IntelliDroid

A Targeted Input Generator for the Dynamic Analysis of Android Malware

Michelle Y. Wong and David Lie

University of Toronto
Department of Electrical and Computer Engineering

NDSS Symposium 2016

Static vs. Dynamic Analysis

- Static analysis: analyze source code or byte code
  - Imprecise
  - No run-time data
- Dynamic analysis: analyze during execution
  - Run-time values → precise

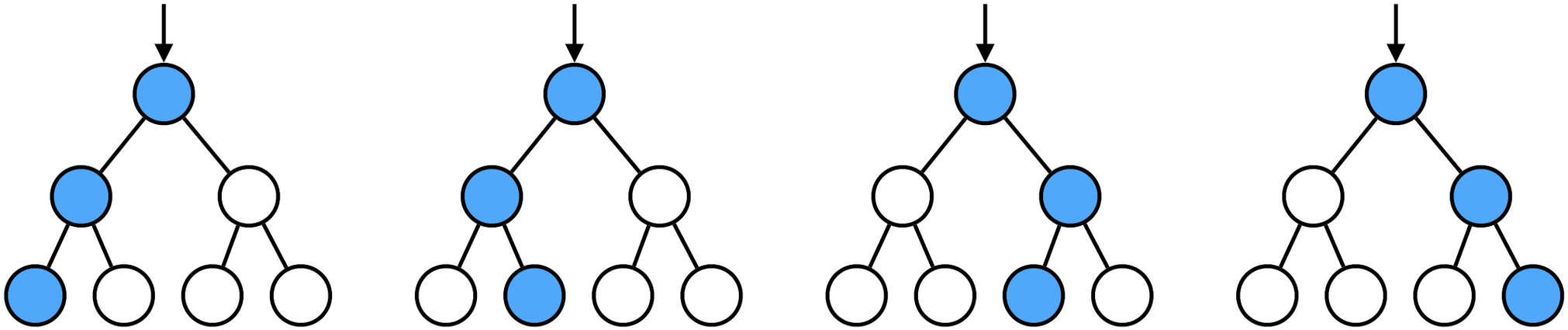
Dynamic Code Coverage

- To detect malicious activity, first have to execute it
- Example:

```
message = <receive confirmation SMS>
if message.number == ‘1234’:
    <malicious action>
```

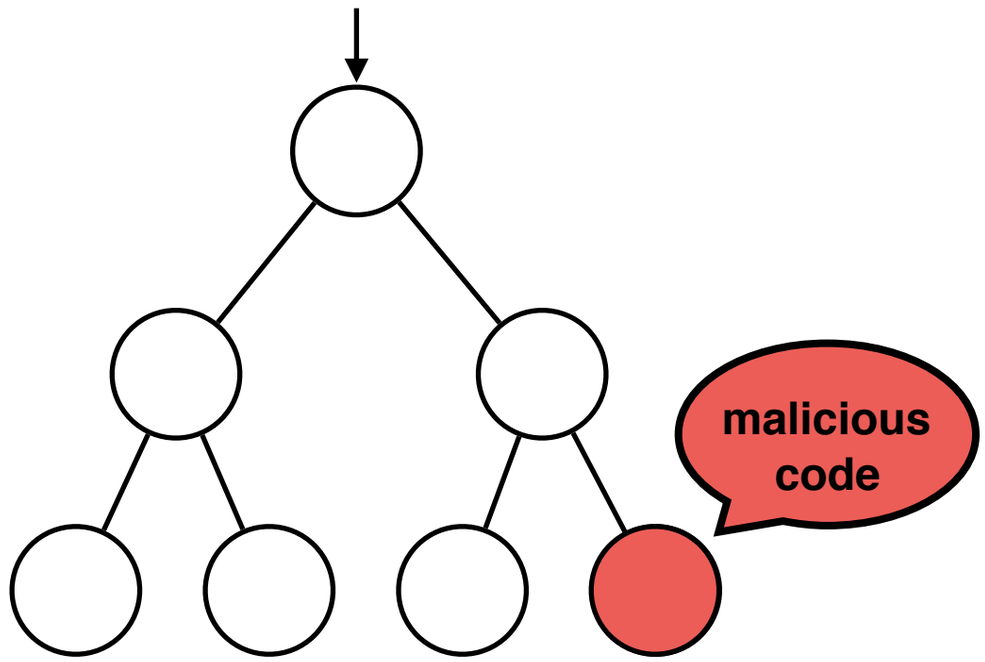
Concolic Testing

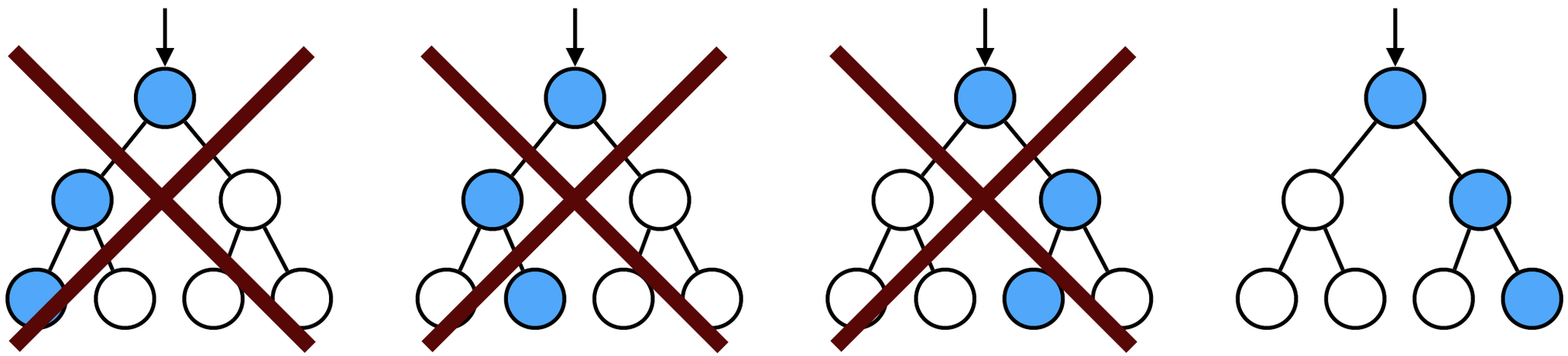
- Run all execution paths in application
- Symbolic execution, solve constraints for inputs



Specific Malicious Paths

- Malicious activity only executed in certain parts of the code



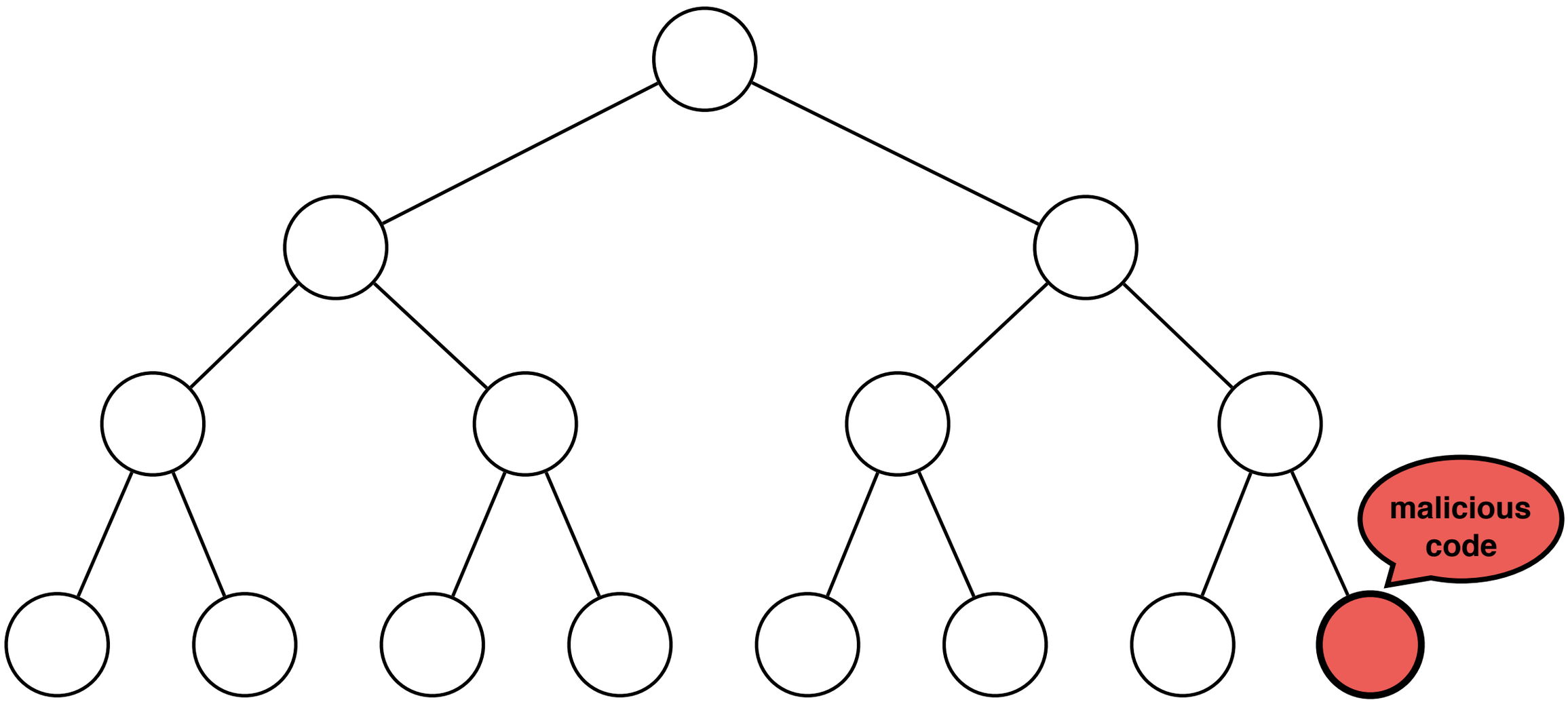


IntelliDroid

- Targets specific parts of the application
  - Input generator for existing dynamic detector
  - Hybrid static and dynamic design
- Implemented for Android
- Improve malware analysis and detection

