IPLI

Image Processing Living Intelligence

SCH IPL

IPLI 작품 설명서

Image Processing Living Intelligence

문제현황

"계산대 줄이 너무 길어 아이스크림 하나만 먼저 계산하겠다는 제가 진상인가요?"

박아영 기자 · ayoung@insight.co.kr 입력 : 2019.05.13 12:06 가- 가+

인사 등 계산 전 통과 의례에 드는 시간은 평균 41초, 물건 하나가 계산대를 통과하는 시간은 약 3초로 물건을 하나만 구매하는 사람이 10명 있더라도, 10번째 고객은 물건 하나를 구매하기 위해 약 8분 정도를 대기해야 한다.

문제현황

제 역할 못하고 불평·불만으로 가득한 무인계산대

김선찬 기자 | 승인 2019.06.27 18:42 | 댓글 0

"무인계산대 꼭 필요하나요?"

무인화 시대에 발맞춰 현재 도내 40여 대가 운영중인 무인계산대가 설치 의도와 다르게 실효성에 대한 지적을 받고 있다.

무인계산대는 시간이 촉박하거나 소량의 상품만을 구매하고자 할 때 일반계산대의 긴 줄을 피해 신속하게 구매가 가능 하도록 설치됐다.

일반계산대 축소로 인해 근무자들의 근무 강도가 높아졌을 뿐만 아니라 대기 시간 증가에 따른 고객들의 불만·불평도 덩달아 높아지는 이중적인 부작용이 발생하고 있는 셈이다.

일반 계산대에서 계산을 한 박진수(25)씨는 "아무리 소량일지라도 일일이 계산하는 것이 귀찮다"면서 "무인계산대보다 오히려 계산원에게 계산하는 것이 더 편하다"고 말했다.

또한, 무인계산대라는 말과 다르게 직원이 계산대 주변에 상주하면서 기계 사용법에 익숙치 않은 노인 등을 대신 계산을 해주는 모습도 눈에 띄였다.

고객들의 편리성과 효율성을 위해 존재하고 있는 무인계산대지만 겉과 속이 다른 실정인 셈이다.

일반계산대의 운영 개수가 줄어든 탓에 고객들이 계산을 하기 위해 대기하는 시간만 증가하는 부용도 적지 않았다.

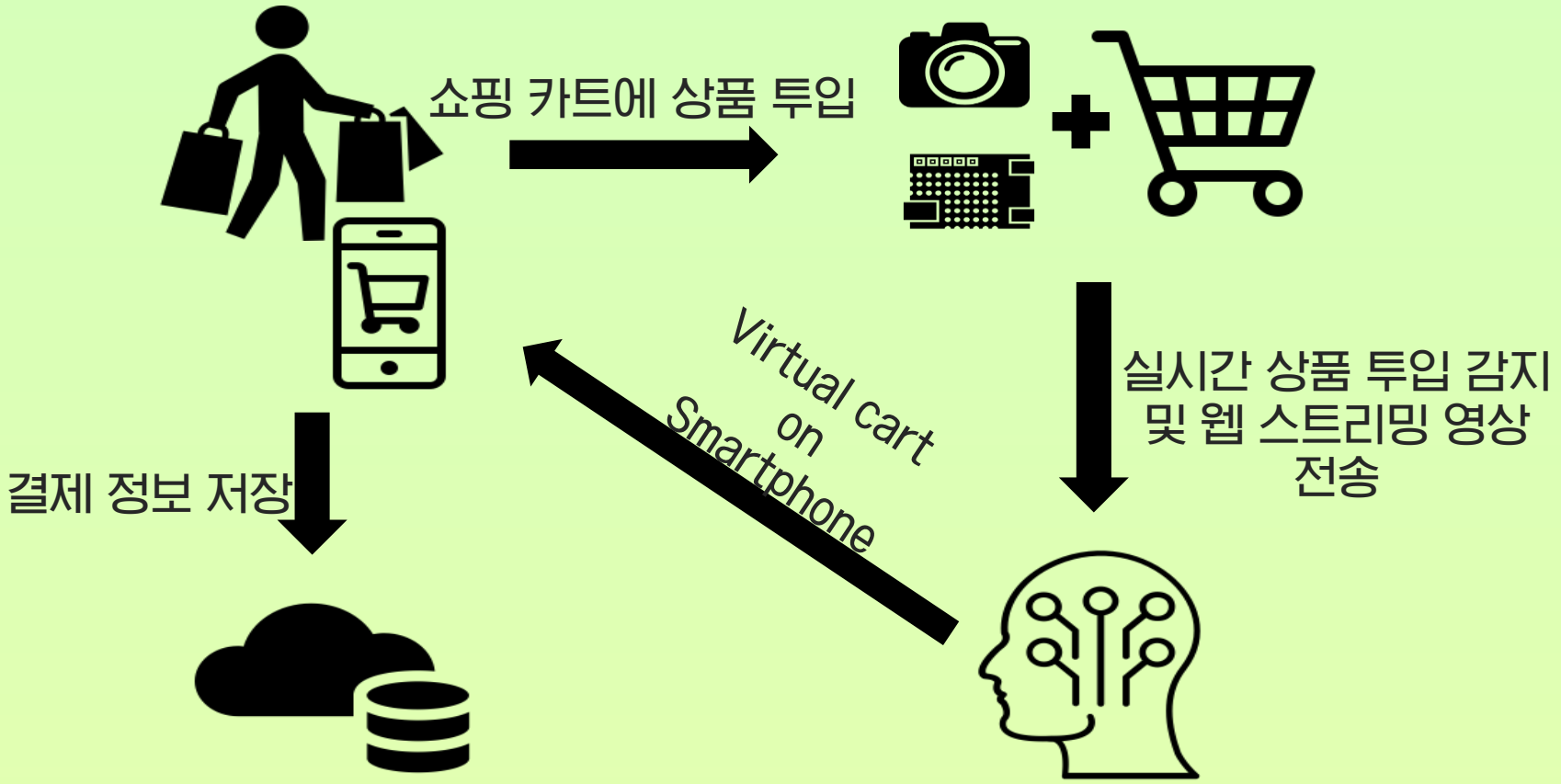
기존에 나와있는 무인 계산 시스템은 교육 받은 숙달된 직원이 하는 일을 고객이 셀프로 하도록 전환되었을 뿐이라 본래의 목적과는 다르게 더 지체되고 불편할 수 밖에 없다.
이러한 문제들은 더 직관적인 인터페이스 결제 시스템의 필요성을 대두한다.

시나리오

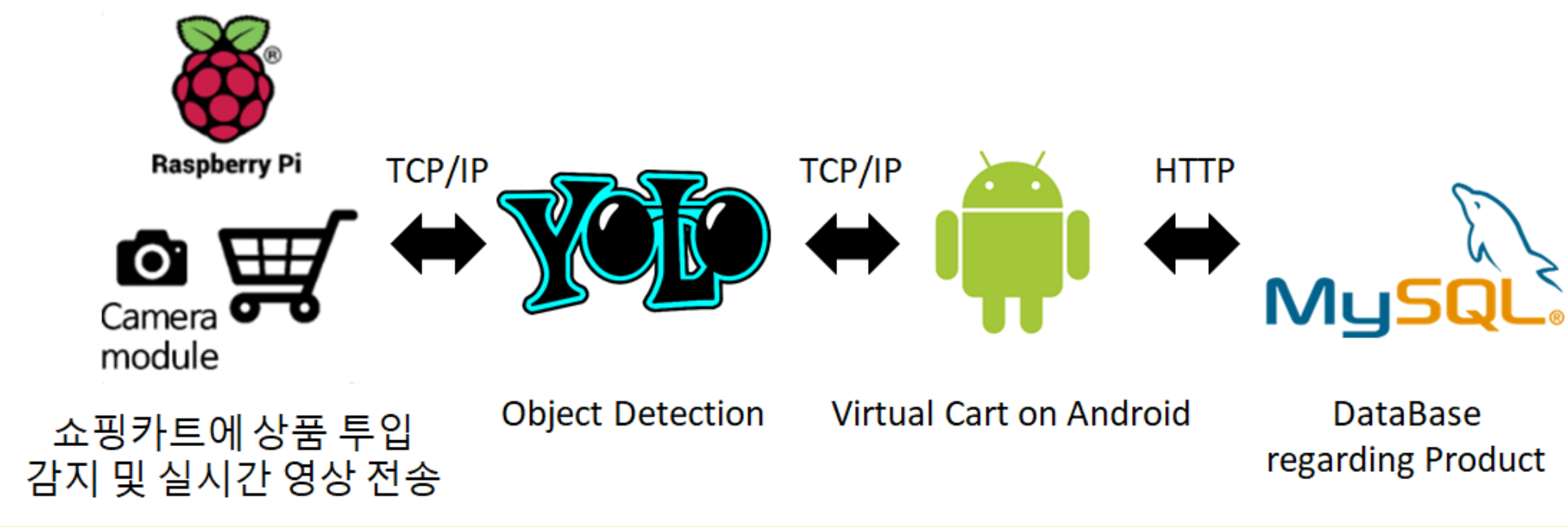
“ 국민 생활에 지능을 더하다. ”

1. A군이 장을 보러 마트에 간다.
2. 충전 중인 쇼핑카트를 빼내어 쇼핑을 시작한다.
3. 구입할 물건을 쇼핑카트에 담는다.
4. 쇼핑카트에 부착된 카메라로부터 획득한 영상을 서버에 전달한다. 서버의 기 학습된 object detection 모델을 통해 상품이 인식된다.
5. 가상의 장바구니에 해당 상품을 추가하고 가격을 산정한다.
6. 3 ~ 5 반복
7. 장바구니를 스마트폰으로 확인 후 결제하고 쇼핑카트를 원래 위치에 두고 가게를 나선다.
8. 쇼핑카트는 충전이 시작된다.

시스템 구성도


쇼핑 카트에 상품 투입
실시간 상품 투입 감지
및 웹 스트리밍 영상
전송
Virtual cart
on
Smartphone
결제 정보 저장

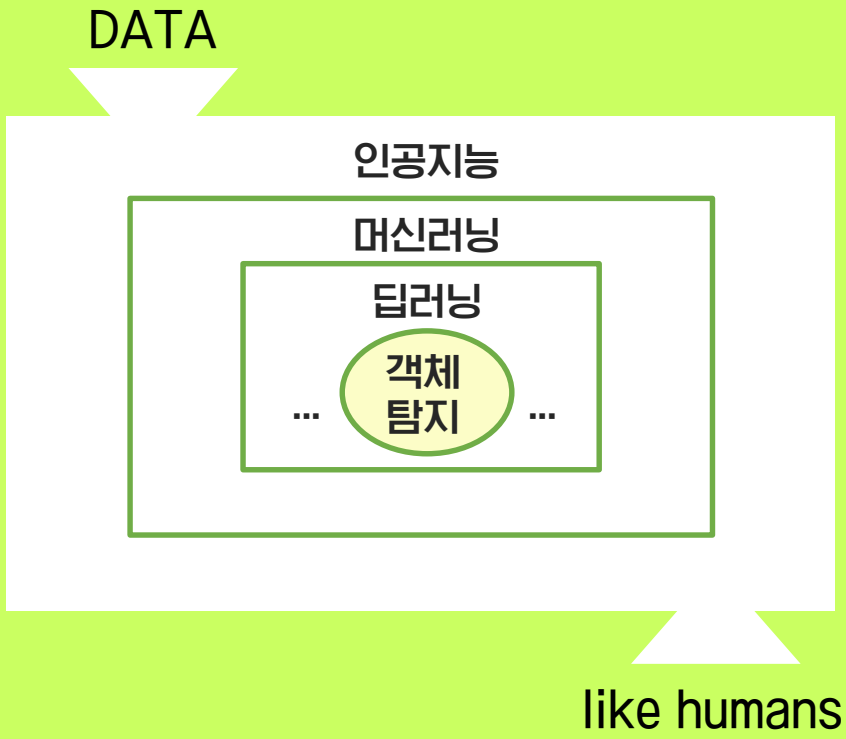
시스템 구성도


Raspberry Pi
Camera module
TCP/IP
TCP/IP
HTTP
쇼핑카트에 상품 투입
감지 및 실시간 영상 전송
Object Detection
Virtual Cart on Android
DataBase
regarding Product

Part1. Server

Image Processing Living Intelligence

Object Detection이란?



Machine Learning : 어떤 입력을 기반으로 (인간다운) 특정 반응을 하도록 하는 일종의 s/w

Object Detection이란?

