Session #1: Fault Injection primer

Albert Spruyt

albert.spruyt@gmail.com

Fault Injection Think Tank (FITT)

- Alyssa Milburn (@noopwafel)
- Cristofaro Mune (@pulsoid)
- Niek Timmers (@tieknimmers)
- Albert Spruyt
- …

ASK QUESTIONS AT ANY TIME!

Today's agenda

Time	Topic	Presenter
9:30 – 10:45	Fault Injection Primer DEMO Fault Injection Attacks	Albert Niek
11:45 – 12:30	Modeling FI	Cristofaro
14:00 – 15:30	Lab: Attacking and hardening Secure Boot	

This presentation

This presentation

- What is glitching?

This presentation

- What is glitching?
- Why glitch?

This presentation

- What is glitching?
- Why glitch?
- How do we glitch?

This presentation

- What is glitching?
- Why glitch?
- How do we glitch?
- What does the process look like?

Fault Injection: a definition

“Introducing faults in a target to alter its intended behavior.”

```
...
if( key_is_correct ) <-- Glitch here!
{
  open_door();
}
else
{
  keep_door_closed();
}
...
```

Fault Injection: a definition

“Introducing faults in a target to alter its intended behavior.”

```
...
if( key_is_correct ) <-- Glitch here!
{
  open_door();
}
else
{
  keep_door_closed();
}
...
```

*How can we introduce these **faults**?*

Overview of injection techniques

Clock

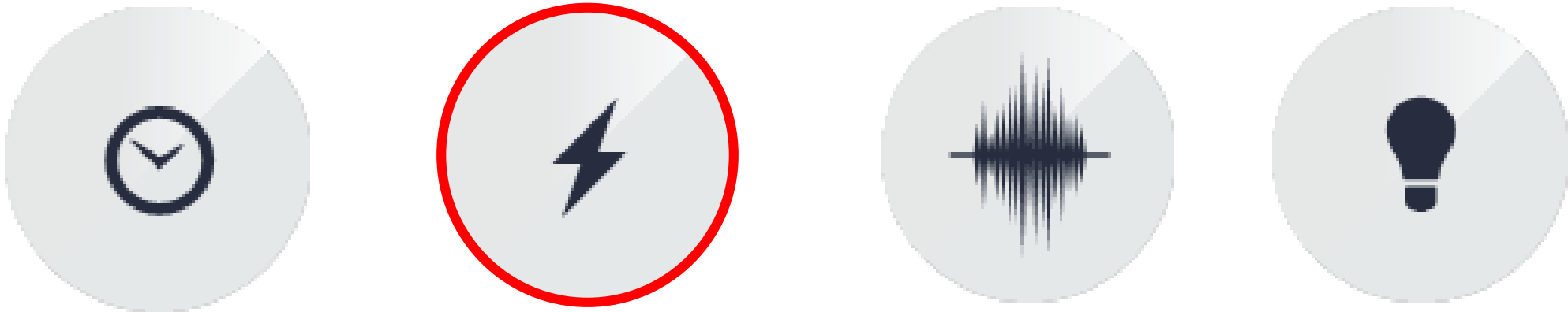
Voltage

EM

Laser

- A controlled environmental change (***glitch***) leads to target misbehavior (***fault***)
- Used for leveraging an existing ***vulnerability*** in hardware

Overview of injection techniques

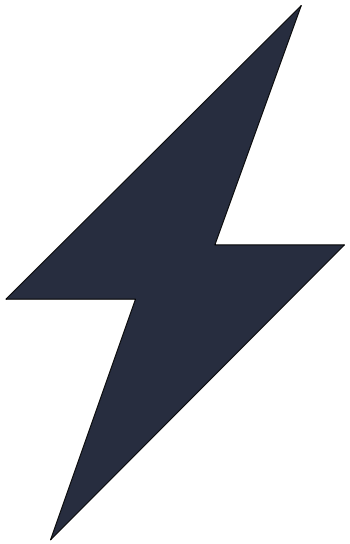


Clock *Voltage* *EM* *Laser*

- A controlled environmental change (***glitch***) leads to target misbehavior (***fault***)
- Used for leveraging an existing ***vulnerability*** in hardware

Glitch

“A controlled environmental change.”



*These **glitches** can result in **fault injection vulnerabilities**!*

Vulnerability

*"Susceptibility of a given **hardware subsystem** to a specific **fault injection technique,** which has an impact on security."*

- Located in hardware
- Cannot be identified by code review only
- Can only be identified by performing a successful attack
- Can only be entirely addressed in hardware

*These **vulnerabilities** lead to **faults**!*

Fault

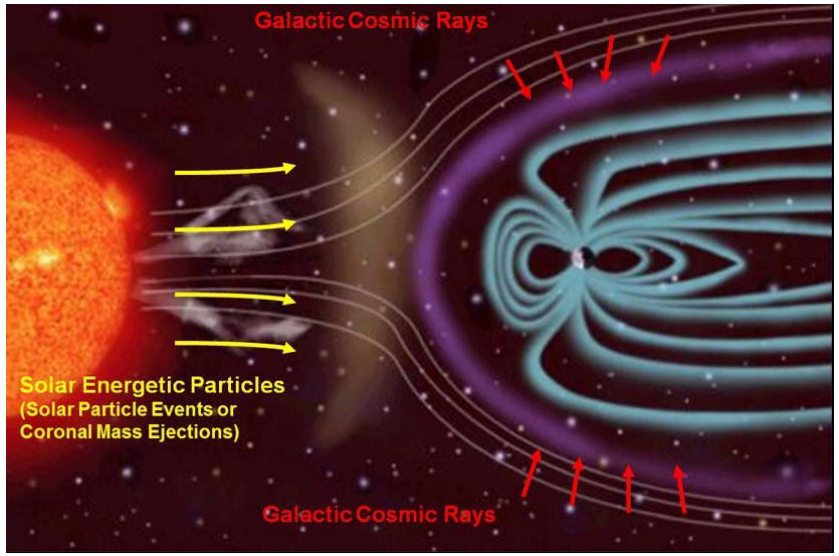
*"An unintended alteration as a consequence of a **vulnerability**."*

- Happens at a specific moment in time
- May be (semi-)persistent
- May be mitigated in software

*These **faults** potentially lead to **compromised systems**!*

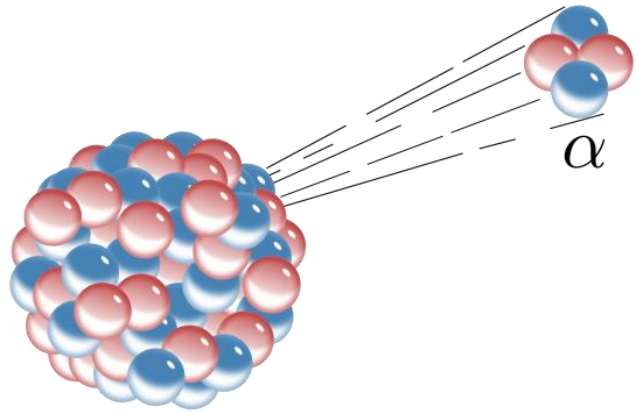
How are faults injected?

Natural phenomena



Cosmic rays

** Ziegler, Lanford – "Effects of cosmic rays on computer memories" (1979)*



Alpha decay

** May, Woods – "Alpha-particle-induced soft errors in dynamic memories" (1979)*

Cost: ???

High-end Tooling

- Great for security labs
- Different techniques:
  - VCC, Clock, EM, Laser,...
- Flexibility, speed, precision
- ***High control → Repeatability***

Cost ($): > $10,000

Other options...

