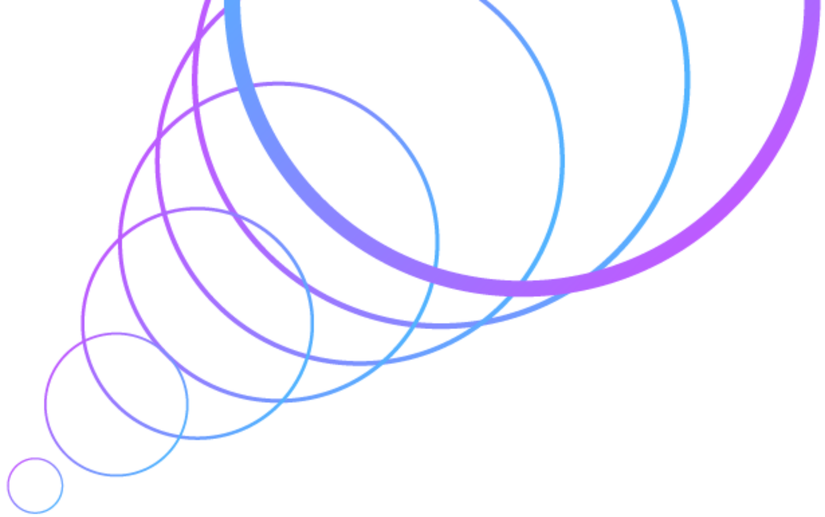


웨일의 차량용 S/W 플랫폼 도전기

웨일 Auto PoC

안영기 NAVER / Whale Platform

최현덕 드림에이스

PROJECT MEMBERS

안영기

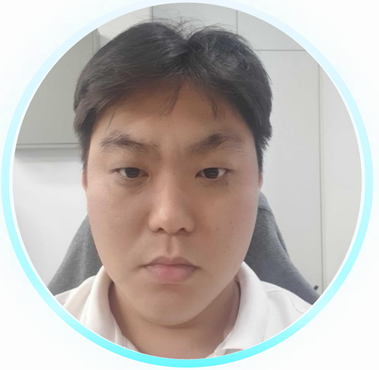
WHALE-Engineer

이원규

WHALE-Engineer

김동훈

WHALE-Engineer



최현덕

DRIMAES-Engineer

홍예브게니

DRIMAES-Engineer

whale X DRIMAES

CONTENTS

1. 웨일 Auto 프로젝트의 시작

1.1 차량의 발전과 차량용 SW의 미래

차량용 OS 패러다임의 변화

가벼운
기존의 OS

기기 별 별도 동작, RTOS 등
가벼운 형태를 띄던 기존 OS체제

>>>

다양한
니즈의 등장

자율주행, AI, LiDAR,
Advanced Infotainment System

>>>

HW /SW
변화의 시작

이를 모두 만족하는
고성능의 HW, 다양한 SW의 필요

갈수록 OS의 크기는 커지고 통합형으로 진화

1.1 차량의 발전과 차량용 SW의 미래

꾸준히 증가하는 Linux 체제의 최신 차량 OS

QNX

AAOS (Android Automotive OS)

AGL

CCOS

SUSE

Web OS Auto

MBOS

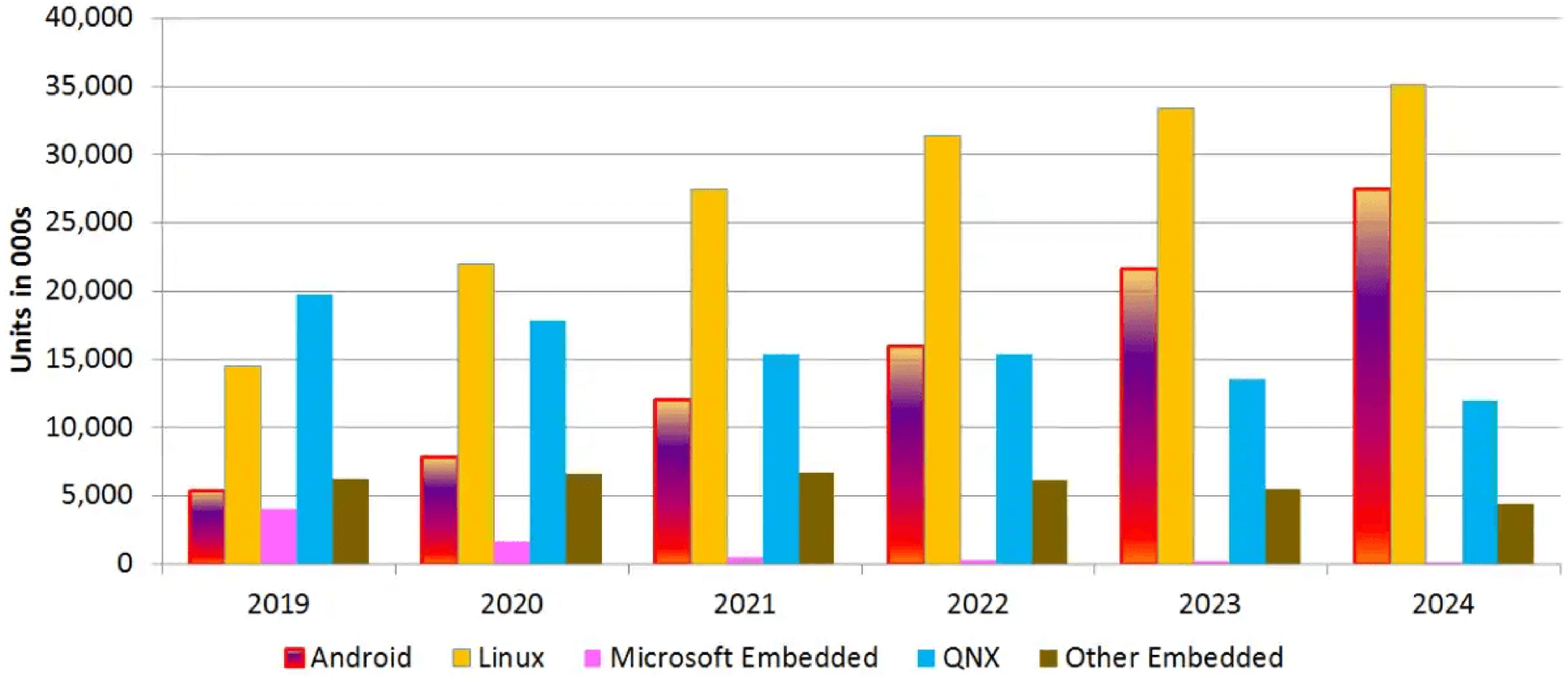
Tesla OS

ETC

1.1 차량의 발전과 차량용 SW의 미래

DEVIEW 2021

Automotive Infotainment OS Market Share



https://www.emizentech.com/blog/android-automotive-in-cabin-experience-in-connected-vehicles.html

1.2 네이버 X 드림에이스

DEVIEW 2021

웨일의 도메인 확장을 꿈꾸는 Whale 팀

브라우저 기술을 바탕으로 더 확장할 수 있는 사업영역이 없을까?

차량용 S/W 플랫폼에서 우리의 강점을 살릴 수 있을지도…

whale

모빌리티 플랫폼의 강자를 꿈꾸는 드림에이스

DRIMAES

차량용 리눅스 (AGL)의 가치와 기술력을 널리 알리고 싶고

컨테이너를 이용한 가상화 기술을 차량에 적용해 봤으면…

1.2 네이버 X 드림에이스

웨일 Auto PoC

NAVER

웨일 Auto 개발

웨일 Auto ↔ AGL 인터페이스 개발

자동차용 웹앱 개발

앱스토어 개발

DRIMAES

SA8155P(보드) 셋업

웨일 브라우저 및 웨일 Auto 포팅

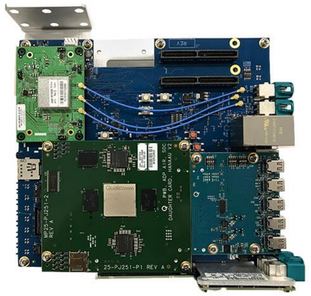
백엔드(AGL) 서비스 개발

좌석별 가상화

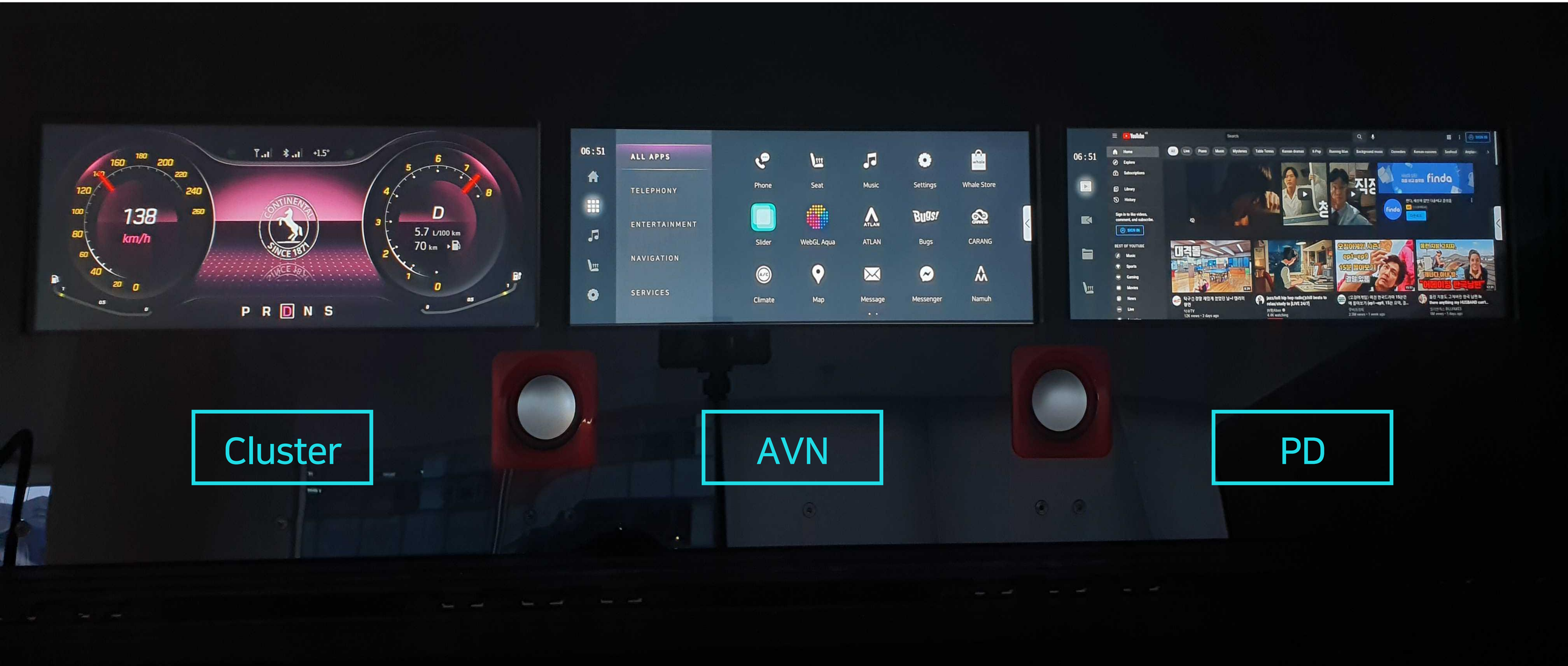
1.3 네이버 X 드림에이스 - 개발 환경

DEVIEW 2021

웨일	Whale Browser M83 코드 기반
AGL	8.0(halibut) + 드림에이스 커스터마이징
타겟 보드	SA8155P (Qualcomm ADP)
커널 버전	4.14
Yocto	2.6 (thud)



