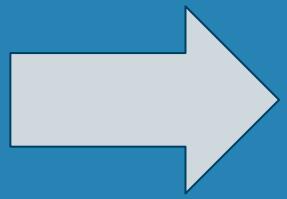
# SPDS: A Scalable Solution for Static Analysis

As presented by:
Bjorn Prollius, Levi Stevenson, and Favour Akinloye

# Porting SPDS to SWAN

SPDS




prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Benefits of porting SPDS to SWAN

**Quickly fix issues without having to wait for the author to fix them**

Use sensible structures for representing the SPDS IR

Eventually wrap the Boomerang solving code with proper exception handling

Increase overall ease of development

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Benefits of porting SPDS to SWAN

Quickly fix issues without having to wait for the author to fix them

**Use sensible structures for representing the SPDS IR**

Eventually wrap the Boomerang solving code with proper exception handling

Increase overall ease of development

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Benefits of porting SPDS to SWAN

Quickly fix issues without having to wait for the author to fix them

Use sensible structures for representing the SPDS IR

**Eventually wrap the BoomerangPDS solving code with proper exception handling**

Increase overall ease of development

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Benefits of porting SPDS to SWAN

Quickly fix issues without having to wait for the author to fix them

Use sensible structures for representing the SPDS IR

Eventually wrap the Boomerang solving code with proper exception handling

**Increase overall ease of development**

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# SPDS

**S**ynchronized

**P**ush

**D**own

**S**ystems

Solves the issue of creating an analysis that is flow, context, and field sensitive

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Context Sensitivity

```
Main(){

    x = 0;
c1: x = id(x);

    x = 1;
c2: x = id(x);
}
```

```
id(y){
    z = y;
    return y;
}
```

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Context Sensitivity

Impractical to store every callsite, especially for recursive programs.

We resort to k-limiting

K-limiting is a source of imprecision

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Field Sensitivity

```
a = source();

while(true){
    b = new DS();
    b.f = a;
    a = b;
}
sink(d);
```

The number of field accesses potentially grows beyond manageability

prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# Combining flow, context and field

A highly precise analysis ideally combines all 3 sensitivities

HOWEVER

Tracking all 3 sensitivities is not scalable.

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Context, Field, and Flow-sensitive

Stored information:

EVERY program statement

prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# Context, Field, and Flow-sensitive

Stored information:

EVERY program statement

EVERY calling context

prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# Context, Field, and Flow-sensitive

Stored information:

EVERY program statement

EVERY calling context

For different combinations of field accesses.

prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# SPDS

Does not resort to k-limiting

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# SPDS

Does not resort to k-limiting

Reduces complexity:
from $|\mathbf{F}|^{3k} \Rightarrow |\mathbf{S}||\mathbf{F}|^2$

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

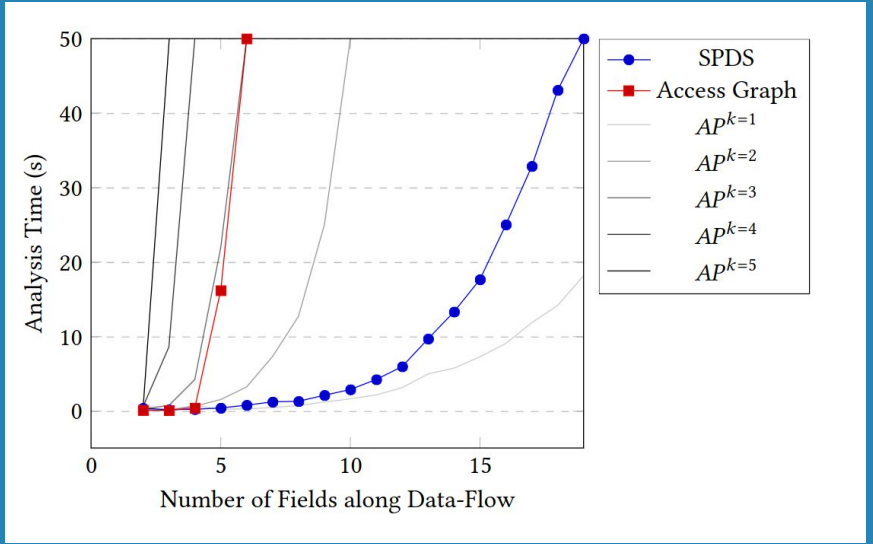
# SPDS

Does not resort to k-limiting

Reduces complexity:
from $|\mathbf{F}|^{3k} \Rightarrow |\mathbf{S}||\mathbf{F}|^2$

Scales similar to:

k=1 with precision of k=∞




prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# Undecidability

False positives exposed only when:

Context-insensitive data-flow path coincides with field-sensitive data-flow

OR

Context-sensitive data-flow coincides with field-insensitive data-flow

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Undecidability

False positives exposed only when:

