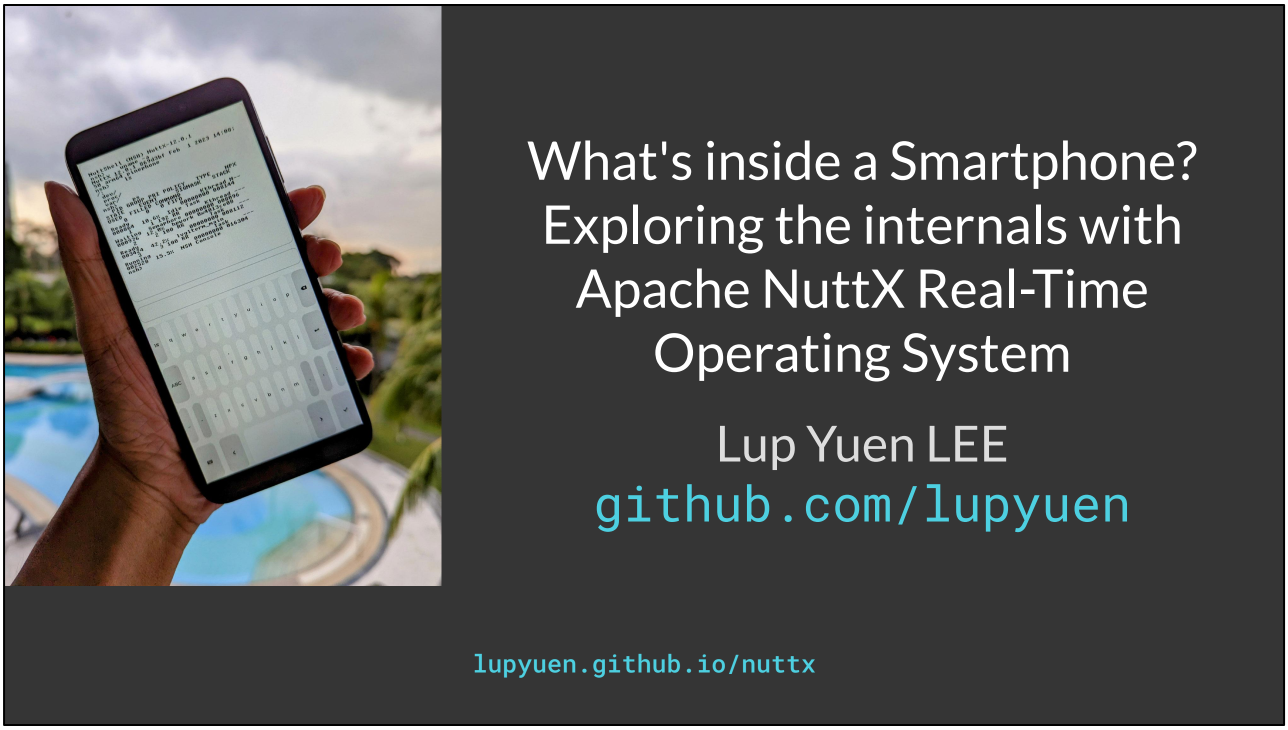


What's inside a Smartphone? Exploring the internals with Apache NuttX Real-Time Operating System

Lup Yuen LEE

github.com/lupyuen

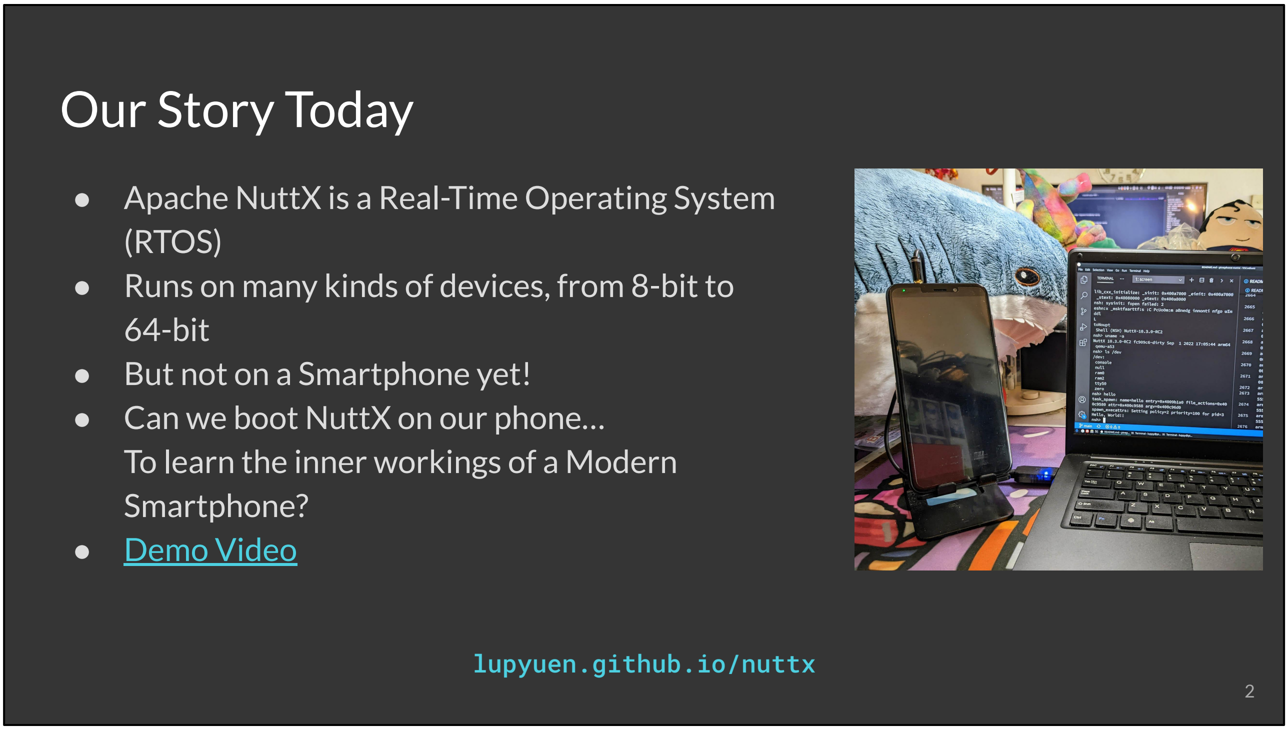
lupyuen.github.io/nuttx

Hello Everybody!
I am Lup
From Sunny Singapore

I used to teach IoT (Internet of Things) in school
Today I'm still in IoT Education
But instead of teaching in a classroom
I write Educational Articles about IoT

I joined the Apache NuttX Project Management Committee
About a year ago
And that's topic of my presentation today

If you would like to download the Presentation Slides
Please head over to this link
lupyuen.github.io/nuttx
That's the letter L, not the digit 1
My presentation transcript, articles and source code
Are available at that link



Our Story Today

- Apache NuttX is a Real-Time Operating System (RTOS)
- Runs on many kinds of devices, from 8-bit to 64-bit
- But not on a Smartphone yet!
- Can we boot NuttX on our phone…
  To learn the inner workings of a Modern Smartphone?
- Demo Video

`lupyuen.github.io/nuttx`

Apache NuttX is a Real-Time Operating System
Works like a tiny version of Linux
But a lot smaller
A lot simpler

NuttX runs on many kinds of devices
from 8-bit to 64-bit devices
Arm and RISC-V
But not on a Smartphone yet!

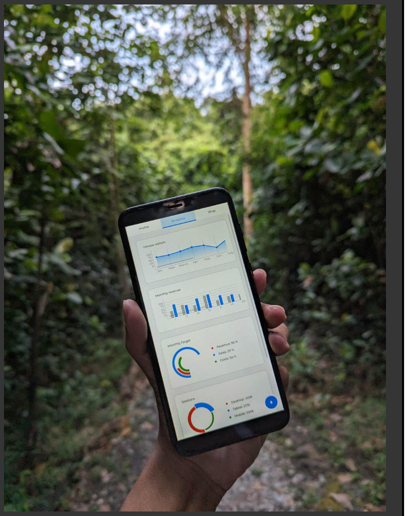
Our Smartphones are incredibly complex gadgets
Super mysterious
Nobody really understands what’s inside

What if we could boot NuttX on our phones
To understand how our phones work?
That would be an awesome way to learn
The internals of a Modern Smartphone!

But most phones are locked down
Difficult to hack
Try hacking an iPhone!
So we take a phone
That was created for Hacking

What is PINE64 PinePhone

- 4G Smartphone designed for Mobile Linux
- Allwinner A64 SoC with 2 GB RAM
- Quad-Core Arm64 Cortex-A53
- LCD Display: Xingbangda XBD599
- Touch Panel: Goodix GT917S
- LCD Controller: Sitronix ST7703 (MIPI DSI)
- LTE Modem: Quectel EG25-G
- WiFi, BLE: Realtek RTL8723CS
- Sensors: Magnetometer, Accelerometer, Proximity
- NuttX boots on microSD, doesn't touch eMMC



lupyuen.github.io/nuttx

PinePhone is a remarkable 4G Smartphone
Assembled with off-the-shelf components
Designed for Mobile Linux

Inside is the Allwinner A64 SoC
System-on-a-Chip
That runs 4 Cores of Arm Cortex-A53
Yep it's a 64-bit CPU

The components are surprisingly familiar
LCD Display
Touch Panel
Display Controller
ST7703
LTE Modem by Quectel
WiFi and Bluetooth by Realtek
MPU-6050 Accelerometer

The components are sourced off-the-shelf
Which makes PinePhone easier for learning
And porting NuttX

To make sure we don't mess up people's phones
We boot NuttX on a MicroSD Card

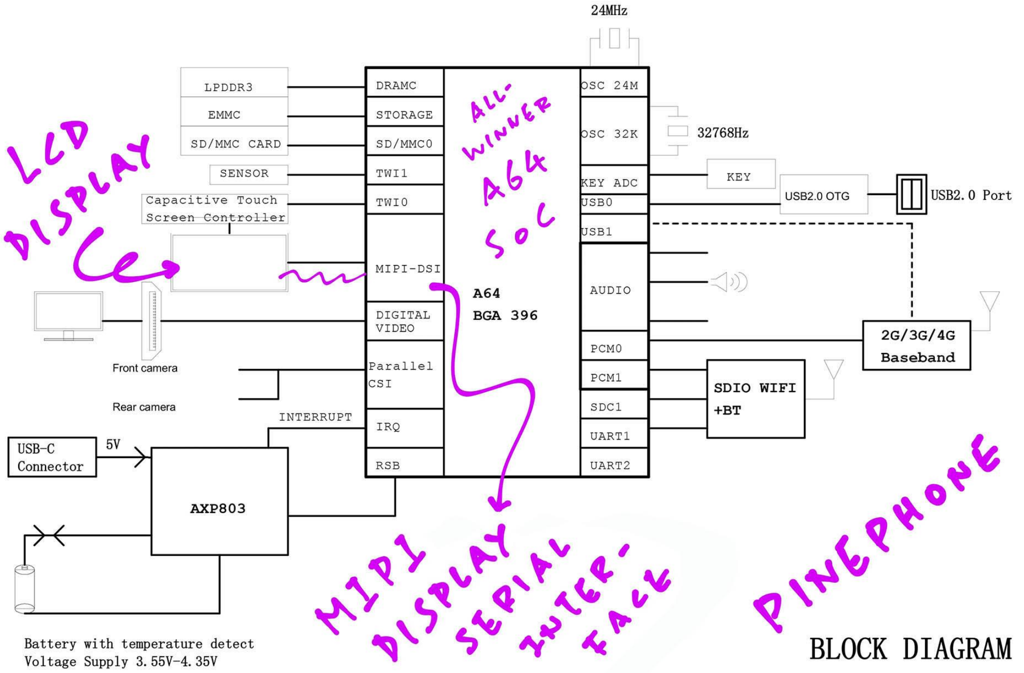
Without touching the Internal Storage
No installation required

(Demo)
How exactly do we boot NuttX on a smartphone?

Well all we need is a PinePhone
And a microSD card
We download the NuttX Image
Copy it to a MicroSD Card
Insert the microSD Card into PinePhone
Power up

And that's it!
NuttX begins to run on our smartphone
No fuss
No installation

How did we get NuttX running on a smartphone?
We took one whole year to build up NuttX for PinePhone
I shall explain the entire
Development process
And the things that we can learn
From NuttX on PinePhone



PinePhone Schematic

So PinePhone works like a normal phone!
Like an old iPhone 5s
Which is also 64-bit Arm

All the components are there
Just simpler, affordable ones

How shall we study the
Smartphone Internals
With NuttX?