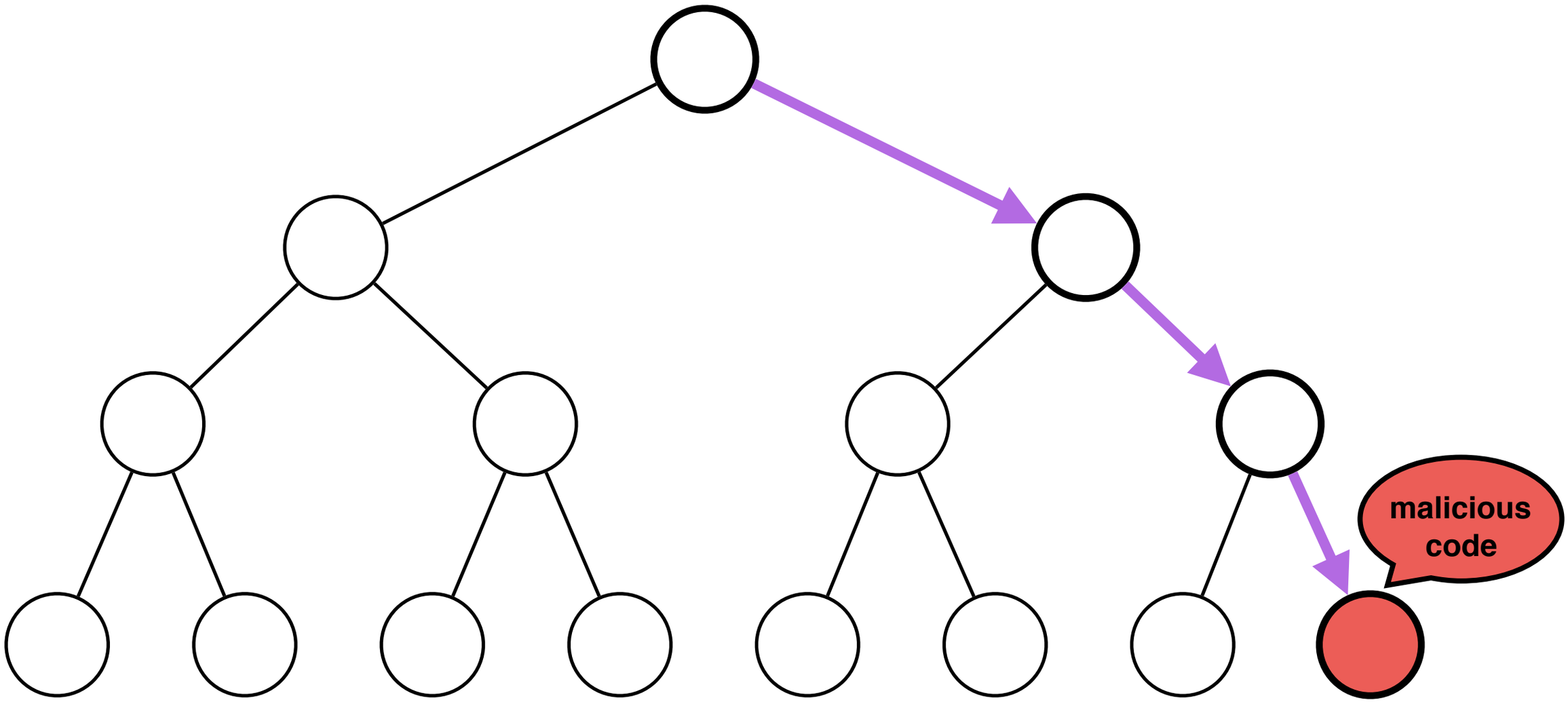
Target Malicious Paths

- Malicious activity present only in certain parts of the code



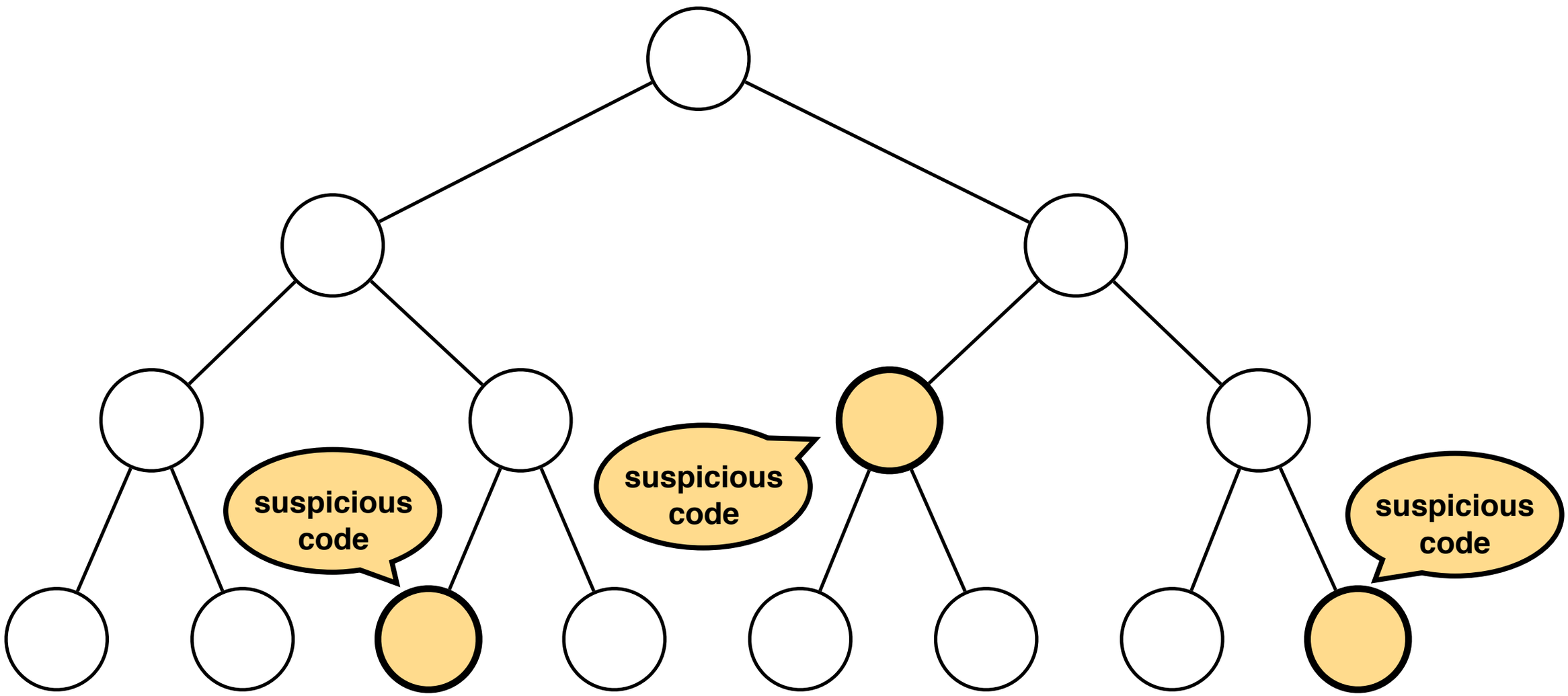
Target Malicious Paths

- Use static analysis to look for call paths to malicious activity



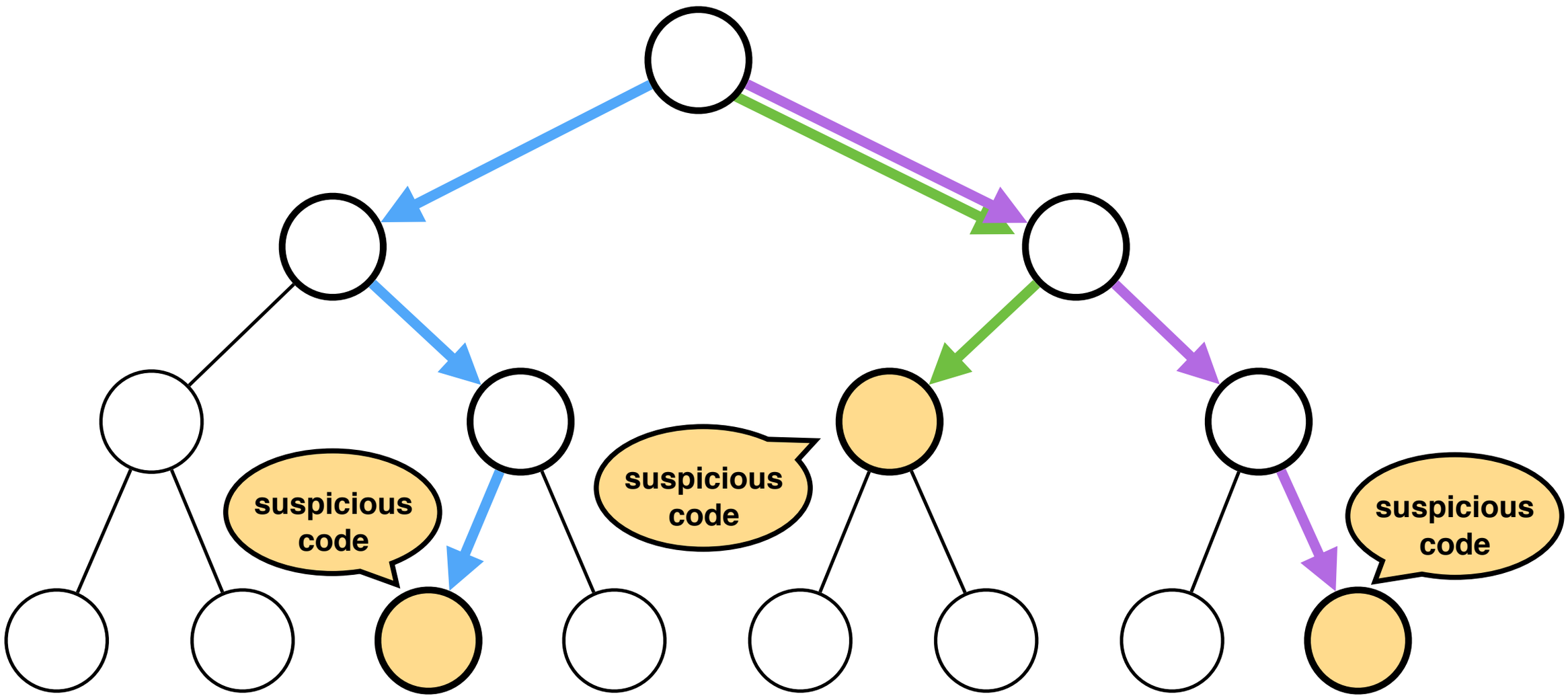
Target Over-Approximation

- Target over-approximation of malicious behaviors



Target Over-Approximation

- Target over-approximation of malicious behaviors



Targeted Methods

- Use method invocations as over-approximation
  - Depends on attached dynamic malware detector
- Existing dynamic detectors for Android:

