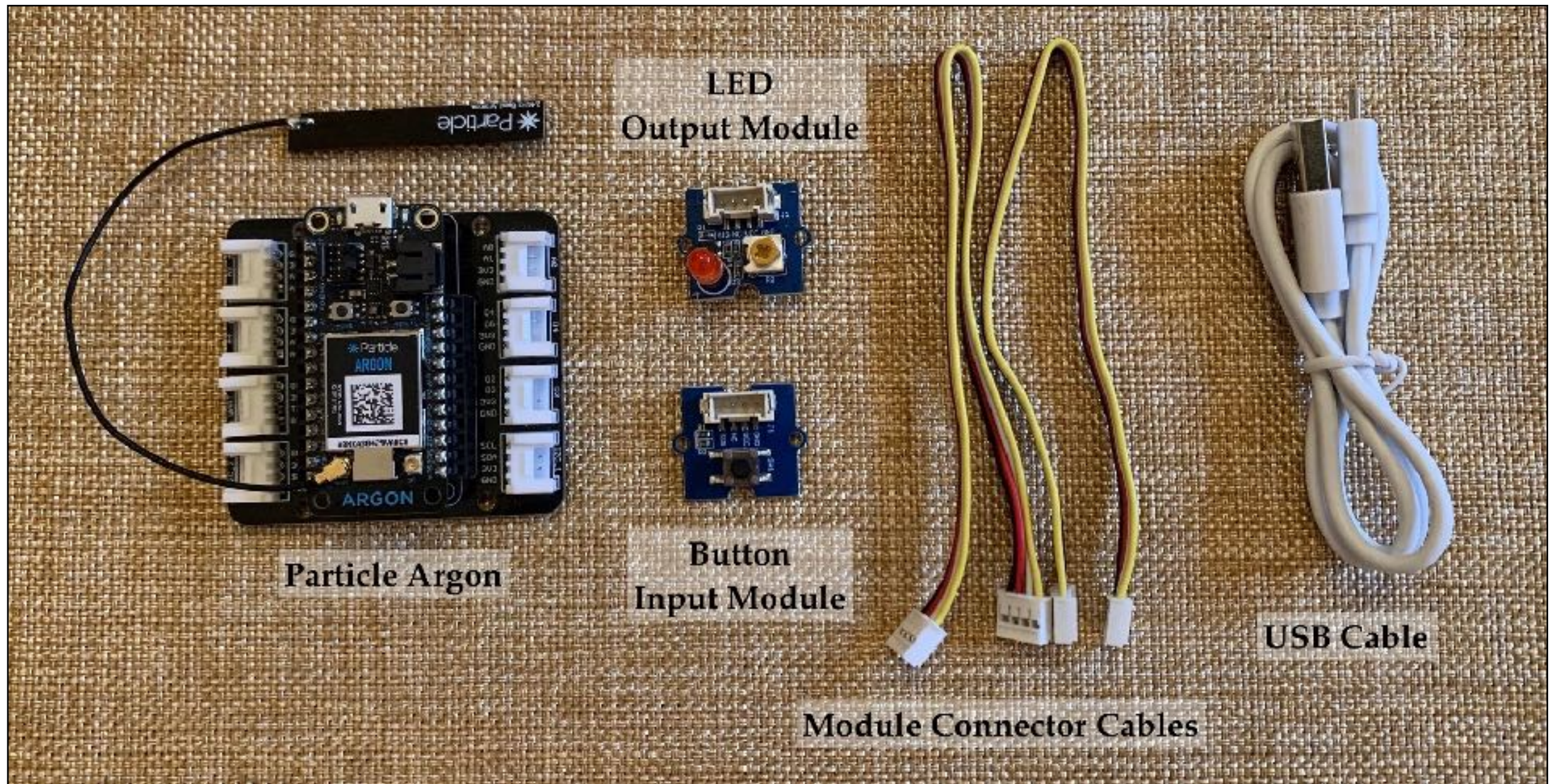


CURIE Academy, Summer 2021

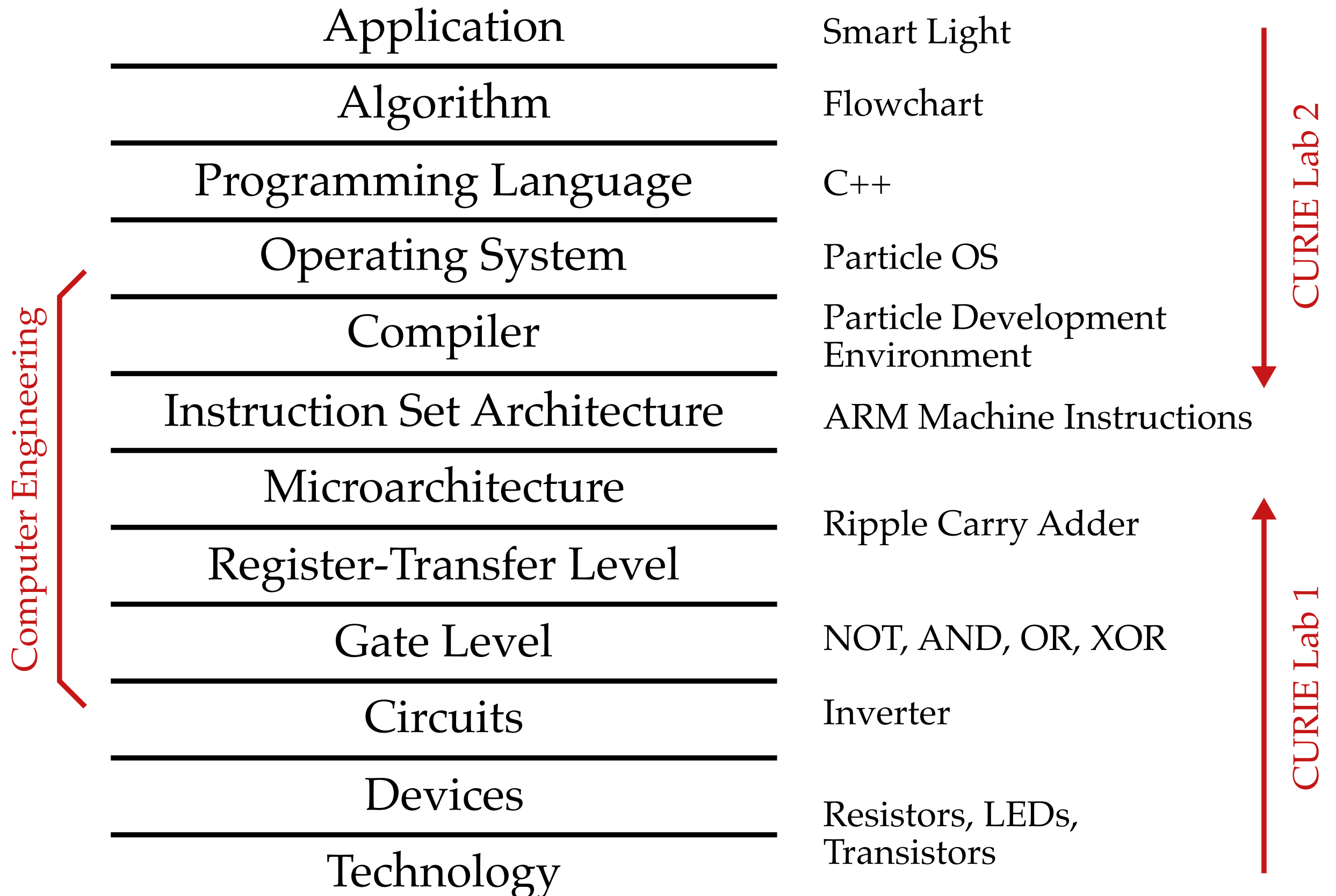
Lab 2: Computer Engineering Software Perspective

Prof. Christopher Batten
School of Electrical and Computer Engineering
Cornell University