We begin with the most complex component
The LCD Touchscreen
That's connected on MIPI DSI
The Display Serial Interface

Inside a Smartphone Display

- LCD Display: Xingbangda XBD599
- LCD Controller: Sitronix ST7703
- MIPI Display Serial Interface (DSI) transmits pixel data to the LCD Panel
- Allwinner Display Engine renders bitmap graphics and pushes the pixels over MIPI DSI
- NuttX Framebuffer exposes the rendering API to NuttX Apps
- Prototype Driver in Zig to de-risk the development

`lupyuen.github.io/nuttx`

My background is in IoT Microcontrollers
I'm familiar with SPI and ST7789 Displays
But smartphone displays are
Overwhelmingly sophisticated!

Inside the Allwinner A64 chip
Is a Display Engine
That renders pixels
And pushes them to the LCD Display
Over the Display Serial Interface

Making this work with NuttX
Was a highly Educational Exercise

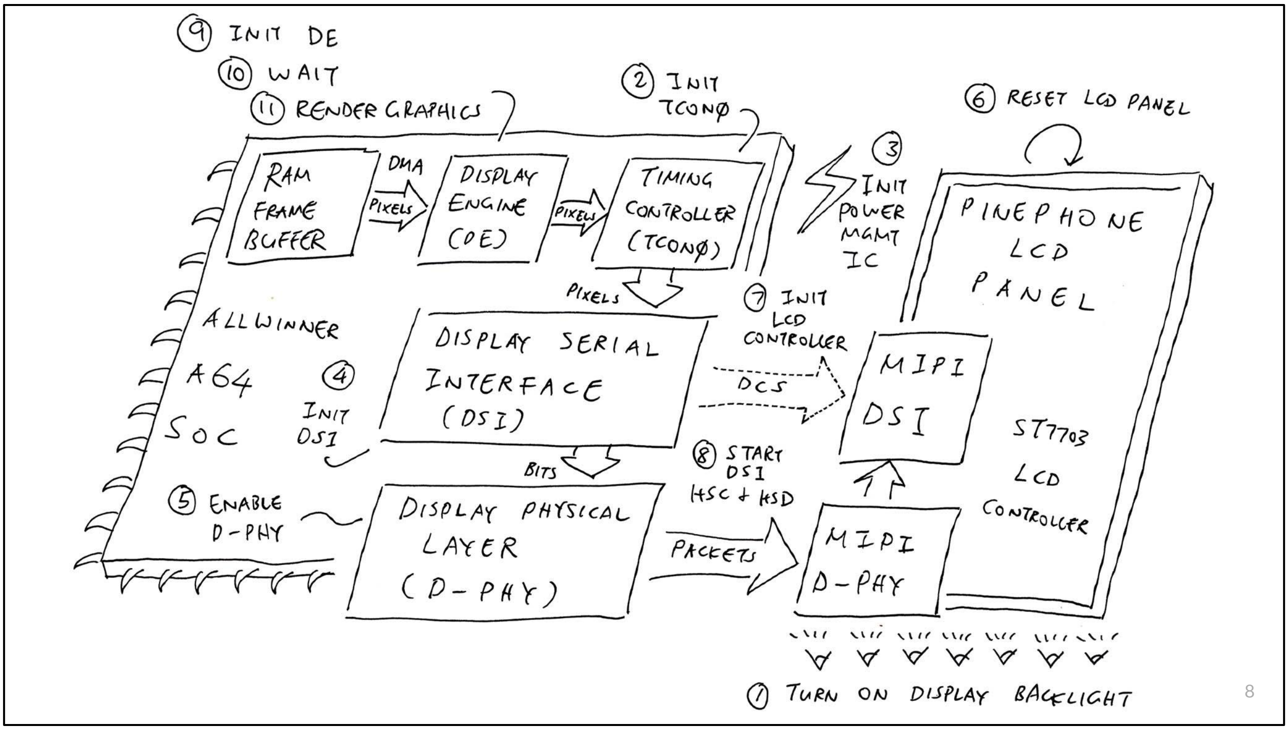
Thankfully we didn't need to build all the drivers
NuttX provides some drivers
Like the Framebuffer Interface
That makes graphics rendering
A little easier

Most of this
Isn't well documented
So we created quick prototypes
Of the Display Drivers

In the Zig Programming Language

Zig is safer
It has Runtime Checks
So we don’t run into Bad Pointers
Inside our Prototype Driver

When everything has tested OK
We converted the Prototype Drivers
From Zig to C
Which is not that hard!
Since they are similar languages



Now if we’re doing all this work
To teach the internals of a Smartphone
We better make sure that
Everything is documented!

That’s why we wrote a stack of Articles
With whimsical sketches like these
To explain what’s really happening
Inside our phones

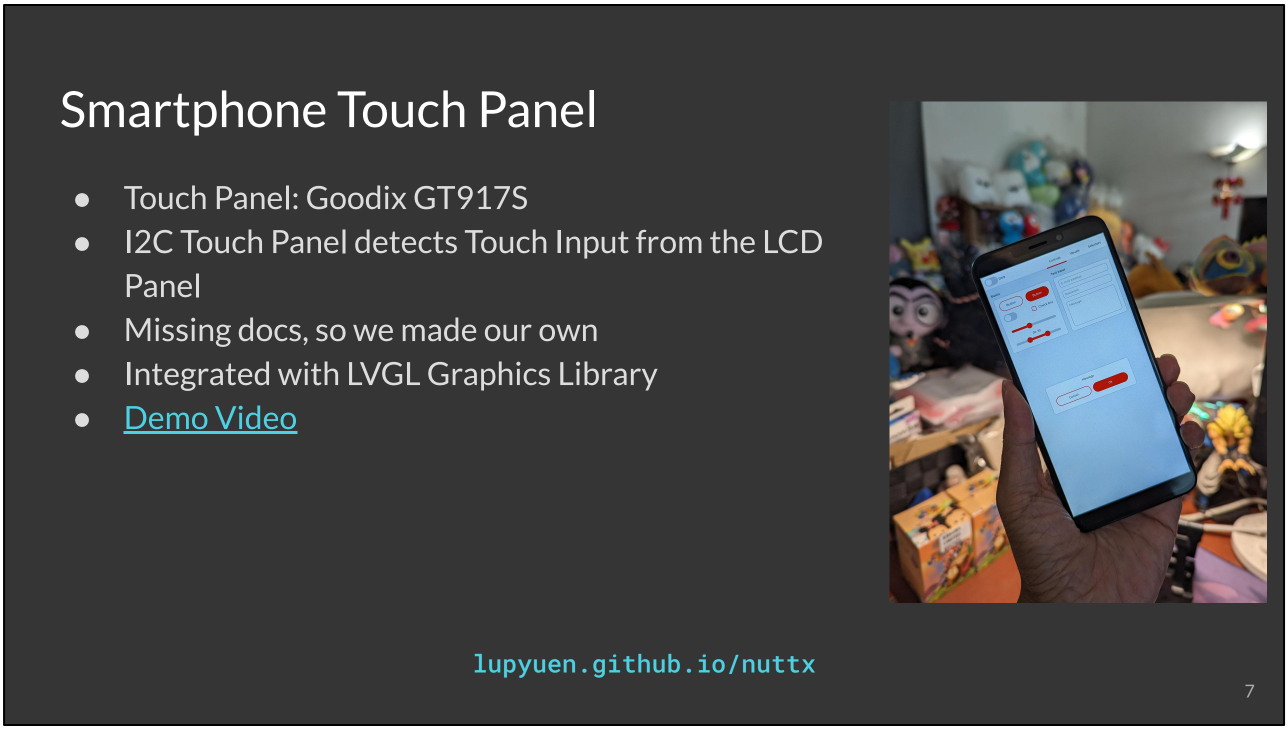
I can’t imagine
How else we could explain this intricate process!
Turning on the Backlight
Starting the Timing Controller
Switching on the Power

Programming the Display Serial Interface
To push pixels over the
Display Physical Layer

Don’t forget the
Framebuffer in RAM
And the Display Engine!

Hopefully our learners will understand
A little better
How our Smartphones
Are pushing pixels to the display

Together with the NuttX Source Code
That we have thoroughly documented



Smartphone Touch Panel

- Touch Panel: Goodix GT917S
- I2C Touch Panel detects Touch Input from the LCD Panel
- Missing docs, so we made our own
- Integrated with LVGL Graphics Library
- Demo Video

lupyuen.github.io/nuttx

Our Smartphone isn't really so Smart
Unless it accepts Touch Input

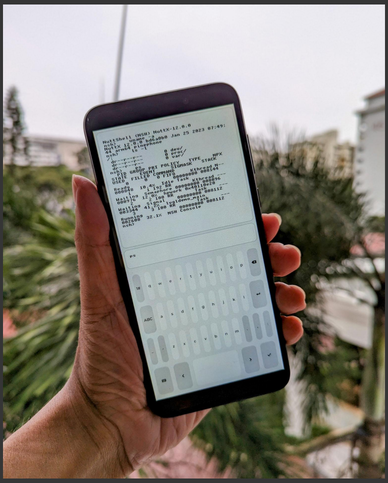
In spite of the missing documentation
We managed to create the
NuttX Driver for the Touch Panel
And we filled in the missing docs

Learners who are familiar with Microcontrollers
Might find it fascinating that
These Touch Panels will talk over a
Simple Protocol
Like I2C!

Now we need a way to write
Touchscreen Applications
Like the ones we see
In App Stores!
Which brings us to an
Interesting problem

Graphical Apps with LVGL

- LVGL Graphics Library for Embedded Devices
- Because NuttX doesn’t have X11, GTK, Qt, SDL
- Proof of Concept: LVGL Terminal App
  Demo Video
- Create LVGL Apps in the Zig Programming Language
- Simpler and Safer for Education



lupyuen.github.io/nuttx

NuttX is a
Real-Time Operating System
Created for Embedded Devices
Simple and easy to learn

But NuttX is not a
General Purpose Operating System
Like Linux
With its X11, GTK, Qt, SDL
And other Graphical Toolkits

So we borrowed something from the
Embedded World
LVGL
The popular Graphics Library
For Embedded Devices

LVGL isn’t something we normally use
To write a Smartphone App
But it works!

(Demo)
Here are two Demo Apps
That are bundled with the LVGL Library

At the left is a Forms-Based App
At the right is a Music Player App

For those of us familiar with LVGL Library
Typically we see these LVGL Demo Apps
Running on a Microcontroller

This is probably the first time
That anyone has attempted to run the
Demo Apps on a Smartphone

We see that they work great with
NuttX on PinePhone!
Quick, smooth and responsive
Just like any smartphone app

And this is the LVGL Terminal App
That we created for NuttX
It runs NuttX Commands
In the NuttX Shell
Looks a bit like Linux

These demos were created by calling
The LVGL Library in C programs
Which might be intimidating
For new learners
Let's talk about it

Since we are targeting
Education and Learners
Can we make LVGL Apps
Easier to write?

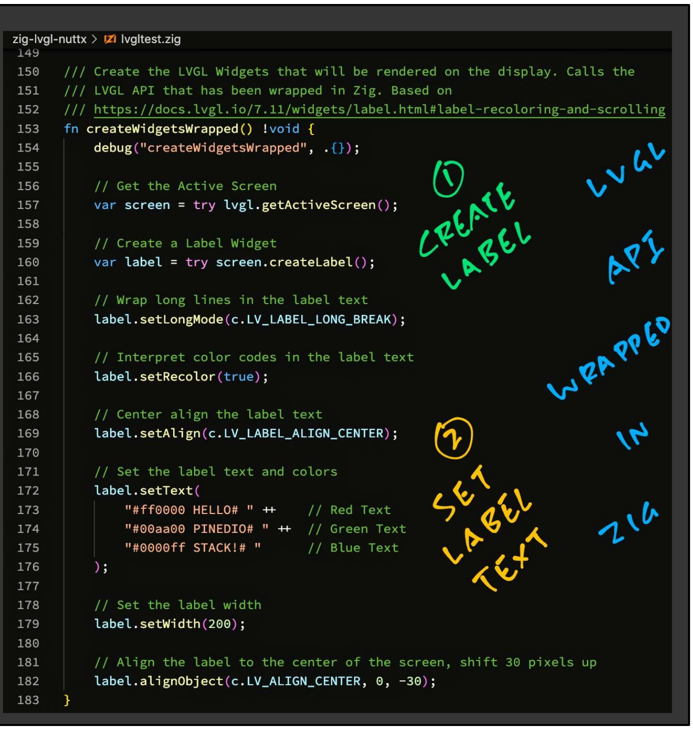
Something simpler than C
Without scary pointers

Something that has a Safety Net
A Safety Harness
That will catch us
If we fall into
Runtime Problems

Once again
The Zig Programming Language!

