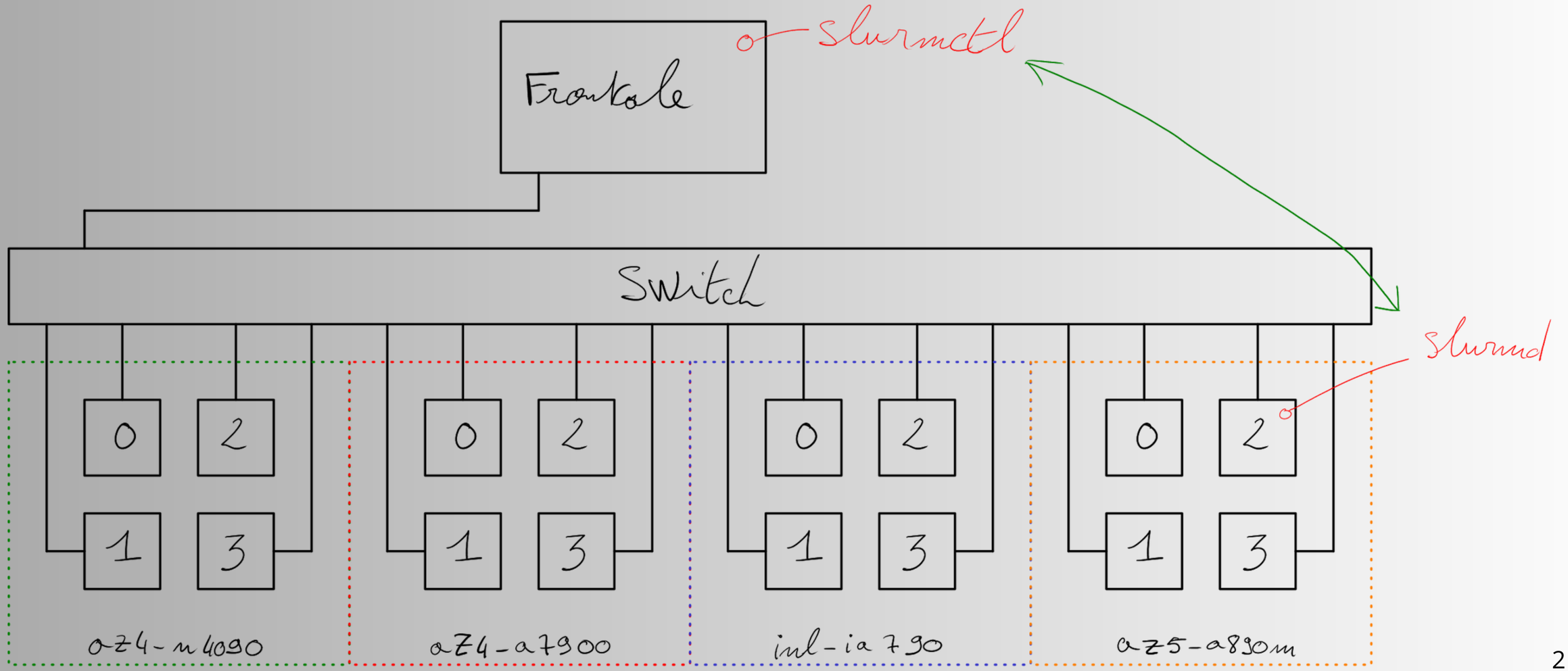
Contrôle des LEDs du cluster DAL3K
Soutenance finale

Rappel du contexte



Rappel des objectifs

- Contrôle de bandeaux ARGB
- Indicateurs visuels d’utilisation
- Extinction automatique des noeuds pour économies d’énergie

