Weekly Report and Discussion

Yutian Tang

Discussion Outline

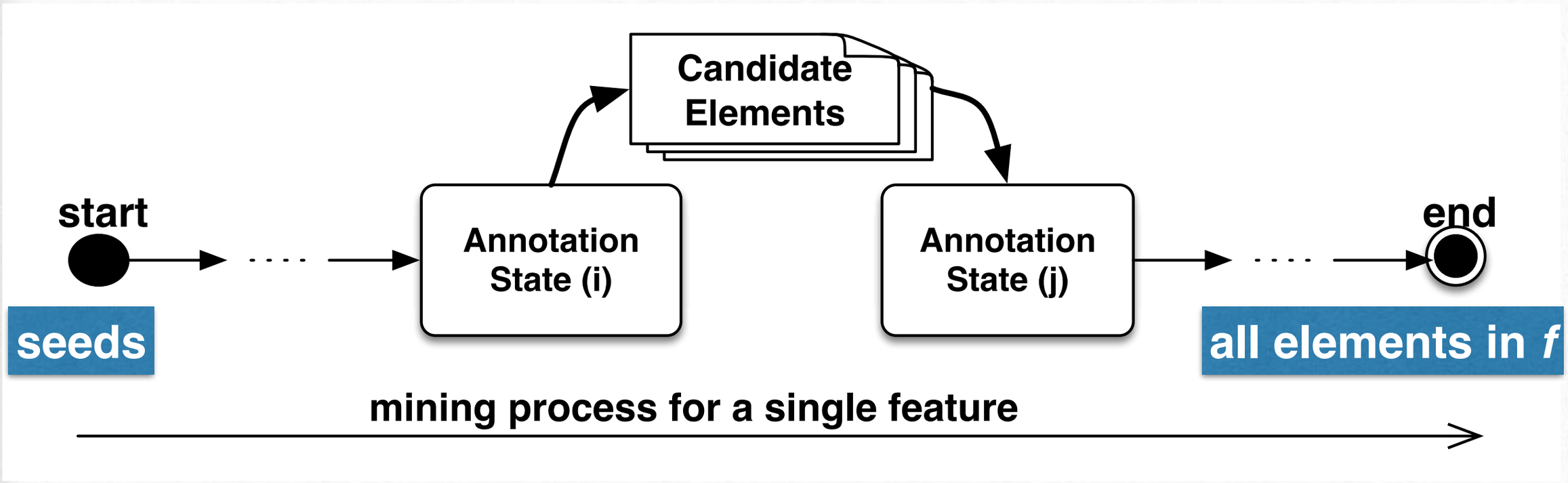
- Extension on StiCProb;
- Discussion on Android Plugin Framework;

StiCProb: Core Idea

- Main contribution:
  - feature location problem with fine granularity;
  - using conditional probability

Annotate Features

$$S(A^*, f, i) = \left\{ e \middle| (e, f) \in A^* \right\}$$



Feature-Element Correlation Coefficient(cond')

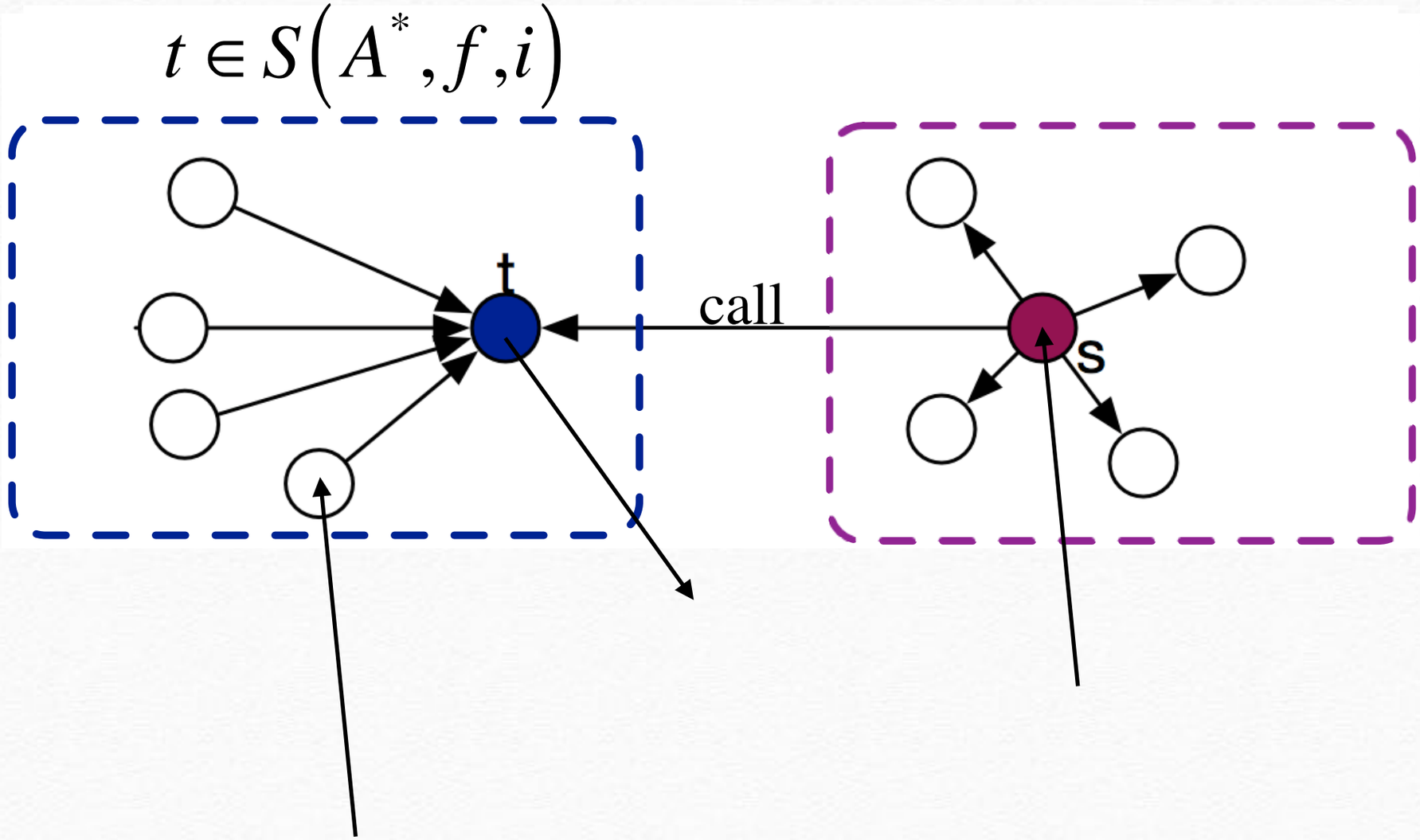
Prog. Elements Cand.

