

# Directed Synthesis of Failing Concurrent Executions

**Malavika Samak\***, Omer Tripp<sup>+</sup>, Murali Krishna Ramanathan\*

\* Indian Institute of Science

+ Google Inc, Mountain View



# Multi-threaded libraries are useful ...

# Multi-threaded libraries are useful ...

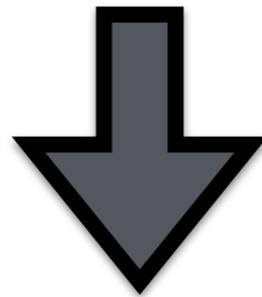
- Provides safety guarantees under concurrency

# Multi-threaded libraries are useful ...

- Provides safety guarantees under concurrency
- Ensures performance benefits

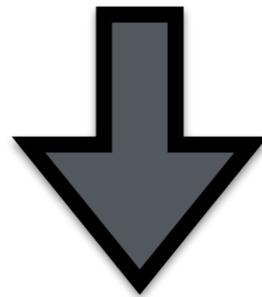
# Multi-threaded libraries are useful ...

- Provides safety guarantees under concurrency
- Ensures performance benefits



# Multi-threaded libraries are useful ...

- Provides safety guarantees under concurrency
- Ensures performance benefits



Makes building multithreaded applications easier

# Building multi-threaded libraries



# Building multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examname(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("P#")) {
            Exam_Name = "PB";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Building multi-threaded libraries

## Class under construction

```
package examz;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examname(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("P#")) {
            Exam_Name = "PB";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Building multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examname(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("P#")) {
            Exam_Name = "PB";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...

Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...

Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("TCB")) {
            Exam_Name = "PCB .NET";
        } else if (examCode.equals("PBB")) {
            Exam_Name = "PB";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...



Need to handle various issues associated with multithreading

# Building multi-threaded libraries

## Class under construction

```
package exam;
public class StudentResults {
    private String Full_Name;
    private String Exam_Score;
    private String Exam_Code;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "John Doe";
        Exam_Score = "85%"; // 85
        Exam_Grade = "B+";
    }

    String FullName() {
        return Full_Name;
    }

    String ExamName(String Exam_Code) {
        if (ExamCode.equals("VB13")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("CSE13")) {
            Exam_Name = "PCB .NET";
        } else if (ExamCode.equals("PWB")) {
            Exam_Name = "PWB";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality** X
- **Performance** ✓
- **Concurrency** X
- **Modularity** X
- **Data Structures** ✓
- **Shared state** ✓
- **Lock field correlation** ✓
- ... ✓

# Building multi-threaded libraries

## Class under construction

```
package examo;
public class StudentResults {
    private String Full_Name;
    private String Exam_Score;
    private String Exam_Code;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "John Doe";
        Exam_Score = "85";
        Exam_Grade = "B+";
    }

    String FullName() {
        return Full_Name;
    }

    String ExamName(String Exam_Code) {
        if (ExamCode.equals("VB13")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("P#P")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality** X
- **Performance** ✓
- **Concurrency** X
- **Modularity** X
- **Data Structures** ✓
- **Shared state** ✓
- **Lock field correlation** ✓
- ... ✓

Simultaneously handling the issues  
can introduce bugs

# Building multi-threaded libraries

## Class under construction

```
package examz;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#P")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "#VB";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...

# Building multi-threaded libraries

## Class under construction

```
package examz;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("J2V")) {
            Exam_Name = "Java";
        } else if (examCode.equals("TCB")) {
            Exam_Name = "PCB .NET";
        } else if (examCode.equals("PVB")) {
            Exam_Name = "PVB";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...

Perform modular testing of various properties

# Building multi-threaded libraries

## Class under construction

```
package examz;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("VB#")) {
            Exam_Name = "VB.NET";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



- **Functionality**
- **Performance**
- **Concurrency**
- **Modularity**
- **Data Structures**
- **Shared state**
- **Lock field correlation**
- ...

Perform modular testing of various properties

# Testing multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examname(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PHP")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Testing multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullNames(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PHP")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Testing multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fulllname(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examname(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PB")) {
            Exam_Name = "PB";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Testing multi-threaded libraries

## Class under construction

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullNames(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = ".NET";
        } else if (examCode.equals("FW")) {
            Exam_Name = "FW";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

```
class ExecTest {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        // generate an index out-of-bounds exception
        num[7] = 10;
        System.out.println("this won't be displayed");
    }
}

class StackTraceHabille {
    static void myMethod() {
        try {
            ExecTest.genException();
        } catch(ArrayIndexOutOfBoundsException exc) {
            exc.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            StackTraceHabille.myMethod();
            ExecTest.genException();
        } catch(ArrayIndexOutOfBoundsException exc) {
            // catch the exception
            System.out.println("standard message is ");
            System.out.println(exc);
            System.out.println("in stack traces ");
            exc.printStackTrace();
        }
        System.out.println("After catch statement.");
    }
}
```



Client

# Testing multi-threaded libraries

## Class under construction

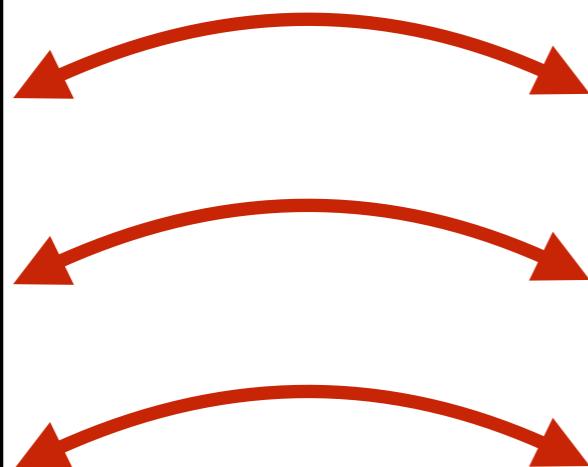
```
package exam2;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = ".NET";
        } else if (examCode.equals("FW")) {
            Exam_Name = "FW";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



```
class ExamTest {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        // generate an index out-of-bounds exception
        num[7] = 10;
        System.out.println("this won't be displayed");
    }

    class StackTraceHabille {
        static void myMethod() {
            try {
                ExamTest.genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }

        public static void main(String[] args) {
            try {
                StackTraceHabille.myMethod();
                ExamTest.genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                // catch the exception
                System.out.println("standard message is ");
                System.out.println(exc);
                System.out.println("in stack traces ");
                exc.printStackTrace();
            }
            System.out.println("After catch statement.");
        }
    }
}
```



Client

# Testing multi-threaded libraries

- Invokes the APIs provided by the class.

Class u  
constructor

```
package exam2;

public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String fullNames(String name) {
        Full_Name = address;
        return Full_Name;
    }

    String examName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("PHP")) {
            Exam_Name = "PHP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

```
class StackTraceable {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        // generate an index out-of-bounds exception
        num[7] = 10;
        System.out.println("this won't be displayed");
    }

    class StackTraceBubble {
        static void myMethod() {
            try {
                StackTraceable.genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }

        public static void main(String[] args) {
            try {
                StackTraceBubble.myMethod();
                StackTraceable.genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                // catch the exception
                System.out.println("standard message is ");
                System.out.println(exc);
                System.out.println("in stack traces ");
                exc.printStackTrace();
            }
            System.out.println("After catch statement.");
        }
    }
}
```



Client

# Testing multi-threaded libraries

## Class under construction

- Invokes the APIs provided by the class.
- Exposes the broken contracts in the library.

```
package exam2;
public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("TCB")) {
            Exam_Name = "PCB .NET";
        } else if (examCode.equals("PBP")) {
            Exam_Name = "PBP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

```
class StackTraceH
{
    static void genException()
    {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        // generate an index out-of-bounds exception
        num[7] = 10;
        System.out.println("this won't be displayed");
    }

    class StackTraceHable
    {
        static void myMethod()
        {
            try
            {
                StackTraceH.genException();
            }
            catch(ArrayIndexOutOfBoundsException exc)
            {
                exc.printStackTrace();
            }
        }

        public static void main(String[] args)
        {
            try
            {
                StackTraceHable.myMethod();
                StackTraceH.genException();
            }
            catch(ArrayIndexOutOfBoundsException exc)
            {
                // catch the exception
                System.out.println("standard message is ");
                System.out.println(exc);
                System.out.println("init stack trace ");
                exc.printStackTrace();
                System.out.println("After catch statement.");
            }
        }
    }
}
```



Client

# Testing multi-threaded libraries

## Class under construction

- Invokes the APIs provided by the class.
- Exposes the broken contracts in the library.
- Facilitates regression testing.

```
package exam2;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String full_name(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String exam_name(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("JV")) {
            Exam_Name = "Java";
        } else if (examCode.equals("TCB")) {
            Exam_Name = "VB.NET";
        } else if (examCode.equals("PBP")) {
            Exam_Name = "PB";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

```
class Emitter {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        // generate an index out-of-bounds exception
        num[7] = 10;
        System.out.println("this won't be displayed");
    }
}

class StackTraceHoller {
    static void myMethod() {
        try {
            Emitter.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            exc.printStackTrace();
        }
    }

    public static void main(String[] args) {
        try {
            StackTraceHoller.myMethod();
            Emitter.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            // catch the exception
            System.out.println("standard message is ");
            System.out.println(exc);
            System.out.println("initial trace ");
            exc.printStackTrace();
            System.out.println("After catch statement.");
        }
    }
}
```



Client

# Testing multi-threaded libraries

```
1 package StackTest;
2
3     static void genException()
4     {
5         int num[] = new int[4];
6         System.out.println("Before exception is generated.");
7         // generate an index out-of-bounds exception
8         num[7] = 10;
9         System.out.println("This won't be displayed.");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             StackTest.genException();
19         }
20         catch (ArrayIndexOutOfBoundsException exc)
21         {
22             exc.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             StackTest.genException();
32         }
33         catch (ArrayIndexOutOfBoundsException exc)
34         {
35             // catch the exception
36             System.out.println("Standard message is ");
37             System.out.println(exc);
38             System.out.println("Stack trace ");
39             exc.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```



**Client**

**Developer**

# Testing multi-threaded libraries

```
1 class EtcTest
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             EtcTest.penException();
19         } catch (ArrayIndexOutOfBoundsException exc)
20         {
21             exc.printStackTrace();
22         }
23     }
24
25     public static void main(String[] args)
26     {
27         try
28         {
29             StackTraceBubble.myMethod();
30             EtcTest.penException();
31         } catch (ArrayIndexOutOfBoundsException exc)
32         {
33             // catch the exception
34             System.out.println("standard message is ");
35             System.out.println(exc);
36             System.out.println("stack trace ");
37             exc.printStackTrace();
38         }
39         System.out.println("After catch statement.");
40     }
41 }
```



**Client**

**Developer**

# Testing multi-threaded libraries

```
1 class EtcTest
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             EtcTest.penException();
19         } catch (ArrayIndexOutOfBoundsException exc)
20         {
21             exc.printStackTrace();
22         }
23     }
24
25     public static void main(String[] args)
26     {
27         try
28         {
29             StackTraceBubble.myMethod();
30             EtcTest.penException();
31         } catch (ArrayIndexOutOfBoundsException exc)
32         {
33             // catch the exception
34             System.out.println("standard message is ");
35             System.out.println(exc);
36             System.out.println("stack trace ");
37             exc.printStackTrace();
38         }
39         System.out.println("After catch statement.");
40     }
41 }
```



**Client**

**Developer**

# Testing multi-threaded libraries



```
1 class ExecTest
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             ExecTest.penException();
19         }
20         catch(ArrayIndexOutOfBoundsException exc)
21         {
22             exc.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             ExecTest.penException();
32         }
33         catch(ArrayIndexOutOfBoundsException exc)
34         {
35             // catch the exception
36             System.out.println("standard message is ");
37             System.out.println(exc);
38             System.out.println("stack trace ");
39             exc.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```

**Client**

**Developer**

**Challenges**

# Testing multi-threaded libraries

```
1 class EtcTest
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             EtcTest.penException();
19         }
20         catch(ArrayIndexOutOfBoundsException exc)
21         {
22             exc.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             EtcTest.penException();
32         }
33         catch(ArrayIndexOutOfBoundsException exc)
34         {
35             // catch the exception
36             System.out.println("standard message is ");
37             System.out.println(exc);
38             System.out.println("and stack trace ");
39             exc.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```



Client

Developer

How many **threads** need to be **created**?

Challenges

# Testing multi-threaded libraries

```
1 class ExecTest
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             ExecTest.penException();
19         }
20         catch(ArrayIndexOutOfBoundsException exc)
21         {
22             exc.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             ExecTest.penException();
32         }
33         catch(ArrayIndexOutOfBoundsException exc)
34         {
35             // catch the exception
36             System.out.println("standard message is ");
37             System.out.println(exc);
38             System.out.println("and stack trace ");
39             exc.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```



**Client**

**Developer**

How many **threads** need to be **created**?

What **methods** need to be **invoked**?

**Challenges**

# Testing multi-threaded libraries

```
1 class StackTraceBubble
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             StackTraceBubble.penException();
19         }
20         catch(ArrayIndexOutOfBoundsException e)
21         {
22             e.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             StackTraceBubble.penException();
32         }
33         catch(ArrayIndexOutOfBoundsException e)
34         {
35             // catch the exception
36             System.out.println("standard message is ");
37             System.out.println(e);
38             System.out.println("and stack trace ");
39             e.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```



**Client**

**Developer**

How many **threads** need to be **created**?

What **methods** need to be **invoked**?

What are the **parameters** to these methods?

**Challenges**

# Testing multi-threaded libraries

```
1 class StackTrace
2 {
3     static void penException()
4     {
5         int sum[] = new int[4];
6         System.out.println("before exception is generated.");
7         // generate an index out-of-bounds exception
8         sum[7] = 10;
9         System.out.println("this won't be displayed");
10    }
11
12 class StackTraceBubble
13 {
14     static void myMethod()
15     {
16         try
17         {
18             StackTrace.penException();
19         }
20         catch(ArrayIndexOutOfBoundsException exc)
21         {
22             exc.printStackTrace();
23         }
24     }
25
26     public static void main(String[] args)
27     {
28         try
29         {
30             StackTraceBubble.myMethod();
31             StackTrace.penException();
32         }
33         catch(ArrayIndexOutOfBoundsException exc)
34         {
35             // catch the exception
36             System.out.println("standard message is ");
37             System.out.println(exc);
38             System.out.println("and stack trace ");
39             exc.printStackTrace();
40         }
41         System.out.println("After catch statement.");
42     }
43 }
```



**Client**

**Developer**

**Challenges**

How many **threads** need to be **created**?

What **methods** need to be **invoked**?

What are the **parameters** to these methods?

What is the **schedule** that needs to be followed?

# Client synthesis for multi-threaded libraries

# Client synthesis for multi-threaded libraries

```
package student;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("P#")) {
            Exam_Name = "P#";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

**Class under  
test**

# Client synthesis for multi-threaded libraries

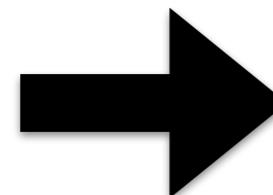
```
package student;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PHP")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



**Class under  
test**

# Client synthesis for multi-threaded libraries

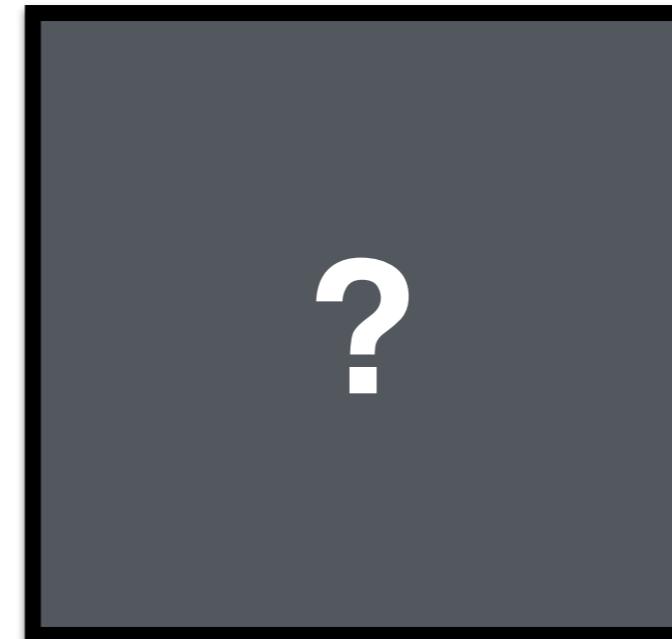
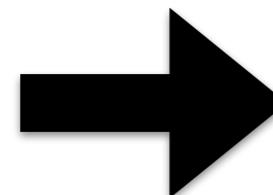
```
package student;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PHP")) {
            Exam_Name = "PERL";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



**Class under  
test**

# Client synthesis for multi-threaded libraries

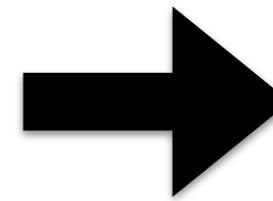
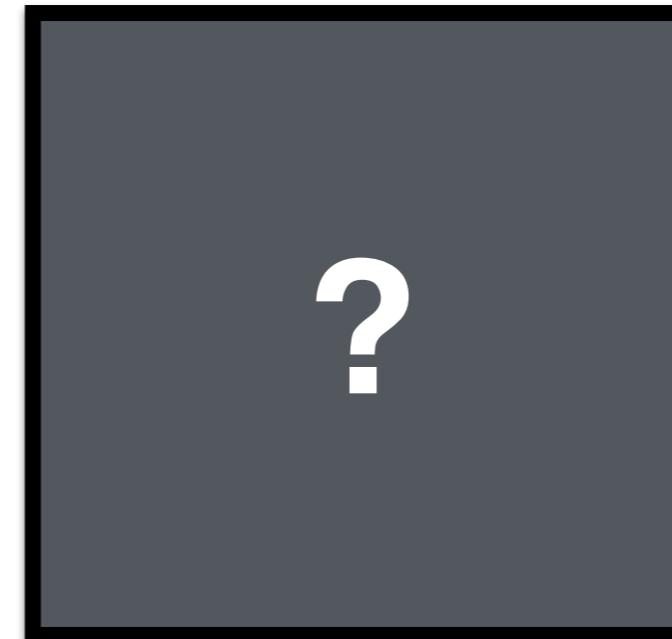
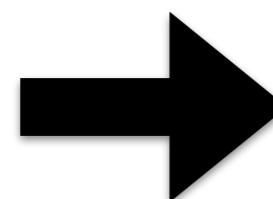
```
package student;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        }
        else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        }
        else if (examCode.equals("C++")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PHP")) {
            Exam_Name = "PEP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



**Class under  
test**

# Client synthesis for multi-threaded libraries

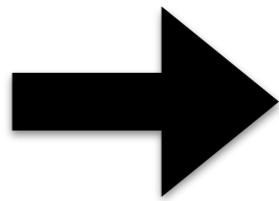
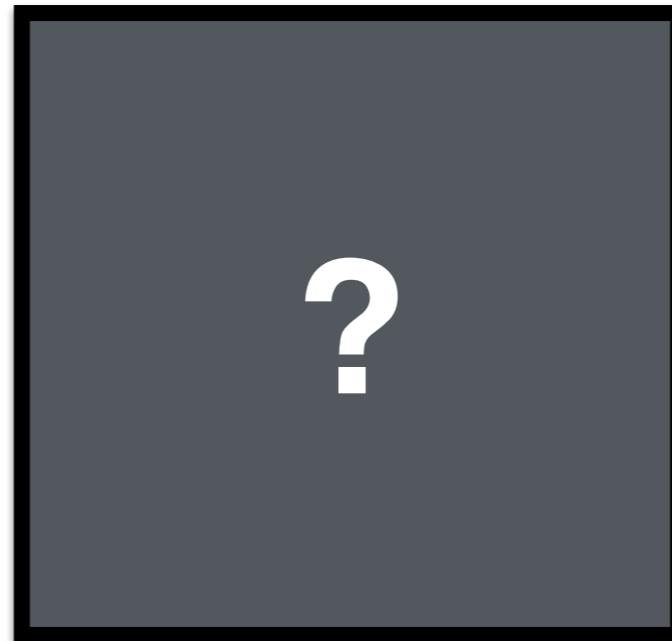
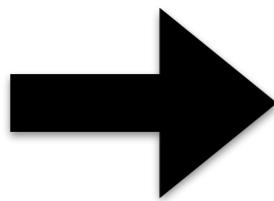
```
package student;

public class StudentResults {
    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("PHP")) {
            Exam_Name = "PEP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



```
class StackTrace {
    static void genException() {
        int sum[] = new int[4];
        System.out.println("before exception is generated.");
        sum[7] = 20;
        System.out.println("this won't be displayed");
    }
}

class StackTraceBubble {
    static void myMethod() {
        try {
            StackTrace.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            exc.printStackTrace();
        }
    }
}

public static void main(String[] args) {
    try {
        StackTraceBubble.myMethod();
    } catch (Exception exc) {
        // catch the exception
        System.out.println("standard message is:");
        System.out.println(exc);
        System.out.println("stack trace is:");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
}
```

**Class under  
test**

**Client**

# Story of the black box so far...

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

Samak and Ramanathan,  
OOPSLA'14

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

Samak and Ramanathan,  
OOPSLA'14

Samak, Ramanathan, Jagannathan  
PLDI'15

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

Samak and Ramanathan,  
OOPSLA'14

Samak, Ramanathan, Jagannathan  
PLDI'15

Samak and Ramanathan,  
FSE'15

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

Samak and Ramanathan,  
OOPSLA'14



Samak, Ramanathan, Jagannathan  
PLDI'15



Samak and Ramanathan,  
FSE'15



**Dynamic  
Analysis  
Engine**

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted generation

Samak and Ramanathan,  
OOPSLA'14



Deadlocks

Samak, Ramanathan, Jagannathan  
PLDI'15



Data races

Samak and Ramanathan,  
FSE'15



Atomicity  
violations

**Dynamic  
Analysis  
Engine**

# Story of the black box so far...

## Randomized generation

Pradel and Gross, PLDI'12



Crashes and Deadlocks

## Targeted ge

Need a **targeted** approach to reveal crashes

Samak and Ramanathan,  
OOPSLA'14



Deadlocks

Samak, Ramanathan, Jagannathan  
PLDI'15



**Dynamic Analysis Engine**



Data races

Samak and Ramanathan,  
FSE'15



Atomicity violations

and now...

# and now...

- ◆ Detecting crashes in a library requires:

# and now...

- ◆ Detecting crashes in a library requires:
  - ◆ Well designed multi-threaded clients

# and now...

- ◆ Detecting crashes in a library requires:
  - ◆ Well designed multi-threaded clients
  - ◆ Specific thread interleavings

# and now...

- ◆ Detecting crashes in a library requires:
  - ◆ Well designed multi-threaded clients
  - ◆ Specific thread interleavings
- ◆ We propose MINION





# Class under test

```
package exam02;

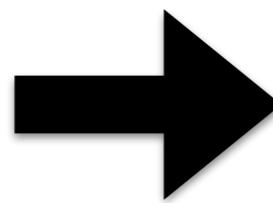
public class StudentResults {

    private String Full_Name;
    private String Exam_Name;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (examCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (examCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (examCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (examCode.equals("PHP")) {
            Exam_Name = "PHP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Class under test

```
package com.droid;

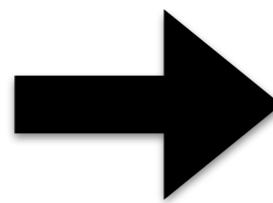
public class StudentResults {

    private String Full_Name;
    private String Exam_Code;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Name = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PHP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```



# Class under test

```
package com.test;

public class StudentResults {
    private String Full_Name;
    private String Exam_Code;
    private String Exam_Score;
    private String Exam_Grade;

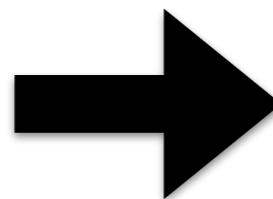
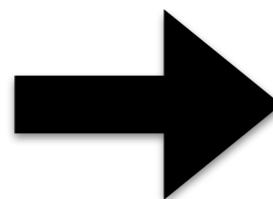
    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Code = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PHP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

Random  
sequential test

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner keys = new Scanner(System.in);
6         int length;
7         double width;
8         System.out.print("Enter the length ");
9         length=keys.nextInt();
10        System.out.print("Enter the width ");
11        width=keys.nextDouble();
12        System.out.println("length=" + length + " width=" +
13           width* area*(length*width));
14
15        System.out.print("Enter your full name (first, last) ");
16        String lastName;
17        String firstName;
18        firstName=keys.next();
19        lastName=keys.next();
20        System.out.println("Your name is "+lastName+" "+firstName);
21    }
22}
```



# Class under test

```
package ex0002;

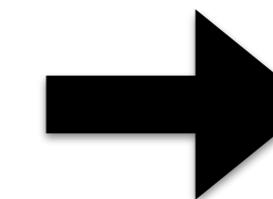
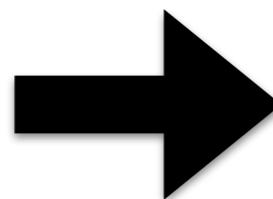
public class StudentResults {
    private String Full_Name;
    private String Exam_Code;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Code = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PEP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

**assert(p1)**



# Multi-threaded client

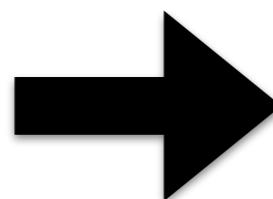
```
class Ex007 {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        num[7] = 10;
        System.out.println("This won't be displayed.");
    }

    class StackTraceBubble {
        static void myMethod() {
            try {
                genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        try {
            StackTraceBubble.myMethod();
            Ex007.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            // catch the exception
            System.out.println("Standard message here.");
            System.out.println(exc);
            System.out.println("Stack trace:");
            exc.printStackTrace();
        }
        System.out.println("After catch statement.");
    }
}
```

# Random sequential test

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner keys = new Scanner(System.in);
6         int length;
7         double width;
8         System.out.print("Enter the length ");
9         length=keys.nextInt();
10        System.out.print("Enter the width ");
11        width=keys.nextDouble();
12        System.out.println("length*"+length+"* width"
13           + width* " area*"+(length*width));
14
15        System.out.print("Enter your full name (first, last) ");
16        String lastName;
17        String firstName;
18        firstName=keys.next();
19        lastName=keys.next();
20        System.out.println("Your name is "+lastName+", "+firstName);
21    }
22}
```



# Class under test

```
package ex0002;

public class StudentResults {
    private String Full_Name;
    private String Exam_Code;
    private String Exam_Score;
    private String Exam_Grade;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Code = "Unknown";
        Exam_Score = "No Score";
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PEP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

**assert(p1)**

# Multi-threaded client

```
class Ex0Test {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        num[3] = 10;
        System.out.println("This won't be displayed.");
    }

    class StackTraceBubble {
        static void myMethod() {
            try {
                genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        try {
            StackTraceBubble.myMethod();
            Ex0Test.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            // catch the exception
            System.out.println("Standard message here.");
            System.out.println(exc);
            System.out.println("Stack trace:");
            exc.printStackTrace();
        }
        System.out.println("After catch statement.");
    }
}
```



# Random sequential test

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner keys = new Scanner(System.in);
        int length;
        double width;
        System.out.print("Enter the length ");
        length=keys.nextInt();
        System.out.print("Enter the width ");
        width=keys.nextDouble();
        System.out.println("length*"+length+"* width"
                           + width* " area*"+(length*width));
    }
}

System.out.print("Enter your full name (first, last) ");
String LastName;
String FirstName;
FirstName=keys.next();
LastName=keys.next();
System.out.println("Your name is "+LastName+" "+FirstName);
```

# Class under test

```
package exam02;

public class StudentResults {
    private String Full_Name;
    private int Exam_Score;
    private String Exam_Grade;
    private String Exam_Code;

    StudentResults() {
        Full_Name = "No Name Given";
        Exam_Score = 0;
        Exam_Grade = "Unknown";
    }

    String FullName(String name) {
        Full_Name = name;
        return Full_Name;
    }

    String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PEP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

**assert(p1)**

# Multi-threaded client

```
class ExTest {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        num[7] = 10;
        System.out.println("This won't be displayed.");
    }

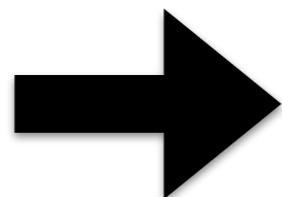
    class StackTraceBubble {
        static void myMethod() {
            try {
                genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }
    }

    public static void main(String[] args) {
        try {
            StackTraceBubble.myMethod();
            ExTest.genException();
        } catch (ArrayIndexOutOfBoundsException exc) {
            // catch the exception
            System.out.println("Standard message here");
            System.out.println(exc);
            System.out.println("Stack trace:");
            exc.printStackTrace();
        }
        System.out.println("After catch statement.");
    }
}
```

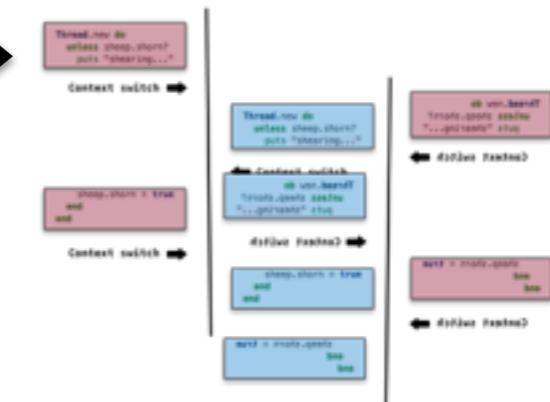


# Random sequential test

```
Project: Main
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner keys = new Scanner(System.in);
6         int length;
7         double width;
8         System.out.print("Enter the length ");
9         length=keys.nextInt();
10        System.out.print("Enter the width ");
11        width=keys.nextDouble();
12        System.out.println("length*"+length+"* width"
13                           + width* " area*"+(length*width));
14
15        System.out.print("Enter your full name (first, last) ");
16        String FirstName;
17        String LastName;
18        FirstName=keys.next();
19        LastName=keys.next();
20        System.out.println("Your name is "+LastName+", "+FirstName);
21    }
22}
```



# Thread interleaving



# Class under test

```
package com.k2;
public class StudentResults {
    private String Full_Name;
    private String Exam_Code;
    private String Exam_Score;
    private String Exam_Grade;
    ...
    public String FullName(String sName) {
        Full_Name = sName;
        return Full_Name;
    }
    public String ExamName(String examCode) {
        if (ExamCode.equals("VB")) {
            Exam_Name = "Visual Basic .NET";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "Java";
        } else if (ExamCode.equals("C#")) {
            Exam_Name = "C# .NET";
        } else if (ExamCode.equals("PHP")) {
            Exam_Name = "PEP";
        } else {
            Exam_Name = "No Exam Selected";
        }
        return Exam_Name;
    }
}
```

**assert(p1)**

# Multi-threaded client

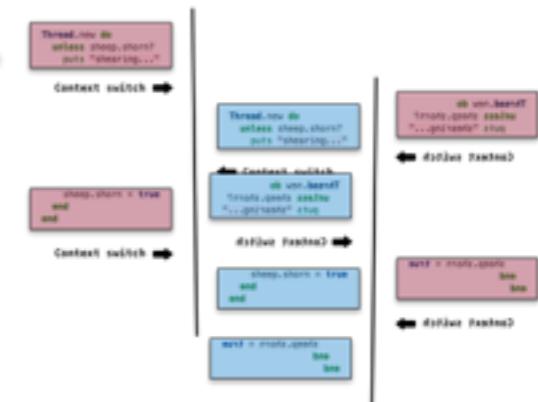
```
class K2Test {
    static void genException() {
        int num[] = new int[4];
        System.out.println("Before exception is generated.");
        num[3] = 10;
        System.out.println("This won't be displayed.");
    }
    class StackTraceBubble {
        static void myMethod() {
            try {
                genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                exc.printStackTrace();
            }
        }
        public static void main(String[] args) {
            try {
                StackTraceBubble.myMethod();
                K2Test.genException();
            } catch (ArrayIndexOutOfBoundsException exc) {
                // catch the exception
                System.out.println("Standard message here");
                System.out.println(exc);
                System.out.println("Stack trace:");
                exc.printStackTrace();
            }
            System.out.println("After catch statement.");
        }
    }
}
```



# Random sequential test

```
Project: Main
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner keys = new Scanner(System.in);
6         int length;
7         double width;
8         System.out.print("Enter the length ");
9         length=keys.nextInt();
10        System.out.print("Enter the width ");
11        width=keys.nextDouble();
12        System.out.println("length*"+length+"* width"
13                           + width* " area* "+(length*width));
14
15        System.out.print("Enter your full name (first, last) ");
16        String firstNames;
17        String lastName;
18        firstNames=keys.nextLine();
19        lastName=keys.nextLine();
20        System.out.println("Your name is "+lastName+" "+firstNames);
21    }
22}
```

# Thread interleaving



**Crash**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ... // returns on len <= 0
112         int avail = buf.length - pos;
113         if (avail > 0) {
114           if (len < avail)
115             avail = len;
116           System.arraycopy(buf, pos, cbuf, off, avail);
117           ...
118         }
119         ...
120       } catch (ArrayIndexOutOfBoundsException e) {
121         throw new IndexOutOfBoundsException();
122       }
123     }
124   }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ... // returns on len <= 0
112             int avail = buf.length - pos;
113             if (avail > 0) {
114                 if (len < avail)
115                     avail = len;
116                 System.arraycopy(buf, pos, cbuf, off, avail);
117             ...
118         }
119         ...
120     } catch (ArrayIndexOutOfBoundsException e) {
121         throw new IndexOutOfBoundsException();
122     }
123 }
124 }
```

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
                     throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             ...
111             ... // returns on len <= 0
112             int avail = buf.length - pos;
113             if (avail > 0) {
114                 if (len < avail)
115                     avail = len;
116             }
117             assert(buf != null);
118             ...
119             ...
120             ...
121             ...
122             ...
123             ...
124             ...
125             ...
126         }
127         ...
128     }
129 }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
                     throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             int avail = buf.length - pos;
111             if (avail > 0) {
112                 if (len < avail)
113                     avail = len;
114                 assert(buf != null);
115             }
116             ...
117             ...
118             int avail = buf.length - pos;
119             if (avail > 0) {
120                 if (len < avail)
121                     avail = len;
122                 assert(buf != null);
123             }
124             ...
125         } catch (ArrayIndexOutOfBoundsException e) {
126             throw new IndexOutOfBoundsException();
127         }
128     }
129 }
```

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
                     throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             int avail = assert(buf != null);
111             if (avail > 0) {
112                 if (len < avail)
113                     avail = len;
114                 assert(buf != null);
115             }
116             ...
117             ...
118         } catch (ArrayIndexOutOfBoundsException e) {
119             throw new IndexOutOfBoundsException();
120         }
121     }
122 }
```

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ... // returns on len <= 0
112         int avail = assert(buf != null);
113         if (avail > 0) {
114           if (len < avail)
115             avail = len;
116           assert(buf != null);
117           ...
118         }
119         ...
120       } catch (ArrayIndexOutOfBoundsException e) {
121         throw new IndexOutOfBoundsException();
122       }
123     }
124   }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ... // returns on len <= 0
112         int avail = assert(buf != null);
113         if (avail > 0) {
114           if (len < avail)
115             avail = len;
116           assert(buf != null);
117           ...
118         }
119         ...
120       } catch (ArrayIndexOutOfBoundsException e) {
121         throw new IndexOutOfBoundsException();
122       }
123     }
124   }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ... // returns on len <= 0
112             int avail = assert(buf != null);
113             if (avail > 0) {
114                 if (len < avail)
115                     avail = len;
116                 assert(buf != null); ← Fail this assert!
117                 ...
118             }
119             ...
120         } catch (ArrayIndexOutOfBoundsException e) {
121             throw new IndexOutOfBoundsException();
122         }
123     }
124 }
```

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
           throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             ... // returns on len <= 0
111             int avail = assert(buf != null); ← This assert
112             if (avail > 0) {                                holds!
113                 if (len < avail)
114                     avail = len;
115                 assert(buf != null); ← Fail this assert!
116             ...
117         }
118         ...
119     } catch (ArrayIndexOutOfBoundsException e) {
120         throw new IndexOutOfBoundsException();
121     }
122 }
123 }
```

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ... // returns on len <= 0
112         int avail = assert(buf != null);
113         if (avail > 0) {
114           if (len < avail)
115             avail = len;
116           assert(buf != null);
117           ...
118         }
119         ...
120       } catch (ArrayIndexOutOfBoundsException e) {
121         throw new IndexOutOfBoundsException();
122       }
123     }
124   }
```

**PushbackReader.java**

# Example from JDK 8

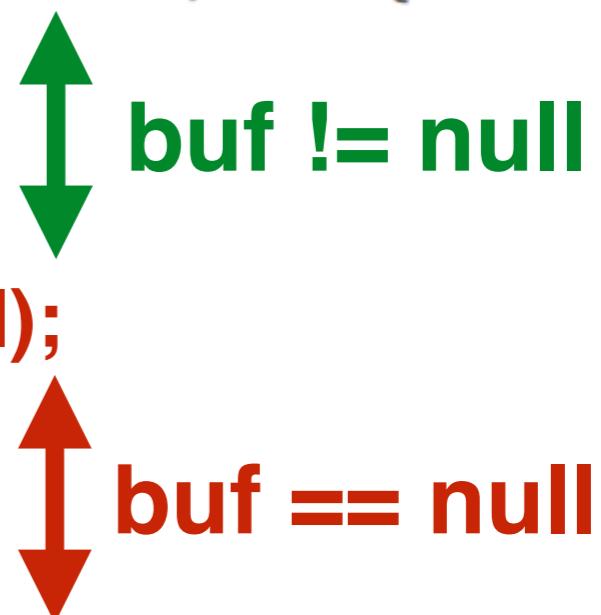
```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ...
112             int avail = assert(buf != null);
113             if (avail > 0) {
114                 if (len < avail)
115                     avail = len;
116                 assert(buf != null);
117                 ...
118             }
119             ...
120         } catch (ArrayIndexOutOfBoundsException e) {
121             throw new IndexOutOfBoundsException();
122         }
123     }
124 }
```



buf != null

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ...
112             // returns on len <= 0
113             int avail = assert(buf != null);
114             if (avail > 0) {
115                 if (len < avail)
116                     avail = len;
117                 assert(buf != null);
118                 ...
119             }
120             ...
121         } catch (ArrayIndexOutOfBoundsException e) {
122             throw new IndexOutOfBoundsException();
123         }
124     }
125 }
```



The diagram consists of two double-headed vertical arrows. A green arrow points downwards from the original code's `assert(buf != null);` to the annotated code's `buf != null`. A red arrow points upwards from the annotated code's `buf == null` back to the original code's `assert(buf != null);`.

