root@host$ 
```

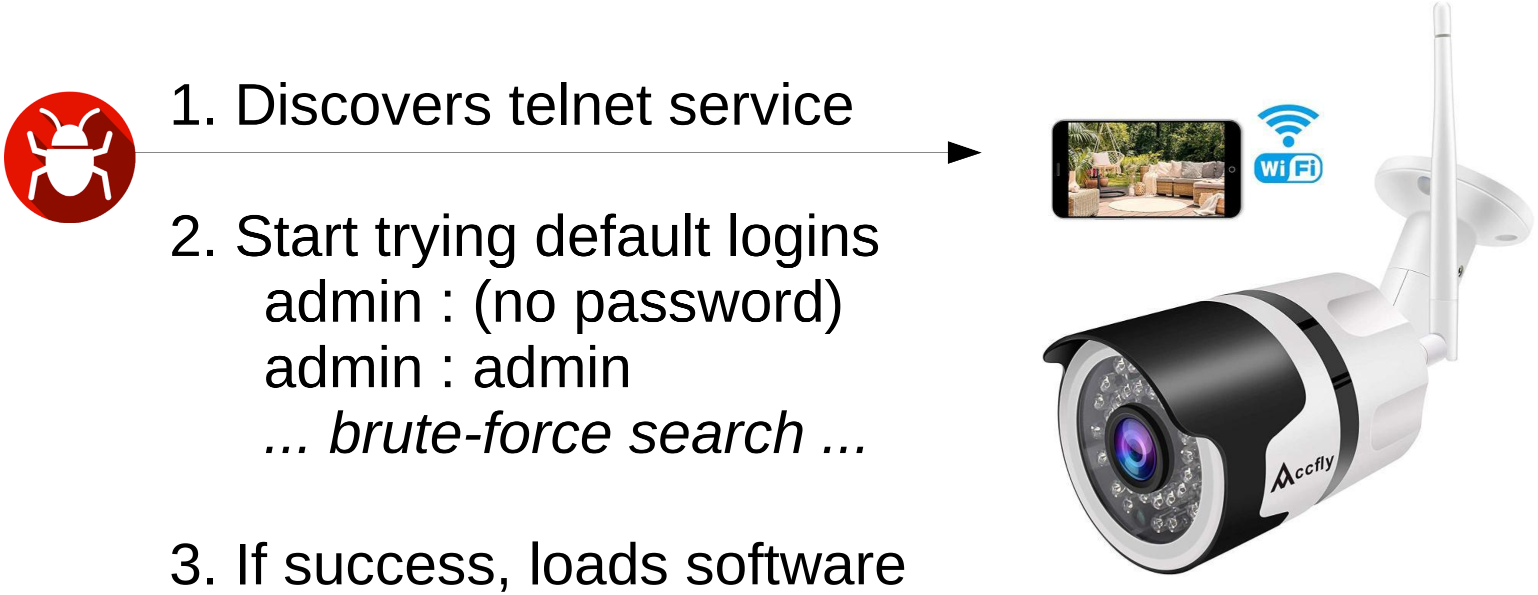
# Common Vulnerabilities
# (Leading to Compromise or Hack)

# Common vulnerability #1

- **Open service ports allowing logins**
  - ssh, telnet, http: login prompt
- *Plus* weak/default passwords

# Common vulnerability #1

- **Open service ports allowing logins**
  - ssh, telnet, http: login prompt
- *Plus* weak/default passwords



1. Discovers telnet service

2. Start trying default logins
   admin : (no password)
   admin : admin
   *... brute-force search ...*

3. If success, loads software

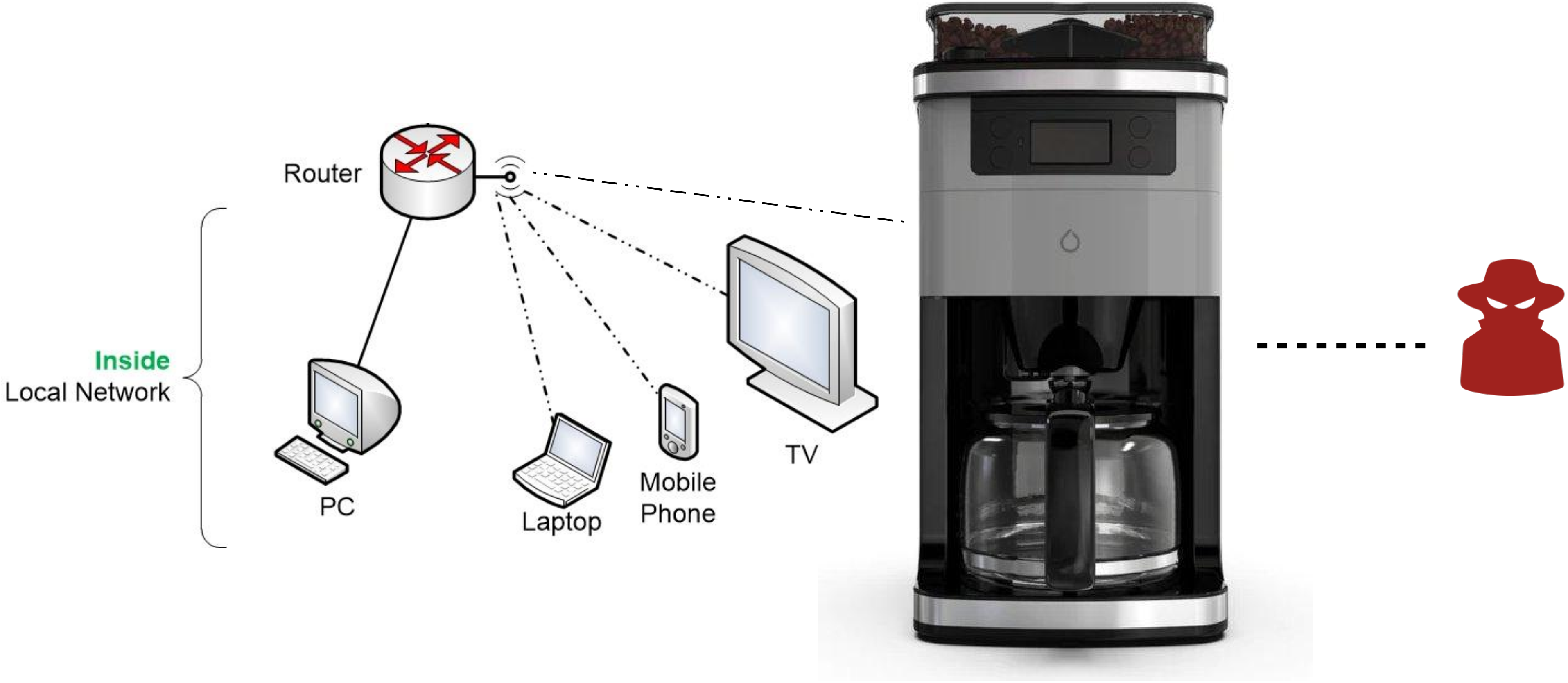
# Common vulnerability #2

- **Unauthenticated open services**
- Anyone can connect!

*See “Avast Hacked a Smart Coffee Maker All Kinds of Ways”*

# Common vulnerability #2

- **Unauthenticated open services**
- Anyone can connect!



*See “Avast Hacked a Smart Coffee Maker All Kinds of Ways”*

# Common vulnerability #3

- **Outdated OS and software**
- Everything needs patching eventually
- Can’t just leave a device alone for 5 years

# Wi-Fi
Security

# Wi-Fi Modes

- **Open**: no password, anyone can connect, unsafe
- **WEP**: old standard, broken, unsafe
- **WPA**: old standard, broken, unsafe
- **WPA2-TKIP**: uses old algorithm, unsafe
- **WPA2-AES**: next best option to WPA3
- **WPA3**: the newest standard, best option

# SSID

- SSID (Service Set Identifier) is hotspot name
- Publicly broadcast and visible to all
- Assume SSID is visible to everyone
- Hiding SSID doesn’t enhance security

# Wi-Fi Can Be Risky

- “KRACK” was a very severe WPA2 attack from 2017-2018
- Some embedded/IoT devices with old firmware
- What can the attacker do?