Chipwhisperer Lite

~$250

Microcontroller

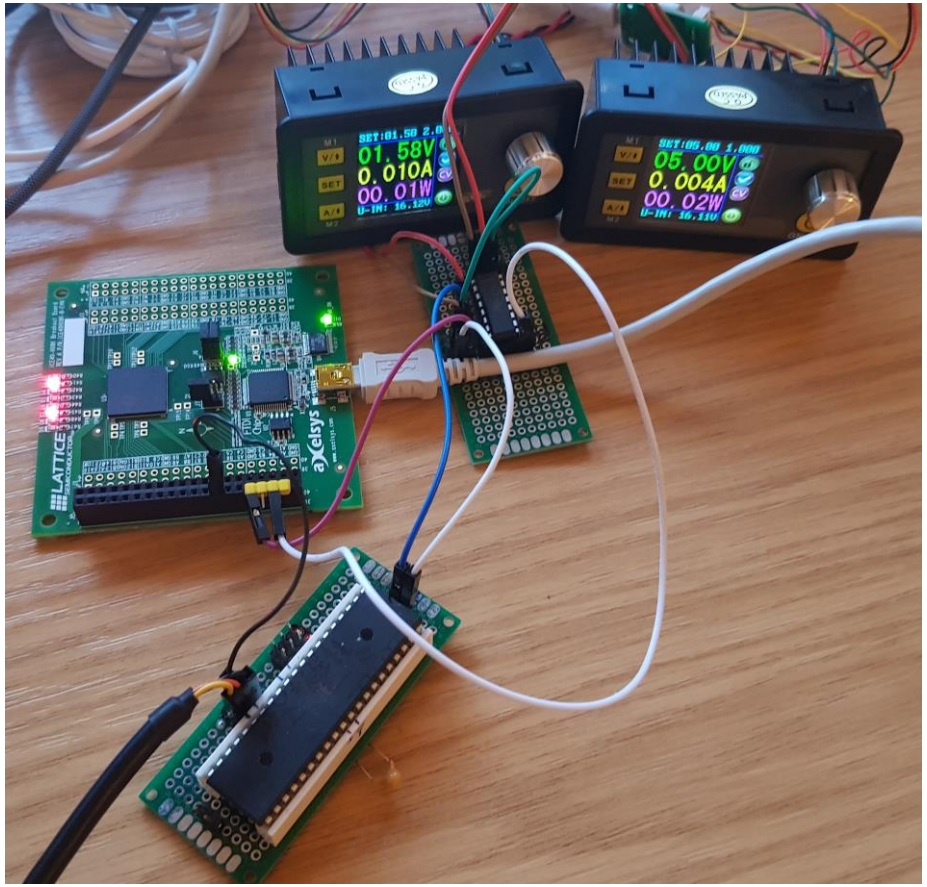
< $30

FPGA

~$99

Cost ($): < $300

Cheap hardware



Cost ($): < $100

Do we always need specialized tooling?

Do we always need specialized tooling?

Not always...

Software activated fault injection

- Possible when software can activate hardware vulnerabilities
- The vulnerabilities and faults are still in hardware!

Some recent examples...

- **Rowhammer** (Kim et al., 2014; many more afterwards)
  - Constantly reading a specific DDR address lead to bit flips
- **CLKSCREW (**Tang et al., 2017**)**
  - Manipulating Digital Voltage Frequency Scaling (DVFS) registers

Software activated fault injection

- Possible when software can activate hardware vulnerabilities
- The vulnerabilities and faults are still in hardware!

Some recent examples...

- **Rowhammer** (Kim et al., 2014; many more afterwards)
  - Constantly reading a specific DDR address lead to bit flips
- **CLKSCREW (**Tang et al., 2017**)**
  - Manipulating Digital Voltage Frequency Scaling (DVFS) registers

No tooling required for software activated fault injection!

Trends

- Specialized equipment is becoming cheaper and available to the masses
- Equipment might not be needed at all (e.g. software activated fault injection)

Why Glitch?

Programmers' Mistakes

- **Arithmetic bugs** (e.g., div by zero, integer overflow, . . .)
- **Logical bugs** (e.g., Infinite loops, . . .)
- **Syntax bugs** (e.g., assignment instead of comparison, . . .)
- **Multi-threaded bugs** (e.g., deadlocks, race conditions, . . .)
- **Interfacing bugs** (e.g., incorrect API use, . . .)
- **Resource bugs** (e.g., uninitialized variables, buffer overflows, . . .)

```
1 #include <string.h>
2
3 void foo (char *bar)
4 {
5     char c[12];
6
7     strcpy(c, bar); // no bounds checking
8 }
9
10 int main (int argc, char **argv)
11 {
12     foo(argv[1]);
13 }
```


Why glitch?

Scenario

You want to break a device.

You don’t have the code.

Question

What now?

An Application's Memory Layout

```
27 //secret.c
28 #include "secret.h"
29
30 static int tries_left = 3;
31 static int PIN = 1234;
32 static int secret = 666;
33
34 int ENTRYPOINT get_secret(int provided_pin) {
35   if (tries_left > 0) {
36     if (PIN == provided_pin) {
37       tries_left = 3;
38       return secret;
39     } else {
40       tries_left--;
41       return 0;
42     }
43   } else
44     return 0;
45 }
```

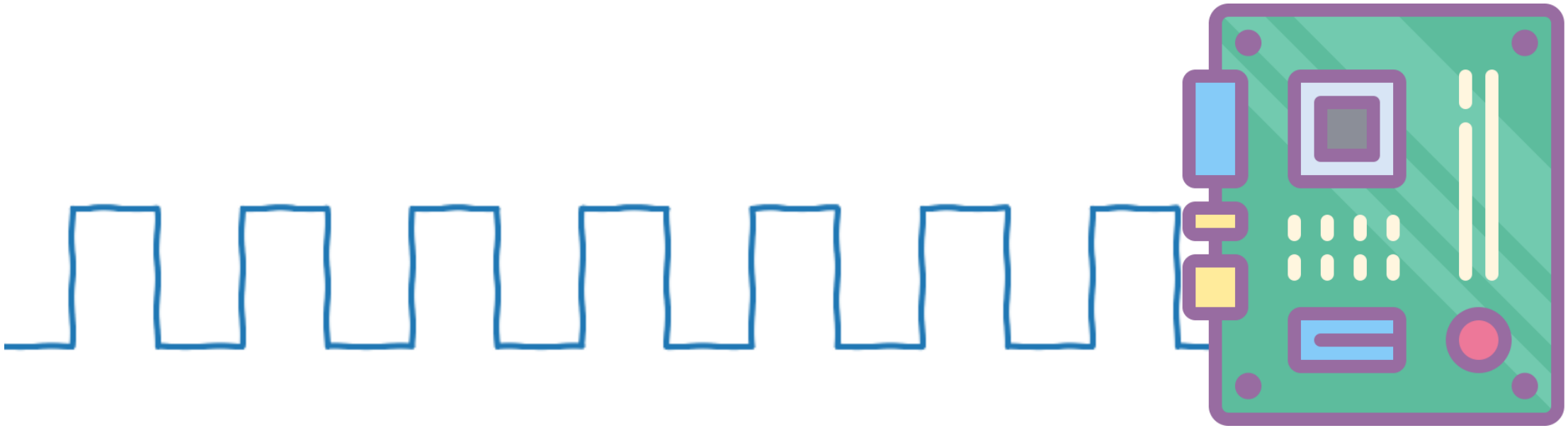
A pin check

```
uint8_t buf[4];
usart_getbuf(buf,sizeof(buf));

uint8_t correct=0;

for(uint8_t i=0;i<sizeof(buf);i++)
    if( buf[i] == pin[i] )
        correct++;

if(correct == sizeof(buf))
    usart_putstring("Pin Correct\r\n");
else
    usart_putstring("Bad pin\r\n");
```

