# Overview

- Modular Design
- Black LED Acrylic
- Powered by LED Charlieplexed Matrix
- Programmed in Python!
- Parts from Adafruit
- Parts Sourced Elsewhere
- Other Tools and Materials

## 3D Printing

- Parts List
- Slicing Parts
- Supports
- Build Plate Adhesion
- Acrylic Panels

## Assemble

- Build Steps
- Prep Wires
- Connect Cases
- Attach Frames
- Solder Driver
- Solder Address Pads
- Solder Connection headers
- Attach Charlie LED Matrices
- Attach Acrylic
- Plug in Jumpers
- Attach Vents
- Mount Pi or Pico
- Attach Bottom Case

## Code (Raspberry Pi)

- System Setup
- Prerequisite Software and Setup
- Download Project Code and Test
- Auto-Starting on Boot-Up

## Code (CircuitPython)

- If this is your first time using CircuitPython on the Pico RP2040 board, please begin with this guide which provides an introduction and download & installation instructions.
- “Do as I say, not as I do”
Overview

The Connection Machine (?) was a groundbreaking massively parallel supercomputer of the mid-1980s and 90s. Just as incredible as the machine’s performance was its industrial design: an ominous black cube-of-cubes, with system activity conveyed through thousands of red LEDs. It looks straight out of sci-fi…but it’s real!

There are a handful of truly iconic computer designs, like the distinctive “C” shape and built-in bench of early Cray supercomputers, but the Connection Machines are simply the best, period. I’m sorry, this isn’t mere subjective opinion, it’s irrefutable science. Look at it!

Not only are Connection Machines exceedingly scarce, with just a few dozen existing in museums and private collections, but their size and power requirements would make them impractical to own.

But now, thanks to 3-D printing and some Adafruit components, you can build a Little Connection Machine for your desk. This can house a Raspberry Pi 4 computer...or, if you don’t want or need the whole Linux-based computer and just want the blinkenlights...a Pico microcontroller board.
Modular Design
The case features an open design with lots of room to mount fans, sensors or even a speaker. Vents on the top, back and bottom help keep components cool.

The case is assembled with screws and press-fit parts that make it easy to customize. One could fit a USB battery bank to make a portable CM-1. Or, keeping on-theme with parallel processing, ambitious folks could probably fit a whole mini Linux cluster inside there using several Raspberry Pis!

Black LED Acrylic
Sheets of black LED acrylic slide over the matrices and diffuses the LEDs.
Powered by LED Charlieplexed Matrix

There are eight 9x16 “Charlieplexed” LED matrices inside this cube. Each is actually a pair of boards: an IS31FL3731 driver “backpack,” and then a 9x16 matrix that gets sandwiched atop this.

The matrices come in different colors if you really want…but classic red is authentic to the Connection Machine design.

Eight of these comprise a whopping 1,152 LEDs!

Programmed in Python!

If building with a Raspberry Pi, there is example code for three different animations. For the Pico microcontroller board, only the first of these is provided, but it’s the one most folks want anyway...
Random Blinkenlights

The classic!

Raspberry Pi: the number of lit elements (0–100%) depends on current CPU load. Since the program itself comprises part of that load, this will never be 0. Try opening a second window and doing a “sudo apt-get update” or “sudo apt-get upgrade” and watch the lights get angry!

Pico: you can configure a percentage or lit elements (0–100%) in the code. The lights simply blink randomly, it doesn’t reflect anything real-world...unless you decide to customize the code to read sensors or other nifty ideas.

Chaser Lights

This animation is inspired by the CM-5 in the Jurassic Park control room. This was a later Connection Machine model that took a different shape, but still had the LEDs...we’ve adapted that pattern to run on the original cube form-factor.

This doesn’t depend on CPU load or anything, it’s simply passive animation.

Audio Visualizer

A trippy sound-reactive display. This requires adding a USB microphone () and some additional software setup on the Raspberry Pi, explained later.
Parts from Adafruit

Required and/or recommended parts for this build:

The full design requires eight each of the LED drivers and matrices.

The “brains” can be one of:

- Any model of Raspberry Pi 4. The basic 1 GB model is adequate if you just want to run the demo code...or opt for bigger models if you expect to be doing real work. Older Pi models could also work, using USB micro-B instead of USB-C for power.
- A Raspberry Pi Pico microcontroller board.

One or other, you do not need both!