Contrôle de bandeaux de LEDs ARGB

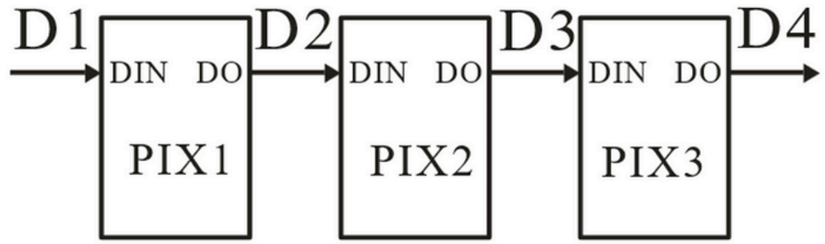
Rappels du protocole WS2811B

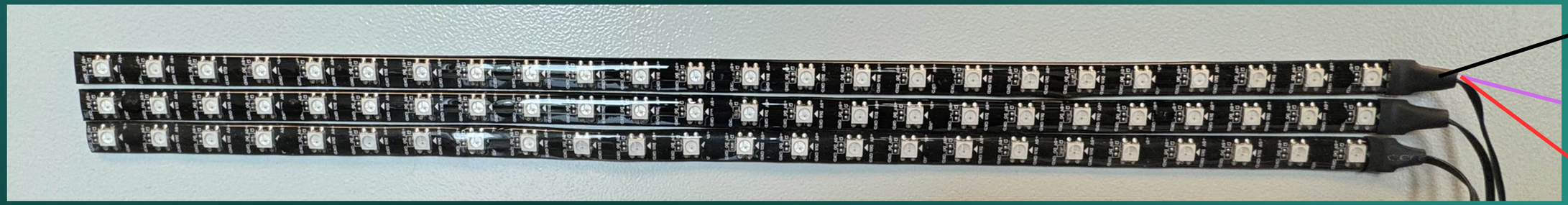
Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

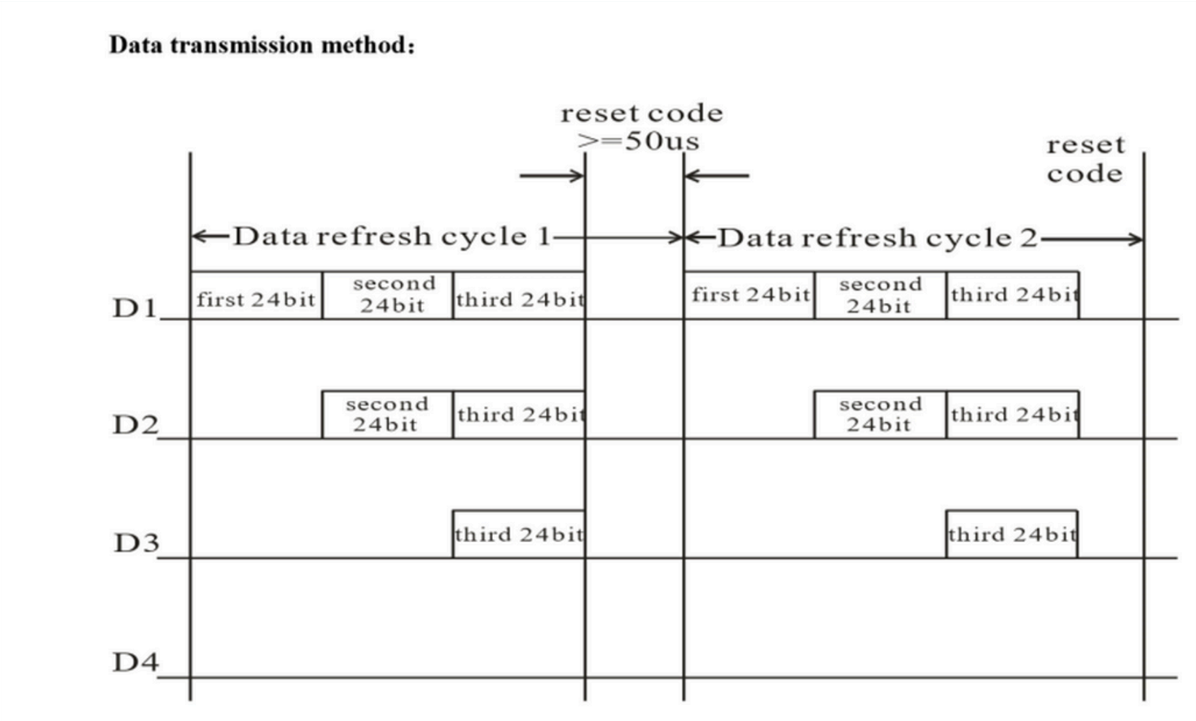
Note: Follow the order of GRB to sent data and the high bit sent at first.