한 이미지 내 여러 객체를 검출 할 수 있음

Object Detection = Classificaion + Localization

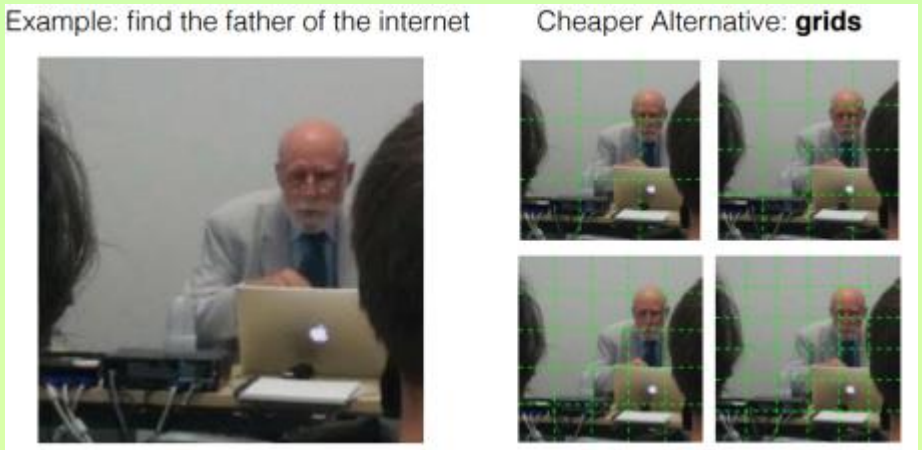
YOLO(You Only Look Once)는 이전까지의 객체 탐지 신경망보다 우수한 성능을 제공하며, 파일 형식으로의 입력 뿐 아니라 웹 캠을 통한 입력에 대해서도 실시간 감지까지 지원하는 강력한 실시간 객체 탐지 시스템

YOLO(You Only Look Once)

이전 object detection 시스템
Proposal 방식

Selective Search 후 Edge Boxes 단계를 거치는 방식(R-CNN)은 CNN 방식보다는 빠른 성능을 보이나, Selective Search 후 반환되는 수 많은 후보 ROI 추정 영역에 대한 경계 박스 위치와 특징데이터를 가지고 분류를 하므로 큰 오버헤드

YOLO object detection 시스템
Grid 방식

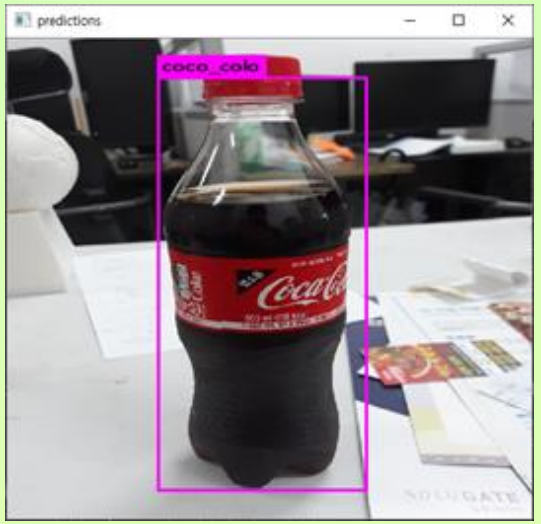


Grid 방식은 grid cell의 개수가 곧 proposal의 수로 후보를 구하는 과정에 대한 오버헤드 전무. R-CNN보다 약 1000배, Fast R-CNN보다 약 100배, Faster R-CNN보다도 약 10배 이상 빠른 성능

YOLO(You Only Look Once)

이미지 grid cell 분할

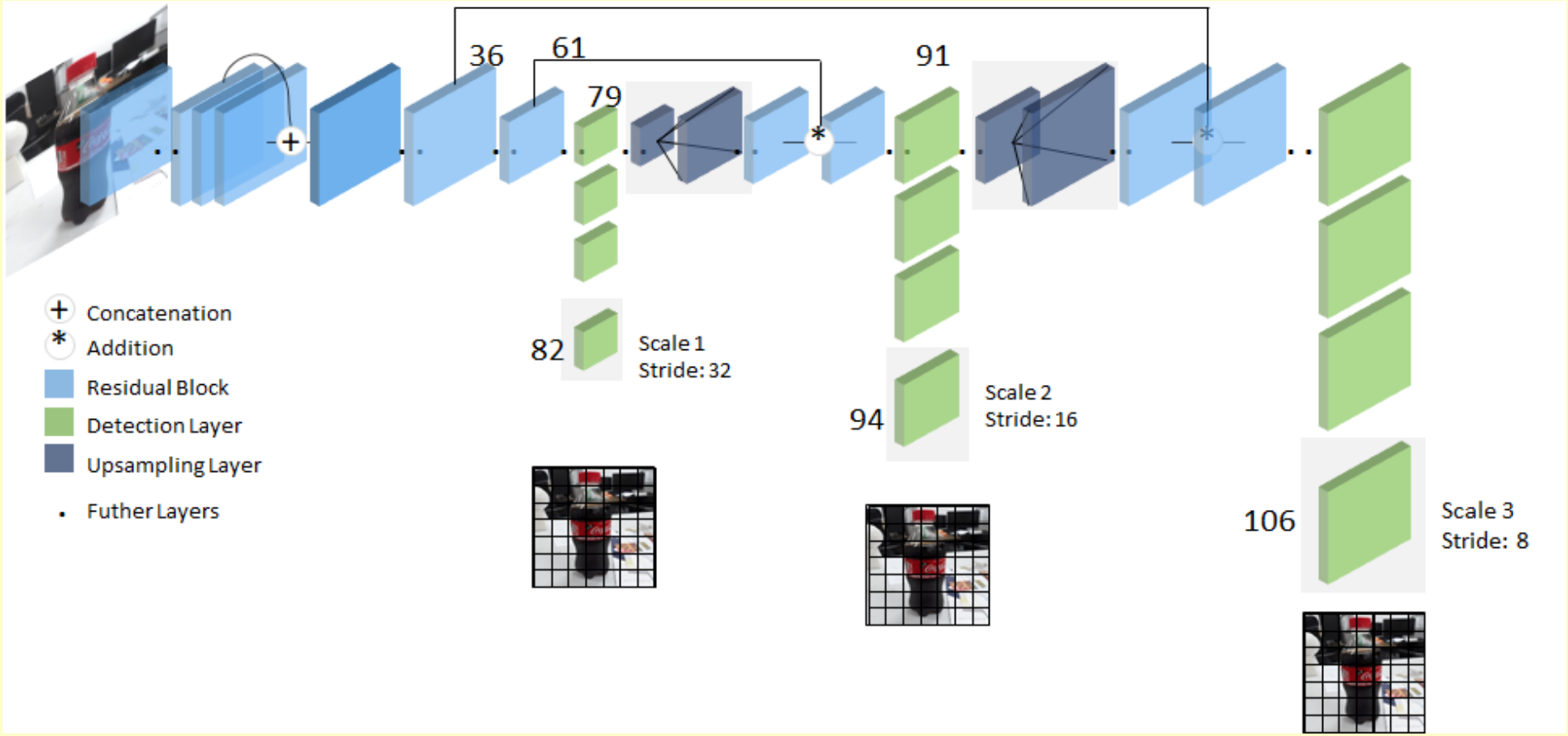
경계박스 생성



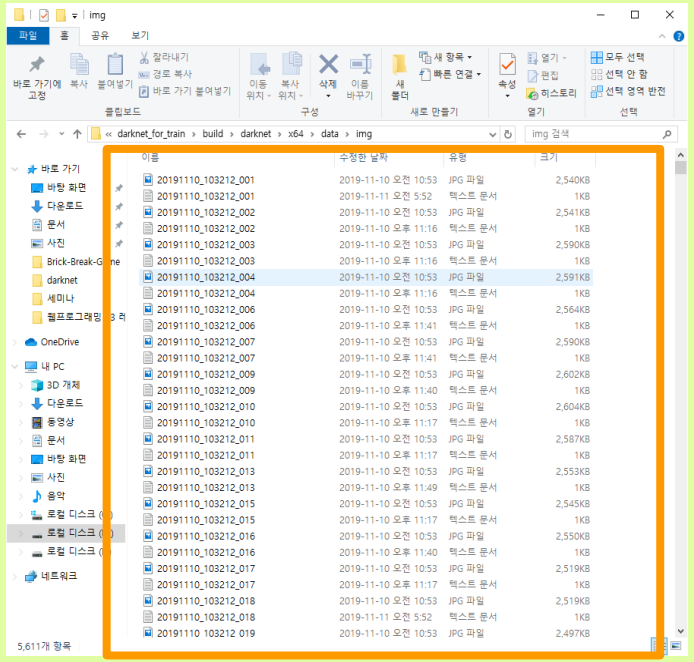
경계박스별 score를 계산해
thresh_hold 이상인
경계박스만 출력

YOLO(You Only Look Once)

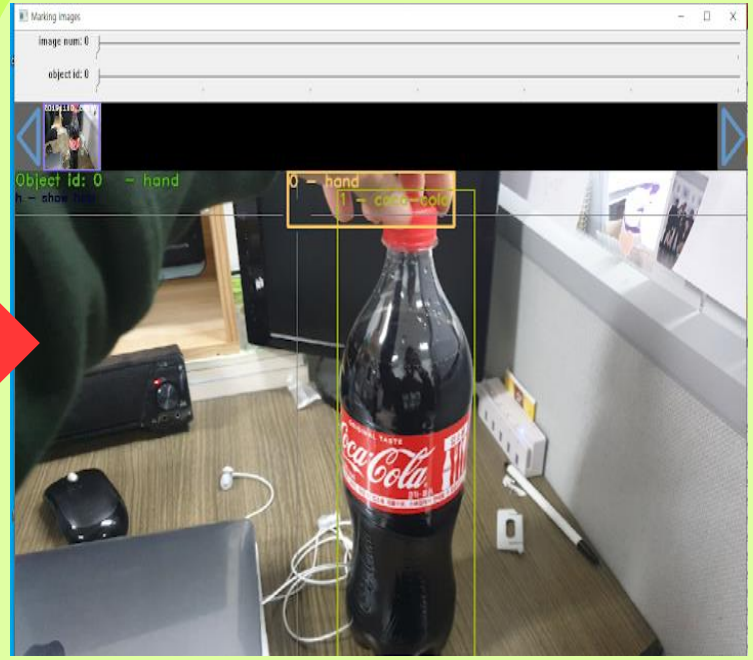
입력 이미지의 크기를 점차 다운 샘플링하여 컨볼루션 기반 아키텍처를 제공



데이터 전처리

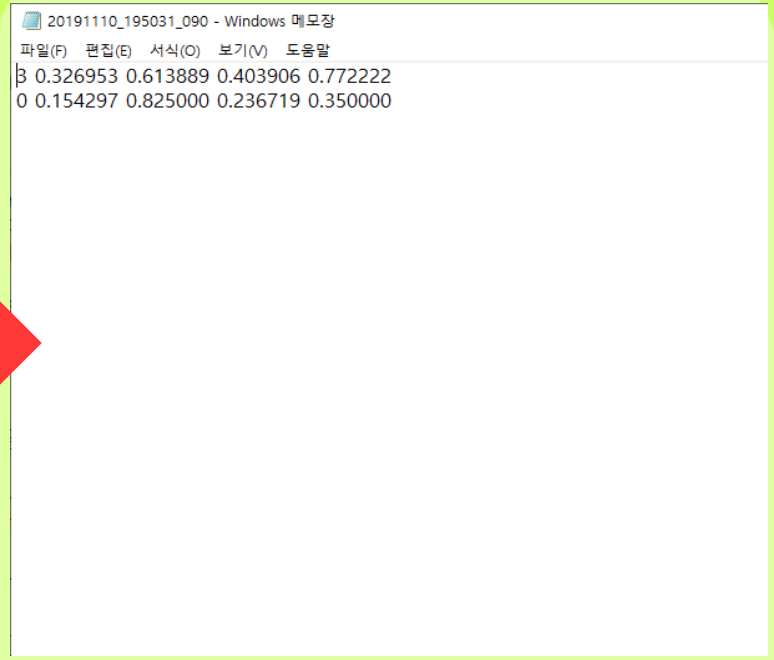


약 2800장의 훈련용 데이터를 수집



데이터 전처리

흥미 영역 외 불필요한 정보를
차단해주는 단계

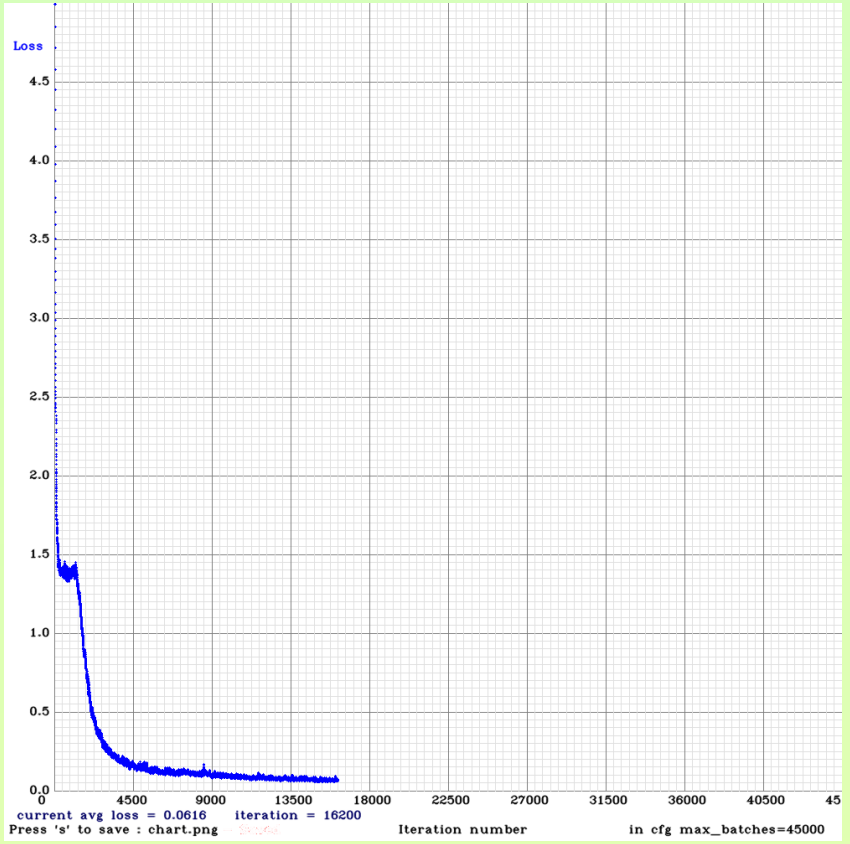


전처리가 완료된 데이터들은 객체의 id,
경계박스의 중앙 x좌표와 y좌표 그리고 width,
height 정보가 저장되게 한다.