  - Intercept wireless traffic, without a password
  - Inject packets and manipulate connections

- HTTPS (aka TLS) helps protect against this

Always use SSL/TLS!

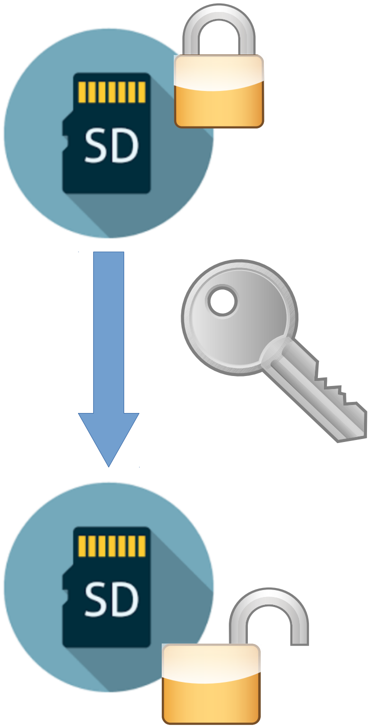
# “Industrial Grade” Concerns

# Physical Tampering

- People have physical access
- They could break open the device
  - Remove SD card
  - Connect to disk interface
- Don’t want people tampering with your embedded sys

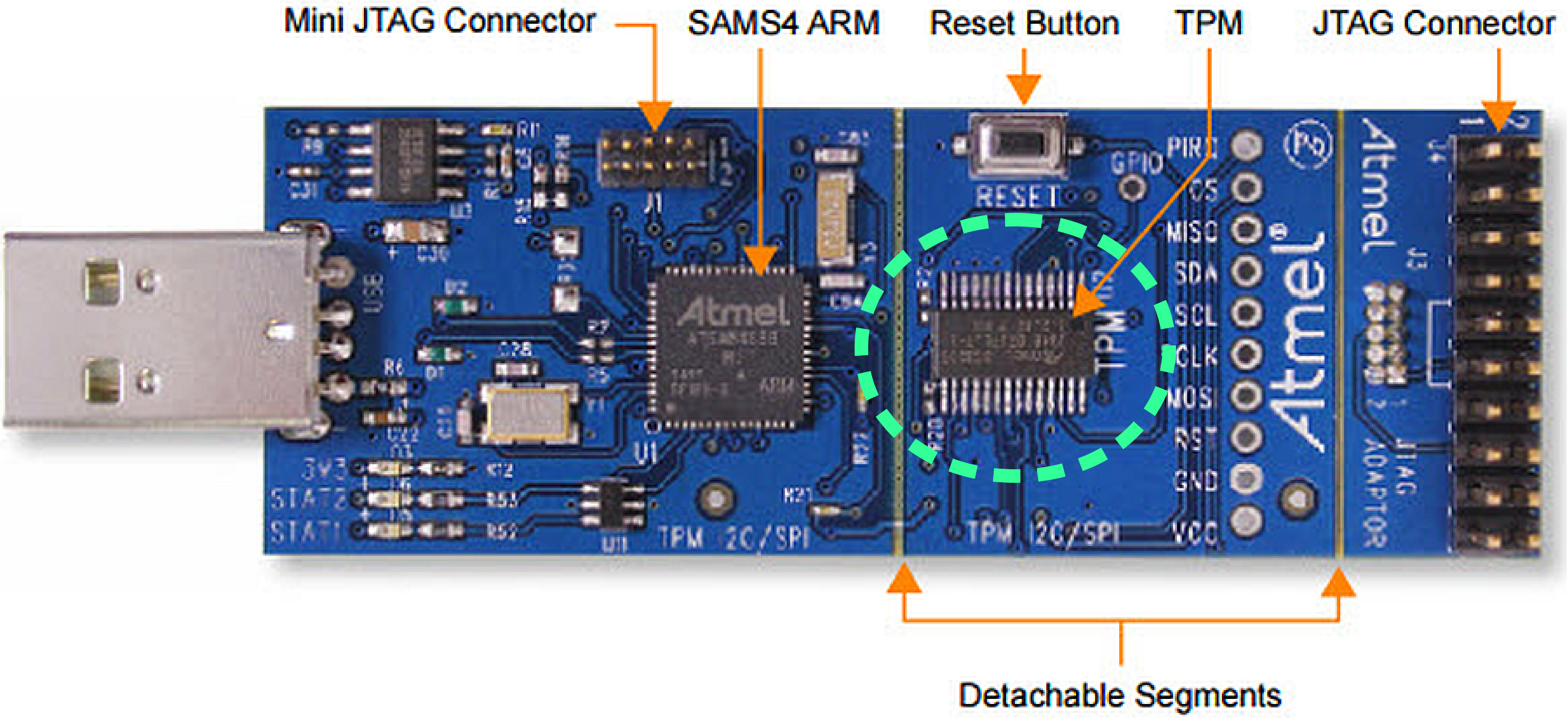
# Protecting System Integrity

- Generally requires more feature-rich processors