feature f

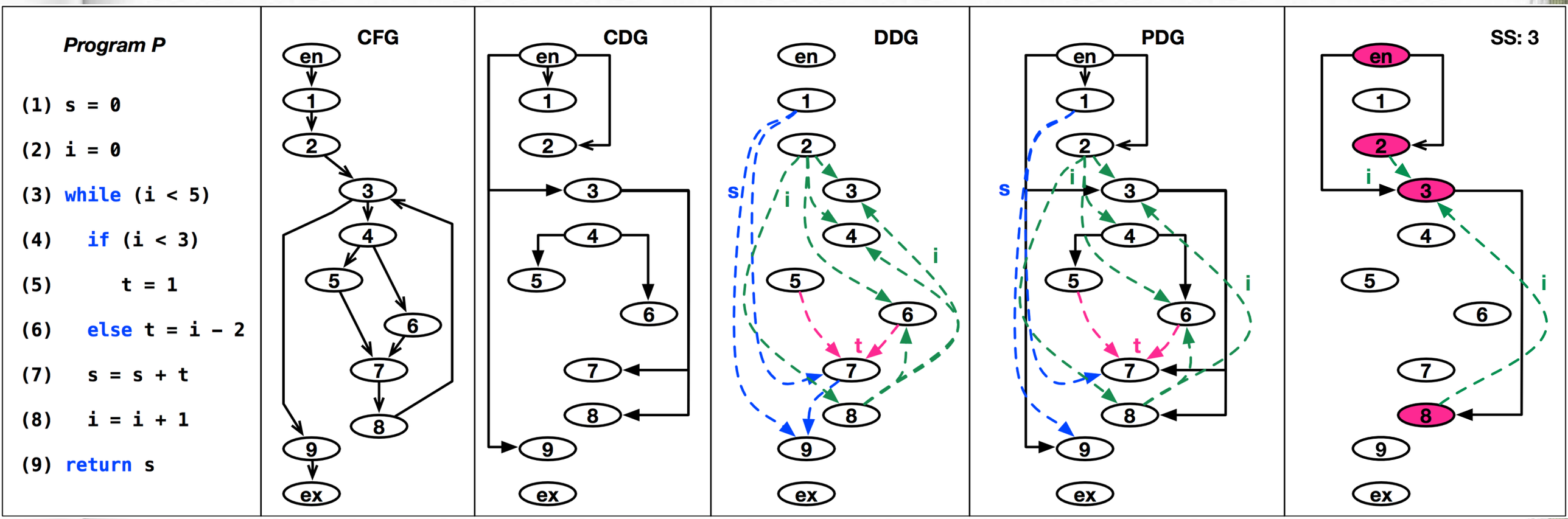
$$S\left(A^*, f, i\right) \cdots\cdots\cdots\cdots\blacktriangleright \qquad \cdots\cdots\cdots\blacktriangleright S\left(A^*, f, i+1\right)$$

$$p\left(e \middle| S\left(A^*, f, i\right)\right)$$

Feature-Element Correlation Coefficient(cond′)



Feature-Element Correlation Coefficient(cond')