데이터 학습

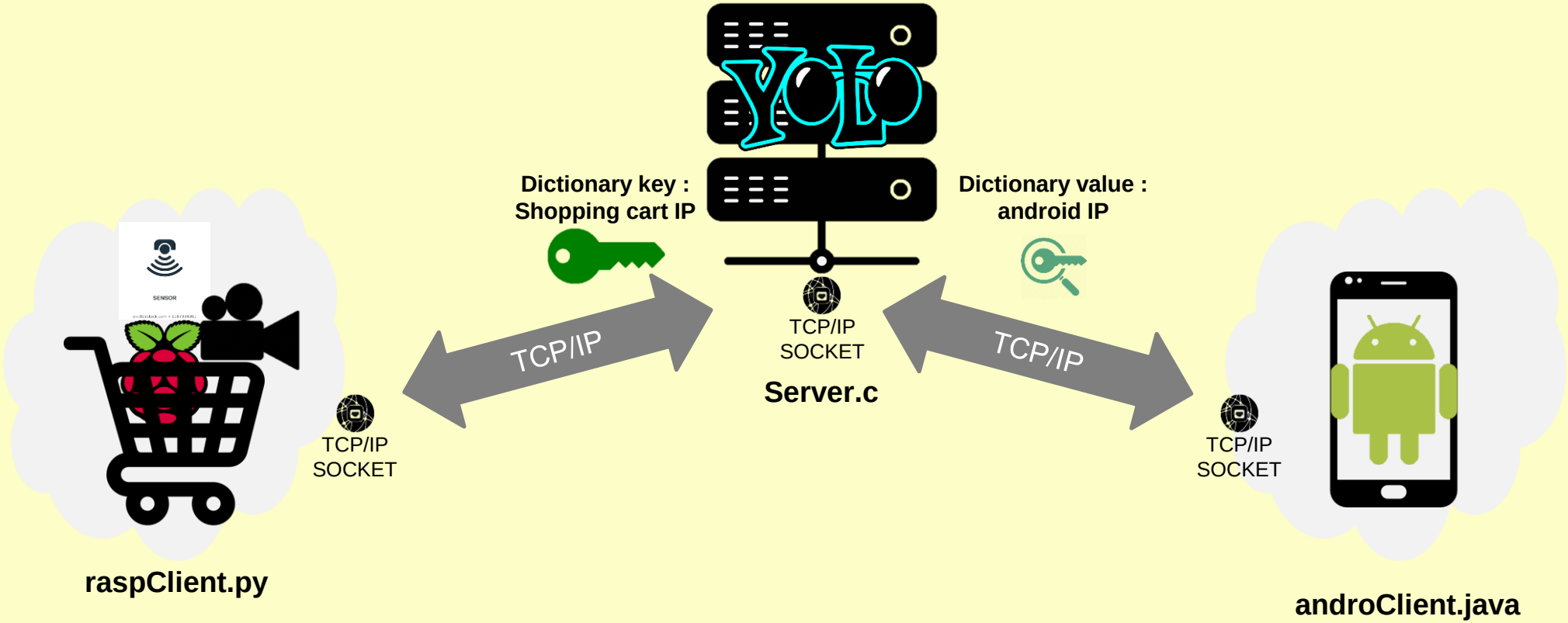
```
명령 프롬프트 - darknet  detector train data/obj.data yolo-obj.cfg data/darknet53.conv.74
Region Avg IOU: 0.598589, Class: 0.166812, Obj: 0.495958, No Obj: 0.496307, Avg Recall: 0.769231,  count: 13
Region Avg IOU: 0.580498, Class: 0.166899, Obj: 0.495939, No Obj: 0.496307, Avg Recall: 0.750000,  count: 12

 Tensor Cores are used.
 4: 15.094563, 15.449919 avg loss, 0.000100 rate, 1.333000 seconds, 256 images
Loaded: 5.037000 seconds
Region Avg IOU: 0.497888, Class: 0.166800, Obj: 0.493836, No Obj: 0.493989, Avg Recall: 0.384615,  count: 13
Region Avg IOU: 0.614703, Class: 0.166759, Obj: 0.493793, No Obj: 0.493989, Avg Recall: 0.812500,  count: 16
Region Avg IOU: 0.619374, Class: 0.166856, Obj: 0.493674, No Obj: 0.493989, Avg Recall: 0.846154,  count: 13
Region Avg IOU: 0.526368, Class: 0.166807, Obj: 0.493933, No Obj: 0.493989, Avg Recall: 0.533333,  count: 15
Region Avg IOU: 0.646920, Class: 0.166674, Obj: 0.493731, No Obj: 0.493989, Avg Recall: 0.857143,  count: 14
Region Avg IOU: 0.542126, Class: 0.166869, Obj: 0.493792, No Obj: 0.493989, Avg Recall: 0.500000,  count: 14
Region Avg IOU: 0.591509, Class: 0.166857, Obj: 0.493779, No Obj: 0.493989, Avg Recall: 0.625000,  count: 16
Region Avg IOU: 0.576640, Class: 0.166765, Obj: 0.493531, No Obj: 0.493989, Avg Recall: 0.583333,  count: 12

 Tensor Cores are used.
 5: 15.013884, 15.406315 avg loss, 0.000100 rate, 1.373000 seconds, 320 images
Loaded: 4.763000 seconds
Region Avg IOU: 0.612982, Class: 0.166787, Obj: 0.490992, No Obj: 0.491252, Avg Recall: 0.666667,  count: 15
Region Avg IOU: 0.637801, Class: 0.166682, Obj: 0.491020, No Obj: 0.491252, Avg Recall: 0.692308,  count: 13
Region Avg IOU: 0.550853, Class: 0.166841, Obj: 0.491034, No Obj: 0.491252, Avg Recall: 0.538462,  count: 13
Region Avg IOU: 0.557801, Class: 0.166712, Obj: 0.491121, No Obj: 0.491252, Avg Recall: 0.538462,  count: 13
Region Avg IOU: 0.564899, Class: 0.166785, Obj: 0.490968, No Obj: 0.491252, Avg Recall: 0.615385,  count: 13
Region Avg IOU: 0.564680, Class: 0.166752, Obj: 0.490953, No Obj: 0.491252, Avg Recall: 0.538462,  count: 13
Region Avg IOU: 0.543331, Class: 0.166772, Obj: 0.491123, No Obj: 0.491252, Avg Recall: 0.666667,  count: 15
Region Avg IOU: 0.595207, Class: 0.166845, Obj: 0.491135, No Obj: 0.491252, Avg Recall: 0.666667,  count: 15

 Tensor Cores are used.
 6: 14.795505, 15.345234 avg loss, 0.000100 rate, 1.287000 seconds, 384 images
```

학습이 진행중인 상태, Loss 함수 값이
단계적 weight가 갱신됨에 따라 점차적으로 감소



16200번 학습을 진행하였을 때의 Loss함수의 그래프

TCP/IP 네트워크 구성

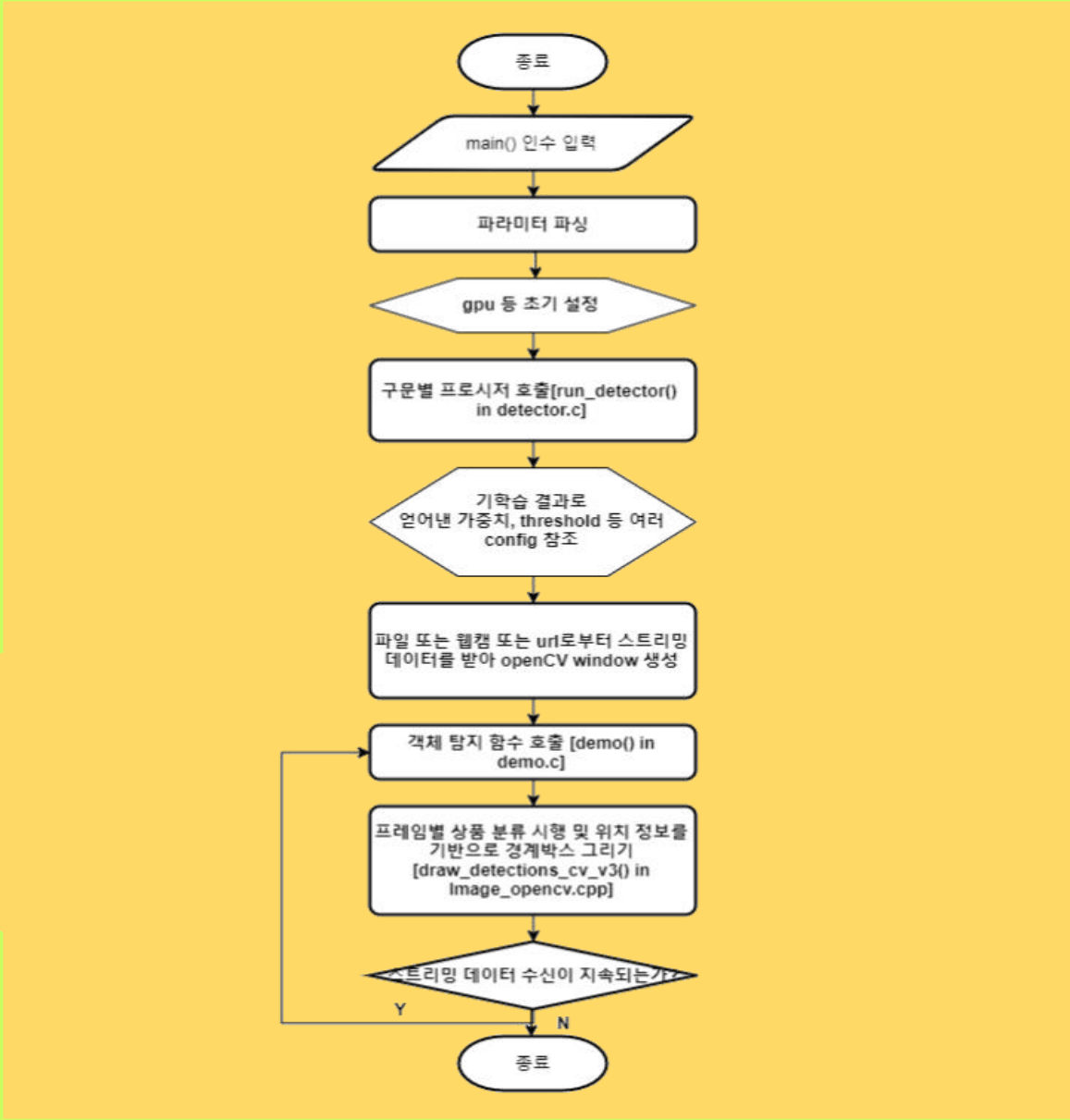


- 쇼핑카드 클라이언트, 안드로이드 클라이언트와의 통신 모두 각각의 스레드를 생성해줌으로써 다중 통신을 지원
- 최초 쇼핑 시나리오에서 사용자의 스마트폰 UI를 통해 연결할 쇼핑 카트의 IP를 입력 받아 서버에 전달해줌으로써 해당 쇼핑카드와 스마트폰을 매핑
- 매핑은 해시 자료구조를 응용

Original YOLO

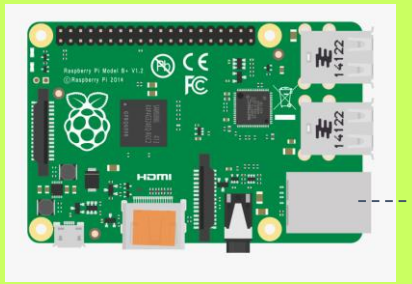
"darknet.exe detector demo data/obj.data data/yolo-obj.cfg yolo-obj_last.weights [영상 스트리밍 제공 url]"

