MOBILE
SECURITY
TECHNOLOGIES
2016

Analysis of Code Heterogeneity for High-precision Classification of Repackaged Malware

Ke Tian, Daphne Yao, Barbara Ryder

Department of Computer Science
Virginia Tech

Gang Tan

Department of Computer Science
and Engineering
Penn State University

✓Motivation:

Repackaged malware skews machine learning results

✓Solution:

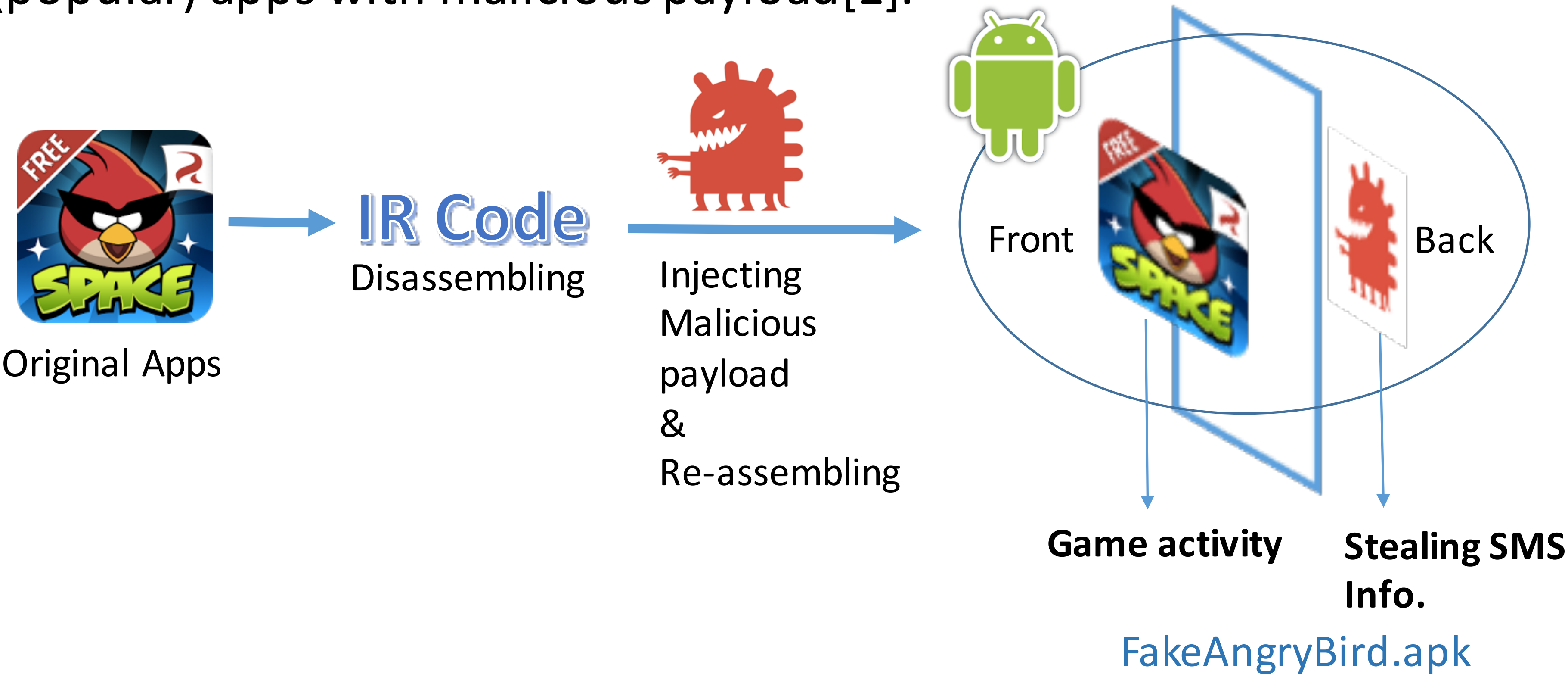
Partition + Machine learning classification

✓Experiment:

30-fold improvement in False Negative than non-partition ML-approach!