Why Zig

- Works well with C
- Runtime Safety
  (But not Memory Safe)
- Type Inference
- Auto-Converts C to Zig



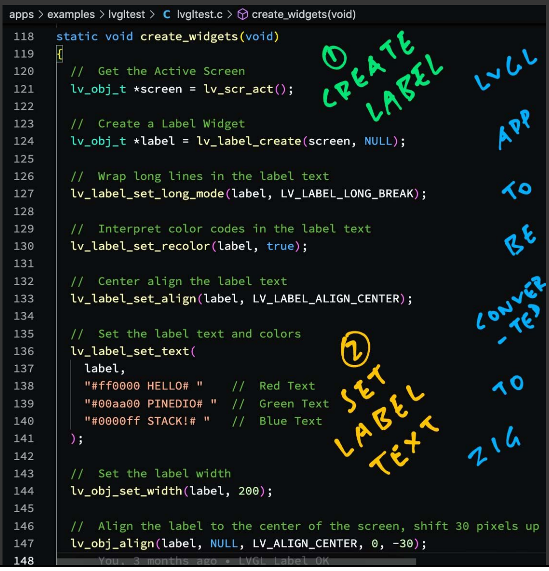
```
/// Create the LVGL Widgets that will be rendered on the display. Calls the
/// LVGL API that has been wrapped in Zig. Based on
/// https://docs.lvgl.io/7.11/widgets/label.html#label-recoloring-and-scrolling
fn createWidgetsWrapped() !void {
    debug("createWidgetsWrapped", .{});

    // Get the Active Screen
    var screen = try lvgl.getActiveScreen();

    // Create a Label Widget
    var label = try screen.createLabel();

    // Wrap long lines in the label text
    label.setLongMode(c.LV_LABEL_LONG_BREAK);

    // Interpret color codes in the label text
    label.setRecolor(true);

    // Center align the label text
    label.setAlign(c.LV_LABEL_ALIGN_CENTER);

    // Set the label text and colors
    label.setText(
        "#ff0000 HELLO# " ++    // Red Text
        "#00aa00 PINEDIO# " ++  // Green Text
        "#0000ff STACK!# "      // Blue Text
    );

    // Set the label width
    label.setWidth(200);

    // Align the label to the center of the screen, shift 30 pixels up
    label.alignObject(c.LV_ALIGN_CENTER, 0, -30);
}
```

Zig works well with C
So importing LVGL Types and Functions into Zig is a piece of cake
Zig will check for Null Pointers and other Runtime Problems that we commonly miss
So our LVGL App will become safer

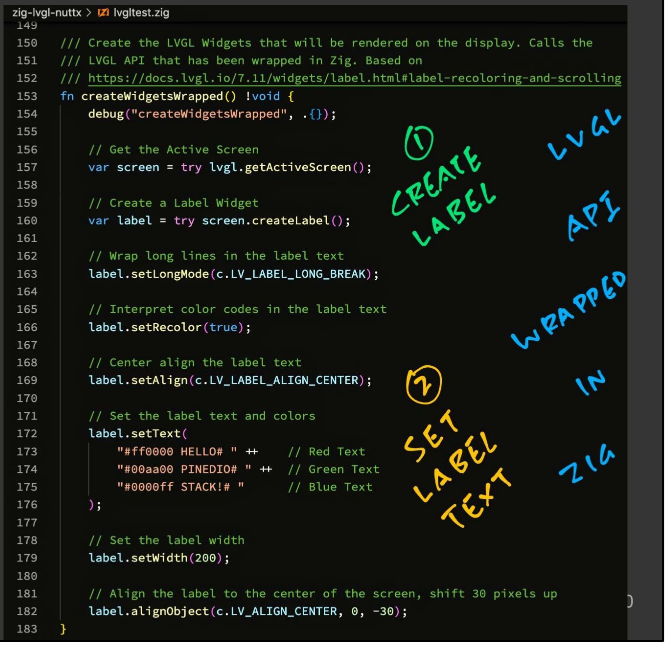
Zig can infer the Missing Types from our code
So there’s less repetition and clutter in our code
Easier to focus

Zig will even convert automatically a program from C to Zig!
Which makes a good reference for studying the
Nuances between C and Zig

Compare C and Zig


apps > examples > lvgltest > C lvgltest.c > create_widgets(void)
① CREATE LABEL
② SET LABEL TEXT
LVGL APP TO BE CONVER-TED TO ZIG

```c
static void create_widgets(void)
{
  // Get the Active Screen
  lv_obj_t *screen = lv_scr_act();

  // Create a Label Widget
  lv_obj_t *label = lv_label_create(screen, NULL);

  // Wrap long lines in the label text
  lv_label_set_long_mode(label, LV_LABEL_LONG_BREAK);

  // Interpret color codes in the label text
  lv_label_set_recolor(label, true);

  // Center align the label text
  lv_label_set_align(label, LV_LABEL_ALIGN_CENTER);

  // Set the label text and colors
  lv_label_set_text(
    label,
    "#ff0000 HELLO# "    //  Red Text
    "#00aa00 PINEDIO# "  //  Green Text
    "#0000ff STACK!# "   //  Blue Text
  );

  // Set the label width
  lv_obj_set_width(label, 200);

  // Align the label to the center of the screen, shift 30 pixels up
  lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, -30);
      You, 3 months ago • LVGL Label OK
```



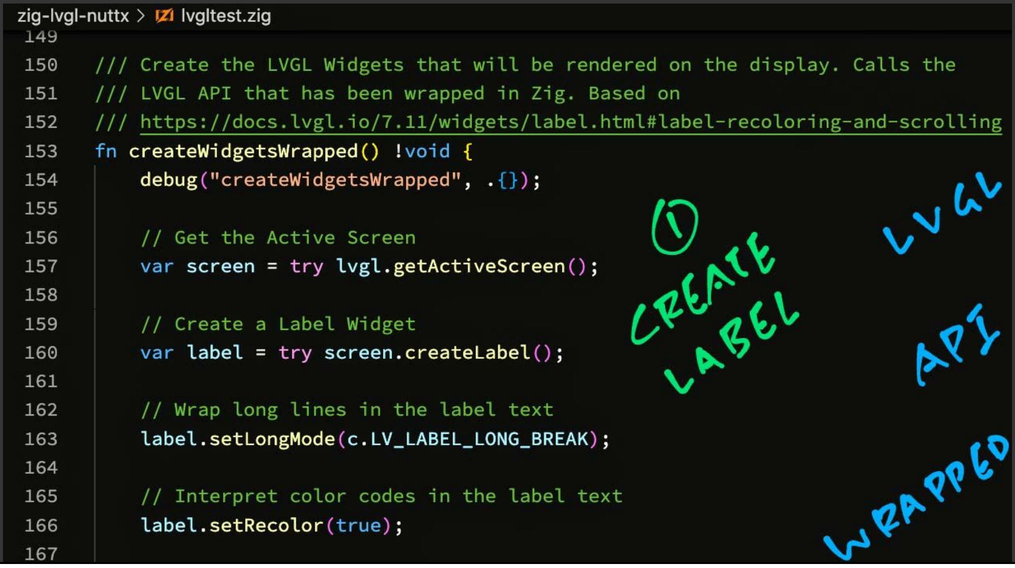
```zig
/// Create the LVGL Widgets that will be rendered on the display. Calls the
/// LVGL API that has been wrapped in Zig. Based on
/// https://docs.lvgl.io/7.11/widgets/label.html#label-recoloring-and-scrolling
fn createWidgetsWrapped() !void {
    debug("createWidgetsWrapped", .{});

    // Get the Active Screen
    var screen = try lvgl.getActiveScreen();

    // Create a Label Widget
    var label = try screen.createLabel();

    // Wrap long lines in the label text
    label.setLongMode(c.LV_LABEL_LONG_BREAK);

    // Interpret color codes in the label text
    label.setRecolor(true);

    // Center align the label text
    label.setAlign(c.LV_LABEL_ALIGN_CENTER);

    // Set the label text and colors
    label.setText(
        "#ff0000 HELLO# " ++    // Red Text
        "#00aa00 PINEDIO# " ++  // Green Text
        "#0000ff STACK!# "      // Blue Text
    );

    // Set the label width
    label.setWidth(200);

    // Align the label to the center of the screen, shift 30 pixels up
    label.alignObject(c.LV_ALIGN_CENTER, 0, -30);
}
```

When we compare the C and Zig versions of our LVGL App
Yep the Zig version looks simpler
Less repetition, less clutter!

Yet Zig is actually doing more behind the scenes
Like watching out for Null Pointers
Underflow, Overflow
Array out of bounds

If something goes wrong at runtime
Zig will halt our program
With a Panic Message
And show a helpful Stack Trace

LVGL in Zig: Create a Label



zig-lvgl-nuttx > lvgltest.zig

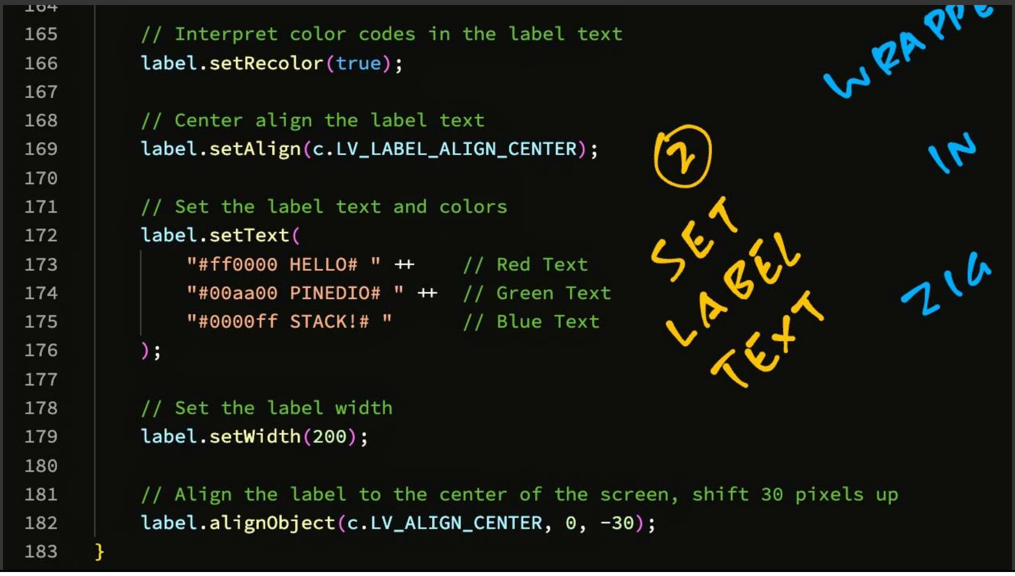
```zig

/// Create the LVGL Widgets that will be rendered on the display. Calls the
/// LVGL API that has been wrapped in Zig. Based on
/// https://docs.lvgl.io/7.11/widgets/label.html#label-recoloring-and-scrolling
fn createWidgetsWrapped() !void {
    debug("createWidgetsWrapped", .{});

    // Get the Active Screen
    var screen = try lvgl.getActiveScreen();

    // Create a Label Widget
    var label = try screen.createLabel();

    // Wrap long lines in the label text
    label.setLongMode(c.LV_LABEL_LONG_BREAK);

    // Interpret color codes in the label text
    label.setRecolor(true);

```

Let us take a peek at our Zig code
We no longer specify the LVGL Types
Like for “screen” and “label”
Zig infers the LVGL Types for us
Our code looks less cluttered now

Notice that we used “try” when calling the LVGL Functions?
This is how we Handle Errors in Zig
If any of these functions return an Error,
The execution stops and the Error is returned to the Caller

That is why we specify the Return Type with a “Bang”
Or Exclamation Mark or Exclamation Point
It means that this Function might return an Error
Nice tidy way to handle errors!