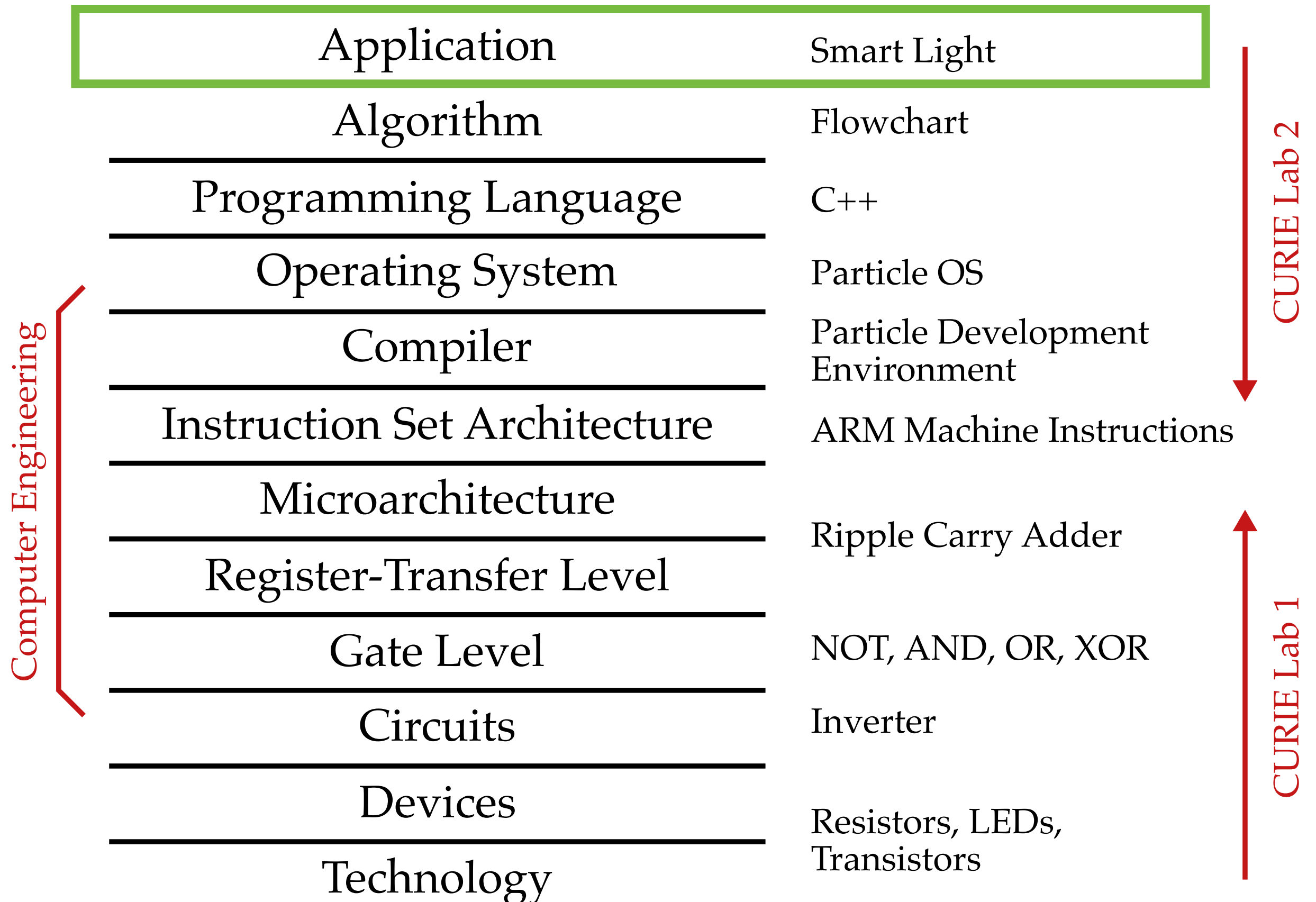
Materials Required for Lab 2



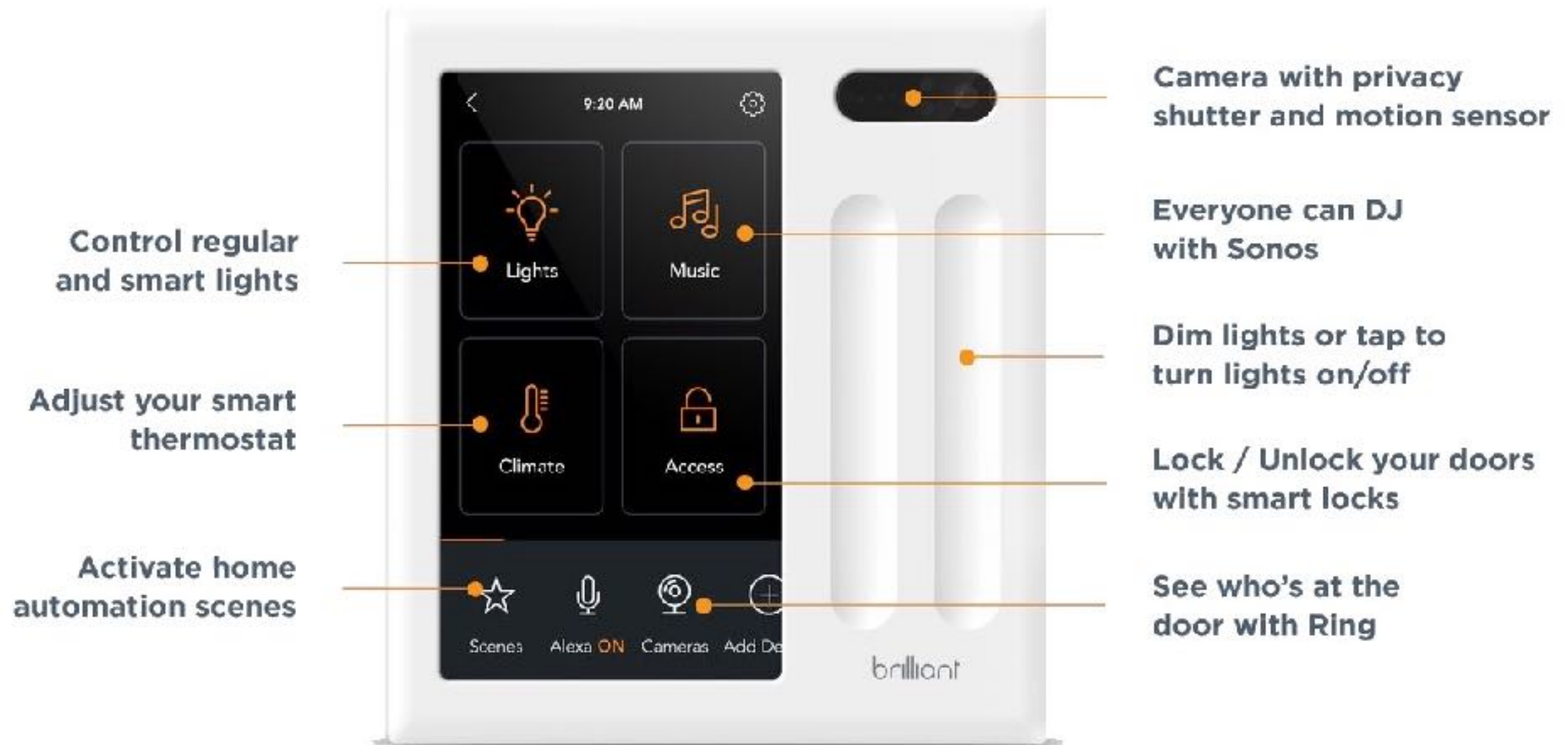
Computer Systems Stack



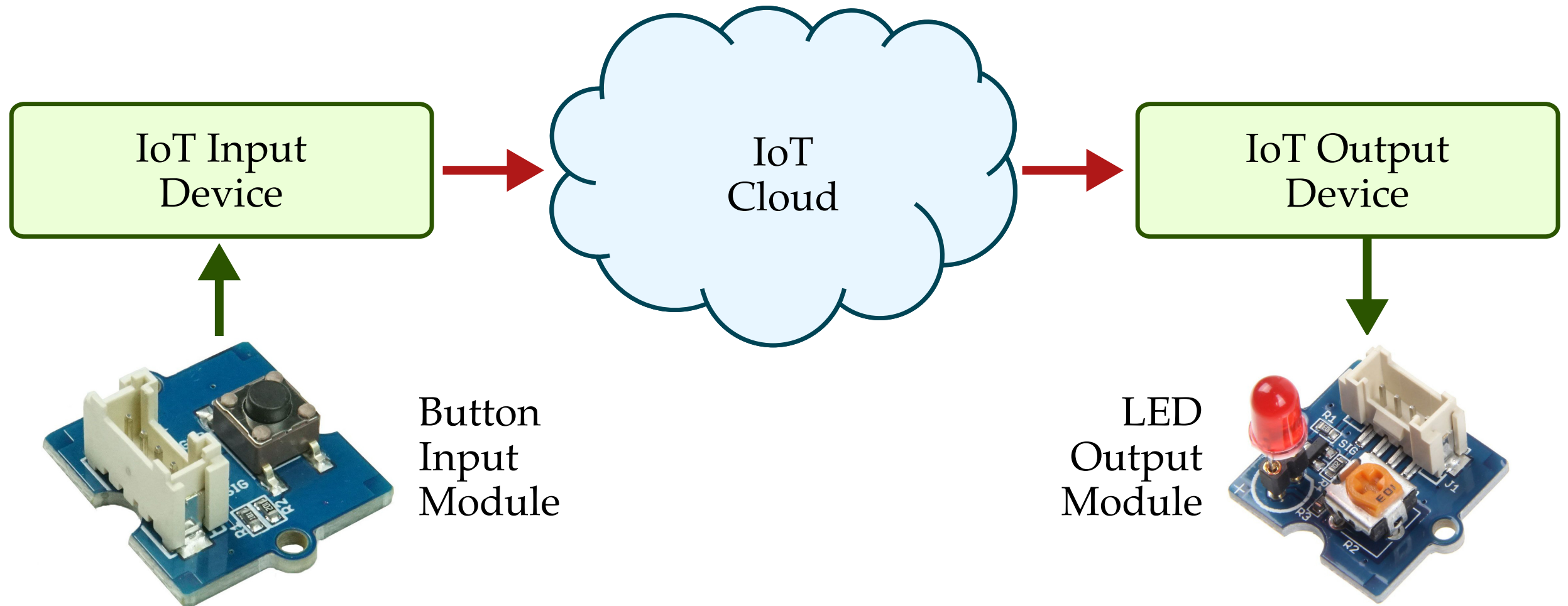
Computer Systems Stack



Application: “Smart Light” System