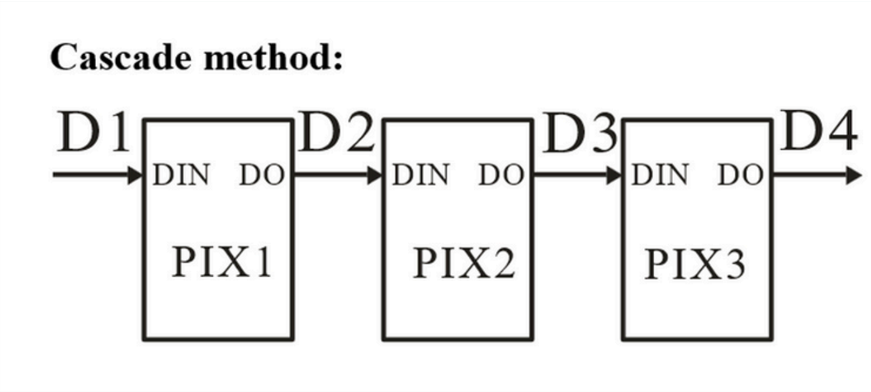
Cascade method:





Rappels du protocole WS2811B

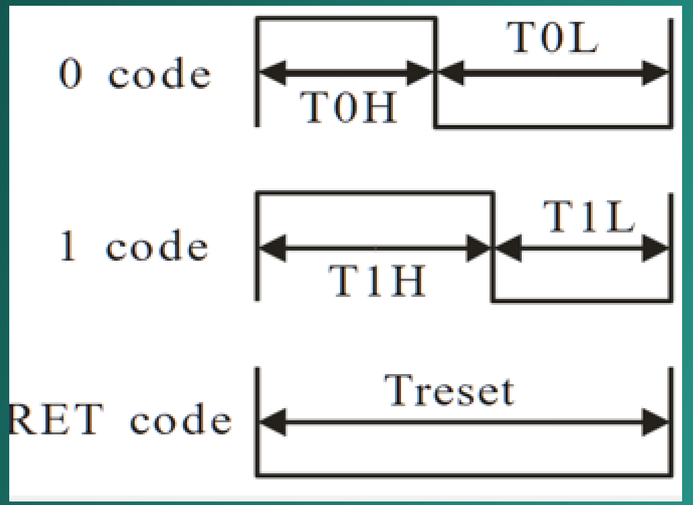




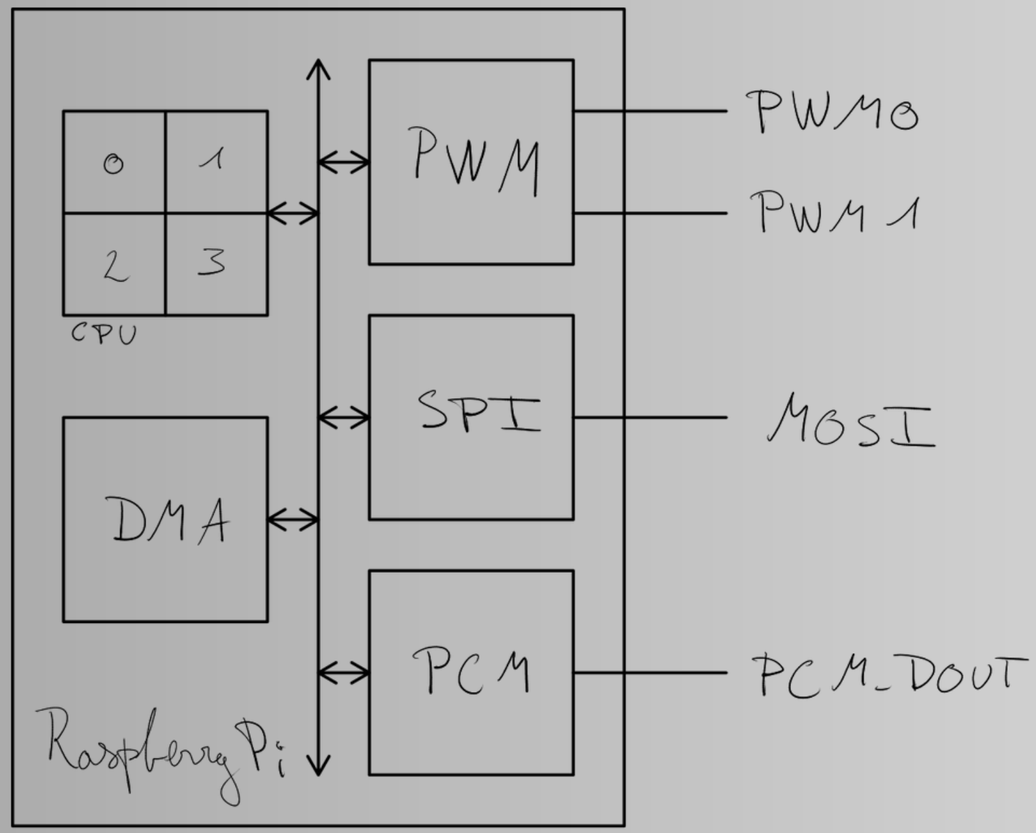
Contrôle bas niveau via WiringPi

Logique d’origine : changer le rapport cyclique à chaque front descendant
Basé sur des interruptions hardware
Fonctionne jusqu’à 400kHz
Trop lent pour 800kHz

- Bit 0: 400ns(±150ns) HL → 850ns(±150ns) LL
- Bit 1: 850ns(±150ns) HL → 400ns(±150ns) LL



Contrôle haut niveau via rpi_281x



3v3 Power	1		2	5v Power
GPIO 2 (I2C1 SDA)	3		4	5v Power
GPIO 3 (I2C1 SCL)	5		6	Ground
GPIO 4 (GPCLK0)	7		8	GPIO 14 (UART TX)
Ground	9		10	GPIO 15 (UART RX)
GPIO 17	11		12	GPIO 18 (PCM CLK)
GPIO 27	13		14	Ground
GPIO 22	15		16	GPIO 23
3v3 Power	17		18	GPIO 24
GPIO 10 (SPI0 MOSI)	19		20	Ground
GPIO 9 (SPI0 MISO)	21		22	GPIO 25
GPIO 11 (SPI0 SCLK)	23		24	GPIO 8 (SPI0 CE0)
Ground	25		26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27		28	GPIO 1 (EEPROM SCL)
GPIO 5	29		30	Ground
GPIO 6	31		32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33		34	Ground
GPIO 19 (PCM FS)	35		36	GPIO 16
GPIO 26	37		38	GPIO 20 (PCM DIN)
Ground	39		40	GPIO 21 (PCM DOUT)

Contrôle haut niveau via rpi_281x

Structure principale : ws2811_t

- Fréquence de base (800kHz)
- Pin du DMA
- Tableau de ws2811_channel_t

Structure channel

- Pin du module PWM/PCM/SPI
- Nombre de LEDs
- Polarité
- Luminosité maximale
- Ordre d’envoi des octets

```
ws2811_t ledstring = {
    .freq = WS2811_TARGET_FREQ,
    .dmanum = 10,
    .channel = {
        [0] = {
            .gpionum = 18,
            .count = 60,
            .invert = 0,
            .brightness = 255,
            .strip_type = WS2811_STRIP_GRB,
        },
    },
};
```

