DSL Design

A conceptual framework for building good DSLs

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter

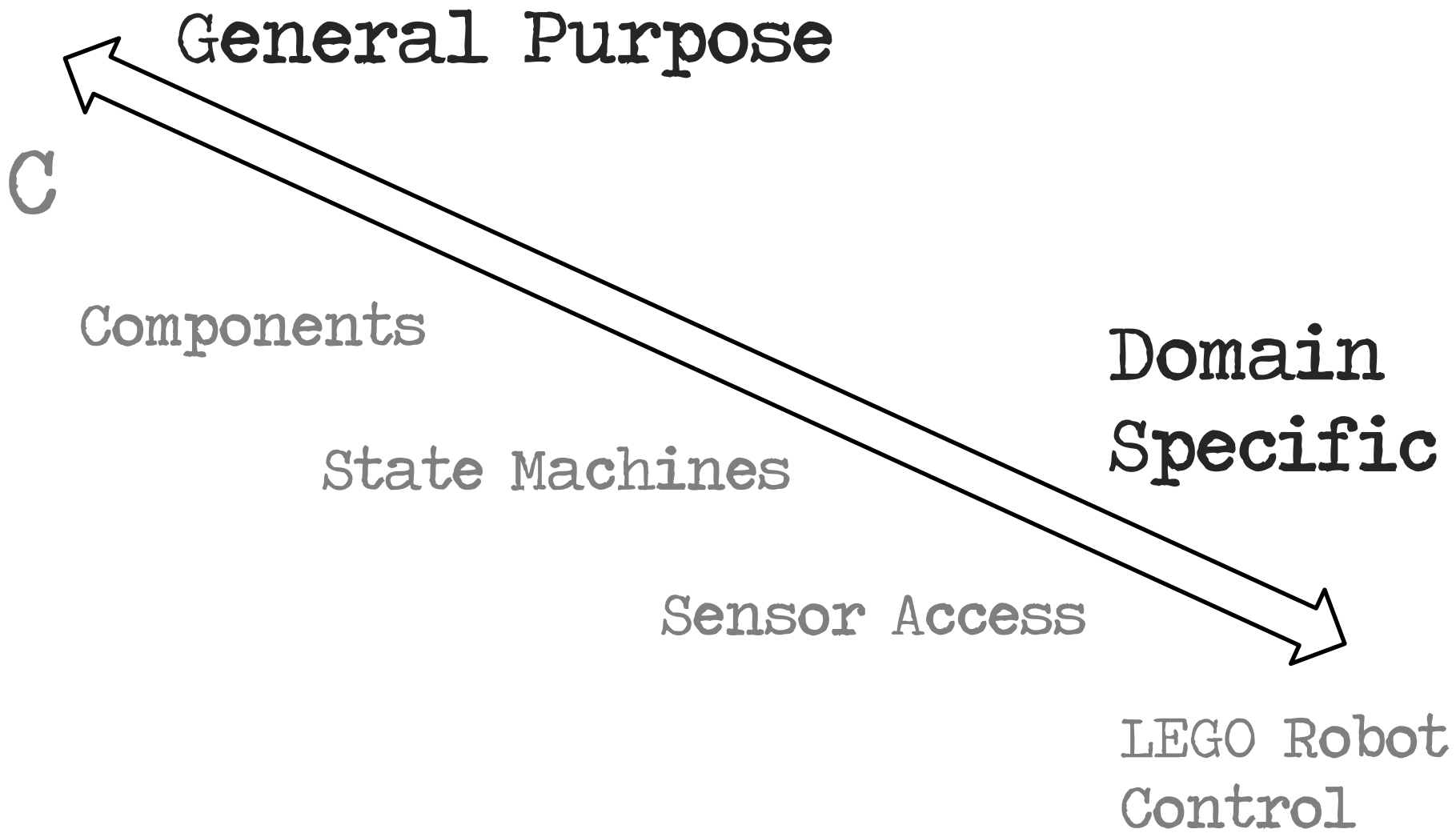
based on material
from a paper written
with Eelco Visser

