Vorbedingungen von Transformationen

Die Anwendbarkeit von Transformationen ist oft an Bedingungen geknüpft:

```
rule eliminate_additive_identity
    replace [expression]
        T1 [expression] + T2 [expression]
    where
        value (T2) = 0
    by
        T1
    end rule
```

Umsetzung von Transformationen

Ersetzung wird als Prozedur auf dem abstrakten Syntaxbaum implementiert:

```
procedure eliminate_additive_identity (n : in out node) is
begin
  if type (n) = plus then    -- Mustersuche
     if value (n.right) = 0 -- Bedingung
        then
                            -- Ersetzung
        replace_child (parent (n), from => n, to => n.left);
        n.left.parent := n.parent;
        delete_tree (n.right); delete (n);
      end if;
  end if;
end;
```

Eigenschaften von Transformationen

- Bestimmte Eigenschaften müssen bei der Transformation erhalten bleiben, andere Eigenschaften sollen sich ändern:
  - Performanz
  - Strukturiertheit
  - Änderbarkeit
  - ...
- In vielen (aber nicht allen) Fällen soll die Semantik erhalten bleiben.

Semantikerhaltende Transformationen

Formale Betrachtung:

- Objekt o hat Attribute und ist definiert in einem Kalkül
  - grundsätzliche Wahrheiten (Axiome) A
  - Menge von Inferenzregeln i
- die Eigenschaften von Objekt o sind dessen Attribute sowie alle Fakten, die sich aus den Axiomen und Attributen mittels Inferenzregeln herleiten lassen
- Transformation t ist **semantikerhaltend** genau dann, wenn für alle Objekte o gilt:
  - die Eigenschaften von o bleiben durch die Transformation erhalten: $P(o) \subseteq P(t(o))$
  - $P(t(o))$ ist konsistent (enthält keine Widersprüche)

  (alte Eigenschaften bleiben erhalten, neue dürfen hinzukommen, es ergeben sich keine Widersprüche)

Semantikerhaltende Transformationen

- Die Erhaltung der Semantik ist in der Praxis nur sehr schwer nachweisbar:
  - Beweis muss nicht nur die Semantik der Programmiersprache, sondern auch die aller aufgerufenen Betriebssystemfunktionen u.Ä. einbeziehen, und ist im Allgemeinen schwierig.
- Nach Transformationen stets Regressionstests durchführen!

Beispiel eines Transformationssystems

TXL[1] (Queens University, Canada): generisches Transformationssystem; ausgeprägt für C, C++, Java, Javascript, Modula, Object Pascal, XML:

- konkreter Syntaxbaum wird generiert
- funktionale Programmiersprache mit Transformationsregeln mit „native Patterns“:
- automatische Traversierung des konkreten Syntaxbaums und Anwendung der Transformationen, solange dies möglich ist

[1] `http://www.txl.ca`

TXL

```
rule eliminate_redundant_declarations
  replace [repeat statement]
      var X [id] : T [type_spec]
      Rest_of_Scope [repeat statement]
  where not
      Rest_of_Scope [references X]
  by
      Rest_of_Scope
end rule

function references X [id]
  match * [id] X
end function
```

Domain Maintenance System (DMS)

- Hersteller: Semantic Designs, Austin, Texas, USA
- generisches Transformationssystem
- parametrisierbar durch so genannte Domains (Grammatiken)
- konkrete Instanzen für C, C++, Java, Cobol, Jovial, ...
- abstrakte Regeln für Parse-Baum
- Clone-Doctor[2] erkennt Klone (mit dem Verfahren von Baxter et al.) und beseitigt sie

[2] Eingetragenes Warenzeichen von Semantic Designs `http://www.semdesigns.com`

DMS-Beispiel

```
default base domain Java;

rule merge-ifs(\condition1,
               \condition2,
               \then-statements)
=
   "if (\condition1)
       if (\condition2)
         { \then-statements }"
  rewrites to
   "if (\condition1 && \condition2)
         { \then-statements }"
;
```

Raincode

- Hersteller: Raincode[3], Brüssel, Belgien
- unterstützte Sprachen: Ada, APS, C, C++, COBOL, CSP, Delphi, Ideal, Informix 4GL, Java, Natural, PL/1
- Skript-Sprache, um Transformationen zu programmieren
- Transformation findet auf dem Text selbst statt

[3] `http://www.raincode.com`

Raincode-Beispiel I

```
PROCEDURE TERMINATE
VAR value;
BEGIN
    -- scan all nodes
    FOR exp IN ROOT.SubNodes DO
        -- if it is an expression that is not a simple literal
        -- and it has not been modified yet
        IF exp IS NonCommaExpression
        AND exp IS NOT Literal
        AND exp["Patched"]<>TRUE THEN
            -- do evaluation
            value := Eval(exp);
            -- if evaluation succeeds
            IF value <> VOID THEN
                -- replace the expression by its value
                PATCH.ReplaceNt(exp, value);
                -- report something on console
                exp.WriteError(LIST.ToString(PATCH.NtImage(exp)
                    || STR.Trim(X)," ")," = "&value);
```