Context-insensitive data-flow path coincides with field-sensitive data-flow

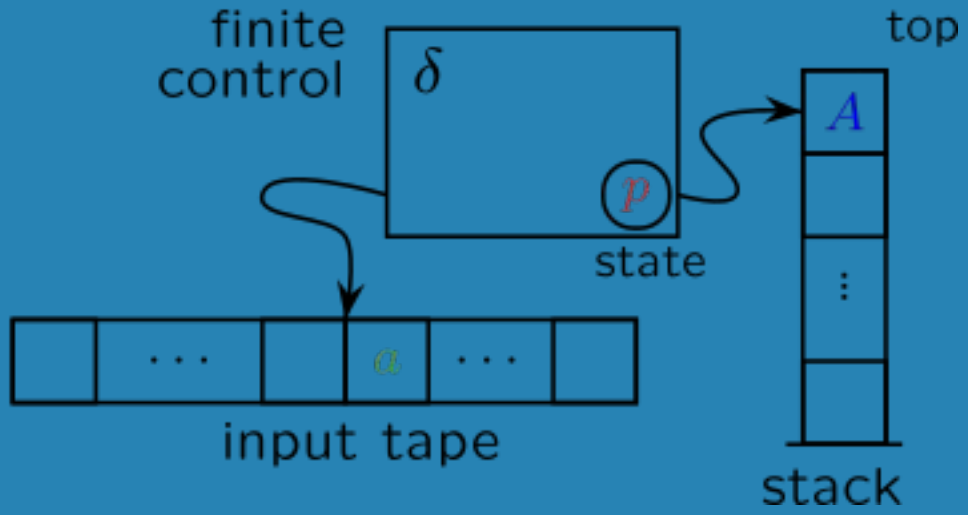
OR

Context-sensitive data-flow coincides with field-insensitive data-flow

**These corner cases are very rare in practice**

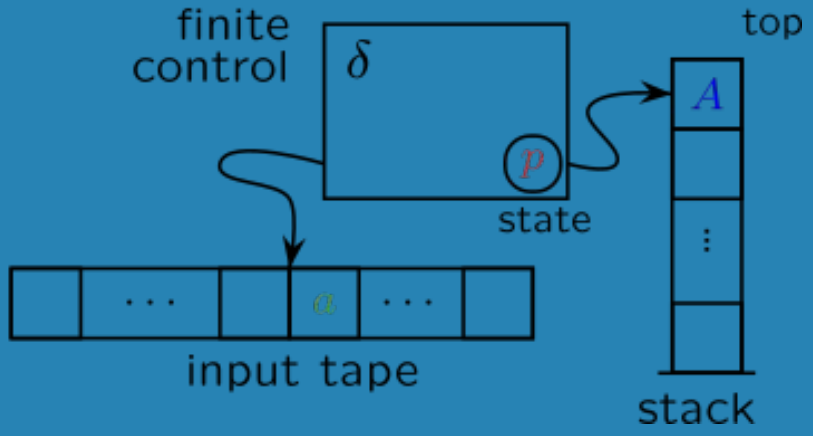
prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

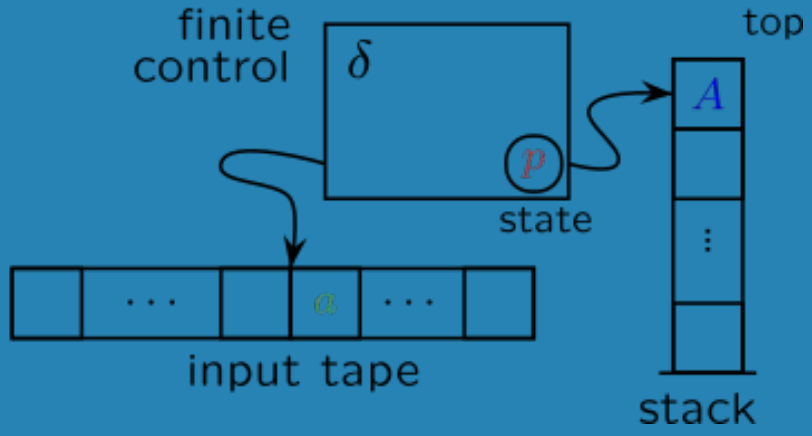
# What is a pushdown system?




prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

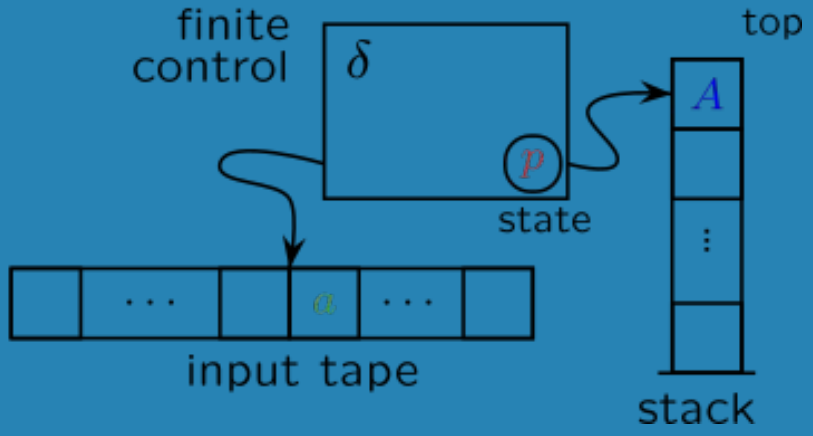
# What is a Synchronized Pushdown System?

