Variability Modelling in the ABS Language

Dave Clarke
Katholieke Universiteit Leuven
based on joint work with
José Proença, Michiel Helvenstijen,
Ina Schaefer, Radu Muschevici

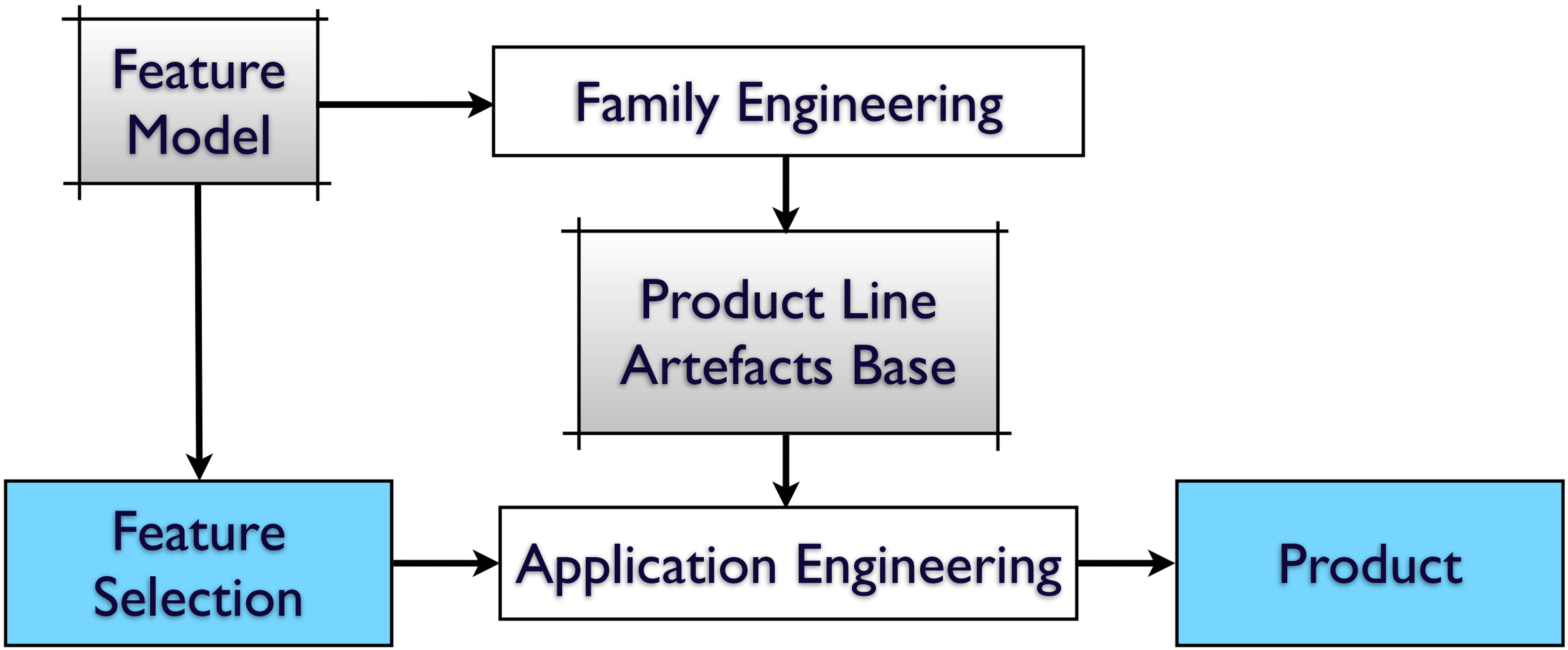
Outline

- Introduction
- Feature Modelling
- Delta Modelling
- Product Line Configuration
- Feature Selection
- Product Generation
- Conclusion

Outline

- Introduction
- Feature Modelling
- Delta Modelling
- Product Line Configuration
- Feature Selection
- Product Generation
- Conclusion

Product Line Development



Running Example

Running Example:
Multi-lingual Hello World

- English: "Hello World"
- German: "Hallo Welt"
- Dutch: "Hallo Wereld"
- Swedish: "Hejsan Allihopa"
- French: "Bonjour tout le monde"
- Possibly with repetition

Ingredients of Variability in ABS

- Core ABS
- Feature Model (μTVL) – describing variability
- Deltas – implementing variability
- Configuration Language
- Feature Selection Language

The Core Functionality

The Core

```
interface Greeting {
     String say_hello();
}

class Greeter implements Greeting {
     String say_hello() {
          return "Hello world";
     }
}
```