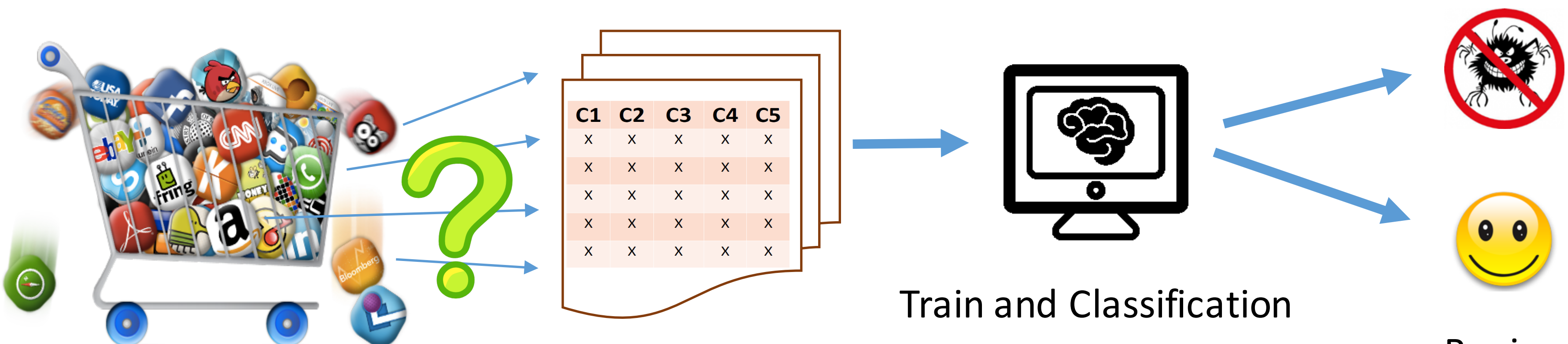
Repackaged Malware

Android Malware writers are repackaging legitimate (popular) apps with malicious payload[1].



[1] http://www.zdnet.com/article/android-malwares-dirty-secret-repackaging-of-legit-apps/

Conventional Machine Learning for Malware Classification



A huge Dataset

Extract Feature vectors

Benign &
Malicious

DroidAPIMiner : Sensitive APIs
Drebin : APIs, constant strings, URLs
Peng et al.: Permission

Is machine learning taking over the world?

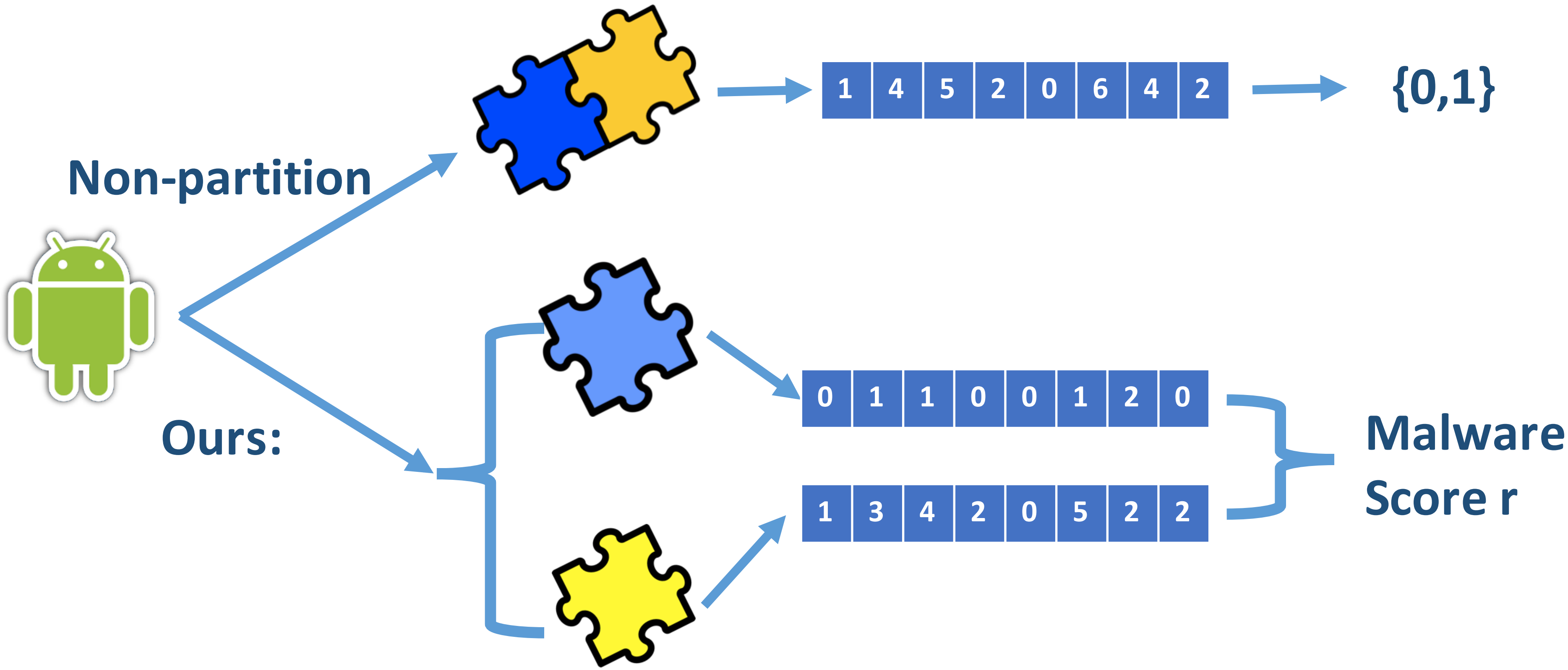
No – What the specific challenges and solutions?