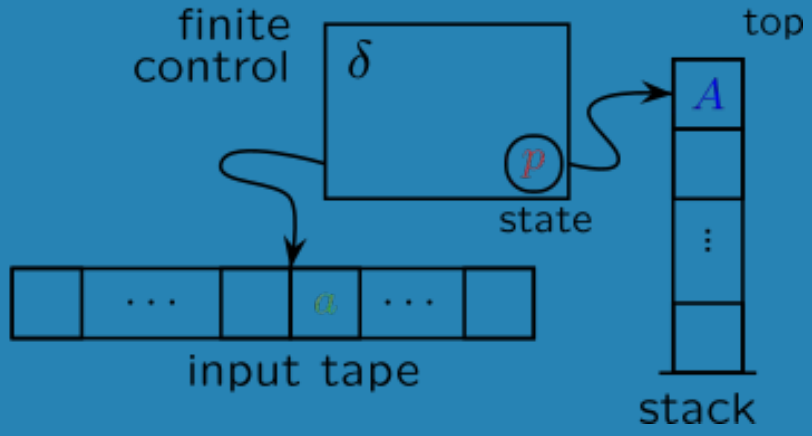



prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What is a Synchronized Pushdown System?

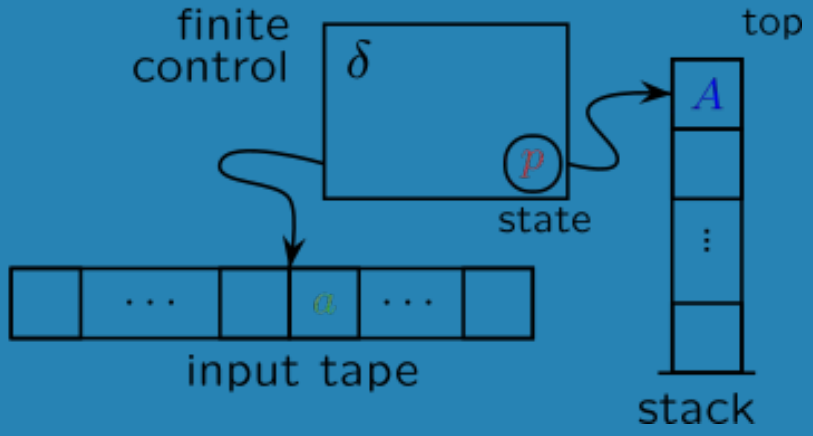


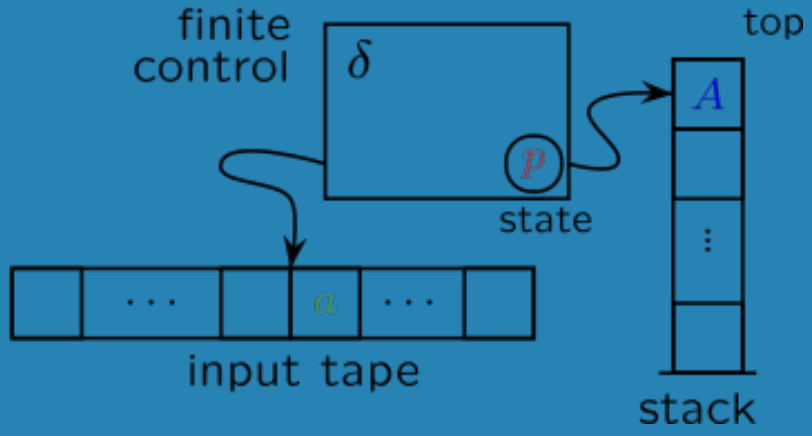

**Context** and **Flow** sensitive Automata

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What is a Synchronized Pushdown System?



