# Directed Synthesis of Failing Concurrent Executions

**Malavika Samak***, **Omer Tripp+**, **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**

```java
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Building multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Building multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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**

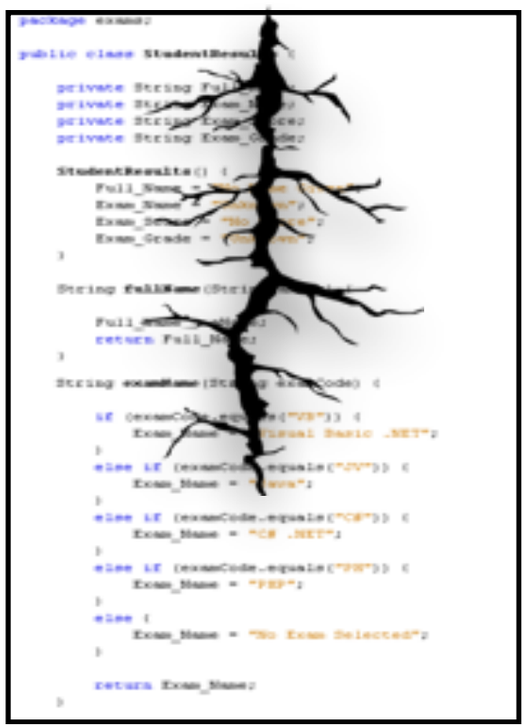
```
package exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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

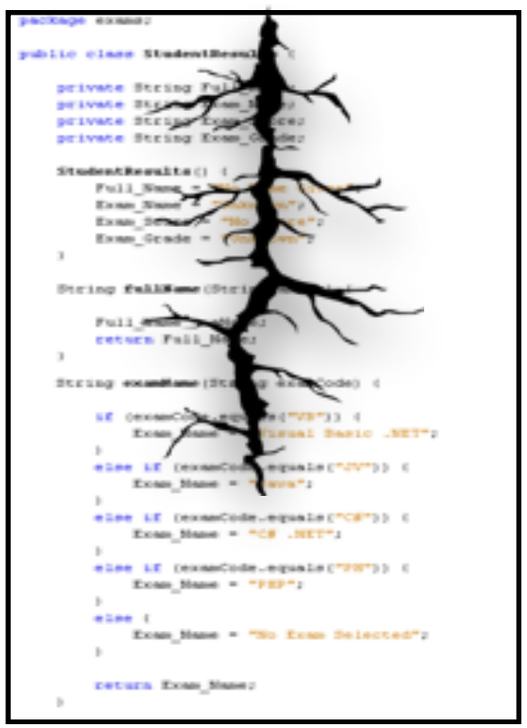
**Class under construction**



- **Functionality** ✗
- **Performance** ✓
- **Concurrency** ✗
- **Modularity** ✓
- **Data Structures** ✓
- **Shared state** ✓
- **Lock field correlation** ✓
- **…**

# Building multi-threaded libraries

**Class under construction**



- **Functionality** ✗
- **Performance** ✓
- **Concurrency** ✗
- **Modularity** ✓
- **Data Structures** ✓
- **Shared state** ✓
- **Lock field correlation** ✓
- **…**

Simultaneously handling the issues can introduce bugs

# Building multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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 exams;

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 sName) {

        Full_Name = sName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        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**

```java
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Testing multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Testing multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Testing multi-threaded libraries

**Class under construction**

```java
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

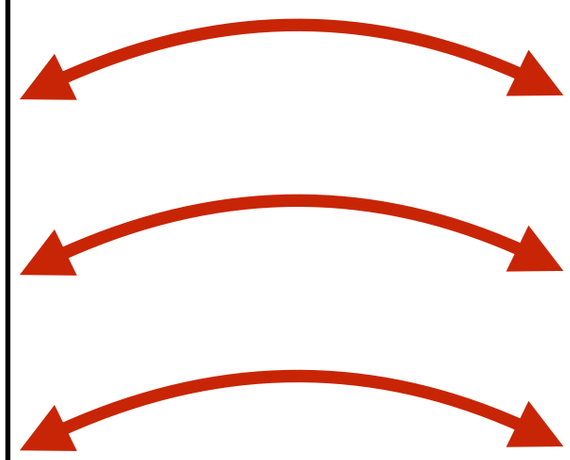
```java
class ExcTest
{
  static void genException()
  {
    int num[] = new int[4];
    System.out.println("Before exception is generated.");
    // generate an index out-of-bounds exception
    num[7] = 10;
    System.out.println("this wos't be displayed");
  }
}
class StackTraceBubble
{
   static void myMethod()
   {
      try
      {
         ExcTest.genException();
      }
      catch(ArrayIndexOutOfBoundsException exc)
      {
         exc.printStackTrace();
      }
   }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
         System.out.println("Standard message is: ");
         System.out.println(exc);
         System.out.println("\nStack trace: ");
         exc.printStackTrace();
      }
    System.out.println("After catch statement.");
  }
}
```

**Client**

# Testing multi-threaded libraries

**Class under construction**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```



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

**Client**

# Testing multi-threaded libraries

- Invokes the APIs provided by the class.

**Class u**
**constru**

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

# Testing multi-threaded libraries

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

**Class u**
**constru**

**Client**

# Testing multi-threaded libraries

**Class u**
**constru**

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

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

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

**Client**

# Testing multi-threaded libraries

```
class ExcTest
{
  static void genException()
  {
    int nums[] = new int[4];
    System.out.println("Before exception is generated.");
    // generate an index out-of-bounds exception
    nums[7] = 10;
    System.out.println("this won't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

# Testing multi-threaded libraries

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

# Testing multi-threaded libraries

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

# Testing multi-threaded libraries

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

**Challenges**

# Testing multi-threaded libraries

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

How many threads need to be created?

**Challenges**

# Testing multi-threaded libraries

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

How many threads need to be created?

What methods need to be invoked?

**Challenges**

# Testing multi-threaded libraries

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

**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

```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**Client**

**Developer**

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?

**Challenges**

# Client synthesis for multi-threaded libraries

# Client synthesis for multi-threaded libraries

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

**Class under test**

# Client synthesis for multi-threaded libraries

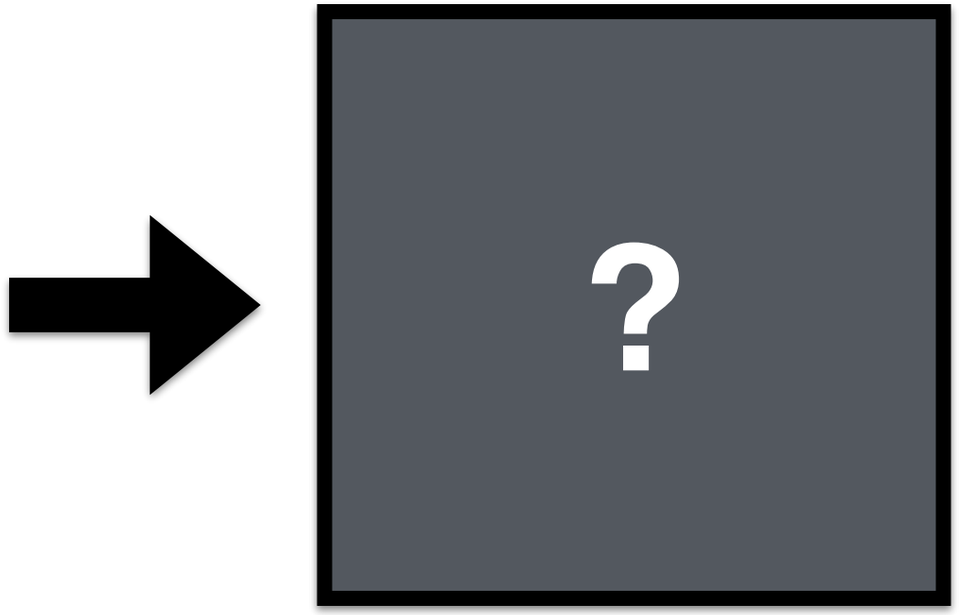
```
package exams;

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 aName) {

        Full_Name = aName;
        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("CS")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

**Class under test**

# Client synthesis for multi-threaded libraries

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

**Class under test**



# Client synthesis for multi-threaded libraries

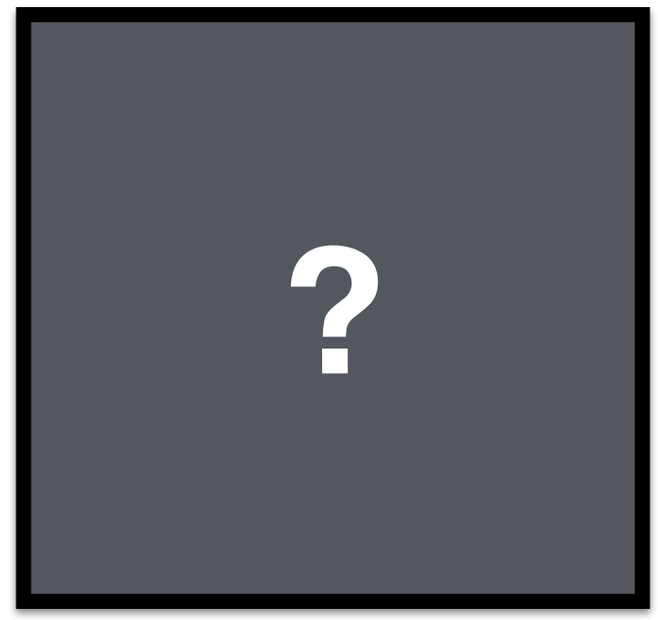
```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

**Class under test**

# Client synthesis for multi-threaded libraries

```
package exams;

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 aName) {

        Full_Name = aName;
        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("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

**Class under test**




```
class ExcTest
{
  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 wos't be displayed");
  }
}
class StackTraceBubble
{
  static void myMethod()
  {
    try
    {
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      exc.printStackTrace();
    }
  }
  public static void main(String[] args)
  {
    try
    {
      StackTraceBubble.myMethod();
      ExcTest.genException();
    }
    catch(ArrayIndexOutOfBoundsException exc)
    {
      // catch the exception
        System.out.println("Standard message is: ");
        System.out.println(exc);
        System.out.println("\nStack trace: ");
        exc.printStackTrace();
    }
    System.out.println("After catch statement.");
  }
}
```

**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 →

Samak, Ramanathan, Jagannathan PLDI'15 →

Samak and Ramanathan, FSE'15 →

**Dynamic Analysis Engine**

→ Deadlocks

→ Data races

→ Atomicity violations

# Story of the black box so far…

**Randomized generation**

Pradel and Gross, PLDI'12 → Crashes and Deadlocks

Need a targeted approach to reveal crashes

**Targeted ge**

Samak and Ramanathan, OOPSLA'14 → Deadlocks

Samak, Ramanathan, Jagannathan PLDI'15 → Data races

Samak and Ramanathan, FSE'15 → Atomicity violations

**Dynamic Analysis Engine**

# 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

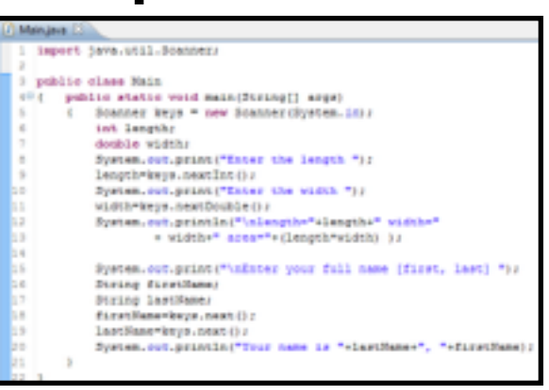
# Class under test

```java
package exams;

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 aName) {

        Full_Name = aName;
        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("CS")) {
            Exam_Name = "C# .NET";
        }
        else if (examCode.equals("PH")) {
            Exam_Name = "PHP";
        }
        else {
            Exam_Name = "No Exam Selected";
        }

        return Exam_Name;
    }
```

# Class under test

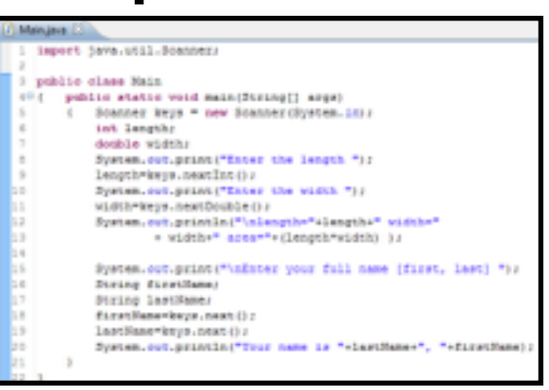

## Class under test


## Random sequential test



## Class under test

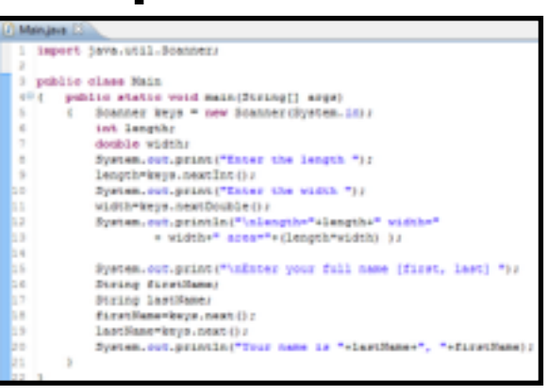

## Random sequential test



## Multi-threaded client

## Class under test

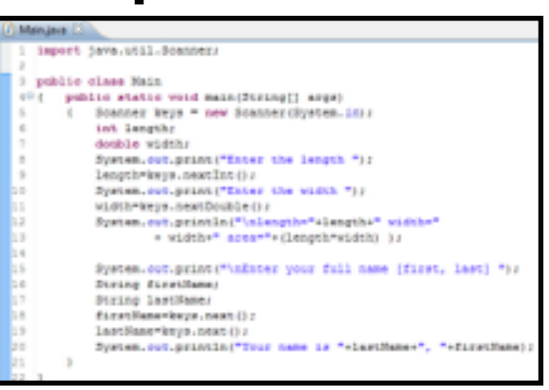

## Random sequential test



## Multi-threaded client

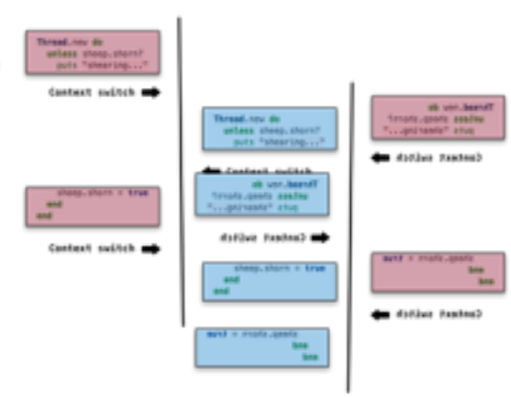
## Class under test


## Random sequential test



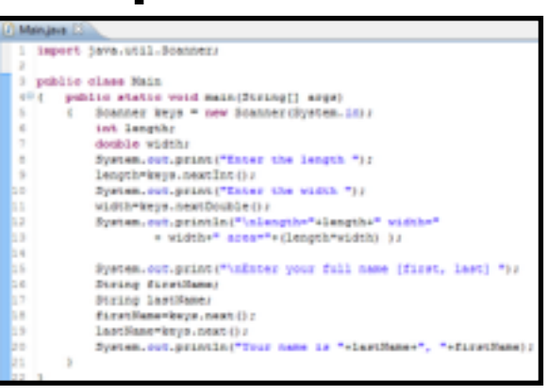
## Multi-threaded client

## Thread interleaving



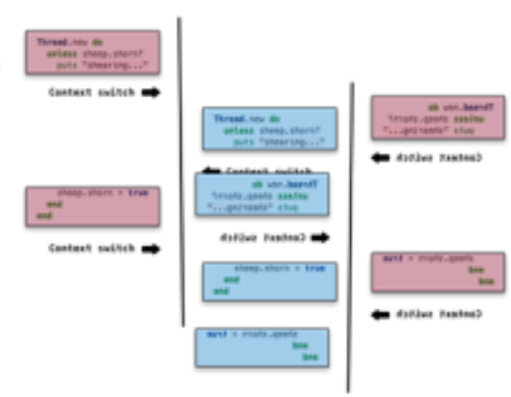
## Class under test


## Random sequential test



## Multi-threaded client

## Thread interleaving



Crash

# Example from JDK 8

```
106  public int read(char cbuf[], int off, int len)
                                        throws IOException {
107    synchronized (lock) {
108      ensureOpen();
109      try {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          System.arraycopy(buf, pos, cbuf, off, avail);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          System.arraycopy(buf, pos, cbuf, off, avail);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122            assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**PushbackReader.java**

# Example from JDK 8

```
public int read(char cbuf[], int off, int len)
                                    throws IOException {
  synchronized (lock) {
    ensureOpen();
    try {
                    // returns on len <= 0
      int avail = buf.length - pos;
      if (avail > 0) {
        if (len < avail)
          avail = len;
          assert(buf != null);
          ...
      }
      ...
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexOutOfBoundsException();
    }
  }
}
```

**PushbackReader.java**

# Example from JDK 8

```
public int read(char cbuf[], int off, int len)
                                    throws IOException {
  synchronized (lock) {
    ensureOpen();
    try {
              // returns on len <= 0
      int avail = assert(buf != null);
      if (avail > 0) {
        if (len < avail)
          avail = len;
          assert(buf != null);
          ...
      }
      ...
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexOutOfBoundsException();
    }
  }
}
```

**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 {
             ... // returns on len <= 0
118        int avail = assert(buf != null);
119        if (avail > 0) {
120         if (len < avail)
121           avail = len;
122         assert(buf != null);
            ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = assert(buf != null);
119        if (avail > 0) {
120         if (len < avail)
121           avail = len;
122         assert(buf != null);
            ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**PushbackReader.java**

# Example from JDK 8

```
public int read(char cbuf[], int off, int len)
                                    throws IOException {
  synchronized (lock) {
    ensureOpen();
    try {
        ... // returns on len <= 0
      int avail = assert(buf != null);
      if (avail > 0) {
        if (len < avail)
         avail = len;
        assert(buf != null);
        ...
      }
      ...
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexOutOfBoundsException();
    }
  }
}
```

Fail this assert!

**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 {
           ... // returns on len <= 0
118        int avail = assert(buf != null);
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136        throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

This assert holds!

Fail this assert!

**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 {
             ... // returns on len <= 0
118        int avail = assert(buf != null);
119        if (avail > 0) {
120         if (len < avail)
121           avail = len;
122         assert(buf != null);
            ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**PushbackReader.java**

# Example from JDK 8

```
public int read(char cbuf[], int off, int len)
                                    throws IOException {
  synchronized (lock) {
    ensureOpen();
    try {
        ... // returns on len <= 0
      int avail =
      if (avail > 0) {
       if (len < avail)
        avail = len;
        ...
      }
      ...
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new IndexOutOfBoundsException();
    }
  }
}
```

**buf != null**

**assert(buf != null);**

**assert(buf != null);**

**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 {
             ... // returns on len <= 0
118        int avail = assert(buf != null);
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**buf != null**

**buf == null**

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122        assert(buf != null);
           ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
120          if (len < avail)
121            avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

Evaluate to true

**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 {
             ... // returns on len <= 0
118        int avail = buf.length - pos;
119        if (avail > 0) {
           if (len < avail)
             avail = len;
122          assert(buf != null);
             ...
126        }
           ...
135      } catch (ArrayIndexOutOfBoundsException e) {
136          throw new IndexOutOfBoundsException();
137      }
138    }
139  }
```

Evaluate to true

Don’t care

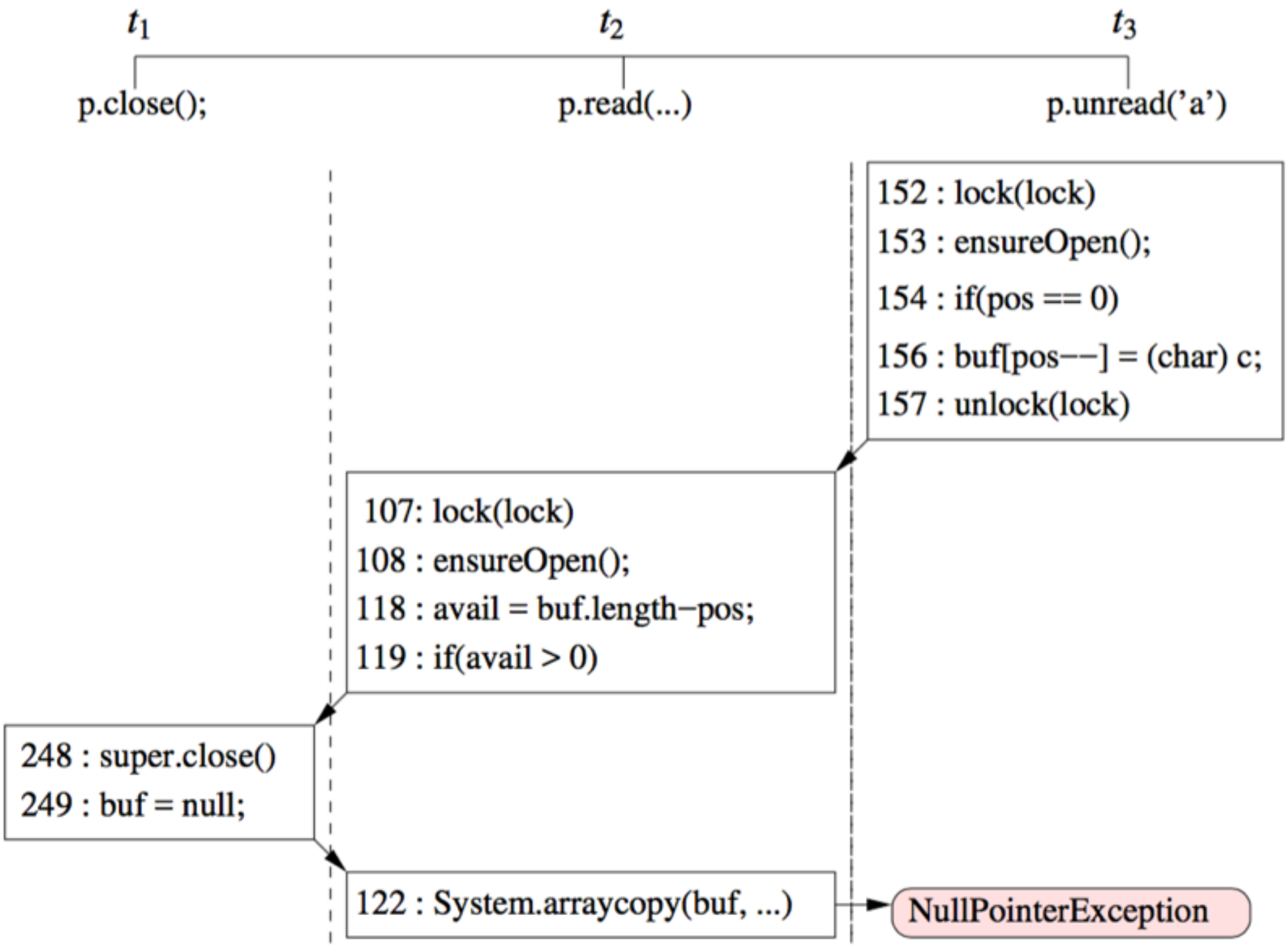
**PushbackReader.java**

# Example from JDK 8