Slicing Scope

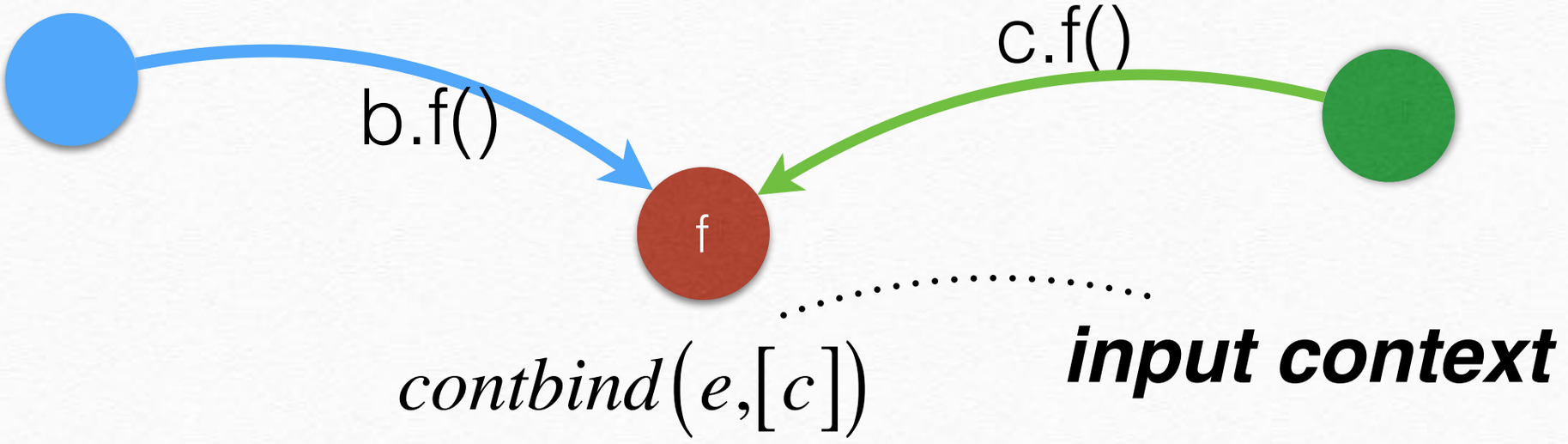
$$sscope(e) = e \cup \left\{ s \middle| s \xrightarrow[cf]{df} e, s \in E \right\}$$

Feature-Element Correlation Coefficient(cond')

Binding

$$bind(e) = def(e) \cup use(e)^* \quad \text{in} \quad sscope(e)$$

Context Binding



* All def and use within e

Context Binding

Context Binding @ Method Invocation

callsite $l = r_0.m(r_1, r_2, ..., r_n)$

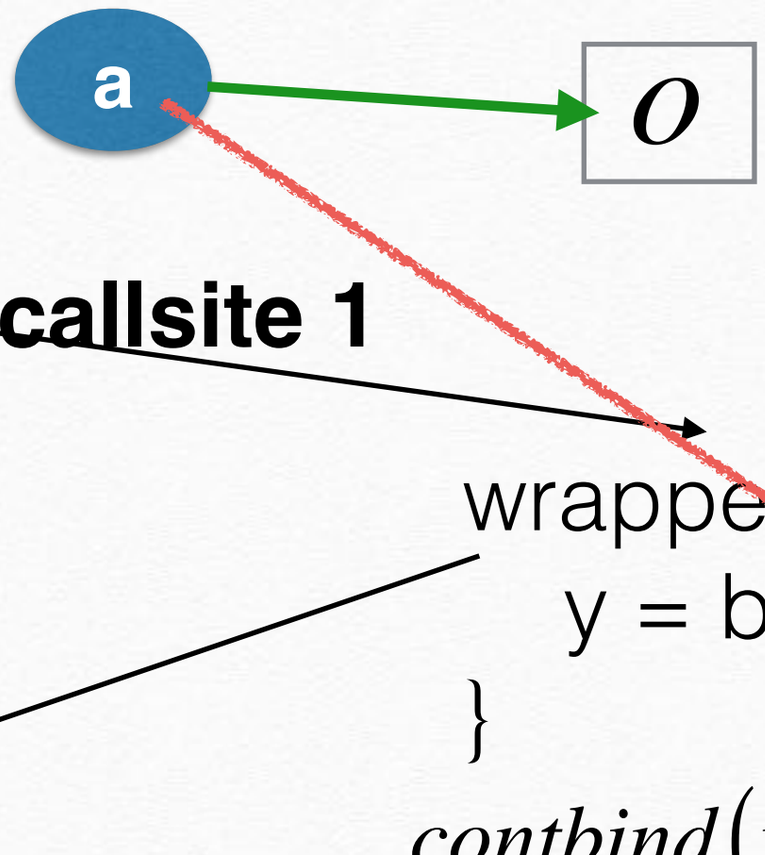
$$contbind(m, [r_1, ..., r_n]) = dispatch(p_i = r_i) \rightarrow bind(m)$$

[13] L. O. Anderson, “Program analysis and specification for c programming language”

Context Binding

Context Binding @ Method Invocation

```
main(){
    A a = new A();
csite 1: z = wrapper(a);
}
```




```
wrapper (A b){
    y = bar(b); return y;
}
```