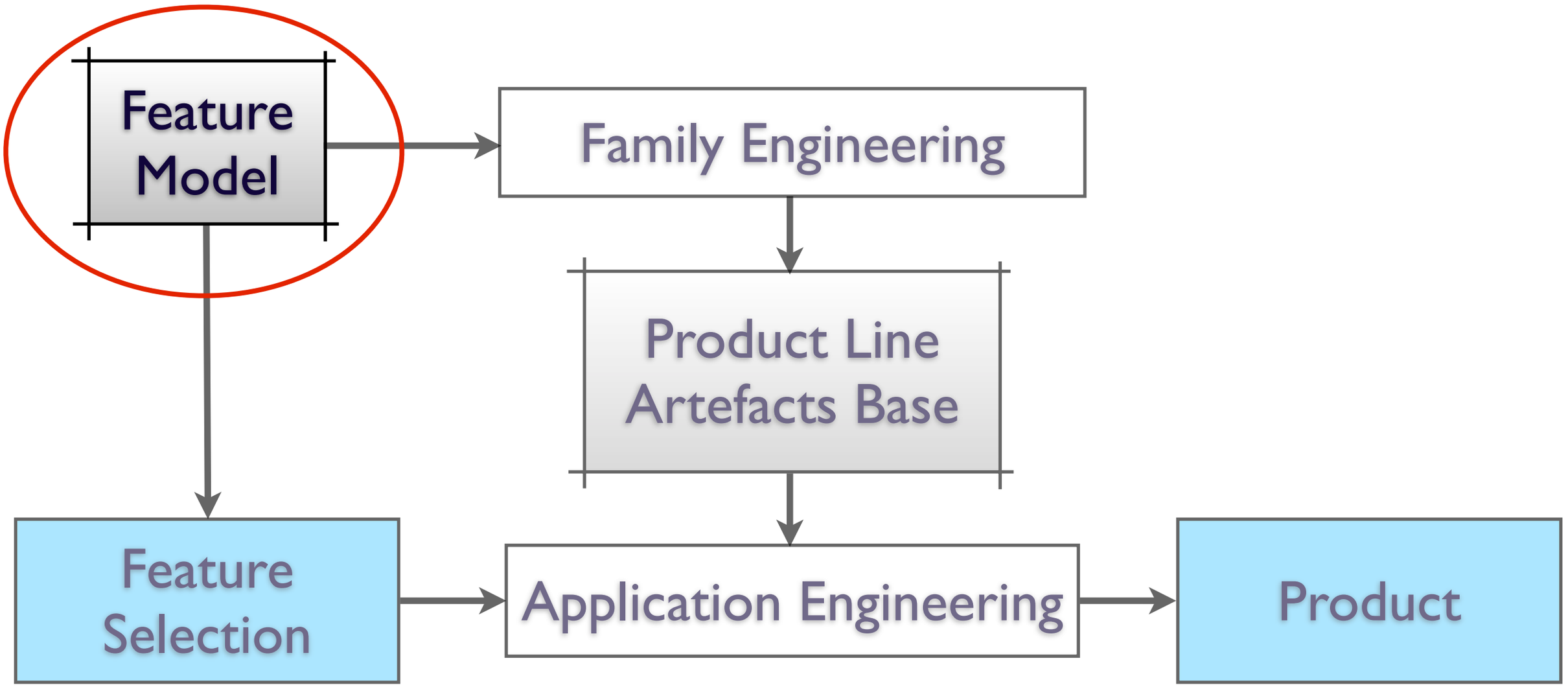
English by default

A fully functioning application
(minus *main*)