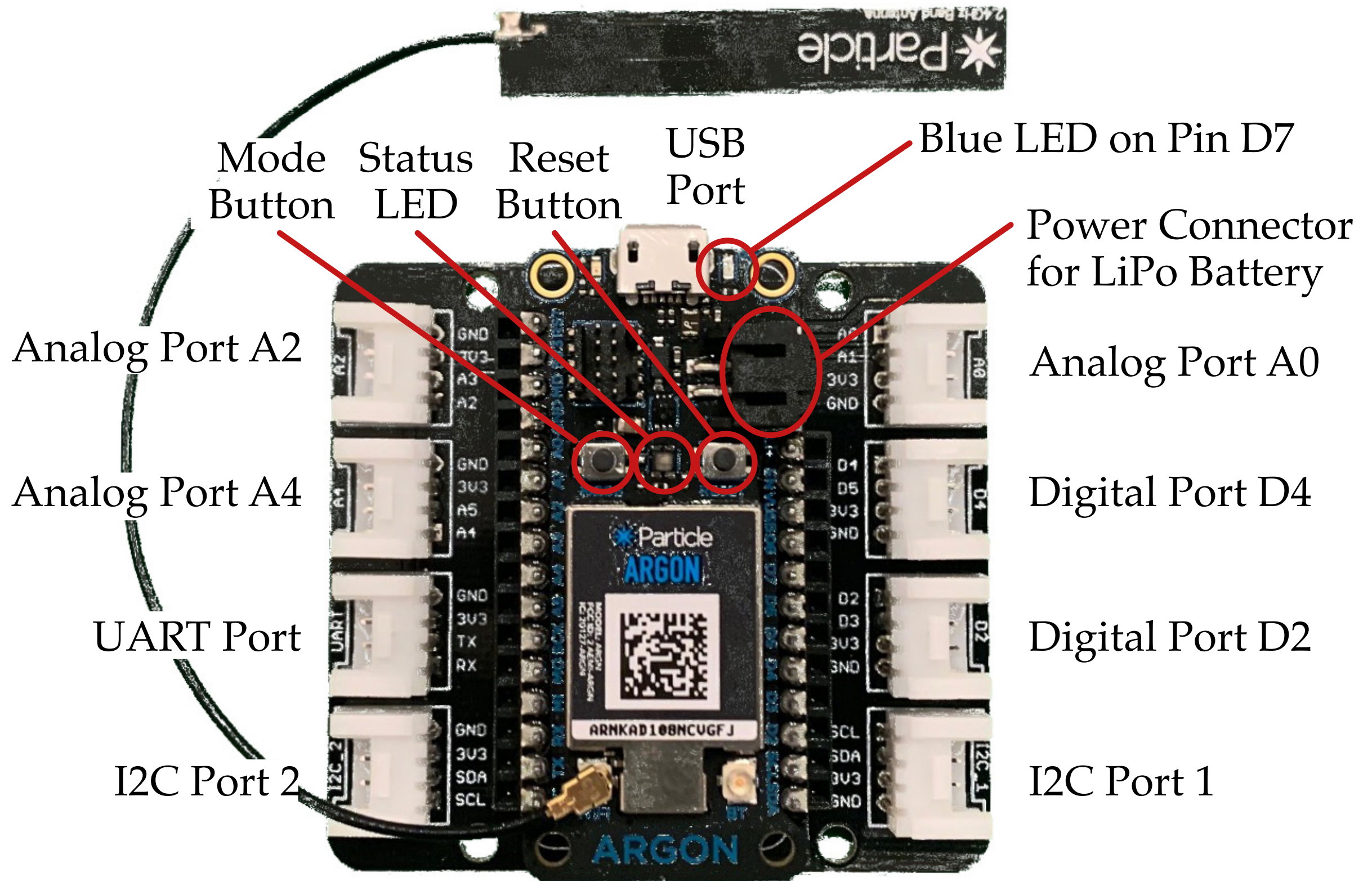


Application: “Smart Light” System

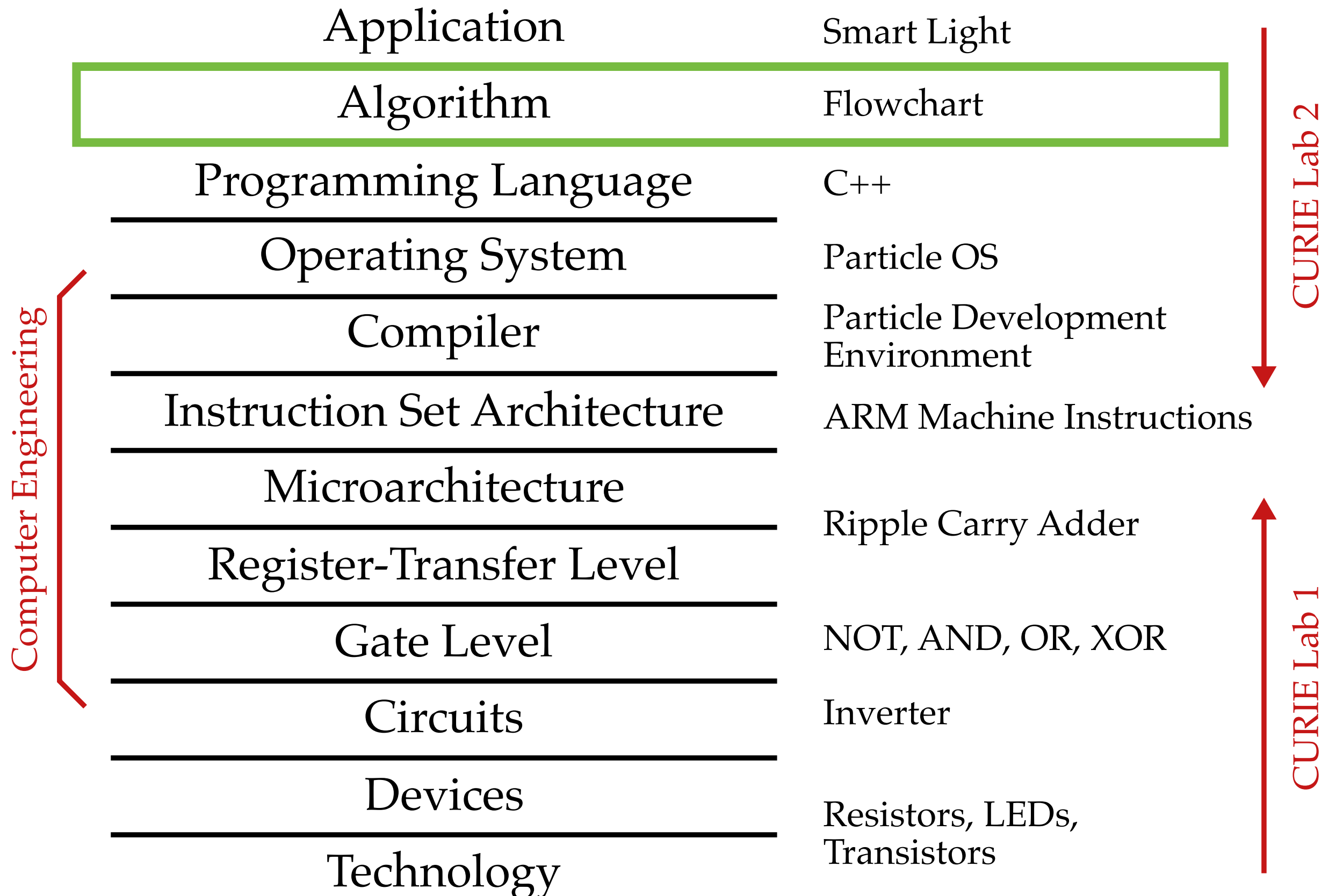


Particle Argon

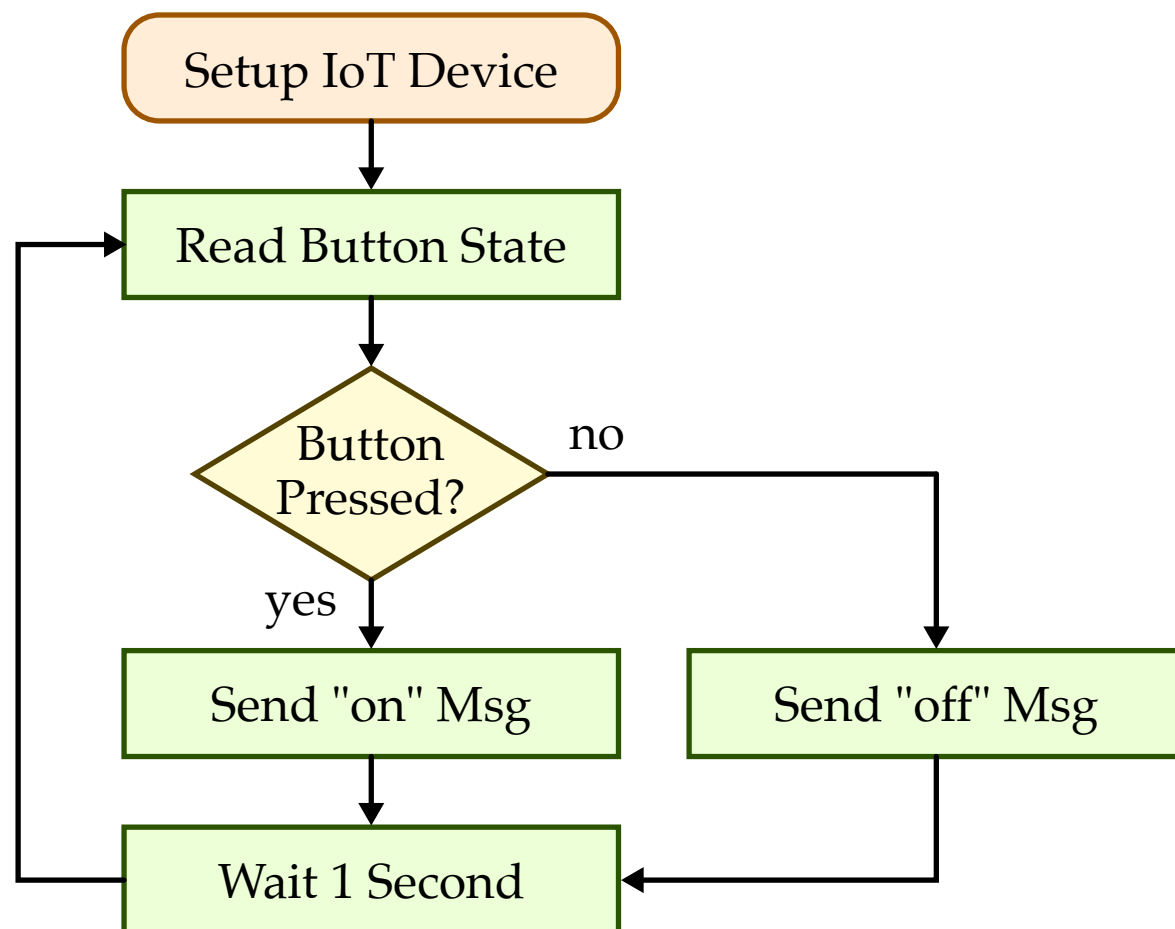
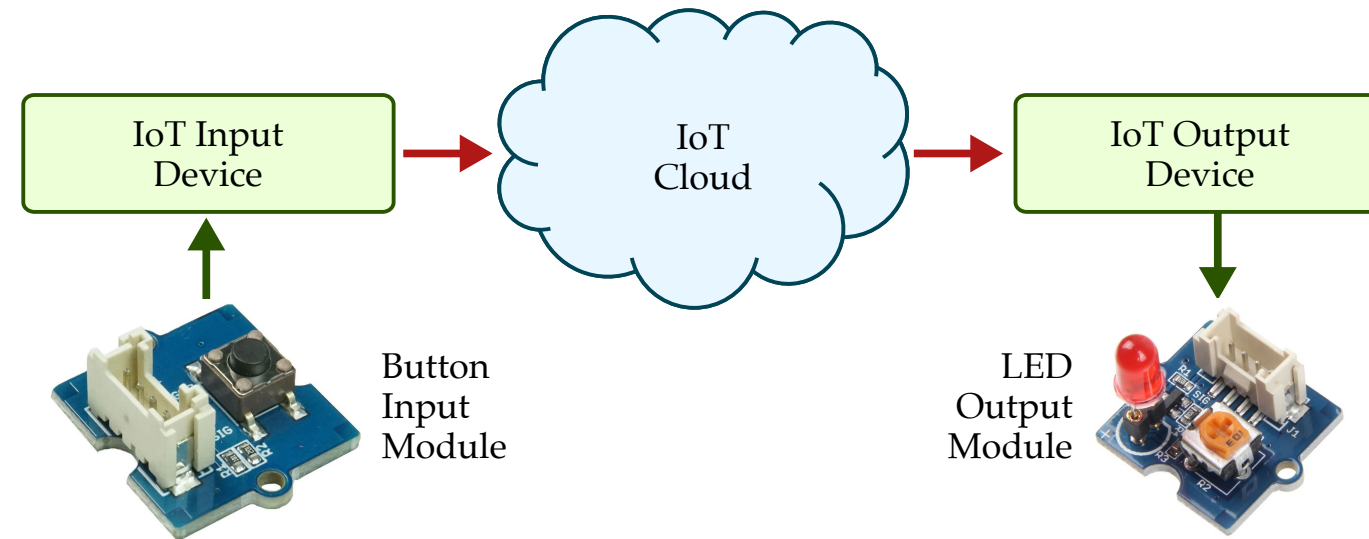
WiFi Antenna