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ...
112         // returns on len <= 0
113         int avail = buf.length - pos;
114         if (avail > 0) {
115           if (len < avail)
116             avail = len;
117           assert(buf != null);
118           ...
119         }
120         ...
121       } catch (ArrayIndexOutOfBoundsException e) {
122         throw new IndexOutOfBoundsException();
123       }
124     }
125   }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ...
112             // returns on len <= 0
113             int avail = buf.length - pos;
114             if (avail > 0) {
115                 if (len < avail)
116                     avail = len;
117             }
118             assert(buf != null);
119             ...
120             ...
121             ...
122         } catch (IOException e) {
123             throw new IndexOutOfBoundsException();
124         }
125     }
126 }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
107           throws IOException {
108     synchronized (lock) {
109       ensureOpen();
110       try {
111         ...
112         // returns on len <= 0
113         int avail = buf.length - pos;
114         if (avail > 0) {
115           if (len < avail)
116             avail = len;
117           assert(buf != null);
118           ...
119         }
120         ...
121       } catch (ArrayIndexOutOfBoundsException e) {
122         throw new IndexOutOfBoundsException();
123       }
124     }
125   }
```

**PushbackReader.java**

# Example from JDK 8

```
106 public int read(char cbuf[], int off, int len)
           throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             ... // returns on len <= 0
111             int avail = buf.length - pos;
112             if (avail > 0) { ← Evaluate to true
113                 if (len < avail)
114                     avail = len;
115                 assert(buf != null);
116                 ...
117             }
118             ...
119         } catch (ArrayIndexOutOfBoundsException e) {
120             throw new IndexOutOfBoundsException();
121         }
122     }
123 }
```

# Example from JDK 8

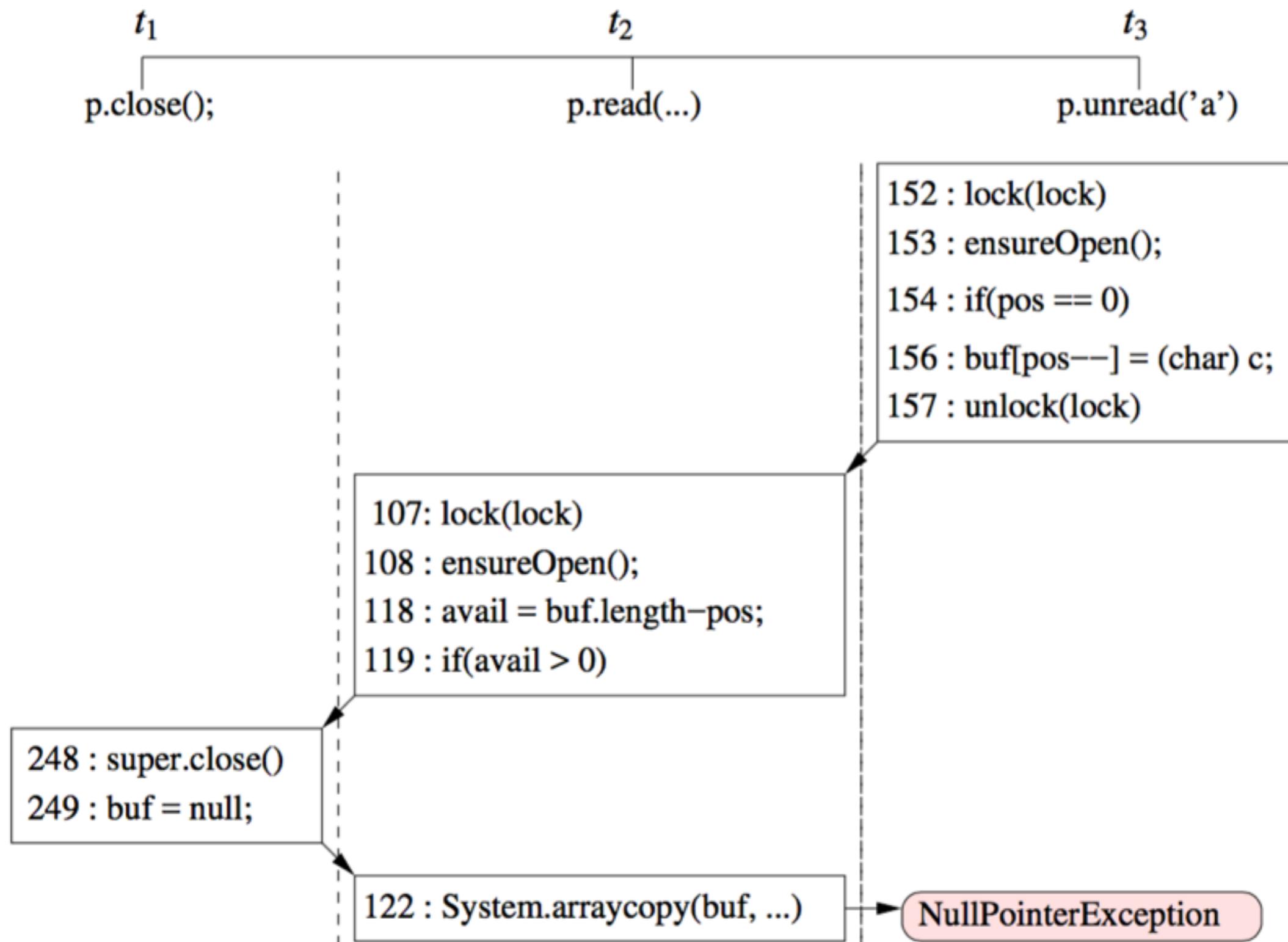
```
106 public int read(char cbuf[], int off, int len)
           throws IOException {
107     synchronized (lock) {
108         ensureOpen();
109         try {
110             ... // returns on len <= 0
111             int avail = buf.length - pos;
112             if (avail > 0) { ← Evaluate to true
113                 if (len < avail)
114                     avail = len;