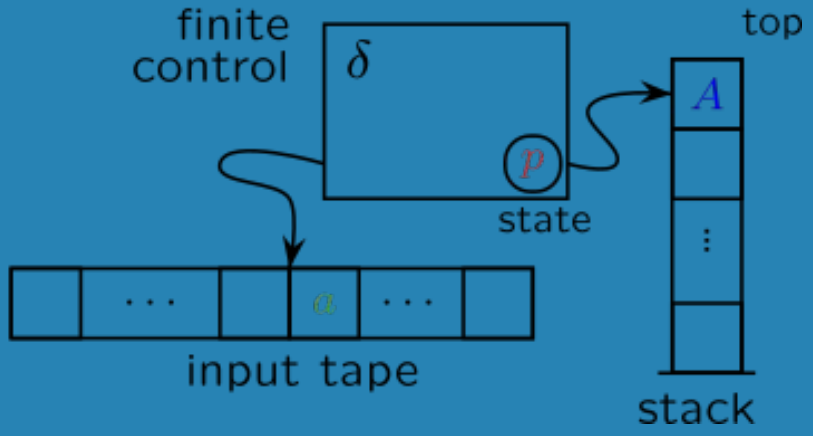

**Context** and **Flow** sensitive Automata



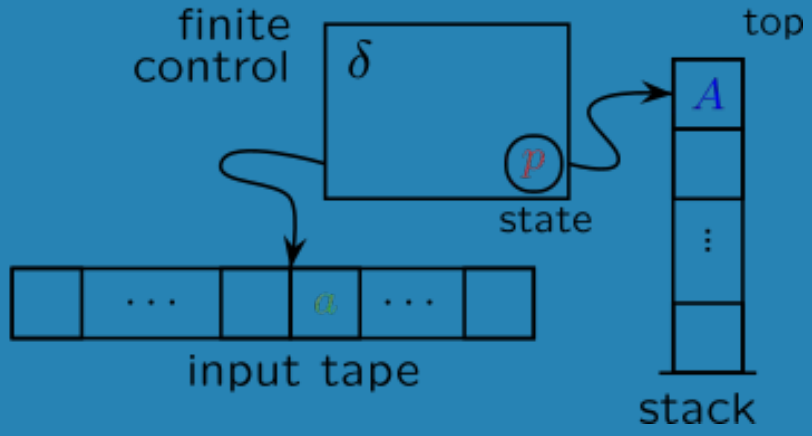

**Field** and **Flow** sensitive Automata

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What is a Synchronized Pushdown System?




**Context** and **Flow** sensitive Automata

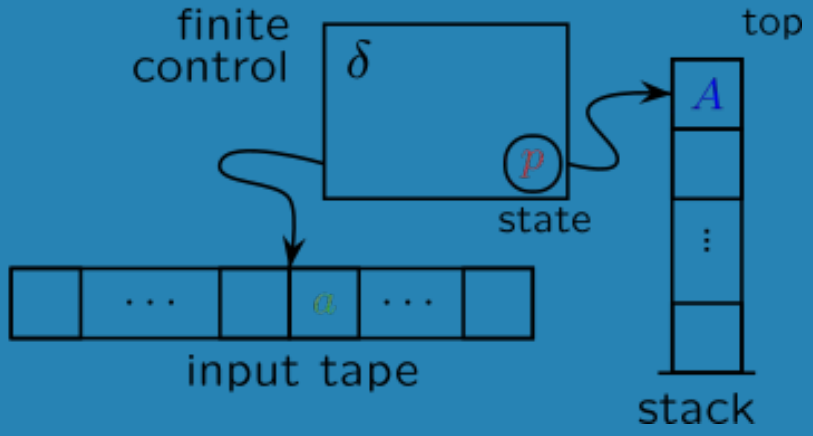



**Field** and **Flow** sensitive Automata

**SYNC POWERS ACTIVATE**

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

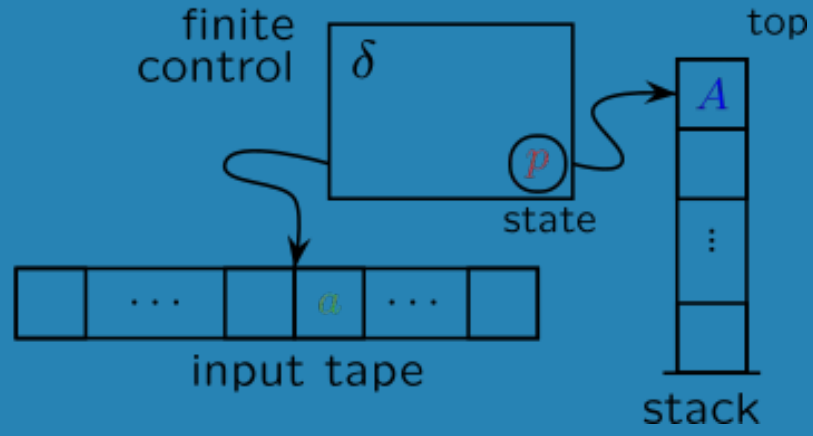
# What is a Synchronized Pushdown System?