LVGL in Zig: Set Label Text and Properties



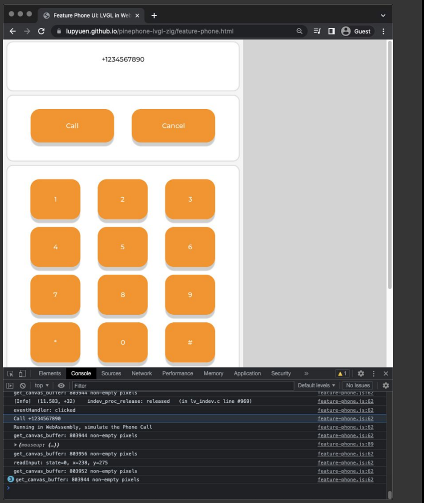
```
    // Interpret color codes in the label text
    label.setRecolor(true);

    // Center align the label text
    label.setAlign(c.LV_LABEL_ALIGN_CENTER);

    // Set the label text and colors
    label.setText(
        "#ff0000 HELLO# " ++    // Red Text
        "#00aa00 PINEDIO# " ++  // Green Text
        "#0000ff STACK!# "      // Blue Text
    );

    // Set the label width
    label.setWidth(200);

    // Align the label to the center of the screen, shift 30 pixels up
    label.alignObject(c.LV_ALIGN_CENTER, 0, -30);
}
```

The rest of the code looks similar to C.
Except that we moved Label to the front
So it appears more “Object-Like”

“++” is for String Concatenation
But it works only for Constant Strings at Compile-Time
So it is safe

Preview Smartphone Apps with WebAssembly

- We can create Touchscreen Apps that will work like a regular Smartphone App
- And preview in a Web Browser with WebAssembly
- By compiling LVGL Library to WebAssembly with Zig Compiler (based on Clang)
- [Demo Video]



```
get_canvas_buffer: 803944 non-empty pixels                                  feature-phone.js:62
[Info]  (11.583, +32)    indev_proc_release: released   (in lv_indev.c line #969)  feature-phone.js:62
eventHandler: clicked                                                        feature-phone.js:62
Call +1234567890                                                             feature-phone.js:62
Running in WebAssembly, simulate the Phone Call                              feature-phone.js:62
get_canvas_buffer: 803944 non-empty pixels                                   feature-phone.js:62
▶ {mouseup: {…}}                                                             feature-phone.js:89
get_canvas_buffer: 803956 non-empty pixels                                   feature-phone.js:62
readInput: state=0, x=238, y=275                                             feature-phone.js:62
get_canvas_buffer: 803952 non-empty pixels                                   feature-phone.js:62
3 get_canvas_buffer: 803944 non-empty pixels                                 feature-phone.js:62
>
```

lupyuen.github.io/nuttx

That was a quick overview of Zig

There's a problem with Smartphone Apps
We need an Actual Smartphone
To test the Smartphone App!

What if we could test Smartphone Apps
In a Web Browser?
That would be incredibly helpful
For Learners!

We have Zig
And Zig compiles to WebAssembly
So yes
We can run our Apps
In the Web Browser!

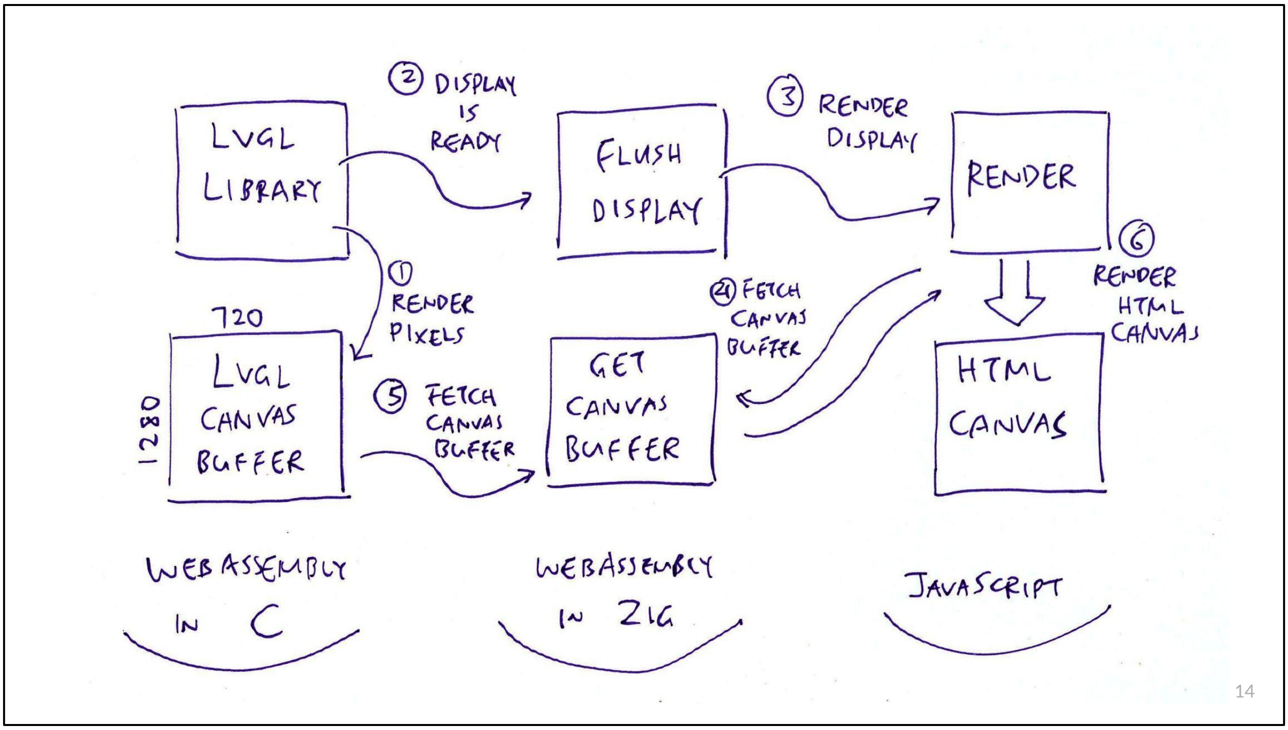
Here is an app
Created in Zig
That runs in the Web Browser
And on an Actual PinePhone

(Demo)
This is great for learners because we can

Tweak and test our app in a web browser
Before testing on an actual phone

And a Web Browser is convenient for
Sharing and collaborating
On our apps

Maybe someday we'll even
Allow apps to be created directly
Inside the browser
By dragging
And dropping
User interface controls



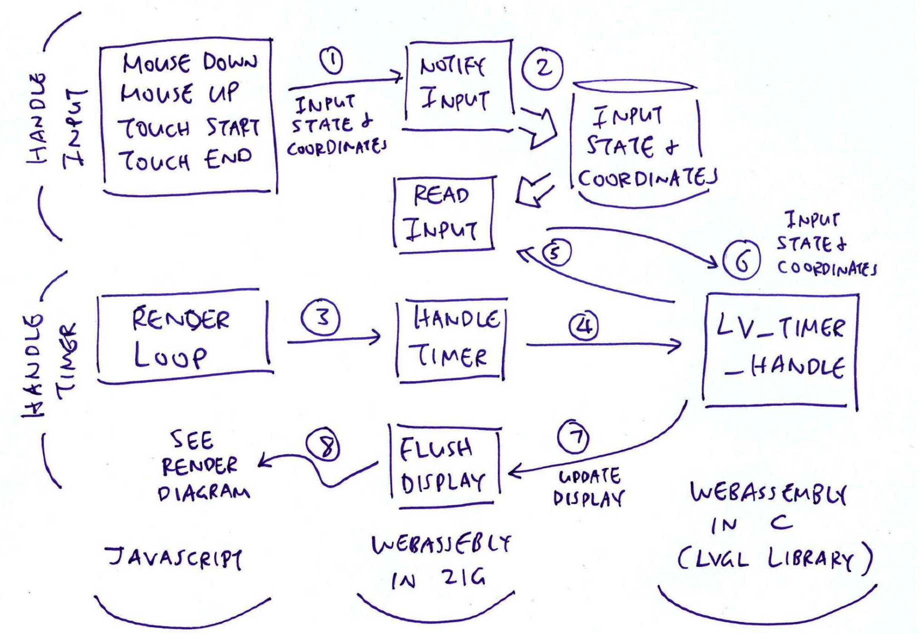
It looks messy
But this is how it works

We compile the LVGL Library
From C to WebAssembly
With the Zig Compiler

Which works because
Inside Zig Compiler
Is the Clang Compiler

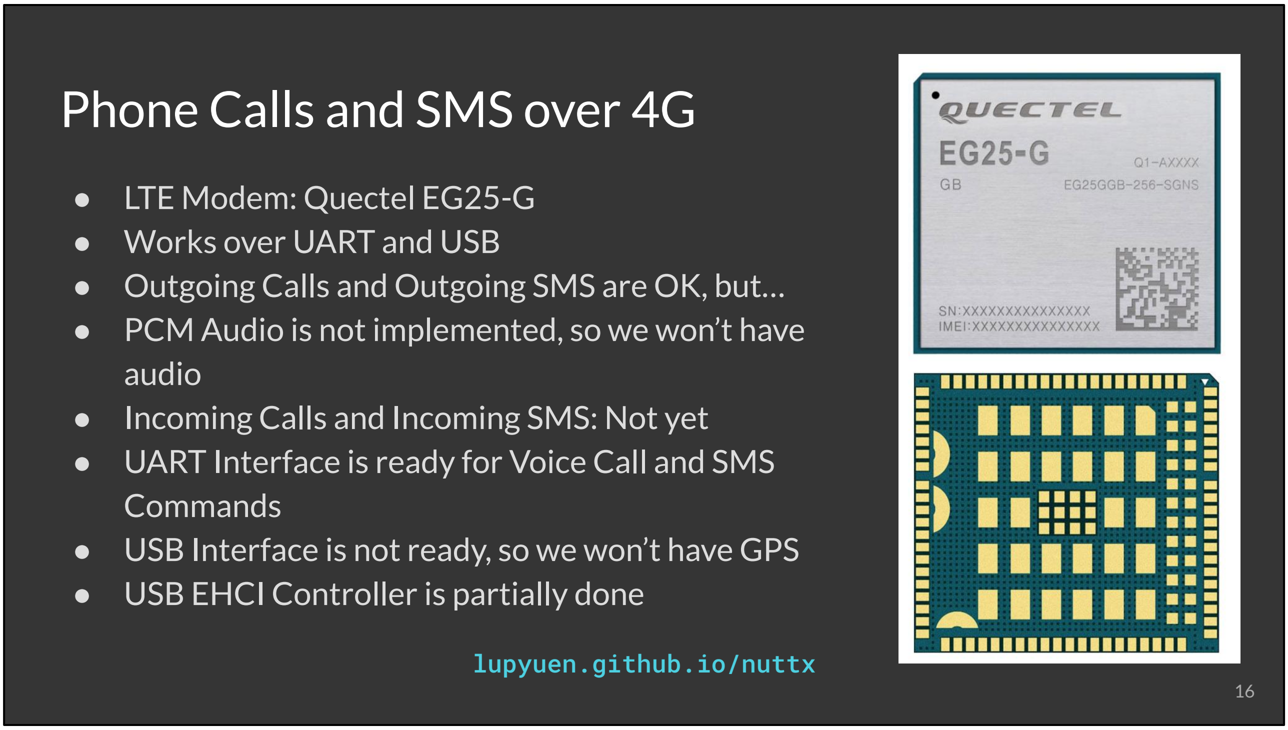
Then we integrate LVGL to the
Web Browser
With some Glue Code
In Zig and JavaScript

This is for rendering LVGL Graphics



Then we handle the Touch Events
Passing them from JavaScript
To Zig
To LVGL

This is a fun experiment
That might change the way
We write Smartphone Apps!



Phone Calls and SMS over 4G

- LTE Modem: Quectel EG25-G
- Works over UART and USB
- Outgoing Calls and Outgoing SMS are OK, but…
- PCM Audio is not implemented, so we won't have audio
- Incoming Calls and Incoming SMS: Not yet
- UART Interface is ready for Voice Call and SMS Commands
- USB Interface is not ready, so we won't have GPS
- USB EHCI Controller is partially done

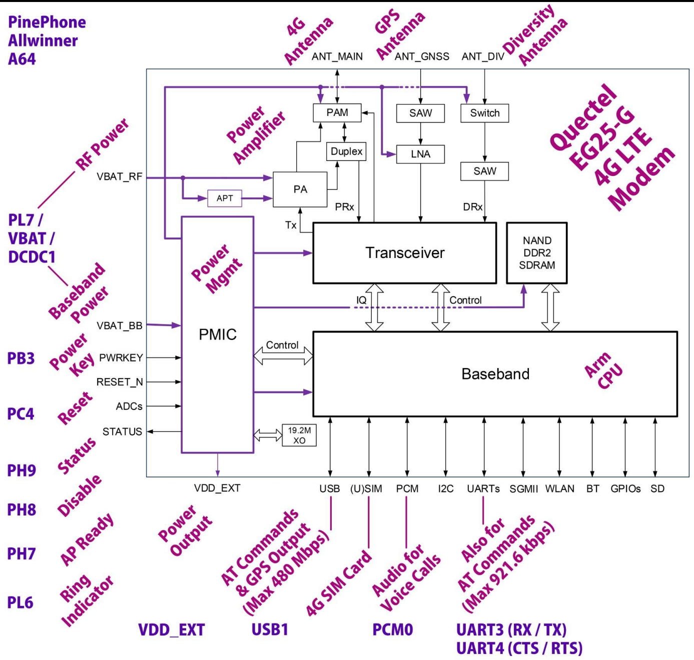
`lupyuen.github.io/nuttx`

What makes a Smartphone
A Phone?

