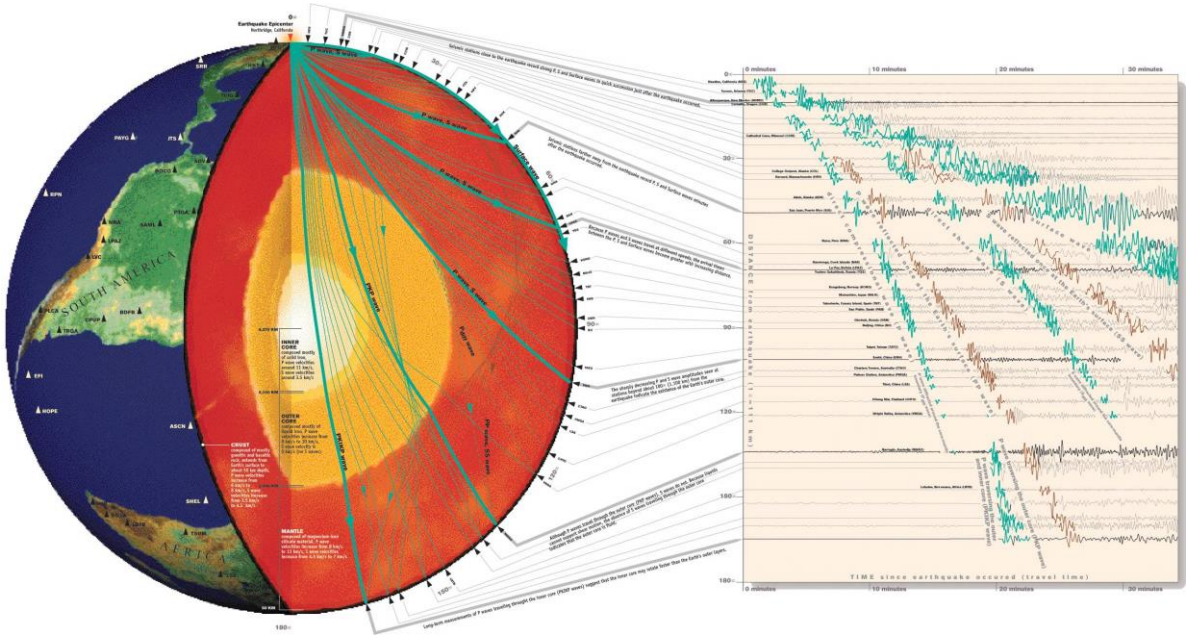
Un flotteur MERMAID pour l'écoute pluridisciplinaire des océans

Sébastien Bonnieux

SERENADE - 29 juin 2022

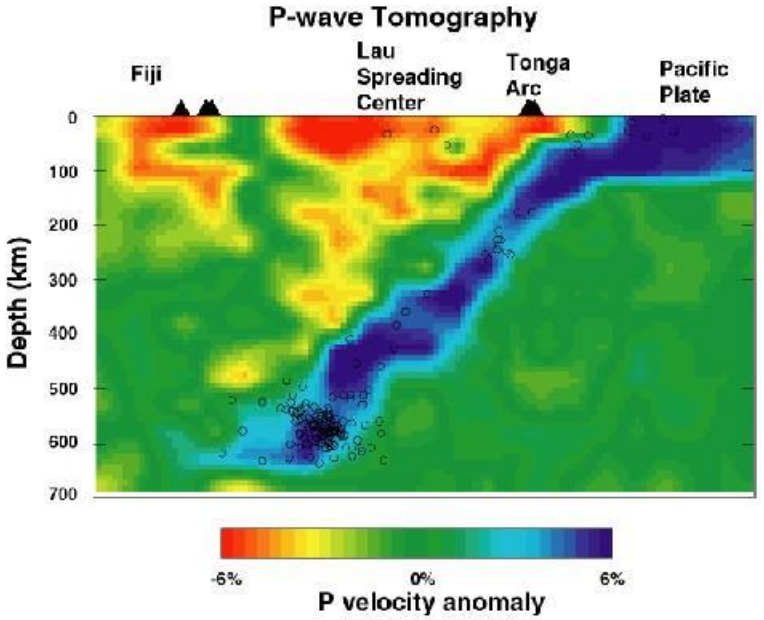
Avec: Karin Sigloch, Guust Nolet, Yann Hello, Olivier Philippe, Mireille Blay-Fornarino et Sébastien Mosser