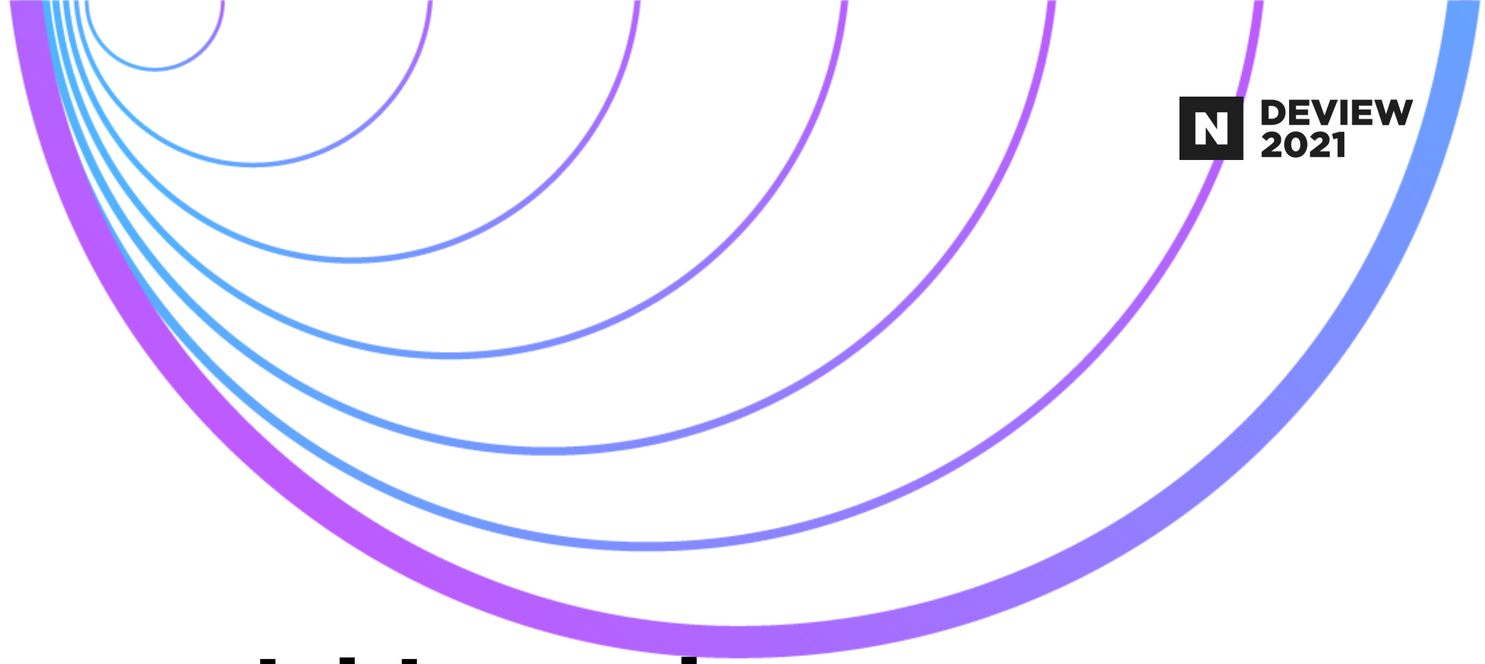
1.4 네이버 X 드림에이스 - 개발 결과



Cluster

AVN

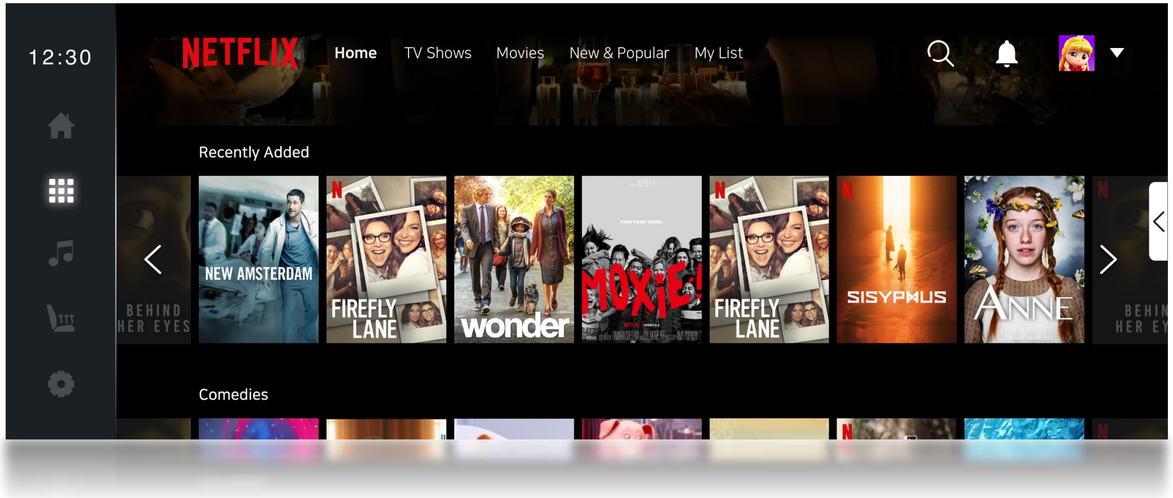
PD

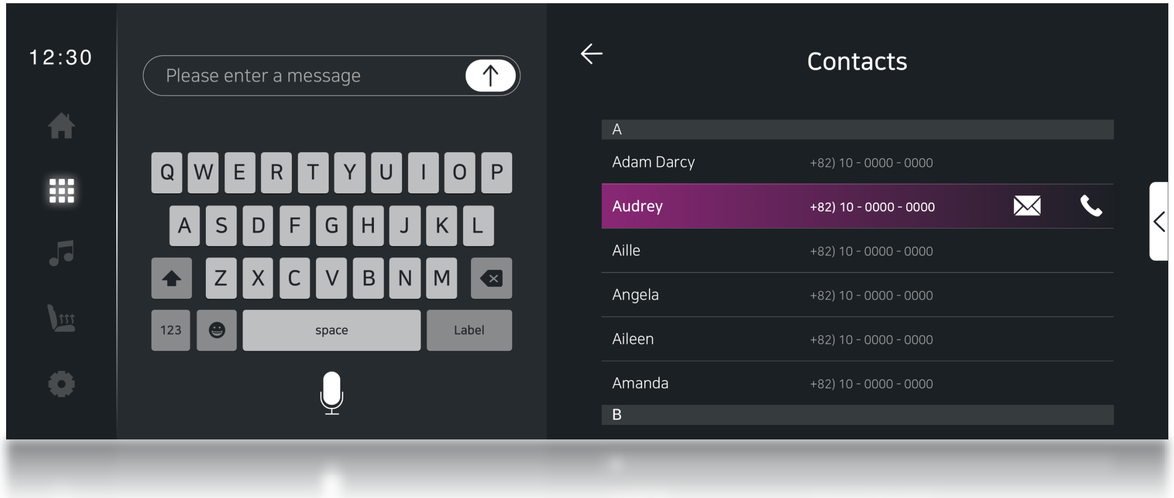


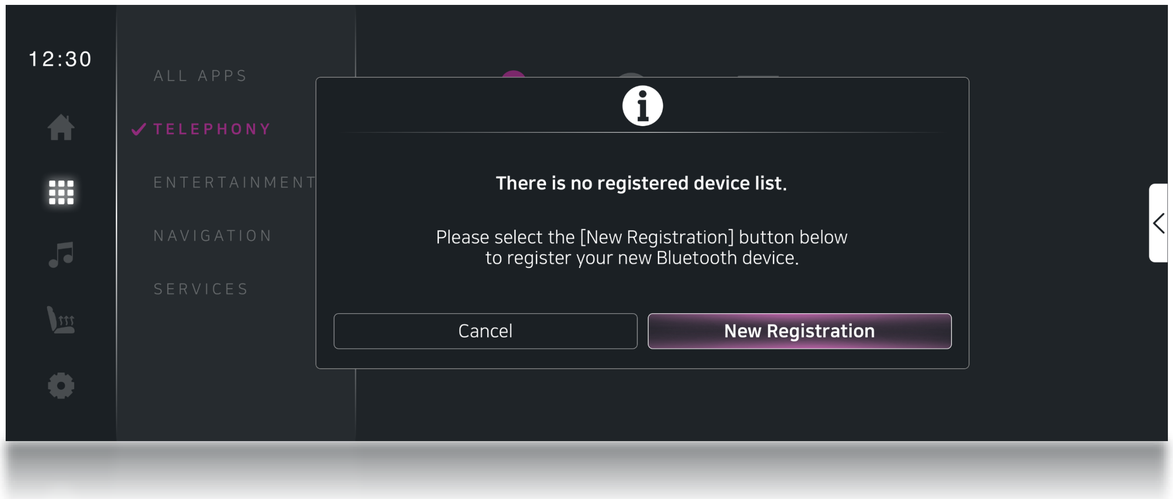
2. 요구사항 도출

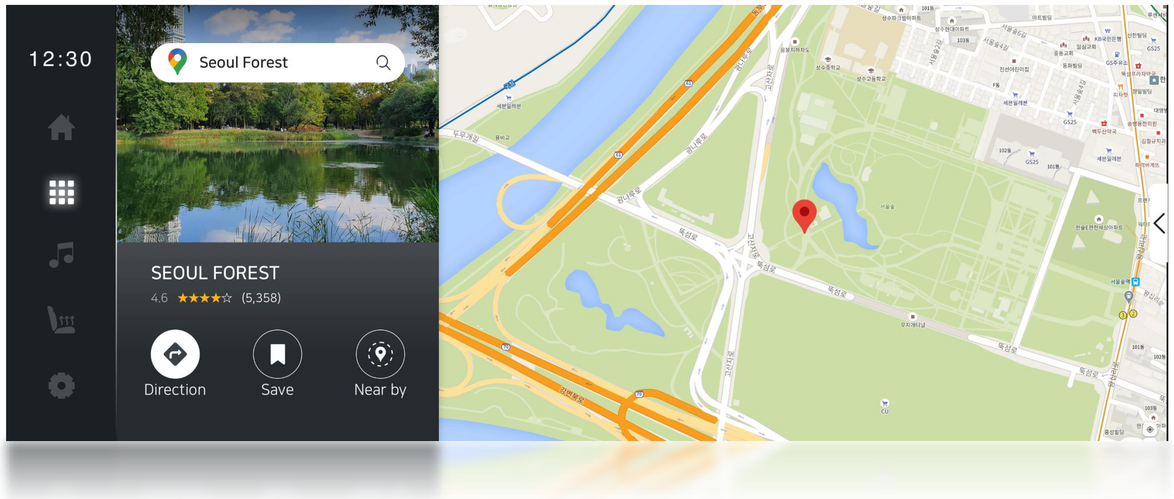
2.1 우리가 만들어야 하는 것

2.1 우리가 만들어야 하는 것

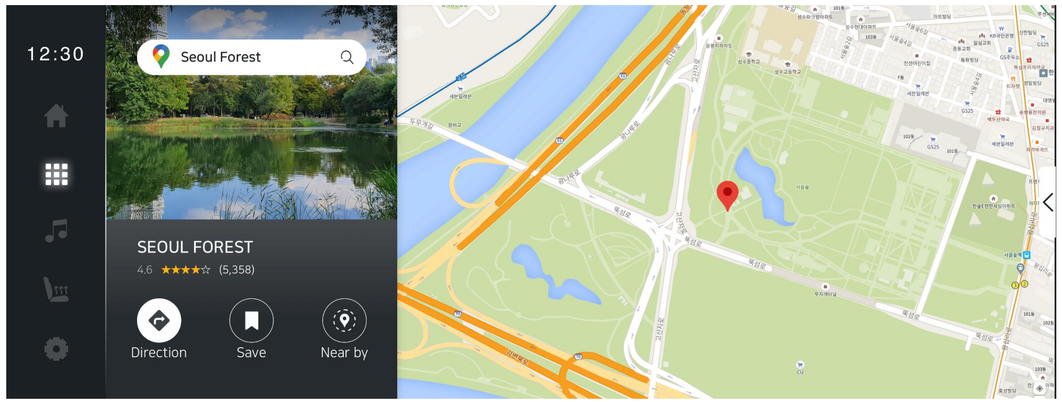






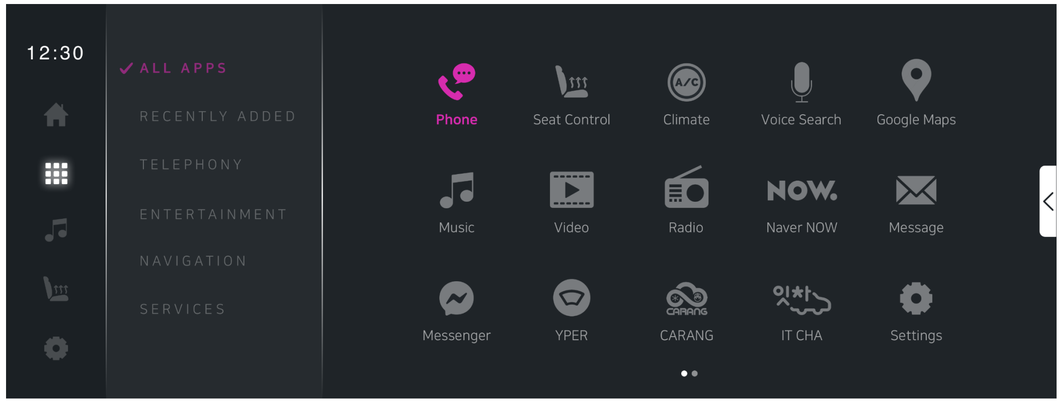


2.1 우리가 만들어야 하는 것



윈도우 시스템

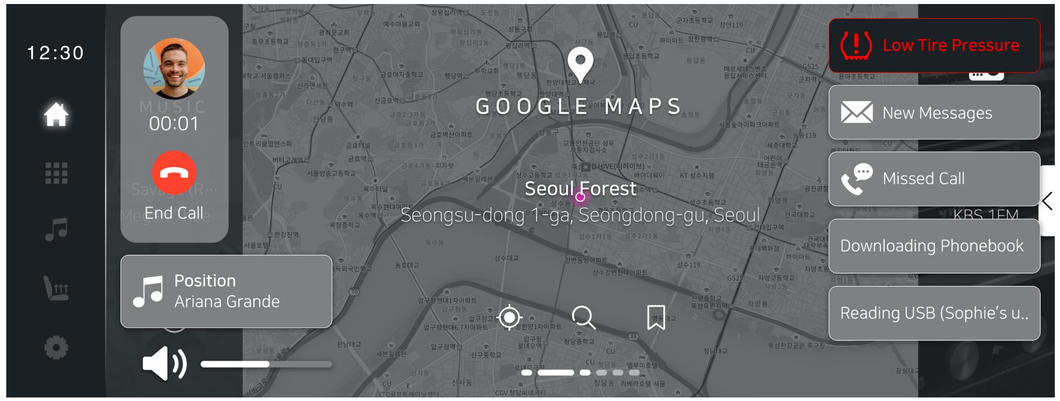
- 다수의 스크린을 동시에 지원
- 스크린당 N개의 윈도우를 지원

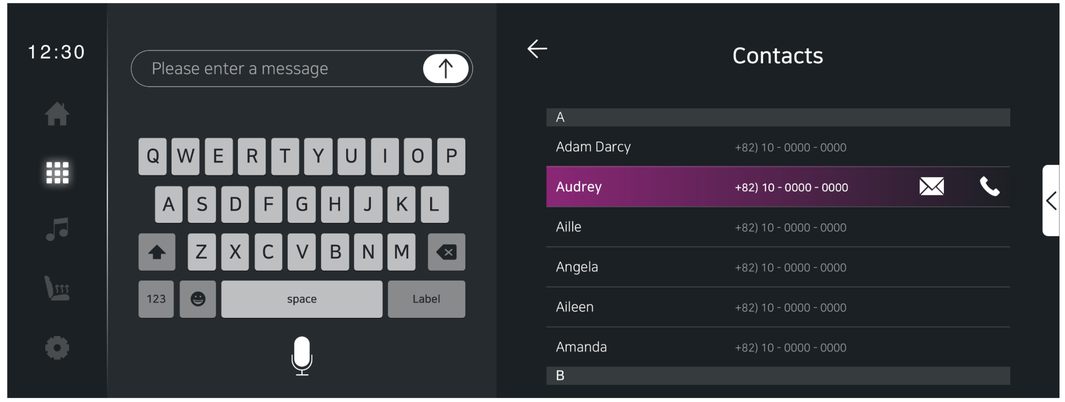


애플리케이션 프레임워크

- 앱을 설치하고 삭제하고 실행
- 앱을 다운로드하기 위한 에코 시스템

2.1 우리가 만들어야 하는 것





알림 시스템

- 시스템의 경고에 대응
- 알림 메시지에 대한 관리
- 윈도우의 우선 순위에 따른 레이어링

그 외,

- 입력기 지원
- 휴대폰과 같은 디바이스 동기화
- 그 이외에 윈도우즈에서 보던 많은 것들

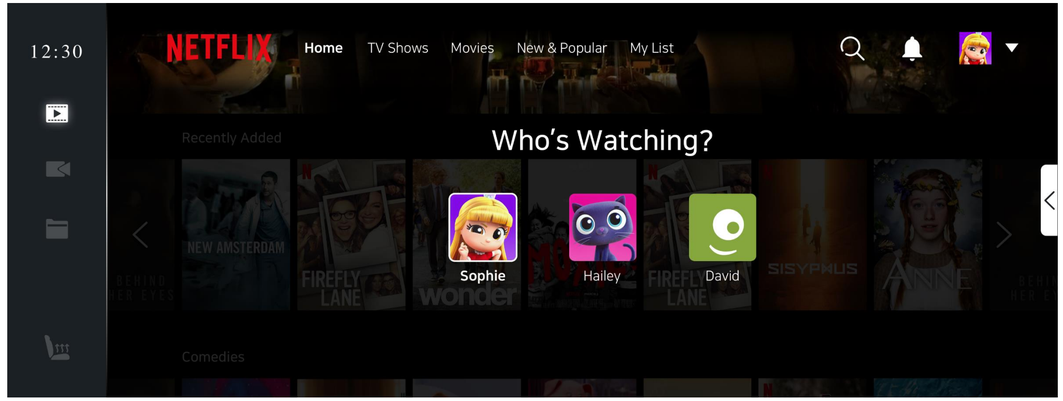
2.1 우리가 만들어야 하는 것

하지만 아직 차량과 관련된 요구 사항은 나오지 않았고..

지원해야 하는 전체 앱 목록도 아직 나오지 않았고..

기획과 디자인을 검토하는 동안에도 시간을 계속 흘러가고..

2.2 차량용 플랫폼이 되기 위해



사용자 프로파일

- 사용자 별로 다른 프로파일을 제공
- 앱 별 로그인 정보를 다르게 취급
- 다양한 접근 권한 제어

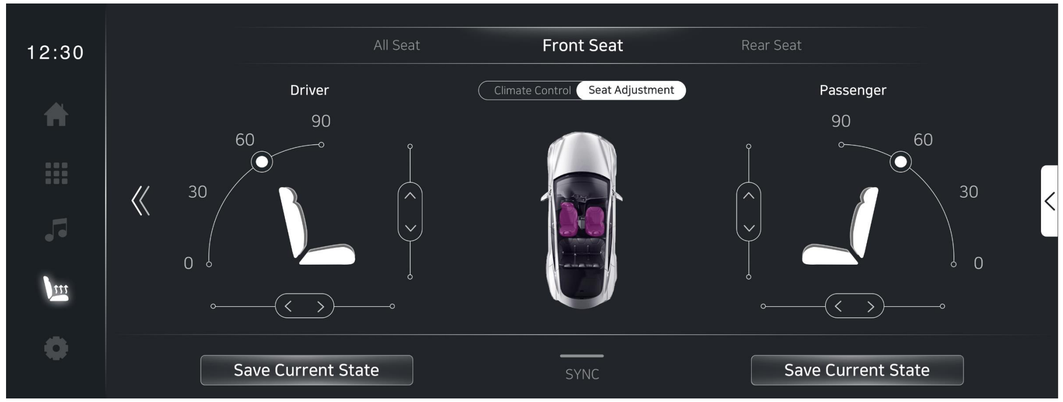
그 외,

- 차량이 알려주는 위험 신호에 반응
- 운전자의 H/W키 조작 반영
- 라디오나 음악같은 것은 당연히 되어야..