E.Visser@tudelft.nl
http://eelcovisser.org/

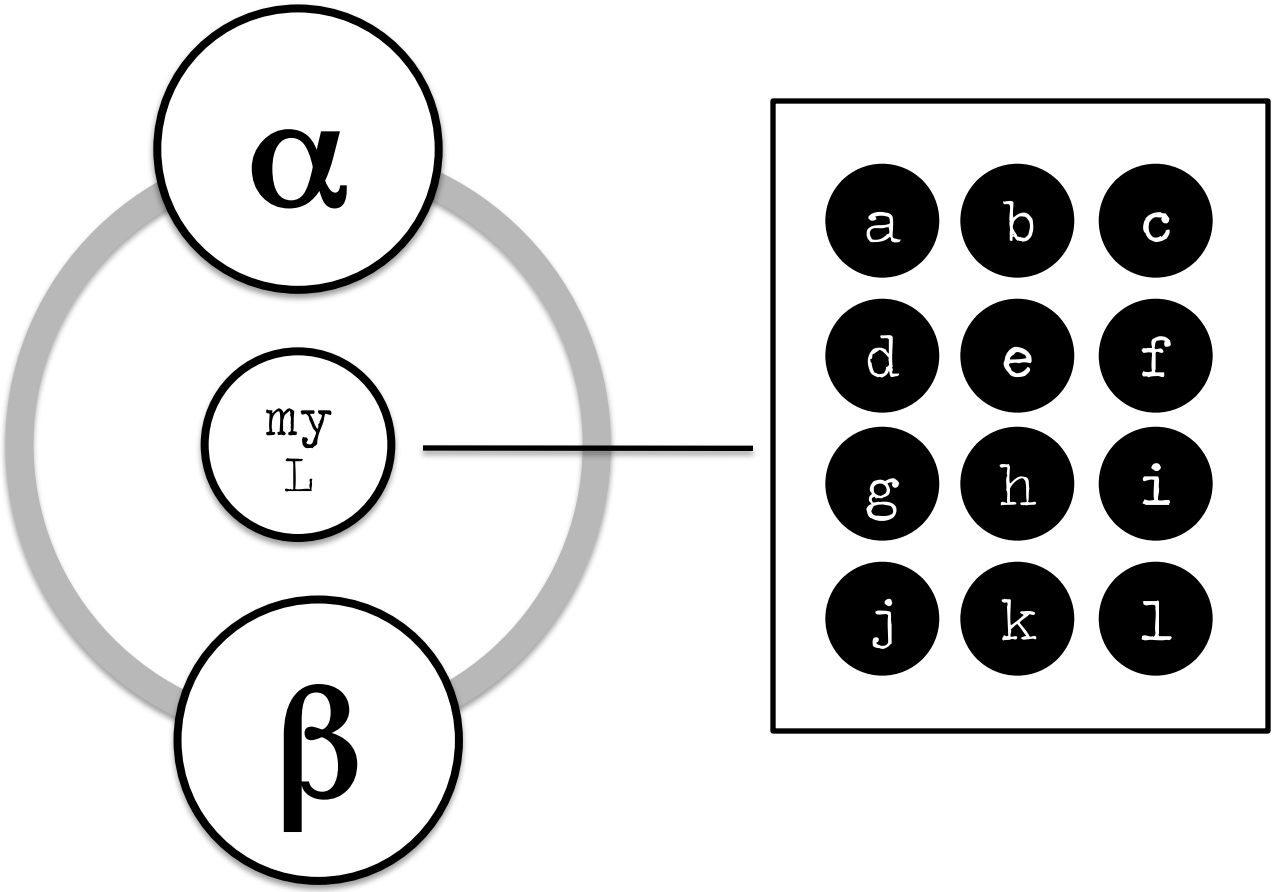
Introduction

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

	more in GPLs	more in DSL
Domain Size	large and complex	smaller and well-defined
Designed by	guru or committee	a few engineers and domain experts
Language Size	large	small
Turing-completeness	almost always	often not
User Community	large, anonymous and widespread	small, accessible and local
In-language abstraction	sophisticated	limited
Lifespan	years to decades	months to years (driven by context)
Evolution	slow, often standardized	fast-paced
Incompatible Changes	almost impossible	feasible


General Purpose
C
Components
Domain Specific
State Machines
Sensor Access
LEGO Robot Control

