# Computer task delegation

* OBD computer
  + Contacts car OBD-II port and keeps a live database of all relevant parameters. Only frequent parameters are requested repeatedly, more specific parameters can be retrieved upon request by other devices.
  + May have other sensors directly attached to it such that specific diagnostic data can be collected even if the car is unable to do so
* Head unit display computer
  + Handles navigation, HUD queries here for navigation details
    - Custom navigation code that pulls directions from Google Maps
    - Integrate Waze
    - Create personal database of speed and red-light cameras
    - Pursuit features
      * Display which streets are a dead end
      * Help choose streets based on traffic conditions and ‘openness’
        + Openness refers to how many downstream route options are available. This should be analyzed by how many dead ends exist along the route up to a certain amount of levels where one level is one intersection ‘deep’ down the chosen road.
      * Ideally sniff out Ford Interceptors using cellular signature if possible
      * Countermeasures on high alert
      * Light kill switch
      * Night vision? Active if not passive
  + Handles media
    - Soundcloud API integration
    - YouTube music integration
    - Bluetooth integration difficult, maybe use external Bluetooth audio receiver
  + Acts as display to all other forms of data
    - Queries network devices to update local information as necessary
* HUD computer
  + Displays navigation, speed, and countermeasure data
  + Also create range display since vehicle monitors fuel consumption
  + Pursuit mode adds other pursuit-relevant details
* Digital gauge cluster computer
  + Later addition most likely
  + Listen to CAN bus and mimic original dispay data and add things
    - Add range
    - Add odometer
    - Add fuel efficiency
    - Add temperature inside and outside
    - Add warnings from countermeasures and diagnostics etc.
* Diagnostic display computer
  + Queries OBD computer and Arduino network to provide warnings about vehicle health
  + Replaces traction control and sport mode buttons
  + Replaces heated seats buttons
* Arduino manager
  + Uses an ethernet shield to generate a server that hosts data from all arduinos in the vehicle
  + Vehicle has many arduinos throughout on a serial network
  + Each Arduino has a specific purpose and limited I/O for that purpose
    - Arduino list (6 MEGA\_EMBED, 4 NANO, 1 SPECIAL, 1 FULL\_MEGA):
      * Rear light manager (MEGA\_EMBED)
        + Police light animations
        + Hazard animations
        + Manage LEDs in fourth brake light and original break light
        + Manage relays for actual brake lighs
      * Head light manager (MEGA\_EMBED)
        + Same as taillight manager but for front lights
      * Siren manager (NANO?)
        + Not sure if Arduino is best for this
        + Manages audio data being sent to bumper siren speaker
      * Mirror managers (x2) (NANO)
        + Keeps track of mirror positions
        + Tilts mirrors when car backs up
      * HVAC manager (MEGA\_EMBED)
        + Taps into vehicle HVAC system
        + Read and modify temperatre/airflow settings
      * Diagnostic display button manager (NANO)
        + Handles relay for replacement buttons
        + Maybe tap into heated seat LEDs to get some insight if doable
      * Steering wheel button manager (https://www.aliexpress.com/item/32840365436.html)
        + Mostly translates steering wheel buttons to keyboard strokes for head unit computer but also provides input for light pattern and siren sounds
      * Power managers (x2) (MEGA\_EMBED)
        + One in front and one in back
        + Handle power delivery to aftermarket and stock components
        + Monitor current draw and various bus voltages
        + Handles active power supply failover
      * Audio manager (MEGA\_EMBED)
        + Handles audio amplifier
        + Switches over from Bluetooth receiver to head unit
        + Manages Bluetooth receiver
      * CONTROLLER (MEGA w/ ethernet shield)
        + Arduino Mega 🡪 3 serial buses
        + Bus 1 – power managers (consistent talkative devices)
        + Bus 2 – steering wheel controls (inconsistent talkative device)
        + Bus 3 – everything else (slave/executor devices)

# Head unit software (Android – Java)

## General theory:

Front end service (FullscreenActivity) provides the general layout which contains a couple fragment containers. Each service has a separate fragment class which is displayed in the fragment containers. This is done because the service must run continuously regardless of whether the fragment is displayed to the user. The display will be set up with a wider section to the left and a slimmer section to the right. The left section will be used for significant tasks such as navigation, defense, and safety related tasks while the smaller fragment will be used for media and defense when the main screen is preoccupied with a non-defense related task. Each fragment can be swapped using steering wheel buttons as described in the UI section.

## UI:

While touch screen functionality will be available, most of the interaction should be done through steering wheel buttons for safety reasons. The steering wheel has the following buttons:

* Arrow keys with enter/back buttons
* Voice command button
* Phone call/hang up
* Source button, media buttons, volume buttons
  + **IMPORTANT:** steering wheel source button will toggle an external relay that switches between the audio input of the separate Bluetooth receiver and the car’s head unit. Furthermore, the media buttons mentioned alongside source button will simulate button presses on the Bluetooth receiver to modify phone’s media. Arduino will have to be able to only do this when the Bluetooth receiver is being used.
  + **IMPORTANT**: second Arduino required to manage audio considering complexity of I/O. This has been added to the Arduino list below. This Arduino will control the amplifier which is separate from the car’s head unit.

Fragments in display can be swapped by pressing the back key on the steering wheel:

* Doing so highlights both fragments after which arrow keys can be used to swap fragments
* Up/down keys swap fragments, left/right keys switch which of the two fragment is selected
* Pressing enter enters the fragment after which the steering wheel buttons control the selected fragment

Each fragment has the following controls:

* Media fragment
  + Media buttons act as media buttons no matter what fragment is selected
  + Arrow keys will be used to navigate playlists/services
* Navigation fragment
  + Try sharing directions through NFC
  + Maybe use web view fragment to display directions?
  + Really lost here

## Classes:

* FullscreenActivity (Front-end manager)
  + Contains all functions to display things
    - Other classes solely provide data or video sources etc
    - Fragments may be contained in other classes however shared display functions such as alerts will be hardcoded here and referenced from external services
* DiagnosticService
  + Talks to OBD and Arduino computers
  + Keeps database of relevant diagnostic data
  + Can execute custom diagnostic requests on demand
  + Observes critical parameters and generates warnings
* MediaService
  + Retrieves media from soundcloud or youtube
  + Receives Bluetooth audio if integrated
    - Maybe figure out what song is played over Bluetooth and show album cover/timeline
    - Not a necessary feature but would be cool
* DefenceService
  + Manages countermeasures
  + Can be deactivated with ‘vanilla’ mode
    - All references to this service should be eradicated upon ‘vanilla’ mode trigger
* SafetyService
  + Manages warnings about impending collisions
  + Obtains footage of blindspot cameras etc.
  + Probably won’t do any CV since those tasks should be done by an nVidia Jetson which won’t be installed for another year at the very least
* NavigationService
  + Interacts with Google Maps API
* NetworkService
  + Handles all network requests
  + https://stackoverflow.com/questions/33229869/get-json-data-from-url-using-android

# Arduino computer software

## Packet structure

* Structure for packets from talkative devices to master:
  + Sender ID (single alphanumerical character should suffice)
  + Packet urgency (possible states: standard and error)
    - This is used in case of power supply failover for example
    - Forces master to listen and forward broadcast on network
    - Standard packets update non-critical information
  + Packet data
* Structure for packets from master to devices
  + Target ID (alphanumerical character describing what computer is addressed)
  + Request type (possible states: execute an action or request data)
  + Request data (only necessary if asking a device to perform an action)
    - Could be used for requesting oddly specific data that may be omitted from regular broadcasts