2.3 추가 개발 사항

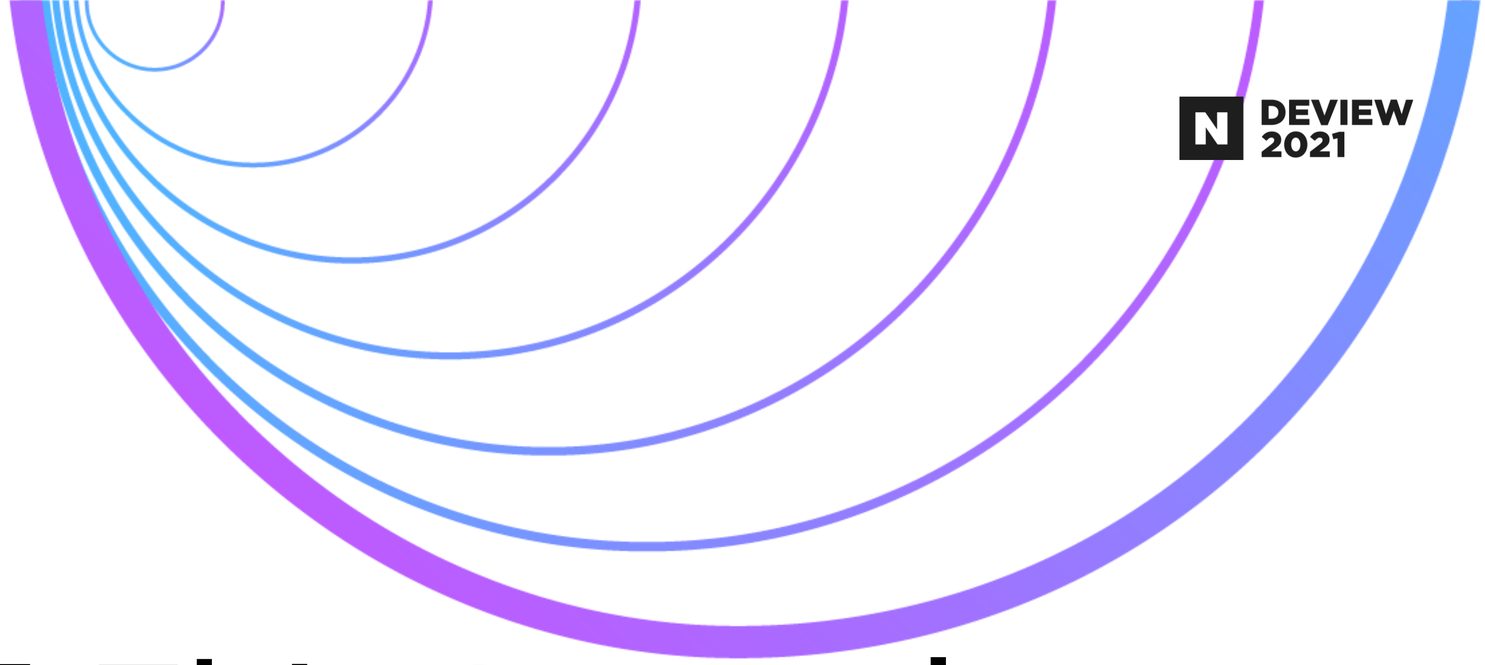
OS의 필수 앱 제작

- 세팅 메뉴
- 애플리케이션 스토어를 위한 앱
- 미디어 플레이어 등



차량용 필수 앱 제작

- 공조/좌석 제어 앱
- 내비게이션
- IVI를 위한 YouTube등의 앱



3. 플랫폼 필수 요소 구축

3.1 윈도우 시스템

DEVIEW 2021

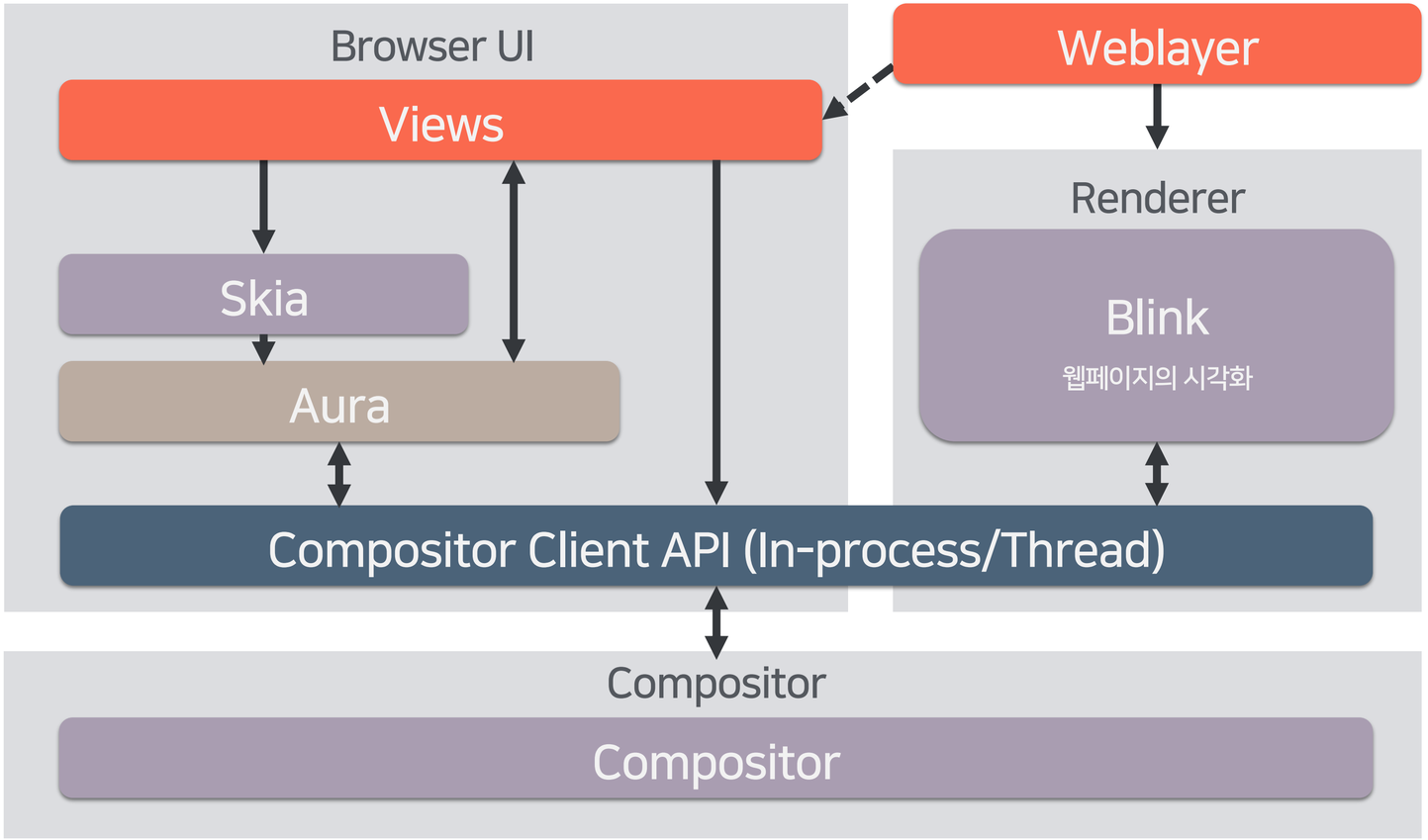
브라우저 엔진을 기반으로 한 윈도우 시스템

- 하나의 윈도우에 하나의 웹페이지를 그릴 수 있도록 분리
- 전화면에 뜬 하나의 웨일 브라우저가
  N개의 윈도우로 분리된 N개의 웨일 브라우저를 자식으로 가지는 형태