**Context** and **Flow** sensitive Automata

Decidable

Context-Sensitive

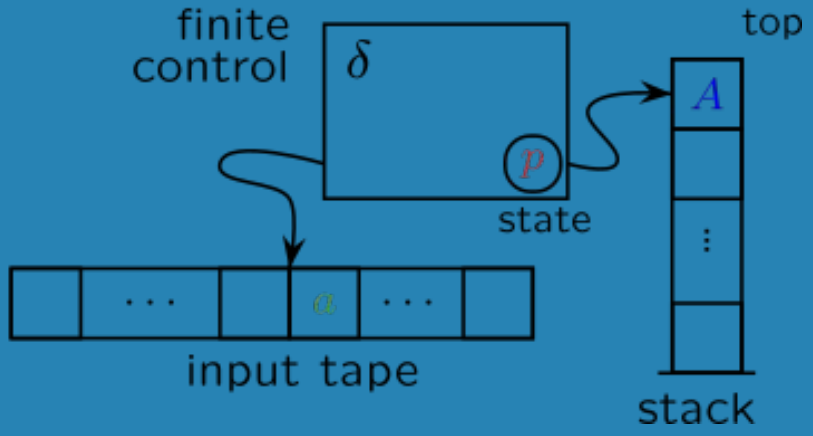



**Field** and **Flow** sensitive Automata

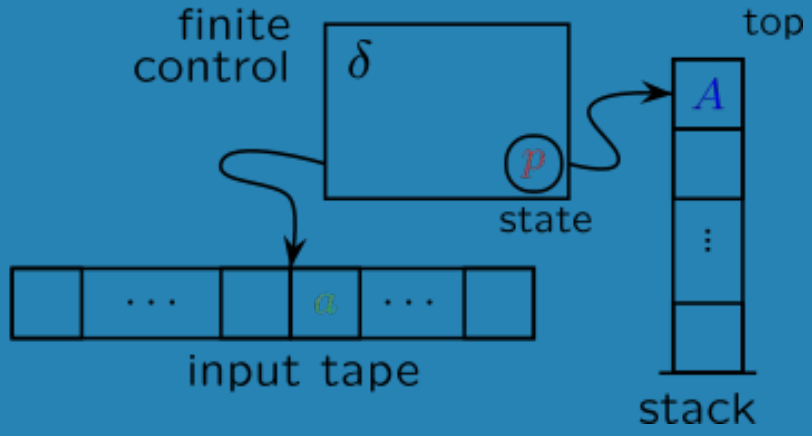
Field-Sensitive

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# How does a PDS work?




**Context** and **Flow** sensitive Automata




**Field** and **Flow** sensitive Automata

prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# Examples!

```
23 main(){
24   A u = new A();
25   A v = u;
26   A w = foo(v);
27 }
```

```
28 foo(A a){
29   if(...){
30     return a;
31   }
32   b = foo(a);
33   return b;
34 }
```

| Normal Rules (main) |
| --- |
| $\langle\!\langle u,24 \rangle\!\rangle \rightarrow \langle\!\langle u,25 \rangle\!\rangle$ |
| $\langle\!\langle u,24 \rangle\!\rangle \rightarrow \langle\!\langle v,25 \rangle\!\rangle$ |
| $\langle\!\langle u,25 \rangle\!\rangle \rightarrow \langle\!\langle u,26 \rangle\!\rangle$ |

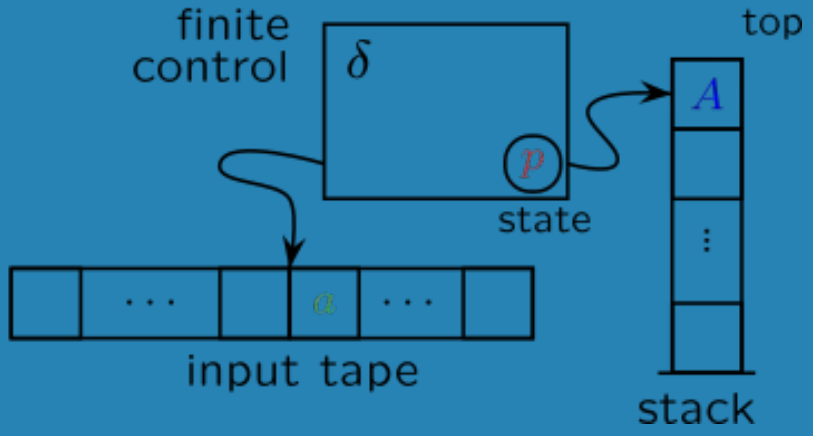
| Normal Rules (foo) |
| --- |
| $\langle\!\langle a,28 \rangle\!\rangle \rightarrow \langle\!\langle a,29 \rangle\!\rangle$ |
| $\langle\!\langle a,29 \rangle\!\rangle \rightarrow \langle\!\langle a,30 \rangle\!\rangle$ |
| $\langle\!\langle a,29 \rangle\!\rangle \rightarrow \langle\!\langle a,31 \rangle\!\rangle$ |
| $\langle\!\langle a,32 \rangle\!\rangle \rightarrow \langle\!\langle a,33 \rangle\!\rangle$ |
| $\langle\!\langle b,32 \rangle\!\rangle \rightarrow \langle\!\langle b,33 \rangle\!\rangle$ |

| Push Rules |
| --- |
| $\langle\!\langle v,25 \rangle\!\rangle \rightarrow \langle\!\langle a,28 \cdot 26 \rangle\!\rangle$ |
| $\langle\!\langle a,31 \rangle\!\rangle \rightarrow \langle\!\langle a,28 \cdot 32 \rangle\!\rangle$ |