- Use full disk encryption (FDE)
- “Encrypted at rest”



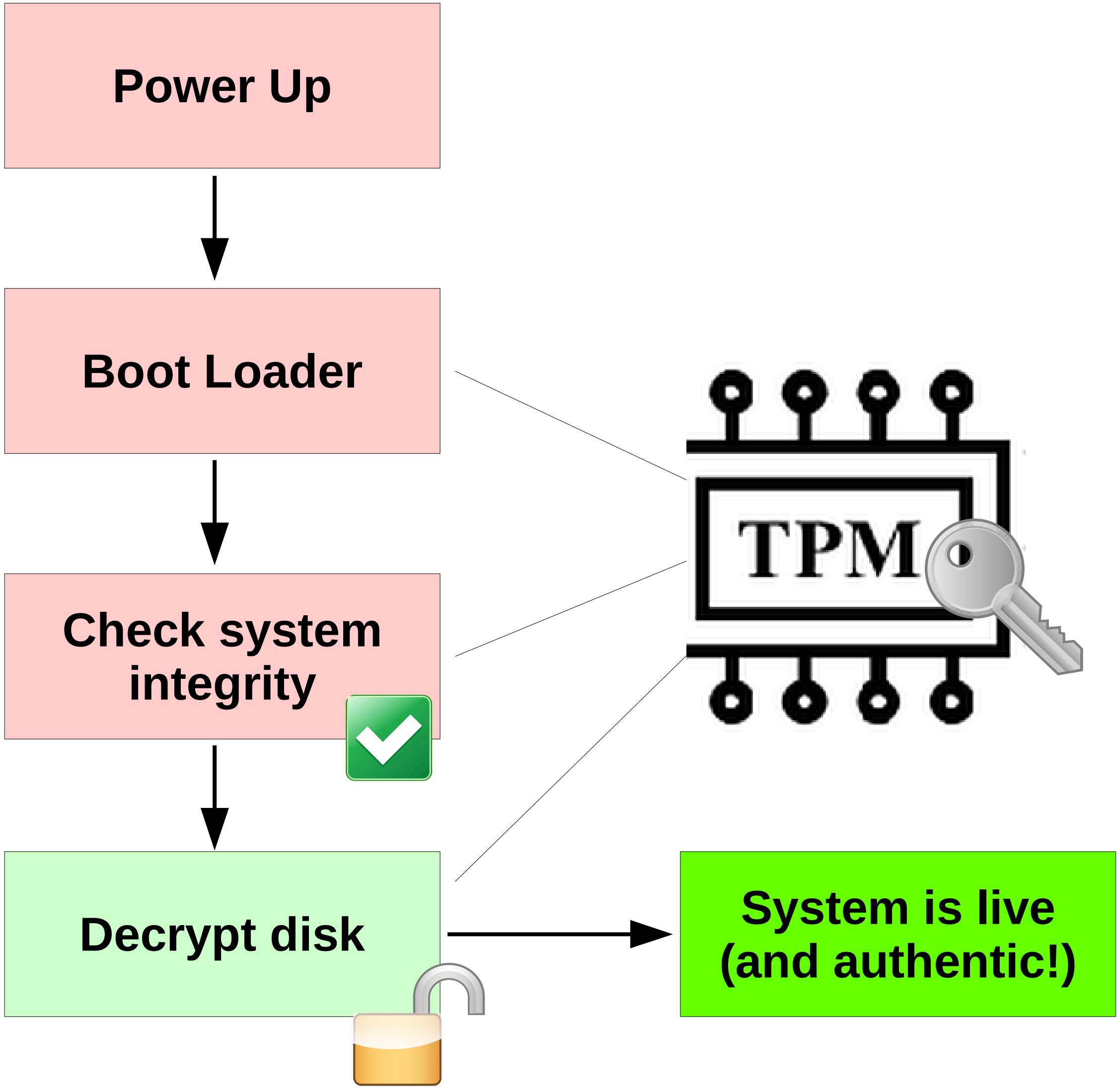
**But the key is exposed and readable, right?**

# Secure Cryptoprocessor (e.g. TPM)



Atmel TPM Development Kit (ARM)

# Secure Boot & Cryptoprocessor



Power Up

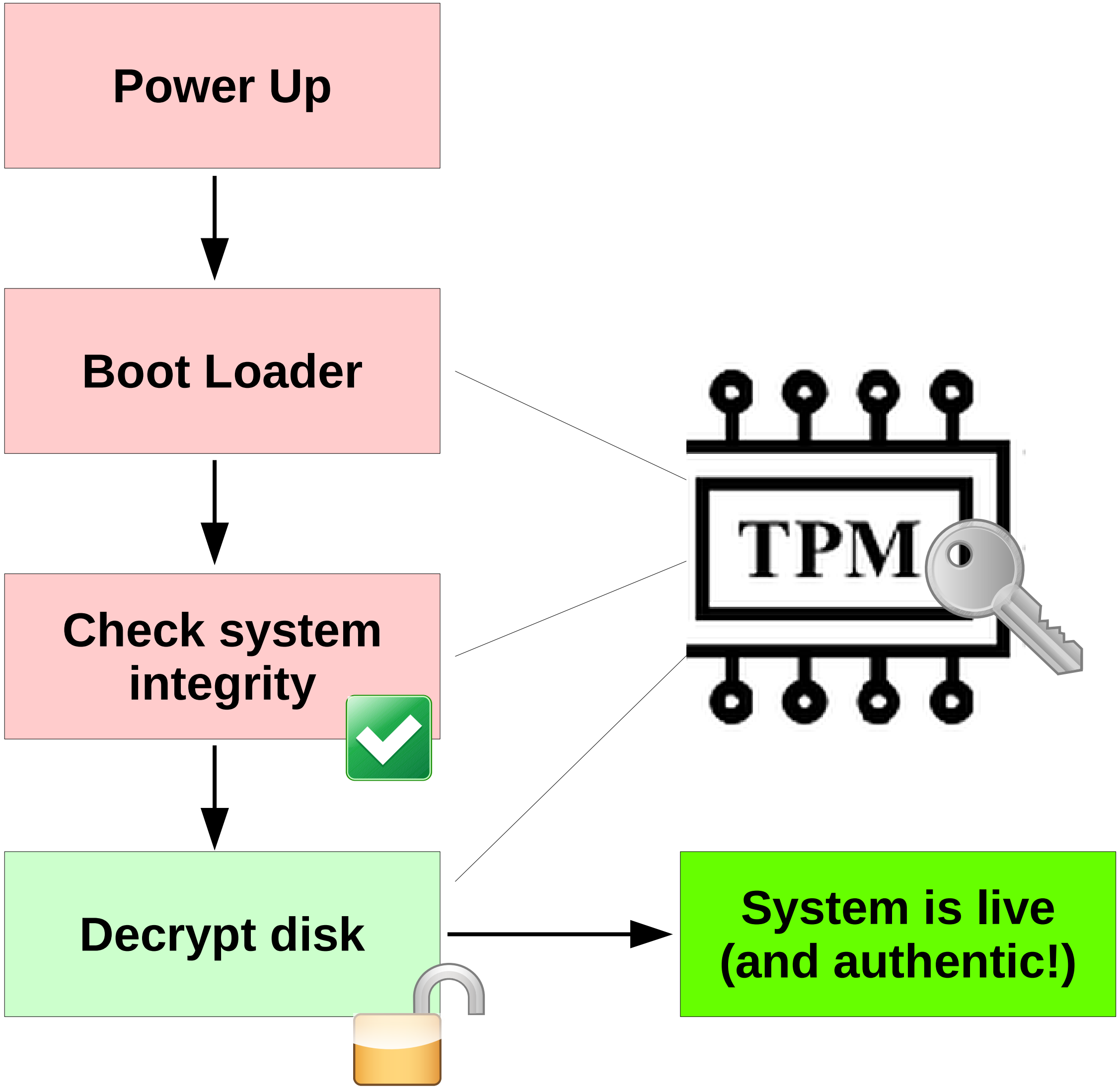
Boot Loader

Check system integrity

Decrypt disk

System is live (and authentic!)

# Secure Boot & Cryptoprocessor



Power Up

Boot Loader

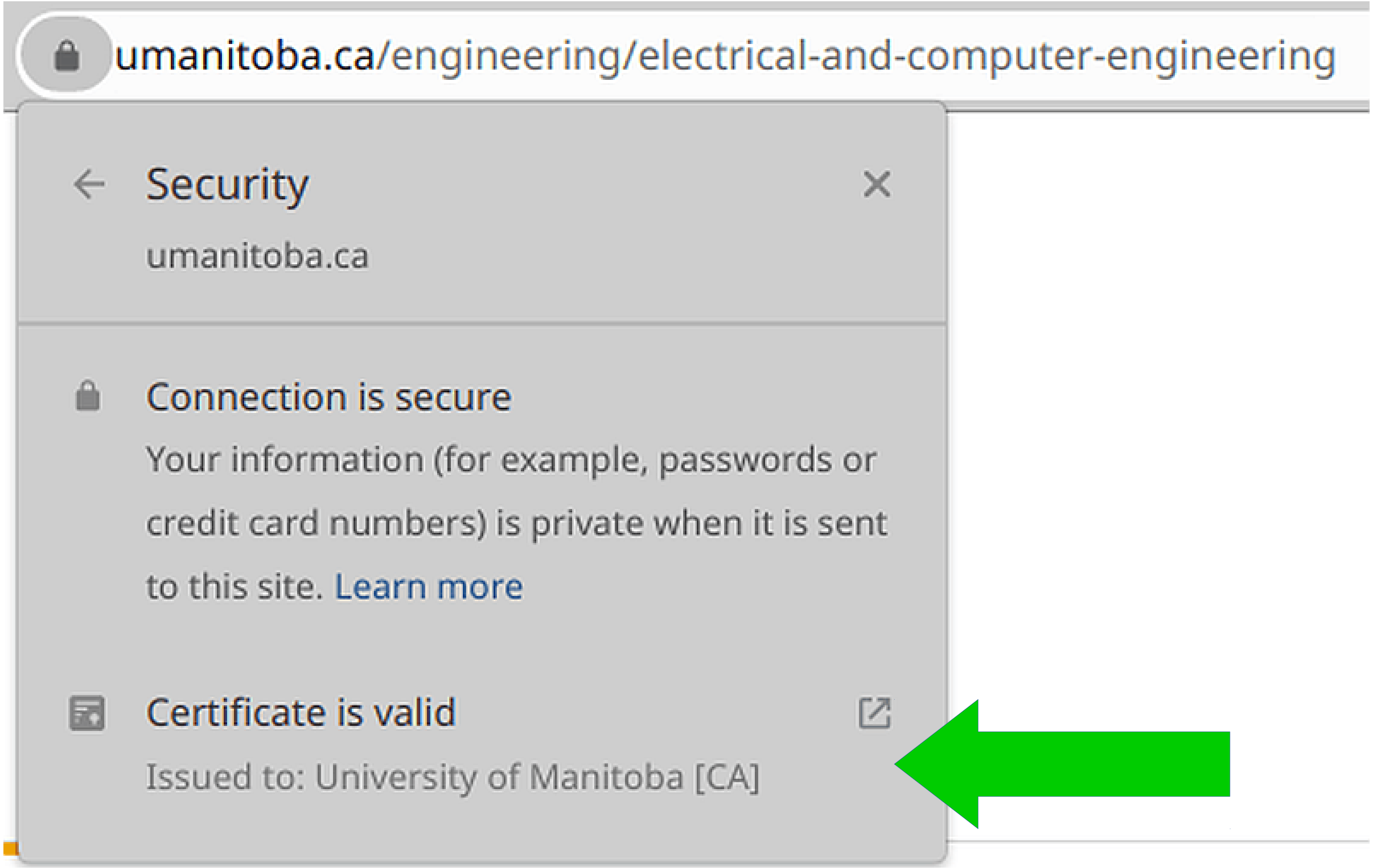
Check system integrity

Decrypt disk

System is live (and authentic!)