기존 웨일 브라우저의 서브 시스템 활용

- Views 기본적인 윈도우/이벤트/포커스 시스템을 가지고 있음
- Weblayer Views내의 컨텐츠를 웹페이지 방식으로 채울 수 있음

3.1 윈도우 시스템 - 구조도

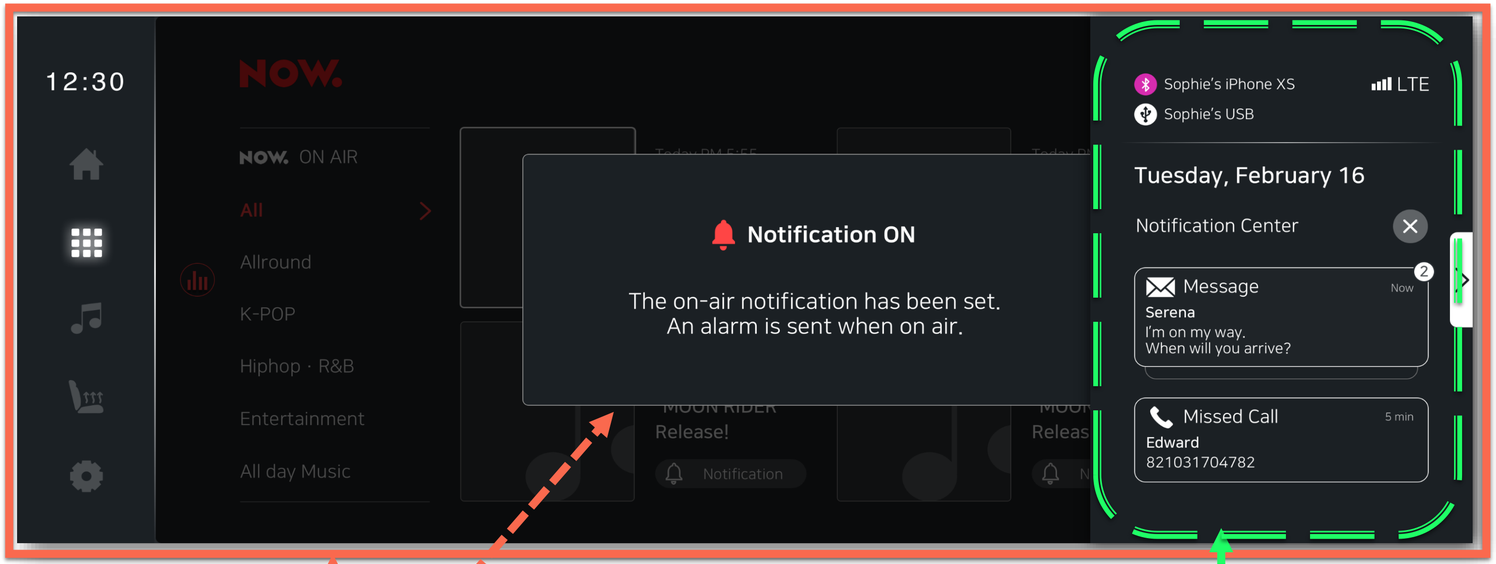


Chromium의 GUI는

브라우저 UI를 만들기 위한 기본 윈도우 시스템과

브라우저의 내용을 채우기 위한 렌더러 시스템으로 구성

3.1 윈도우 시스템 – 예제 화면



윈도우/포커스 시스템

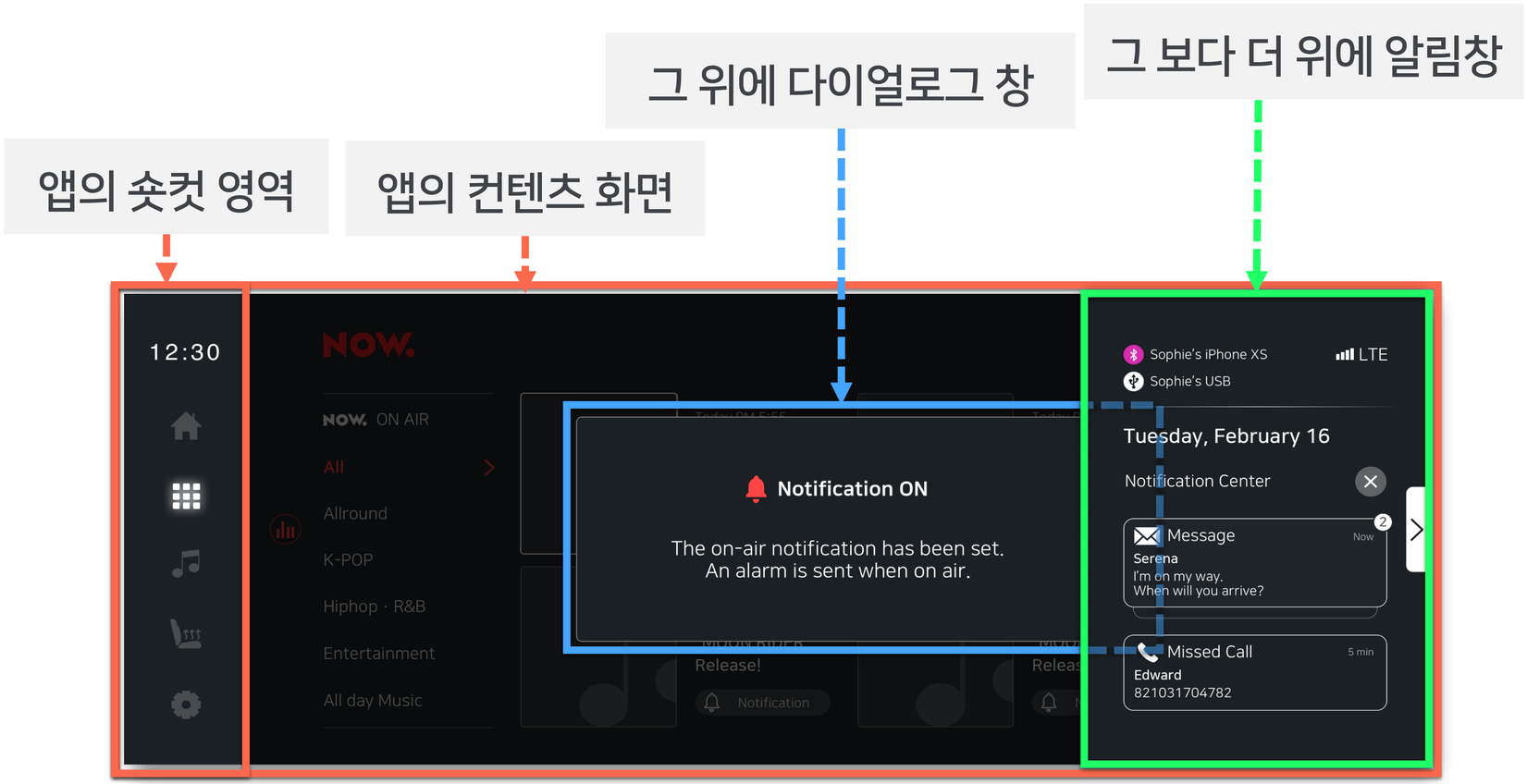
Chromium에 내장된 Views를 이용

윈도우 안의 내용

Chromium에 내장된 Weblayer 이용,

웹 컨텐츠로서 출력

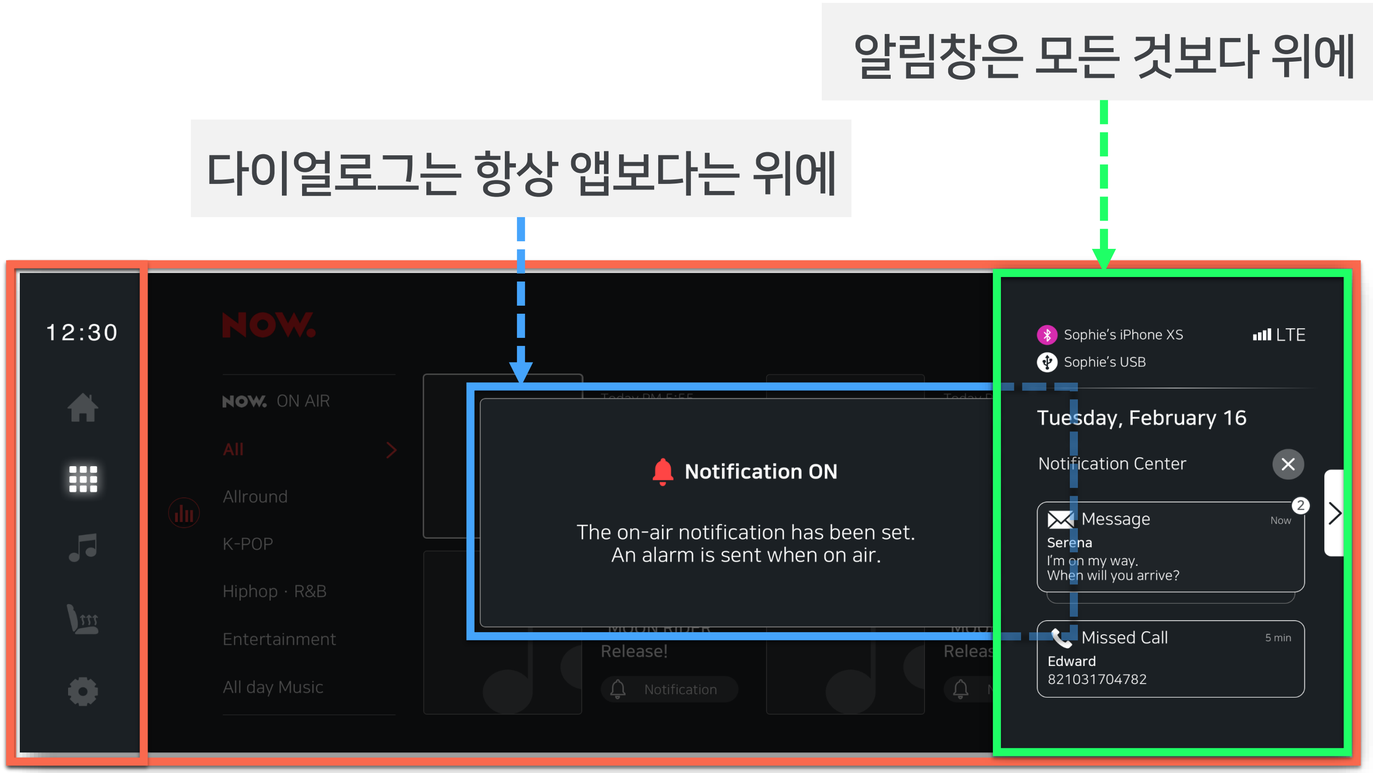
3.1 윈도우 시스템 – 예제 화면



예제 화면

일반적으로 발생 가능한, 다중 윈도우가 배치된 예

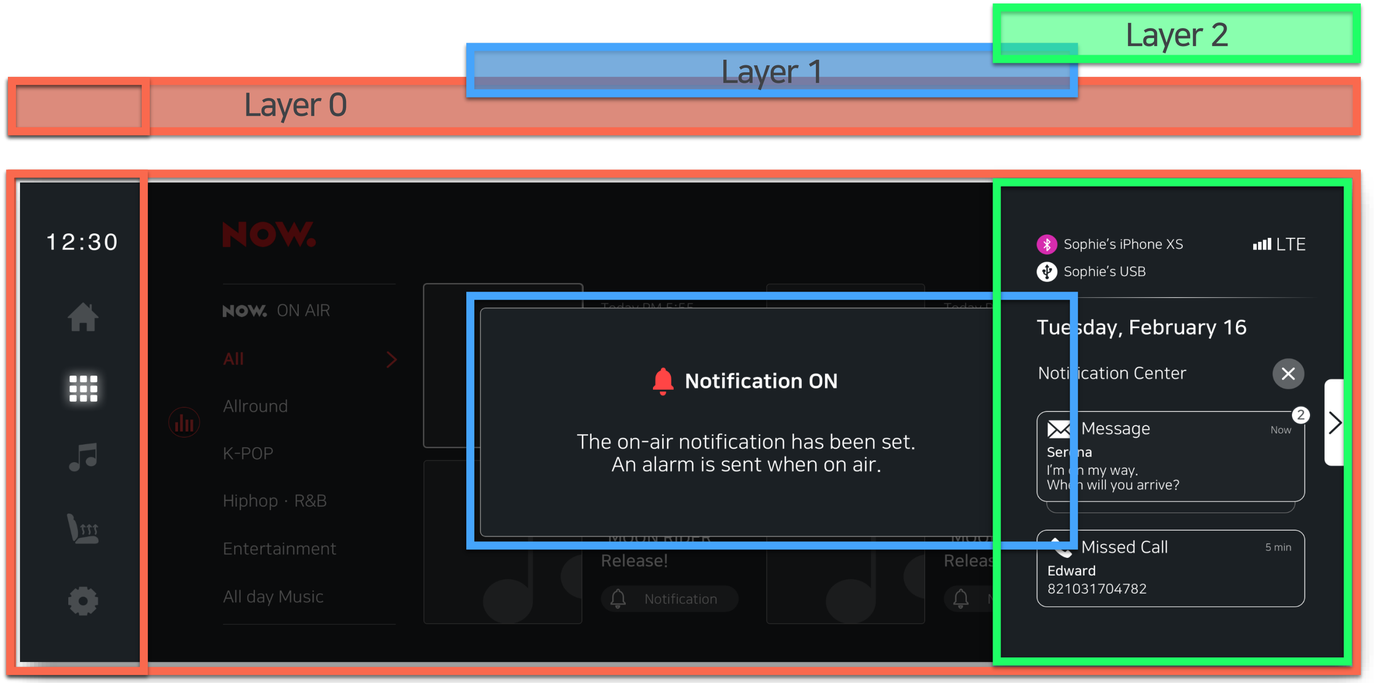
3.1 윈도우 시스템 – 예제 화면



윈도우의 z-order

특정 윈도우는 일반 윈도우보다 항상 위에, 또 어떤 윈도우는 그 무엇보다도 항상 위에

3.1 윈도우 시스템 – 예제 화면



윈도우의 z-order 보장

- 3개의 레이어로 나누고, 각 레이어 내에서 각각 윈도우 시스템을 운용
- 마지막에 3개의 레이어 합성

3.1 윈도우 시스템

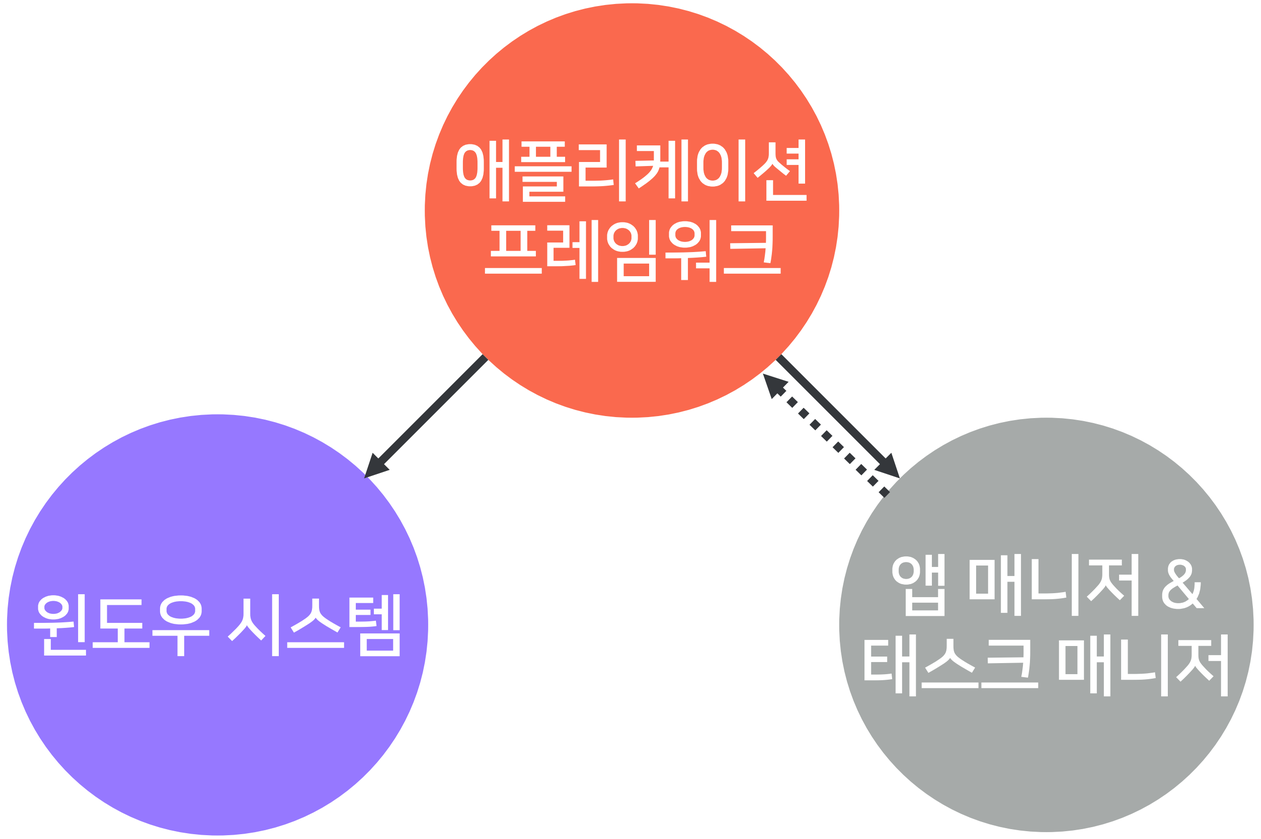
DEVIEW 2021

각각의 윈도우는 독립적인 시스템으로

- 각 윈도우는 샌드박싱된 별도의 프로세스로 동작
  - 서로 독립공간에서 보호됨
  - 앱에 치명적인 오류가 발생해도 전체 시스템에 영향을 주지 않음
- 각각 별도의 애니메이션과 투명도를 가짐

별도의 프로세스이지만 서로 통신이 가능해야 함

3.2 애플리케이션 프레임워크



윈도우 시스템의 최종 목적은 여러 앱을 화면 상에서 구동하는 것

위의 역할을 하는 애플리케이션 프레임워크가 필요

3.2 애플리케이션 프레임워크

웨일 Auto에서 앱의 범위를 먼저 정의

앱은 네이티브(C++)앱과 웹앱으로 구성

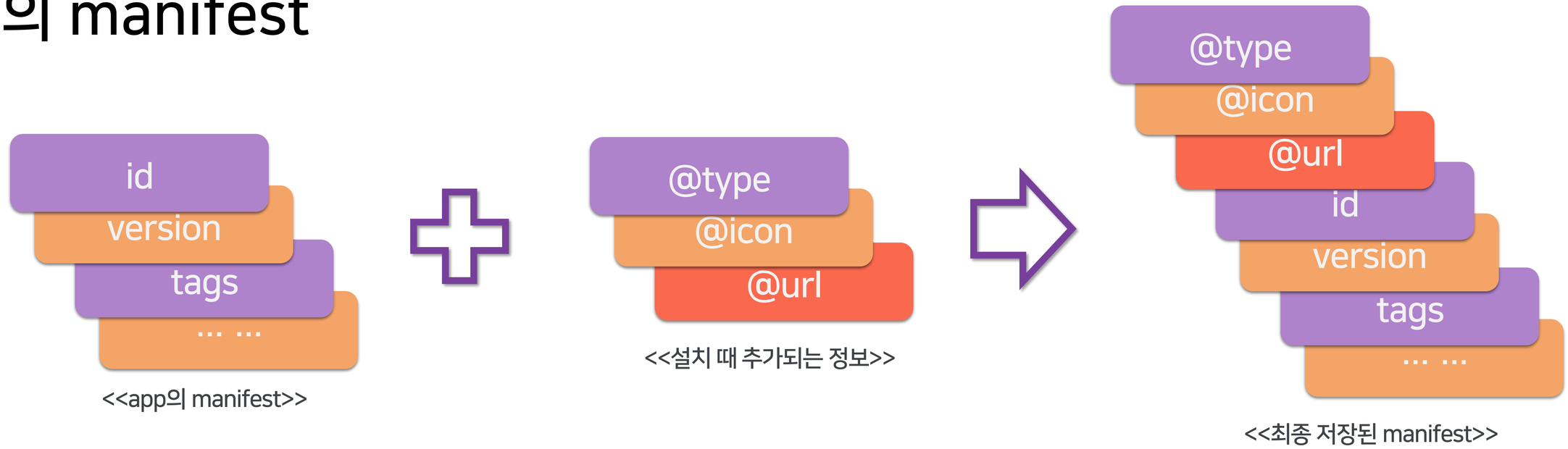
- 속도를 요구하는 기본앱은 네이티브(C++)로 제작
- 최근 대부분의 FE툴로 웹앱을 만들 수 있음
- PWA 방식이나 webpage를 직접 지정하는 방식의 앱도 지원

3.2 애플리케이션 프레임워크 – 앱 패키지

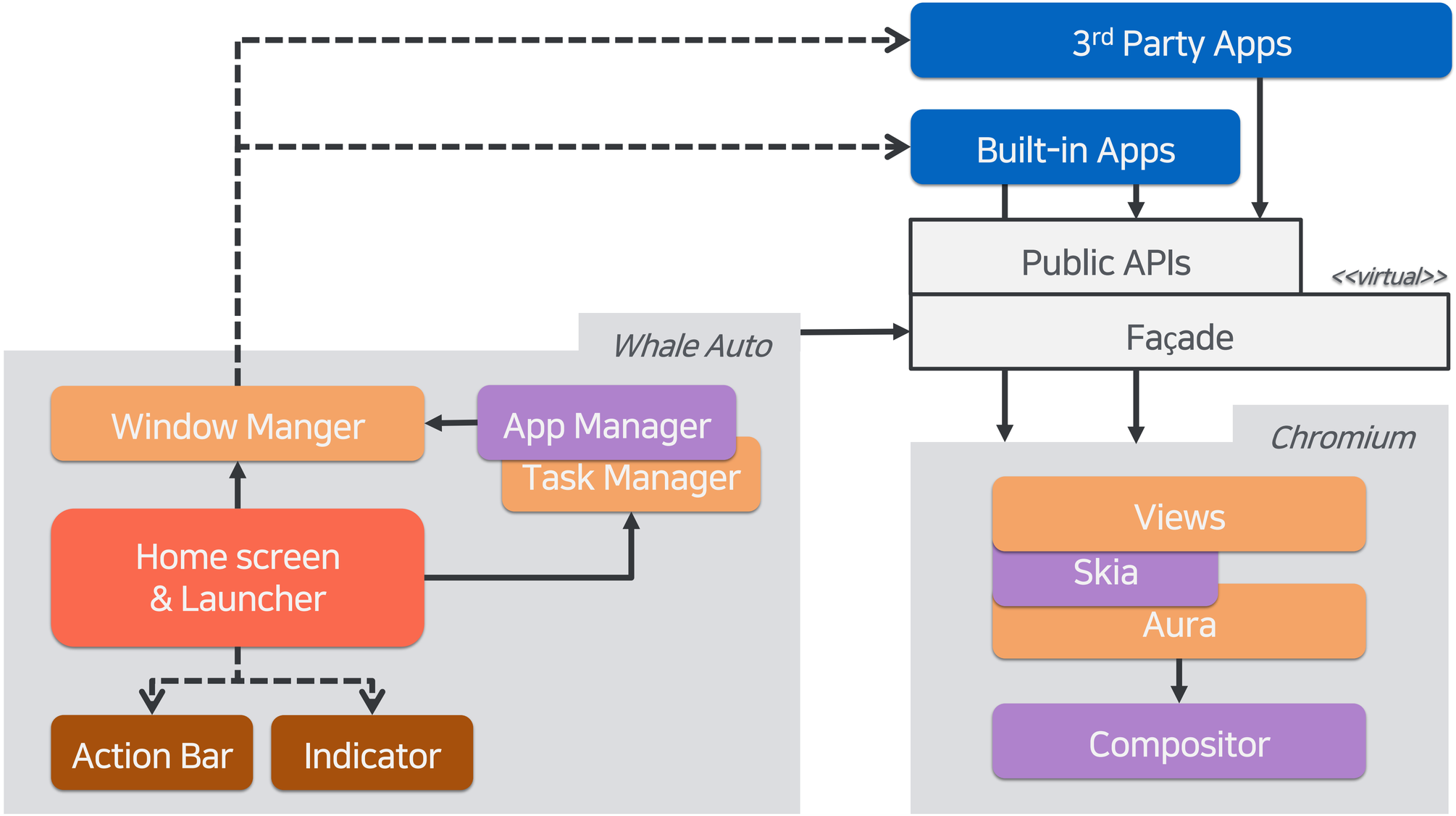
앱의 패키지 스펙이 필요

- 웨일 Auto용 웹앱을 위한 manifest를 만들고 패키징 형태를 구성
- 기존의 웹앱 패키징이나 안드로이드 패키징 방법 등을 참고

앱의 manifest



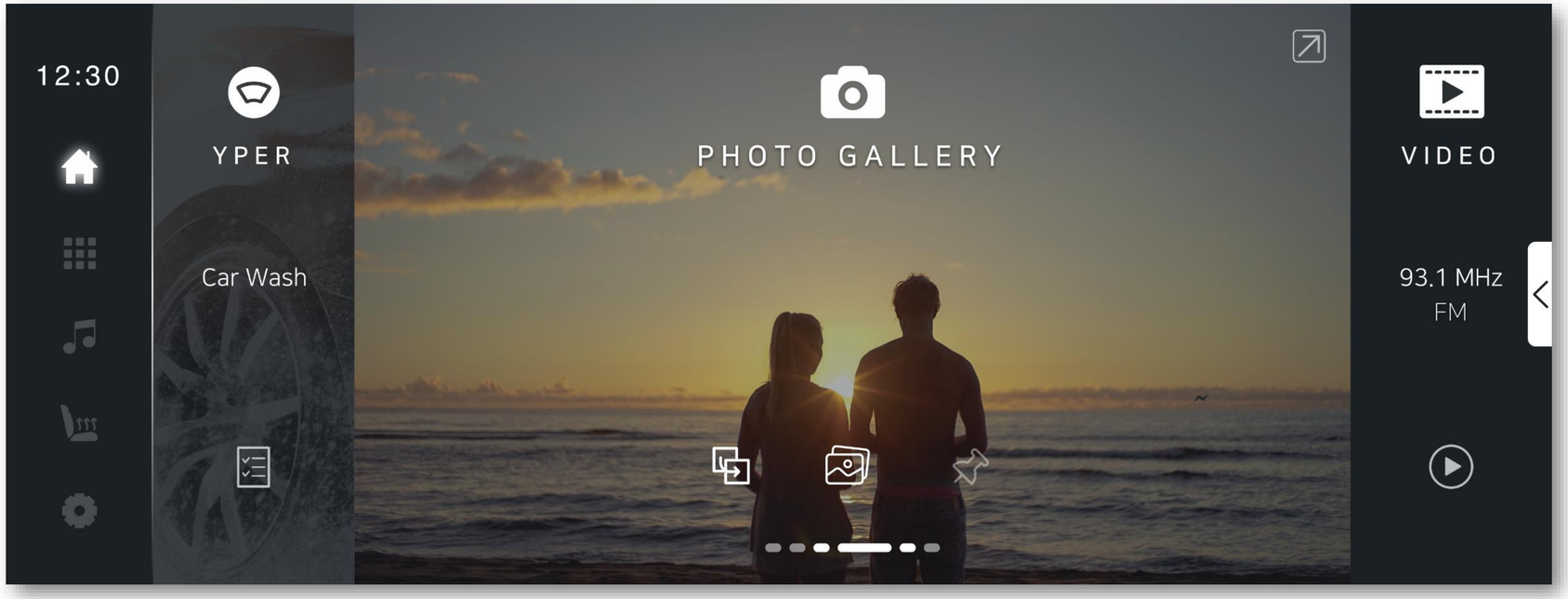
3.2 애플리케이션 프레임워크 - 구조도



3.2 애플리케이션 프레임워크 – 홈스크린

윈도우 시스템의 기본 앱인 홈스크린 앱

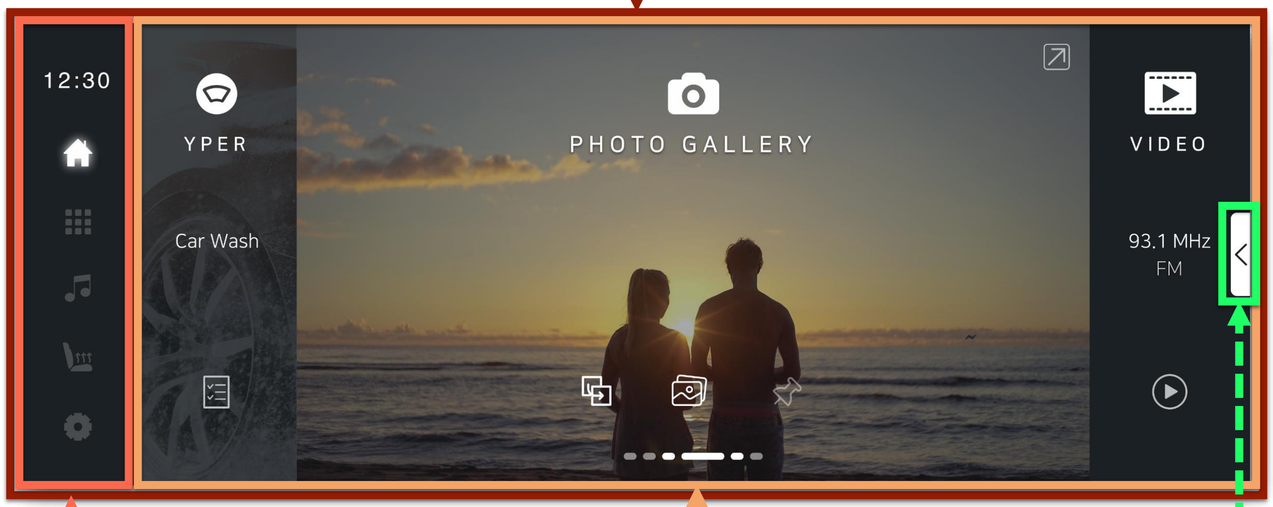
- 가운데의 컨텐츠 화면 / 왼쪽의 메뉴 화면 / 오른쪽의 알림 상태창
- 빠른 응답 속도와 높은 수준의 시스템 접근 권한이 필요
  네이티브(C++)로 제작



3.2 애플리케이션 프레임워크 – 홈스크린

<<Layer 0 + 2>>

전체 구성은 C++



메뉴는 web 형식

<<Layer 0>>

메인 앱은 web app

<<Layer 0>>

빠른 반응을 위해 C++

<<Layer 2>>

홈스크린 앱의 내부 구성

윈도우의 바탕화면과 같은 곳이므로, 빠른 응답성과 안정성이 요구됨

3.2 애플리케이션 프레임워크

앱의 설치 / 제거 / 실행 기능

- 설치되는 유형에 따른 앱의 타입 분류

앱 유형	업데이트 가능여부	예시
시스템 앱	단독 업데이트 불가	대화 모달 창, 알림창, 런쳐, 앱목록, 시스템 기본 앱 등
	업데이트 가능	내부 웹 브라우저, 차량용 기본앱 등
3rd party 앱	업데이트 가능	네이게이션, 미디어 플레이어, 기타 대부분의 앱