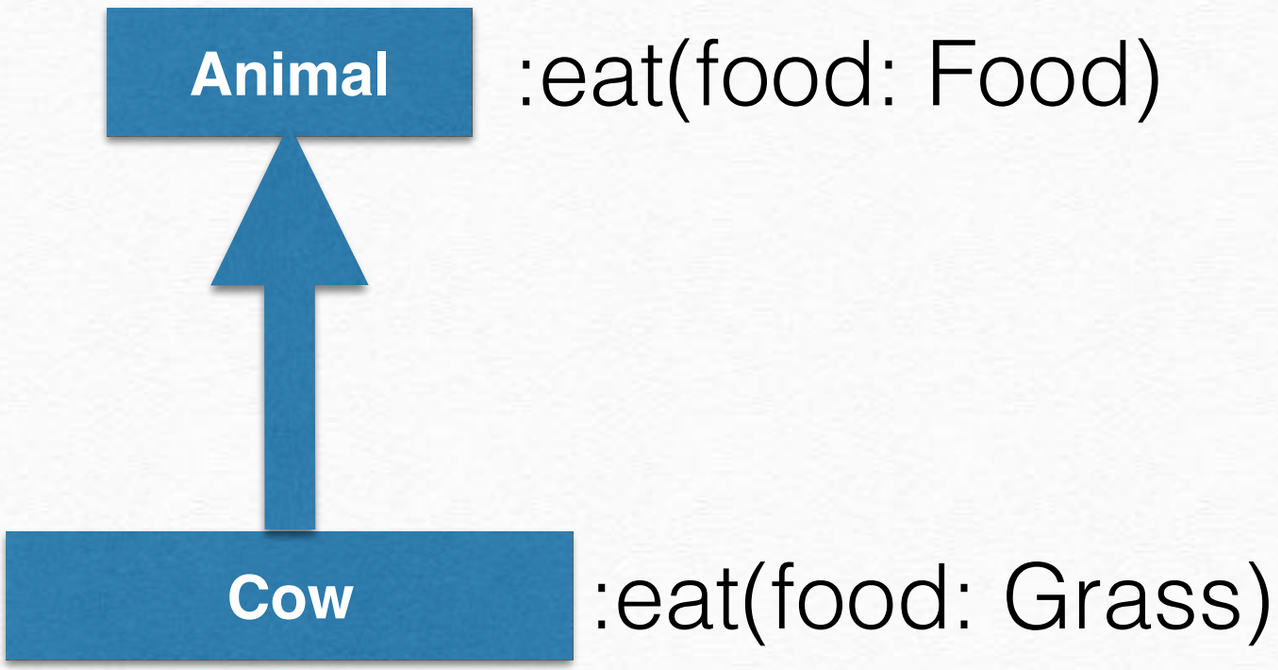
$contbind(wrapper(..),[a])$

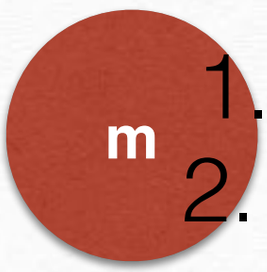
```
bar (A c){
    x = c.f; return x;
}
```

[13] L. O. Anderson, “Program analysis and specification for c programming language”