Modular Language



with many optional,
composable modules

Case Studies

```
namespace com.mycompany {
  namespace datacenter {
    component DelayCalculator {
      provides aircraft: IAircraftStatus
      provides console: IManagementConsole
      requires screens[0..n]: IInfoScreen
    }
    component Manager {
      requires backend[1]: IManagementConsole
    }
    public interface IInfoScreen {
      message expectedAircraftArrivalUpdate
              (id: ID, time: Time)
      message flightCancelled(flightID: ID)
    }
    public interface IAircraftStatus …
    public interface IManagementConsole …
  }
}
```

Example

Compo
nents

```
namespace com.mycompany.test {
  system testSystem {
    instance dc: DelayCalculator
    instance screen1: InfoScreen
    instance screen2: InfoScreen
    connect dc.screens to
      (screen1.default, screen2.default)
  }
}
```

Example

Compo
nents

```
appliance KIR {

    compressor compartment cc {
        static compressor c1
        fan ccfan
    }

    ambient tempsensor at

    cooling compartment RC {
        light rclight
        superCoolingMode
        door rcdoor
        fan rcfan
        evaporator tempsensor rceva
    }

}
```

Example

Refrigerators

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

Example

Refrigerators

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

Refrigerators

```
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;
  constant int BLACK = 700;
  constant int SLOW = 20;
  constant int FAST = 40;

  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> running
    }
    state running { }
  }

  initialize {
    ecrobot_set_light_sensor_active
                (SENSOR_PORT_T::NXT_PORT_S1);
    event linefollower:initialized
  }

  terminate {
    ecrobot_set_light_sensor_inactive
                (SENSOR_PORT_T::NXT_PORT_S1);
```

```
task run cyclic prio = 1 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor
                        (SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_C, left, 1);
  nxt_motor_set_speed(MOTOR_PORT_T::NXT_PORT_B, right, 1);
}
```

Example

Exten
ded C

Capgemini Pension Workbench

File Edit Projection Navigation Search Format Tools Dev Generate Pension Team NN

NNLCPA-14w2-21112008 * ×

Table of Contents ×

All

Library
- Documentation
- Foundation
  - Value sets
    - Value set Groottebepalingsmethode
      - Value set member Salaris-diensttijd
      - Value set member Verzekerde bedrage
      - Value set member Afgeleide toezeggin
    - Value set Salaris-diensttijd
      - Value set member Middelloon
      - Value set member Eindloon
    - Value set Verzekerde bedragen
      - Value set member Vast bedrag
      - Value set member Percentage van gron
      - Value set member Percentage van gron
      - Value set member Opgegeven bedrag
      - Value set member ANW-hiaat
      - Value set member AOP bedrag
    - Value set Indicatie Opbouw / Risico
      - Value set member Opbouw
      - Value set member Risico
    - Value set Deelnemerstatus
      - Value set member Aspirant
      - Value set member Actief
      - Value set member Premievrij
      - Value set member Slapend
      - Value set member Uitkerend
      - Value set member Overleden
      - Value set member Vervallen
  - Tag definitions
    - Tag Basisberekening
    - Tag Ouderdomspensioen
    - Tag Partnerpensioen
    - Tag Wezenpensioen
    - Tag ANW extra
    - Tag WIA excedent AOV

¤ 3.3 Commutatiegetallen op 1 leven¶

$$D_x = v^x * \frac{l_x}{100}$$ ¤6 Dec(3)¶

Implemented in ¤ V9401¶

¶

$$N_x = \sum_{t=0}^{\omega - x} D_{x+t}$$ ¤7 Dec(3)

¶

¤ 3.6 Contante waarde 1 leven/ 2 levens¶

$$_nE_x = \frac{D_{x+n}}{D_x}$$ ¤19 Dec(4)

¶

$$a_x = \ddot{a}_x - 1$$ ¤21 Dec(3)

¶

$$\bar{a}_x = \ddot{a}_x - 0{,}5$$ ¤22 Dec(3)

¶

$$\ddot{a}_{x\overline{n}|} = \frac{N_x - N_{x+n}}{D_x}$$ ¤23 Dec(3)

$$\bar{a}_{x\overline{n}|} = \ddot{a}_{x\overline{n}|} - 0{,}5 + 0{,}5 * {}_nE_x$$ ¤25 Dec(3)