✔ Method invocations

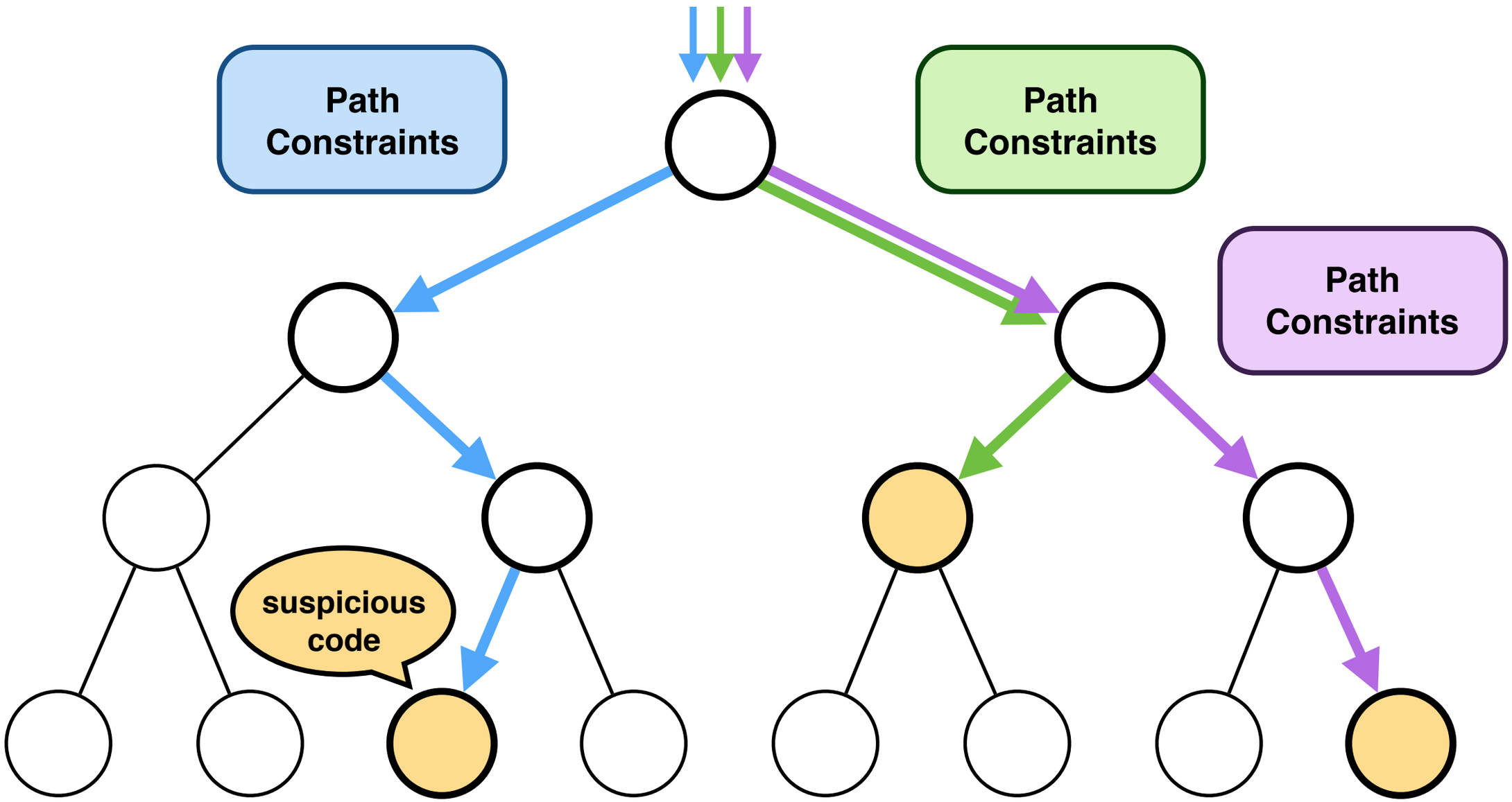
✔ System call traces

✘ Anomaly detection

Dynamic Tool	Goal	Features for Analysis
AASandbox [10]	Monitor behavior via tracking of system calls	System calls
Andromaly [36]	Malware detection via system resource usage	Low-level device features (e.g. battery usage, CPU load)
CopperDroid [39]	Monitor behavior via system call tracking	System calls
Crowdroid [12]	Monitor behavior via tracking of system calls	System calls
DroidBox [18]	Sandbox to monitor external accesses	Sink API methods
DroidRanger [50]	Detect malware using pre-specified behavioral footprints and heuristics	Sequence of API method invocations and parameters
DroidScope [39]	Plugins for API tracking, instruction tracing, and taint tracking	API methods; source/sink API methods
RiskRanker [39]	Detect malware using known vulnerability signatures	Sequence of API method invocations
TaintDroid [19]	Detect privacy leakage	Source/sink API methods
VetDroid [47]	Malware detection via permission use behavior	Permission requests (can be mapped to API methods)

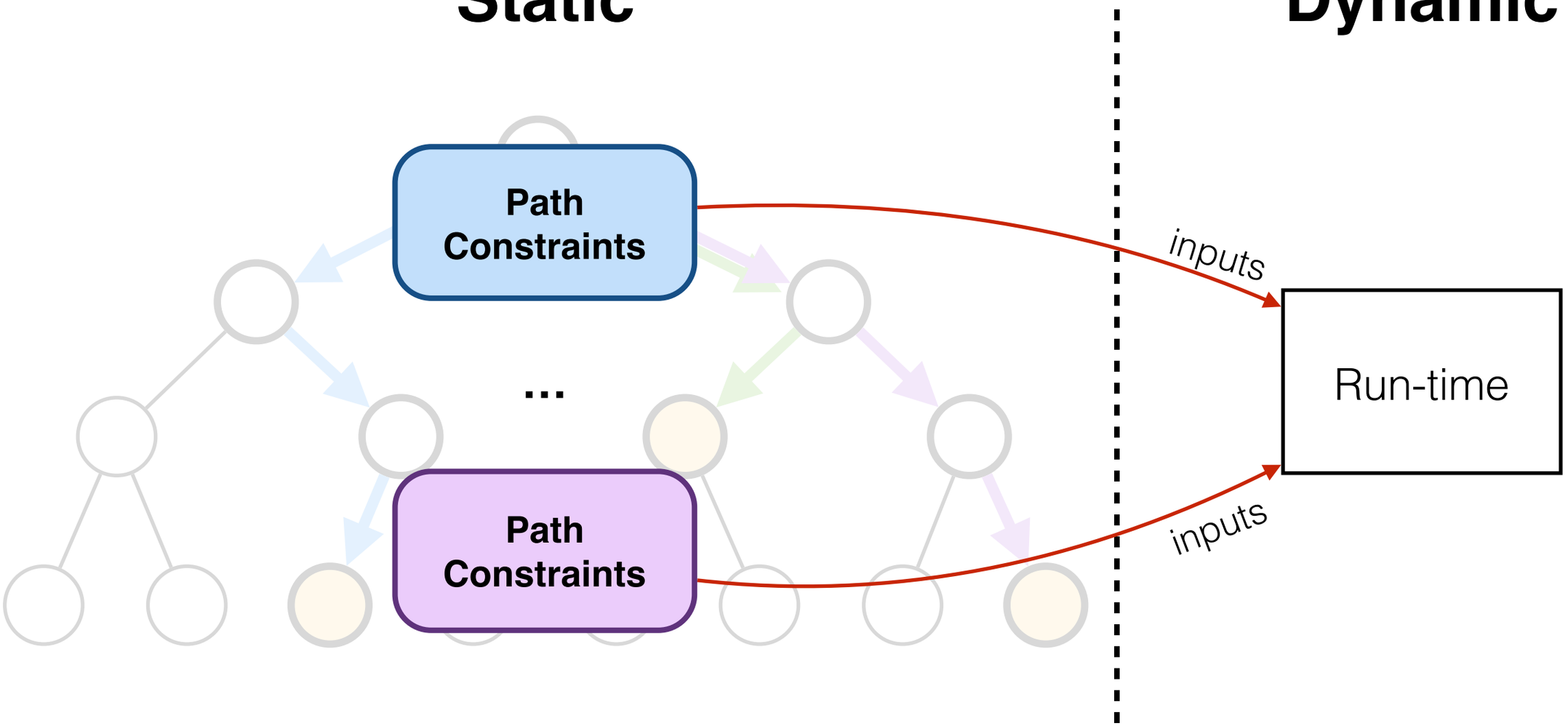
Static Constraint Extraction

- Extract constraints on inputs that can trigger targeted paths



Targeted Input Injection

- Inject constrained inputs to execute paths at run-time



Challenges

- Finding targeted paths using static analysis
  - Imprecision?
- Executing path to suspicious code
  - Dependencies between paths?
- Run-time input injection
  - Where to inject?