Context Binding

Context Binding @ Overriding

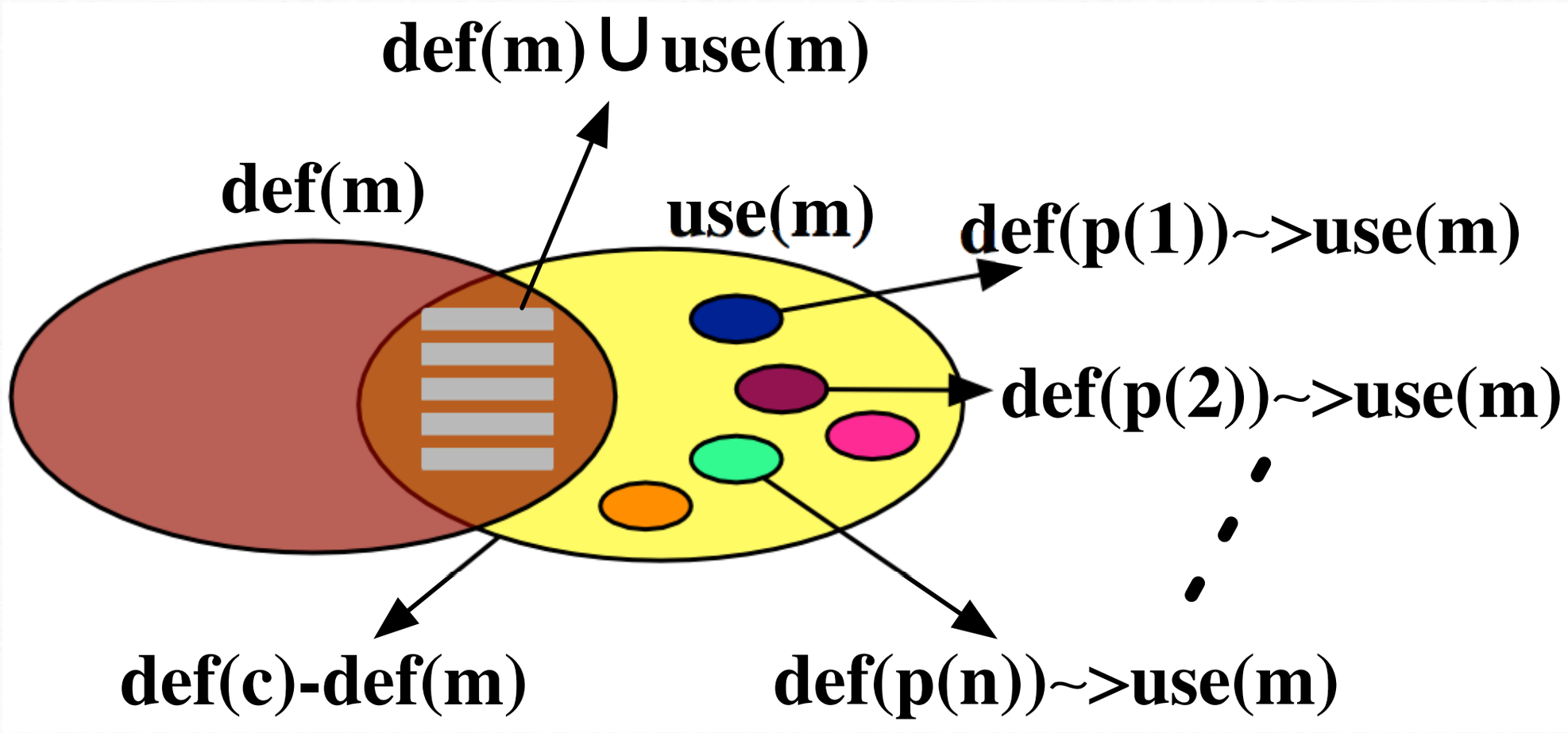




1. def and use in m
2. def in parent class/interface, used in m

Context Binding

Context Binding @ Overriding



Context Binding

Context Binding @ Overriding

defined in m: $def(m)$

used in m: $use(m)$

1. used in m and defined in m
2. used in m and not defined in m

$$contbind\left(m,\left[p_1,....,p_n\right]\right)=def\left(m\right)\cup\bigcup_{i=1}^{n}\left(use(m)\sim>def\left(p_i\right)\right)$$

~> used to specify the source of the context

Context Binding

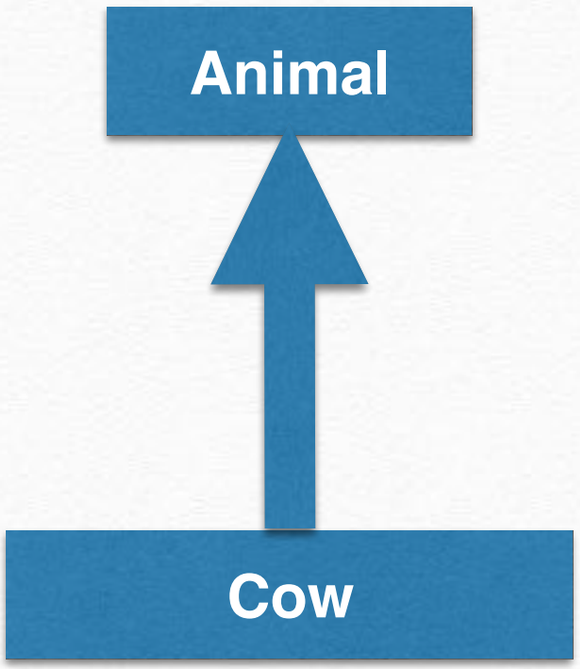
Context Binding @ Overriding : **Example**

```
1 public class FlyingCar implements OperateCar {
2    public int startEngine(int encryptedValue) {
3        OperateCar.super.startEngine(OperateCar.encryptedValue);
4    }
5 }
-------------------------------------------------------------
6 public interface OperateCar {
7    int encryptedValue = 1;
8    default public int startEngine(int value) {...}
9 }
```

$$contextbind(startEngine) = \{encryptedValue, OperateCar.encryptedValue\}$$

Context Binding

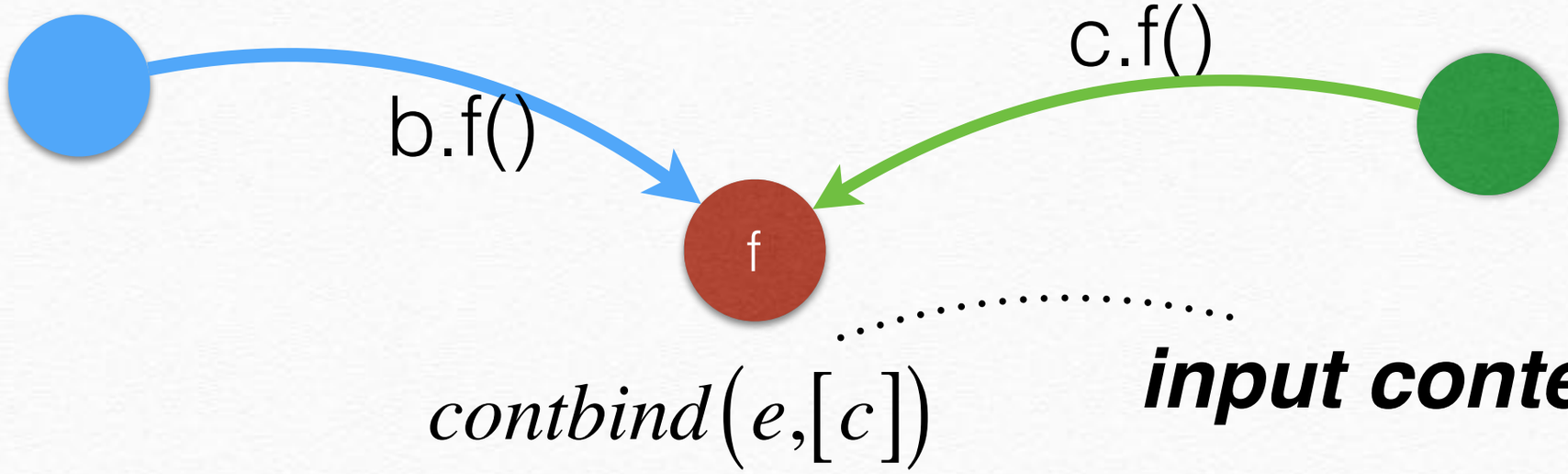
Context Binding @ Inheritance



$$contbind\left(c,\left[p_1,..,p_n\right]\right)=bind\left(c\right)\cup\bigcup_{i=1}^{n}def\left(p_i\right)$$