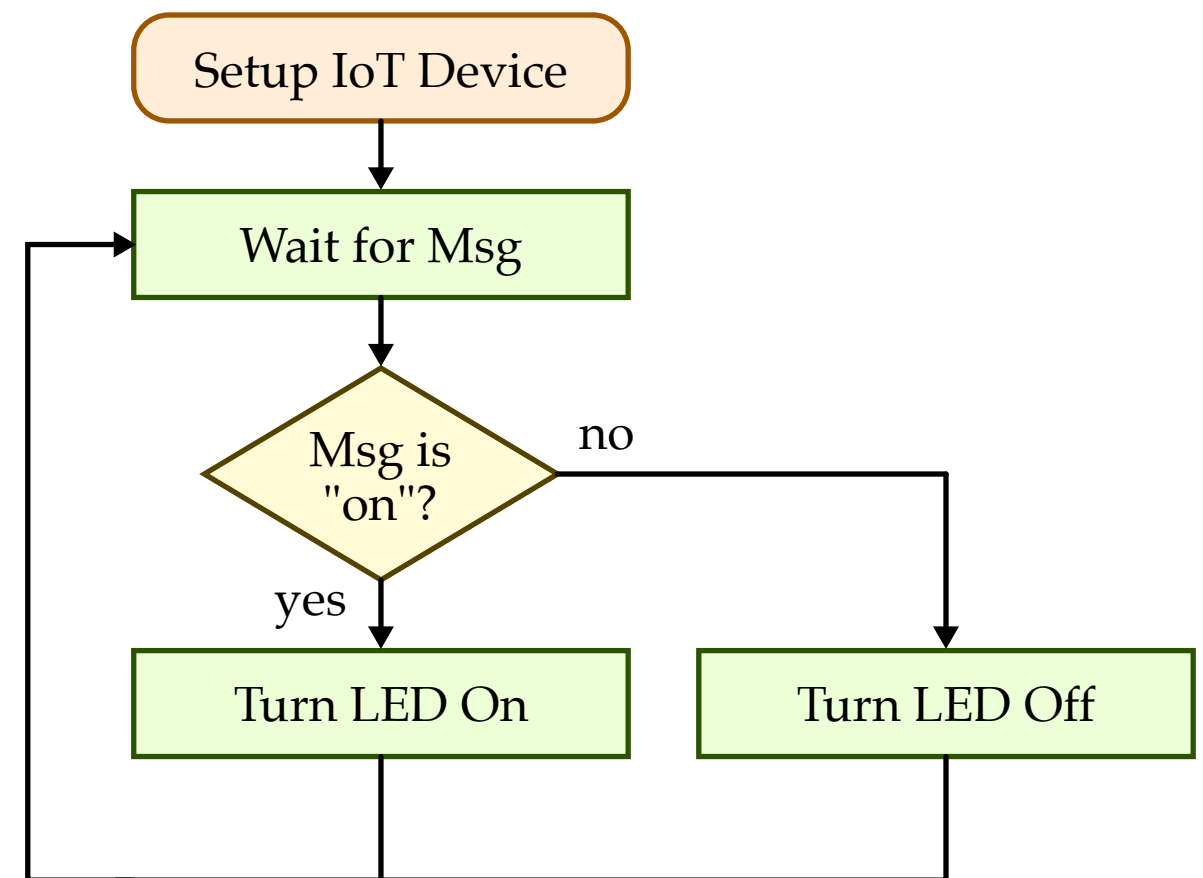
Computer Systems Stack



Algorithm: Flowcharts

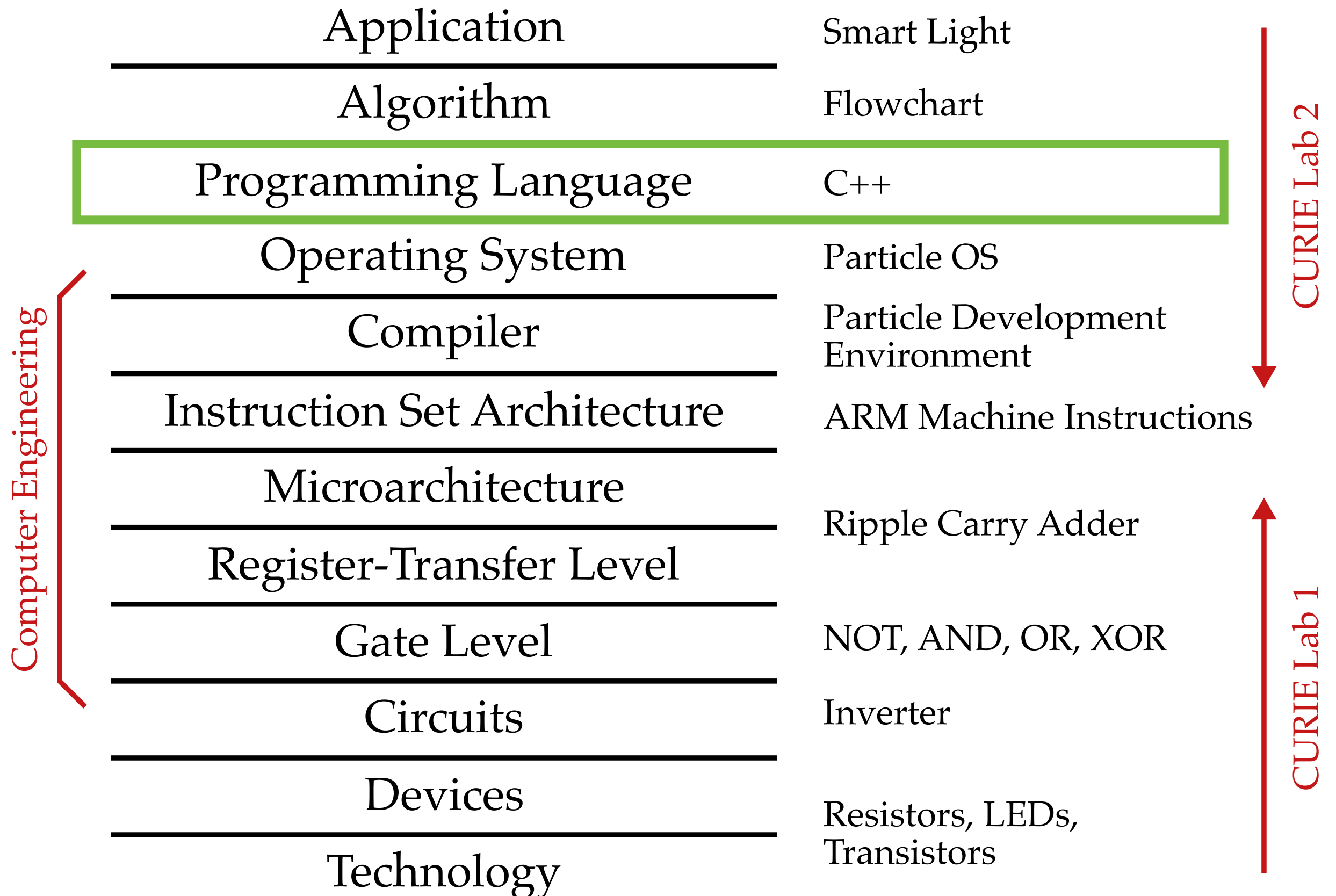


(a) Flowchart for IoT Input Device



(b) Flowchart for IoT Output Device

Computer Systems Stack



Programming Language: C++

1	<code>int a;</code>	<code>// declaration</code>	1	<code>// function to add two integers</code>		
2	<code>a = 2;</code>	<code>// assignment</code>	2	<code>int add(int a, int b)</code>		
3	<code>int b = 3;</code>	<code>// initialization</code>	3	<code>{</code>		
4			4	<code>int sum;</code>		
5	<code>int c;</code>		5	<code>sum = a + b;</code>		
6	<code>c = a + b;</code>		6	<code>return sum;</code>	1	<code>int c;</code>
			7	<code>}</code>	2	<code>c = add(2, 3);</code>
	(a)			(b)		(c)

Figure 6: Example C++ Code Snippets

Programming Language: C++

<pre>1 int a; // declaration 2 a = 2; // assignment 3 int b = 3; // initialization 4 5 int c; 6 c = a + b;</pre>	<pre>1 // function to add two integers 2 int add(int a, int b) 3 { 4 int sum; 5 sum = a + b; 6 return sum; 7 }</pre>	<pre>1 int c; 2 c = add(2, 3);</pre>
(a)	(b)	(c)

Figure 6: Example C++ Code Snippets

```
1  int button_state;
2  button_state = read_button_state( button_pin );
3
4  if ( button_state == 1 ) {
5      // send "on" msg
6  }
7  else {
8      // send "off" msg
9  }
10
11 // wait 1 second
```

