Android's CHRE and Open Source Frameworks

Yuval Peress
peress@google.com

Zephyr Developer Summit, 2022-06-08

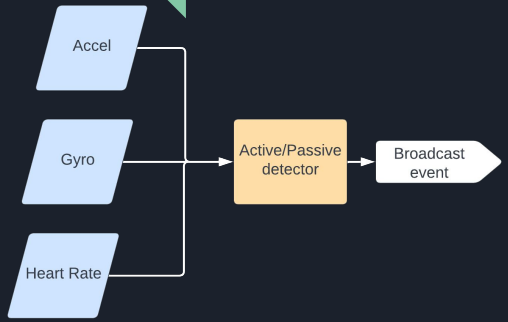

yuval#5515

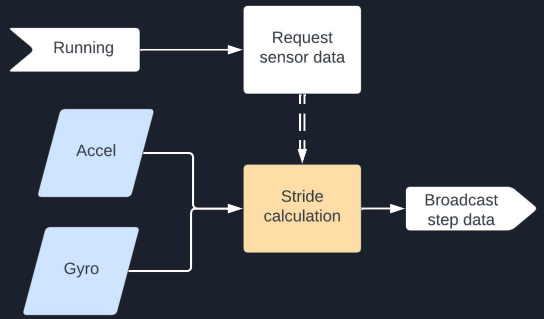

yperess

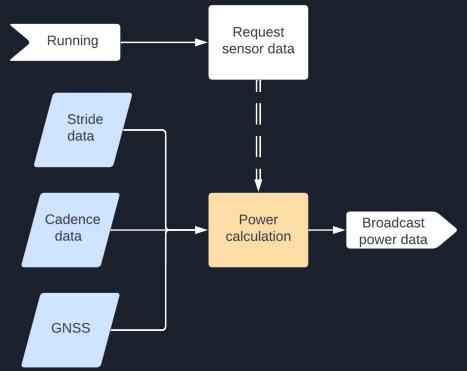
Building a fitness tracker

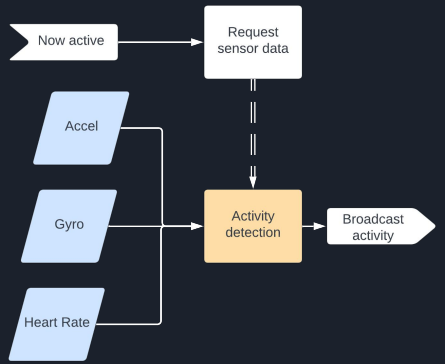
- Relevant sensor data:
  - Accelerometer
  - Gyroscope
  - Heart rate
  - Blood oxygen
- Conserve power by differentiating active and passive
- If active, figure out which activity the user is participating in:
  - Swimming
  - Biking
  - Running
- If known activity, start collecting/calculating specific metrics

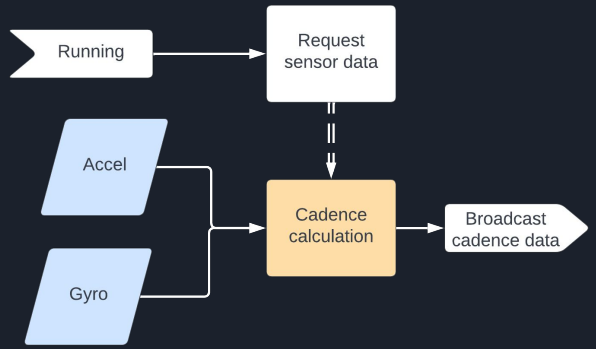
The components...