Raincode-Beispiel II

```
                -- add annotation so we do not process subnodes
                FOR IN exp.SubNodes DO
                    X.Patched := TRUE;
                END;
            END;
        END;
    END;
    -- save the processed source
    PATCH.Save(ROOT.SourceName&".patched");
END;
```

Dynamische Analyse

2. Dynamische Analyse
   - Probleme statischer Analysen
   - Information durch dynamische Analyse
   - Testfälle
   - Instrumentierung
   - Probleme der Instrumentierung
   - Weiterverarbeitung dynamischer Information
   - Anwendungsbeispiele
   - Vergleich mit statischer Analyse

Probleme statischer Analysen

Bisher: statische Analyse

- Aliasing
- Polymorphismus und dynamisches Binden
- Reflection
- verteilte Systeme
- Laufzeitinstanzen versus statischer Einheiten (z.B. Klassen)

$\longrightarrow$ viele Entscheidungen erst zur Laufzeit

$\longrightarrow$ statische Analyse problematisch

Dynamische Analyse

Definition

Dynamische Analyse: Analyse der Eigenschaften eines laufenden Programms.

Dynamische Analyse

Definition

Dynamische Analyse: Analyse der Eigenschaften eines laufenden Programms.

Ablauf:

1. Ausführen des Programms
2. Beobachtung der Ausführung
3. Analyse der Ergebnisse

Dynamische Analyse

Definition

Dynamische Analyse: Analyse der Eigenschaften eines laufenden Programms.

Ablauf:

1. Ausführen des Programms
2. Beobachtung der Ausführung
3. Analyse der Ergebnisse

Notwendig:

1. Wahl der Testfälle

Dynamische Analyse

Definition

Dynamische Analyse: Analyse der Eigenschaften eines laufenden Programms.

Ablauf:

1. Ausführen des Programms
2. Beobachtung der Ausführung
3. Analyse der Ergebnisse

Notwendig:

1. Wahl der Testfälle
2. Instrumentierung oder Interpretation

Dynamische Analyse

Definition

Dynamische Analyse: Analyse der Eigenschaften eines laufenden Programms.

Ablauf:

1. Ausführen des Programms
2. Beobachtung der Ausführung
3. Analyse der Ergebnisse

Notwendig:

1. Wahl der Testfälle
2. Instrumentierung oder Interpretation
3. Analysemethoden/-werkzeuge

Was messen?

Beobachtung der Ausführung durch Messen

- Codeabdeckung oder Häufigkeit
  - Anweisungen, Zweige, Pfade, Aufrufe, ...
- berechnete Werte
- Laufzeit, Speicherverbrauch
- Reihenfolge von Operationen
- ...