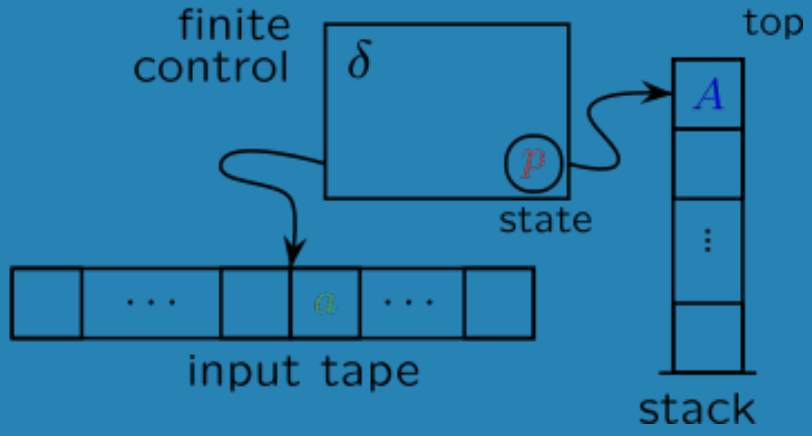
| Pop Rules |
| --- |
| $\langle\!\langle a,30 \rangle\!\rangle \rightarrow \langle\!\langle a,\epsilon \rangle\!\rangle$ |
| $\langle\!\langle a,30 \rangle\!\rangle \rightarrow \langle\!\langle v,\epsilon \rangle\!\rangle$ |
| $\langle\!\langle a,33 \rangle\!\rangle \rightarrow \langle\!\langle a,\epsilon \rangle\!\rangle$ |
| $\langle\!\langle a,33 \rangle\!\rangle \rightarrow \langle\!\langle v,\epsilon \rangle\!\rangle$ |
| $\langle\!\langle b,33 \rangle\!\rangle \rightarrow \langle\!\langle w,\epsilon \rangle\!\rangle$ |
| $\langle\!\langle b,33 \rangle\!\rangle \rightarrow \langle\!\langle b,\epsilon \rangle\!\rangle$ |

Späth, Johannes, Karim Ali, and Eric Bodden. "Context-, flow-, and field-sensitive data-flow analysis using synchronized pushdown systems." Proceedings of the ACM on Programming Languages 3.POPL (2019): 1-29.

# How does a PDS work?




**Context** and **Flow** sensitive Automata



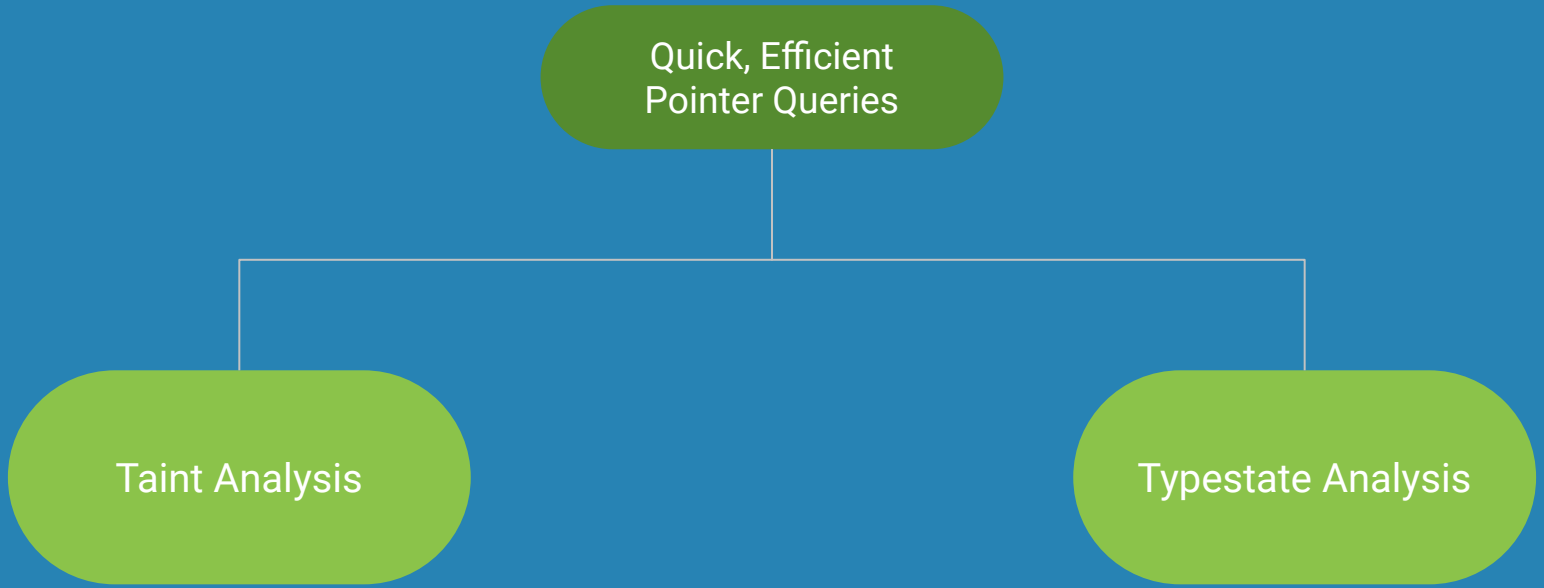

**Field** and **Flow** sensitive Automata

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Why are they used in static analyses like SWAN?