Feature-Element Correlation Coefficient(cond')

Context Binding



by default: ***context-aware points-to analysis***

Feature-Element Correlation Coefficient(cond′)

$$S(A^*, f, i) = \left\{ e \middle| (e, f) \in A^* \right\}$$

$$S(A^*, f, i) = \bigcup_{a \in S\left(A^*, f, i\right)}^{a} contextbind(a)$$

StiCProb: Uniquess Table

element *s* and *t* with a relation r $s \xrightarrow{r} t$

$$U\left(E,T,R,P_{forward},P_{backward}\right)$$

$P_{forward}$

the uniqueness of *t* to *s* if *s* has been annotated to a feature ***f***

$$s \in S\left(A^*,f,i\right) \qquad S(A^*,f,i)=\left\{e\middle|(e,f)\in A^*\right\}$$

StiCProb: Uniquess Table(cond')

$P_{forward}$

the uniqueness of *t* to *s* if *s* has been annotated to a feature ***f***

$$s \in S(A^*, f, i) \qquad S(A^*, f, i) = \{e | (e, f) \in A^*\}$$

$$p_{forward}\left(s \xrightarrow{r} t \middle| (s, f) \in A^*\right) = \frac{contbind(t, [s])}{contbind(s)}$$

StiCProb: Uniquess Table(cond')

$P_{backward}$

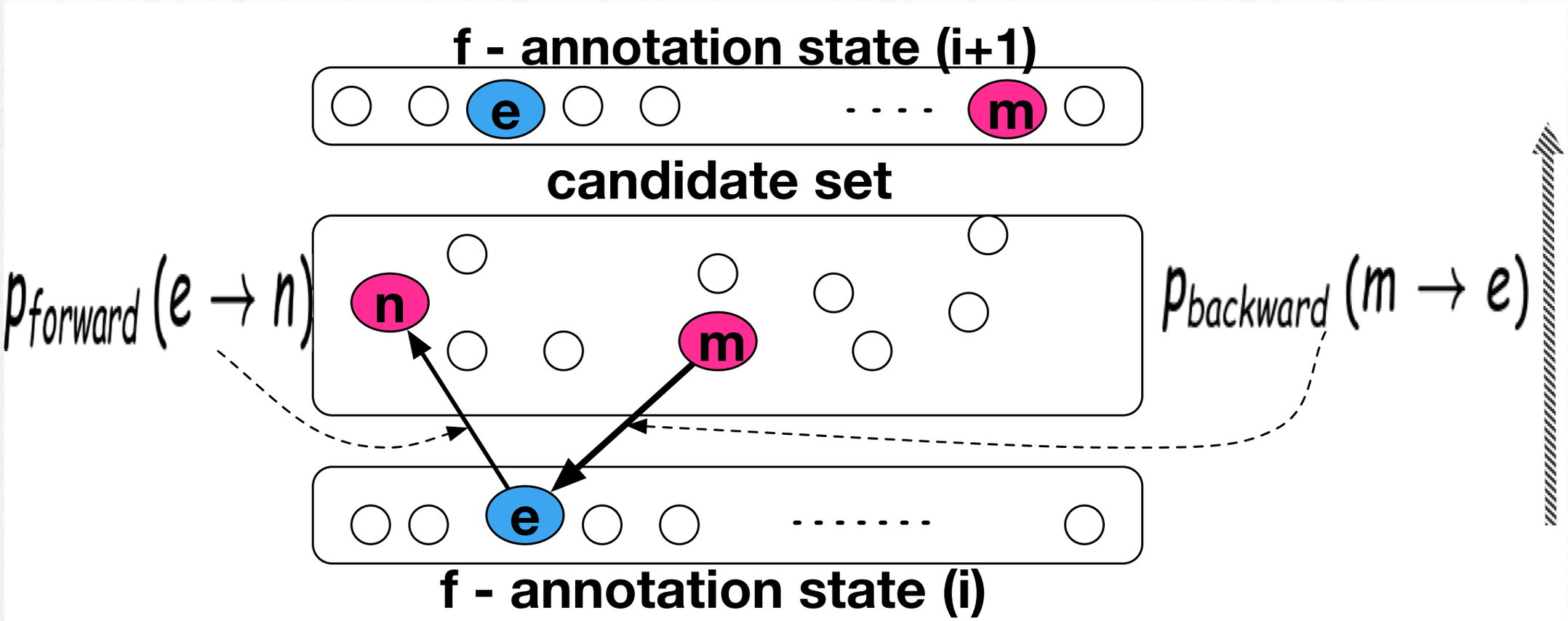
the uniqueness of s to t if *t* has been annotated to a feature ***f***

$$p_{backward}\left(s \xrightarrow{r} t \middle| (t,f) \in A^*\right) = \frac{contbind(t,[s])}{\bigcup_{i \xrightarrow{r} t} contbind(t,[i])}$$

$$s \in S(A^*,f,i) \qquad S(A^*,f,i) = \left\{e \middle| (e,f) \in A^*\right\}$$

$\bigcup_{i \xrightarrow{r} t} contbind(t,[i])$ a collection of context binding from all prog. elements, which have relation *r* with *t*.

StiCProb


f - annotation state (i+1)
e
m
candidate set
$p_{forward}(e \rightarrow n)$
n
m
$p_{backward}(m \rightarrow e)$
e
f - annotation state (i)

StiCProb: Extension

- 1.