```
public int read(char cbuf[], int off, int len)
                                        throws IOException {
    synchronized (lock) {
        ensureOpen();
        try {
            ... // returns on len <= 0
          int avail = buf.length - pos;
          if (avail > 0) {
           if (len < avail)
            avail = len;
            assert(buf != null);
            ...
          }
          ...
        } catch (ArrayIndexOutOfBoundsException e) {
            throw new IndexOutOfBoundsException();
        }
    }
}
```

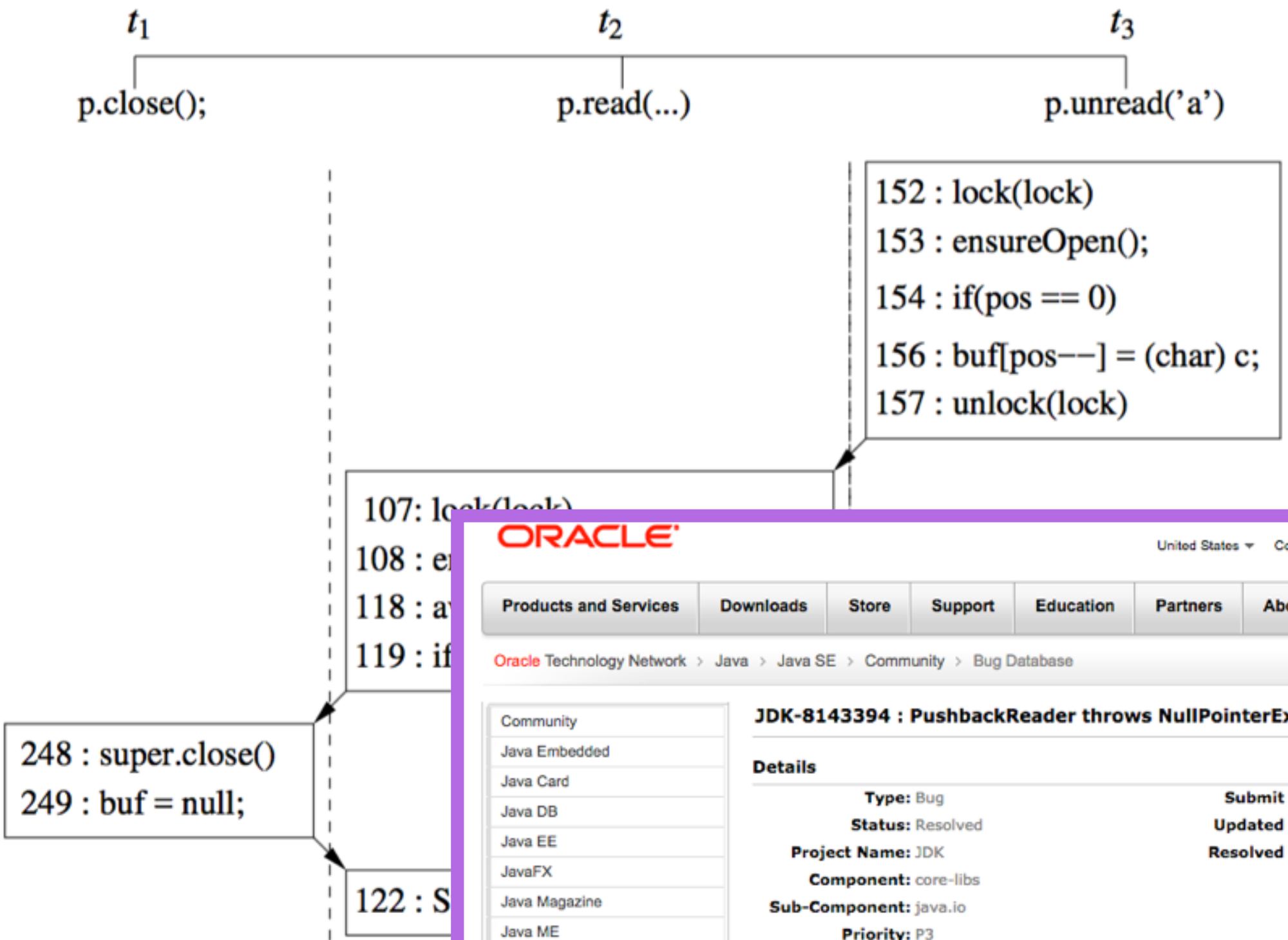
**PushbackReader.java**

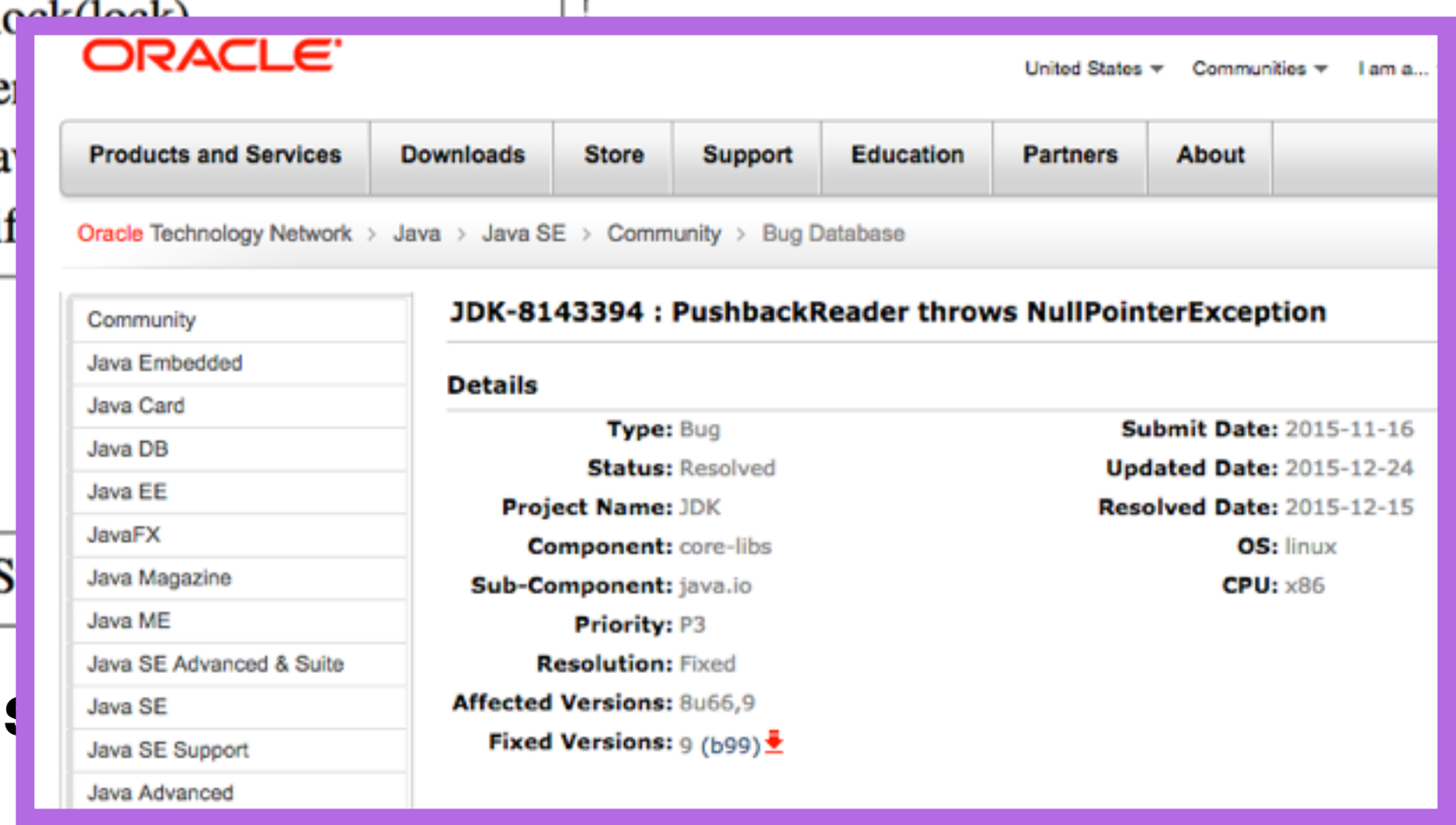
# Crash found by Minion in JDK8




**PushbackReader.java**

# Crash found by Minion in JDK8

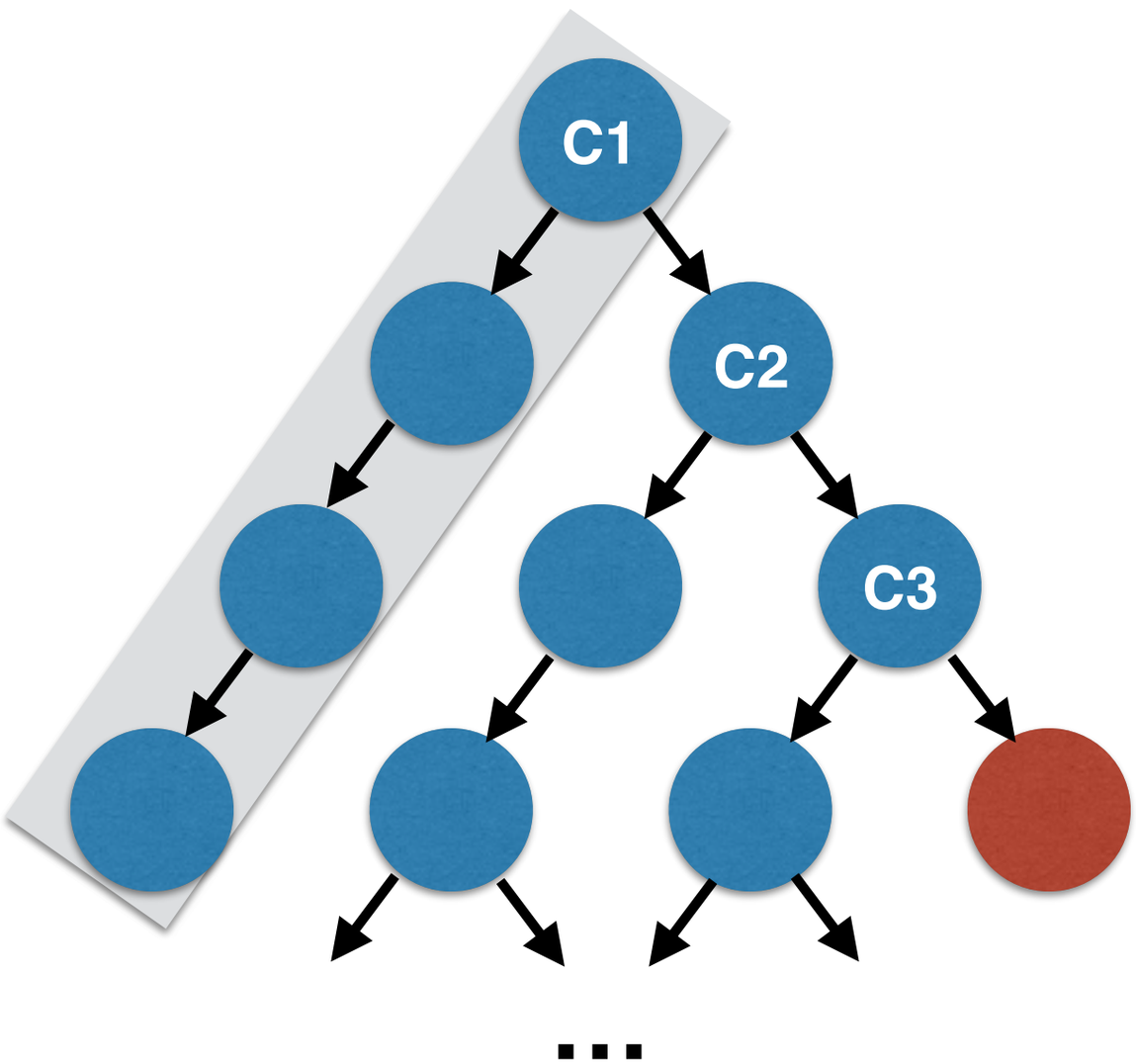


# Execution synthesis

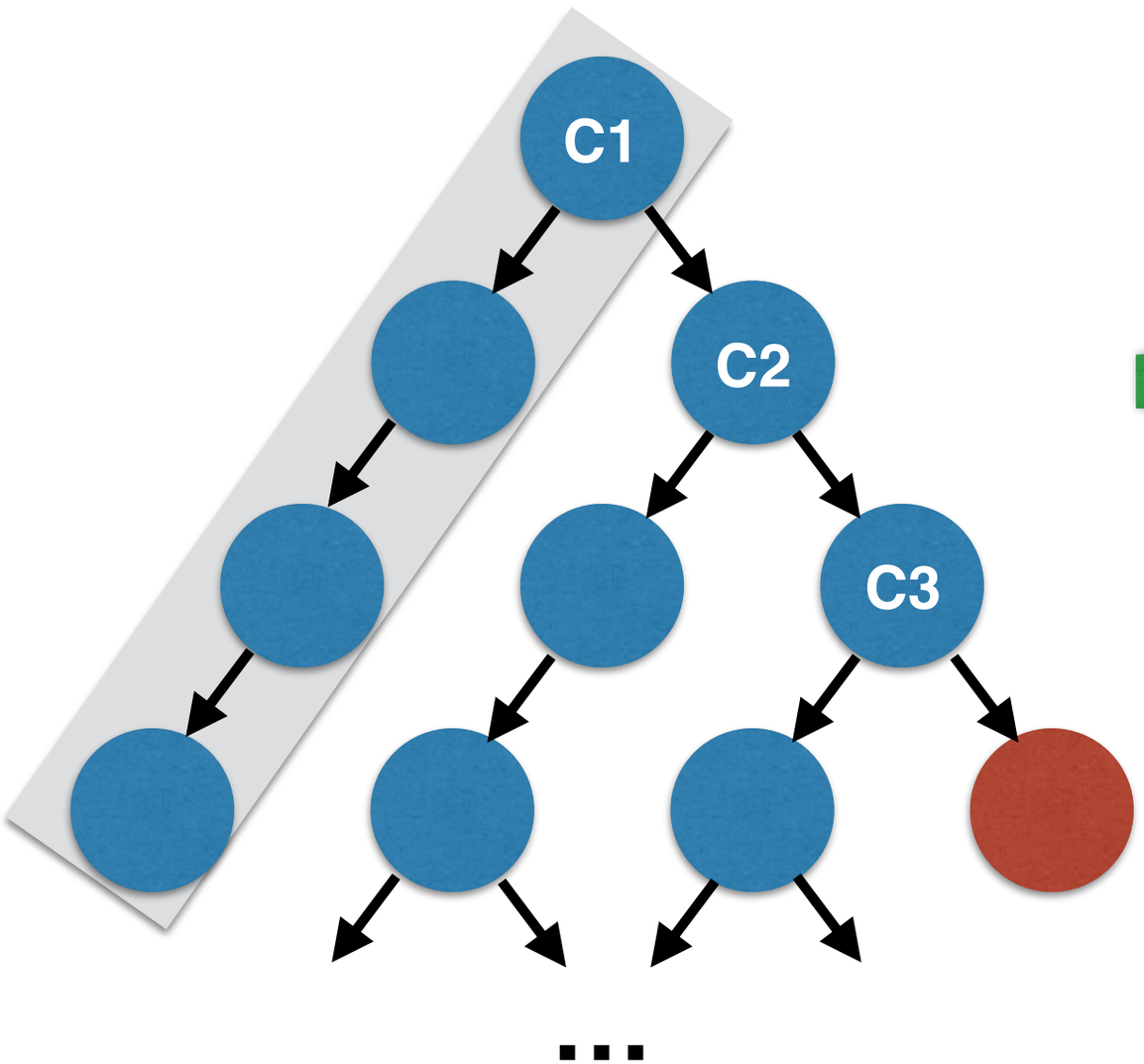
# Execution synthesis

Random execution



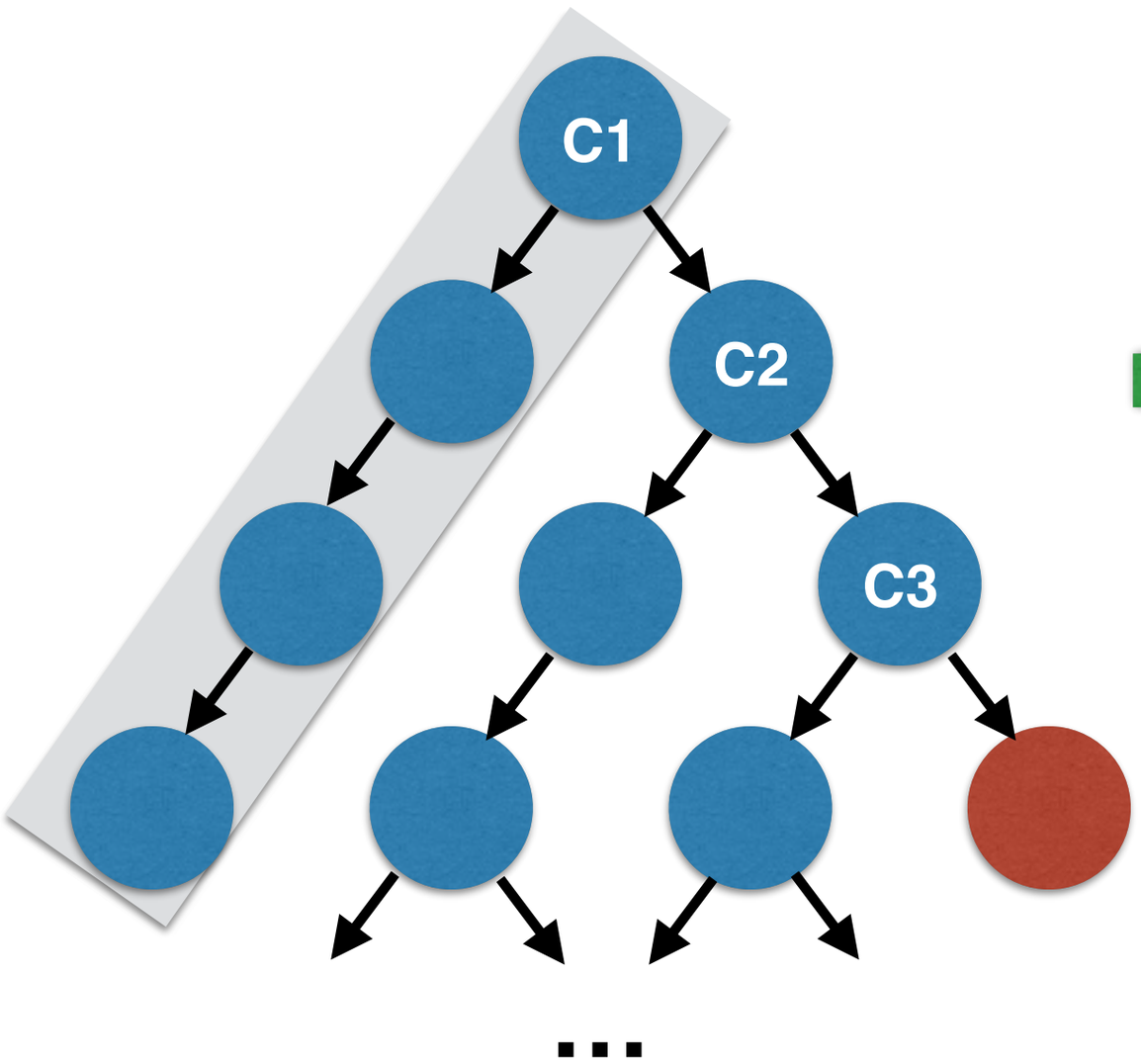
# Execution synthesis

Random execution

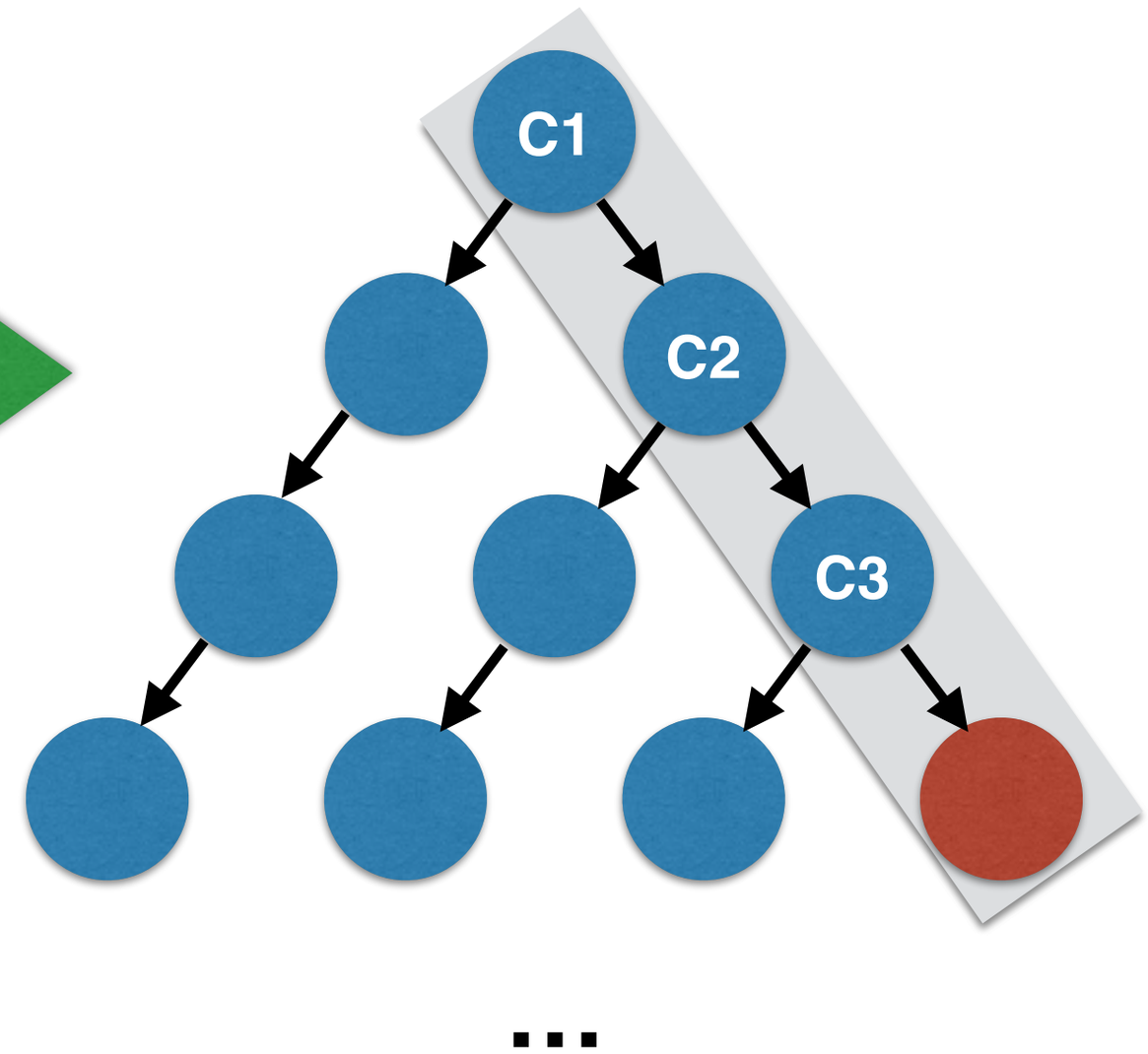


# Execution synthesis

Random execution

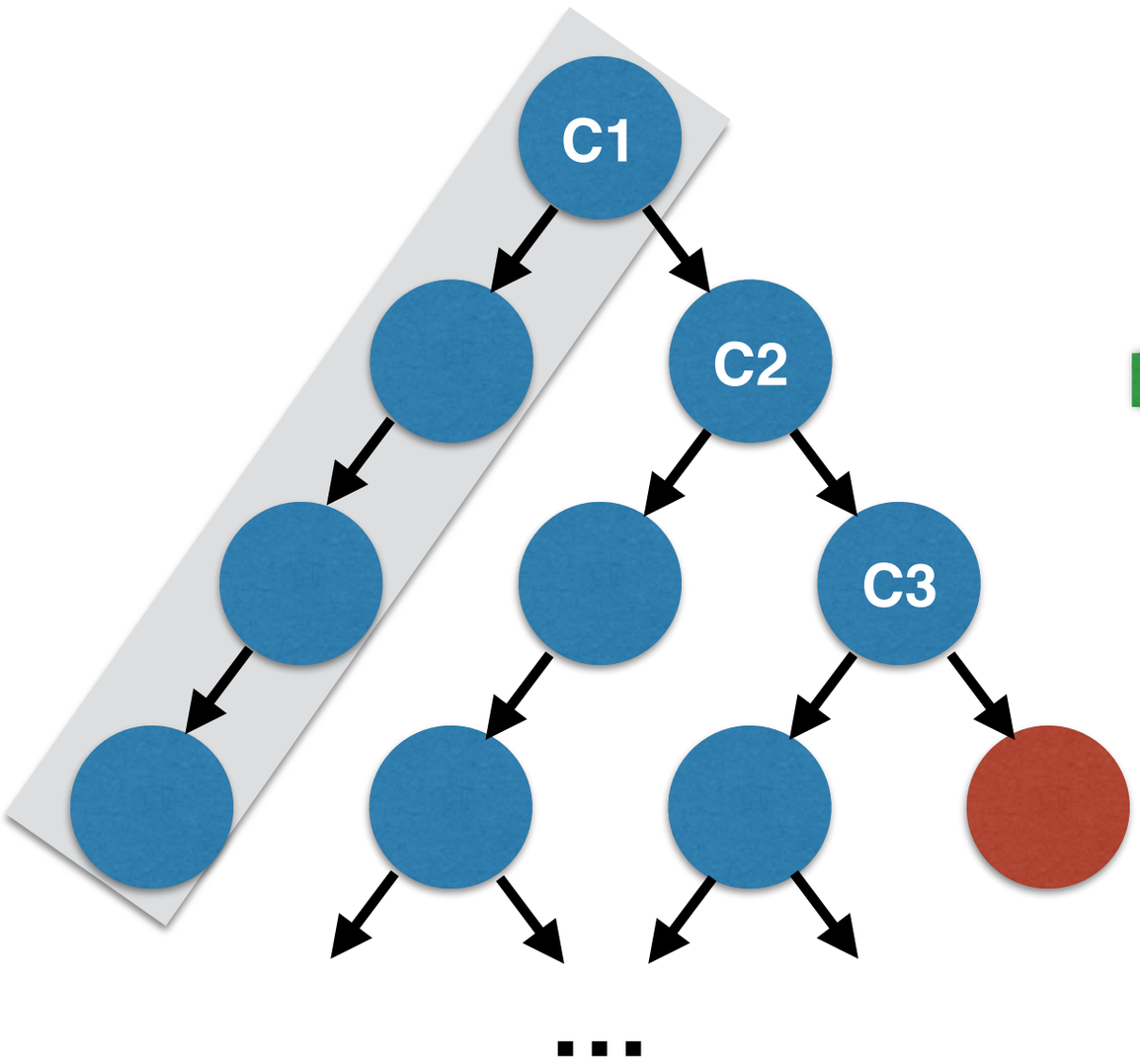



Target execution

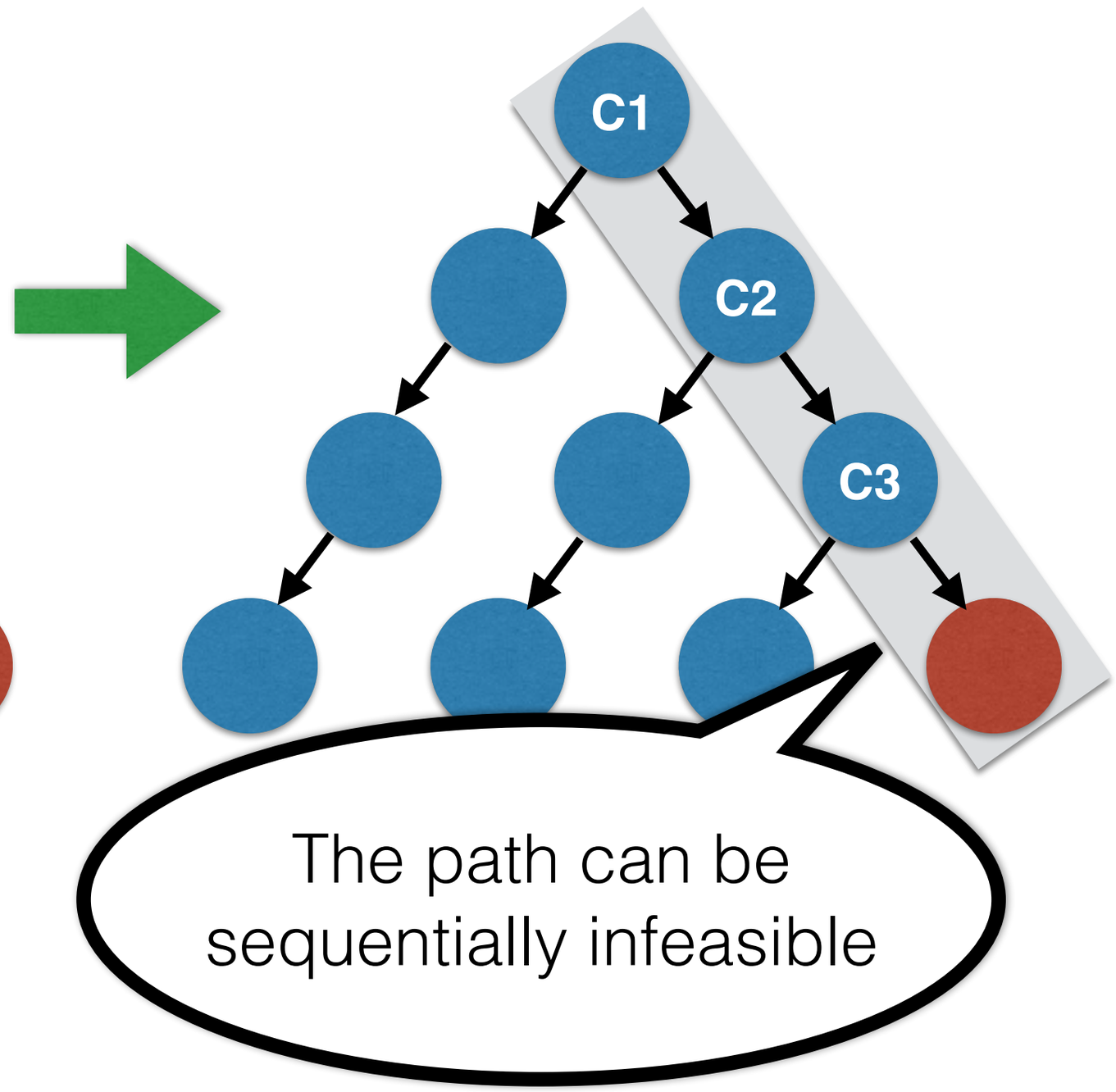


# Execution synthesis

Random execution




Target execution



# Execution synthesis

**$t_1$: m3** **$t_2$: m1** **$t_3$: m2**

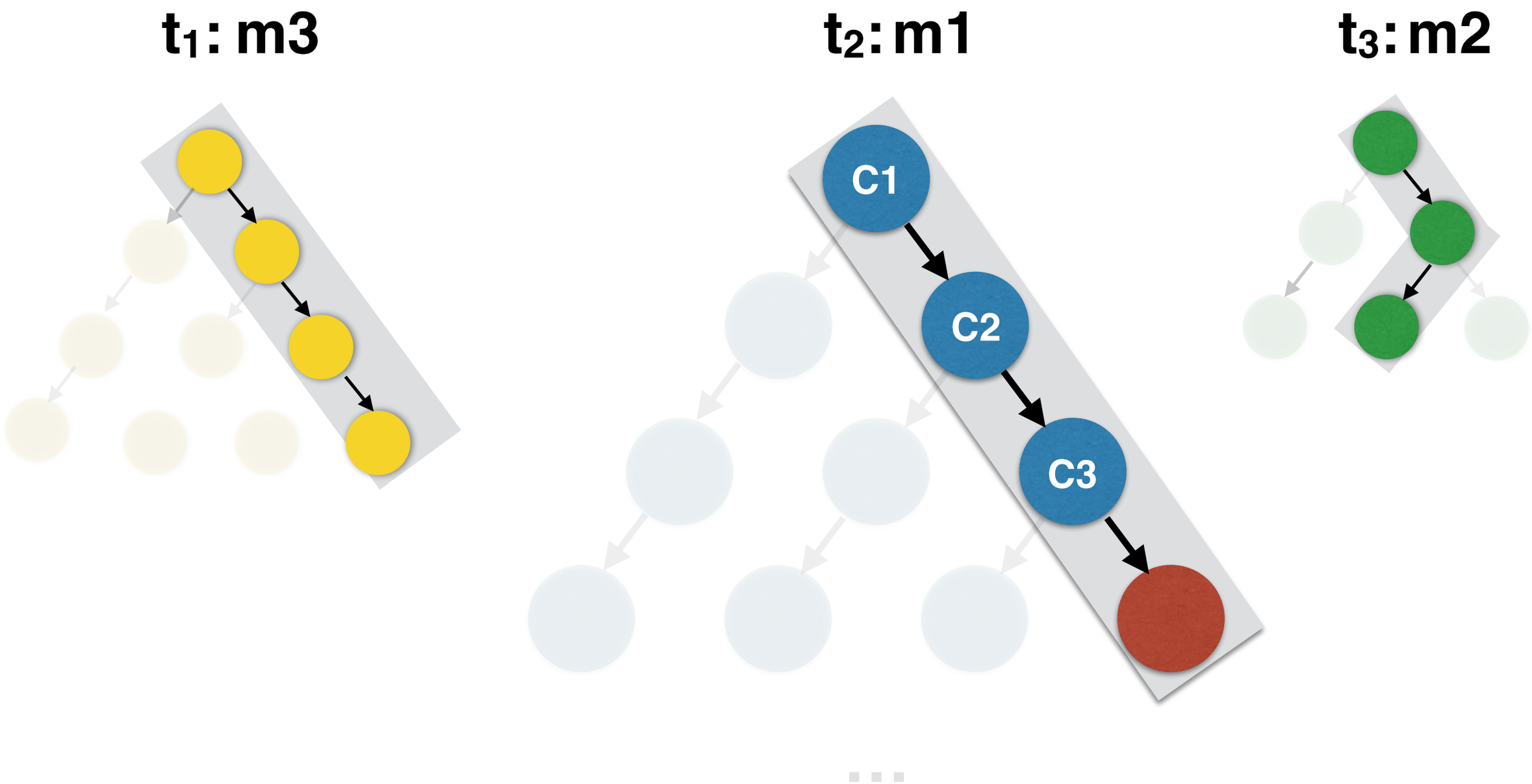