Contrôle haut niveau via rpi_281x

Exemples minimalistes

```
ws2811_init(&ledstring);

ledstring.channel[0].leds[0] = 0x00FF00; // LED verte

ws2811_render(&ledstring);

sleep(1);

ws2811_fini(&ledstring);
```

```
void clear_channel(ws2811_channel_t *channel) {
    for (int i = 0; i < LED_COUNT; i++) {
        channel->leds[i] = 0;
    }
}
void set_color(ws2811_channel_t *channel, uint8_t r, uint8_t g, uint8_t b) {
    for (int i = 0; i < LED_COUNT; i++) {
        channel->leds[i] = (r << 16) | (g << 8) | b;
    }
}
```

Contrôle haut niveau via rpi_281x

Modes et changements dynamiques

```
enum Mode {
    MODE_STATIC = 0,
    MODE_BLINK = 1,
    MODE_CHASE = 2,
    MODE_FILL = 3,
    MODE_FILLREV = 4,
    MODE_FLOW = 5,
    MODE_BREATHE = 6,
    MODE_IDLE = 7,
    MODE_RAINBOW = 8,
    MODE_COMET = 9,
    MODE_WAVE = 10
};
```

- Lecture du pipe
- Récupération de l’état
- Récupération de l’utilisation CPU
- Mise à jour du mode

```
0: mode[i] = MODE_STATIC; r[i] = v[i] = b[i] = 0; break;
1: mode[i] = MODE_IDLE; r[i] = r_def[i]; v[i] = v_def[i]; b[i] = b_def[i];b
```

```
case 2:
    r[i] = r_def[i]; v[i] = v_def[i]; b[i] = b_def[i]
    if (cpu >= 0 && cpu < 20) {
        mode[i] = MODE_BREATHE;
    }
    else if (cpu < 40) {
        mode[i] =MODE_FILLREV;
    }
    else if (cpu < 60) {
        mode[i] =MODE_COMET;
    }
    else if (cpu < 80) {
        mode[i] =MODE_WAVE;
    }
    else{
        mode[i] =MODE_RAINBOW;
    }
```

Contrôle haut niveau via rpi_281x

Boucle principale

```
        for (int i = 0; i < nb_sections; i++) {
            apply_mode(&ledstring.channel[0], &sections[i], mode[i], r[i], v[i], b[i], tick, taux_cpu[i]);
            if (parallel) apply_mode(&ledstring.channel[1], &p_sections[i], p_mode[i], p_r[i], p_v[i],
p_b[i], tick, p_taux_cpu[i]);
        }


        ws2811_render(&ledstring);
        usleep(10000);
        tick = (tick + 1) % 1000000;
```