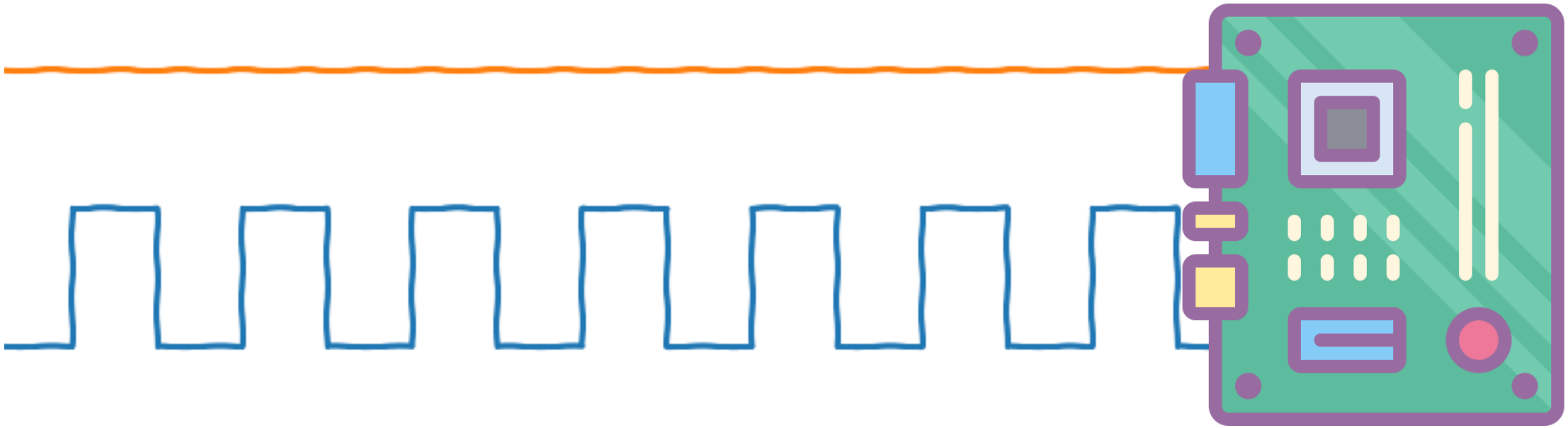
Classic FI goals

- Bypassing security measures
- Recovering keys

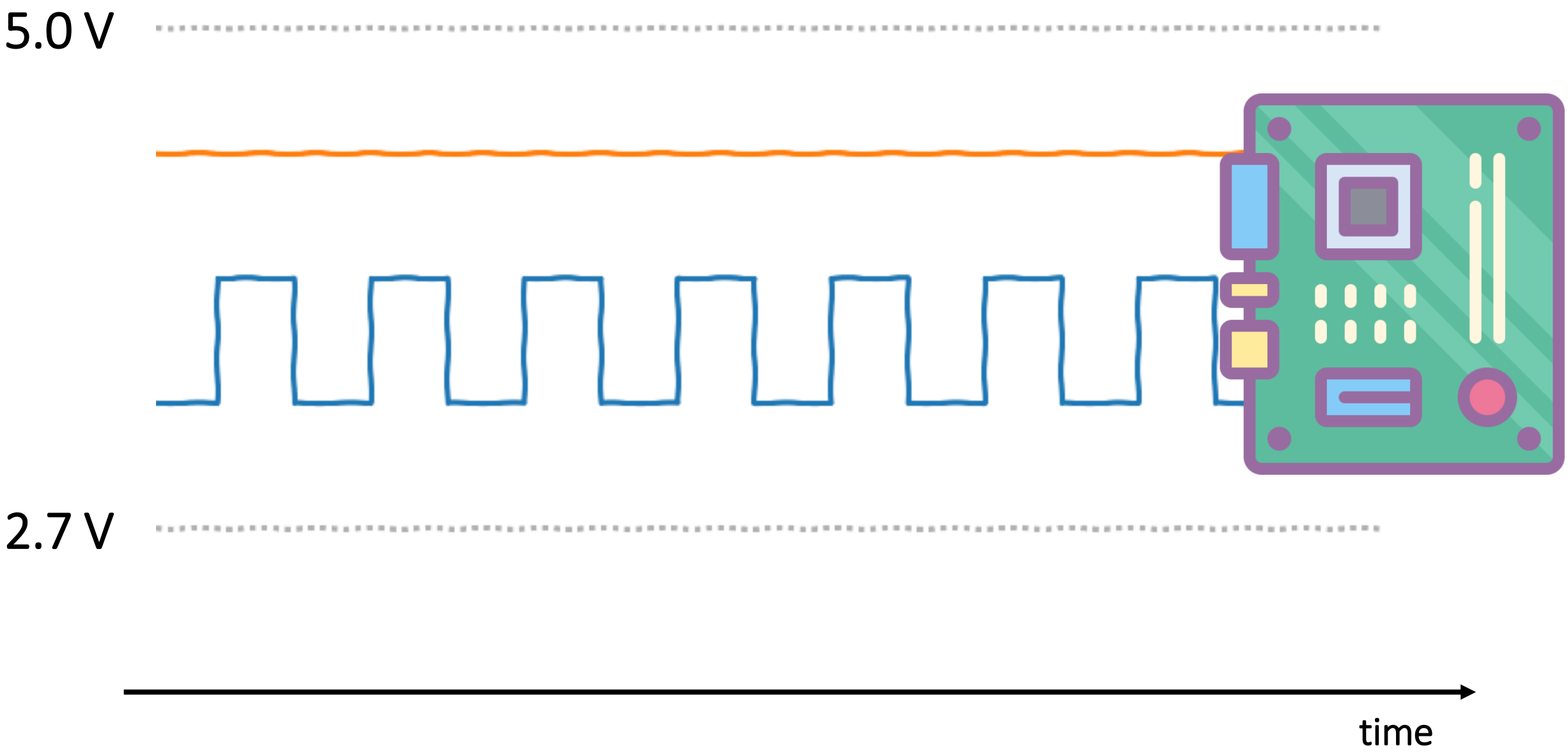
How do we glitch?