At the time this is being written, Pi 4 supplies are constrained...one could build with the Pico for now, and swap out for a “real” Pi later.

LED Charlieplexed Matrix - 9x16 LEDs - Red
These are 9x16 Charlieplexed LEDs designed to match with the Adafruit 16x9 Charlieplexed PWM LED Matrix Driver... https://www.adafruit.com/product/2947

Adafruit 16x9 Charlieplexed PWM LED Matrix Driver - IS31FL3731
The IS31FL3731 will let you get back to that classic LED matrix look, with a nice upgrade! This I2C LED driver chip has the ability to PWM each individual LED in a 16x9...
https://www.adafruit.com/product/2946
Black LED Diffusion Acrylic Panel - 10.2" x 5.1"
nice whoppin' rectangular slab of some lovely black acrylic to add some extra diffusion to your LED Matrix project. This material is 2.6mm (0.1") thick and is made of...
https://www.adafruit.com/product/4749

These wires and heat-shrink bits make for a luxurious build...though if you've been doing electronics for a while you might already have suitable bits around:

Premium Silicone Covered Female-Female Jumper Wires - 200mm x 40
These premium female-female jumper wires are handy for making wire harnesses or jumpering between headers on PCBs. They're 200mm (“7.8”) long and come loose as a pack of...
https://www.adafruit.com/product/4447

Heat Shrink Pack
Heat shrink is the duct tape of electronics, it keeps your stuff all safe and kept together. Especially when wiring and soldering, use heat shrink to add mechanical strength to cables....
https://www.adafruit.com/product/344

You do not need this Pico RP2040 board if building around a Raspberry Pi 4:
Raspberry Pi Pico RP2040
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...
https://www.adafruit.com/product/4864

You only need the following right-angle connector if building with Raspberry Pi 4. Pico does not require it.

Using this particular right-angle connector does unfortunately block the HDMI ports; it’s implied that such a Pi would be used “headless,” as in an OctoPrint () server. If you require HDMI out, you might find an upward-facing USB-C adapter elsewhere, a super-flexy cable, or more advanced folks might change up the wiring to allow using a 5V power adapter into the Pi’s breakout header (shared with the matrices).

Right Angle USB Type C Adapter - USB 3.1 Gen 4 Compatible
As technology changes and adapts, so does Adafruit, and speaking of adapting, this right angle adapter is USB C...
https://www.adafruit.com/product/4432

Parts Sourced Elsewhere
A well-stocked local hardware store might carry metric fasteners...but if not, these can be ordered online...

12 x M2.5x8mm Screw
M2.5x8mm Screw
4 x M2.5x12mm Screws
M2.5x12mm Screws

4 x M3x6mm Screw
M3x6mm Screw

Other Tools and Materials

You will need a soldering iron and related paraphernalia. None of the soldering is especially challenging…but with eight of those matrices, it does require patience.

Wire cutters/strippers, small screwdriver, typical electronics workbench stuff.

Most of the case is 3-D printed, so you’ll need a printer, or access to one. This can be made with basic black filament, or read on for alternatives…
3D Printing

Parts List
STL files for 3D printing are oriented to print "as-is" on FDM style machines. Parts are designed to 3D print without any support material. Original design source may be downloaded using the links below.

We used Sparkly Glitter PLA Filament from ERYONE ()

Minimum print bed dimensions are 150x150x150mm

Slicing Parts
Supports are required. Slice with setting for PLA material.

The parts were sliced using CURA using the slice settings below.

PLA filament 220c extruder
0.2 layer height (charlie-frame @ .15)
10% gyroid infill
60mm/s print speed
60c heated bed
Supports

Support Extrusion Width: .2
Support Density: 4%
Support Overhang Angle: 70
Support Z Height: .21
Interface: Off
Support Roof: Off
Support Pattern: Zig Zag

Build Plate Adhesion

Type: Brim
Line Count: 4
Brim on outside

Acrylic Panels

The four LED diffusion panels are slightly rectangular: 62 mm wide by 60.5 mm tall. There are a few ways these can be made:

- The score-and-snap method, using a straightedge and plastic scoring tool or back of an X-Acto blade. Wear eye protection!
- CNC milling.
- Laser cutting. If purchasing the LED diffusion acrylic from a retail source like TAP Plastics, some offer laser cutting services for a modest fee...or some makerspaces have a laser cutter for member use.
Assemble

Here's a wiring schematic for the Raspberry Pi version. Refer back to this when getting to later steps:

Color-coded by function:

<table>
<thead>
<tr>
<th>Color</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>+5V</td>
</tr>
<tr>
<td>Black</td>
<td>Ground</td>
</tr>
<tr>
<td>Blue</td>
<td>I2C SDA</td>
</tr>
<tr>
<td>Yellow</td>
<td>I2C SCL</td>
</tr>
</tbody>
</table>

Notice there are two groups of four matrices each. Each group gets wired to a different I2C bus. Specific pin numbers will be discussed later.