¶

¤ 4 BN(_ris) koopsommen¶

Section ▸ title ▸ Paragraph : Text Dev

Doc | Splitter | Pension | PensionDecorated | AM

Example

Pension Plans

Capgemini Pension Workbench

File Edit Projection Navigation Search Format Tools Dev Generate Pension Team NN

NNLCPA-14w2-21112008 * ×

Table of Contents ×

- Library
  - Documentation
  - Foundation
    - Value sets...
    - Tag definitions
      - Tag Basisberekening
      - Tag Ouderdomspensioen
      - Tag Partnerpensioen
      - Tag Wezenpensioen
      - Tag ANW extra
      - Tag WIA excedent AOV
      - Tag Eindkapitaal
  - Shared
    - Elements...
    - Rules
      - Rule Bereken Mutatieperiode
      - Rule Bereken Salaris ontwikkeling
      - Rule Bereken Pensioengrondslag
      - Rule Bereken Pensioengrondslag
      - Rule Bereken pensioengrondslag
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Delta deelaanspraak uit doorl
      - Rule Bereken Deelaanspraak opgebouwd
      - Rule Bereken Toekomstige dienstjaren
      - Rule Bereken Deelaanspraak uitzicht
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verkoopkosten
      - Rule Bereken Netto Eindwaarde
      - Rule Bereken einddatum opbouw
      - Rule Bereken premie
      - Rule Bereken IS-Opslag
      - Rule Bereken administratiekosten

All

Elements...

Rules

Rule Bereken Mutatieperiode

Result:
Mutatieperiode

Name:
Bereken Mutatieperiode

Documentation:
Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen.¶
De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verslagjaar.¶
Dit wordt niet afgevangen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.¶

Tags:
Basisberekening

Algorithm:
if maximum(Mutaties per datum) == 1 **then daysof**(**duration**(**valid**(Mutaties per datum))) **else** 0

Test cases:

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Gelijke datums	03/01/2008		Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig			3	0
Periode < 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (binnen 1 maand)			15	15
Periode > 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (meerdere maanden)			60	60

Bereken Mutatieperiode ▸ Test cases ▸ Unit test: Gelijke datums : ^Place Dev

Doc | Splitter | Pension | PensionDecorated | AM

Example

Pension Plans

```
entity Post {
  key        :: String (id)
  blog       → Blog
  urlTitle   :: String
  title      :: String (searchable)
  content    :: WikiText (searchab
  public     :: Bool (default=fals
  authors    → Set<User>
  function isAuthor(): Bool {
    return principal() in authors
  }
  function mayEdit(): Bool {
    return isAuthor();
  }
  function mayView(): Bool {
    return public || mayEdit();
  }
```

```
access control rules
  rule page post(p: Post, title: String) {
    p.mayView()
  }
  rule template newPost(b: Blog) {
    b.isAuthor()
  }
section posts
  define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloglayout(p.blog){
      placeholder view { postView(p) }
      postComments(p)
    }
  }
  define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
  }
```

Example

Web DSL

Terms
&
Concepts

Model

A schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics

www.answers.com/topic/model

Model

A representation of a set of components of a process, system, or subject area, generally developed for understanding, analysis, improvement, and/or replacement of the process

www.ichnet.org/glossary.htm

Model

which ones?

an abstraction or simplification of reality

what should we leave out?

ecosurvey.gmu.edu/glossary.htm

Model Purpose

… code generation
… analysis and checking
… platform independence
… stakeholder integration
…

drives design of language!