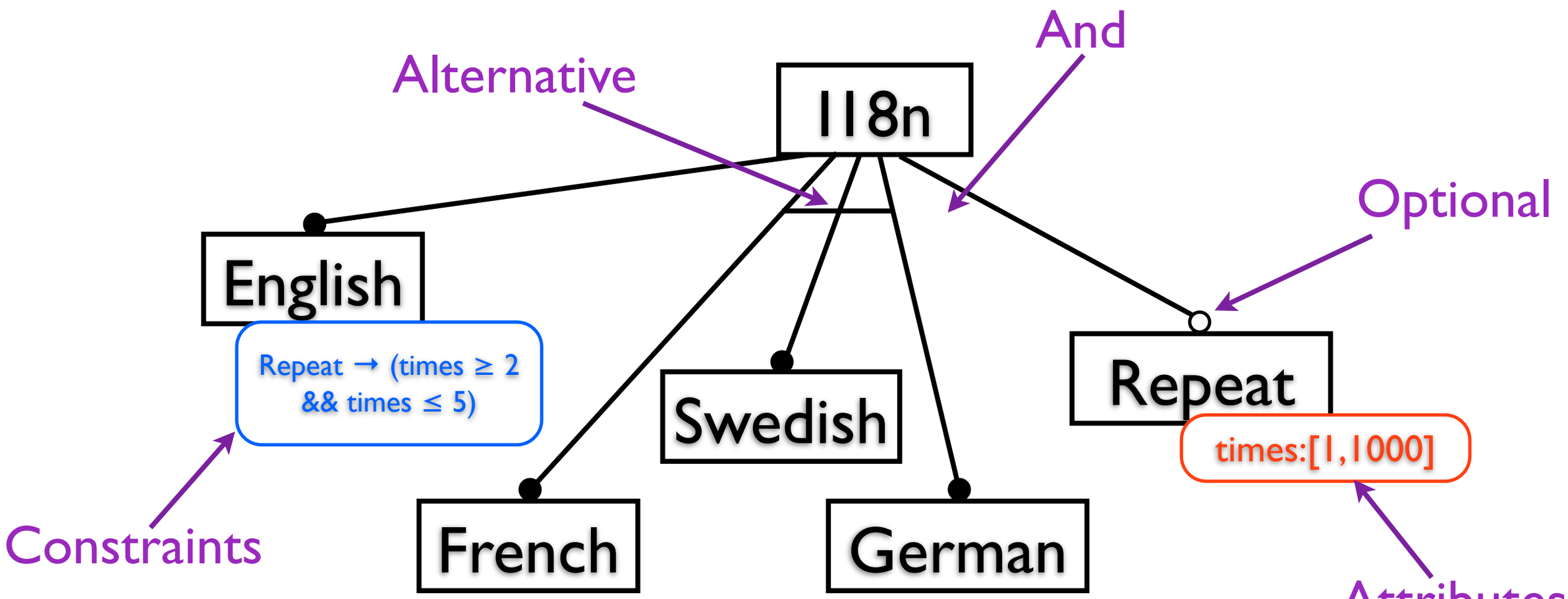
Outline

- Introduction
- **Feature Modelling**
- Delta Modelling
- Product Line Configuration
- Feature Selection
- Product Generation
- Conclusion

Product Line Development



Feature Models



Or Requires Excludes Cardinalities

Feature Model in μTVLs

```
root I18n {
  group allof {
    Language {
      group oneof {
        English, Dutch, French, German, Swedish
      }
    },
    opt Repeat {
      int [0,1000] times;
    }
  }
}

extension English {
  ifin: Repeat ->
    (Repeat.times >= 2 && Repeat.times <= 5);
}
```