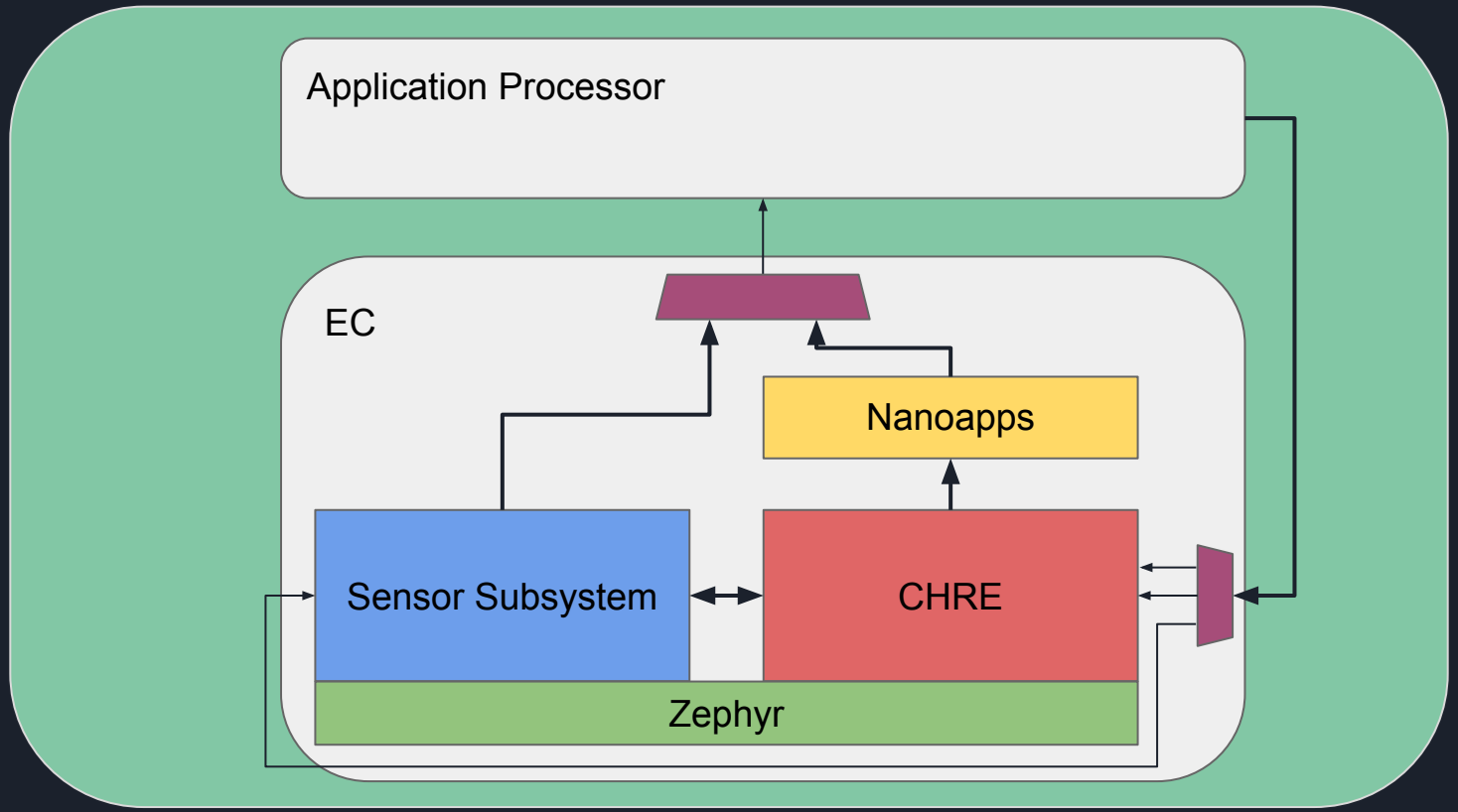






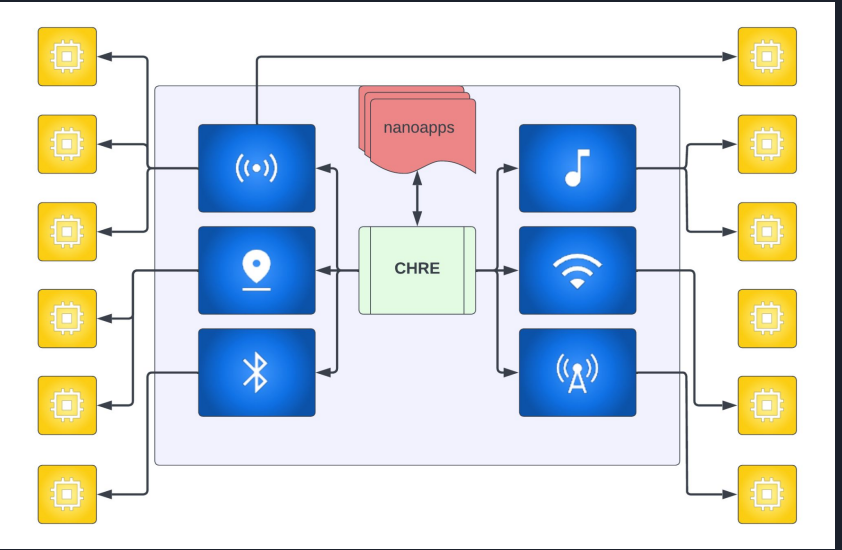
Nanoapps

Top level view



Context Hub Runtime Environment

- Android's event delivery mechanism in the EC
- Runtime environment for nanoapps
  - load/unload
  - start/stop
- Provides APIs for various frameworks: sensors, GNSS, BLE, audio, WIFI, & WWAN



Disclaimer: *all Kconfig, devicetree, and APIs presented here are preliminary and are likely to change.*

Why/what are nanoapps?

- They focus on a feature instead of how data is collected
- They are easy to test
- They're portable

Examples:

- Fall detection
- Car crash detection
- Activity detection
- Glass breaking detection
- Geo fencing
- BT/WIFI scanning

Nanoapps are particularly useful to encapsulate some work.
They consume O(n) data and produce fewer events.