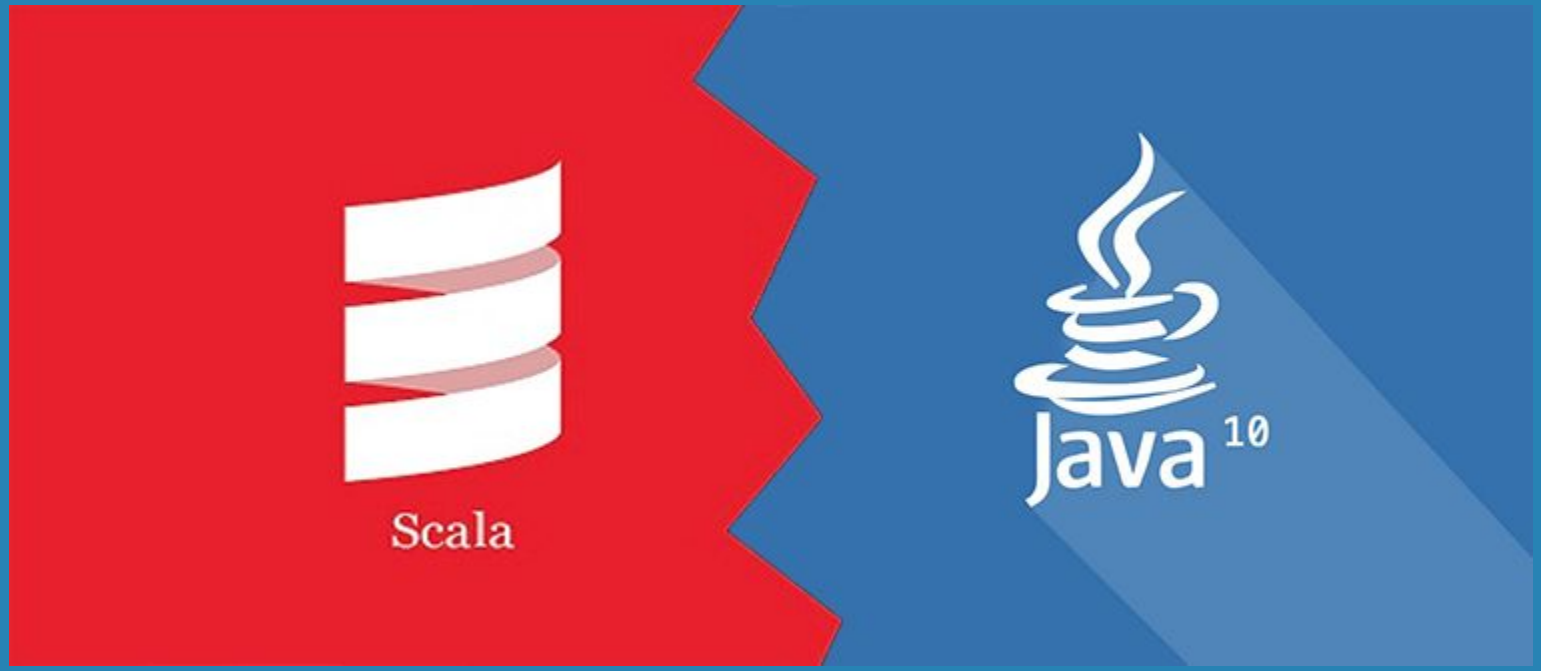
For exactly the reason that they are decidable, context and field sensitive and lets us avoid k-limiting issues in recursive programs.

# SPDS in SWAN: Developer uses




prollius@ualberta.ca
lqs@ualberta.ca
tqakinlo@ualberta.ca

# What benefits are there to using scala over java?




prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

- While Scala has static typing like Java, it integrates type inference, eliminating the need for explicit declaration of variable types and allowing for better code streamlining.

- Scala streamlines concurrent and parallel programming through its actor model and Akka framework, simplifying the development of scalable and efficient programs compared to Java's less effective thread-based model.

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

- Scala seamlessly interoperates with Java, enabling the use of existing Java frameworks and libraries concurrently with Scala programming.

- This encourages a smooth transition from Java to Scala or the concurrent use of both languages within a project.

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

- Additionally, Scala includes advanced language features like pattern matching, high-order functions, and case classes, leading to neater, better-maintained code and aiding in bug resolution.

- We can see this in a small example of a program that determines if a person(class) is an adult or not, written in both Java and Scala

- The Next 3 slides feature the classes used/required of the adult_filter program in Java and Scala

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

Filter.java

```java
import java.util.ArrayList;
import java.util.List;

class Filter {
    static <T> List<T> filter(List<T> list, Predicate<T> predicate) {
        List<T> result = new ArrayList<>();
        for (T item : list) {
            if (predicate.test(item)) {
                result.add(item);
            }
        }
        return result;
    }
}
```

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

Person.java

```java
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Predicate.java

```java
interface Predicate<T> {
    boolean test(T t);
}
```

# What benefits are there to using scala over java?

adult_filter.scala

```scala
case class Person(name: String, age: Int)

def isAdult(person: Person): Boolean = {
  person.age >= 18
}

val people = List(
  Person("Alice", 25),
  Person("Bob", 17),
  Person("Charlie", 30)
)

val adults = people.filter(isAdult)

adults.foreach {
  case Person(name, _) => println(s"$name is an adult.")
}

```

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?

main.scala | adult_filter.scala

main.scala > Main > main > ...