115                 assert(buf != null);
116                 ...
117             }
118             ...
119         } catch (ArrayIndexOutOfBoundsException e) {
120             throw new IndexOutOfBoundsException();
121         }
122     }
123 }
```

# Example from JDK 8



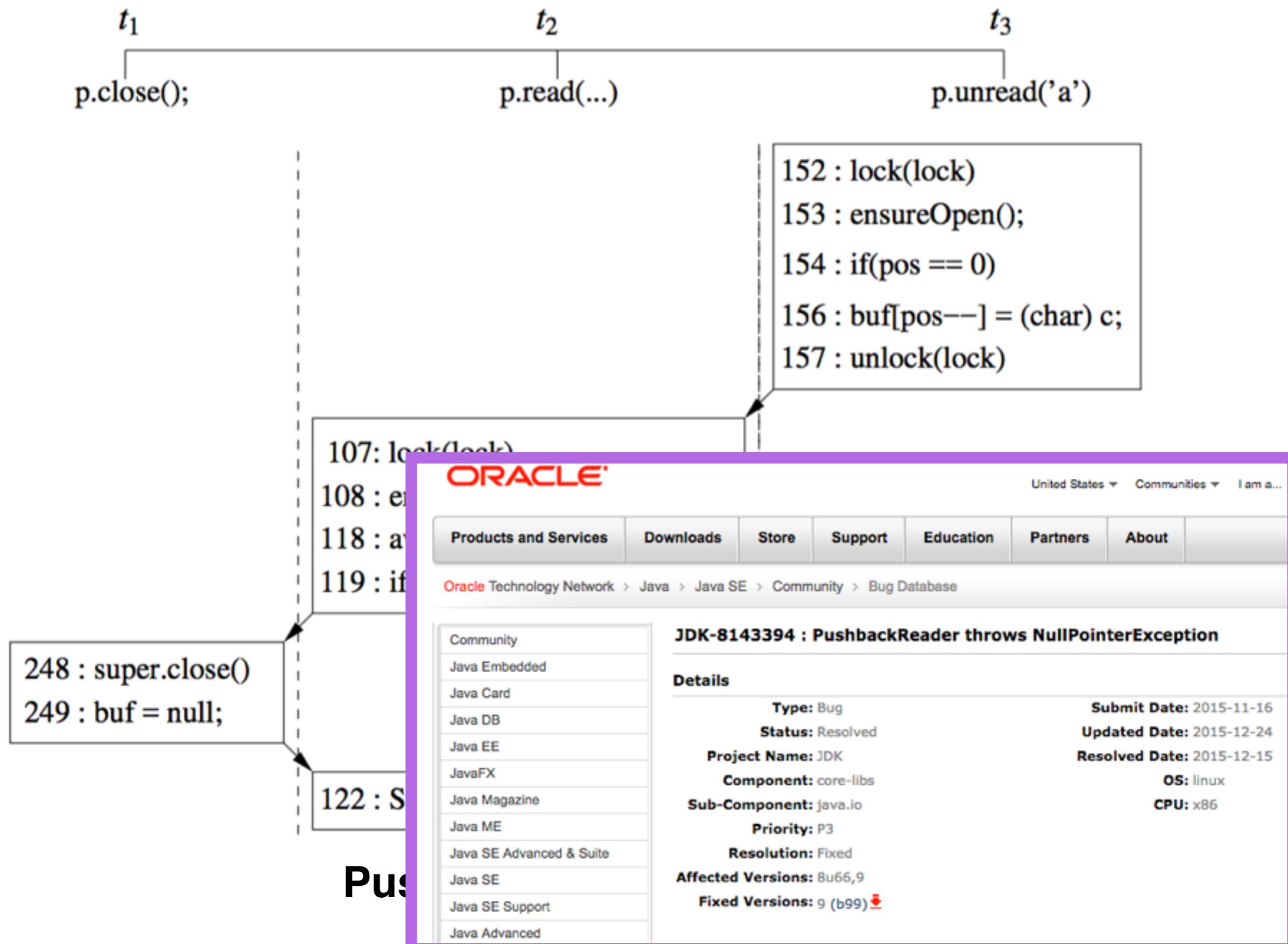
```
106 public int read(char cbuf[], int off, int len)
107     throws IOException {
108     synchronized (lock) {
109         ensureOpen();
110         try {
111             ...
112             // returns on len <= 0
113             int avail = buf.length - pos;
114             if (avail > 0) {
115                 if (len < avail)
116                     avail = len;
117                 assert(buf != null);
118                 ...
119             }
120             ...
121         } catch (ArrayIndexOutOfBoundsException e) {
122             throw new IndexOutOfBoundsException();
123         }
124     }
125 }
```

# Crash found by Minion in JDK8



**PushbackReader.java**

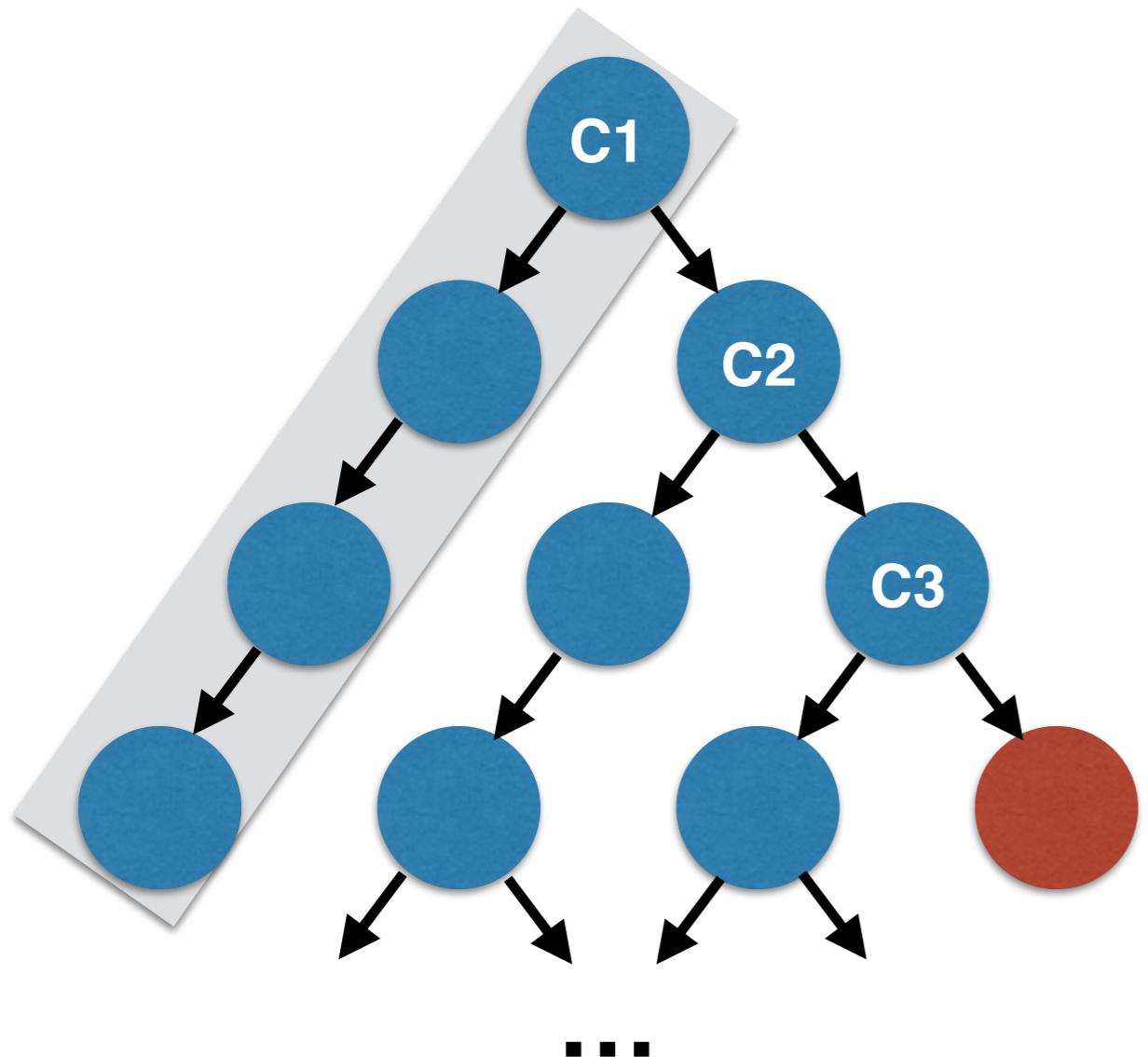
# Crash found by Minion in JDK8



# Execution synthesis

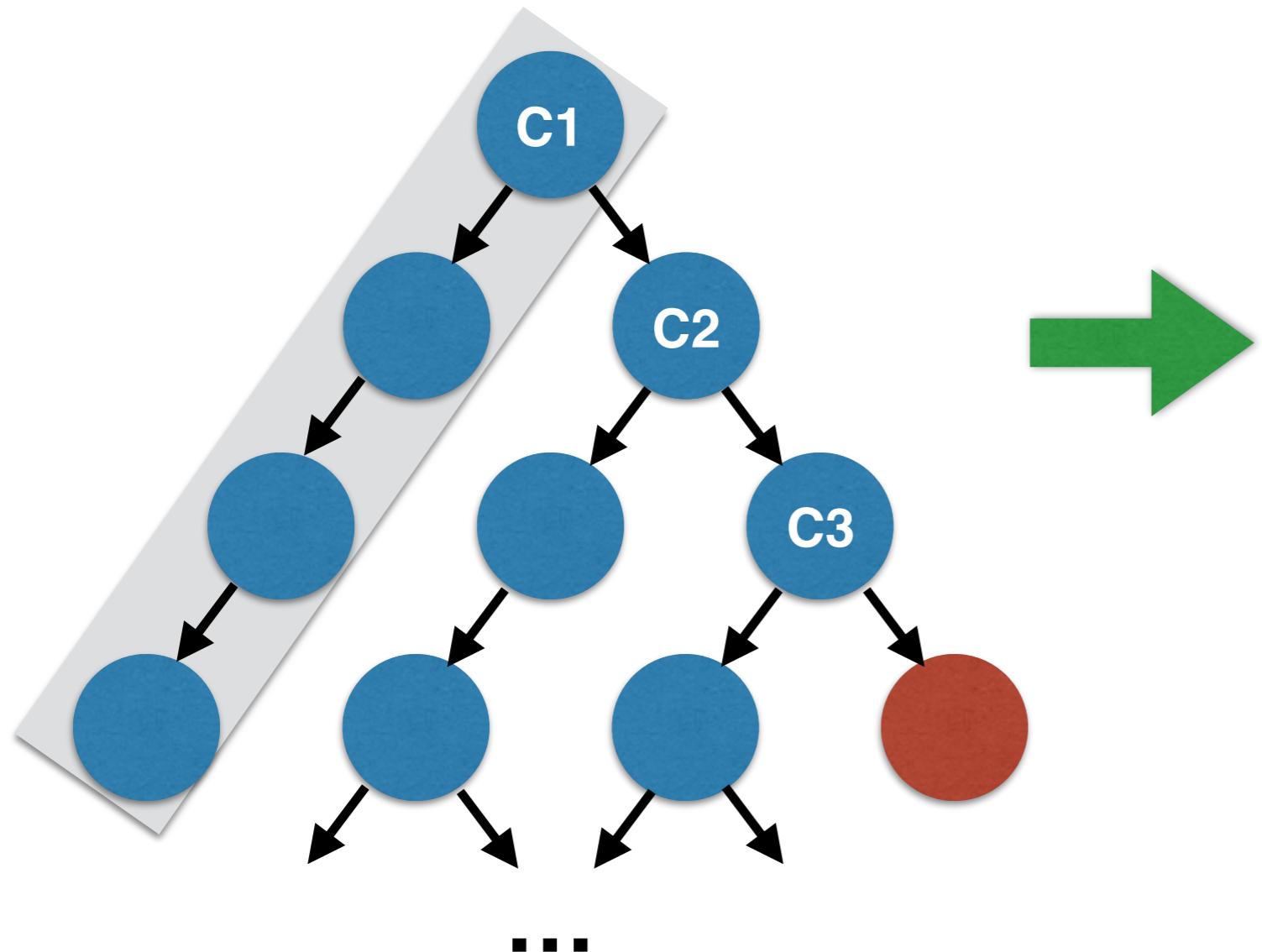
# Execution synthesis

Random execution



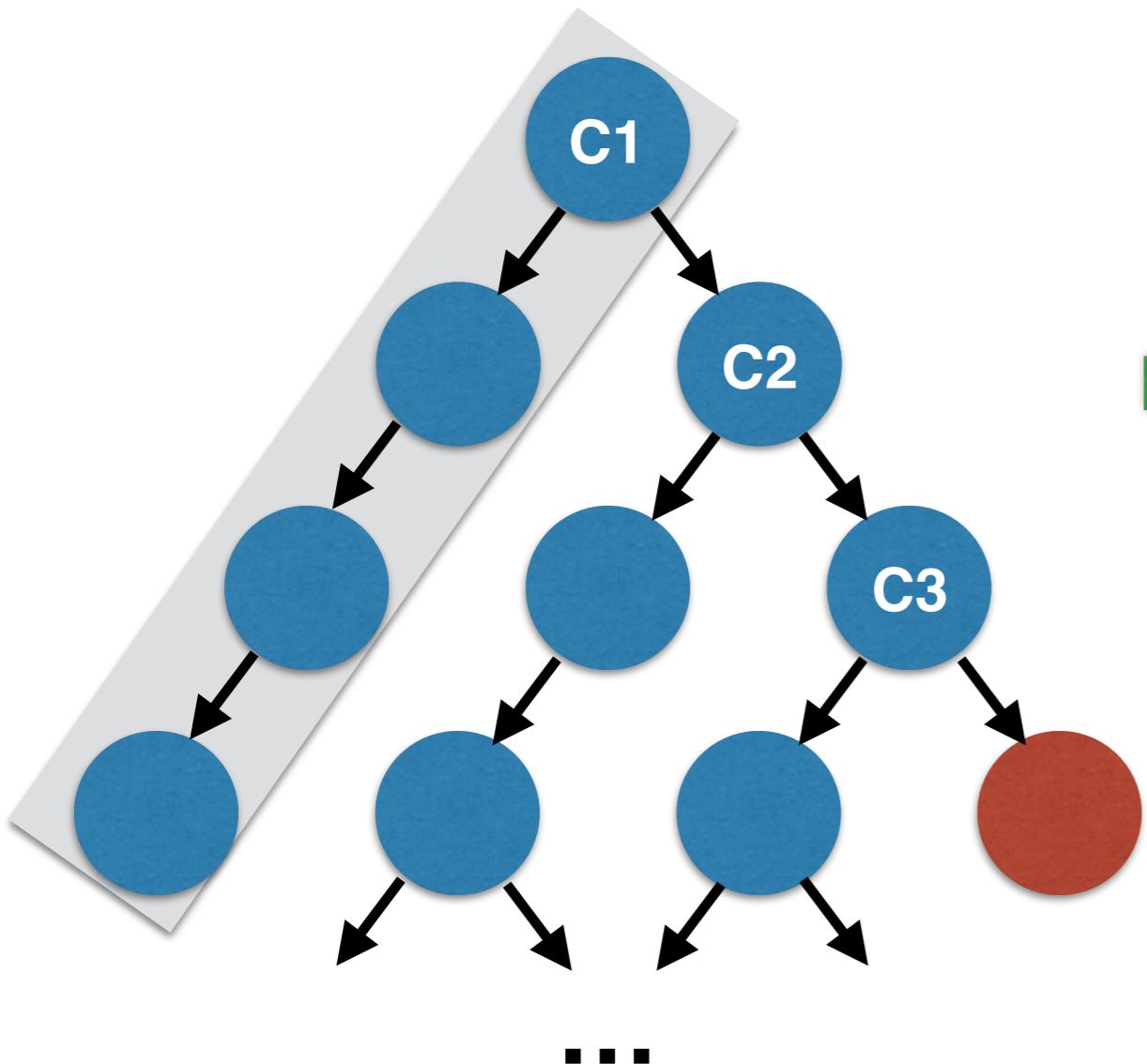
# Execution synthesis

Random execution

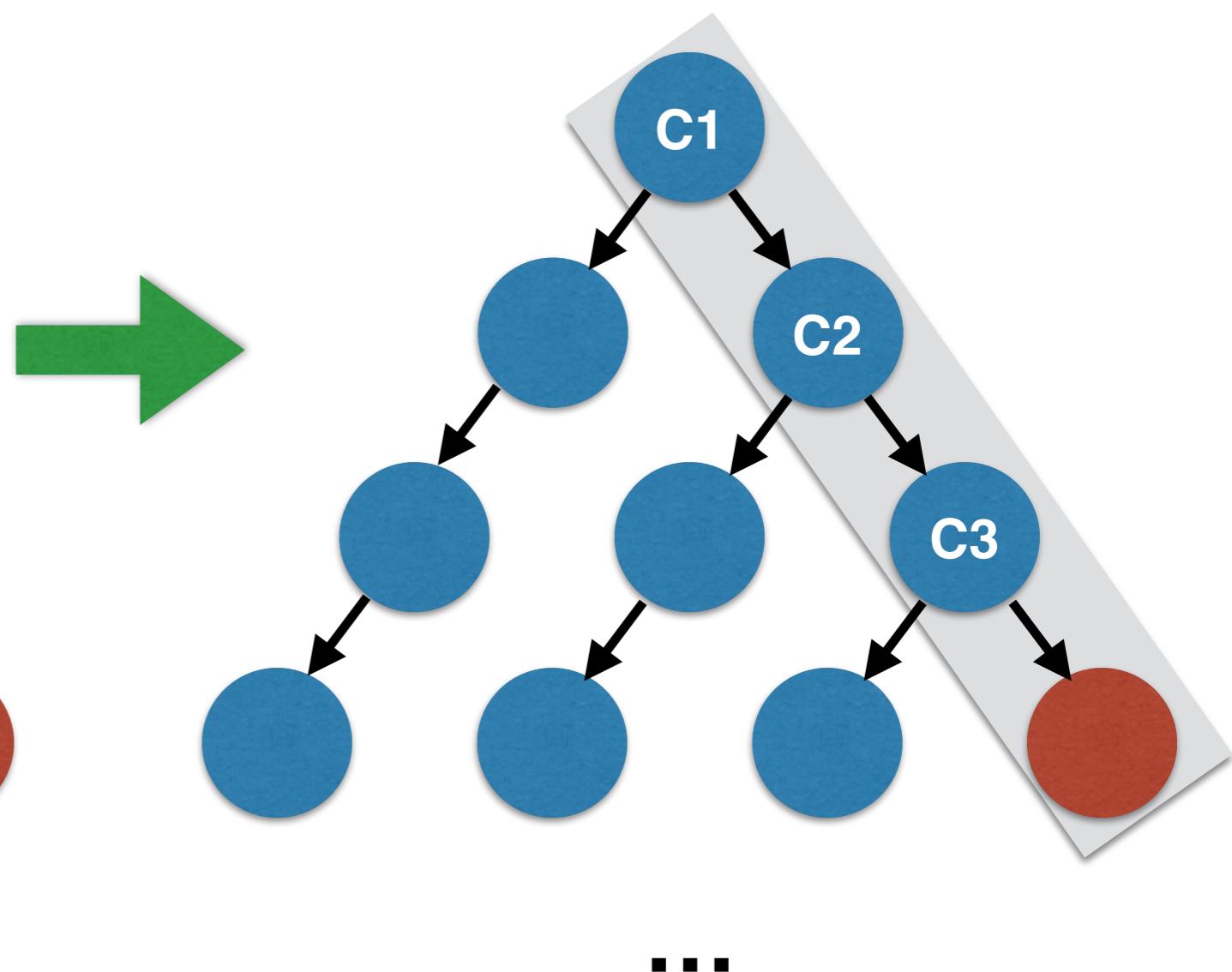


# Execution synthesis

Random execution

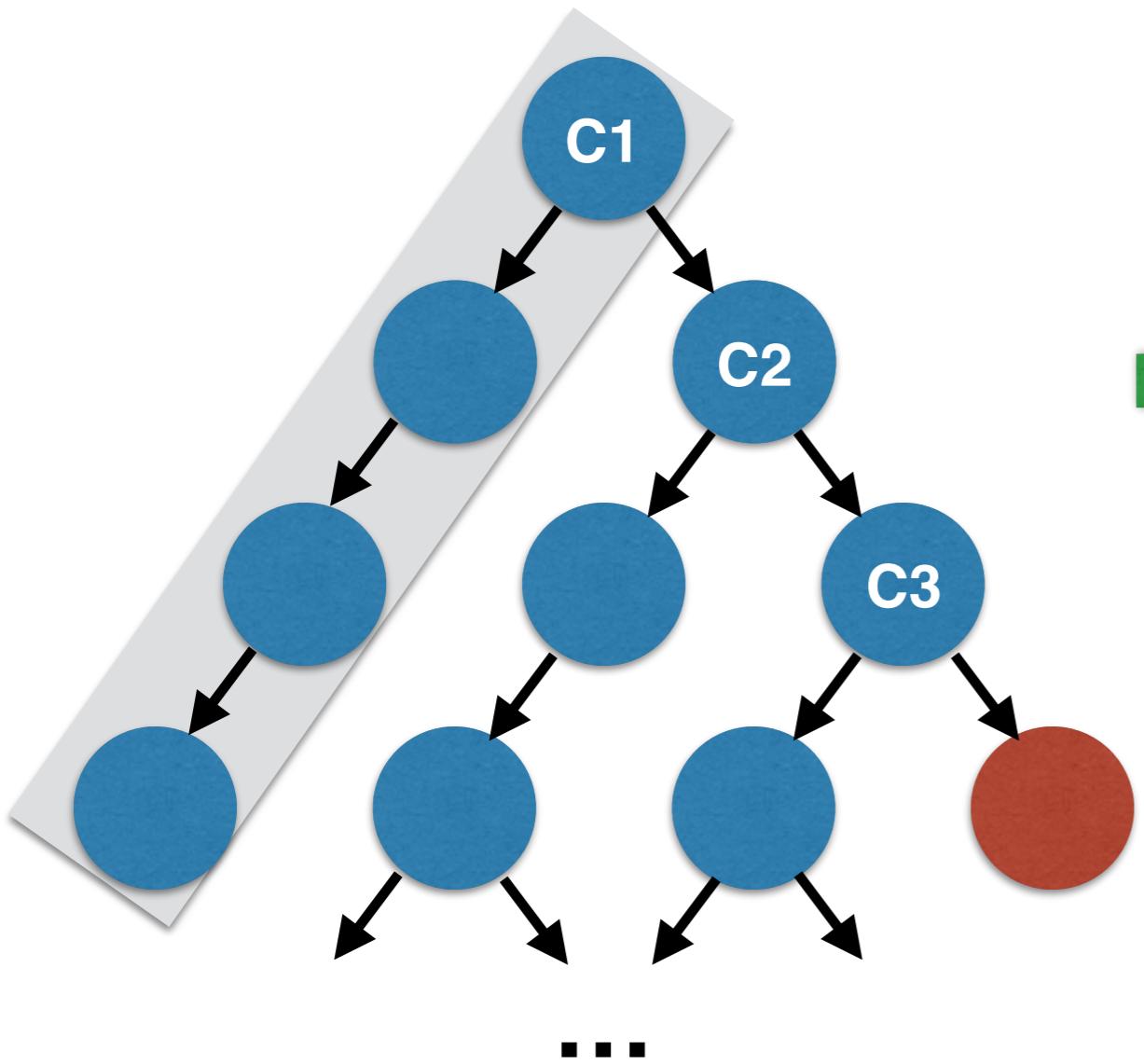


Target execution

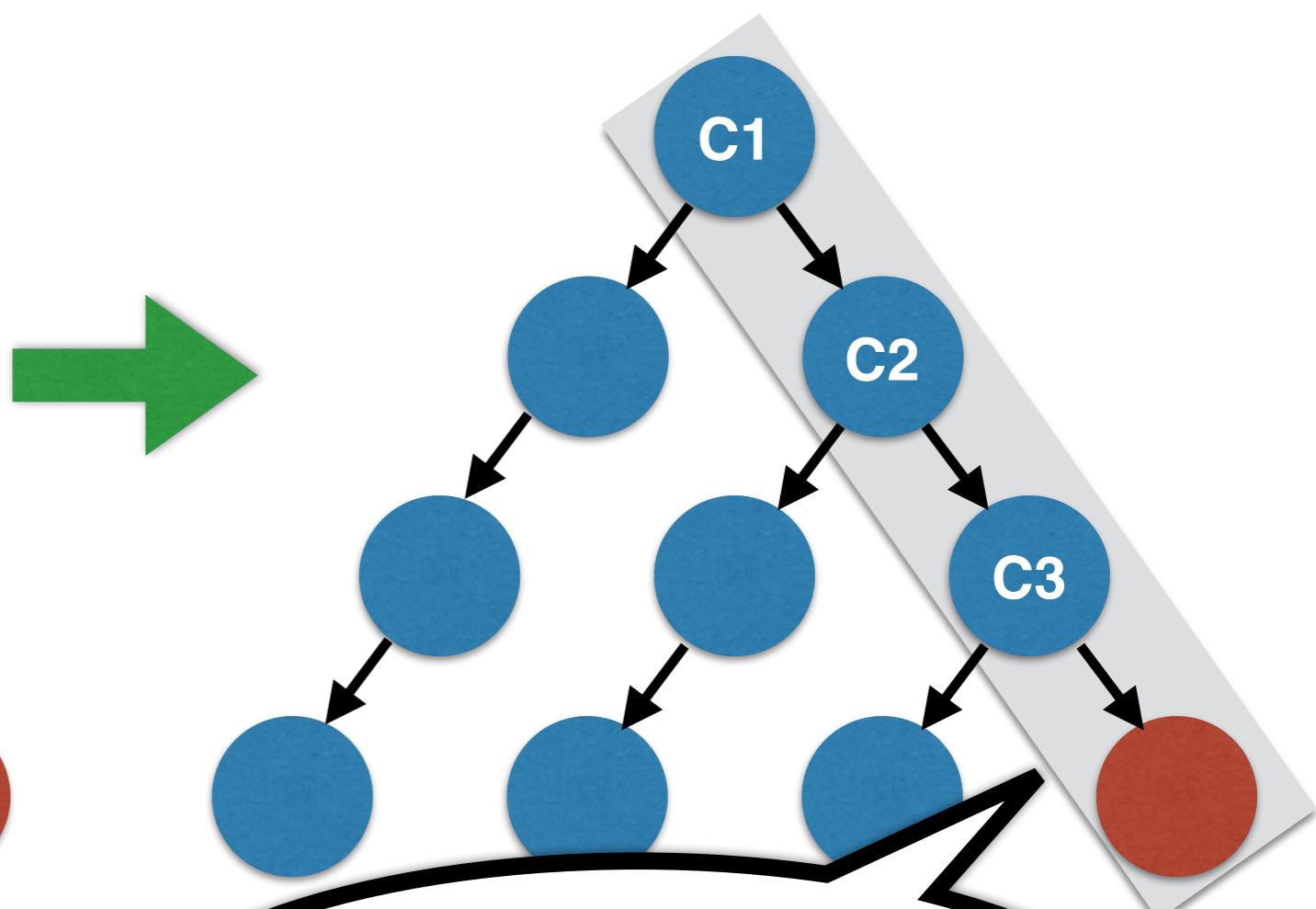


# Execution synthesis

Random execution



Target execution



The path can be  
sequentially infeasible

# Execution synthesis

**$t_1: m_3$**

**$t_2: m_1$**

**$t_3: m_2$**

# Execution synthesis

- Methods invoked from different threads appropriately

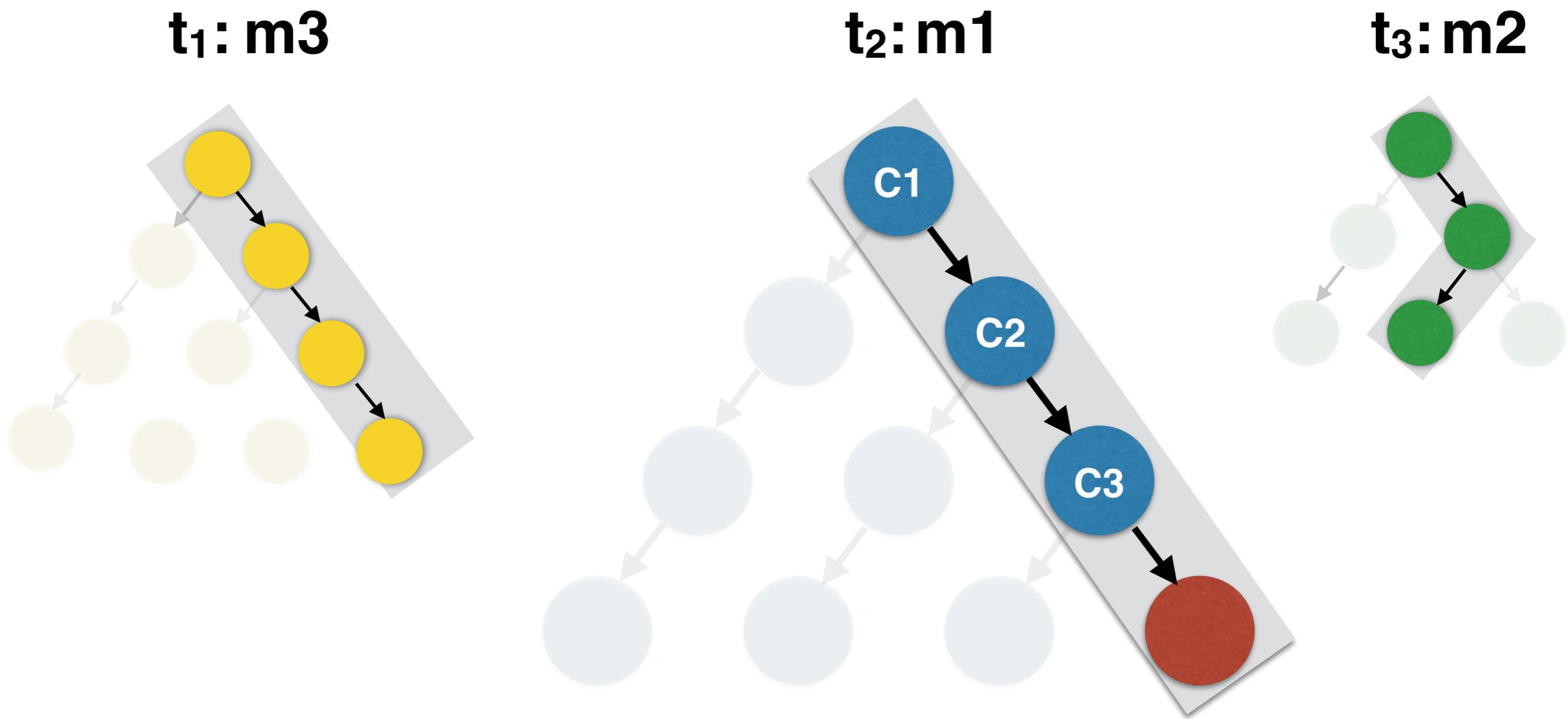
**$t_1: m_3$**

**$t_2: m_1$**

**$t_3: m_2$**

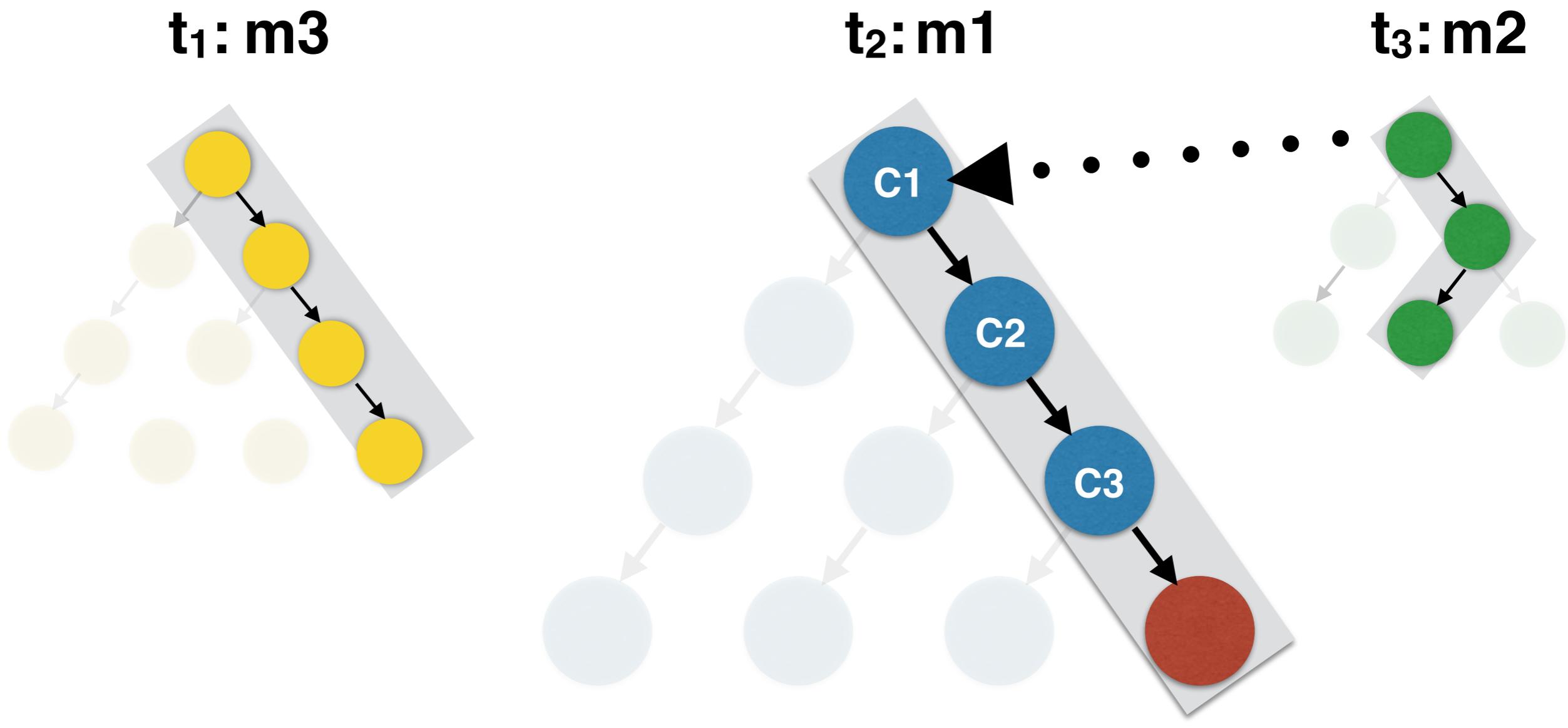
# Execution synthesis

- Methods invoked from different threads appropriately



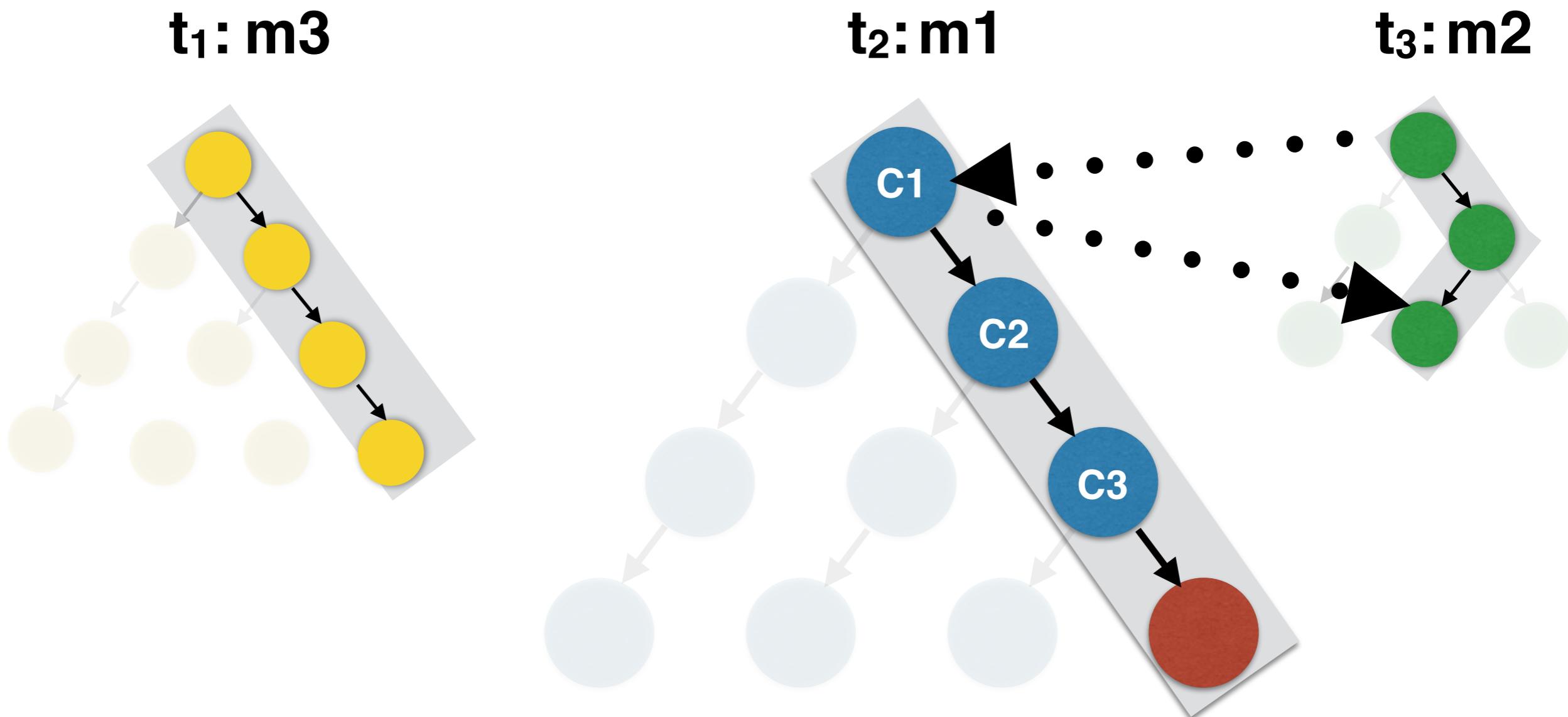
# Execution synthesis

- Methods invoked from different threads appropriately
- Specific interleaving needs to be followed



# Execution synthesis

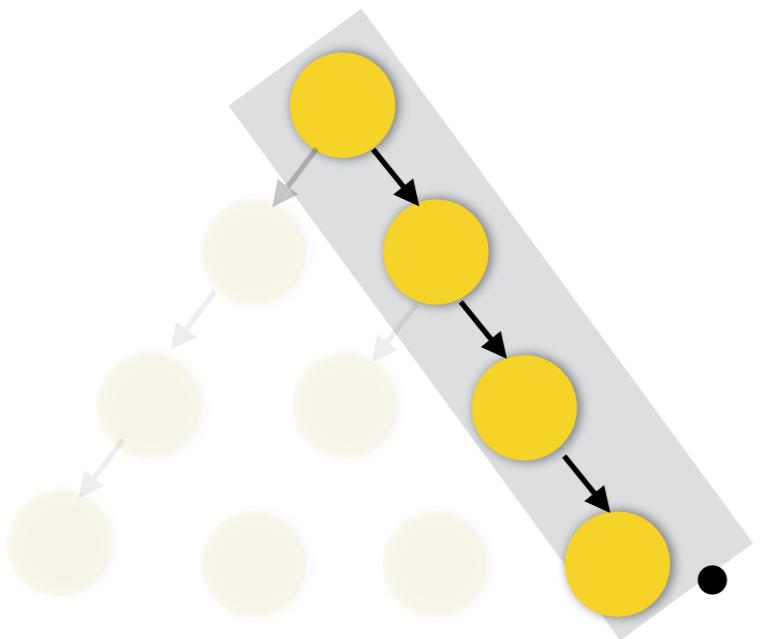
- Methods invoked from different threads appropriately
- Specific interleaving needs to be followed



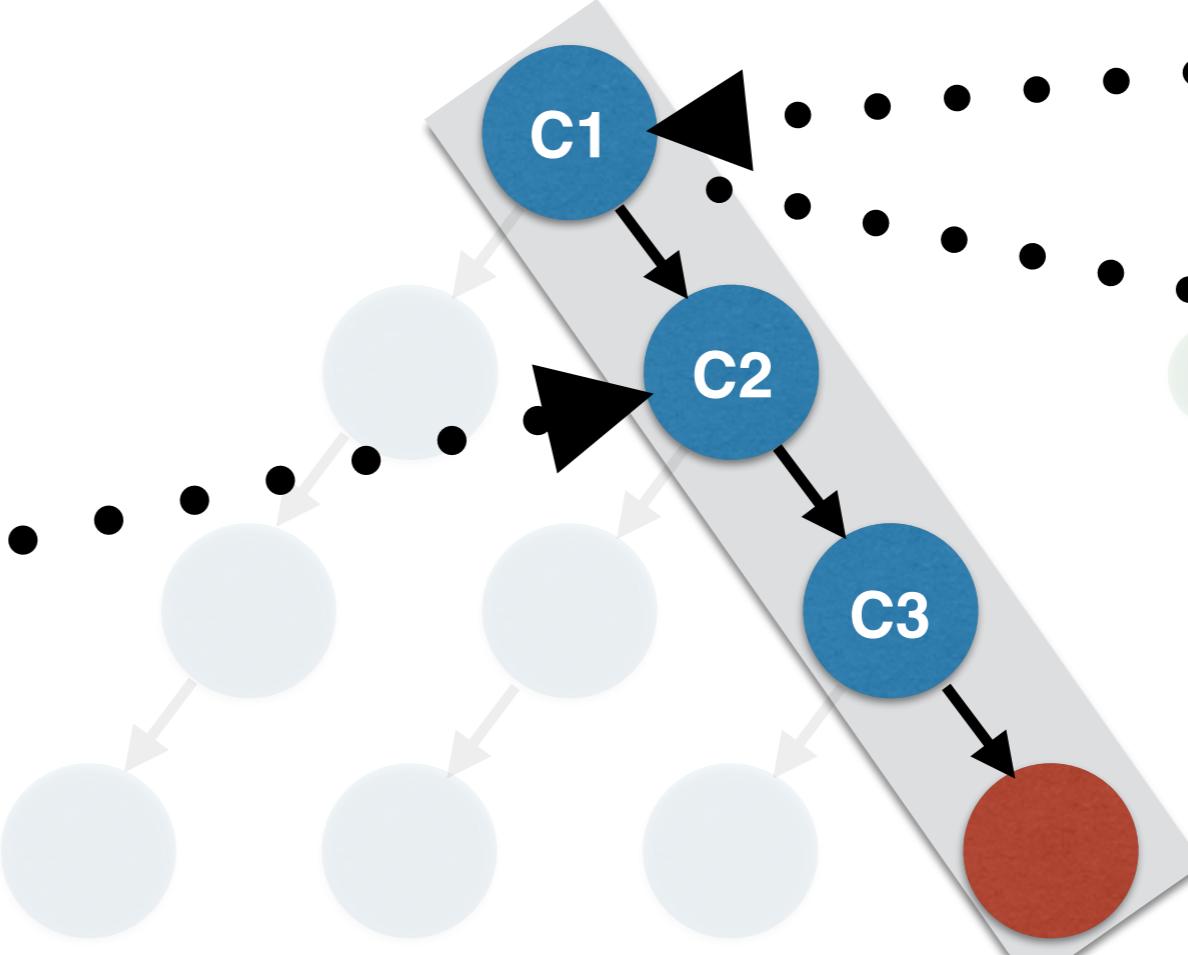
# Execution synthesis

- Methods invoked from different threads appropriately
- Specific interleaving needs to be followed

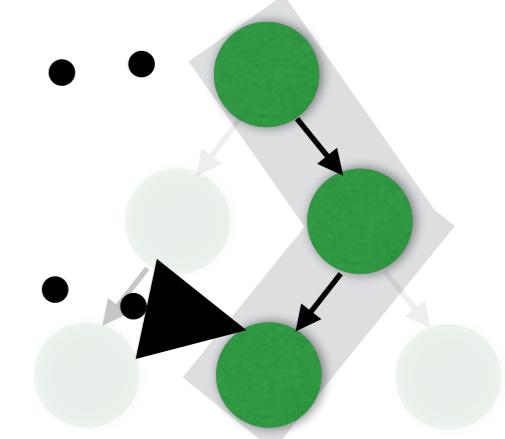
$t_1: m_3$



$t_2: m_1$



$t_3: m_2$



# Approach Overview

# Approach Overview

- ◆ **Static analysis**

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields