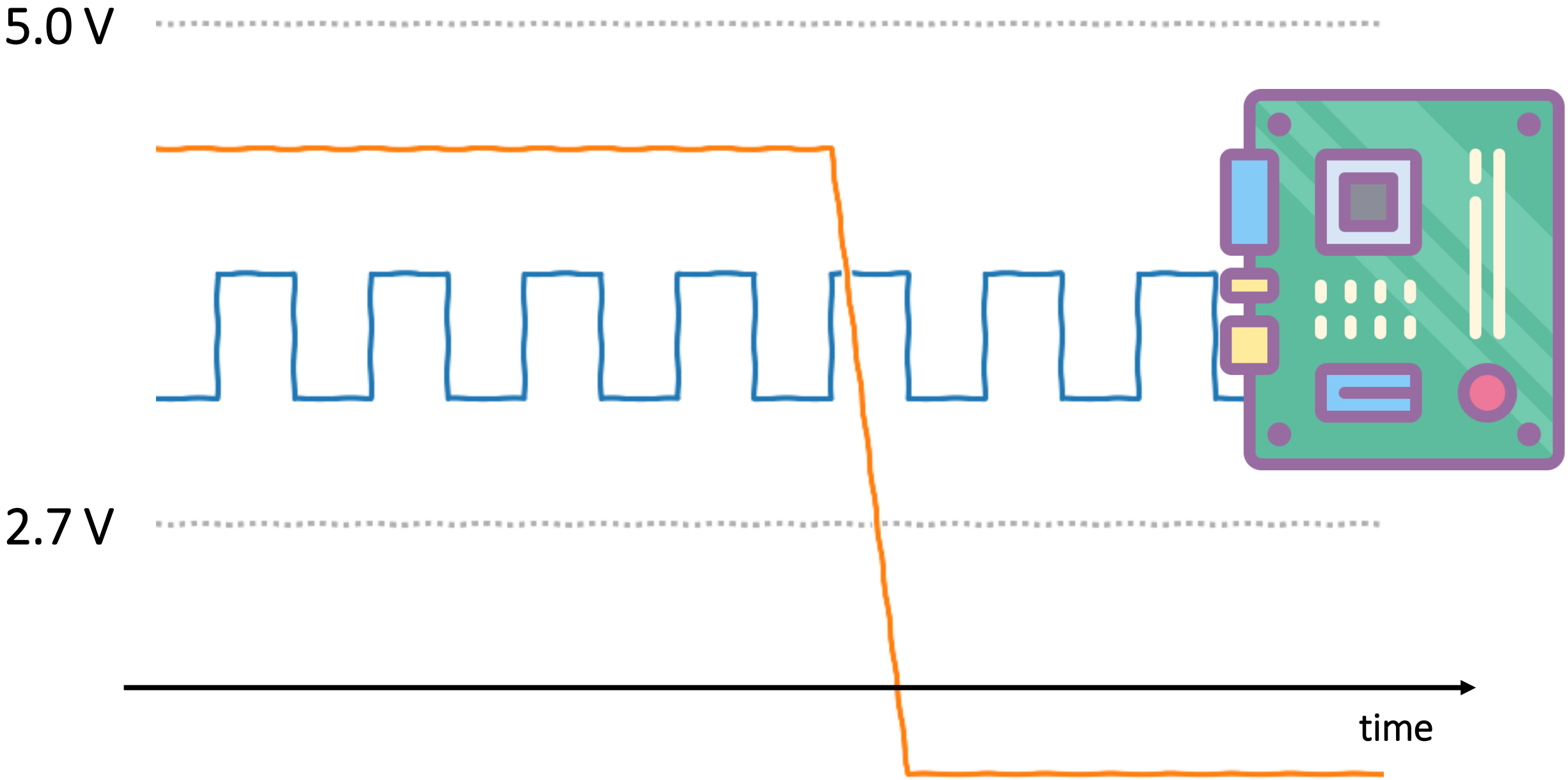


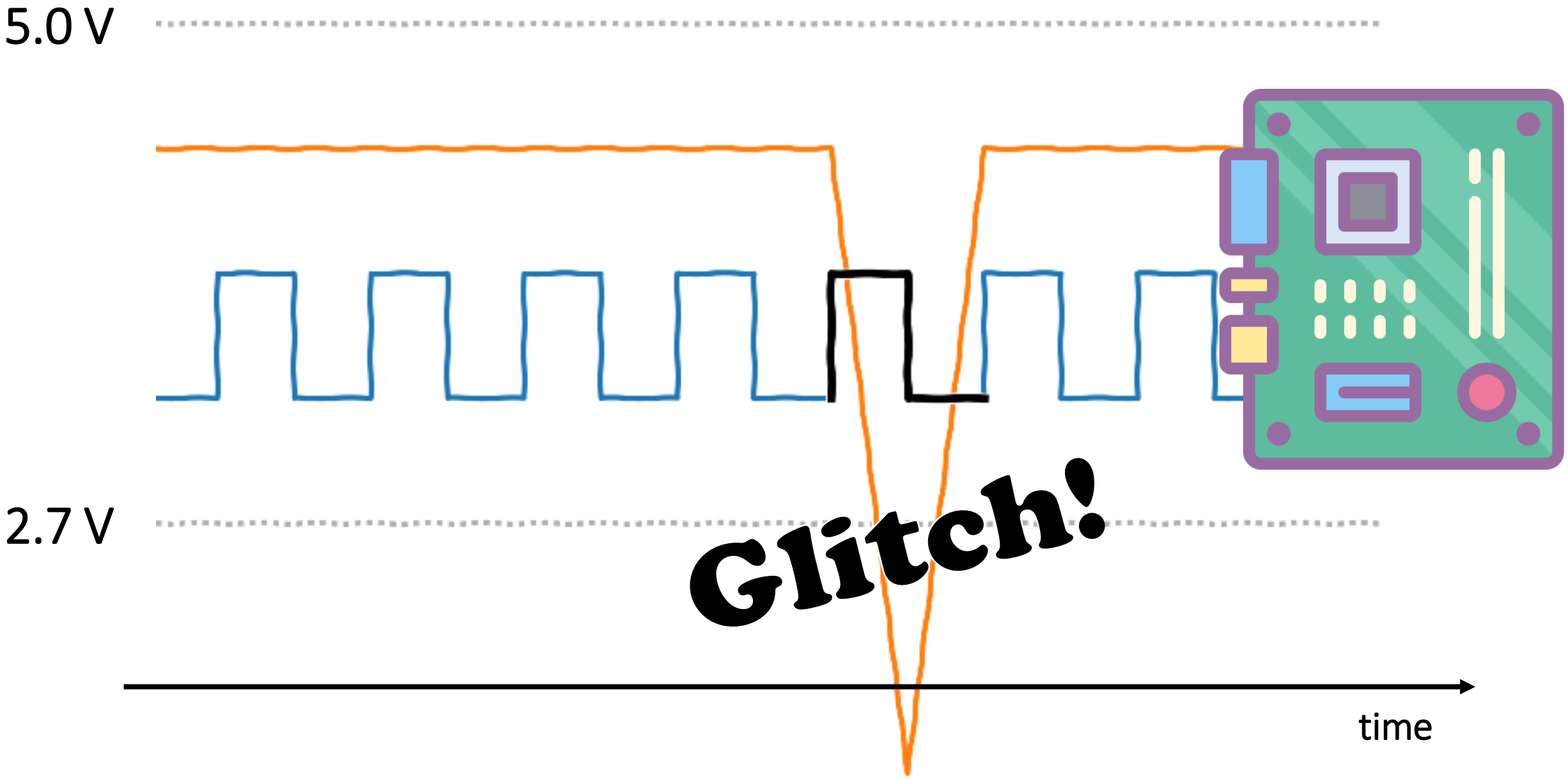
time



time


5.0 V
2.7 V
time


5.0 V
2.7 V
time


5.0 V
2.7 V
Glitch!
time

What happens when we glitch?

Things go wrong!