RePlugin vs. VirtualAPK

RePLUGIN

- **极其灵活**：主程序无需升级（无需在Manifest中预埋组件），即可支持新增的四大组件，甚至全新的插件
- **非常稳定**：Hook点仅有一处（ClassLoader）。其崩溃率仅为“万分之一”，**并完美兼容市面上近乎所有的Android ROM**
- **特性丰富**：支持近乎所有在“单品”开发时的特性。**包括静态Receiver、Task-Affinity坑位、自定义Theme、进程坑位、AppCompat、DataBinding等**
- **易于集成**：无论插件还是主程序，**只需“数行”就能完成接入**
- **管理成熟**：拥有成熟稳定的“插件管理方案”，支持插件安装、升级、卸载、版本管理，甚至包括进程通讯、协议版本、安全校验等
- **数亿支撑**：有360手机卫士庞大的**数亿**用户做支撑，**三年多的残酷验证**，确保App用到的方案是最稳定、最适合使用的

什么时间进行绑定

讨论一下 支持的框架 和 SPL进行比较 哪一种 框架化 更好

RePlugin

RePLUGIN

	Dynamic-Load-Apk	Atlas/ACDD	DroidPlugin	卫士Plugin
不改主程序Manifest	不支持	部分支持	支持	支持
插件与宿主交互	一般	容易	较复杂	容易
插件接入成本	一般	一般	无成本	很低（编译期）
Hook量	很少	很少	很多	很少，只有一处
UI插件所处进程（性能 / 独立性）	单进程	单进程	多进程	单进程 & 进程坑位
场景	简单插件	组件解耦化	应用免安装	独立发布 全面插件化

360 www.360.cn GMTC

Virutal APK
框架也加入 进行比较

比较 binding的时间
运行的 先后 场景啊

插件之间的 interaction
如何调用资源

RePlugin

Hook了太多

这里所说的“Hook”是指通过 Java 反射手段，获取并修改与系统 Server 等交互的 Internal API，来让框架正常工作的行为，如上面所列部分。正常情况下的反射（例如反射类内部自己的字段）不属于 Hook。

看似灵活，然而下列三种情况，将很有可能导致插件甚至应用，彻底不能工作：

- **Android 升级**：既然是内部 API，那么 Android 自然不会认为是“不能修改的”，一旦系统升级时做了改动，轻则功能不正常，重则直接 Crash。例如有的插件化框架曾遇到在 Android 7.0 上出现异常，必须升级主程序才能解决的事故，历历在目。
- **ROM 修改**：比 Android 升级更可怕的，是第三方 ROM 对内部 API 的“各种改”。这个适配难度是可想而知的。例如自定义 Resource、自定义 WiFiService 等造成的“插件化血案”，不一而足。
- **使用不当**：“常在河边站，哪有不湿鞋”，Hook 的点多了，一旦对某一点的实现原理理解不透而出了错。结果，前功尽弃不说，还可能出现更严重，且更难以察觉的崩溃事故，细思恐极。

RePlugin

研究全新占坑插件化框架（注意，此时 DroidPlugin 类方案还没有出现）时，就定了个“小目标”：**让 Hook 越少越好**。经过一次次的研究讨论，最终确定只 Hook 一个点：ClassLoader，且要求“坚持到底”，所有改动都是基于此来展开。

- **非坑位方案**：标准的一一对应关系。例如，插件有个 XXXActivity，那么主程序则要求必须也有个 XXXActivity（名字未必一样）来对应。
  一旦插件要添加一个新的 Activity，则对应的，主程序也必须得添加，否则就无法使用这个 Activity。
  坑位方案：有的（如 2013 年的我们）会通过 Fragment 来模拟 Activity，从现一定程度上的灵活。
  真的遇到大需求，如和其它应用 Activity 的交互等，就局限百出，很难称是完整坑位思想。
- **坑位方案**：可以做到“一对多”的关系。例如插件有个 XXXActivity，则运行时，主程序可以将自己的 N1ST1 对应到这个 XXXActivity 上。一旦该 Activity 退出，则 N1ST1 就“空闲”出来。而当 YYYActivity 进来后，又会重新占用 N1ST1。这样就可以做到一个坑位（如 N1ST1）对应多个实体。

  此外，这个 YYYActivity 既可以是已有的，也可以是插件新增的。这样无论插件如何升级，主程序都可以不用动，即可支持新的 Activity。

坑位的思想 如何在product line中体现 ==》 考虑一下

向完美前进——动态编译方案

通过刚才的叙述，像 PM.startActivity 等确实可通过一些方法，来让插件“无成本使用”。但是，像 Provider 的调用（本质是 IContentProvider），Service 的 stopSelf（是 final 的），以及因涉及坑位分配，而不得不需复写相应方法的 Activity 等。这里面存在两个矛盾点：