# Approach Overview

- ◆ **Static analysis**

- ◆ targets: locate assertions to violate and updates to fields
- ◆ derive path conditions to reach target instructions

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions
- ◆ **Dynamic analysis**

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions
- ◆ **Dynamic analysis**
  - ◆ obtain concrete data by executing provided tests

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions
- ◆ **Dynamic analysis**
  - ◆ obtain concrete data by executing provided tests
    - ◆ path conditions traversed, value of fields, etc

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions
- ◆ **Dynamic analysis**
  - ◆ obtain concrete data by executing provided tests
    - ◆ path conditions traversed, value of fields, etc
- ◆ **Constraint solvers**

# Approach Overview

- ◆ **Static analysis**

- ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions

- ◆ **Dynamic analysis**

- ◆ obtain concrete data by executing provided tests
    - ◆ path conditions traversed, value of fields, etc

- ◆ **Constraint solvers**

- ◆ encode path constraints, read-write constraints, lock constraints and parameter constraints using information from static and dynamic analysis

# Approach Overview

- ◆ **Static analysis**

- ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions

- ◆ **Dynamic analysis**

- ◆ obtain concrete data by executing provided tests
    - ◆ path conditions traversed, value of fields, etc

- ◆ **Constraint solvers**

- ◆ encode path constraints, read-write constraints, lock constraints and parameter constraints using information from static and dynamic analysis
  - ◆ synthesize structure of new clients and schedules

# Approach Overview

- ◆ **Static analysis**
  - ◆ targets: locate assertions to violate and updates to fields
  - ◆ derive path conditions to reach target instructions
- ◆ **Dynamic analysis**
  - ◆ obtain concrete data by executing provided tests
    - ◆ path conditions traversed, value of fields, etc
- ◆ **Constraint solvers**
  - ◆ encode path constraints, read-write constraints, lock constraints and parameter constraints using information from static and dynamic analysis
  - ◆ synthesize structure of new clients and schedules
- ◆ **Leverage the above components; iterate until target is reached**

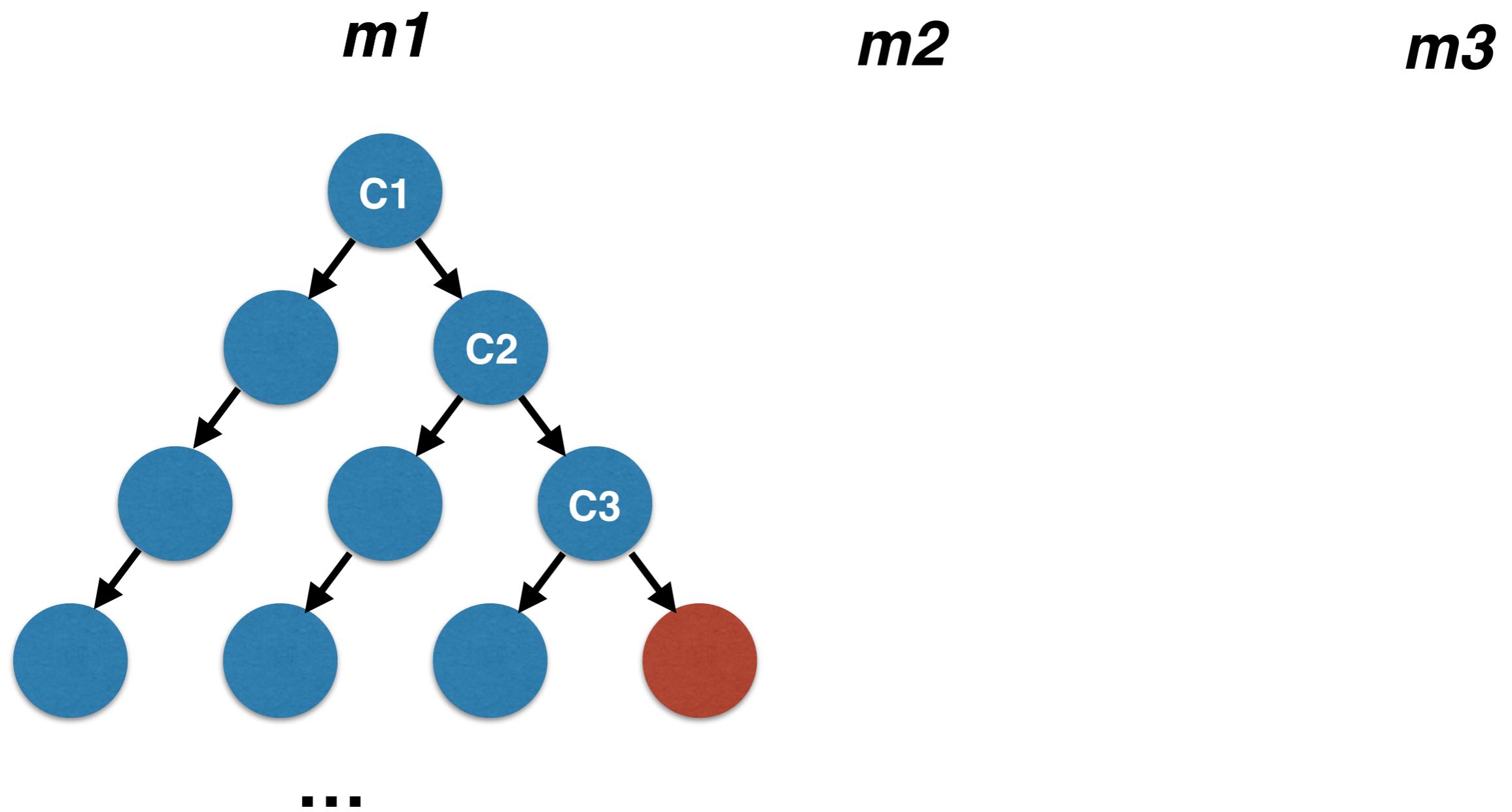
# Iteration - 1 : Static analysis

*m1*

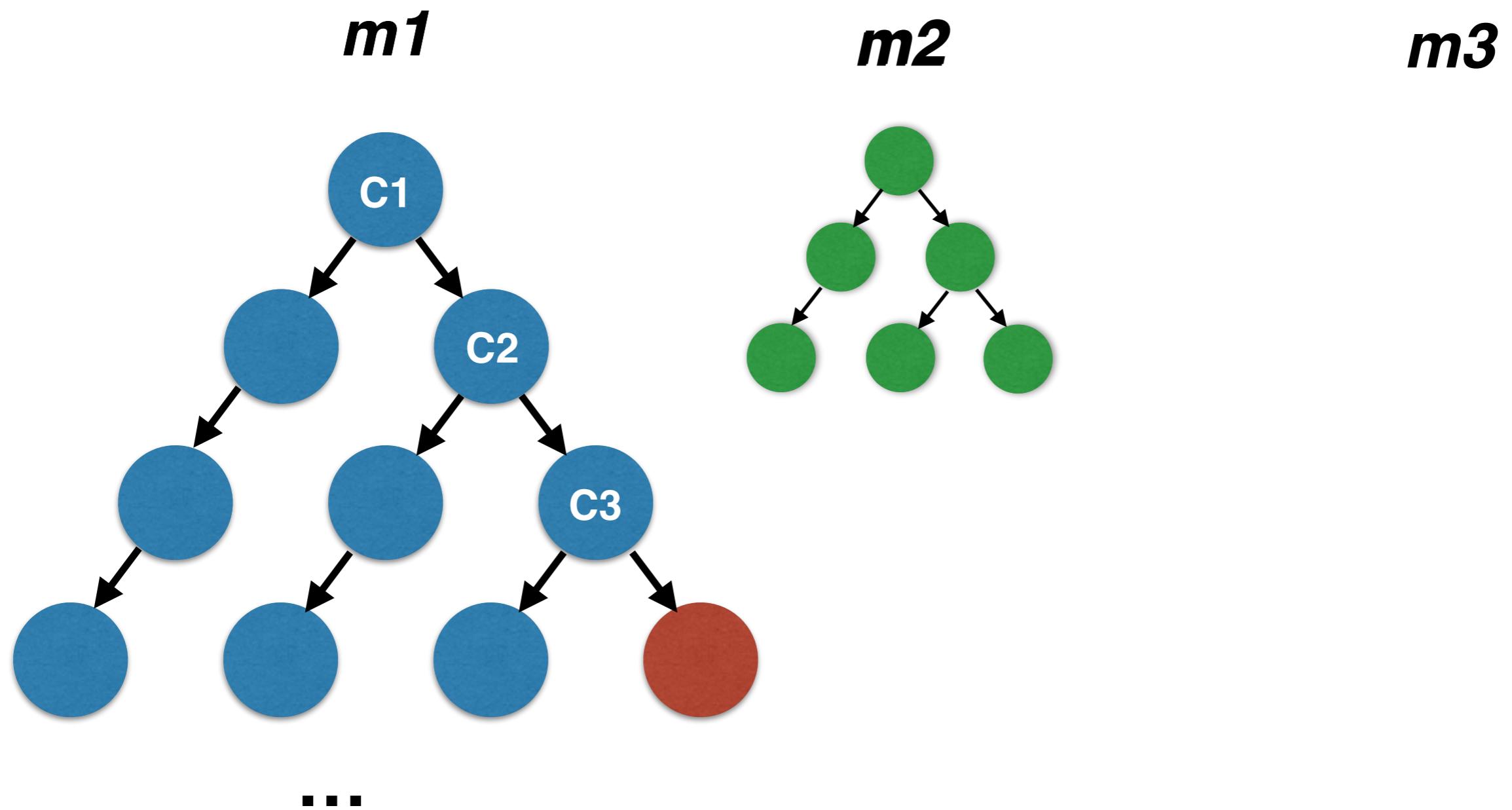
*m2*

*m3*

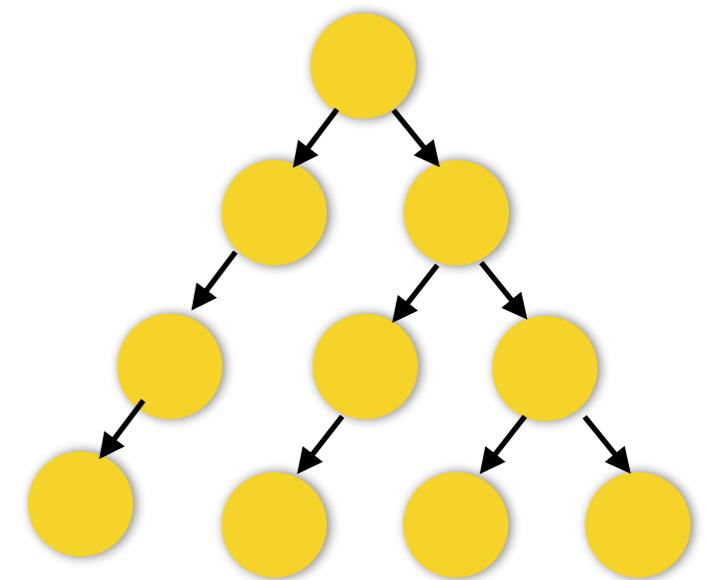
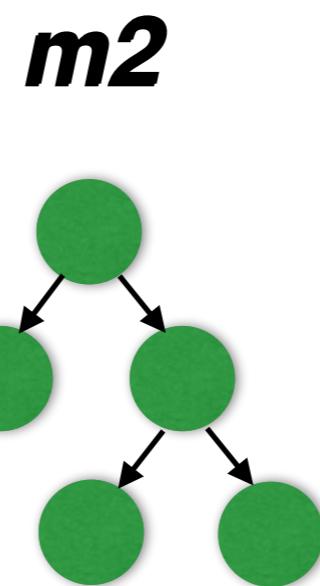
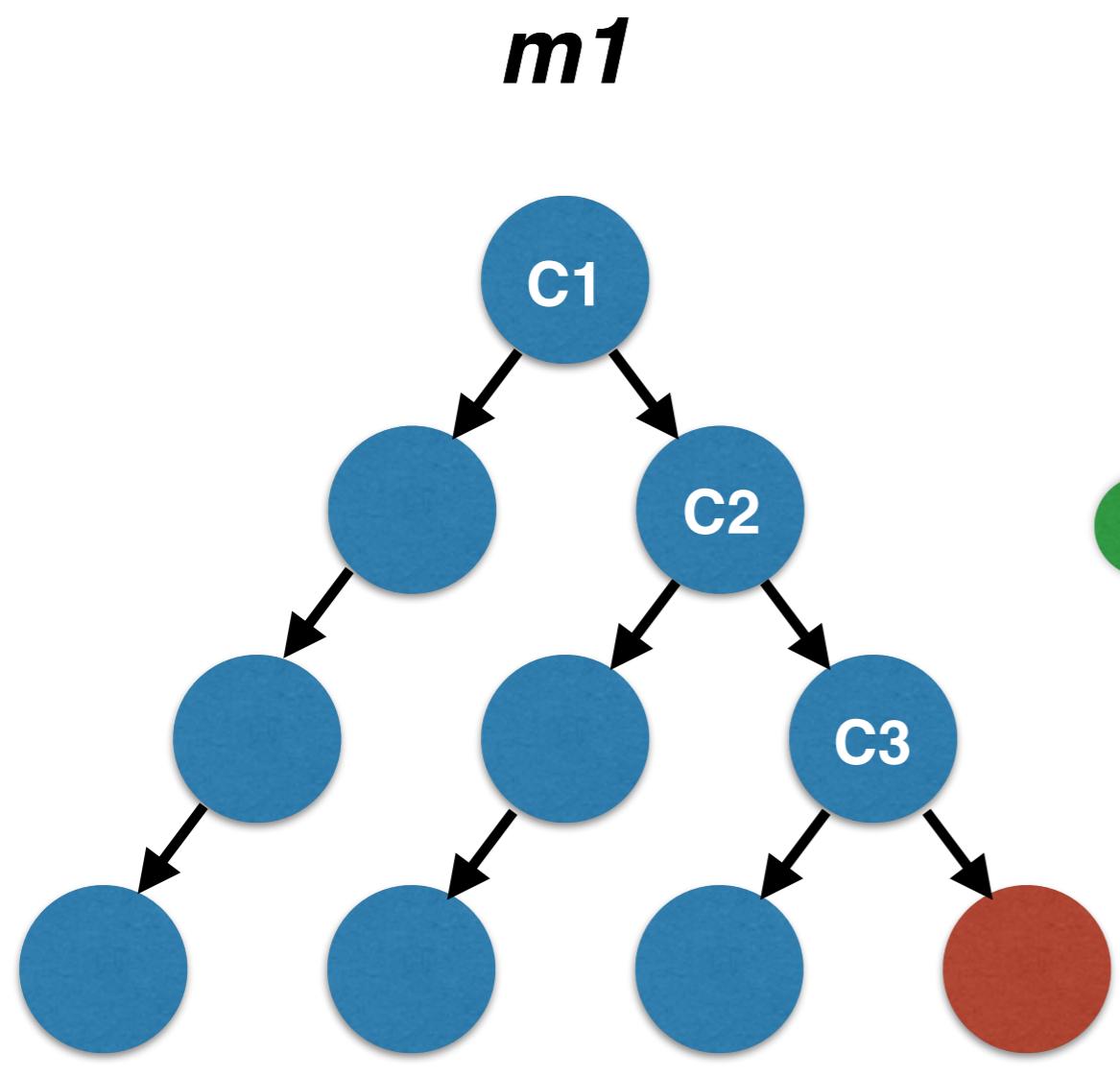