Heterogeneous Code:
Code with different Security behaviors in different code portions

FakeAngryBird.apk

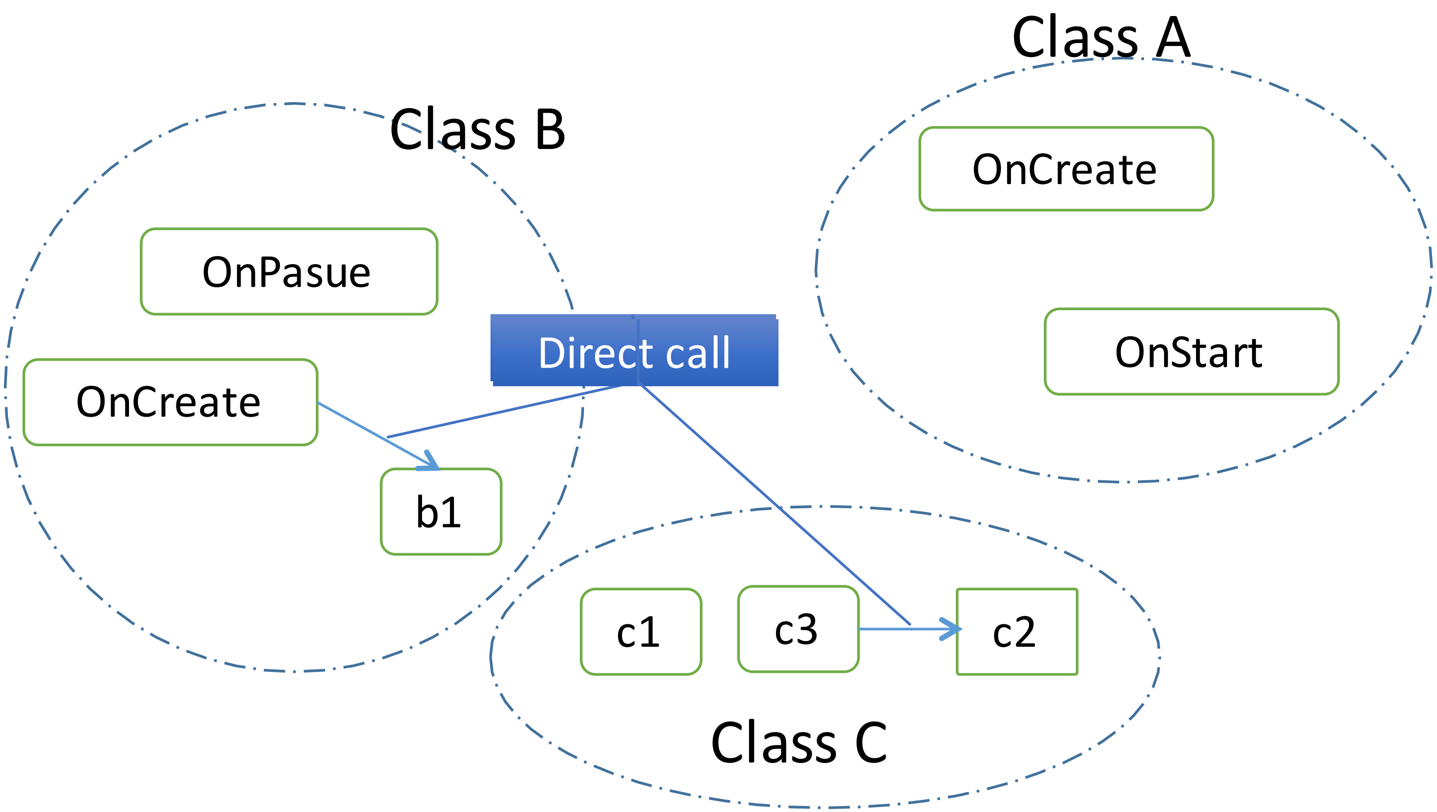
Existing machine learning techniques extracts features from the entire app, repackaged malware skews classification results (i.e., introduce false negatives)

Research Question: How to recognize heterogeneity in code?

