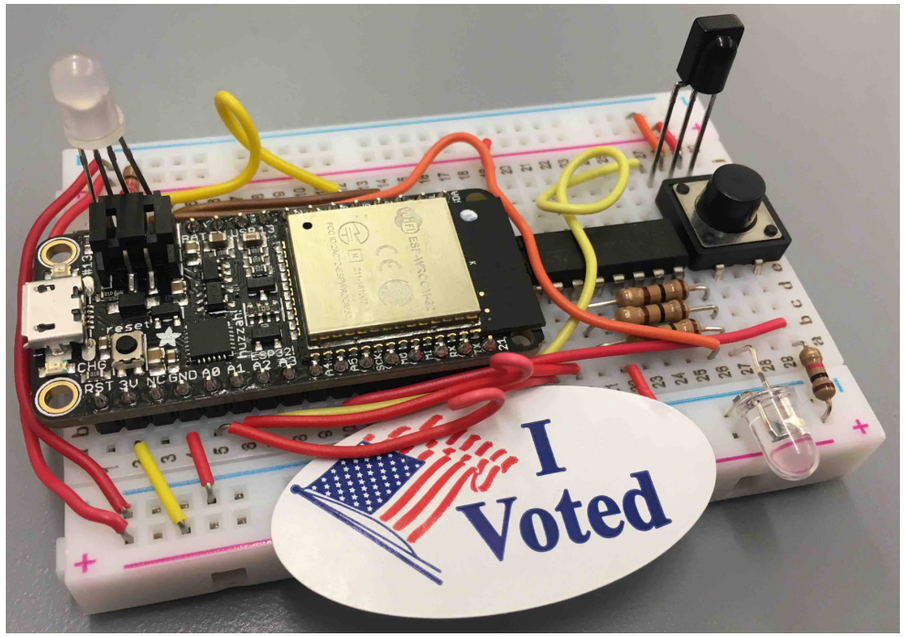
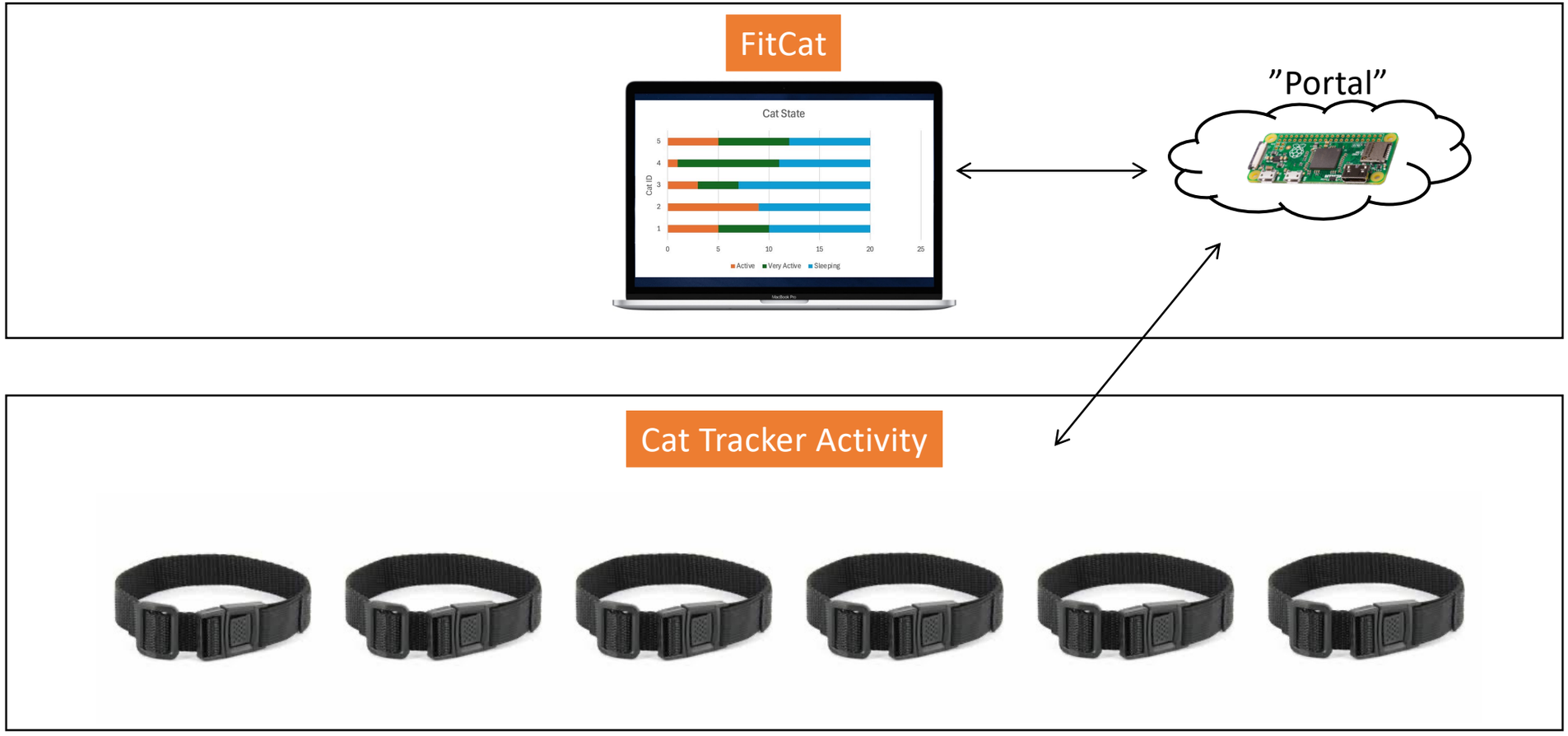


BU-EC444

Fall 2024
Prof. Little

Review of Quest: FitCat – Activity Tracking for Cats

Quest 3: FitCat Concept



Engages multiple Cat Trackers (and Cats)

Quest 3: FitCat

Any remote client

FitCat

“Cloud Server”

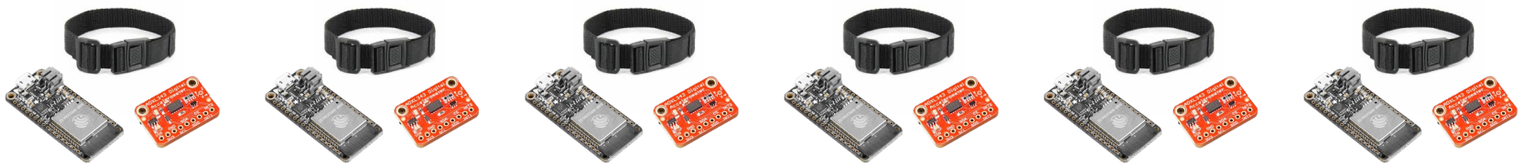
”Portal”