# Execution synthesis

- Methods invoked from different threads appropriately

**$t_1$: m3** **$t_2$: m1** **$t_3$: m2**

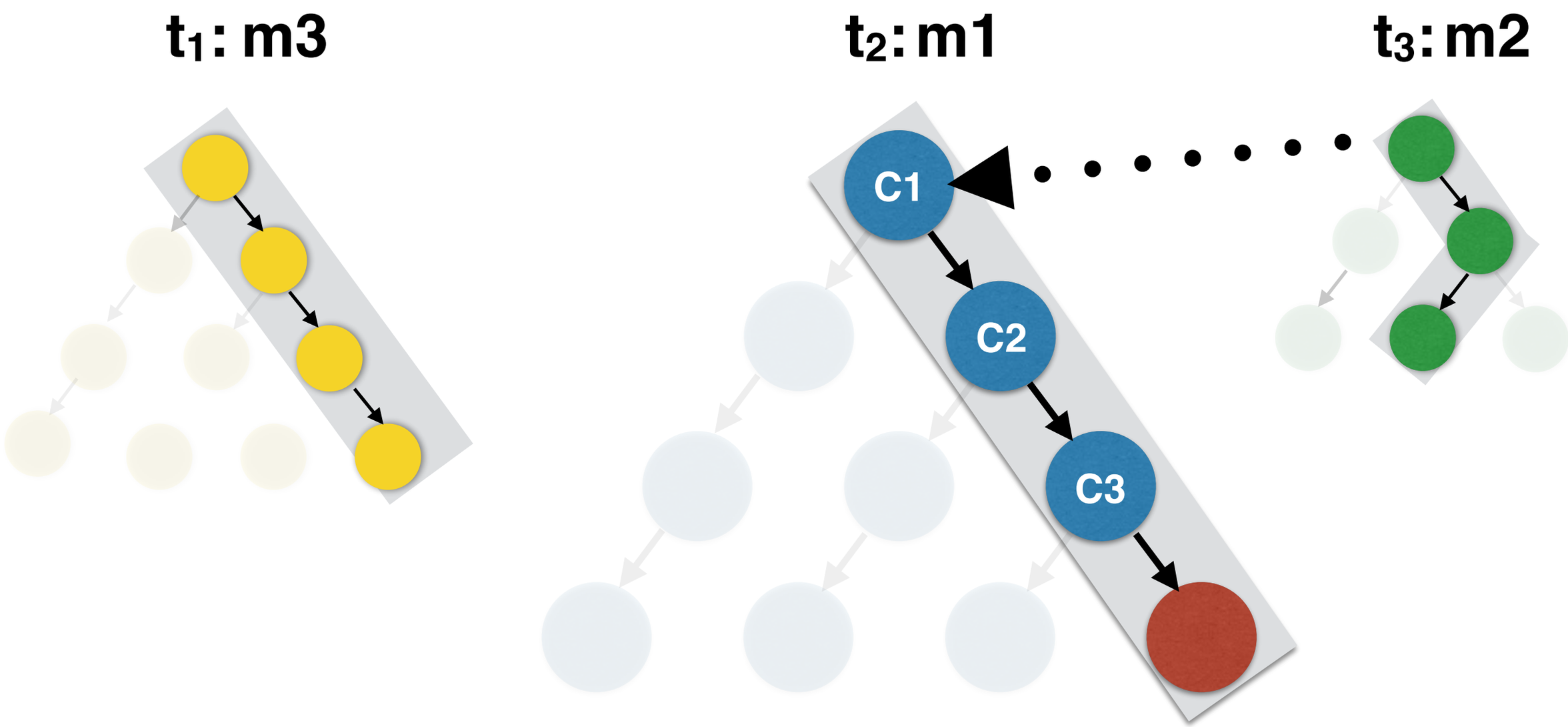
# Execution synthesis

- Methods invoked from different threads appropriately



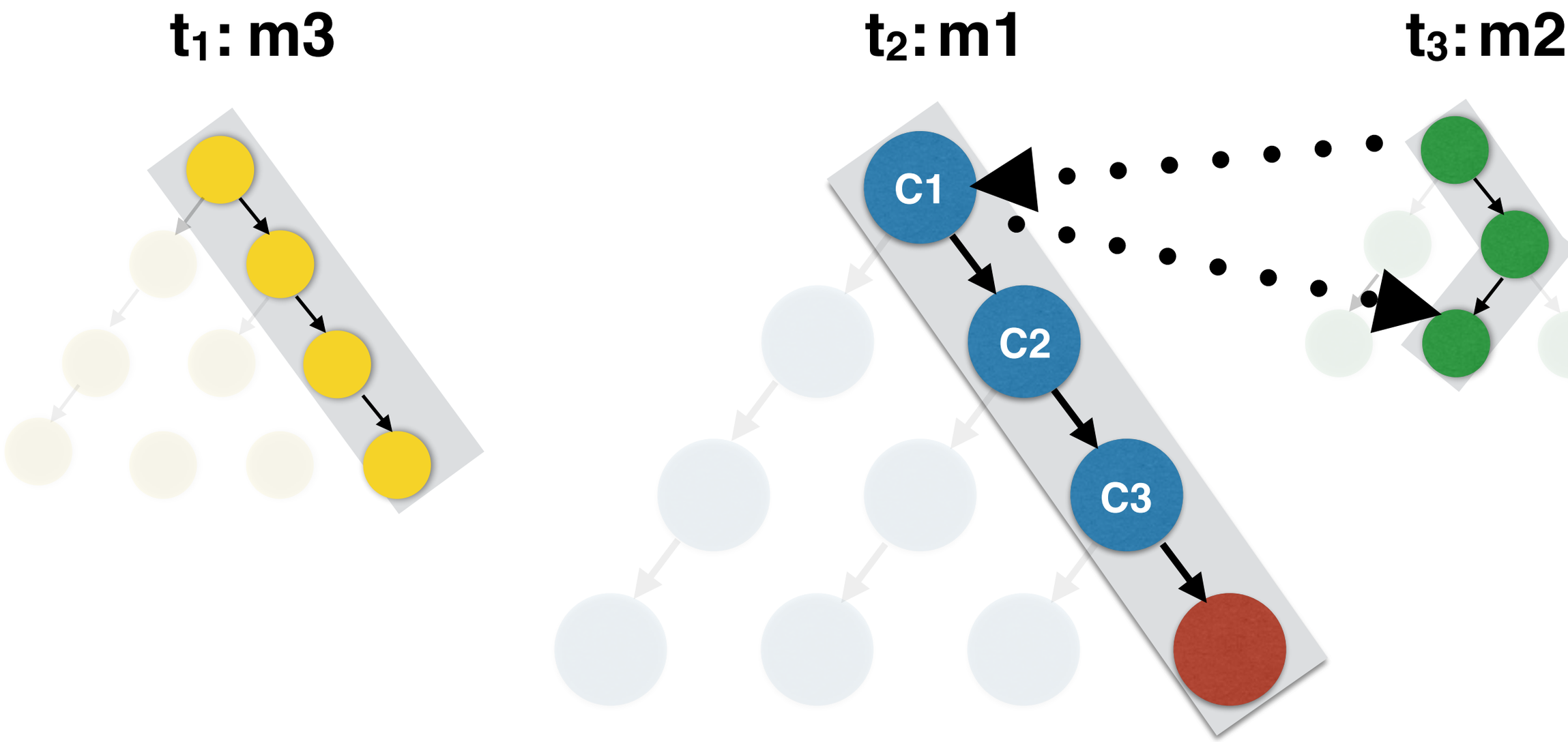
# Execution synthesis

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



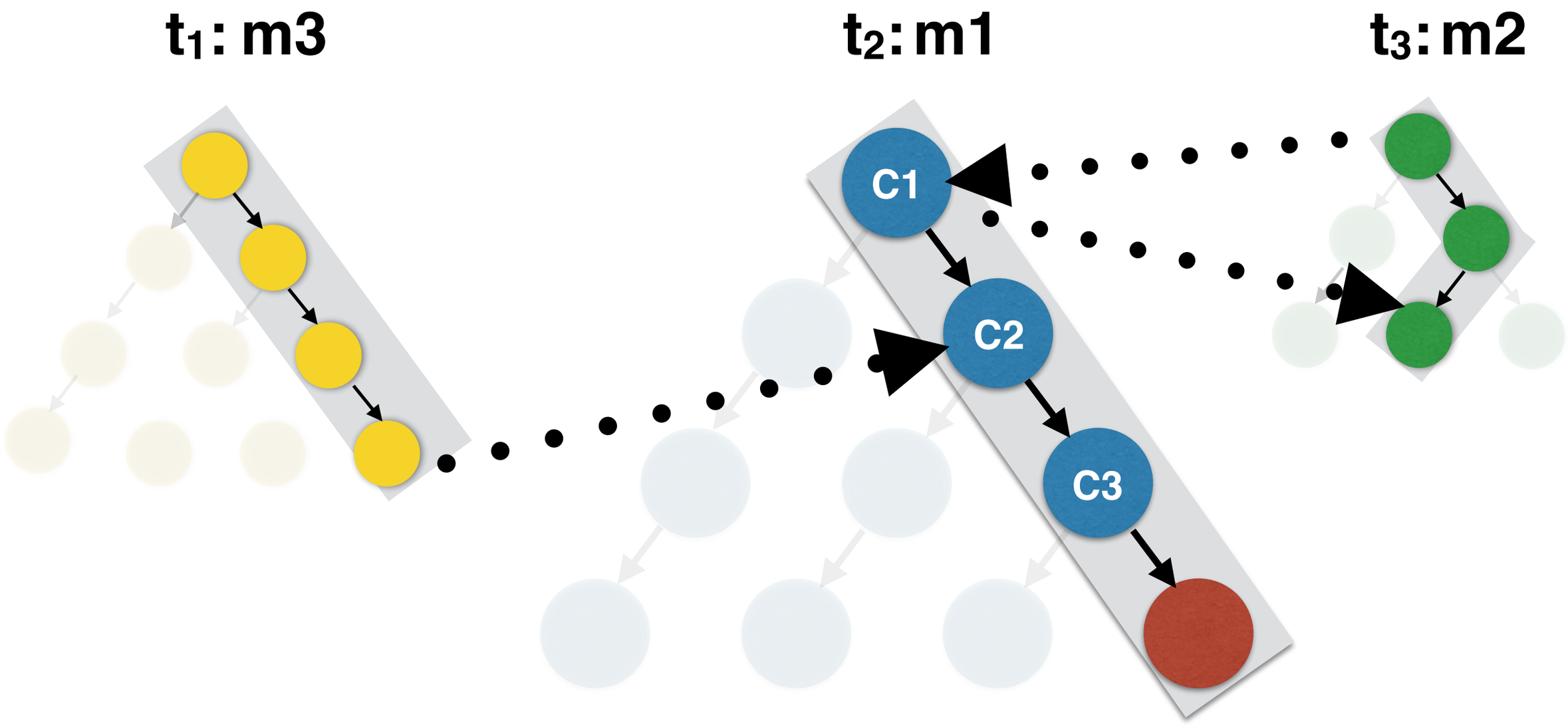
# Execution synthesis

- 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



# 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

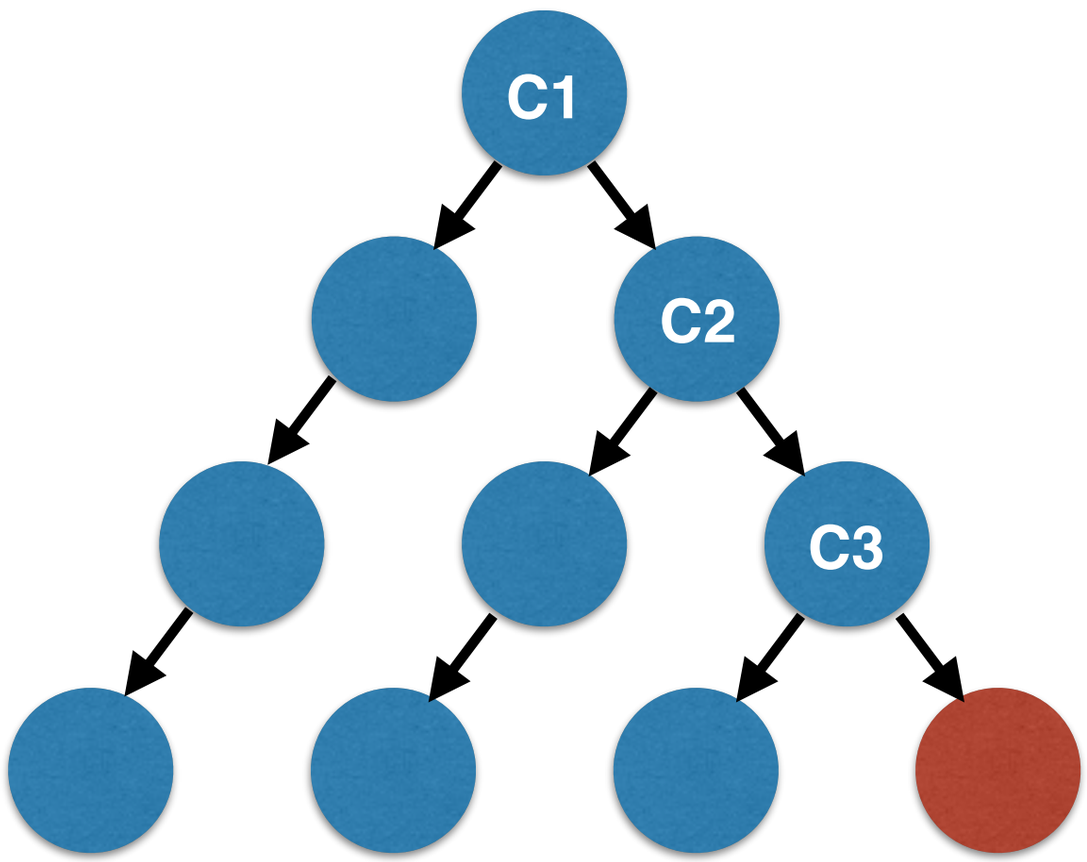
- **Static analysis**
  - targets: locate assertions to violate and updates to fields
  - derive path conditions to reach target instructions
- **Dynamic analysis**
  - obtain concrete data by executing provided tests
    - path conditions traversed, value of fields, etc
- **Constraint solvers**
  -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

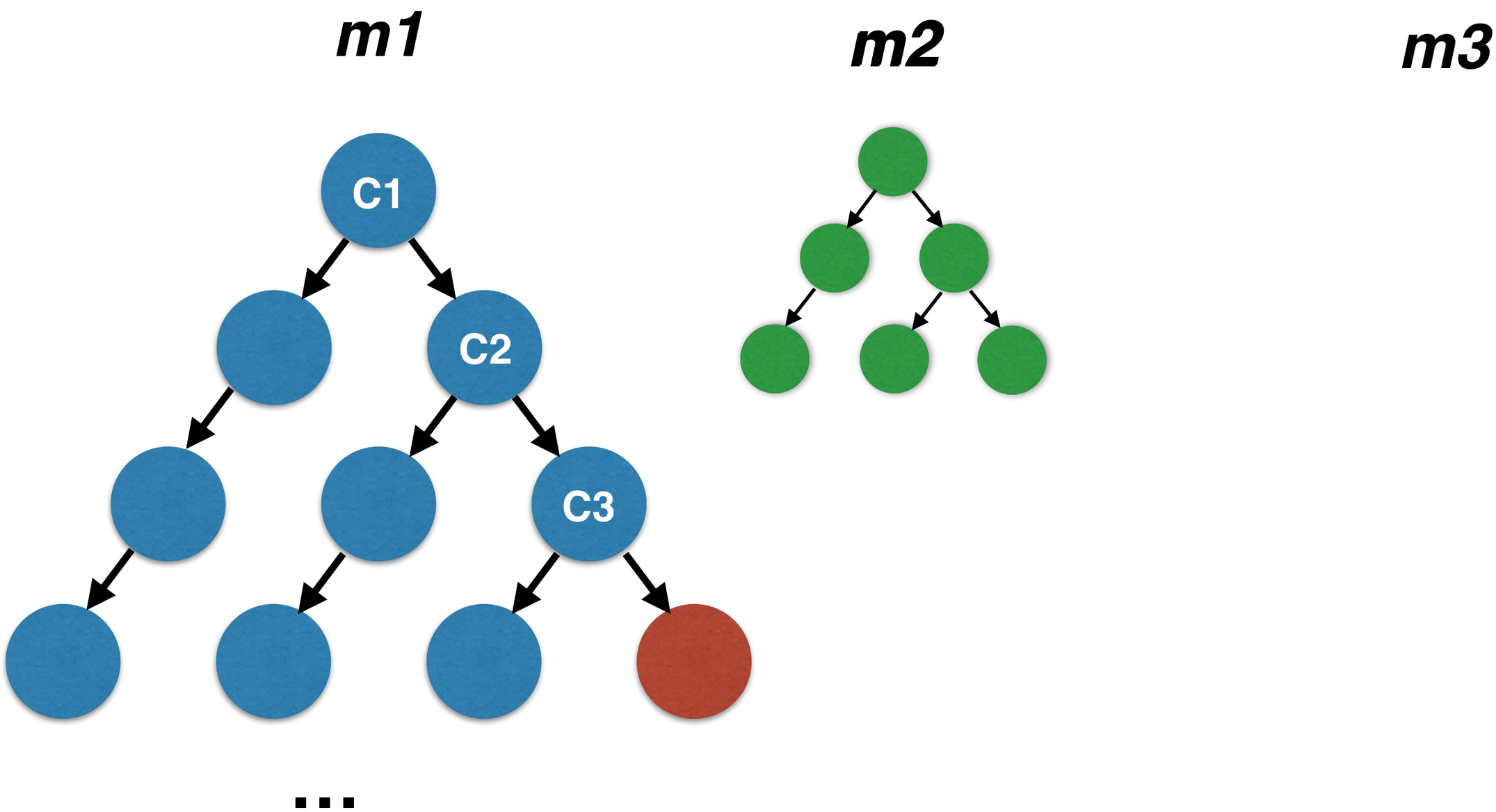
# Iteration - 1 : Static analysis

***m1*** ***m2*** ***m3***

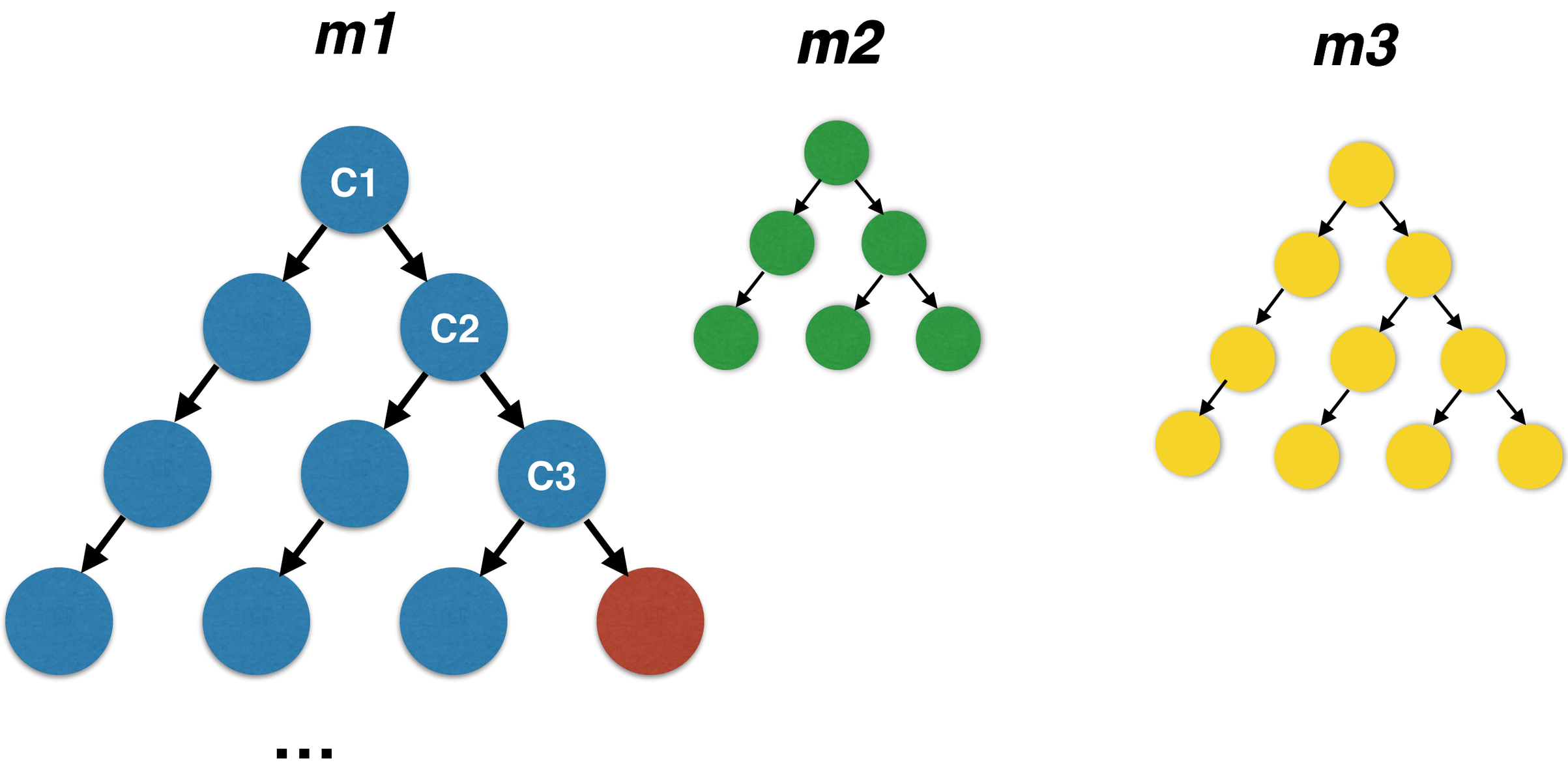



...

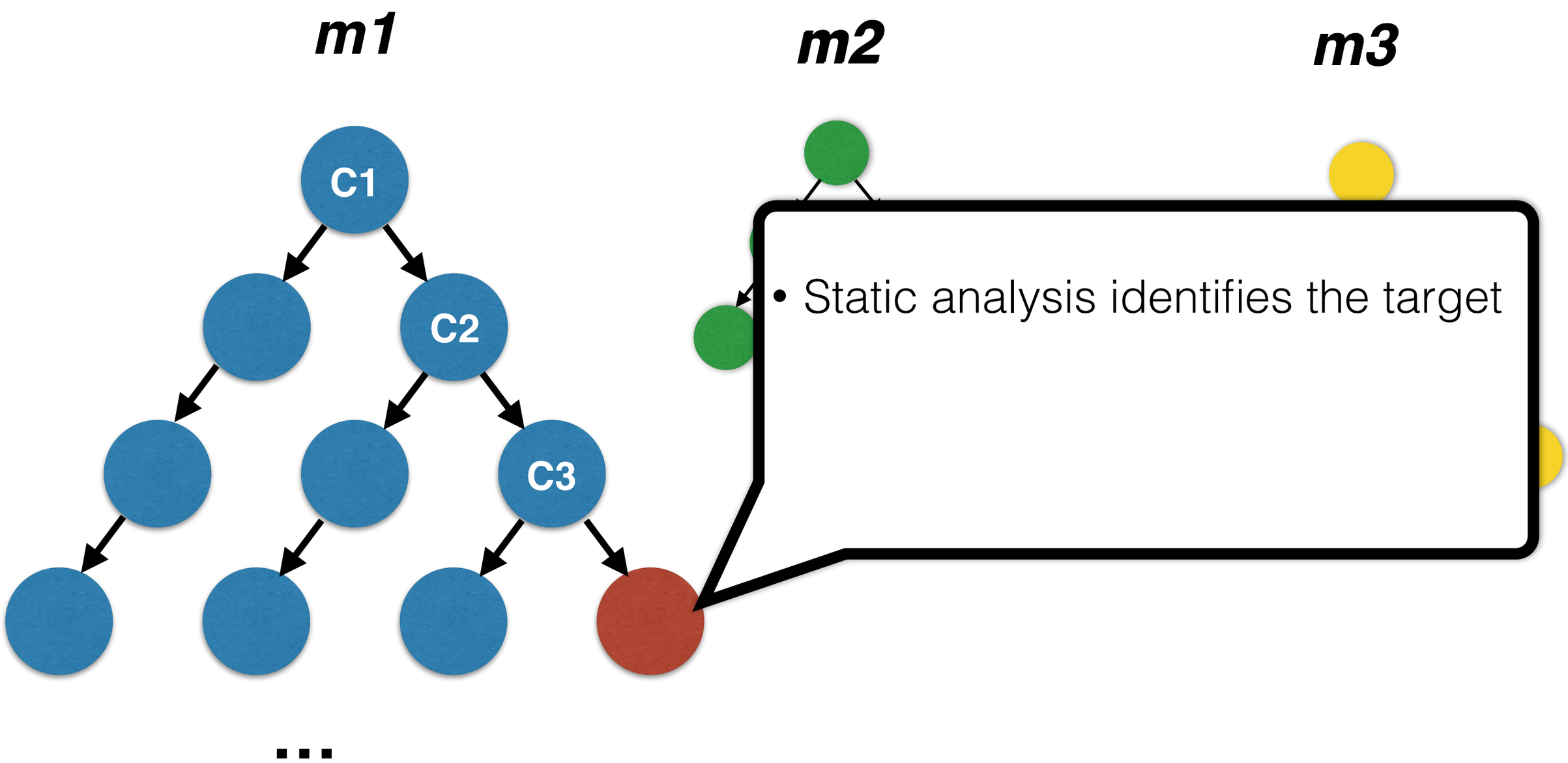
# Iteration - 1 : Static analysis



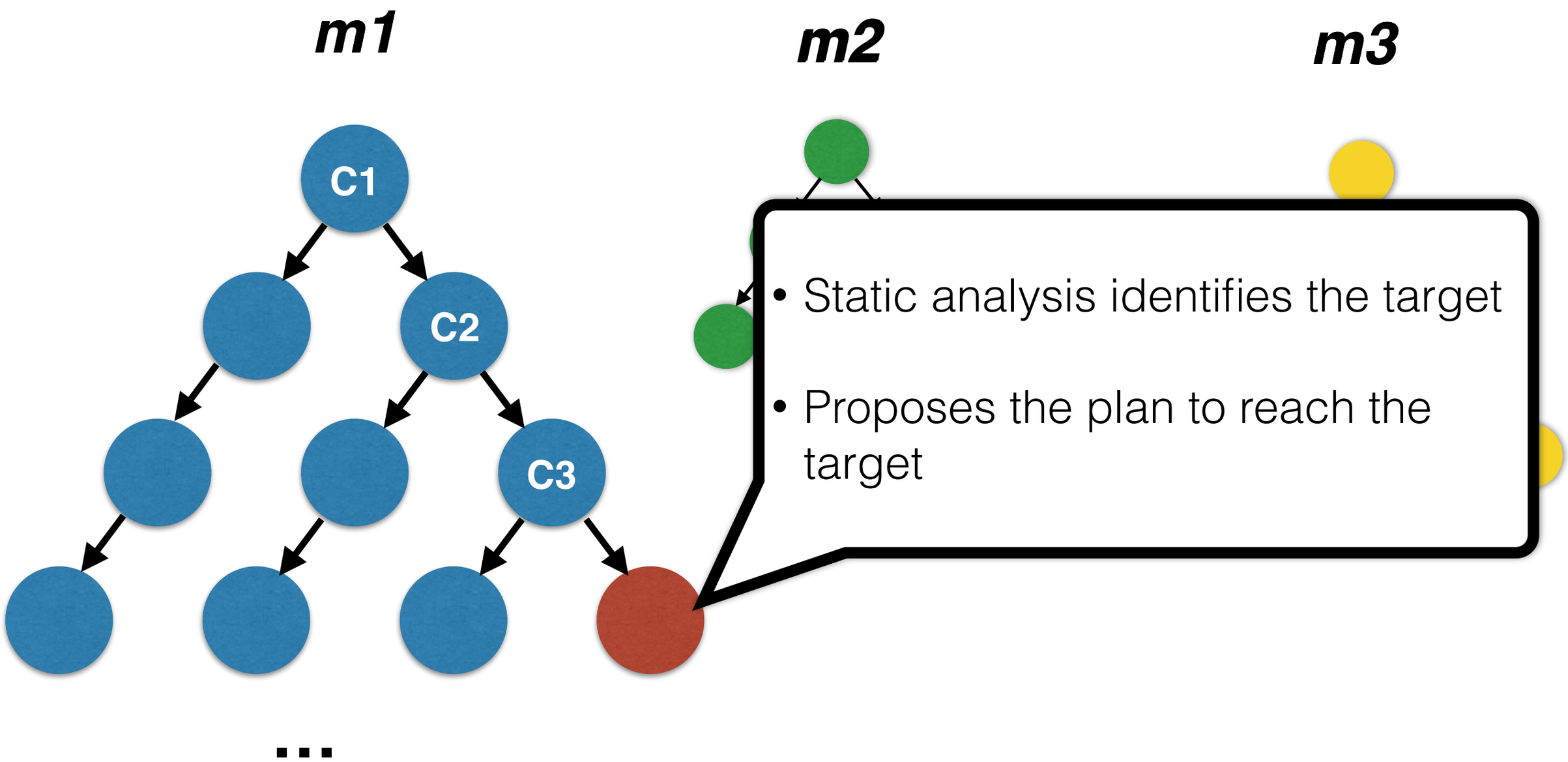