Contrôle haut niveau via rpi_281x

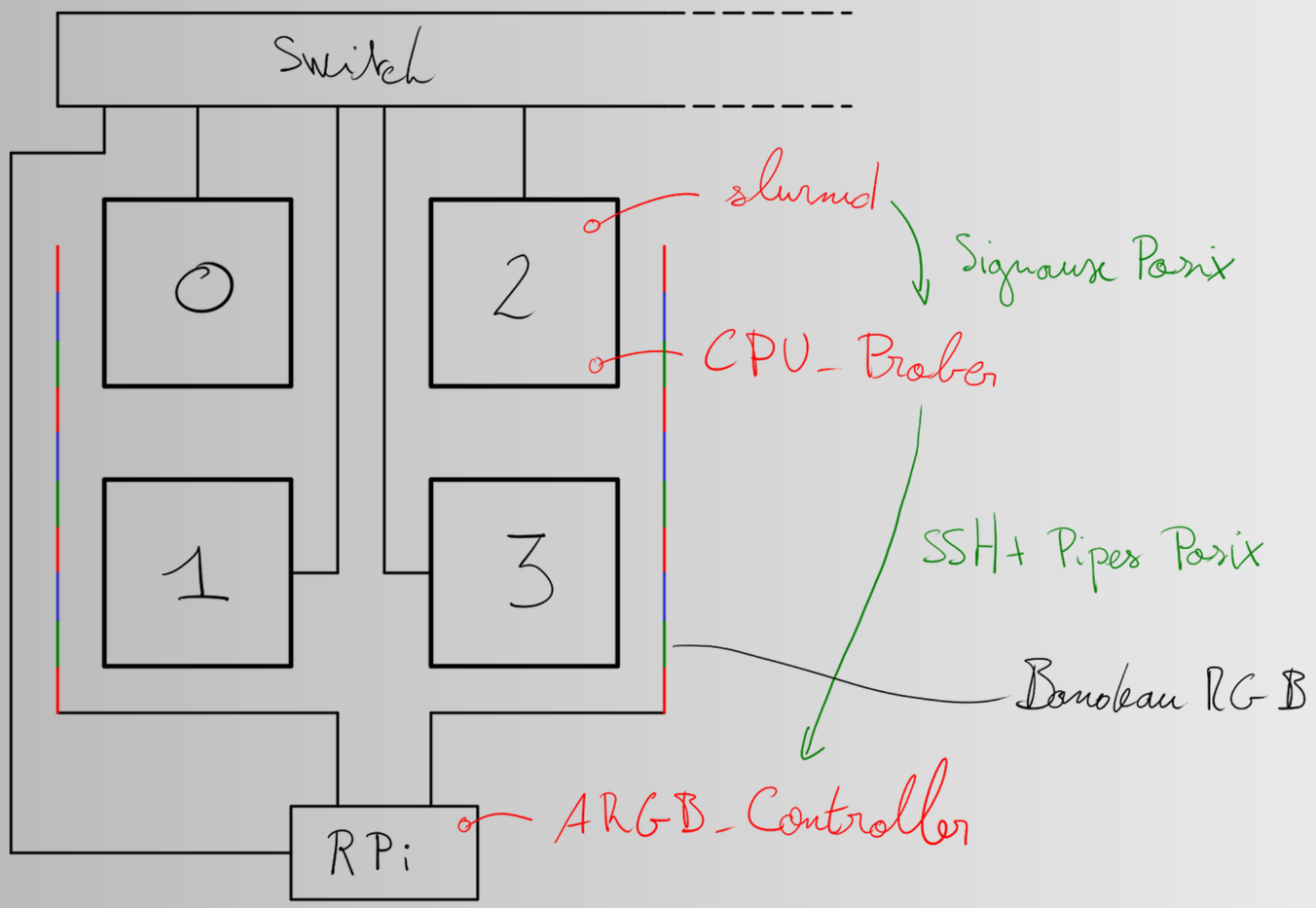
Paramètres de lancement :

- Nombre de sections
- Mode parallèle ou unique
- Nombre de LEDs par section
- Couleurs de base
- Effet de base
- Température simulée
- Préfixe des pipes

```
sudo ./final_1_fifo --sections 6 --parallel 1 \
  --l1 8  --r1 100 --g1 0 --b1 0 --m1 static --tcpu1 80 \
  --l2 8  --r2 0 --g2 255 --b2 0 --m2 blink --tcpu2 70 \
  --l3 8  --r3 0 --g3 0 --b3 255 --m3 chase --tcpu3 60 \
  --l4 8  --r4 128 --g4 128 --b4 0 --m4 fill --tcpu4 50 \
  --l5 8  --r5 0 --g5 128 --b5 128 --m5 fillrev --tcpu5 40 \
  --l6 8  --r6 128 --g6 0 --b6 128 --m6 flow --tcpu6 30 \
  --pl1 8 --pr1 255 --pg1 0 --pb1 0 --pm1 static --ptcpu1 10 \
  --pl2 8 --pr2 0 --pg2 255 --pb2 0 --pm2 blink --ptcpu2 20 \
  --pl3 8 --pr3 0 --pg3 0 --pb3 255 --pm3 static --ptcpu3 60 \
  --pl4 8 --pr4 128 --pg4 128 --pb4 0 --pm4 fill --ptcpu4 40 \
  --pl5 8 --pr5 0 --pg5 128 --pb5 128 --pm5 fillrev --ptcpu5 40 \
  --pl6 8 --pr6 128 --pg6 0 --pb6 128 --pm6 fillrev --ptcpu6 30 \
  --fifo-prefix fifo --pfifo-prefix fifop
```