- Detects tampering
- Ensures integrity
- Protects the key
- “Root of Trust”

# SSL/TLS Certificates



umanitoba.ca/engineering/electrical-and-computer-engineering

Security

umanitoba.ca

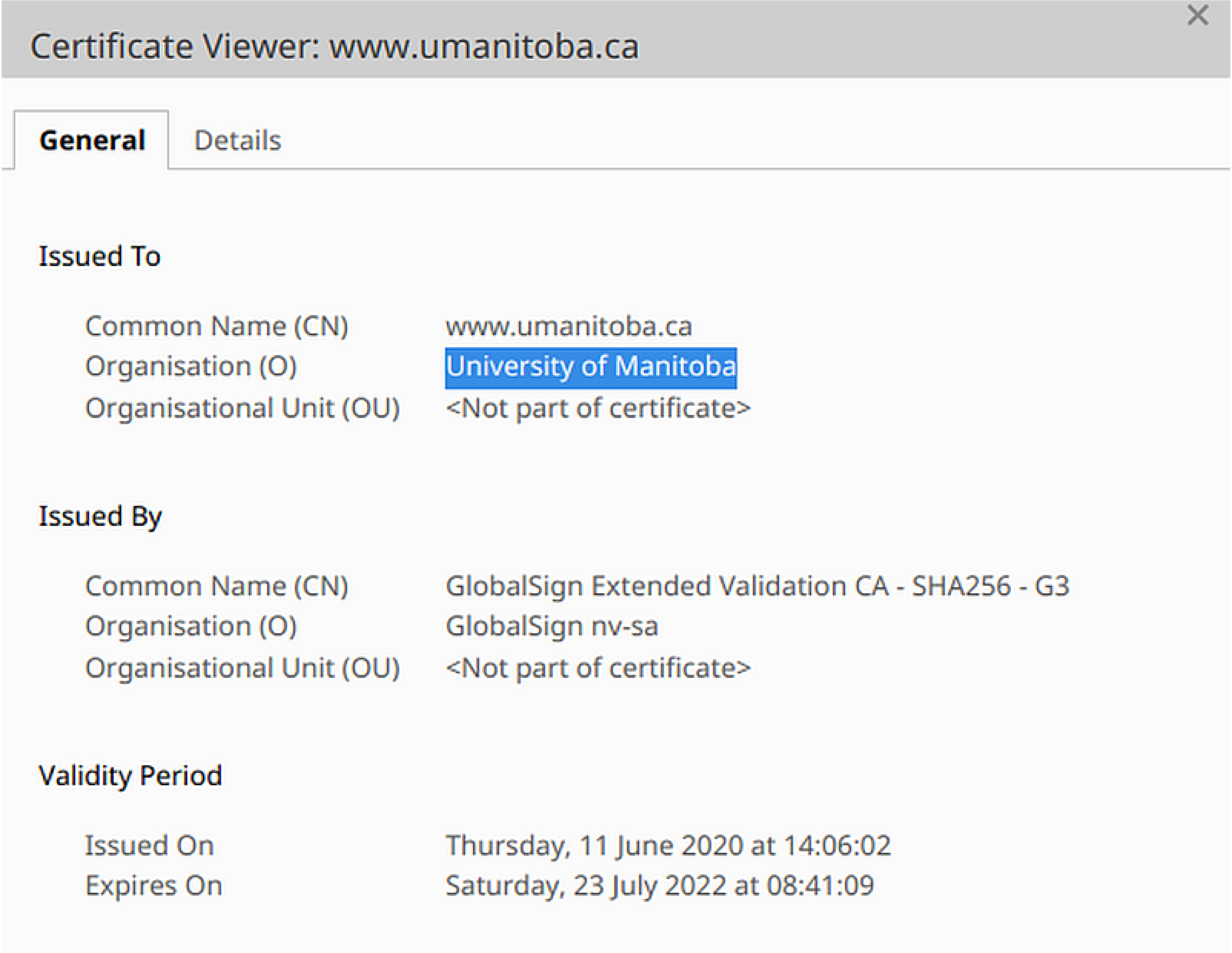
Connection is secure

Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

Certificate is valid

Issued to: University of Manitoba [CA]

# SSL/TLS Certificates



Certificate Viewer: www.umanitoba.ca

Issued To

| | |
|---|---|
| Common Name (CN) | www.umanitoba.ca |
| Organisation (O) | University of Manitoba |
| Organisational Unit (OU) | <Not part of certificate> |

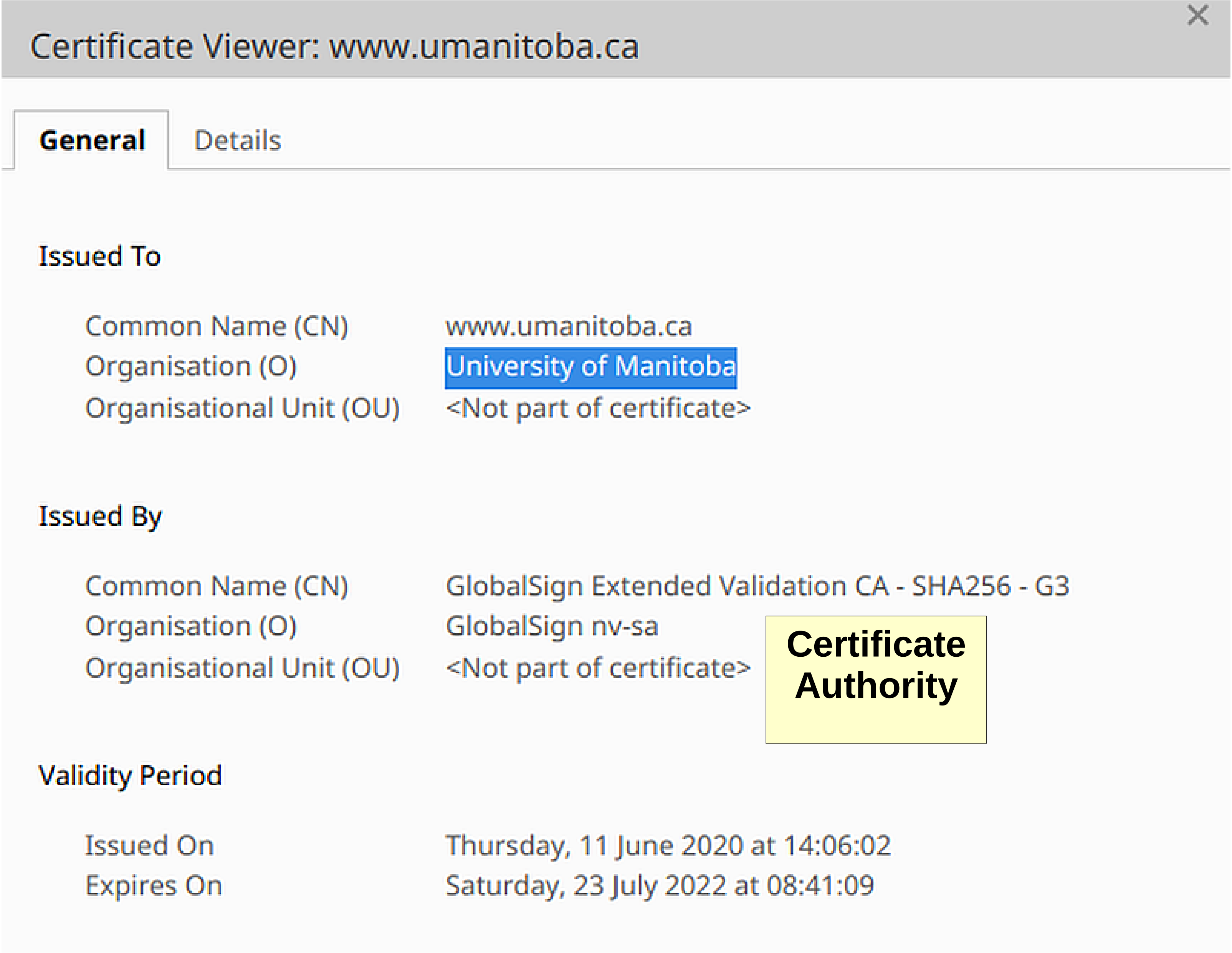
Issued By

| | |
|---|---|
| Common Name (CN) | GlobalSign Extended Validation CA - SHA256 - G3 |
| Organisation (O) | GlobalSign nv-sa |
| Organisational Unit (OU) | <Not part of certificate> |

Validity Period

| | |
|---|---|
| Issued On | Thursday, 11 June 2020 at 14:06:02 |