Problem: Messung ändert Verhalten

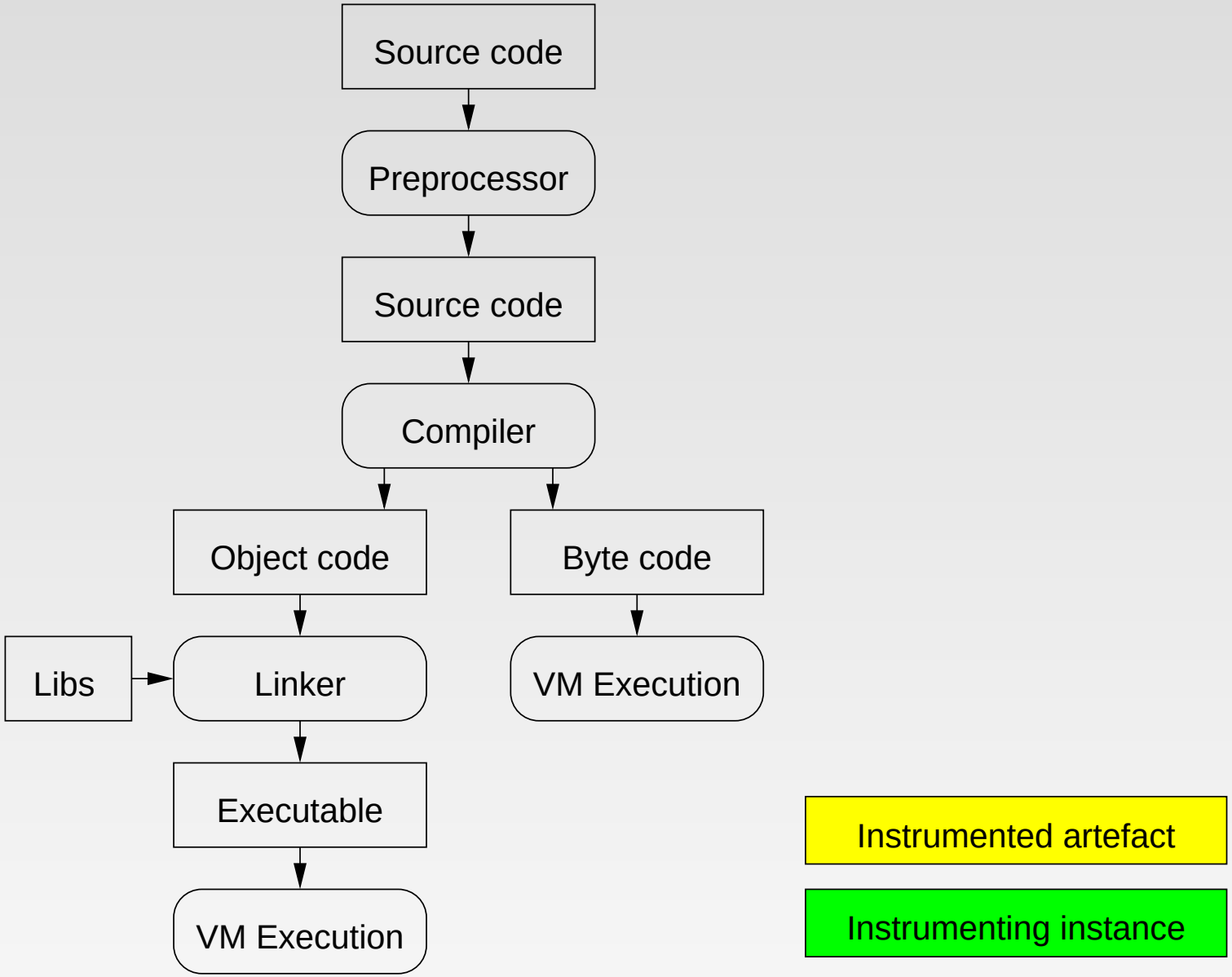
Wahl der Testfälle

Testsuite bestimmt

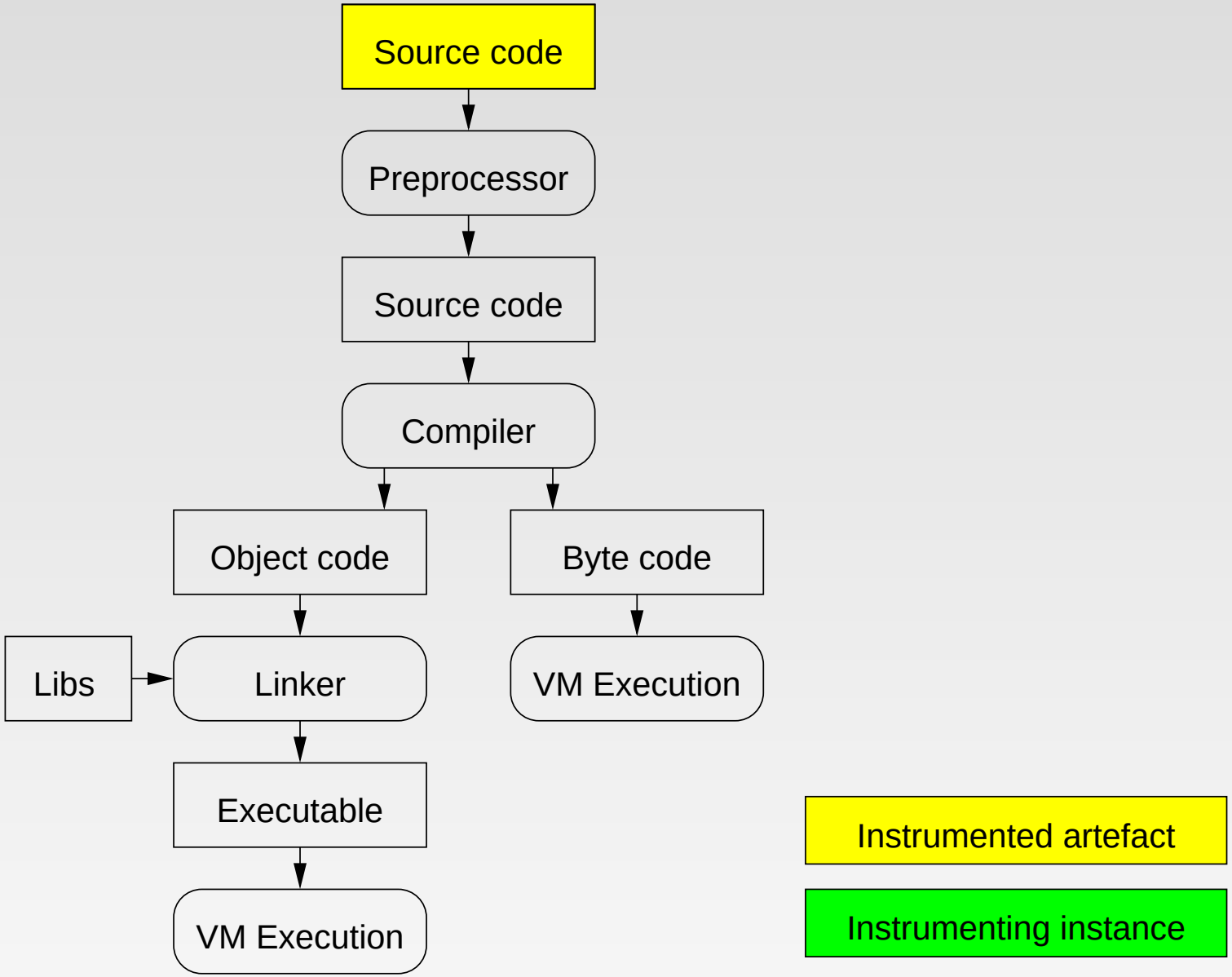
- Kosten (Zeit und Raum)
- Genauigkeit (was wird nie ausgeführt)