Récupération des données d’utilisation

Organisation des communications et scripts



Plugin SPANK

Interface de plugin SLURM

- Bas niveau (compilé avec SLURM)

OU

- Par appels via SPANK (Slurm Plug-in Architecture for Node and job (K)control

5 contextes SPANK : local, **remote**, allocator, slurmd, job_script

Fonction slurm_spank_init() exécutée au lancement d'un job
Fonction slurm_spank_fini() exécutée à la fin du job

Envois de signaux Posix vers le prober

CPU Prober

Lecture du fichier /proc/stat

```
proc_stat_stream = freopen("/proc/stat", "r", proc_stat_stream);
fscanf(proc_stat_stream, "%s %lu %lu %lu %lu", line, &user, &user_nice, &system_time, &idle);
```

Temps CPU en ticks : user, nice, system, idle, iowait, irq, softirq

```
current_user = user - old_user;
current_user_nice = user_nice - old_user_nice;
current_system = system_time - old_system;
current_idle = idle - old_idle;

total = current_user + current_user_nice + current_system + current_idle;
used_sum = current_user + current_user_nice + current_system;

old_user = user;
old_user_nice = user_nice;
old_system = system_time;
old_idle = idle;

if (total <= 0)
{
    UC = 0;
}
else
{
    UC = ((double)used_sum / (double)total) * 100;
}

values = "\"2," + std::to_string(UC) + "\"";
```

CPU Prober

Communication via SSH

```
if (fork() == 0)
{
    execl("/bin/ssh", "-v", "-p", port.c_str(), (ssh_user + "@" + host).c_str(), "echo", values.c_str(), ">>", link.c_str(), (char *)NULL);
}
```

Contrôle via signaux Posix

```
void sig_handler(int sig)
{
    printf("signal recu : %d\n", sig);
    switch (sig)
    {
    case SIGINT:
    case SIGTERM:
        stop = 1;
        break;

    case SIGUSR1:
        enabled = 1;
        signal(SIGUSR1, sig_handler);
        break;

    case SIGUSR2:
        enabled = 0;
        signal(SIGUSR2, sig_handler);
        break;
    }
}
```

```
while (!stop)
{
    if (!enabled)
    {
        if (fork() == 0) // on envoie une trame "idle"
        {
            execl("/bin/ssh", "-v", "-p", port.c_str(), (ssh_user + "@" + host).c_str(), "echo", "\"1,0\"", ">>", link.c_str(), (char *)NULL);
        }
        sigsuspend(&suspend_set); // suspend l'envoi des trames ==> noeud allumé mais sans job

        if (stop) // si le signal de réveil est SIGINT ou SIGTERM
        {
            break;
        }

        if (!enabled) // si on a été réveillé par autre chose que les signaux définis, ignorer la boucle
        {
            continue;
        }
    }
```

CPU Prober

Transformation en service systemd
Lancement, arrêt, et contrôle du prober

```
[Unit]
Description=CPU prober for RaspiRGB

[Service]
Type=simple

User=argb
Group=argb

ExecStart=/usr/bin/CPU_prober.x -d 3000 -u argb -h "az4-n4090-rpi" -p 22 -l "/home/argb/fifo0"
ExecStop=/bin/kill -15 -$MAINPID
```