| Expires On | Saturday, 23 July 2022 at 08:41:09 |

# SSL/TLS Certificates



Certificate Viewer: www.umanitoba.ca

Issued To

| | |
|---|---|
| Common Name (CN) | www.umanitoba.ca |
| Organisation (O) | University of Manitoba |
| Organisational Unit (OU) | <Not part of certificate> |

Issued By

| | |
|---|---|
| Common Name (CN) | GlobalSign Extended Validation CA - SHA256 - G3 |
| Organisation (O) | GlobalSign nv-sa |
| Organisational Unit (OU) | <Not part of certificate> |

**Certificate Authority**

Validity Period

| | |
|---|---|
| Issued On | Thursday, 11 June 2020 at 14:06:02 |
| Expires On | Saturday, 23 July 2022 at 08:41:09 |

# Certificate Authorities (CA)

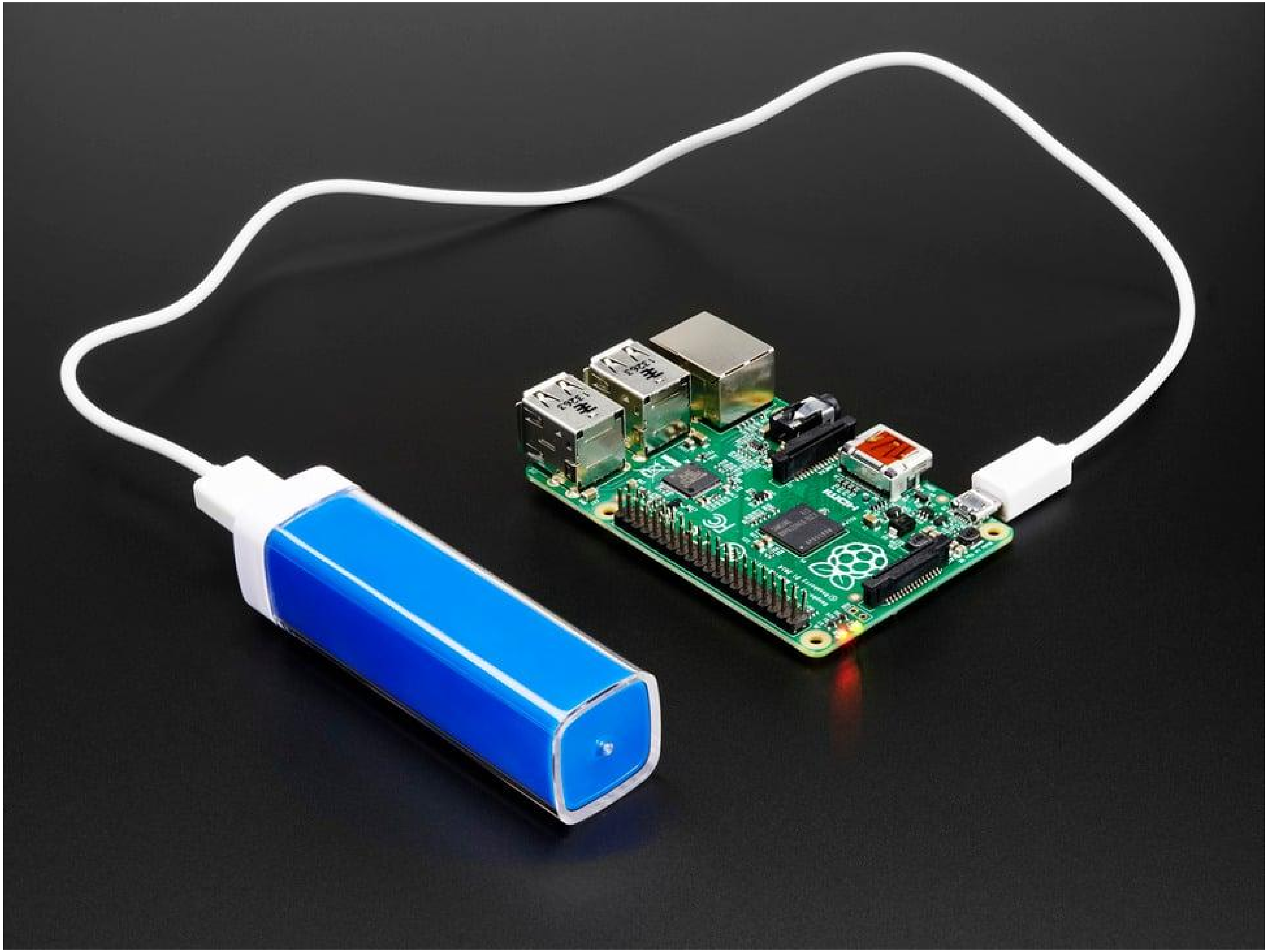
- Another special concern for embedded
- SSL/TLS certificates are verified against CAs
- But IoT devices can’t get certs from actual root CAs
- What happens when a client checks the cert?