Problem: Vollständige Testabdeckung im Allgemeinen nicht zu erreichen.

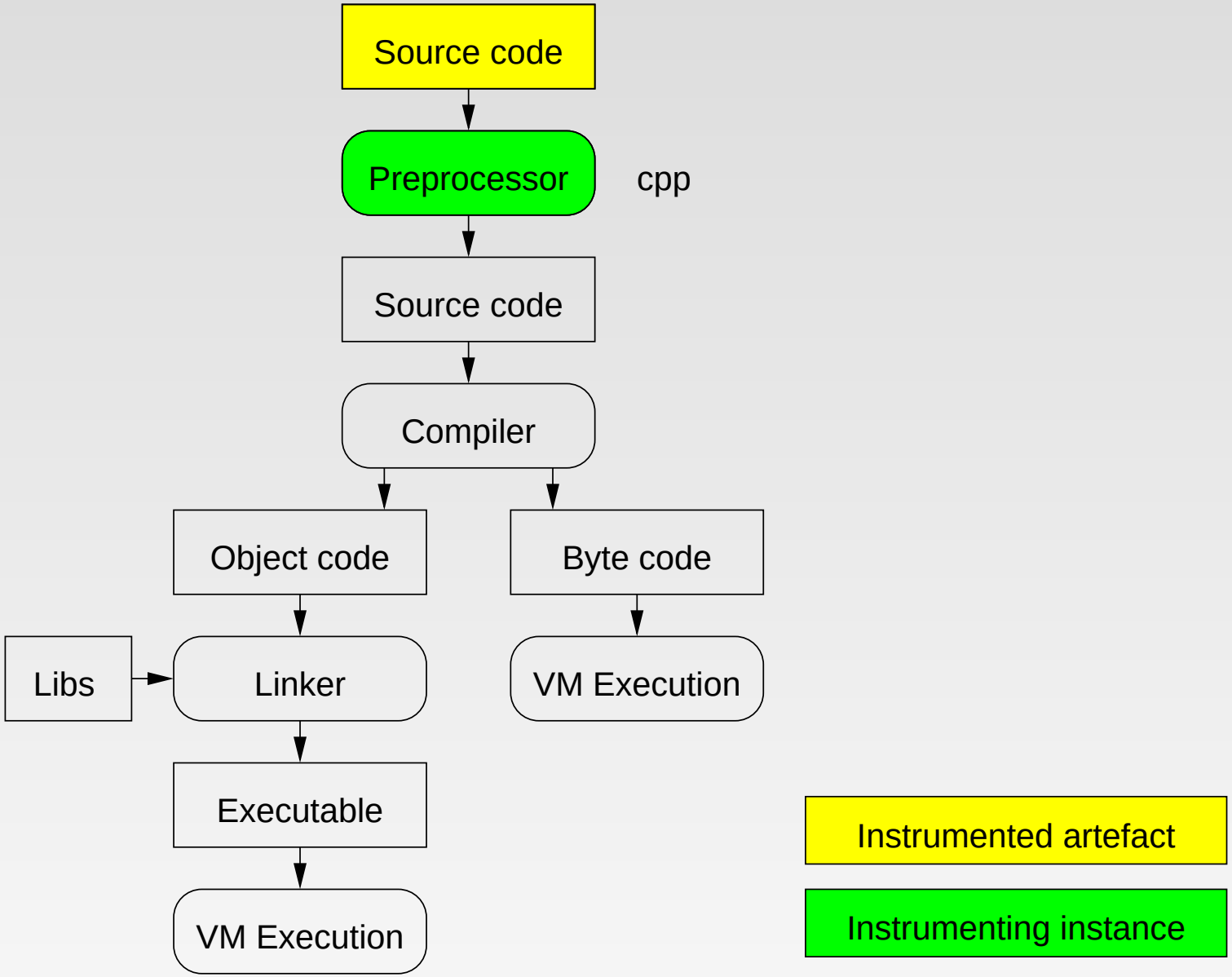