Static Imprecision

- Static analysis cannot determine run-time values
- Example:

```
message = <receive confirmation SMS>
if message.number == <file A>.text:
    <malicious action>
```

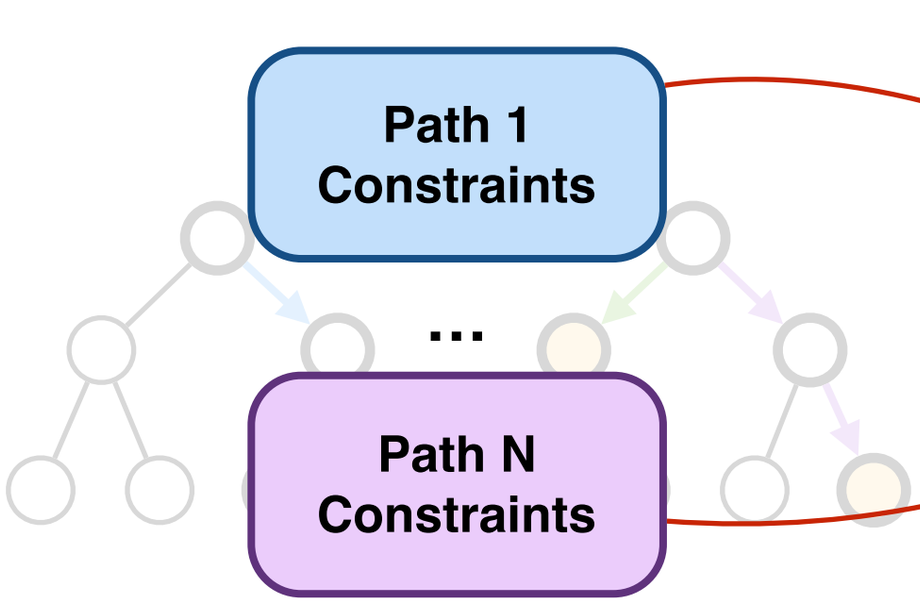
Constraint

<SMS message>.number == <file A>

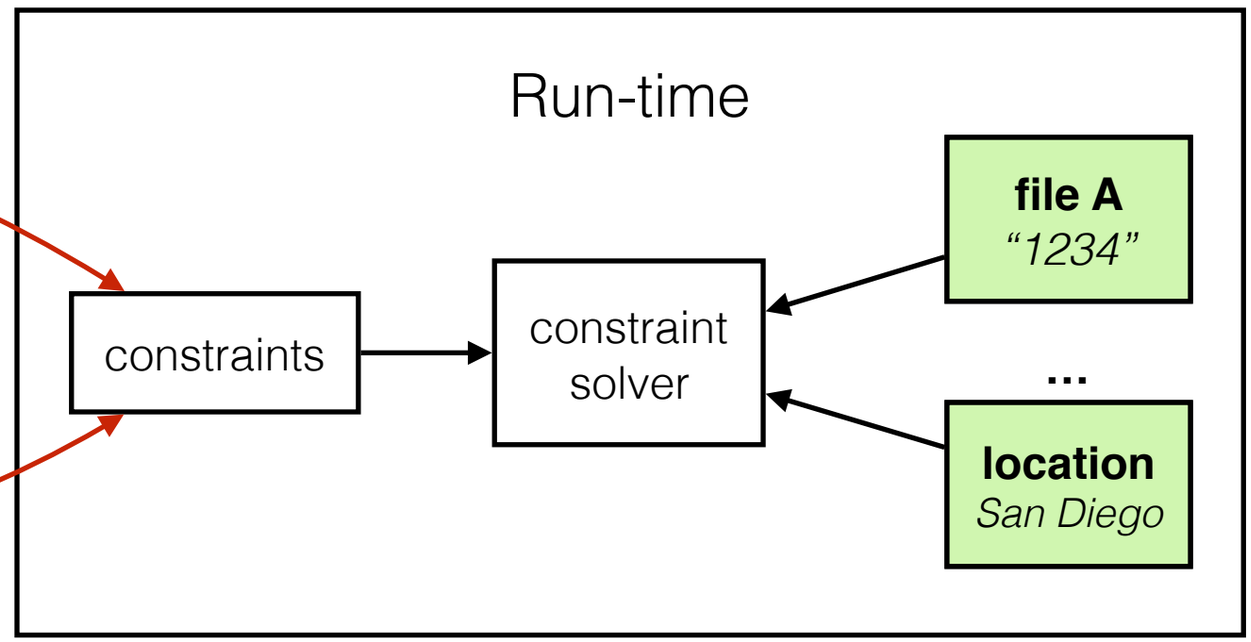
Using Run-time Data

- Solve constraints at run-time (with run-time data)

Static



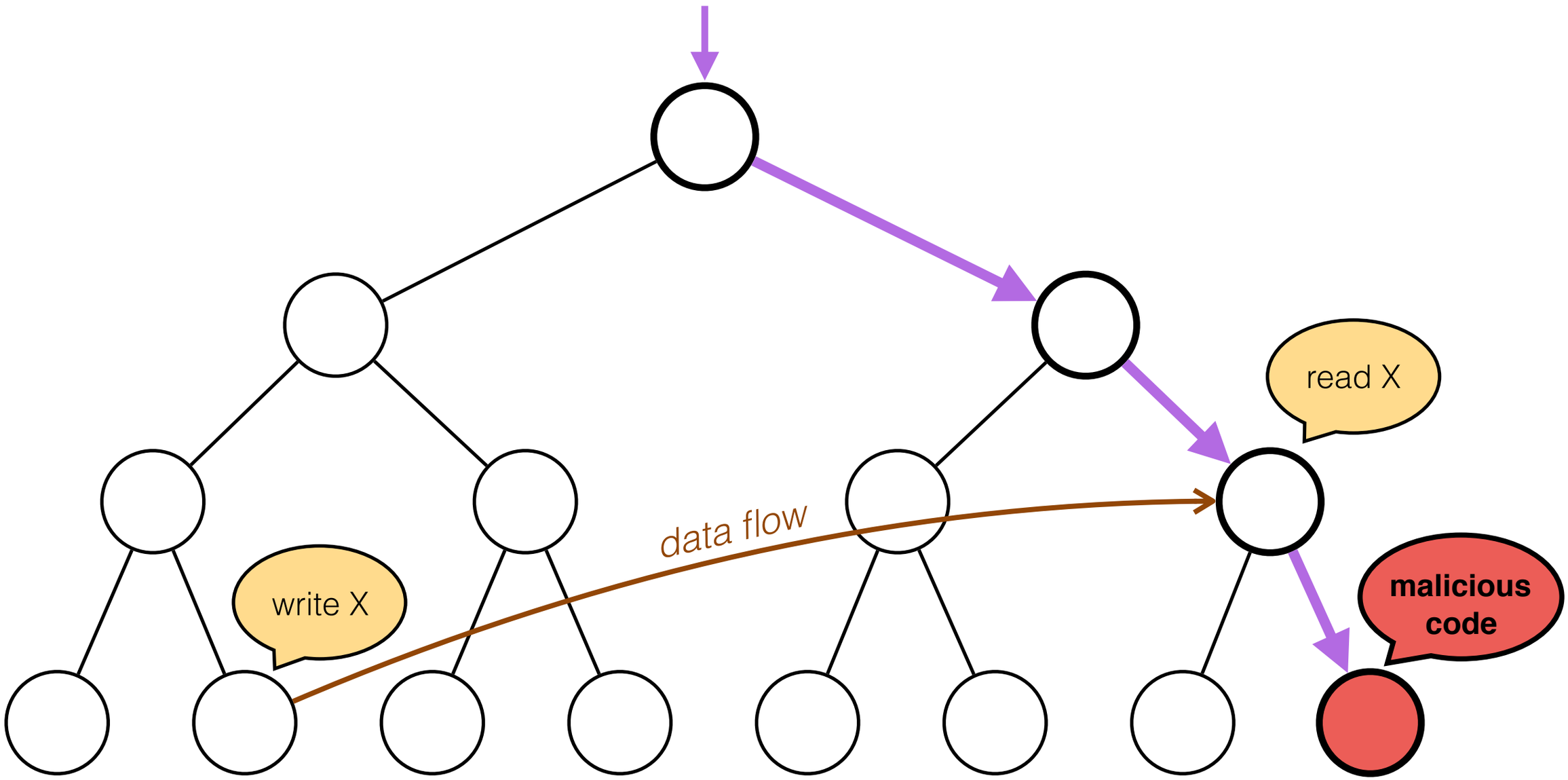
Dynamic




```
<SMS message>.number == "1234"
```