And here's a similar diagram for Pico RP2040:
Build Steps

Prep Wires

Silicone female-to-female jumper wires will be used to link the matrices to header pins on the Raspberry Pi or Pico.

5 each of four colors are sufficient to build two sets of wiring harnesses. Cut these in half to produce 40 half-wires total, and strip about 1 cm of insulation from all the cut wire ends.

Twist four same-color wires together facing the same way, then twist one more wire from the other direction to produce a four-way splitter cable. Solder where these wires all meet, then cover the exposed solder joint with heat-shrink tubing.

Repeat the process above to produce eight such 4-way splitters in total, two each for power, ground, I2C data and clock. You can optionally use a bit more heat-shrink to tidy up the wire bundles.
Connect Cases
Align the case parts and use four M2.5x12mm screws to fasten together.

Attach Frames
Use M2.5x8mm screws to attach the LED frames to the stand-offs inside the case.
Solder Driver
Solder the longer end of the header pins towards the back of the driver board. The short side of the header pins face the LEDs and the acrylic sheets.

Solder Address Pads
Each matrix controller within a group of four must be assigned a unique I2C address, settable with solder bridges on the back. The first matrix can be left alone, using the default address of 0x74. The other three should get a single solder bridge, setting them to address 0x75, 0x76 and 0x77, respectively. Repeat for both sets of four.

Assembling the matrix-and-driver sandwich is explained in greater detail in this guide.

Solder Connection headers
Solder four header pins to each matrix. Solder the shorter pins to the driver board. The longer pins face away from the driver board and connect to the female jumper wires.

EXPERT TIP: experienced builders will always connect all wiring outside the case first, then skip ahead to the software setup. Once everything is tested and known working, disconnect and re-assemble in the case as we’ll show next. This adds extra steps but is SO MUCH EASIER to troubleshoot than after everything’s enclosed!
Attach Charlie LED Matrices
Position each matrix in the order shown in the schematic at top, and with the four pin header connection towards the bottom. Leftmost matrix as seen from the front will be 0x74. From the back, these will be in the opposite order.

Attach Acrylic
The Black LED Acrylic slides over the matrices.
Plug in Jumpers
Connect each jumper to the corresponding header pin on each matrix, and to the Pi or Pico as shown in the corresponding wiring diagram at the top of this page.

Attach Vents
The back vents press fit into the walls around each cube.
Mount Pi or Pico

Use M3x6mm long screws to mount Raspberry Pi computer. The USB port align the cutout on the bottom case.

The Raspberry Pico mounts what an addition frame over the stand-offs on the bottom case.

Attach Bottom Case
Use four M2.5x8mm screws to attach the bottom case.

As mentioned on the Overview page, the USB right-angle connector will block the HDMI ports. This is fine for “headless” use, as with an OctoPrint server. If you require HDMI out, you’ll need to improvise with a different USB adapter or cable.
Glad you’re here! The Raspberry Pi code has all the bells and whistles. This will be discussed in a moment, but first…

System Setup

Basic Raspberry Pi setup is already covered in other guides... so if this is your first time, start there. You can do these initial steps with the Pi outside the case and without the matrices wired up.

Pi power users may have their own setup ritual, but for most folks we recommend using Raspberry Pi Imager () to create a bootable micro SD card.

Choose “Raspberry Pi OS Lite (32-bit)” as the operating system. If you require a GUI, use “Desktop (32-bit)” instead...but see prior notes that the HDMI ports may be inaccessible once in the case. Avoid the 64-bit OS versions for now...these are cutting-edge and often problematic.

Click the gear icon, or press Control+Shift+X (Windows) or ⌘+Shift+X (Mac), to show advanced options. Essential system info (hostname, WiFi network, etc.) can all be configured before the system is even booted the first time...it’s so much easier this way! Tip: remember to also set the Wireless LAN country to get WiFi working.
So suppose at this point you have a Raspberry Pi that’s booted and accessible via an ssh remote session...