Instrumentierung



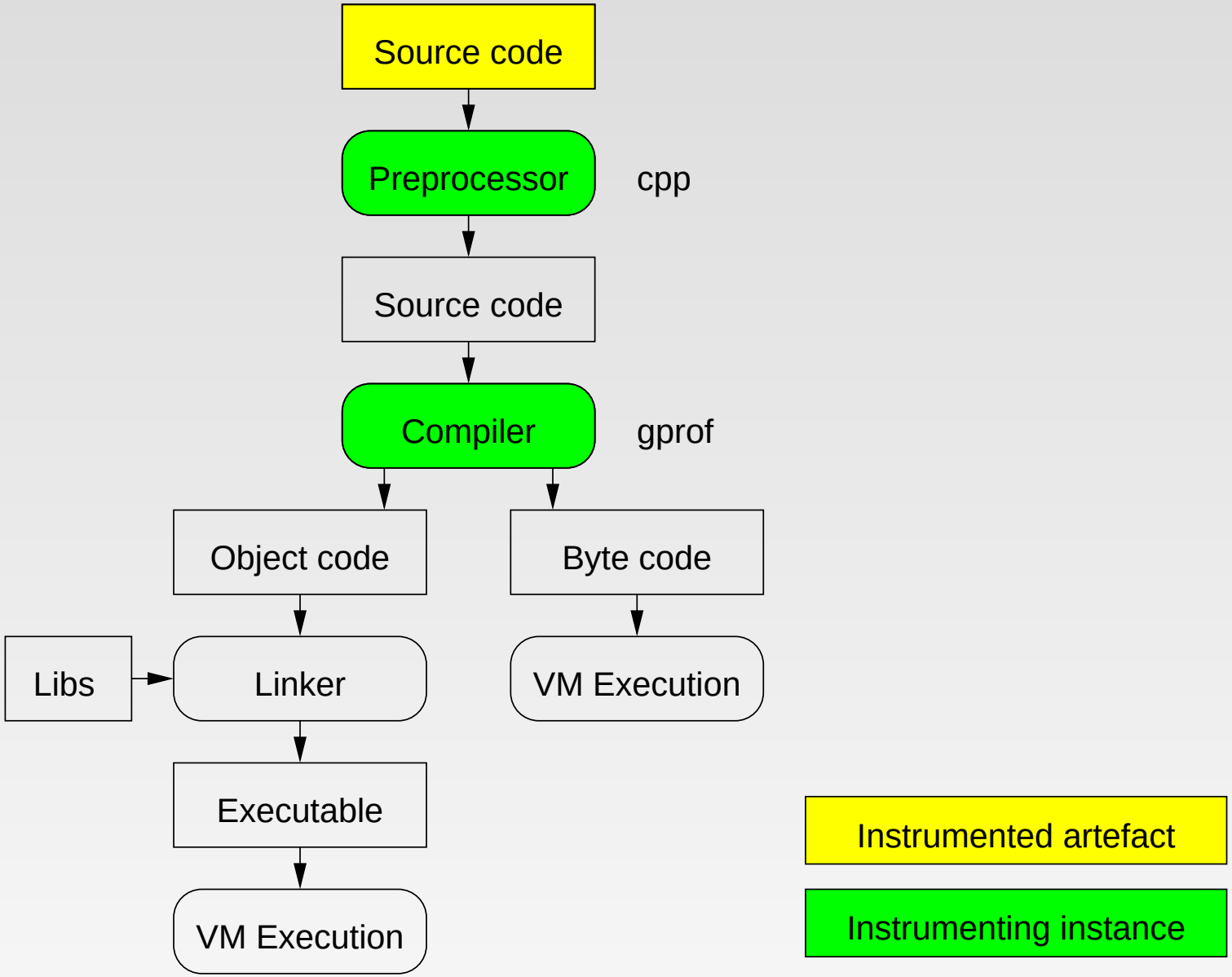
Instrumentierung