La tomographie sismique

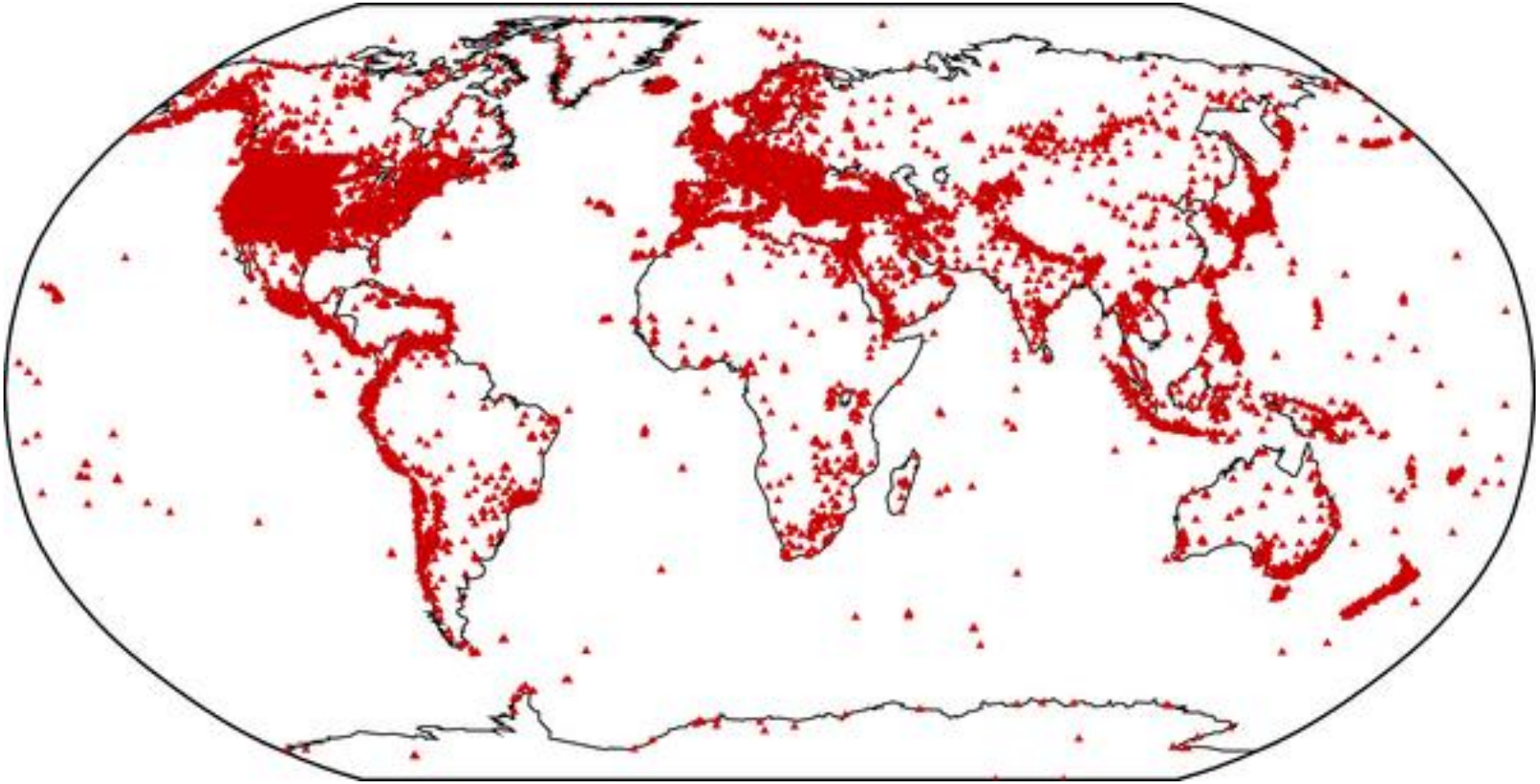


Sismogrammes

Image tomographique

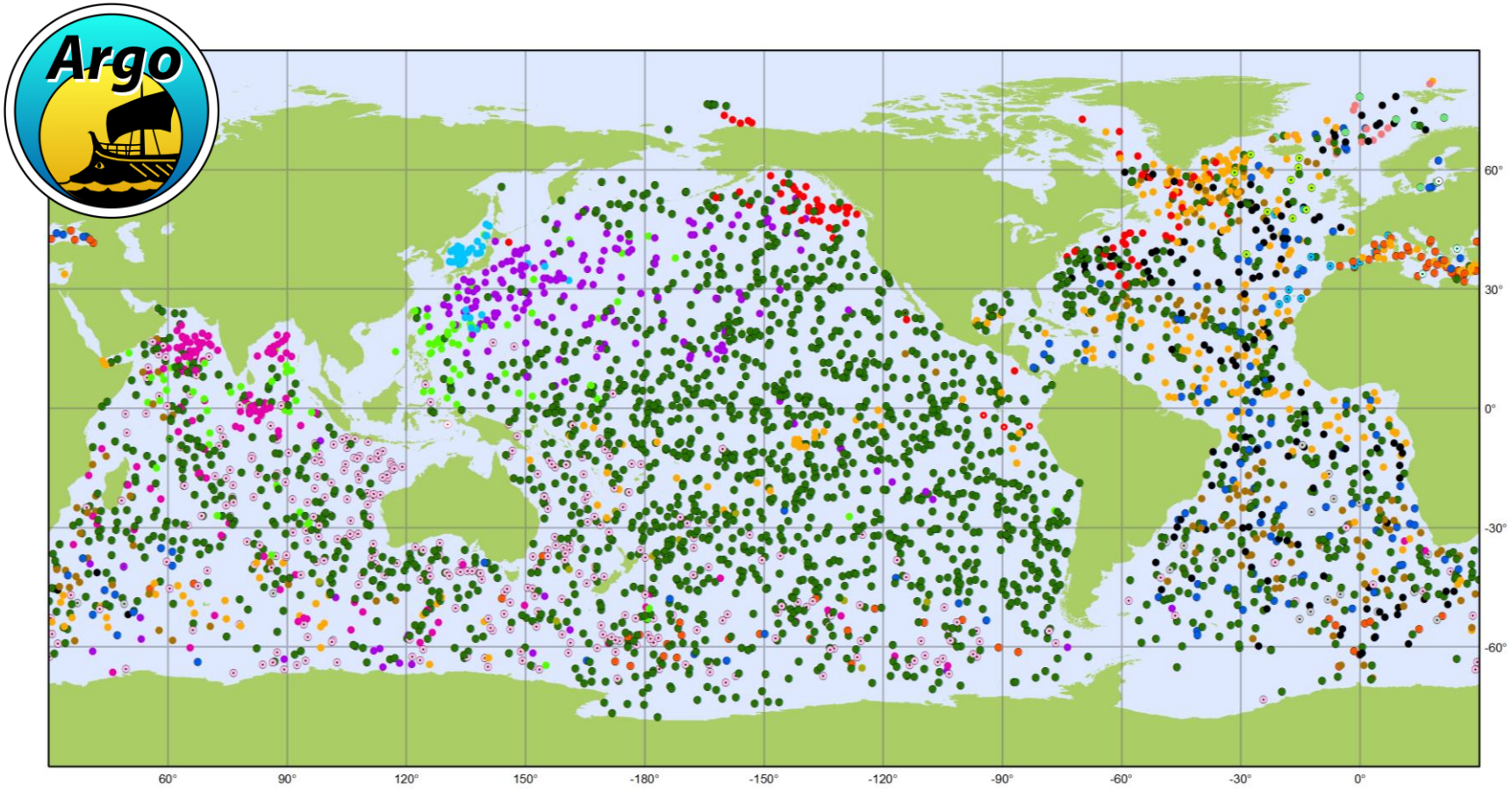


Carte de l'International Registry of Seismograph Stations



Il y a peu de stations sismologiques en mer !