Prerequisite Software and Setup

First is to edit the system /boot/config.txt file, which must be done as root via the “sudo” command:

```bash
sudo nano /boot/config.txt
```

(You can substitute your editor of preference in place of “nano.” I’m strange and actually like vi.)

Append the following two lines to the end of the file. Just copy-and-paste them verbatim, as the syntax is very persnickety:

```bash
dtoverlay=i2c-gpio,bus=2,i2c_gpio_scl=17,i2c_gpio_sda=27,i2c_gpio_delay_us=1
dtoverlay=i2c-gpio,bus=3,i2c_gpio_scl=23,i2c_gpio_sda=24,i2c_gpio_delay_us=1
```

Save changes to the file. This will enable two “soft” I2C buses on specific GPIO pins. You can use different pins if both the software and wiring are changed to match, but don’t do this unless other hardware you’re interfacing specifically demands it. Most users should stick with the plan.

Extra Steps for the Audio Visualizer Only

This optional part of the project requires the [Mini USB Microphone](https://example.com) mentioned on the Overview page. If you don’t have this part or don’t want the visualizer, you can skip ahead to “Continue Setup” below.

We’ll do the audio setup before the other software, as it then requires fewer reboots.

Enter these commands:

```bash
sudo apt-get install python3-pip libatlas-base-dev libportaudio2
pip3 install numpy pyaudio
```

Answer “Y” when asked about installing packages.

Following installation, one must then edit the system’s ALSA configuration, which is related to sound input and output:
sudo nano /usr/share/alsa/alsa.conf

(As before, substitute your editor of preference in place of nano.)

Look for these two lines, starting around line 76:

```plaintext
defaults.ctl.card 0
defaults.pcm.card 0
```

Change these to:

```plaintext
defaults.ctl.card 1
defaults.pcm.card 1
```

Save changes to this file and exit the editor.

**Continue Setup**

With or without the audio visualizer, all installations will require the following sequence of commands (please use the “Copy Code” button and paste into the Pi’s terminal window...some of these are single long lines that might appear wrapped here):

```plaintext
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python3-pip libopenjp2-7 subversion
sudo pip3 install --upgrade setuptools adafruit-python-shell psutil adafruit-
extended-bus adafruit-circuitpython-is31fl3731 pillow
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/
master/raspi-blinka.py
sudo python3 raspi-blinka.py
```

Answer “Y” when asked about upgrading or installing packages.

That last step—the raspi-blinka.py script—will ask to reboot the system when finished. Play along, answer “Y,” and log back into the system once it's rebooted.

FYI: Blinka is a package for running CircuitPython (microcontroller) code on “real” computers like Raspberry Pi. It allows leveraging hardware-specific libraries for talking to the LED matrices, for instance. Pretty cool. [There’s an entire guide](https://github.com/adafruit/Raspberry-Pi-Installer-Scripts) if you’d like to learn more.

**Download Project Code and Test**

All the system configuration is done, and now to fetch the project code:
Start with the simplest of the examples, some animated lights inspired by the CM-5 control room computer in Jurassic Park:

```
python chaser.py
```

If all goes well, this will simply run, lighting all 8 matrices with animated bit patterns. Press CONTROL+C to stop.

If you’re seeing a RuntimeError or ValueError message from the system, it’s usually one of these things:

- An SDA/SCL wire pair is swapped, either on the Raspberry Pi header or on one or more matrices.
- SDA and/or SCL are connected to the wrong pin(s).
- Address select pads on one or more matrices are not correctly set; multiple matrices on the same I2C bus are set to the same address.
- A poorly-soldered connection in one of the 4-way splitters.

Remember there are two I2C buses at play here, each with four matrices, each of which should have a unique address on that bus.

For posterity, here’s that wiring diagram again:
From a programming perspective, chaser.py is the simplest of the examples, as it doesn’t tie into any obscure system services…it just animates the lights. If creating your own programs for the Little Connection Machine, this is a good starting point.

Then there’s the classic supercomputer blinkenlights:

```bash
python cpuload.py
```

The amount of pixels lit is proportional to the current system workload. Left running in the background...when compiling and debugging two million lines of code to run a dinosaur theme park, you’ll see this really light up! CONTROL-C to stop.