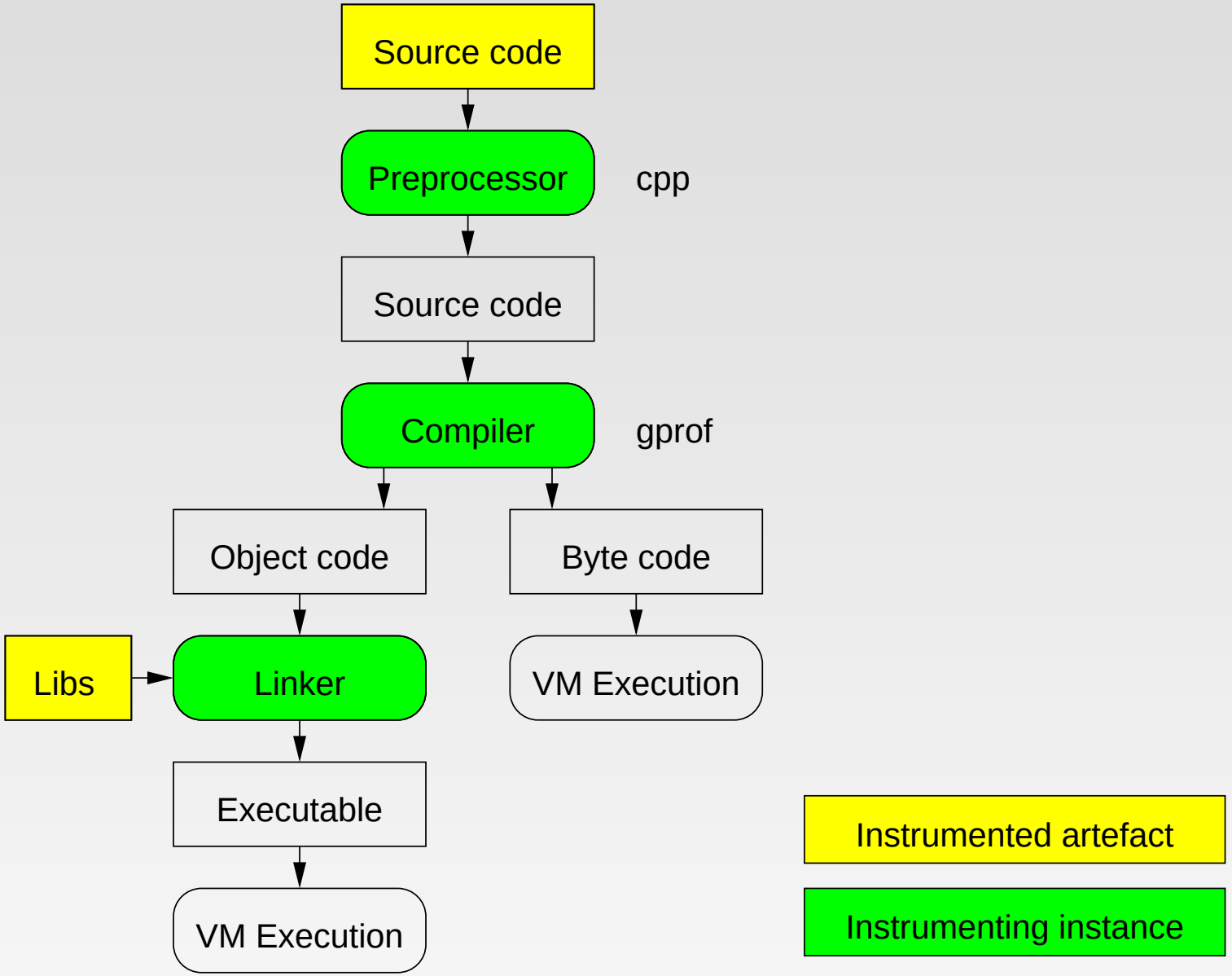
Instrumentierung



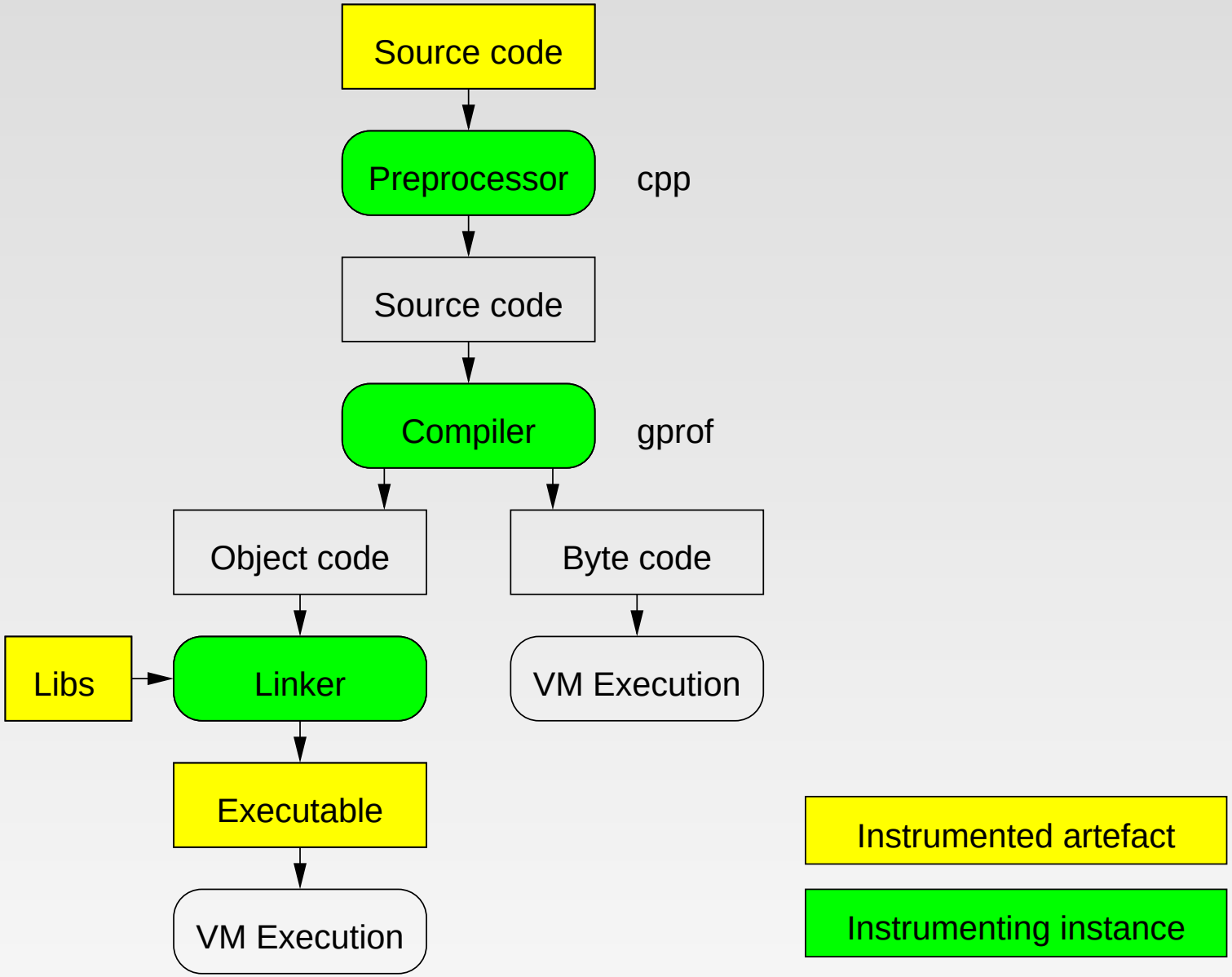