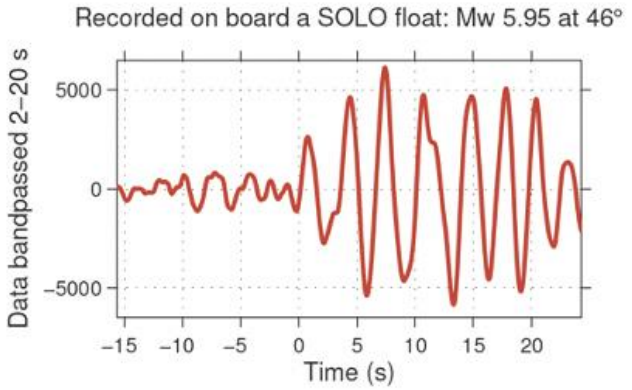
Un réseau à l'échelle globale



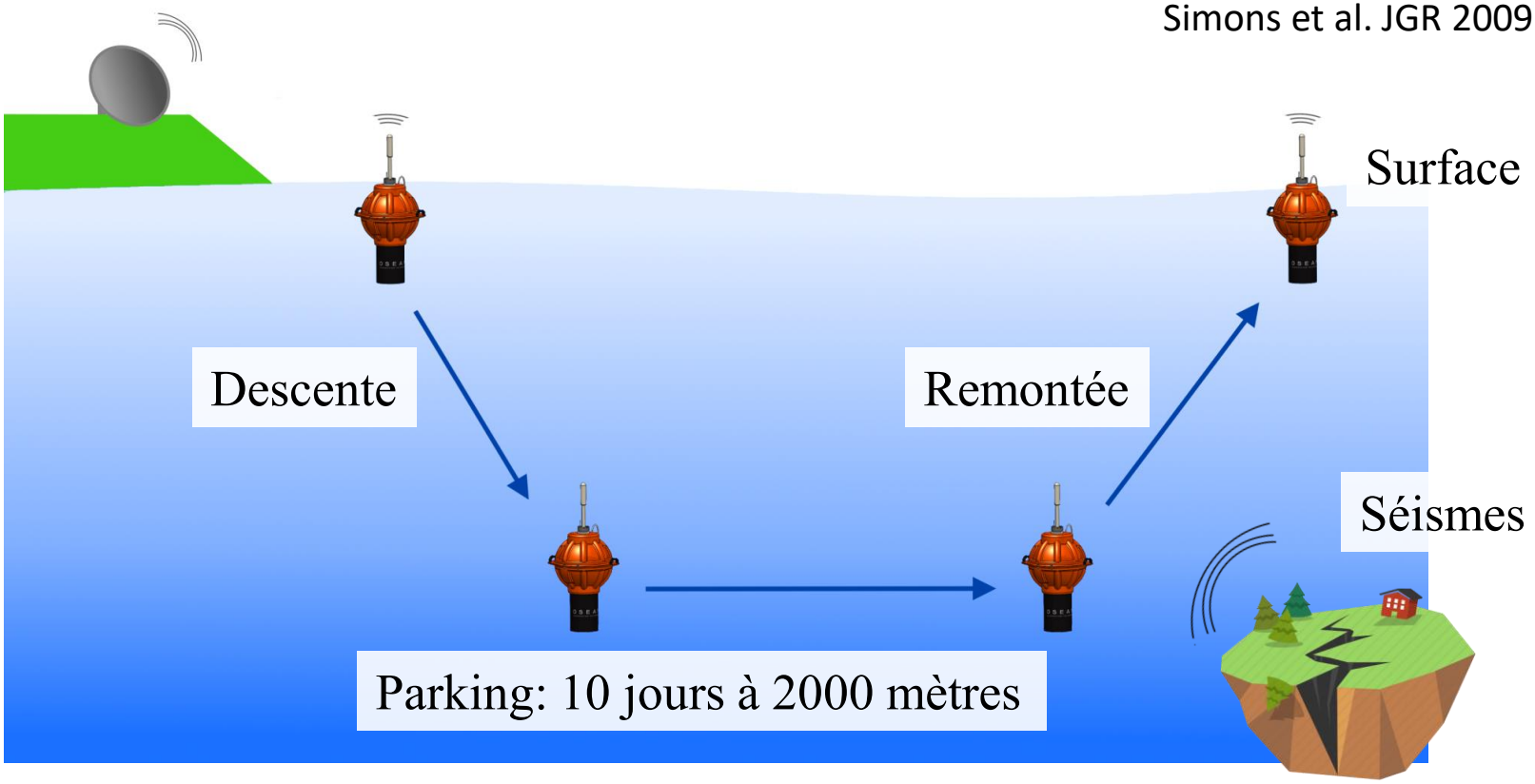
Le réseau ARGO mesure la température et la salinité des océans
Il est constitué d'environ 4000 flotteurs