Process of (voltage) glitching

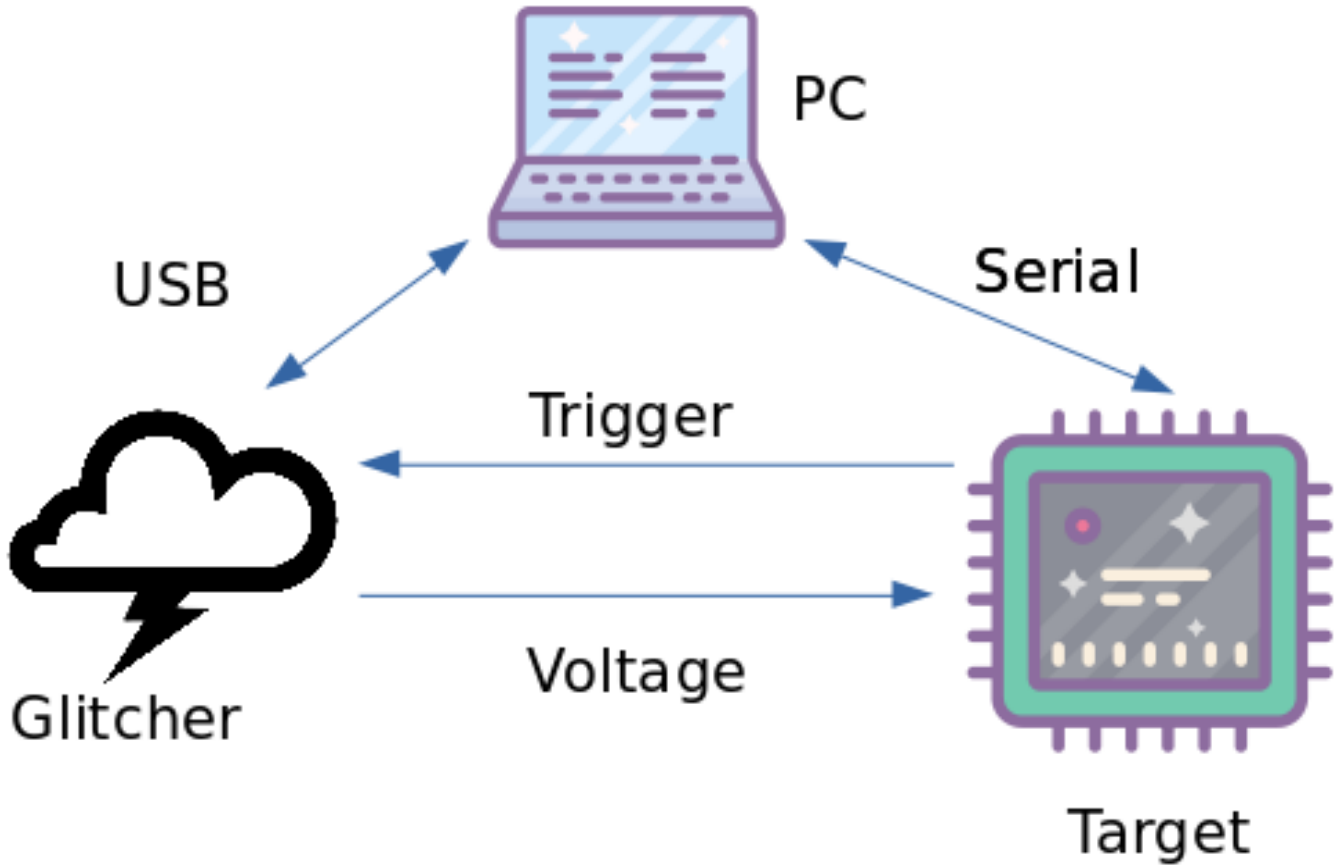
Step 1: Prepare the setup

Step 2: Prepare the target

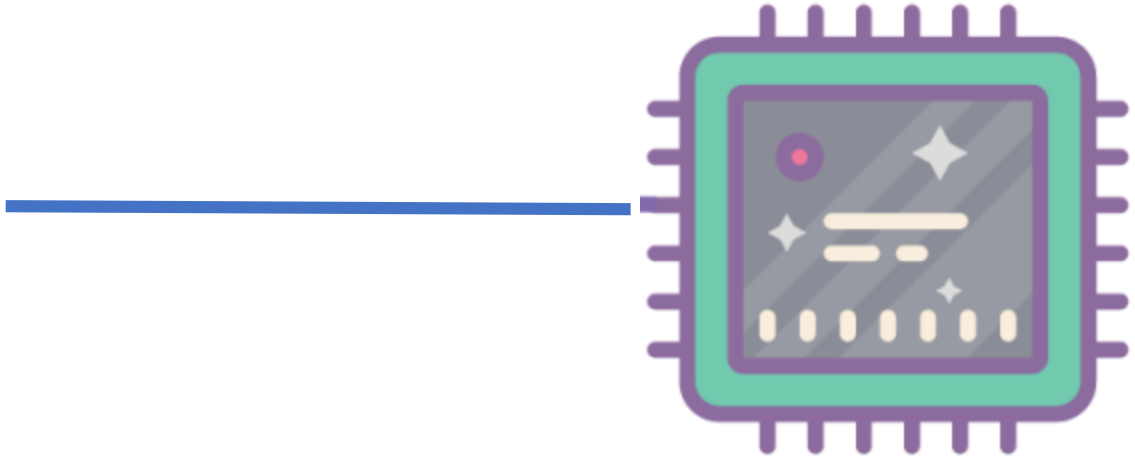
Step 3: Characterize the target

Step 4: Perform the attack