Node.js

Cat Kennel

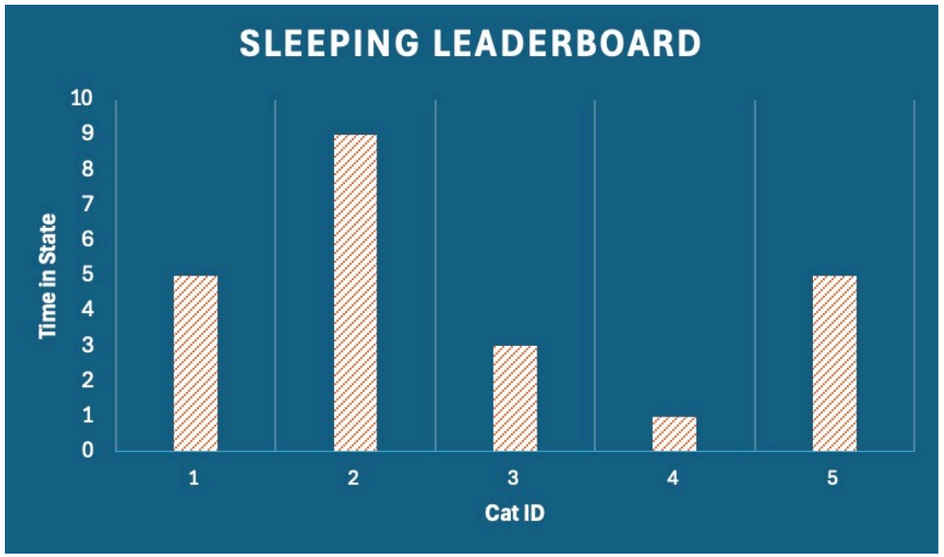
ESP32

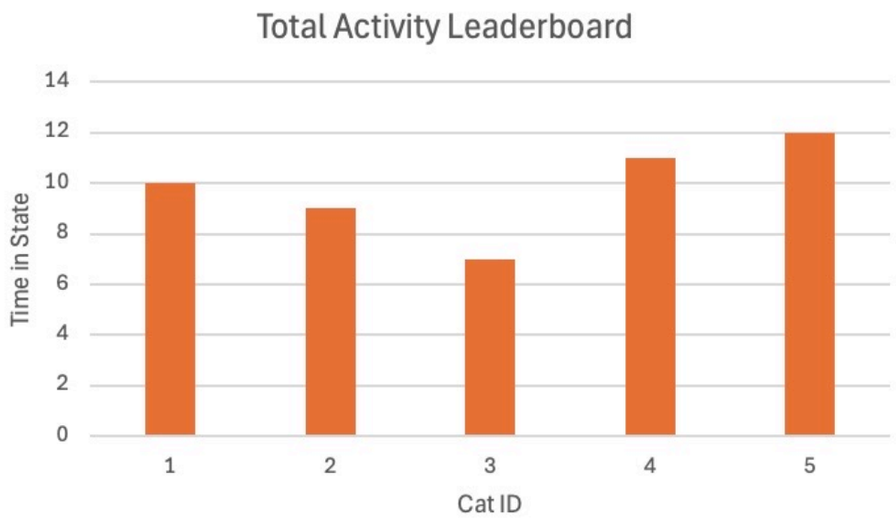
Cat Trackers

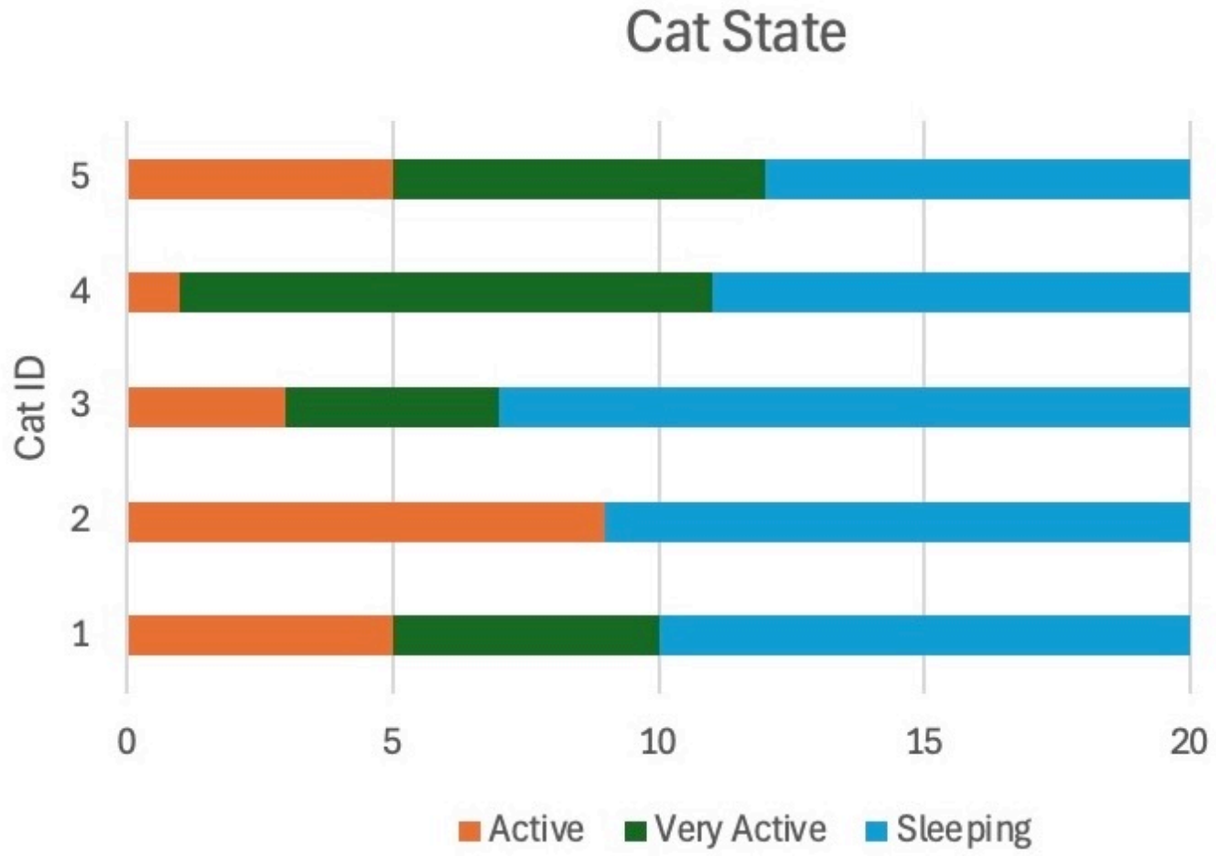


Pi Cam

Quest 3: FitCat – some ideas for data display

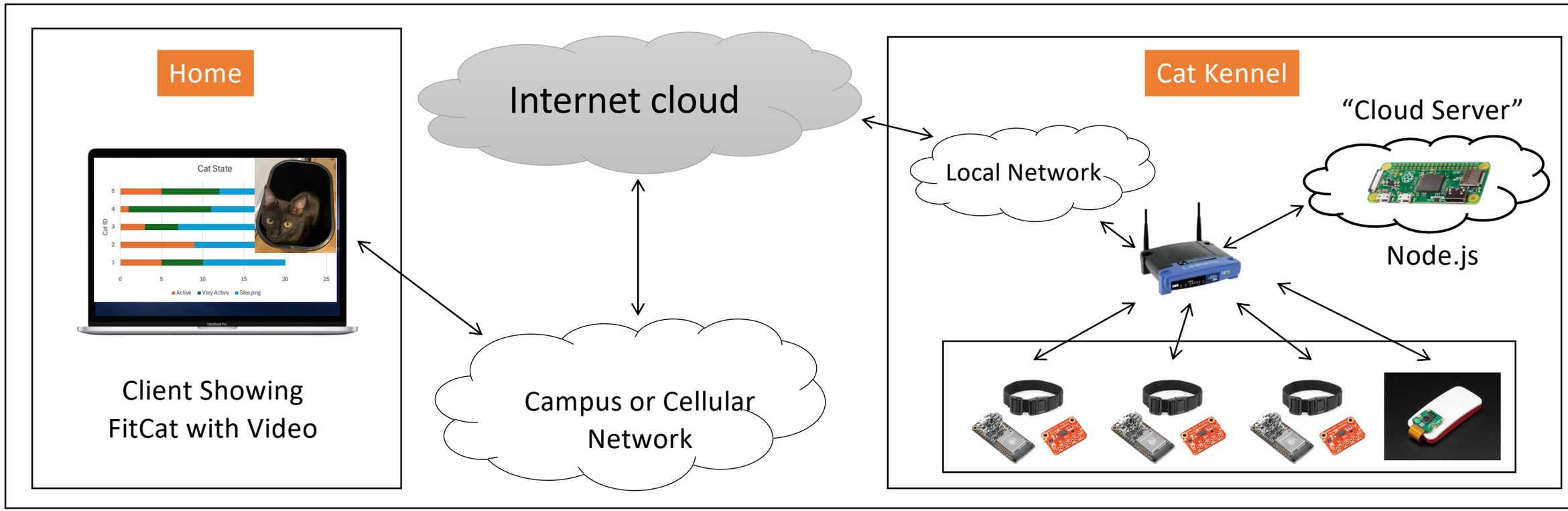






Nice to have: leader floats to top

Quest 3: Hosting the FitCat Social Platform



Key takeaways:

- Data aggregation
- Remote IP control
- Wireless embedded system
- Webcam

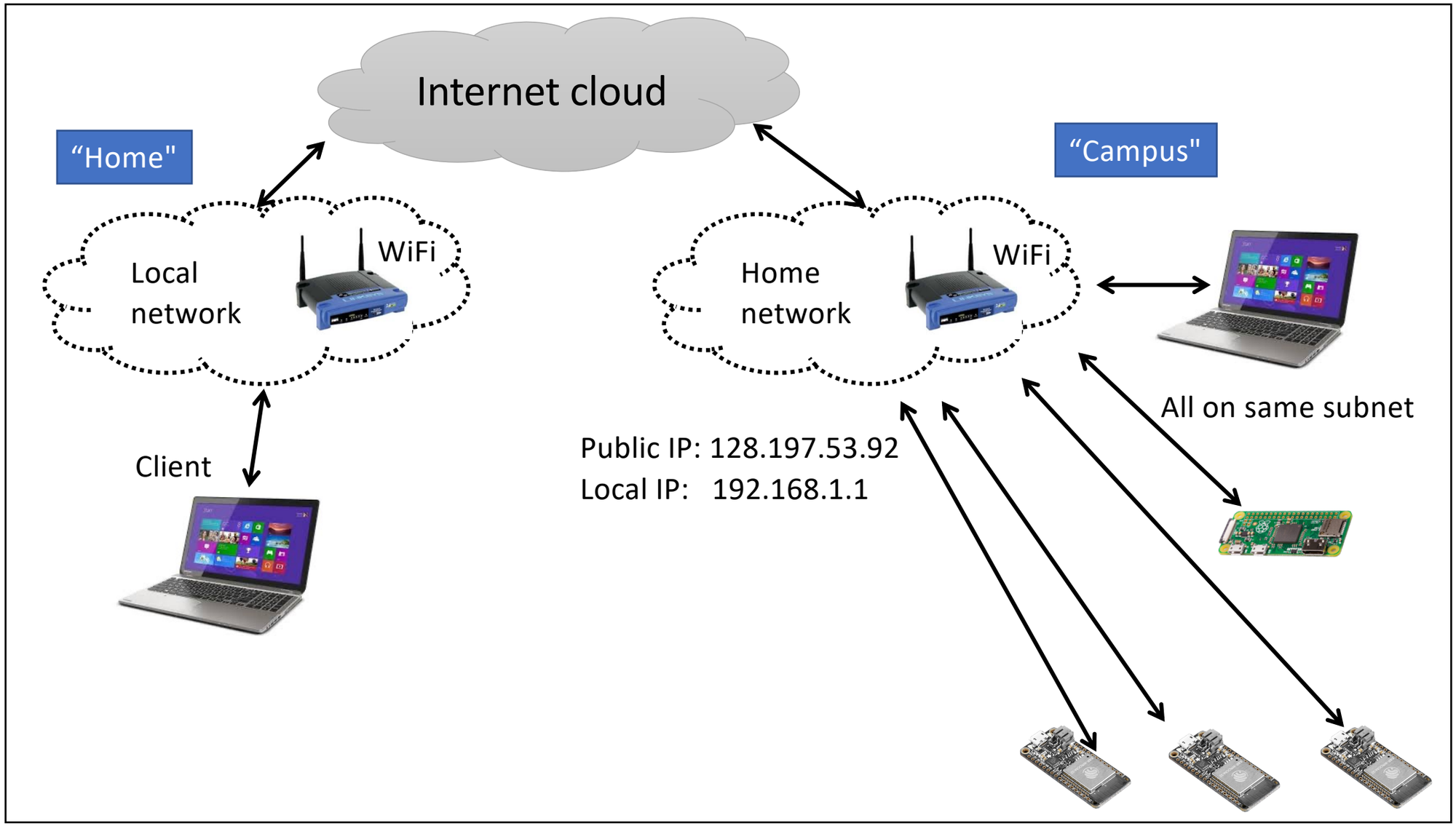
- Multiple ESPs
- Nodejs on pi
- Video from pi added
- WiFi
- Hook to future database

Quest 3 Cluster

- Router Configuration (team)
- DDNS on Router (team)
- WiFi-ESP-Station
- Client and Server with UDP
- Raspberry Pi and Node.js
- Pi Web Cam

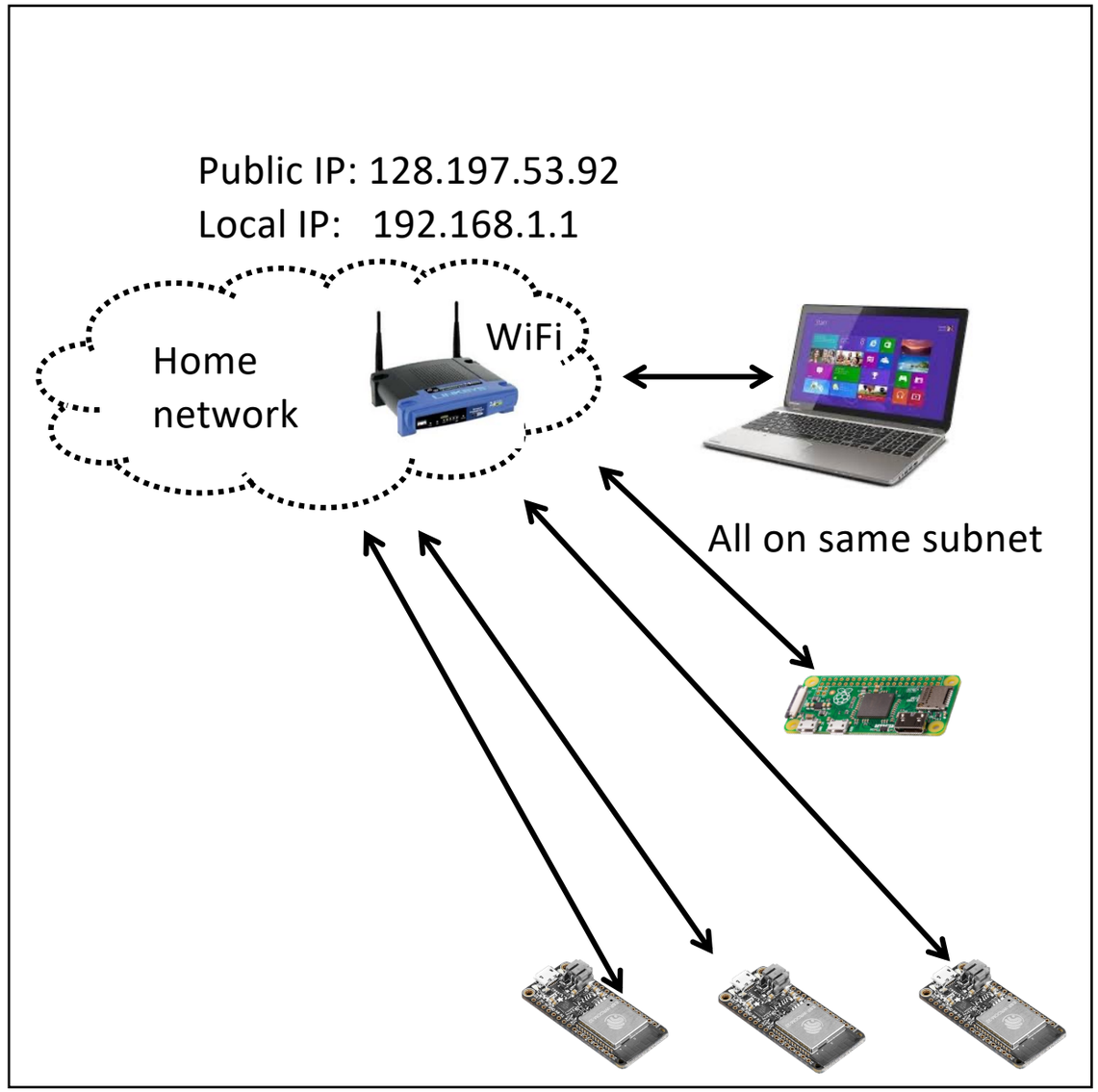
Router Configuration

Typical Wireless Setup



Typical Wireless Setup

- Router serves to create a **local subnet with its own IP address group**
- (Different from a **switch** which just **replicates ports** on same subnet)
- Can serve as firewall, limiting types of traffic
- Supports many different functions (VPNs, bit torrent etc.)
- Web access typically at :80 or :8080
- DHCP – **assigns local IP address** to local machines



Two router types – WRT54GS and E1200 v2

Router details

- WAN port
- Local ports
- Reset button
- Antennae

- Typical access address of 192.168.1.1
- UID: root, PWD: admin

Flash the router firmware using the existing admin interface of the router.