Les flotteurs profileurs MERMAID

Transmission
de données

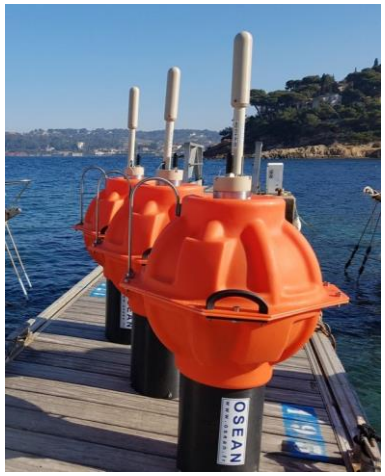


Simons et al. JGR 2009



EarthScope-Oceans

European Research Council

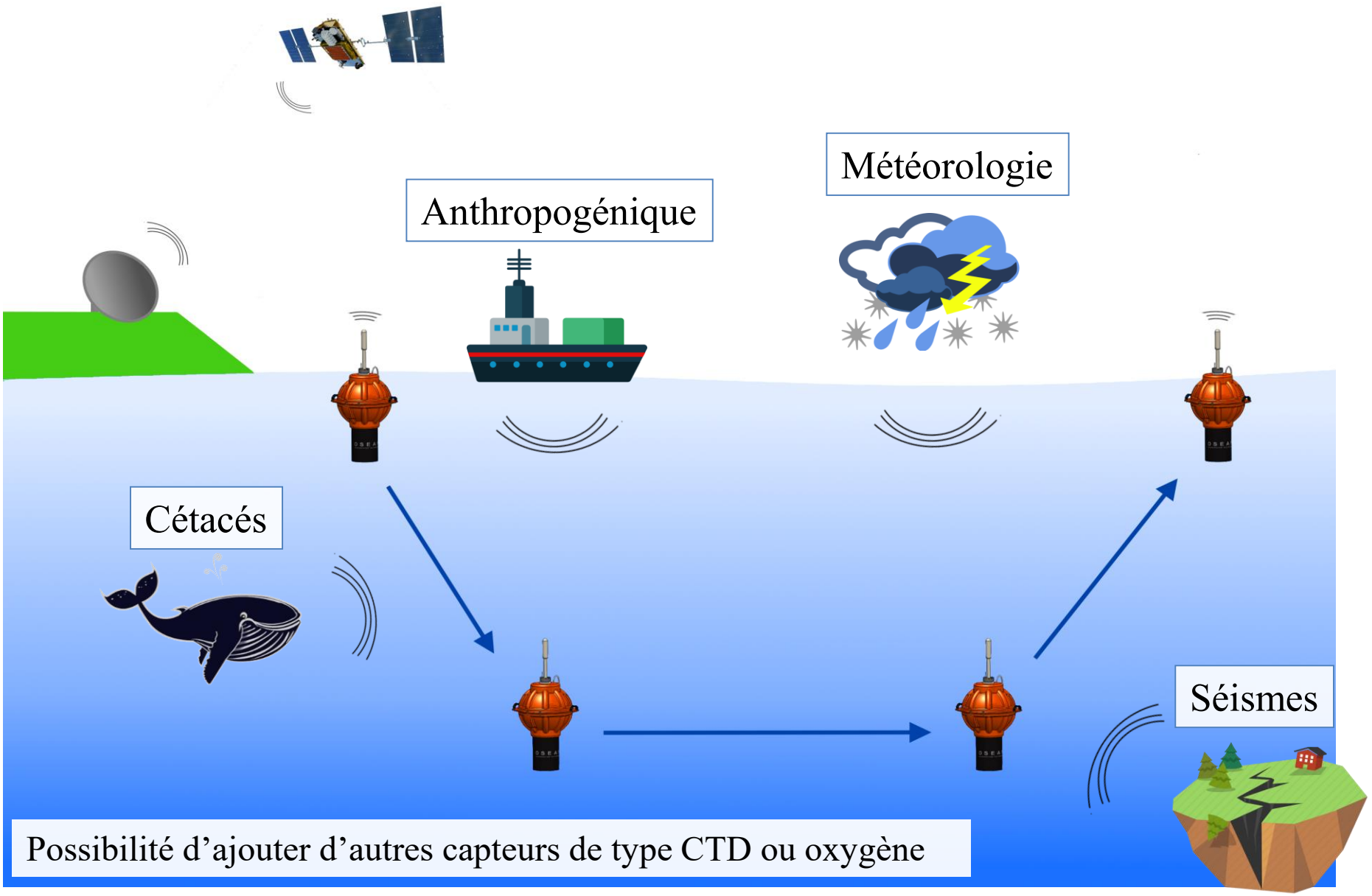


MERMAID développé par
Géoazur et OSEAN

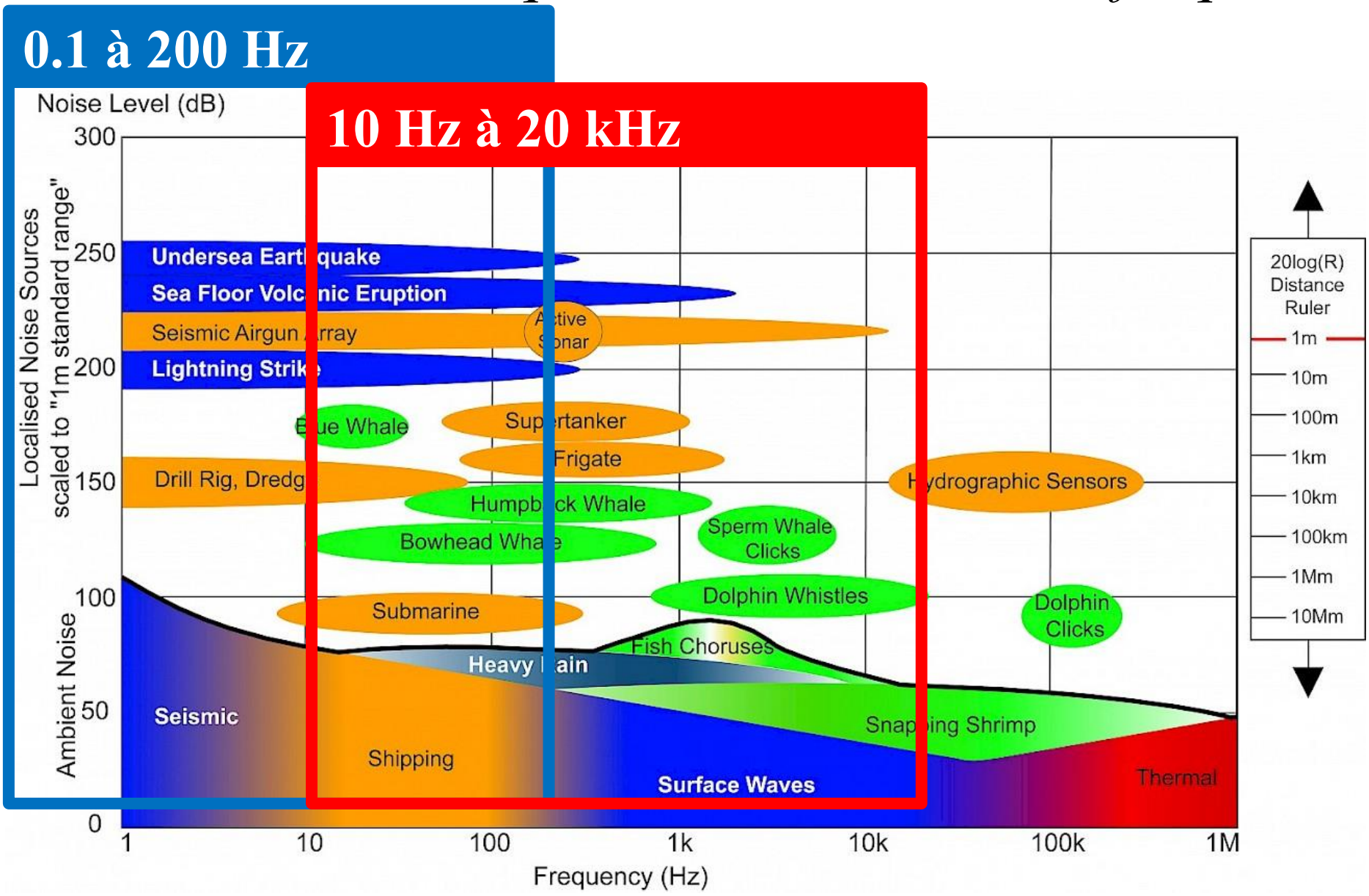
earth scope oceans

63 flotteurs actifs principalement dans l'océan Pacifique

Vers un flotteur acoustique pluridisciplinaire



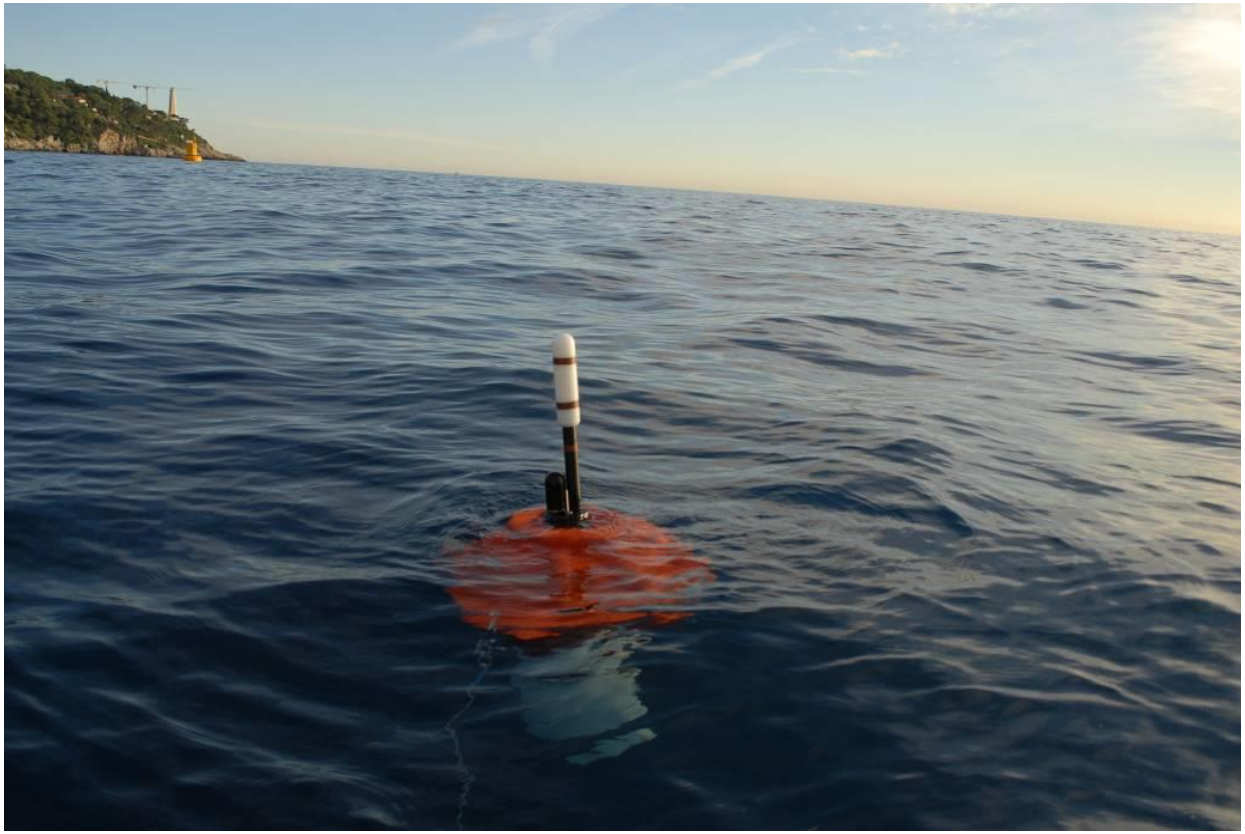
Nouvelle carte d'acquisition basse et haute fréquence