Setup

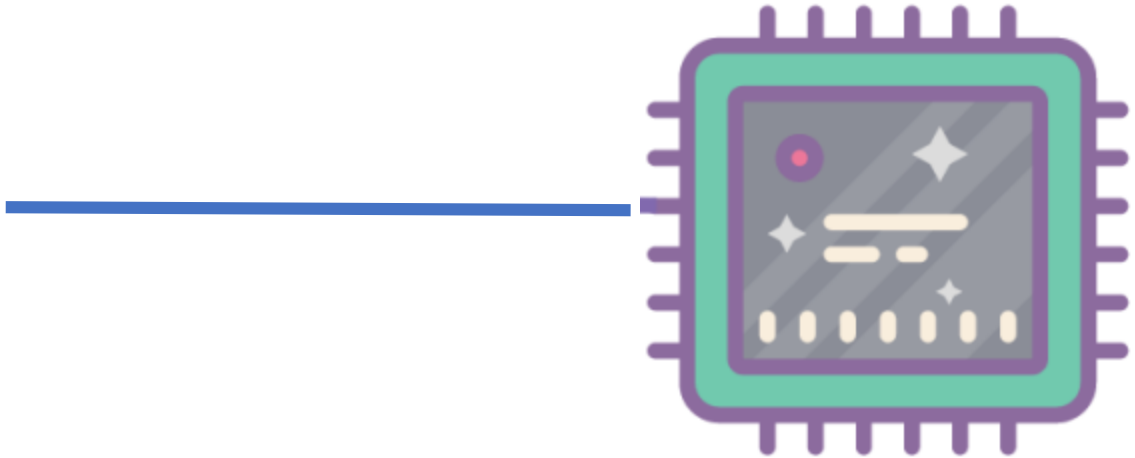


Prepare target



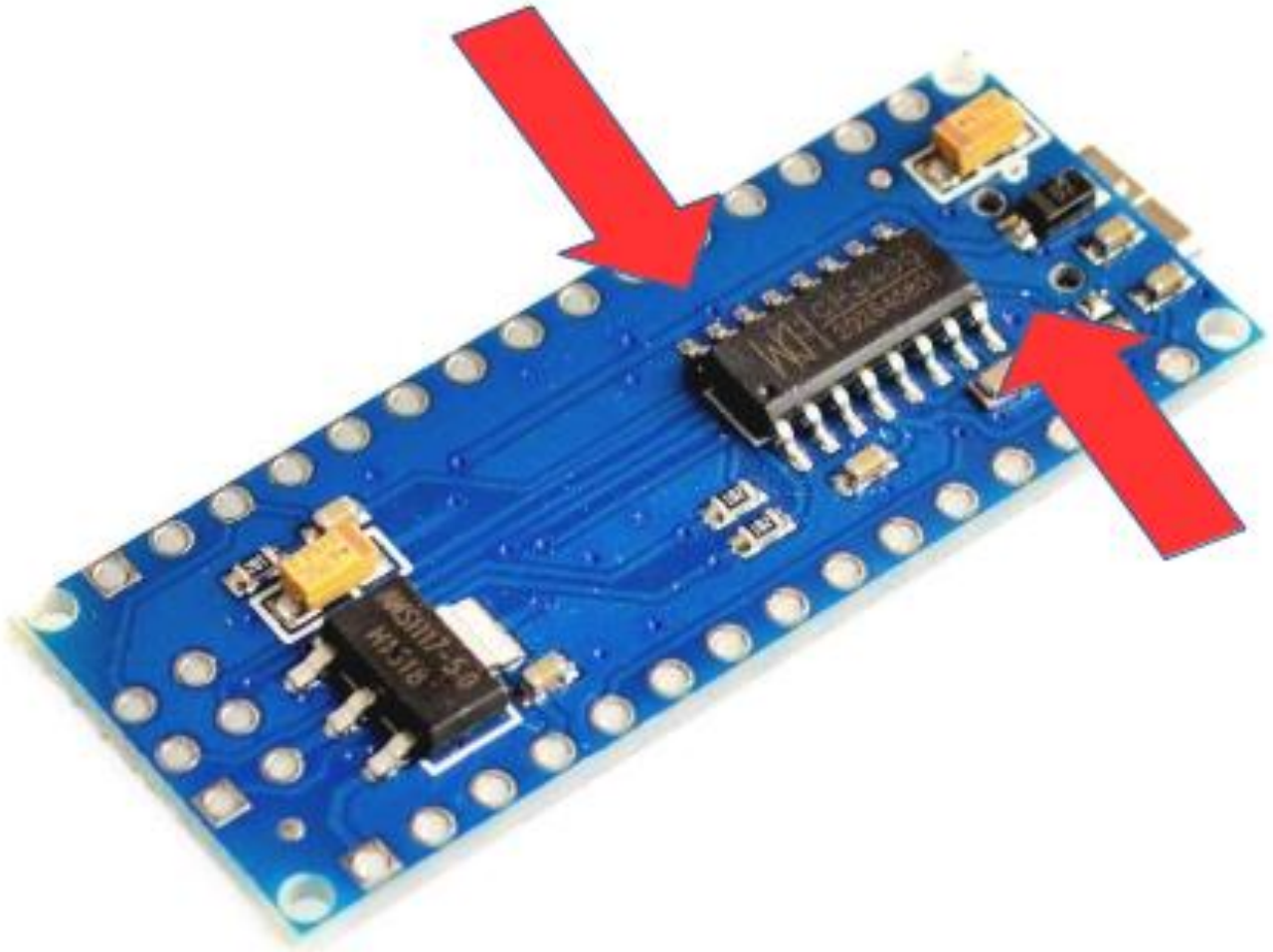
Prepare target

+5v



Prepare target

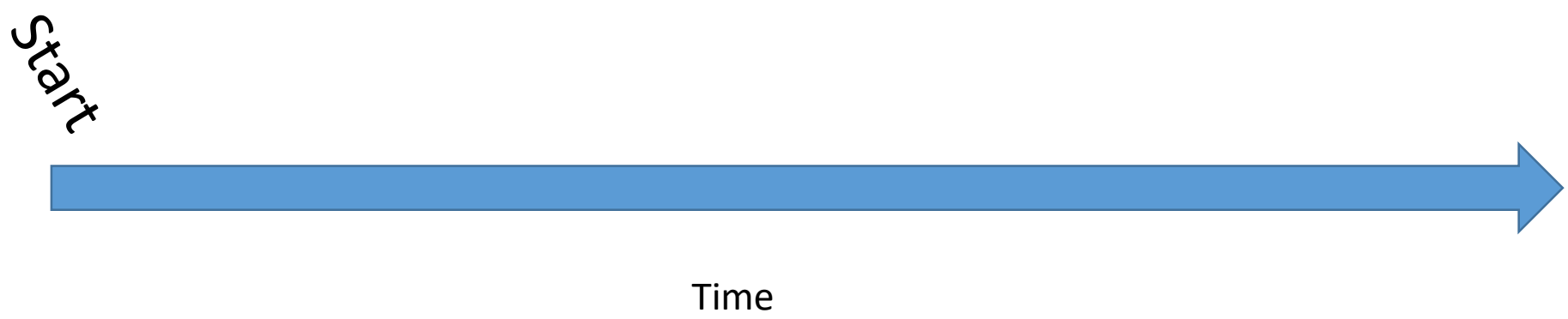
Simple right?