# Iteration - 1 : Static analysis



# Iteration - 1 : Static analysis

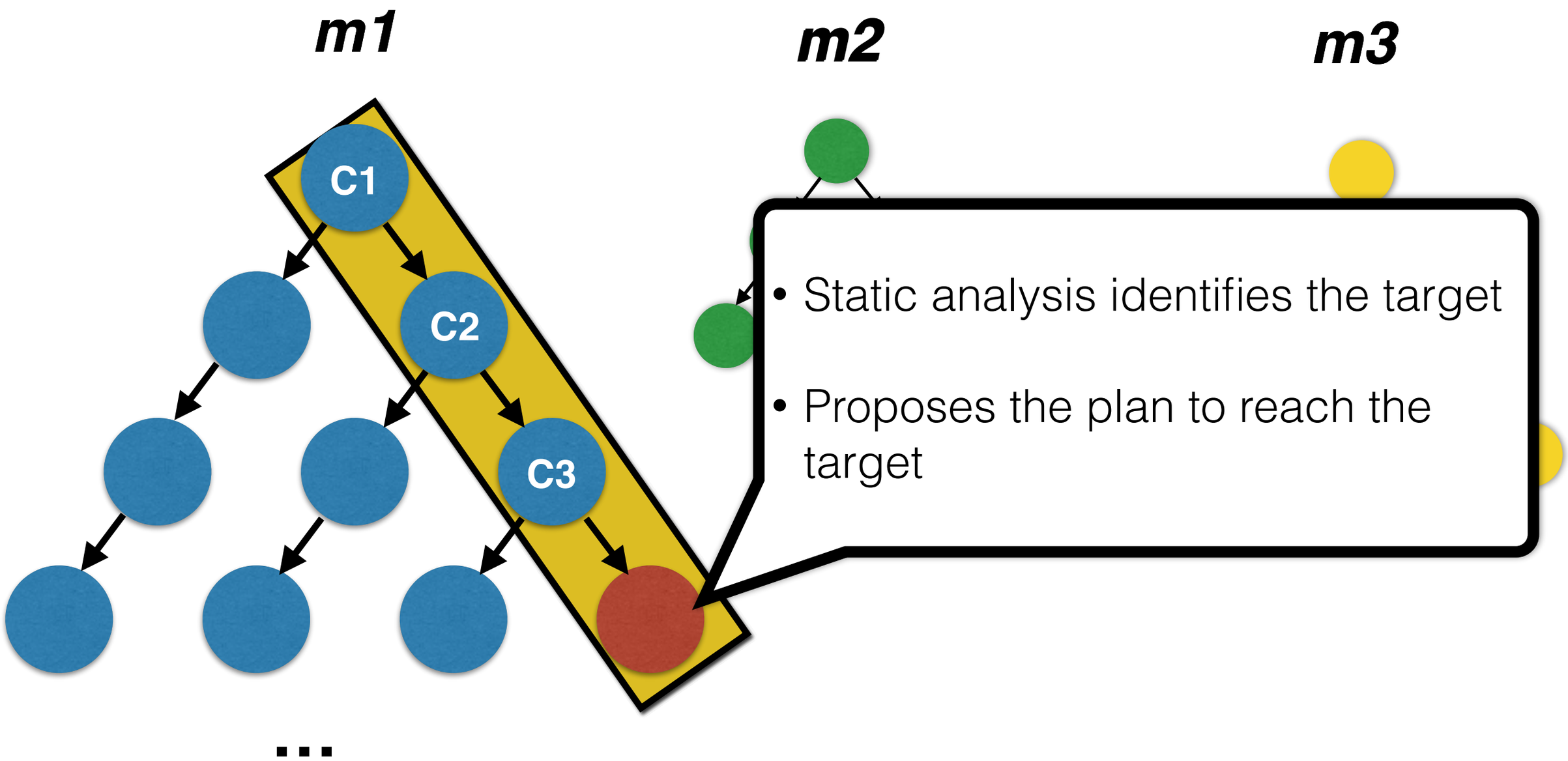


# Iteration - 1 : Static analysis

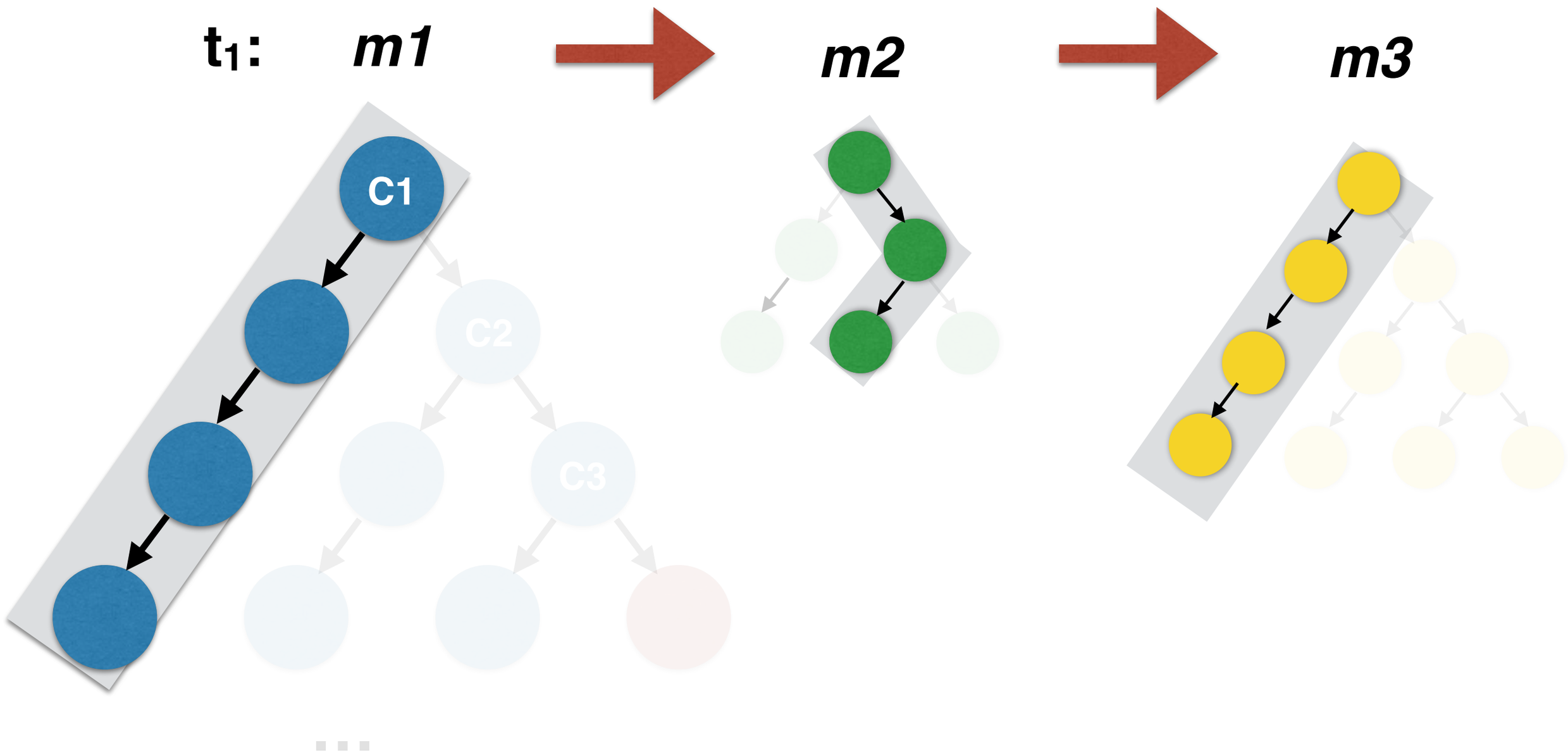



- Static analysis identifies the target
- Proposes the plan to reach the target

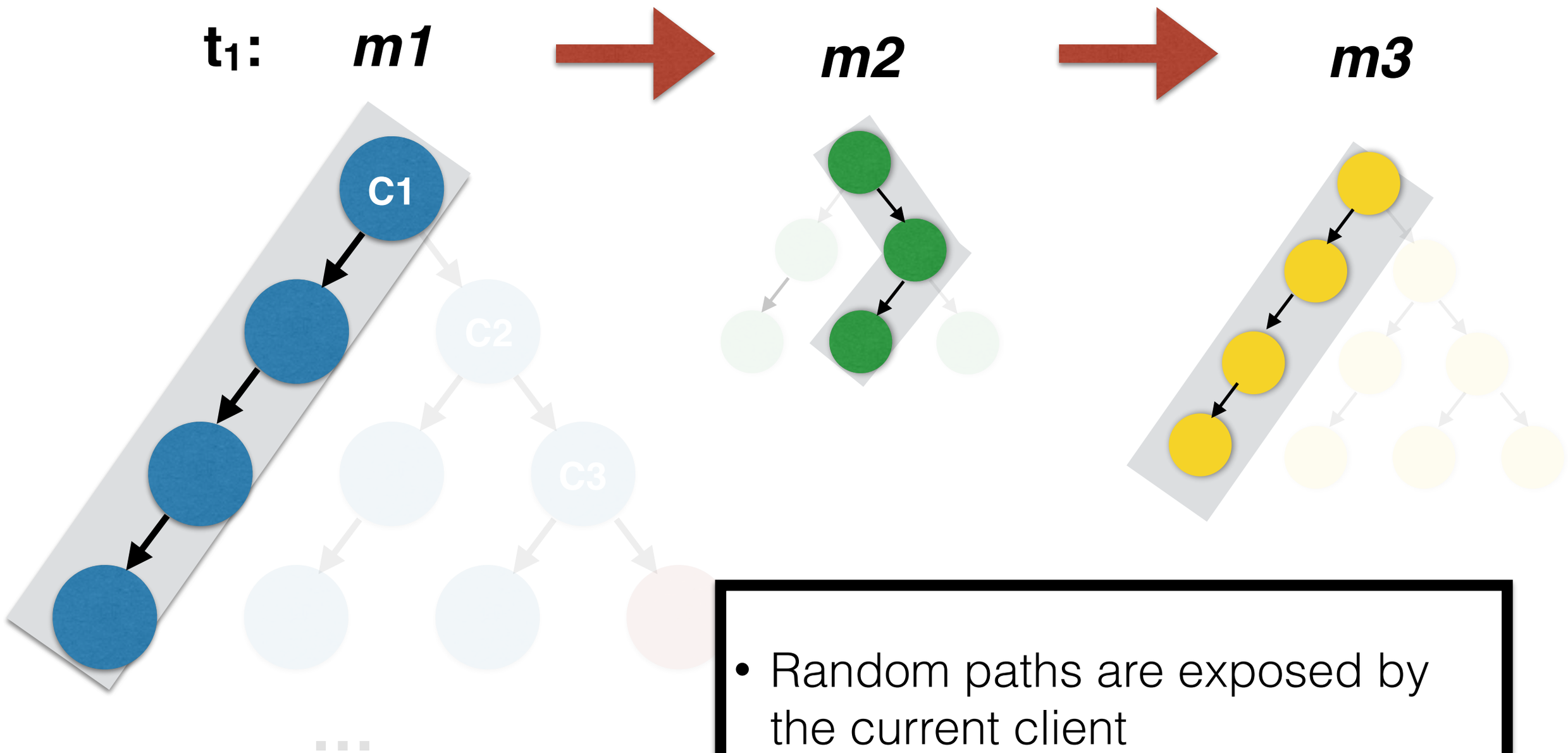
# Iteration - 1 : Static analysis



# Iteration - 1 : Concrete Execution

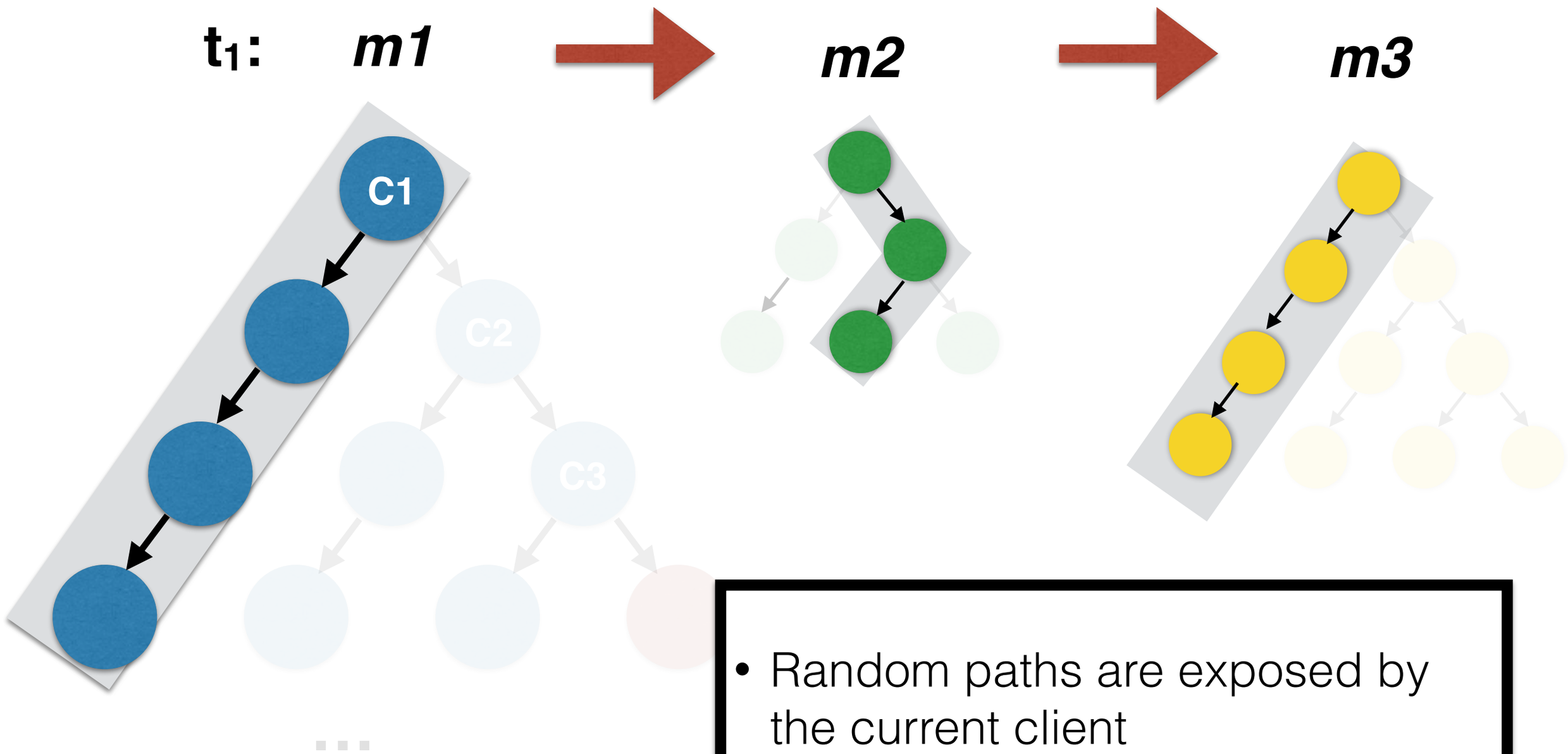


# Iteration - 1 : Concrete Execution



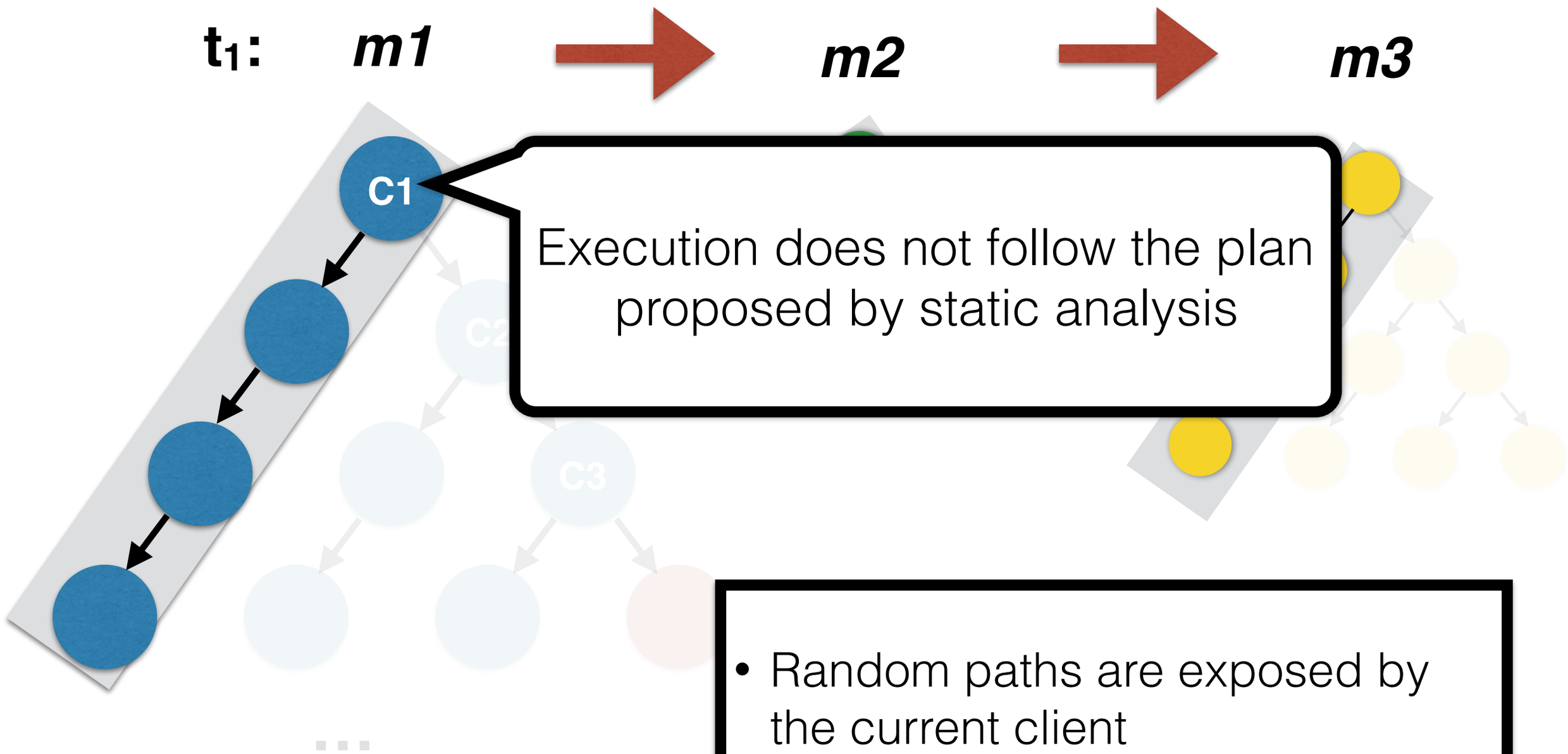

- Random paths are exposed by the current client

# Iteration - 1 : Concrete Execution




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

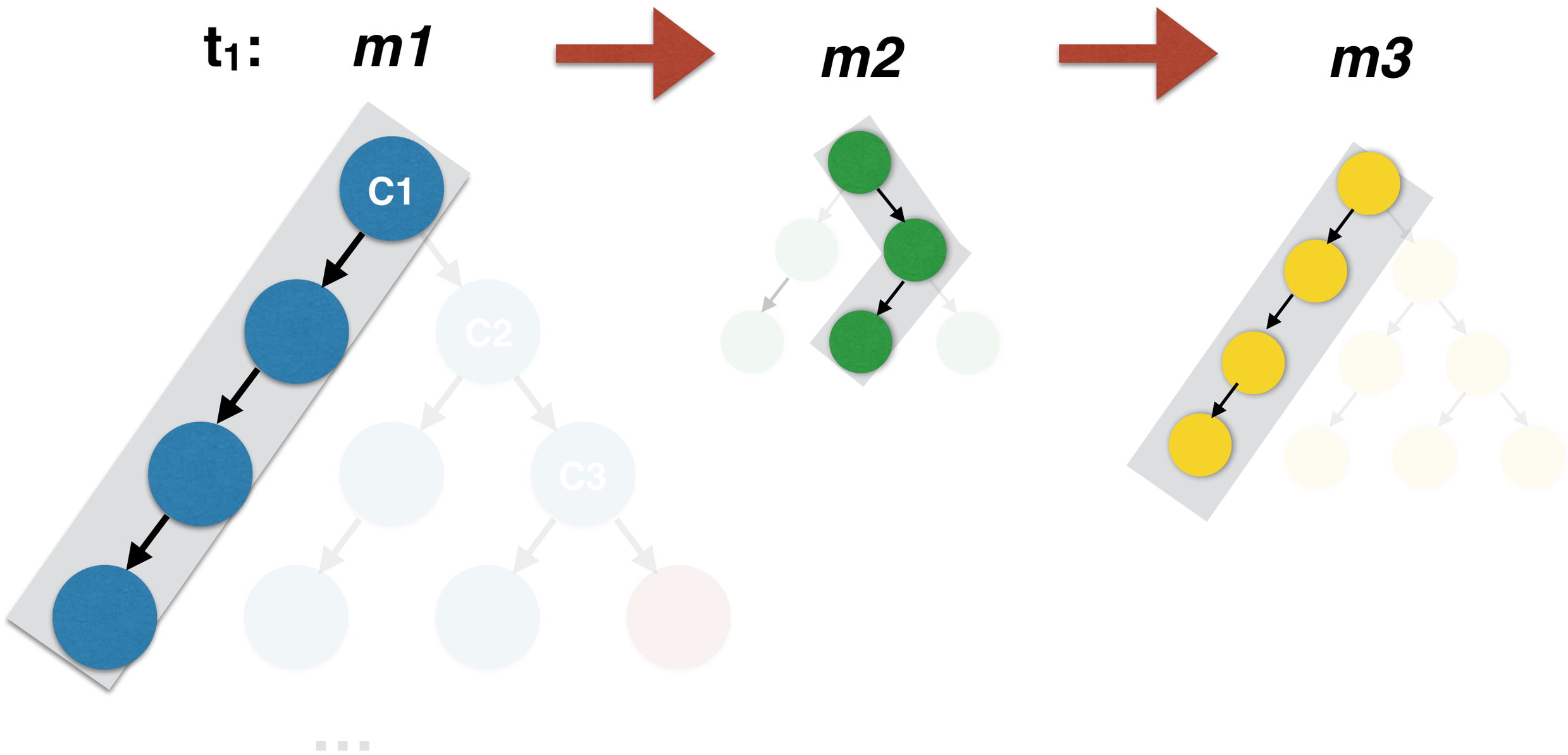
# Iteration - 1 : Concrete Execution



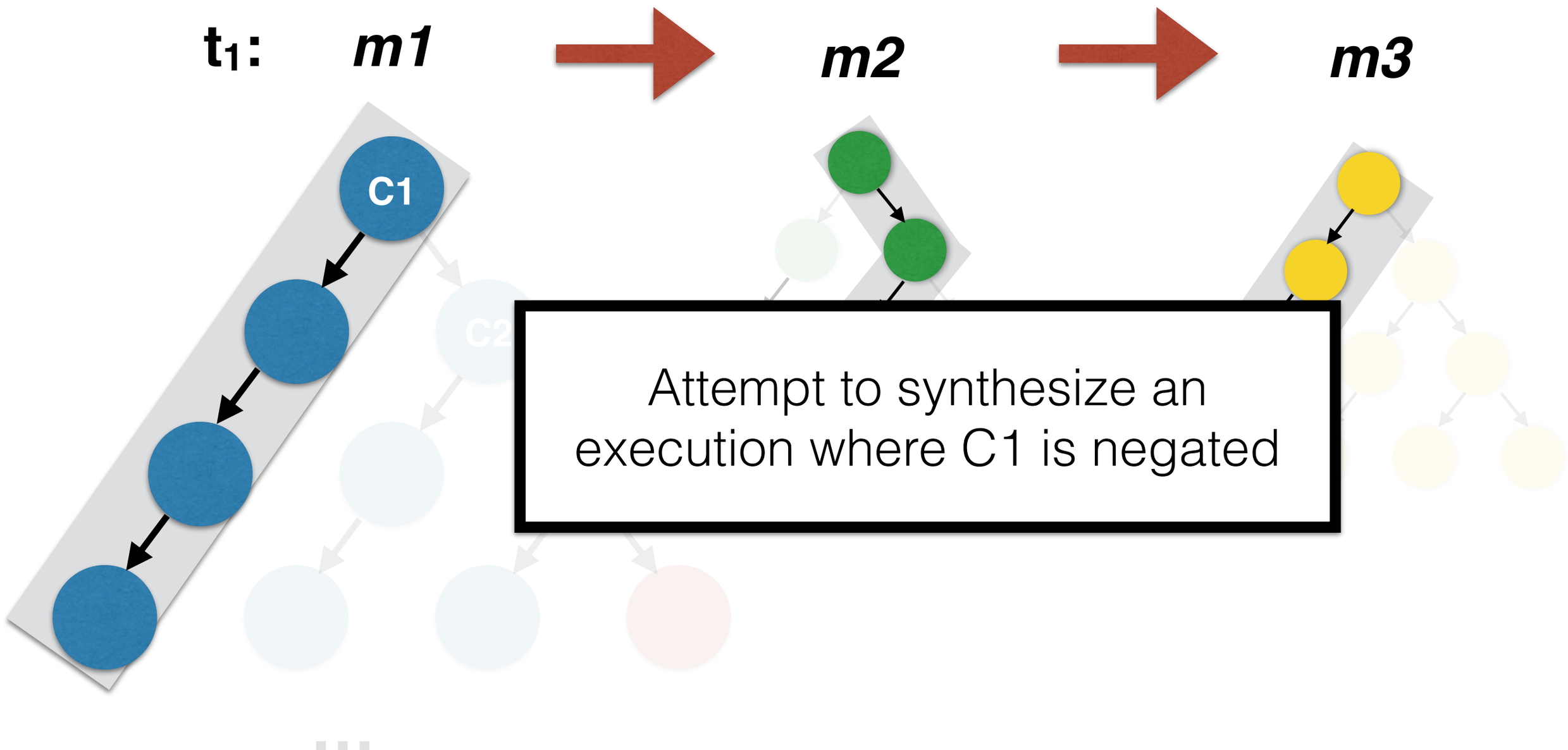

Execution does not follow the plan proposed by static analysis

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

# Iteration - 1 : Concrete Execution

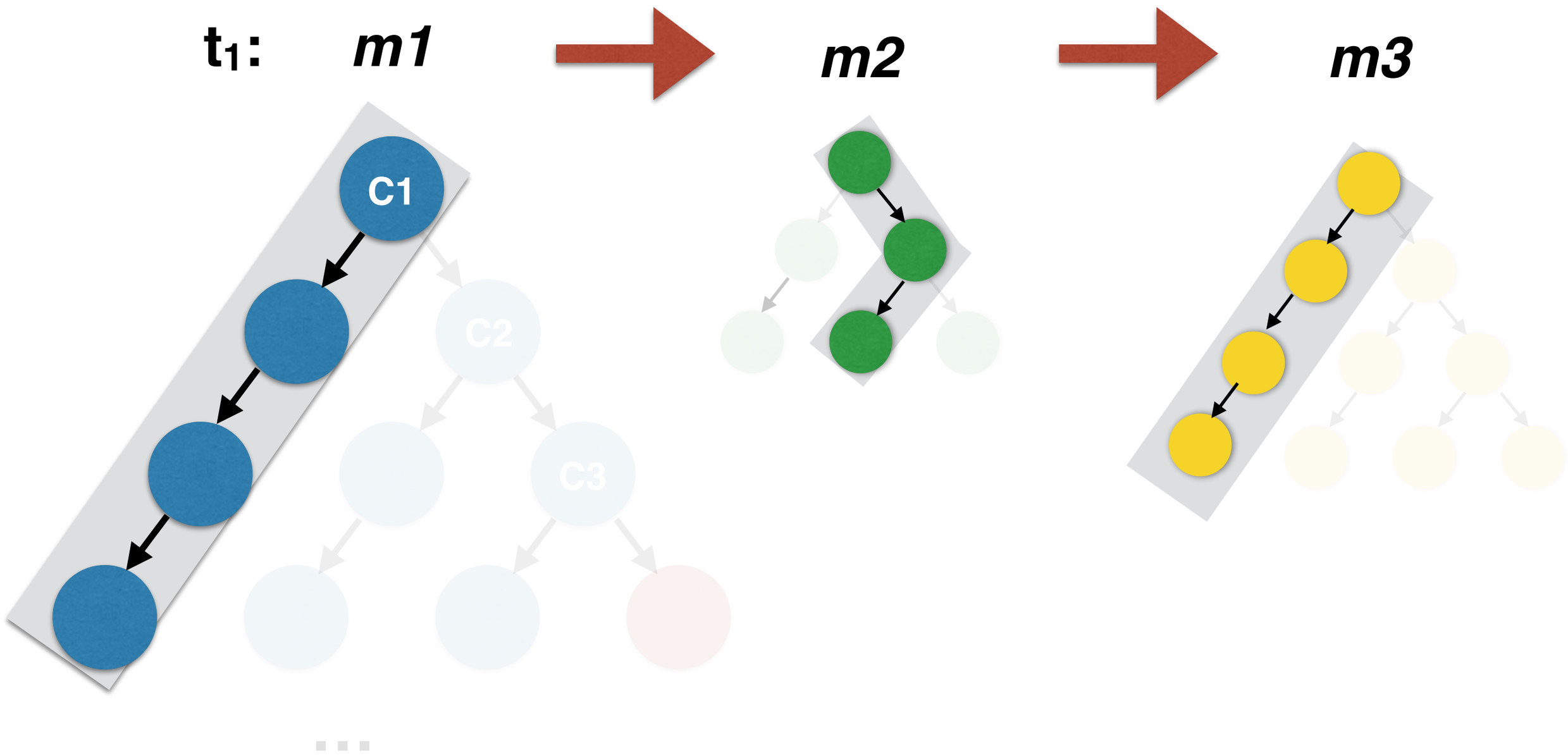


# Iteration - 1 : Concrete Execution

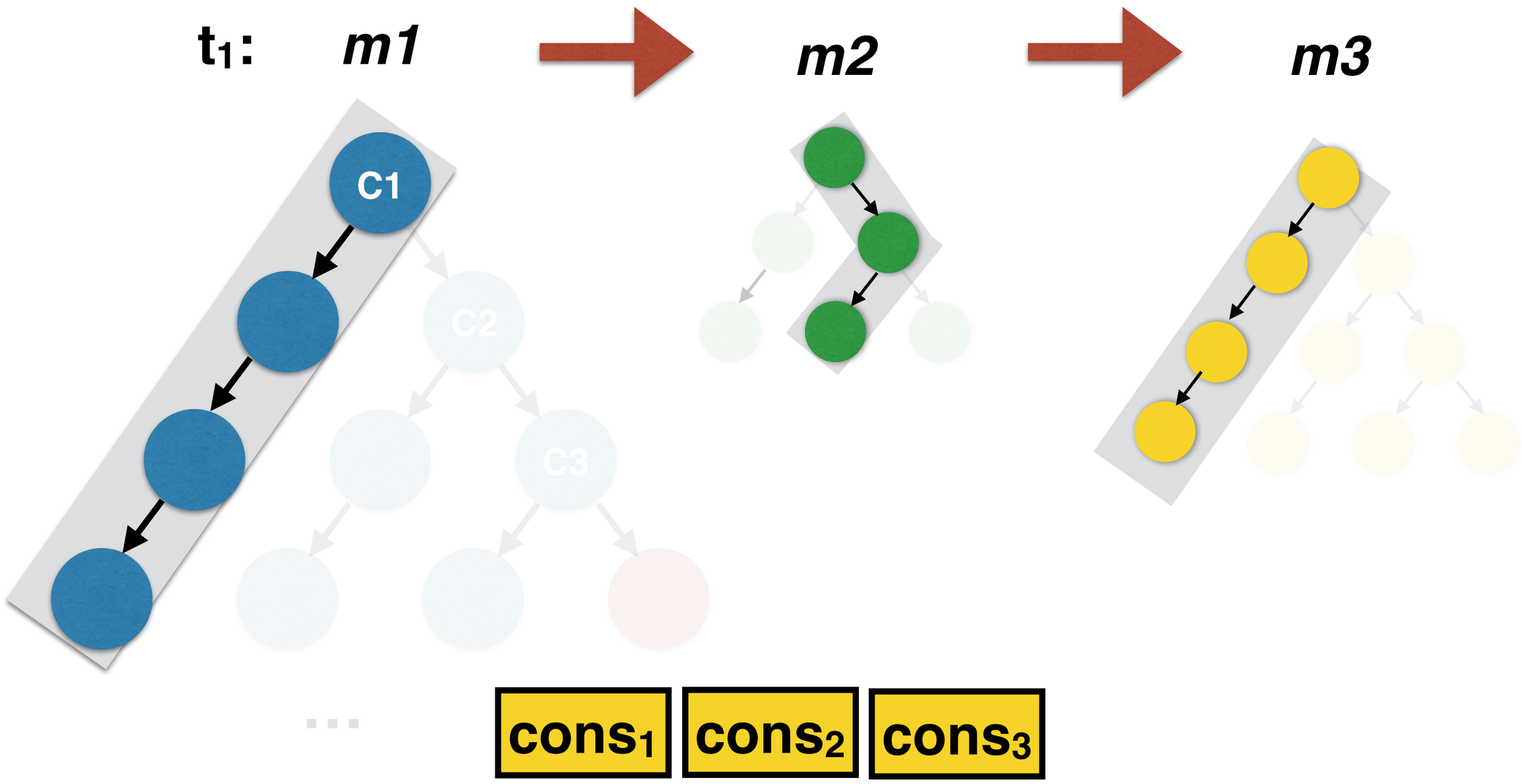



Attempt to synthesize an execution where C1 is negated

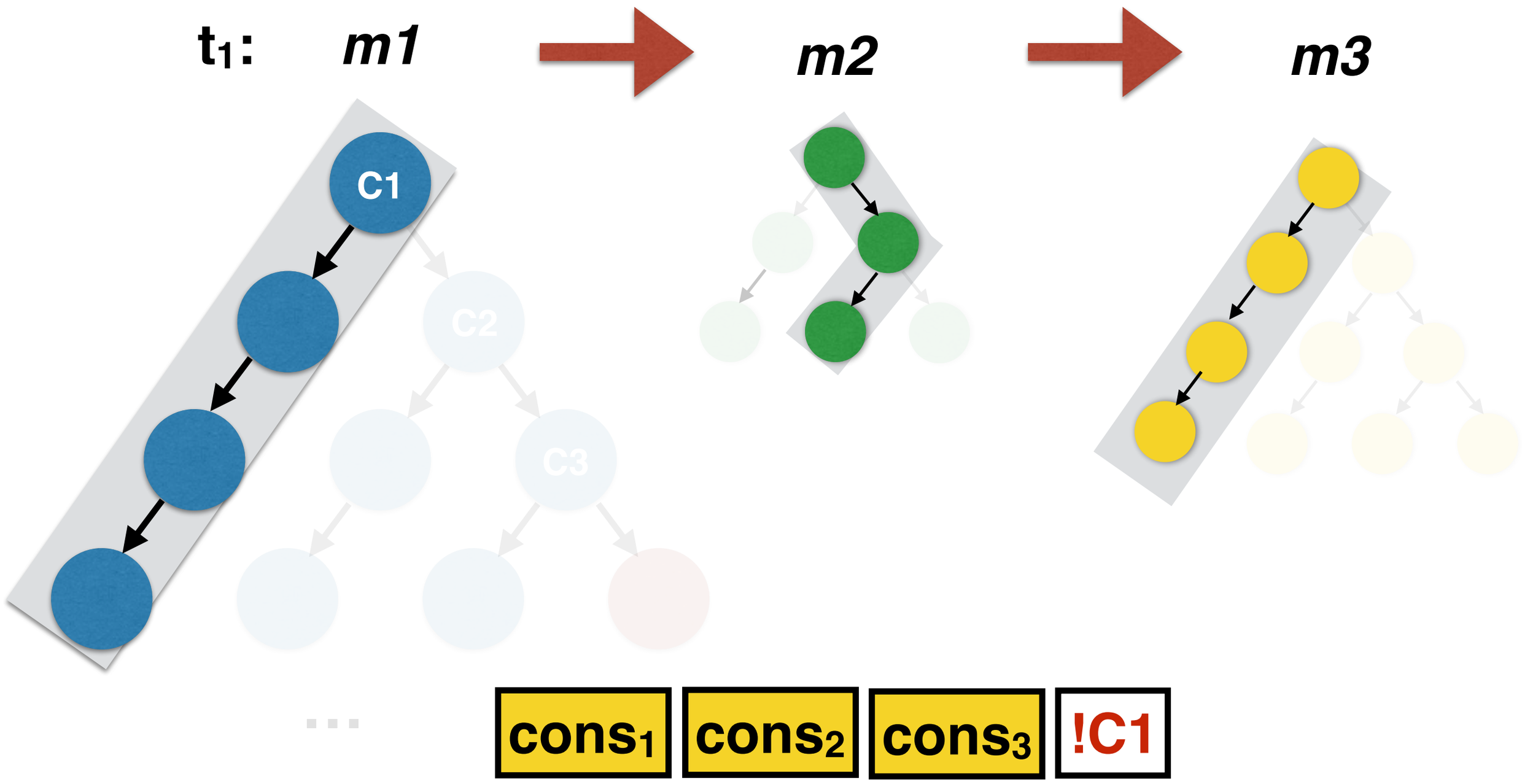
# Iteration - 1 : Concrete Execution



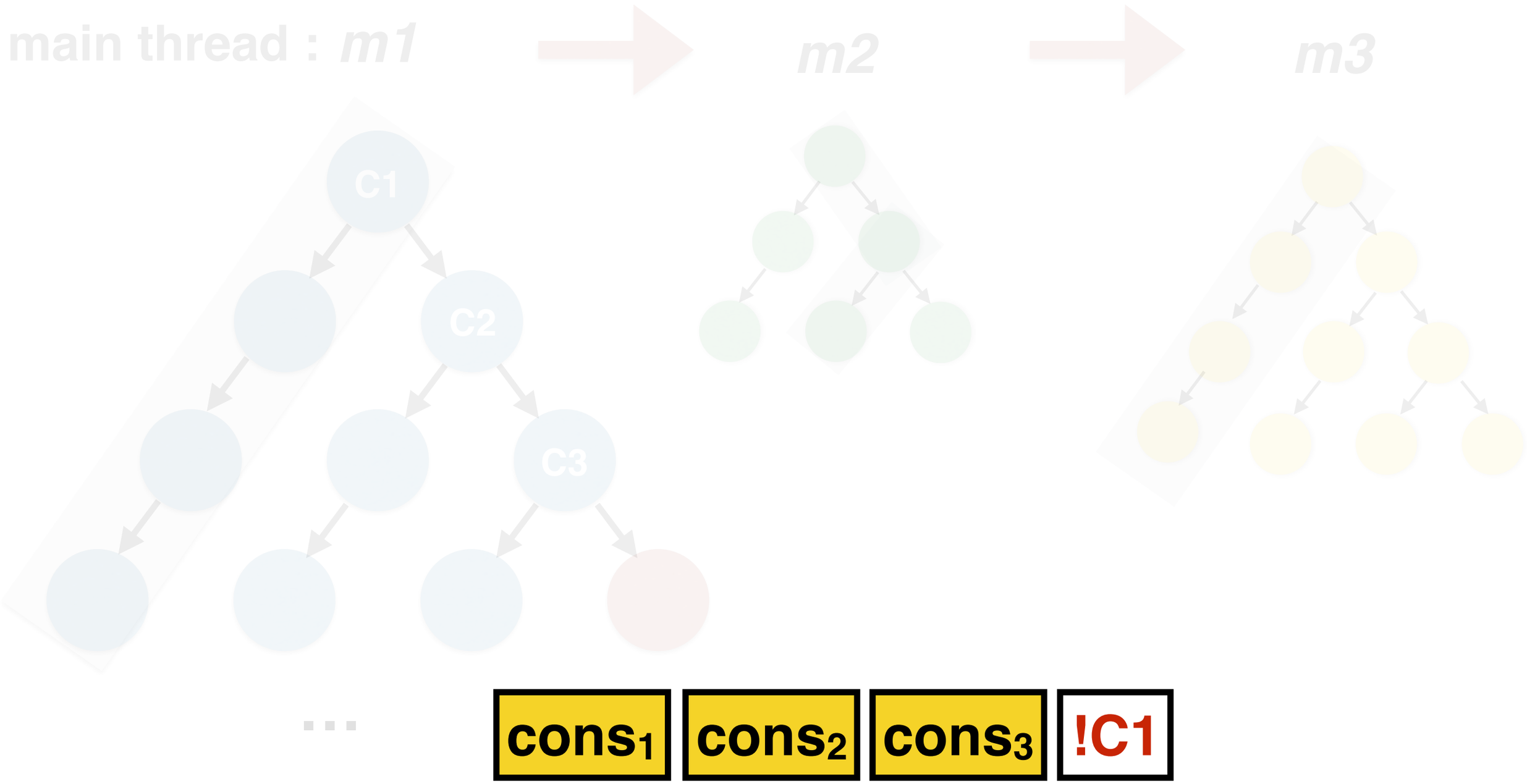
# Iteration - 1 : Concrete Execution



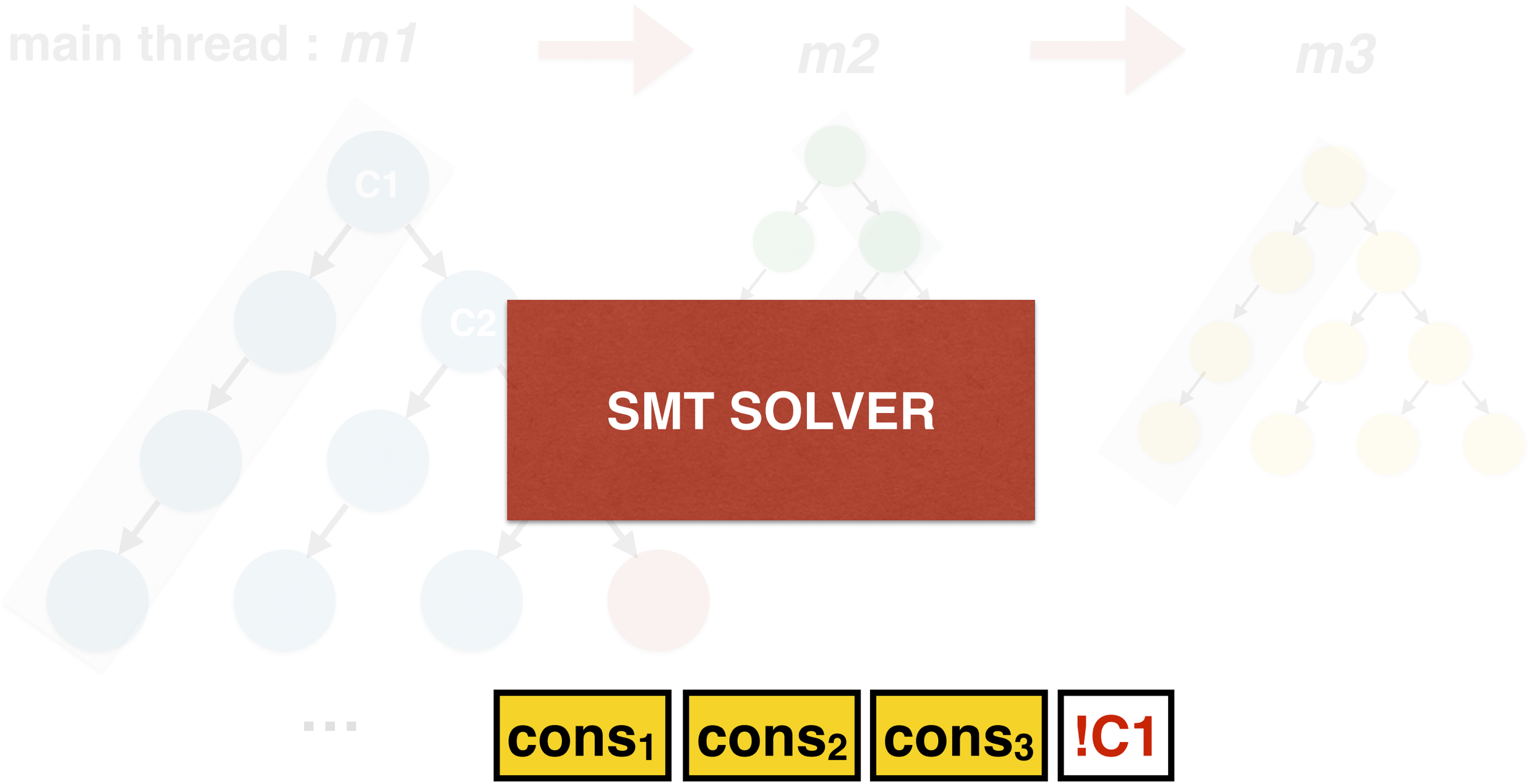