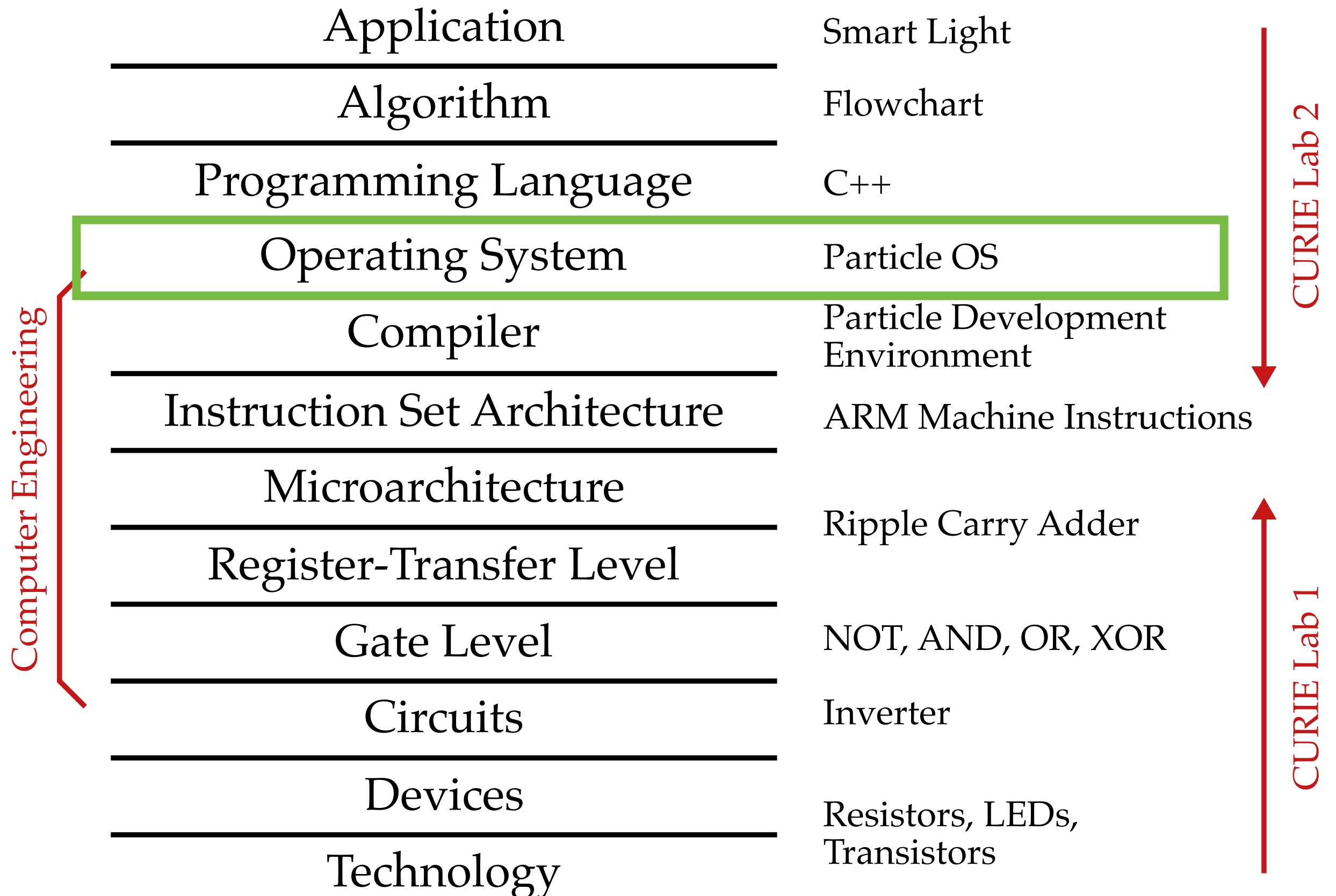
(a) Sketch of IoT Input Device Program

```
1  void receive_msg( msg )
2  {
3      if ( msg == "on" ) {
4          // turn light on
5      }
6      else {
7          // turn light off
8      }
9  }
```

(b) Sketch of IoT Output Device Program

Figure 7: Sketch of C++ Programs for Smart Light

Computer Systems Stack



Compiler: Particle OS

```
1 // Global constants for pin assignments and global variables
2
3 int led_pin = D7;
4
5 int x = 2;
6 int y = 3;
7 int z = 0;
8
9 // Helper functions
10
11 int add( int a, int b )
12 {
13     int sum;
14     sum = a + b;
15     return sum;
16 }
17
18 // The setup routine runs once when you press reset
19
20 void setup()
21 {
22     // Configure led_pin as digital output
23     pinMode( led_pin, OUTPUT );
24 }
```

1

2

3

```
26 // The loop routine runs over and over again
```

```
27
```

```
28 void loop()
```

```
29 {
```

```
30     // Do the addition
```

```
31     z = add( x, y );
```

```
32
```

```
33     // Blink LED z times
```

```
34     for ( int i = 0; i < z; i++ ) {
```

```
35         digitalWrite( led_pin, HIGH ); // Turn on the LED
```

```
36         delay(500); // Wait 0.5 seconds
```

```
37         digitalWrite( led_pin, LOW ); // Turn off the LED
```

```
38         delay(500); // Wait 0.5 seconds
```

```
39     }
```

```
40
```

```
41     // Wait four seconds
```

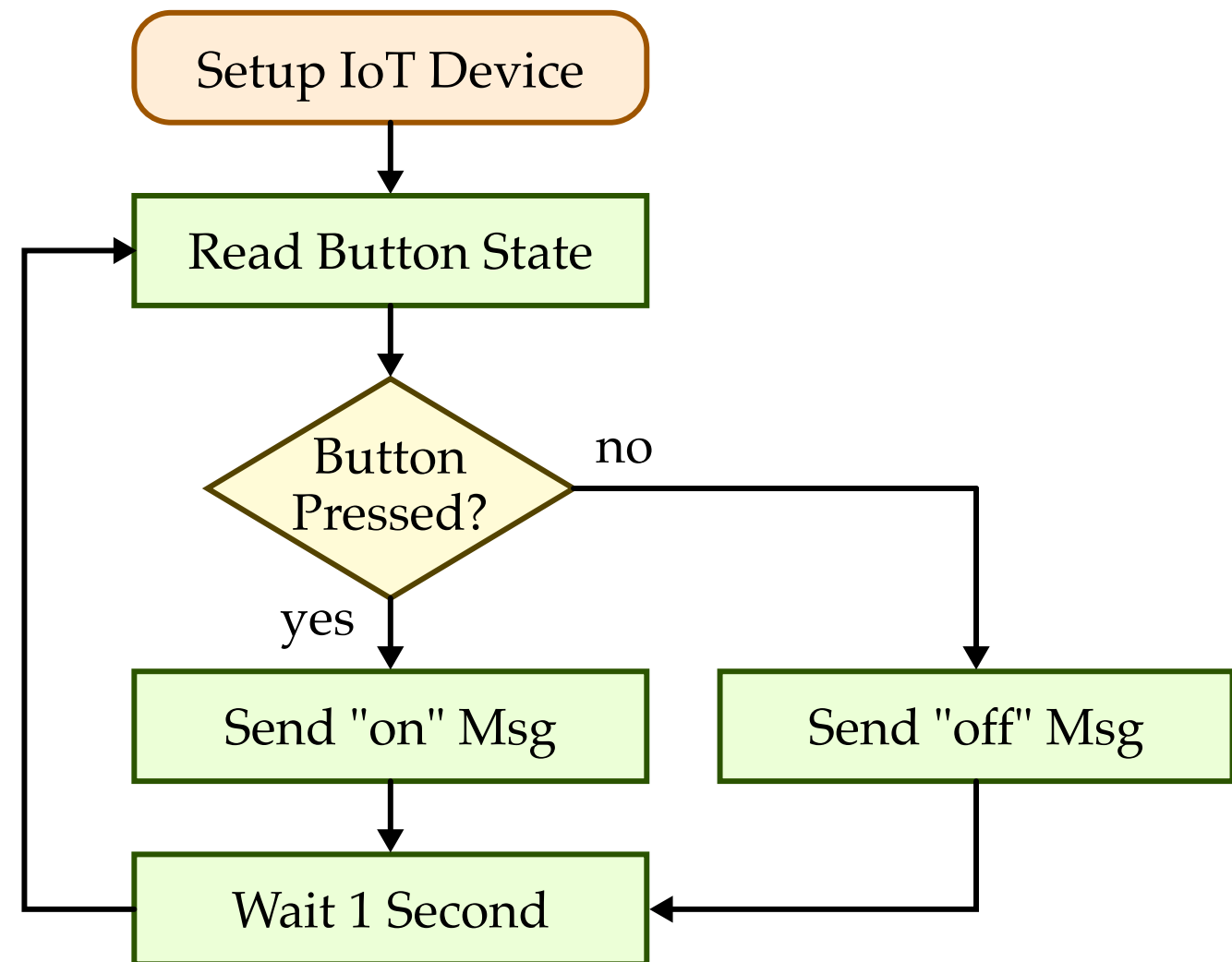
```
42     delay(4000);
```

```
43 }
```

4

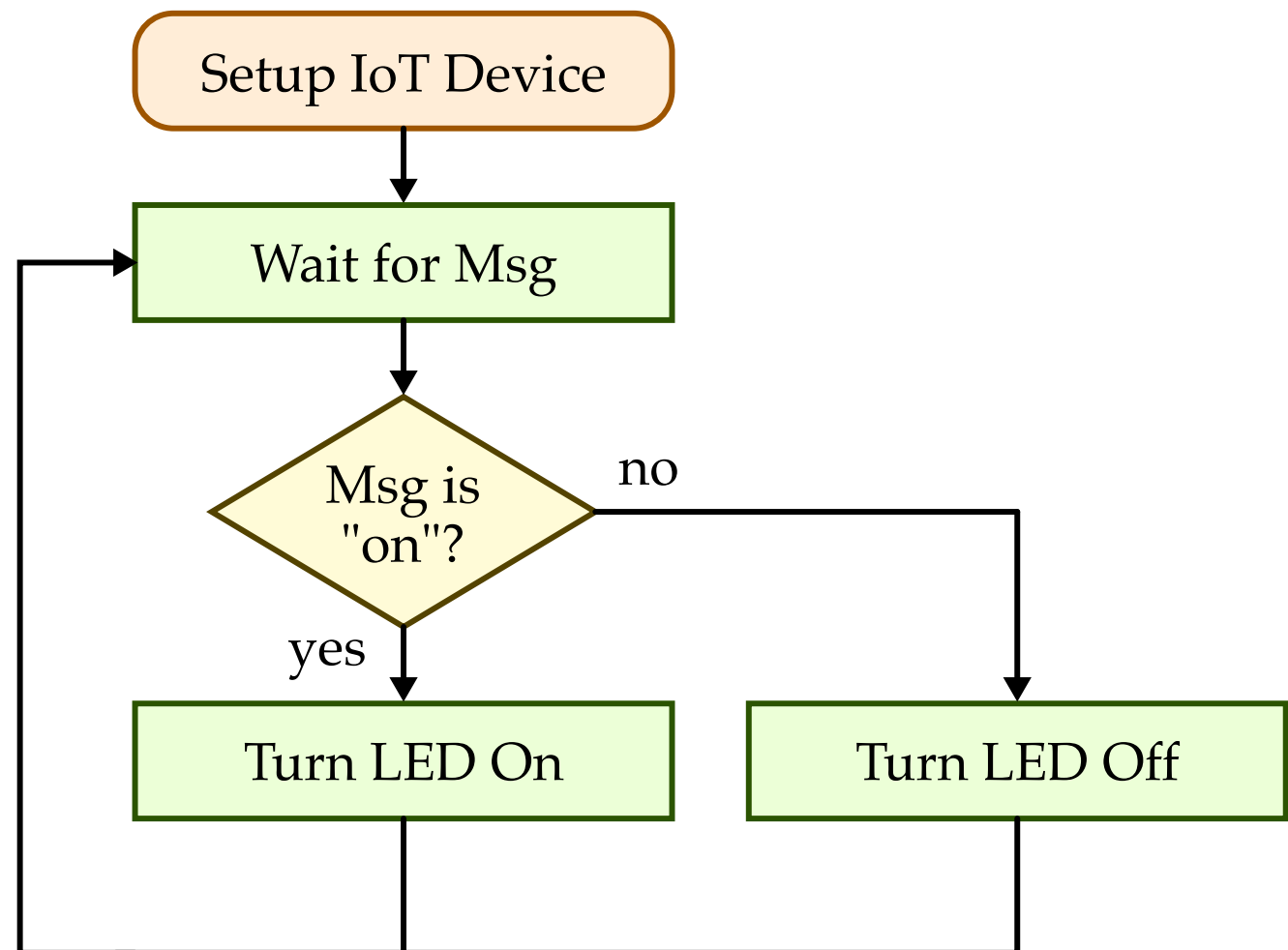
Compiler: Particle OS

```
1 int button_pin    = D4;
2 int button_state = -1;
3
4 void setup()
5 {
6   pinMode( button_pin, INPUT );
7 }
8
9 void loop()
10 {
11   button_state = digitalRead( button_pin );
12   if ( button_state == 1 ) {
13     Particle.publish( "button_state", "on" );
14   }
15   else {
16     Particle.publish( "button_state", "off" );
17   }
18   delay(1000);
19 }
```