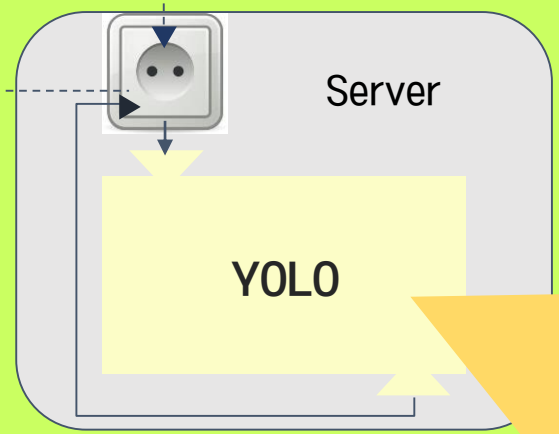
YOLO



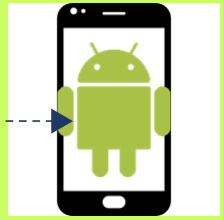
Socket Programming + Original YOLO

"darknet.exe detector demo data/obj.data data/yolo-obj.cfg yolo-obj_last.weights http://'+cart_ip+':8091/?action=stream"

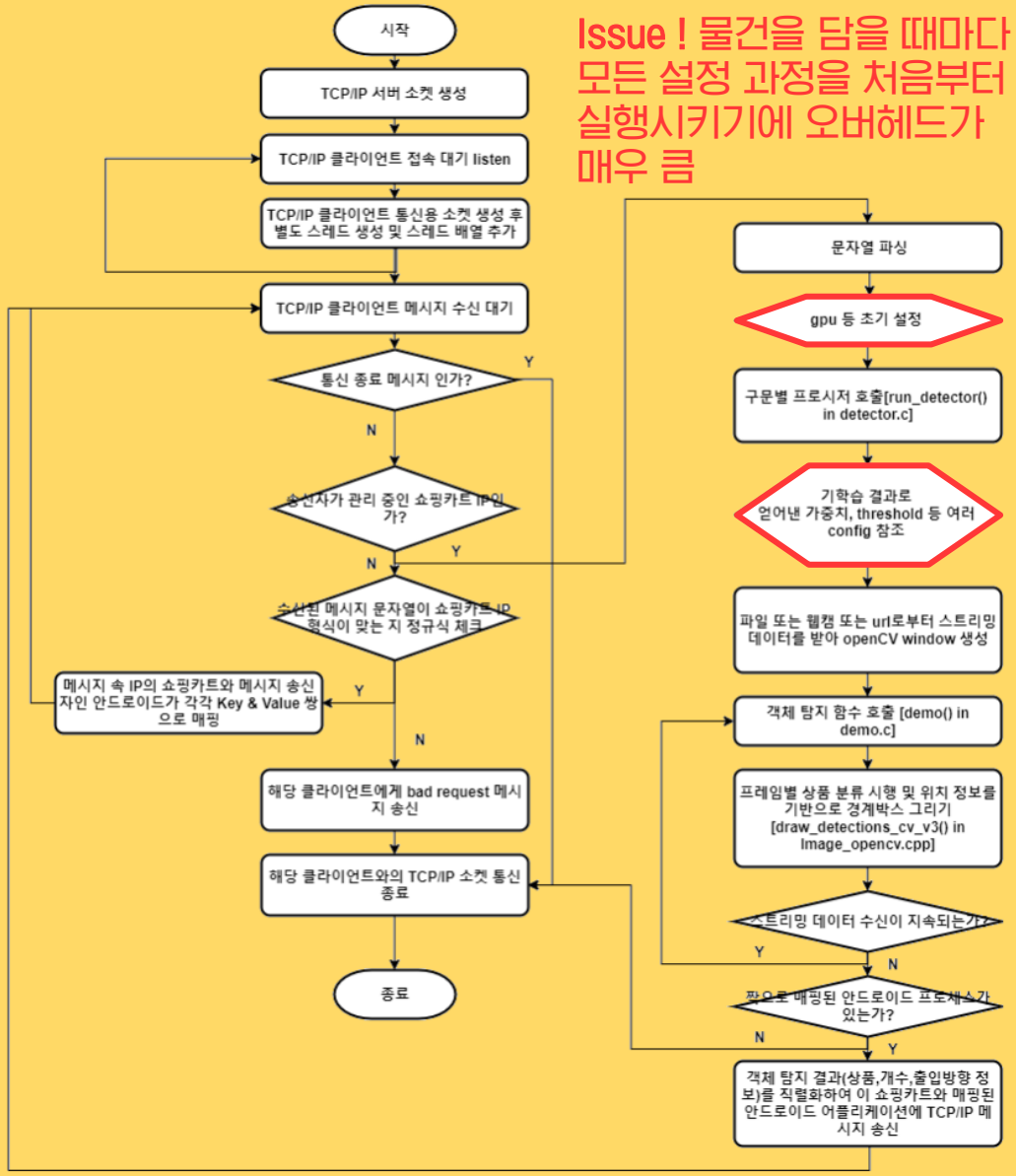




상품id@상품개수@출입방향[+|-];

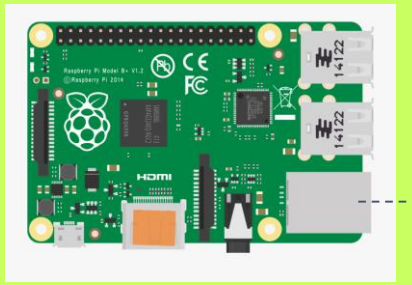


Issue ! 물건을 담을 때마다 모든 설정 과정을 처음부터 실행시키기에 오버헤드가 매우 큼

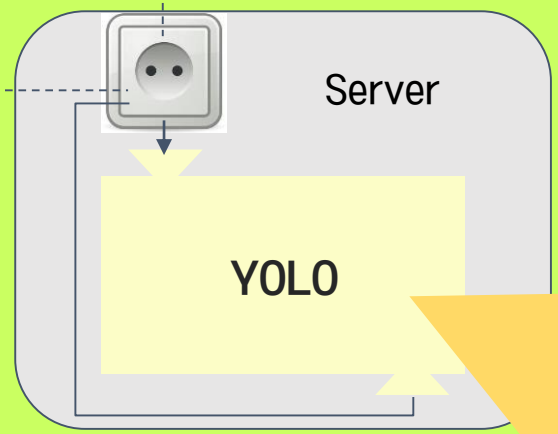


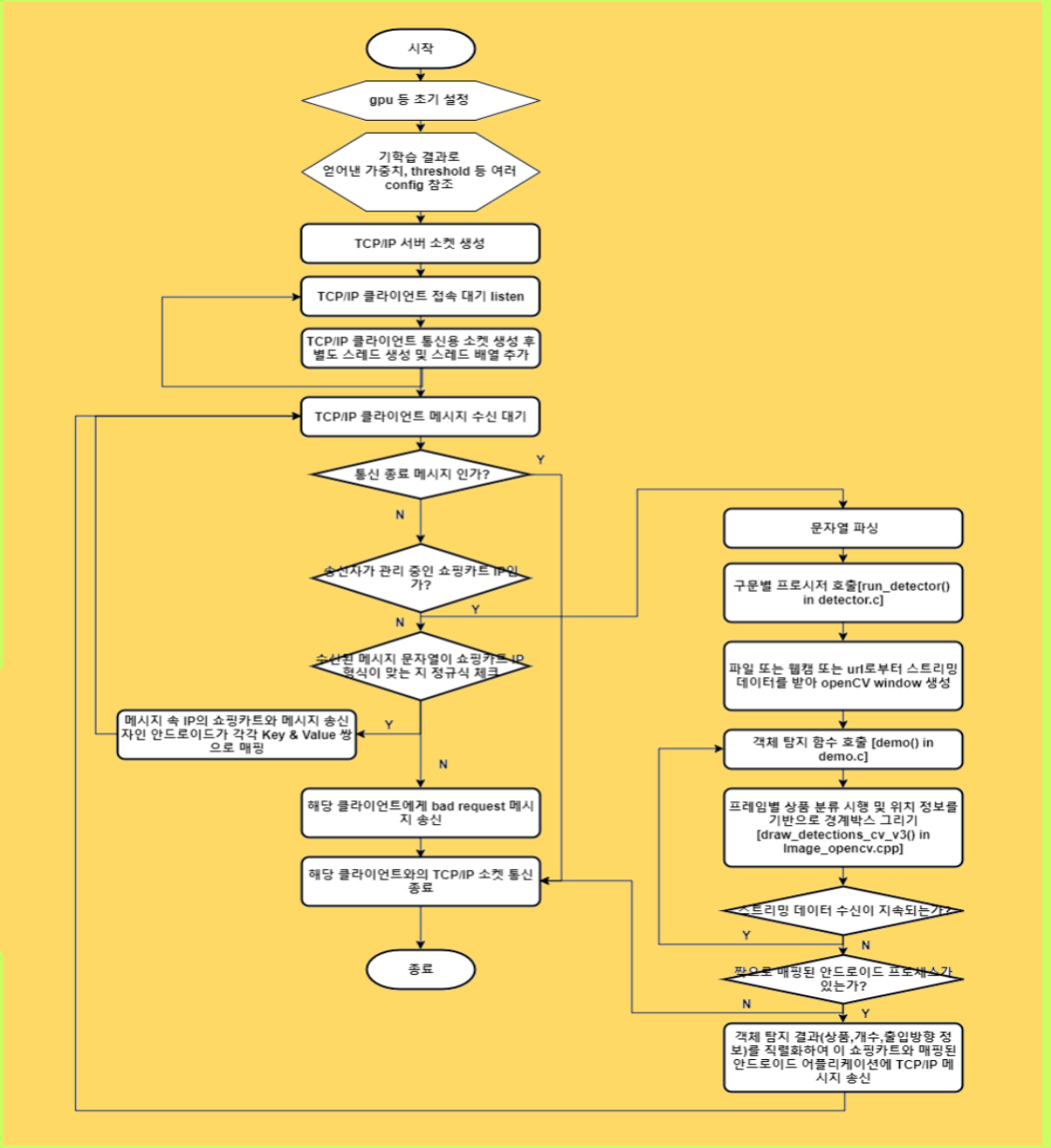
최적화

"darknet.exe detector demo data/obj.data data/yolo-obj.cfg yolo-obj_last.weights http://'+cart_ip+':8091/?action=stream"



상품id@상품개수@출입방향[+|-];





Part2. H/W on cart

Image Processing Living Intelligence

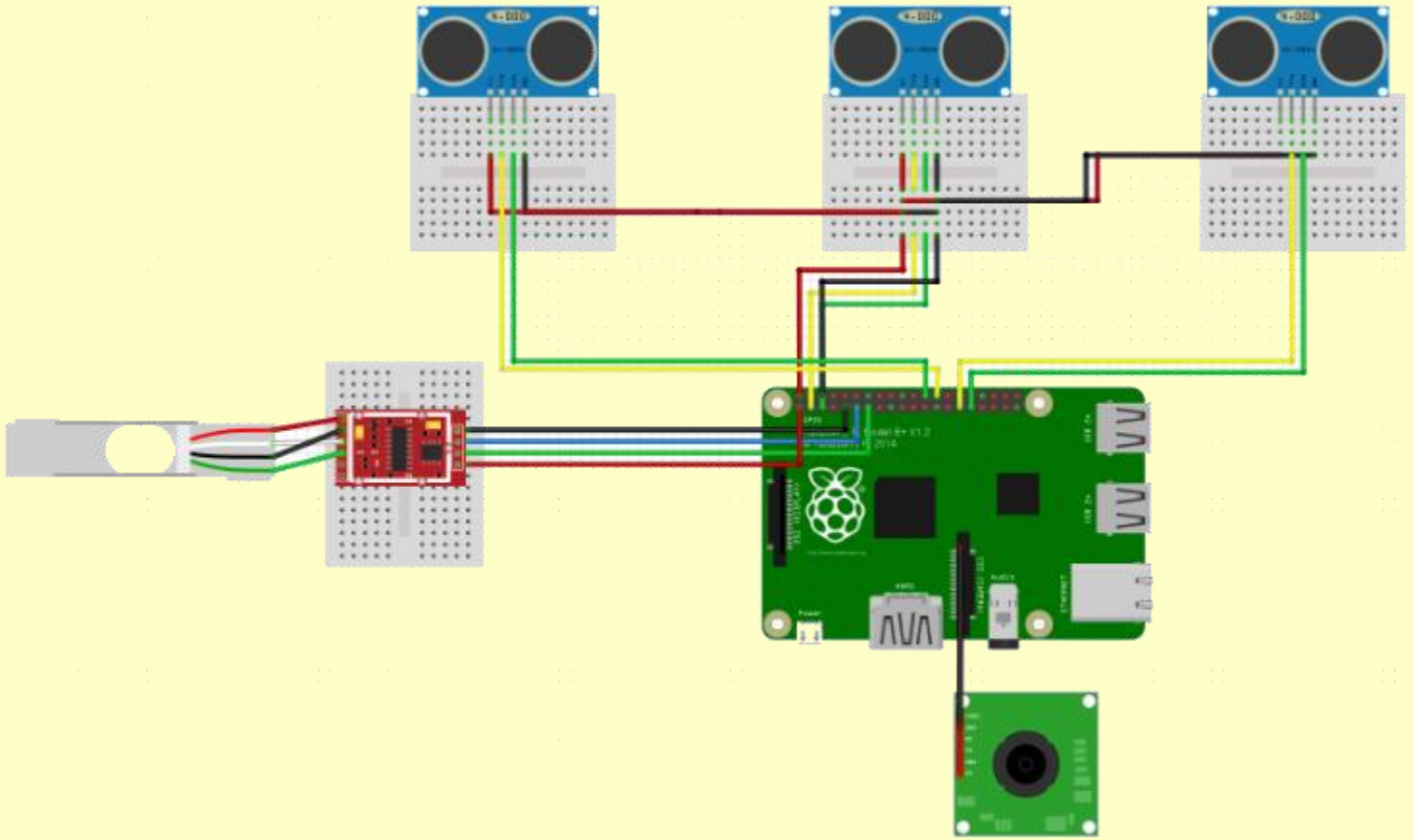
사용 H/W 모듈 정보

사용한 H/W 모듈	기능
Raspberry Pi	- Raspberry Pi 는 대표적 Micro Processor Unit 보드 중 하나로 IoT 시스템을 구현함에 있어서 영상 전송에 적합한 소형 컴퓨팅 장치로 널리 사용 - Raspberry Pi 3b+에 내장 WiFi NIC를 이용하여 Internet 망에 접속
파이카메라	- Raspberry Pi와 함께 흔히 사용되는 Raspberry Pi 전용 카메라 모듈 - 본 프로젝트에서는 MJPG Streamer를 활용하여 웹으로 영상을 스트리밍 하기 위한 H/W 인터페이스 장치로서 사용
초음파 거리센서	- 쇼핑 카트 내 상품 출입을 감지하기 위한 트리거 모듈로 사용
무게센서	- 물체가 들어오고 나가는 것을 더욱 정밀하게 판별하기 위하여 사용