# Iteration - 1: Static analysis



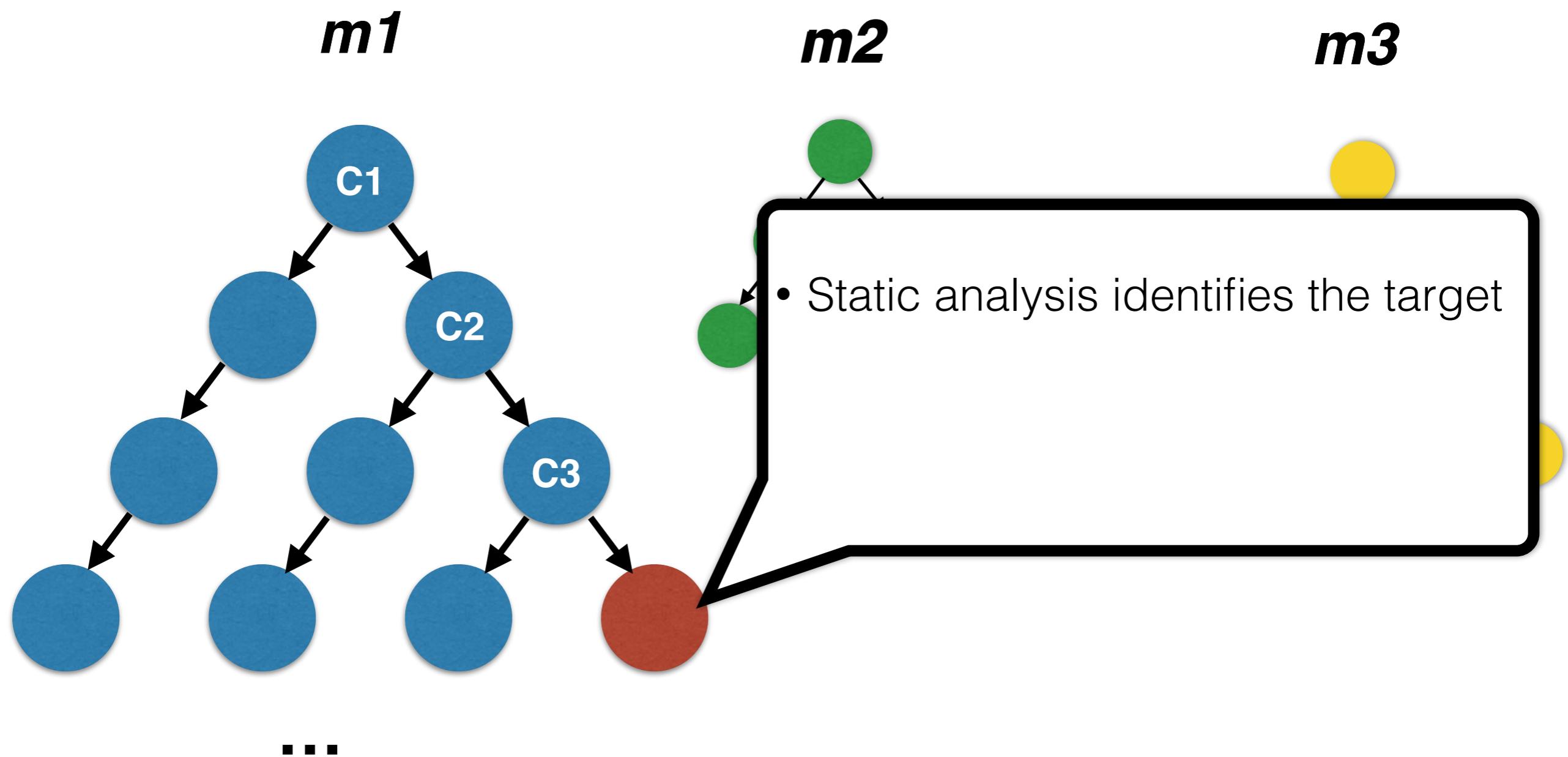
# Iteration - 1 : Static analysis



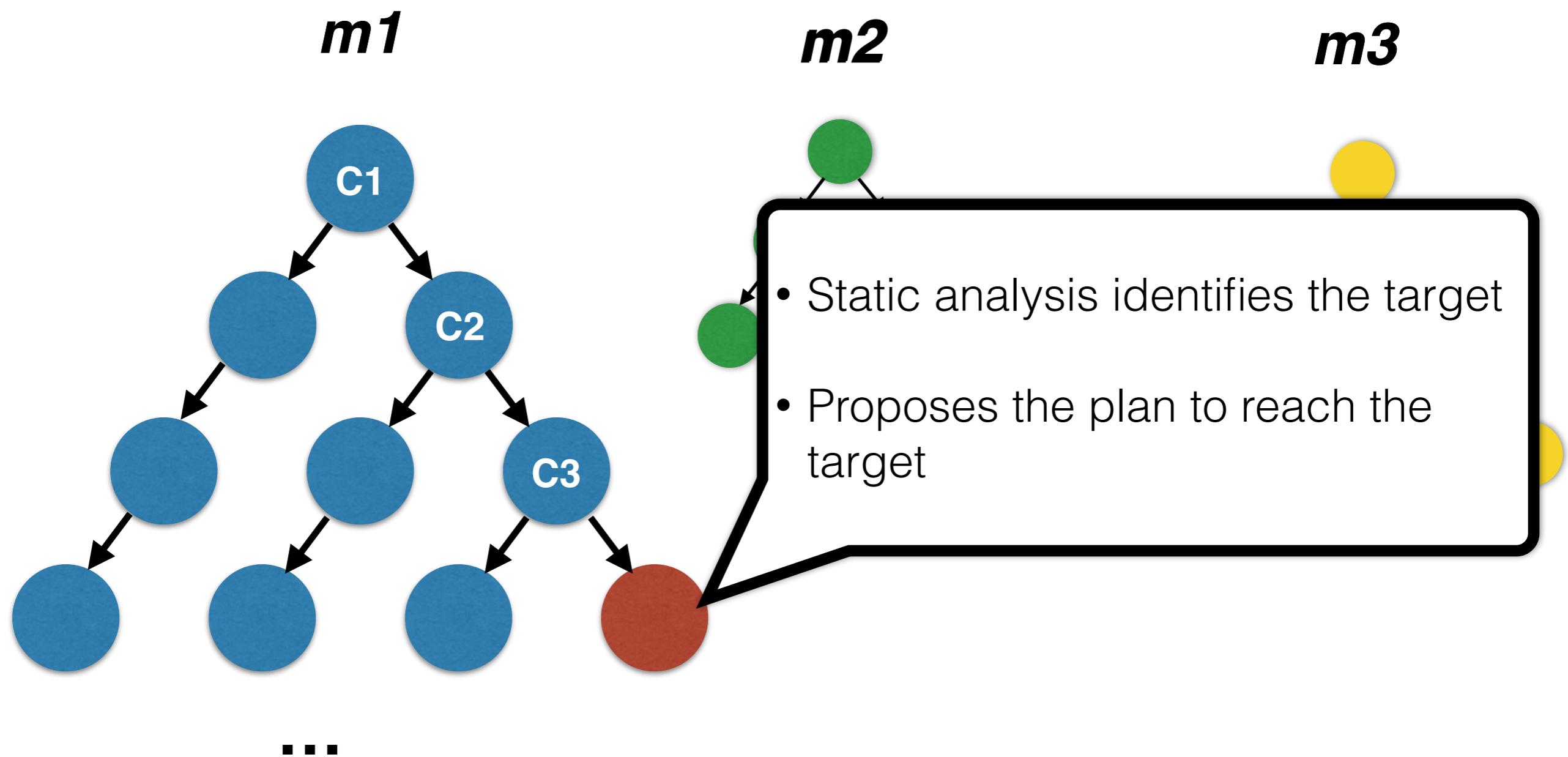
# Iteration - 1: Static analysis



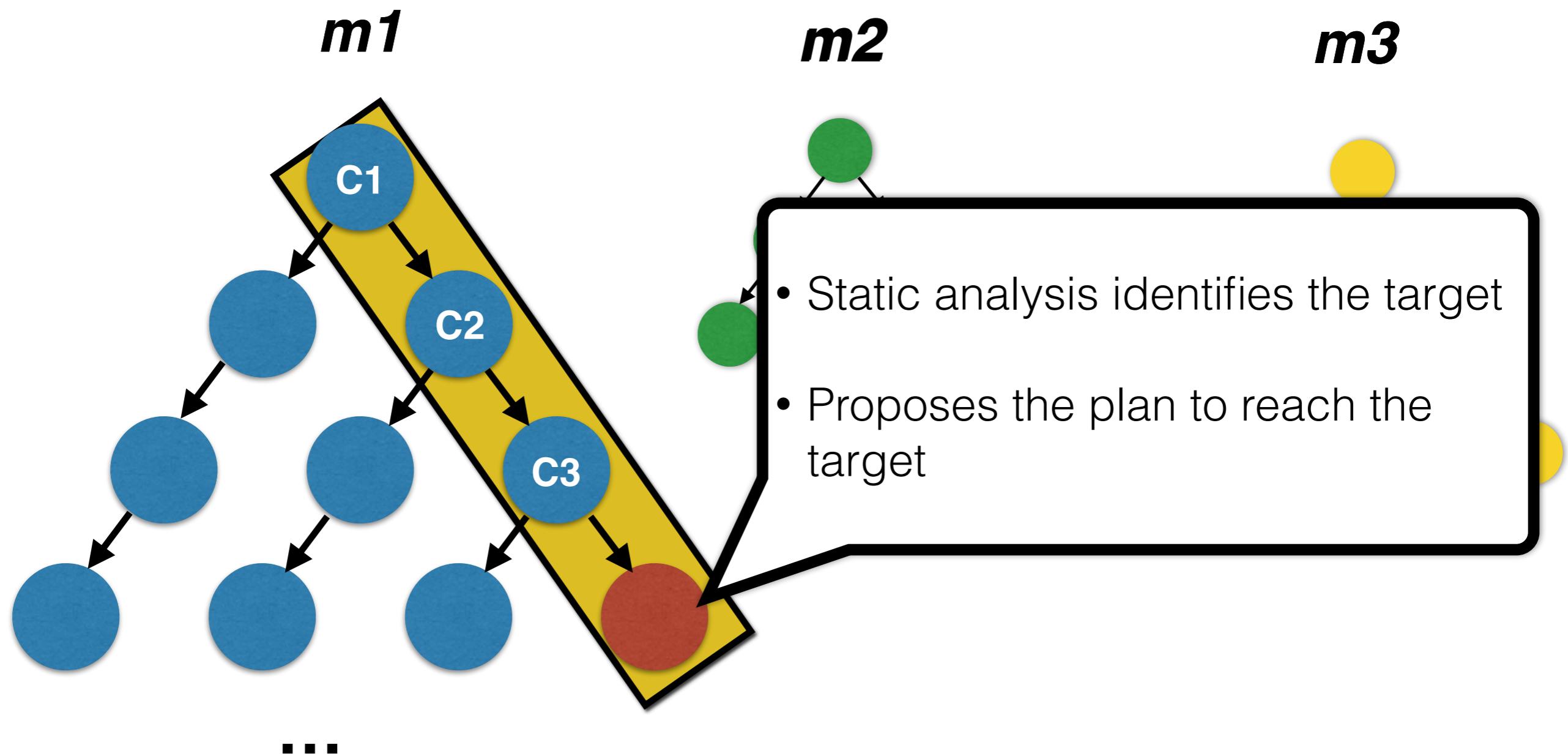
# Iteration - 1 : Static analysis



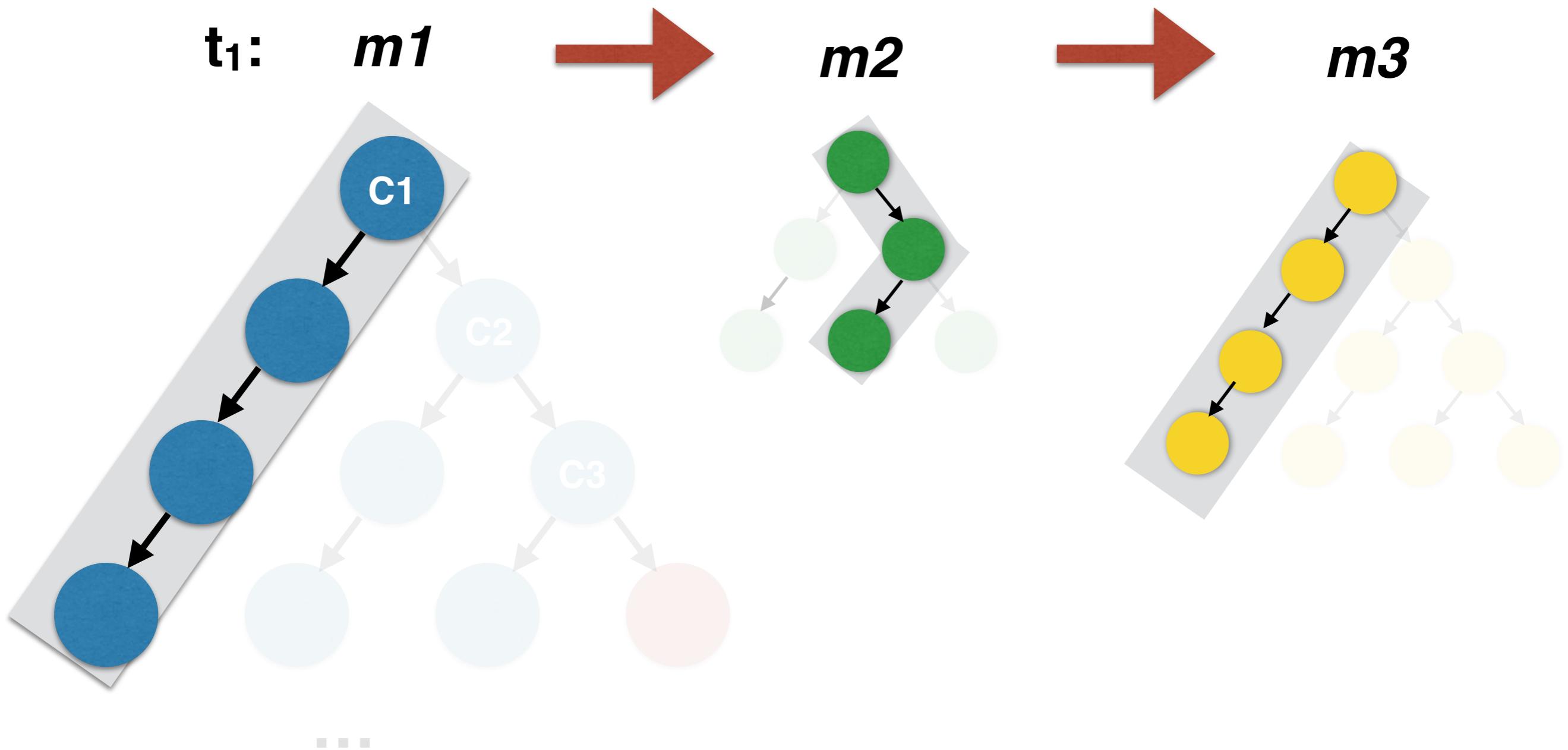
# Iteration - 1 : Static analysis



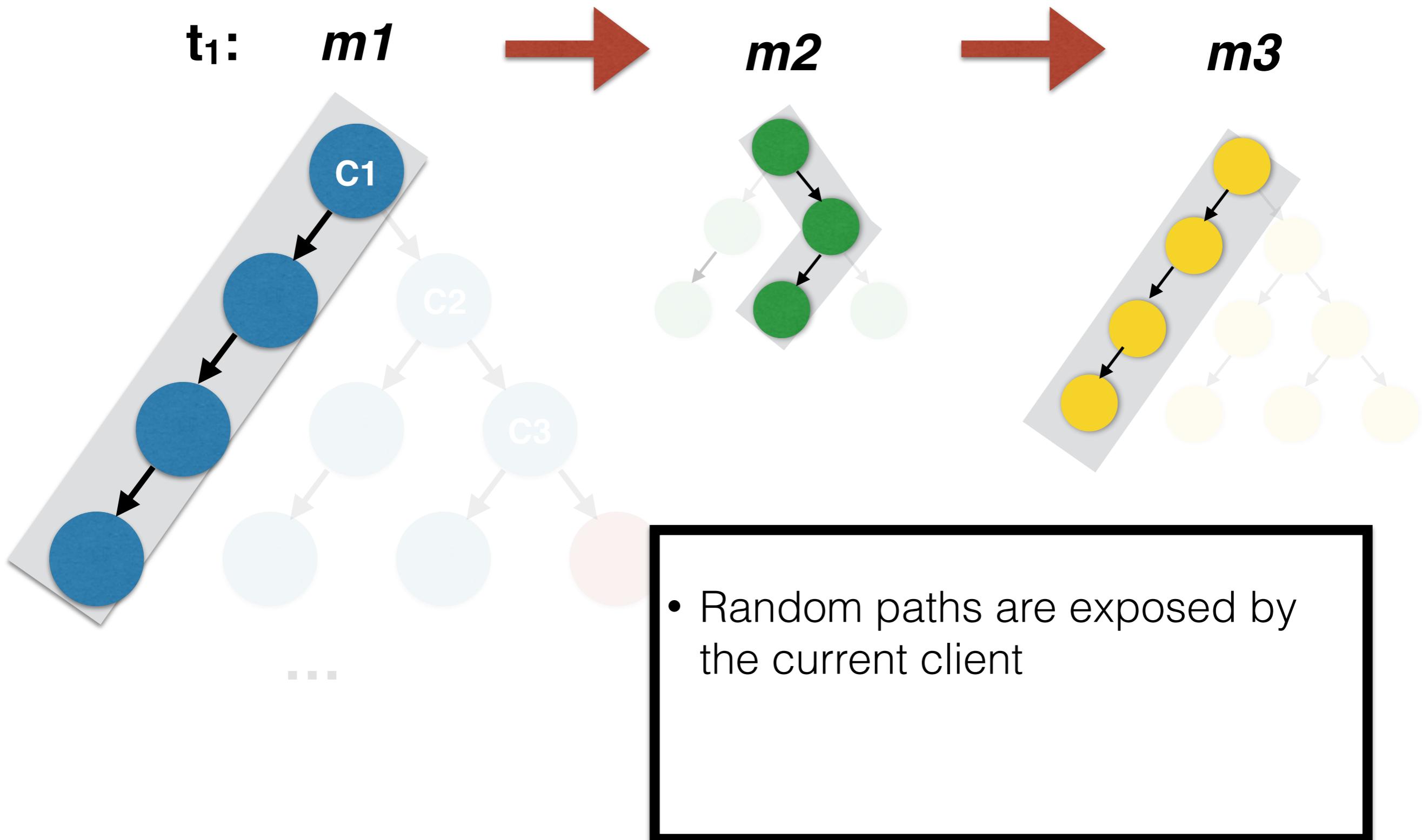
# Iteration - 1 : Static analysis



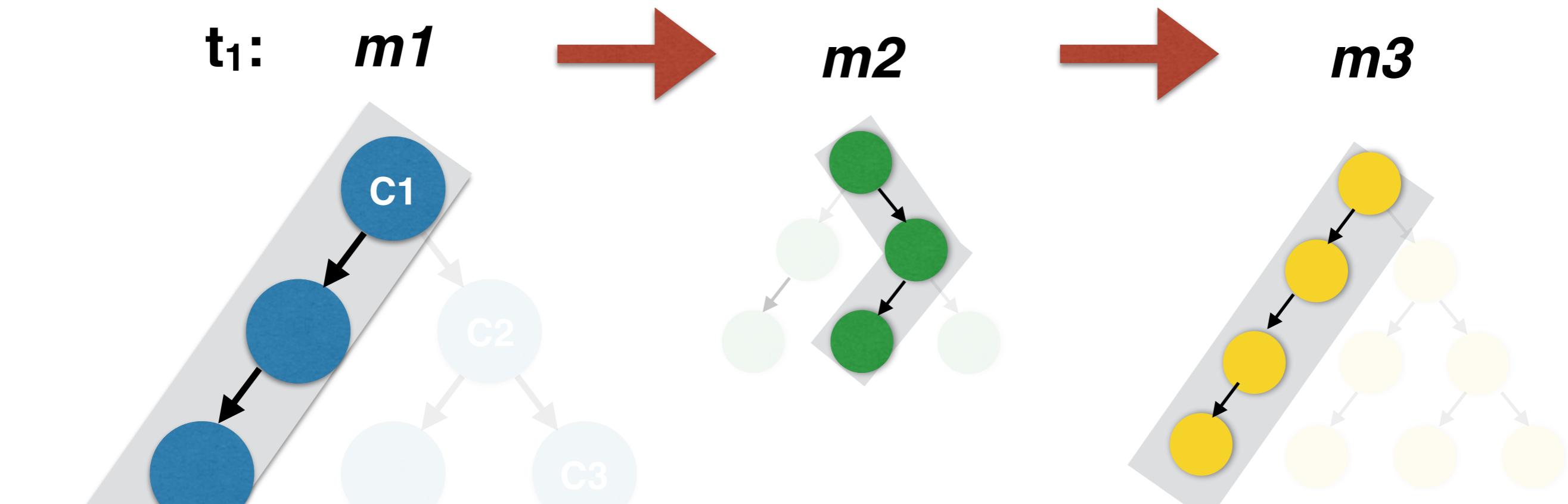
# Iteration - 1 : Concrete Execution



# Iteration - 1 : Concrete Execution

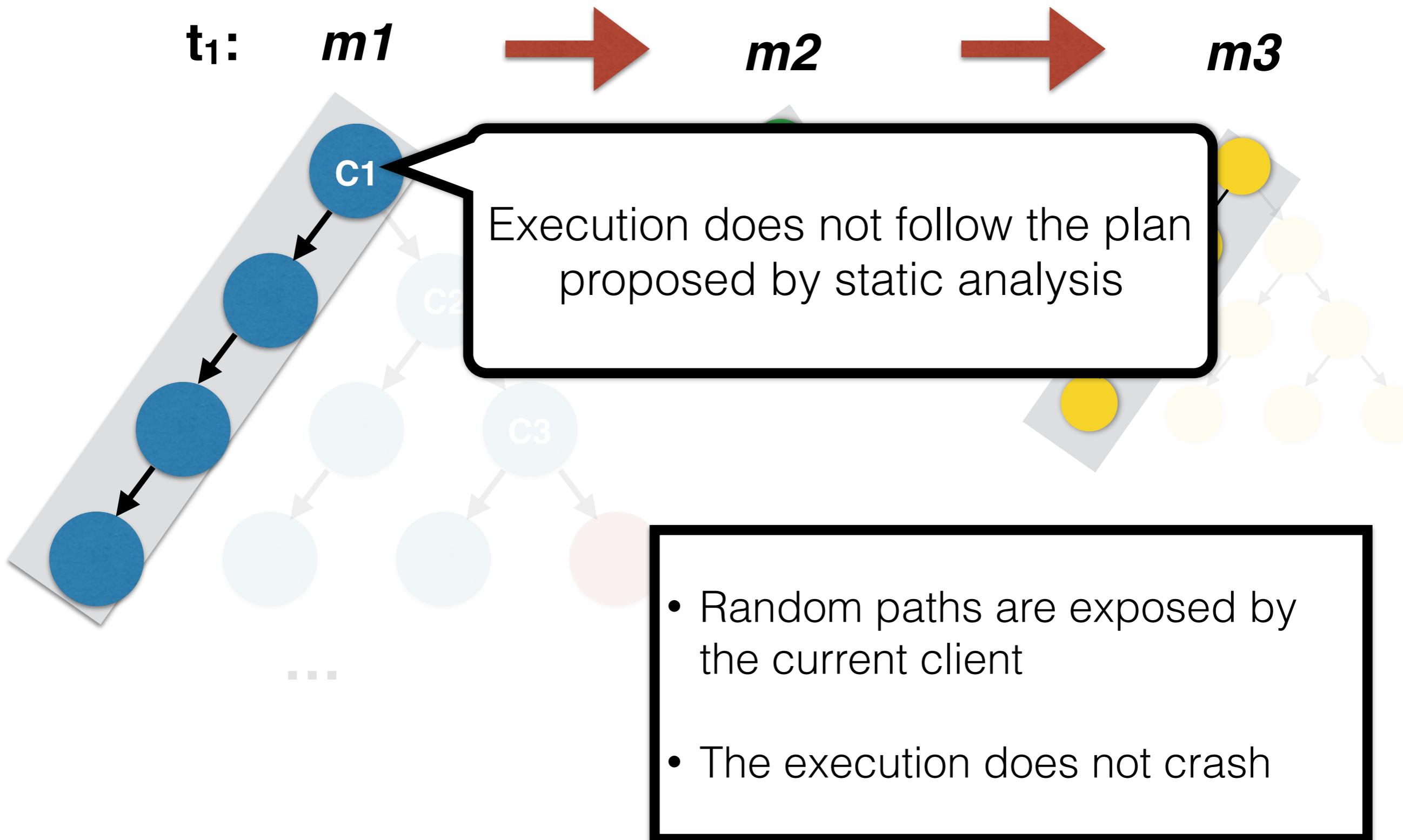


# Iteration - 1 : Concrete Execution

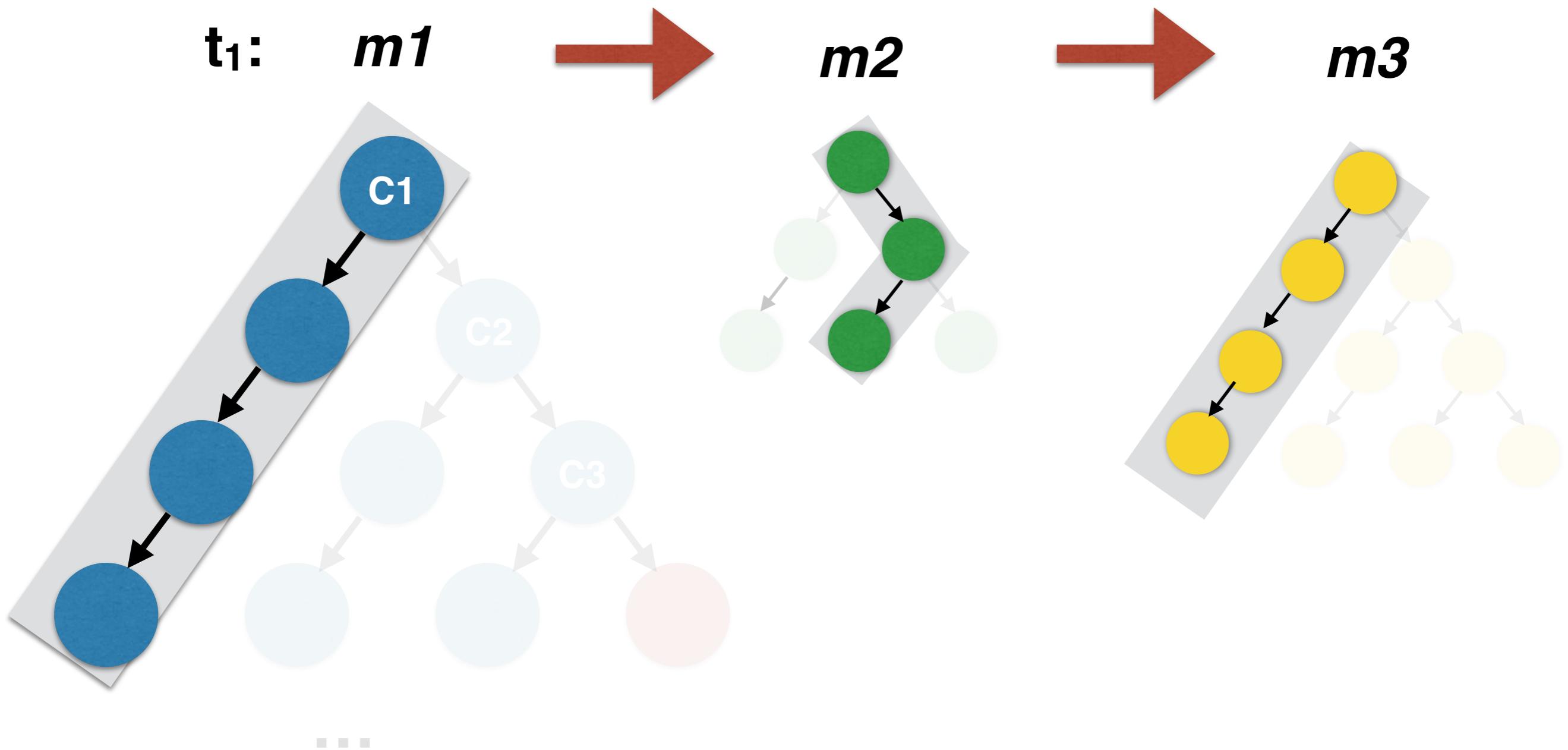


- Random paths are exposed by the current client
- The execution does not crash

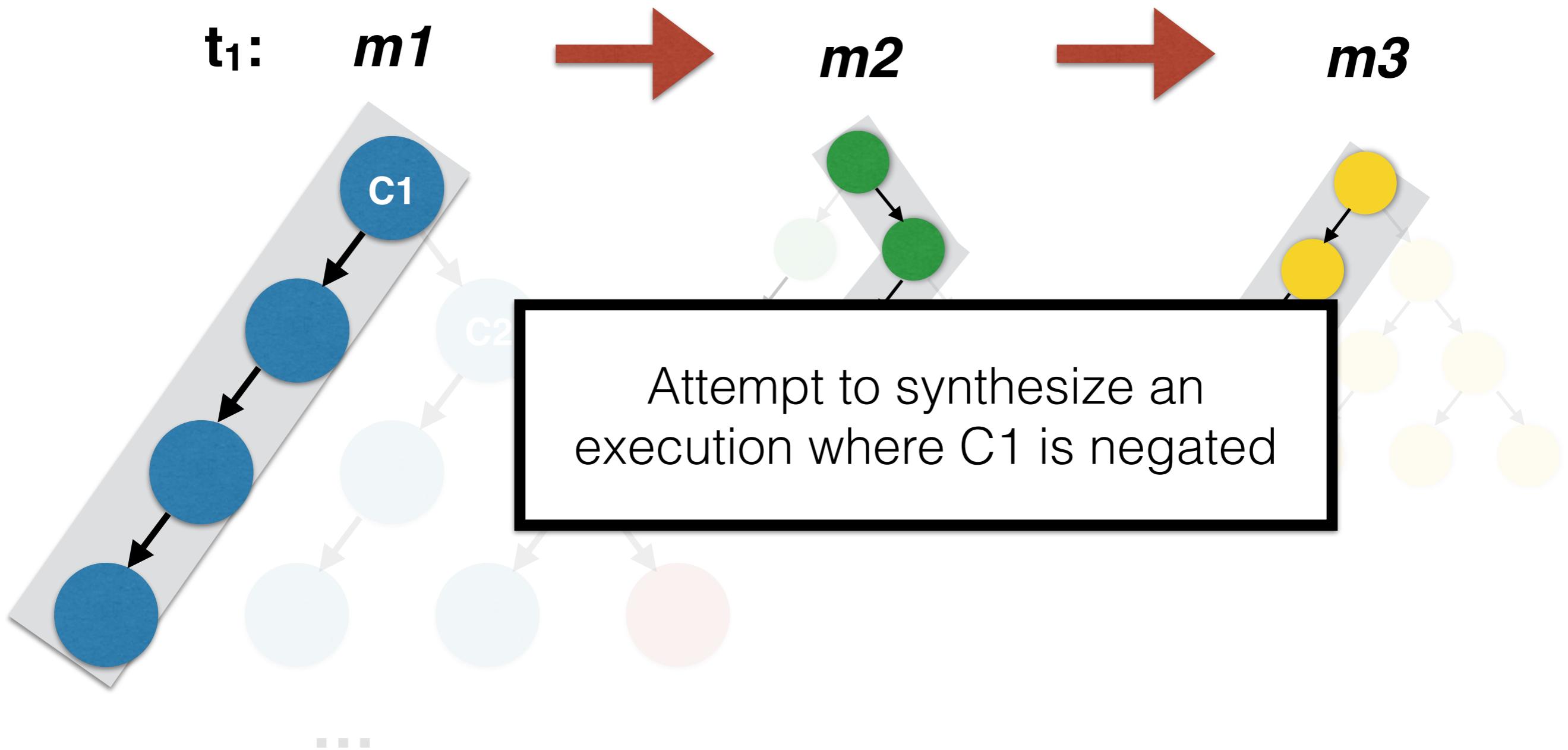
# Iteration - 1 : Concrete Execution



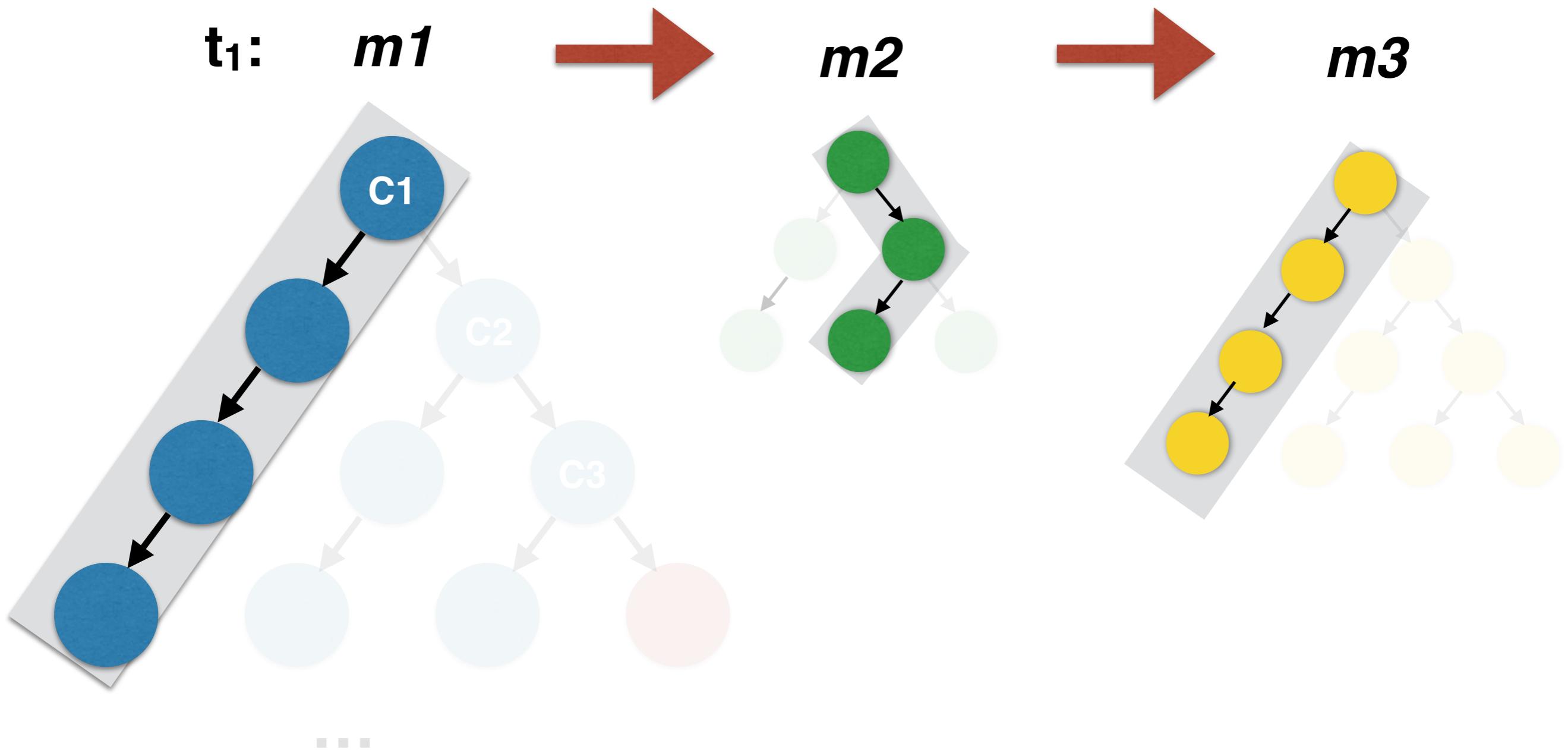
# Iteration - 1 : Concrete Execution



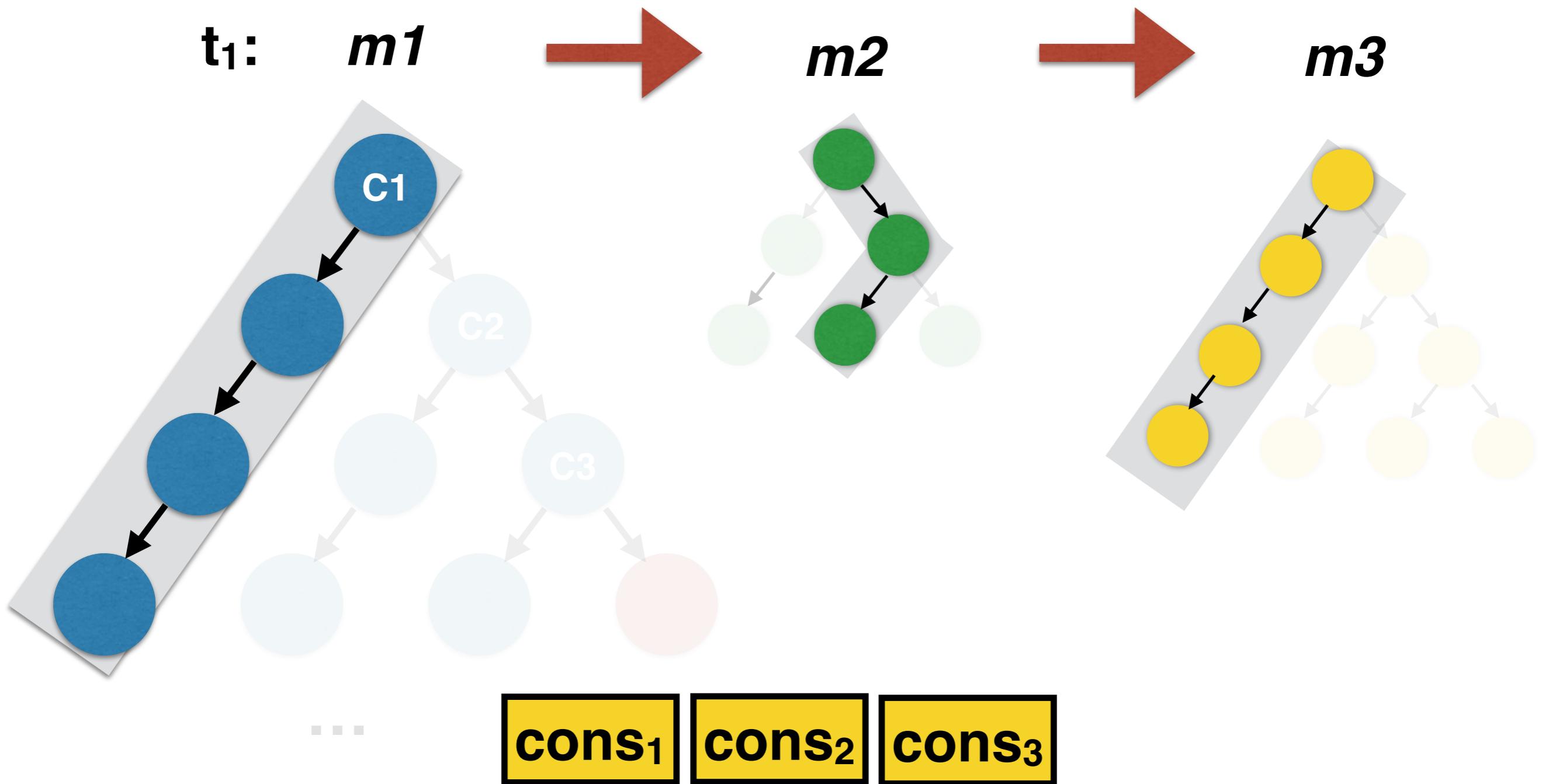
# Iteration - 1 : Concrete Execution



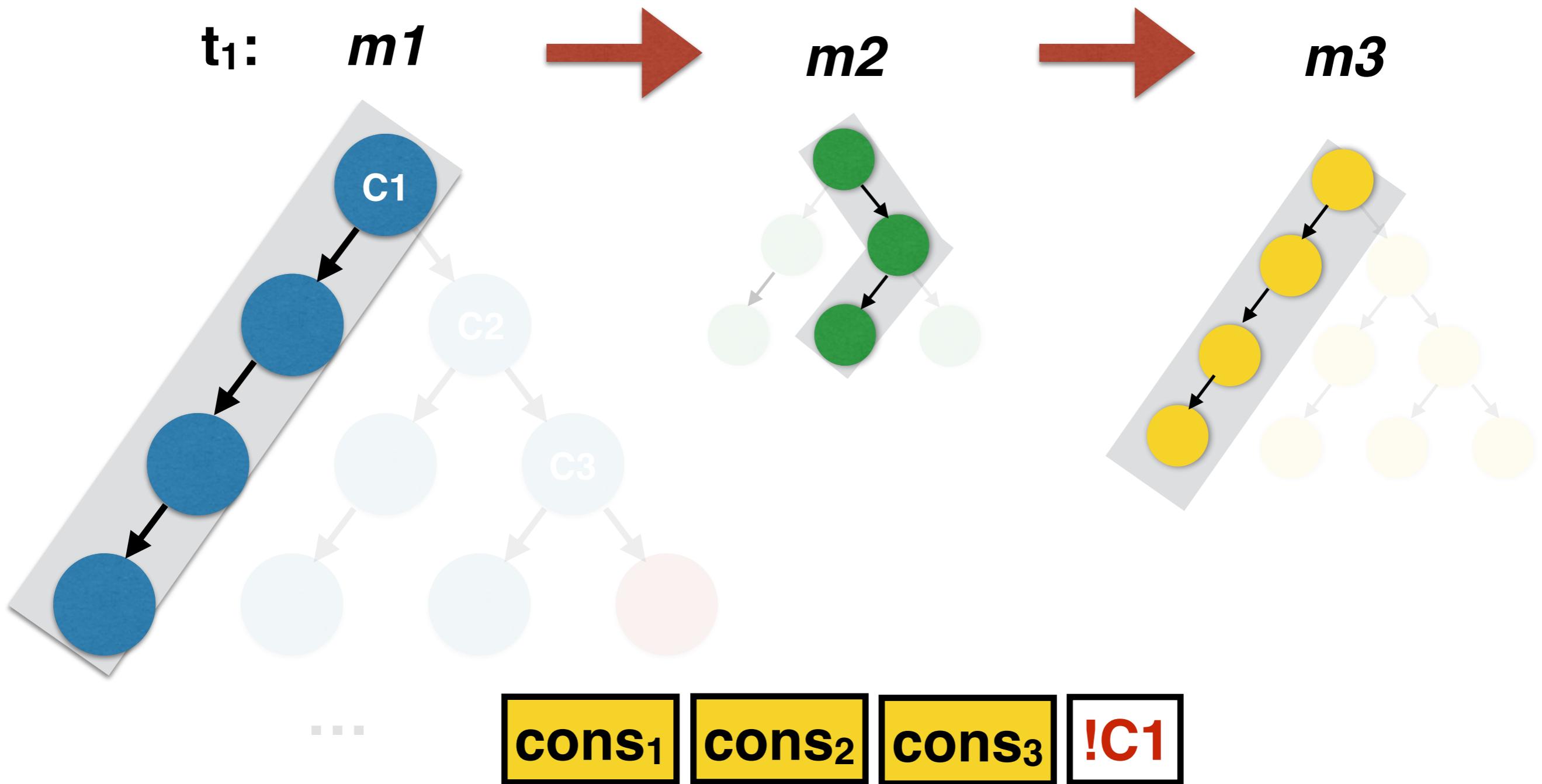
# Iteration - 1 : Concrete Execution



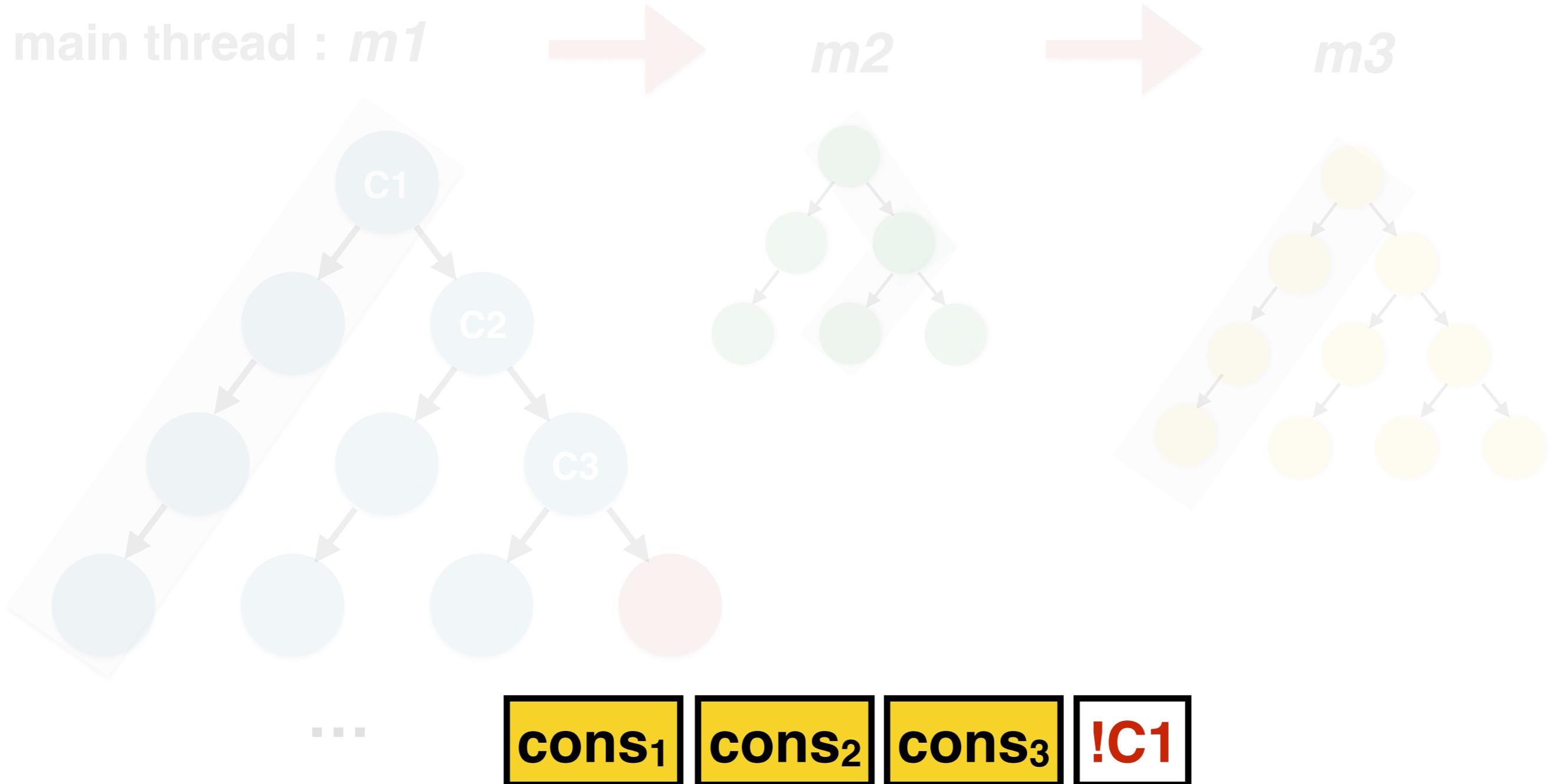
# Iteration - 1 : Concrete Execution



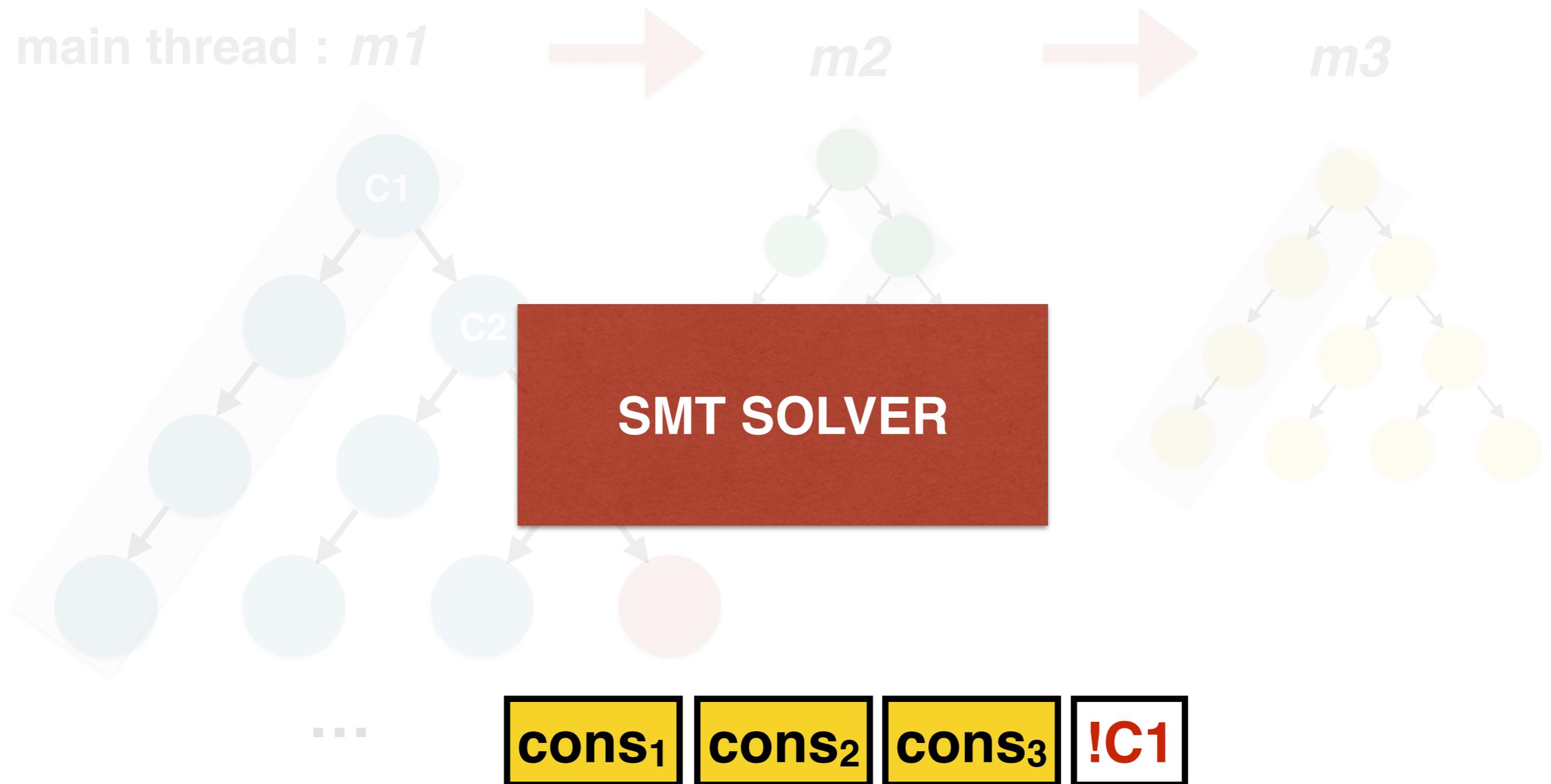
# Iteration - 1 : Concrete Execution



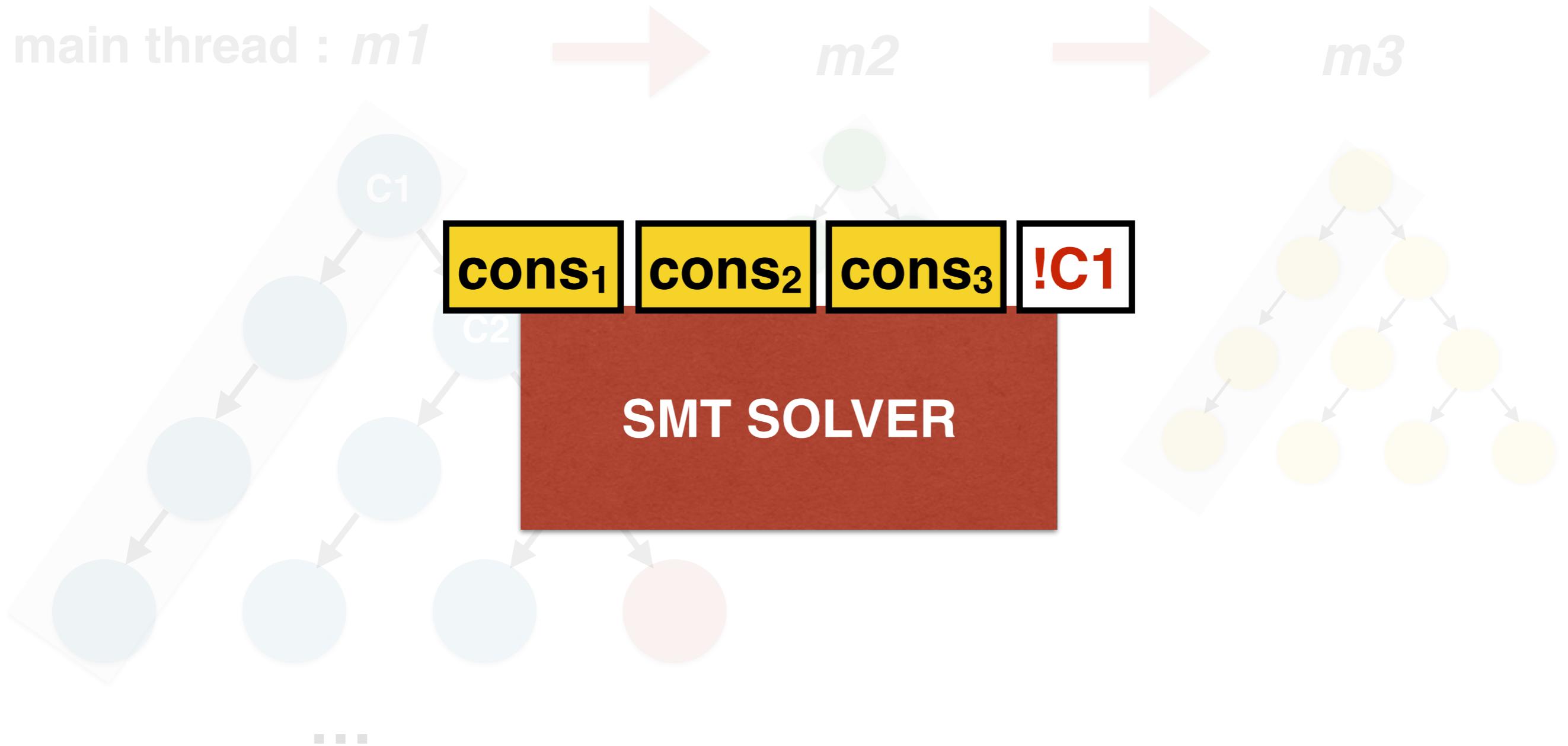
# Iteration - 1 : Client Synthesis



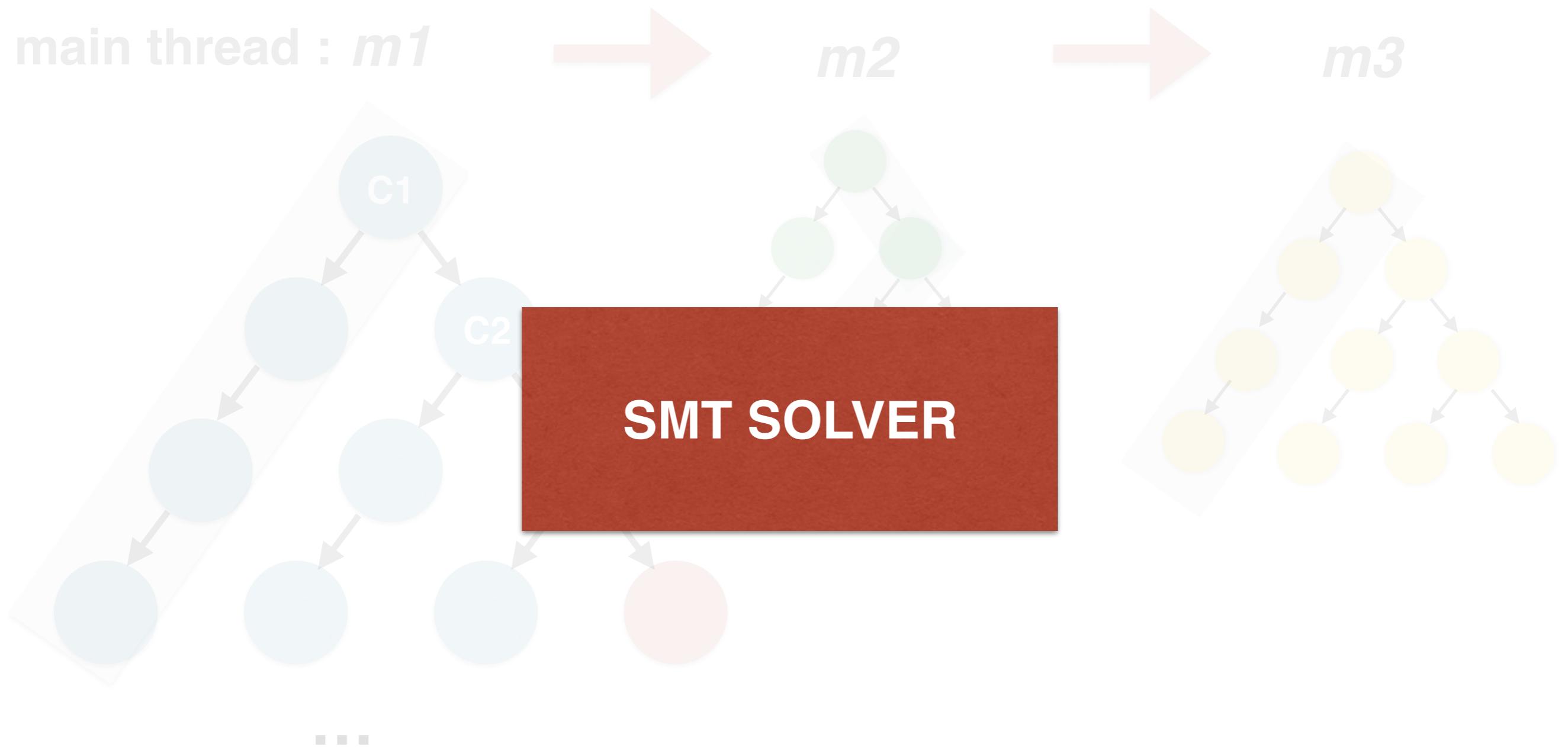
# Iteration - 1 : Client Synthesis



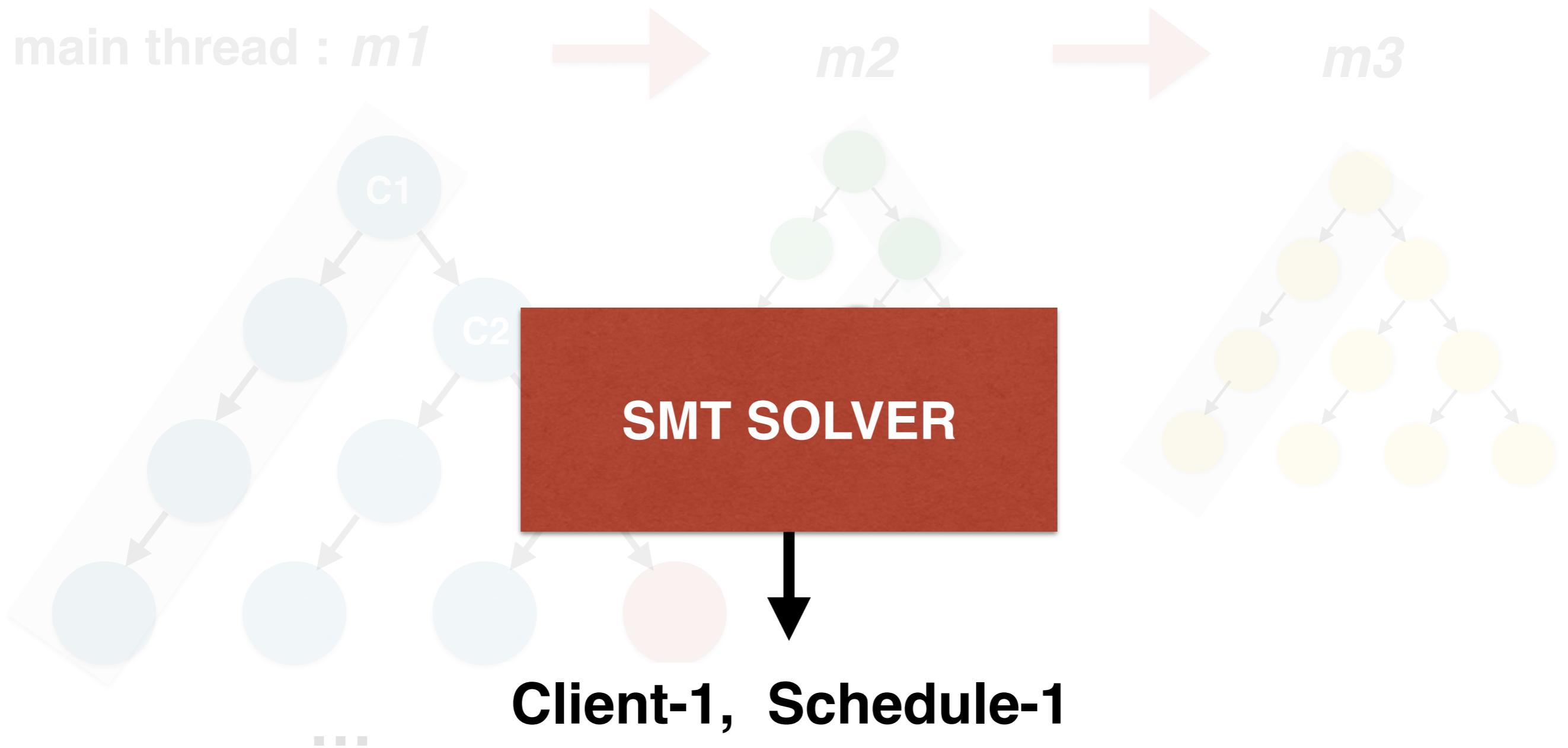
# Iteration - 1 : Client Synthesis



# Iteration - 1 : Client Synthesis



# Iteration - 1 : Client Synthesis

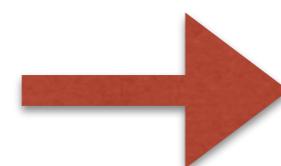


# Iteration - 2 : Concrete Execution

New  
client

$t_1 : m1$

$t_2 : m2$



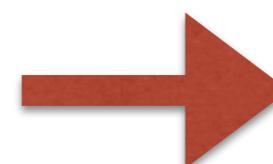
$m3$

# Iteration - 2: Concrete Execution

# New client

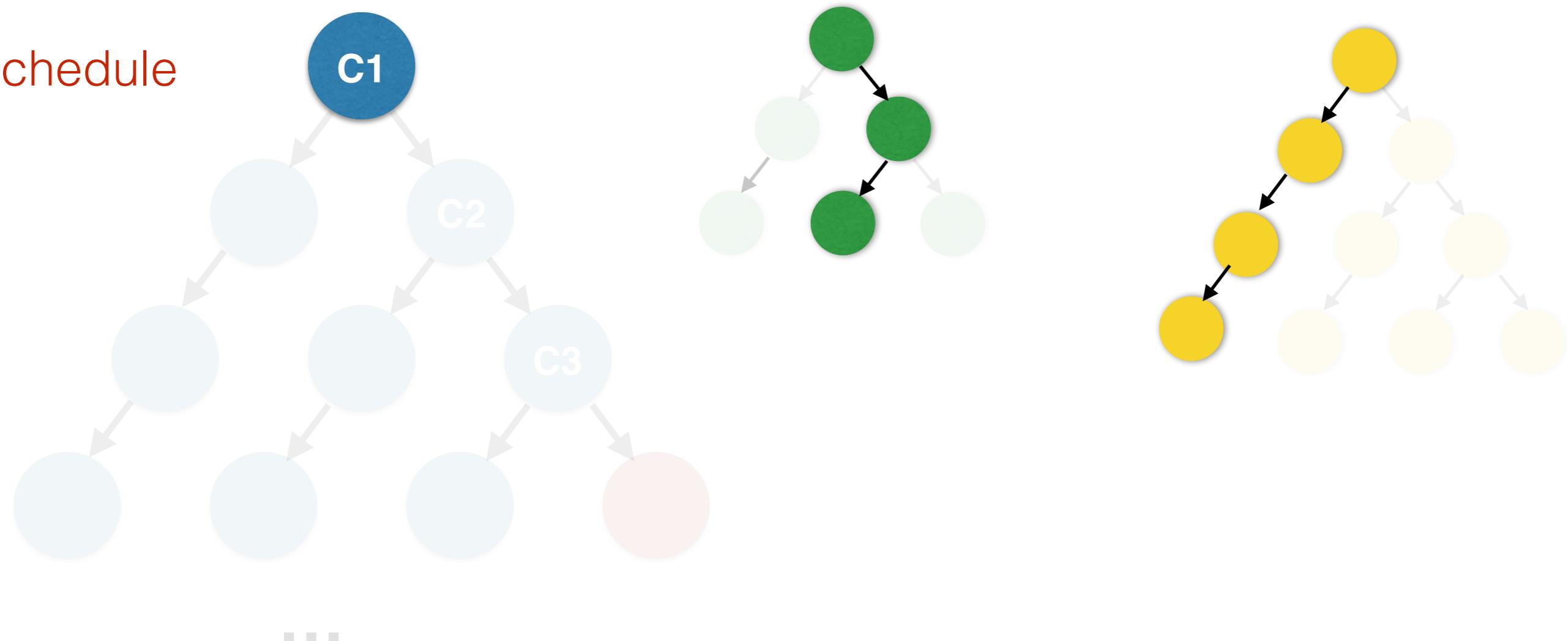
**t<sub>1</sub>** : *m1*

**t<sub>2</sub>** : *m2*

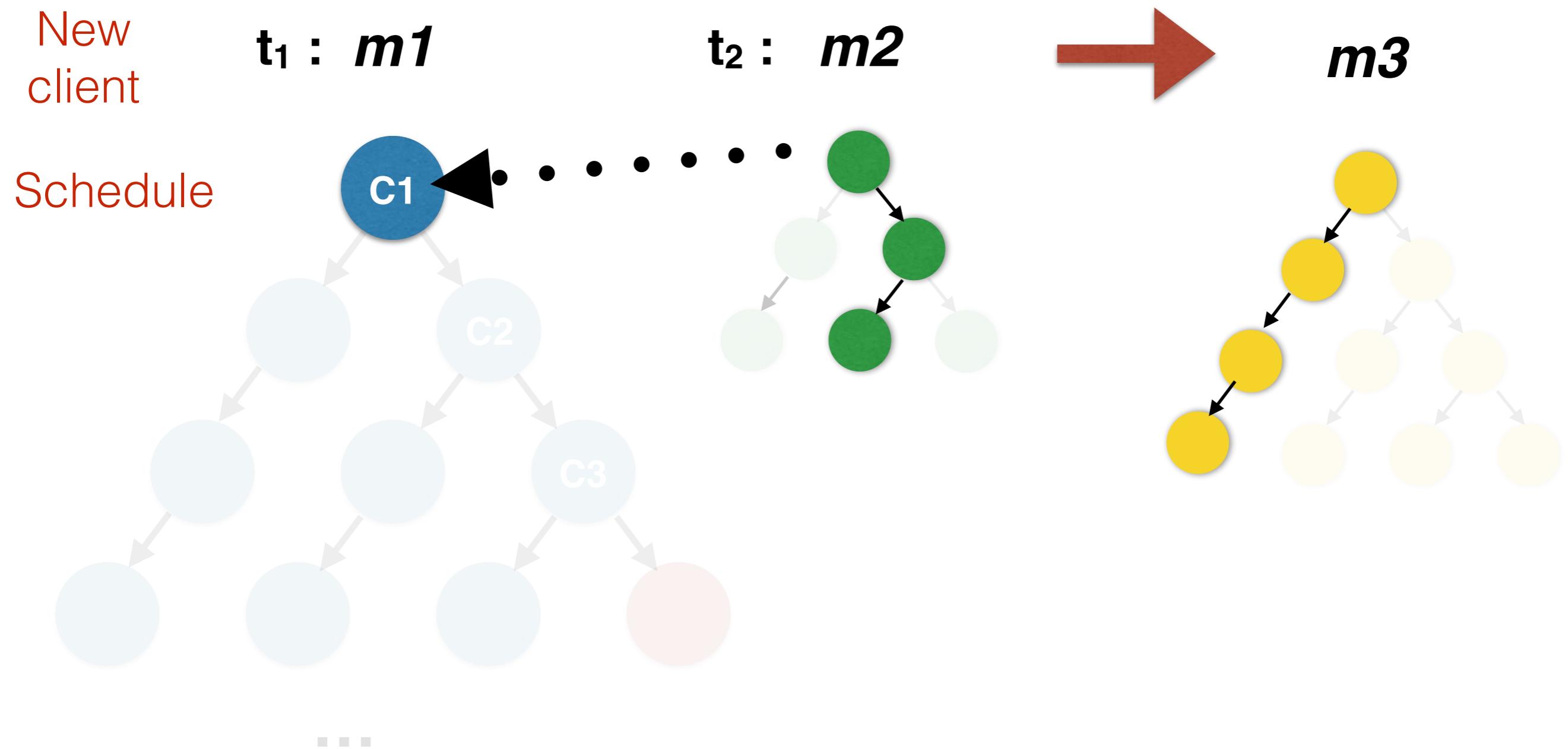


m3

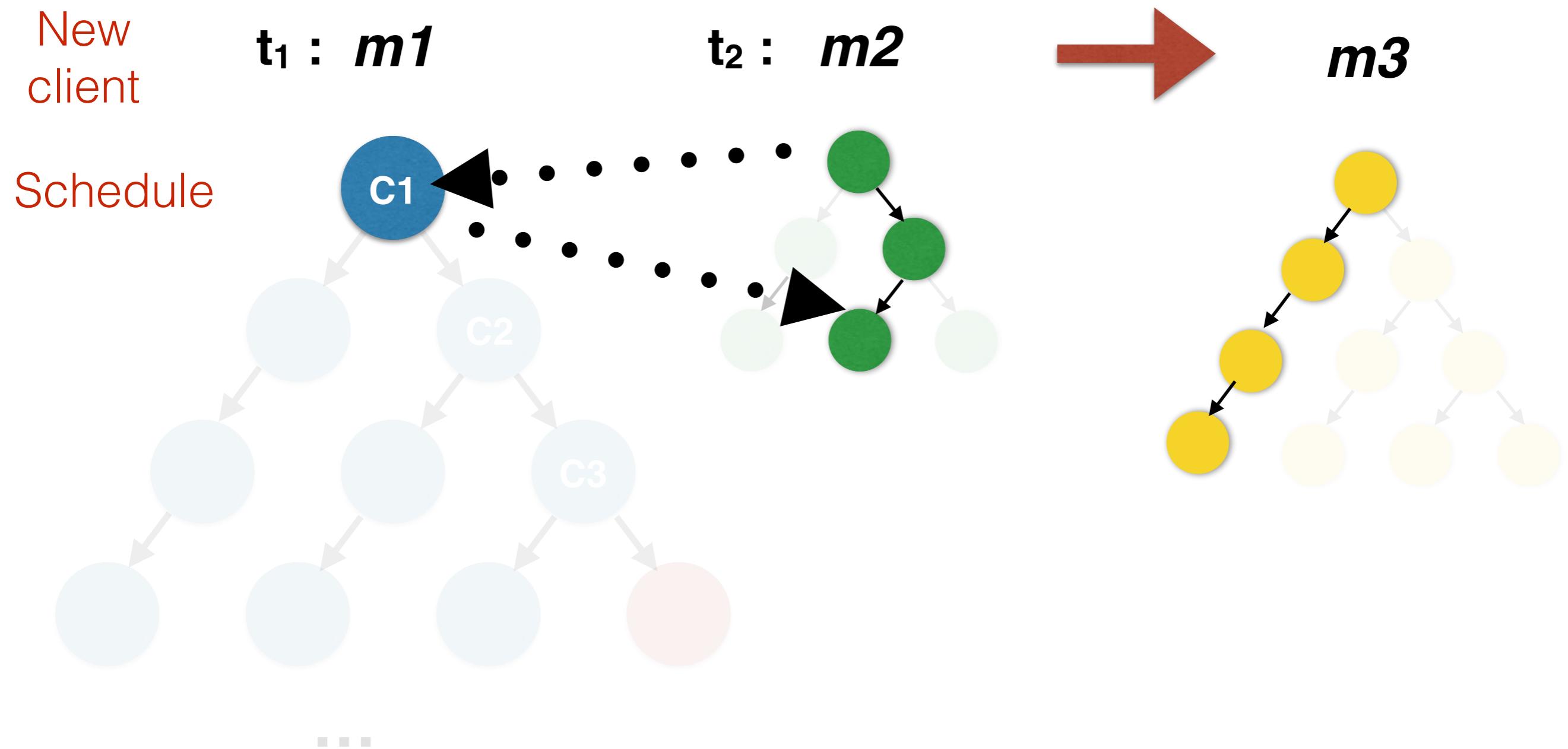
# Schedule



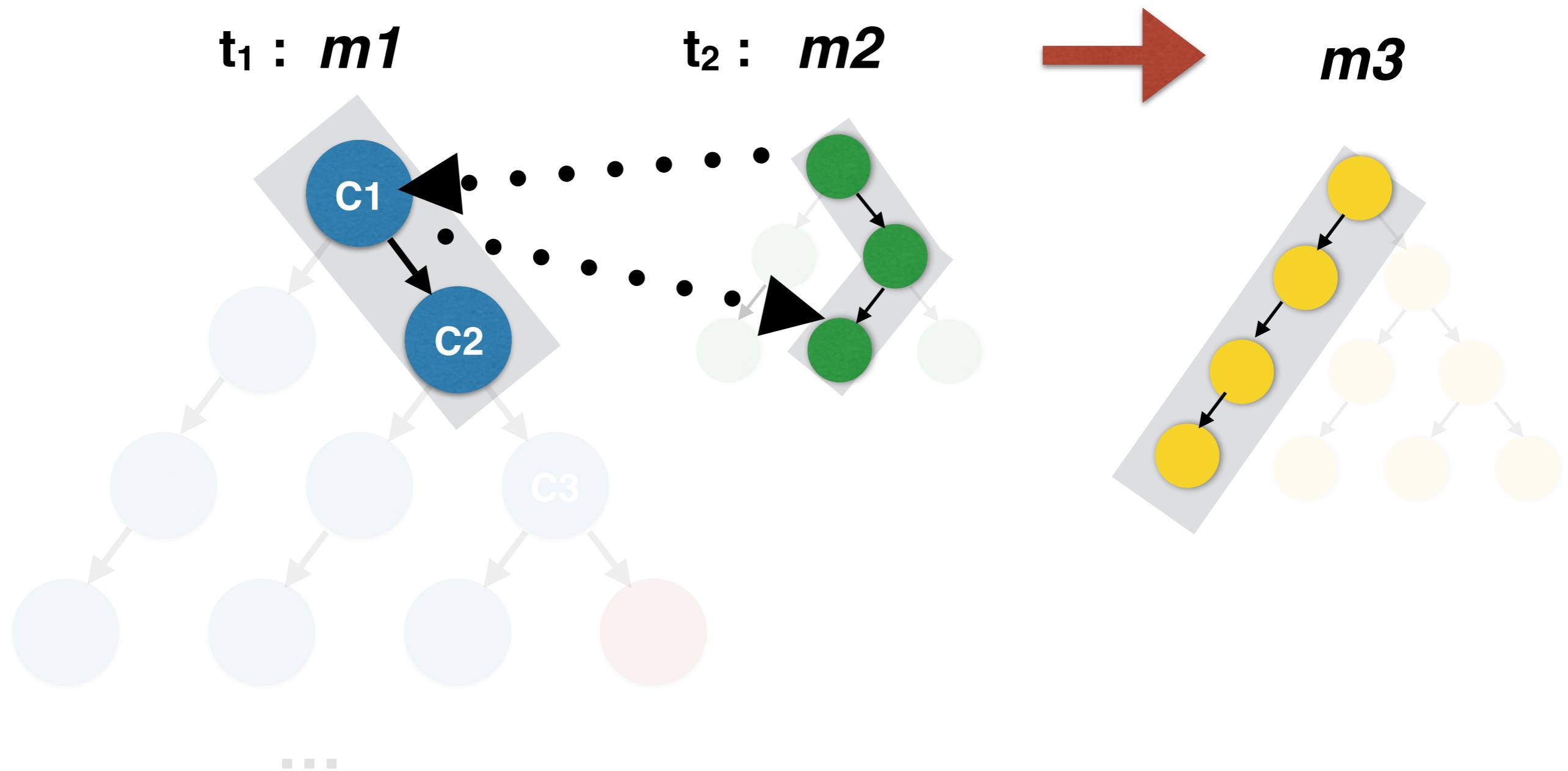
# Iteration - 2 : Concrete Execution



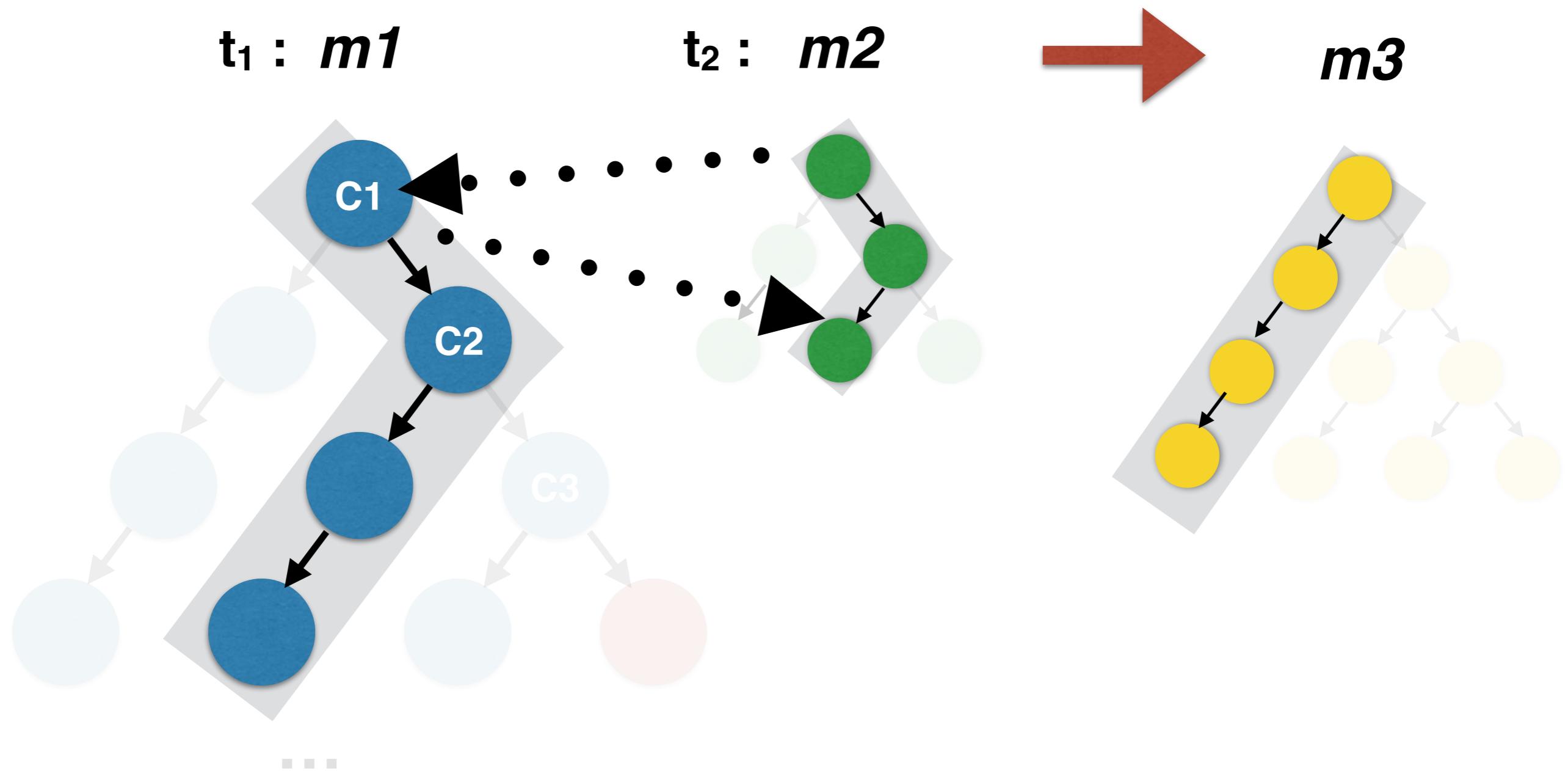
# Iteration - 2 : Concrete Execution



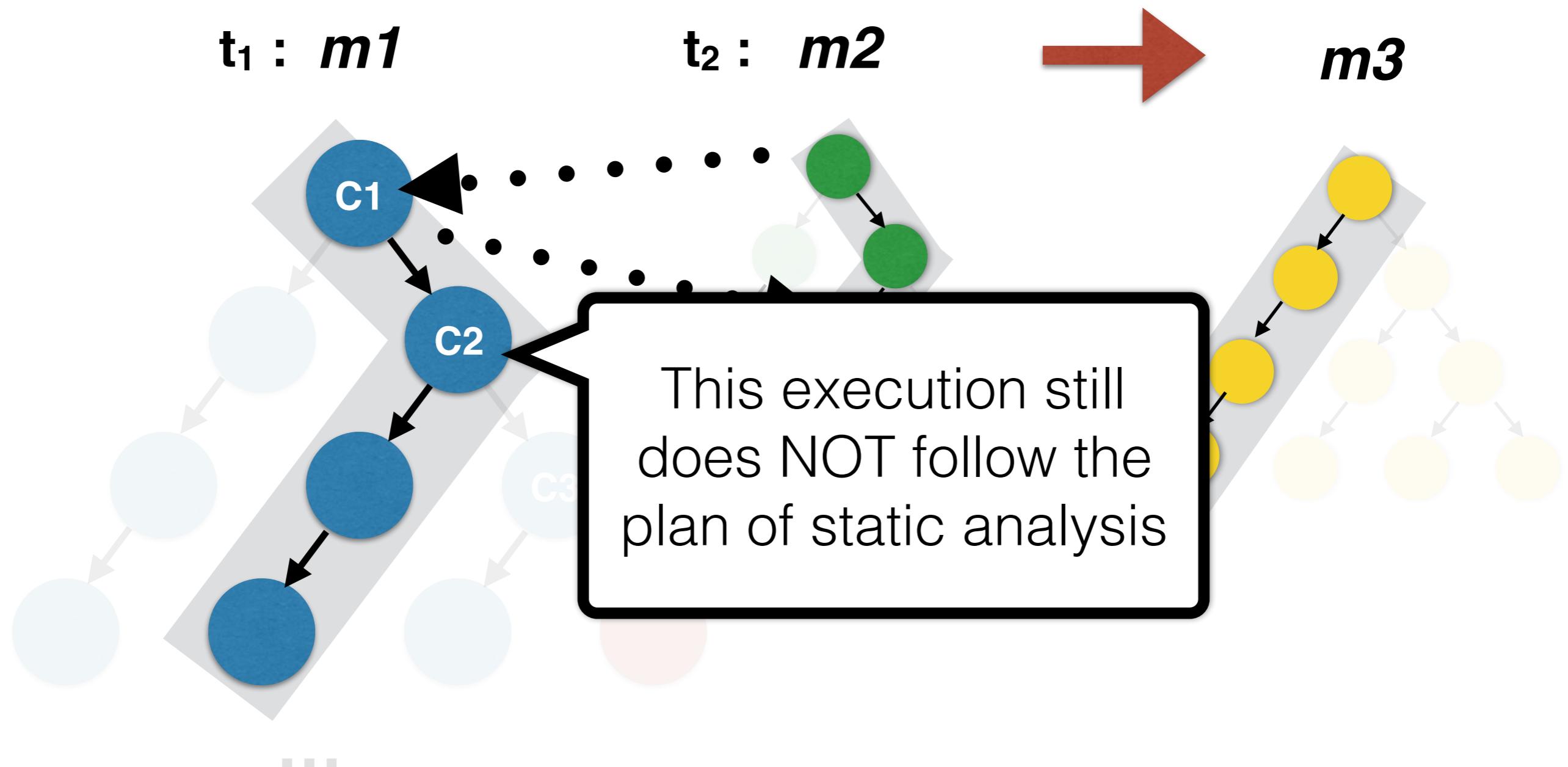
# Iteration - 2 : Concrete Execution



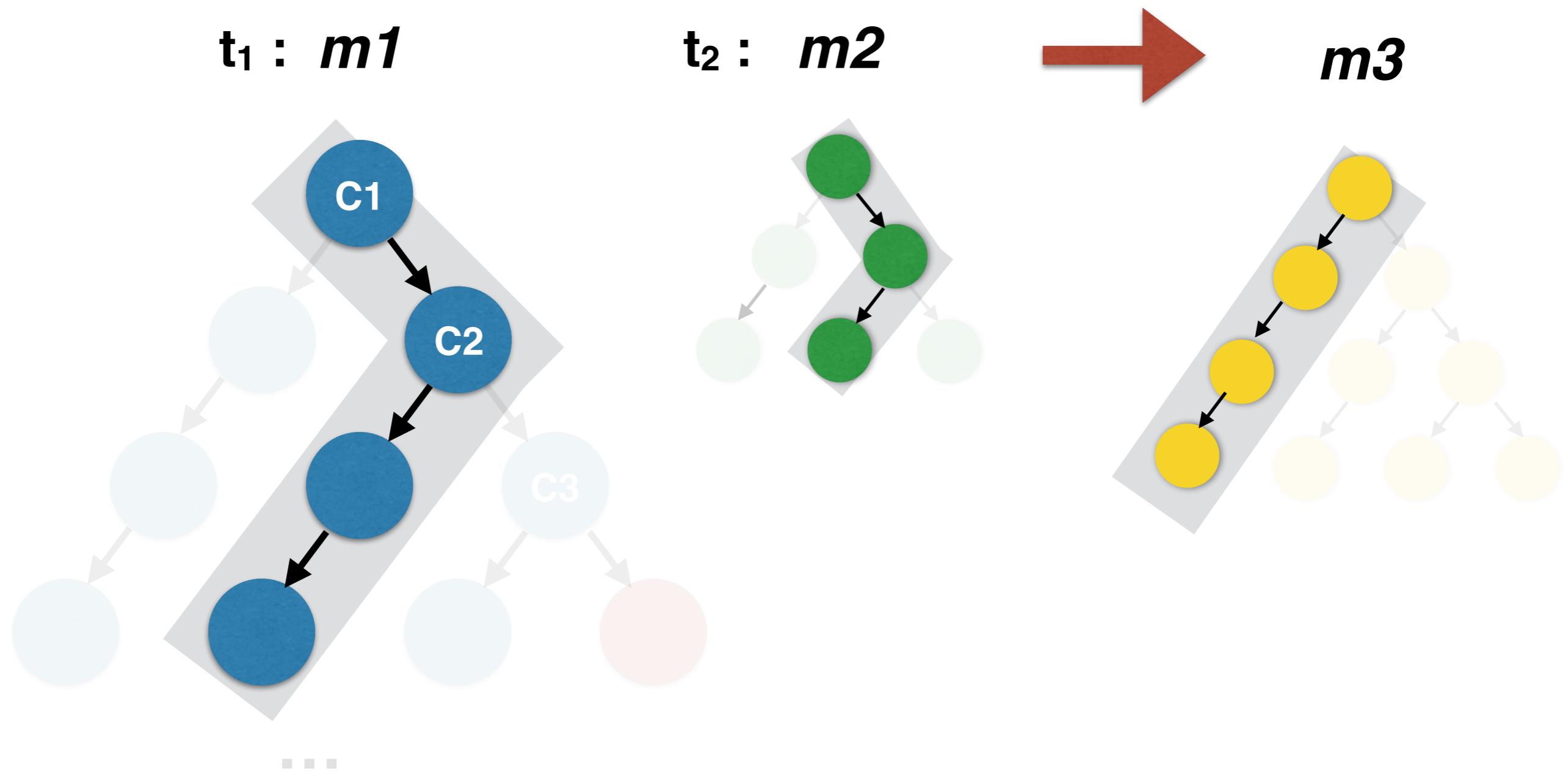
# Iteration - 2 : Concrete Execution



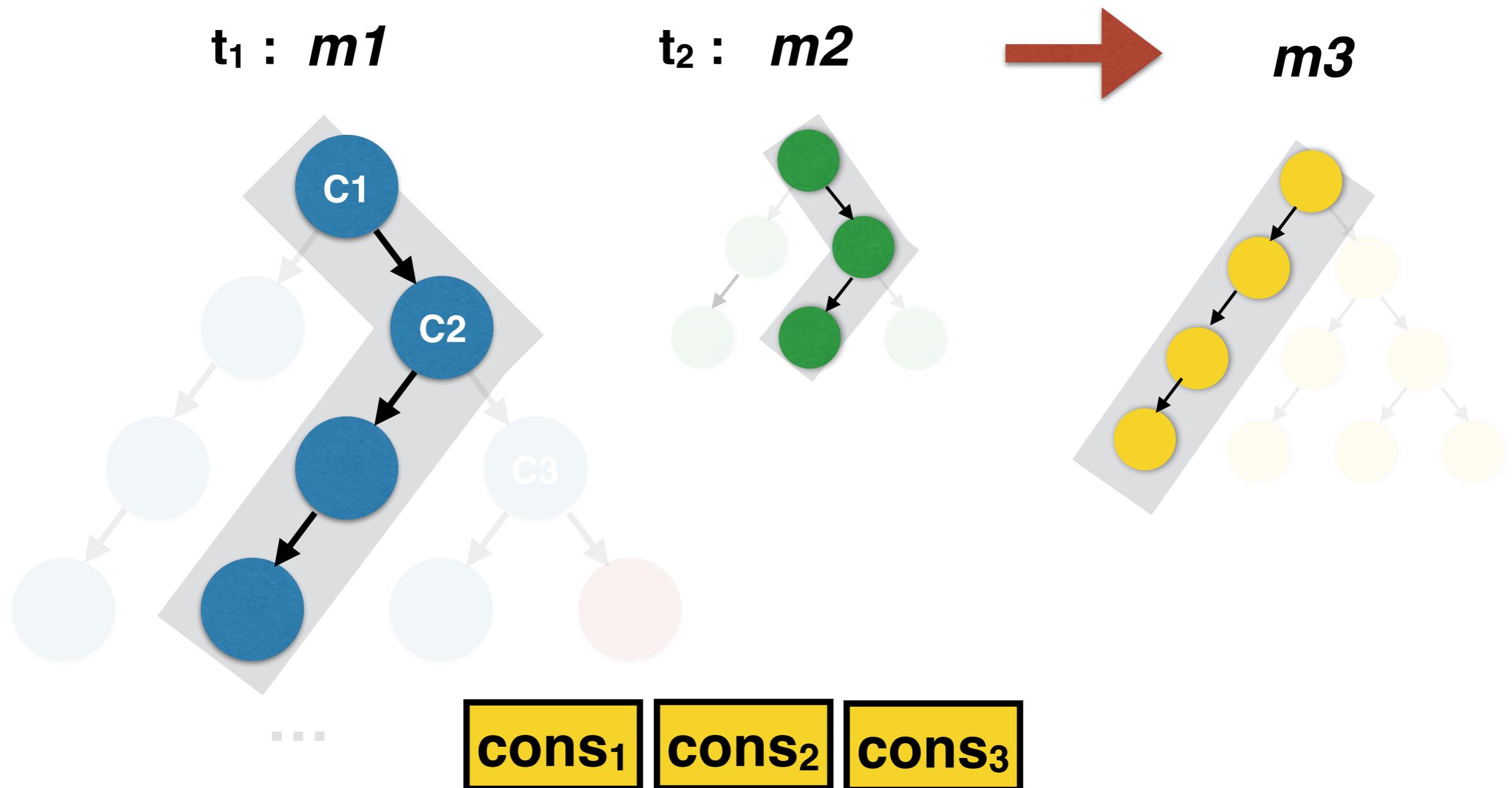
# Iteration - 2 : Concrete Execution



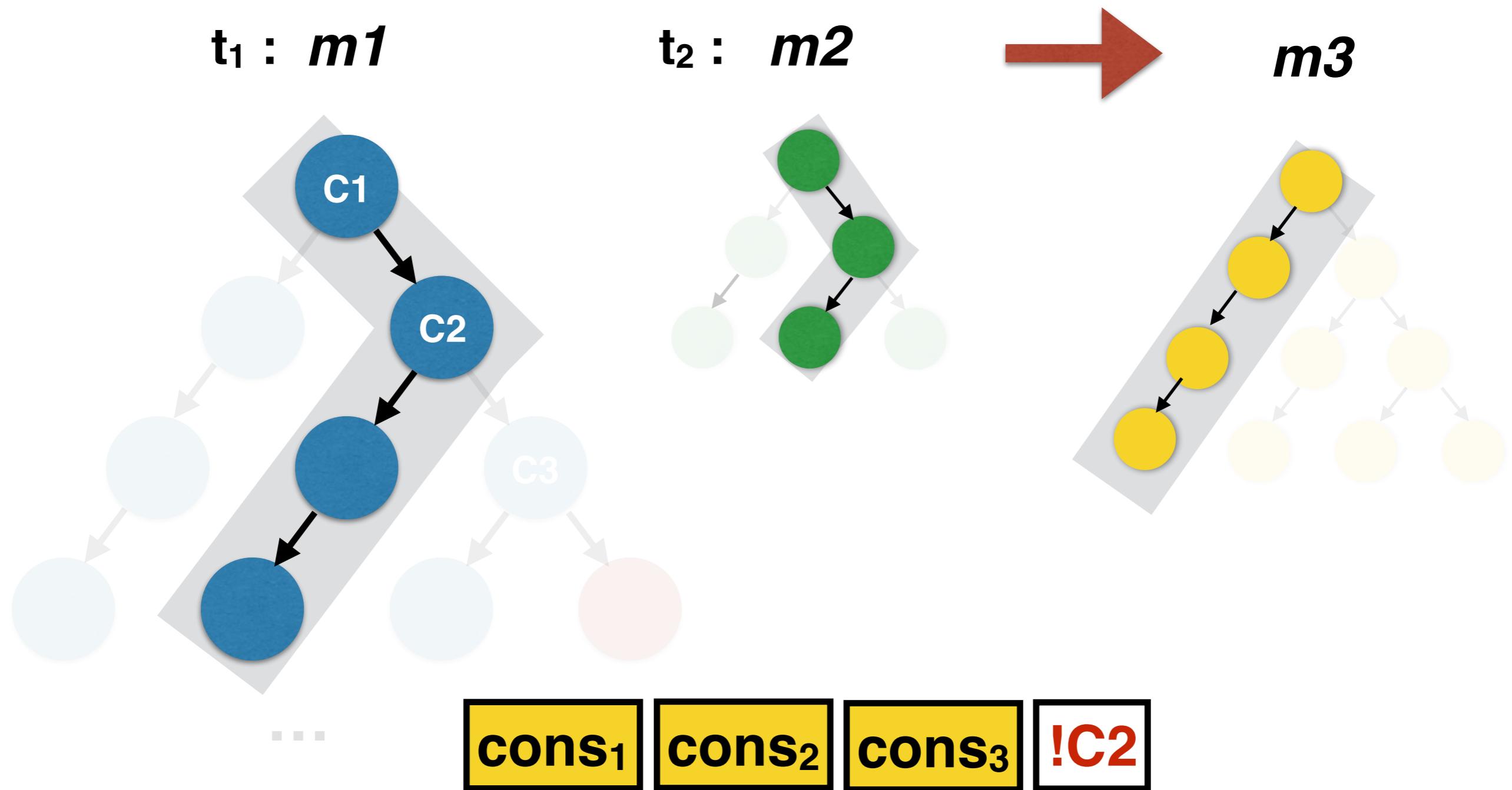
# Iteration - 2 : Concrete Execution



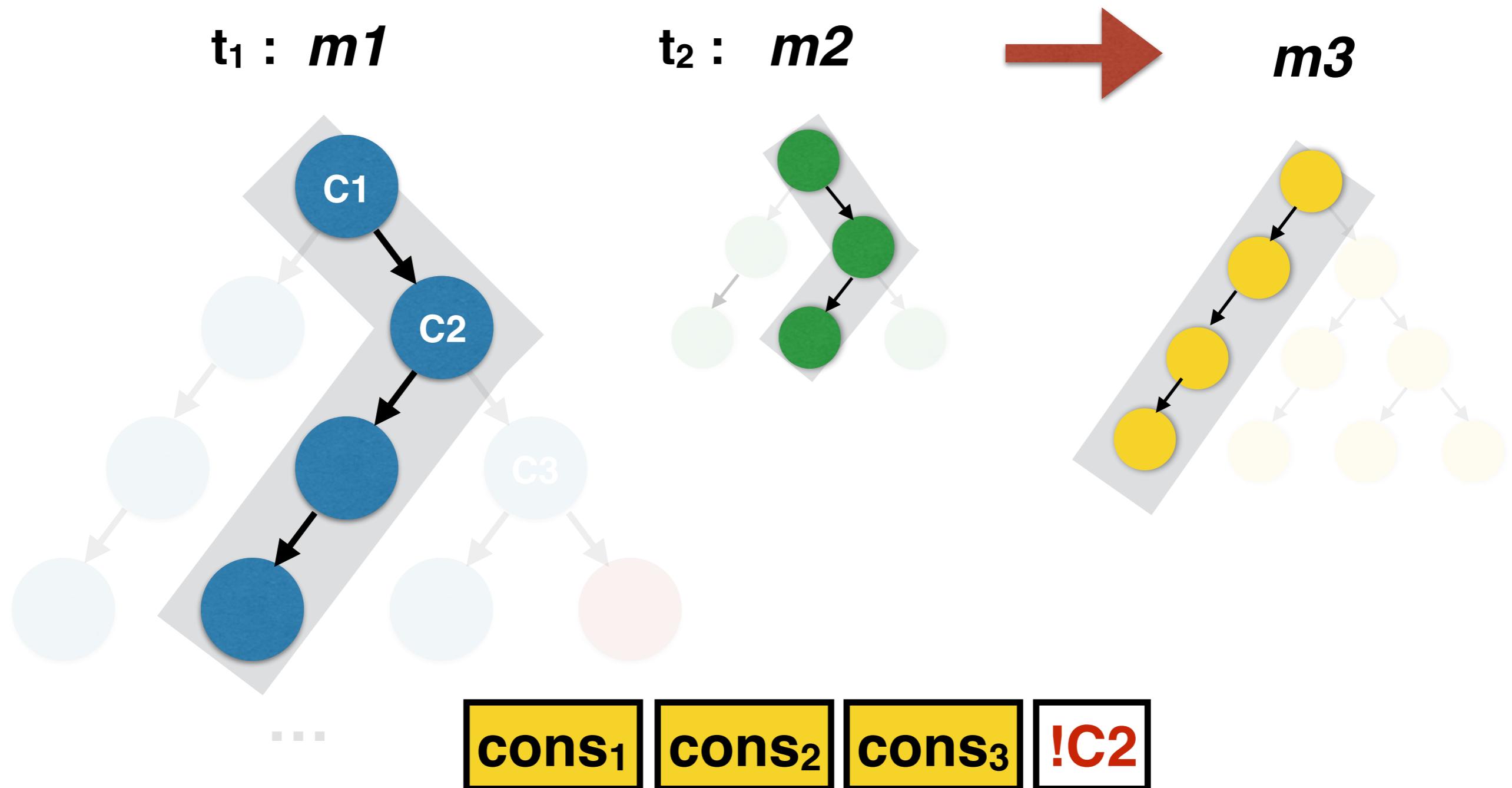
# Iteration - 2 : Concrete Execution



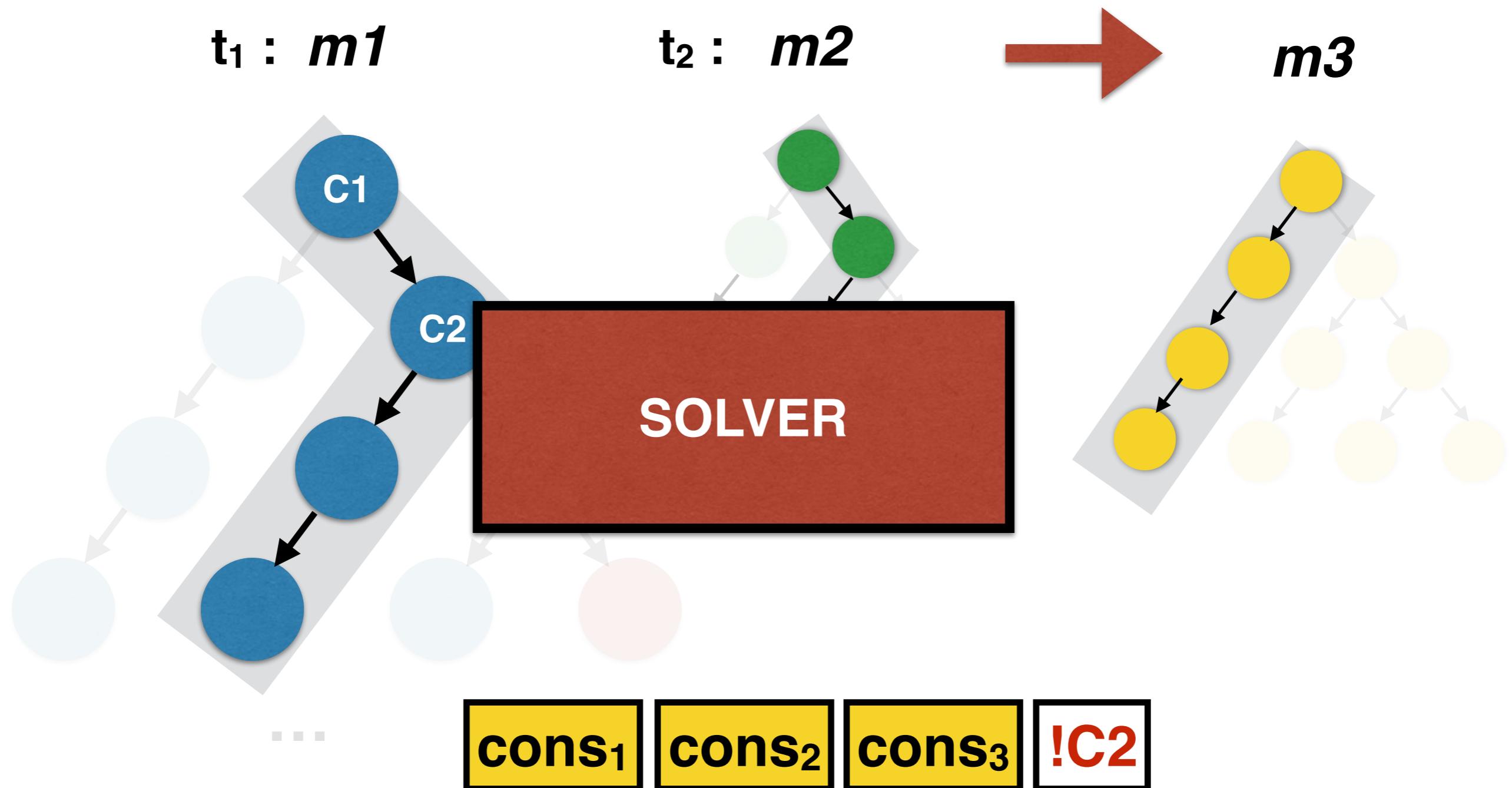