Attach **your computer** to one of the **LAN ports**

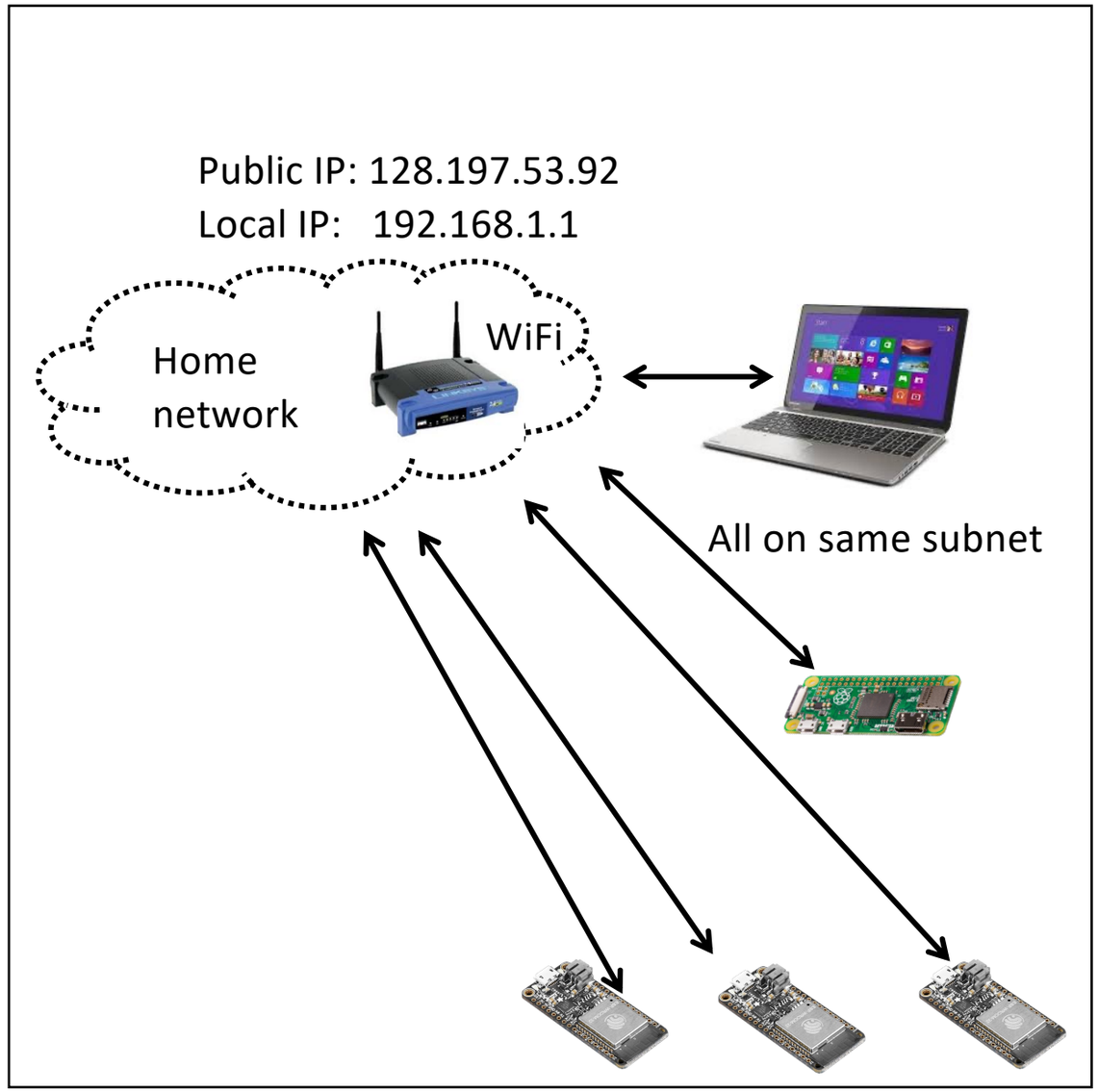
Attach **LAN port** to a **campus LAN port**

Power up. Press and hold reset button for 5s to reset password

Log in with default credentials

Typical Wireless Setup: Configuration Settings

- Local subnet of 192.168.1.1 (or similar)
- Enable local DHCP server (should be set up already)
- Set SSID with your group number (e.g., ‘Group01’)
- Set security to WPA/WPA2 Personal
- Set shared key to be ‘smartsys’
- Enable NTP server (separate step)
- Enable Dynamic DNS as per (separate step)

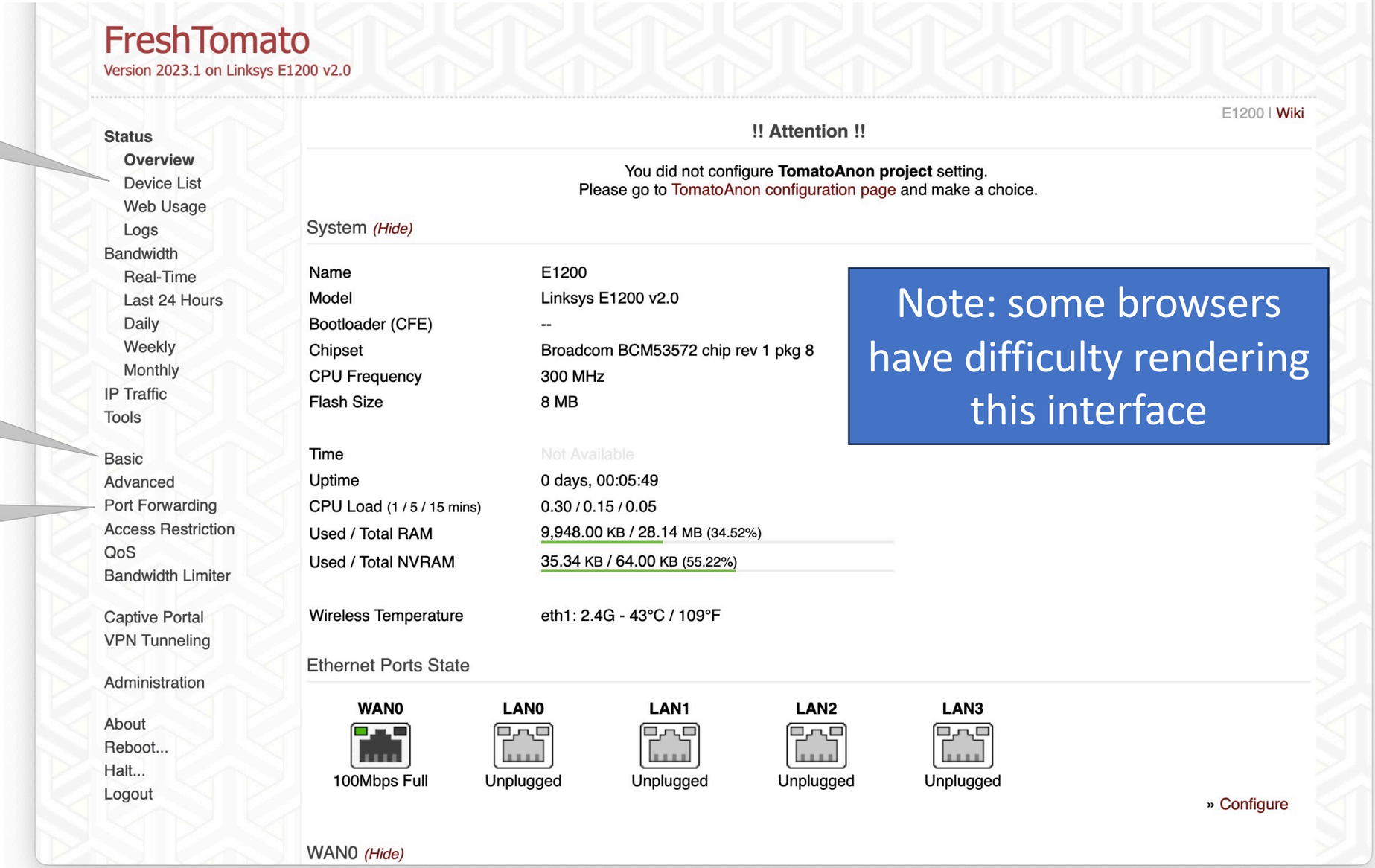


Opening screen (http://192.168.1.1 from LAN port or WiFi)

To identify devices on the local LAN

Most of the important configuration menus

Port forwarding



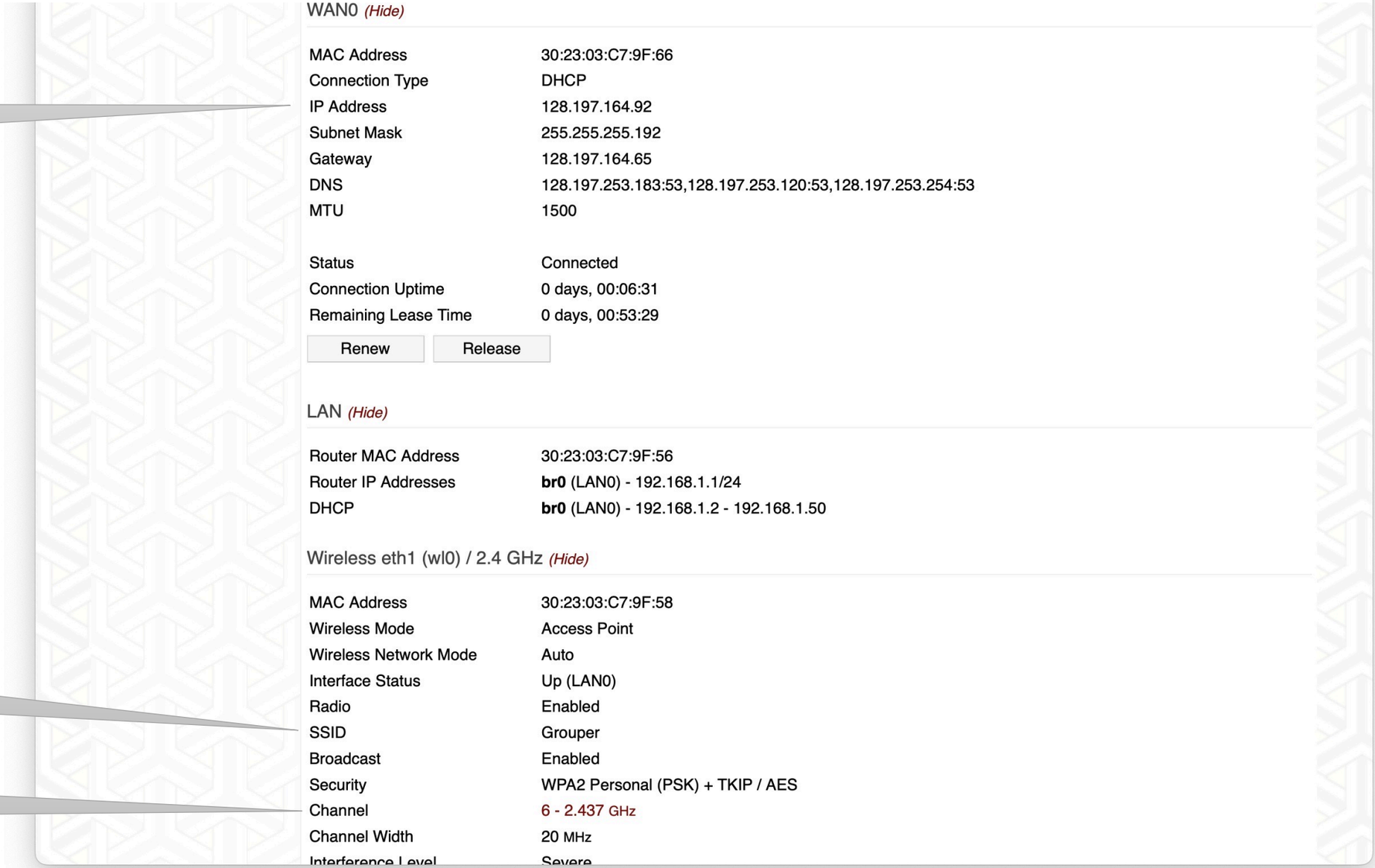
Note: some browsers have difficulty rendering this interface

More opening screen (status)