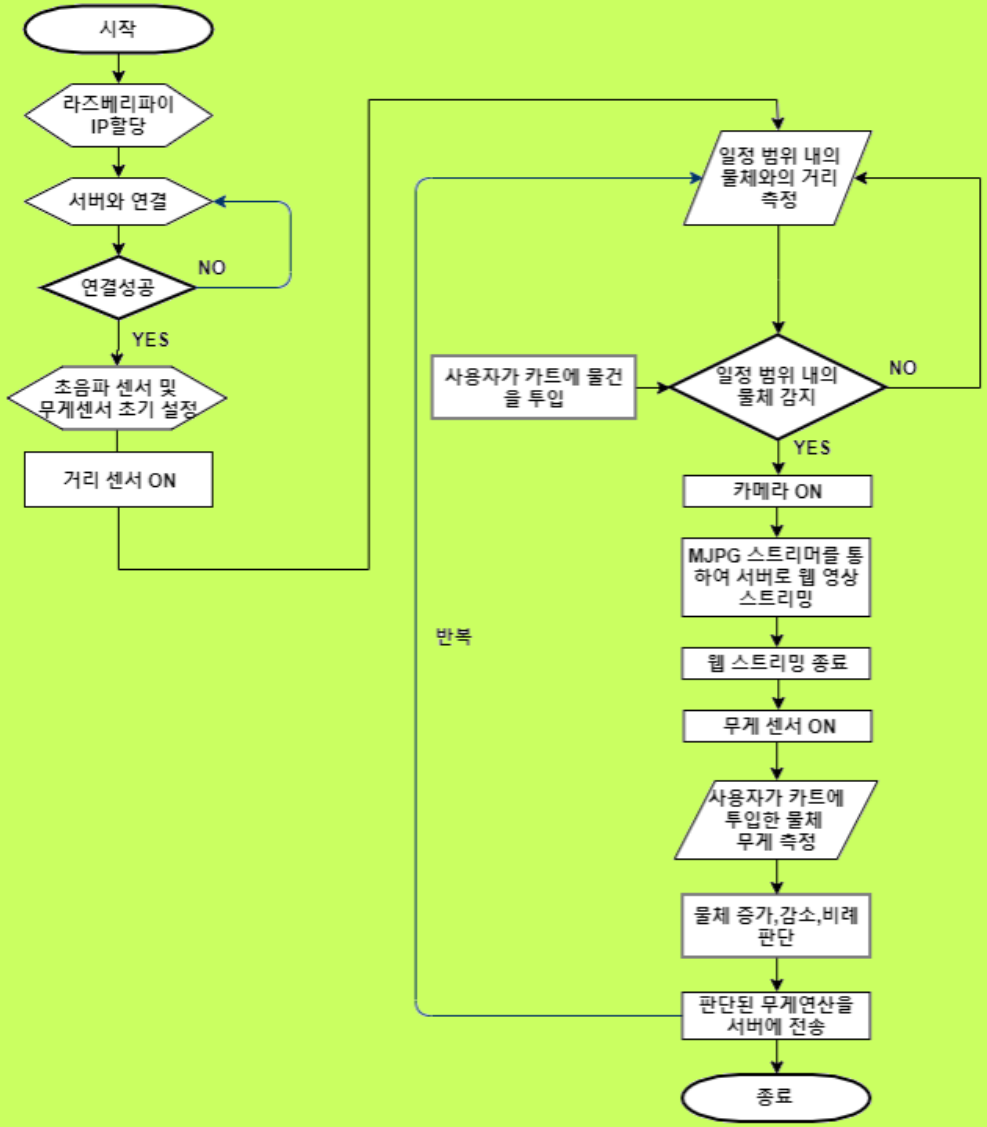
사용 S/W 모듈 정보

사용한 S/W 모듈	기능
Raspbian OS	- 저전력을 목표로 한 ARM 아키텍처 사용 - Raspberry pi에서 지원하는 대표적인 OS - 무선배터리를 활용하여 부팅 시 모듈 내 프로그램 자동 실행
Python	- 유연한 언어를 제공 - 각종 라이브러리를 방대하게 지원 - Raspberry pi 에서의 접근성이 좋음
Mjpg-streamer	- jpg 촬영을 연속으로 수행해서 마치 영상처럼 보이도록 해주는 솔루션(MJPG) - 일련의 설치 과정을 거치면 특정 port 로 접근할 수 있는 HTTP Server 역할을 제공 - 이 페이지를 통해 MJPG영상을 볼 수 있도록 해주기 때문에 브라우저로 접속하거나, URL을 통한 접근이 가능 - Raspberry pi에서 영상 스트리밍 데이터를 C로 구현된 서버까지 전달하기에 용이

H/W 회로도



H/W Flow Chart



MOCKUP MODEL

물건을 담는 이벤트에 대하여
거리센서 트리거가 발생

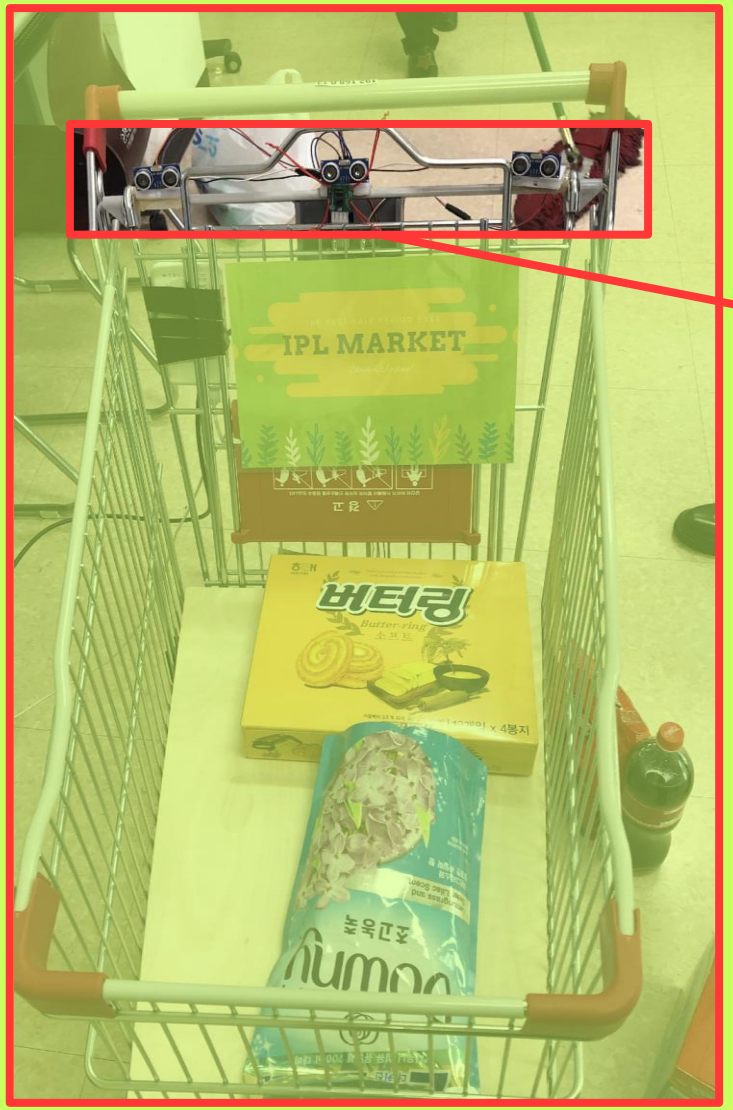
RaspberryPi에 부착된
Picam과 MJPG Streamer를
이용하여 영상 웹
스트리밍을 시작

서버에게 스트리밍이
시작되었음을 tcp/ip 소켓을
이용하여 NOTIFY

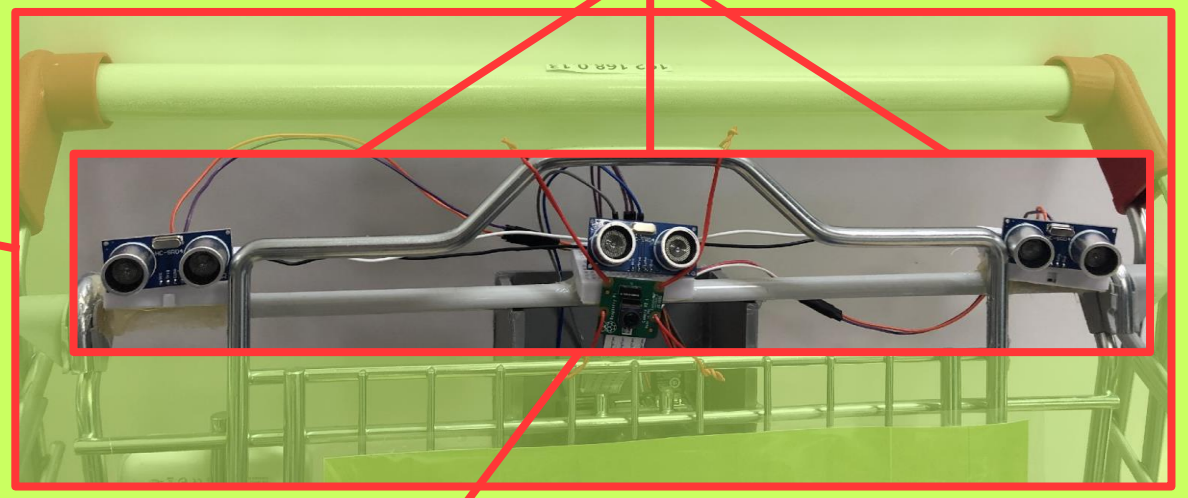
배터리 소모 최소화

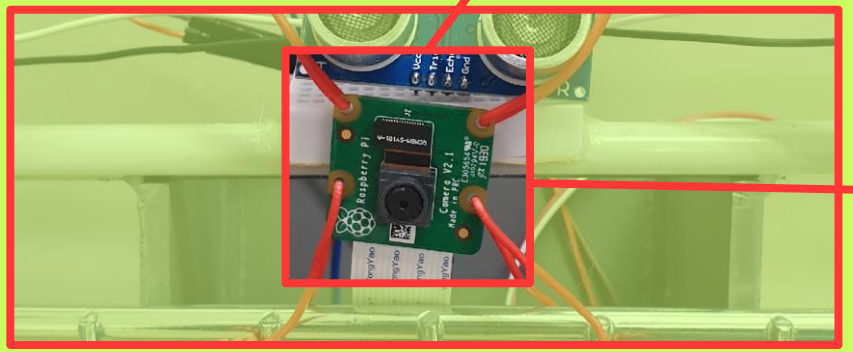
MOCKUP MODEL

- 거리센서 / PI-CAM



초음파 거리센서





파이카메라

MOCKUP MODEL

웹 영상 스트리밍 종료

물체가 들어왔는지 나갔는지를
정밀하게 판단하기 위해
이전의 무게와 현재 무게를
비교하여 증가, 감소, 비례를
판단

무게의 증가, 감소, 비례 판단
정보 전송의 정확도를 높이기
위해 3번의 무게측정

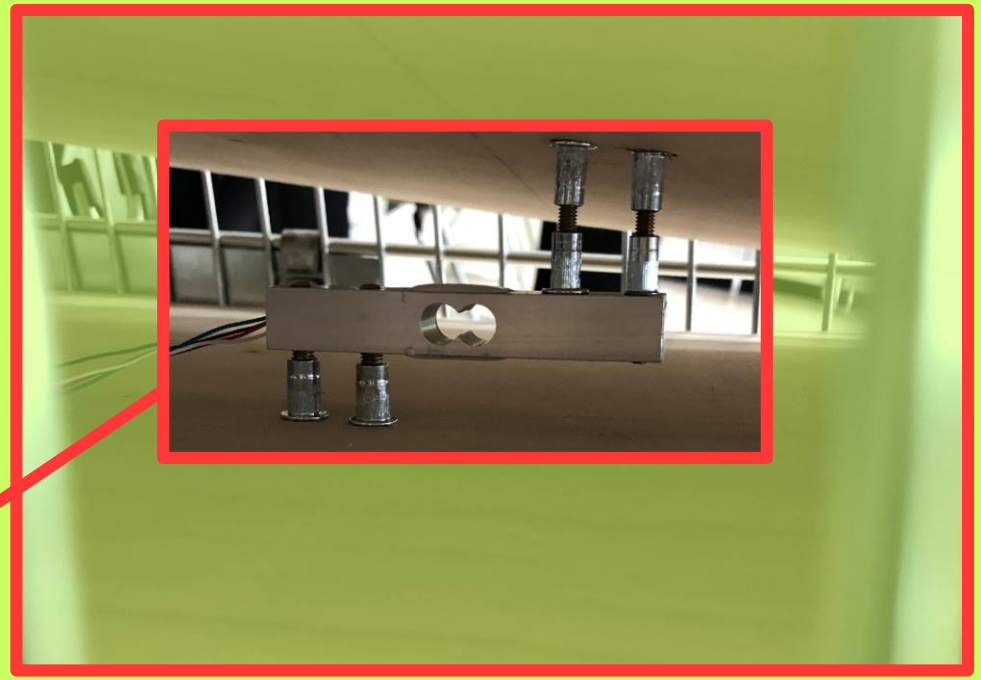
3번 반복

물체에 대한
정확도 증가

MOCKUP MODEL

- 무게센서(Load Cell)