# Iteration - 2 : Concrete Execution



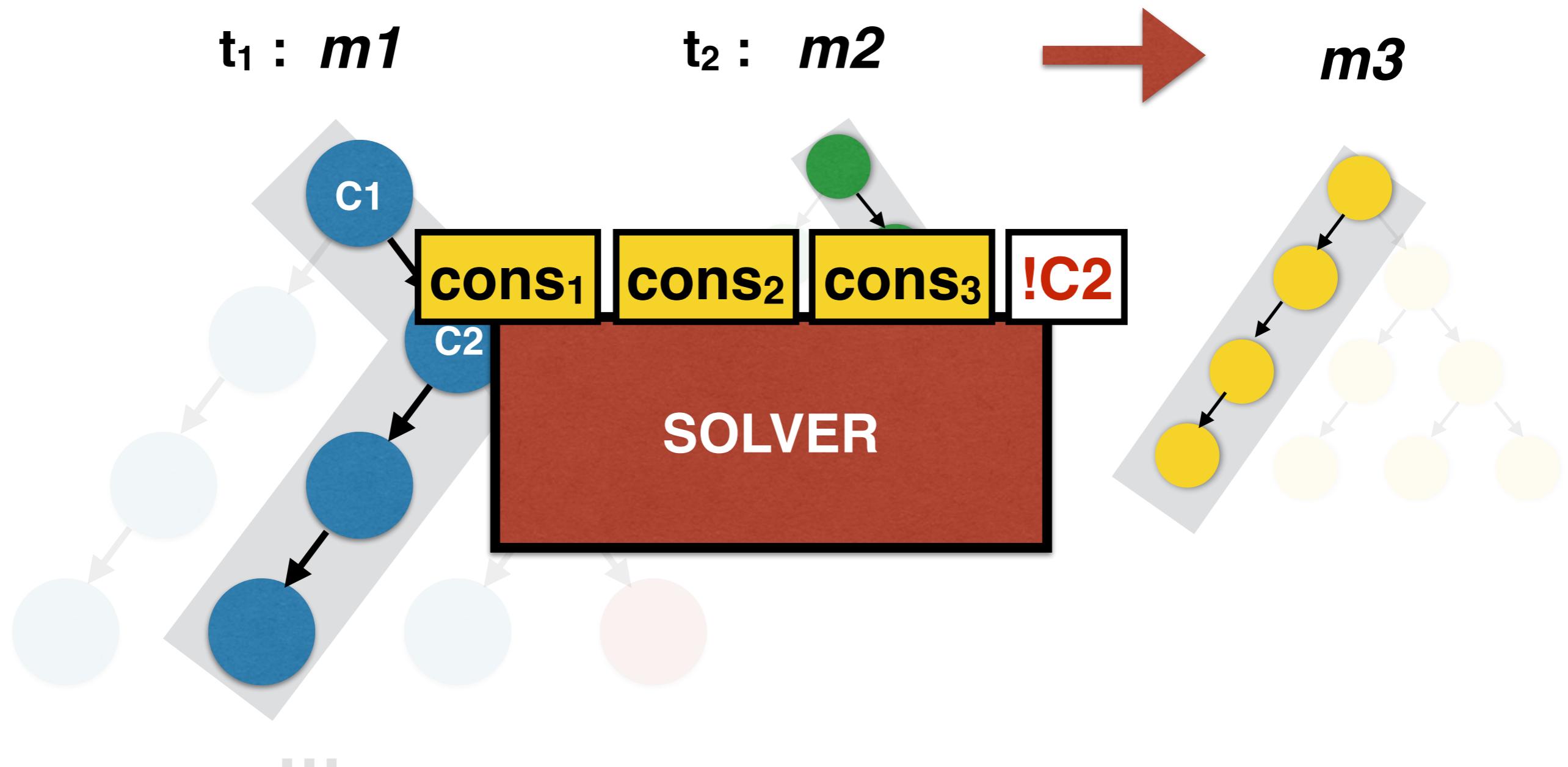
# Iteration - 2 : Concrete Execution



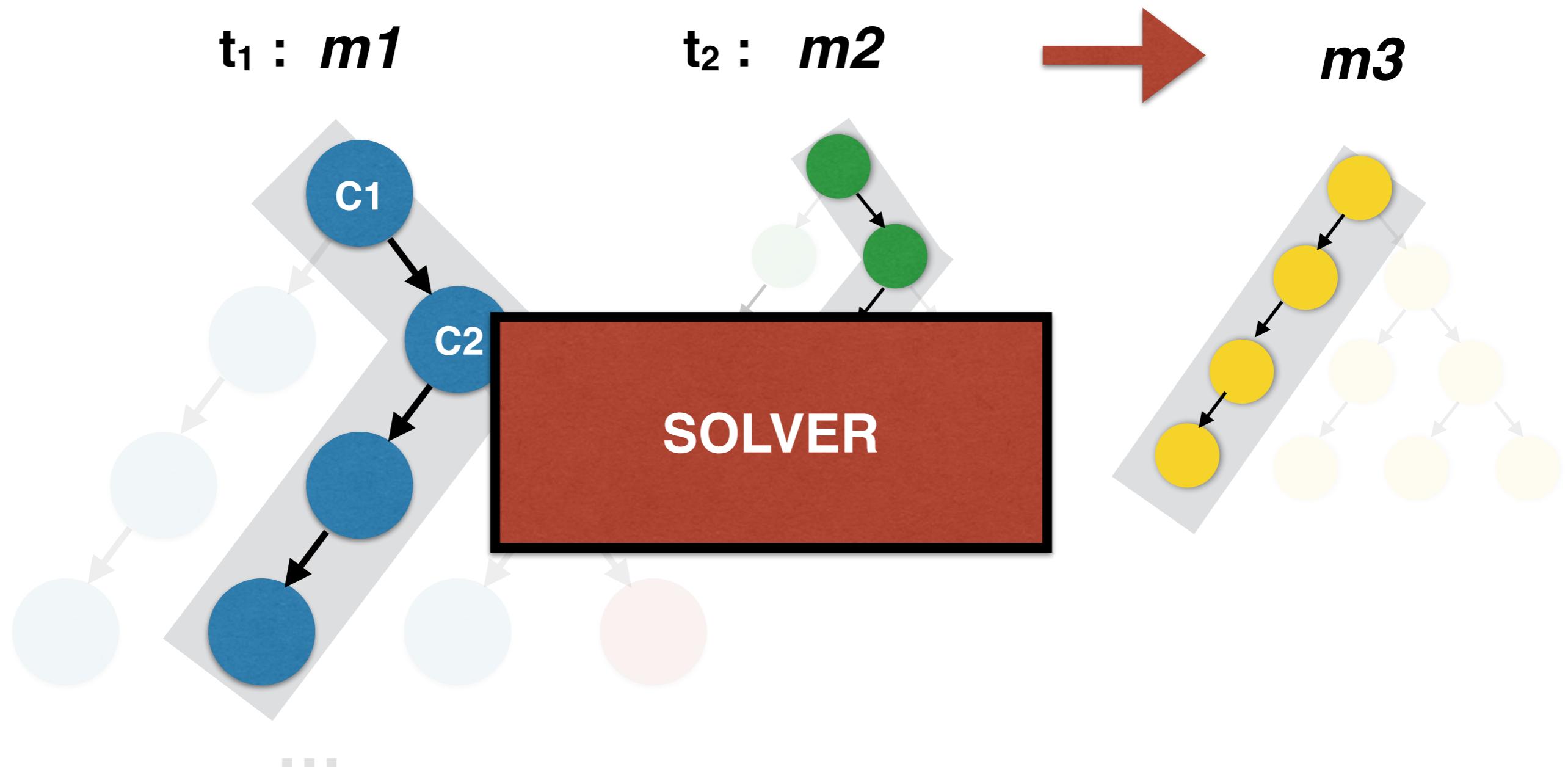
# Iteration - 2 : Concrete Execution



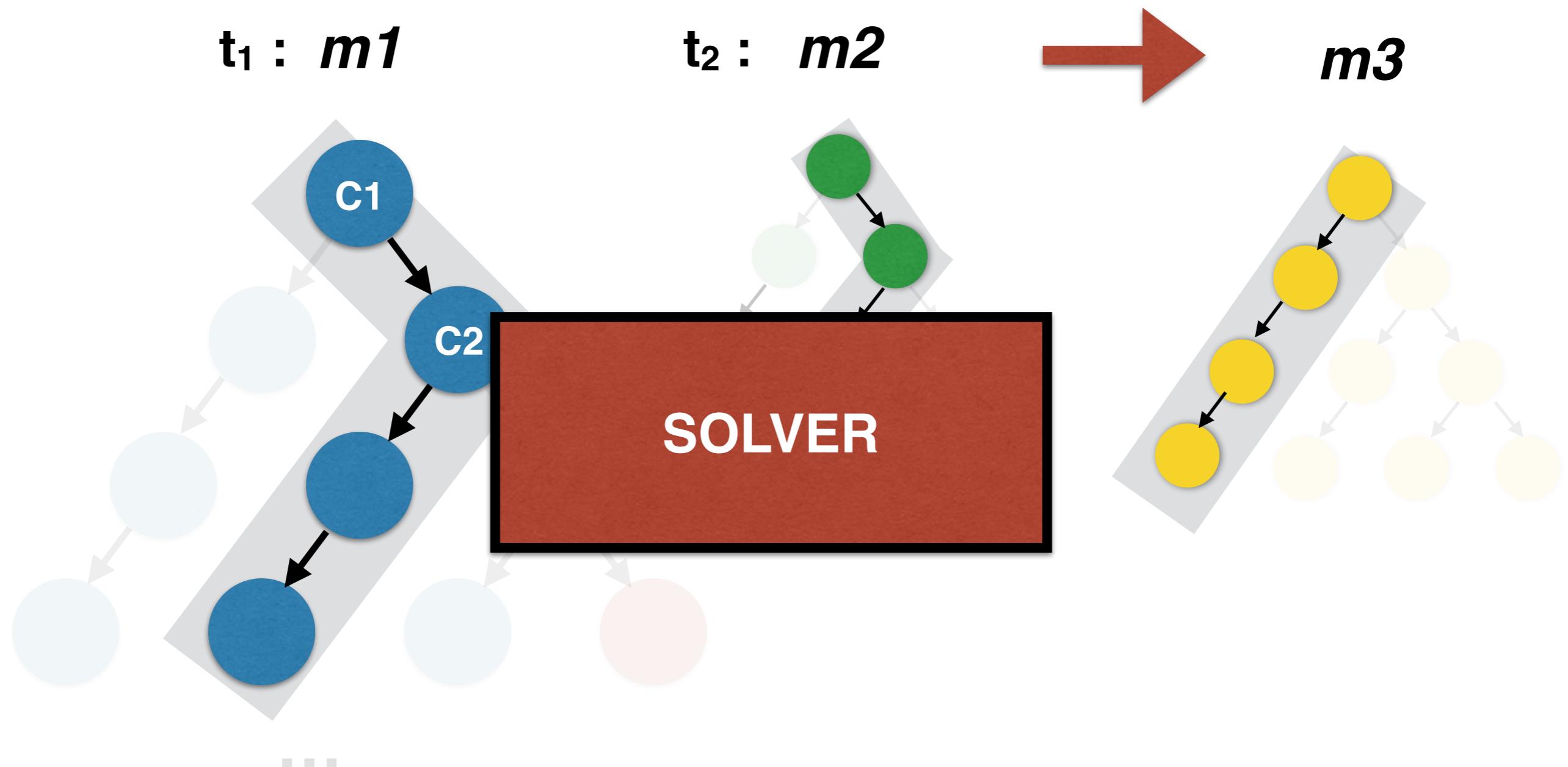
# Iteration - 2 : Concrete Execution



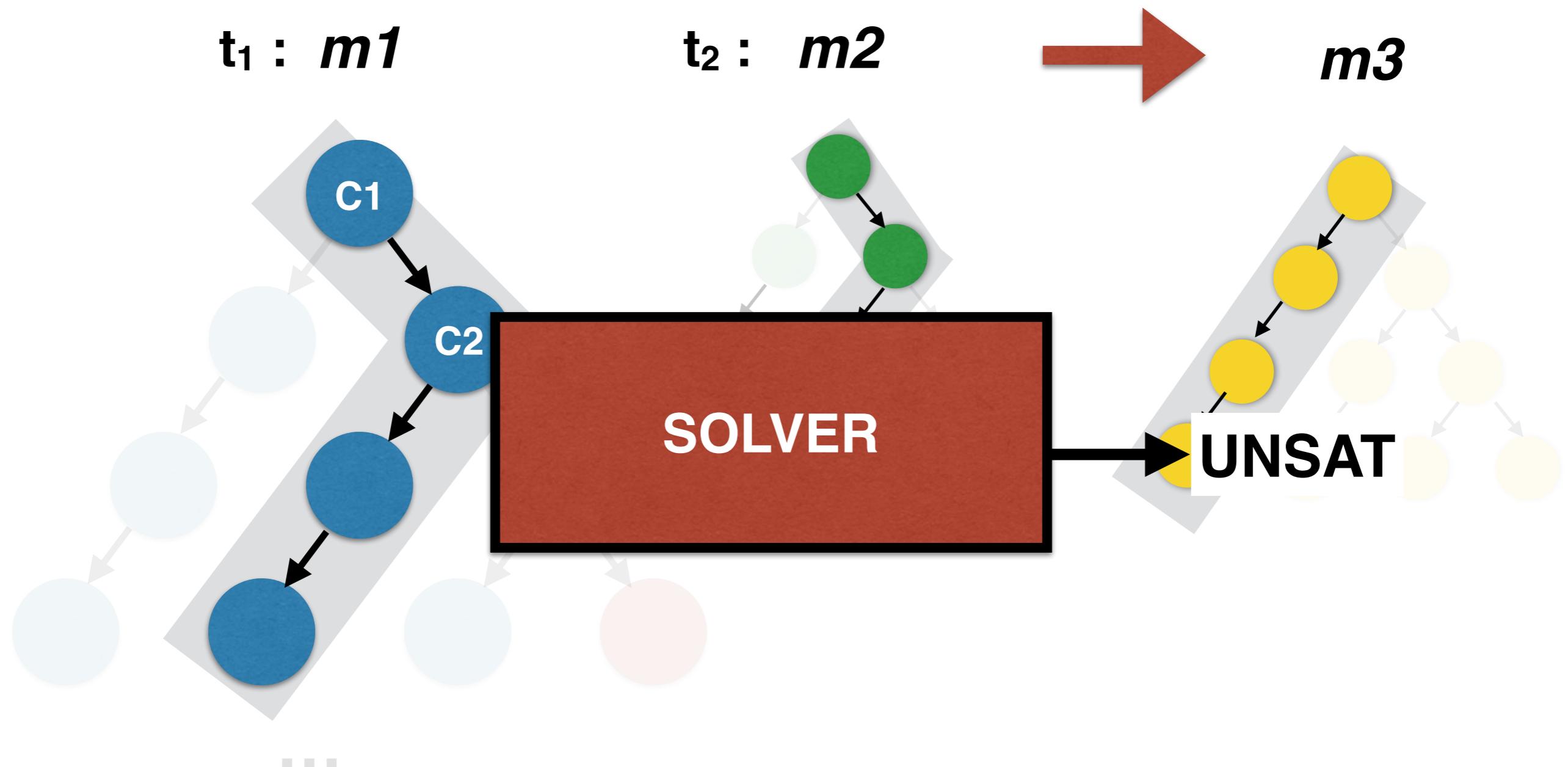
# Iteration - 2 : Concrete Execution



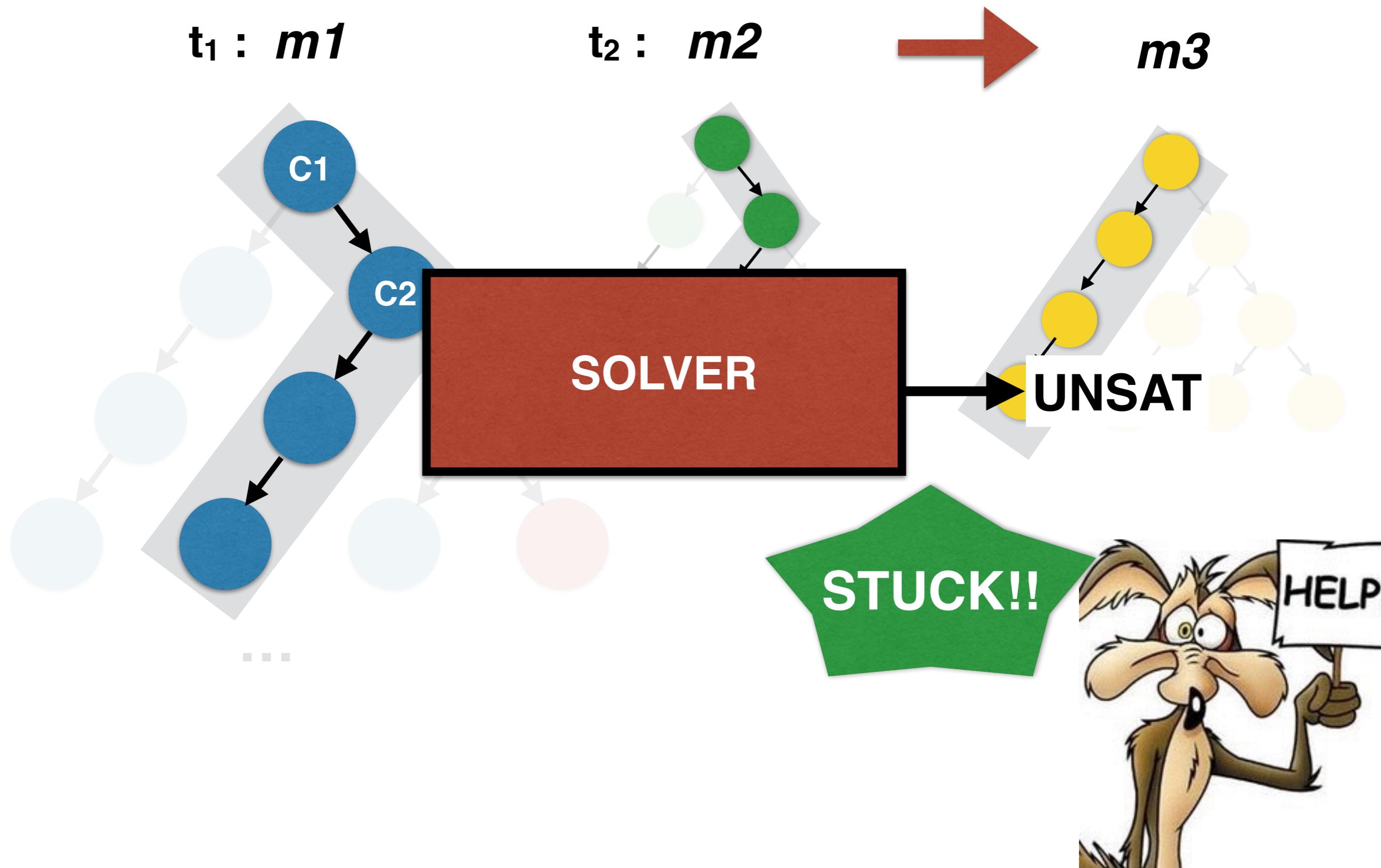
# Iteration - 2 : Concrete Execution



# Iteration - 2 : Concrete Execution

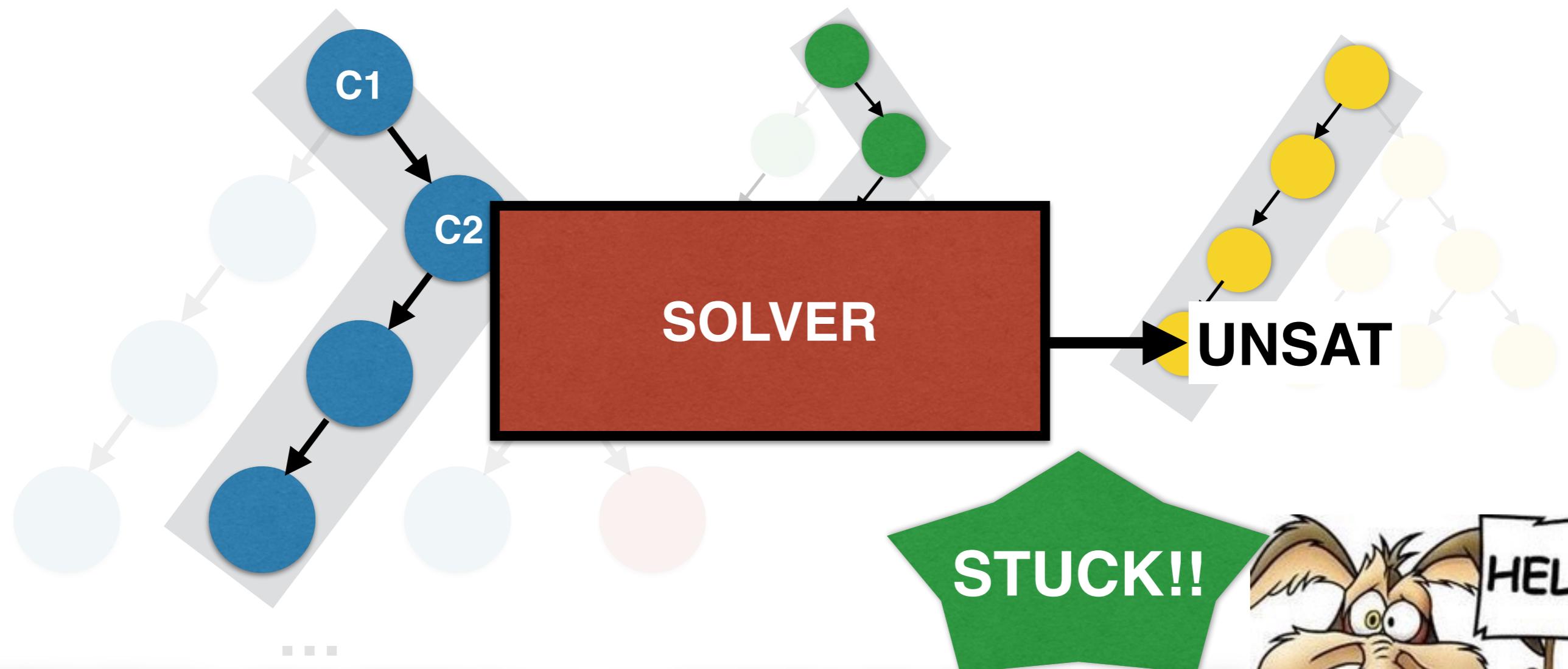


# Iteration - 2 : Concrete Execution



# Iteration - 2 : Concrete Execution

$t_1 : m1$        $t_2 : m2$        $\rightarrow$        $m3$

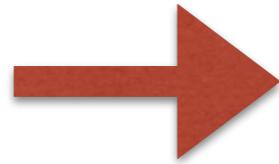


Updates seen on executed paths in  $m2$ ,  $m3$   
cannot negate  $C2$ ; Use static analysis

# Iteration - 3 : Static Analysis

$t_1 : m1$

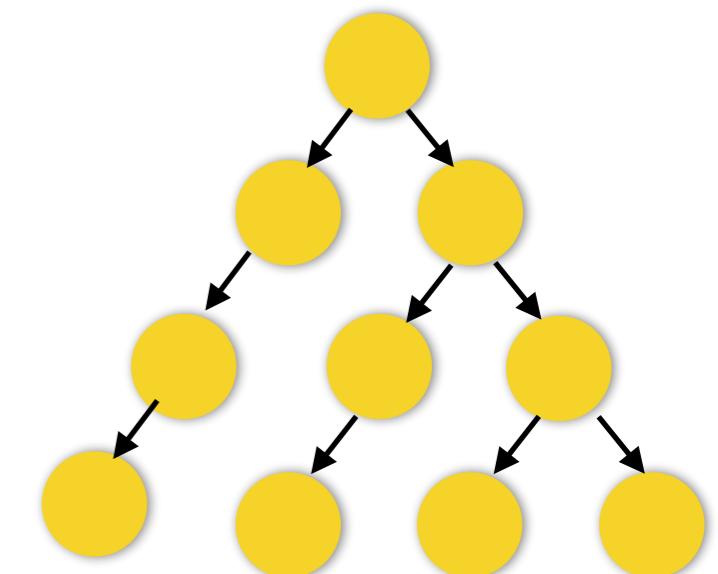
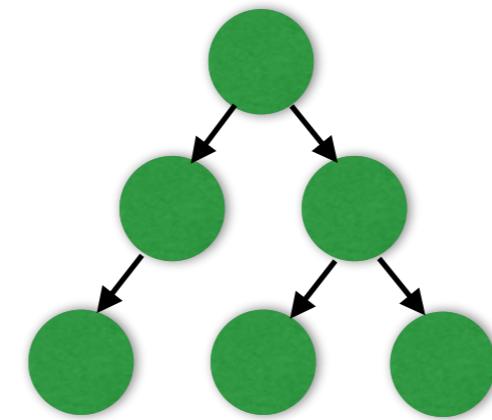
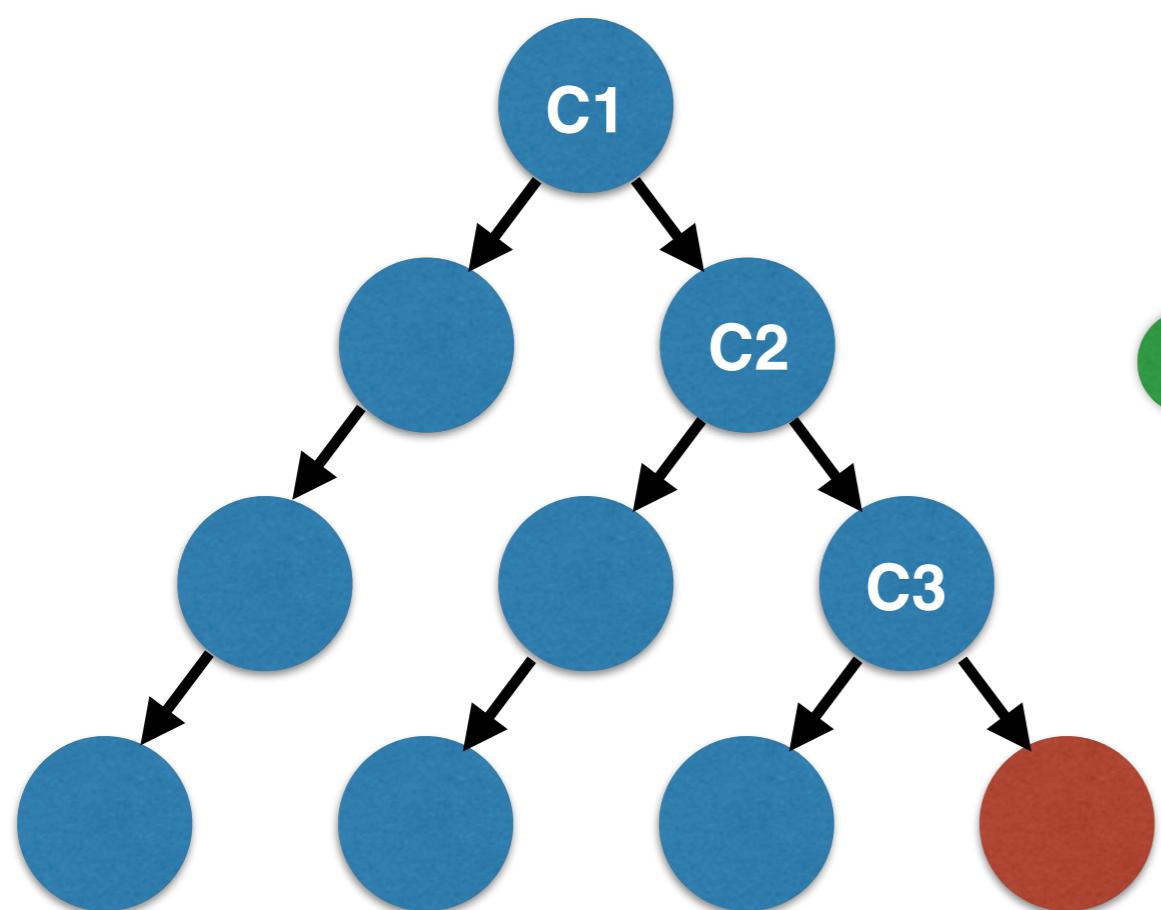
$t_2 : m2$



$m3$

# Iteration - 3 : Static Analysis

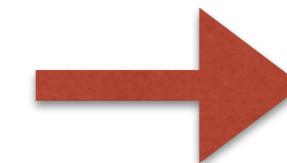
$t_1 : m1$        $t_2 : m2$        $\rightarrow$        $m3$



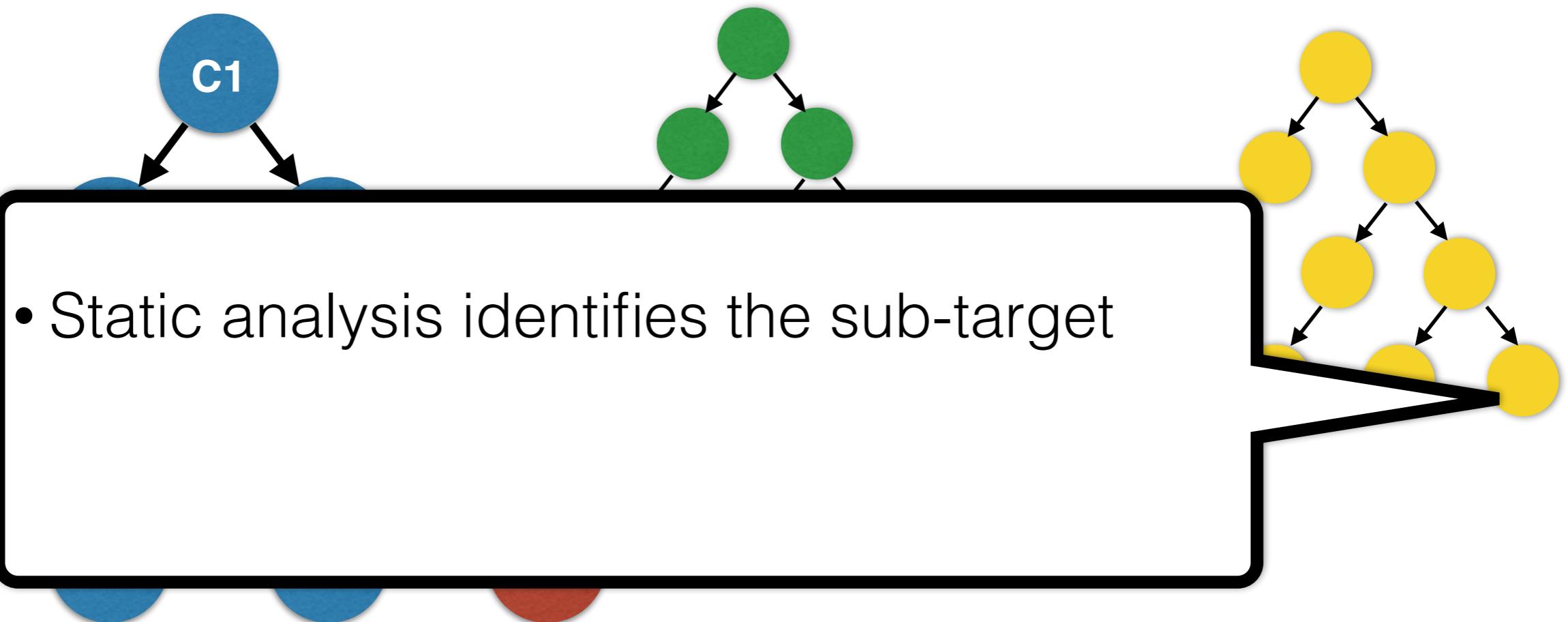
# Iteration - 3 : Static Analysis

$t_1 : m1$

$t_2 : m2$

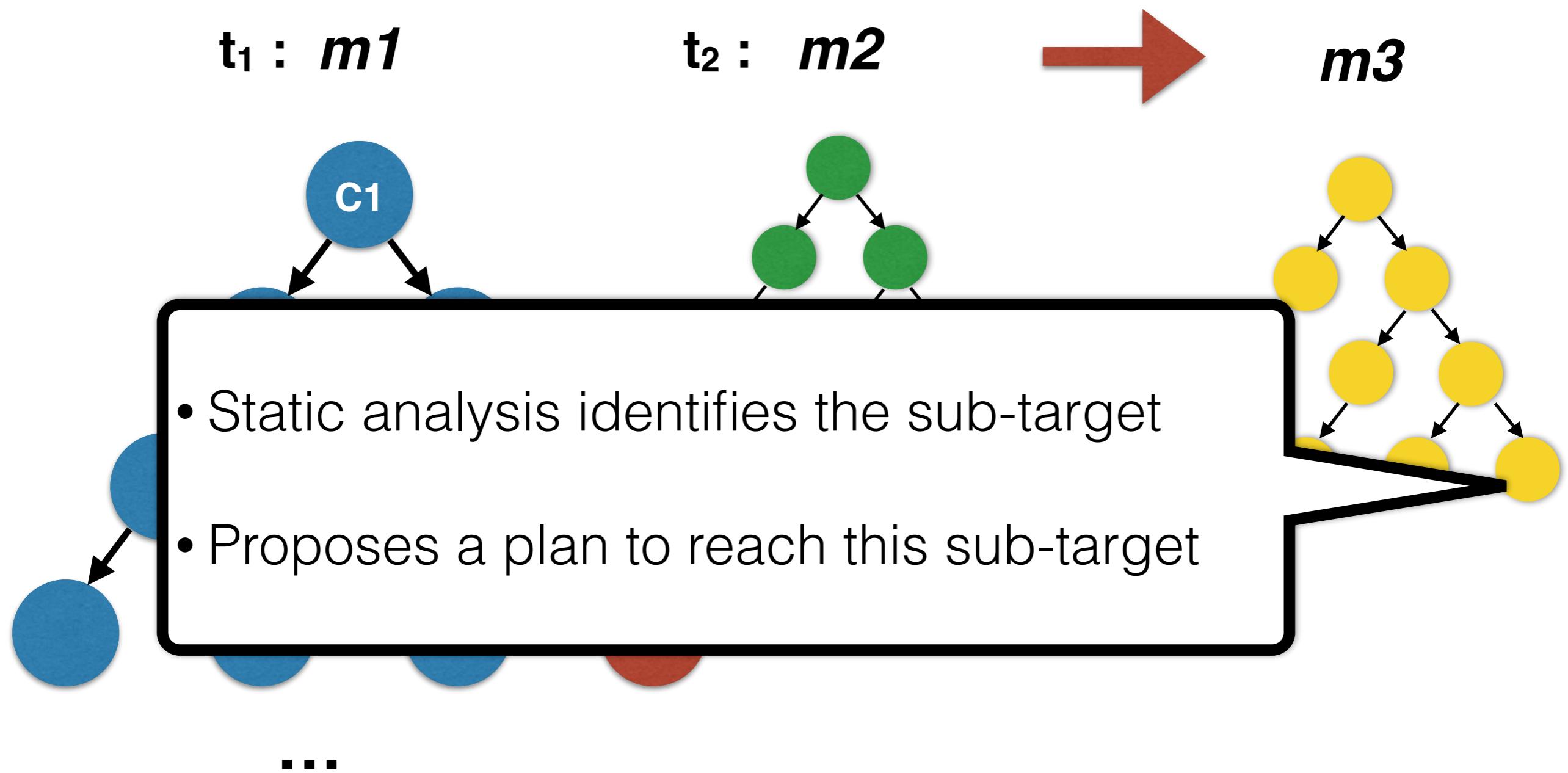


$m3$

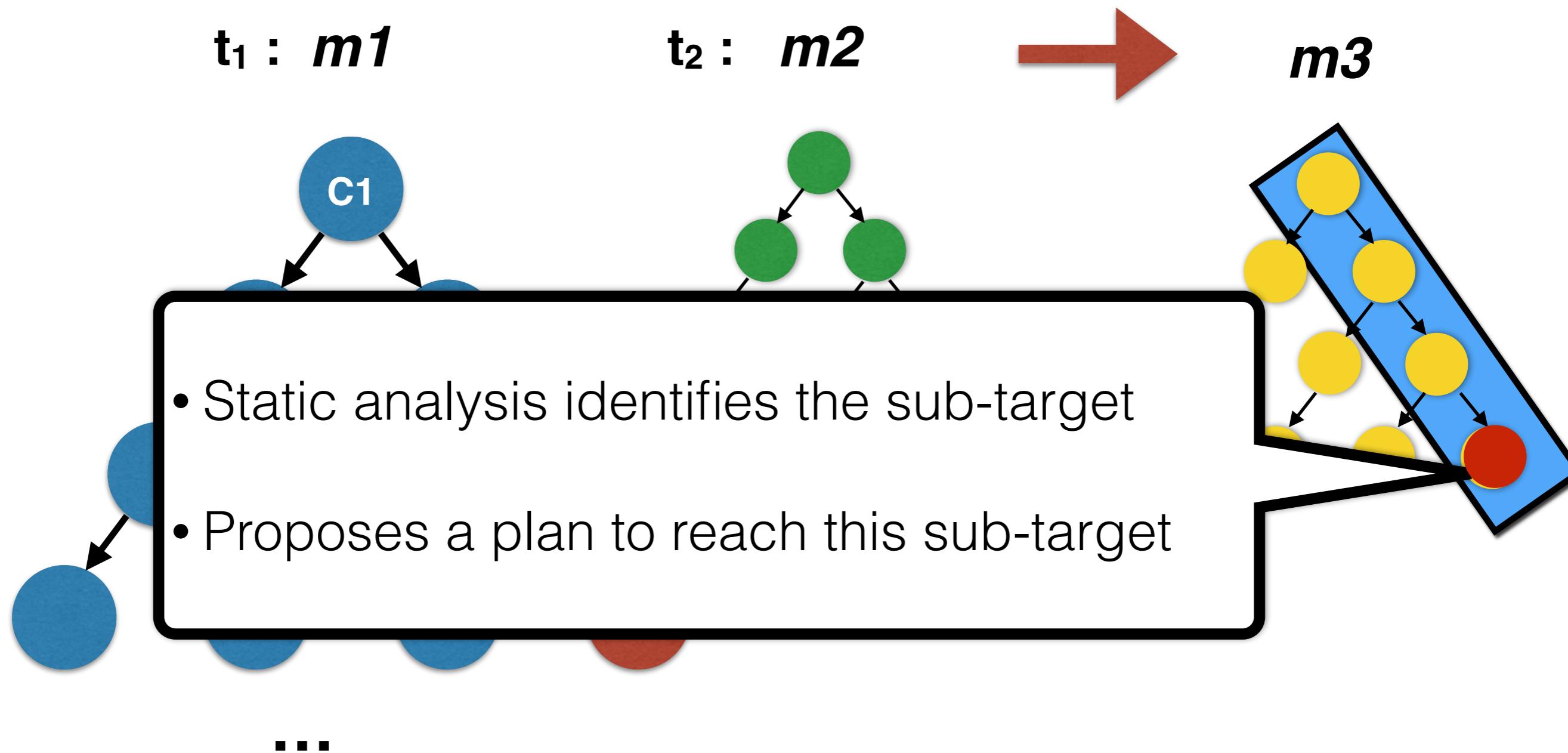


...

# Iteration - 3 : Static Analysis

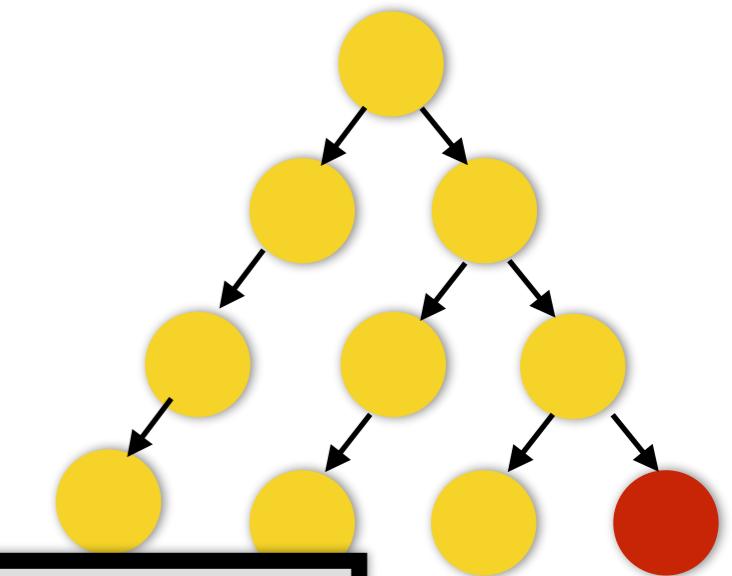
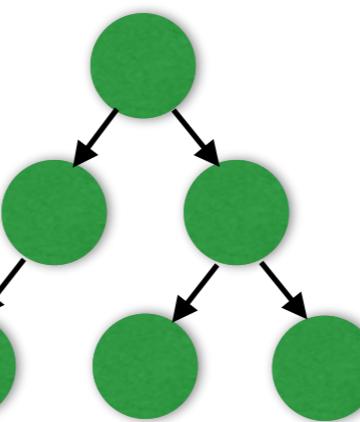
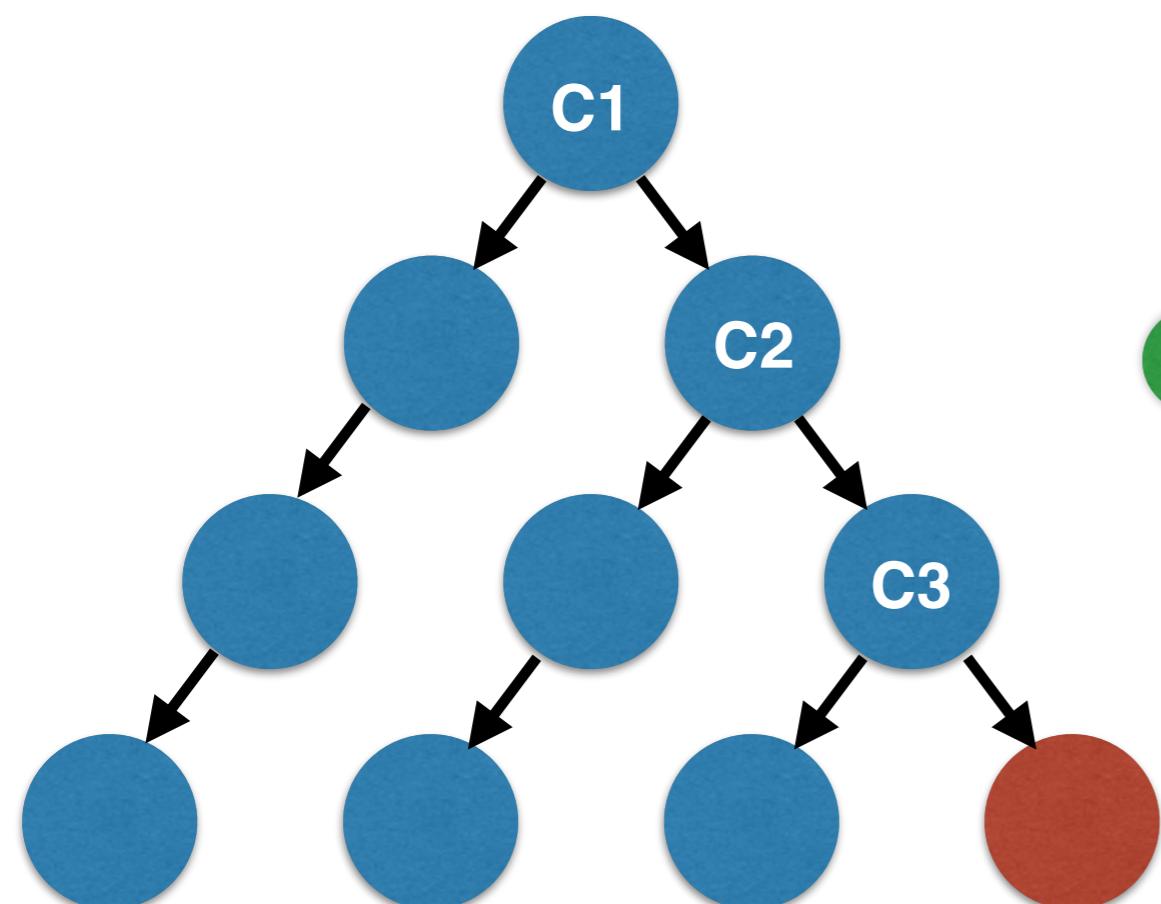


# Iteration - 3 : Static Analysis



# Iteration - 3 : Concrete Execution

$t_1 : m1$        $t_2 : m2$        $\rightarrow$        $m3$



Apply MINION for  
reaching this target

...

After a few more iterations...

# Iteration - 4 : Concrete Execution

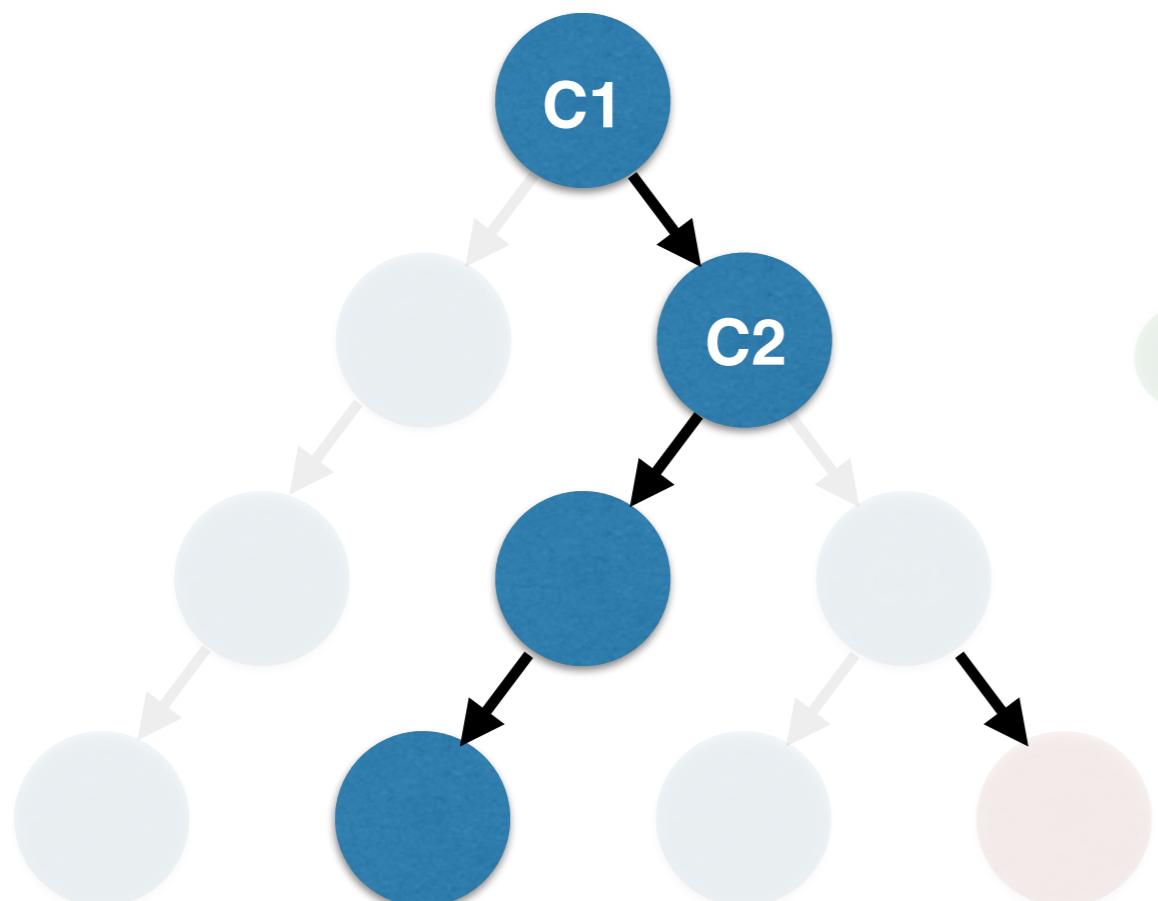
**$t_1 : m1$**

**$t_2 : m2$**

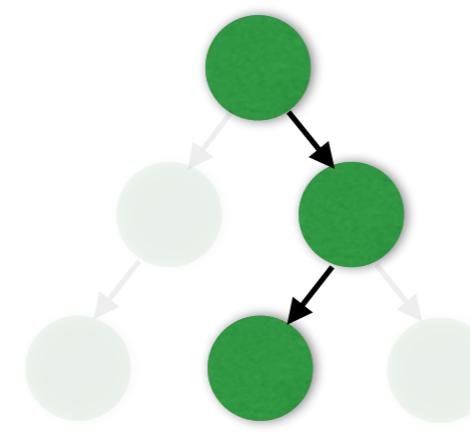
**$t_3 : m3$**

# Iteration - 4 : Concrete Execution

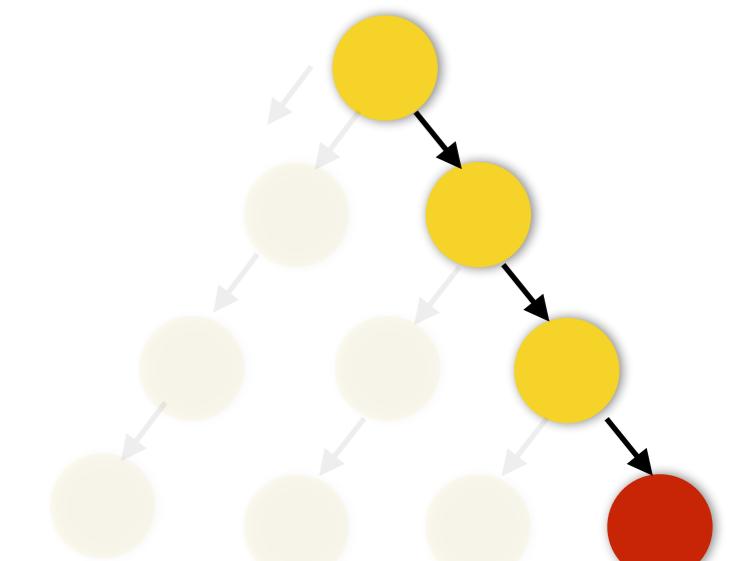
$t_1 : m1$



$t_2 : m2$



$t_3 : m3$

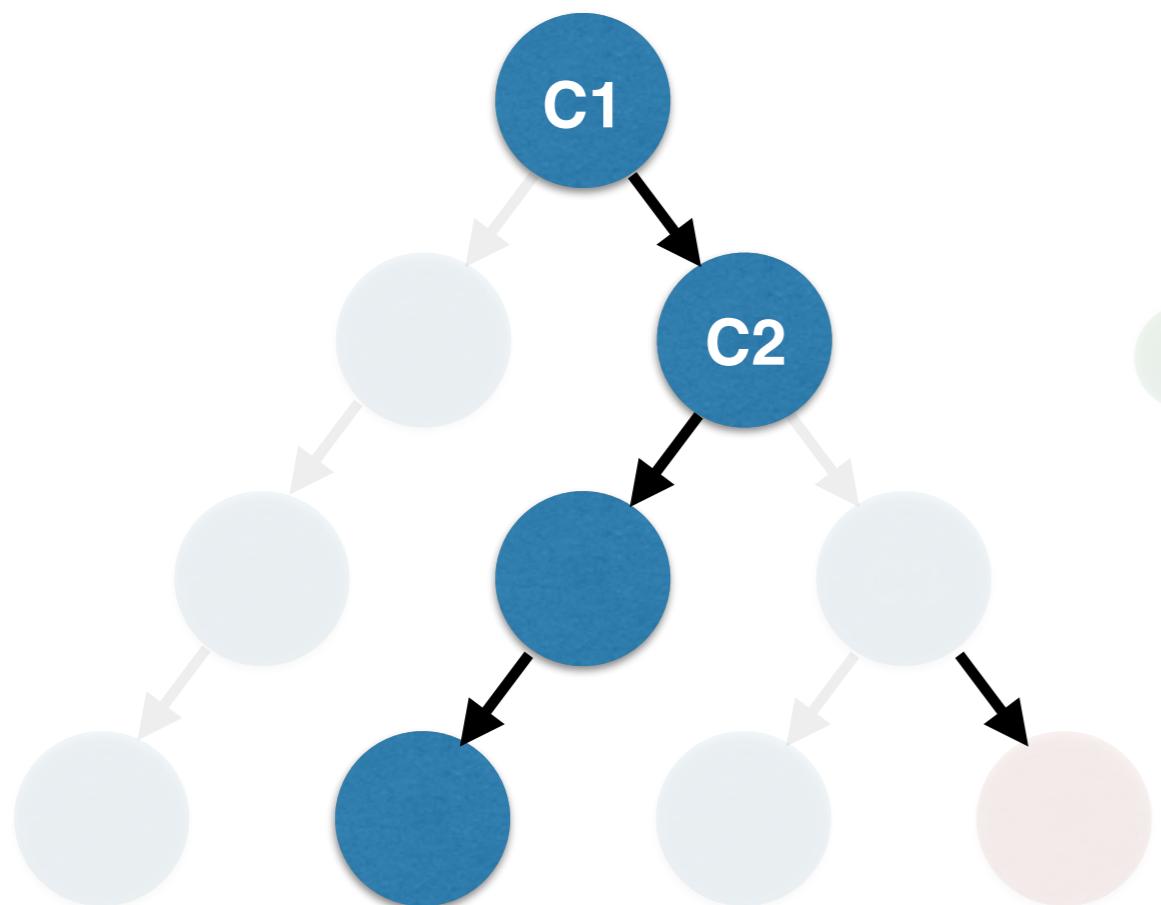


...

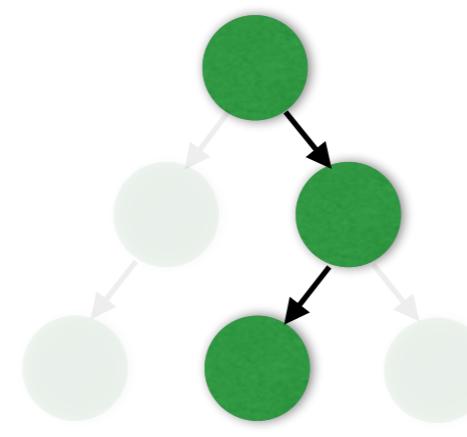
...

# Iteration - 4 : Concrete Execution

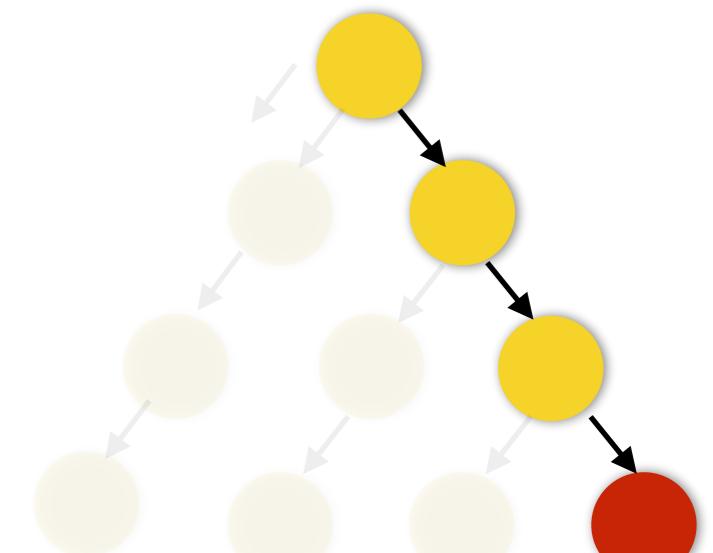
$t_1 : m1$



$t_2 : m2$



$t_3 : m3$



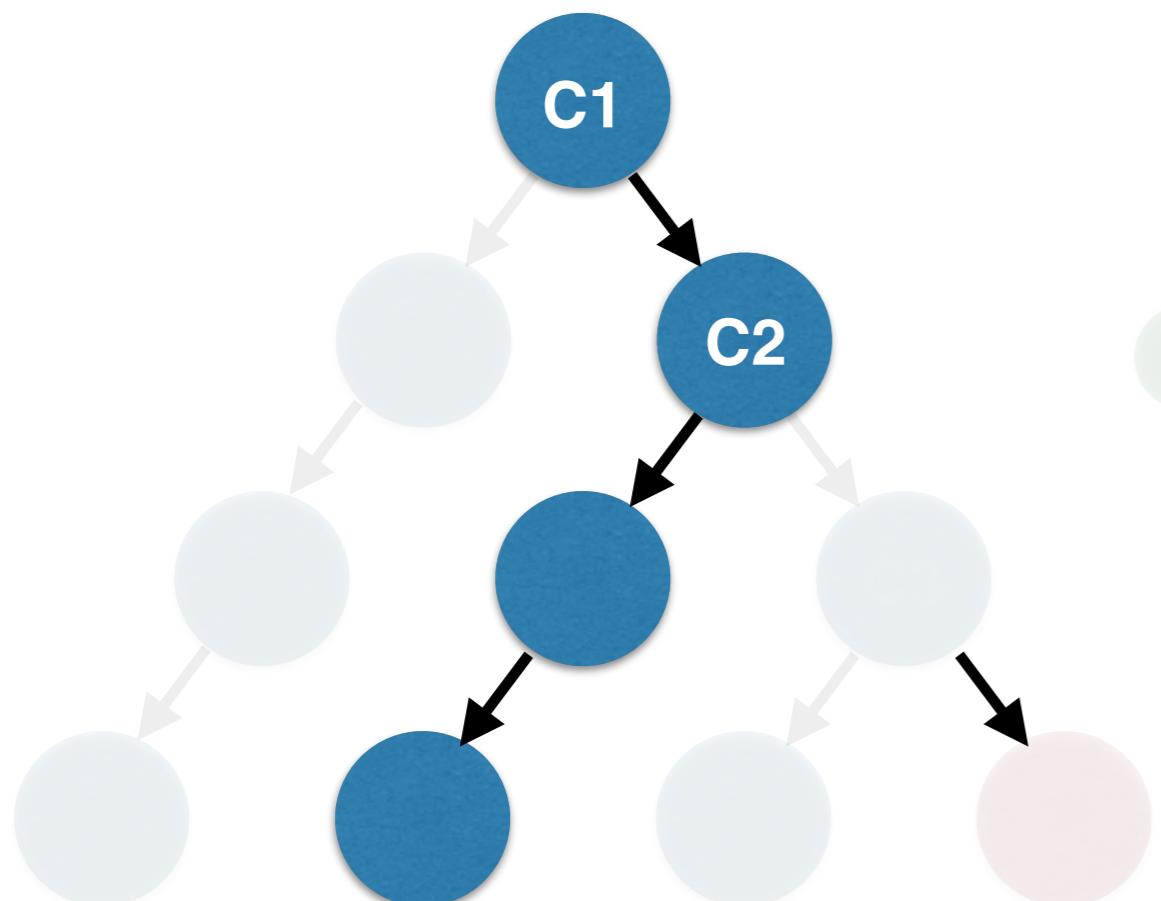
**cons<sub>1</sub>**

**cons<sub>2</sub>**

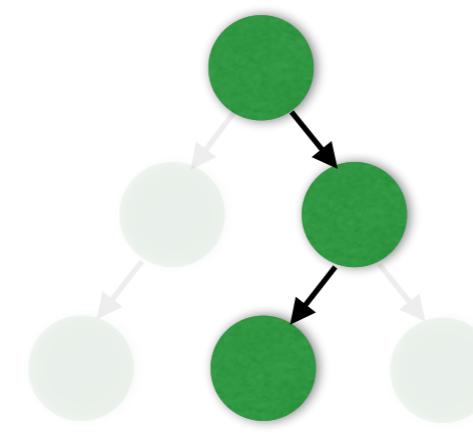
**cons<sub>3</sub>**

# Iteration - 4 : Concrete Execution

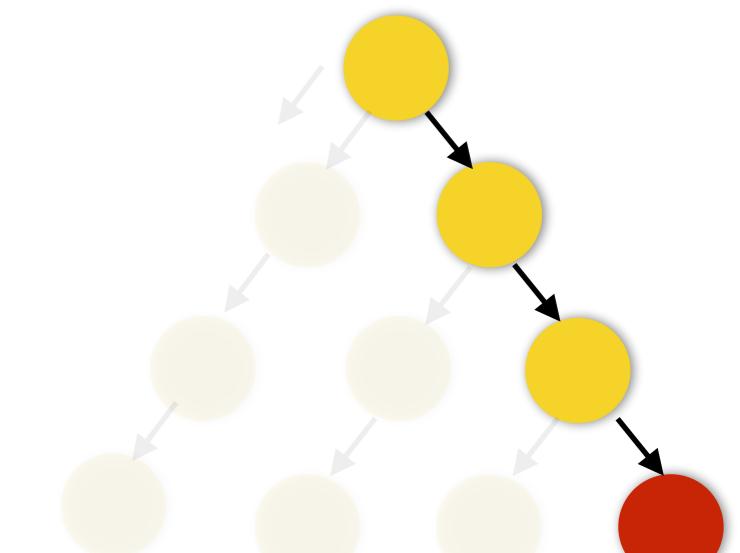
$t_1 : m1$



$t_2 : m2$



$t_3 : m3$



cons<sub>1</sub>

cons<sub>2</sub>

cons<sub>3</sub>

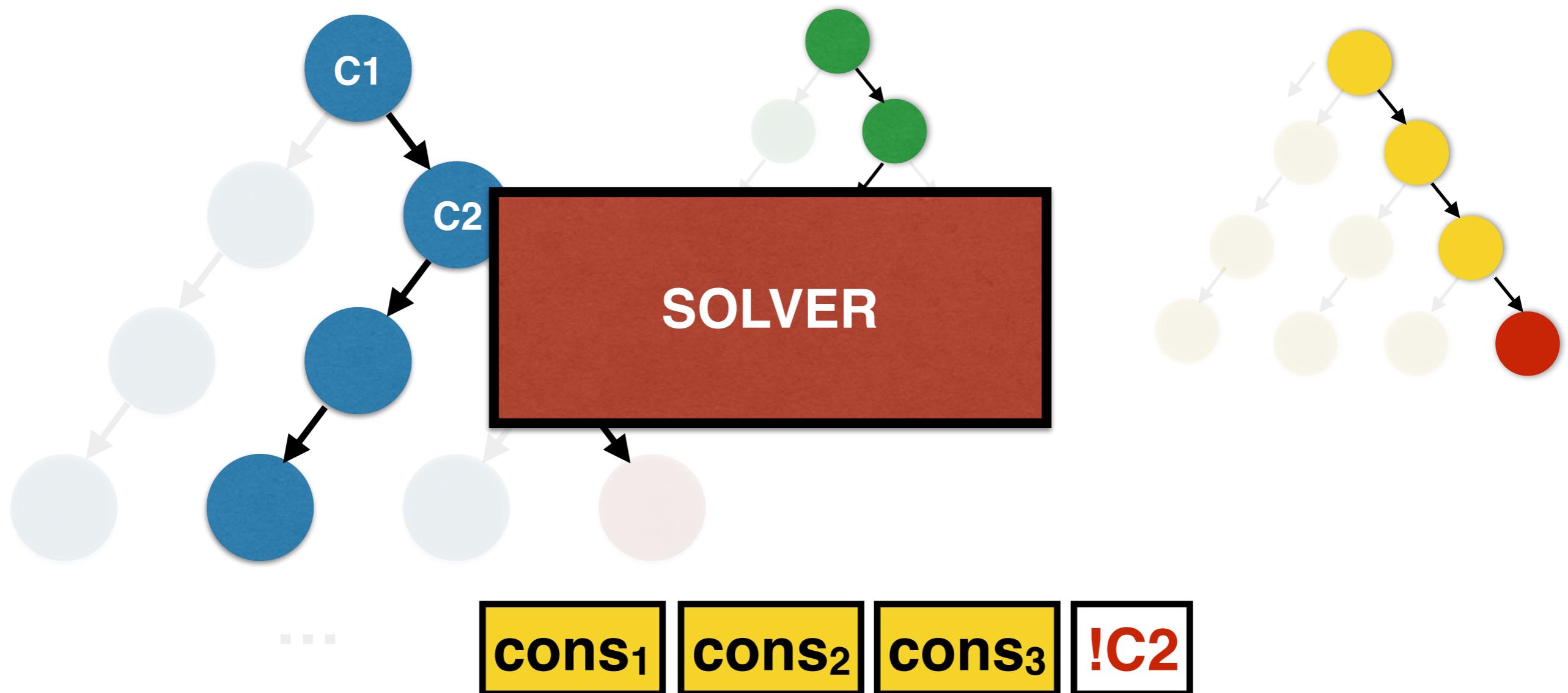
!C2