WAN address from ISP

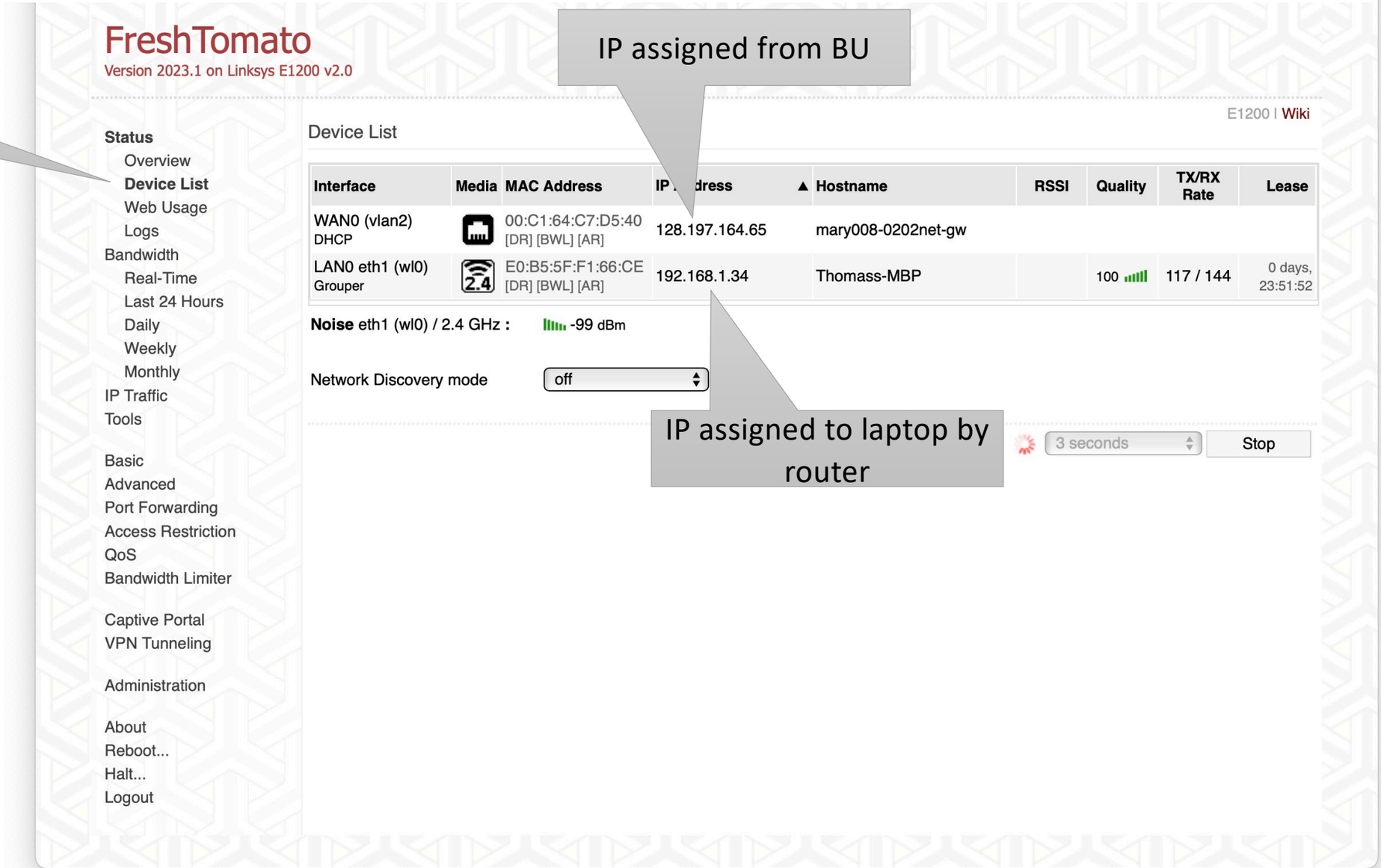
SSID

WiFi channel

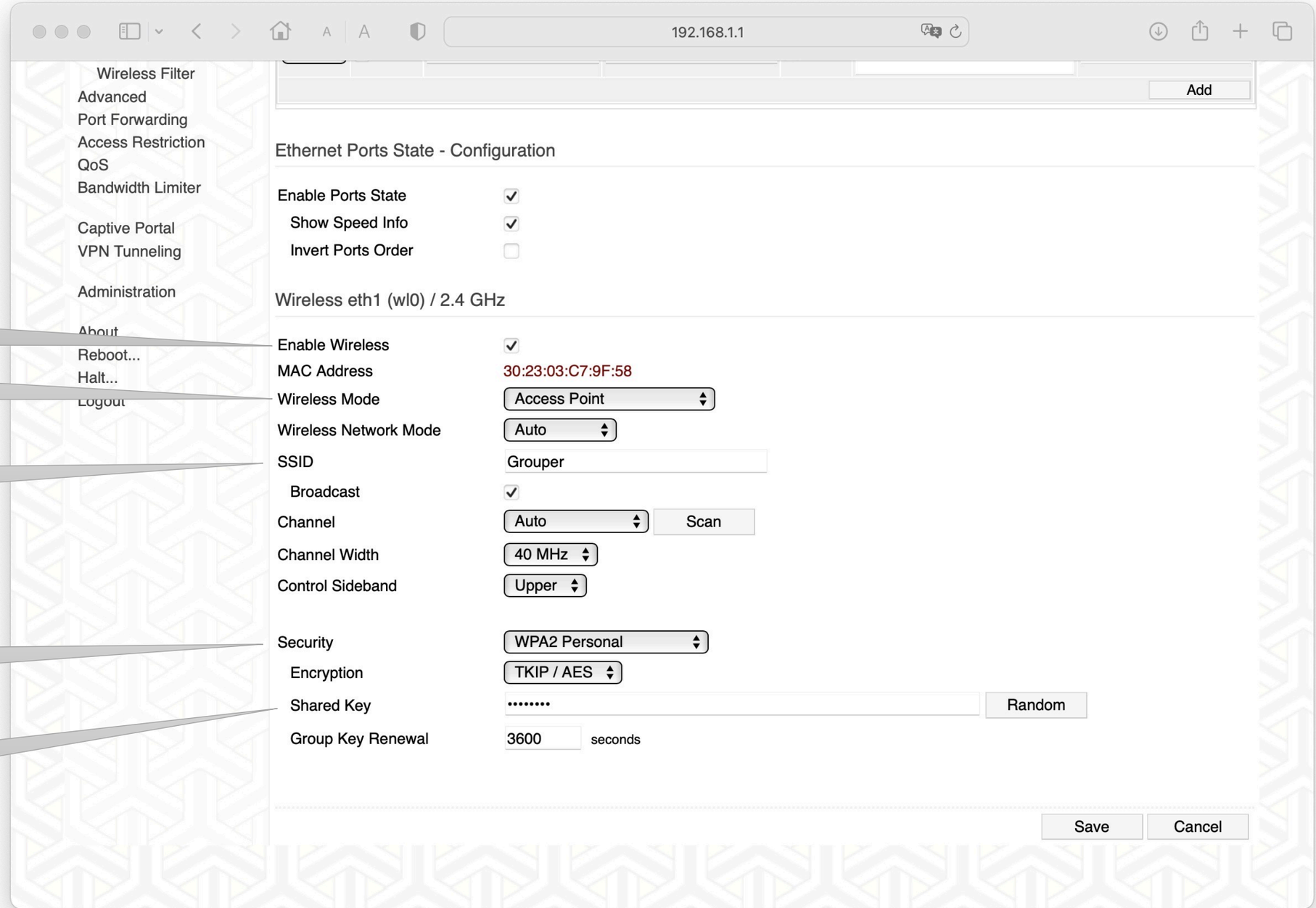


Device List

To identify devices on the local LAN



Basic -- wireless



Enable wireless

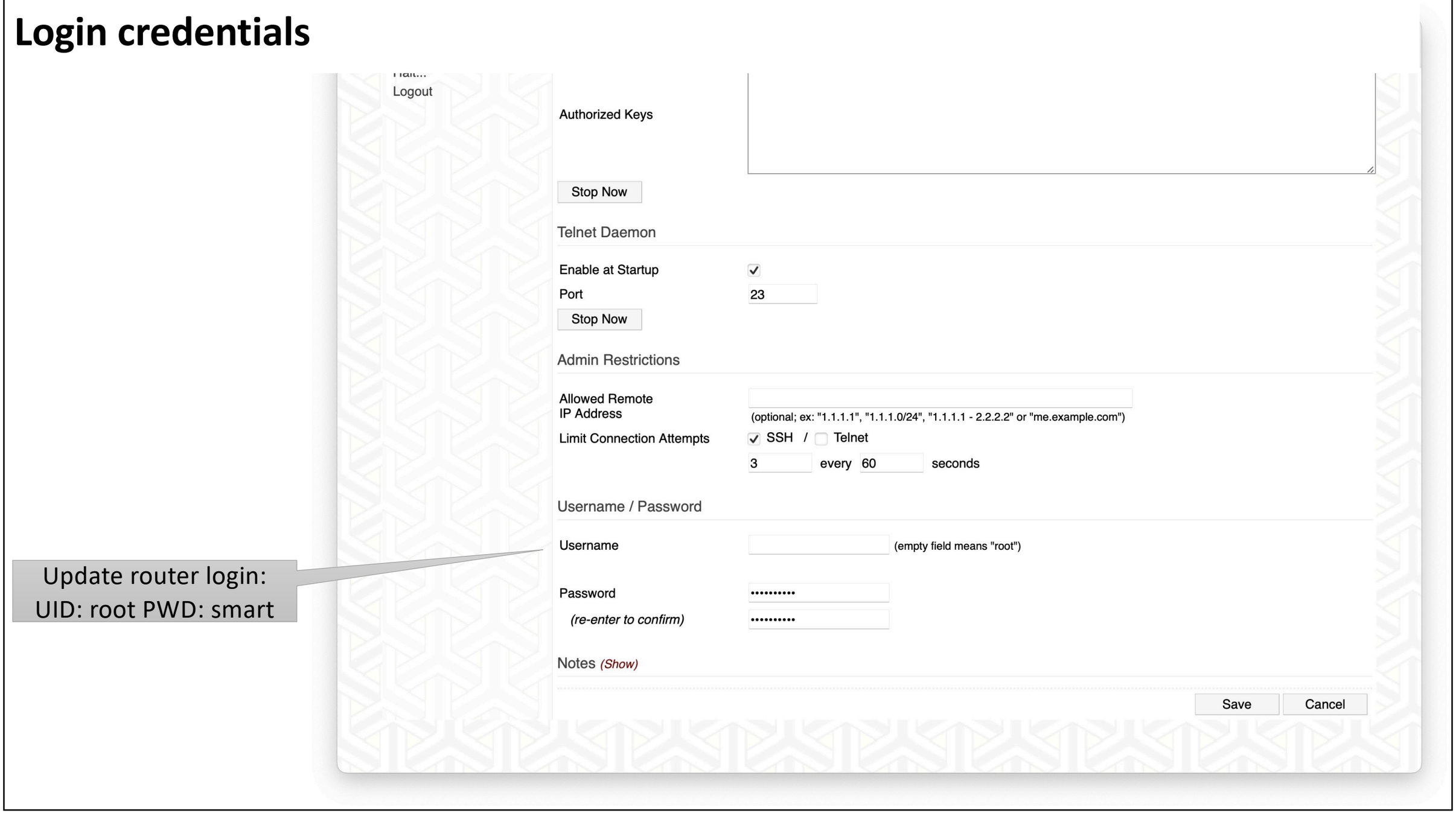
As Access Point

Set SSID

Set security

Set key to ‘smartsys’