Let us not forget the
4G LTE Modem
That is inside PinePhone!

This Quectel LTE Modem
Works like any typical LTE Modem
It uses AT Commands

Which works well with NuttX
For making Phone Calls
And sending SMS Text Messages

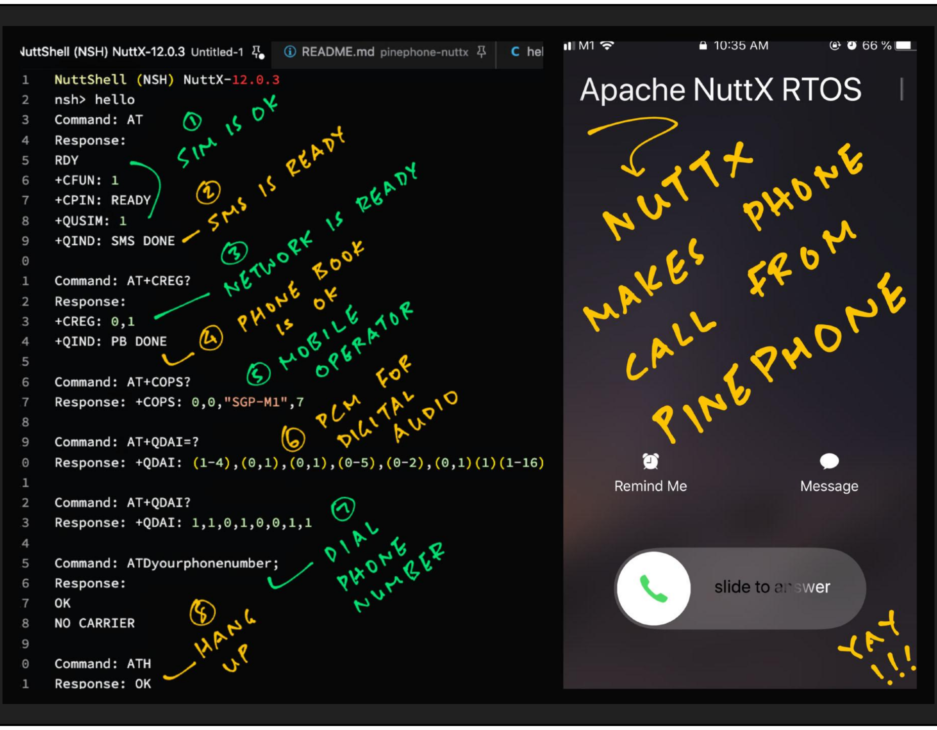


Quectel EG25-G Hardware Design

Here is a Fun Fact about Smartphones
There is an Entire Computer inside our Smartphone!

Inside the LTE Modem
That talks to the 4G Network
We have an Arm CPU
With Serial Ports
Digital Audio
USB
Wireless
GPS

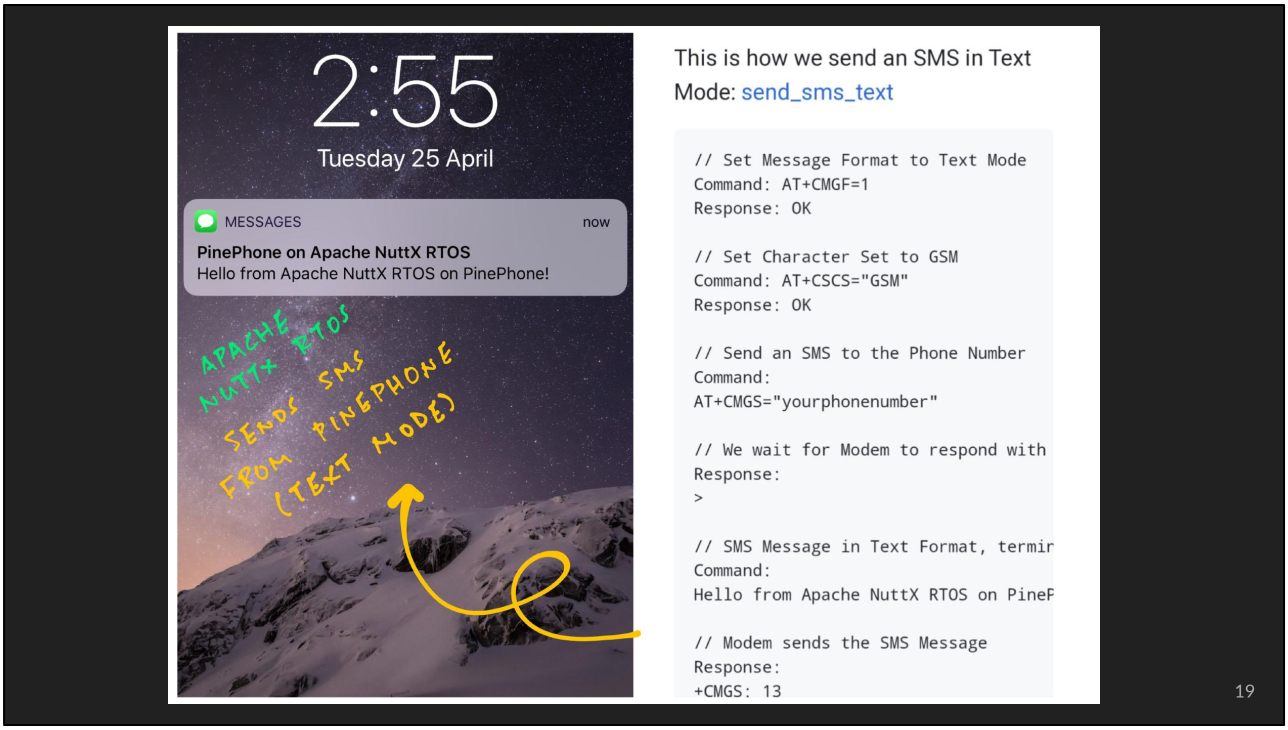
This is a computer in itself!
Fantastic opportunity
For learning
Something totally new
Inside a Smartphone



```
NuttShell (NSH) NuttX-12.0.3
nsh> hello
Command: AT
Response:
RDY
+CFUN: 1
+CPIN: READY
+QUSIM: 1
+QIND: SMS DONE

Command: AT+CREG?
Response:
+CREG: 0,1
+QIND: PB DONE

Command: AT+COPS?
Response: +COPS: 0,0,"SGP-M1",7

Command: AT+QDAI=?
Response: +QDAI: (1-4),(0,1),(0,1),(0-5),(0-2),(0,1)(1)(1-16)

Command: AT+QDAI?
Response: +QDAI: 1,1,0,1,0,0,1,1

Command: ATDyourphonenumber;
Response:
OK
NO CARRIER

Command: ATH
Response: OK
```

Here we see the AT Commands
Sent by NuttX to the LTE Modem
For making a Phone Call

Yep NuttX works OK
For making Phone Calls
Though the Audio Part
Is not quite there yet



2:55

Tuesday 25 April

MESSAGES now

PinePhone on Apache NuttX RTOS
Hello from Apache NuttX RTOS on PinePhone!

APACHE NUTTX RTOS SENDS SMS FROM PINEPHONE (TEXT MODE)

This is how we send an SMS in Text Mode: send_sms_text

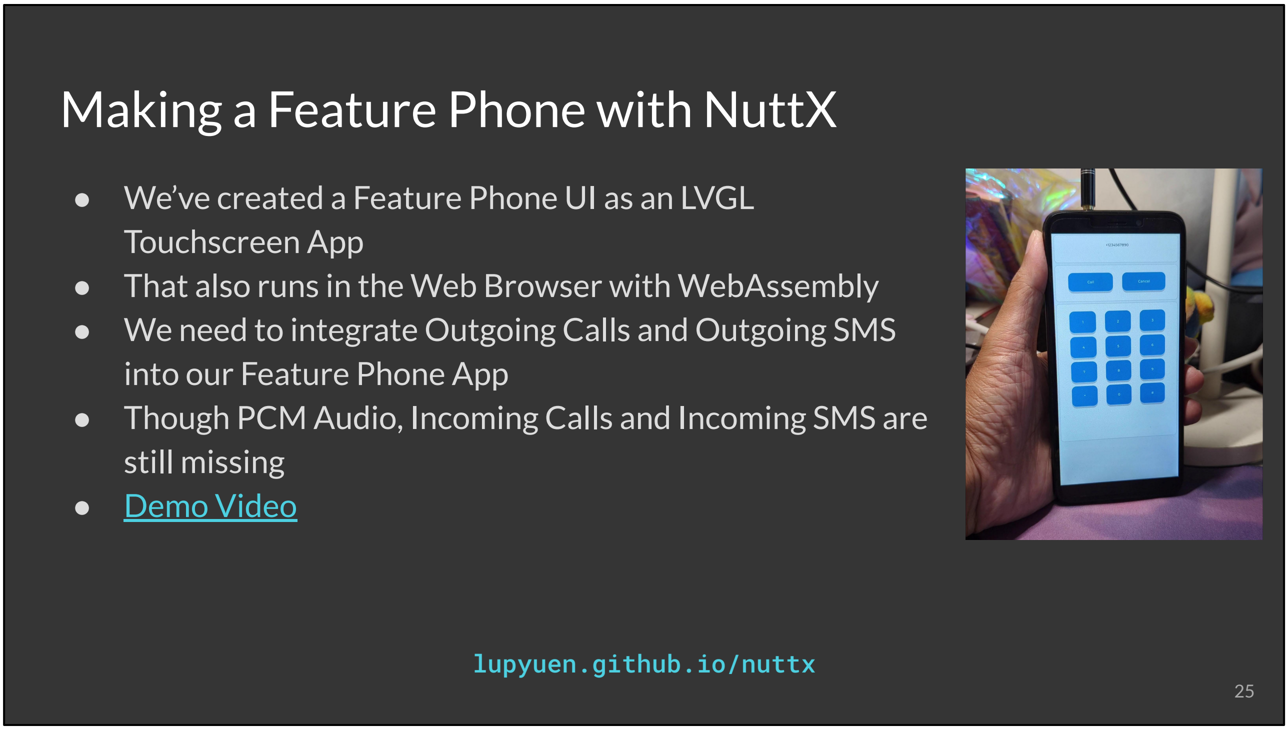
```
// Set Message Format to Text Mode
Command: AT+CMGF=1
Response: OK

// Set Character Set to GSM
Command: AT+CSCS="GSM"
Response: OK

// Send an SMS to the Phone Number
Command:
AT+CMGS="yourphonenumber"

// We wait for Modem to respond with
Response:
>

// SMS Message in Text Format, termir
Command:
Hello from Apache NuttX RTOS on PineP

// Modem sends the SMS Message
Response:
+CMGS: 13
```

19

And here are the AT Commands
For sending an SMS Text Message

This is just hilarious
Inside a Smartphone
We are still sending the same AT Commands
That have been around since
1981!
That’s 42 years ago!

Now with NuttX
Our Learners can truly appreciate
The Power of AT Commands
Inside our Smartphones



Making a Feature Phone with NuttX

- We've created a Feature Phone UI as an LVGL Touchscreen App
- That also runs in the Web Browser with WebAssembly
- We need to integrate Outgoing Calls and Outgoing SMS into our Feature Phone App
- Though PCM Audio, Incoming Calls and Incoming SMS are still missing
- Demo Video

lupyuen.github.io/nuttx

Remember the Feature Phones from 1999?
The plain and simple Nokia Phones
That will only make
Phone Calls and
Send Text Messages?

We are almost ready to turn NuttX
Into a Feature Phone!