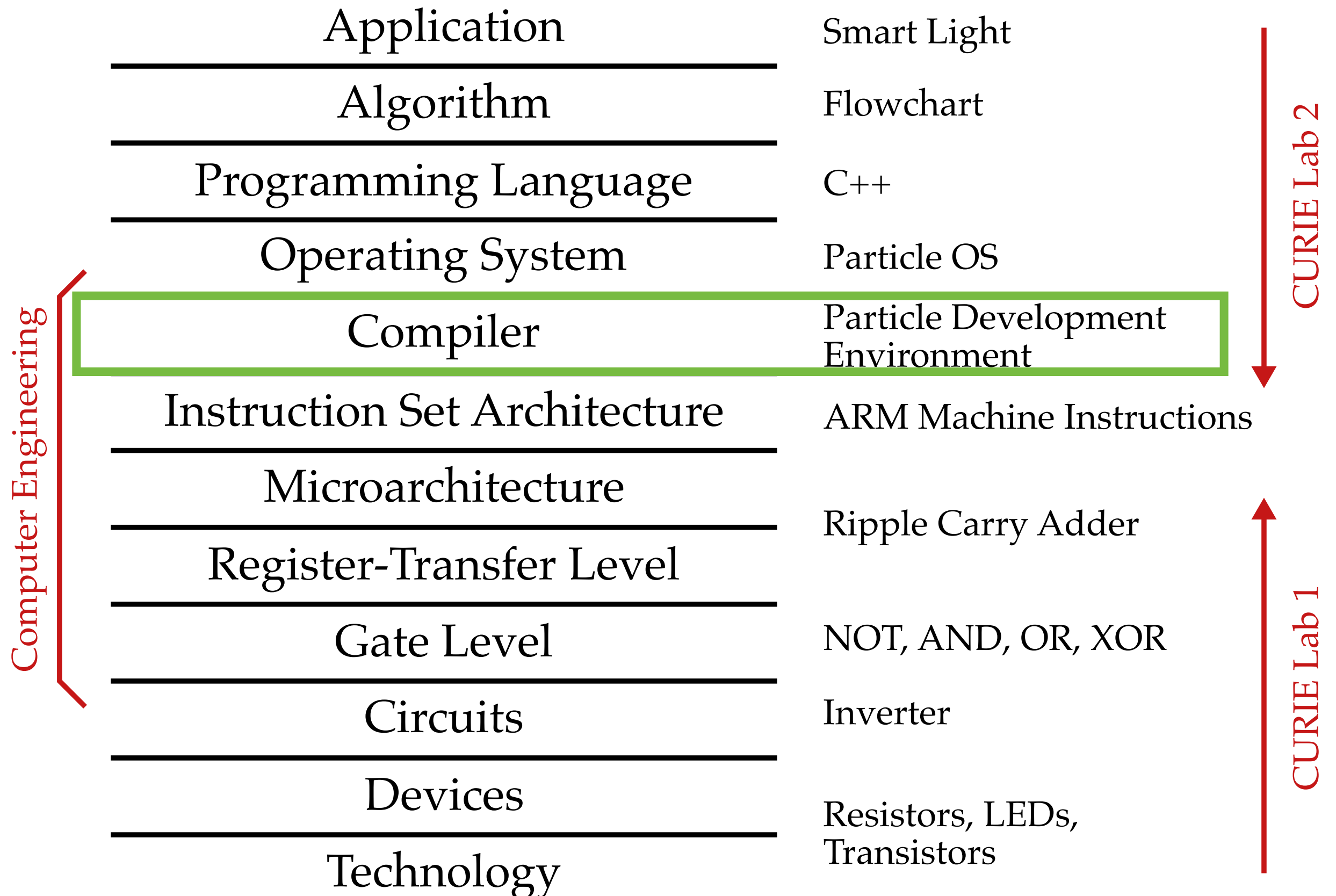


Compiler: Particle OS

```
1  int led_pin = D4;
2
3  void receive_msg( const char* event, const char* msg )
4  {
5      if ( strcmp( msg, "on" ) == 0 ) {
6          digitalWrite( led_pin, HIGH );
7      }
8      else {
9          digitalWrite( led_pin, LOW );
10     }
11 }
12
13 void setup()
14 {
15     pinMode( led_pin, OUTPUT );
16     Particle.subscribe( "button_state", receive_msg );
17 }
18
19 void loop()
20 {
21     // empty
22 }
```



Computer Systems Stack



Compiler: Particle Devel Environment

Flash

Verify

Save

Code

Library

Help

Docs

Devices

Console

Settings

★ brg-argon-139 >

★ brg-argon-140 >

★ brg-argon-141 >

★ brg-argon-142 >

★ brg-argon-143 >

★ brg-argon-144 >

★ brg-argon-145 >

★ brg-argon-148 >

★ brg-argon-153 >

★ brg-argon-154

Device ID:

Device OS target:
Default (2.1.0) v

On the device: 2.1.0

SIGNAL

★ brg-argon-155 >

brg-argon-154 v2.1.0

blink-an-led.ino

```
1 int button_pin = D4;
2 int button_state = -1;
3
4 void setup()
5 {
6   pinMode( button_pin, INPUT );
7 }
8
9 void loop()
10 {
11   button_state = digitalRead( button_pin );
12   if ( button_state == 1 ) {
13     Particle.publish( "button_state", "on" );
14   }
15   else {
16     Particle.publish( "button_state", "off" );
17   }
18   delay(1000);
19 }
```

Ready.

Compiler: Particle Devel Environment

- Always confirm your Particle Argon status LED is breathing cyan
- Always confirm that your Particle Argon is selected in the device list as indicated by the yellow star
- Always confirm that your Particle Argon is selected as indicated by the device name in the lower left-hand corner
- Always prefix the names your Particle Argon C++ programs with your first name (e.g., jane-blink-led)

Compiler: Particle Devel Environment

The screenshot displays the Particle Console interface for a specific device. The browser address bar shows the URL `console.particle.io/devices/e00fce684d7c11c254664147`. The interface includes a sidebar with navigation icons, a top navigation bar with links to Docs, Contact Sales, Support, and Notifications, and a user profile dropdown.

View Device

Device ID: `e00fce684d7c11c254664147` Name: `brg-argon-154`
Device OS: `2.1.0` Type: `Argon`
Serial Number: `ARNKAB8429VABCH` Last Handshake: `Jul 19th 2021, 10:35 am`
Last Heard: `Jul 19th 2021, 10:35 am`

Events

Search for events **ADVANCED**

NAME	DATA	DEVICE	PUBLISHED AT
Unpause to reveal 12 queued events.			
button_state	off	brg-argon-154	7/19/21 at 10:35:36 ...
button_state	off	brg-argon-154	7/19/21 at 10:35:35 ...
button_state	off	brg-argon-154	7/19/21 at 10:35:34...
button_state	off	brg-argon-154	7/19/21 at 10:35:33 ...
button_state	on	brg-argon-154	7/19/21 at 10:35:32 ...
button_state	on	brg-argon-154	7/19/21 at 10:35:32 ...
button_state	on	brg-argon-154	7/19/21 at 10:35:30...
spark/device/dia...	{ "device": {"networ...	brg-argon-154	7/19/21 at 10:35:30...
button_state	on	brg-argon-154	7/19/21 at 10:35:29 ...
button_state	off	brg-argon-154	7/19/21 at 10:35:28 ...
button_state	off	brg-argon-154	7/19/21 at 10:35:27 ...
particle/device/u...	false	brg-argon-154	7/19/21 at 10:35:27 ...
button_state	off	brg-argon-154	7/19/21 at 10:35:26 ...
particle/device/u...	false	brg-argon-154	7/19/21 at 10:35:26 ...
particle/device/u...	true	brg-argon-154	7/19/21 at 10:35:26 ...
spark/device/las...	power_down	brg-argon-154	7/19/21 at 10:35:26 ...
spark/status	online	brg-argon-154	7/19/21 at 10:35:25 ...
spark/status	offline	brg-argon-154	7/19/21 at 10:31:45 ...

button_state
Published by `e00fce684d7c11c254664147` on 7/19/21 at 10:35:32 am

LAST VITALS

Jul 19th, 2021, 10:27AM

- Strong Wi-Fi signal
- 2119ms round-trip time
- 50kB of 165kB RAM used
- 0 rate-limited publishes

FIRMWARE

OTA Updates: Enabled

Force Enable OTA
Force enable OTA updates to override device firmware setting

FUNCTIONS

No functions found on device. Read more about functions [here](#).