Login credentials

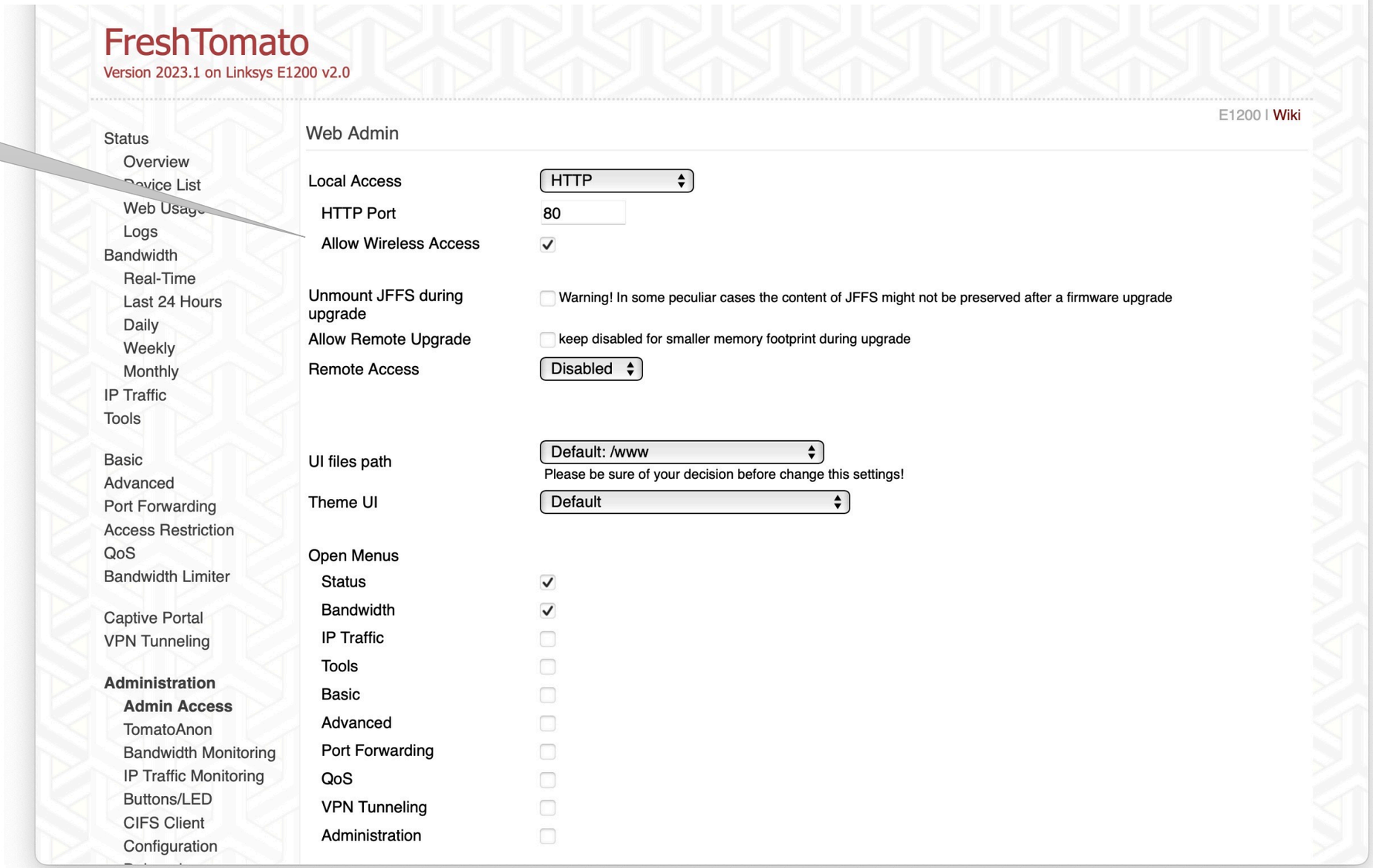


Enable router login access constraints

Enable wireless access on port 80

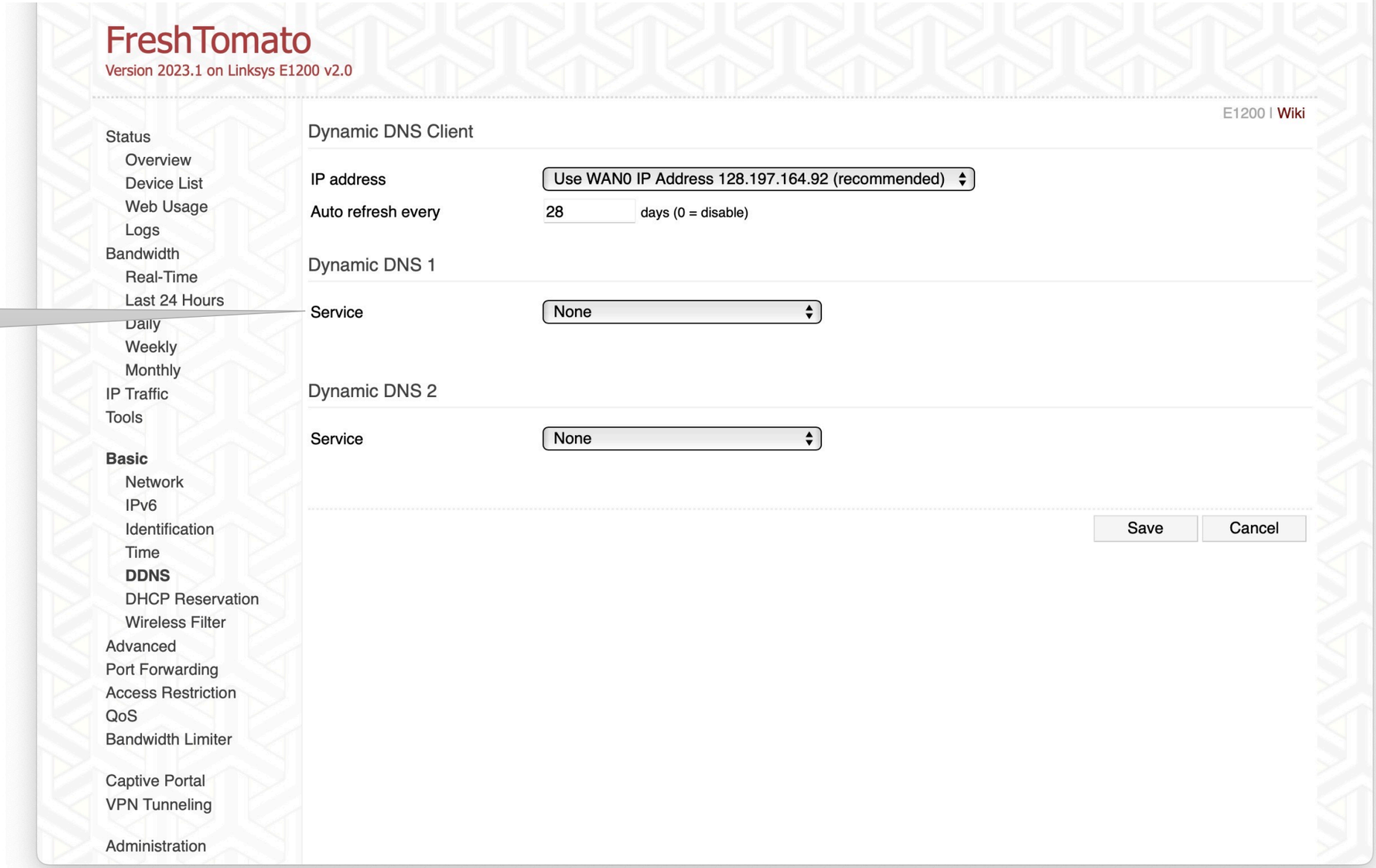
Ways to make more secure:

- HTTPS
- No wireless login
- No remote access
- Limit access by certain domains, IP addresses, or time windows

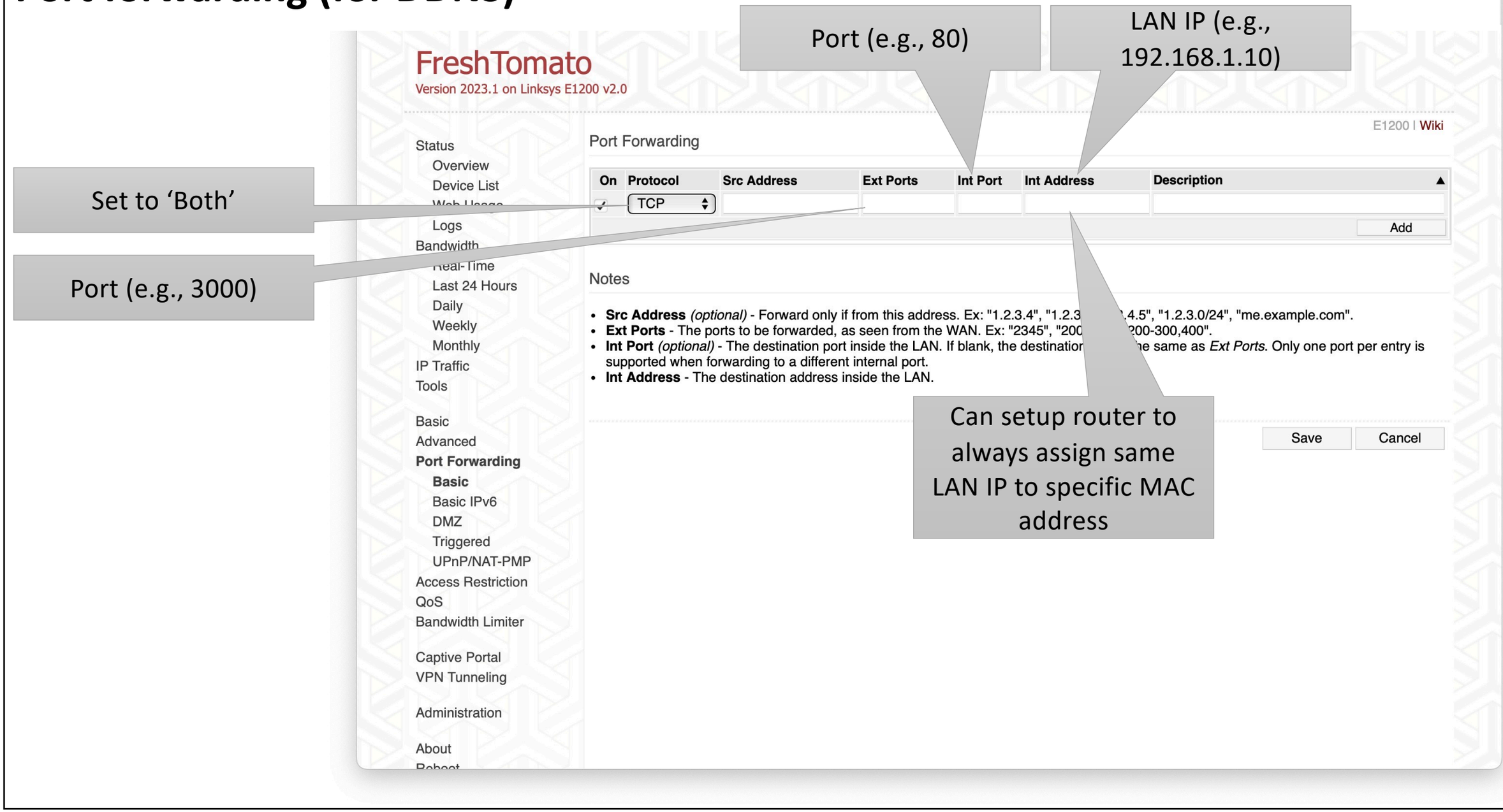


DDNS setup on router (not on Pi or laptop)

Set up DDNS

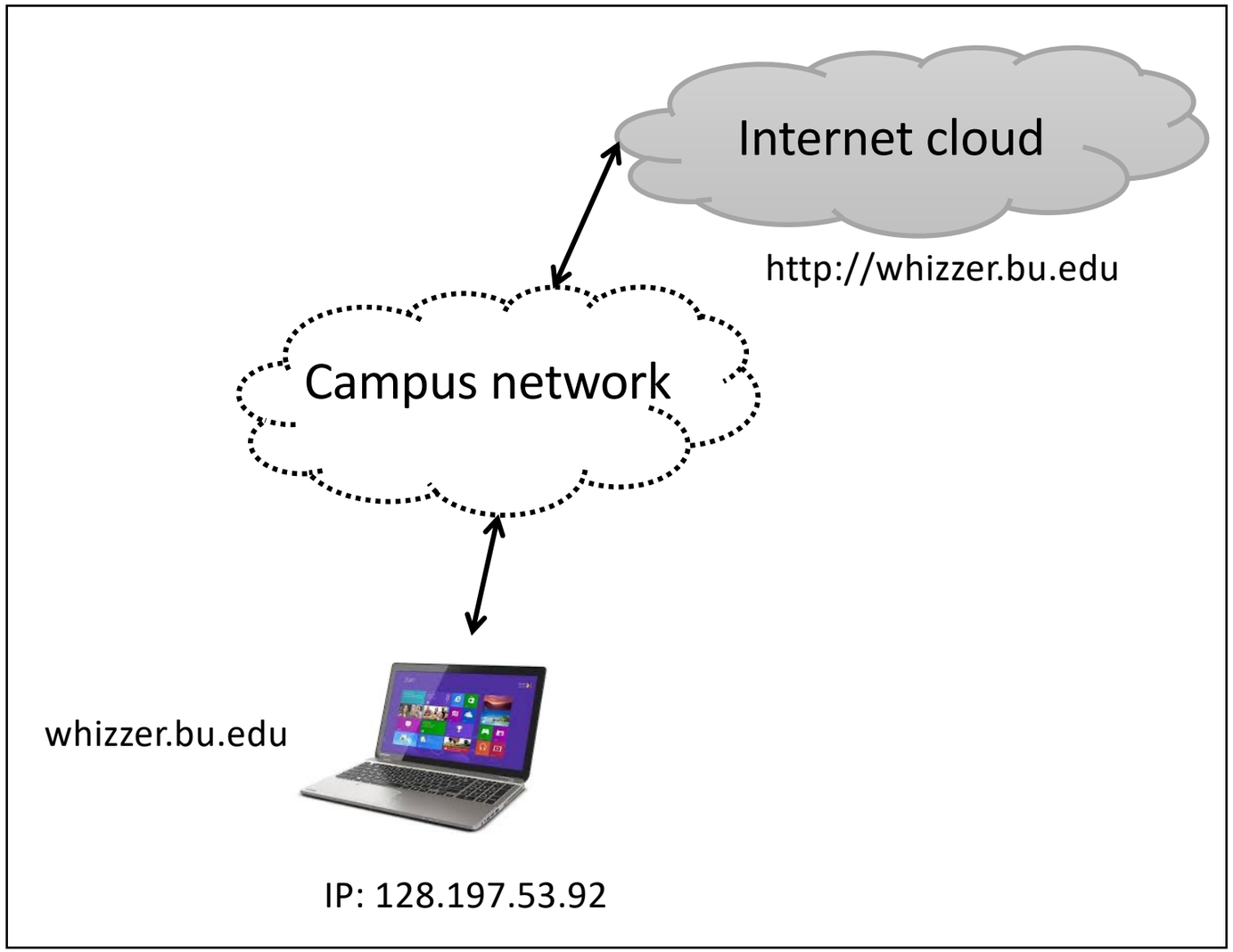


Port forwarding (for DDNS)