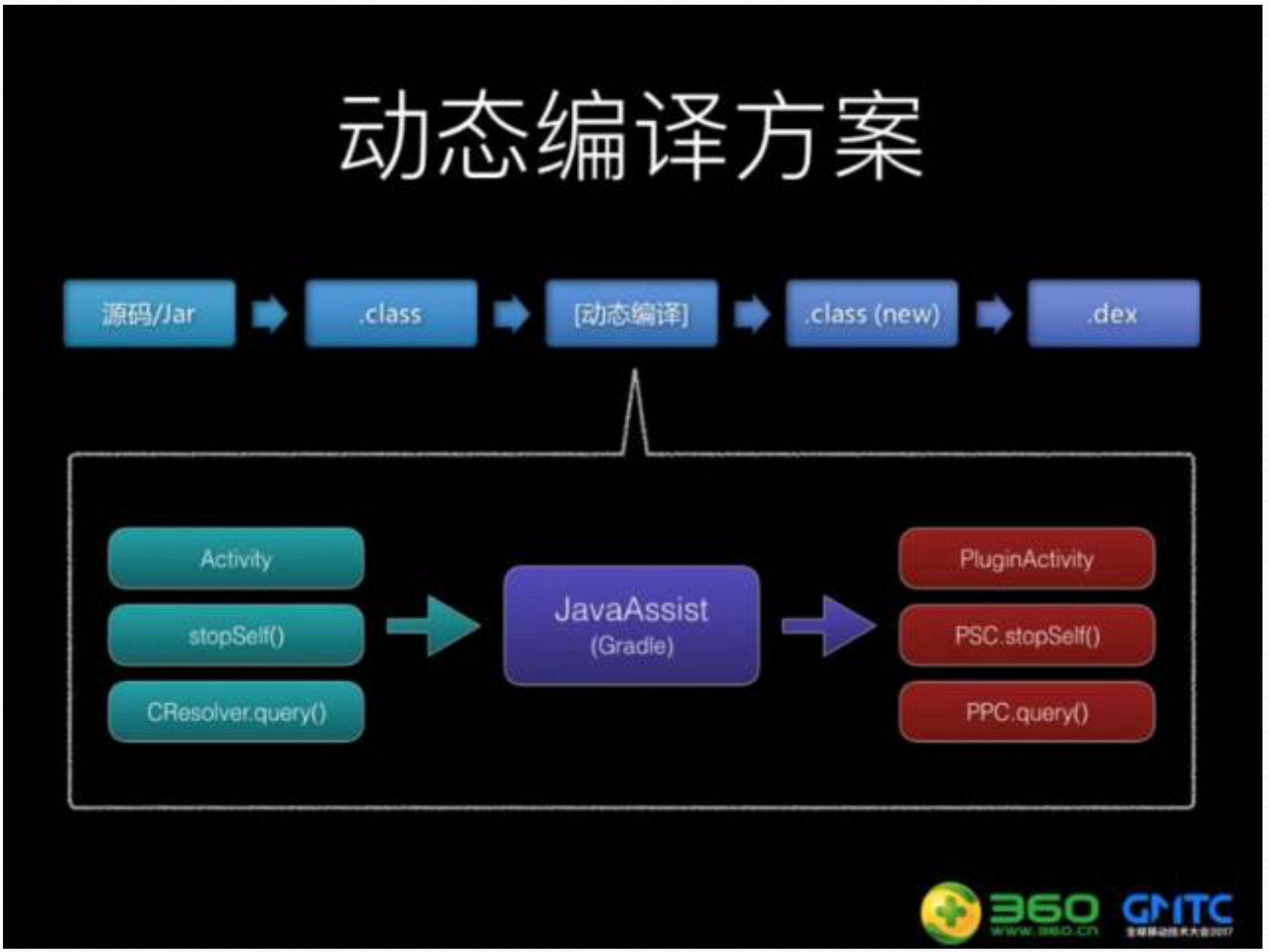
- 若不 Hook，则必须要插件开发者“自行处理”，稍显繁琐，不够完美；
- 一旦 Hook（如尝试 Hook AMS、IContentProvider 等），又破坏了我们坚持的“1 Hook 原则”，进而担心未来出现兼容性问题

利弊之间如何取舍，令人头疼。

针对这个问题，我们的核心理念是：“绝不在 Hook 及稳定性上做任何妥协”，转而创新性的做一套“动态编译方案”，力图从“编译期”来解决这个难题。

大体而言，就是把一些我们认为需要开发者修改的类和方法，借助神奇的 JavaAssist 来做自动化修改，这样可节省开发者的改动成本，达到想要的效果。

动态编译方案


源码/Jar
.class
[动态编译]
.class (new)
.dex
Activity
stopSelf()
CResolver.query()
JavaAssist
(Gradle)
PluginActivity
PSC.stopSelf()
PPC.query()

例如，有个名叫“360 桌面”的应用，它想把自己变成“插件”跑起来。那么，有了“动态编译方案”，结合“插件类库”和框架的支持，最终的效果是——只需改几行 Gradle，就能直接生成一个 APK。这个 APK：

Ideas

- 1. Android app — product line — feature , feature model, constraints.

- 2. Plugin framework: compare with desktop software product line: advantages, disadvantage, usage, modularity, security issue.

- 3. Module load strategy vs. product line configurator

Ideas

- 4. compare Android plugin frameworks:
- 5. use ‘1-n’ mapping from Android to product line. <product line reorganisation>

结论（1）论文扩展的建议

- 1. 把原来方法的 seed部分修改 生成 context-aware的 程序信息 及描述
- 2. 做成一个环
- 3. 在每一次迭代中加入检查

结论2：

- 1. 看看 lili的论文 看suvery 都类似app 坐了什么工作 与SPL有关
- 2. grandle怎么实现
- 3. anroid实验多做一些
- 4. 方法层到应用层
- product line 到 app 有哪些不同点 有哪些约束