We have the Feature Phone App
Created with Zig and LVGL Library
Tested in the Web Browser
Running on PinePhone

Outgoing Phone Calls and
Text Messages over 4G
Are working OK

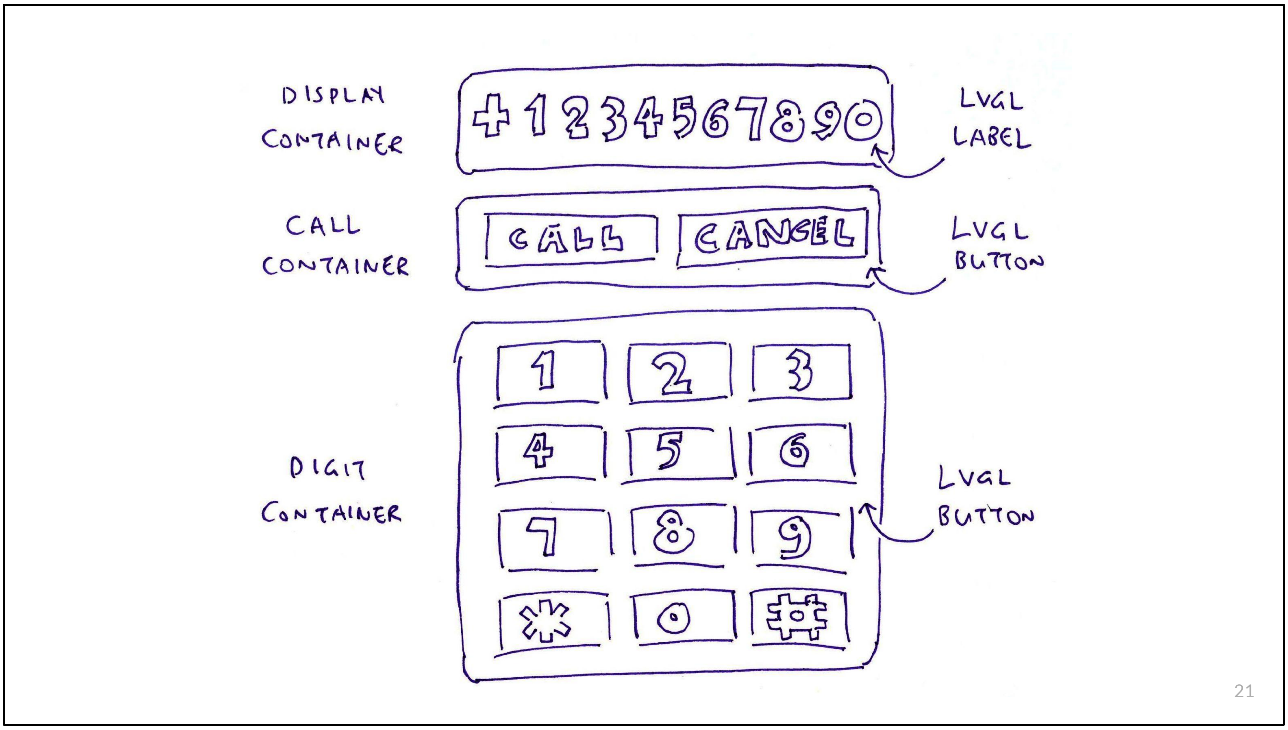
Now we need to work on the
Incoming Phone Calls,
Text Messages
And Digital Audio

(Demo)

Remember the Zig app we saw earlier
Running in a Web Browser?
Here is the same app running on
A real PinePhone

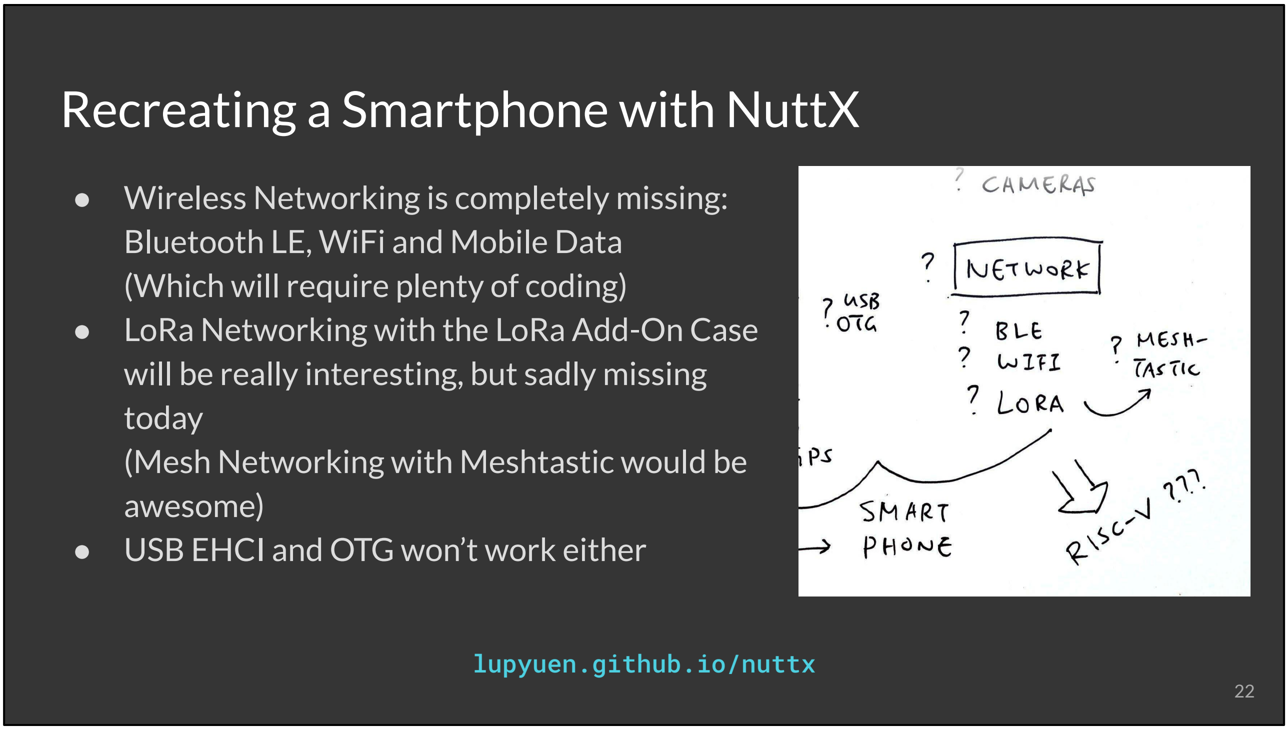
This Might sound strange
And counterintuitive
Why are we're turning a smartphone
Into a Feature Phone
Like regressing back to the past
Going retro

When we structure this as a
Learning Experience
Actually it makes a lot of sense
A smartphone is a phone after all
Hence we start with the basics
Making phone calls
Sending text messages
Then the rest will follow naturally
Mobile networking
Smartphone apps



It’s amazing that
Learners will be able to
Understand the Inner Workings of
Phone Calls
And Text Messages
Inside out!

And maybe inspire our Learners
To create their own
Feature Phones!


? CAMERAS
? NETWORK
? USB OTG
? BLE
? WIFI
? LORA
? MESH-TASTIC
PS
SMART PHONE
RISC-V ???

Recreating a Smartphone with NuttX

- Wireless Networking is completely missing: Bluetooth LE, WiFi and Mobile Data (Which will require plenty of coding)
- LoRa Networking with the LoRa Add-On Case will be really interesting, but sadly missing today (Mesh Networking with Meshtastic would be awesome)
- USB EHCI and OTG won't work either

lupyuen.github.io/nuttx

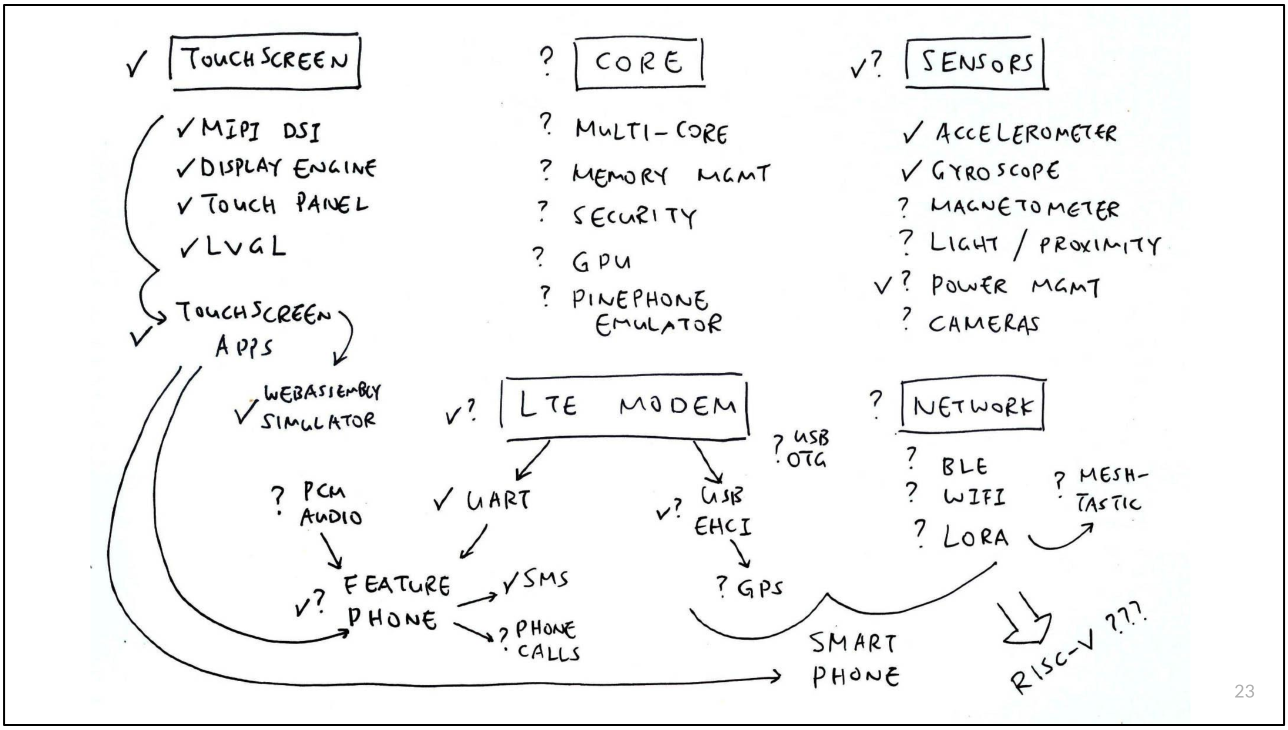
Yet we have plenty of room to grow
For Advanced Learners

They can build the Bluetooth
WiFi and Mobile Data Drivers
For NuttX

I am particularly fond of
LoRa
That's the Long Range
Low Power
Low Bandwidth
Wireless Radio Network

PinePhone has a Back Cover
With a LoRa Chip inside
Absolutely brilliant!

Would be great to have
LoRa Mesh Networking
On a Smartphone
An alternative
Peer-to-peer
Wireless Comms Network



But a Modern Smartphone has so many things inside
How close are we
In getting everything
To work with NuttX?