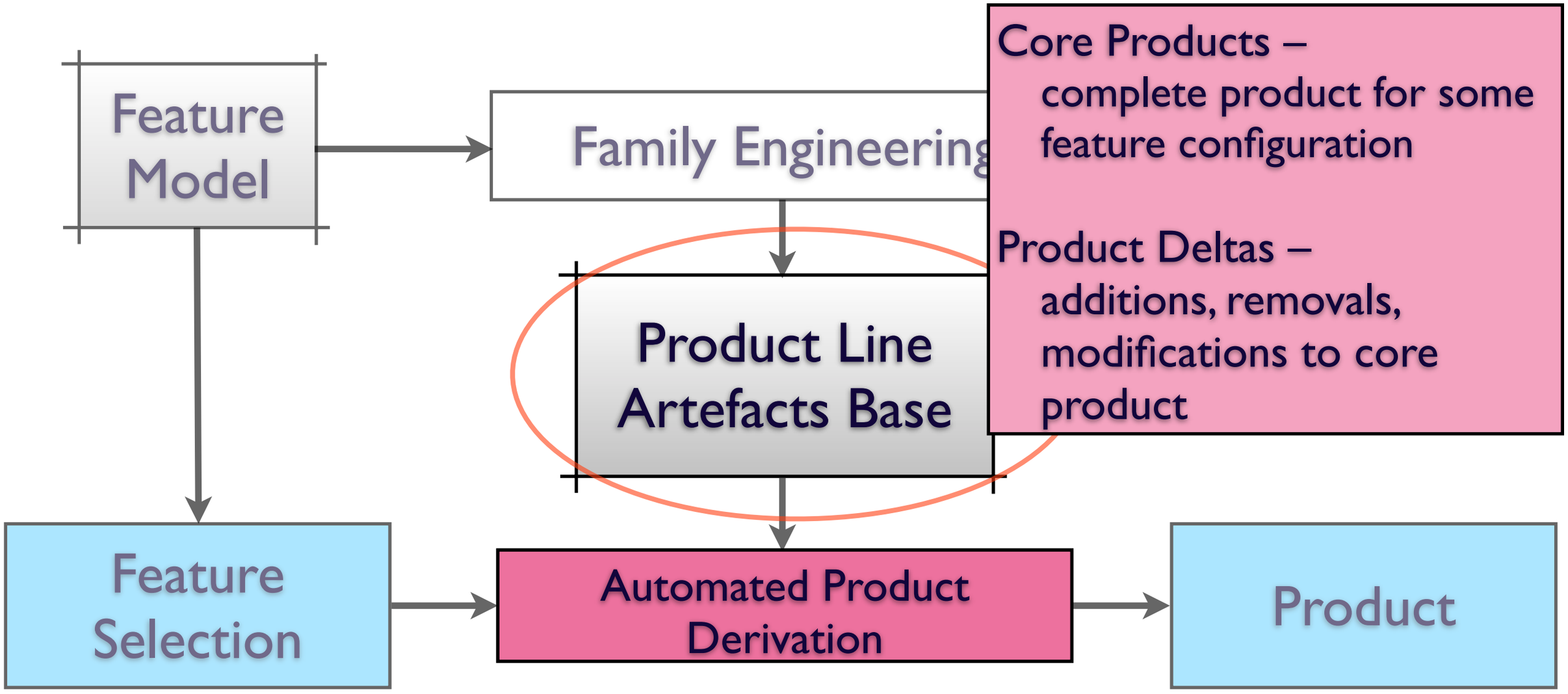
μTVL Feature Modelling Language

- μTVL adapts existing TVL language
  [Claessen, FUNDP, Namur U]
- Constraint solver written in Choco:
  - finds valid feature selections matching specified constraints
  - checks validity of feature selections

Outline

- Introduction
- Feature Modelling
- **Delta Modelling**
- Product Line Configuration
- Feature Selection
- Product Generation
- Conclusion

Delta-oriented Programming



Delta-oriented Programming

- Modifications on Class Level:
  - Addition, Removal and Modification of Classes
- Modifications of Class Structure:
  - Changing Super Class and Constructor
  - Adding/Removing Fields/Methods
  - Modifying Methods (wrapping original call)

[Delta-oriented Programming of Software Product Lines
Schaefer, Bettini, Bono, Damiani, Tanzarella. SPLC 2010.]

The Repeat Delta

```
delta Rpt (Int times) {
    modifies Greeter {
        modifies String say_hello() {
            String result = "";
            Int i = 0;
            while (i < times) {
                result = result + original();
                i = i + 1;
            }
            return result;
        }
    }
}
```

The German Delta

```
delta De {
    modifies Greeter {
        modifies String say_hello() {
            return "Hallo Welt";
        }
    }
}
```

The Dutch Delta

```
delta Nl {
    modifies Greeter {
        modifies String say_hello() {
            return "Hallo wereld";
        }
    }
}
```

The French Delta

```
delta Fr {
    modifies Greeter {
        modifies String say_hello() {
            return "Bonjour tout le monde";
        }
    }
}
```