Instrumentierung



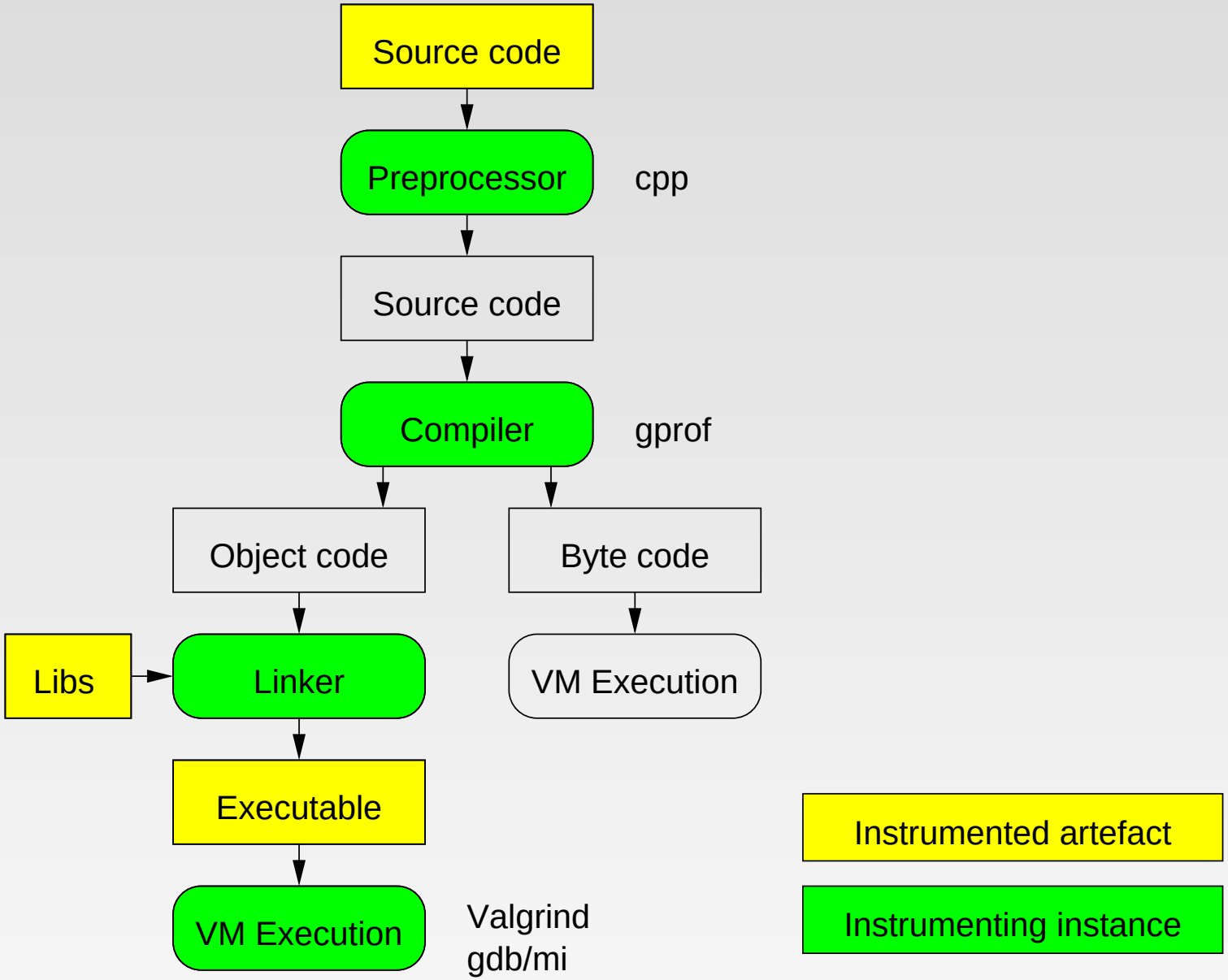
Instrumentierung