This is our Roadmap for
NuttX on PinePhone
The features we have built
The features we are going to build
And the features we built halfway through

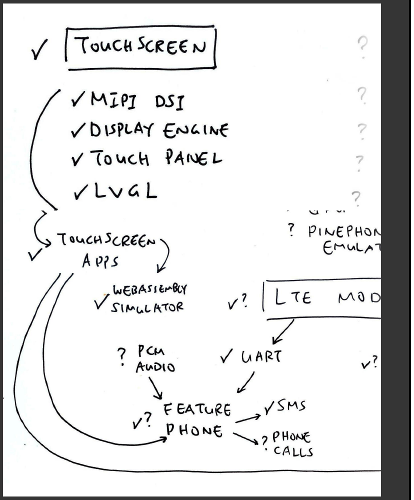
The Touchscreen works great
Even with Touchscreen Apps

Our support for Phone Sensors
Is a little spotty

We have some glaring gaps
In our Roadmap
USB
Networking
Core Security Features

Our SmartPhone Roadmap

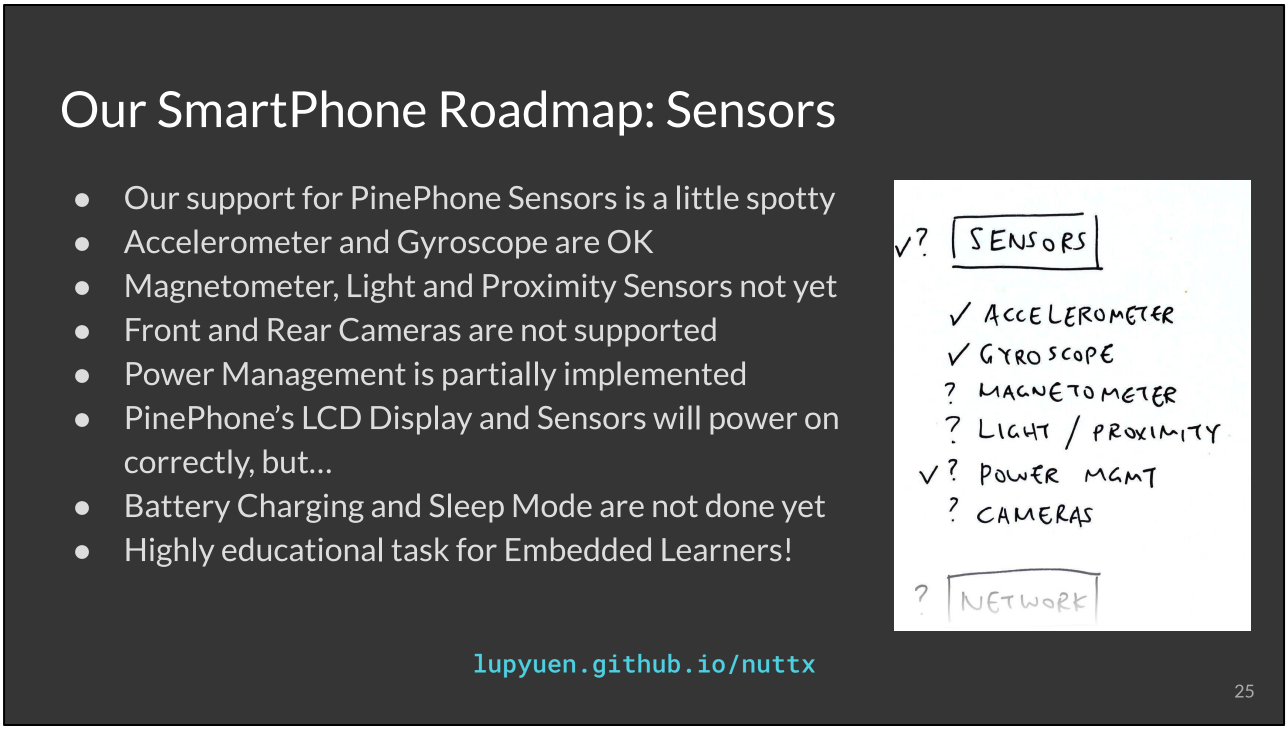
- Touchscreen Apps are well-supported
- LTE Modem is partially supported
- Feature Phone: Almost there
- Smartphone: Needs work
- Sensors: Partially supported



lupyuen.github.io/nuttx

That is because
We are a very tiny team
Building NuttX on PinePhone

Let’s talk about the
Sensors on PinePhone
Why they are especially interesting
For Embedded Learners



Our SmartPhone Roadmap: Sensors

- Our support for PinePhone Sensors is a little spotty
- Accelerometer and Gyroscope are OK
- Magnetometer, Light and Proximity Sensors not yet
- Front and Rear Cameras are not supported
- Power Management is partially implemented
- PinePhone's LCD Display and Sensors will power on correctly, but…
- Battery Charging and Sleep Mode are not done yet
- Highly educational task for Embedded Learners!

lupyuen.github.io/nuttx

Those of us familiar with Embedded Development
Arduino and IoT Microcontrollers
Quite often we connect sensors to our
Microcontroller Boards
So we can program and test the Sensors
Plug in, plug out on a Breadboard
Again and again

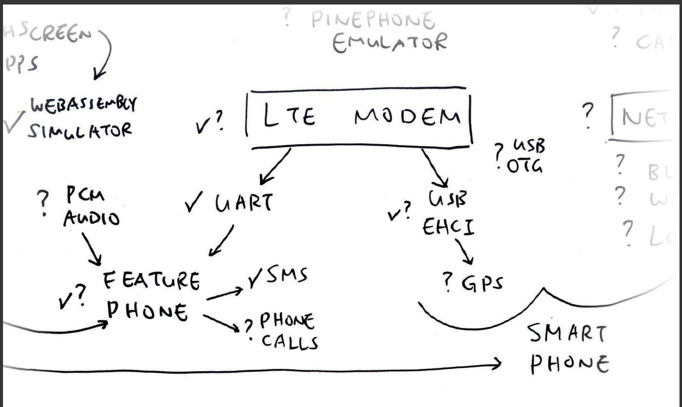
With a Smartphone
There's no need to connect sensors!
A whole bunch of sensors are already inside
Sensors for Gyroscope, Accelerometer, Temperature
Magnetometer, Light, Proximity
Even GPS

Much easier for learning
And experimenting

Not all drivers are available in NuttX today
But it would be an awesome
Educational Exercise
For students to contribute
The missing NuttX Drivers!

Our Smartphone Roadmap: LTE Modem and USB

- USB EHCI needs work



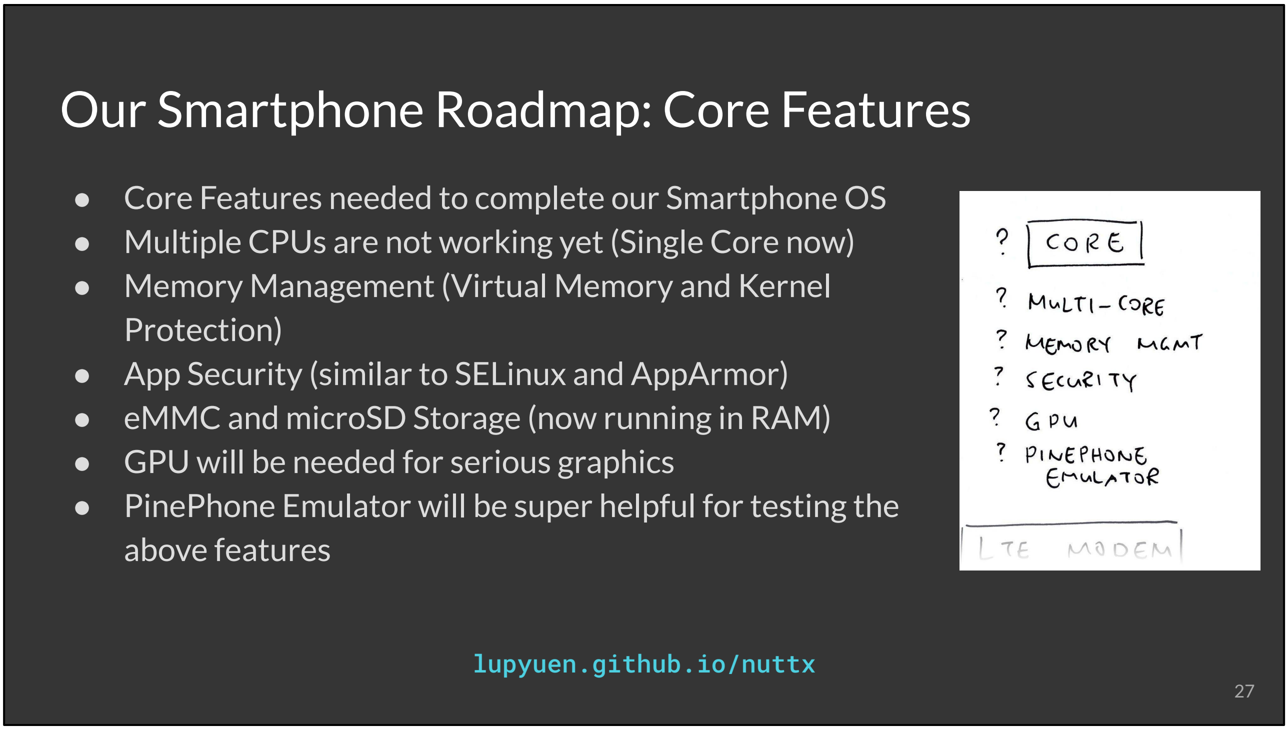
lupyuen.github.io/nuttx

Now we step up from Smartphone Sensors
And head into something
More challenging for Learners
USB in a Smartphone

USB is needed by the LTE Modem
For GPS to work
Because the GPS talks over the USB Interface

We are building the USB EHCI Driver
That’s the Enhanced Host Controller Interface
EHCI supports only USB Host Mode
Not USB Device Mode

For USB Device Mode
We need the USB On-The-Go Driver
For the Mentor Graphics USB Controller
Which we haven’t started



Our Smartphone Roadmap: Core Features

- Core Features needed to complete our Smartphone OS
- Multiple CPUs are not working yet (Single Core now)
- Memory Management (Virtual Memory and Kernel Protection)
- App Security (similar to SELinux and AppArmor)
- eMMC and microSD Storage (now running in RAM)
- GPU will be needed for serious graphics
- PinePhone Emulator will be super helpful for testing the above features

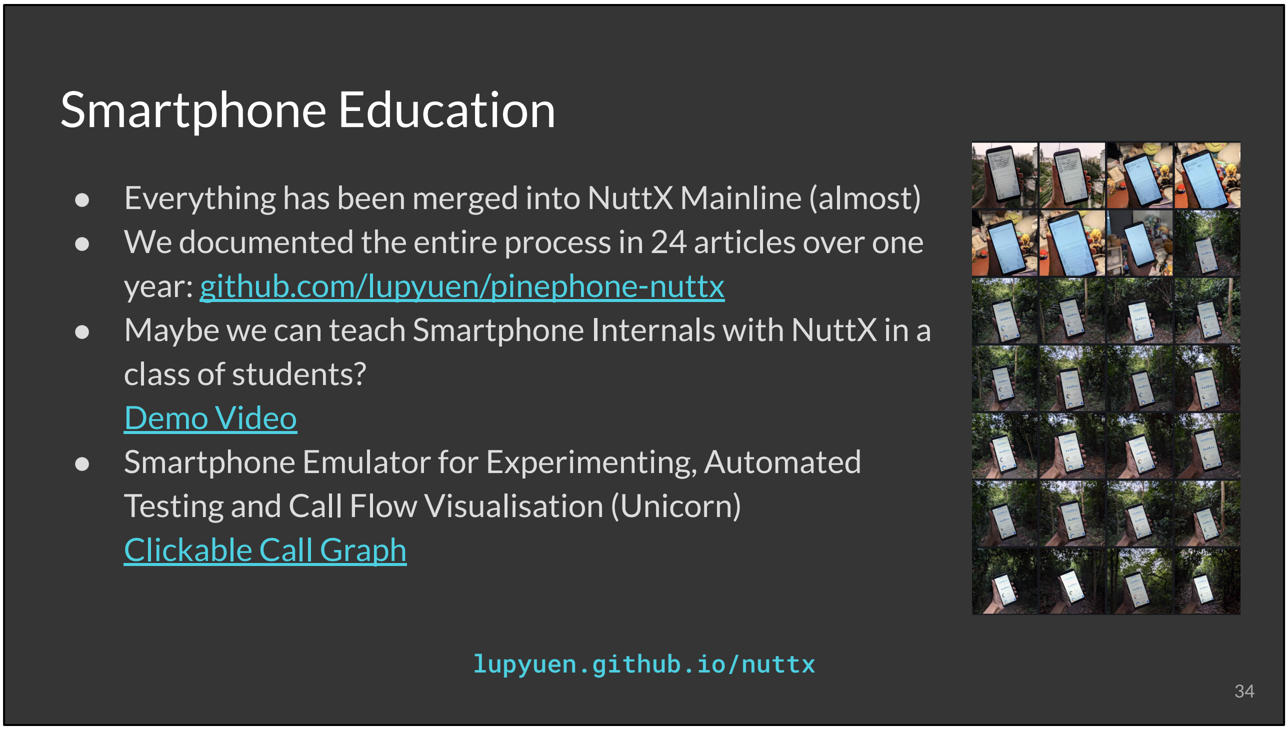
lupyuen.github.io/nuttx

There are many things that we take for granted in a
Modern Smartphone
Multiprocessing for Multiple CPUs
Security
For Memory and Apps
Graphical Processing Unit

For Advanced Learners
This could be a serious opportunity
For research and development

We have plenty of features in our Smartphone Roadmap
Would be great if we had a PinePhone Emulator
For testing all the new features

We might have a solution for that
We explain in a while



Smartphone Education

- Everything has been merged into NuttX Mainline (almost)
- We documented the entire process in 24 articles over one year: github.com/lupyuen/pinephone-nuttx
- Maybe we can teach Smartphone Internals with NuttX in a class of students?
  Demo Video
- Smartphone Emulator for Experimenting, Automated Testing and Call Flow Visualisation (Unicorn)
  Clickable Call Graph

lupyuen.github.io/nuttx

Now imagine a
Classroom of Students
Learning about Smartphone Internals
By booting NuttX on a phone
Are we ready teach NuttX in school?