Skills: Dynamic DNS

DNS and static address mapping – static IPs in this illustration



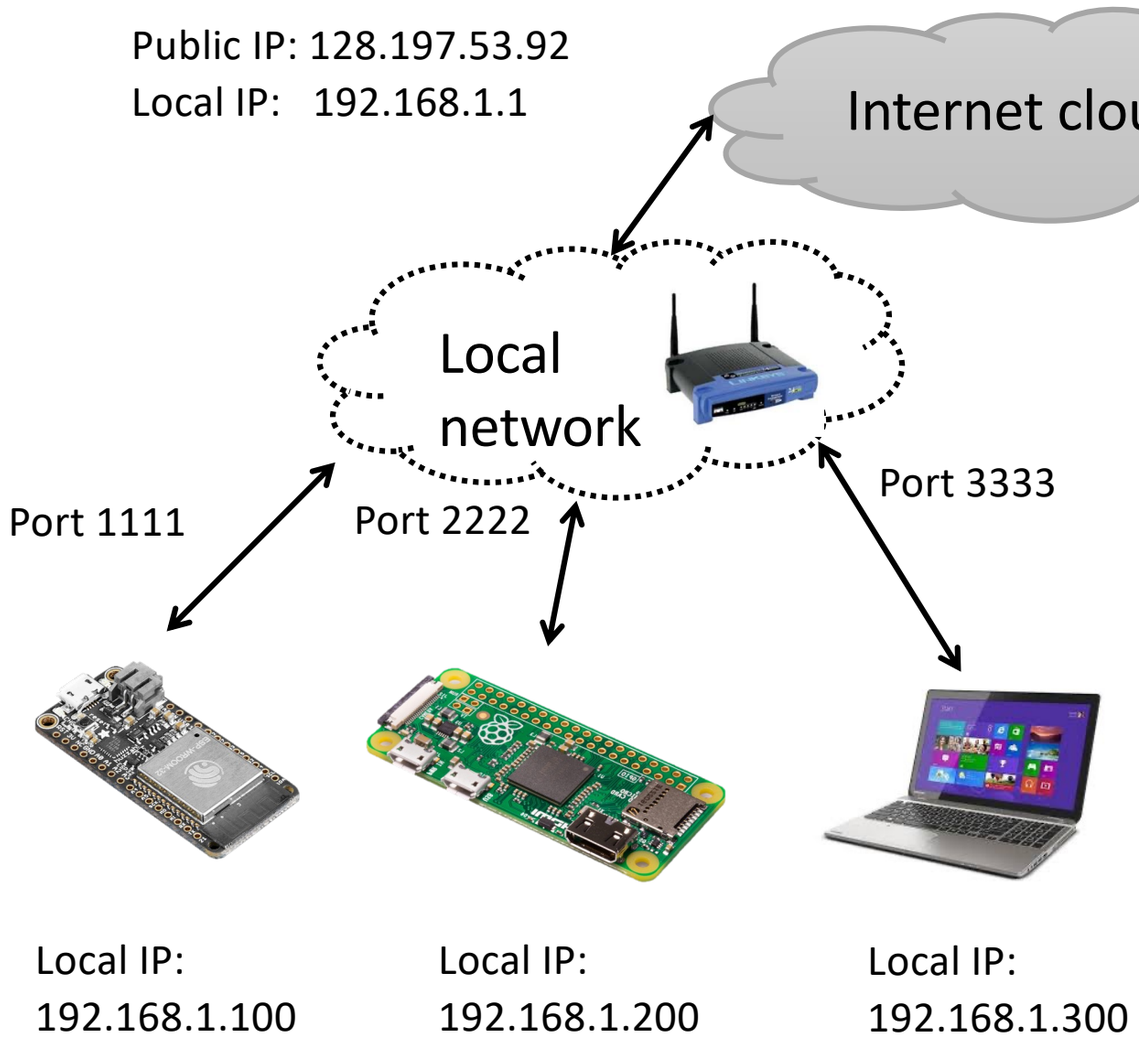
Typical static configuration

DNS Table (e.g., /etc/hosts)

Name	**IP Address**
whizzer.bu.edu	128.197.53.92
hulk.bu.edu	128.197.53.89
spiderman	128.197.53.88
…	…

But if IP addresses are not static, the table needs to be updated

Dynamic DNS – address remapping and port forwarding



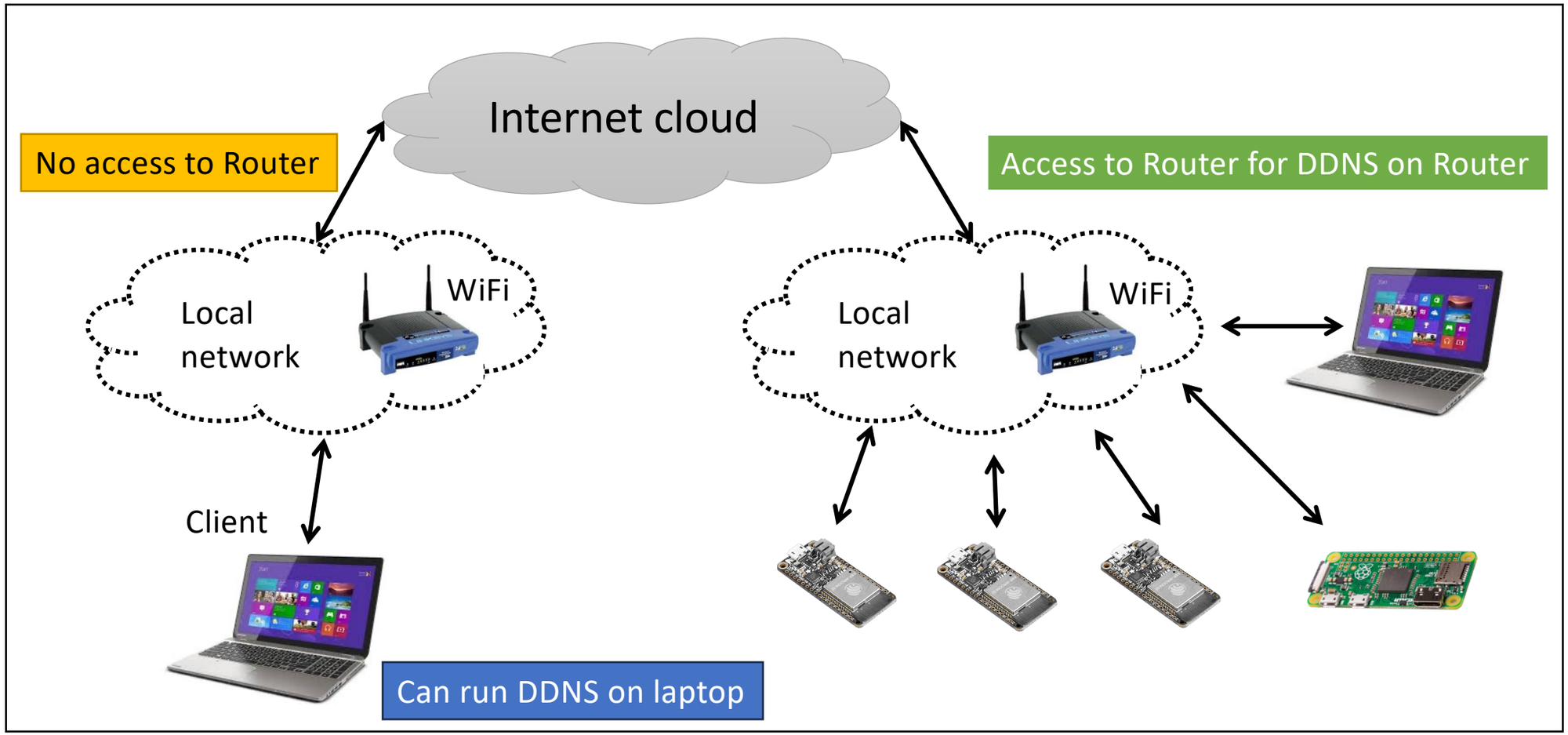
Store alias in cloud
Ec444.dyn.com→128.197.53.92

Routing Table		
Ext Port	Local IP	Int Port
1111	192.168.1.100	1111
2222	192.168.1.200	2222
3333	192.168.1.300	3333

Address 128.197.53.92:1111 → goes to ESP32
Address 128.197.53.92:2222 → goes to Pi
Address 128.197.53.92:1111 → goes to laptop
Address 128.197.53.92:80 → goes to router

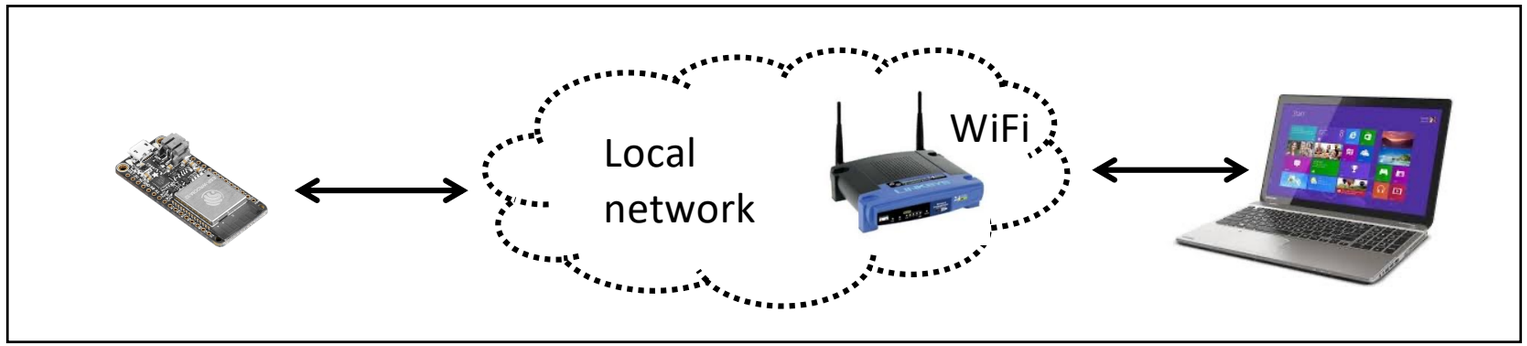
Can also re-map the port: 1111→ 80

Sometimes a problem: one end can’t do DDNS



Skills: WiFi-ESP-Station

Skill: WiFi for ESP -- WiFi “station” or “client”



ESP32 program

- Find SSID
- Authenticate → “connection establishment”

Task:

- Bring up ESP32 for connecting to your router
- Code: https://github.com/espressif/esp-idf/tree/master/examples/wifi/getting_started/station

Skills: Client and Server UDP with ESP32 and Node

Next: getting some data across the WiFi

- Some definitions
- **IP**: Internet Protocol (IPv4 / IPv6): rules and formats to send (route) packets in and between networks
- **Datagrams**: header and payload (packets); connectionless; order order of arrival not guaranteed (no acknowledgements) not specifically reliable
- **UDP** (User Datagram Protocol): protocol for sending and receiving datagrams; connectionless, checksums, no handshaking
- **TCP** (Transmission Control Protocol): Connection-oriented and reliable (with retransmissions), flow control
- **Socket:** enables application multiplexing (IP address : port)
- **Port**: communication endpoint – binding point between service and network; a number

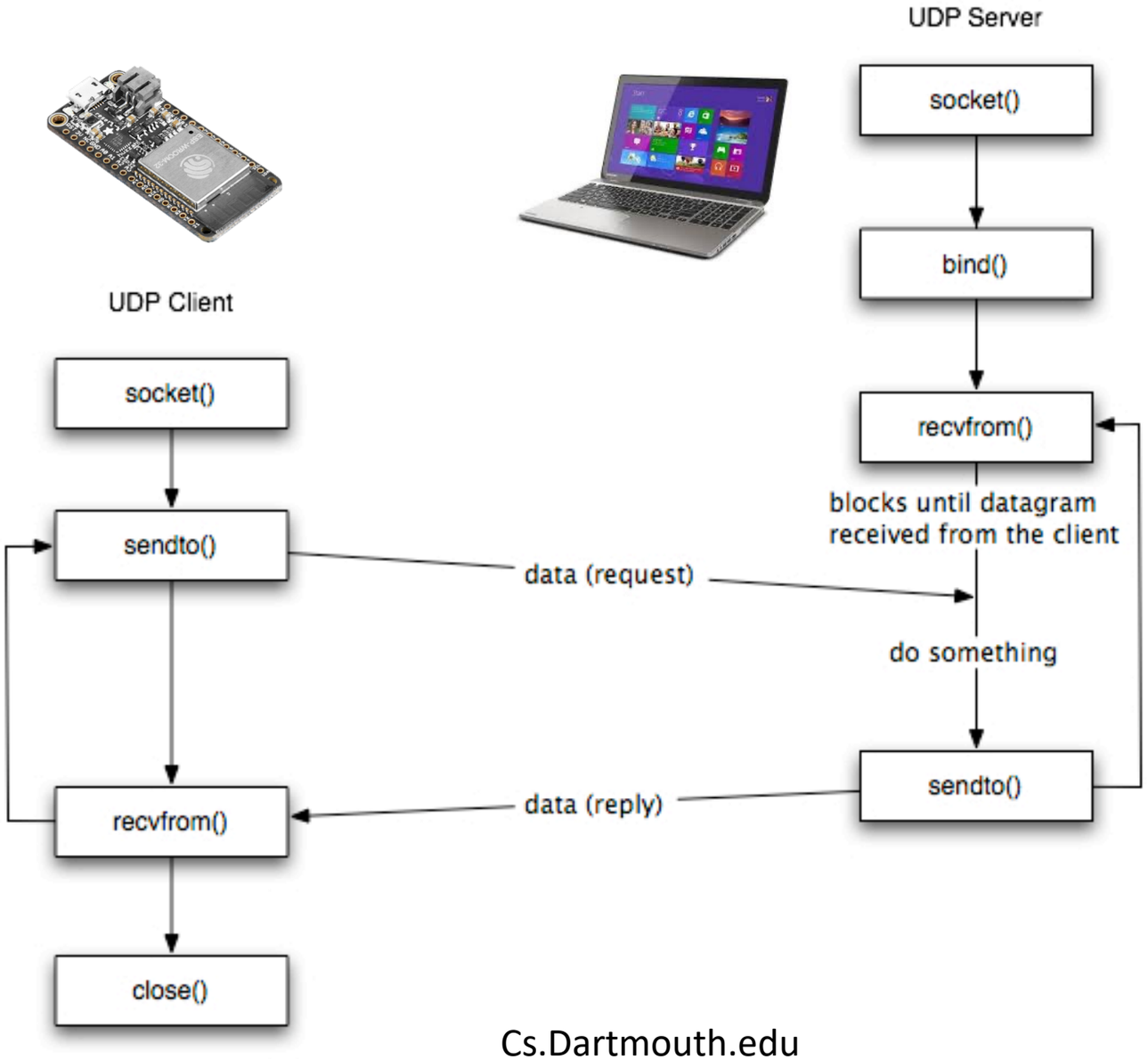
UDP enabled with a client and a server

Essentials

- Client and server aware of IP address and port numbers
- Server set up to receive and respond
- Client set up to request and receive