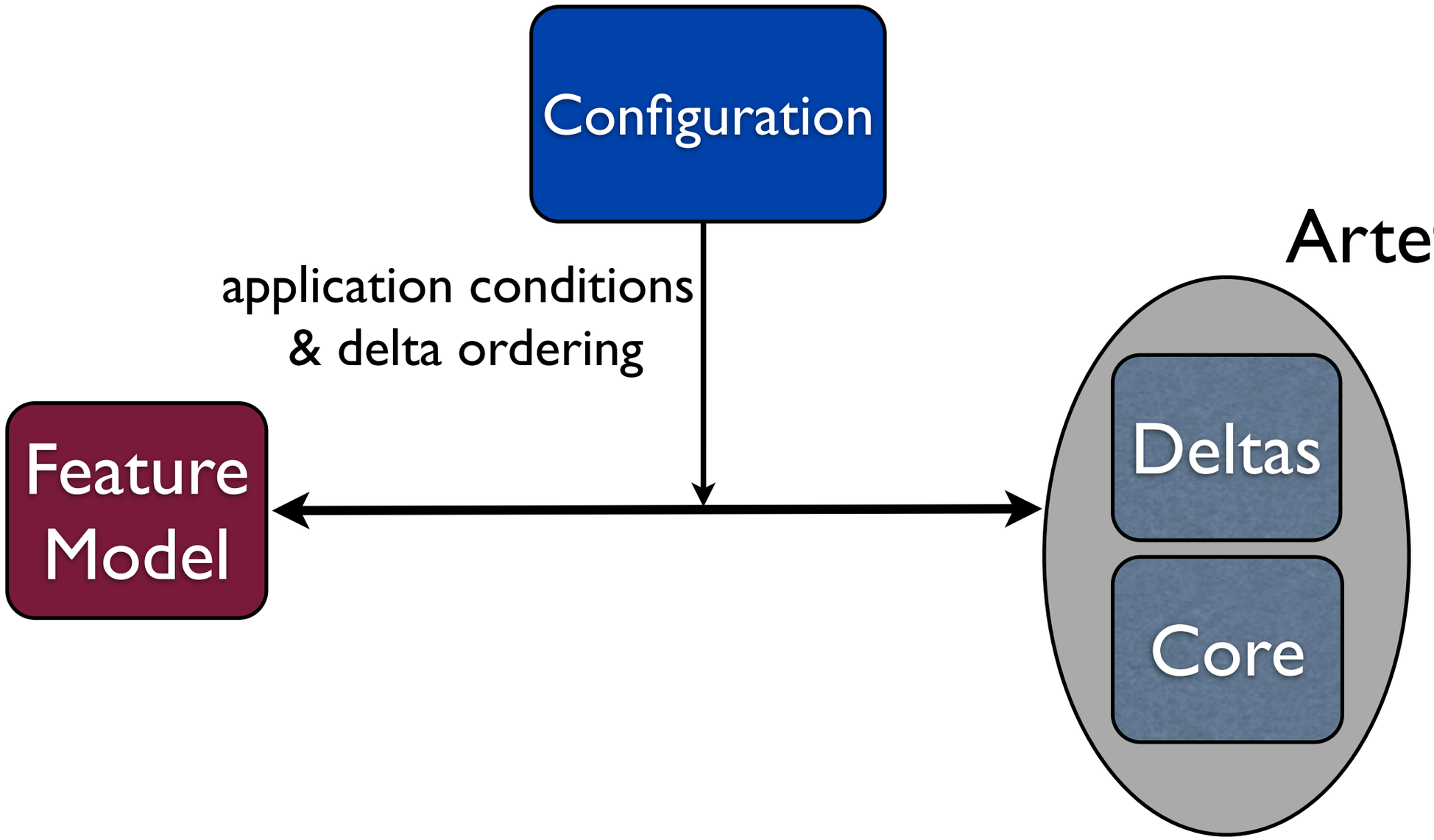
The Swedish Delta

```
delta Sv {
    modifies Greeter {
        modifies String say_hello() {
            return "Hejsan allihopa";
        }
    }
}
```

Outline

- Introduction
- Feature Modelling
- Delta Modelling
- **Product Line Configuration**
  - Feature Selection
  - Product Generation
  - Conclusion

Features-Delta Mapping



Configuration

- Links feature model with deltas
- Adds application conditions to deltas:
  - constraints on features and attributes
- Specifies ordering between deltas
- Nests deltas to ensure atomic application

Configuration

```
product line HelloMultiLingual {
    features Repeat, German, French, Dutch, Swedish;
    core English;

    delta De when German && not Repeat;
    delta Fr when French;
    delta Nl when Dutch;
    delta Sv when Swedish && Repeat;
    delta Rpt(Repeat.times)
        after De, Fr, Nl, Sv when Repeat;
}
```