Model Purpose

… code generation

… analysis and checking

… platform independence

… stakeholder integration

Example

Compo
nents

Model Purpose

… code generation

… analysis and checking

… platform independence

… stakeholder integration

Example

Refrige
rators

Model Purpose

… code generation
… analysis and checking
… platform independence
… stakeholder integration

Example

Exten
ded C

Model Purpose

… code generation

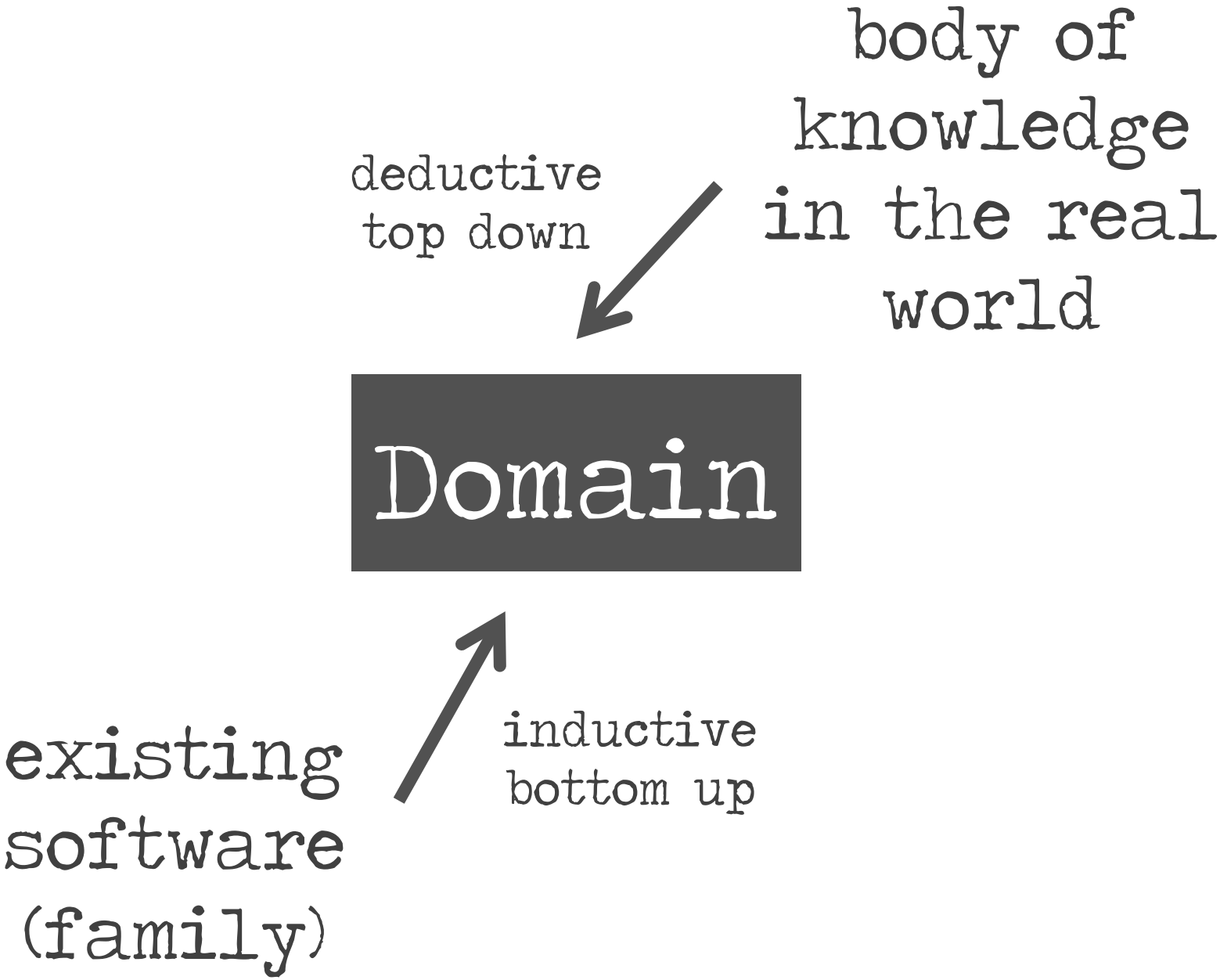
… analysis and checking

… platform independence

… stakeholder integration

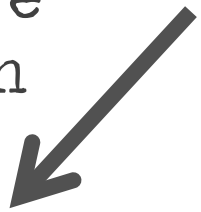
Example

Pension Plans


body of knowledge in the real world
deductive top down
Domain
existing software (family)
inductive bottom up