  - e.g. user visits HTTPS (web) server

# CA Solutions

- Create your own Certificate Authority (using OpenSSL)
- Install your own “root” CA cert on every device
- Also called a Private CA
- Each of *your* devices can then recognize each other
  - But someone else (e.g. smart phone) will still get an “invalid cert”

# Low-Power Design

# Limited Power (battery)



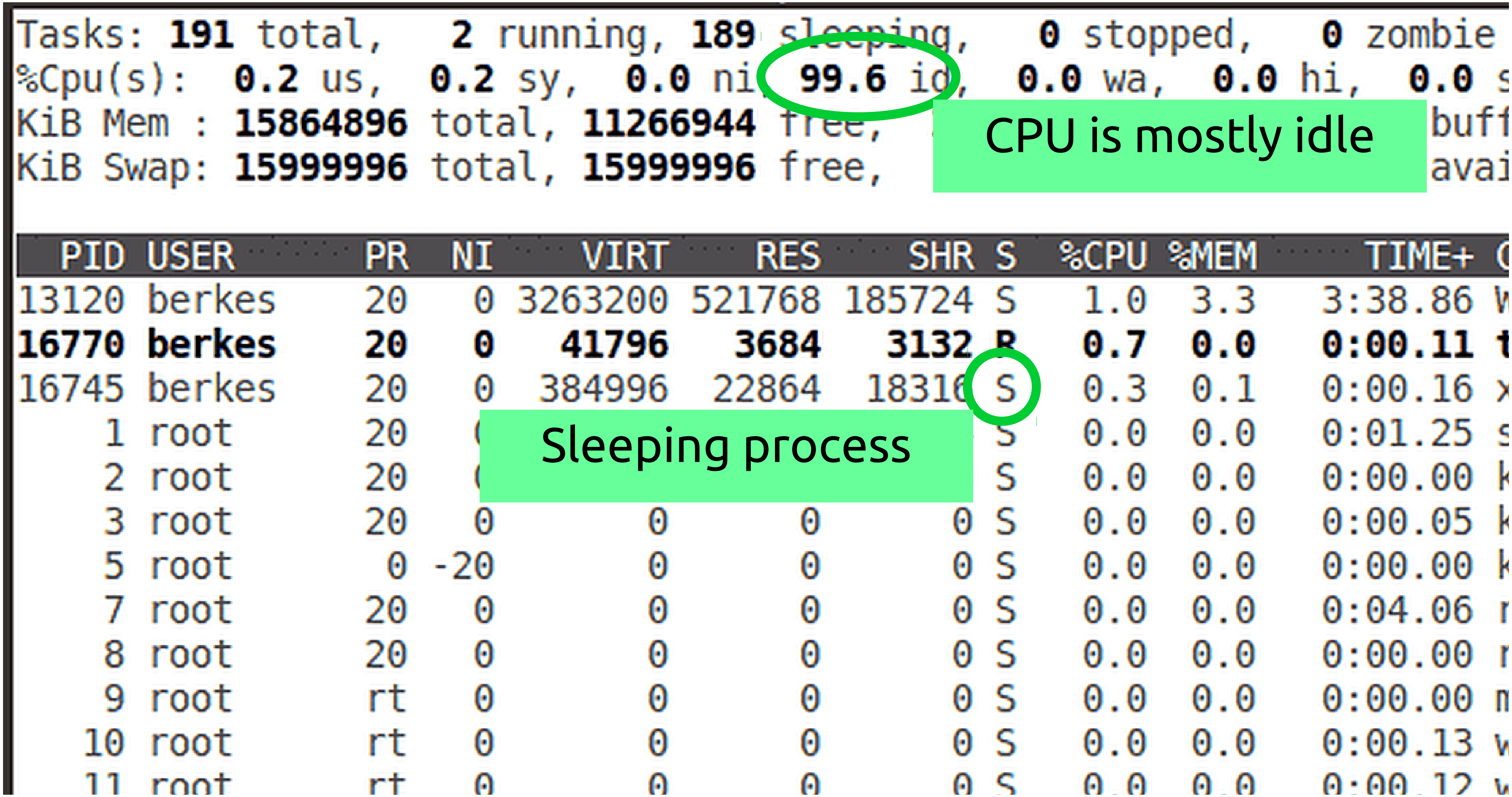
# Best Practices

- Turn off unused interfaces
  - USB, HDMI video, Wi-Fi, Bluetooth, etc.
- Idle is good!
  - Read sensors intermittently (low sample rate)
  - Allows CPU to save power
- System-wide sleep/suspend... maybe

# Idle is Good!

- Wait in the right way; avoid “busy wait”
- Can suspend and wait for event (system-specific)
  - UNIX signals, timers
  - External inputs
- Sample external sensors at low rates
  - Sleep in between

# ‘top’ gives clues



```
Tasks: 191 total,   2 running, 189 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.2 sy,  0.0 ni, 99.6 id,  0.0 wa,  0.0 hi,  0.0 s
KiB Mem : 15864896 total, 11266944 free,                      buff
KiB Swap: 15999996 total, 15999996 free,                      avai

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ C
13120 berkes    20   0 3263200 521768 185724 S   1.0  3.3   3:38.86 W
16770 berkes    20   0   41796   3684   3132 R   0.7  0.0   0:00.11 t
16745 berkes    20   0  384996  22864  18316 S   0.3  0.1   0:00.16 x
    1 root      20   (                       S   0.0  0.0   0:01.25 s