application conditions

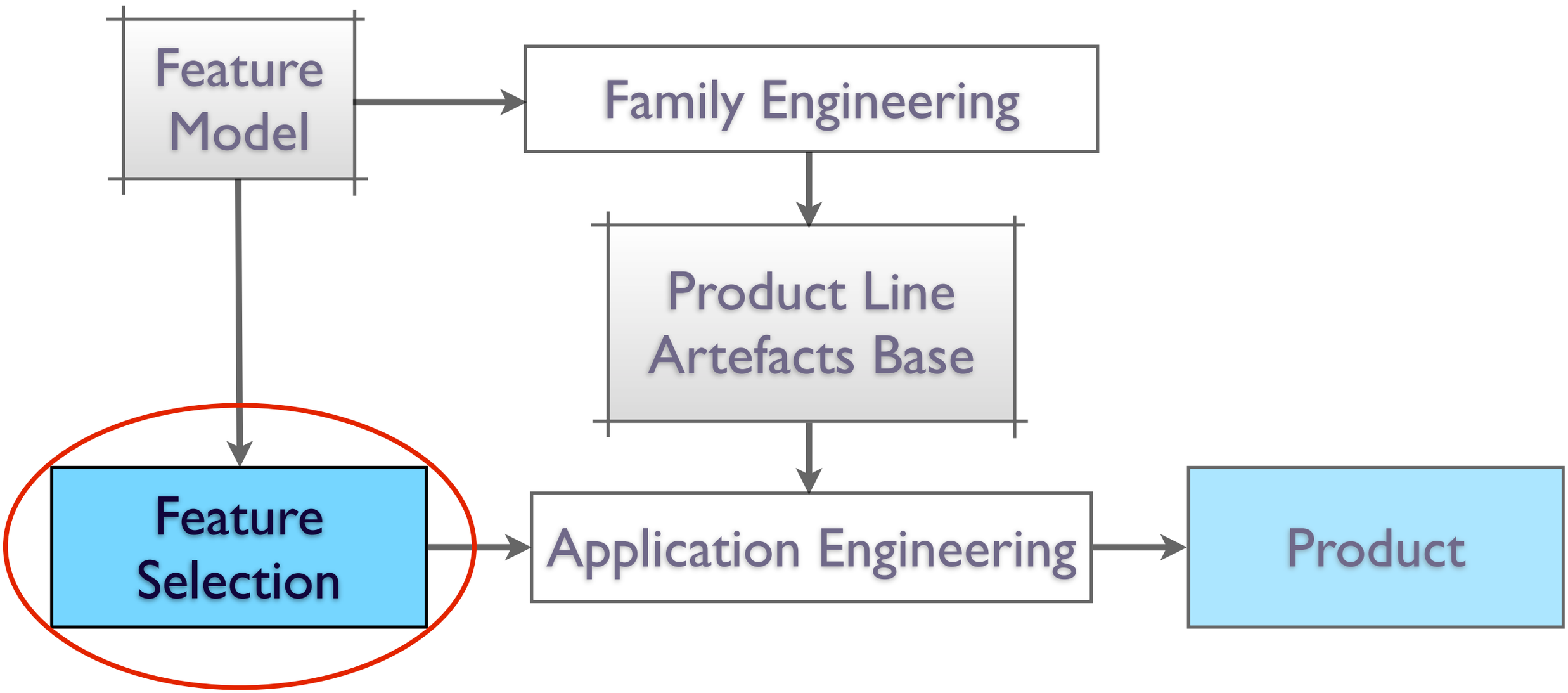
parameter passing

ordering

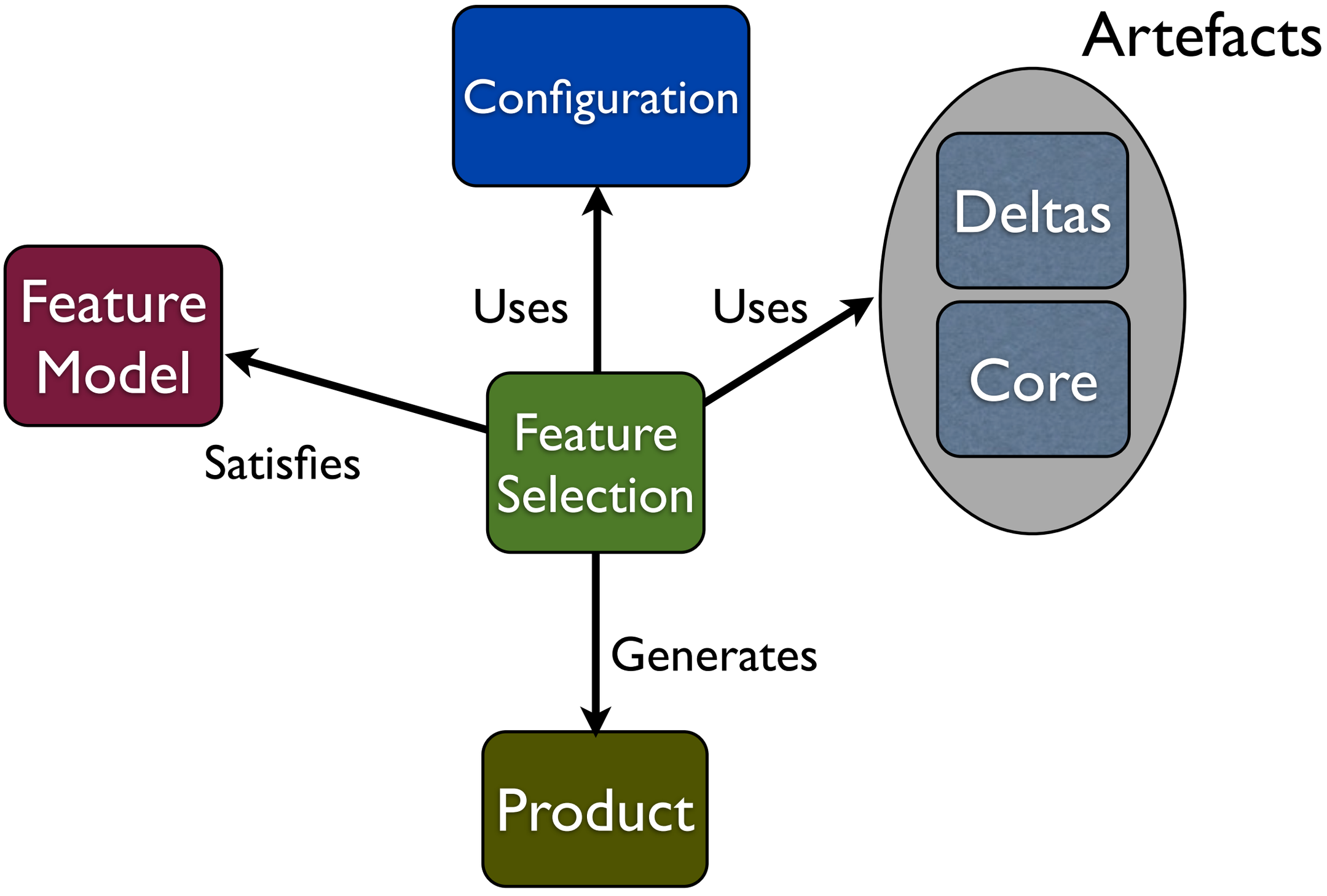
Outline

- Introduction
- Feature Modelling
- Delta Modelling
- Product Line Configuration
- **Feature Selection**
- Product Generation
- Conclusion

Product Line Development



Feature Selection



Feature Selection

- Specifies selected features and their attributes.

- Final Ingredient required to Generate Product.

- Checked against Feature Model.