- 설치를 하면 앱이 아이콘 형태로 노출되게 하는 매커니즘
- 앱 아이콘을 통해 실행과 삭제가 가능하도록 연결

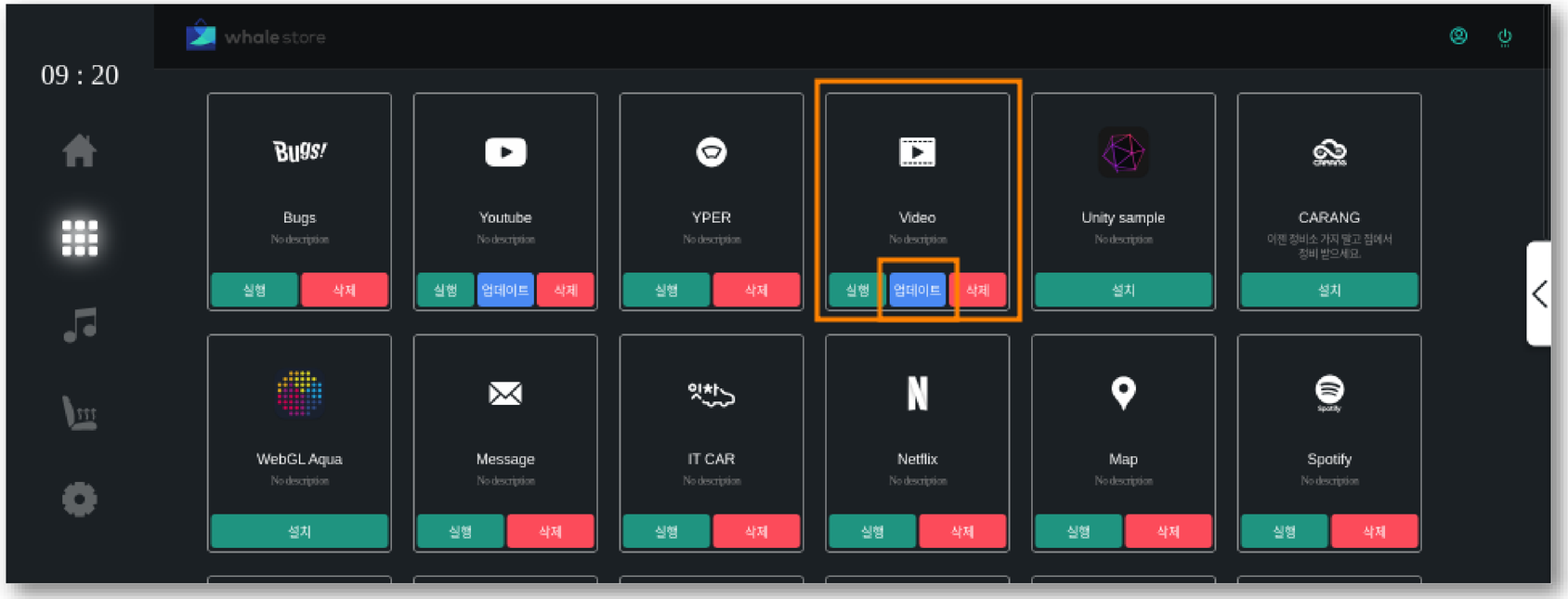
3.2 애플리케이션 프레임워크 – 웨일 스토어

웨일 Auto용 앱 등록

- (원래부터 있던) 웨일 스토어에서 서브 도메인을 추가

웨일 스토어,기본 앱으로 탑재 필요

- 스토어 앱에서, 설치 / 제거 / 업데이트 / 실행이 가능하도록

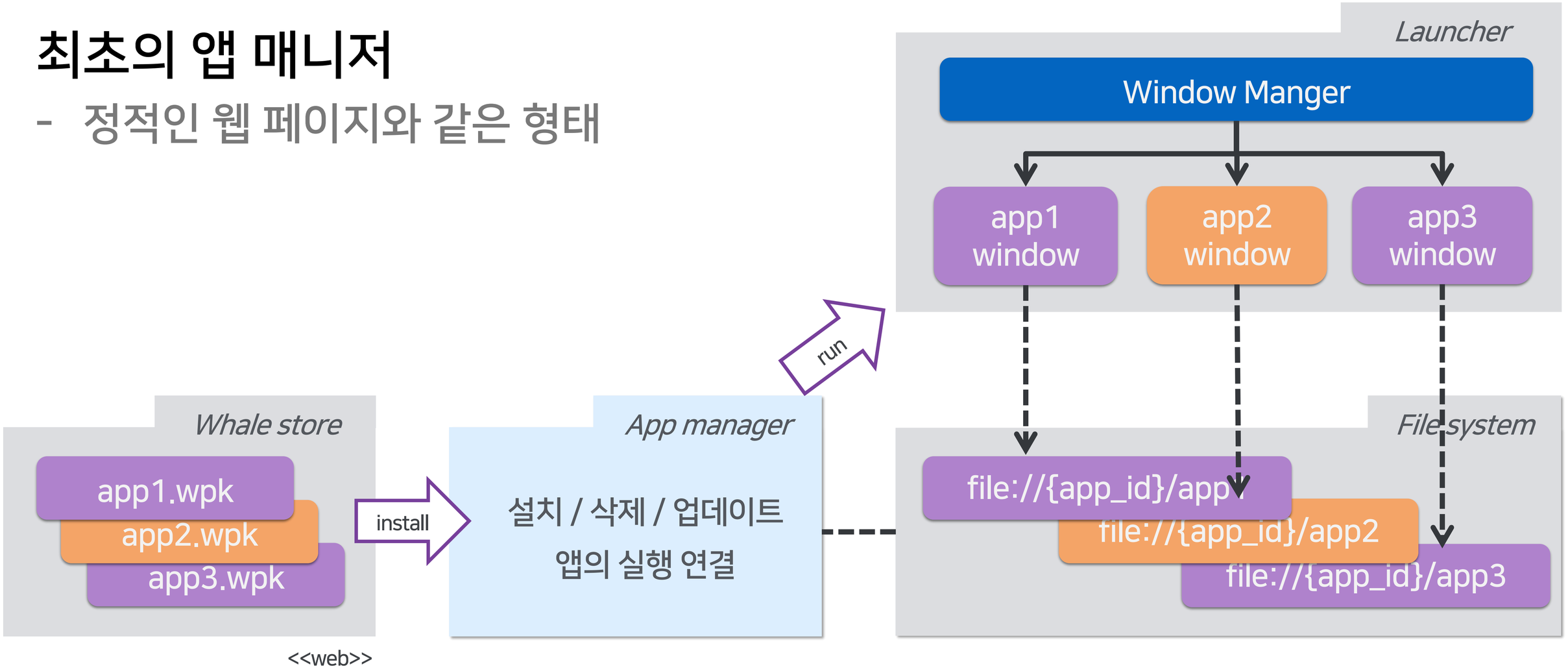


3.2 애플리케이션 프레임워크 – 앱 매니저

DEVIEW 2021

최초의 앱 매니저

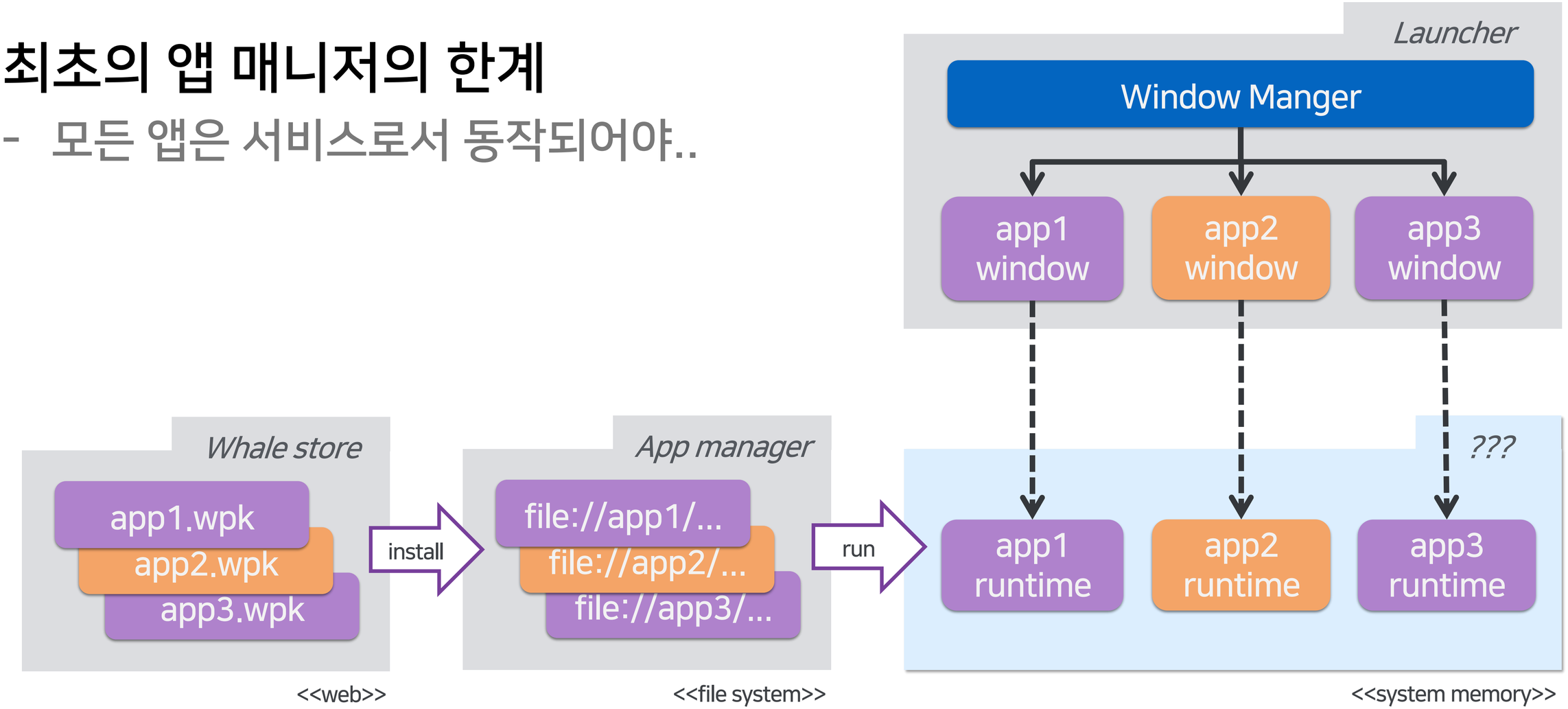
- 정적인 웹 페이지와 같은 형태



3.2 애플리케이션 프레임워크 – 앱 매니저

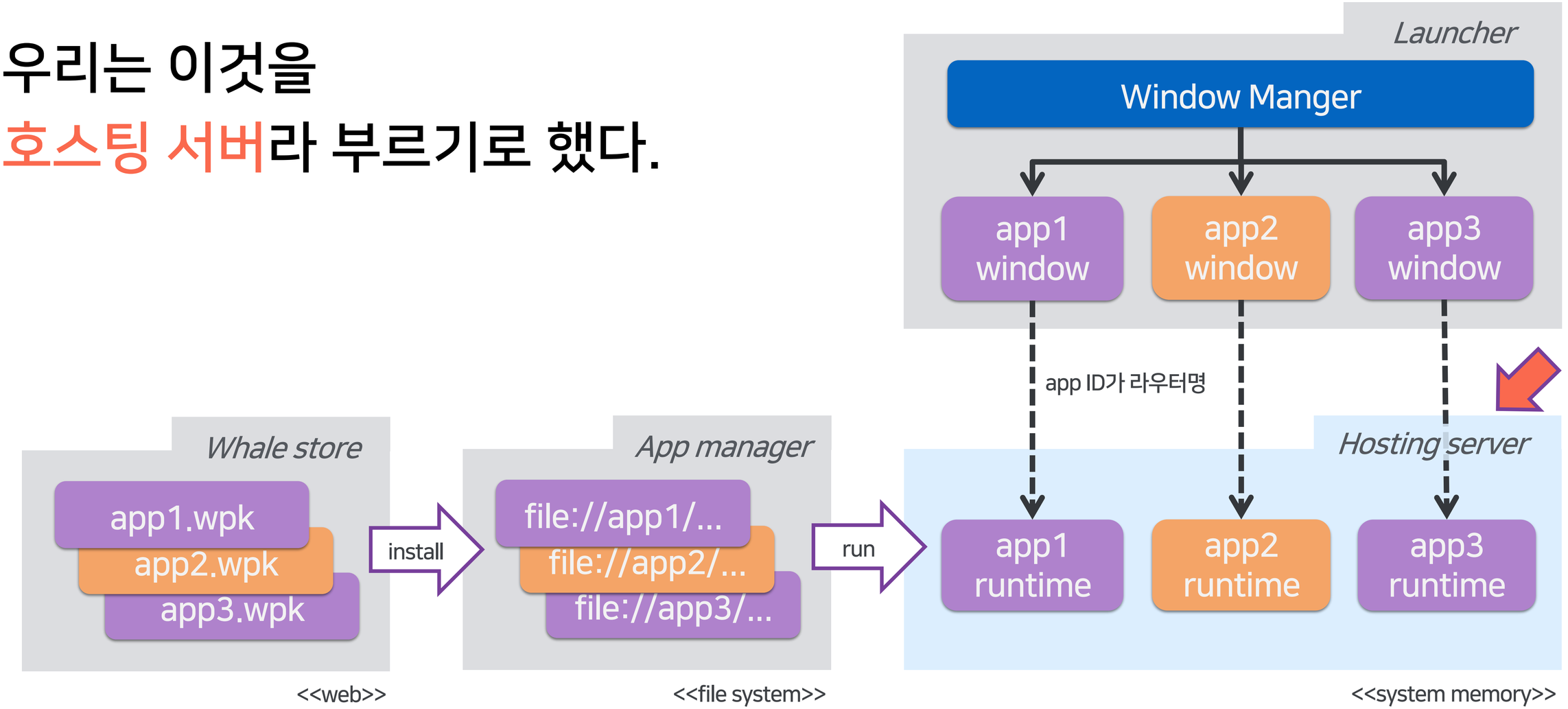
최초의 앱 매니저의 한계

- 모든 앱은 서비스로서 동작되어야..



3.2 애플리케이션 프레임워크 – 앱 호스팅 서버

우리는 이것을
호스팅 서버라 부르기로 했다.



3.2 애플리케이션 프레임워크

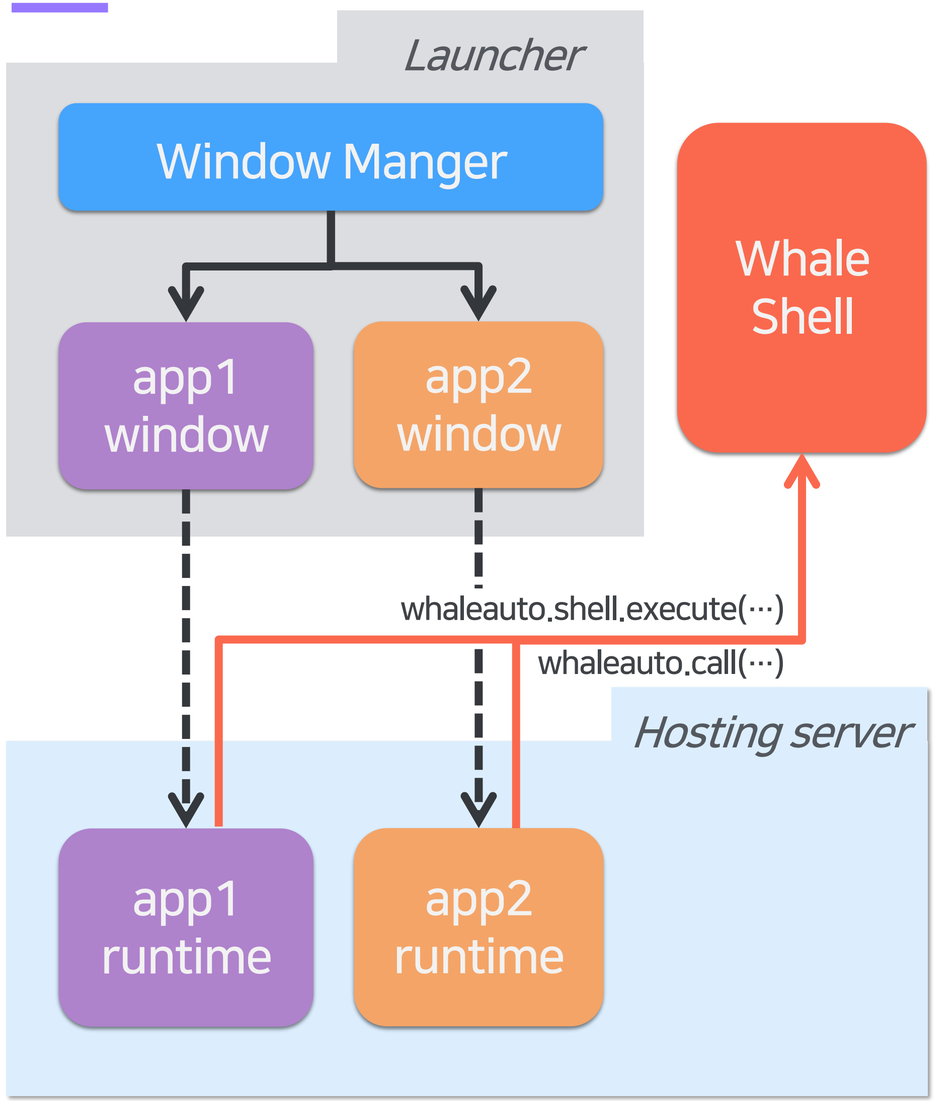
모든 웹 기반의 앱은 내부의 호스팅 서버를 통해서 실행

- 각 앱의 샌드박싱
  - 각 앱의 리소스는 다른 앱이 접근 불가
  - 각 app의 허용 권한에 대해 중앙 제어 가능
- 하부에서 전달되는 시스템 이벤트의 분배 기능
  - 기본적인 시스템 알림의 처리
- 서비스로서의 앱
  - PWA와 같은 service worker로 동작
  - 앱이 비활성화 일 때도 동작 가능