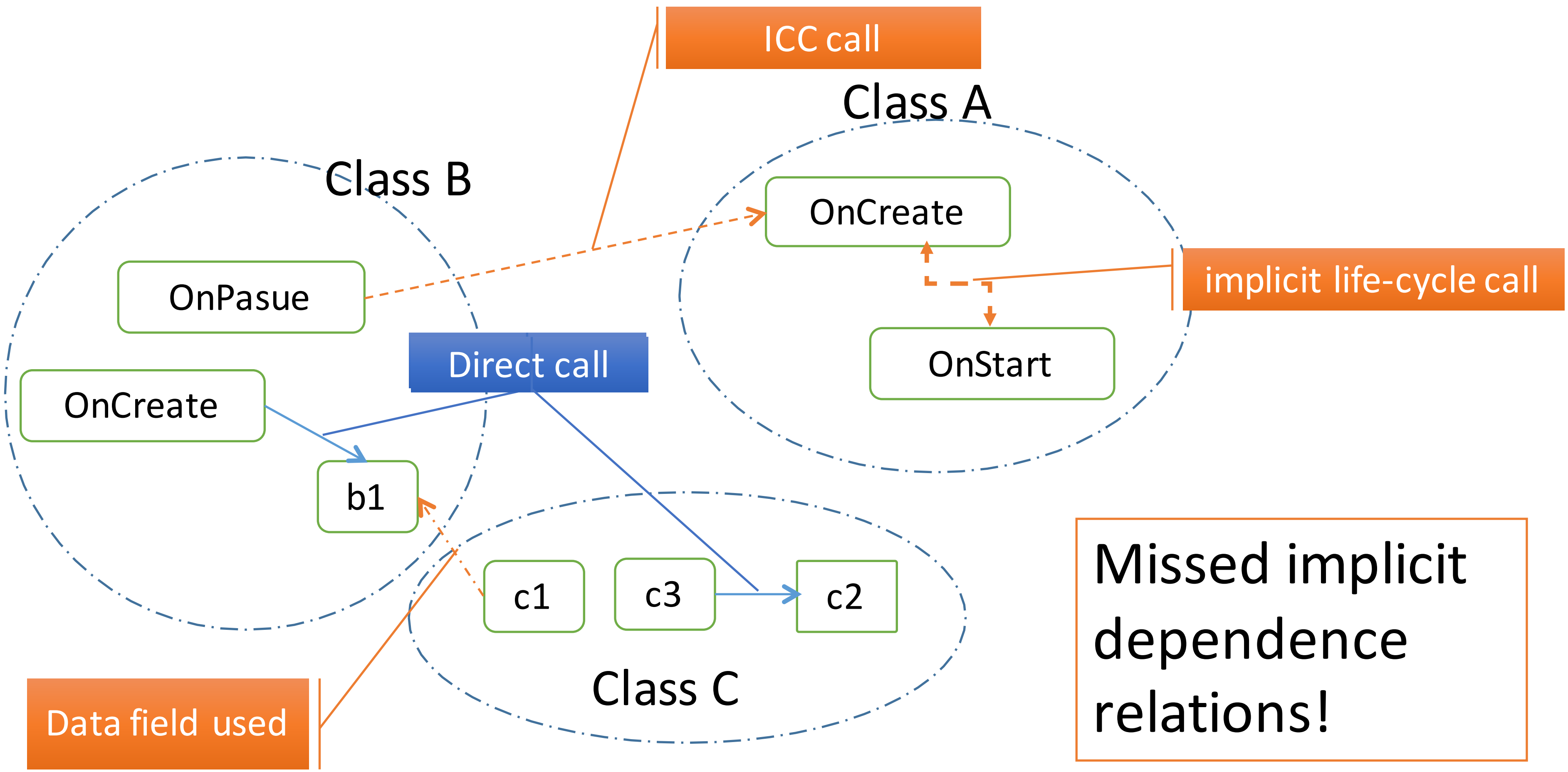


Tasks:
- How to partition the code?
- How to extract efficient features?
- How to calculate the malware score?