Feature Selection

```
// basic product with no deltas
product P1 {
     Greeting bob;
     bob = new Greeter();
     String s = "";
     s = bob.say_hello();
}
```

Initialisation
(aka main)

Feature Selection

Feature Selection

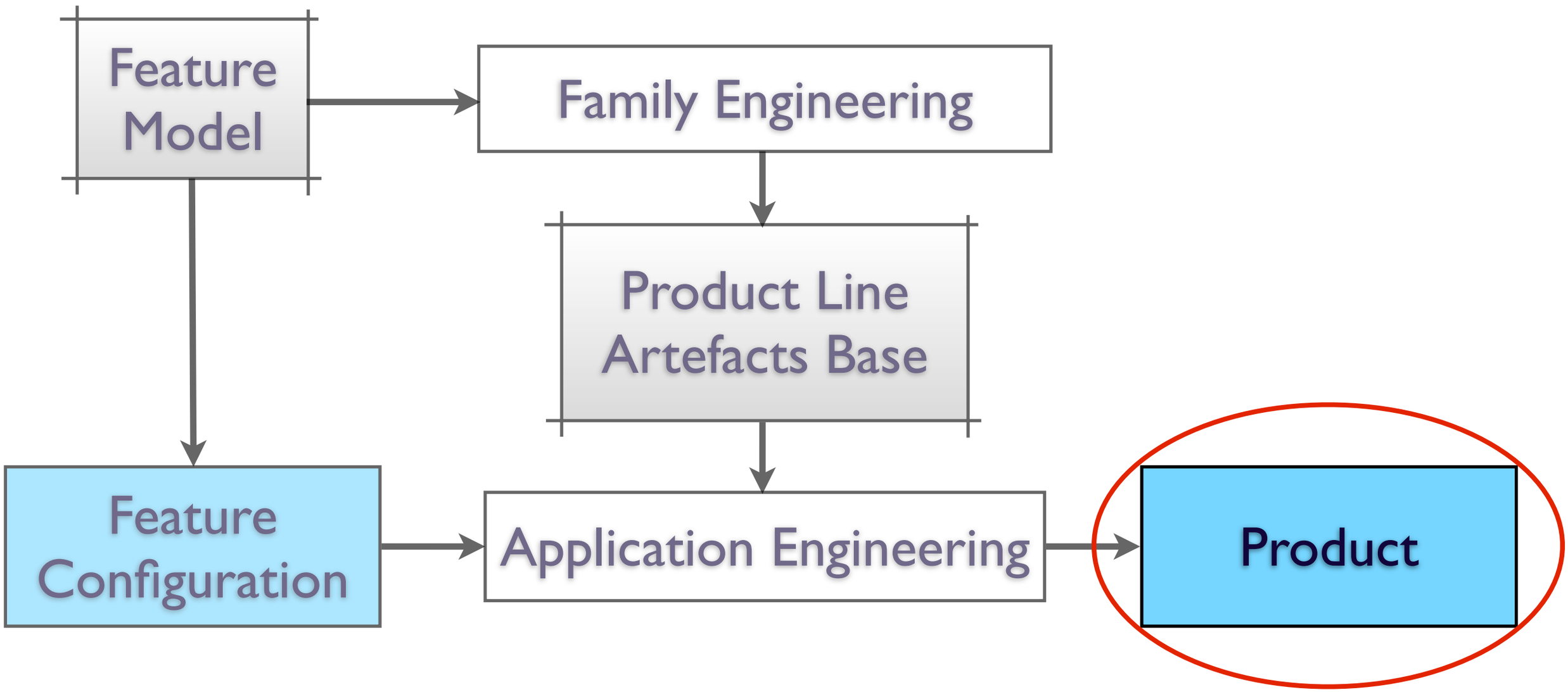
Attribute Specification

```
// apply delta Fr and Repeat
product P3 (French, Repeat{times=10}) {
    Greeting bob;
    bob = new Greeter();
    String s = "";
    s = bob.say_hello();
}
```