Niveaux et fréquences pour différent types de bruits (OSPAR)

Essais en Méditerranée
fin 2023 - 3 flotteurs

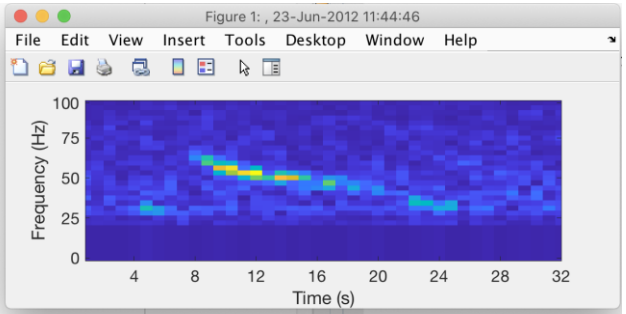
Il faut développer des applications de traitement de données
(les applications pourront être mises à jour par satellite)



Développement des applications

1. Développement d'un algorithme avec un langage type Matlab ou Python

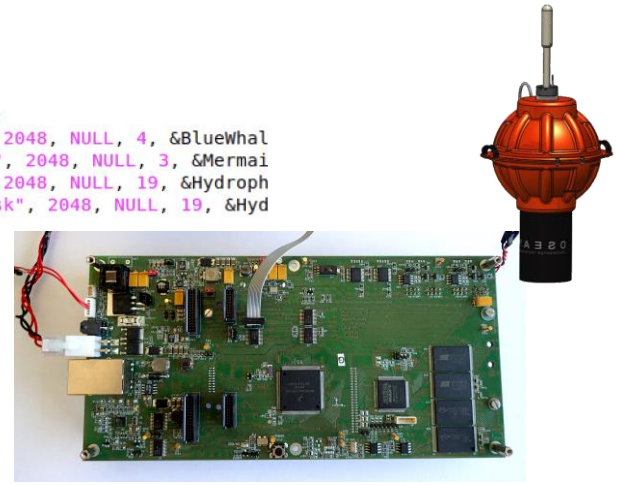
```
fid = fopen(acousticFile);
y = fread(fid, 'int32');
fclose(fid);
for i = plotStartIndex:plotStartIndex+maxPlotNb-1
    type = "";

    staltaoffset = 40*32;
    lim1 = 15*32;
    lim2 = 35*32;

    d1 = (round(seconds(toPlot(i) - startDate)))*200;
    d2 = (round(seconds(toPlot(i) - startDate)))*200;

    yspectro = y(d1-lim1:d2+lim2);
    yfeatures = y(d1-lim1-staltaoffset:d2+lim2);

    [spectrogram] = whalesSpectroCopy(yspectro, 200);
    [allMaxFreq1, allMaxfSNR1, stalta] = computeFeaturesCopy(yfeatures);

    allMaxFreq1 = allMaxFreq1(staltaoffset/32:end);
    allMaxfSNR1 = allMaxfSNR1(staltaoffset/32:end);
    stalta = stalta(staltaoffset/32:end);
```




```
"439 false negatives"

ans =

    "Total "
```

2. Intégration et test dans le flotteur

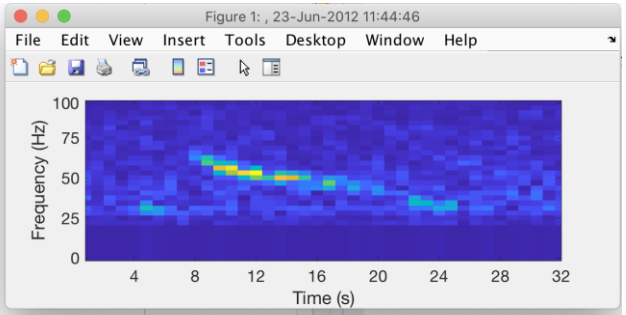
```
void Coordinator_task(void* parametres) {
    init_board(0);
    BlueWhales_semaphore = xSemaphoreCreateMutex();
    MermaidAlgo_semaphore = xSemaphoreCreateMutex();
    xTaskCreate(BlueWhales_task, "BlueWhales_task", 2048, NULL, 4, &BlueWhal
    xTaskCreate(MermaidAlgo_task, "MermaidAlgo_task", 2048, NULL, 3, &Mermai
    xTaskCreate(Hydrophone_task, "Hydrophone_task", 2048, NULL, 19, &Hydroph
    xTaskCreate(HydrophoneHF_task, "HydrophoneHF_task", 2048, NULL, 19, &Hyd
    while(1) {
        _update_mission_sequence();
        _wait_low_power_mode(1000);
    }
}
```



Développement des applications

1. Développement d'un algorithme avec un langage type Matlab ou Python

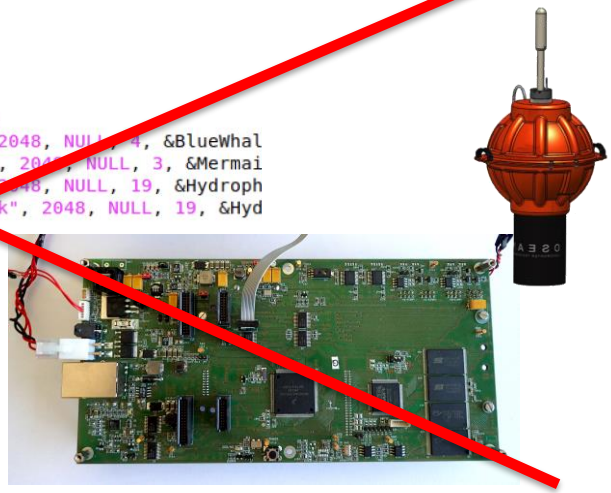
```
fid = fopen(acousticFile);
y = fread(fid, 'int32');
fclose(fid);
for i = plotStartIndex:plotStartIndex+maxPlotNb-1
    type = "";

    staltaoffset = 40*32;
    lim1 = 15*32;
    lim2 = 35*32;

    d1 = (round(seconds(toPlot(i) - startDate)))*200;
    d2 = (round(seconds(toPlot(i) - startDate)))*200;

    yspectro = y(d1-lim1:d2+lim2);
    yfeatures = y(d1-lim1-staltaoffset:d2+lim2);

    [spectrogram] = whalesSpectroCopy(yspectro, 200);
    [allMaxFreq1, allMaxfSNR1, stalta] = computeFeaturesCopy(yfeatures);

    allMaxFreq1 = allMaxFreq1(staltaoffset/32:end);
    allMaxfSNR1 = allMaxfSNR1(staltaoffset/32:end);
    stalta = stalta(staltaoffset/32:end);
```




```
"439 false negatives"

ans =

    "Total "
```