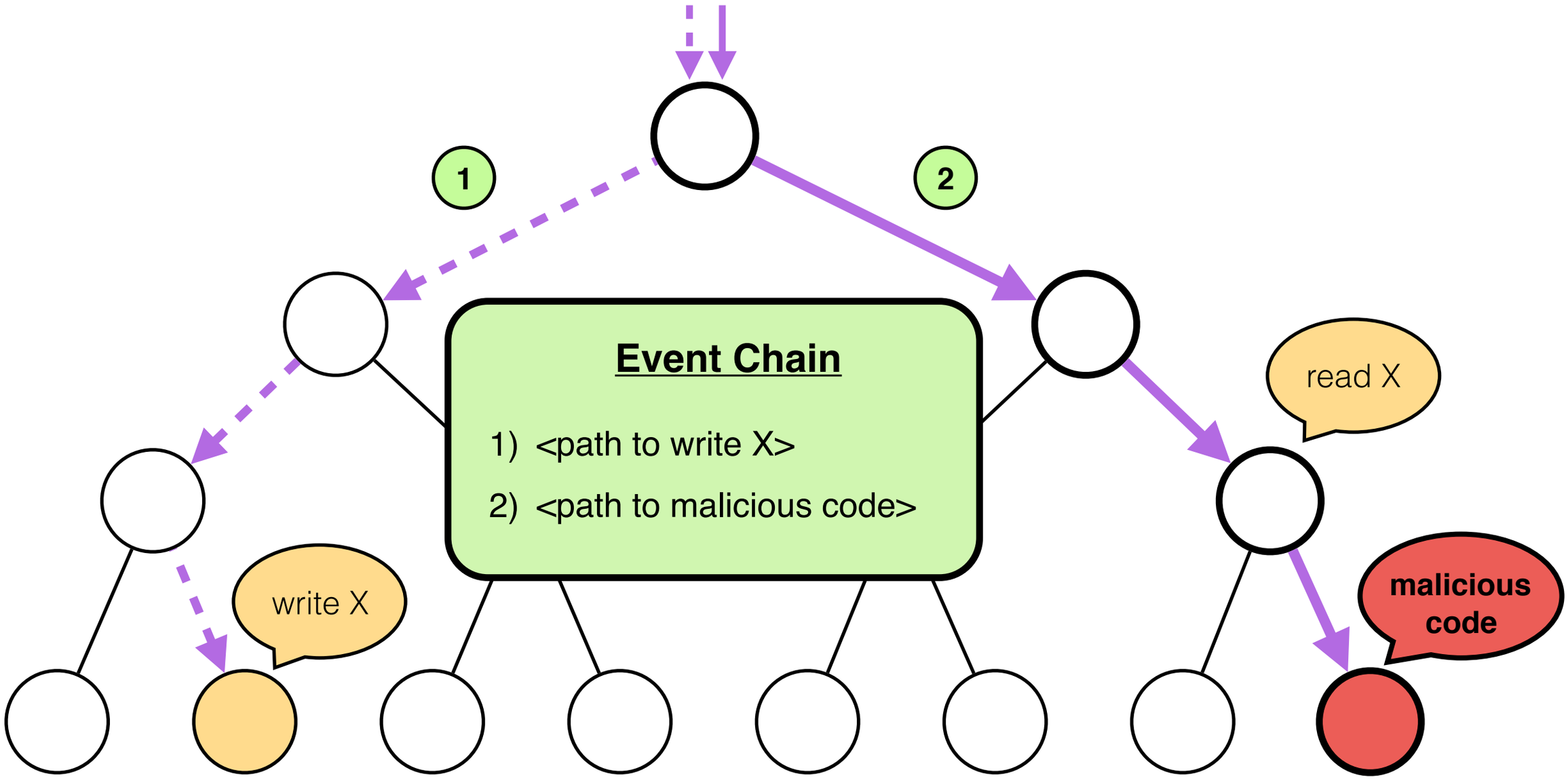
Path Dependencies

- Data- and control-flow dependencies between call paths

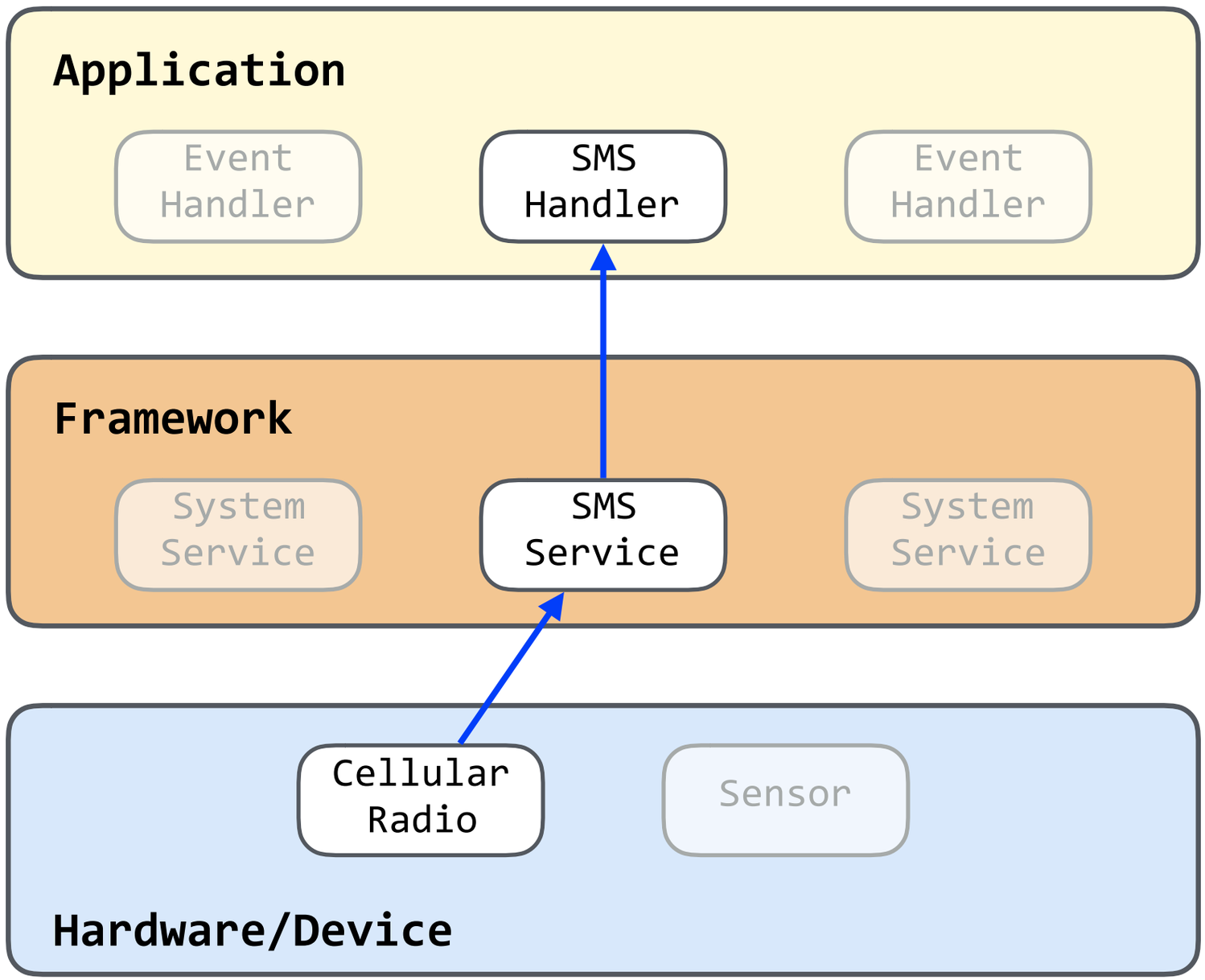


Path Dependencies

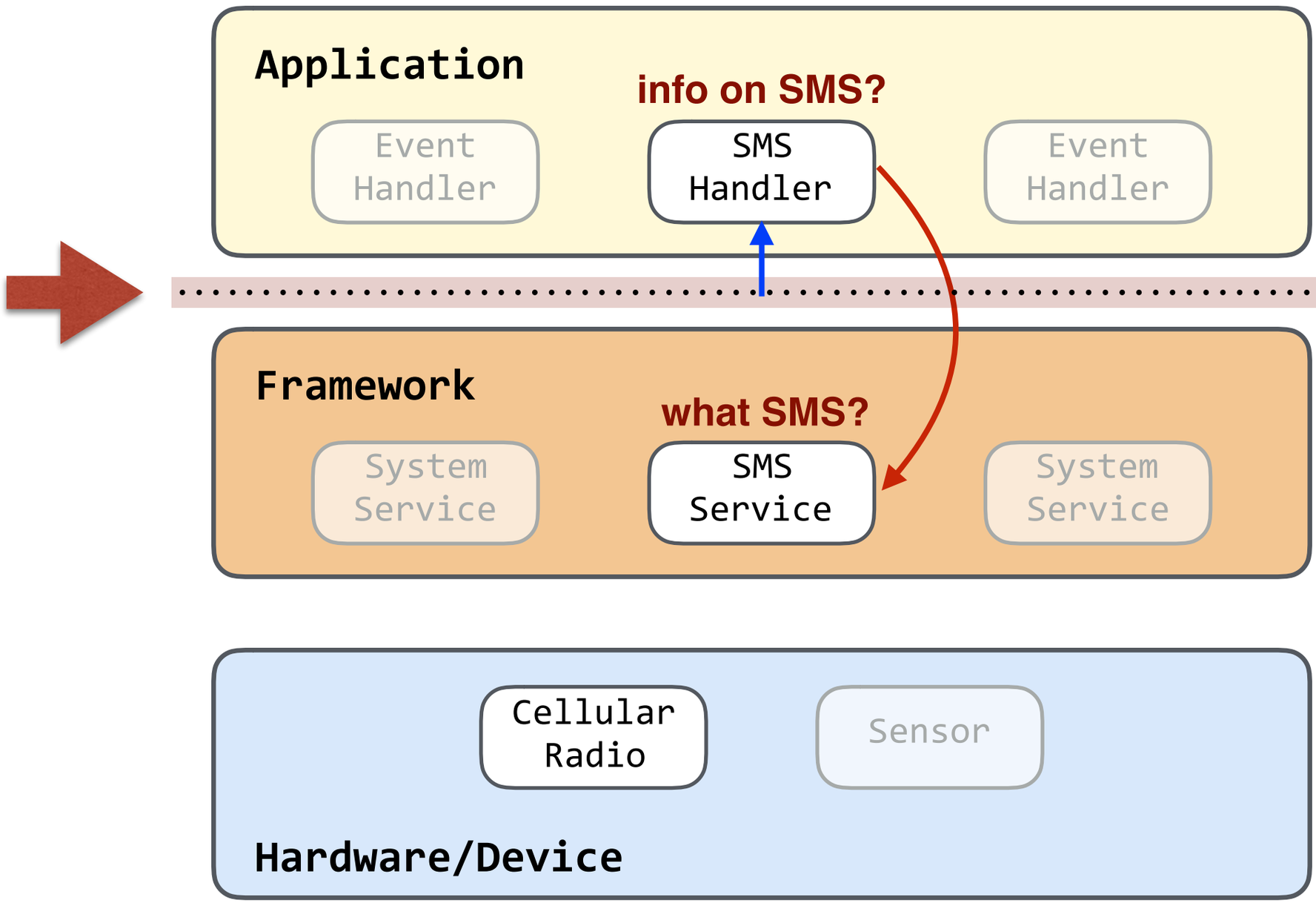
- Data- and control-flow dependencies between call paths