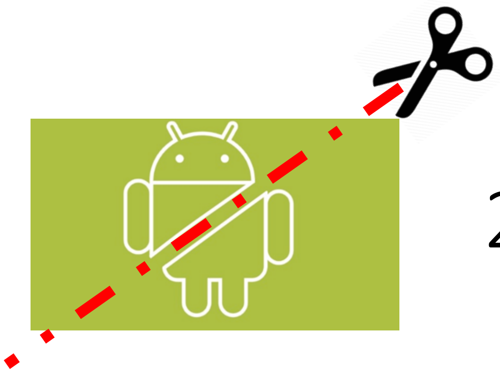
First Attempt: partition based on direct method call relations



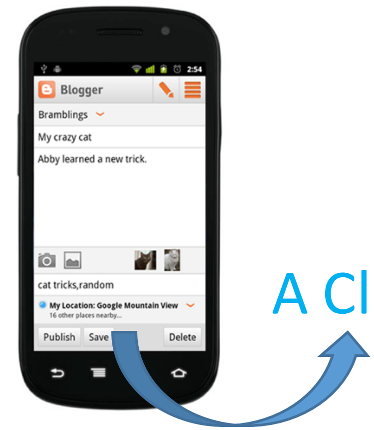
First Attempt: not wok well



Missed implicit dependence relations!



2-level graph



A Class

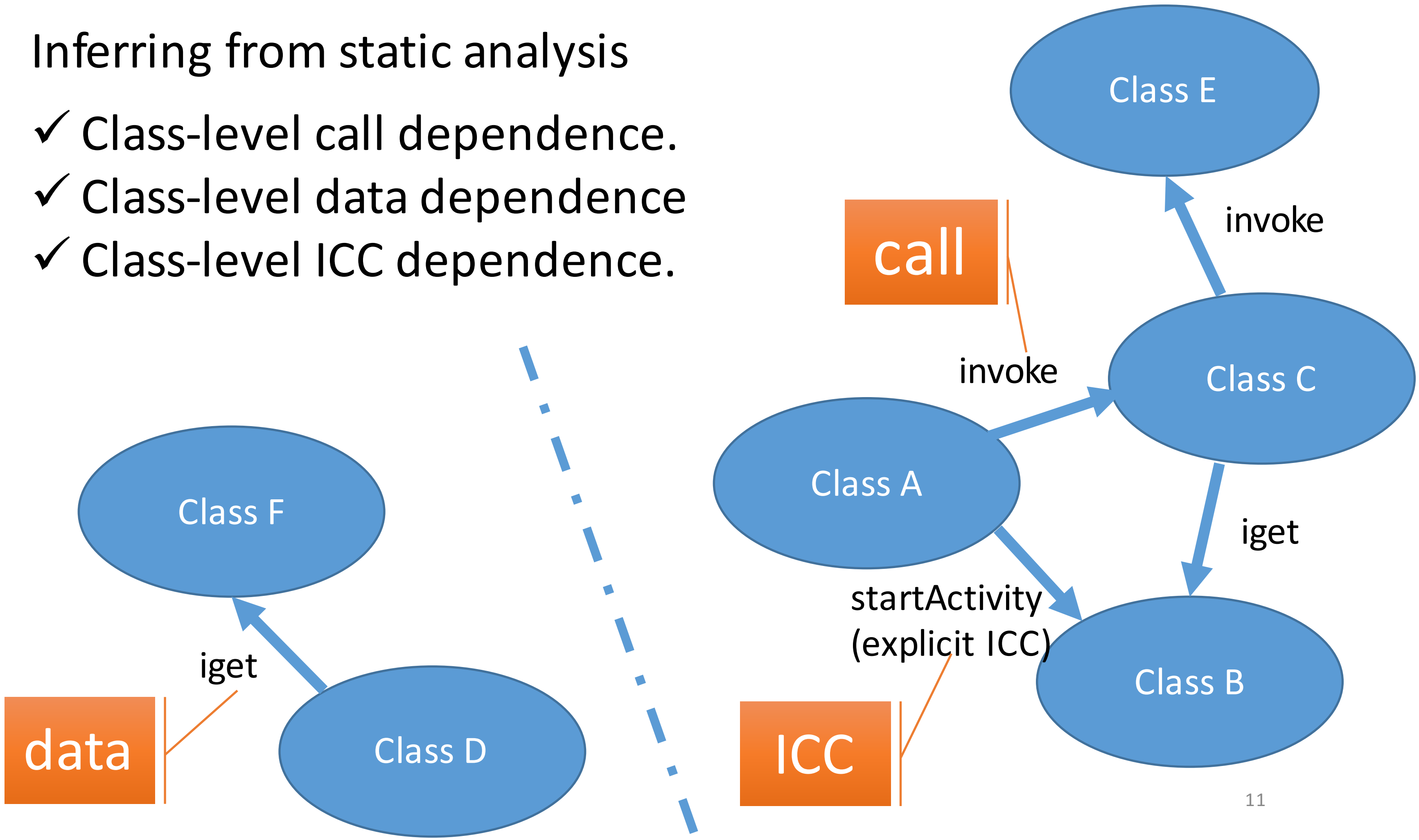
Class-level Dependence Graph (CDG) to capture event (activity) relations.