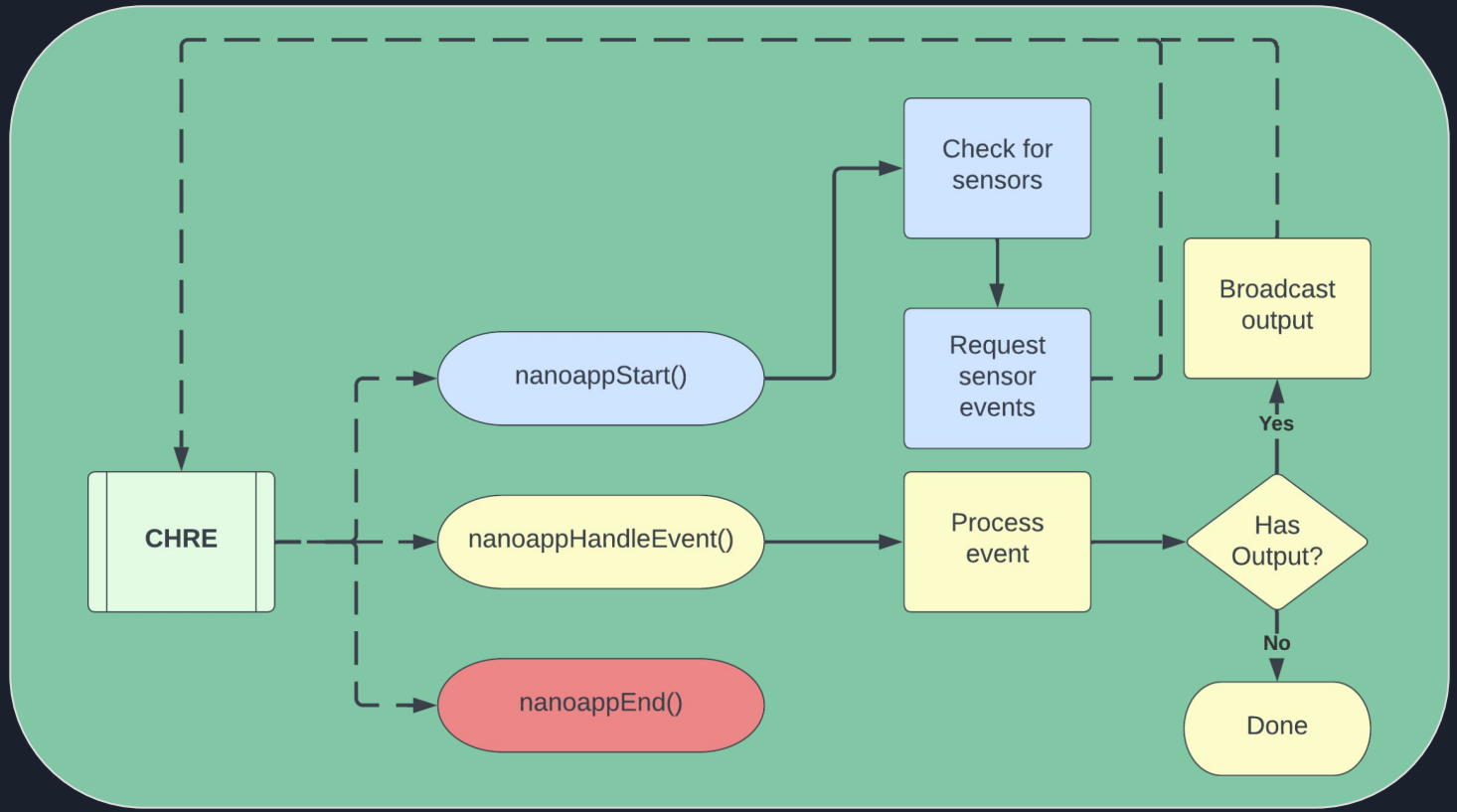
Nanoapps

Each nanoapp is required to implement 3 APIs:

- `bool nanoappStart(void)`
  Called when CHRE starts the nanoapp.
- ```
  void nanoappHandleEvent(uint32_t senderInstanceId,
                          uint16_t eventType,
                          const void *eventData)
  ```
  Called when events are delivered to the nanoapp.
- `void nanoappEnd(void)`
  Called when the nanoapp is stopped by CHRE and will no longer receive events.

Nanoapps are registered with CHRE using the `Nanoapp` class.
  ```

Nanoapp flow



Loading nanoapps

```
namespace {
constexpr const uint64_t kAppId = 1;
constexpr const uint32_t kAppVersion = 1;

chre::Nanoapp *nanoapp = nullptr;

bool nanoappStart(void) {
  printk("EchoApp::nanoappStart()\n");
  nanoapp = chre::EventLoopManagerSingleton::get()->getEventLoop().getCurrentNanoapp();
  nanoapp->registerForBroadcastEvent(CHRE_EVENT_MESSAGE_FROM_HOST);
  return true;
}

void nanoappHandleEvent(uint32_t sender_instance_id, uint16_t event_type, const void *event_data) {
  printk("EchoApp::nanoappHandleEvent(sender_instance_id=%u, event_type=%u, event_data@%p)\n",
         sender_instance_id, event_type, event_data);
}

void nanoappEnd() {
  nanoapp->unregisterForBroadcastEvent(0);
  nanoapp = nullptr;
  printk("EchoApp::nanoappEnd()\n");
}