3.2 애플리케이션 프레임워크

웹앱의 API 확장

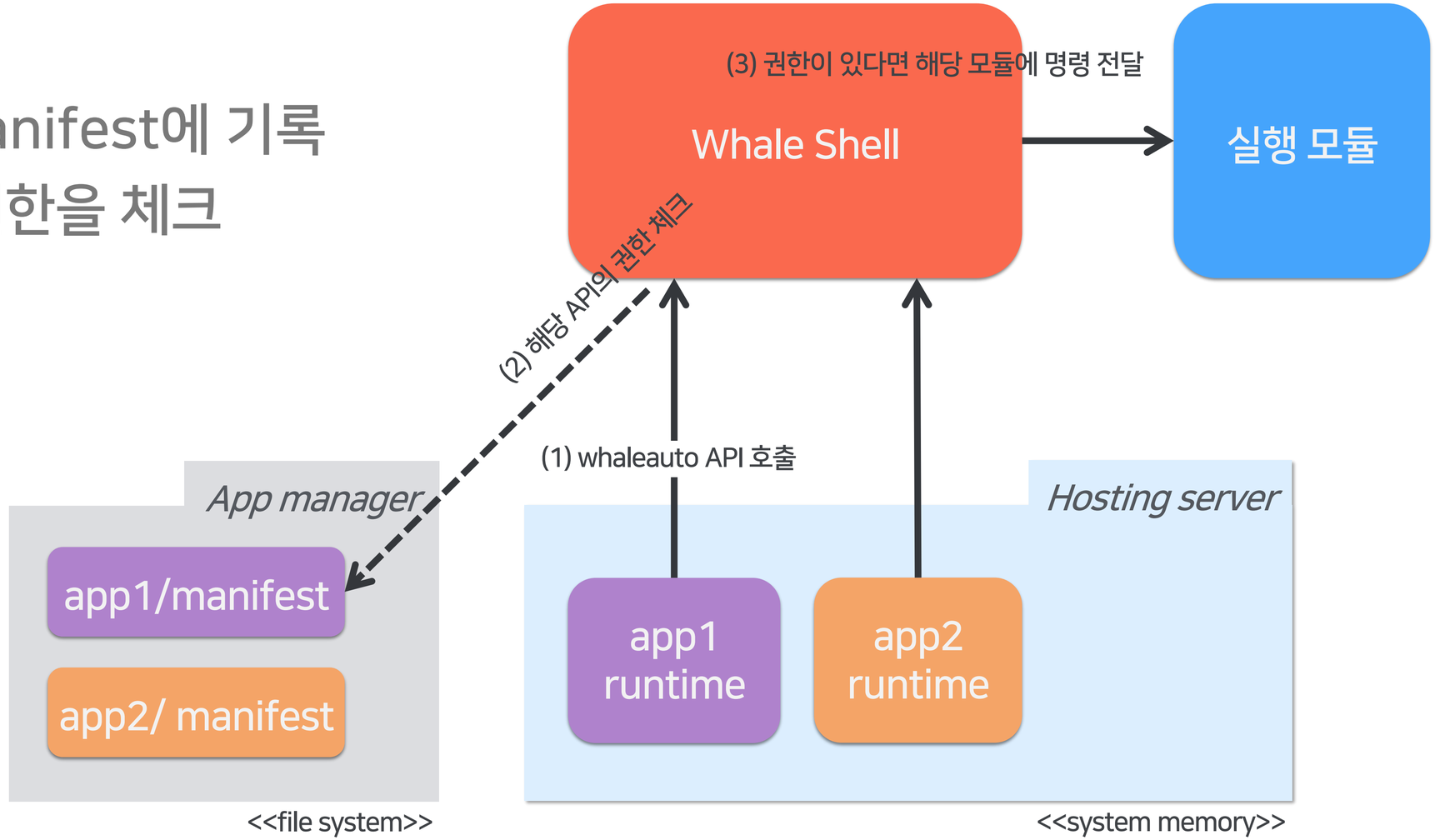
- 웨일 Auto의 독자 기능 사용
- ‘whaleauto’ 네임스페이스 제공
- 이 API는 웨일 Auto의 shell에 의해 시스템에 전파



3.2 애플리케이션 프레임워크

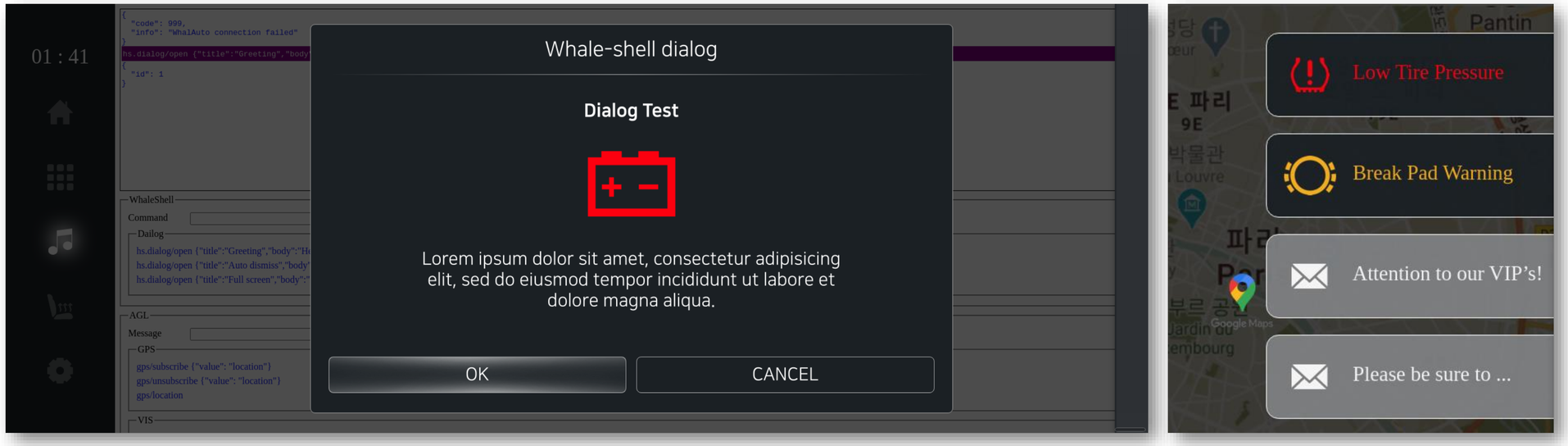
웹앱의 권한

- 앱의 권한은 앱의 manifest에 기록
- Shell은 앱의 호출 권한을 체크



4. 차량용 플랫폼 목적의 개발사항

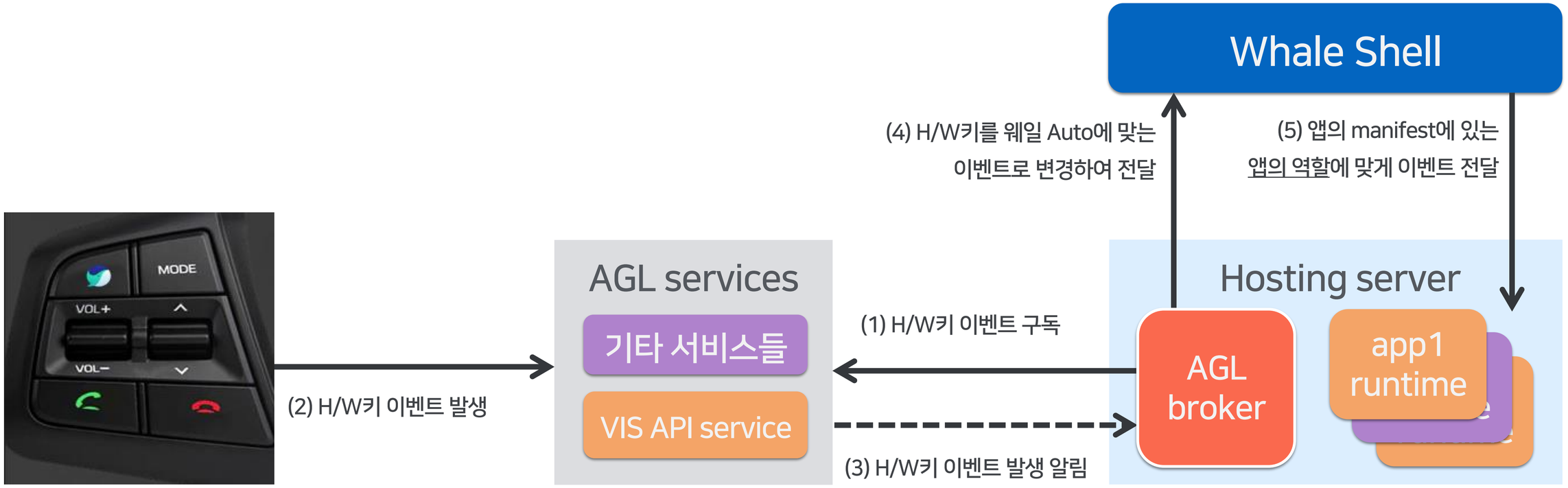
4.1 차량의 신호에 대응



차량의 이상 신호를 받아서 GUI 시스템에 표시

타이어 압력 브레이크 패드 교체 리덕션 기어 오일 부족 엔진 과열 배터리 경고

4.2 차량의 H/W 인터페이스에 대응

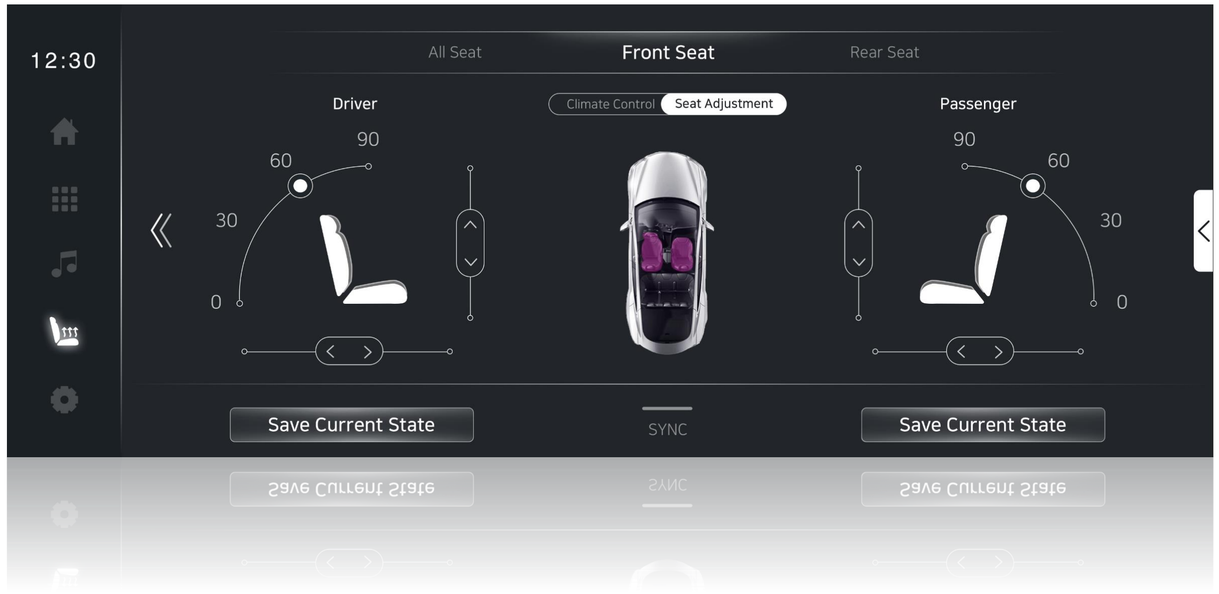


HW 키에서 발생하는 이벤트와 연동

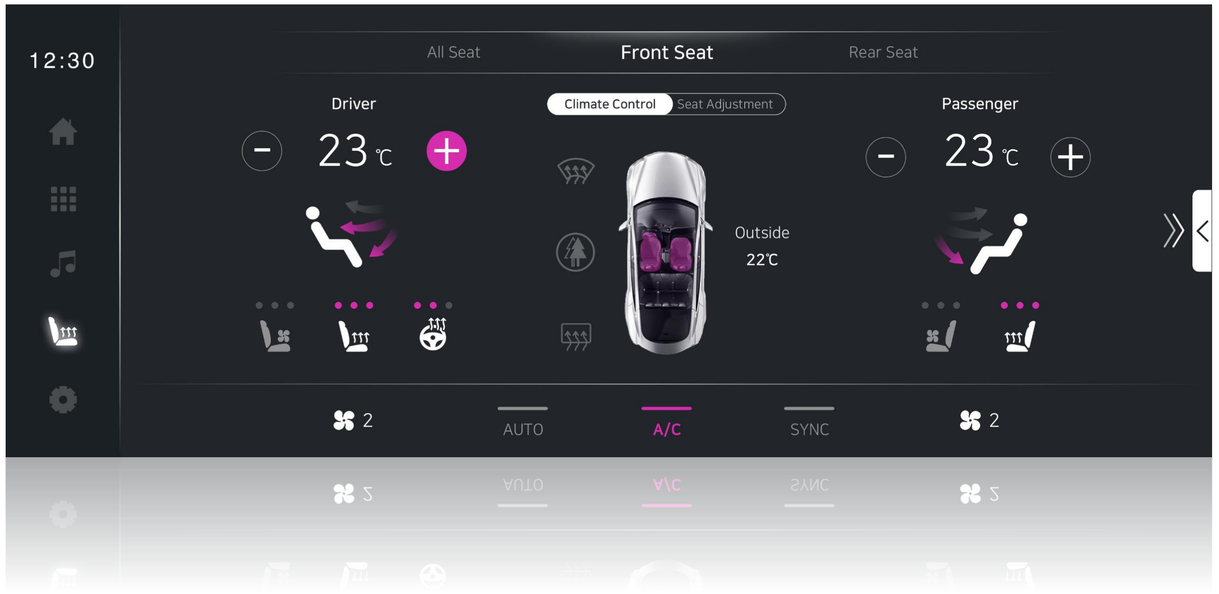
오디오 볼륨 전화 받기/끊기 앱 바로가기 버튼

4.3 차량용 기본 앱 제작

[차량 좌석 제어 앱]



[차량 내부 공조 제어 앱]



4.4 그 외 개발 사항

전화면에 비디오 화면을 출력 하는 문제

WebRTC를 통한 화상 회의

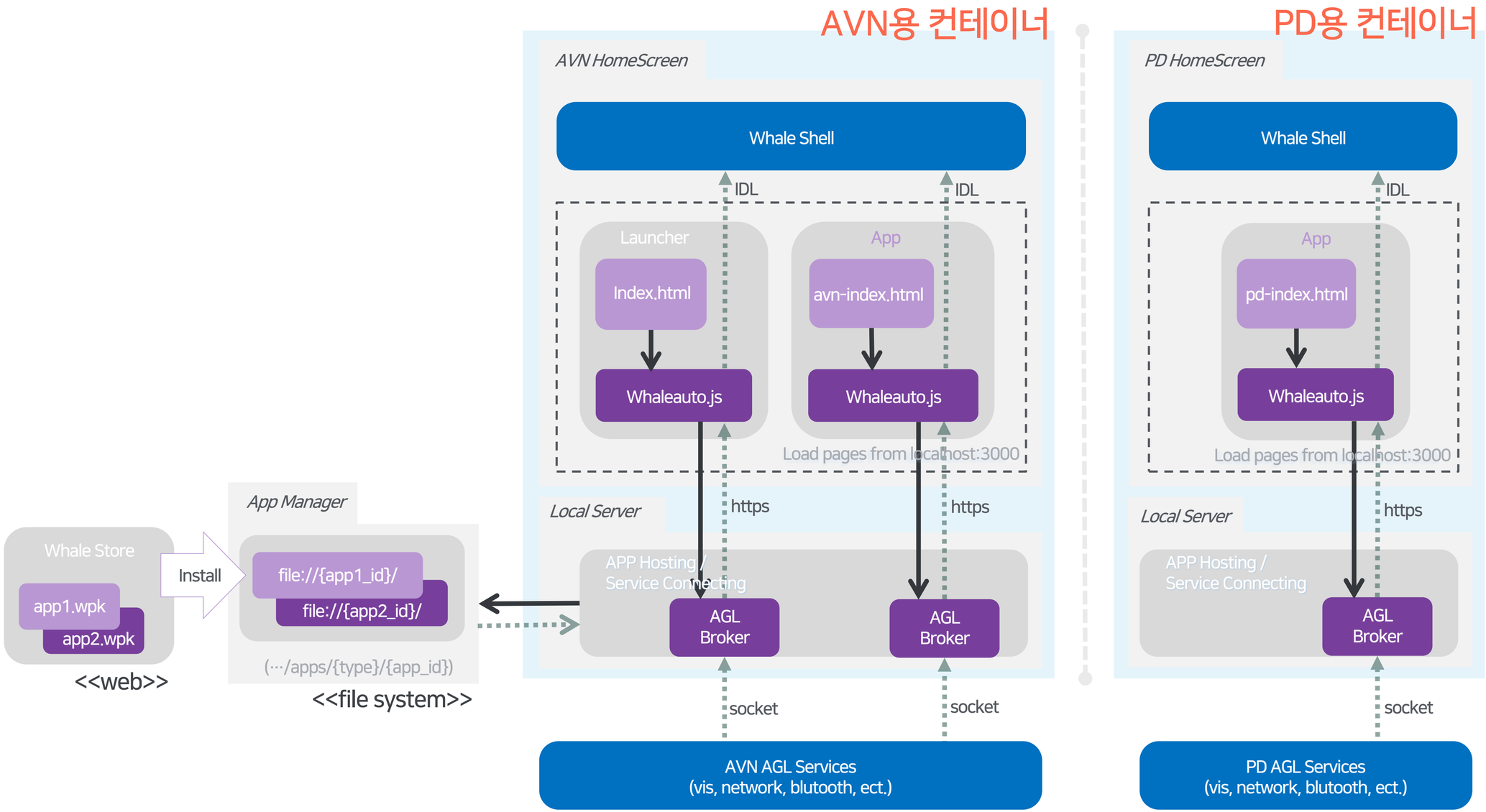
내부 브라우저와 연동

4.4 그 외 개발 사항

그리고 각 스크린별로 물리적인 컨테이너로 분리하는 일

- 화면은 Cluster / AVN / PD로 분리
- 각각을 별도의 독립적인 스크린으로 구성
- 각 스크린별로 독립적인 home 화면이 필요

4.5 필수 요소들로 구성된 시스템 구조도



DRIMAES

발표자-최현덕

5. AGL에서 구동하는 웨일 브라우저

5.1 Chromium 68 → Whale 83 포팅

DEVIEW 2021

문제상황 : 버전 차이가 심한 모듈의 포팅

- AGL 8.0 브라우저 - Chromium 68(2018)
- 타겟 브라우저 : Whale 83(2020)
- 담당자는 Chromium에 대한 지식이 낮음
- 그럼 포팅을 어떻게 하지?