Method-level Call Graph (MCG) for subsequent feature extraction.

Class-level Dependence Graph (CDG)

Inferring from static analysis

- ✓ Class-level call dependence.
- ✓ Class-level data dependence
- ✓ Class-level ICC dependence.



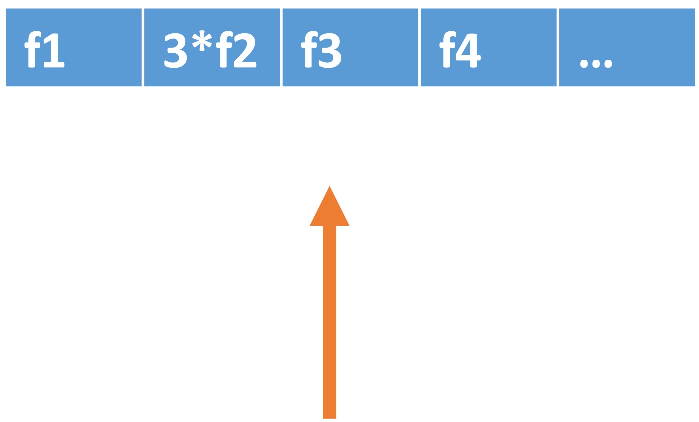
So far, we got code partitioned at class-level dependence graph

Can feature extraction be done on class-level call graph?

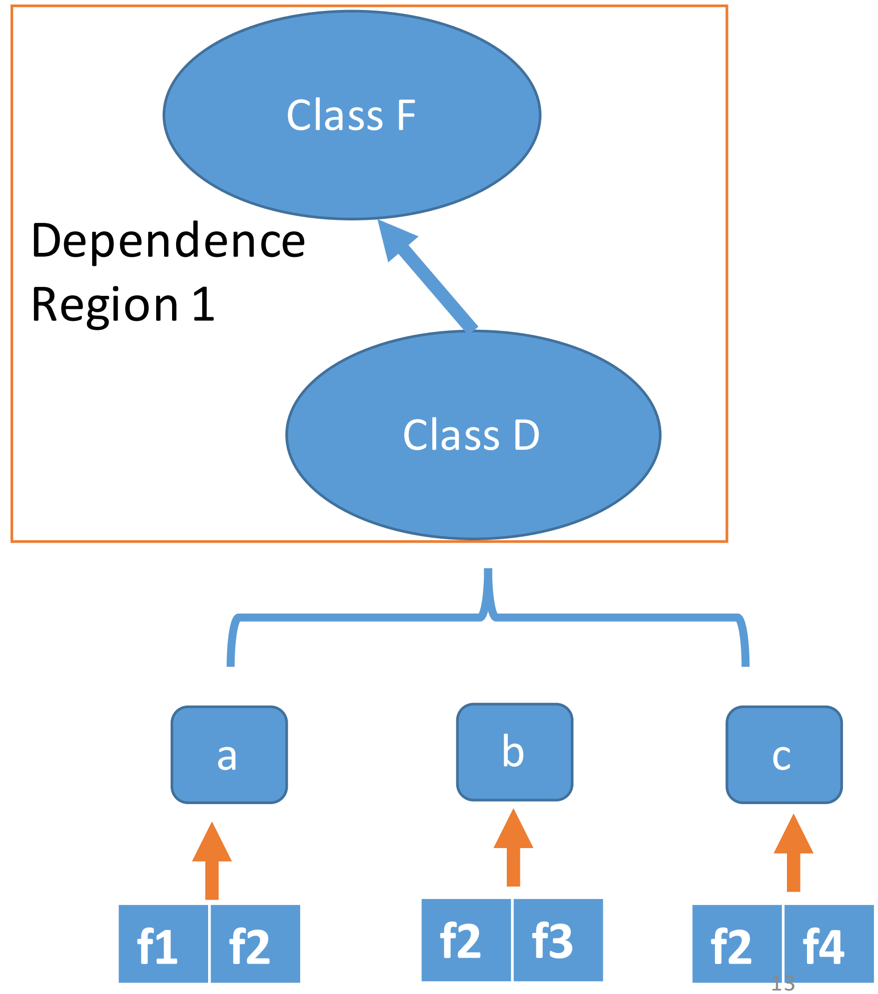
No. Why?