2. Intégration et test dans le flotteur

```
void Coordinator_task(void* parametres) {
    init_board(0);
    BlueWhales_semaphore = xSemaphoreCreateMutex();
    MermaidAlgo_semaphore = xSemaphoreCreateMutex();
    xTaskCreate(BlueWhales_task, "BlueWhales_task", 2048, NULL, 4, &BlueWhal
    xTaskCreate(MermaidAlgo_task, "MermaidAlgo_task", 2048, NULL, 3, &Mermai
    xTaskCreate(Hydrophone_task, "Hydrophone_task", 2048, NULL, 19, &Hydroph
    xTaskCreate(HydrophoneHF_task, "HydrophoneHF_task", 2048, NULL, 19, &Hyd
    while(1) {
        _update_mission_sequence();
        _wait_low_power_mode(1000);
    }
}
```



Un langage de programmation dédié: MeLa

Comment développer des applications sans spécialistes des systèmes embarqués ?

$\Rightarrow$ Langage de programmation MeLa dédié au flotteur MERMAID

Ce langage permet:

1. De cacher les aspects bas niveaux orientés systèmes embarqués
2. Prendre en compte les limites de l'instrument (puissance de calcul, énergie)
3. D'assurer la fiabilité et l'efficacité des applications
4. D'installer plusieurs applications sur un même flotteur

MeLa: calcul de FFT

```
1    Mission:
2       ParkTime: 7 days
3       ParkDepth: 1500 meters
4
5    Coordinator:
6       ParkAcqModes: helloFFT during 10 minutes every 1 hour
7
8    ContinuousAcqMode helloFFT:
9    Input:
10      sensor: HydrophoneHF(20000)
11      data: x(128)
12      overlap: 32
13
14   Variables:
15      Array spectrum(128)
16
17   RealTimeSequence main:
18      spectrum = fft(x, 128, HANNING)
19   endseq;
20   endmode;
```

MeLa: génération de code

MeLa

1 fichier / **20** lignes de code

```
Mission:
   ParkTime: 7 days
   ParkDepth: 1500 meters

Coordinator:
   ParkAcqModes: helloFFT during 10 minutes every 1 hour

ContinuousAcqMode helloFFT:
Input:
   sensor: HydrophoneHF(20000)
   data: x(128)
   overlap: 32

Variables:
   Array spectrum(128)

RealTimeSequence main:
   spectrum = fft(x, 128, HANNING)
endseq;
endmode;
```

Langage C

10 fichiers / **442** lignes de code

```
SRC_ROOT=src

CPPFLAGS+=-I$(SRC_ROOT)

LD_PATH=boards
############################################################
SRC_SRC+=$(SRC_ROOT)/Coordinator_task.c
SRC_SRC+=$(SRC_ROOT)/shared.c

SRC_SRC+=$(SRC_ROOT)/Simple_task.c
SRC_SRC+=$(SRC_ROOT)/HydrophoneBF_task.c


void Coordinator_task(void* parametres) {
    init_board(0);
    Simple_semaphore = xSemaphoreCreateMutex();
    xTaskCreate(Simple_task, "Simple_task", 2048, NULL, 3, &Simple_task_handle);
    xTaskCreate(HydrophoneBF_task, "HydrophoneBF_task", 2048, NULL, 19, &HydrophoneBF
    while(1) {
        _update_mission_sequence();
        _wait_low_power_mode(1000);
    }
}



void HydrophoneBF_task(void * parameters){
    int32_t sample;
    int i = 0;
    xSemaphore = xSemaphoreCreateMutex();
    xSemaphoreTake(Simple_semaphore, portMAX_DELAY);
    vTaskSuspend(NULL);
    while (1) {
        sample = get_HydrophoneBF_sample();
        if (Simple_require_samples) {
            x_Simple_fill->data[i] = sample;
            i ++;
            if (i >= x_Simple_fill->len) {
                if (x_Simple_fill == &x_Simple_buff_1) {
                    x_Simple_process = &x_Simple_buff_1;
                    x_Simple_fill = &x_Simple_buff_2;
                }
                else {
                    x_Simple_process = &x_Simple_buff_2;
                    x_Simple_fill = &x_Simple_buff_1;
                }
                i = 0;
                xSemaphoreGive(Simple_semaphore);
            }
        }
    }
}
```

```
if (mission_step == DESCENT) {
    descent_cycle_time_s = getsdate() - descent_start_time_s;
    if (!Simple_is_running) {
        resume_Simple_task();
    }
}
else if (mission_step == PARK) {
    park_cycle_time_s = getsdate() - park_start_time_s;
    if (!Simple_is_running) {
        resume_Simple_task();
    }
    if (Simple_is_running) {
        suspend_Simple_task();
    }
}
else if (mission_step == ASCENT) {
    ascent_cycle_time_s = getsdate() - ascent_start_time_s;
    if (!Simple_is_running) {
        resume_Simple_task();
    }
    if (Simple_is_running) {
        suspend_Simple_task();
    }
}
else if (mission_step == SURFACE) {
    surface_cycle_time_s = getsdate() - surface_start_time_s;
    if (Simple_is_running) {
        suspend_Simple_task();
    }
    if (buffer_to_flush == true) {
        serialPrintString("Write on files\n");
        buffer_to_flush = false;
    }


void resume_Simple_task() {
    Simple_require_samples = true;
    ask_start_HydrophoneBF();
    vTaskResume(Simple_task_handle);
    Simple_is_running = true;
}


void suspend_Simple_task() {
    Simple_require_samples = false;
    ask_stop_HydrophoneBF();
    vTaskSuspend(Simple_task_handle);
    Simple_is_running = false;
}



static int32_t x_data[10];
static array_i32_t x = {x_data, 10, x_data, 10};
static int32_t y = 0;


void Simple_task(void * parametres){
    vTaskSuspend(NULL);
    while (1){
        xSemaphoreTake(Simple_semaphore, portMAX_DELAY);
        meanArrayI32(&x, &y);
    }
}
```

MeLa: estimation de la consommation d'énergie

Success

Autonomy:

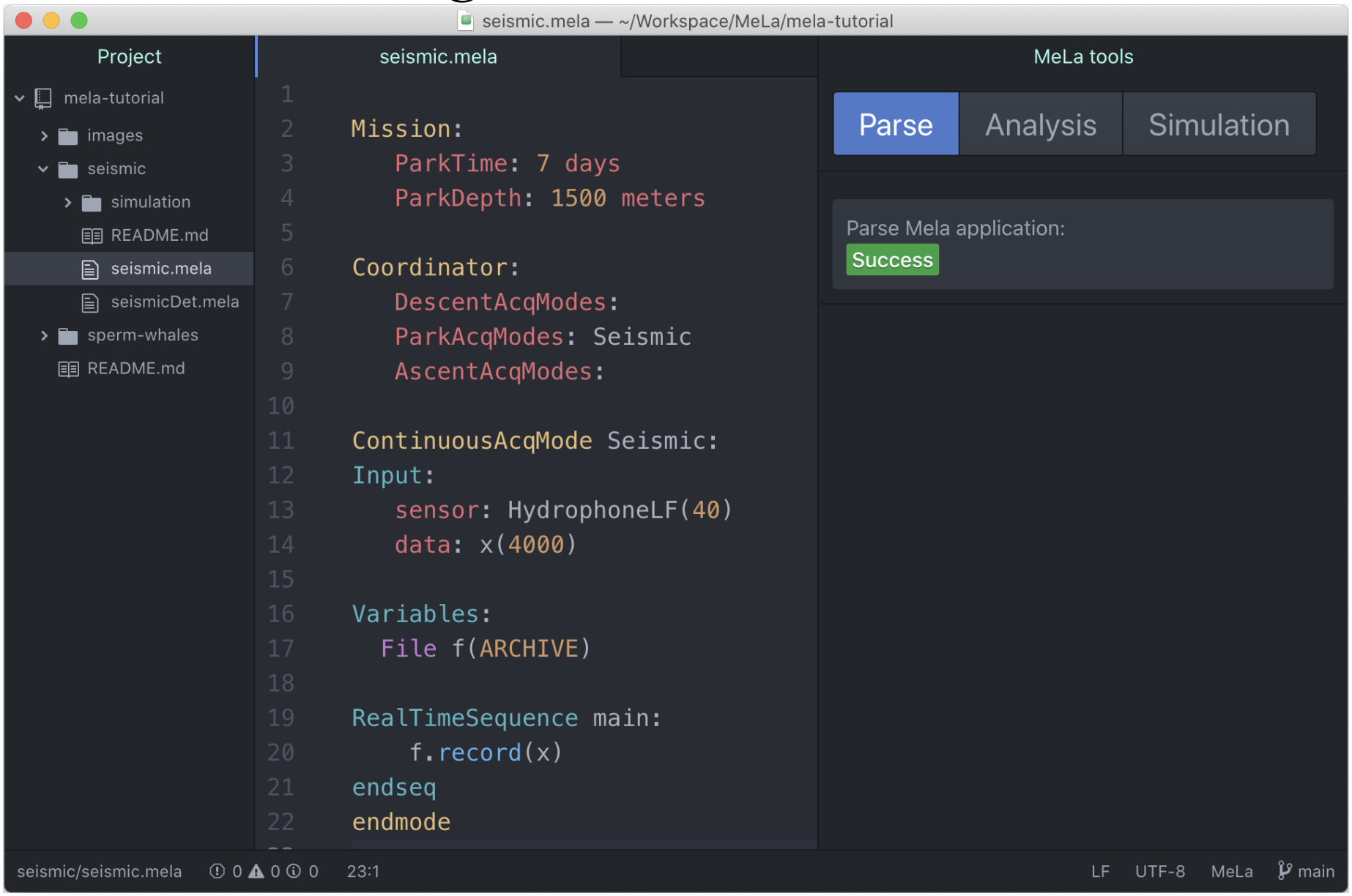
242 cycles

5.2 years

Energy consumption rate:

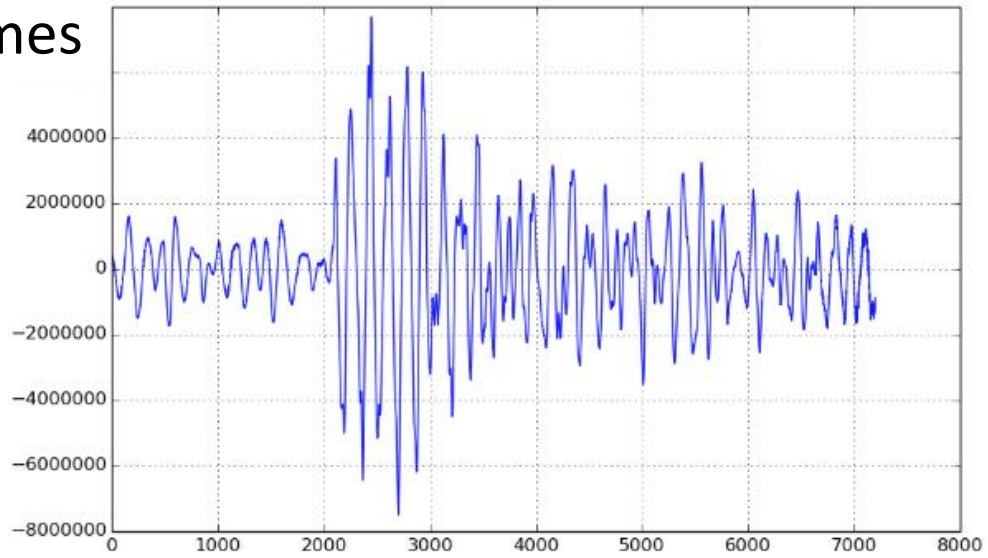
Mission step	Processor (%)	Sensors (%)	Transmission (%)	Actuators (%)
Descent	0.3	0	0	1.1
Park	33.3	37.7	0	0
Ascent	0.1	0	0	25.9
Surface	0	0	1.6	0