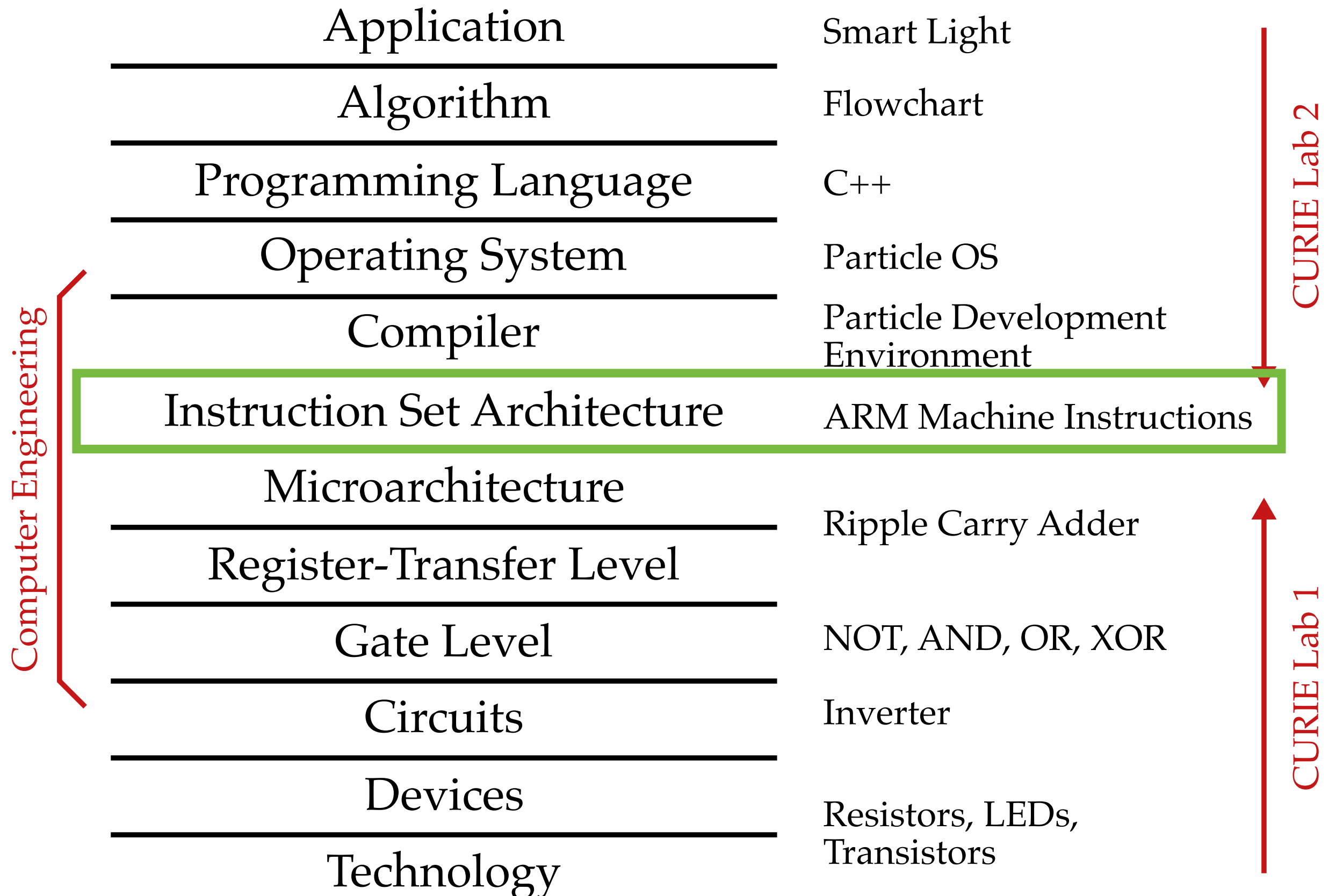
VARIABLES

No variables found on device. Read more about variables [here](#).

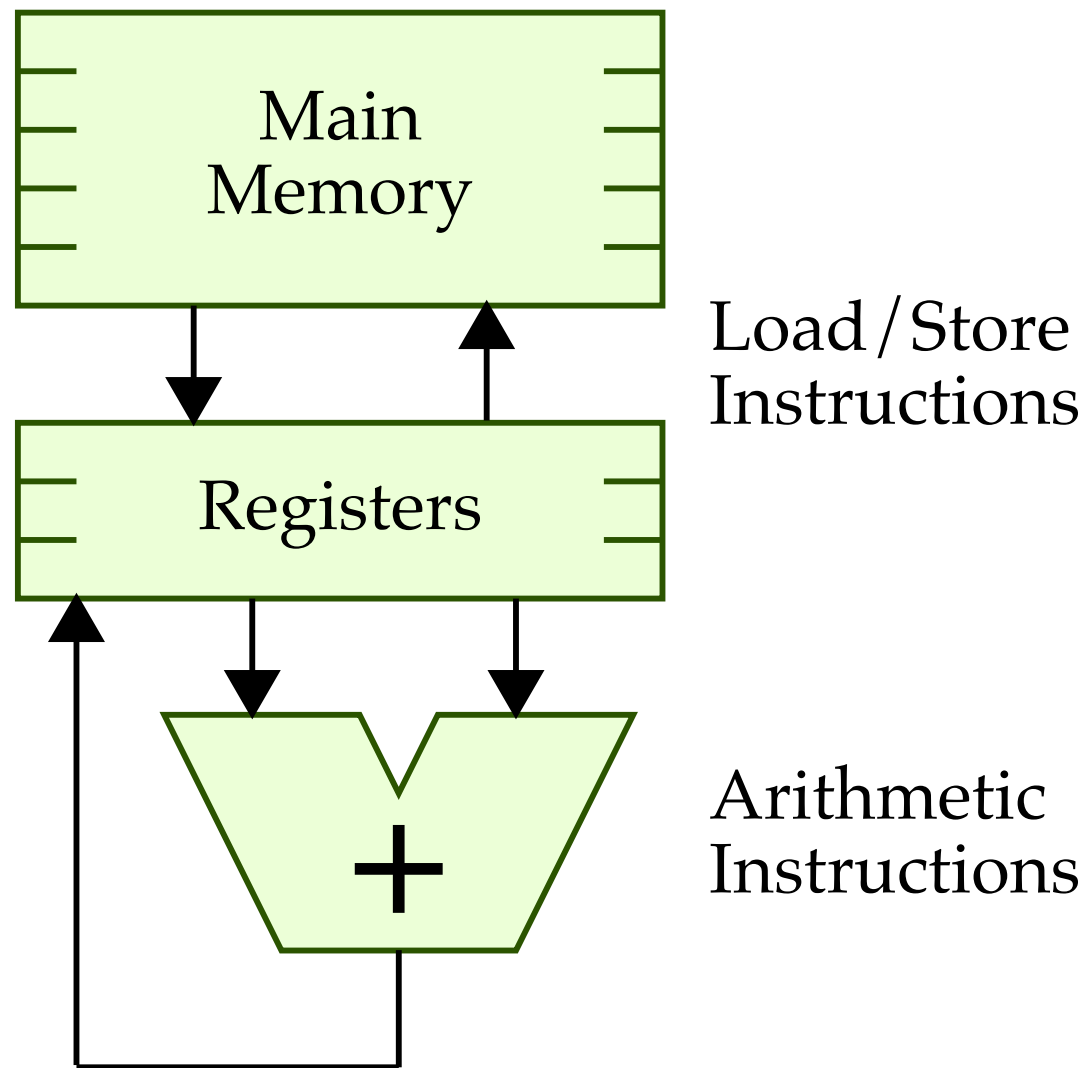
ACTIONS

UNCLAIM

Computer Systems Stack



ISA: ARM Machine Instructions



```
11  int add( int a, int b )
12  {
13      int sum;
14      sum = a + b;
15      return sum;
16  }
```



```
1  # load two values from main
2  # memory into two registers
3  ldr      r2, [r7, #4]
4  ldr      r3, [r7]
5
6  # do the actual addition
7  add      r3, r3, r2
8
9  # store the sum from a register
10 # back into main memory
11 str      r3, [r7, #12]
```

ISA: ARM Machine Language

The screenshot shows the Godbolt Compiler Explorer interface. The browser address bar shows `godbolt.org`. The Compiler Explorer logo is in the top left. The main interface is divided into two panes. The left pane, titled "C++ source #1", contains the following C++ code:

```
1 // Type your code here, or load an
2 int square(int num) {
3     return num * num;
4 }
```

The right pane, titled "x86-64 gcc 11.1 (Editor #1, Compiler #1) C++", shows the generated assembly code:

```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul    eax, eax
7     pop     rbp
8     ret
```

Annotations on the image include:

- A red arrow pointing to the compiler selection dropdown menu, which currently shows "x86-64 gcc 11.1". A red text label "Select ARM gcc trunk (linux) here" is placed above the arrow.
- A red oval around the "Compiler options..." button. A red arrow points to this button with the text "Try entering -O3 here".
- Red text "Write your C++ code here ..." is placed below the C++ source code pane.
- Red text "Compiler will generate the corresponding machine instructions here ..." is placed below the assembly code pane.

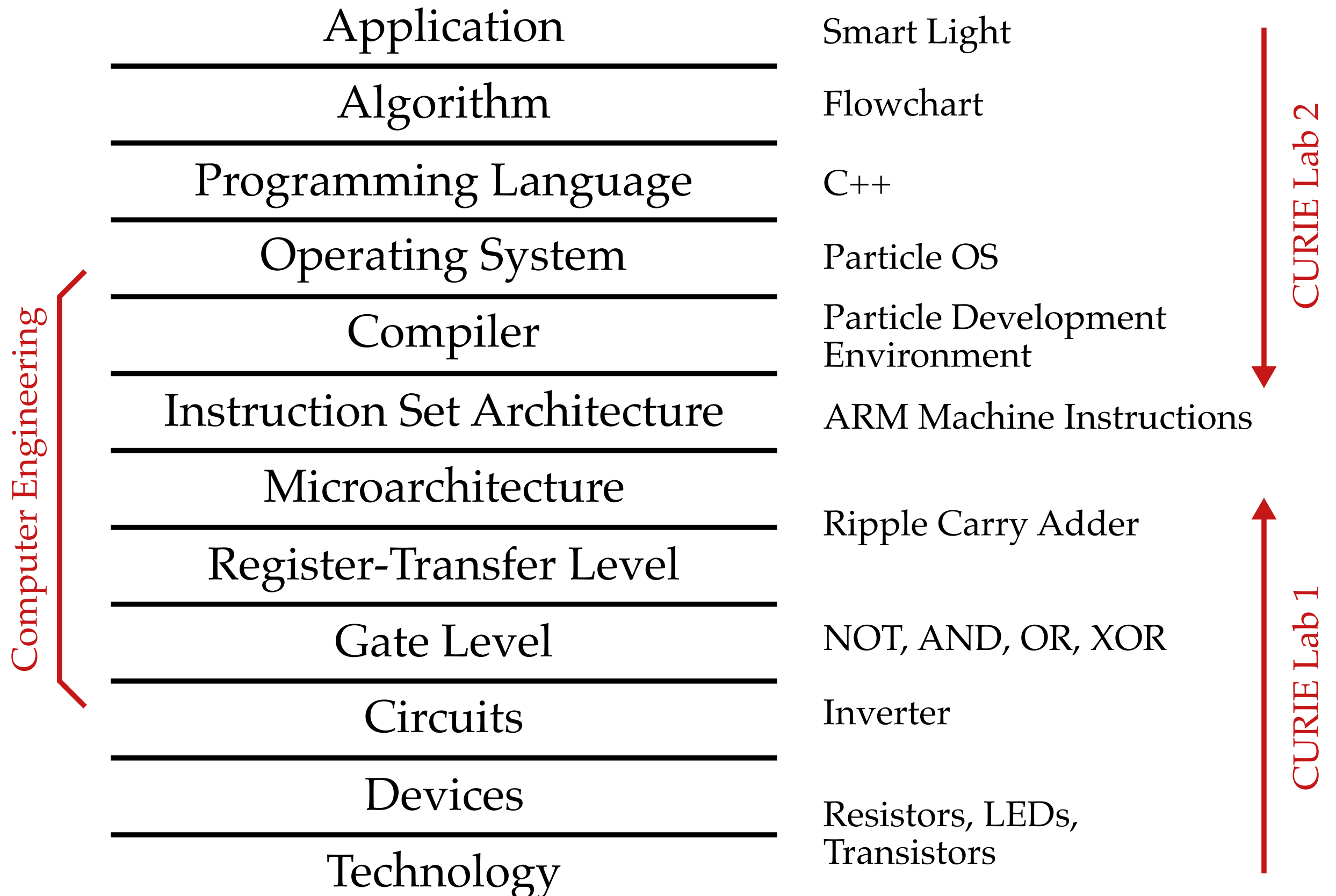
The bottom status bar shows "Output (0/0) x86-64 gcc 11.1 - 77ms (2792B) ~170 lines filtered".