body of knowledge in the real world

deductive top down



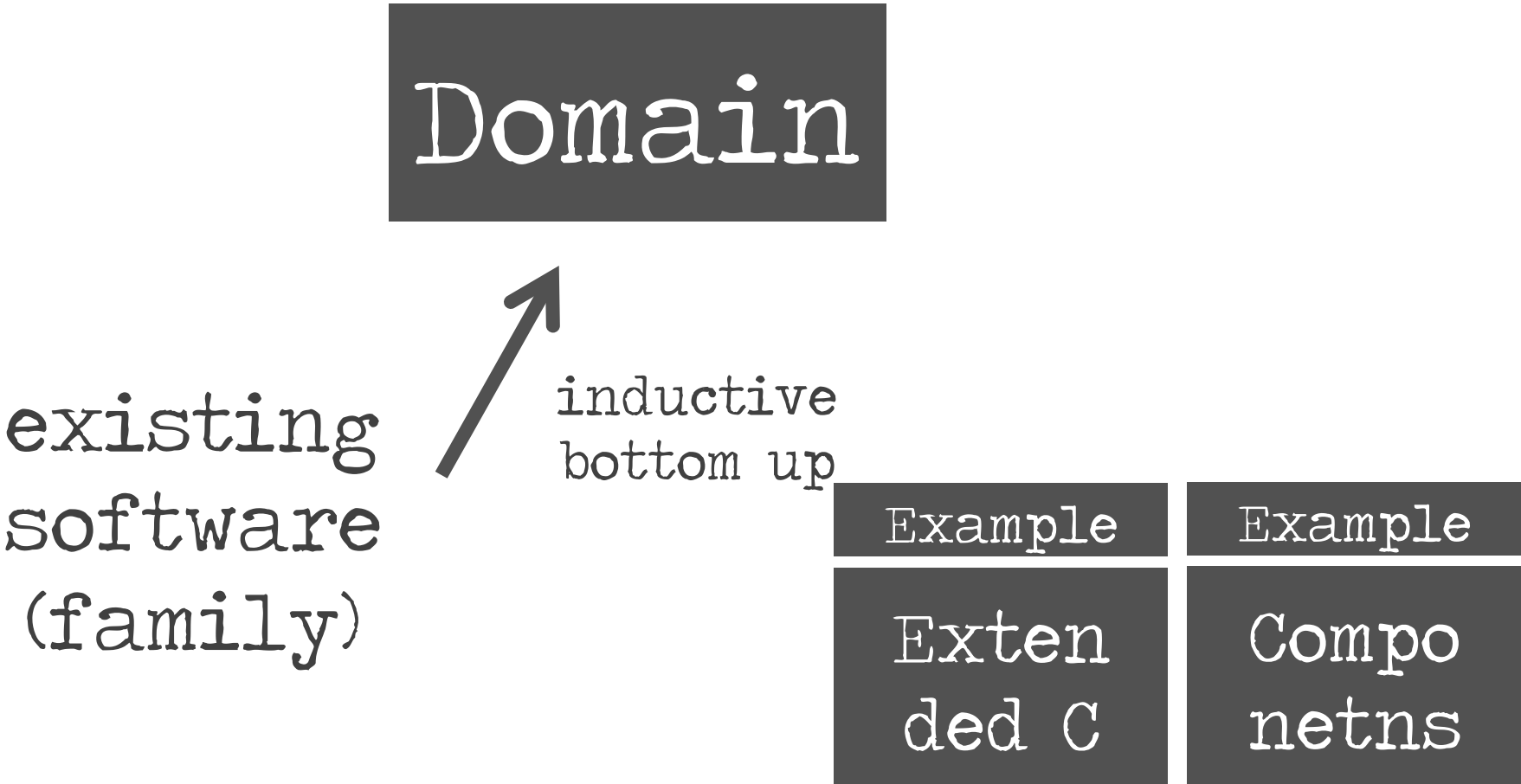
Domain

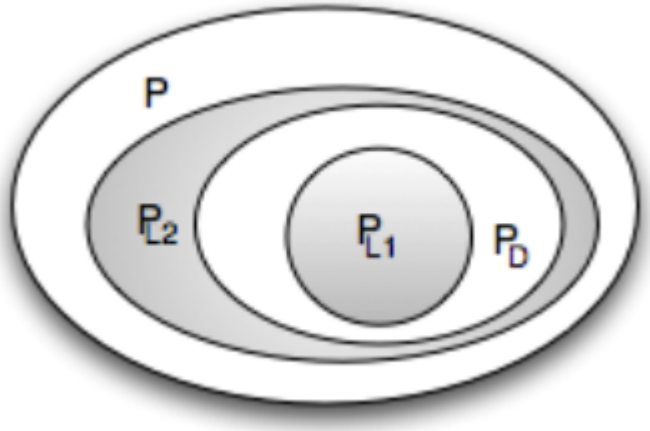
Example

Penion Plans