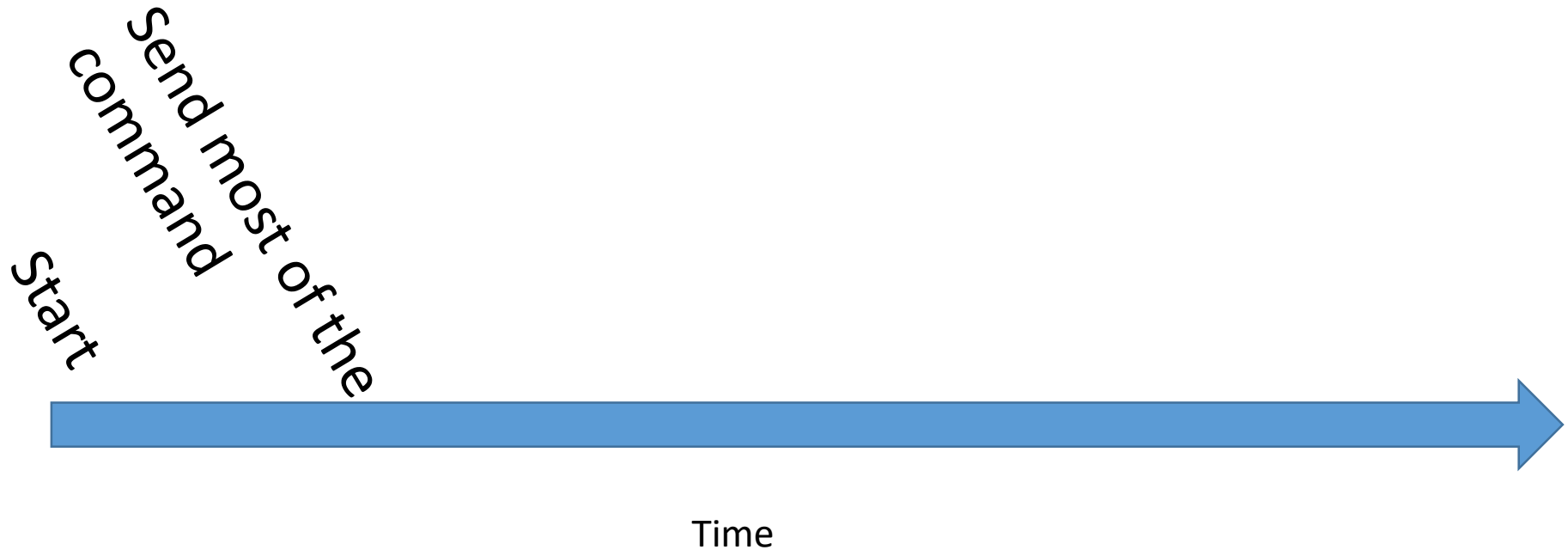
Setup timeline

Time

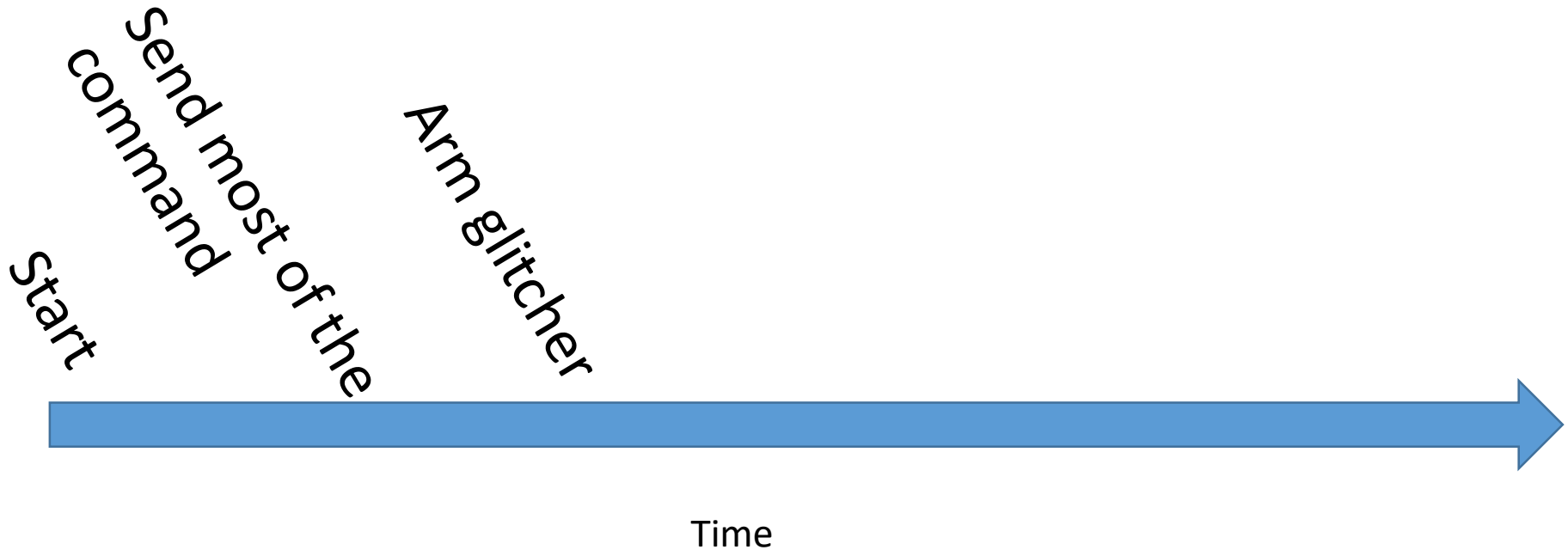
Setup timeline



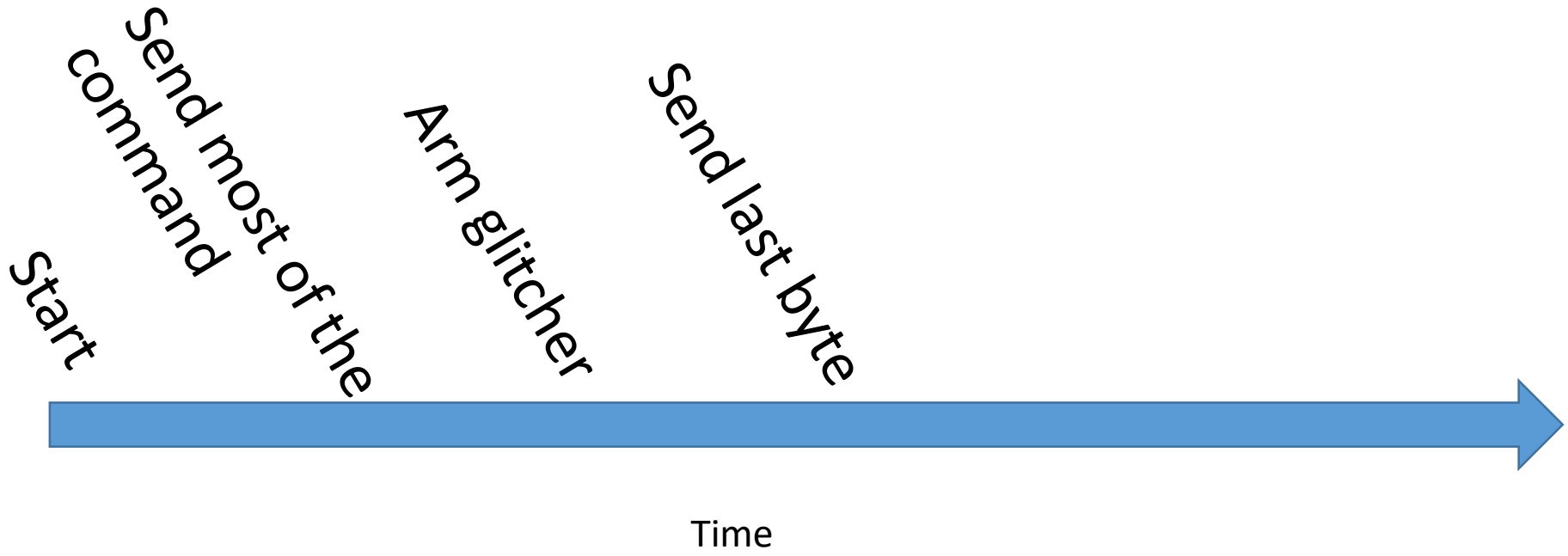
Setup timeline



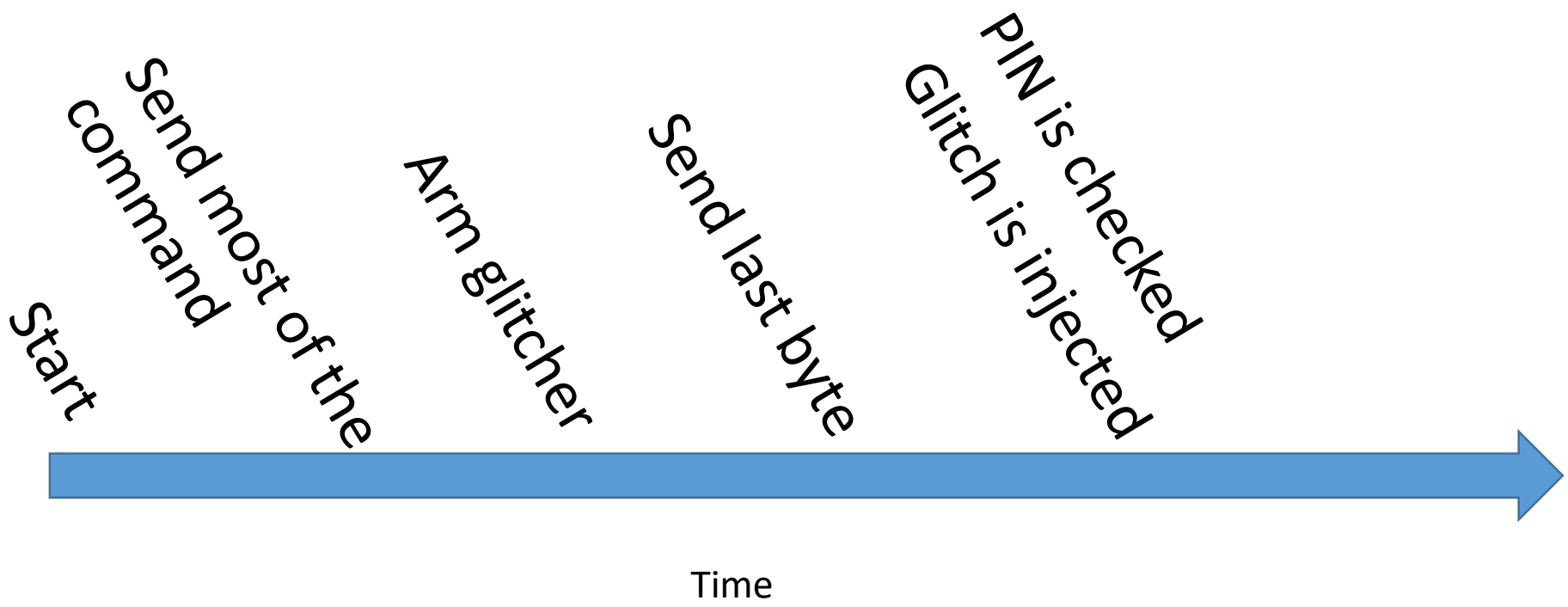
Setup timeline



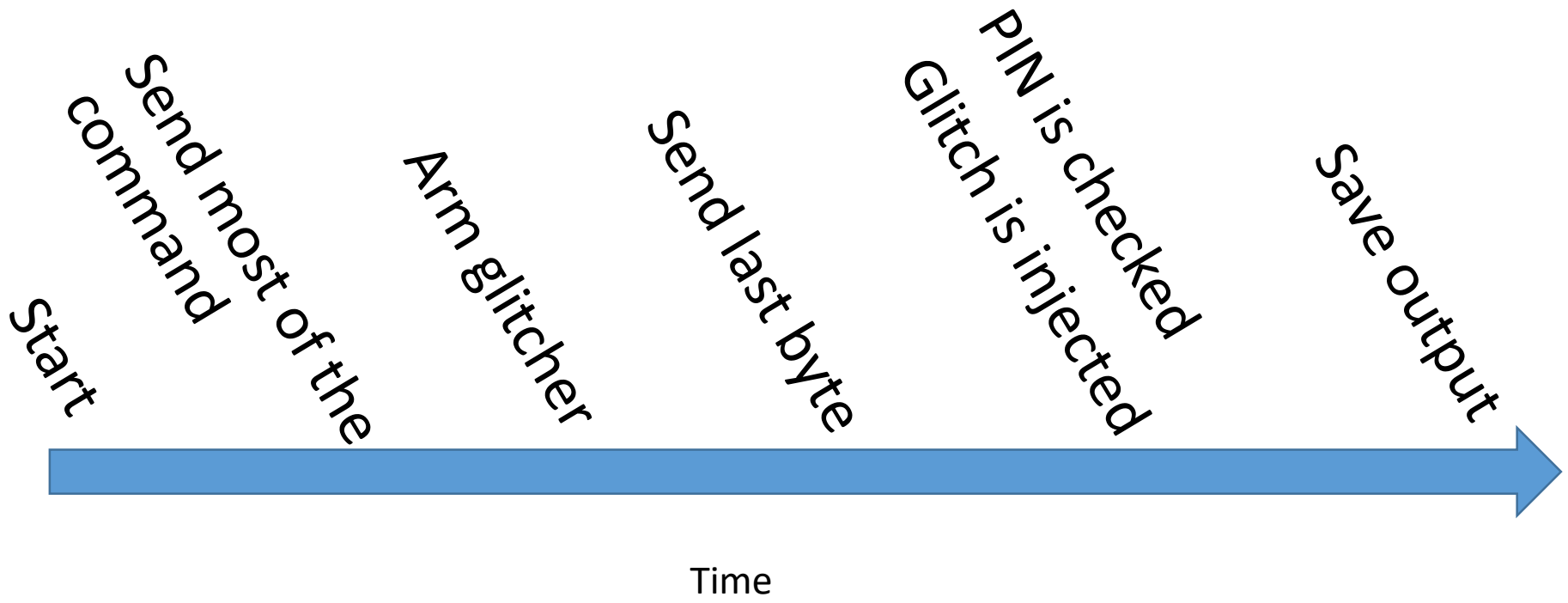
Setup timeline



Setup timeline