    2 root      20   (                       S   0.0  0.0   0:00.00 k
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.05 k
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 k
    7 root      20   0       0      0      0 S   0.0  0.0   0:04.06 r
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 r
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 m
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.13 w
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.12 w
```

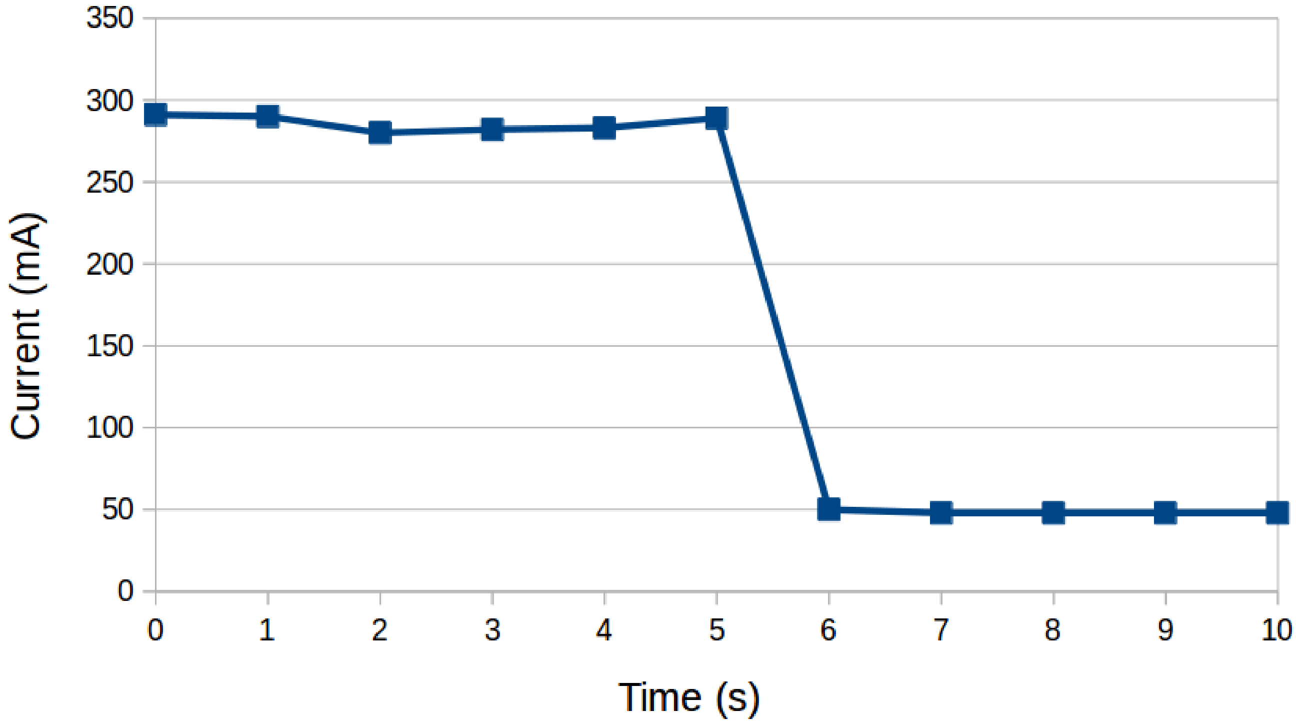
CPU is mostly idle

Sleeping process

# System-wide sleep/suspend

- Support varies by embedded system

```
$ cat /sys/power/state
freeze standby mem disk
$ echo standby > /sys/power/state
```



# Wi-Fi Design Considerations

- Some systems automatically go into power-saving
- The Wi-Fi interface might sleep
  - Latency/dropped packets
  - Connections might *break*

# Wi-Fi Design Considerations

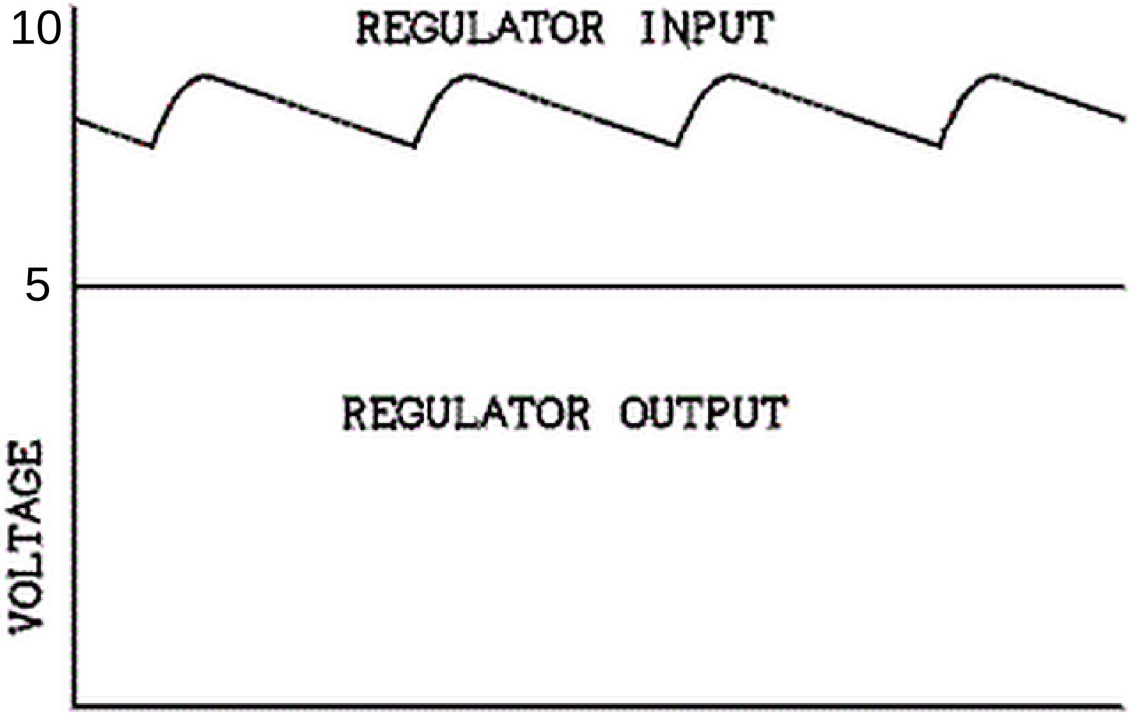
- Design a robust communication protocol
  - Beware that wireless connections may break
  - Don’t assume Wi-Fi is continuously connected

- You might want to **turn off** Wi-Fi power savings

# *Extra slides:*

# *Power Supplies & Batteries*

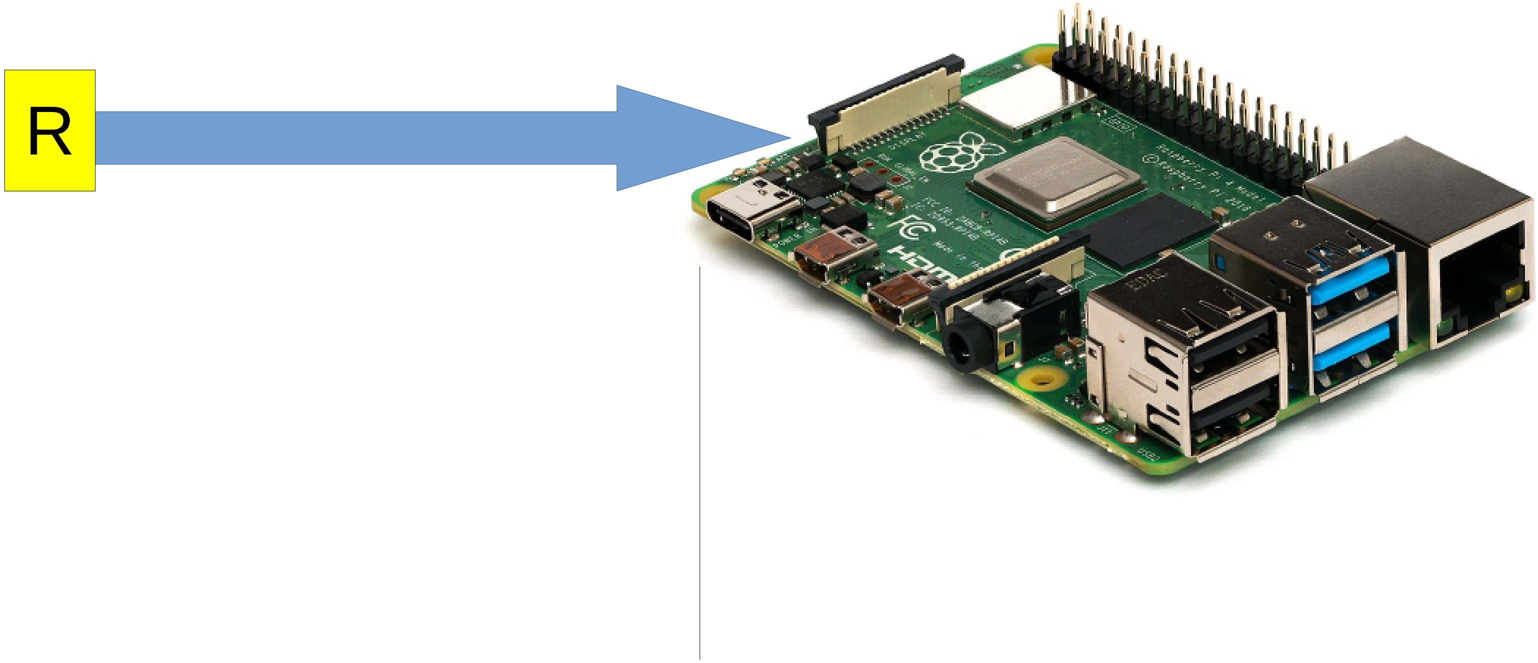
# Voltage Regulation



Uneven input

Steady output

# *Where* is regulator? (Pi)

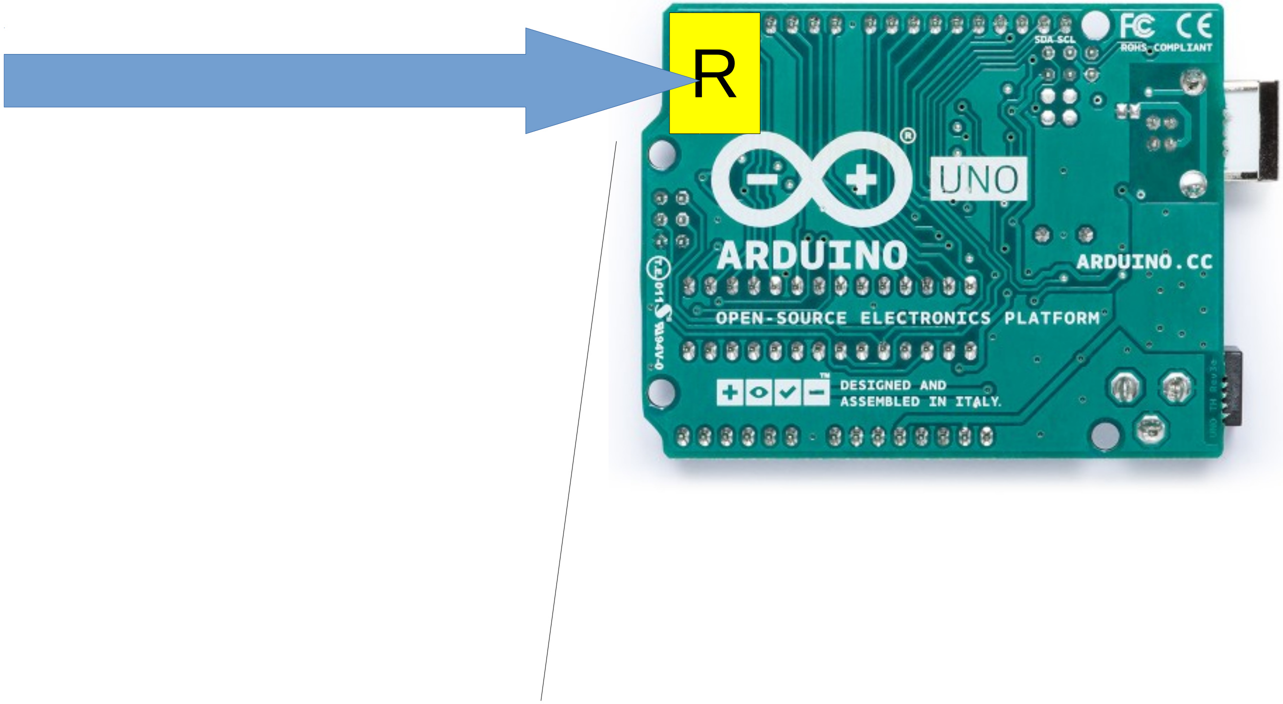


R

Expects 4.8 – 5.2 V

Input must be **regulated**