# Iteration - 1 : Concrete Execution



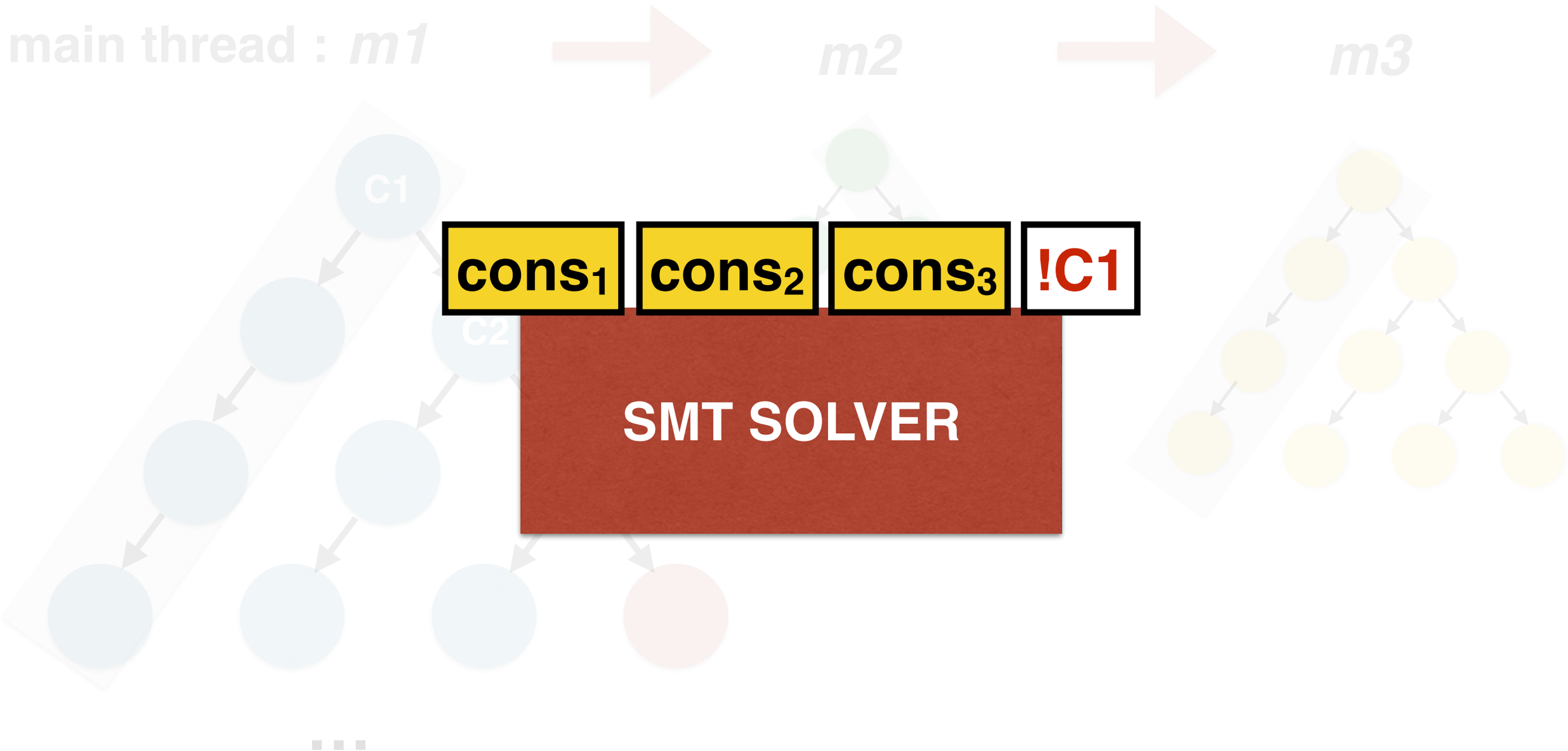
# Iteration - 1 : Client Synthesis



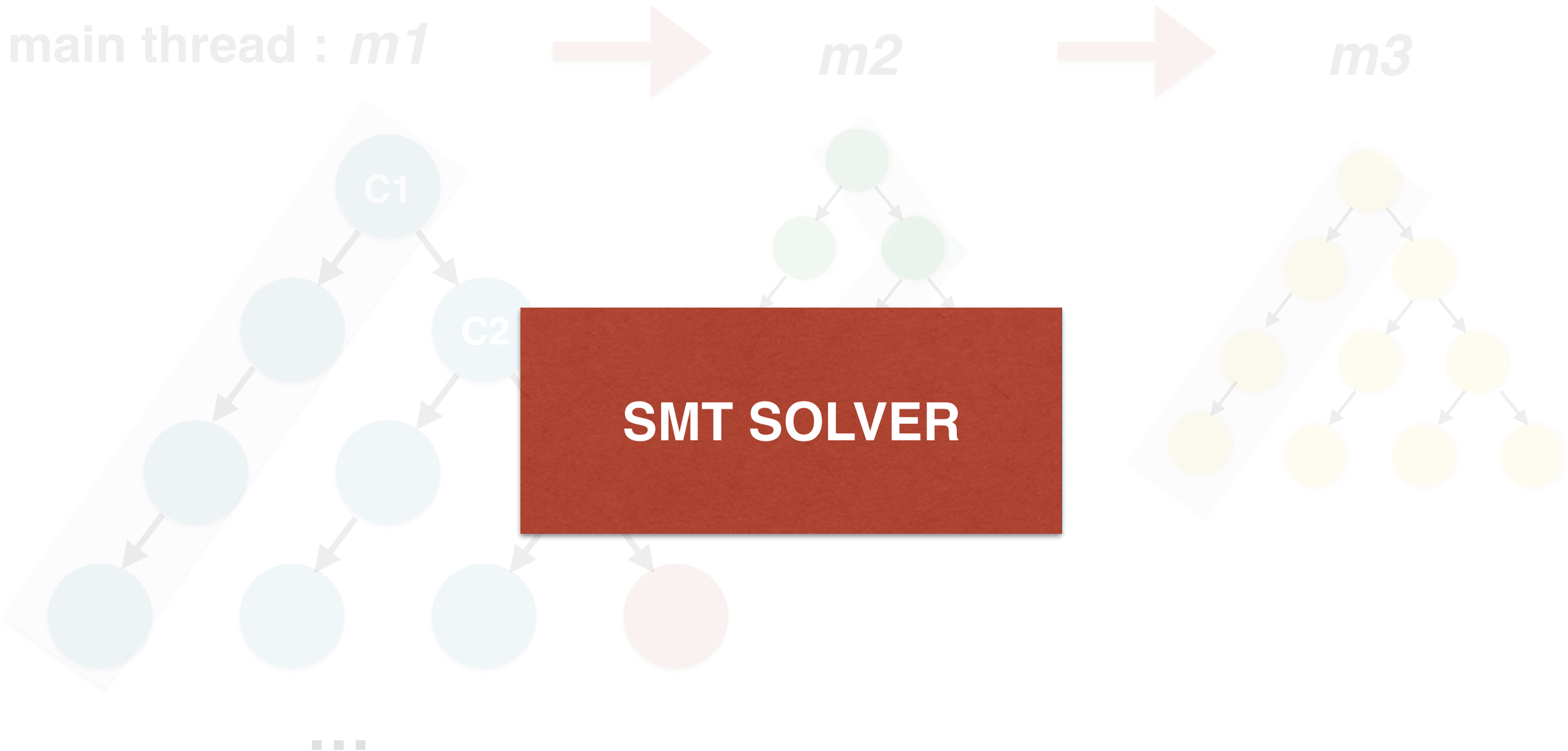
# Iteration - 1 : Client Synthesis



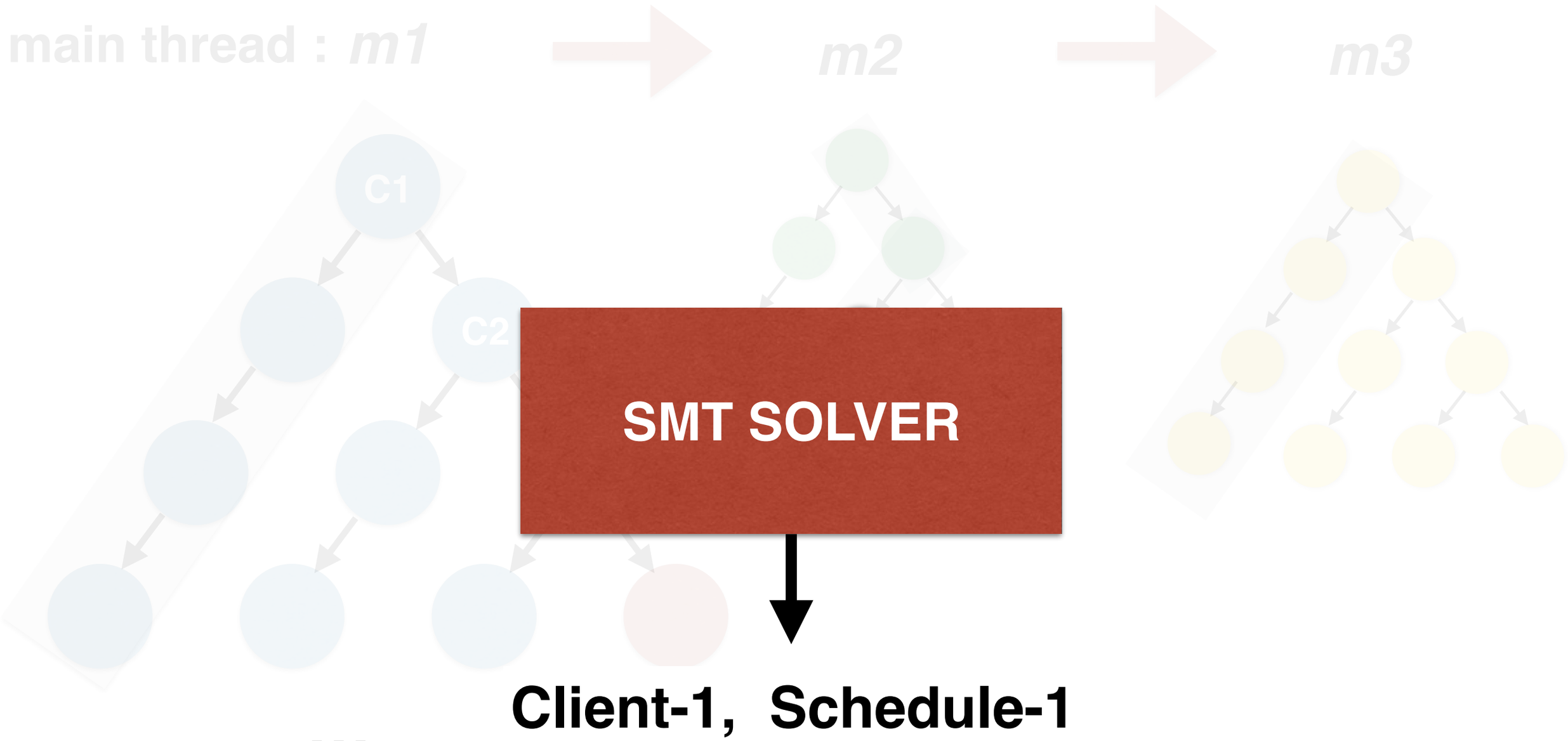
# Iteration - 1 : Client Synthesis



# Iteration - 1 : Client Synthesis



# Iteration - 1 : Client Synthesis



# Iteration - 2 : Concrete Execution

New client

$t_1$ : ***m1***

$t_2$ : ***m2***

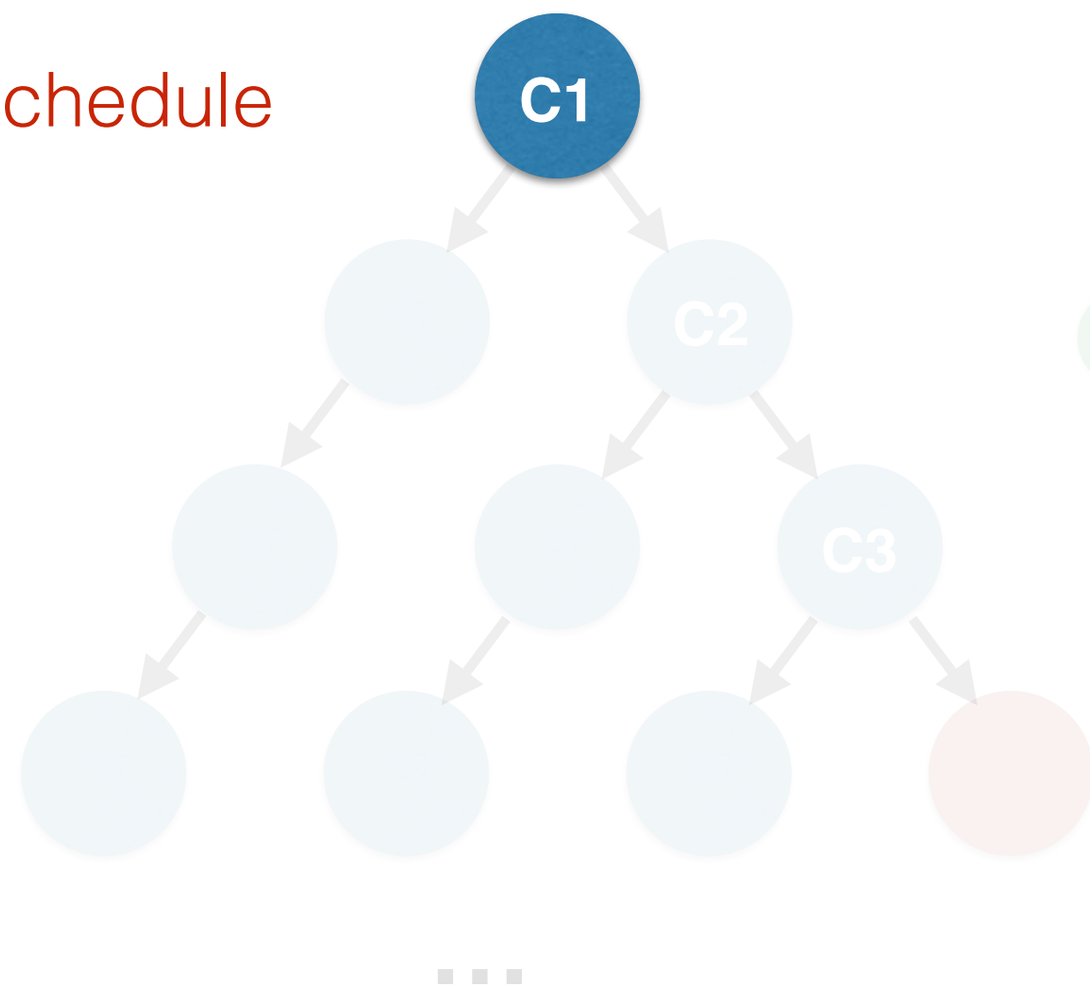
→ ***m3***

# Iteration - 2 : Concrete Execution

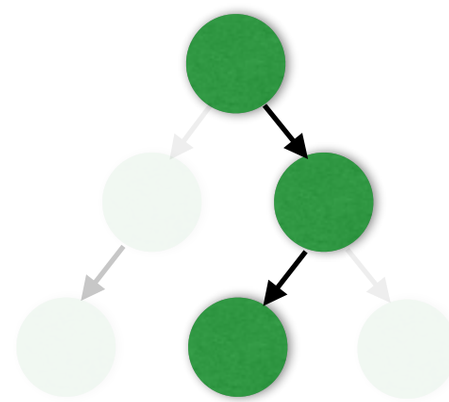
New client

Schedule

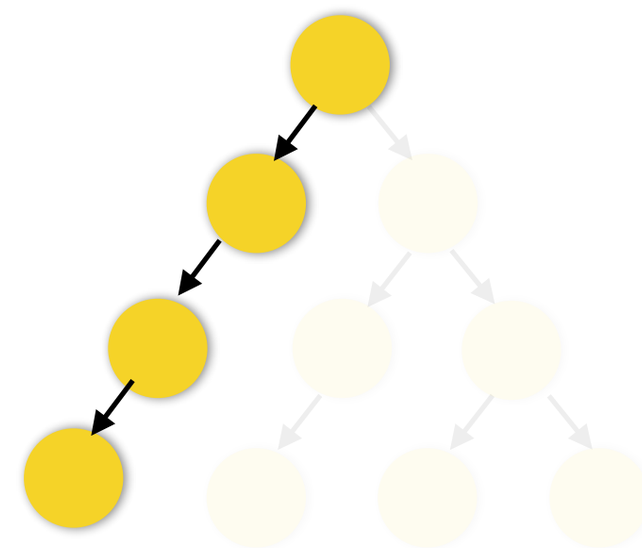
$t_1$ : ***m1***



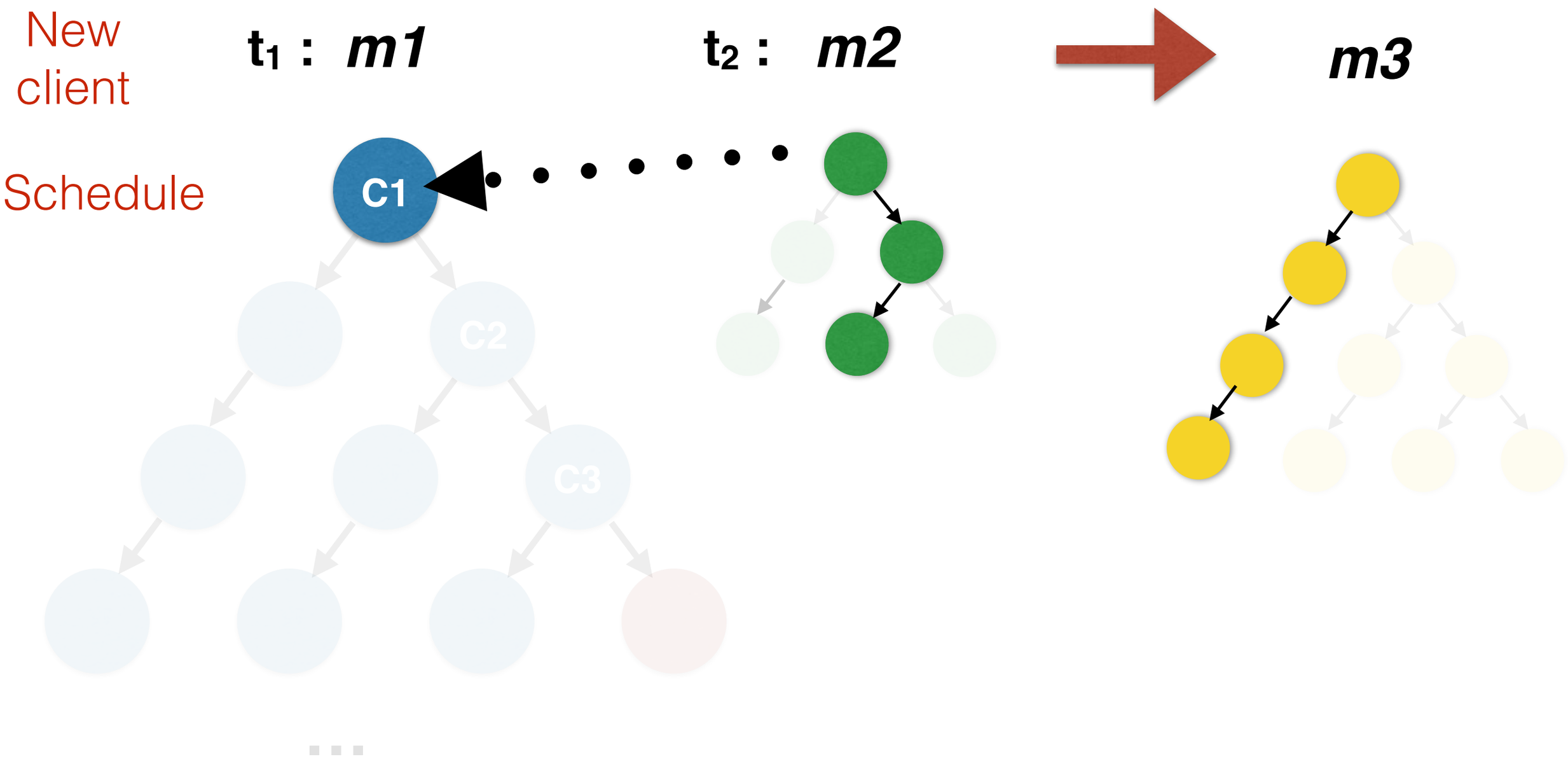

$t_2$ : ***m2***



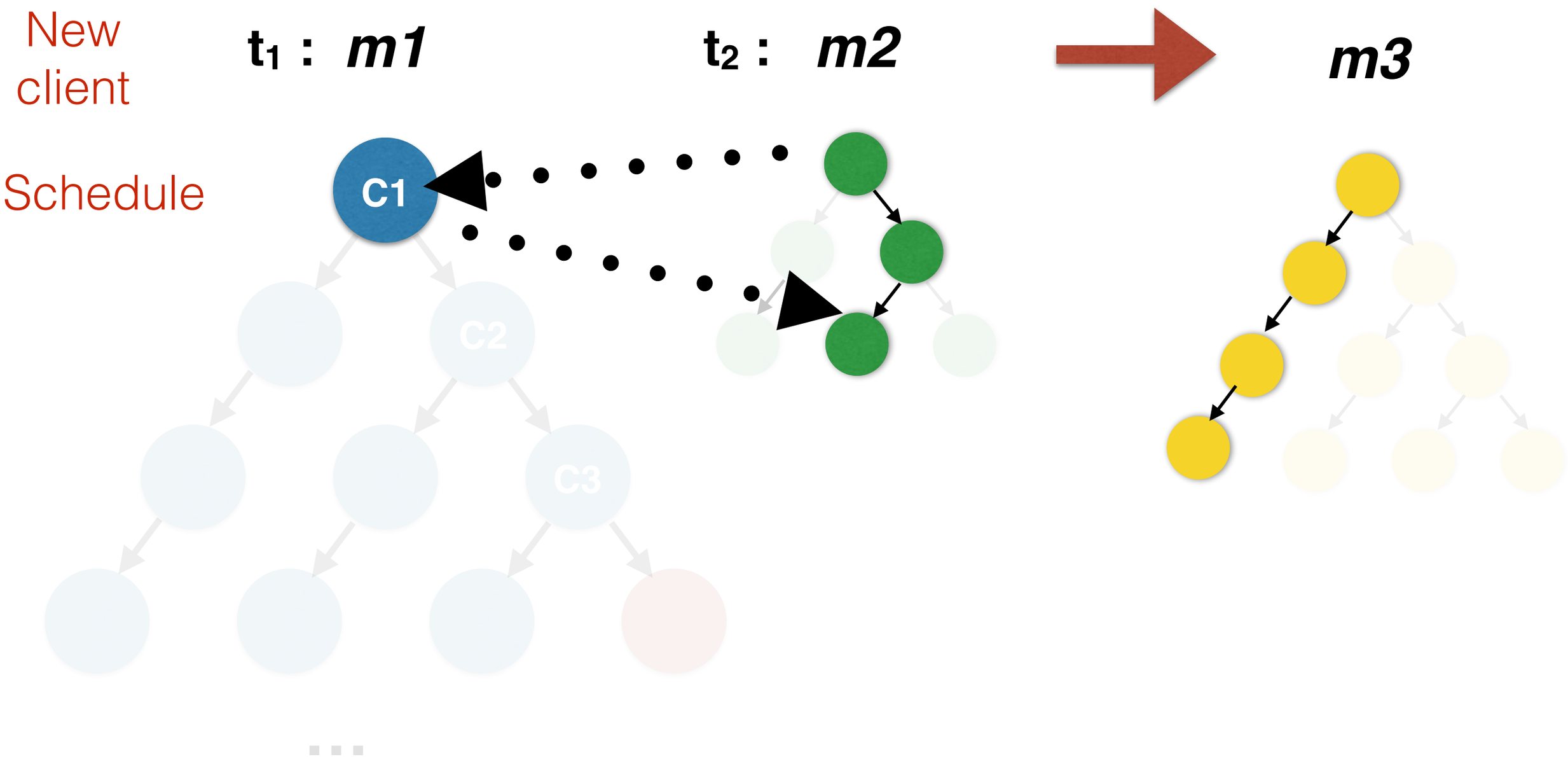
***m3***



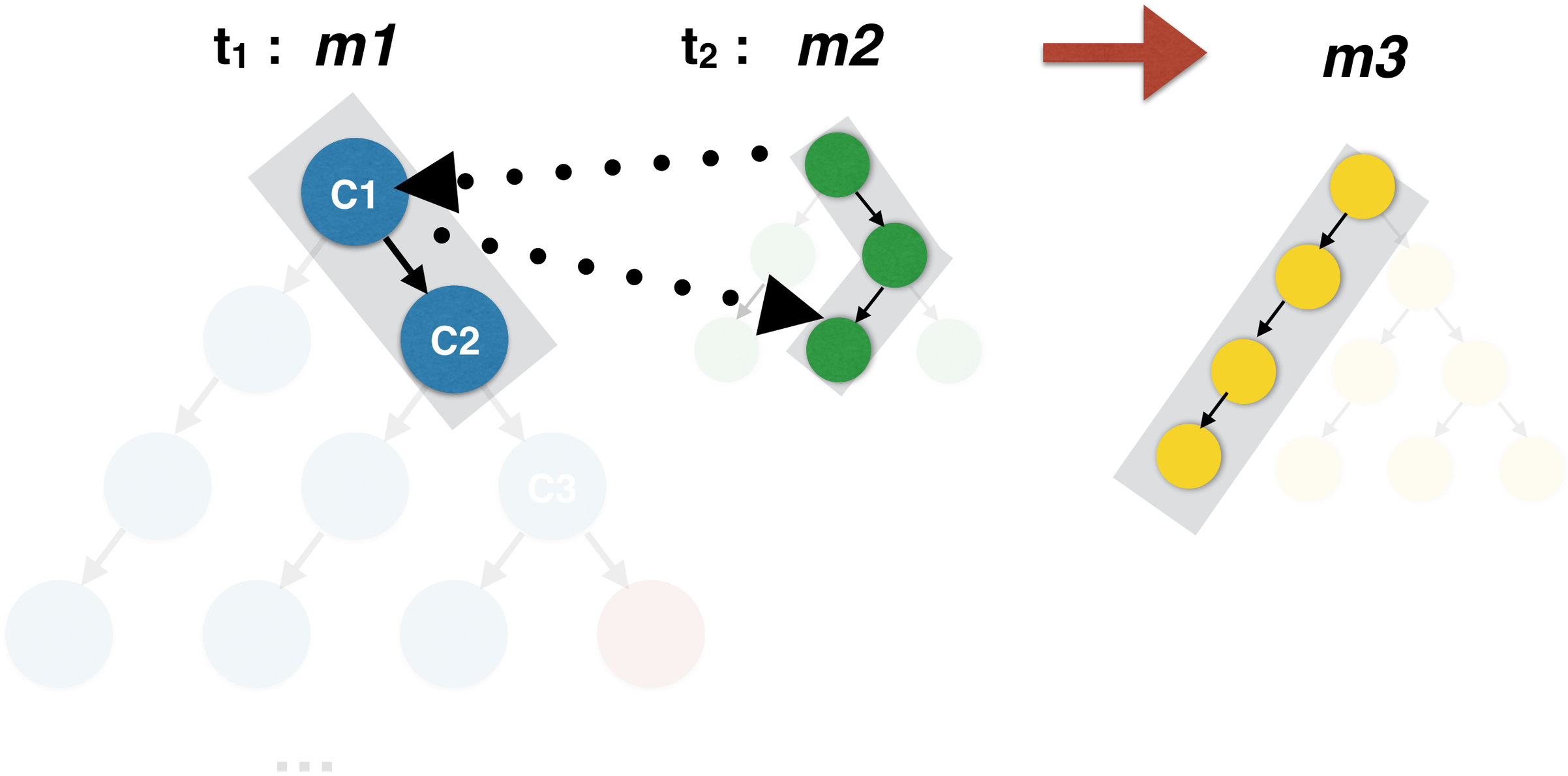
# Iteration - 2 : Concrete Execution



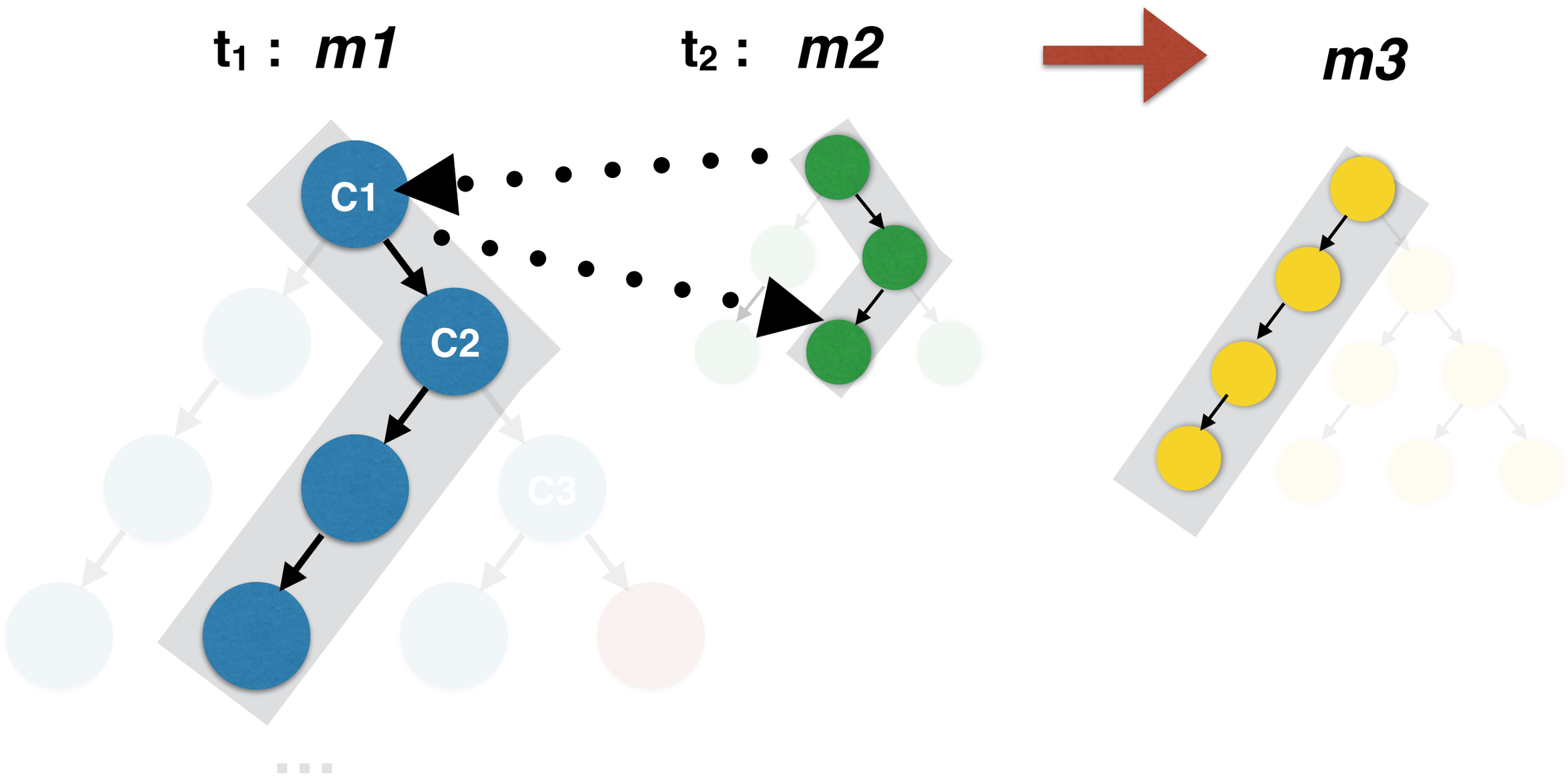
# Iteration - 2 : Concrete Execution



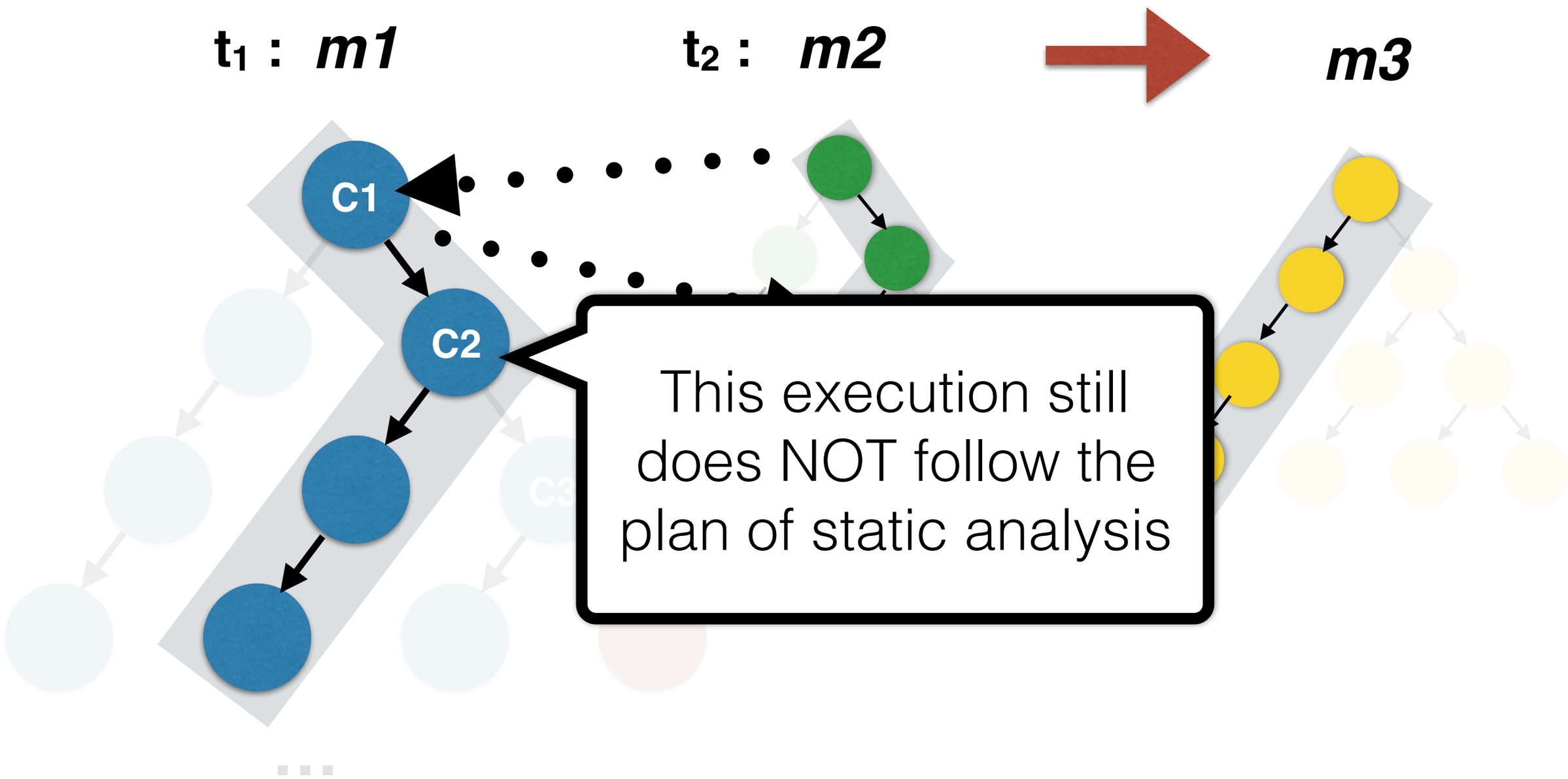
# Iteration - 2 : Concrete Execution



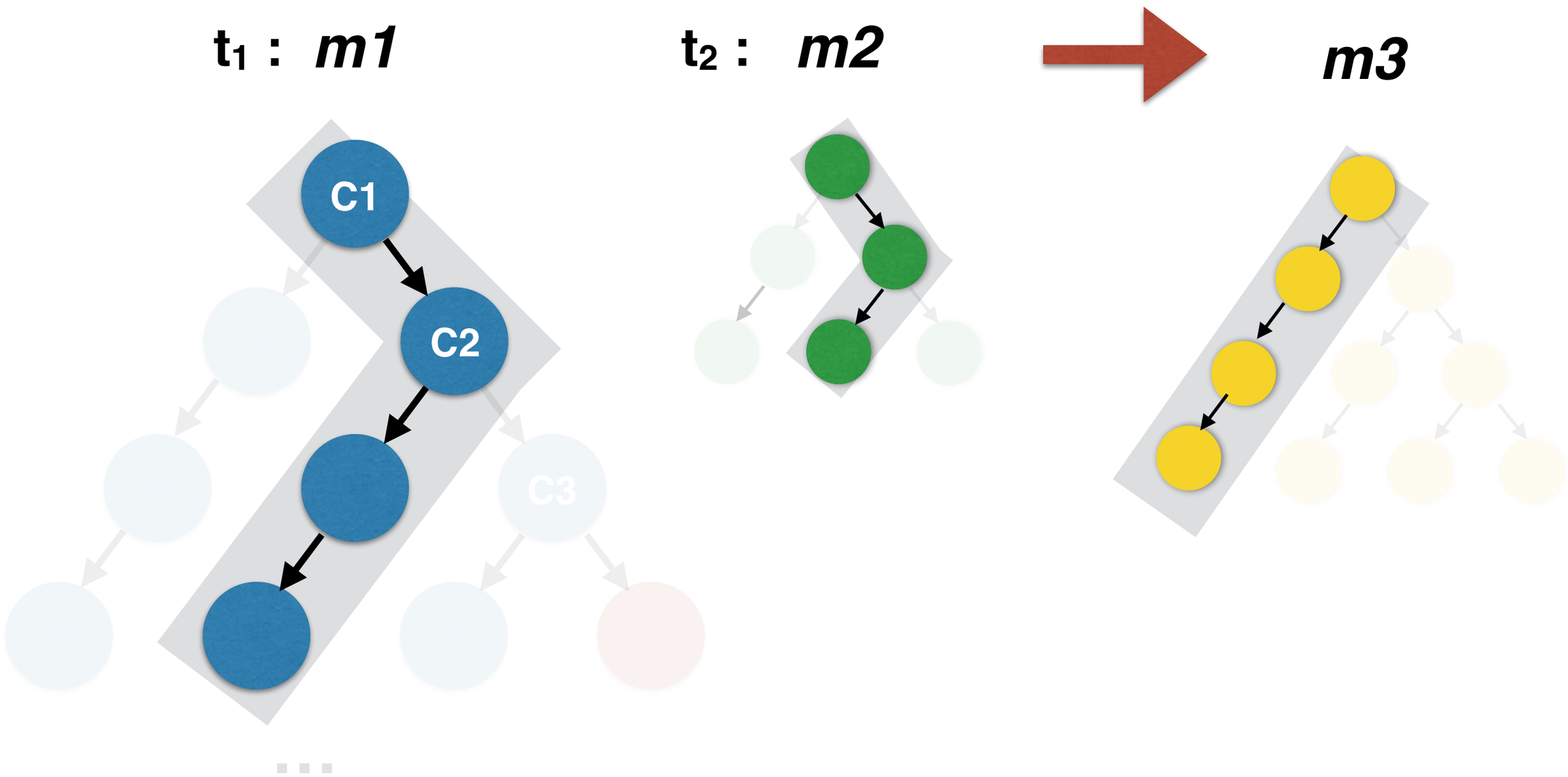
# Iteration - 2 : Concrete Execution



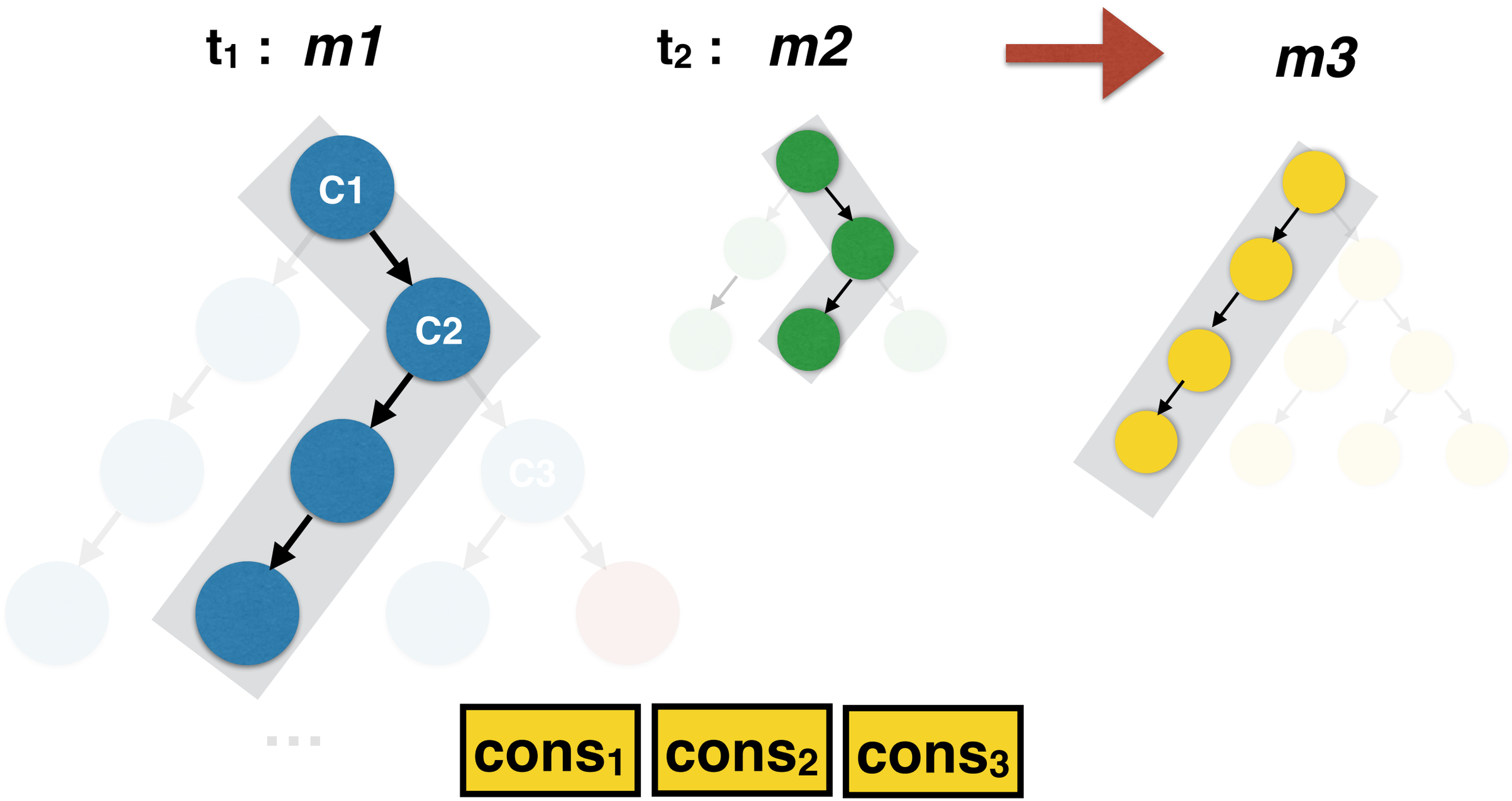
# Iteration - 2 : Concrete Execution



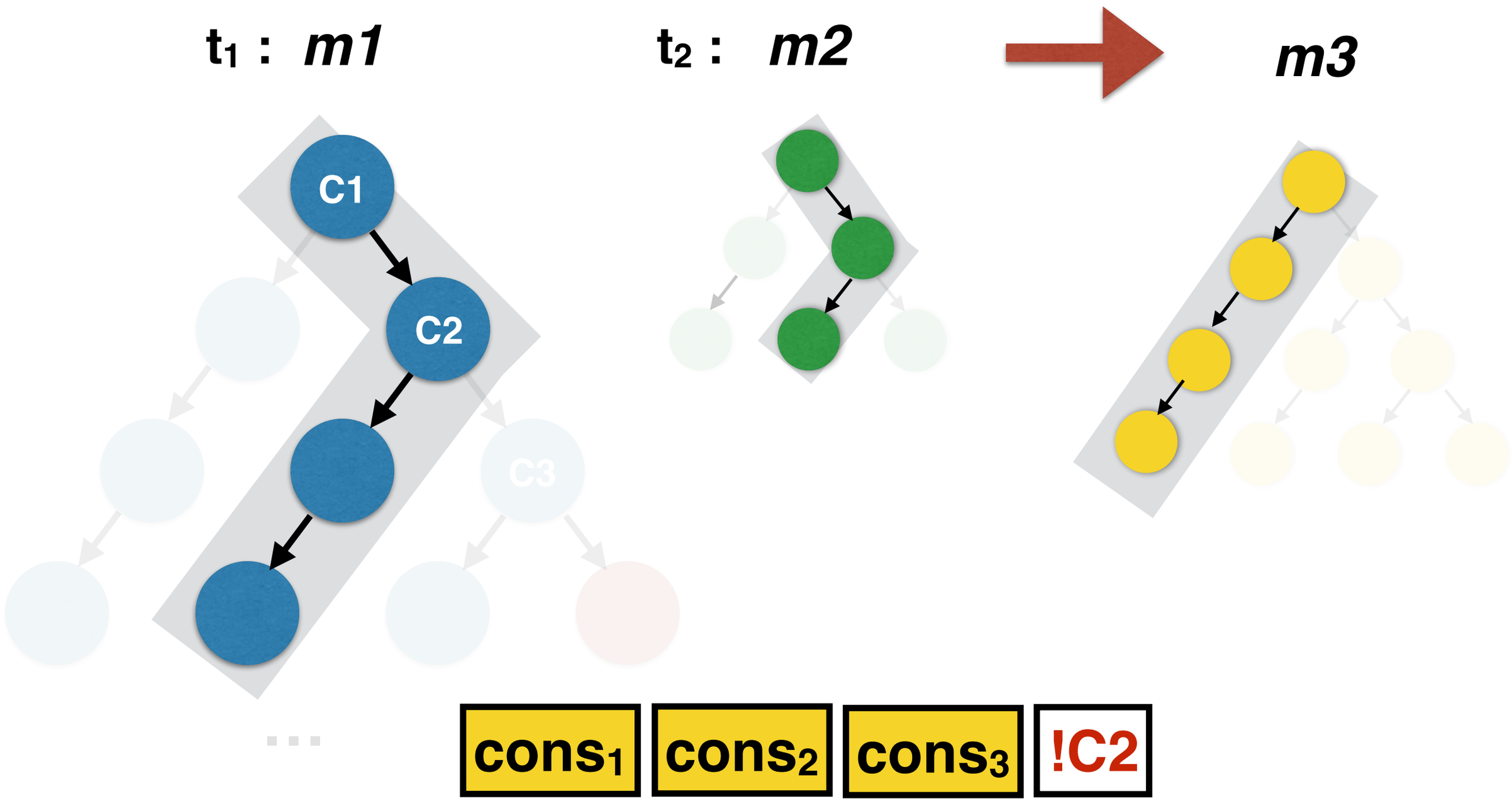
# Iteration - 2 : Concrete Execution



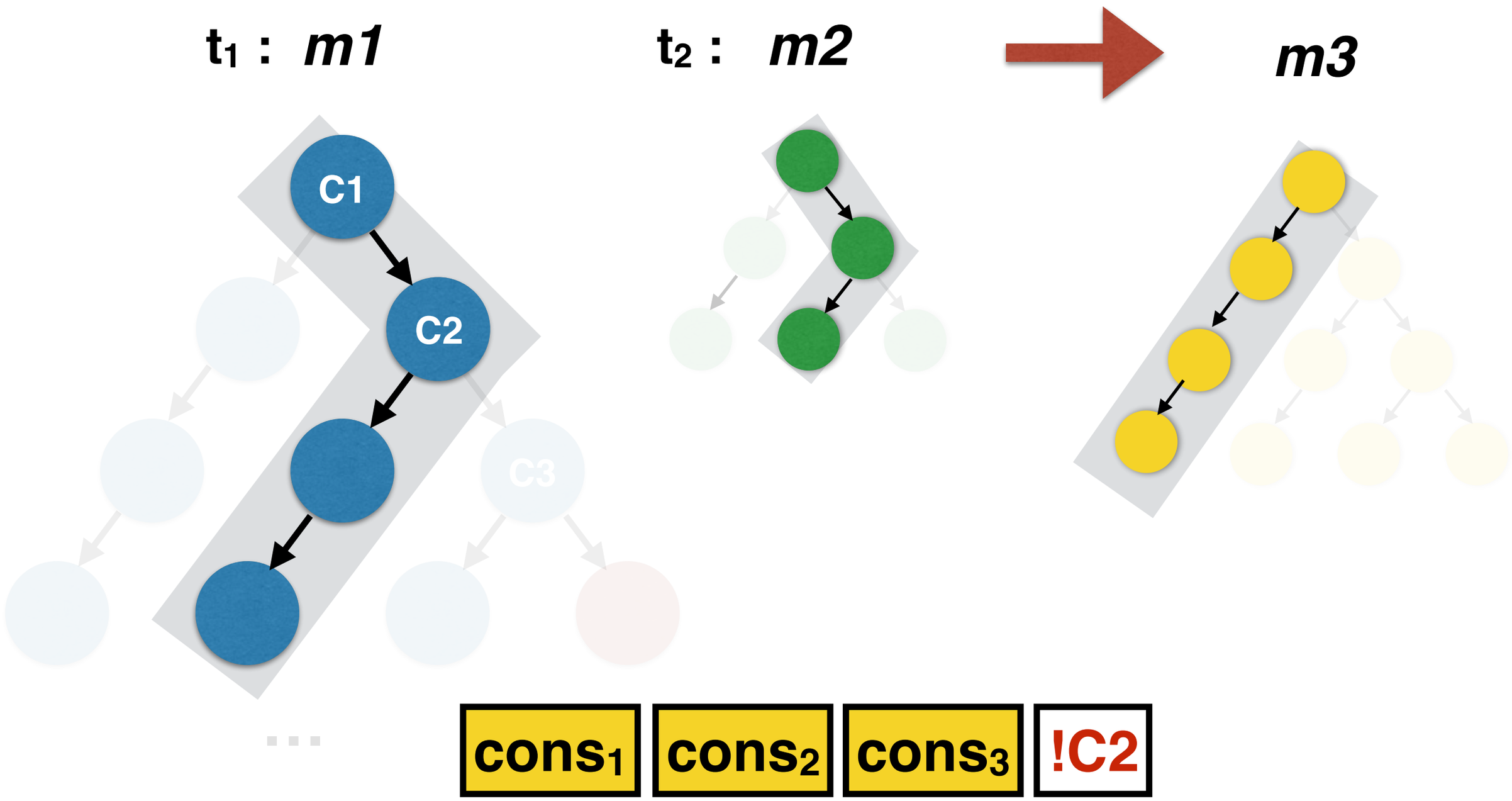