If you have the USB microphone installed and did the extra setup earlier, you can try the third example:

```bash
python audio.py
```

This will react to music and sounds within earshot...higher frequencies toward the top of the matrices, lower frequencies toward the bottom. CONTROL+C to exit.

If this fails to run, double-check the ALSA configuration that was described earlier. It’s notoriously picky.

If this runs but the image appears jumbled, the LED matrices are arranged in the wrong order. The wiring diagram shows the address for each matrix.

### Auto-Starting on Boot-Up

The preferred way to do this nowadays is via the systemd service. Begin by creating a unit file for this service...

```bash
sudo nano /lib/systemd/system/cm1.service
```

Copy-and-paste the following into the new file, editing the “ExecStart” line to reference which of the Python scripts you’d like to run. Can also change the directory name, if you’ve moved or called it something else:

```ini
[Unit]
Description=CM-1 Service
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python /home/pi/LittleConnectionMachine/RaspberryPi/chaser.py

[Install]
WantedBy=multi-user.target
```

© Adafruit Industries
Save this file and exit the editor.

A few commands then enable this new service and will launch it on startup:

```
sudo systemctl daemon-reload
sudo systemctl enable cm1.service
sudo reboot
```

To change which program is loaded on startup, just edit `/lib/systemd/system/cm1.service` and change the ExecStart line, and reboot. No need to repeat the systemctl commands.

---

**Code (CircuitPython)**

If you’d be satisfied with just the blinkenlights part of the project (this is what most people are after!), we can do this with a Pico RP2040 microcontroller board replacing the Raspberry Pi 4. No complex operating system to boot up, it’s even easier to set up and use.

If you know your way around code and hardware a bit, actually any CircuitPython-compatible board with two I2C buses can work. The Pico is super inexpensive and readily available though, so it is used as the reference design.

If this is your first time using CircuitPython on the Pico RP2040 board, please begin with this guide () which provides an introduction and download & installation instructions.

So at this point, you should have CircuitPython installed on the Pico board.
To download the code for this project, click the “Download Project Bundle” button below. This brings together some required CircuitPython libraries, so everything is quickly and easily copied to the CIRCUITPY drive.

If you currently have a CircuitPython project on the Pico board, copy the files off to somewhere for safe keeping.

Unzip the project bundle archive, look inside the resulting folder (a couple layers down if needed), and copy the code.py file and lib folder to the CIRCUITPY drive. The drive contents should then appear like so (you can ignore the boot_out.txt file, it's auto-generated on startup):

```python
# SPDX-FileCopyrightText: 2022 Phillip Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# CircuitPython random blinkenlights for Little Connection Machine. For Raspberry
# Pi Pico RP2040, but could be adapted to other CircuitPython-capable boards
# with two or more I2C buses. Requires adafruit_bus_device and adafruit_is31fl3731
# libraries.

# This code plays dirty pool to get fast matrix updates and is NOT good code
to learn from, and might fail to work with future versions of the IS31FL3731
library. But doing things The Polite Way wasn't fast enough. Explained as
we go...

# pylint: disable=import-error
import random
import board
import busio
from adafruit_is31fl3731.matrix import Matrix as Display

BRIGHTNESS = 40  # CONFIGURABLE: LED brightness, 0 (off) to 255 (max)
PERCENT = 33  # CONFIGURABLE: amount of 'on' LEDs, 0 (none) to 100 (all)

# This code was originally written for the Raspberry Pi Pico, but should be
# portable to any CircuitPython-capable board WITH TWO OR MORE I2C BUSES.
# IS31FL3731 can have one of four addresses, so to run eight of them we
# need *two* I2C buses, and not all boards can provide that. Here's where
# you'd define the pin numbers for a board...
```
I2C1_SDA = board.GP18  # First I2C bus
I2C1_SCL = board.GP19
I2C2_SDA = board.GP16  # Second I2C bus
I2C2_SCL = board.GP17

# pylint: disable=too-few-public-methods
class FakePILImage:
    """Minimal class meant to simulate a small subset of a Python PIL image, so we can pass it to the IS31FL3731 image() function later. THIS IS THE DIRTY POOL PART OF THE CODE, because CircuitPython doesn't have PIL, it's too much to handle. That image() function is normally meant for robust "desktop" Python, using the Blinka package...but it's still present (but normally goes unused) in CircuitPython. Having worked with that library source, I know exactly what object members its looking for, and can fake a minimal set here...BUT THIS MAY BREAK IF THE LIBRARY OR PIL CHANGES!""
    def __init__(self):
        self.mode = "L"  # Grayscale mode in PIL
        self.size = (16, 9)  # 16x9 pixels
        self.pixels = bytearray(16 * 9)  # Pixel buffer

    def tobytes(self):
        """IS31 lib requests image pixels this way, more dirty pool.""
        return self.pixels