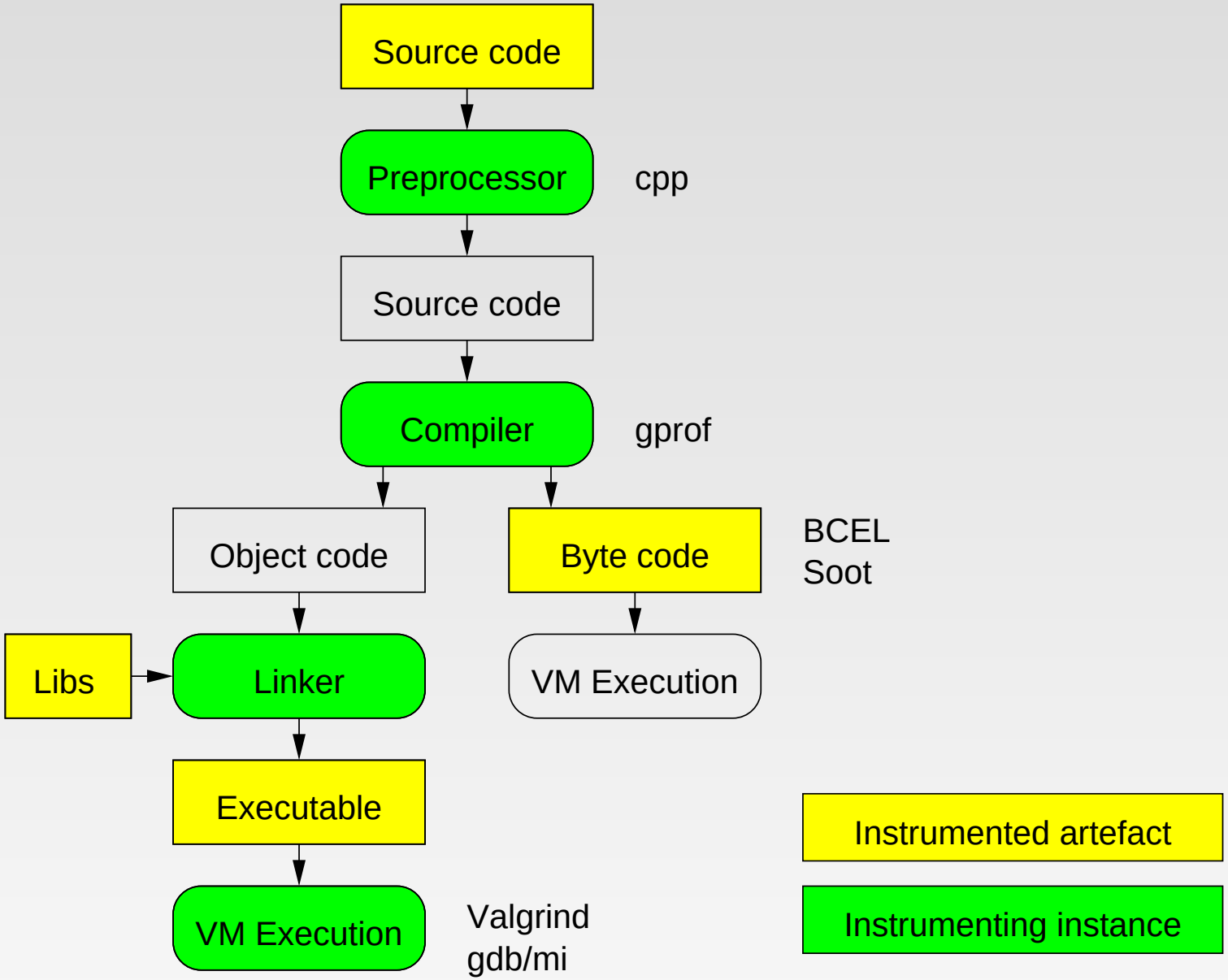
Instrumentierung



Instrumentierung



Instrumentierung



Instrumentierung