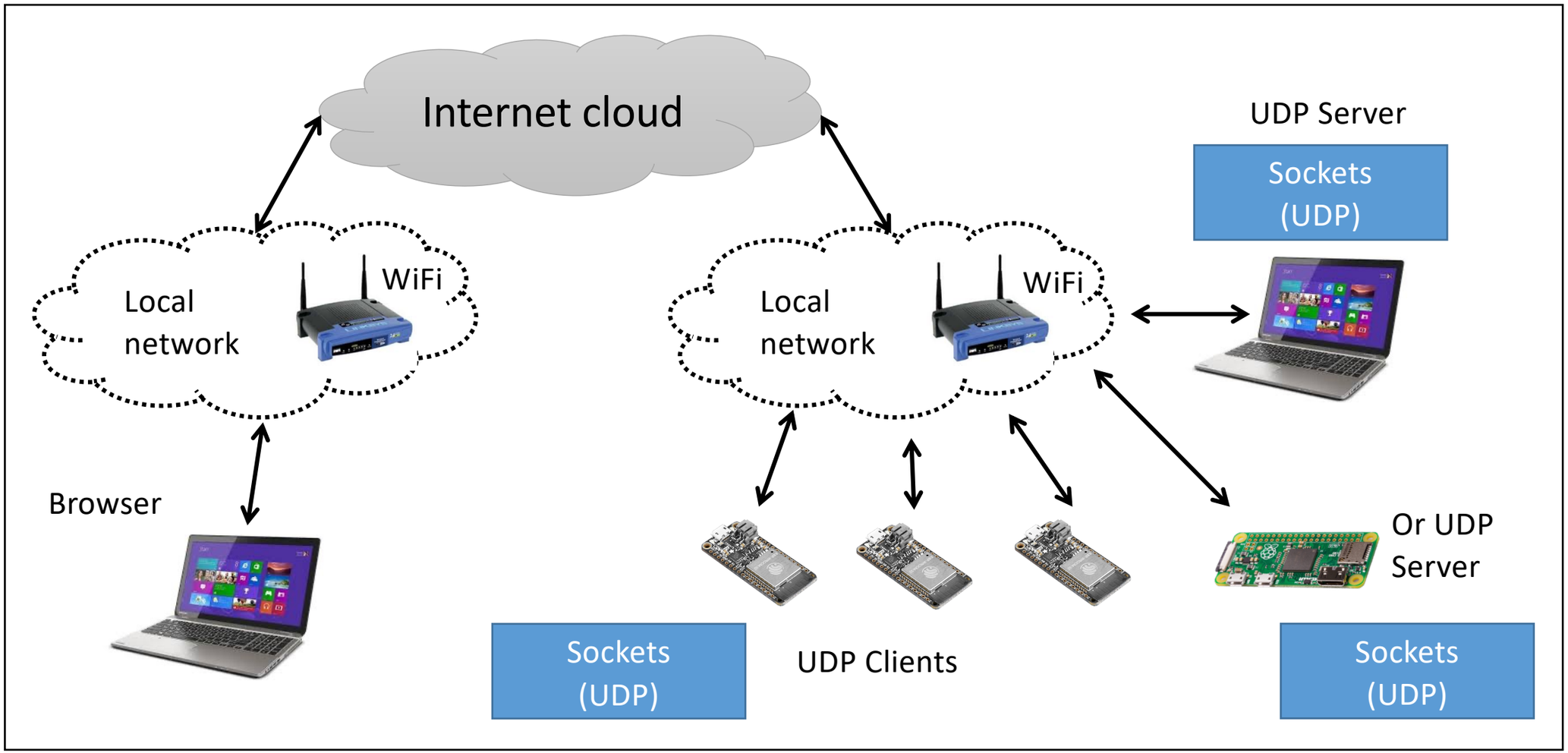
Client initiates in this model

Also in node.js socket model

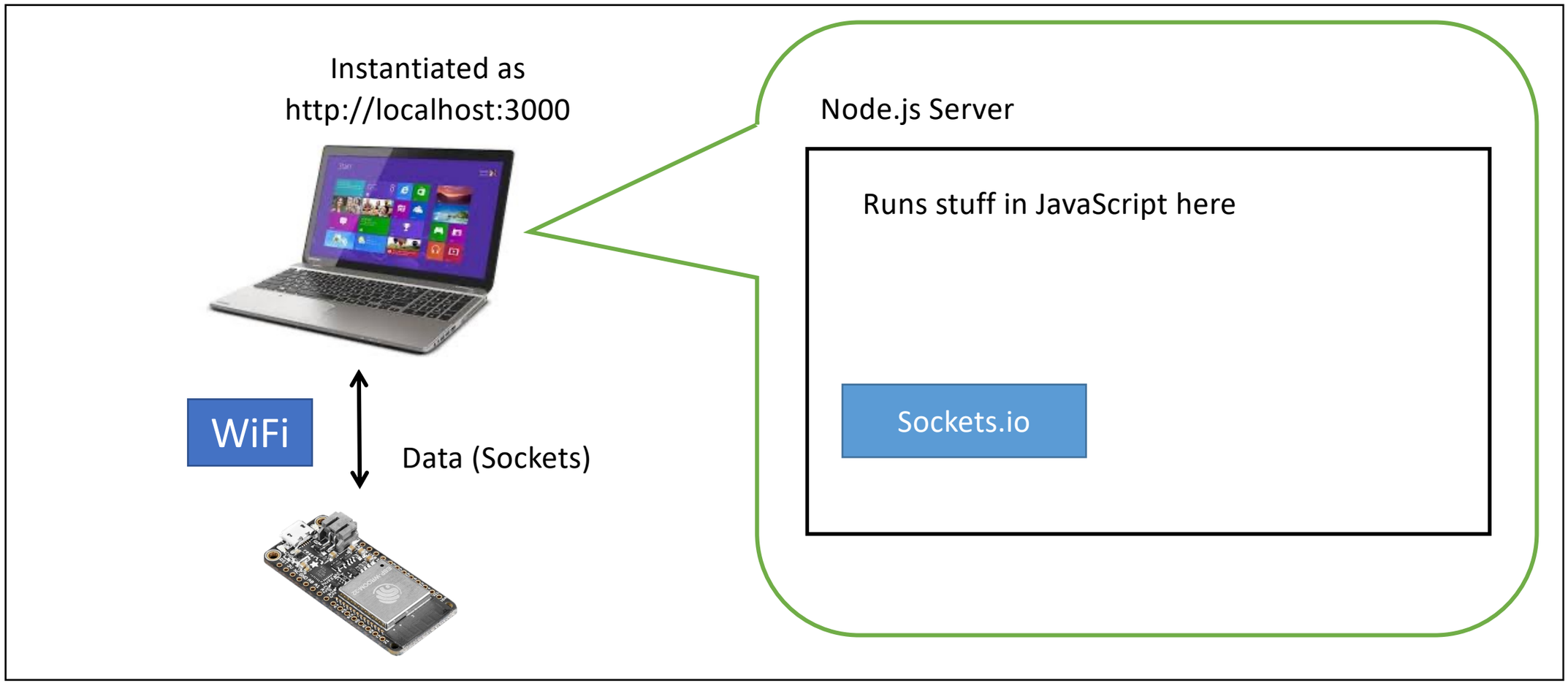


Cs.Dartmouth.edu

In our context



Data Flow – node.js and comms



***Not the only way to do it

UDP enabled with a client and a server

Skill Part 1:

- Read **design pattern** for UDP Sockets
- Bring up a UDP server in the node app
- Bring up a UDP client at the ESP32 including WiFi
- The ESP32 should send a message requesting the blink time from the laptop (node server)
- Then node server responds with a new blink time
- The ESP32 receives the blink time and flashes the onboard LED

Skill Part 2:

- Bring up a UDP client app on the node server
- Bring up a UDP server at the ESP32 including WiFi
- The node app should send a message to change the status LED on the ESP32
- The ESP32 changes the LED and and then acknowledges the change back to the node server.

https://github.com/BU-EC444/01-EBook/blob/main/docs/briefs/design-patterns/dp-sockets.md

Skills: Raspberry Pi and Node.js

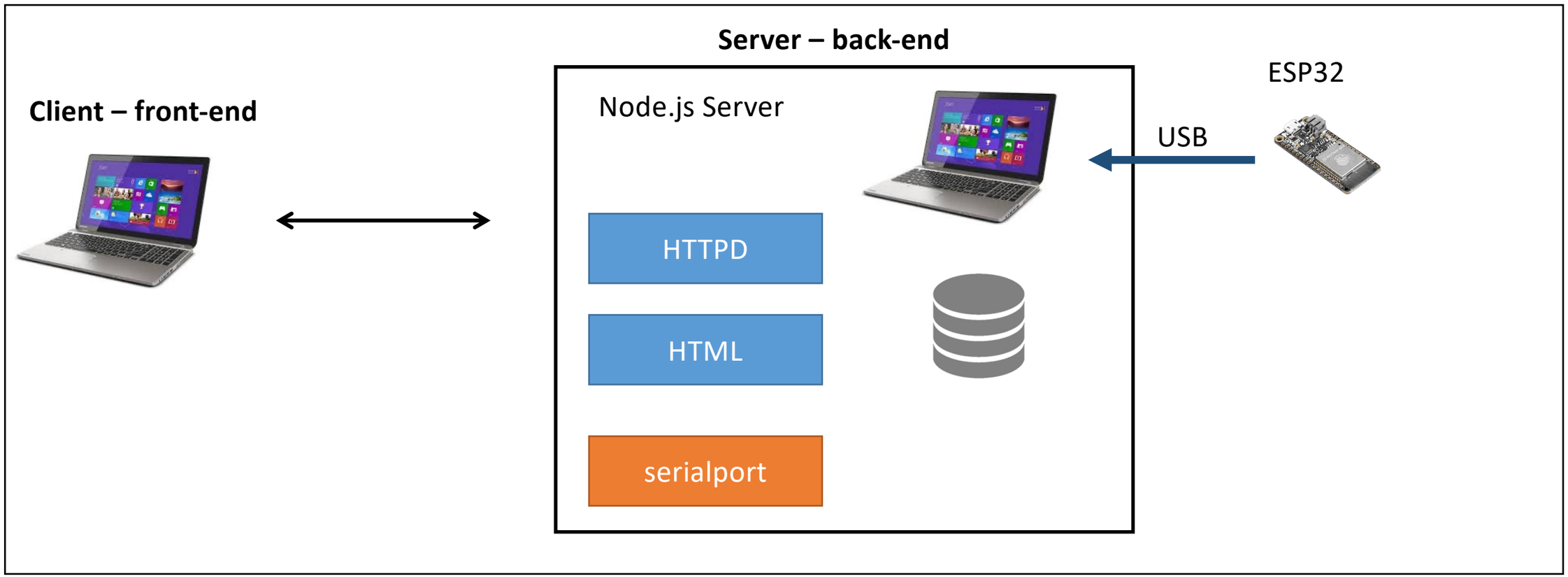
Raspberry Pi Zero

- 1GHz single-core CPU
- 512MB RAM
- Mini HDMI port
- Micro USB OTG port (on the go – for ext drives)
- Micro USB power
- HAT-compatible 40-pin header
- Composite video and reset headers
- Camera connector