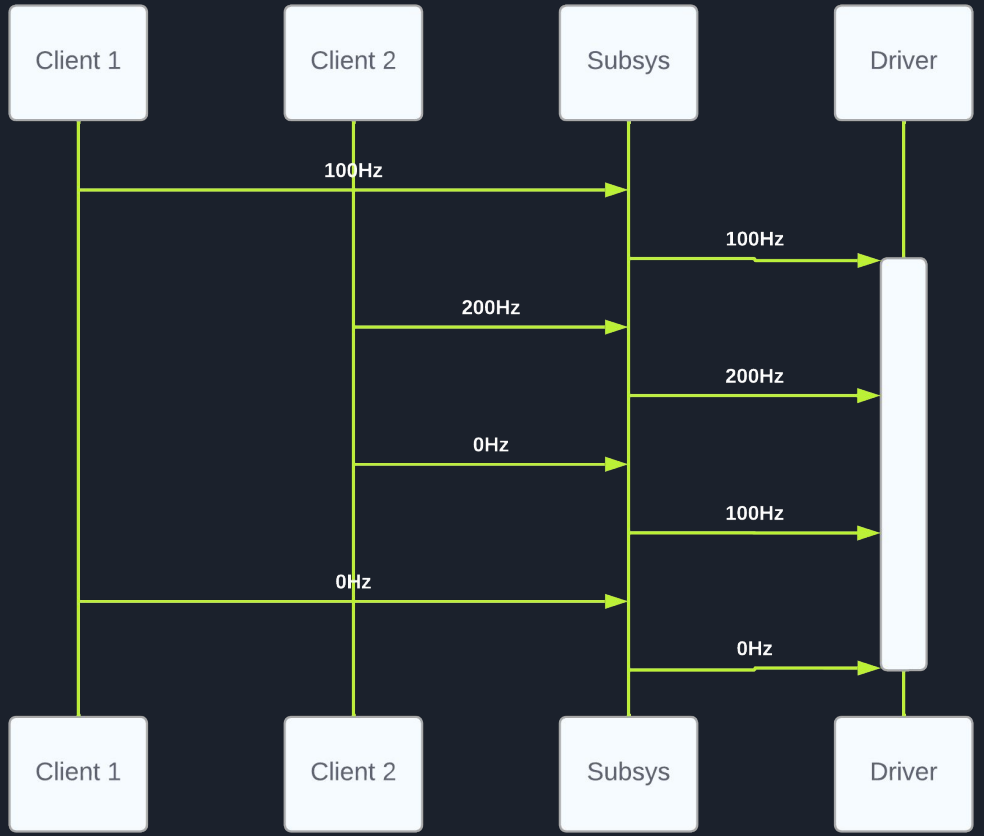
} /* anonymous namespace */

CHRE_STATIC_NANOAPP_INIT(EchoApp, kAppId, kAppVersion, chre::NanoappPermissions::CHRE_PERMS_NONE);
```

Request Arbitration

Support:

- Multiple clients
- Power modes
- Sample rates
- Hardware FIFO management
- Range / resolution



Host communication

Communicating off chip

Nanoapps:

- Post events via `chreSendMessageWithPermissions(...)`
- Use the `HostMessage` struct containing a raw `Buffer<uint8_t>` to send the payload

Application must:

- Set `CONFIG_CHRE_MESSAGE_TO_HOST_MAX_SIZE=<size_in_bytes>`
- Implement:
  - `HostLink::sendMessage()`
  - `HostLink::flushMessagesSentByNanoapp()`

Enter PIGWEED

What is pigweed

- Grab a copy of Pigweed from https://pigweed.googlesource.com/pigweed/pigweed
- Add Pigweed to your modules and set `CONFIG_PIGWEED=y`
- Many Pigweed module already support Zephyr:

```
CONFIG_PIGWEED_LOG
CONFIG_PIGWEED_ASSERT
CONFIG_PIGWEED_SYNC
CONFIG_PIGWEED_SYNC_MUTEX
CONFIG_PIGWEED_SYNC_BINARY_SEMAPHORE
CONFIG_PIGWEED_RPC_NANOPB
CONFIG_PIGWEED_PREPROCESSOR
CONFIG_PIGWEED_CONTAINERS
CONFIG_PIGWEED_POLYFILL
CONFIG_PIGWEED_RESULT
CONFIG_PIGWEED_STATUS
CONFIG_PIGWEED_STREAM
CONFIG_PIGWEED_CHRONO
CONFIG_PIGWEED_INTERRUPT
CONFIG_PIGWEED_CHRONO_SYSTEM_CLOCK
CONFIG_PIGWEED_SYS_IO
CONFIG_PIGWEED_RPC
CONFIG_PIGWEED_HDLC