# Iteration - 2 : Concrete Execution



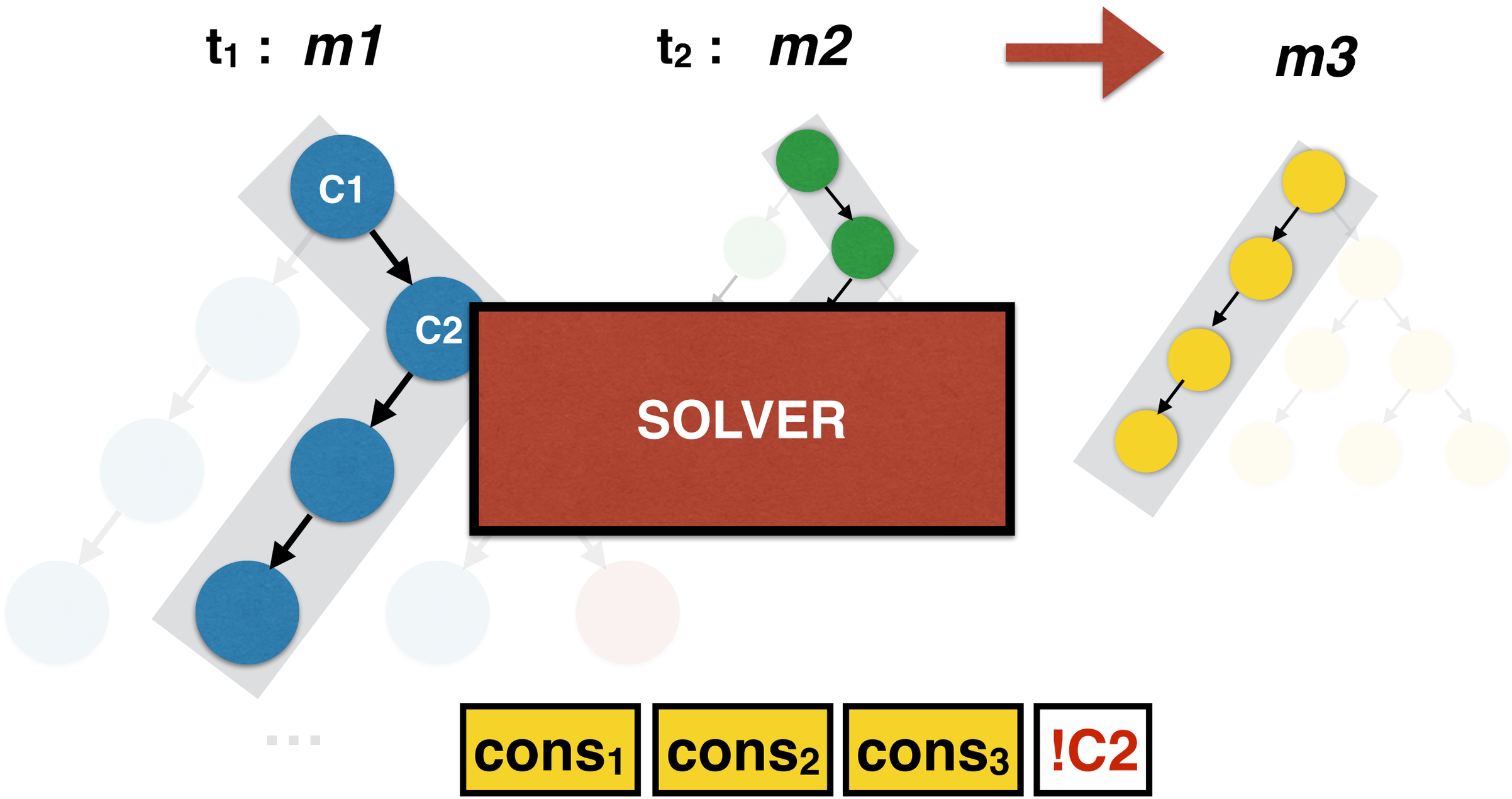
# Iteration - 2 : Concrete Execution



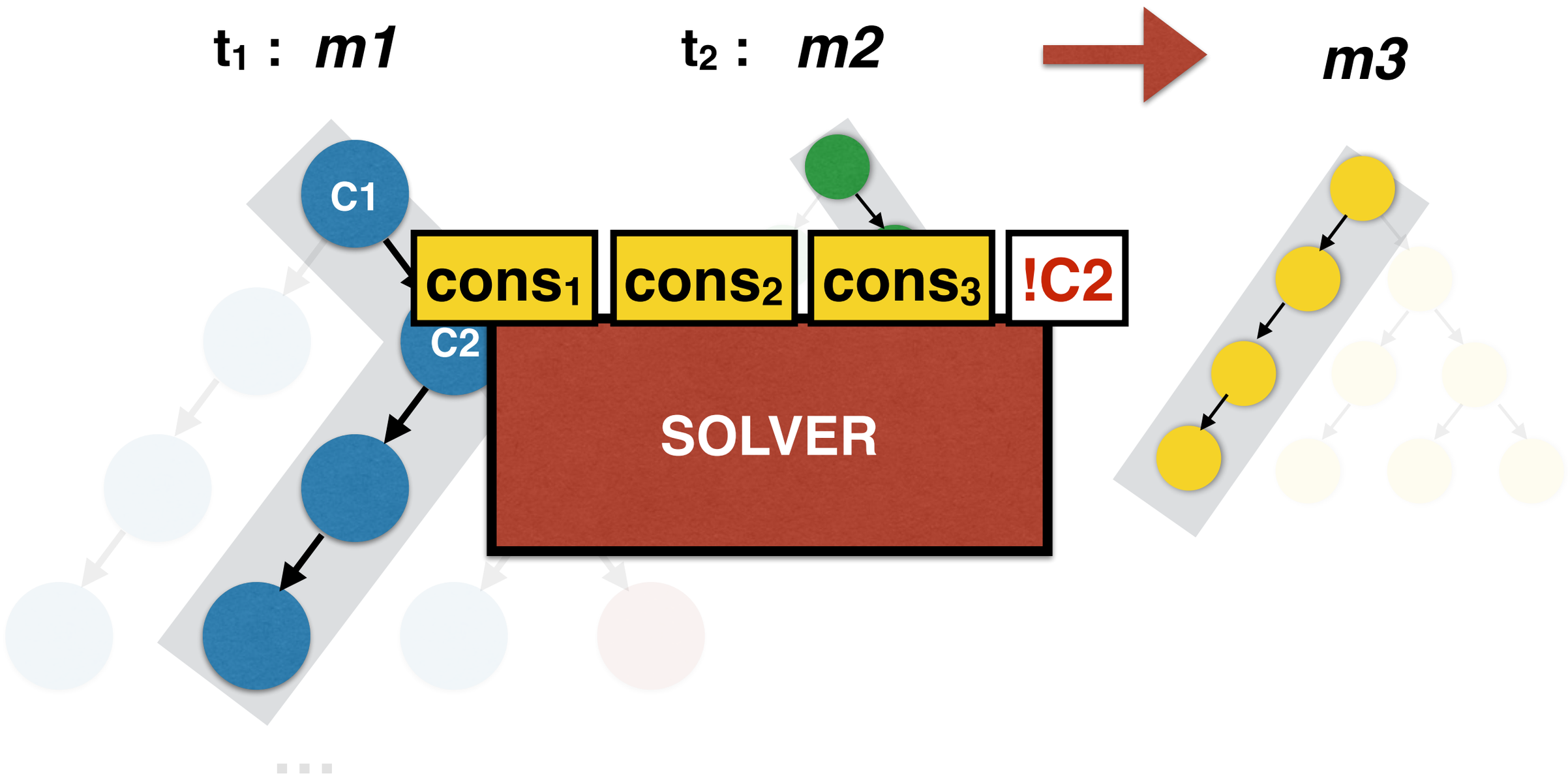
# Iteration - 2 : Concrete Execution



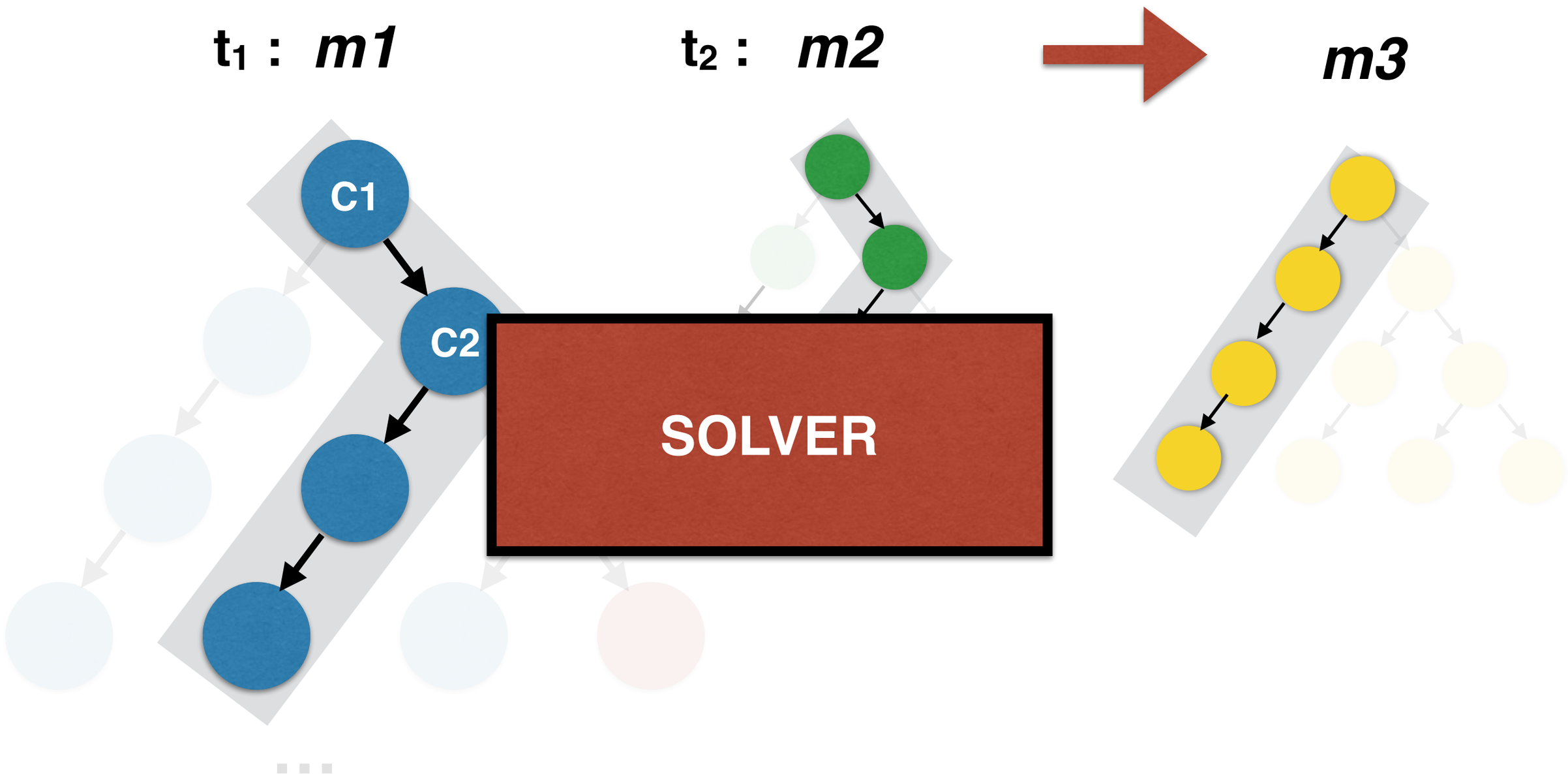
# Iteration - 2 : Concrete Execution



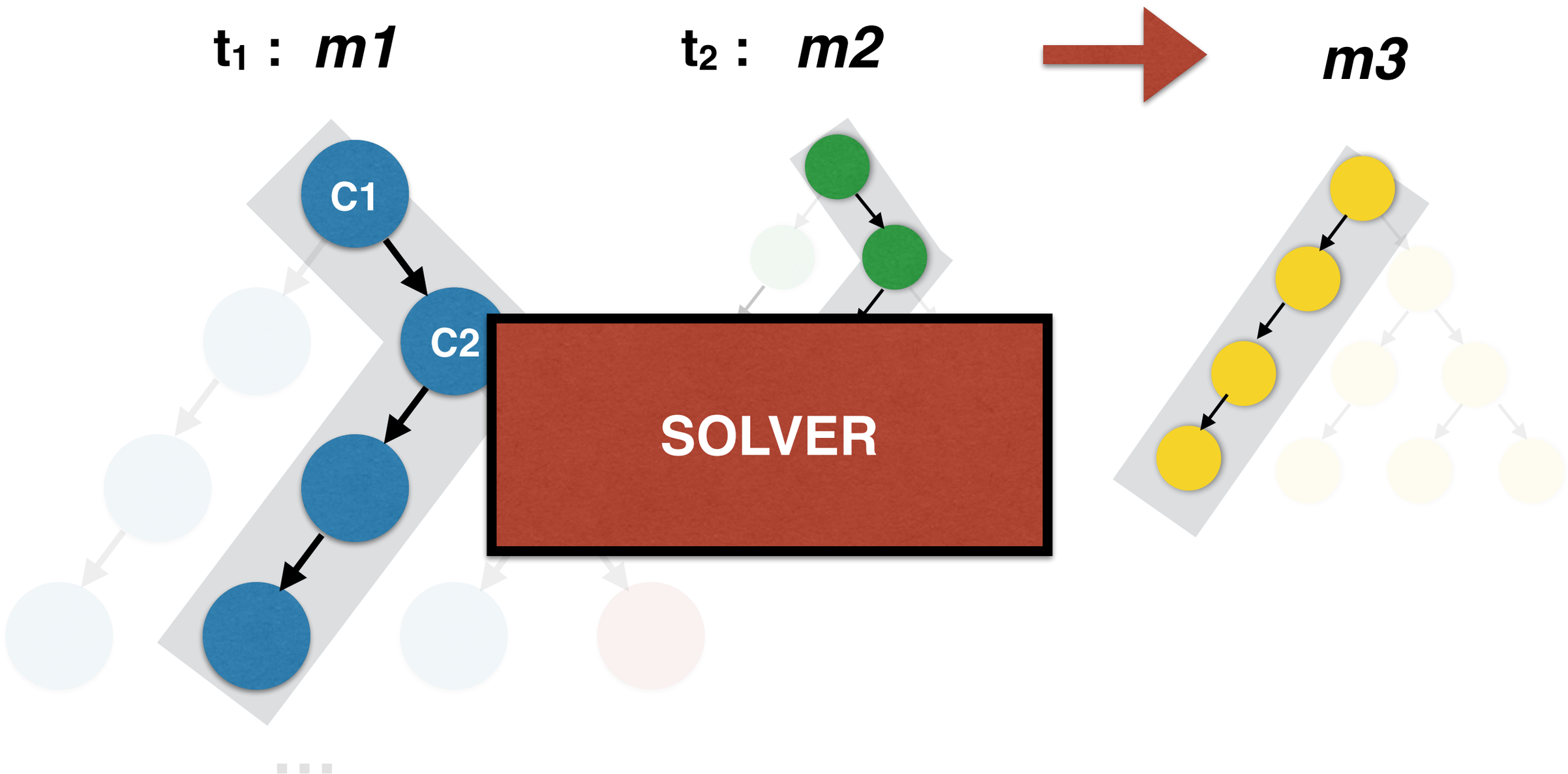
# Iteration - 2 : Concrete Execution



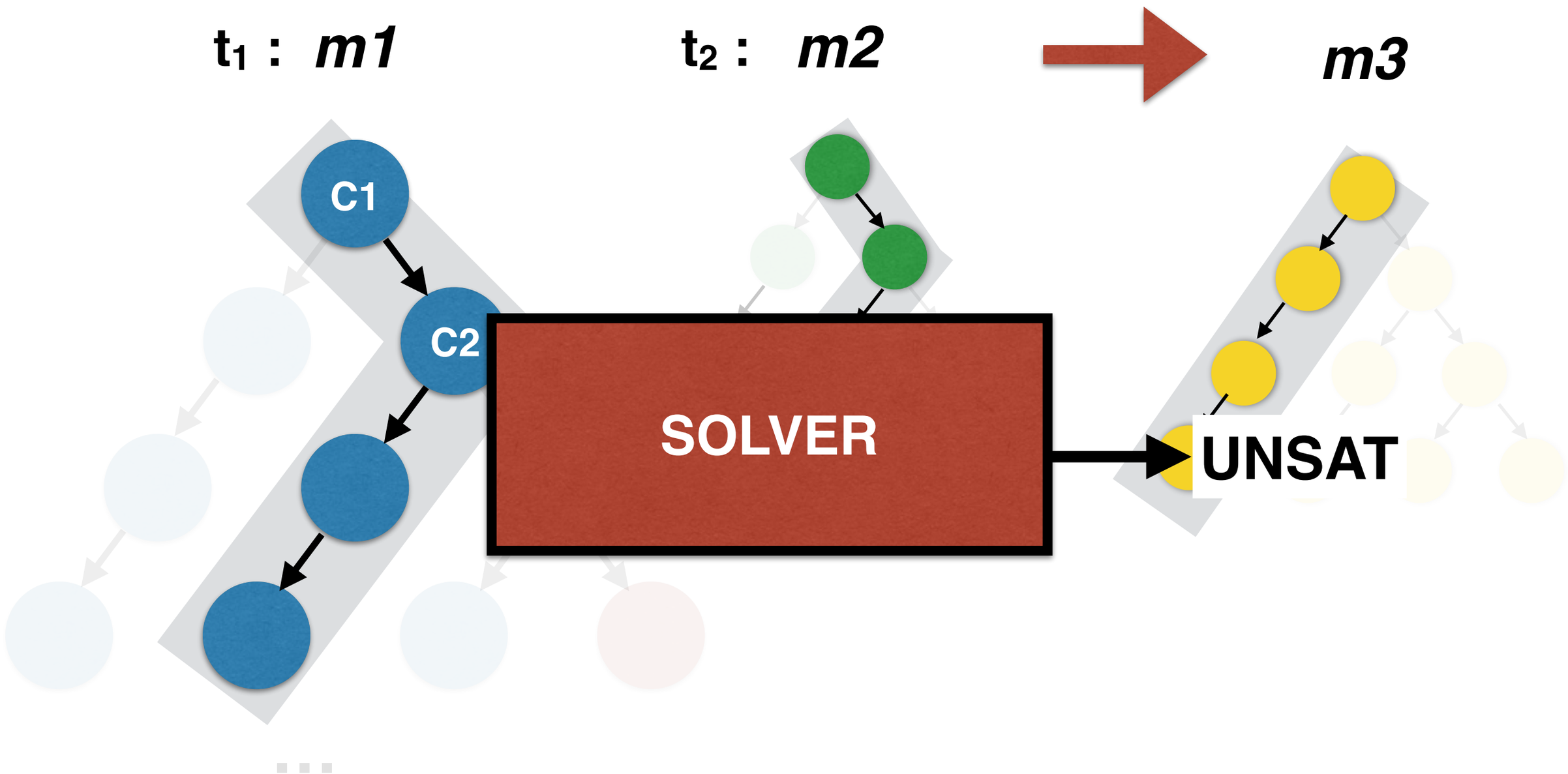
# Iteration - 2 : Concrete Execution



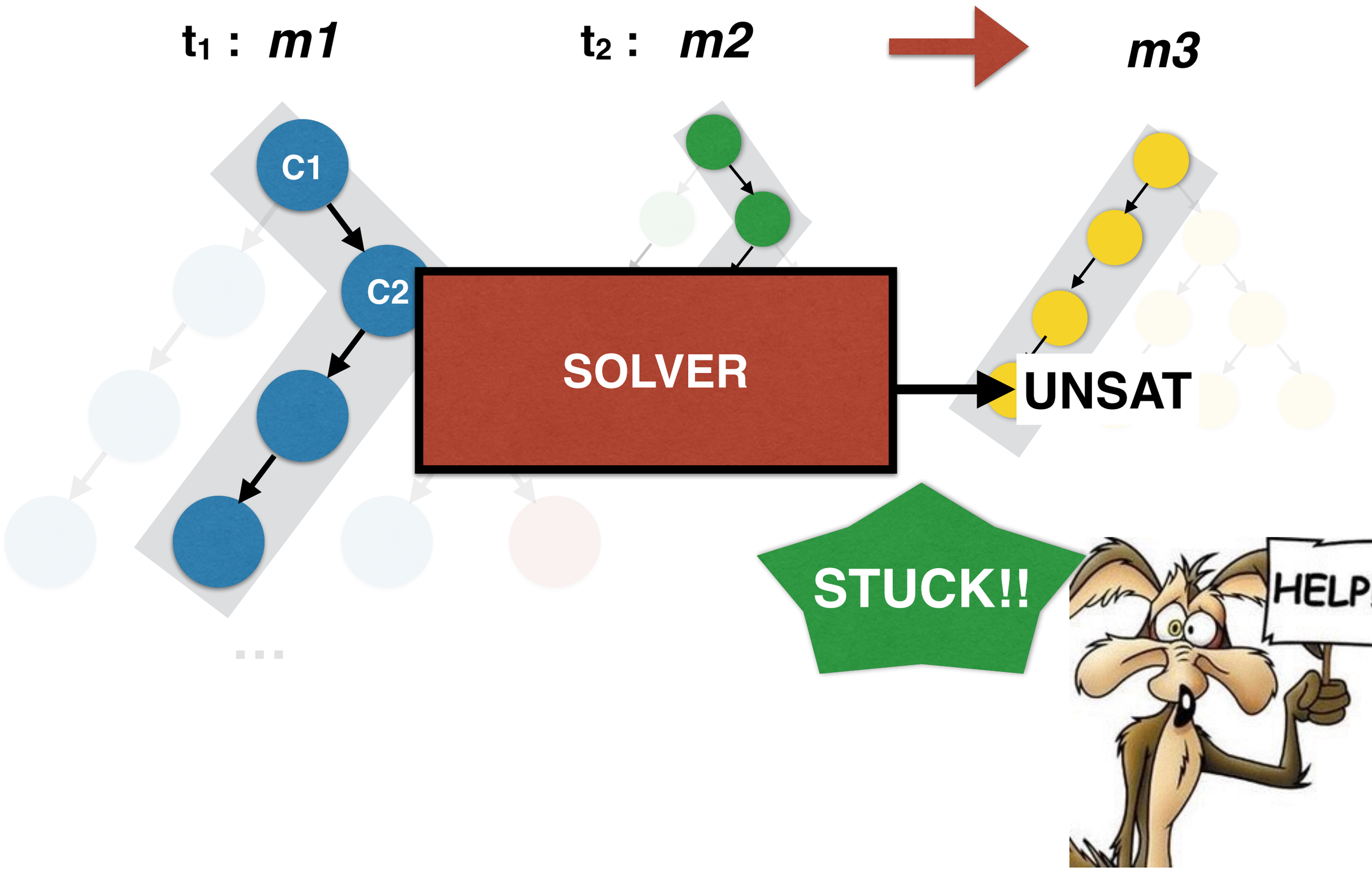
# Iteration - 2 : Concrete Execution



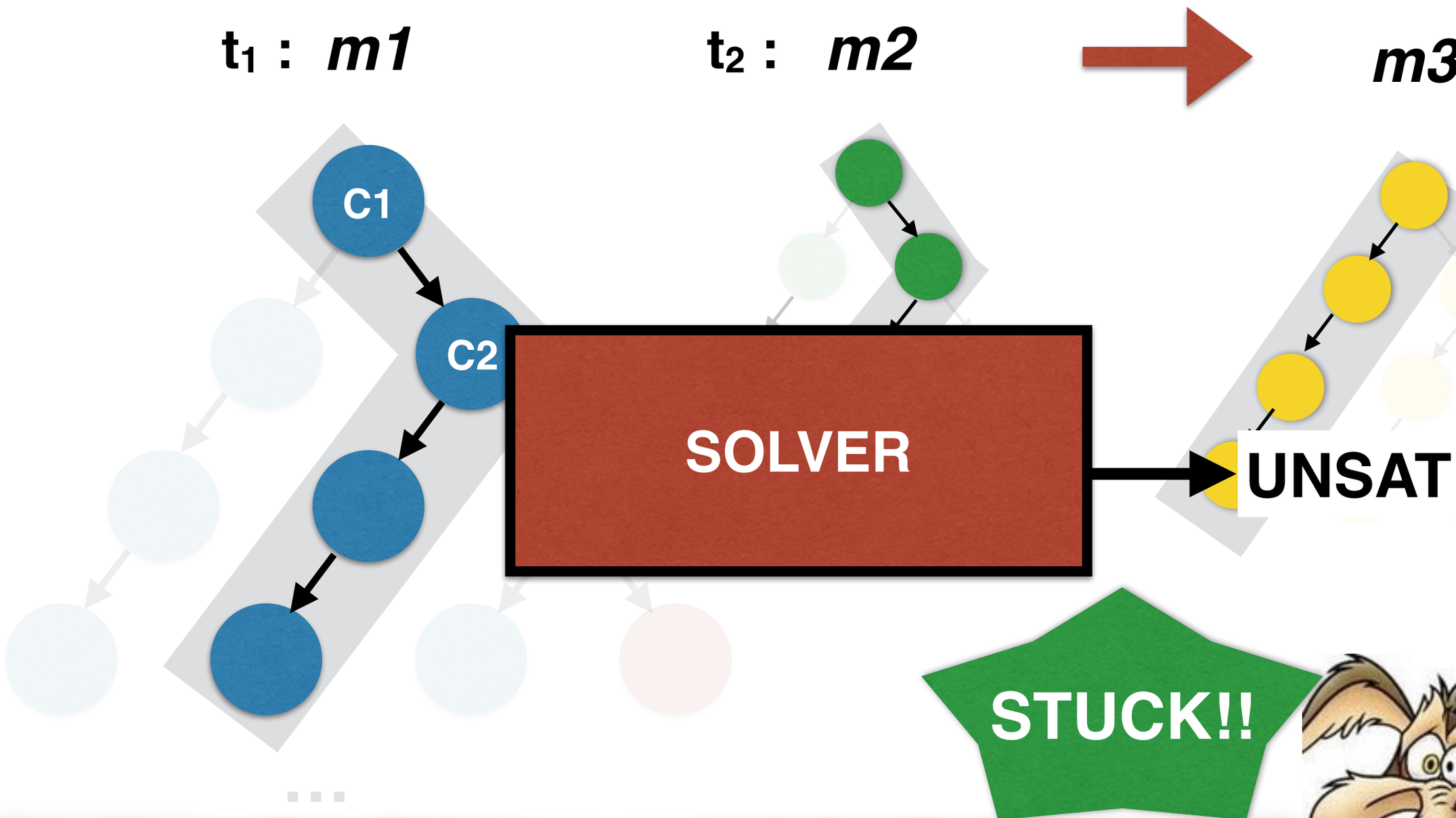
# Iteration - 2 : Concrete Execution



# Iteration - 2 : Concrete Execution



# Iteration - 2 : Concrete Execution





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

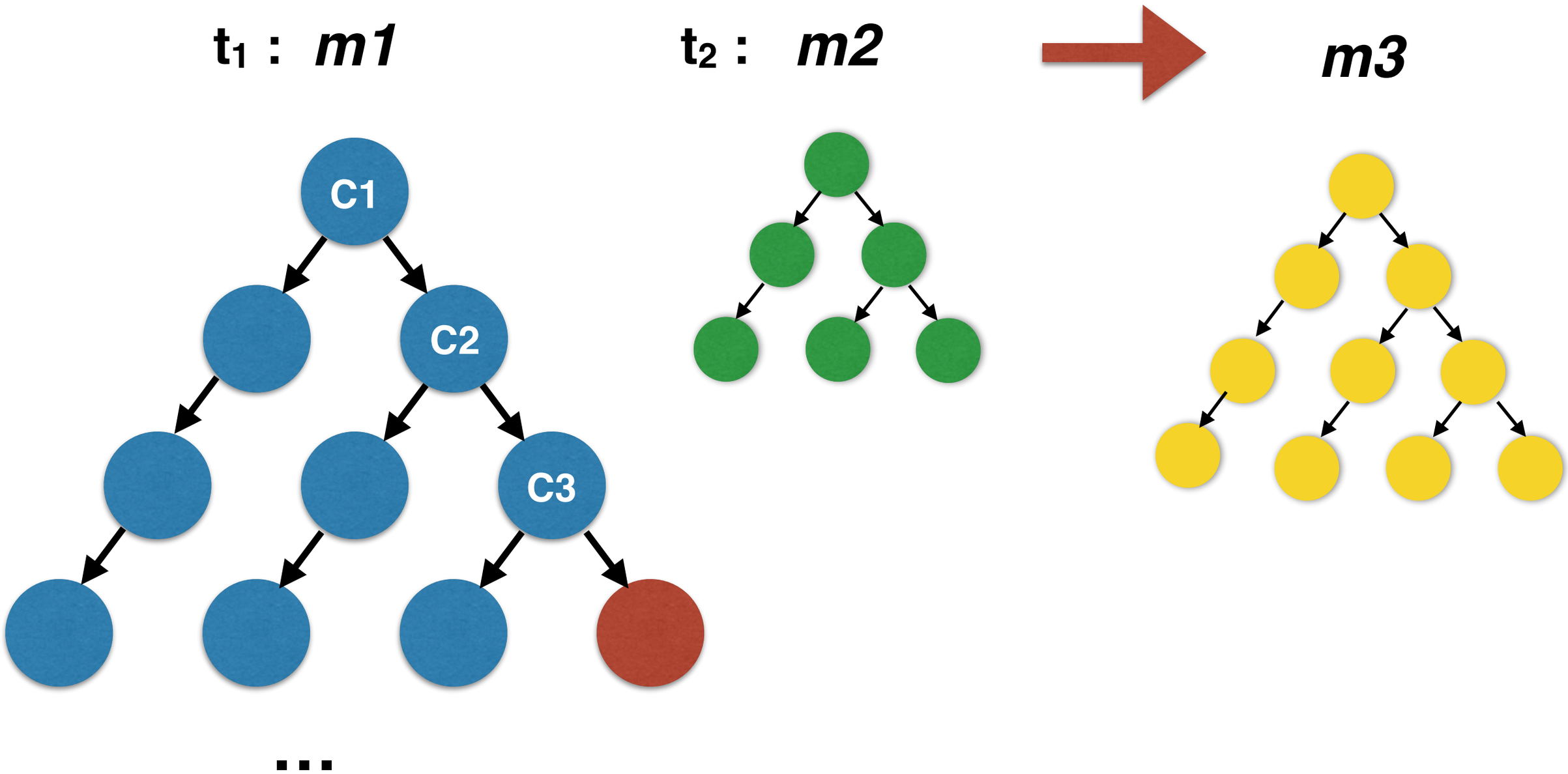
# Iteration - 3 : Static Analysis

$t_1$ : ***m1***

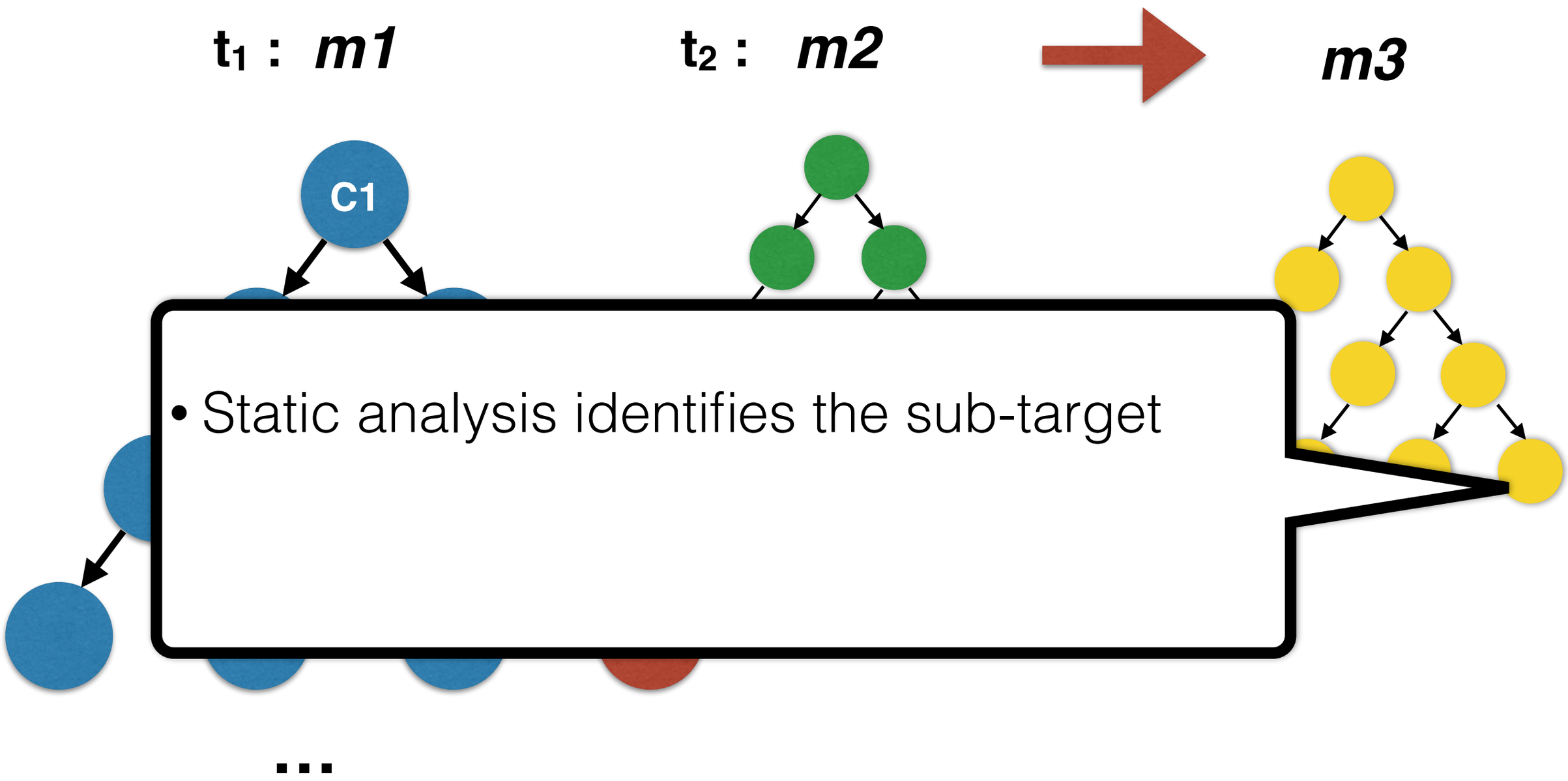
$t_2$ : ***m2***

→ ***m3***

# Iteration - 3 : Static Analysis

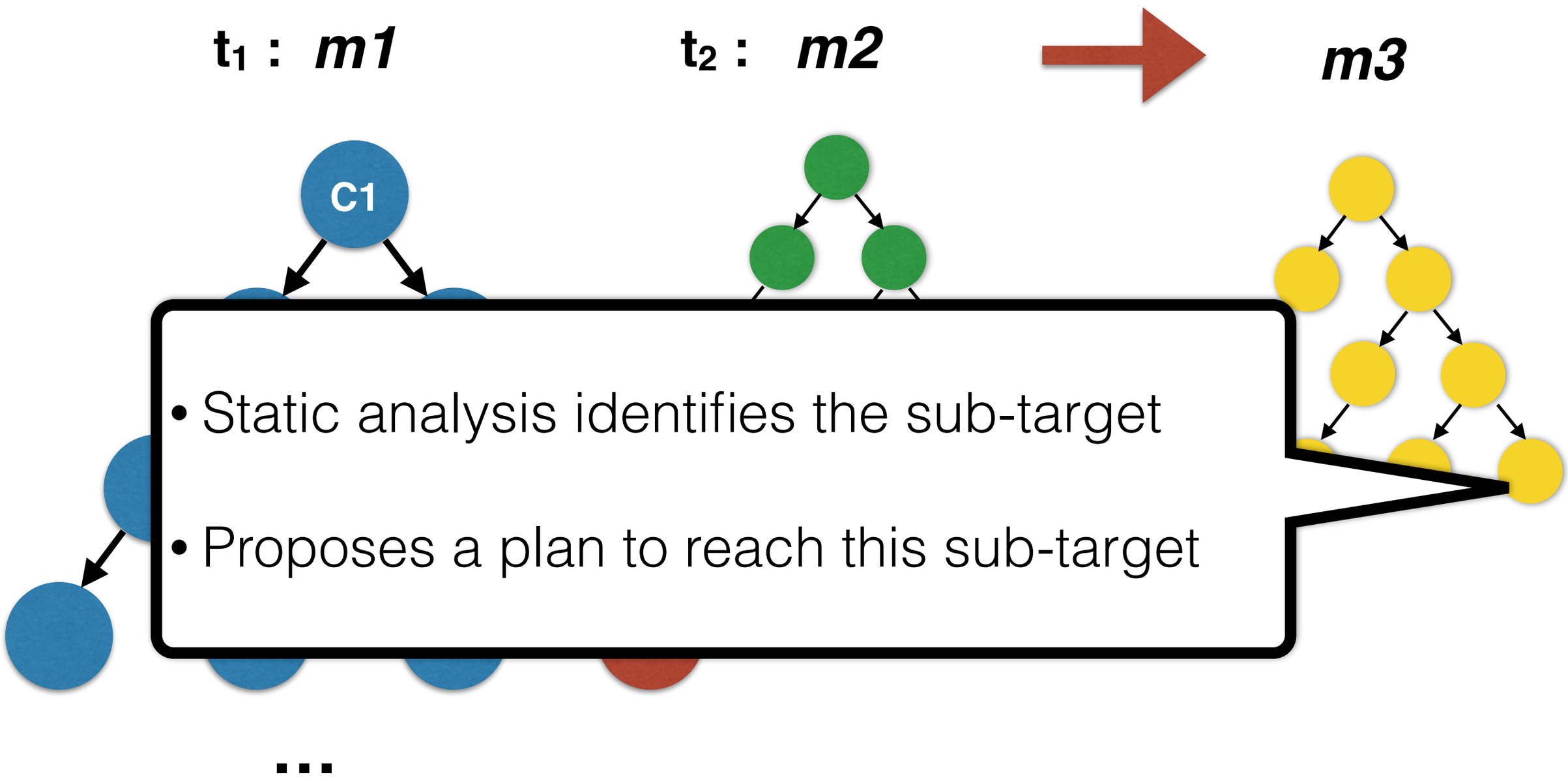


# Iteration - 3 : Static Analysis

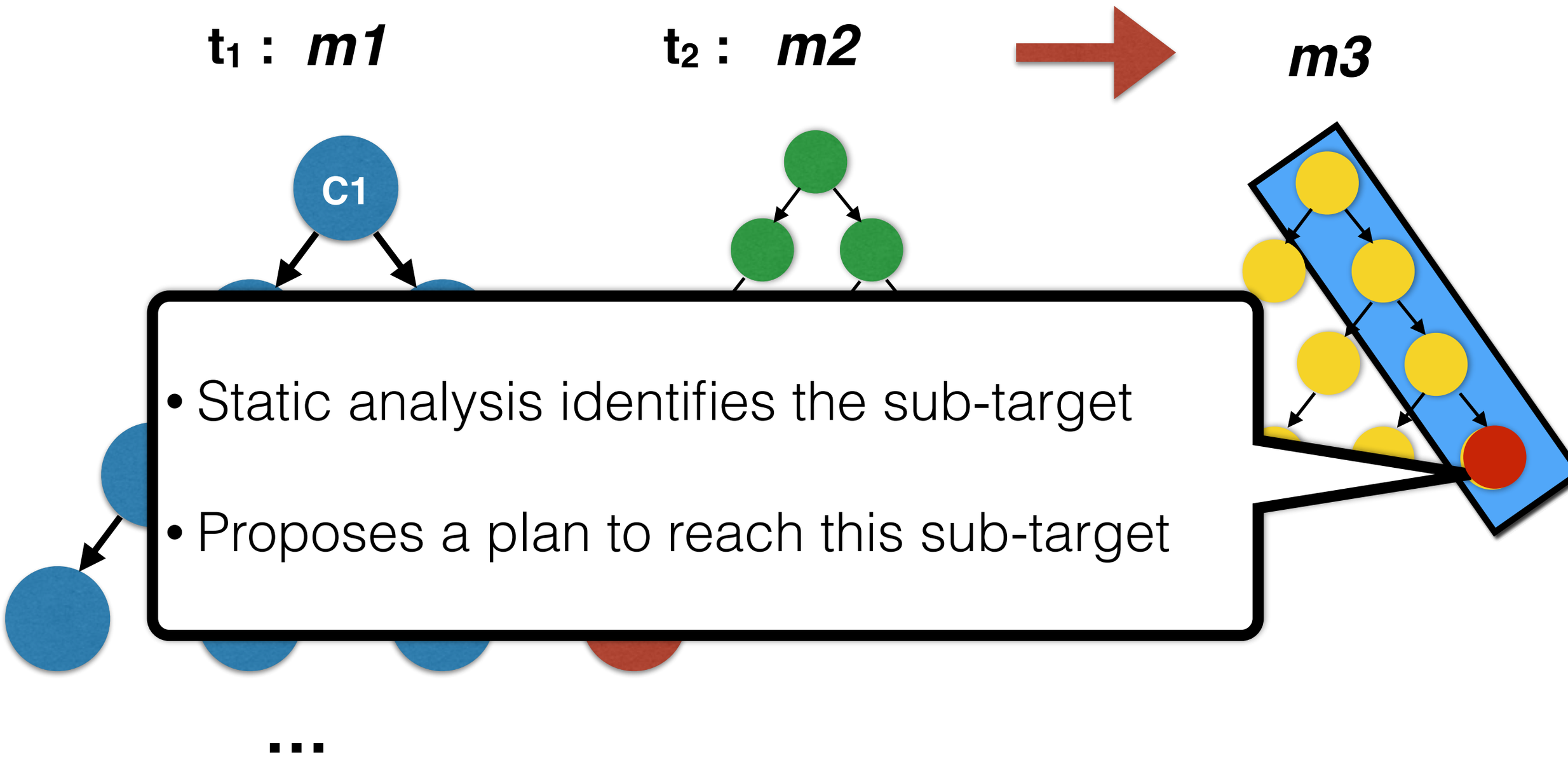



- Static analysis identifies the sub-target

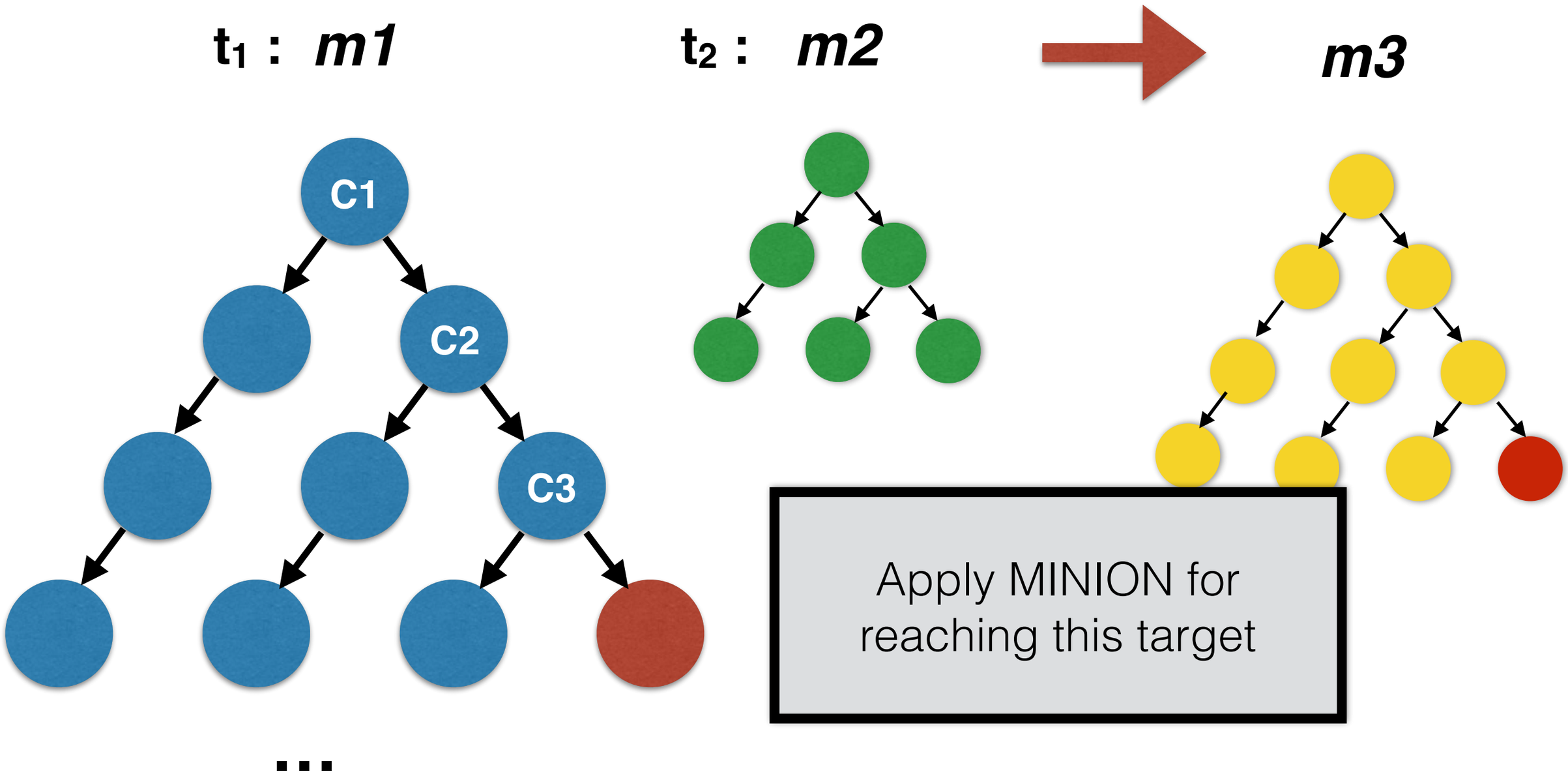
# Iteration - 3 : Static Analysis



# Iteration - 3 : Static Analysis



# Iteration - 3 : Concrete Execution



After a few more iterations…