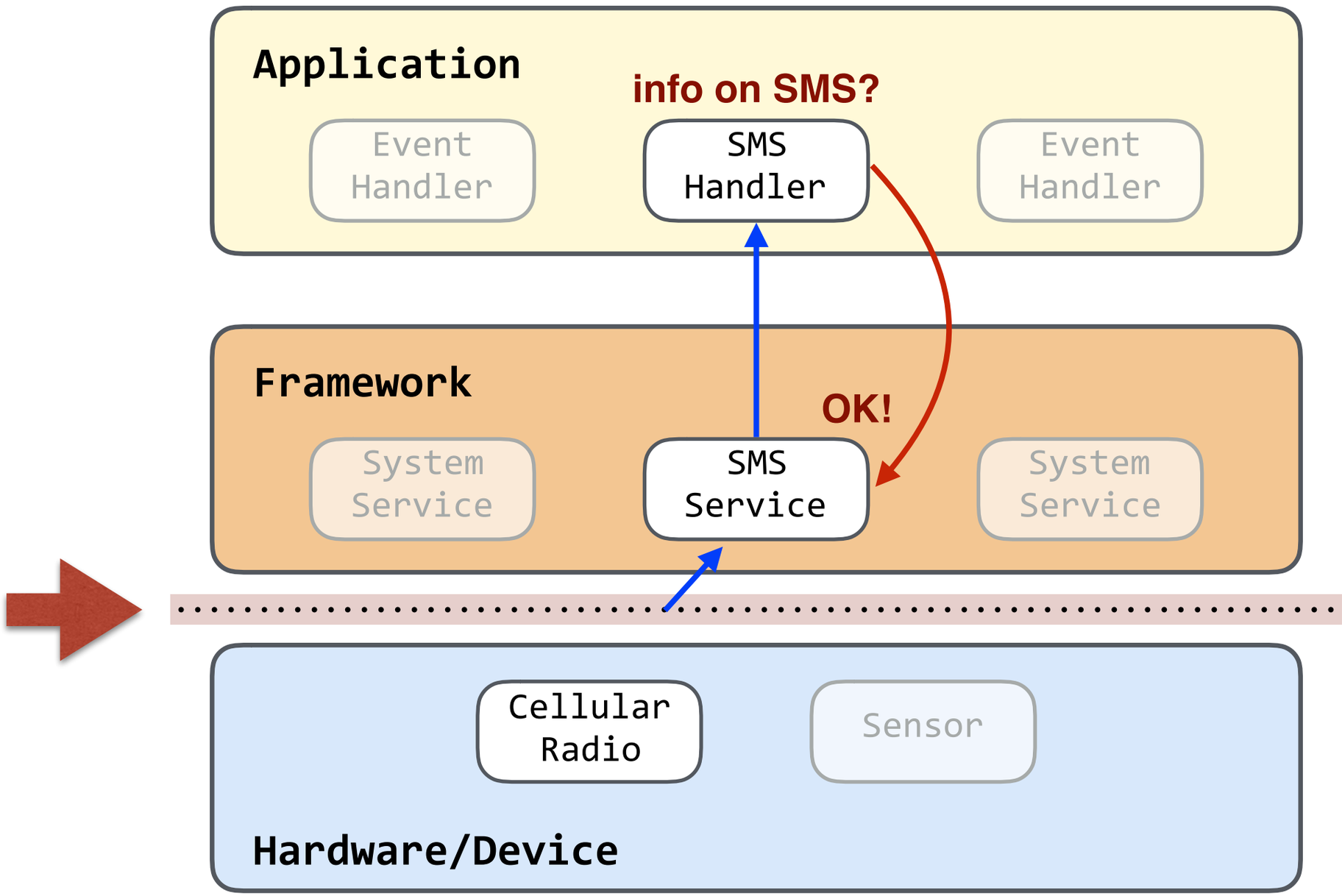
Run-Time Injection



Application Injection



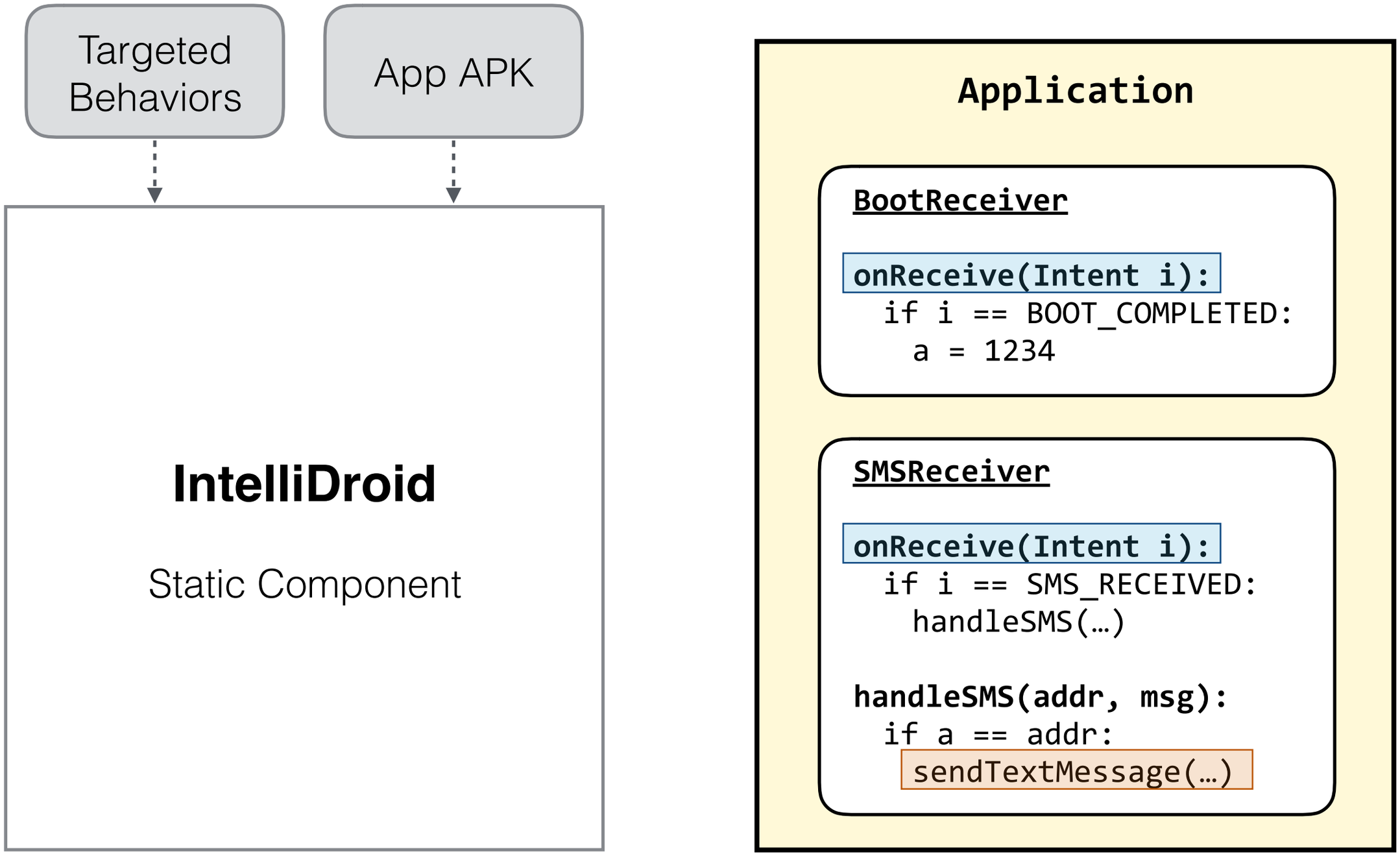
Device-Framework Injection



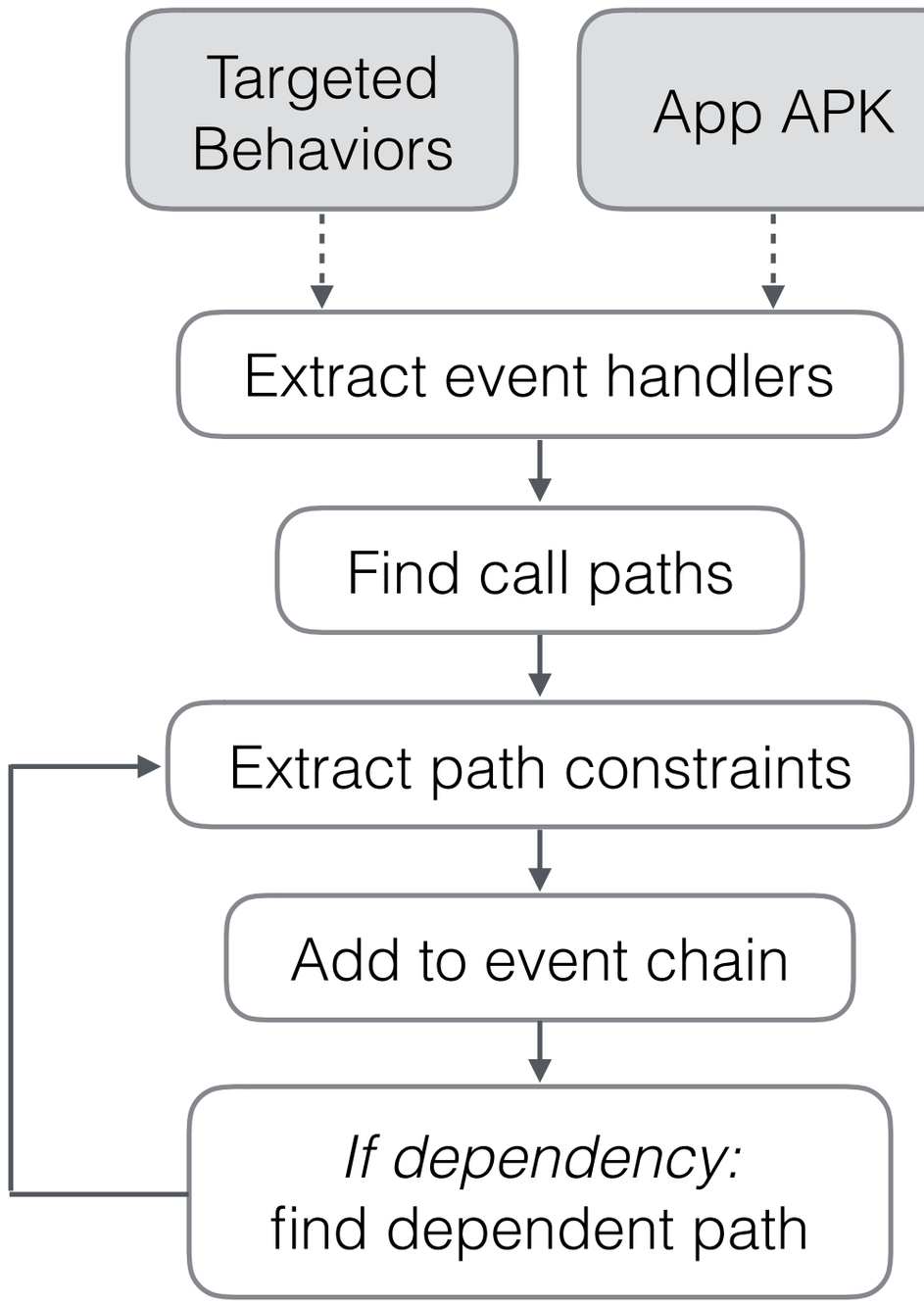
Contributions

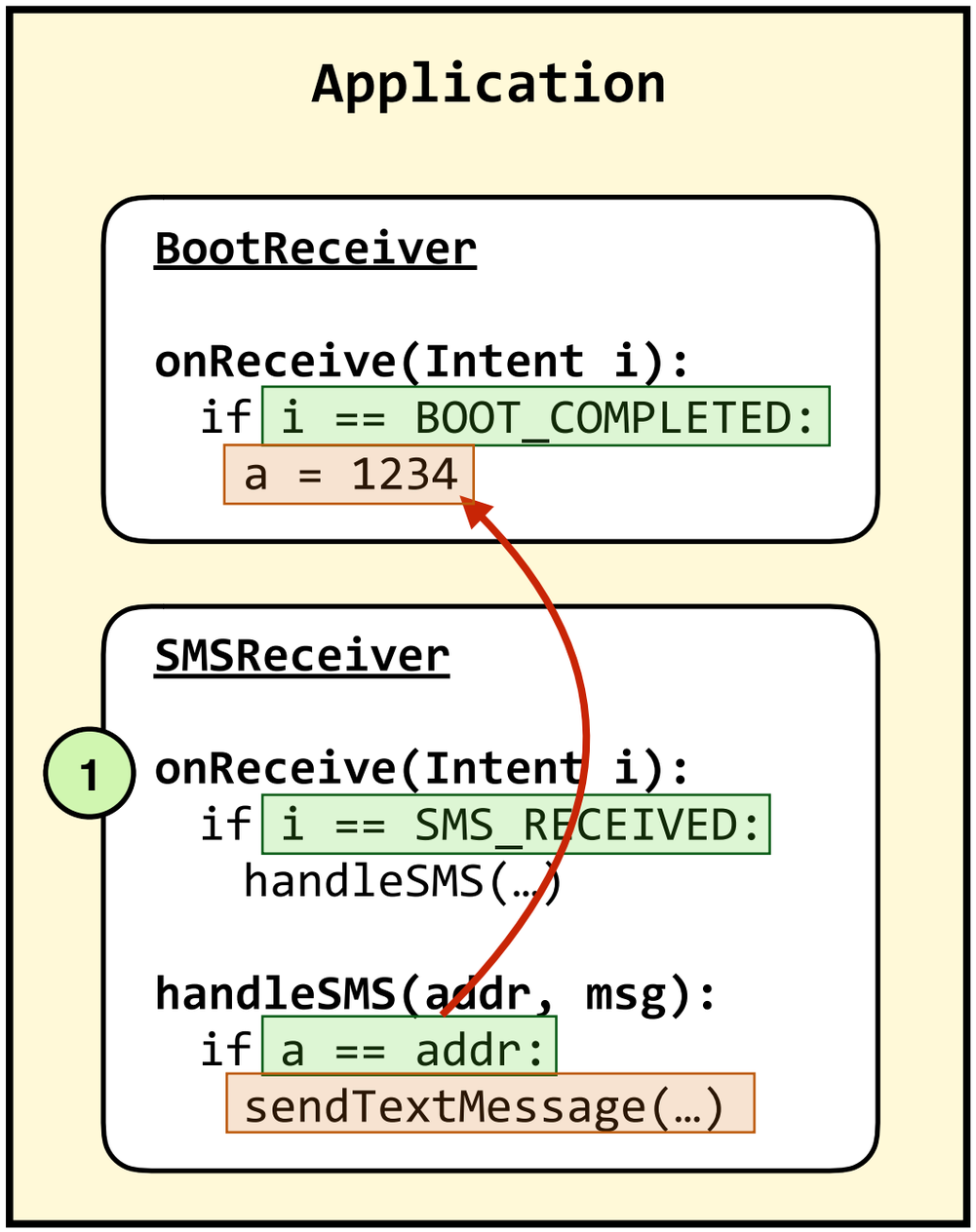
- Static imprecision
  - Dynamic constraint solving with run-time values
- Path dependencies
  - Event chains
- Consistent input injection
  - Device-framework injection

Static Component