CONFIG_PIGWEED_SPAN
CONFIG_PIGWEED_FUNCTION
CONFIG_PIGWEED_STRING
CONFIG_PIGWEED_VARINT
CONFIG_PIGWEED_ROUTER
CONFIG_PIGWEED_CHECKSUM
```

Nanoapps additional resources

Example nanoapp: https://github.com/zephyrproject-rtos/zephyr/blob/main/samples/modules/chre/src/echoapp.cpp

CHRE on Zephyr: https://github.com/zephyrproject-rtos/chre

CHRE on AOSP: https://source.android.com/devices/contexthub

CHRE Frameworks / subsystems

The Sensor Subsystem

- Can be used without CHRE
- Easily added with `CONFIG_SENSOR_SUBSYSTEM=y`
- Easily configured with devicetree
- Will provide the following once implemented:
  - Timestamp spreading
  - Virtual sensor support
  - Sensor orientation
  - Hot plugging of sensors (not yet scheduled)

Physical sensors

```
/* Create a sensor subsystem with an accelerometer and gyroscope.
 * Both sensors point to the same physical sensor.
 */
sensor-subsystem {
  compatible = “zephyr,sensor-subsystem”;

  /* Add a rotation matrix to N sensors. */
  rot {
    compatible = “zephyr,sensor-subsystem-rotation-matrix”;
    /* Invert Z axis */
    matrix = <1 0   0
              0 1   0
              0 0 (-1)>;
    apply-to = <&bmi160 ...>
  };

  /* Add a 3D accelerometer to the sensor subsystem. */
  accel {
    compatible = “zephyr,sensor-subsystem-accel3d”;
    accel = <&bmi160>;
  };