MeLa: Intégration dans l'éditeur ATOM

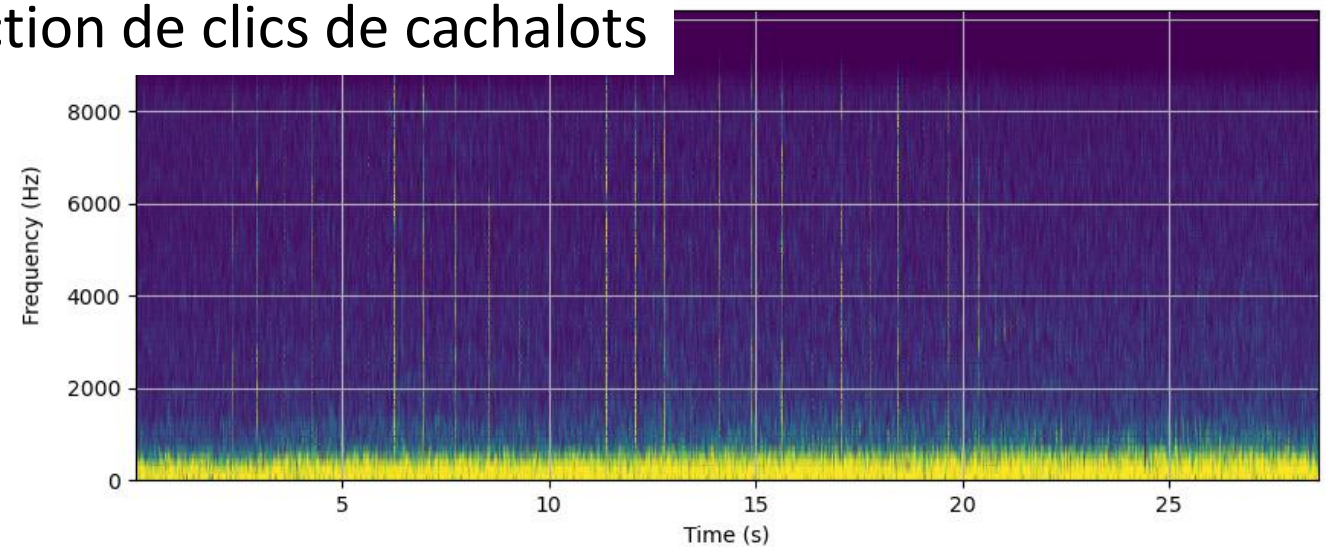


MeLa: tutoriels

Détection de séismes



Détection de clics de cachalots



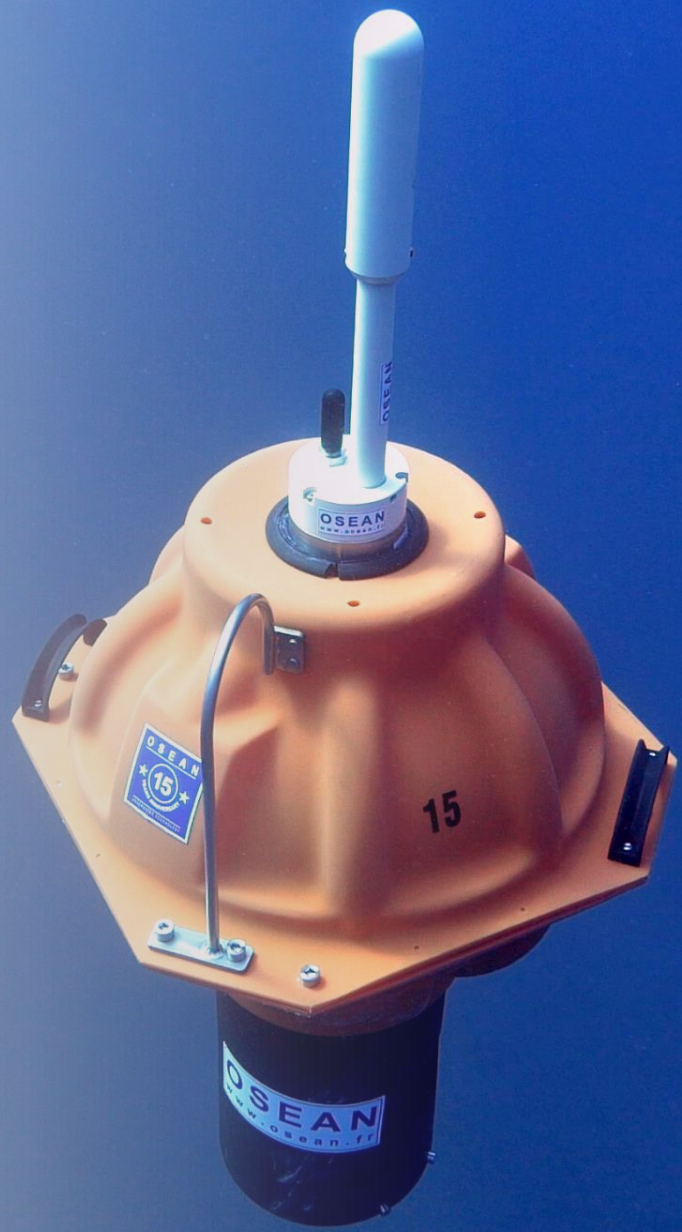
Nous recherchons des spécialistes pour participer au développement d'applications:

- Météo
- Cétacés
- Bruit ambiant

Permettra d'améliorer les fonctionnalités de MeLa

Déploiement de 3 flotteurs en Méditerranée fin 2023 !

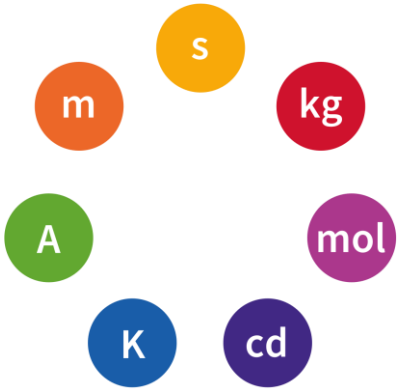
Mail: bonnieux@geoazur.unice.fr



Perspectives

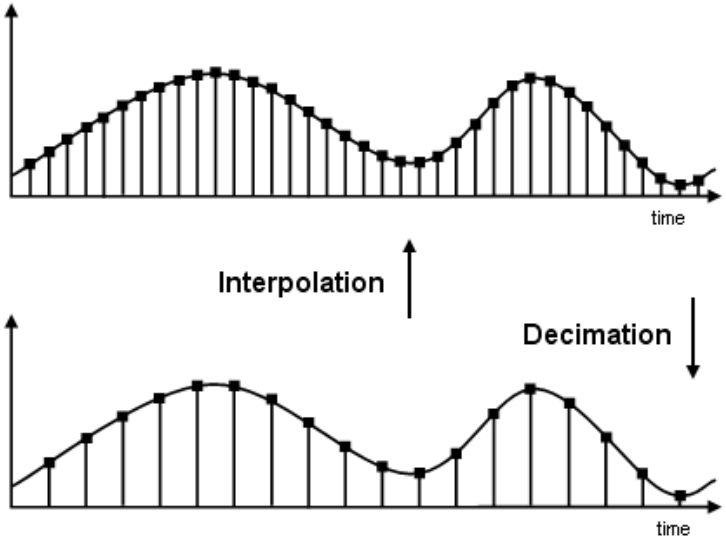
Calcul d'utilisation de mémoire

Unités physiques

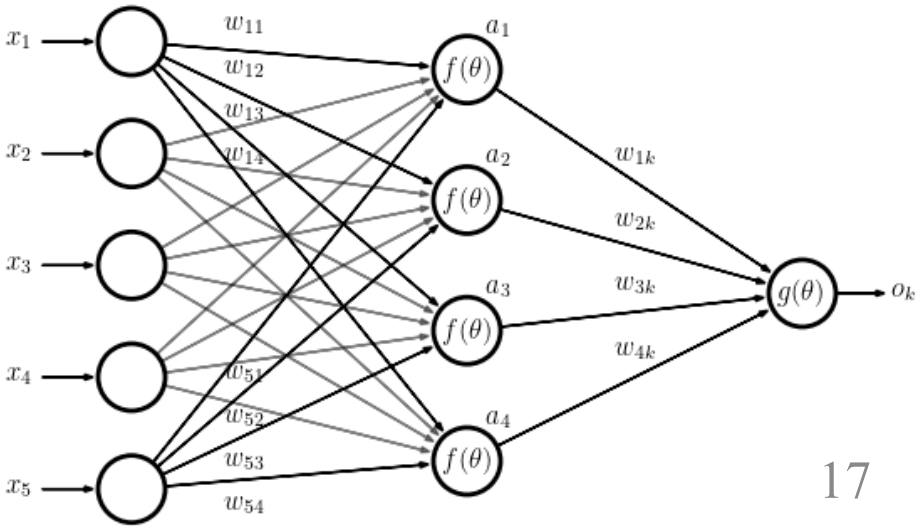


Programmer d'autres d'instruments

Décimation



Apprentissage automatique



Compilateur MeLa