Class-level call graph is too coarse-grained, lacking useful method information. Need method-level details

Mapping Through Projection (to prepare for feature extraction)



Aggregation features in each region



Feature Extraction for Regions

✓ Type I: User Interaction Features
*user-related functions and the graph-related impact features

✓ Type II: Sensitive API Features.
*sensitive Java and Android APIs

✓ Type III: Permission Request Features.
*permissions used in each region

1.Features are used to profile the region’s behaviors.
2.Combined with traditional features, user interaction and graph properties

Classification of Apps

- Binary Classification for each dependence region.
- Computing the malware score for an app based on results from all regions.

$Malware\ score\ r_m$

$$r_m = \frac{N_{mali}}{N_{total}}$$

Malicious regions

Total regions in the app

Continuous value in [0,1]

Solution summaries:

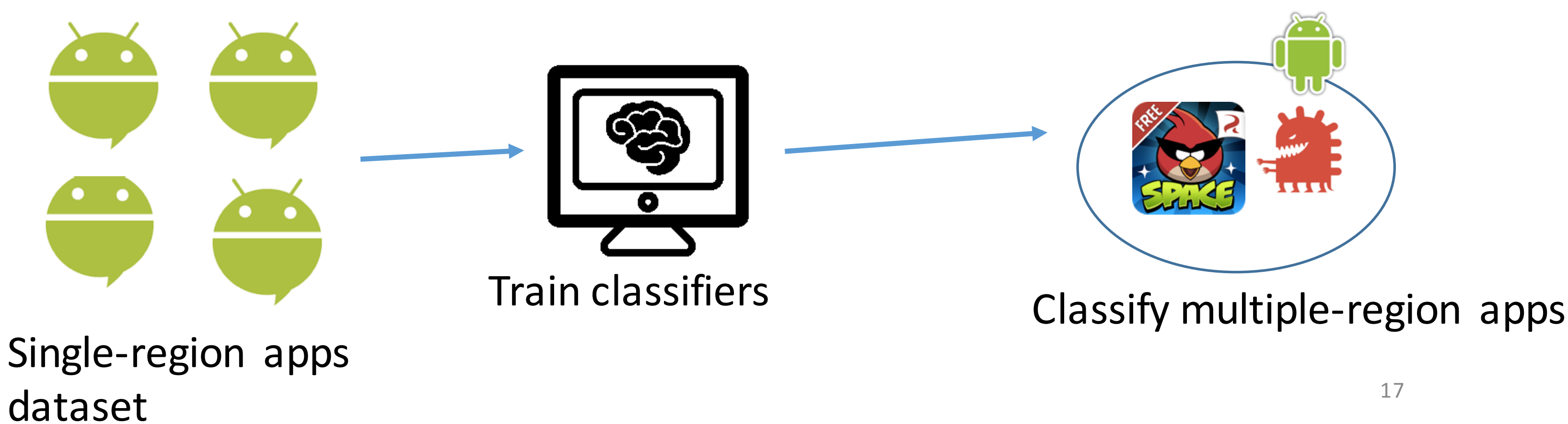
- *(Partition)* Partition the app into different Regions –> Class-level Dependence Graph (CDG)
- *(Feature)* Independently classify each Region –> Method-Level Call Graph(MCG)
- *(Classification)* Mapping the features through projection, calculating Malware Score

Limitations :

- Graph Accuracy. -- More accurate program analysis
- Dynamic Code -- Native Libraries
- Integrated Malware – Hard to partition