  /* Add a 3D gyroscope to the sensor subsystem. */
  gyro {
    compatible = “zephyr,sensor-subsystem-gyro3d”;
    gyro = <&bmi160>;
  };
};
```

Virtual sensors

```
/* Create a sensor subsystem with two step counters (virtual and physical).
 * Both sensors point to the same physical sensor, but the virtual sensor
 * will calculate steps in software.
 */
sensor-subsystem {
  compatible = “zephyr,sensor-subsystem”;

  /* Add a physical step counter to the sensor subsystem. */
  pstep {
    compatible = “zephyr,sensor-subsystem-step”;
    accel = <&bmi160>;
  };

  /* Add a virtual step counter to the sensor subsystem.
   * This assumes we’ve implemented virtual step counters in the sensor subsystem.
   */
  vstep {
    compatible = “zephyr,sensor-subsystem-vstep”;
    accel = <&bmi160>;
  };

  /* Add a virtual angle sensor to the sensor subsystem base on 2 accelerometers.
   * This assumes we’ve implemented virtual angle sensors in the sensor subsystem.
   */
  lid_angle {
    compatible = “zephyr,sensor-subsystem-accel-angle”;
    accels = <&bmi160_lid &bma255_base>;
  };
};
```

Writing custom virtual sensors

Drop detection nanoapp example:

- Single event when the device dropped
- Register for accelerometer data
- Listen for `~0g` lasting `min_drop_duration_ms`
- A sharp spike in `accel_magitude`^{2} (the impact) will trigger the event

```
/* If the sensor is static and doesn't
 * need to be unloaded, it can also
 * have a devicetree representation.
 */
vsens_drop {
  compatible = "my_app,vsens_drop";
  accel = <&bmi160>;
  min_drop_duration_ms = <1500>;
};
```

What sensors to expect?

- Initial push for all the sensors listed in CHRE's `sensor_types.h` (`CHRE_SENSOR_TYPE_*`)

```
CHRE_SENSOR_TYPE_ACCEROMETER
CHRE_SENSOR_TYPE_ACCELETOMETER_TEMPERATURE
CHRE_SENSOR_TYPE_GEOMAGNETIC_FIELD
CHRE_SENSOR_TYPE_GEOMAGNETIC_FIELD_TEPERATURE
CHRE_SENSOR_TYPE_GYROSCOPE
CHRE_SENSOR_TYPE_GYROSCOPE_TEMPERATURE
CHRE_SENSOR_TYPE_HINGE_ANGLE
CHRE_SENSOR_TYPE_INSTANT_MOTION_DETECT
CHRE_SENSOR_TYPE_LIGHT
CHRE_SENSOR_TYPE_PRESSURE
CHRE_SENSOR_TYPE_PROXIMITY
CHRE_SENSOR_TYPE_STATIONARY_DETECT
CHRE_SENSOR_TYPE_STEP_COUNTER
CHRE_SENSOR_TYPE_STEP_DETECT
CHRE_SENSOR_TYPE_UNCALIBRATED_ACCELEROMETER
CHRE_SENSOR_TYPE_UNCALIBRATED_GEOMAGNETIC_FIELD
CHRE_SENSOR_TYPE_UNCALIBRATED_GYROSCOPE
```

- Quickly followed by HIDs sensor types
- Additional sensor types may be added to Zephyr or implemented in the applications directly

Using the sensor subsystem

```
sensor_subsystem_open(callbacks) //< Start the subsystem and set the
                                 //  callbacks
sensor_subsystem_get_sensors(&list, &size) //< Get the sensor list
sensor_subsystem_configure_sensor(
    sensor_handle, mode, interval, latency);

...

/* Will handle any events generated by the sensor subsystem. The data
 * pointer is owned by the sensor subsystem and should be released
 * when done by calling sensor_subsystem_release_data_event().
 */
static void my_app_data_event_callback(
    uint32_t sensor_info_index, void *data) { ... }
```

Other Frameworks

- CHRE additionally supports:
  - ***audio*** - hotword detection, alarm detection, custom gain, etc
  - ***BLE*** - background scanning & advertising
  - ***GNSS*** - geofencing, activity tracking, etc
  - ***WiFi*** - background scanning, range, NAN, etc
  - ***WWAN*** - get current cell tower information
- These will be implemented as needed

Questions?