# *Where* is regulator?


R
UNO
ARDUINO
ARDUINO.CC
OPEN-SOURCE ELECTRONICS PLATFORM
DESIGNED AND ASSEMBLED IN ITALY.

Expects 7 – 12 V

Board has regulator

# Current Draw

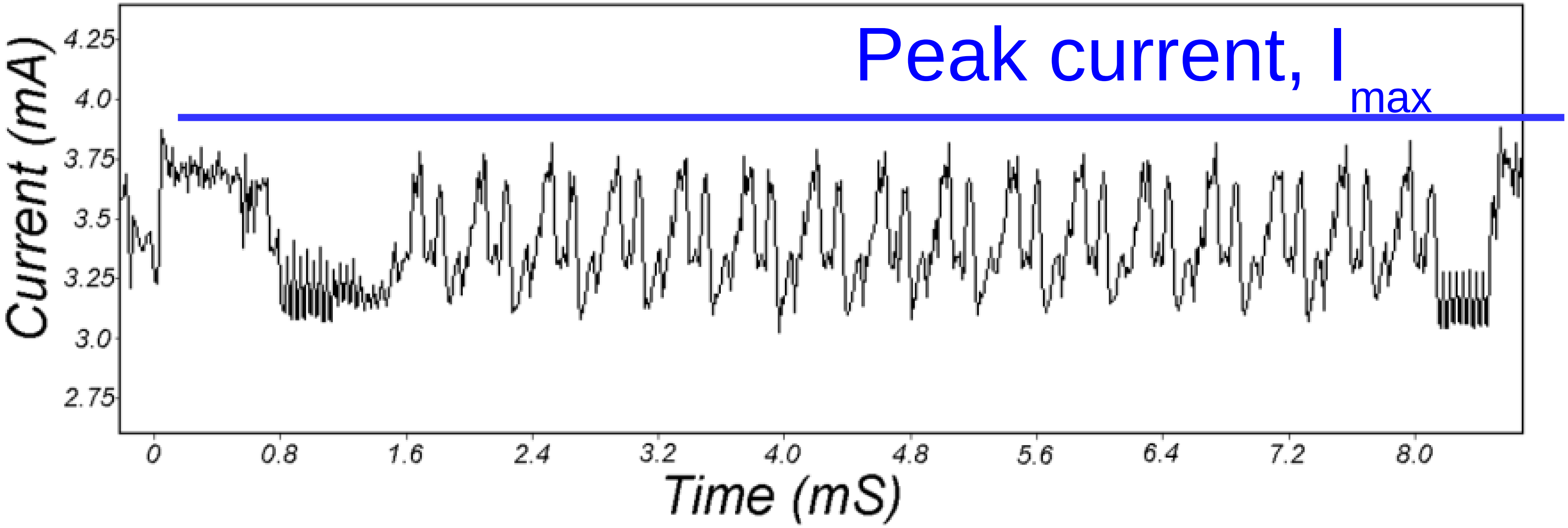
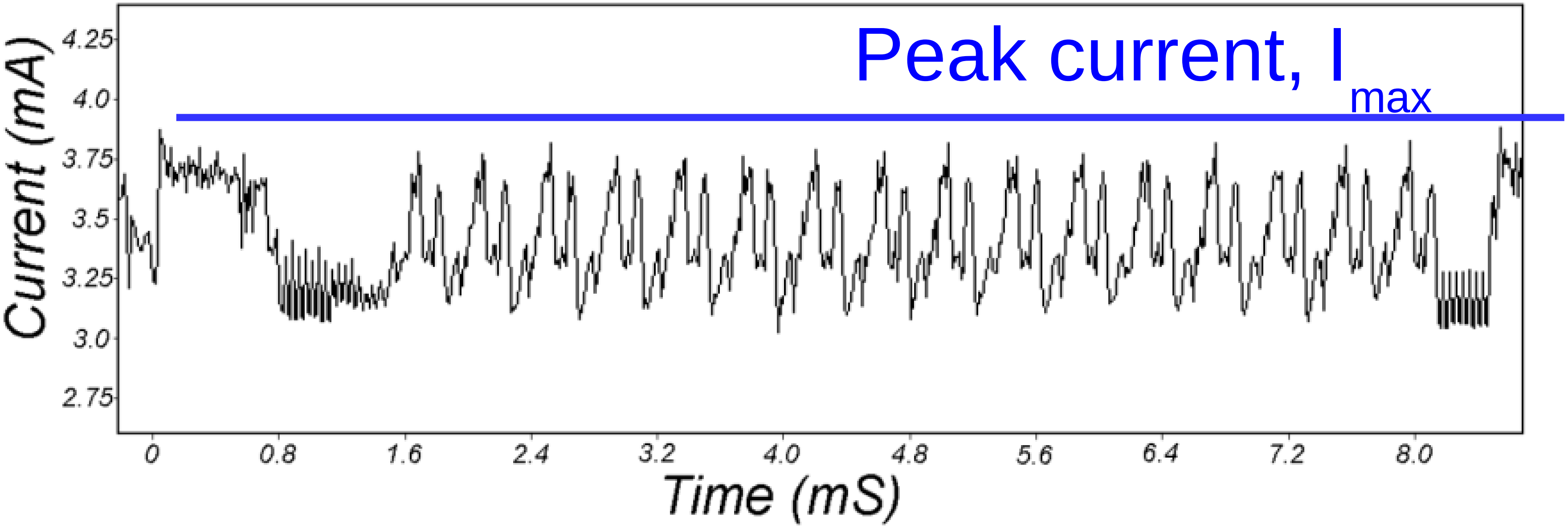


Image from *Differential Power Analysis* (Kocher, Jaffe, Jun)

# Current Draw



- Current can spike; plan conservatively
- Power supplies are rated for max current
- If power supply can’t keep up, **device malfunctions**

Image from *Differential Power Analysis* (Kocher, Jaffe, Jun)

# Battery Packs

## (USB mobile phone chargers / power banks)

# How to Calculate

- Determine $I_{max}$ and ensure supply can provide it
- Determine $I_{avg}$
- Learn battery’s milliamp-hours (mAh) rating
- Caveats
  - Voltage regulators lose power
  - Batteries age
  - Power packs often over-state mAh

# Rule-of-Thumb Adjustments

- Reduce battery mAh rating by 30%
  - Accounts for regulator loss
- Plan for 50% empty battery
  - Accounts for aging and safety margin

# Example Calculation

- Raspberry Pi with $I_{avg}$ = 600 mA and $I_{max}$ = 1200 mA
- USB 5V Mobile Charger, “5000 mAh”, max 2500 mA
- Check

  $I_{max}$ < 2500 mA (ok)
- Adjust battery capacity down to 3500 mAh
- Time on battery = 3500 mAh / 600 mA = 5.8 hours
- Plan for 50% battery
  - **Conservative answer is 2.9 hours**