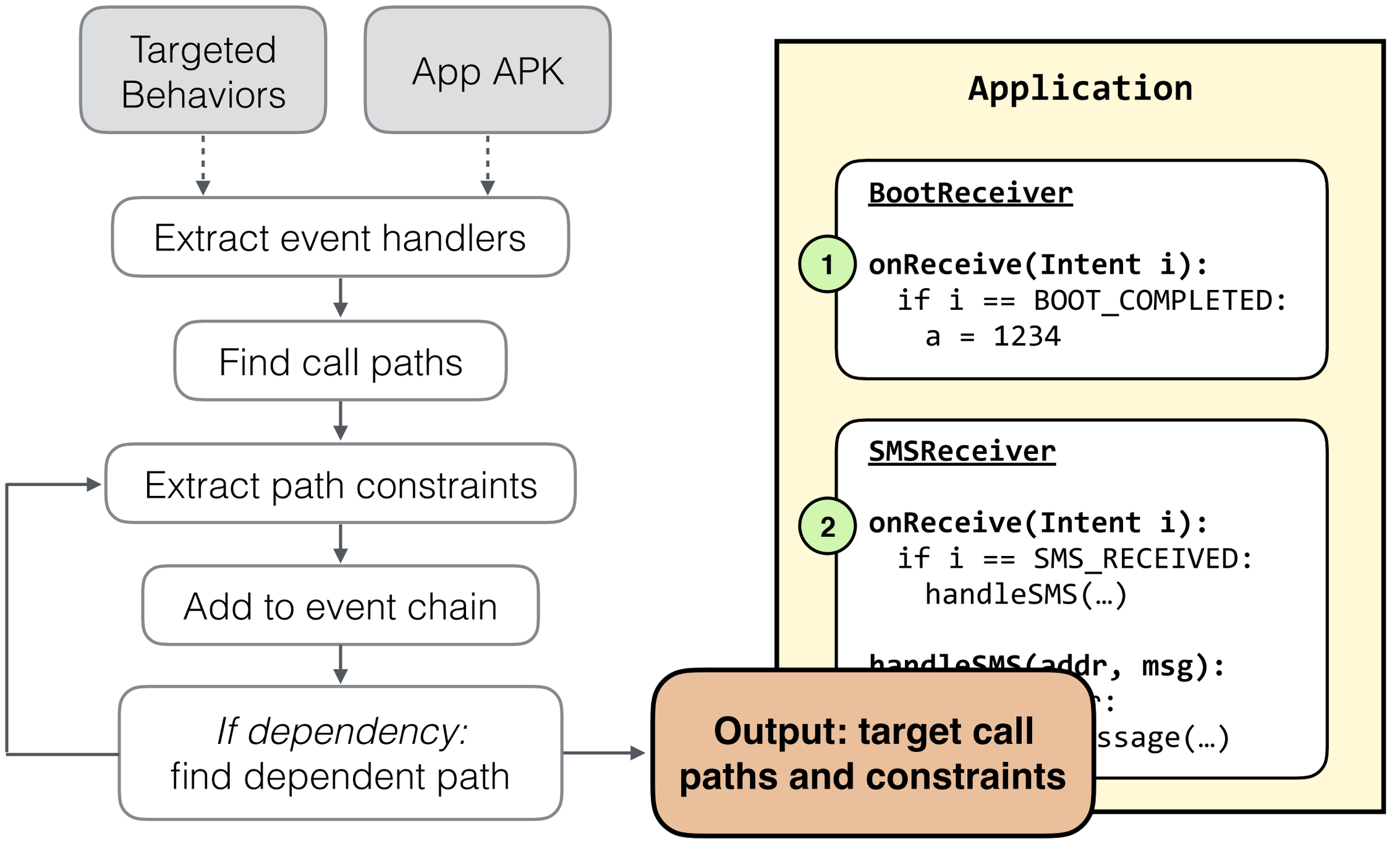


Static Component





Static Component



Implementation

- Static analysis (Android-specific): WALA [1]
- Dynamic component:
  - Client program (Python)
    - Constraint solver: Z3 [2]
  - Custom Android OS
    - `IntelliDroidService`: system service to receive input information and inject events

[1] Watson libraries for analysis. http://wala.sourceforge.net. Accessed: September 2014.