# Iteration - 4 : Concrete Execution

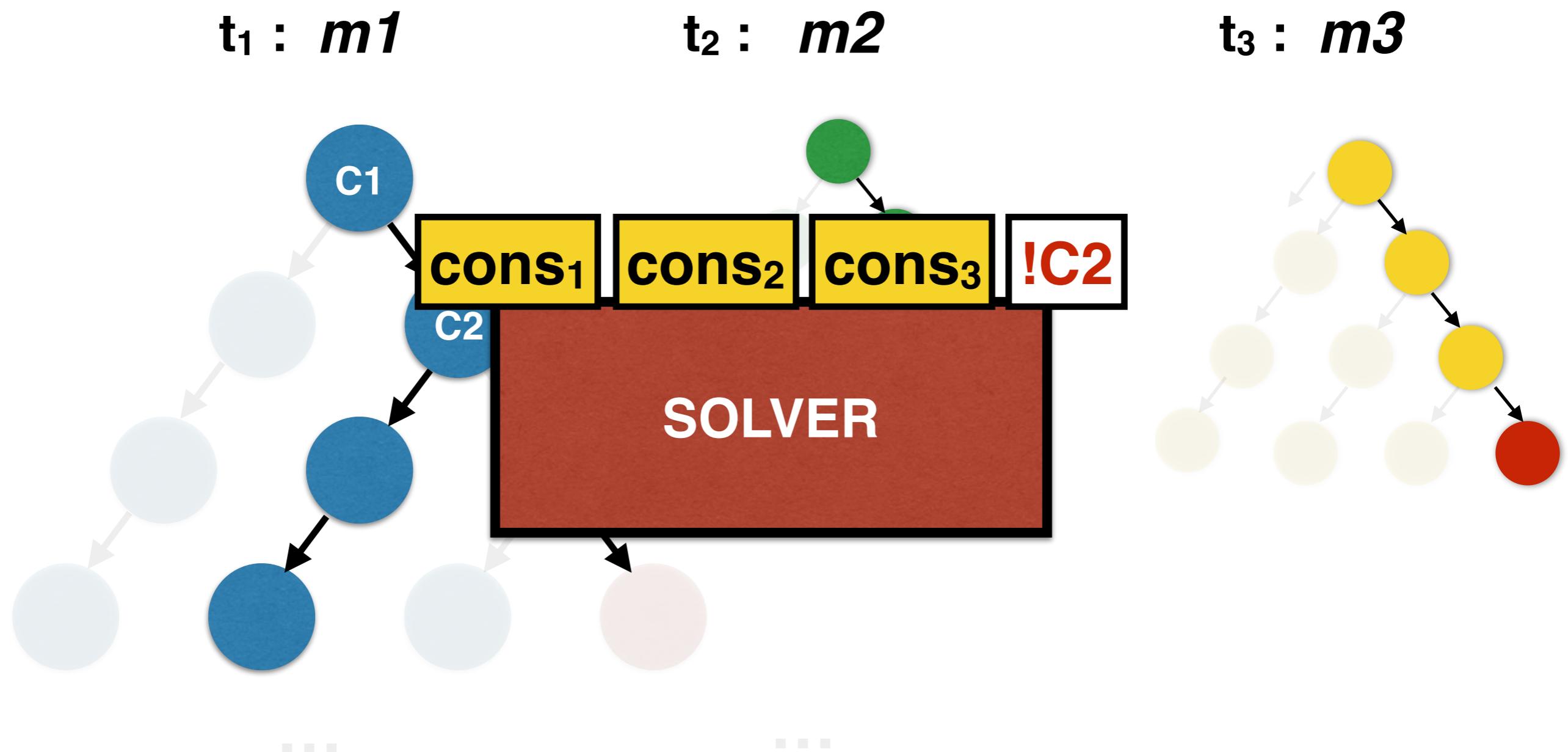
$t_1 : m1$

$t_2 : m2$

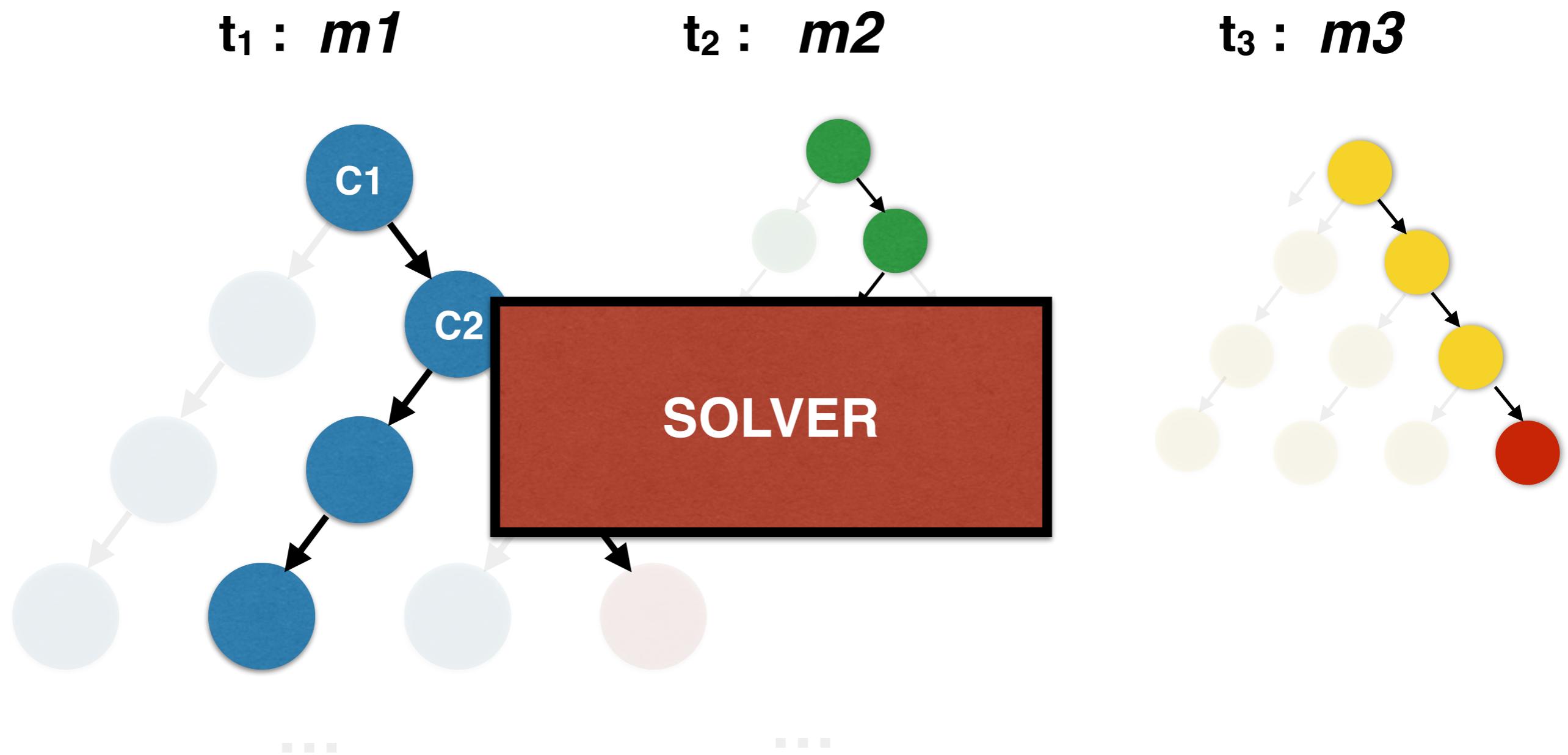
$t_3 : m3$



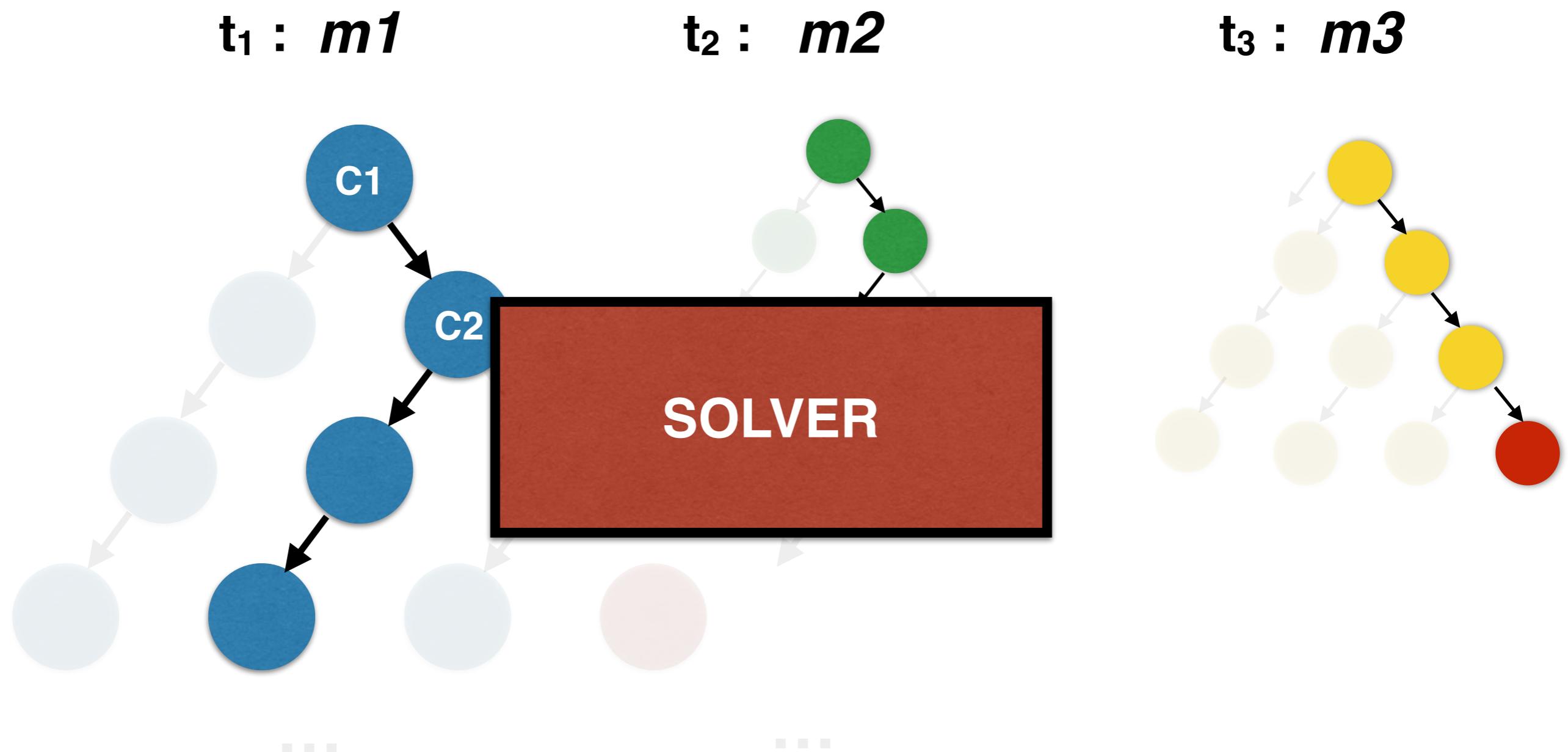
# Iteration - 4 : Concrete Execution



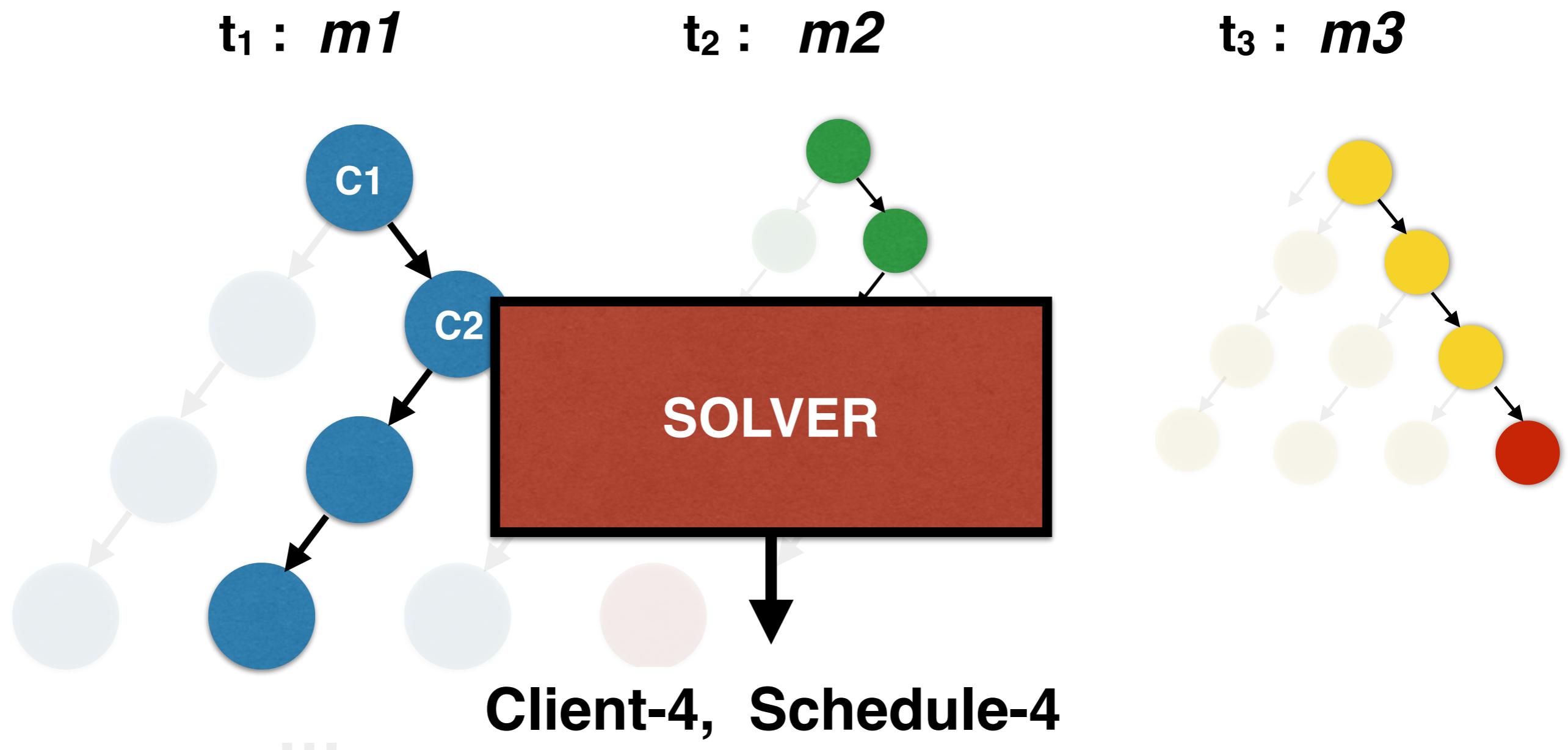
# Iteration - 4 : Concrete Execution



# Iteration - 4 : Concrete Execution

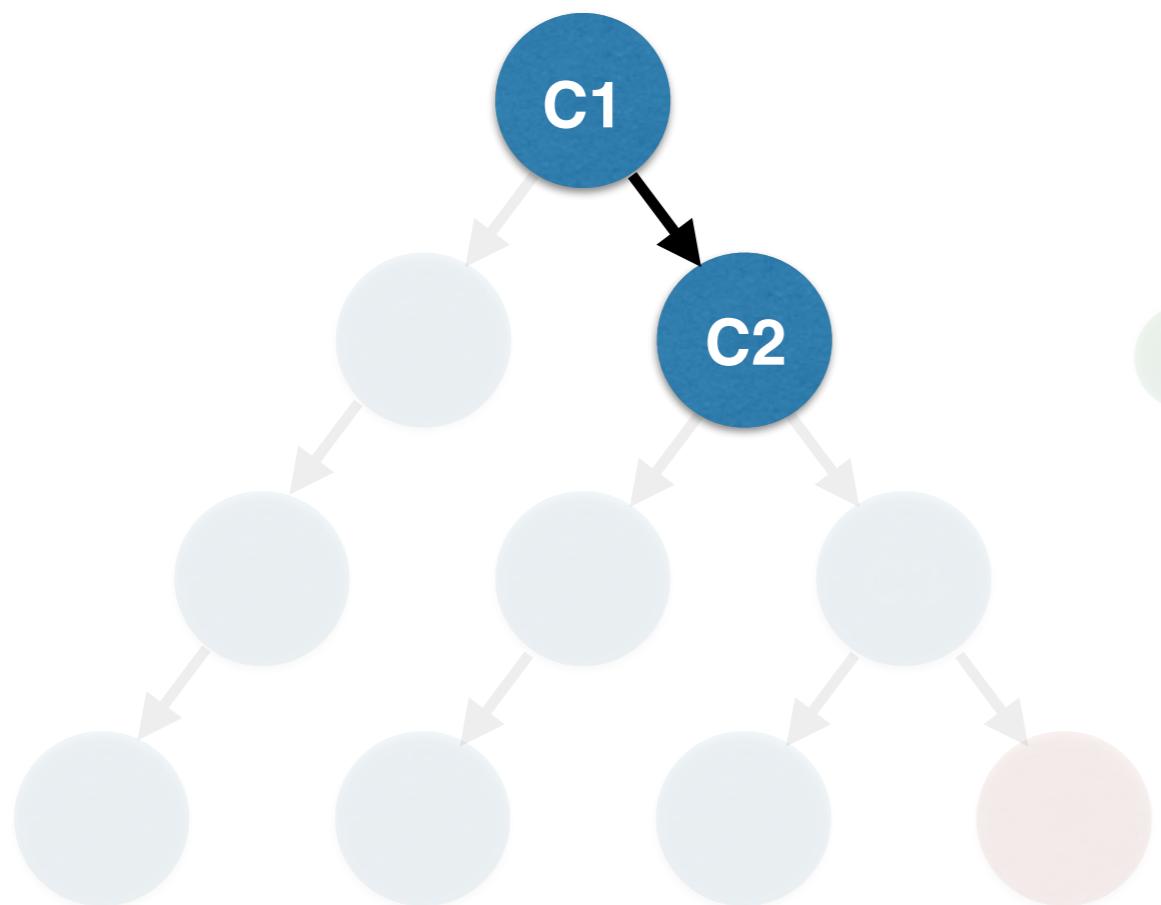


# Iteration - 4 : Concrete Execution

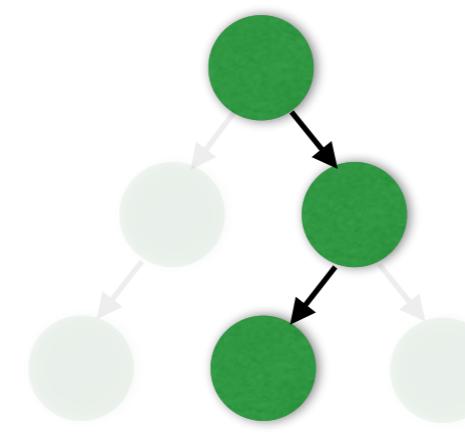


# Iteration - 4 : Concrete Execution

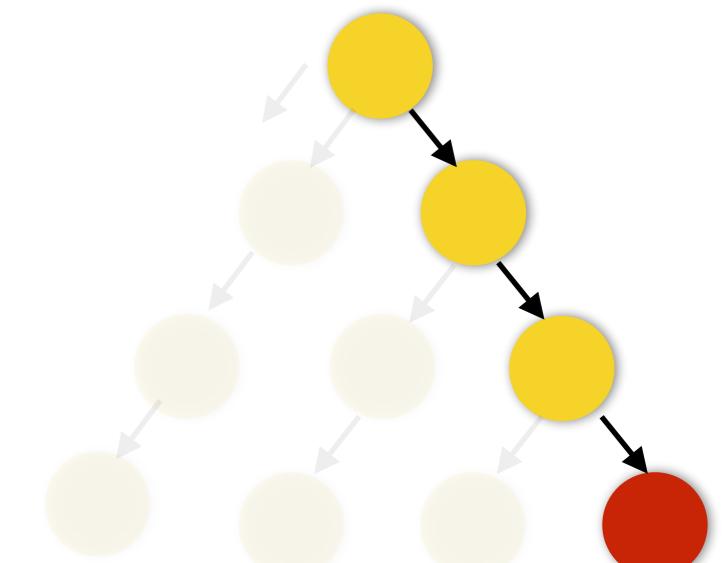
$t_1 : m1$



$t_2 : m2$

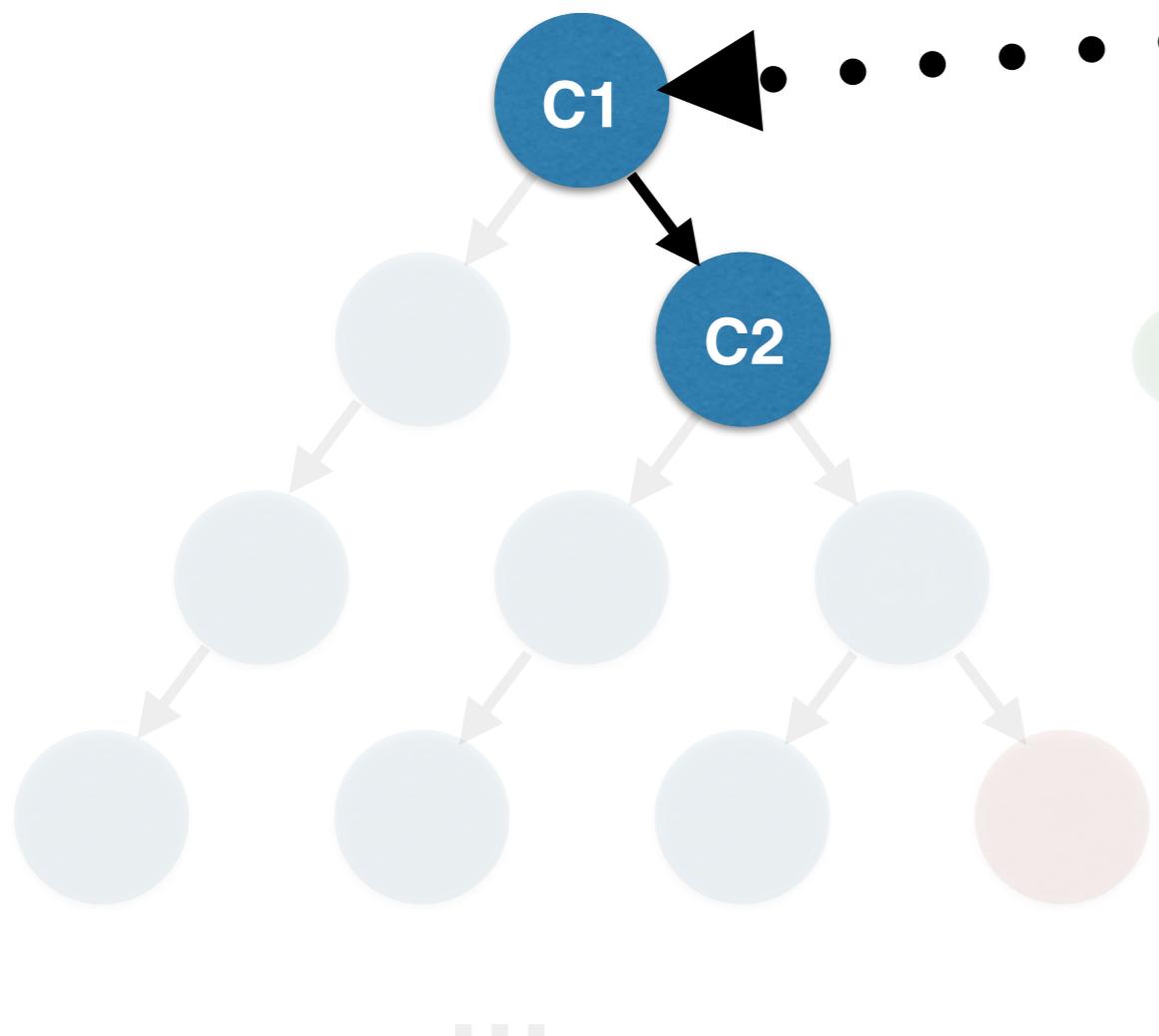


$t_3 : m3$

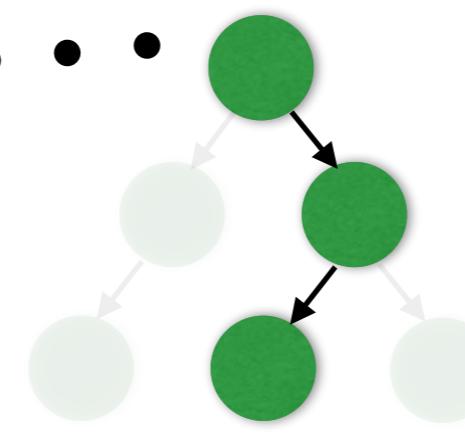


# Iteration - 4 : Concrete Execution

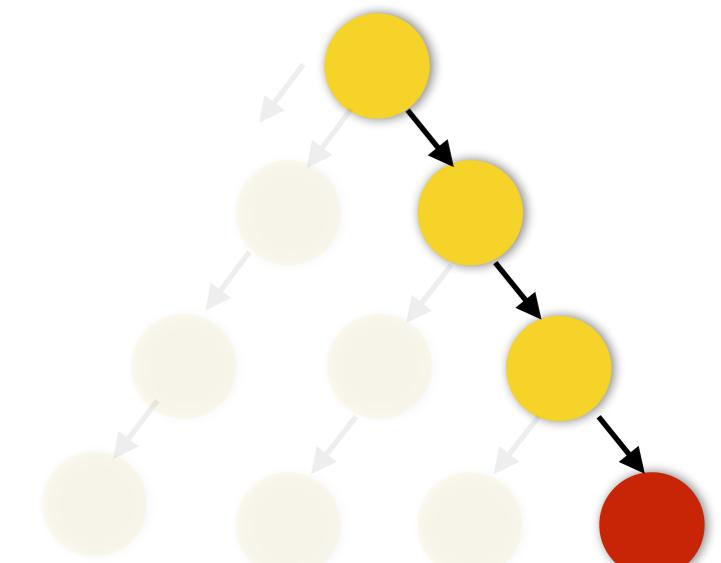
$t_1 : m1$



$t_2 : m2$

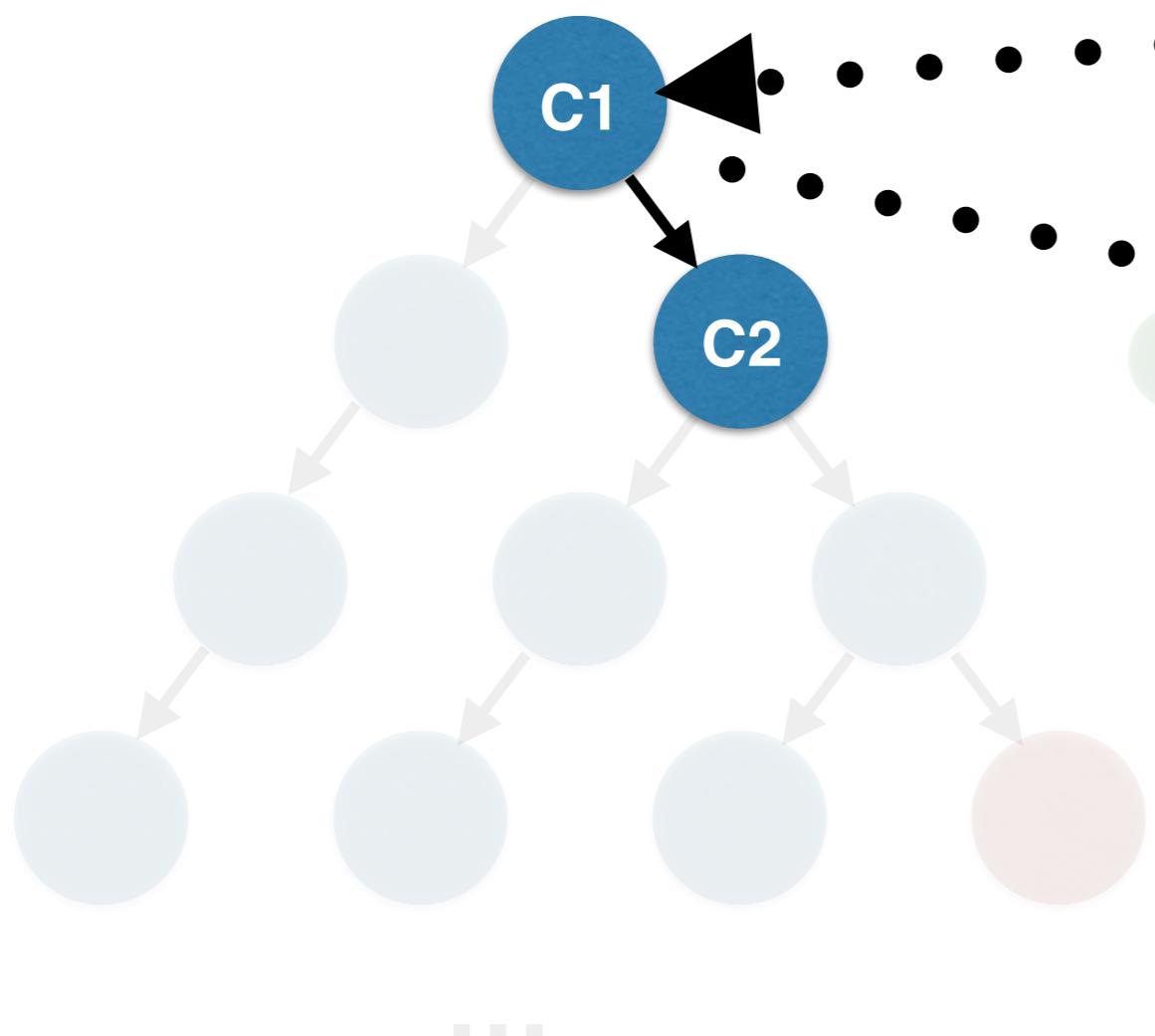


$t_3 : m3$

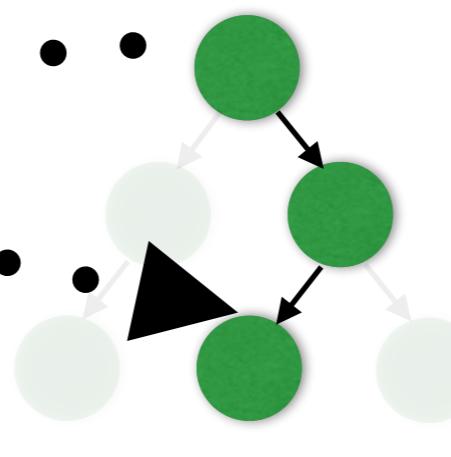


# Iteration - 4 : Concrete Execution

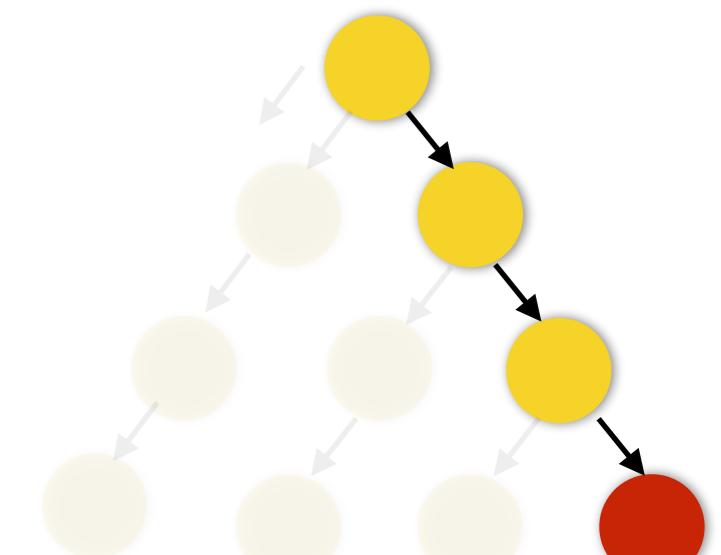
$t_1 : m1$



$t_2 : m2$



$t_3 : m3$

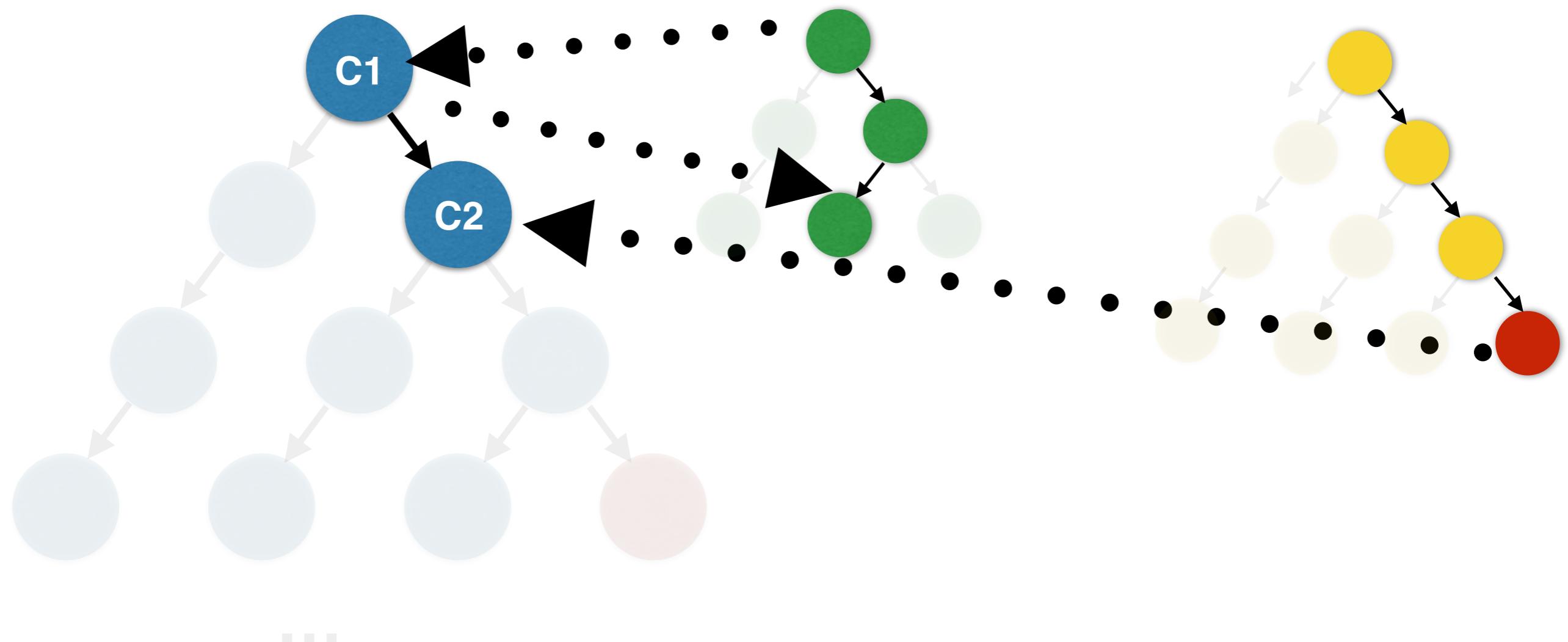


# Iteration - 4 : Concrete Execution

$t_1 : m1$

$t_2 : m2$

$t_3 : m3$

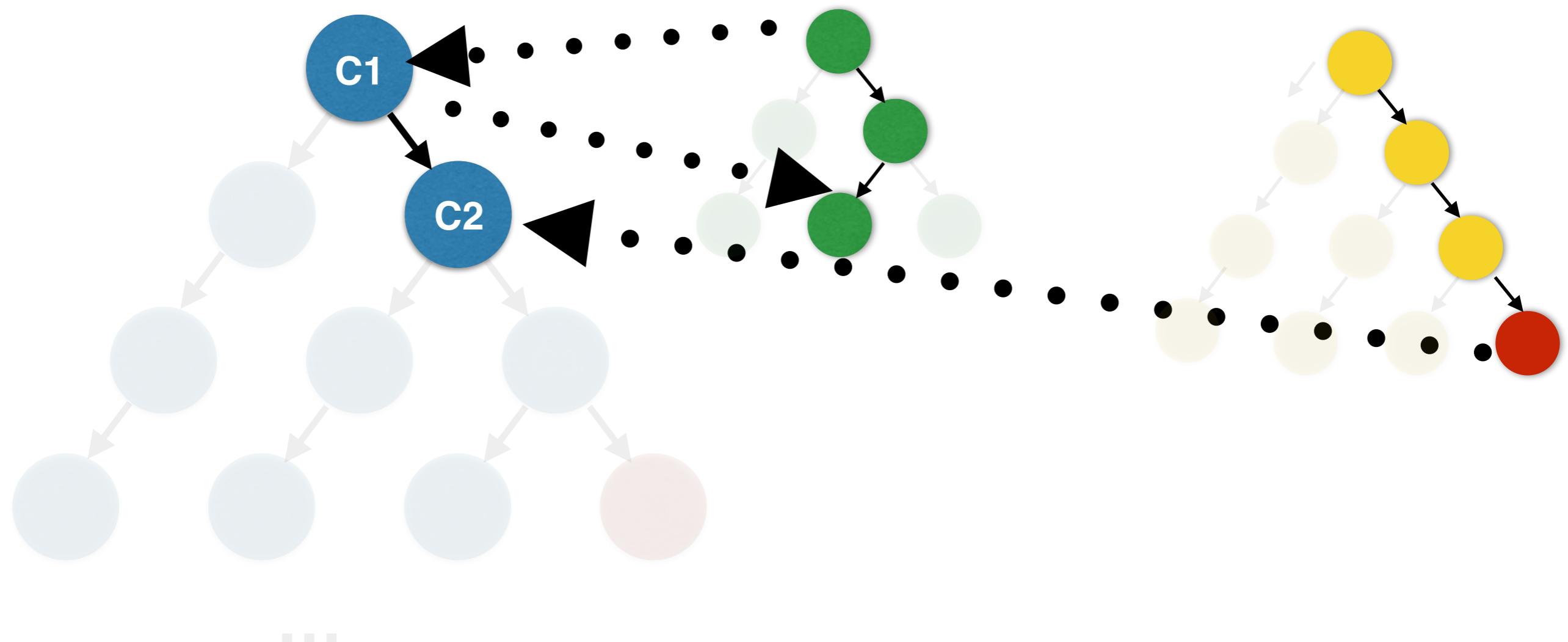


# Iteration - 4 : Concrete Execution

$t_1 : m1$

$t_2 : m2$

$t_3 : m3$

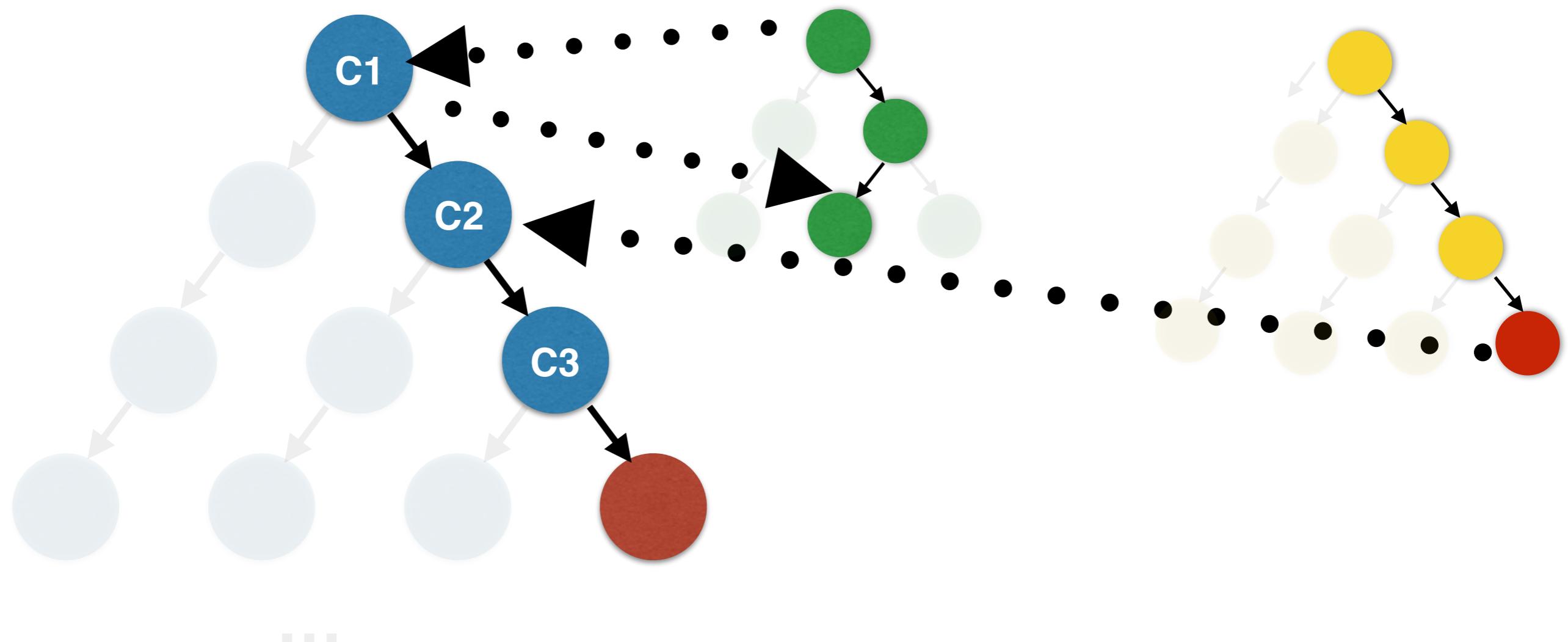


# Iteration - 4 : Concrete Execution

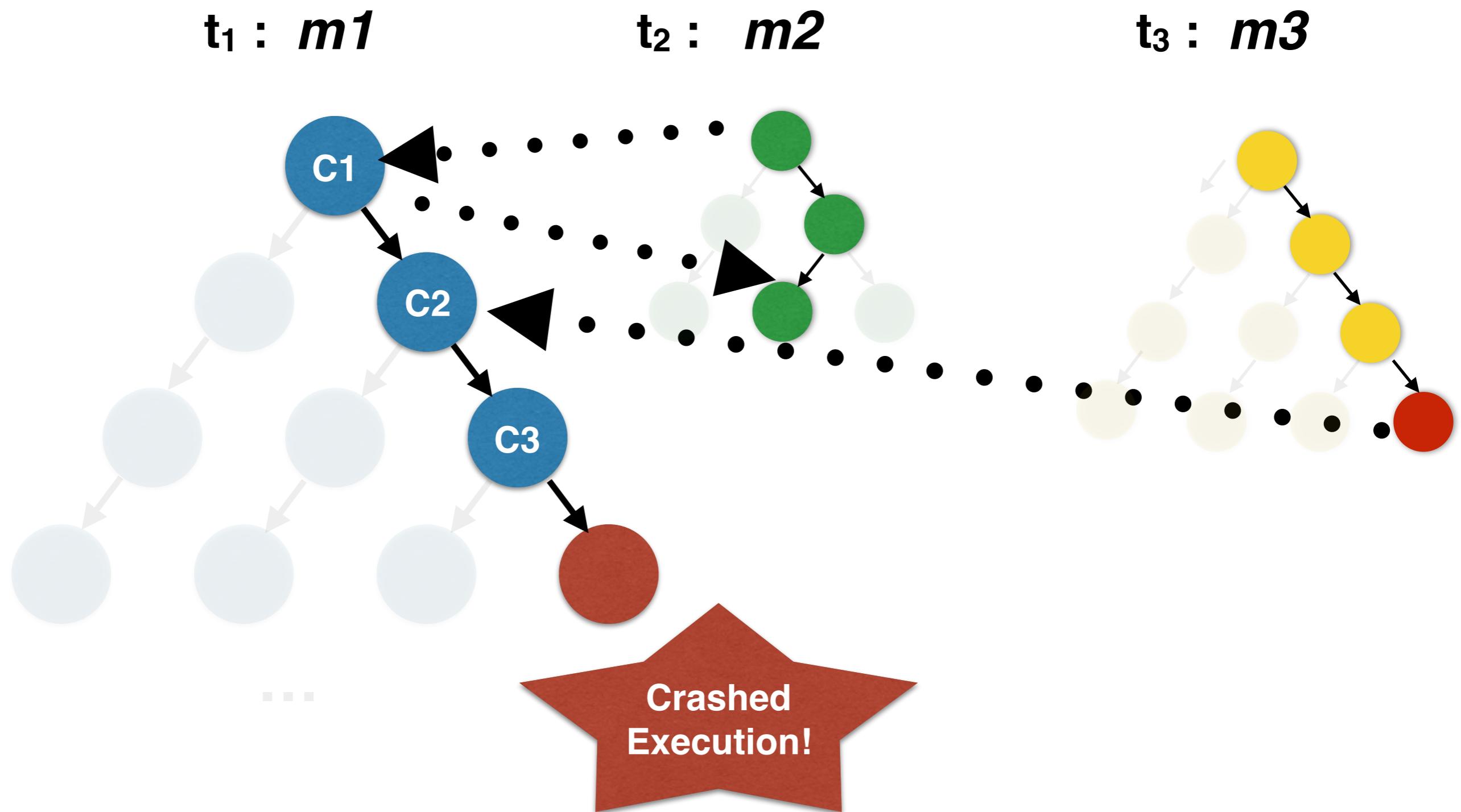
$t_1 : m1$

$t_2 : m2$

$t_3 : m3$



# Iteration - 4 : Concrete Execution



# Implementation

# Implementation

- ◆ Built on top of the **soot** bytecode analysis framework

# Implementation

- ◆ Built on top of the **soot** bytecode analysis framework
- ◆ Used the **Z3** constraint solver

# Implementation

- ◆ Built on top of the **soot** bytecode analysis framework
- ◆ Used the **Z3** constraint solver
- ◆ Evaluated on open source Java libraries

# Implementation

- ◆ Built on top of the **soot** bytecode analysis framework
- ◆ Used the **Z3** constraint solver
- ◆ Evaluated on open source Java libraries
- ◆ Input sequential client: invoke each method in a class once with random objects

# Benchmark Information

Benchmark	Version	Class name	M
cache4j	0.4	CacheCleaner (C1)	3
classpath	0.99	BufferedInputStream (C2)	10
guava	18.0	SimpleStatsCounter (C3)	6
hsqldb	2.3.3	DoubleIntIndex (C4)	32
java.lang	1.7	StringBuffer (C5)	50
java.io	1.8	CharArrayReader (C6)	8
		PipedReader (C7)	5
		PushbackReader (C8)	11
		StringReader (C9)	8
java.util	1.7	Vector (C10)	43

# Benchmark Information

Benchmark	Version	Class name	M
cache4j	0.4	CacheCleaner (C1)	3
classpath	0.99	BufferedInputStream (C2)	10