5.1 Chromium 68 → Whale 83 포팅

단계적 접근

- 임베디드 환경에서의 개발에서 어려운 점 중 하나는 포팅
- Yocto로 시스템 및 라이브러리 버전의 기준 생성 (AGL 8.0 – Yocto 2.6 사용)
- Yocto 2.6에서의 동작을 보장하는 버전은 Chromium 80
- AGL의 Chromium 포팅 지원 범위 확인 필요 → 76, 80, 83 버전 확인

5.1 Chromium 68 → Whale 83 포팅

DEVIEW 2021

포팅 결과

Chromium 버전	포팅 결과
Chromium 76	정상 동작
Chromium 80	메뉴 및 우측 버튼 클릭 시, 팝업 미출력
Chromium 83	

결과 분석

- 데스크탑용 리눅스가 아니므로, 예상치 못한 문제는 나올 수 있음
- 76~80 버전의 사이에서 큰 변경사항이 있을 것으로 예상
  → Ozone wayland 구조 변경 : Wayland window와 Wayland popup으로 분류
- 디버깅 결과 메뉴 및 우측 버튼은 Wayland popup으로 실행

5.1 Chromium 68 → Whale 83 포팅

문제점

- IVI(In-vehicle Infotainment) shell 에는 Popup이라는 개념 자체가 없음
- Ozone wayland는 Desktop shell을 기반으로 코드 작성

해결방안 : 없으면 직접 만들자!

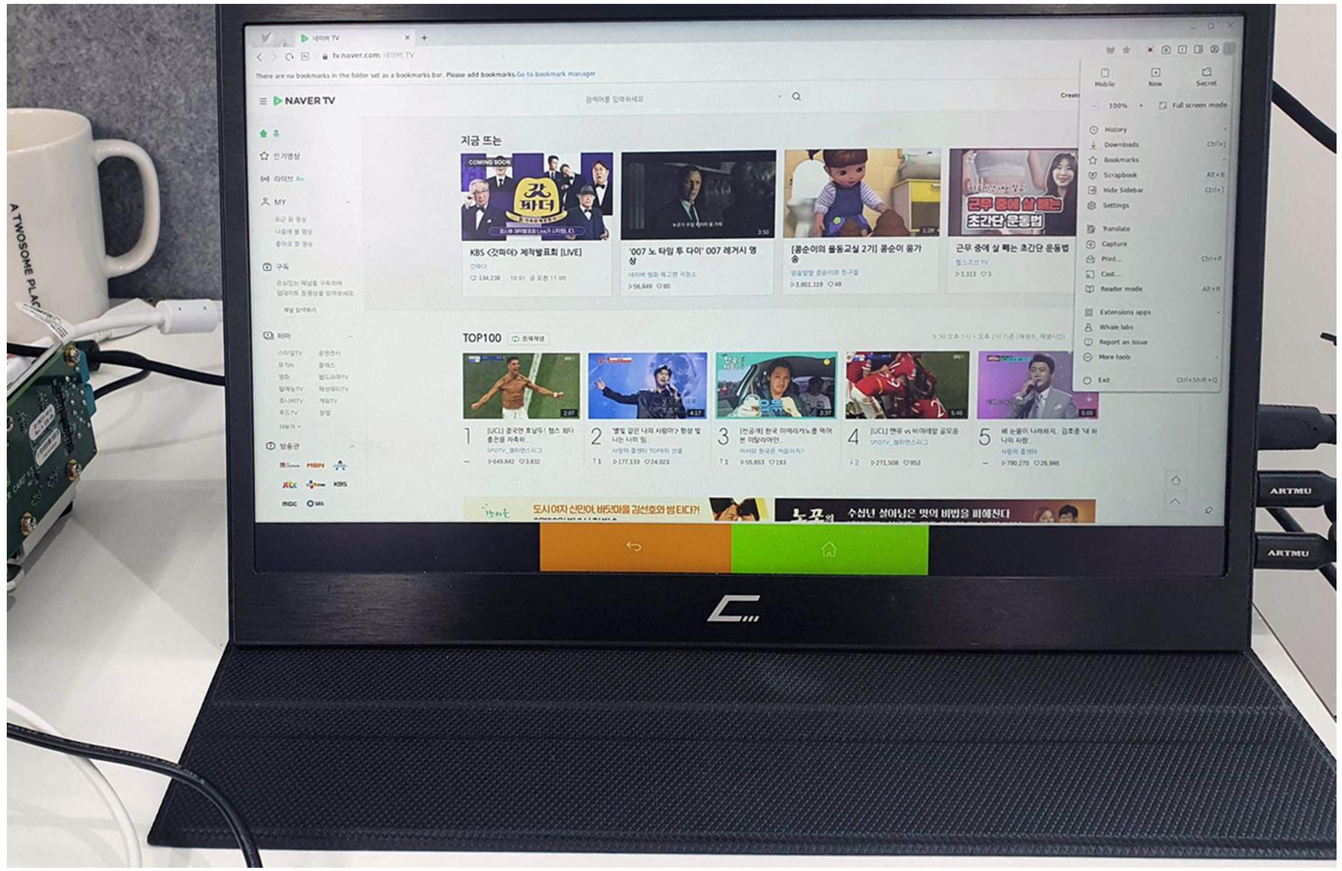
- IVI shell 컨셉 + Ozone wayland 컨셉 결합
- Chromium 에서는 Popup처럼 사용하되 IVI shell에서는 Surface로 인식
- Chromium 동작 여부를 확인 후 수정 사항을 웨일 브라우저에 적용

5.1 Chromium 68 → Whale 83 포팅

DEVIEW 2021

결과

- 타겟 보드에서 웨일 브라우저 구동
- Popup도 정상적으로 동작



5.1 Chromium 68 → Whale 83 포팅

DEVIEW 2021

Yocto에 적용

- Bitbake 작성
- 브라우저 수정 관련 내용을 패치 파일로 작성

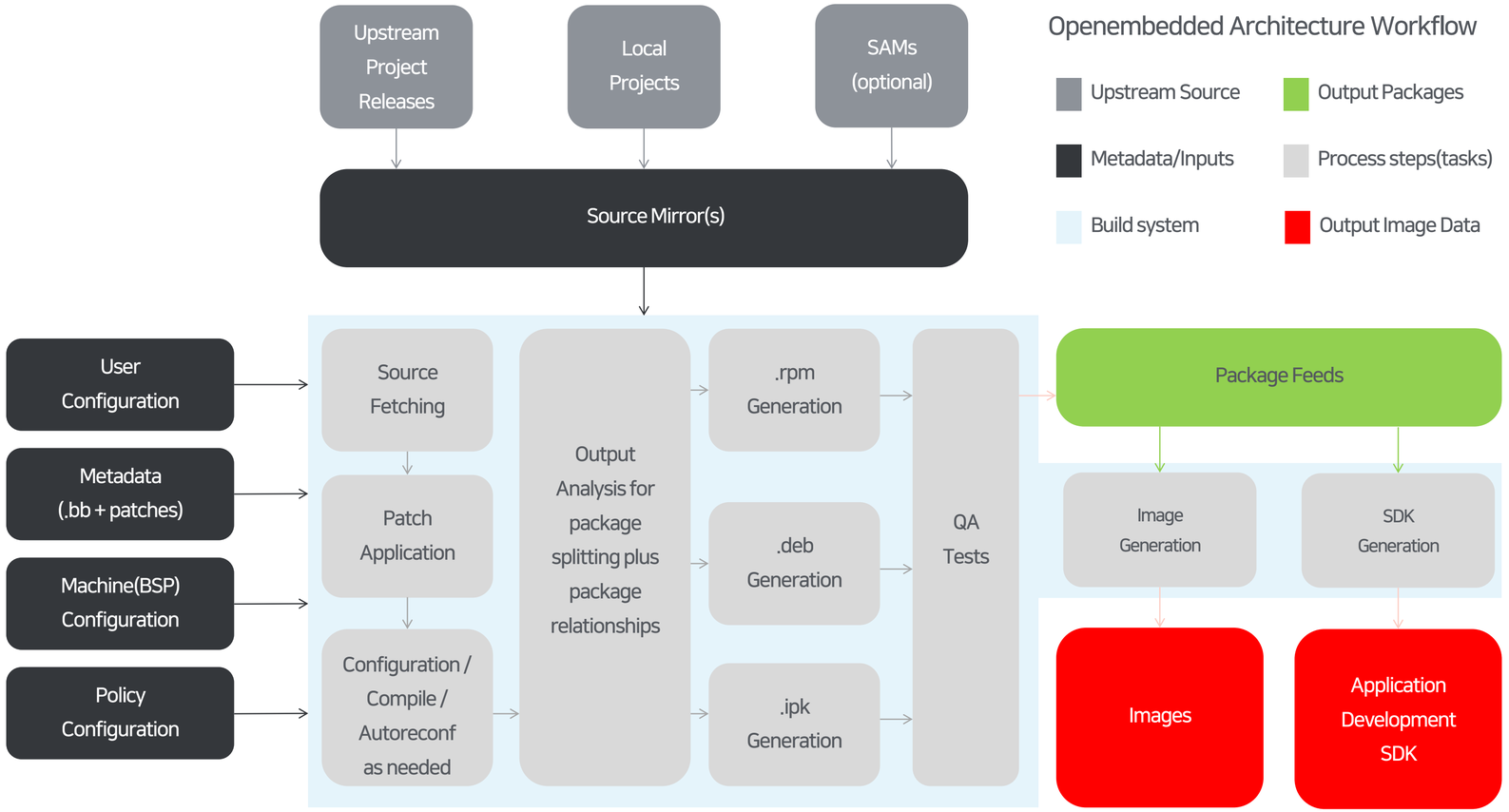
빌드 테스트

- 빌드 오류가 나지 않고 빌드가 정상적으로 되어야 함
- 퓨징 후 정상동작 해야 함

5.1 Chromium 68 → Whale 83 포팅

DEVIEW 2021

Yocto 동작 프로세스



https://slowbootkernelhacks.blogspot.com/2016/12/yocto-project.html

5.2 Web Feature 구현 - PWA

PWA(Progressive Web App)

- Chromium 72에서 지원
- Chromium 68 기반의 AGL 에서는 지원 불가

PWA 구조 및 실행 방식에 대한 이해

- 브라우저에서 설치 → Chromium 내부에 설치, 바탕화면에 바로가기 생성
- 각각의 고유 ID가 있으며, 이를 실행 인자로 사용하여 실행

5.2 Web Feature 구현 - PWA

문제점

- AGL에서는 일반적인 앱 실행 방식과 완전히 다름
- 현재의 방법으로는 사용 불가

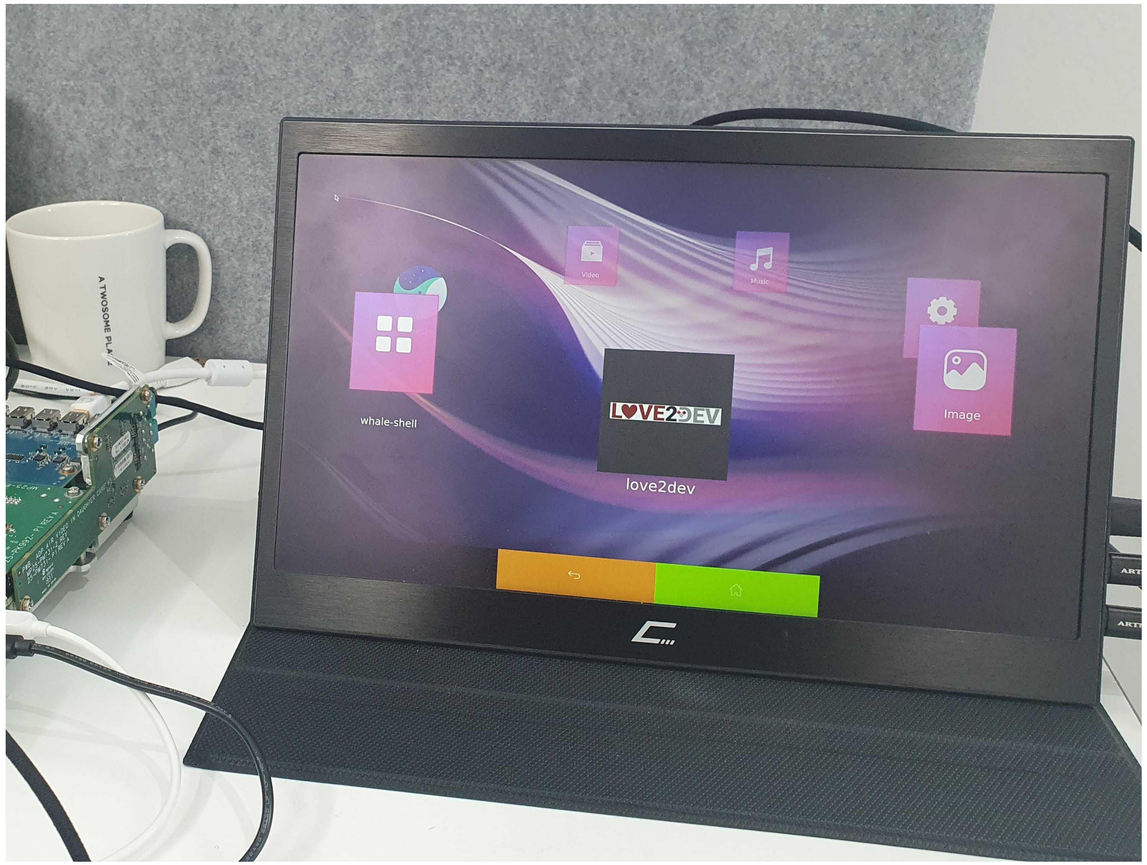
해결방안 : AGL 버전을 만들어서 이식

- PWA 설치시 AGL 앱으로 설치되게끔 만들자
- AGL App은 고유의 ID를 가지므로 PWA의 ID와 연동하여 고유ID 생성
- 설치 뿐만 아니라 삭제도 가능하게 구현

5.2 Web Feature 구현 - PWA

DEVIEW 2021

결과



6. AGL에서 구동하는 웨일 Auto

6.1 웨일 Auto를 위한 AGL의 기능 확장

요구사항 분석

요구사항	내용
① AGL 서비스와 웨일 Auto와의 연동을 위한 API 필요	와이파이, 블루투스, 차량 정보, 사운드, GPS 등
② 타겟 보드에서 원활한 동작을 보장해야 함	SA8155P
③ 협업을 원활하게 하기 위한 방법 필요	[기존] 서로 다른 사용 TOOL (웨일 : Github / 드림에이스 : Gitlab) [협업 방안] 소스관리 - Github API 문서관리 - Notion

6.1 웨일 Auto를 위한 AGL의 기능 확장

DEVIEW 2021

설계

재사용 가능 모듈과 신규 개발 모듈 분리

- 재사용 가능 모듈 : 와이파이, GPS 등
- 신규 개발 모듈 : 블루투스, 차량 정보 등

Yocto 메타 구조 설계

- 고려사항 : 타겟 보드에 종속되지 않는 메타 구성

개발

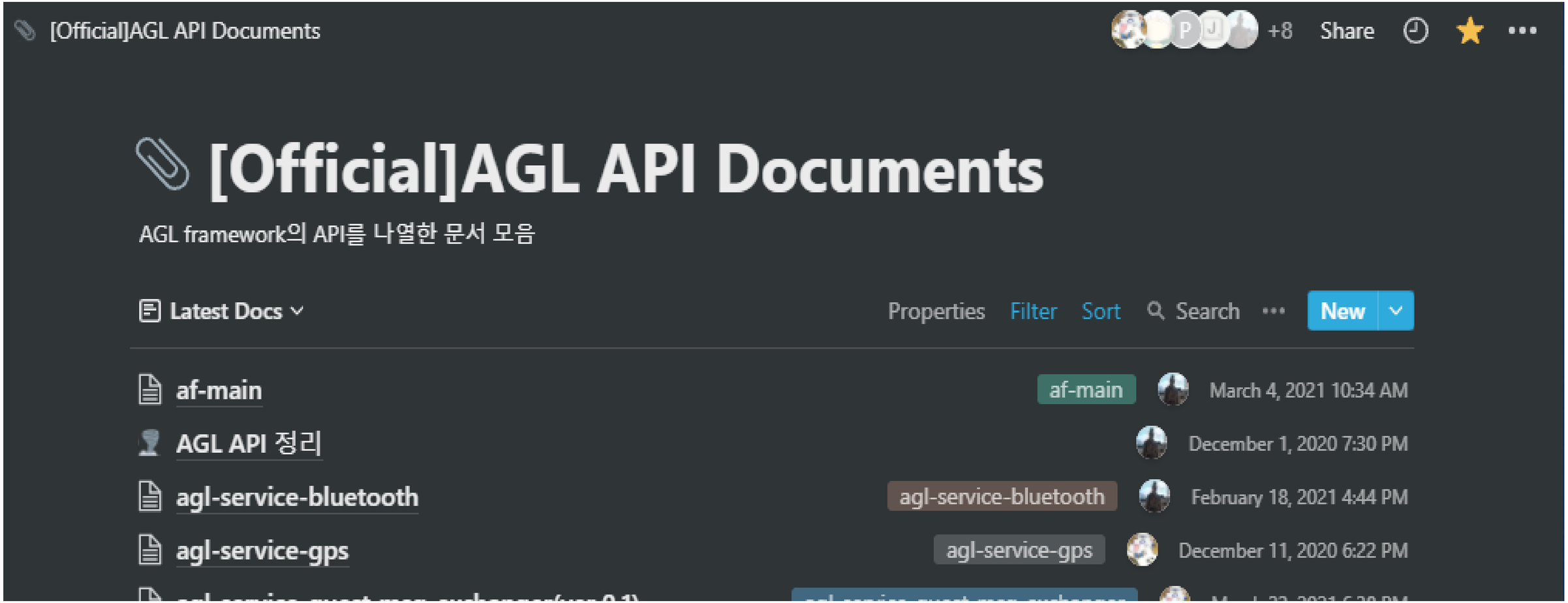
유닛 테스트

통합 테스트

6.1 웨일 Auto를 위한 AGL의 기능 확장

DEVIEW 2021

기능별 AGL API 문서



6.1 웨일 Auto를 위한 AGL의 기능 확장

DEVIEW 2021

agl-service-gps API 문서

6.2 오토모티브 웹 API 도입 - VIS

DEVIEW 2021

도입의 계기

- Web에 친화적인 환경을 만들자!
- 새로운 오토모티브 웹 API 도입은 어떨까?
- 자료 조사

VIS(Vehicle Information Service)

- W3C Automotive 에서 제공하는 각종 차량 정보에 대한 관리 등을 명세
- VSS(Vehicle Signal Specification) 데이터를 기반으로 동작
- VIS의 오픈 소스가 존재