# Iteration - 4 : Concrete Execution

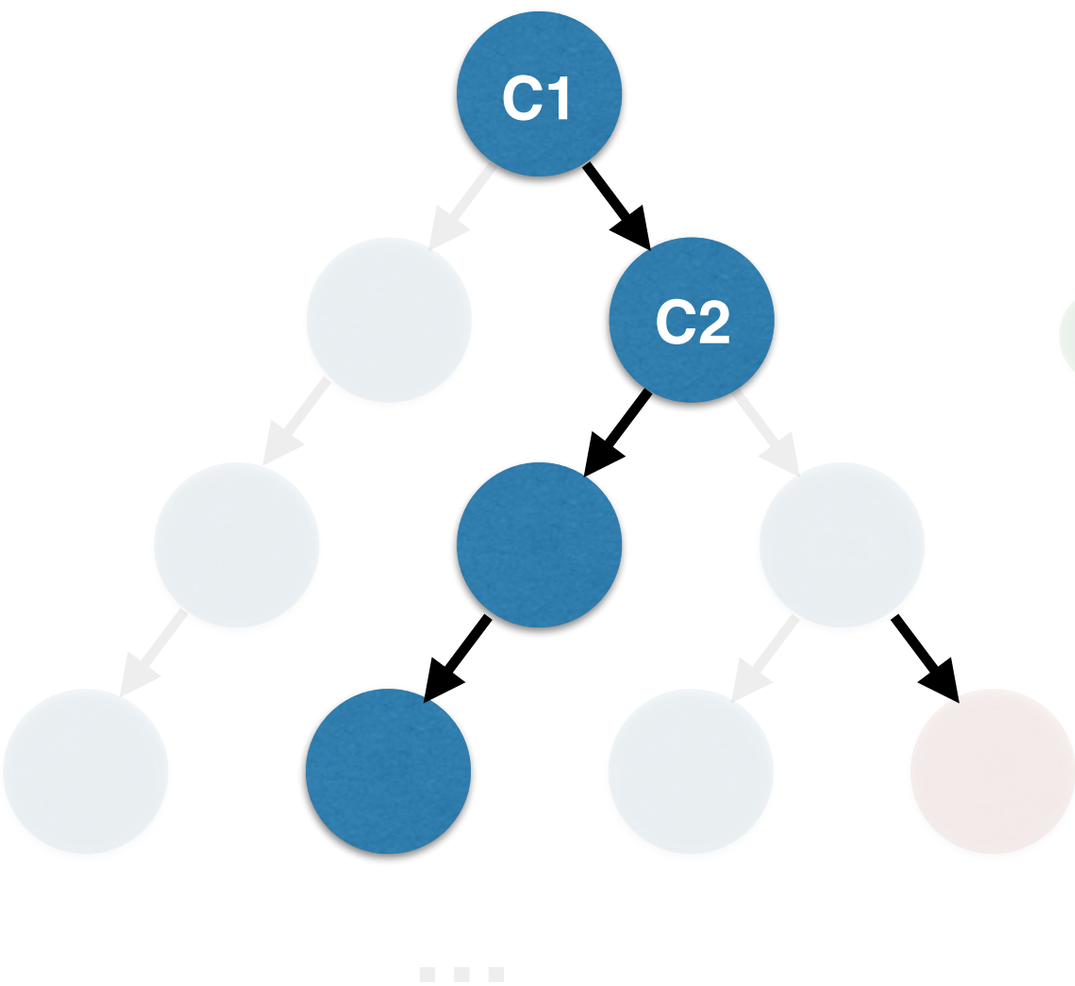
$t_1$ : ***m1***

$t_2$ : ***m2***

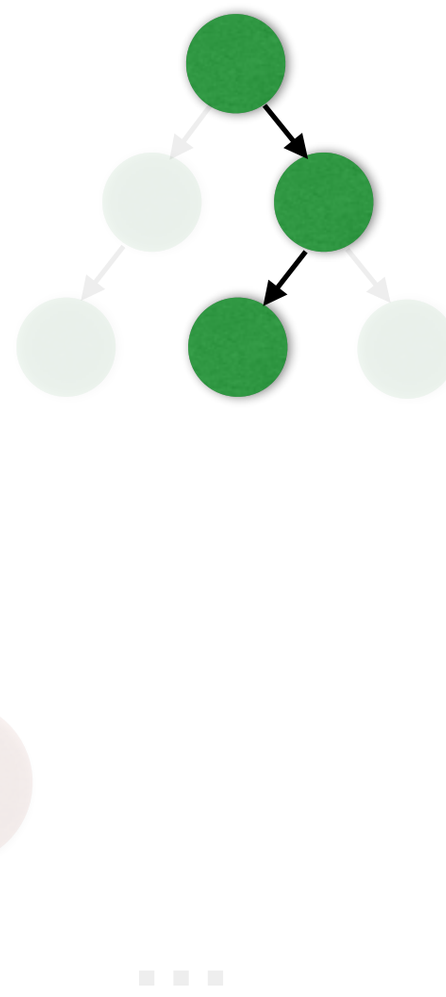
$t_3$ : ***m3***

# Iteration - 4 : Concrete Execution

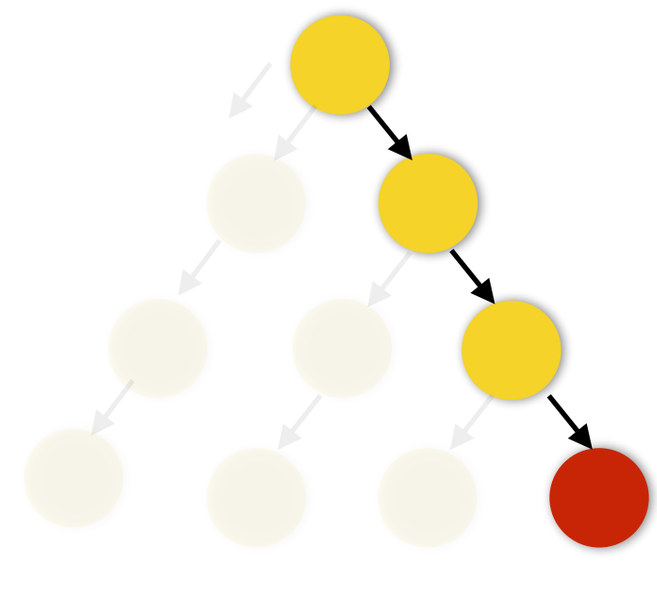
$t_1$ : ***m1***



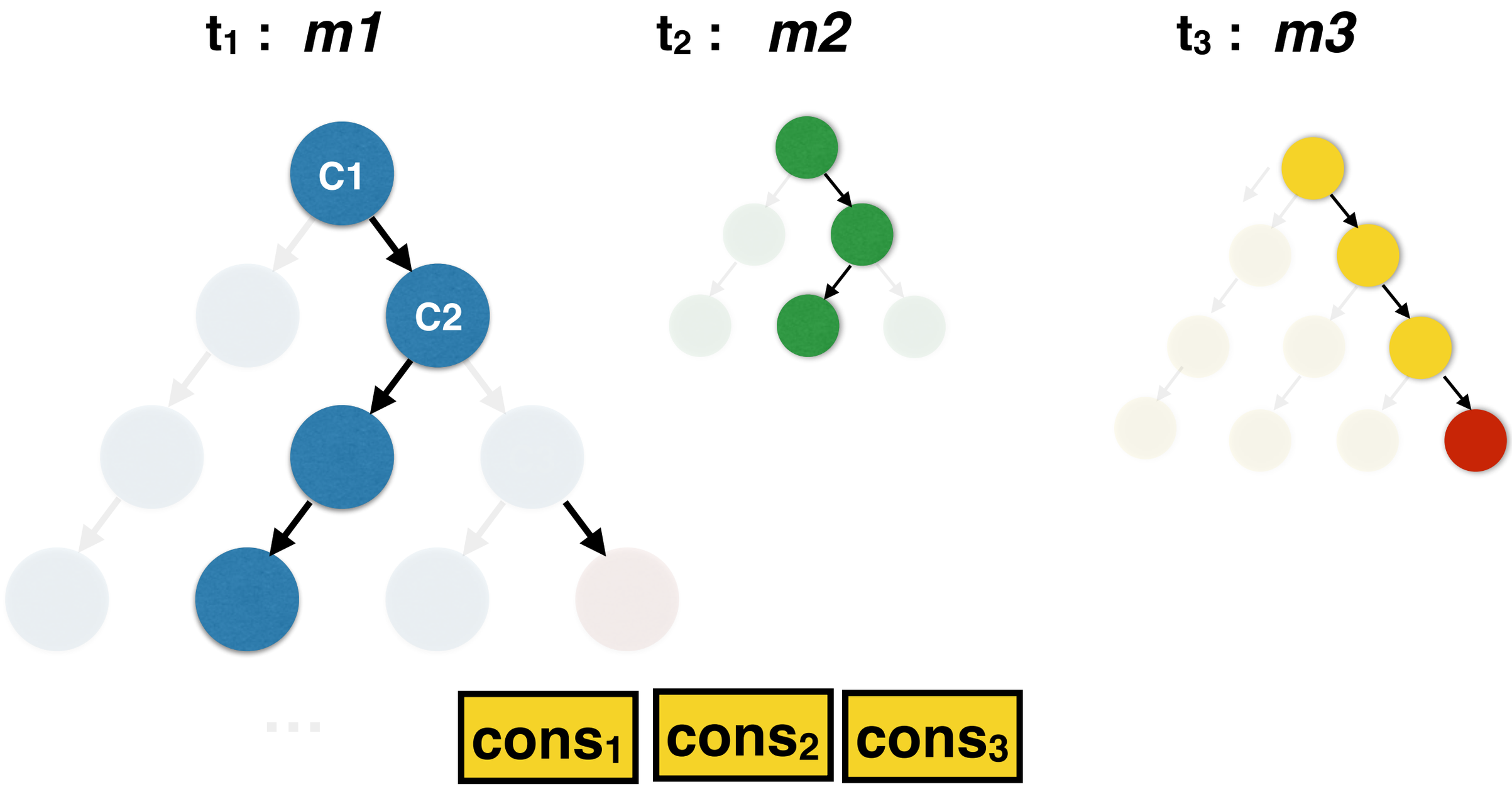

$t_2$ : ***m2***




$t_3$ : ***m3***

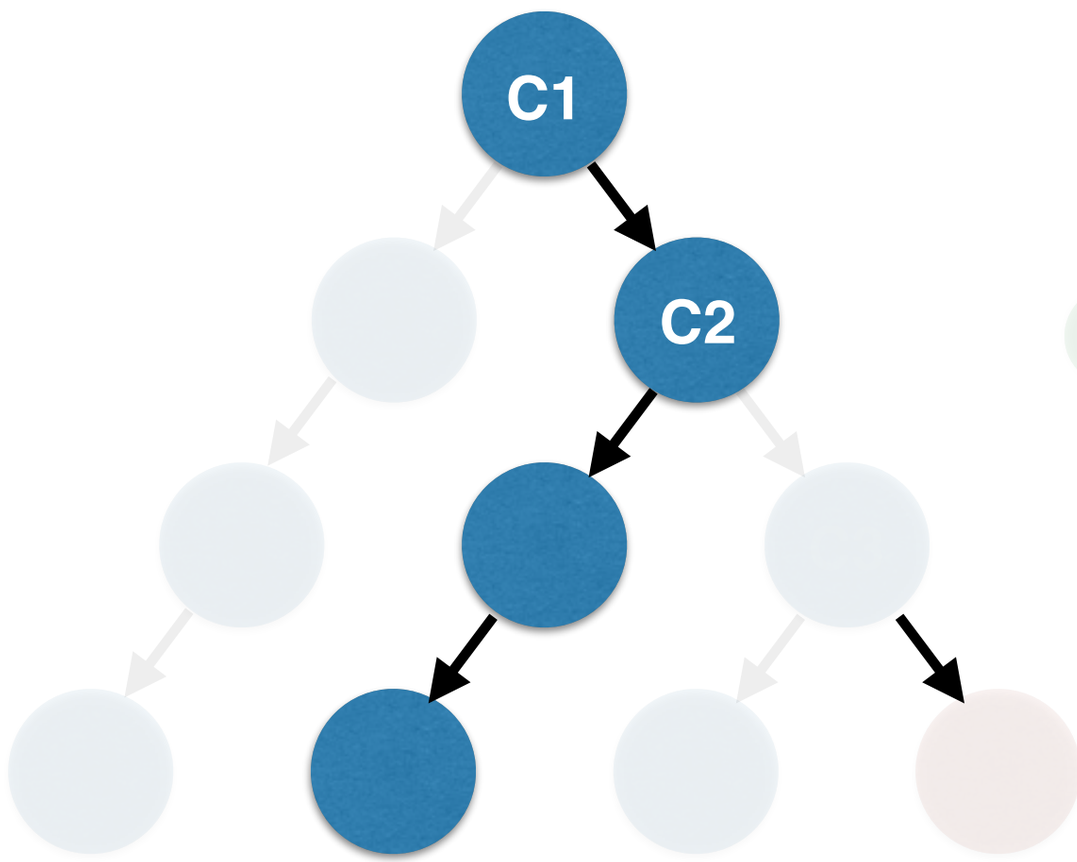


# Iteration - 4 : Concrete Execution

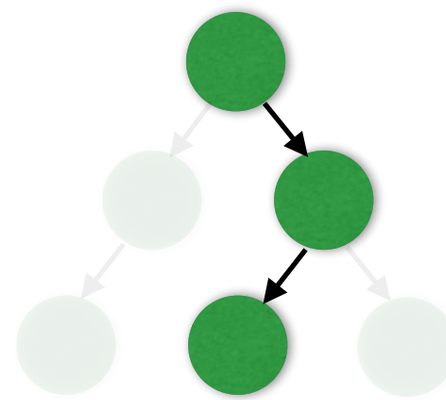


# Iteration - 4 : Concrete Execution

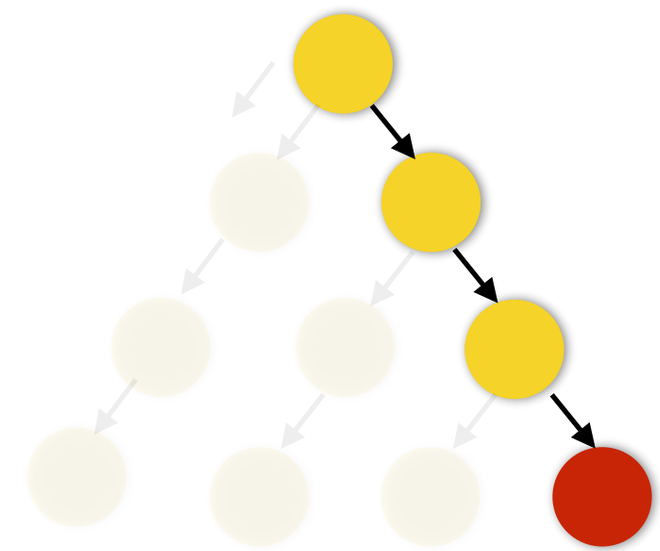
$t_1$ : ***m1***




$t_2$ : ***m2***



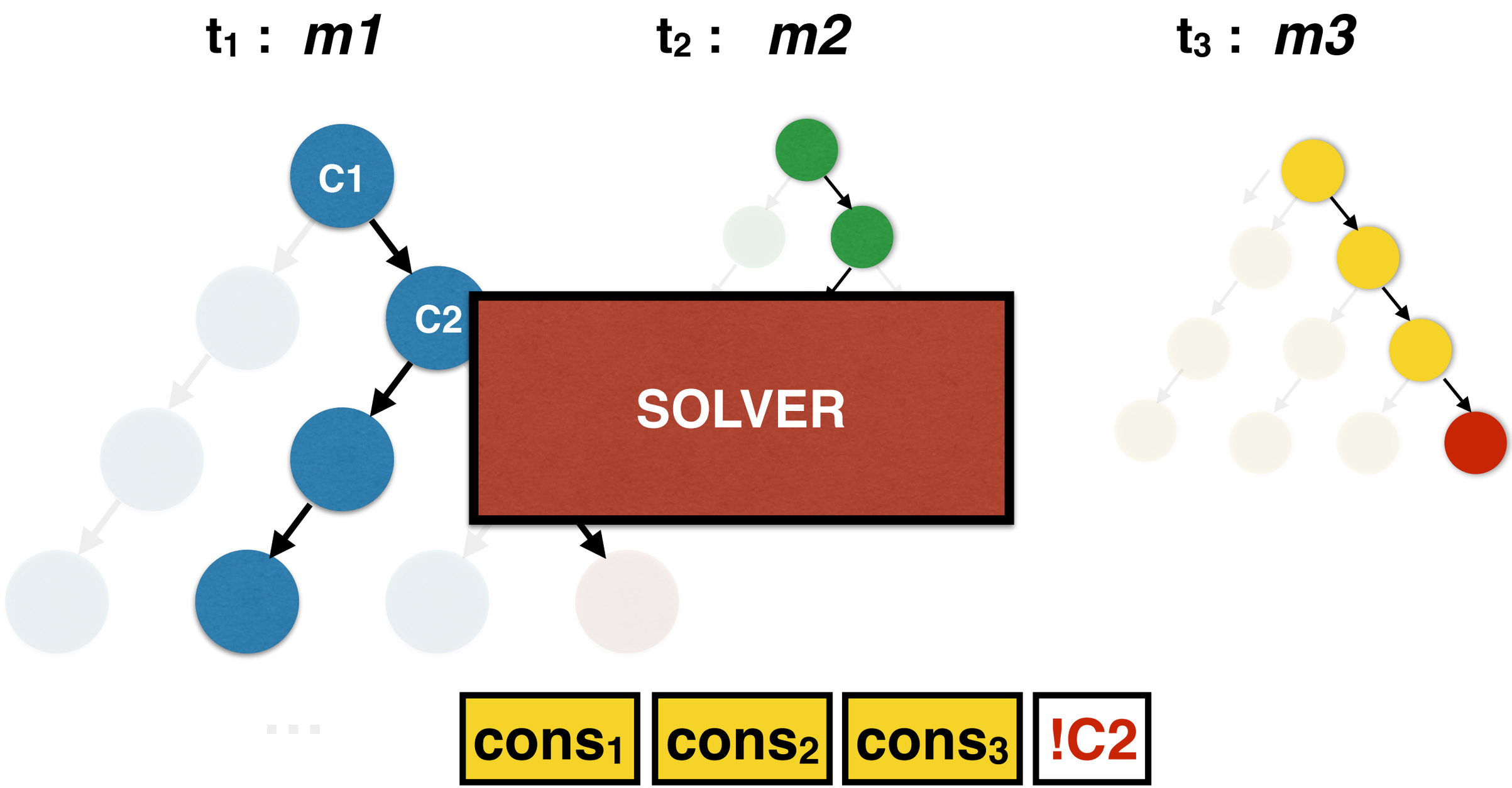
$t_3$ : ***m3***



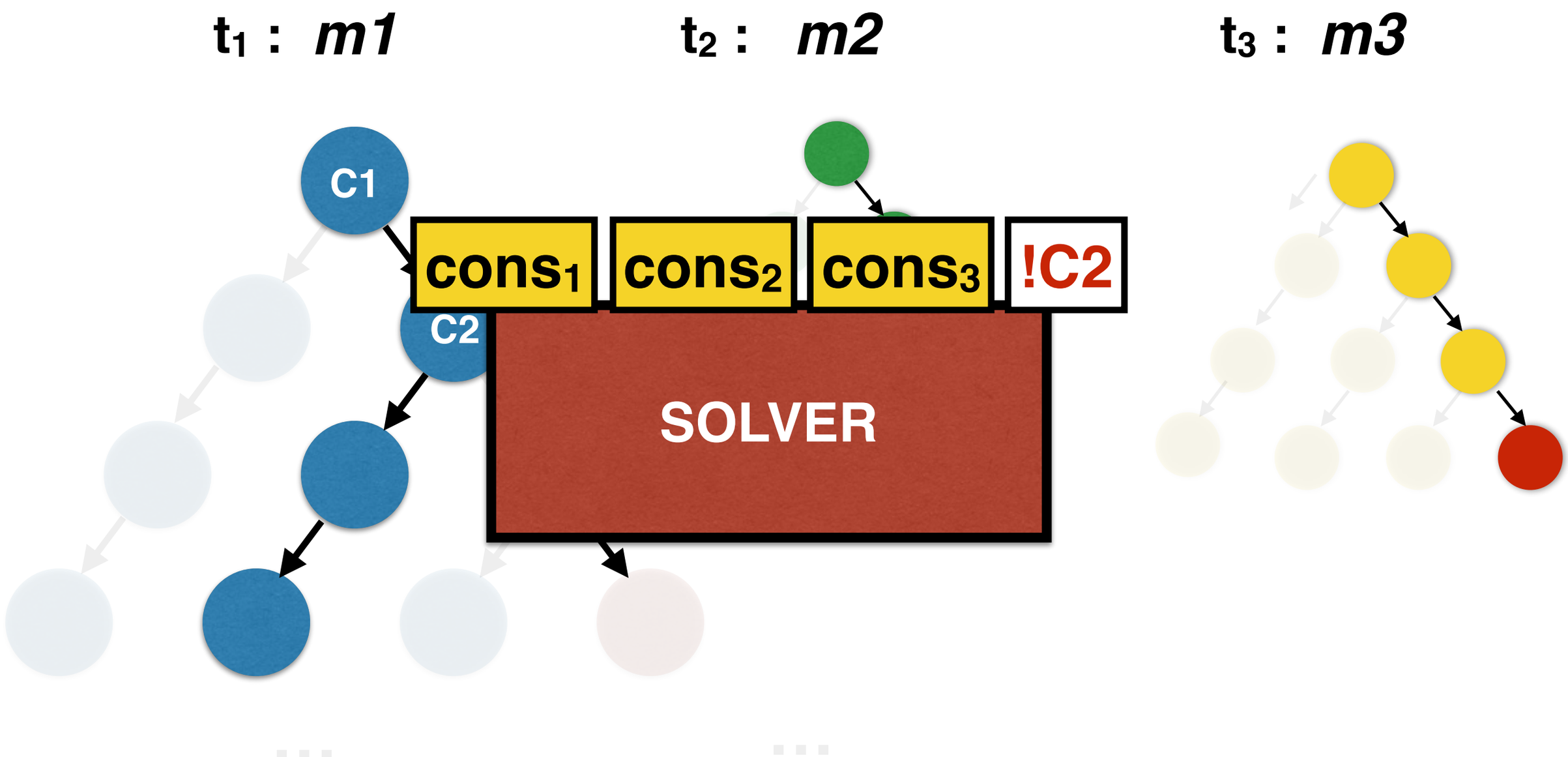
...

$cons_1$ $cons_2$ $cons_3$ !C2

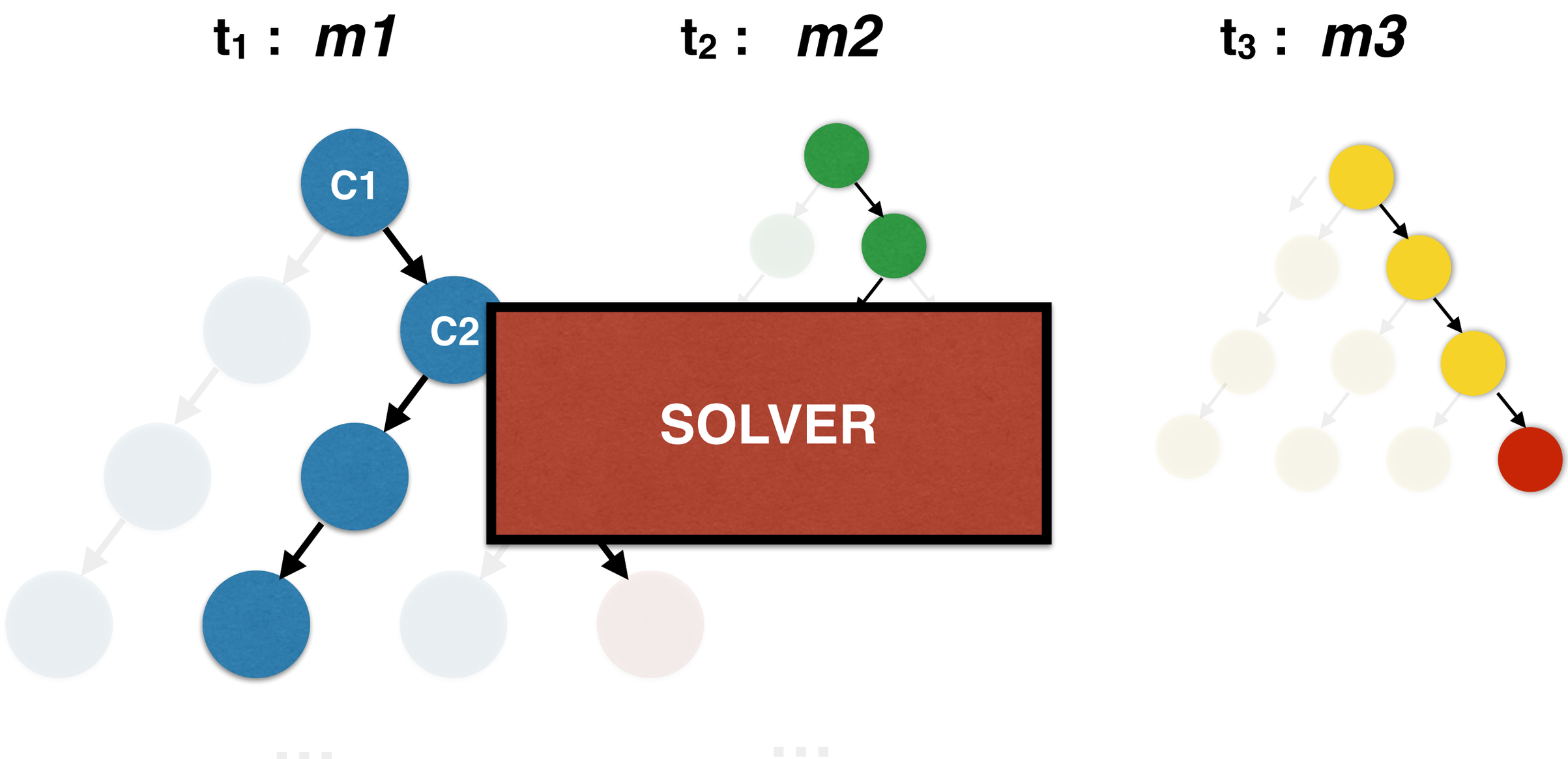
# Iteration - 4 : Concrete Execution



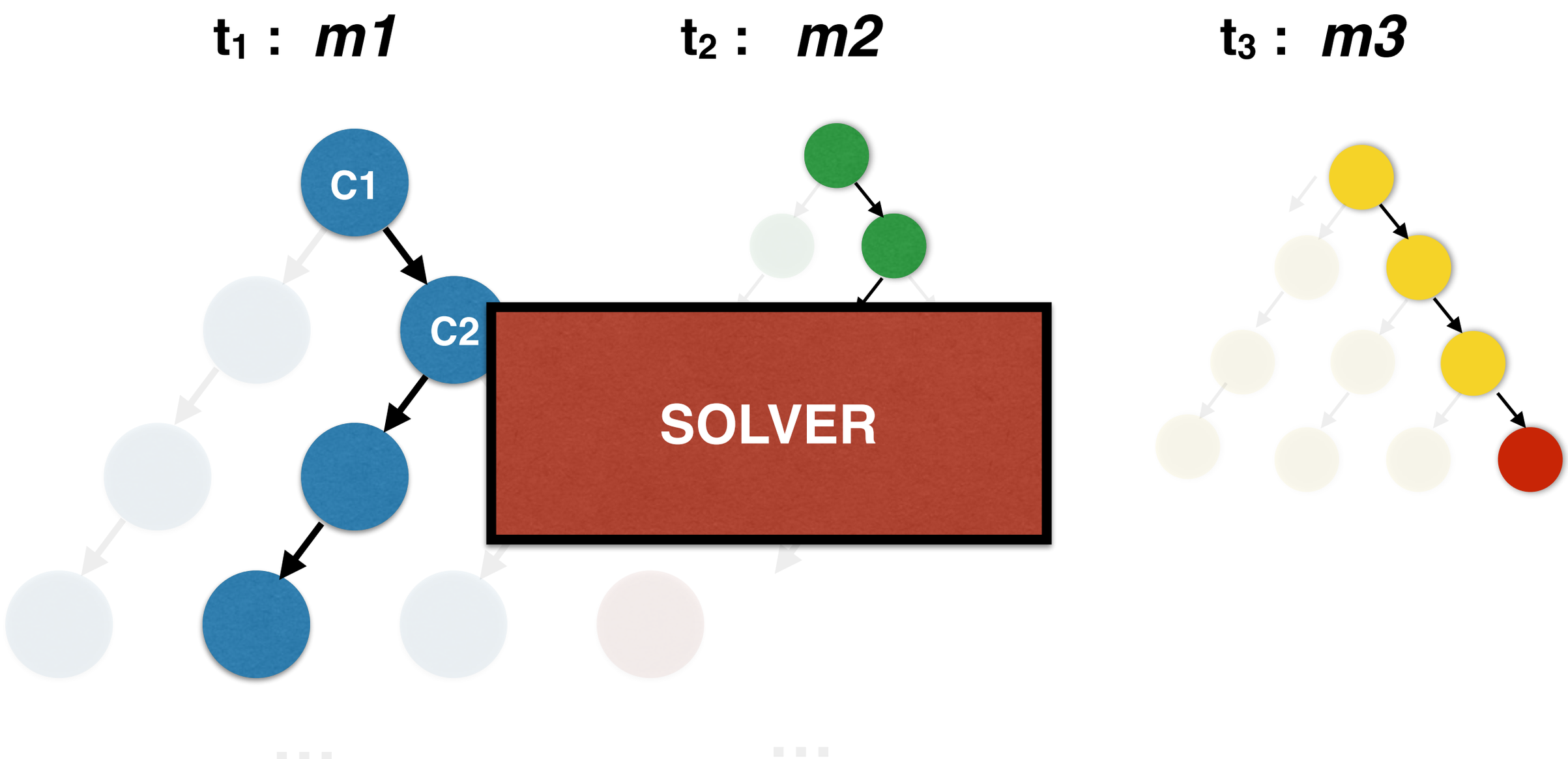
# Iteration - 4 : Concrete Execution



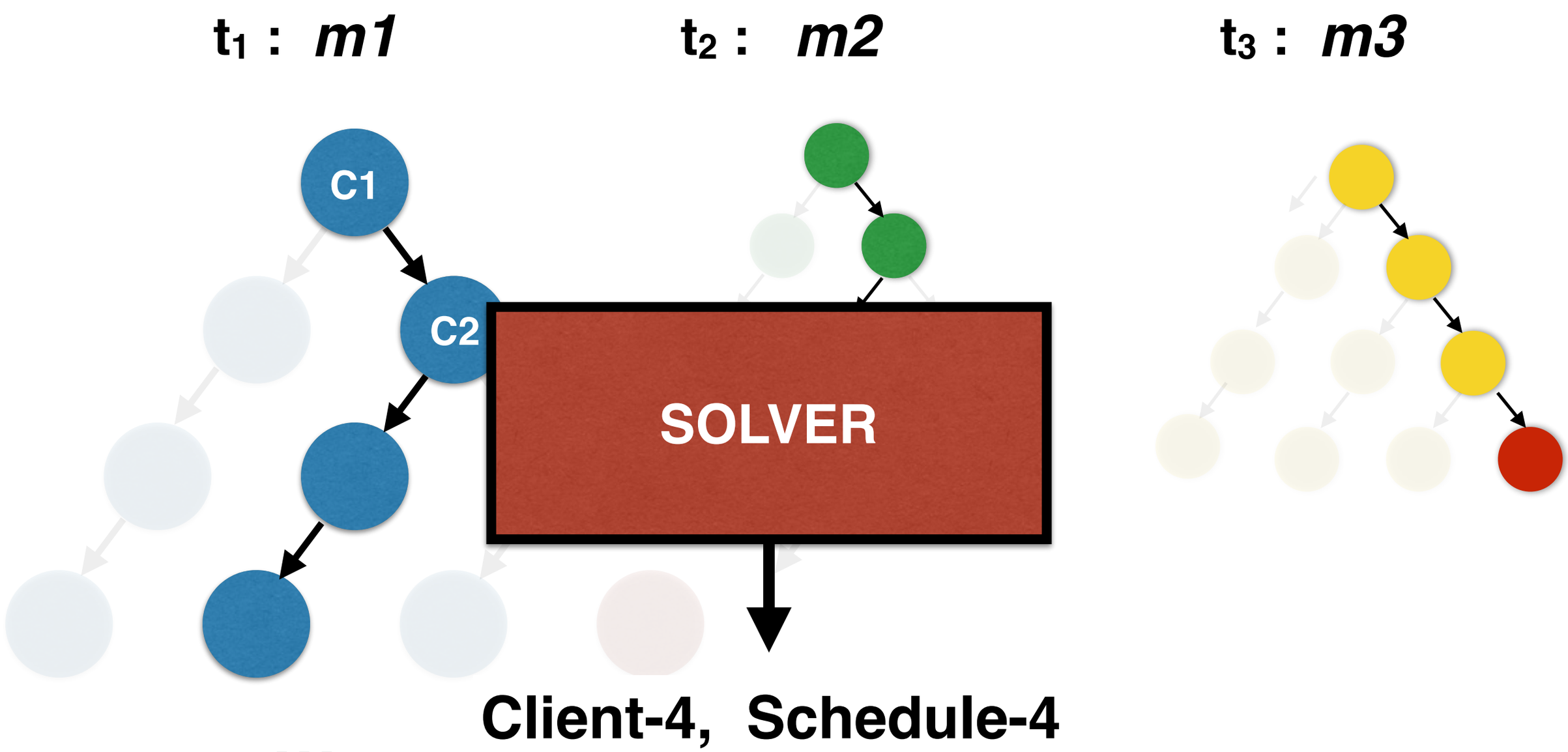
# Iteration - 4 : Concrete Execution



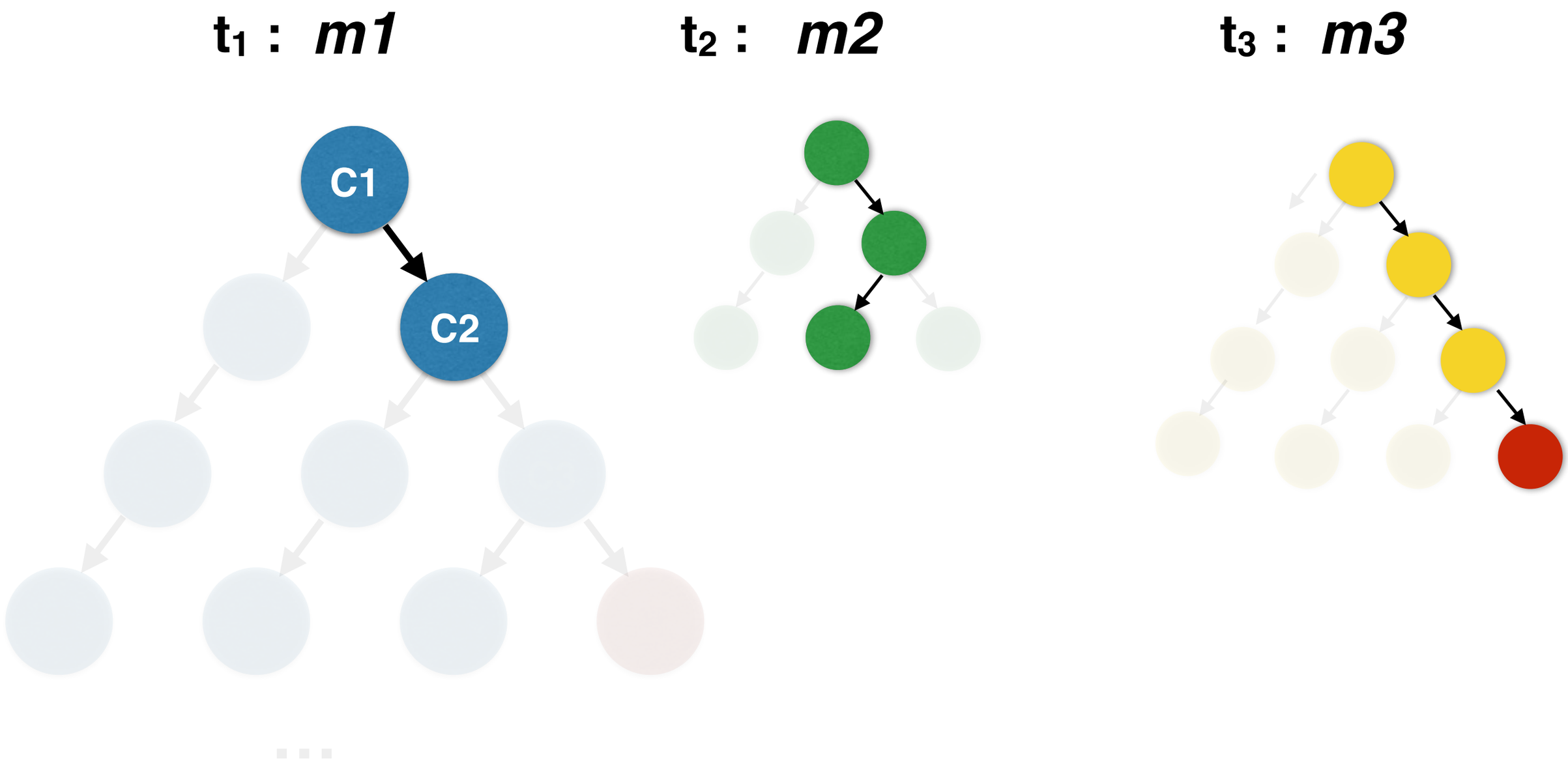
# Iteration - 4 : Concrete Execution



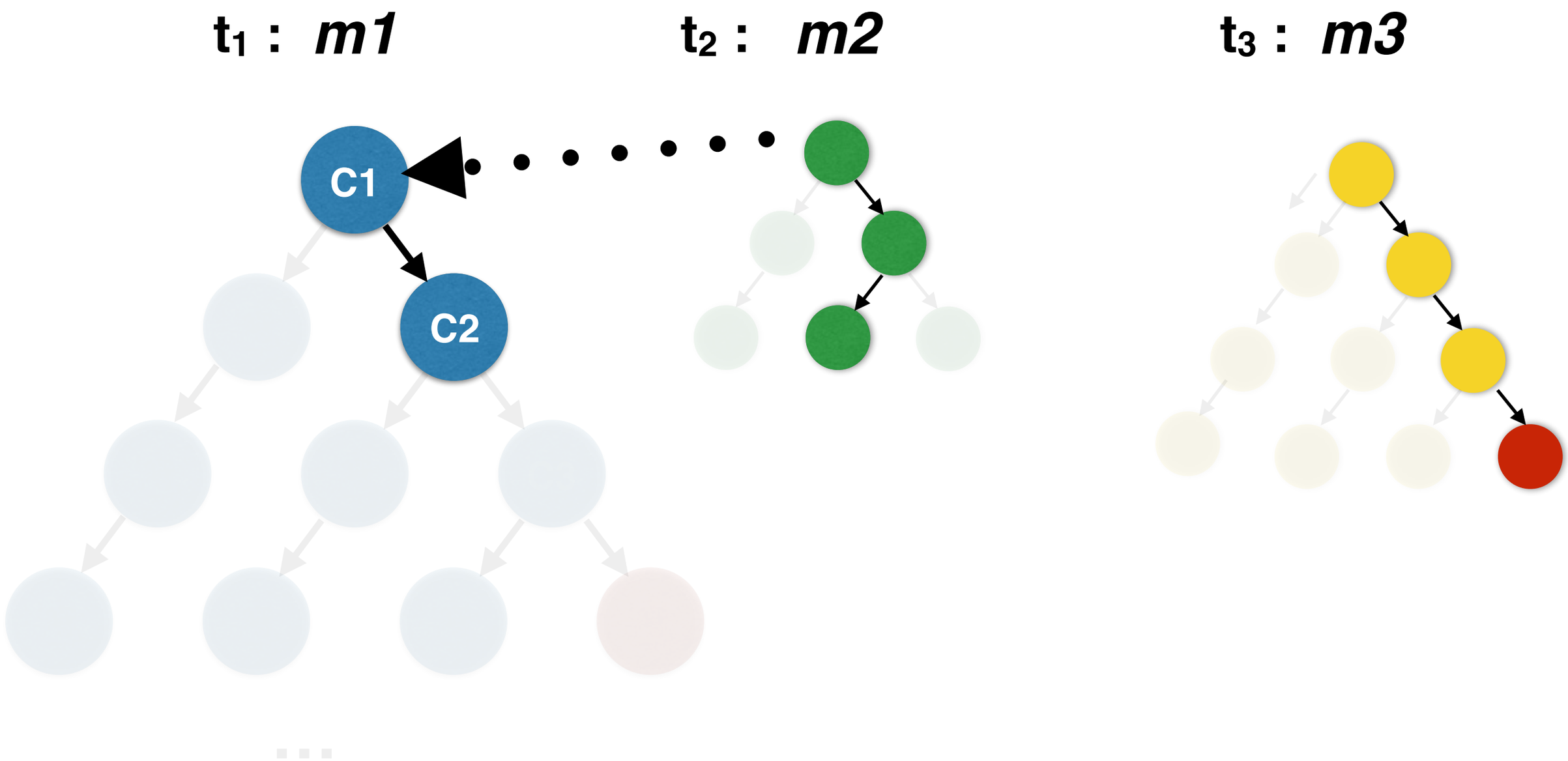
# Iteration - 4 : Concrete Execution



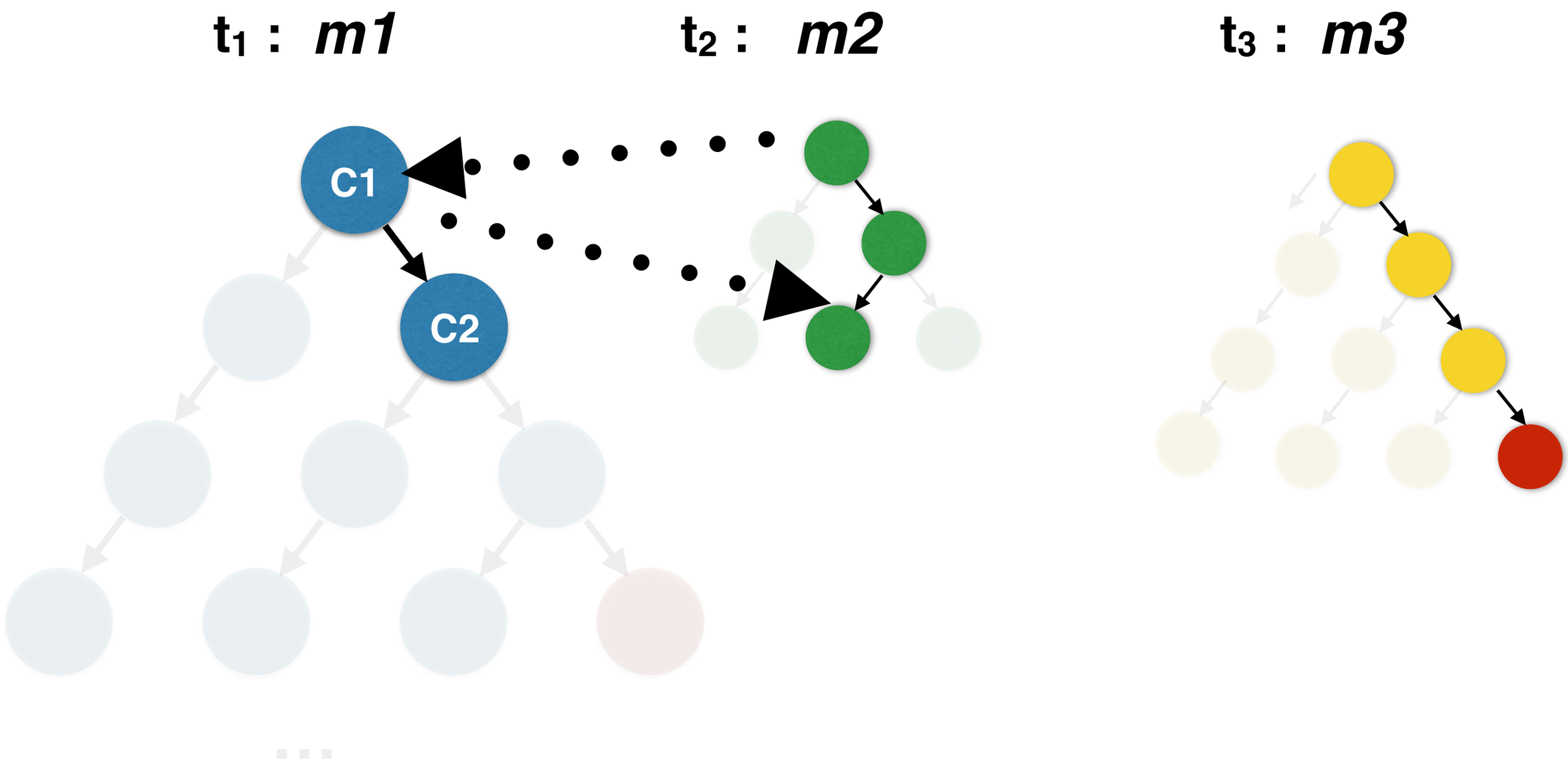
# Iteration - 4 : Concrete Execution



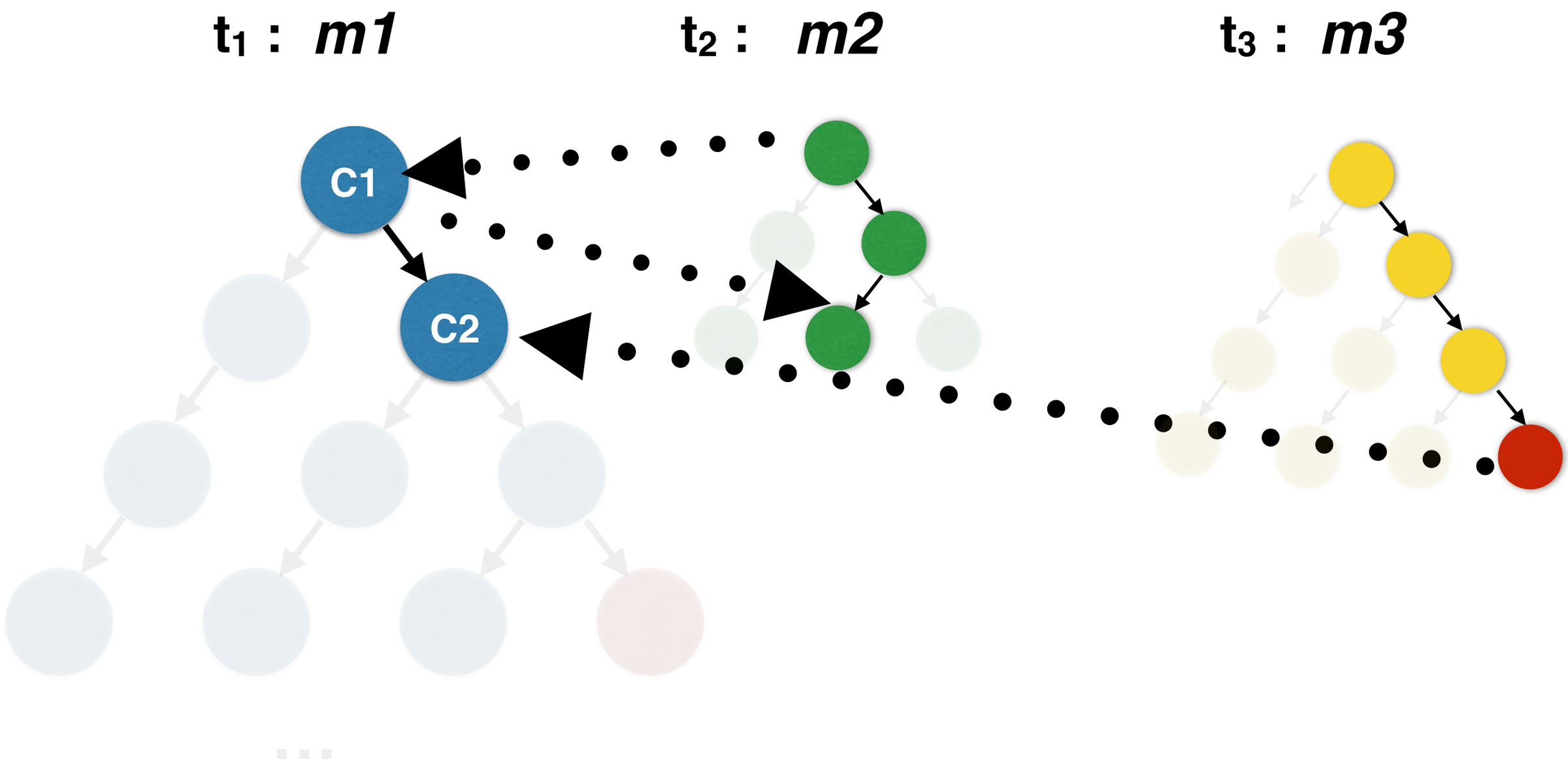