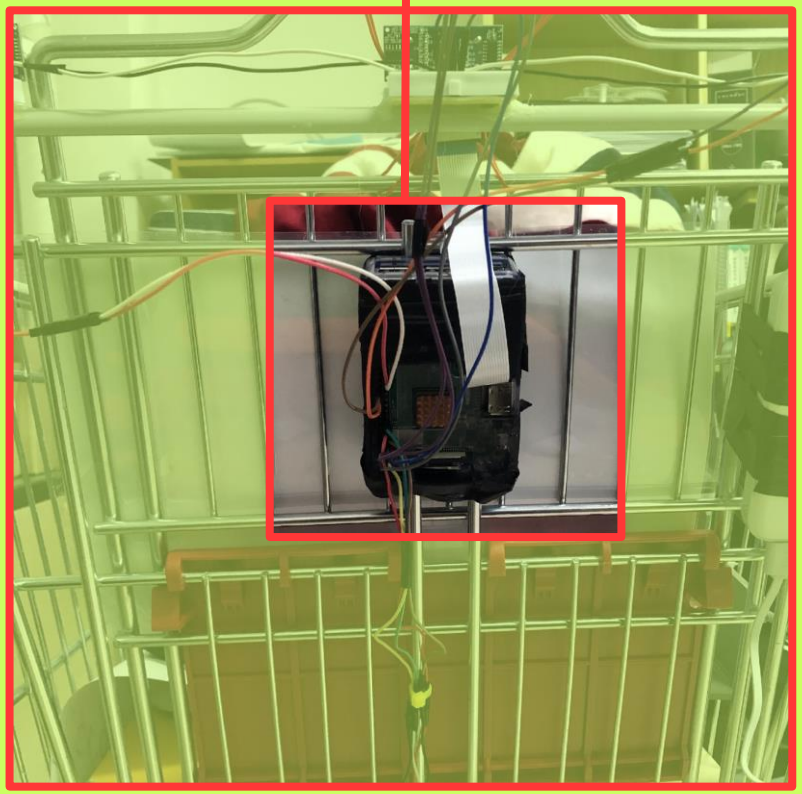
무게센서(Load Cell)

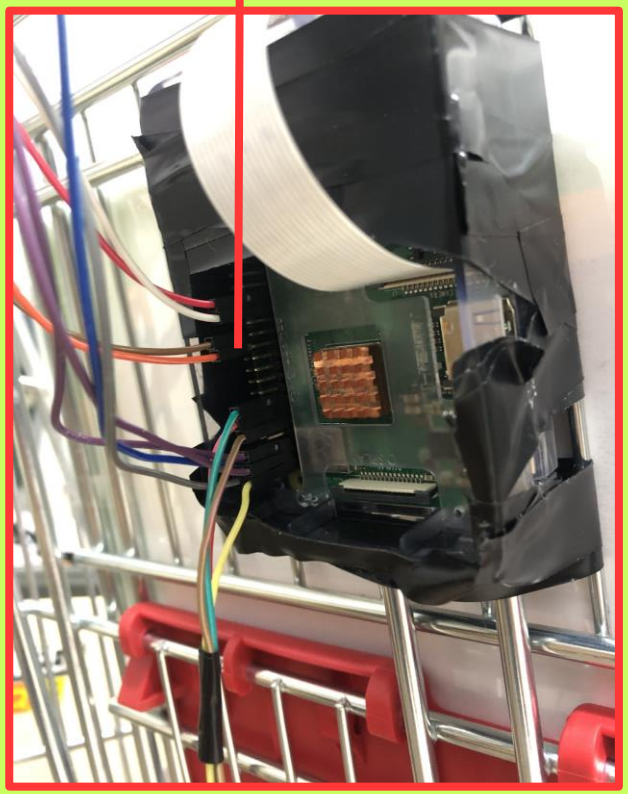


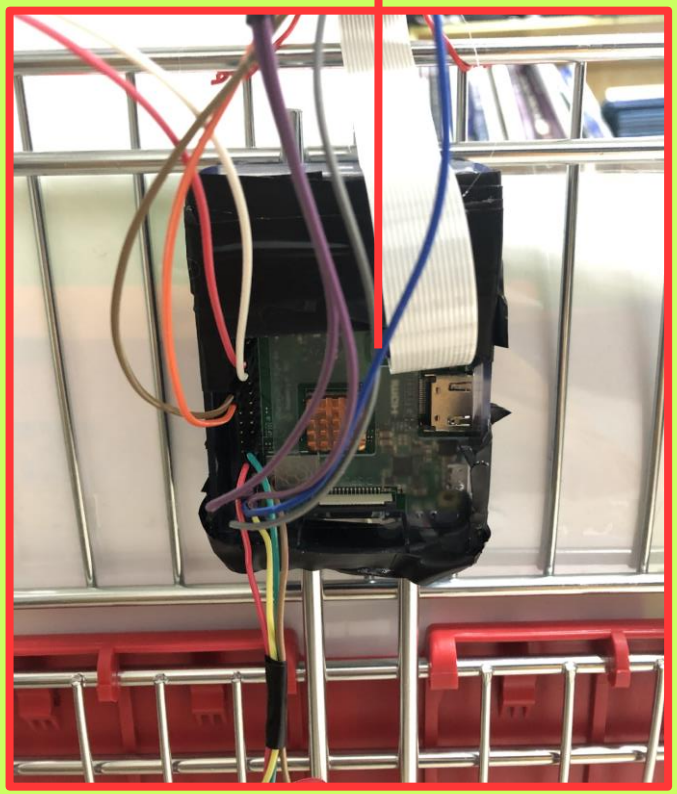
MOCKUP MODEL

- Raspberry Pi GPIO

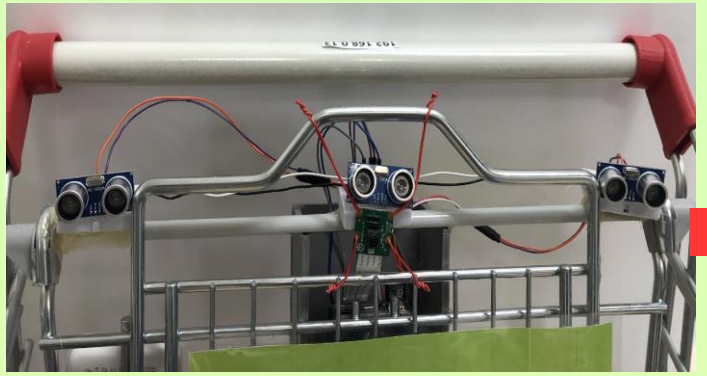
카트 뒷면
라즈베리파이와 센서 모듈간 GPIO 통신



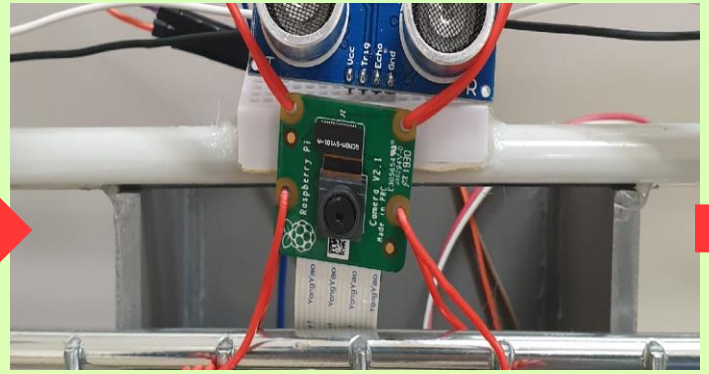




MOCKUP MODEL 작동 순서



초음파 거리 센서 설정 범위 내에
물체 인식 -> 트리거 발생



트리거 발생 -> 설정 시간동안
카메라 작동 후 종료

카메라 종료 -> 현재 담긴 물체의 무게
3번 측정

Part 3. Android

Image Processing Living Intelligence

Android Part

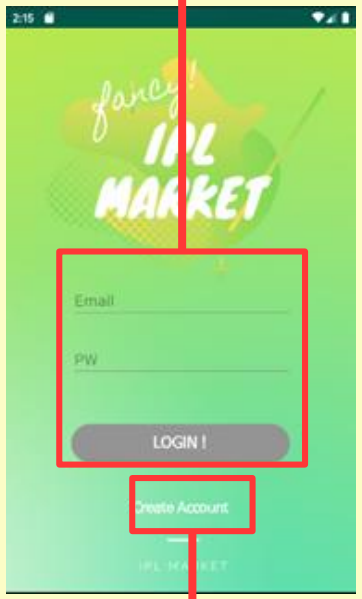
Android App에서의 UI를 제공

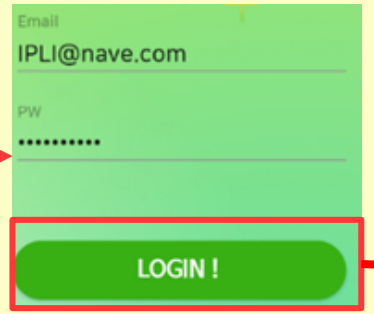
- ➔ Raspberry Pi에서 전송한 영상에 대해 서버가 인식한 상품의 레이블명과 개수, 출입 여부에 대한 직렬화된 Data를 Android 측으로 송신하면 그 물건을 DB에서 조회하여 Android의 가상 장바구니 안에 추가 또는 삭제하는 등 CRUD를 제공

- ➔ 셀프 결제와 관련된 결제 정보 관리를 총 망라하는 유저 관리를 제공

- ➔ 전 연령대가 사용하는 어플인 만큼 깔끔하고 단정한 디자인에 알아보기 쉽도록 하는 것이 디자인의 목적이며 어플명은 IPL 마켓이다.

UI 설계 (Login Interface)





계정을 입력하면
로그인 버튼 활성화

로그인 버튼 터치 후
메인 화면으로 이동

계정 정보를 모두 입력하면
계정생성 버튼 활성화

DB에 정보 저장

UI 설계 (Main Interface)

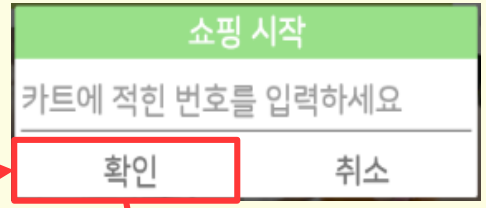
상단의 메뉴바를 누르면 다양한 엑티비티로 진입 가능

<기획전>

<메인화면>

<이벤트>

UI 설계 (Shopping Cart Interface)