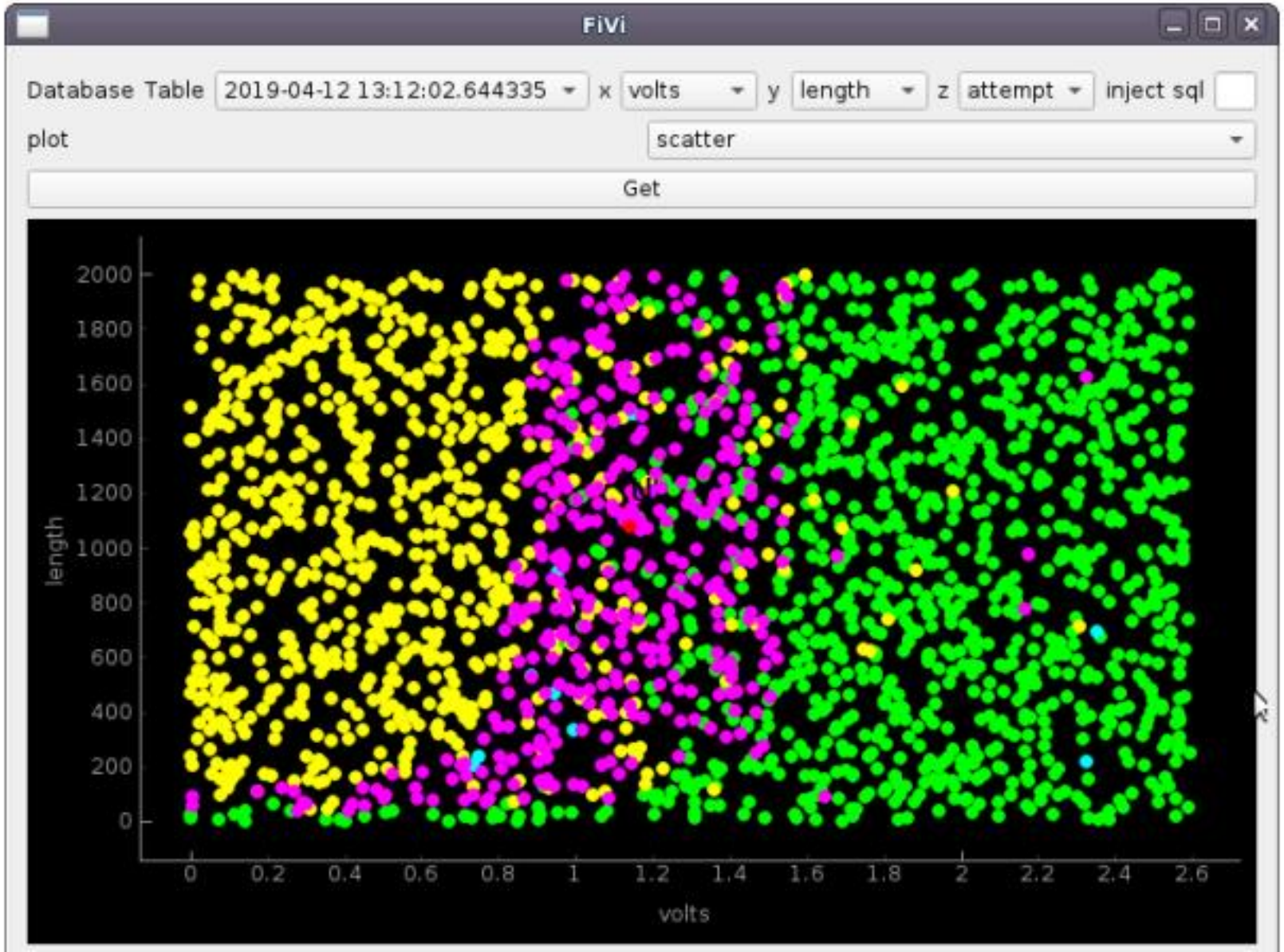
Setup timeline



Characterization

- Test code on an open target
- Determine:
  - if we can glitch
  - good parameters

Looooooots of attempts



DEMO

- Cheap hardware setup
- Pin check: The “Hello World” of FI

Fault injection breaks things!

You cannot expect that your software executes as intended!

Questions?!