guava	18.0	SimpleStatsCounter (C3)	6
hsqldb	2.3.3	DoubleIntIndex (C4)	32
java.lang	1.7	StringBuffer (C5)	50
java.io	1.8	CharArrayReader (C6)	8
		PipedReader (C7)	5
		PushbackReader (C8)	11
		StringReader (C9)	8
java.util	1.7	Vector (C10)	43

# Benchmark Information

Benchmark	Version	Class name	M
cache4j	0.4	CacheCleaner (C1)	3
classpath	0.99	BufferedInputStream (C2)	10
guava	18.0	SimpleStatsCounter (C3)	6
hsqldb	2.3.3	DoubleIntIndex (C4)	32
java.lang	1.7	StringBuffer (C5)	50
java.io	1.8	CharArrayReader (C6) PipedReader (C7) PushbackReader (C8) StringReader (C9)	8 5 11 8
java.util	1.7	Vector (C10)	43

# Benchmark Information

Benchmark	Version	Class name	M
cache4j	0.4	CacheCleaner (C1)	3
classpath	0.99	BufferedInputStream (C2)	10
guava	18.0	SimpleStatsCounter (C3)	6
hsqldb	2.3.3	DoubleIntIndex (C4)	32
java.lang	1.7	StringBuffer (C5)	50
java.io	1.8	CharArrayReader (C6)	8
		PipedReader (C7)	5
		PushbackReader (C8)	11
		StringReader (C9)	8
java.util	1.7	Vector (C10)	43

# Results

Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

# Results

Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

# Results

Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

# Results

Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

# Results

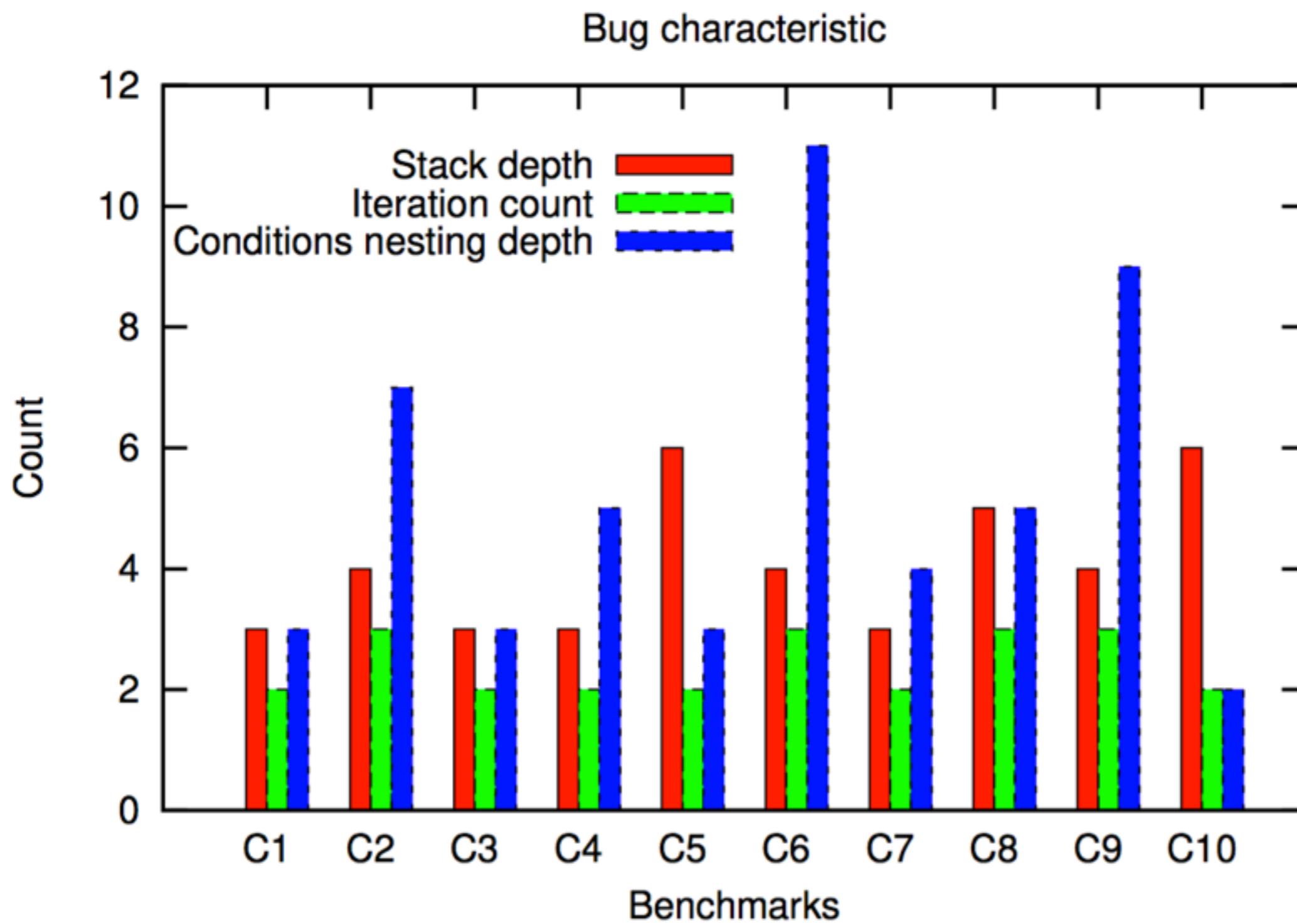
Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

# Results

Class	# of Targets	# of Constraints	Schedule length	Time (sec)	Violations	
					Seq.	Conc.
C1	2	254	37	21	0	1
C2	8	8189	25	70	2	6
C3	3	1898	43	10	0	2
C4	8	4105	10	94	4	4
C5	12	14646	116	664	5	1
C6	4	717	14	20	1	3
C7	10	3355	30	6	2	1
C8	23	7276	19	324	8	9
C9	4	900	20	21	1	3
C10	6	56045	74	120	0	1
Total	80			1350	23	31

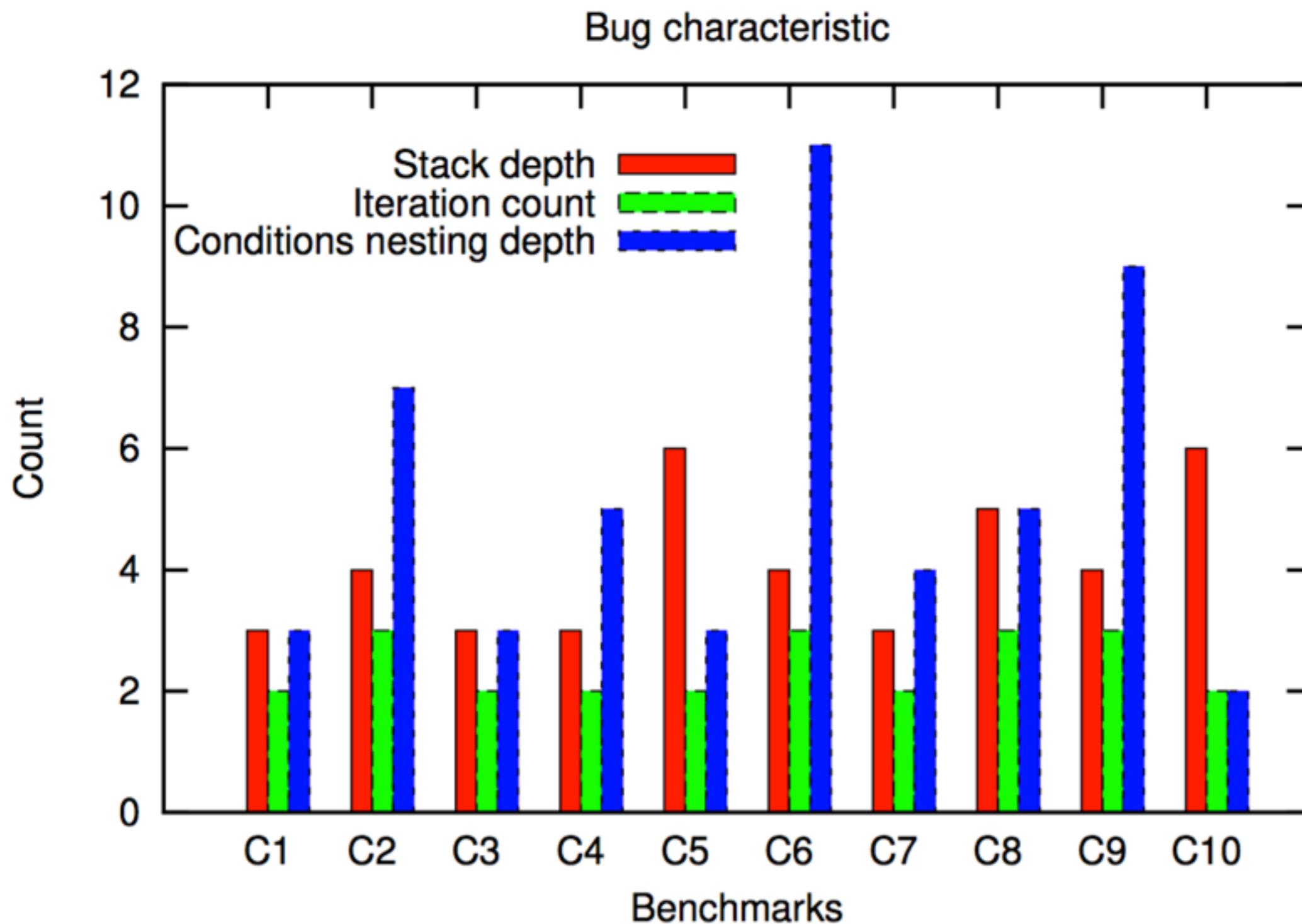
31 crashes due to concurrency

# Bug Characteristics



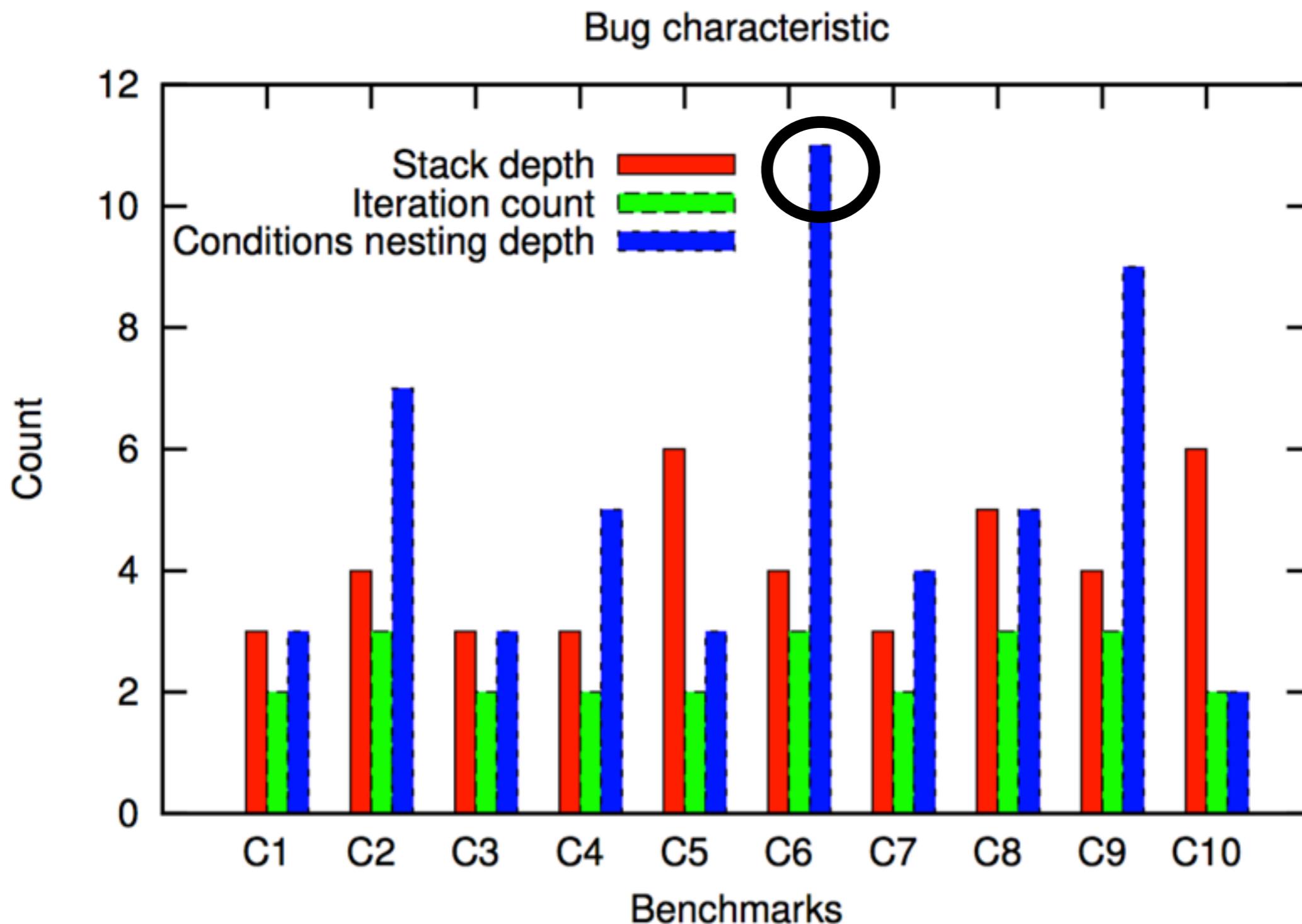
# Bug Characteristics

MINION detects complex concurrency issues



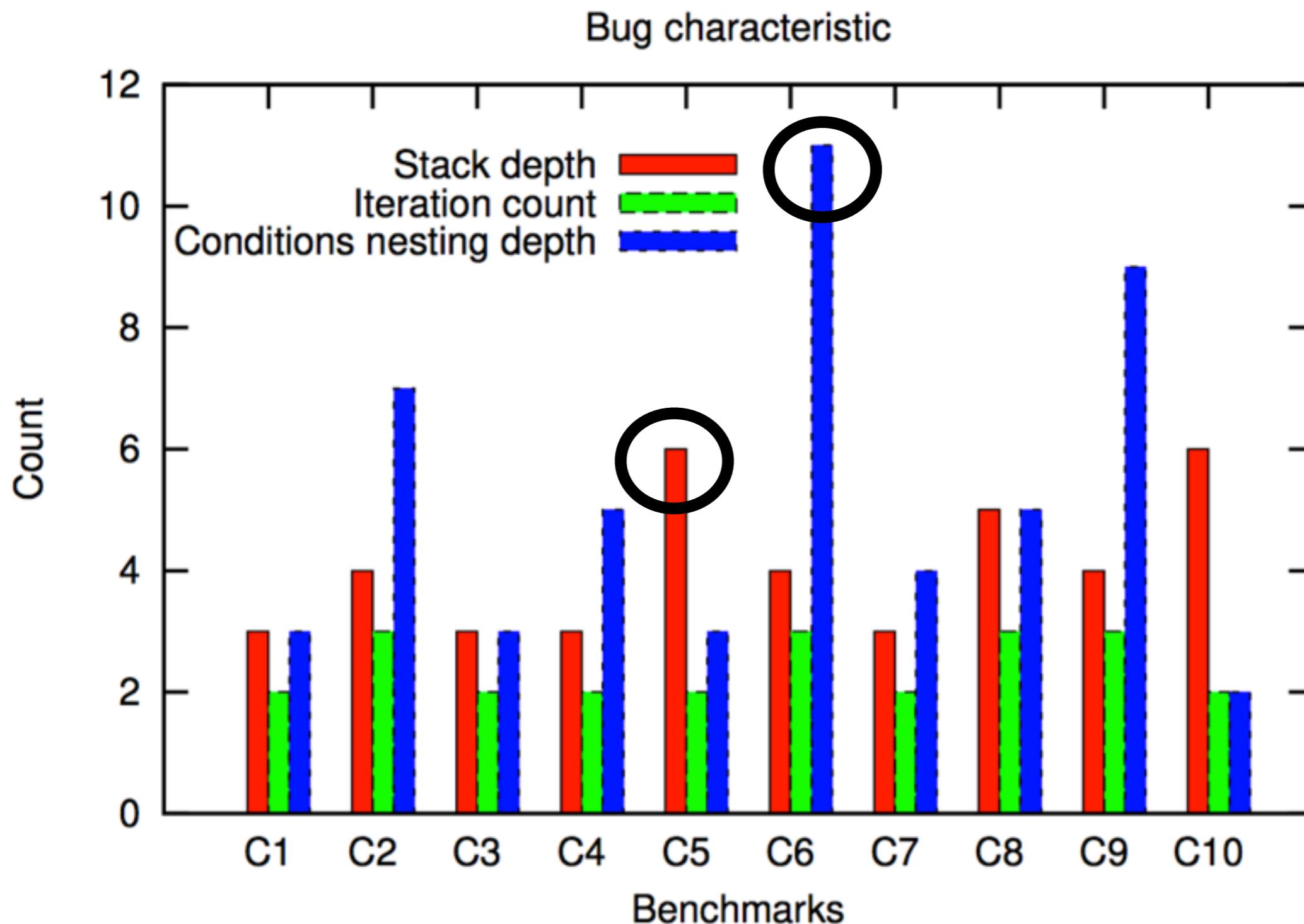
# Bug Characteristics

MINION detects complex concurrency issues



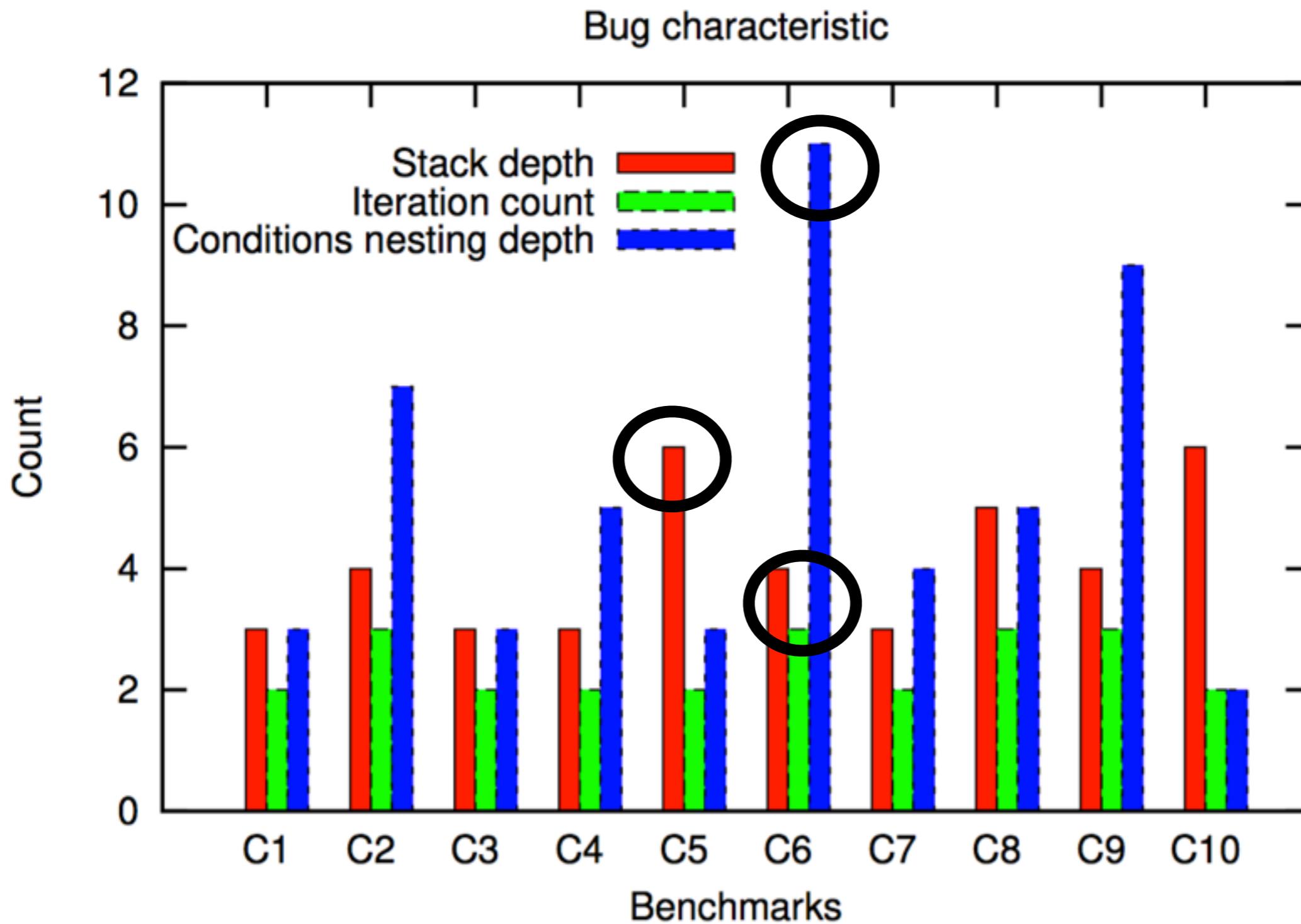
# Bug Characteristics

MINION detects complex concurrency issues



# Bug Characteristics

MINION detects complex concurrency issues



# Summary

- ◆ Designed a directed synthesis of failing concurrent executions
  - ◆ Integrates testing, symbolic execution and static analysis
- ◆ Validated on **10** well tested and popular Java classes
- ◆ Detected **31** crashes .
  - ◆ Resulted in fixes (includes classes in **JDK 8**)
  - ◆ Total time for analyzing all classes is approximately 23 minutes
  - ◆ Maximum nested path conditions: 11, stack depth: 6