# Iteration - 4 : Concrete Execution



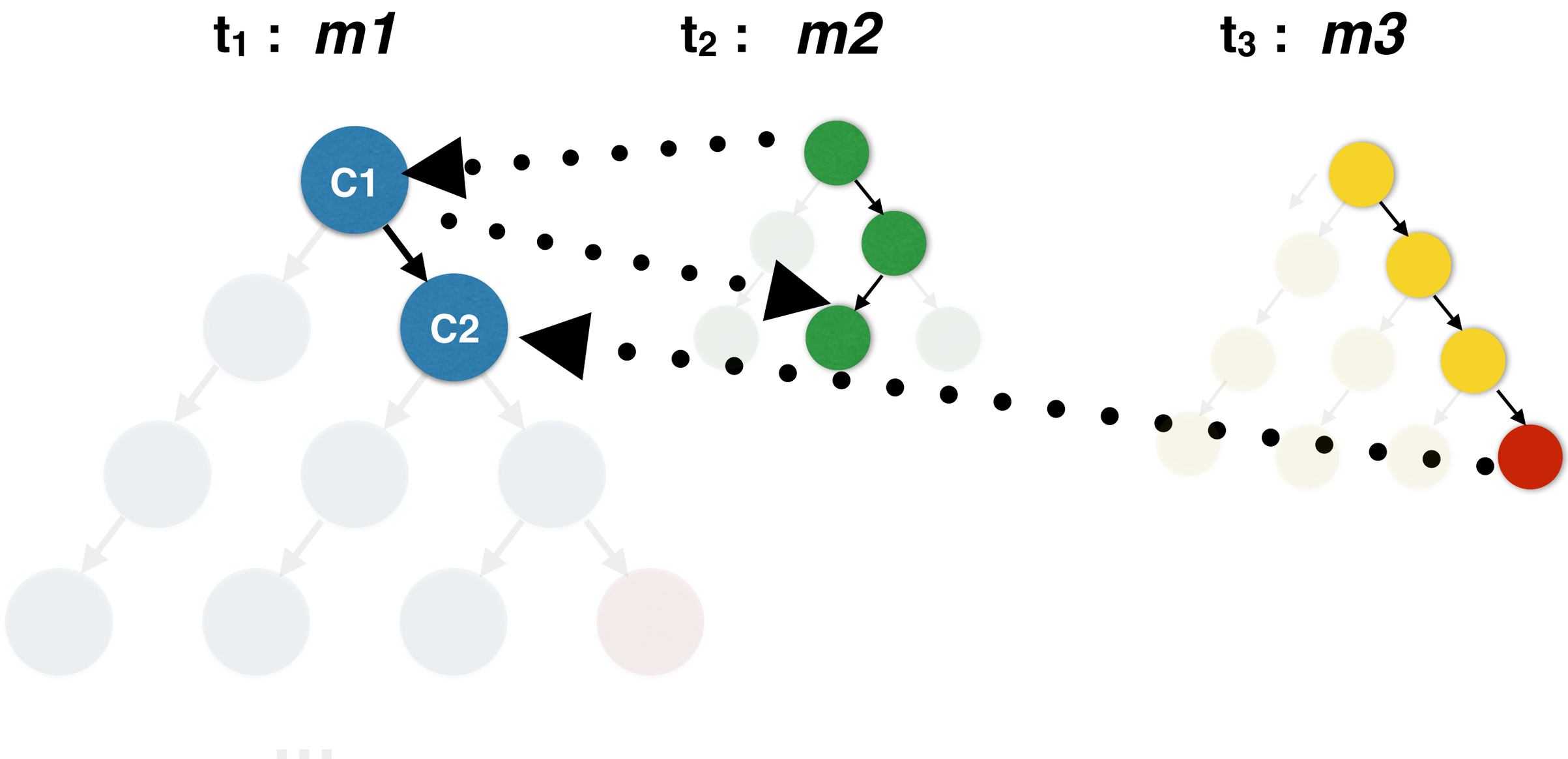
# Iteration - 4 : Concrete Execution



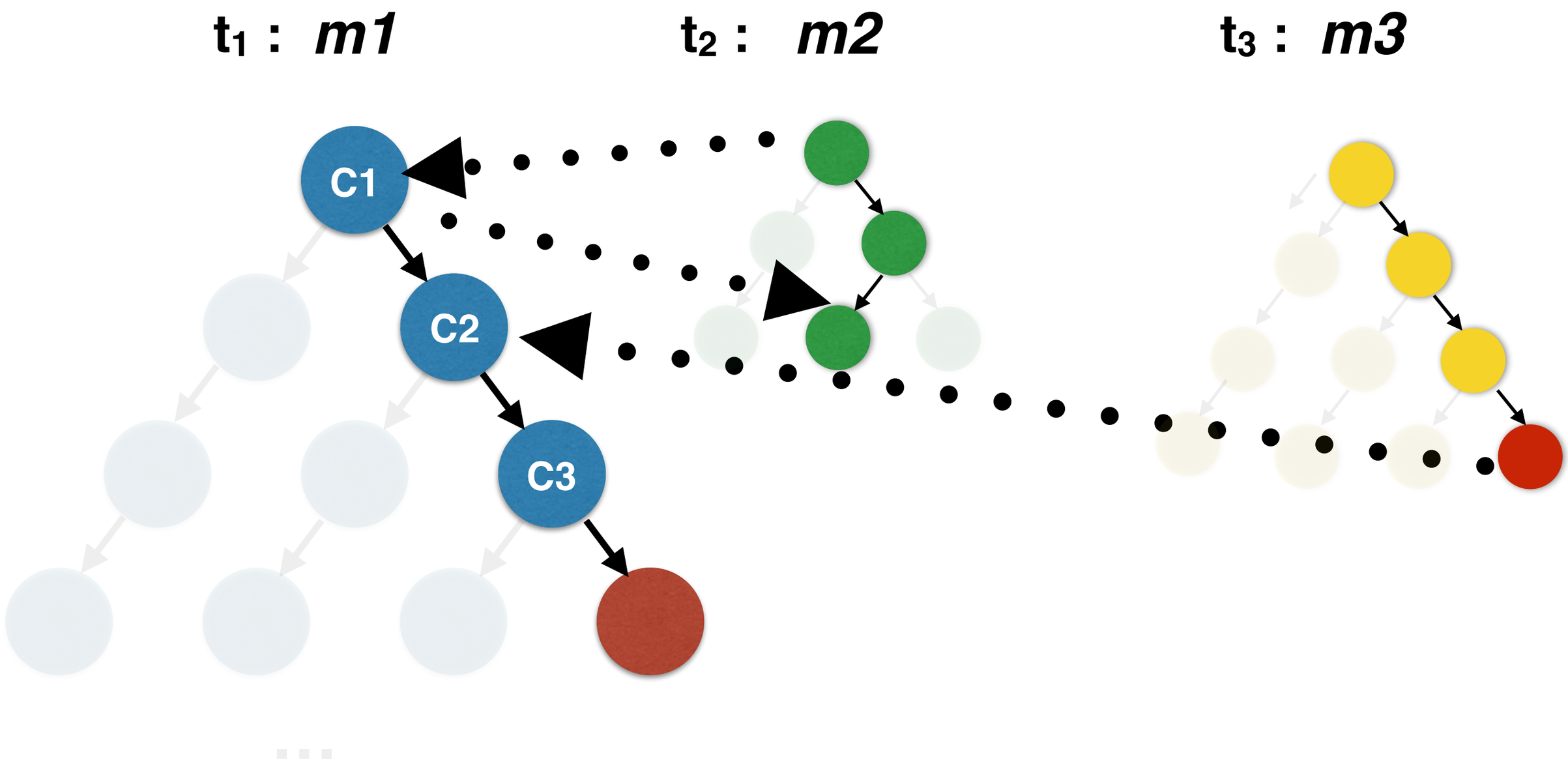
# Iteration - 4 : Concrete Execution



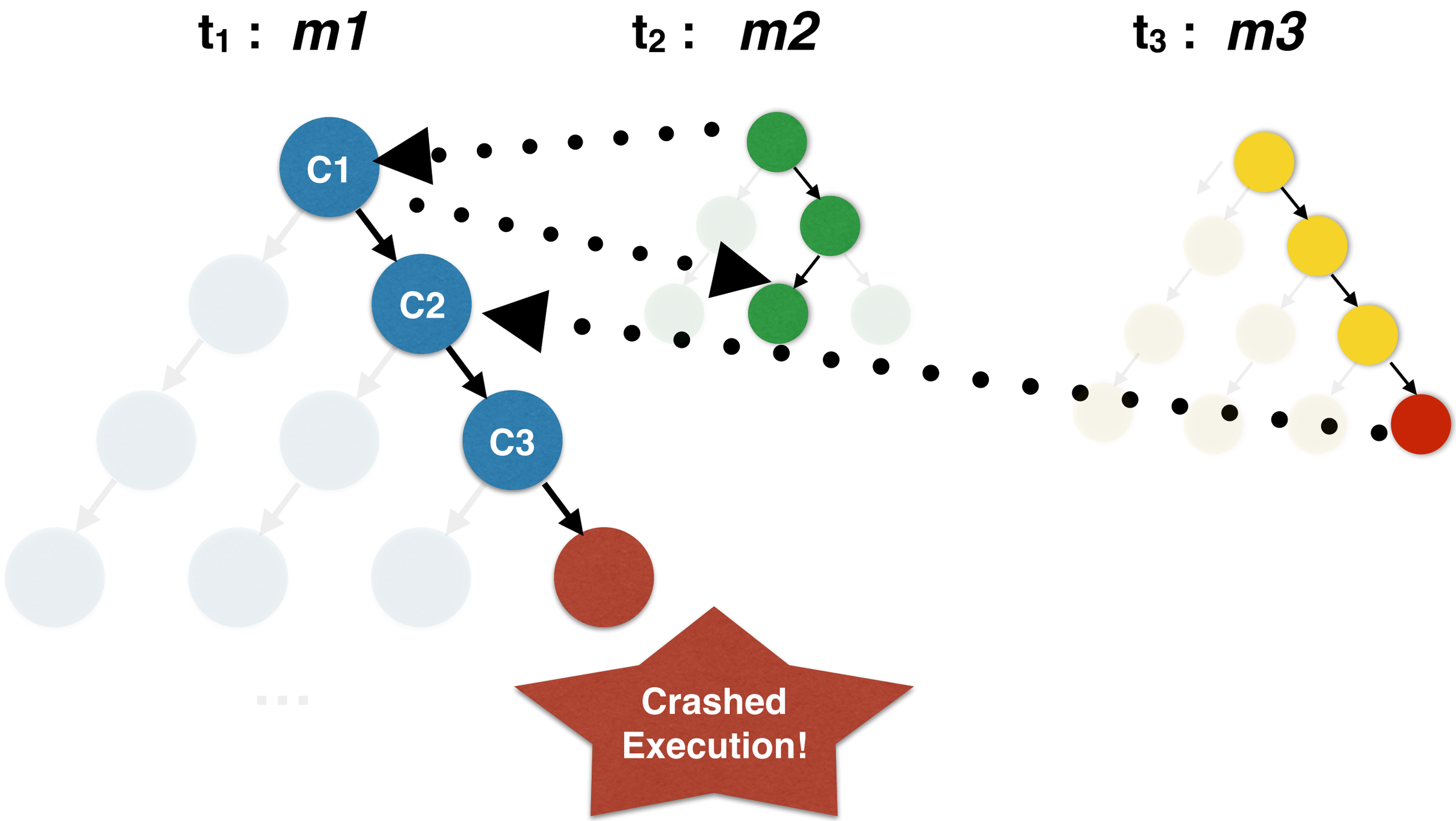
# Iteration - 4 : Concrete Execution



# Iteration - 4 : Concrete Execution



# 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) | 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 |

# 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

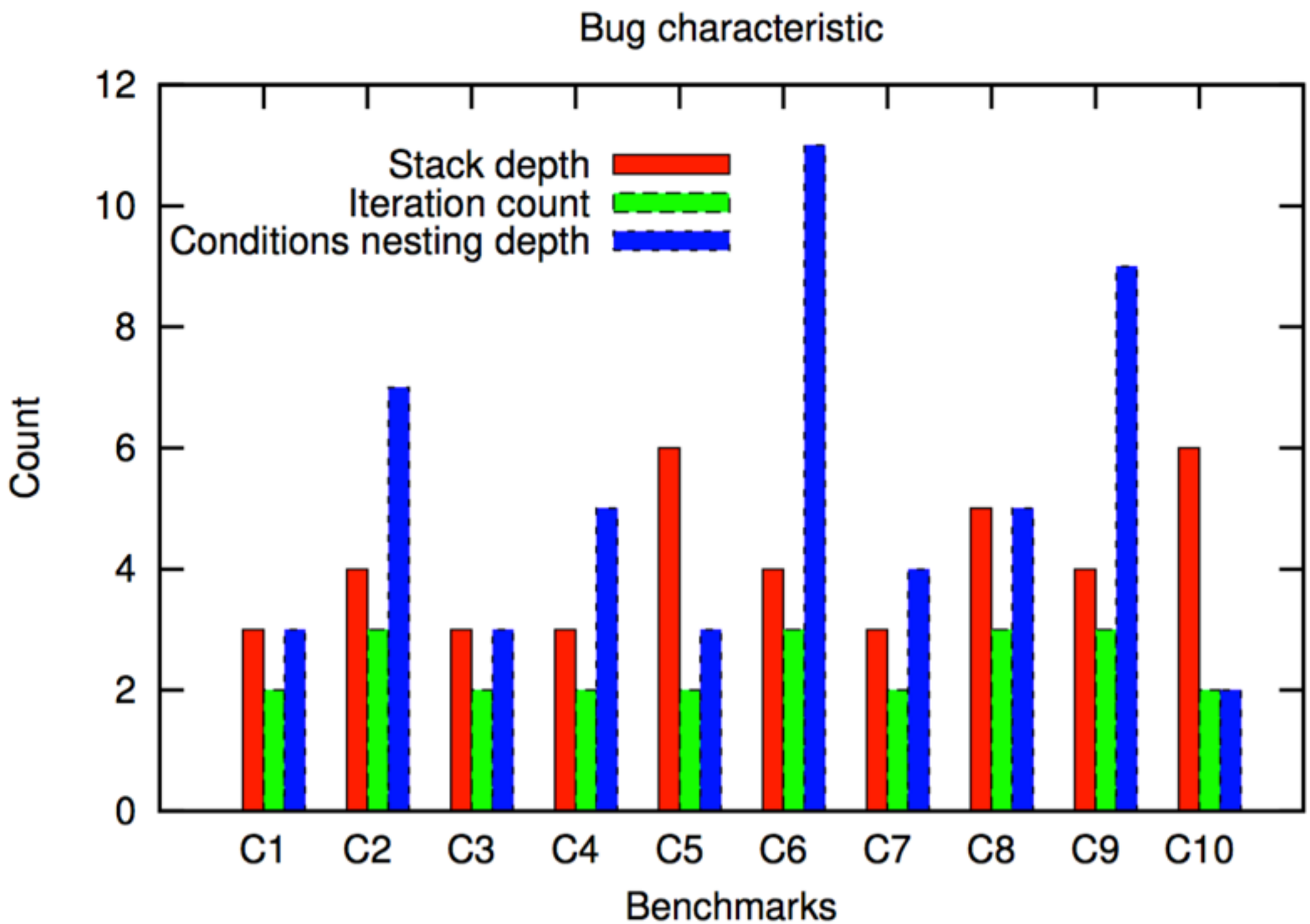
| Class | # of Targets | # of Constraints | Schedule length | Time (sec) | Violations | |
|---|---|---|---|---|---|---|
| | | | | | Seq. | Conc. |
| C1 | 2 | 254 | 37 | 21 | 0 | 1 |
| C2 | 8 | 8189 | 25 | 70 | 2 | 6 |
| C3 | 3 | 1898 | 43 | 10 | 0 | 2 |
| C4 | 8 | 4105 | 10 | 94 | 4 | 4 |
| C5 | 12 | 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** |

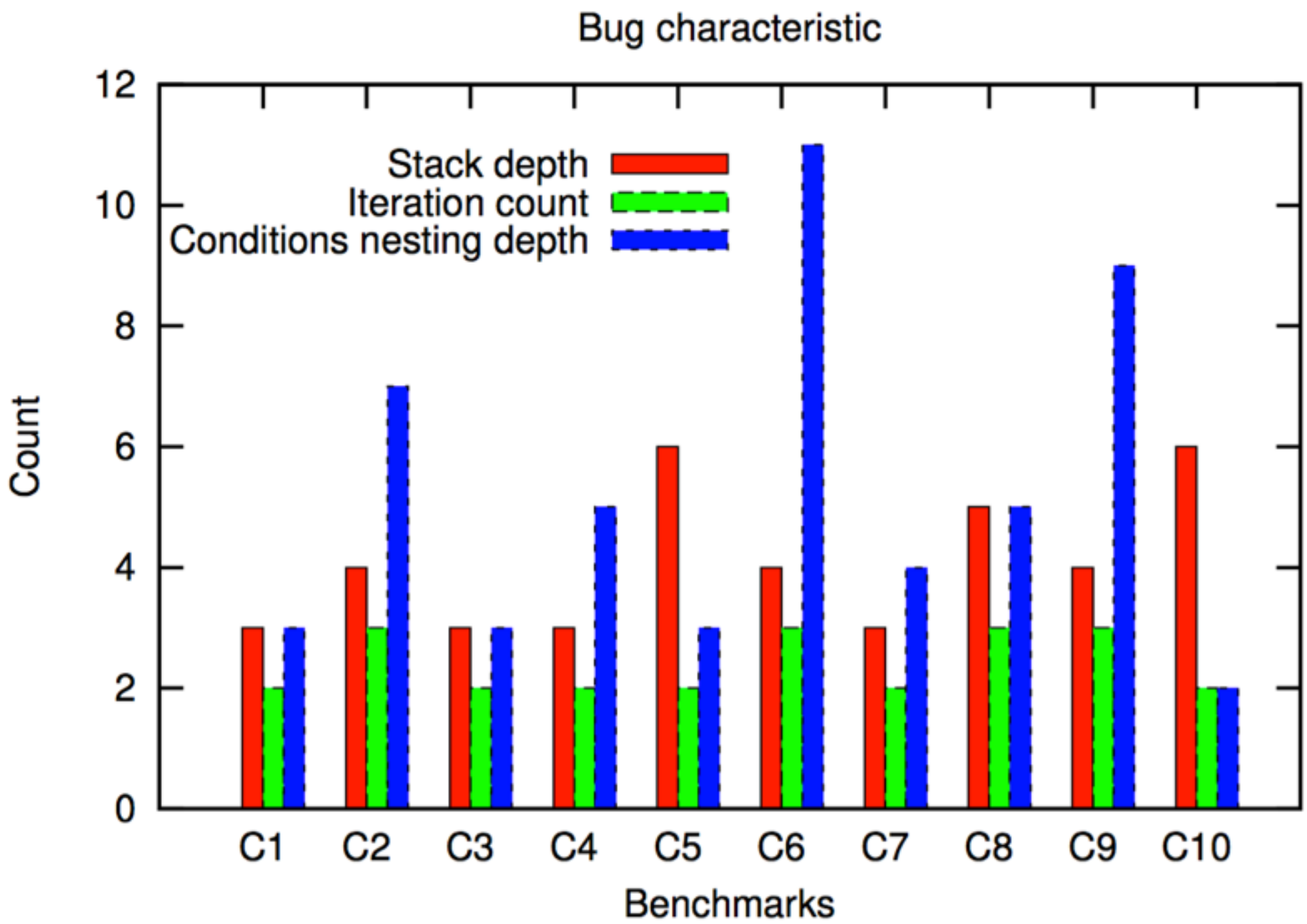
31 crashes due to concurrency

# Bug Characteristics



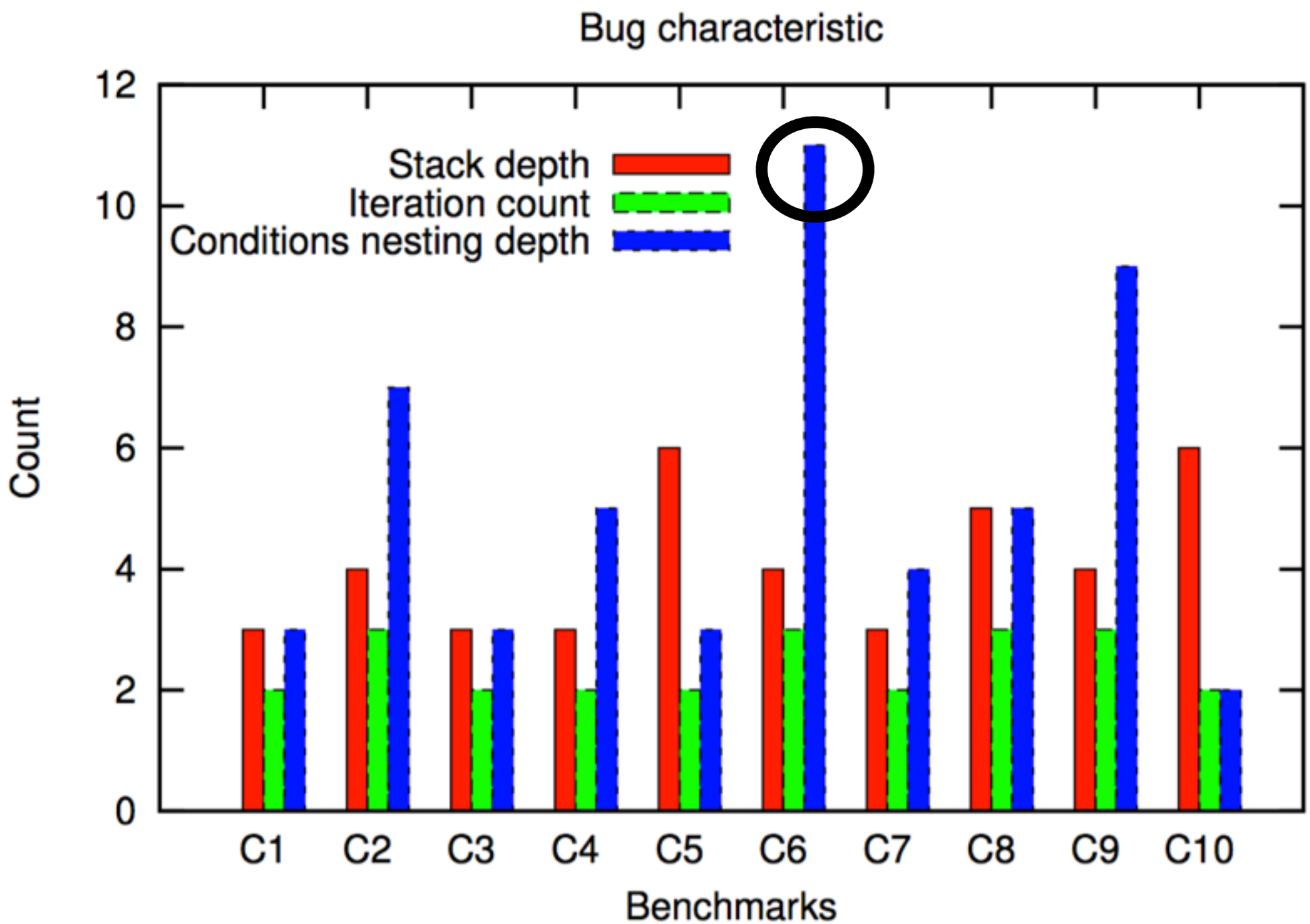
# Bug Characteristics

MINION detects complex concurrency issues



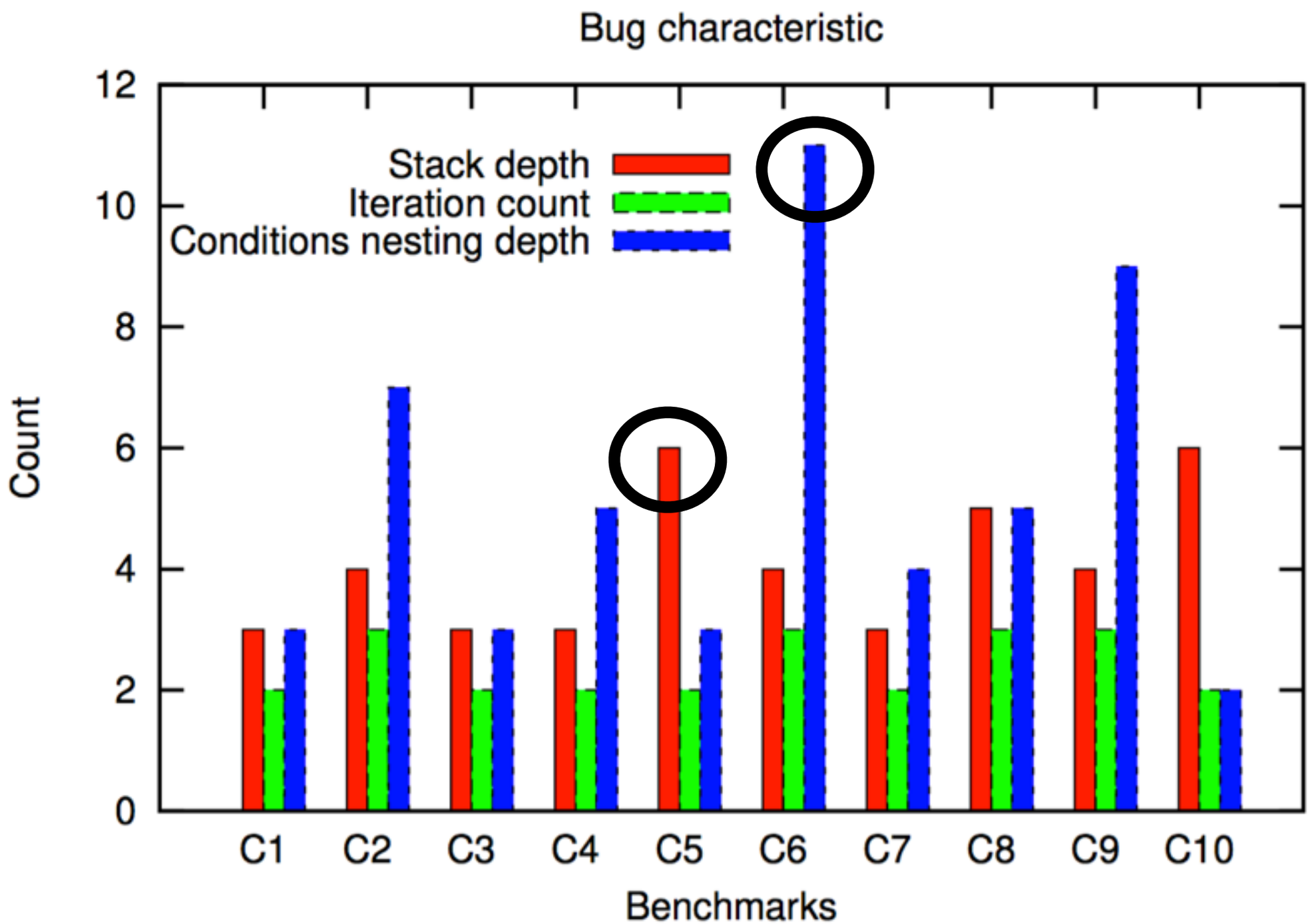
# Bug Characteristics

MINION detects complex concurrency issues



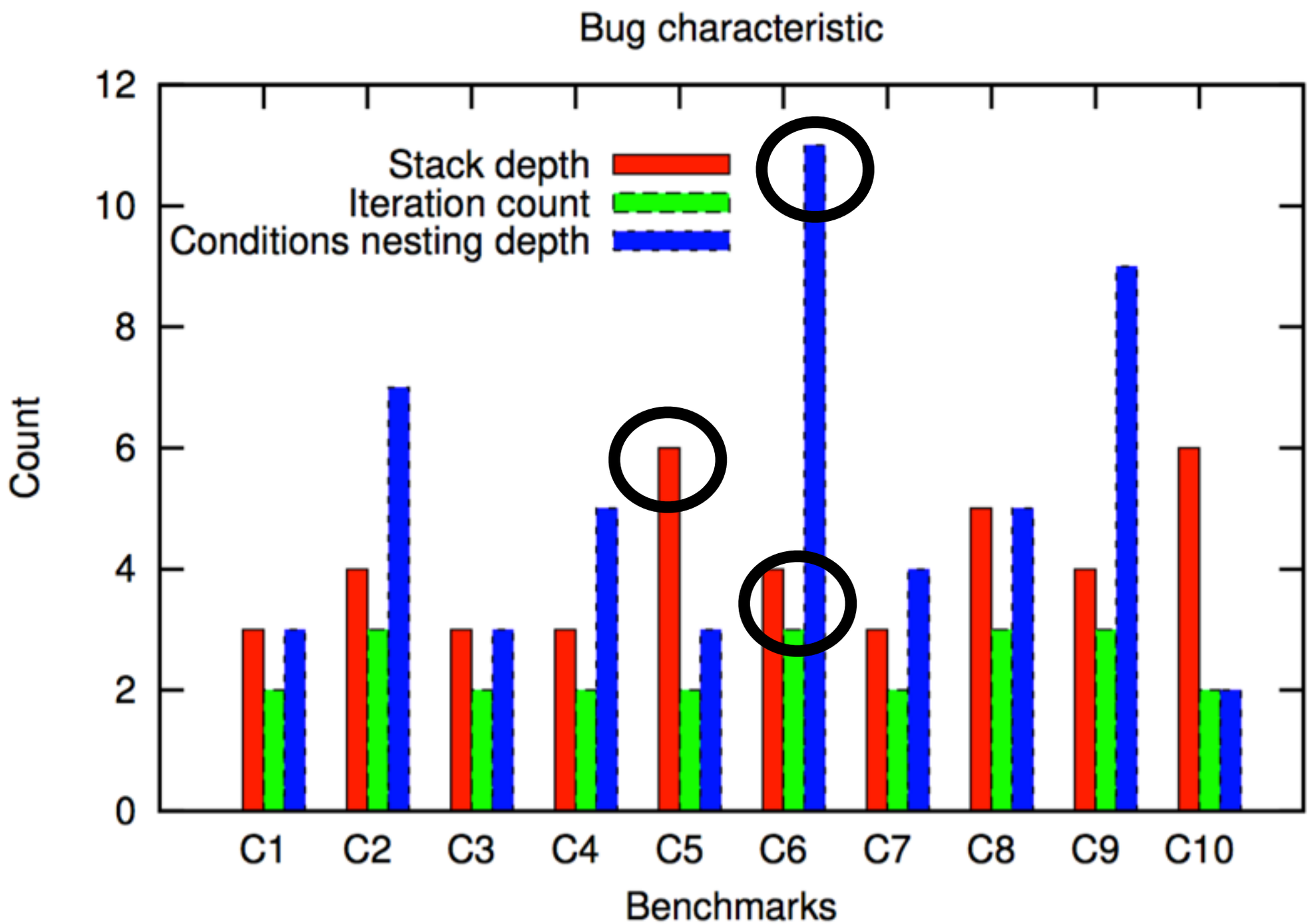
# Bug Characteristics

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