```scala
object Main {
  def main(args: Array[String]): Unit = {
    // Define a case class representing a person
    case class Person(name: String, age: Int)

    // Function to check if a person is an adult
    def isAdult(person: Person): Boolean = {
      person.age >= 18
    }

    // List of people
    val people = List(
      Person("Alice", 25),
      Person("Bob", 17),
      Person("Charlie", 30)
    )

    // Filter adults using a high-order function and pattern matching
    val adults = people.filter(isAdult)

    adults.foreach {
      case Person(name, _) => println(s"$name is an adult.")
    }
  }
}
```

Main.java | Filter.java | Person.java | Predicate.java

Main.java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

class Main {
    // Function to check if a person is an adult
    static boolean isAdult(Person person) {
        return person.age >= 18;
    }
    public static void main(String[] args) {
        // List of people
        List<Person> people = new ArrayList<>();
        people.add(new Person("Alice", 25));
        people.add(new Person("Bob", 17));
        people.add(new Person("Charlie", 30));
        // Filter adults using the Filter class and the isAdult predicate
        List<Person> adults = Filter.filter(people, Main::isAdult);
        // Print the names of adults
        for (Person person : adults) {
            System.out.println(person.name + " is an adult.");
        }
    }
}
```

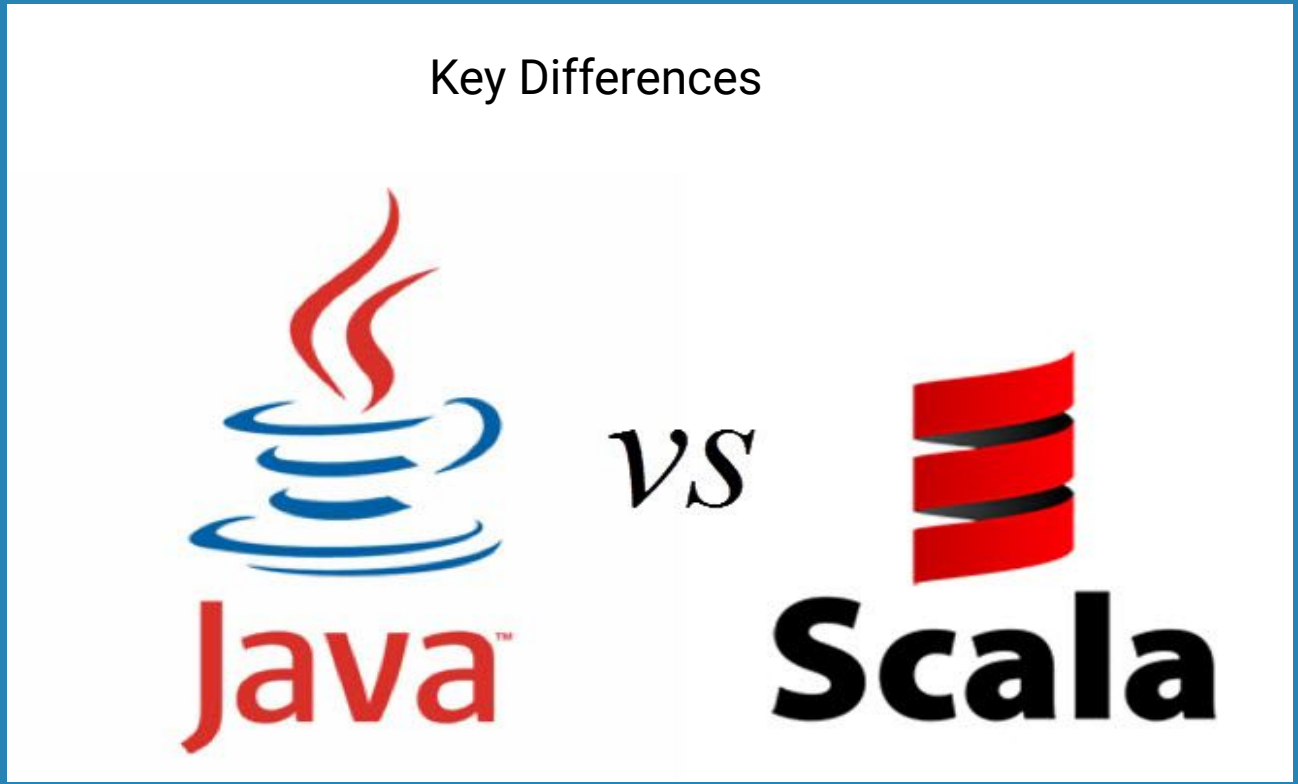
Console | Shell

Run

```
Alice is an adult.
Charlie is an adult.
```

prollius@ualberta.ca
los@ualberta.ca
tgakinlo@ualberta.ca

# What benefits are there to using scala over java?




prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# How are each of these contributing to improving the project

- Overall, Scala is more suitable for complex algorithms and intricate data flow analysis, such as the case of SPDS (assuming it refers to some specific domain or application).

- As a hybrid functional and object-oriented language, Scala supports functional programming paradigms that align with the mathematical and algorithmic nature of SPDS.

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Current Progress

All files within the SPDS project have been fully translated

Now working on ironing out some compatibility issues that arose during translation

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca

# Questions?

prollius@ualberta.ca
lgs@ualberta.ca
tgakinlo@ualberta.ca