Classification of non-repackaged Apps

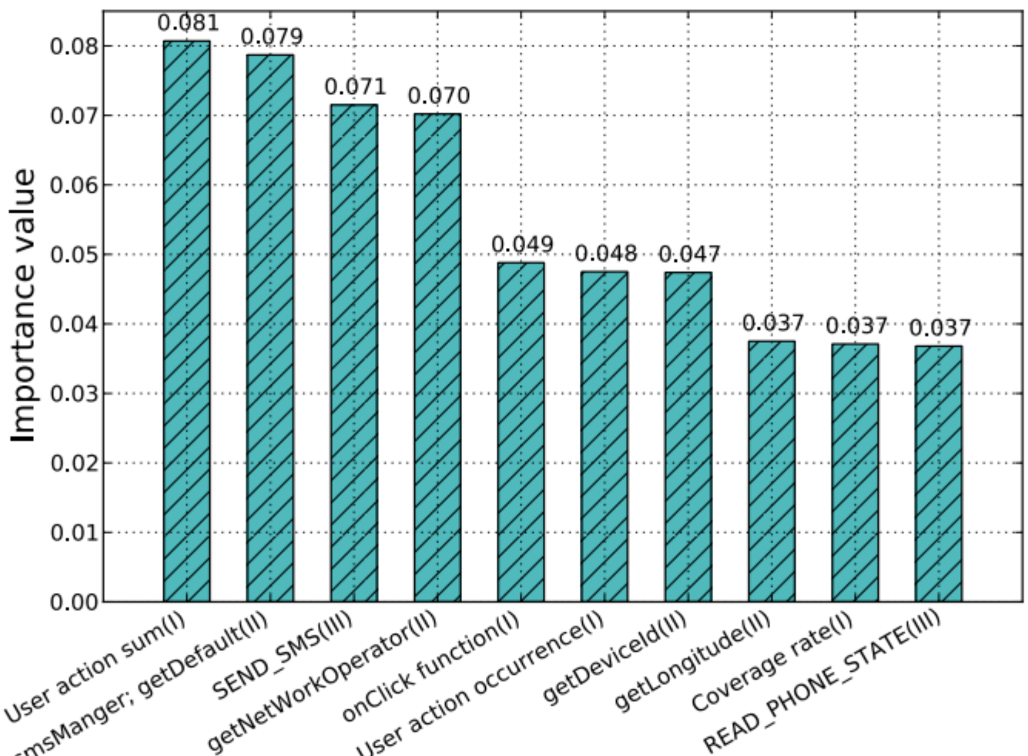
- Each of apps contains just a single region (dependence region).
- The region is labeled as benign or malicious from dataset
- Used to evaluate the features and get trained classifiers



Classification of non-repackaged Apps

Cases	FNR(%)	FPR(%)	ACC(%)
KNN	6.43 ± 5.22	6.50 ± 2.67	93.54 ± 3.33
D.Tree	4.78 ± 2.90	3.52 ± 1.57	95.79 ± 2.14
R.Forest	3.85 ± 3.27	1.33 ± 0.78	97.30 ± 1.96
SVM	7.42 ± 4.85	1.46 ± 0.58	95.28 ± 2.58

Random Forest performs Best



All three types of features are effective

Use Random Forest as the standard classifier to test repackaged malware

Classification of Repackaged Malware

Malware Families	Geinimi		Kungfu		AnserverBot		Average
	FN	FNR(%)	FN	FNR(%)	FN	FNR(%)	FNR(%)
Partition-based	0(62)	0	4(374)	1.07	0(185)	0	**0.35**
Non-partition-based	12(62)	19.36	12(374)	9.89	3(185)	1.62	**10.29**

Test three repackaged malware families:

1 Geinimi

2 Kungfu

3 AnserverBot

Comparison:

1 Entire-app classification (Basic)

2 Our partition classification

Use the Same trained Random Forest to test

our FNR gets 30-fold improvement than the non-partition!

Case Study of Heterogeneous Properties

DroidKungfu1–881e*.apk		Partition (ours)		Non-partition
Feature	Description	DRegion1	DRegion2	N/A
Type III	READ_PHONE_STATE permission	**0**	**1**	1
	READ_LOGS permission	**0**	**1**	1
Type II	getDeviceId function in Landroid/telephone/telephoneManager	**0**	**1**	1
	read function in Ljava/io/InputStream	**0**	**3**	3
	write function in Ljava/io/FileOutput	**0**	**1**	1
Type I	onClick function occurrence	**16**	**2**	18
	# of distinct user-interaction functions	**5**	**1**	5
	onKeyDown function occurrence	**3**	**0**	3
Classification		Benign	Malicious	Benign
Correctness		✓(Yes)		✗(No)

- Malicious Region with sensitive permissions& APIs
- Benign Region with user-interaction functions

Need to look into the code structure!

Region analysis in popular apps

- Analyzing 1,617 free popular apps from Google Play.
- 158/1,617= 9.7% Apps contain multiple regions
- Ad Libraries introduce multiple regions in Apps.
- Some aggressive ads libraries introduce alerts in the detection.

	w/o Ads	w/ Group 1 Ads	w/ Group 2 Ads
% of Alerts	2.96%	2.96%	5.18%

Table. Alerts made by Group 2 Ads library (Group 1:admob | Group 2:adlantis)

False Negatives:

1) Integrated benign and malicious behaviors.
2) Not enough malicious behaviors in malicious components

False Positives:

Some aggressive packages and libraries, e.g., Adlantis, results in a false alarm in our detection.

Conclusions:

- Our approach achieves 30-fold improvement than the non-partition-based approach.
- Our approach is able to identify malicious code in repackaged malware.
- Partition can be used to label malicious code or Isolate inserted code (Ads packages or dead code)

Future work:

More Effort on Partition/Detection for Code Provenance!

Thanks!