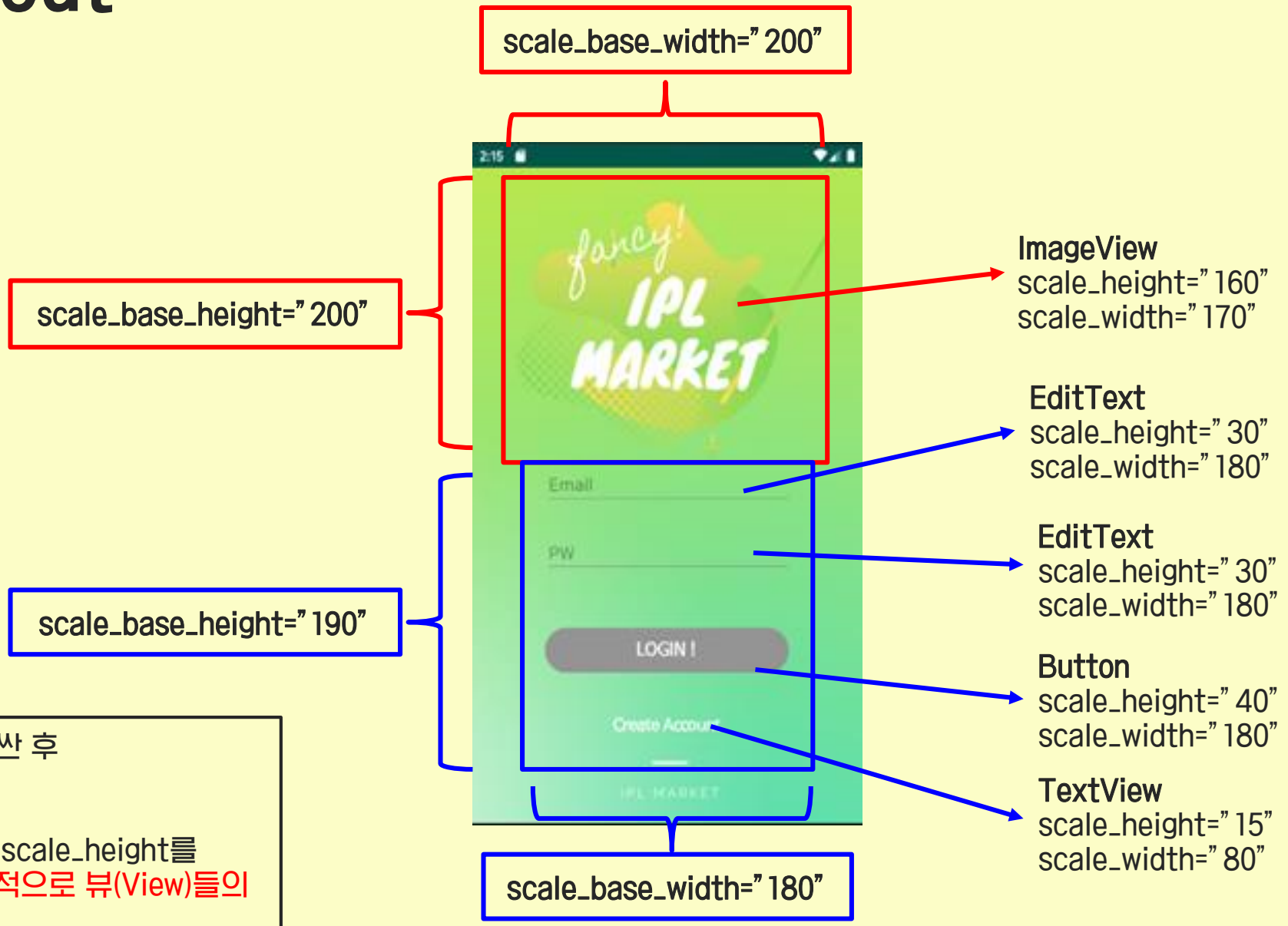


Scarable Layout

Scarable Layout 1

Scarable Layout 2

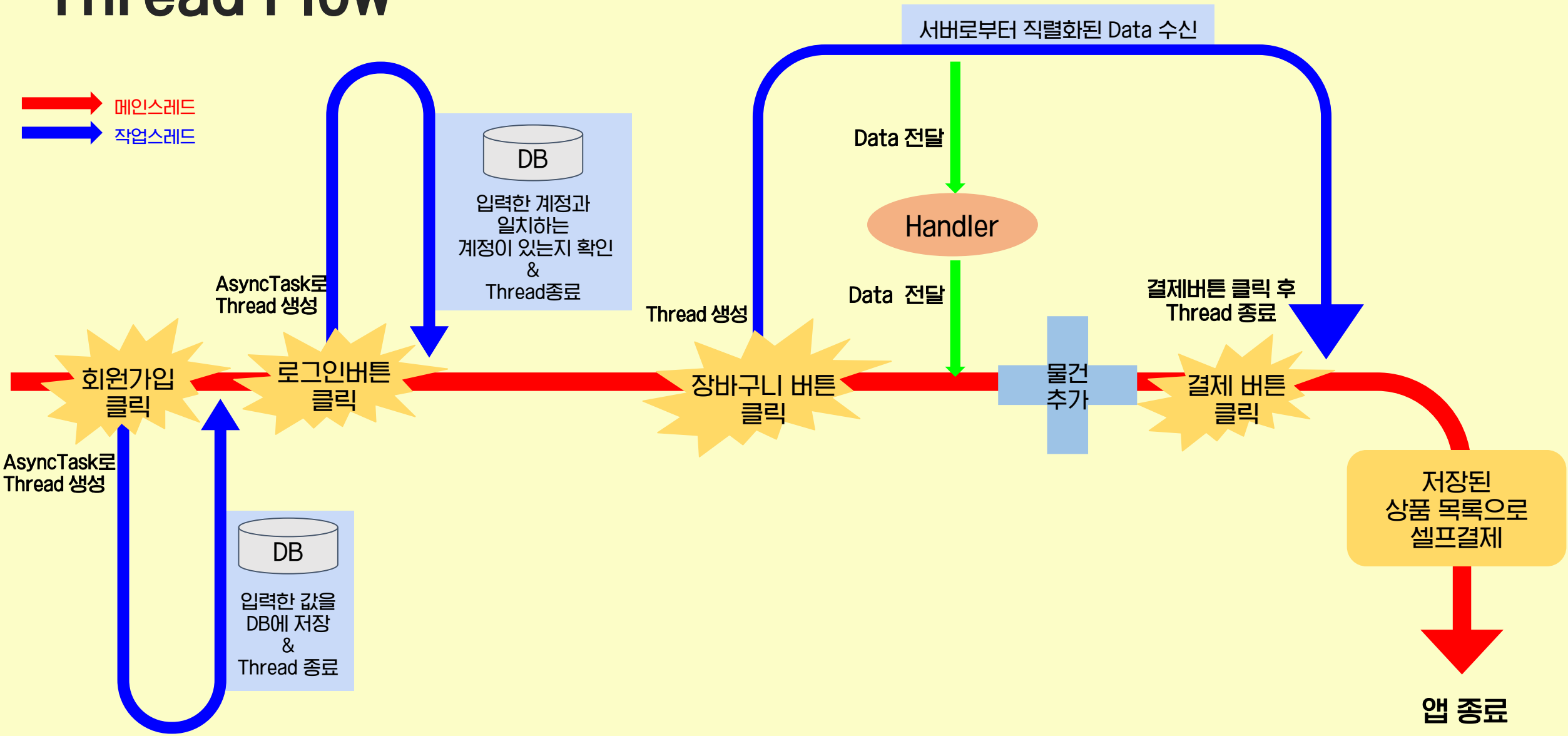
<Scarable Layout 예시>



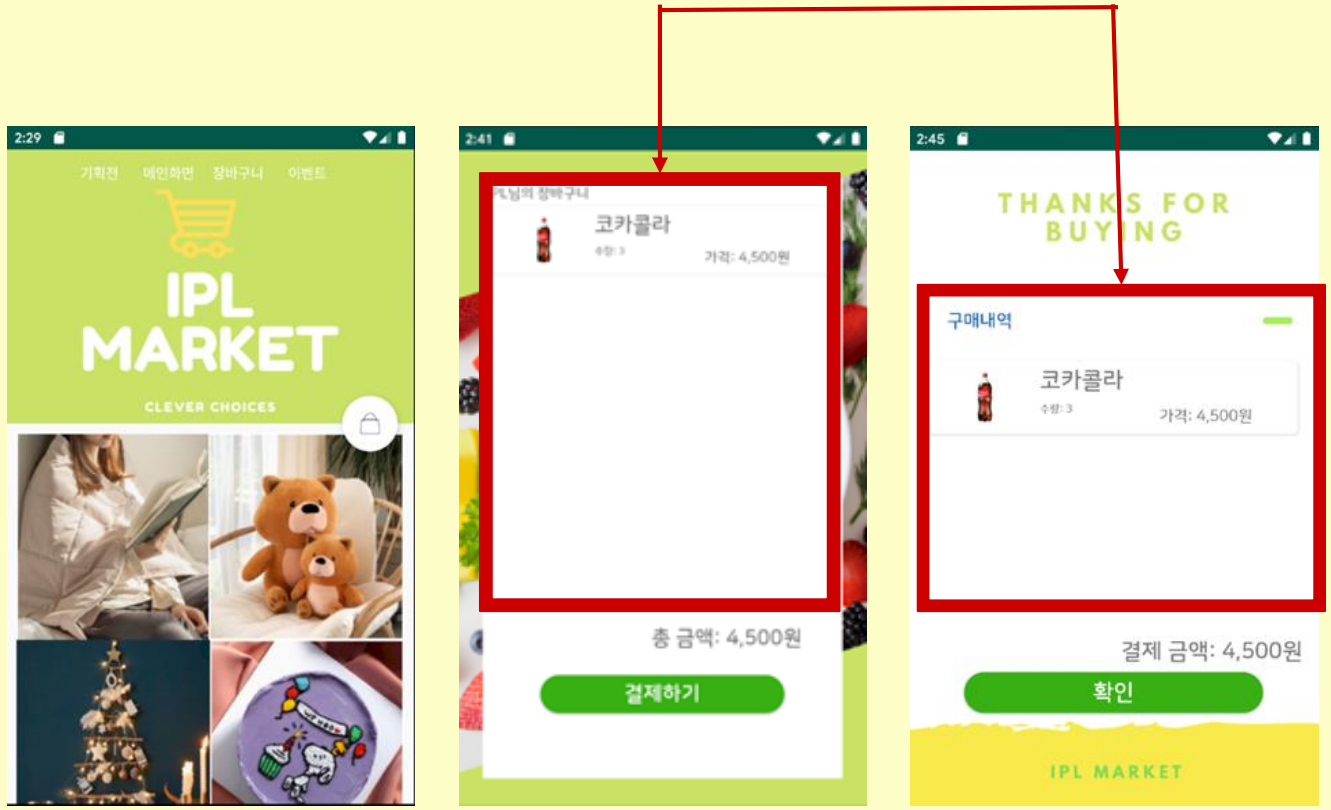
scalable layout으로 뷰(View)들을 감싼 후
base_width, base_height를 설정

각 뷰(View)의 속성으로 scale_width, scale_height를 설정하면 base의 크기에 따라서 상대적으로 뷰(View)들의 비율이 유지됨

Thread Flow


메인스레드
작업스레드
서버로부터 직렬화된 Data 수신
Data 전달
Handler
Data 전달
DB
입력한 계정과
일치하는
계정이 있는지 확인
&
Thread종료
AsyncTask로
Thread 생성
Thread 생성
결제버튼 클릭 후
Thread 종료
회원가입
클릭
로그인버튼
클릭
장바구니 버튼
클릭
물건
추가
결제 버튼
클릭
AsyncTask로
Thread 생성
DB
입력한 값을
DB에 저장
&
Thread 종료
저장된
상품 목록으로
셀프결제
앱 종료

ListView와 RecyclerView



리사이클러뷰를
응용한 그리드 뷰 사용

두 개의 액티비티에서 장바구니,
구매내역은 리사이클러뷰를
사용하여 스크롤을 했을 때,
컨텐츠를 재사용

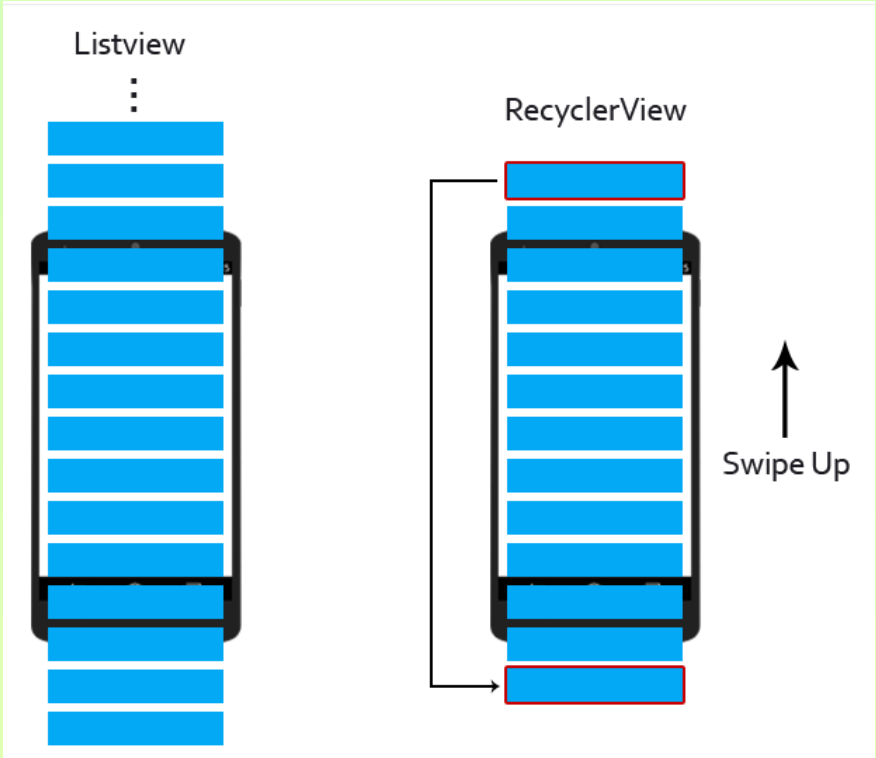
ListView와 RecyclerView

기존의 리스트 뷰 구성

➔ **커스터마이징, 스크롤하면 계속 생성해야 하는 성능 문제**

이를 해결하기 위해서 RecyclerView 사용

- 기존 ListView의 향상되고 유연해진 버전
- 화면에 표시할 만한 개수의 리스트 item을 만들고 **스크롤** 이벤트에 따라 **최 상단 view부터 하단으로 이동**하여 **컨텐츠만 바꾸어 재사용**하도록 설계