Outline

- Introduction
- Feature Modelling
- Delta Modelling
- Product Line Configuration
- Feature Selection
- **Product Generation**
- Conclusion

Product Line Development



Product Generation

- For a Feature Configuration:
  - Select product deltas with valid application condition
  - Determine linear ordering of product deltas compatible with partial ordering
  - Apply changes specified by product deltas to core product in the linear order

Given Feature Selection

```
// apply delta Fr and Repeat
product P3 (French, Repeat{times=10})
{
    Greeting bob;
    bob = new Greeter();
    String s = "";
    s = bob.say_hello();
}
```

Apply Delta Fr

```
class Greeter implements Greeting {
    String say_hello() {
        return "Hello world";
    }
}
```

+

```
delta Fr {
    modifies Greeter {
        modifies String say_hello() {
            return "Bonjour tout le monde";
        }
    }
}
```

Apply Delta Fr

```
class Greeter implements Greeting {
    String say_hello() {
        return "Bonjour tout le monde";
    }
}
```

Configure Repeat


```
delta Rpt (Int times) {
    modifies Greeter {
        modifies String say_hello() {
            String result = "";
            Int i = 0;
            while (i < times) {
                result = result + original();
                i = i + 1;
            }
            return result;
        }
    }
}
```

Configure Repeat

```
delta Rpt                    {
    modifies Greeter {
        modifies String say_hello() {
            String result = "";
            Int i = 0;
            while (i < 10    ) {
                result = result + original();
                i = i + 1;
            }
            return result;
        }
    }
}
```

Apply Repeat

```
class Greeter implements Greeting {
    String say_hello() {
        return "Bonjour tout le monde";
    }
}
```

+

```
delta Rpt {
    modifies Greeter {
        modifies String say_hello() {
            String result = "";
            Int i = 0;
            while (i < 10) {
                result = result + original();
                i = i + 1;
            }
            return result;
        }
    }
}
```

Apply Repeat

```
class Greeter implements Greeting {
    String __say_hello_original() {
        return "Bonjour tout le monde";
    }

    String say_hello() {
        String result = "";
        Int i = 0;
        while (i < 10) {
            result = result + __say_hello_original();
             i = i + 1;
        }
        return result;
    }
}
```

Adding Initialisation

```
class Greeter implements Greeting {
    String __say_hello_original() {
        return "Bonjour tout le monde";
    }

    String say_hello() {
        String result = "";
        Int i = 0;
        while (i < 10) {
            result = result + __say_hello_original();
             i = i + 1;
        }
        return result;
    }
}
{
    Greeting bob;
    bob = new Greeter();
    String s = "";
    s = bob.say_hello();
}
```

Typing Issues

- Errors
  - Field, method, class missing
  - Incorrect type: field/method
- Warnings
  - Field/method/class not expected one
  - Conflicting implementation orderings
- Product line level checking instead of per product (= after generation) checking

Outline

- Introduction
- Feature Modelling
- Delta Modelling
- Product Line Configuration
- Feature Selection
- Product Generation
- **Conclusion**

HATS ABS

- HATS ABS is a collection of 5 languages
  - Core Product
  - Product Deltas
  - Feature Modelling
  - Configuration: Links FM and Core+Deltas
  - Feature Selection
- Implementation efforts are well underway

Questions?