Example

Refrige rators

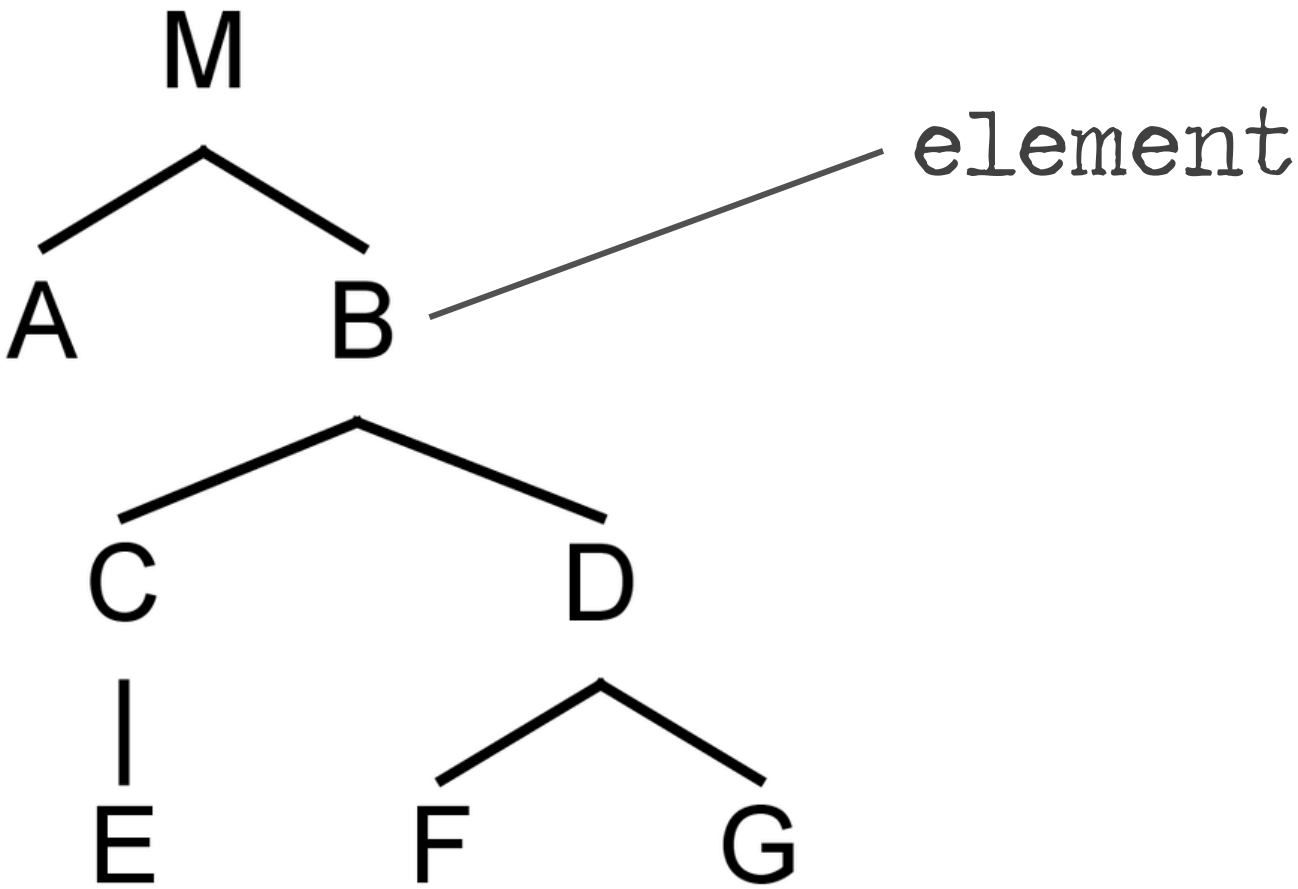

Domain
existing software (family)
inductive bottom up
Example
Example
Exten ded C
Compo netns