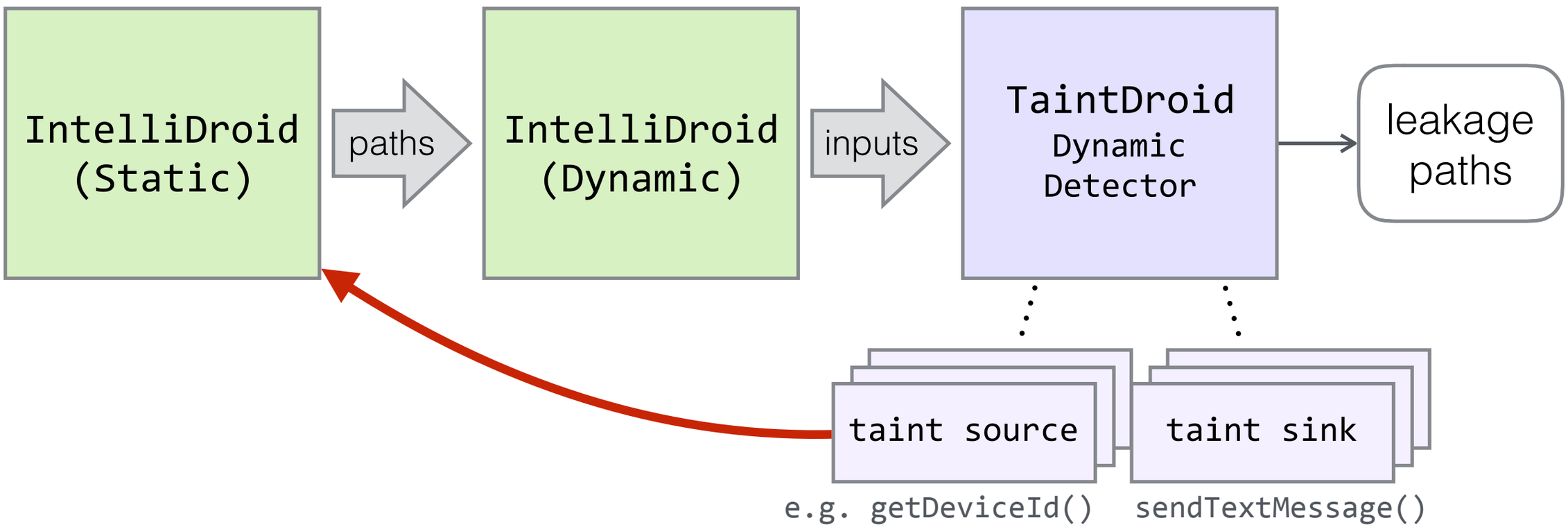
[2] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In Tools and Algorithms for the Construction and Analysis of Systems, pages 337–340. Springer, 2008.

Evaluation

- Can IntelliDroid be integrated with existing dynamic malware detectors?
- Can it execute targeted behaviours at run-time?
- Is the analysis time reasonable?

Integration with TaintDroid

- Attached to TaintDroid (dynamic taint tracking tool)
- Input generator to execute taint sources and sinks



IntelliDroid-Driven TaintDroid

- Tested on 26 privacy leaks in 17 malicious apps [1,2]
- IntelliDroid: Triggered and detected all leaks
  - Monkey: Missed 21 leaks
- Executed < 5% of application code

[1] Yajin Zhou and Xuxian Jiang. Dissecting Android malware: Characterization and evolution. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, pages 95–109. IEEE, 2012.

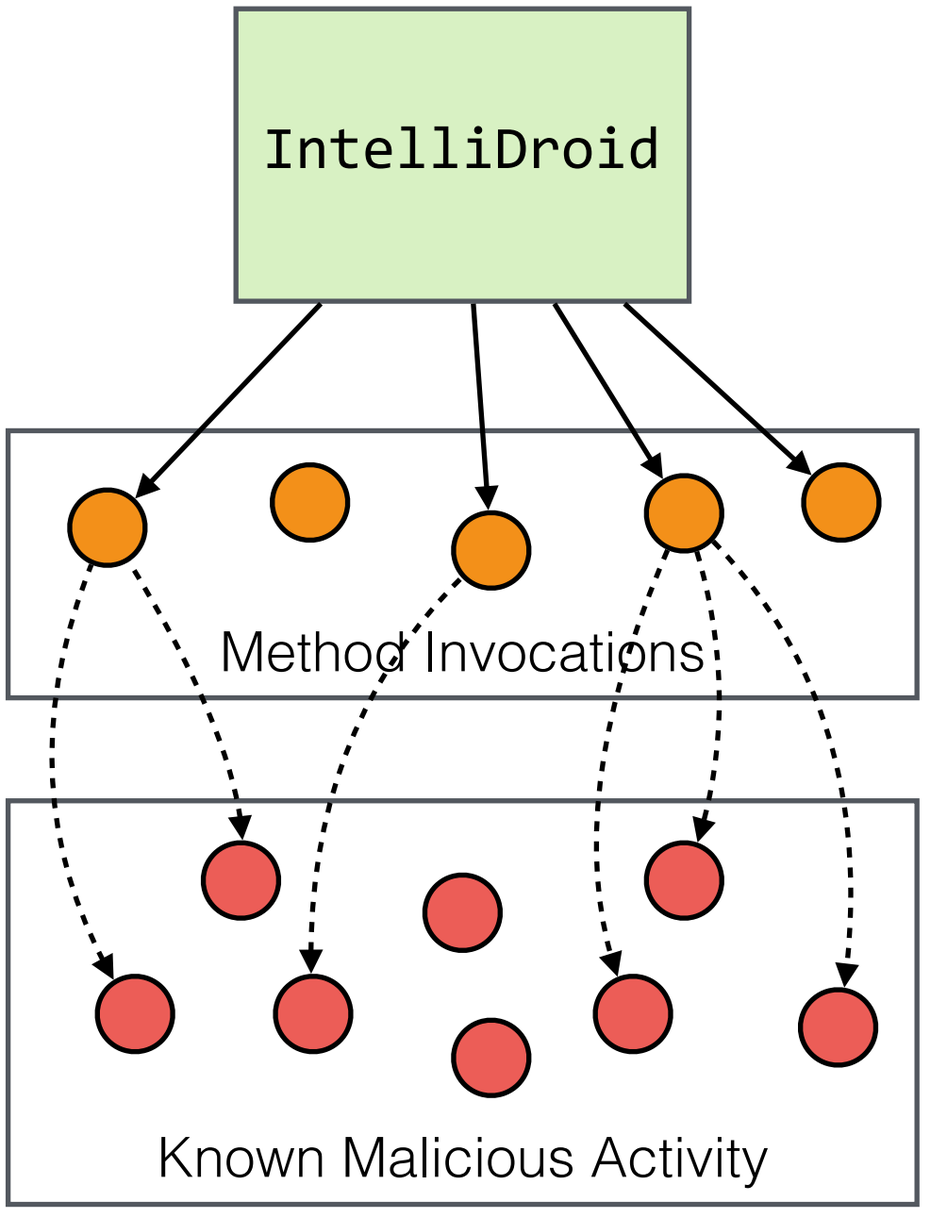
[2] M. Parkour, “Contagio mobile,” 2015, http://contagiominidump. blogspot.ca/, Last Accessed Aug, 2015.

Targeted Input Injection

- Target malicious behaviours in Android Malware Genome and Contagio

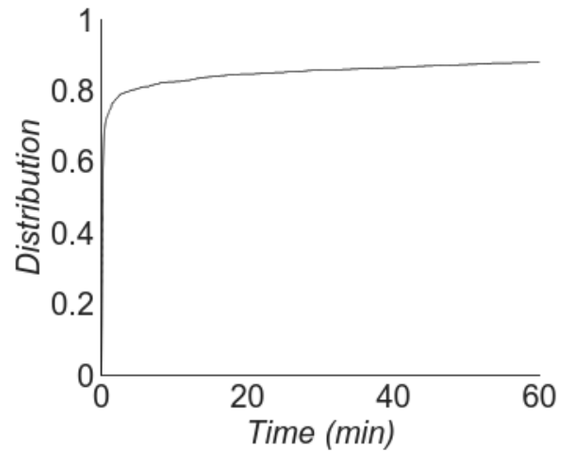
- Triggered 70 out of 75 behaviours

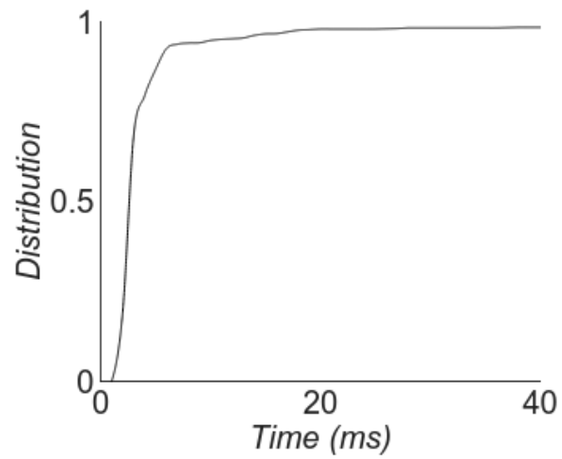
- Missed behaviors:
  - Encoding
  - File dependencies (currently not supported)



Performance

- Scales for large-scale analysis of applications
- Static analysis:
  - 138.4s per application
- Dynamic constraint solving:
  - 4.22ms per targeted call path





[1] David Barrera, Jeremy Clark, Daniel McCarney, and Paul C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In Proceed- ings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '12, pages 81–92, New York, NY, USA, 2012. ACM.

Conclusion

- Targeted input generation for effective dynamic malware detection
- IntelliDroid
  - Static constraint extraction with run-time data
  - Event chains and framework injection
- Integrated with existing dynamic tools (TaintDroid)
- Improve effectiveness, reduce amount of code to be executed (< 5%)