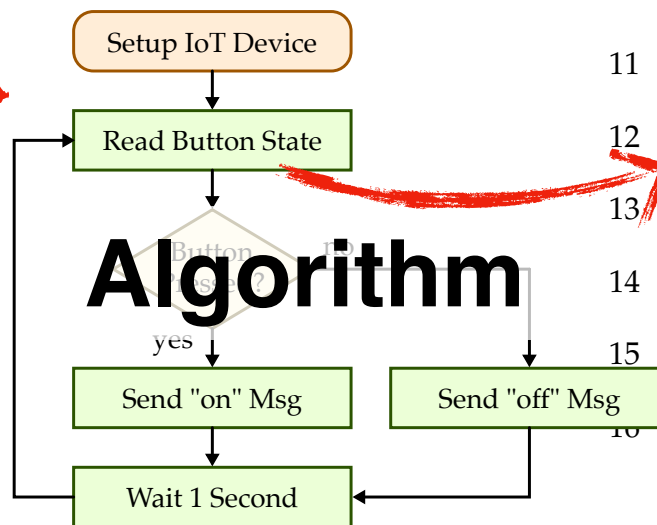
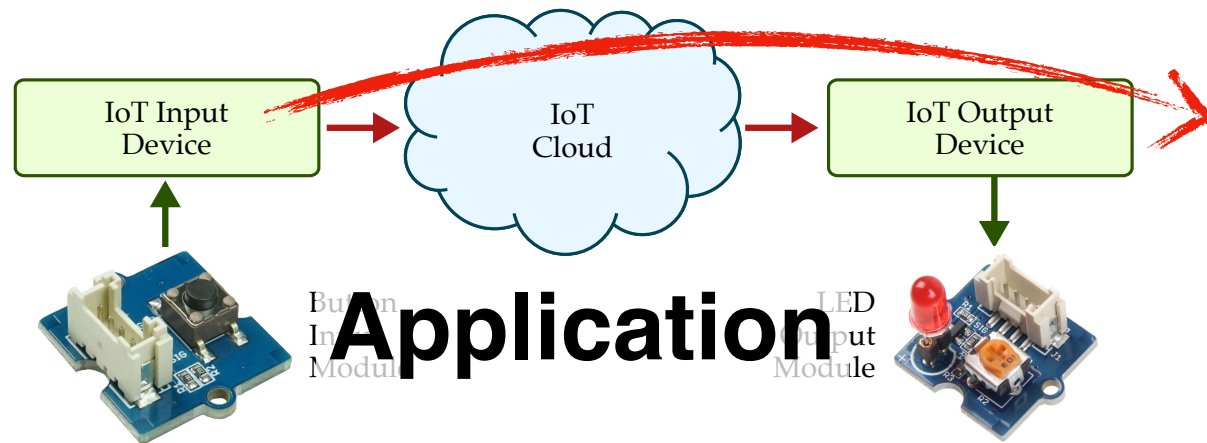
Lab 2 Overview

- Part 1.A Test Simple Addition Program
- Part 1.B Examine Machine Instructions
- Part 2.A Experiment with LED Output
- Part 2.B Experiment with Button Input
- Part 3.A Experiment with Particle Variables
- Part 3.B Experiment with Sending Particle Events
- Part 3.C Experiment with Receiving Particle Events
- Part 4.A Develop a “Smart Light” System
- Part 4.B Share Photo or Video of IoT System
- Experiment with IoT Geolocation System

Let's write a simple blinking LED program

Computer Systems Stack





```
11 int add( int a, int b )
12 {
13     int sum;
14     sum = a + b;
15     return sum;
16 }
```

Prog Lang

This block shows a C program for adding two integers. It is labeled as the 'Prog Lang' layer.

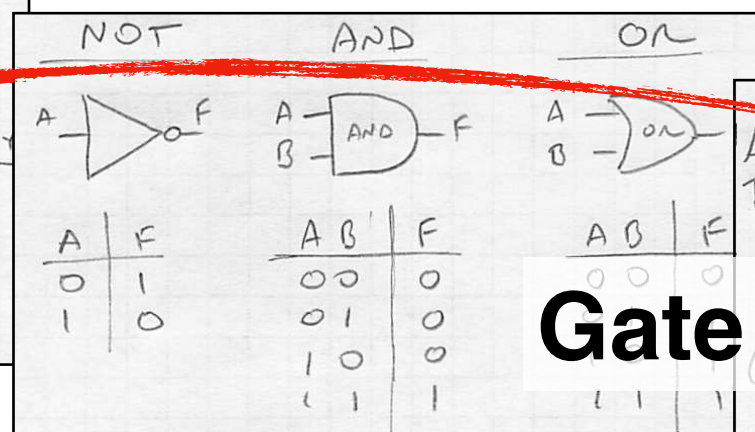
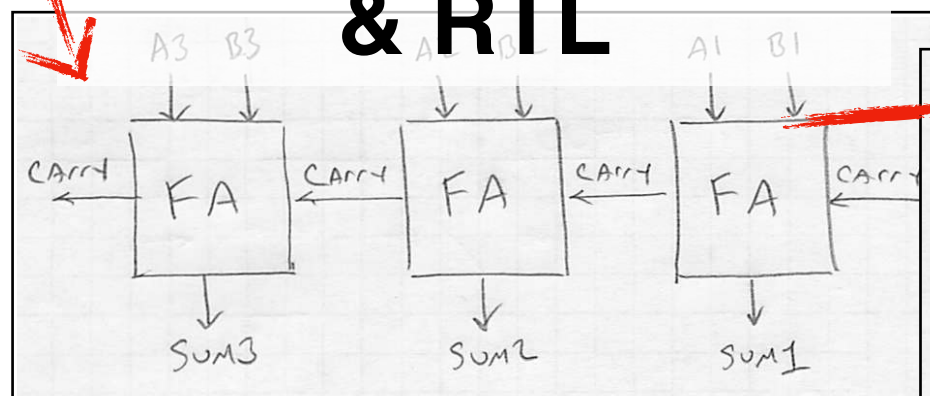
Instruction Set Architecture

```
1 # load two values from main
2 # memory into two registers
3 ldr    r2, [r7, #4]
4 ldr    r3, [r7]
5
6 # do the actual addition
7 add    r3, r3, r2
8
9 # store the sum from a register
10 # back into main memory
11 str    r3, [r7, #12]
```

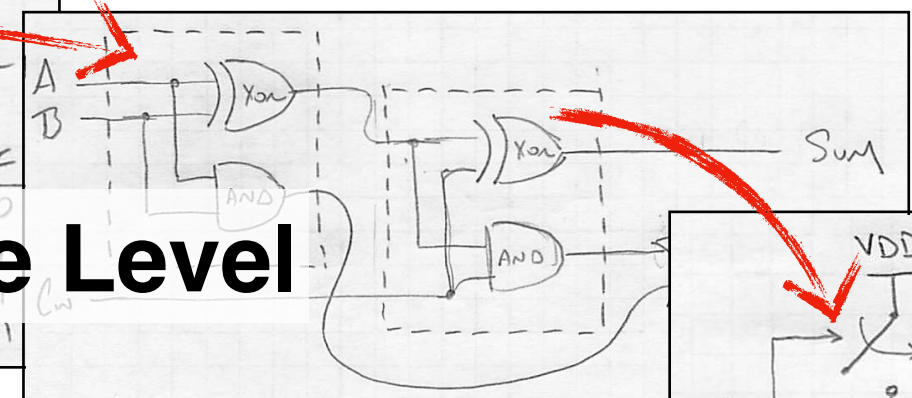
Arithmetic Instructions

This block shows assembly code for loading values from memory, performing an addition, and storing the result back to memory. It is labeled as the 'Instruction Set Architecture' layer.

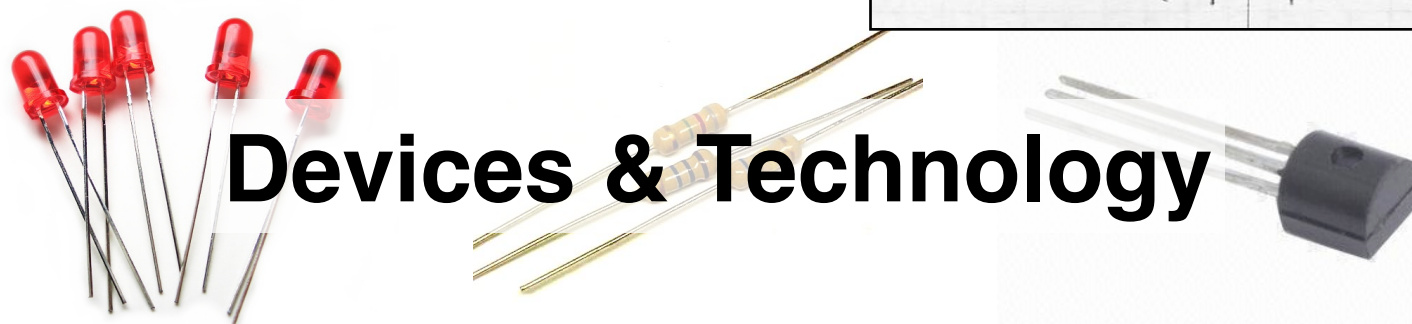
Microarchitecture & RTL



Gate Level



Devices & Technology



Circuits