# Okay, back to business...
# Instantiate the two I2C buses. 400 KHz bus speed is recommended.
# Default 100 KHz is a bit slow, and 1 MHz has occasional glitches.
I2C = [
    busio.I2C(I2C1_SCL, I2C1_SDA, frequency=400000),
    busio.I2C(I2C2_SCL, I2C2_SDA, frequency=400000),
]
# Four matrices on each bus, for a total of eight...
DISPLAY = [
    Display(I2C[0], address=0x74, frames=(0, 1)),  # Upper row
    Display(I2C[0], address=0x75, frames=(0, 1)),
    Display(I2C[0], address=0x76, frames=(0, 1)),
    Display(I2C[0], address=0x77, frames=(0, 1)),
    Display(I2C[1], address=0x74, frames=(0, 1)),  # Lower row
    Display(I2C[1], address=0x75, frames=(0, 1)),
    Display(I2C[1], address=0x76, frames=(0, 1)),
    Display(I2C[1], address=0x77, frames=(0, 1)),
]

IMAGE = FakePILImage()  # Instantiate fake PIL image object
FRAME_INDEX = 0  # Double-buffering frame index

while True:
    # Draw to each display's "back" frame buffer
    for disp in DISPLAY:
        for pixel in range(0, 16 * 9):  # Randomize each pixel
            IMAGE.pixels[pixel] = BRIGHTNESS if random.randint(1, 100) <= PERCENT

    # Here's the function that we're NOT supposed to call in CircuitPython, but is still present. This writes the pixel data to the display's back buffer. Pass along our "fake" PIL image and it accepts it.
    disp.image(IMAGE, frame=FRAME_INDEX)

    # Then quickly flip all matrix display buffers to FRAME_INDEX
    for disp in DISPLAY:
        disp.frame(FRAME_INDEX, show=True)
    FRAME_INDEX ^= 1  # Swap buffers

    # This is actually the LESS annoying way to get fast updates. Other involved # writing IS31 registers directly and accessing intended-as-private methods
If everything’s connected correctly and the code and libraries installed in the right place, you should get the blinkies. And that’s it! Just plug it into USB any time you need it.

If things do not work, you’ll want to open a serial connection to the CircuitPython console (such as with the Mu editor) and see what error messages it’s displaying. This is explained in the introductory CircuitPython guide previously mentioned.

For posterity, here’s that wiring diagram, showing the connections for this board:

If you see a RuntimeError or ValueError message in the serial console, it’s usually one of these things:

- An SDA/SCL wire pair is swapped, either on the Pico board or on one or more matrices.
- SDA and/or SCL are connected to the wrong pin(s).
- Address select pads on one or more matrices are not correctly set; multiple matrices on the same I2C bus are set to the same address.
- A poorly-soldered connection in one of the 4-way splitters.

Remember there are two I2C buses at play here, each with four matrices, each of which should have a unique address on that bus.
“Do as I say, not as I do”

This code does things that would get you an “F” in any programming class, so please don’t look to it as a shining example of anything.

The adafruit_is31fl3731 library normally provides easy X/Y pixel-setting functions and robust exception handling. Any well-behaved code really should be using the functions provided there, as documented in this guide. If you want to display other things on your Little Connection Machine and don’t need super-frequent updates, please program it that way...or build the Raspberry Pi version.

In order to get fast updates on the LED matrices, this project’s code plays games with the adafruit_is31fl3731 library to dump data directly to the matrices, bypassing the aforementioned functions. It relies on the fact that it’s just a random bit pattern, that there’s no coherent order that must be followed to produce an image. Handling that would add inordinate complexity to this code.

Such complexity is something the Raspberry Pi is perfectly equipped to handle... and if you go looking into the Pi code, you’ll see it treats all the matrices as if a single contiguous image, with numerous drawing and image-loading functions available through PIL, the Python Imaging Library. The CircuitPython blinky code is just parlor tricks...and that’s why it’s the only example we provide there.