The code that we have seen today
Is already available in the NuttX Mainline Repository
Open source, free to access on GitHub
Contributed by the fantastic NuttX Community

To teach NuttX in a classroom
We need textbooks right?
Well we have a huge stack of
24 articles
Written over the past year

Imagine a Thick Textbook on NuttX
With 24 chapters inside!

Covering all the topics we discussed today
Touchscreen, Smartphone Apps, Zig Programming
LTE Modem, Sensors, USB
Even a step-by-step guide

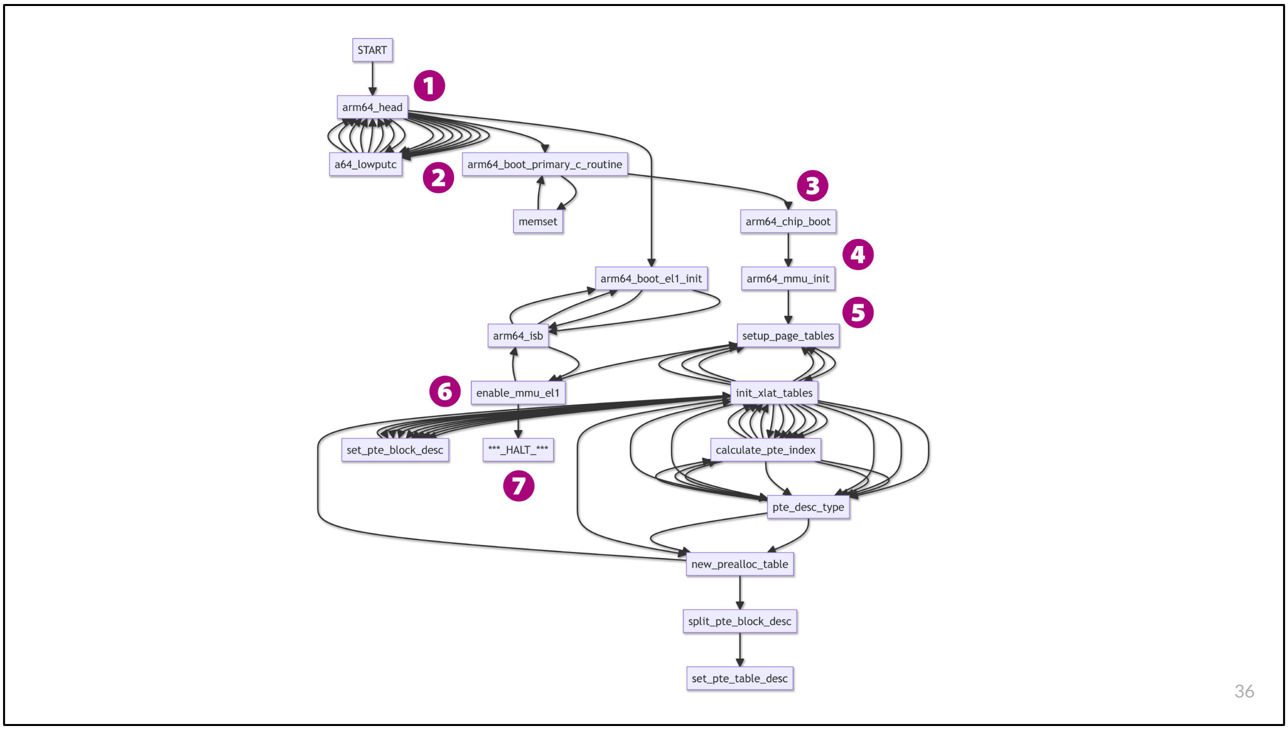
For submitting a Pull Request for NuttX

Our Textbook for NuttX
Should be sufficient
For self study
On Smartphone Internals

Sometimes it's helpful to have a
Smartphone Emulator
For Experimenting
And Automated Testing

We are building a PinePhone Emulator
With Unicorn Emulator
Which is a programmable variant of QEMU

Here is something fun we created
With Unicorn Emulator
A Clickable Call Graph
That visualises the Call Flow in NuttX



Have we ever wondered
What happens when an Operating System
Boots on our Computer?

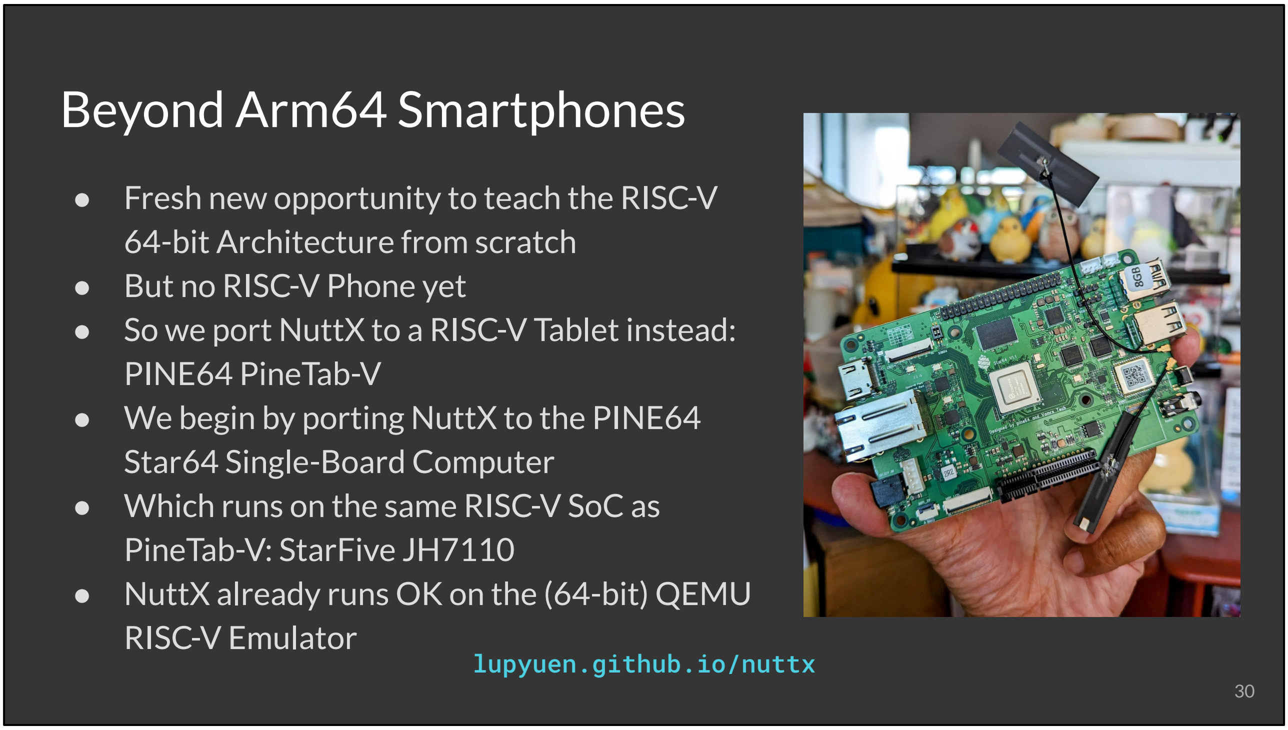
Thanks to Unicorn Emulator
We can visualise everything that happens
When NuttX boots on our Smartphone!

We are building a PinePhone Emulator
With Unicorn Emulator
Which is a programmable variant of QEMU

Unicorn emulates the entire Startup Process of NuttX
That is how we render the Call Graph that
Visualises the Call Flow
At NuttX Startup

This is interactive too
When we click a Box in the Call Graph
It jumps to the NuttX Source Code

This is a fun new way to use
Software Emulation
For Smartphone Education



Beyond Arm64 Smartphones

- Fresh new opportunity to teach the RISC-V 64-bit Architecture from scratch
- But no RISC-V Phone yet
- So we port NuttX to a RISC-V Tablet instead: PINE64 PineTab-V
- We begin by porting NuttX to the PINE64 Star64 Single-Board Computer
- Which runs on the same RISC-V SoC as PineTab-V: StarFive JH7110
- NuttX already runs OK on the (64-bit) QEMU RISC-V Emulator

`lupyuen.github.io/nuttx`

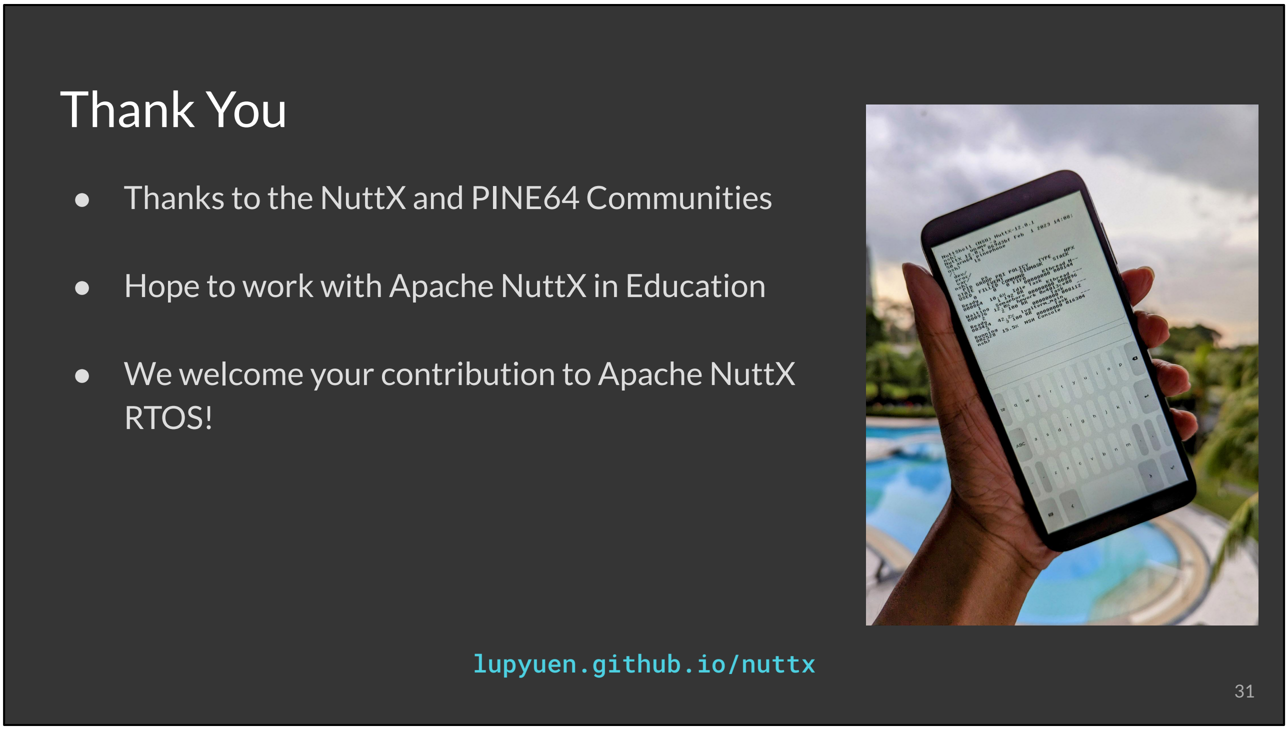
Arm64 is an extremely popular platform today
For all kinds of gadgets

Just follow the exact same steps
We’ve meticulously documented today
And NuttX will (probably) run on any Arm64 Device:
iPhone, Samsung Phones, Tablets, Laptops, Gaming Handhelds, …

Now we have a fresh new opportunity to teach
A new computing architecture from scratch:
The 64-bit RISC-V Architecture
Sadly there isn’t a RISC-V Phone yet
Thus we’ll port NuttX to a RISC-V Tablet instead:
The PineTab-V

We begin by porting NuttX to the
Star64 Single-Board Computer
Which runs on the same RISC-V SoC as PineTab-V:
The StarFive JH7110 SoC

NuttX works great on
RISC-V QEMU Emulator
So we are taking the same code
And running it on the Star64 SBC



Thank You

- Thanks to the NuttX and PINE64 Communities
- Hope to work with Apache NuttX in Education
- We welcome your contribution to Apache NuttX RTOS!

lupyuen.github.io/nuttx

Apache NuttX on PinePhone has been
An incredibly rewarding journey
Thanks to the awesome NuttX and Pine64 Communities

We have seen so many
Educational Opportunities today
I hope to do more with Apache NuttX in Education

And we welcome your contribution to the
Apache NuttX Project!

Thank you!