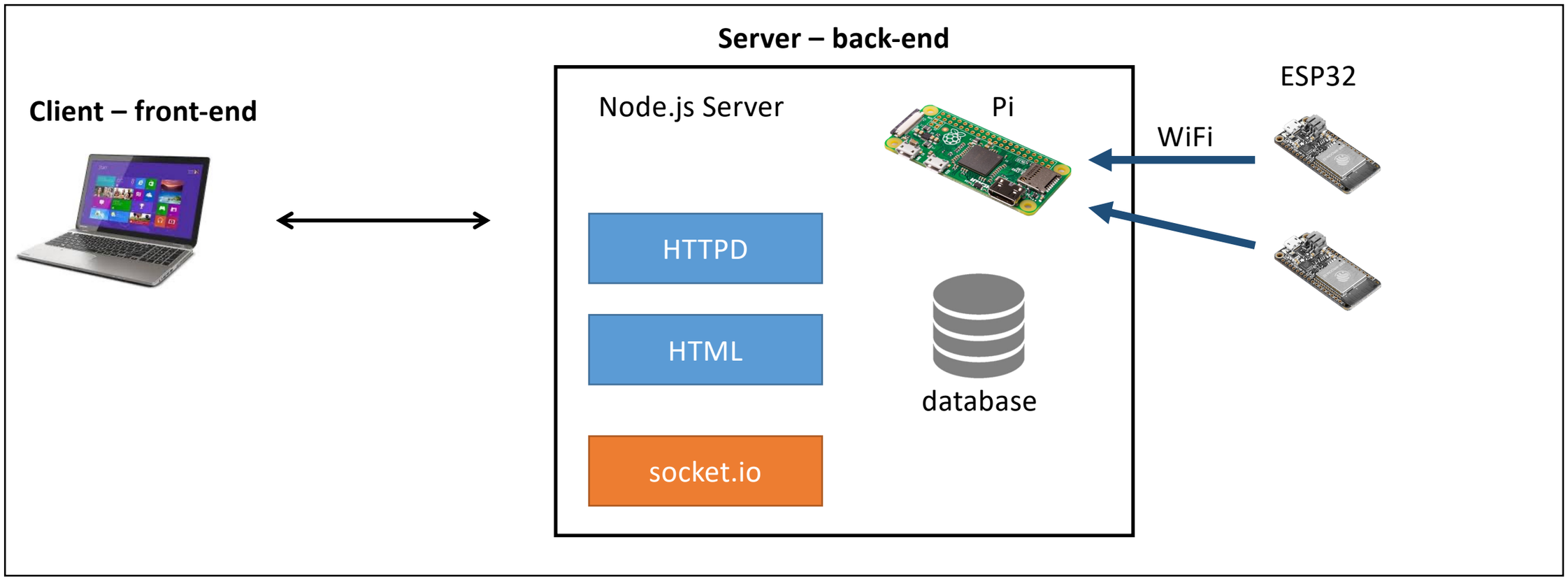
From the Pi zero web site

We use this primarily as an embedded server, not for the I/O

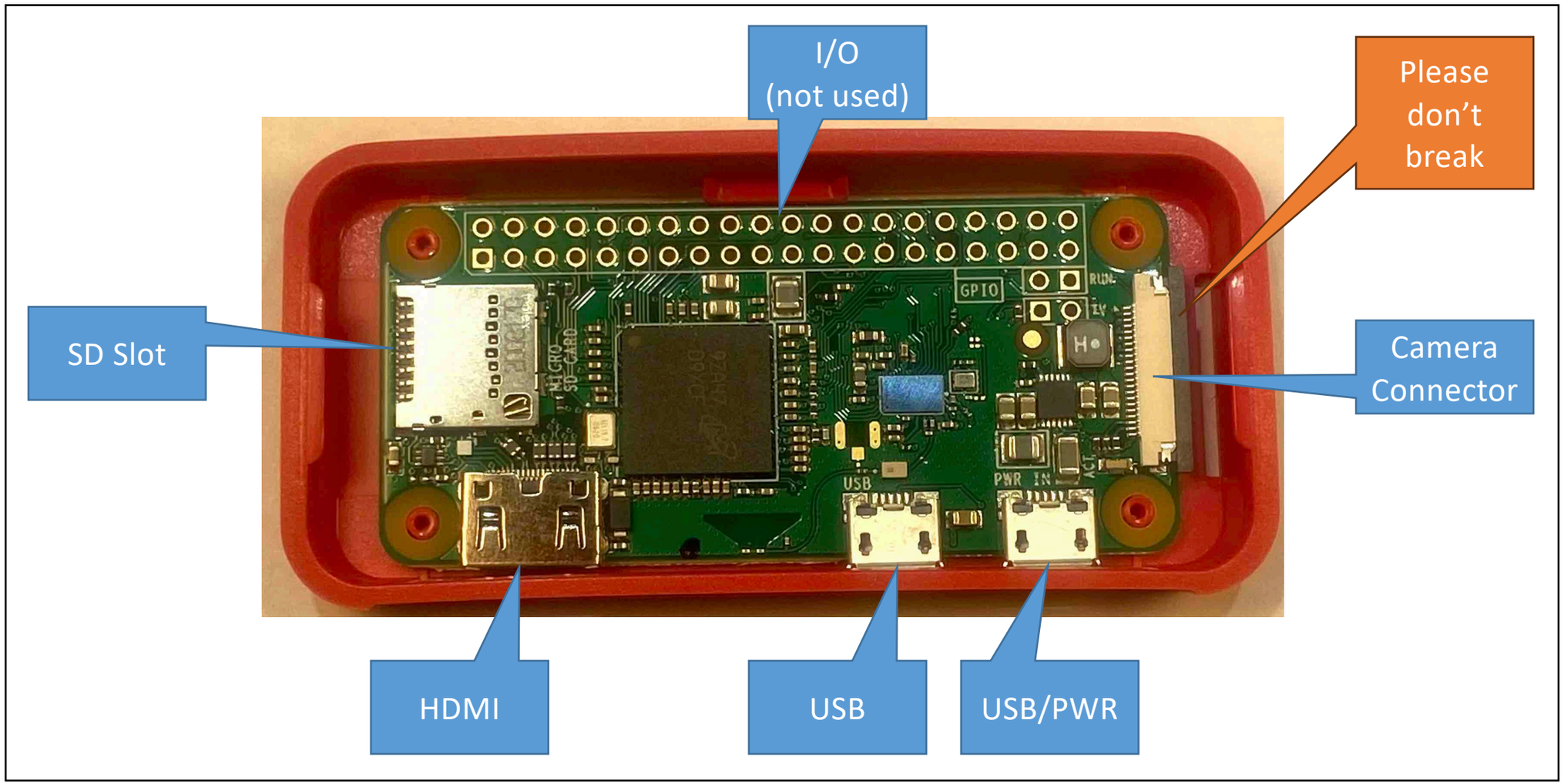
This is what we previously did for hosting data from the ESP32



Then move to WiFi and Pi

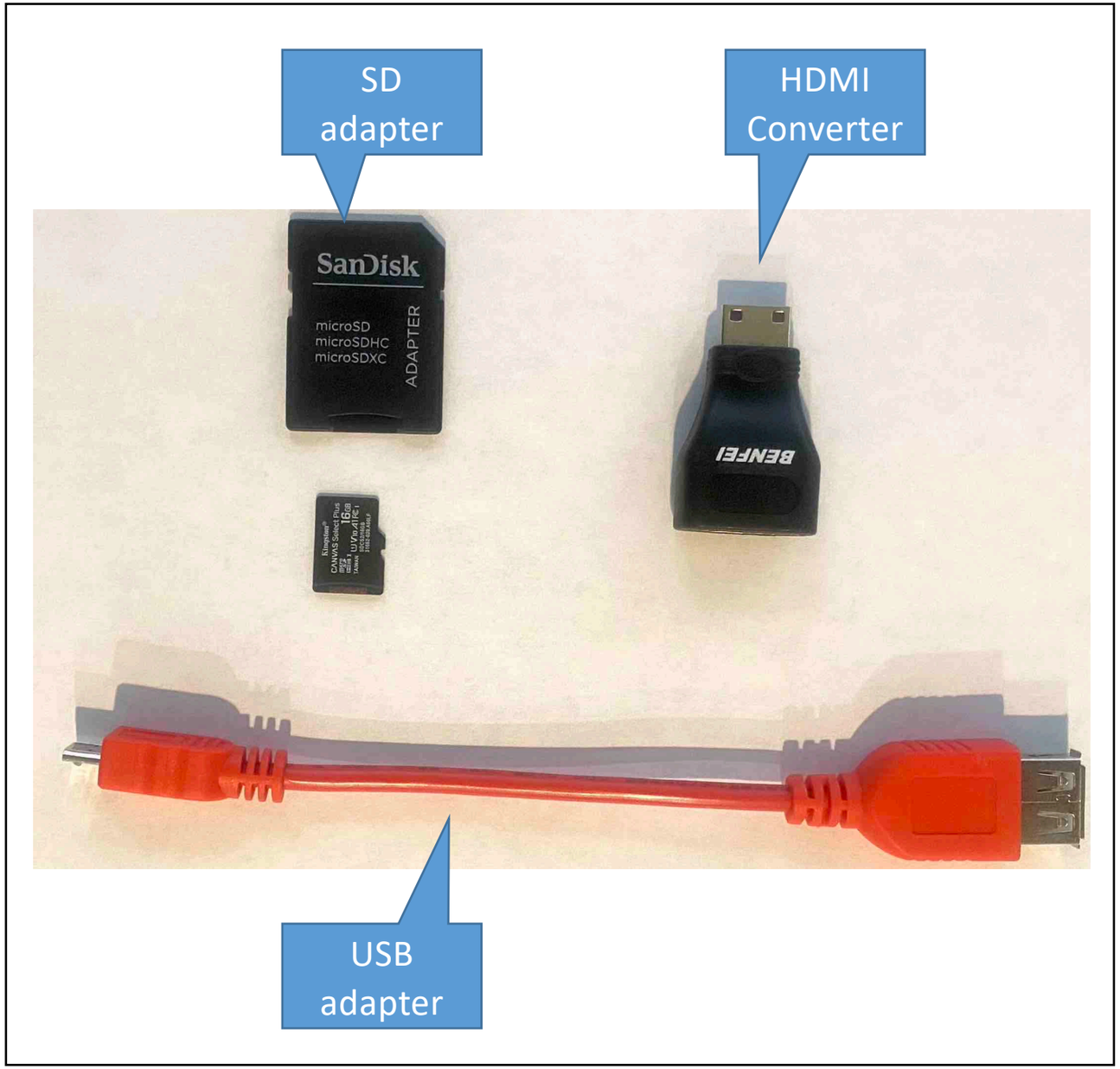


Notes – Navigating the Pi Zero board



Notes – Pi Zero kit

- Each kit has a 16GB SD card
- Micro SD to SD adapter
- HDMI Mini converter (or cable)
- Mini USB to USB adapter cable



Bringing up the RPi – skill part (1)

- Download the Raspberry Pi Imager.
- Image your SD card with the Raspberry Pi OS
- Use the menu under the gear for setting SSID/PWD and UID/PWD for your router (Group_X/smartsys) and pi (pi/smart)
- You are doing a “headless” install – without keyboard, mouse, or monitor
- Unplug your Pi, insert the SD card you just flashed, and plug the pi into power (do not unplug until it is done)
- After boot (could be 5 min), you should see the pi appear on your local network (log into the router and look at the **device list**)
- Now you can open a command prompt and `ssh pi@ip-address-of-pi`
- UID pi, PWD smart

* https://www.raspberrypi.com/documentation/computers/configuration.html#configuring-networking

Flashing the drive from laptop

Brings up menu to configure SSID, UID/PWD

Bringing up the pi – skill (part 2)

Install node.js and npm on pi – follow instructions on skill assignment

- This method gets the latest distribution for the ARMv6 used on our pi and does installation
- Bring up ‘hello world’ using node (from earlier skill, just now on pi)

Configure git on your pi to pull code from your repo

- Set up your pi to pull code from your EC444 git repo
- For example: (on the pi)

```
git clone https://github.com/BU-EC444/myrepo
```

- And to refresh with latest code:

```
git pull https://github.com/BU-EC444/myrepo
```

Using git pull for pi

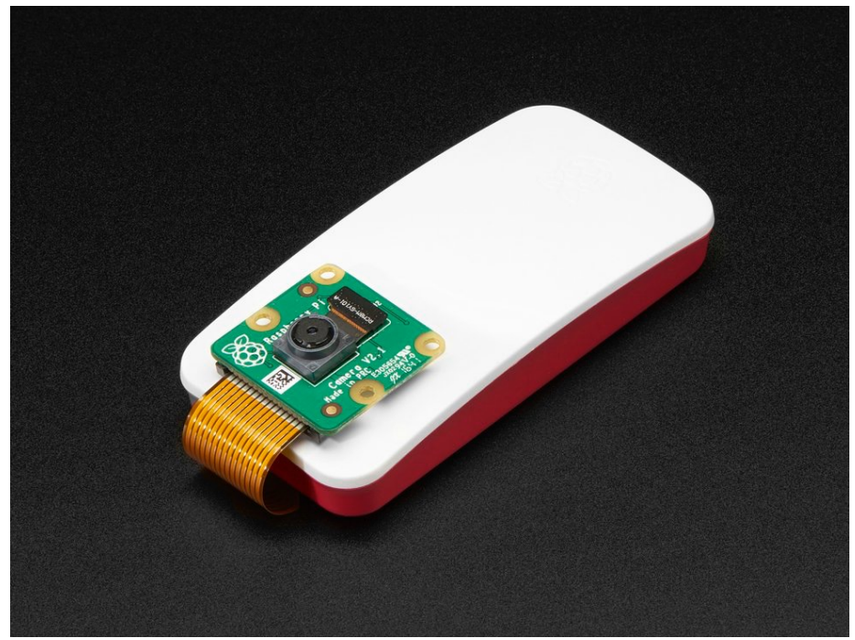
- We recommend that you test node.js code on your laptop before deploying on the pi.
- Using git on the RPi will greatly simplify deploying code on the RPi as you will not need to write and debug on the RPi.
  - Write the code on your laptop
  - Push to your git repo
- Pull from the pi to transfer working files to the pi

- ***Remember: don't save your build folders on git***.
- Do an `idf.py fullclean` before you push from your laptop

Skills: Pi Web Cam

Pi web cam

- Pi zero with Camera Board v2
- We're doing this 'headless' – no keyboard or monitor
- Be careful with the camera connector



- Multiple options available for software we will use rpicam-apps: https://www.raspberrypi.com/documentation/computers/camera_software.html#rpicam-apps

- Demonstrate streaming on browser either from pi or from local network browser
  - Use rpicam-vid to source the video
  - Vlc to play the video from another machine
  - https://www.raspberrypi.com/documentation/computers/camera_software.html#stream-video-over-a-network-with-rpicam-apps