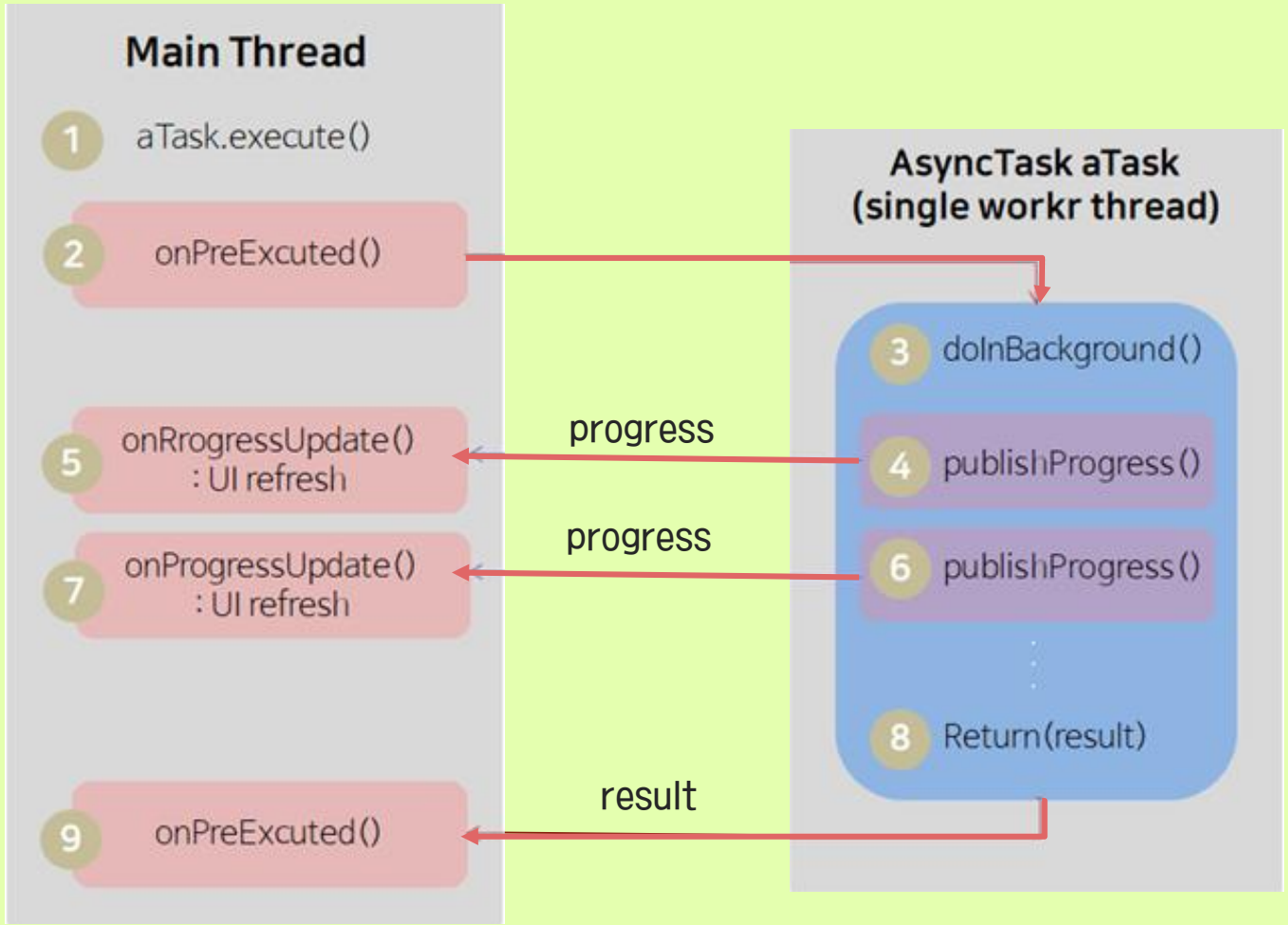


<리스트 뷰>

<리사이클러 뷰>

AsyncTask

<AsyncTask의 동작 순서>



앱이 실행되면
안드로이드(Android)는
메인 쓰레드(Main Thread)를 생성

-> 메인 쓰레드는 UI에 관련된 작업 외에 다른 작업을 수행하지 못함

UI 반응성 향상 , 처리시간이 오래 걸리는 작업 처리를 같이 해결 하기 위해서는 별도의 쓰레드(Thread)가 필요

AsyncTask 사용

AsyncTask

AsyncTask 란?
쓰레드(Thread)나 메시지 루프, 핸들러(Handler) 등의 원리를 이해하지 않아도 **하나의 클래스에서 Background 작업과 UI 작업을 가능하게 하는 비동기식 스레드**

AsyncTask의 단점
AsyncTask를 실행한 **Activity가 종료되었을 때 별도의 지시가 없다면 AsyncTask는 종료되지 않고 계속 돌아가거나** , 한번 실행한 AsyncTask를 **다시 실행하면 에러가 발생**하게 됨

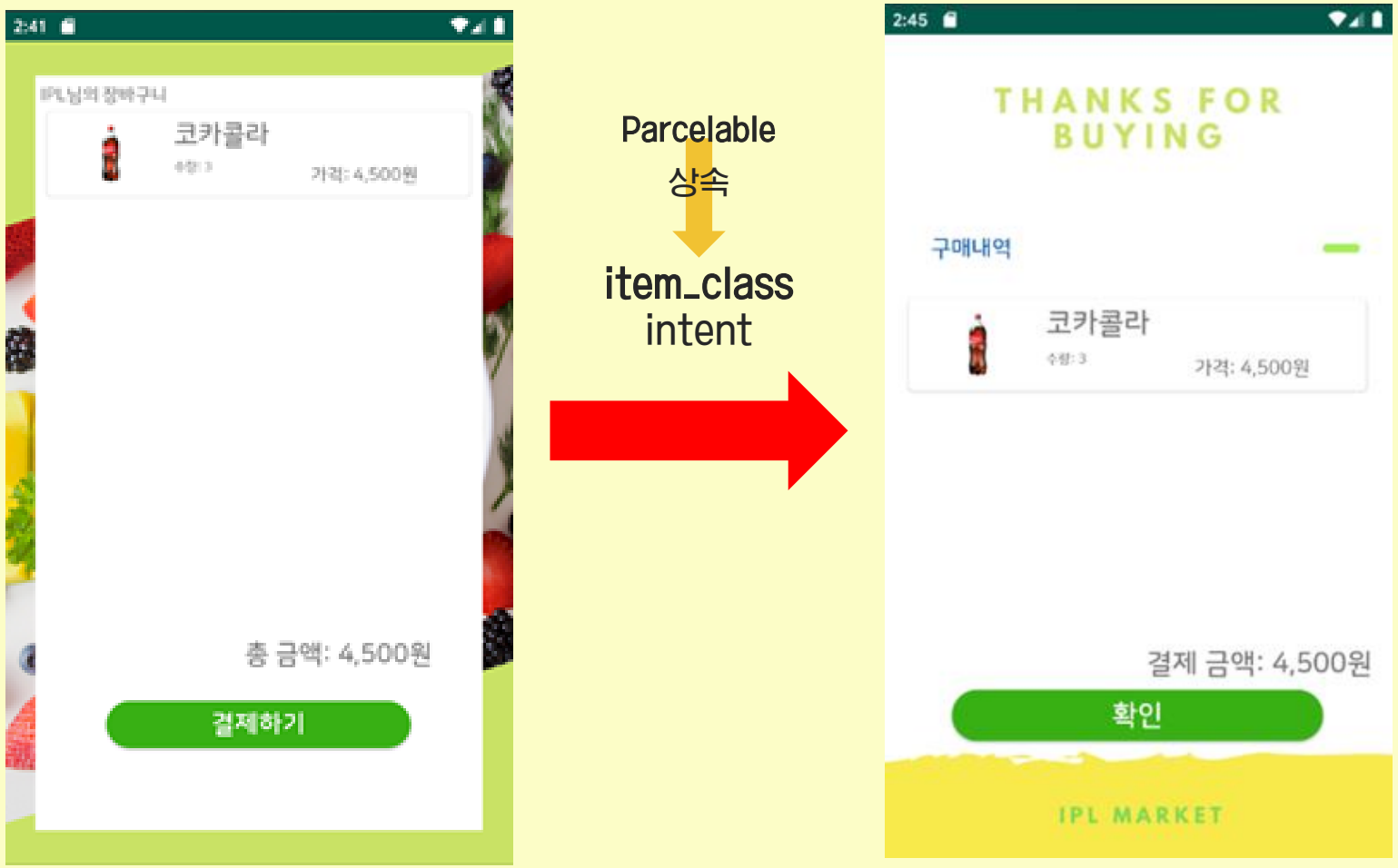
한번만 실행되는 작업에만 AsyncTask 사용

Glide vs Picasso

- 최근 많이 쓰이는 Google에서 개발한 이미지 로딩 라이브러리
- 사용하는 함수 방식은 피카소(Picasso)와 비슷하지만 성능이 더 좋음
- Picasso에 비해 Glide가 메모리 용량을 50%이상 적게 사용함
- Picasso는 원본 이미지를 메모리로 가져와 gpu에서 실시간으로 리사이징하나,
  Glide는 리사이징된 크기로 메모리에 가져오기 때문에 성능면에서 더 효율적

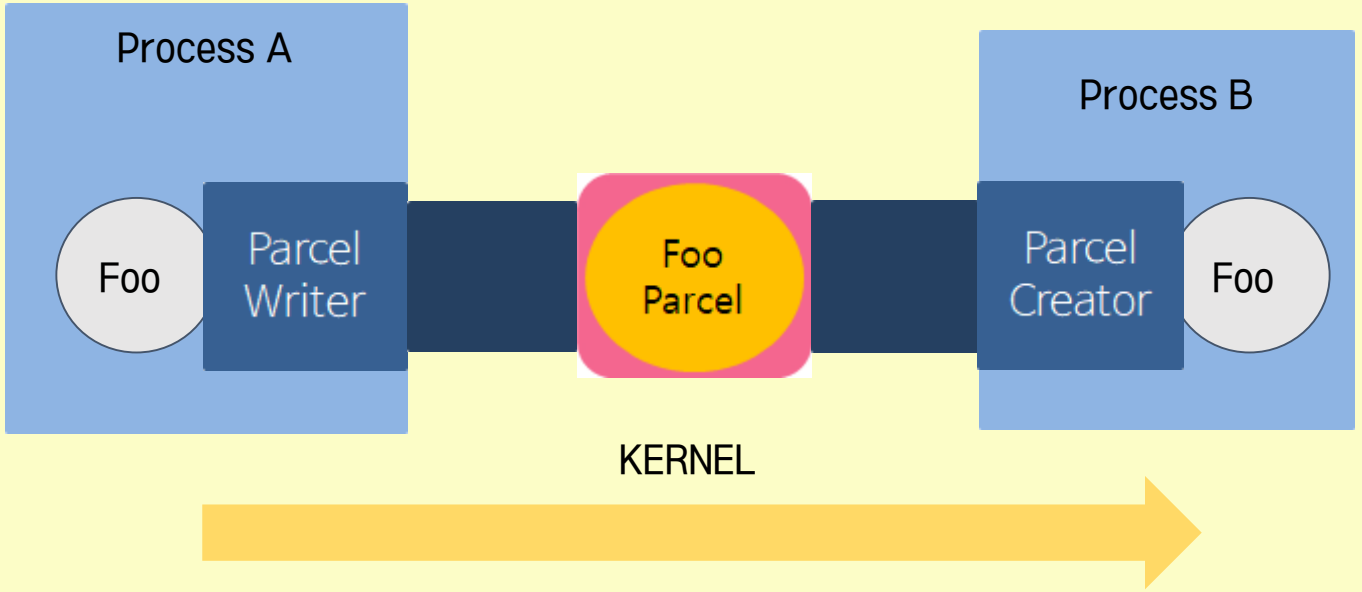
➡ 장바구니의 이미지는 화질보다 빠른 캐시가 우선이기 때문에 Glide 라이브러리 사용

Parcelable



item_class 객체를 intent로 넘기기 위해서 해당 클래스(item_class)에 Parcelable을 상속받음

Parcelable



- Parcelable 은 Reflection을 사용하지 않도록 설계되었음
  - Serializable 과는 달리 직렬화 처리 방법을 사용자가 명시적으로 작성하기 때문에 자동으로 처리하기 위한 Reflection이 필요 하지 않음
- Parcelable는 IPC(프로세스간 통신)을 이용하기 때문에 속도가 빠름
- 프로세스 간의 메모리 영역을 공유할 수 없기 때문에 Parcelable 인터페이스는 커널메모리를 통해 데이터를 다른 프로세스로 전달하는 통로를 만들어 줌

THANK YOU ☺