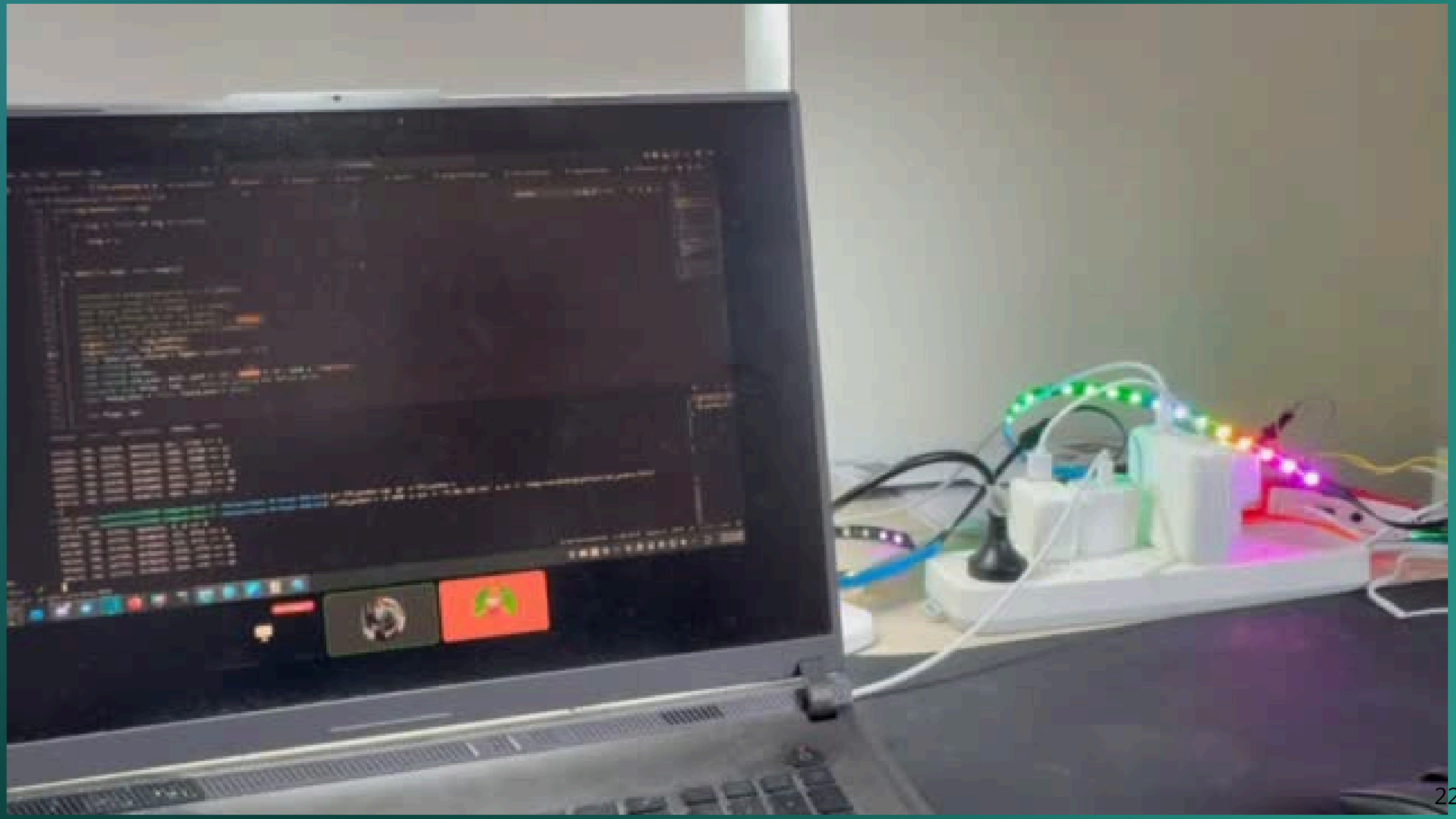
Contrôle automatique des nœuds

Contrôle des nœuds

Basé sur les scripts nodesuspend et noderesume définis pour slurmctl

```
#!/bin/bash
hosts=$(scontrol show hostnames "$1")
adresses=$(scontrol getaddrs "$(hosts)")
username="frontale"
logfile=/var/log/power_save.log
echo "$(date) Suspend invoked $0 $*" >>$logfile
for addr in $adresses
do
        ssh "$username@$addr" sudo poweroff
done
exit 0
```

```
#!/bin/bash
hosts=$(scontrol show hostnames "$1")
logfile=/var/log/power_save.log
echo "$(date) Resume invoked $0 $*" >>$logfile
for host in $hosts
do
        wol --name "$host"
done
exit 0
```



Merci de votre attention