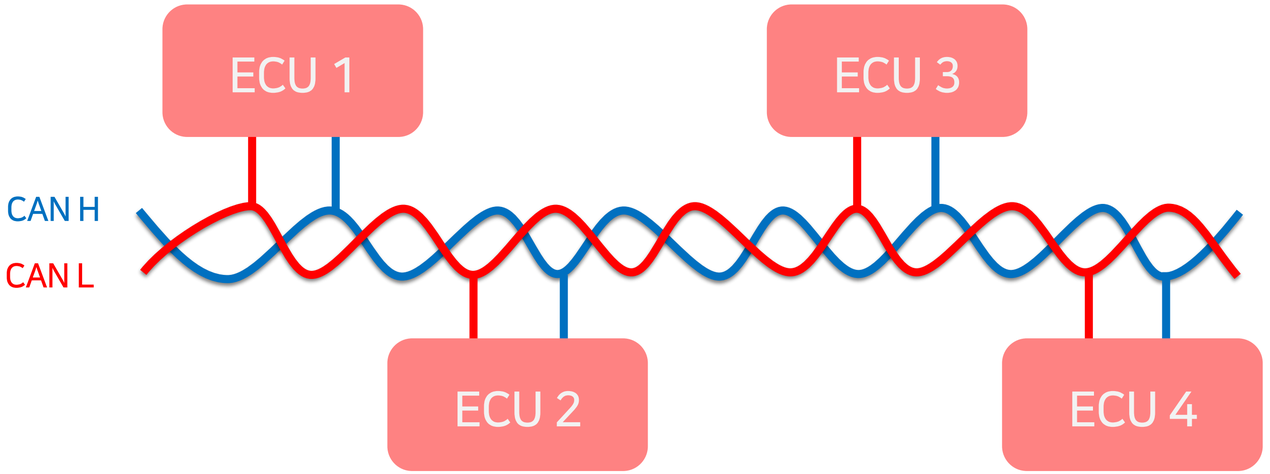
https://w3c.github.io/automotive/vehicle_data/vehicle_information_service.html

https://github.com/COVESA/vehicle_signal_specification

6.2 오토모티브 웹 API 도입 - VIS

사전지식 - CAN(Controller Area Network) 통신

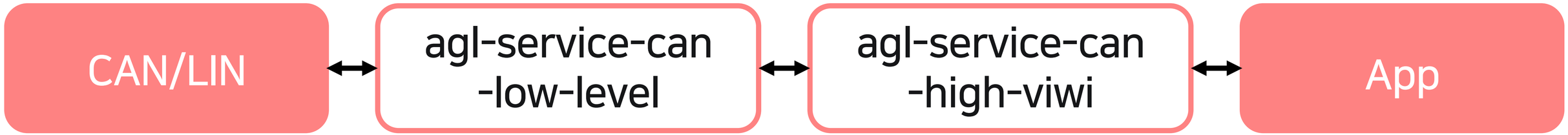
- CAN L(Low) / CAN H(High)의 2가닥으로 구성
- 노이즈나 간섭에 강함
- 차량용 ECU간 통신에서 많이 사용
- Master-Slave 구조



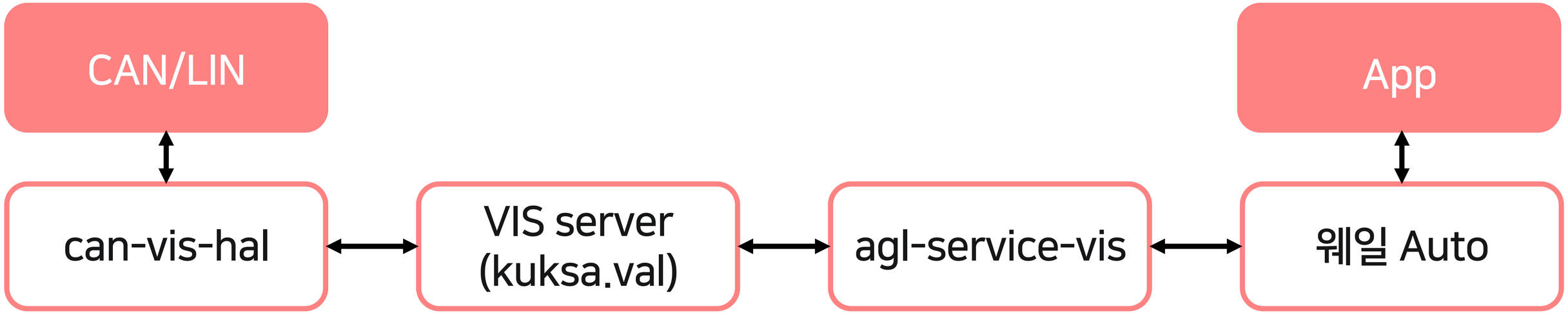
ECU : Electronic Control Unit

6.2 오토모티브 웹 API 도입 - VIS

기존 AGL 의 차량 정보 통신 방식



VIS 를 적용한 차량 정보 통신 방식



6.2 오토모티브 웹 API 도입 - VIS

기존에 있는 모듈

- VIS Server(kuksa.val) : 오픈 소스 활용

추가 개발이 요구되는 모듈

- can-vis-hal : CAN ↔ VIS 데이터 변환 모듈
- agl-service-vis : VIS 데이터를 받아서 AGL API의 형태로 변환해서 제공

VIS server source code : https://github.com/eclipse/kuksa.val

6.2 오토모티브 웹 API 도입 - VIS

늘어난 레이어, 문제 없을까?

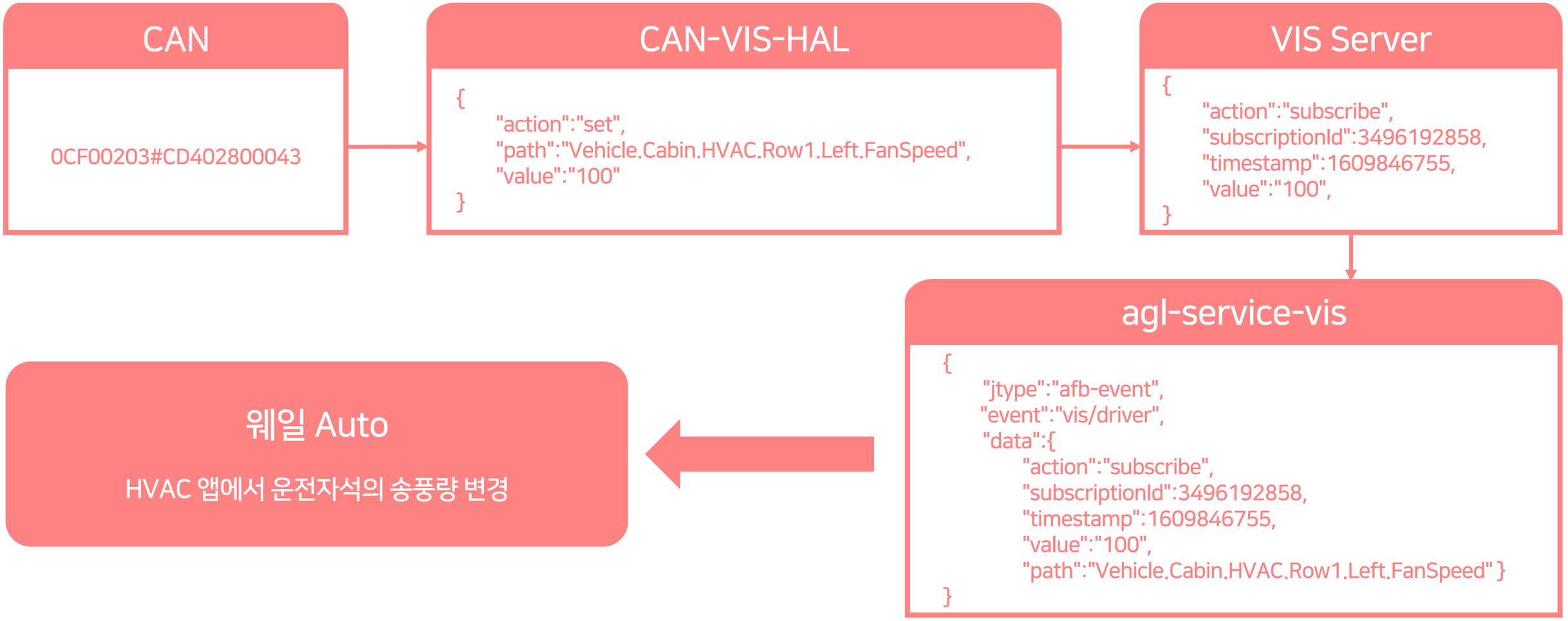
- 테스트 결과 시간적으로 크게 차이 나지 않았음
- 클러스터가 제외된 IVI이기 때문에 시간 지연에 대해 상대적으로 덜 민감

도입시 예상되는 장점

- 표준화 된 API로 다른 차량 및 다른 OS를 사용하더라도 VIS를 적용한 환경이라면 모두 사용 가능
- 기존 차량용 데이터는 CAN통신을 하므로 바이트 단위의 데이터로 구성되어 있으나, VIS 이용 시 차량 데이터는 high level의 형태를 띄기 때문에 개발자 입장에서 용이

6.2 오토모티브 웹 API 도입 - VIS

완성된 시스템의 실제 데이터 흐름



6.3 통합 콕핏 시스템의 가상화

DEVIEW 2021

컨테이너의 도입

- 요구사항의 분석 결과 AVN과 PD의 기능은 독립적으로 동작하는 것들이 대부분
- 서로 격리한다면 시스템 복잡도가 상당부분 낮아짐
- 안정된 하이퍼바이저는 대부분 유료이며, 구동시 오버헤드 비용이 비쌈
- IVI(In-Vehicle Infotainment) 시스템은 클러스터보다 안전에 상대적으로 덜 민감
- 서로 다른 OS를 사용하는 경우가 아니라면 컨테이너가 더 적합

6.3 통합 콕핏 시스템의 가상화

LXC를 이용한 가상화

- 공유 장치를 제외한 나머지는 완벽한 격리

ex) 디스플레이 : 디스플레이는 커널 디바이스 드라이버 레벨에서부터 격리
오디오 : 스피커별 장치 할당은 큰 의미가 되지 않으므로 서로 공유

- 실행 환경 등도 완전 격리화 : 앱 레벨이 아닌 프레임워크 레이어부터 격리
- CPU나 메모리도 컨테이너별 중요도에 따른 성능 상한 설정

컨테이너 격리를 위한 모듈 개발

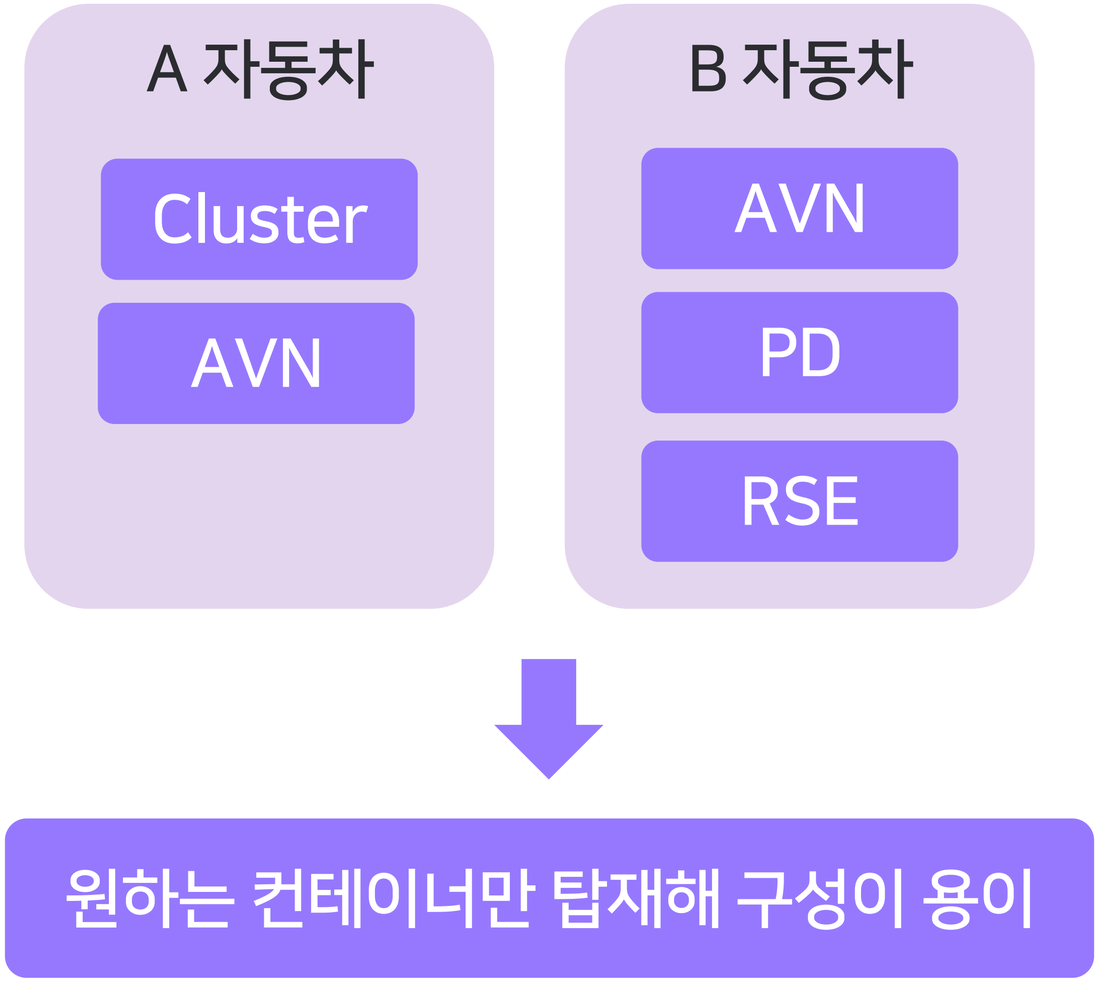
- 컨테이너간 통신할 수 있는 커뮤니케이션 채널
- 호스트 - 컨테이너간 브릿지 모듈

6.3 통합 콕핏 시스템의 가상화

DEVIEW 2021

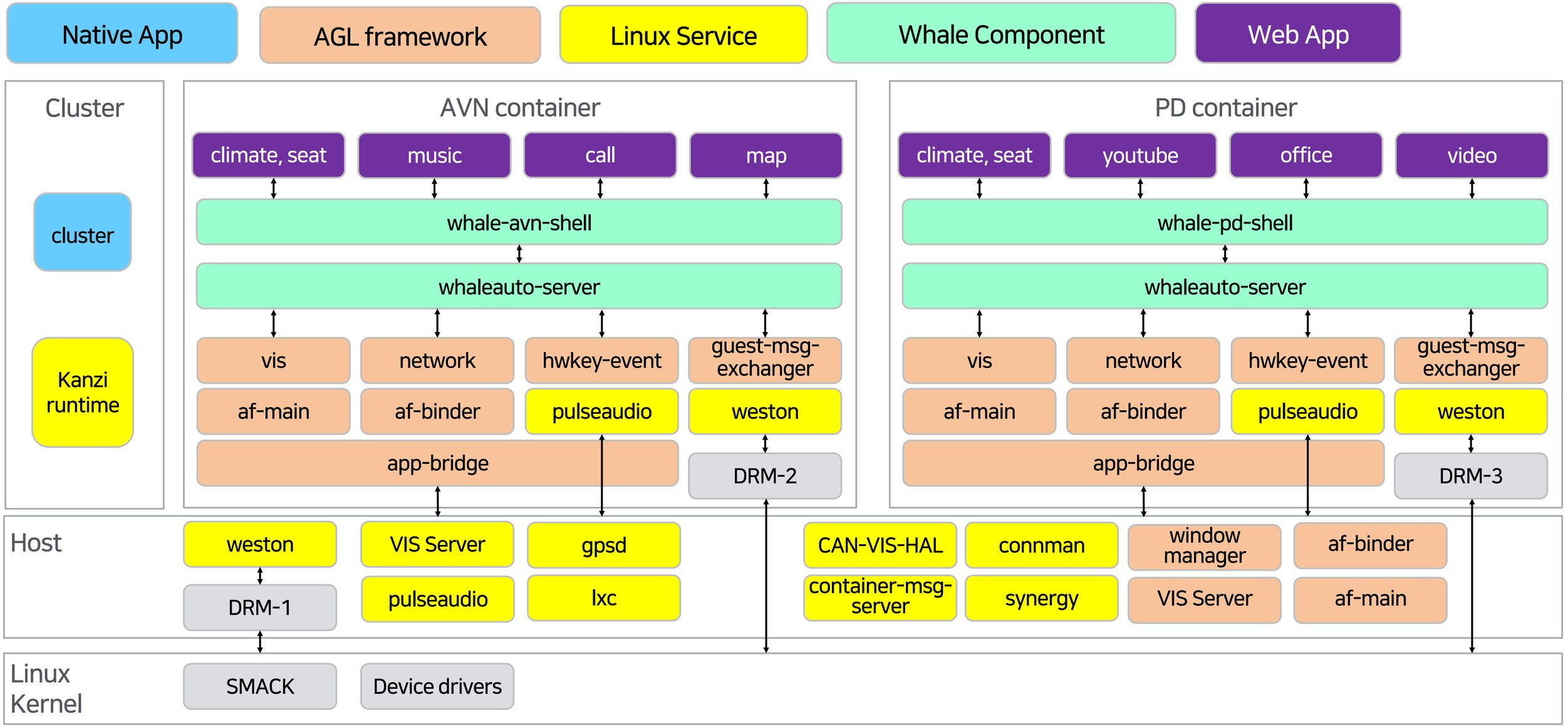
가상화 장점

- 디버깅, 로깅이 편리해짐
- 다양한 구성이 쉬워짐



RSE: Rear Seat Entertainment(뒷좌석 모니터)

6.4 통합 콕핏 시스템 구조도



6.5 통합 콕핏의 실제 구동 사진

6.6 TO-DO

1. 웨일 및 AGL의 버전 최신화
2. 더욱 다양한 기능과 경험 제공 (ex. 웨일 브라우저의 웨일 ON)
3. Ecosystem 구축

Thank You