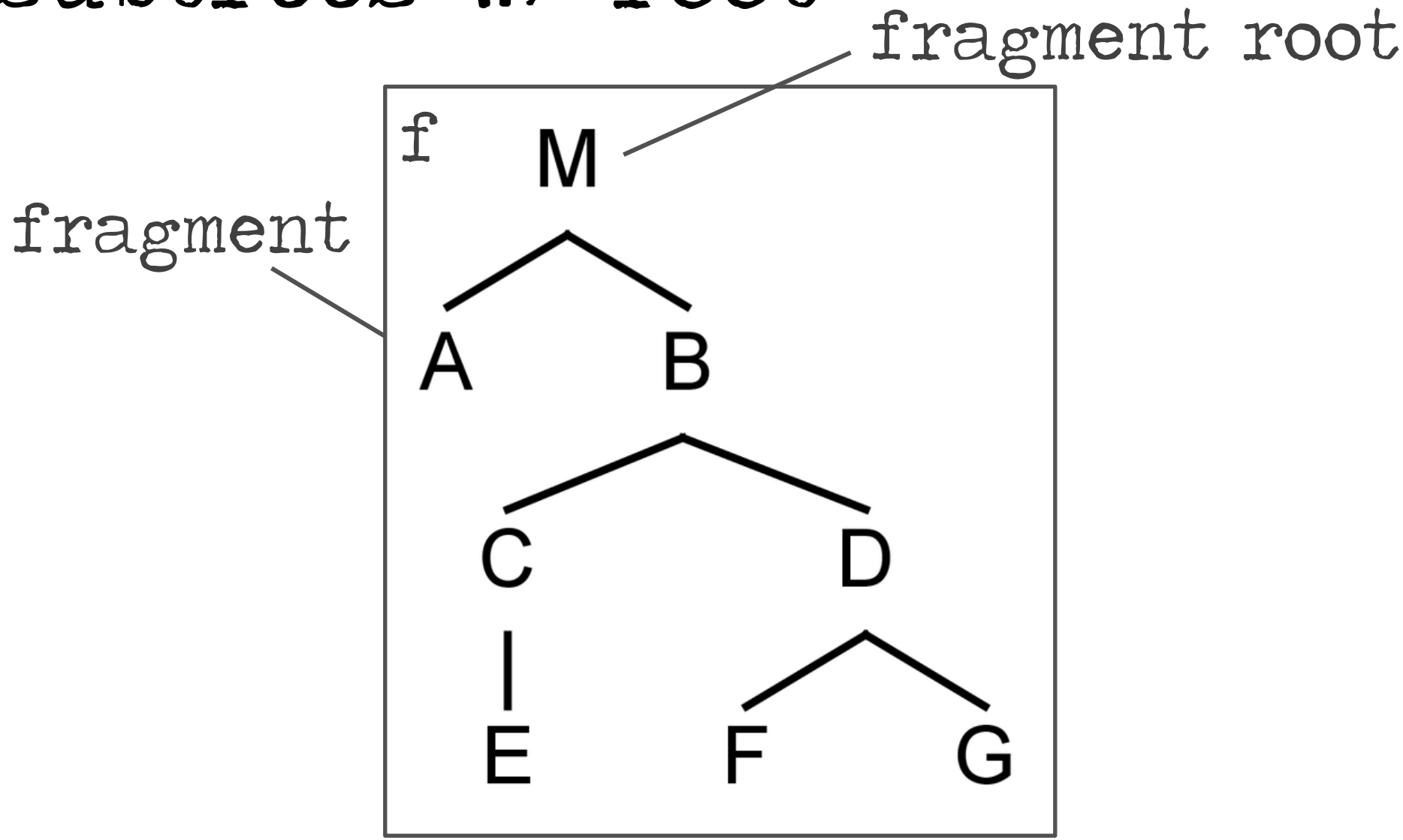
A DSL L_D for D is a language that is **specialized** to encoding P_D programs.

more efficient
smaller

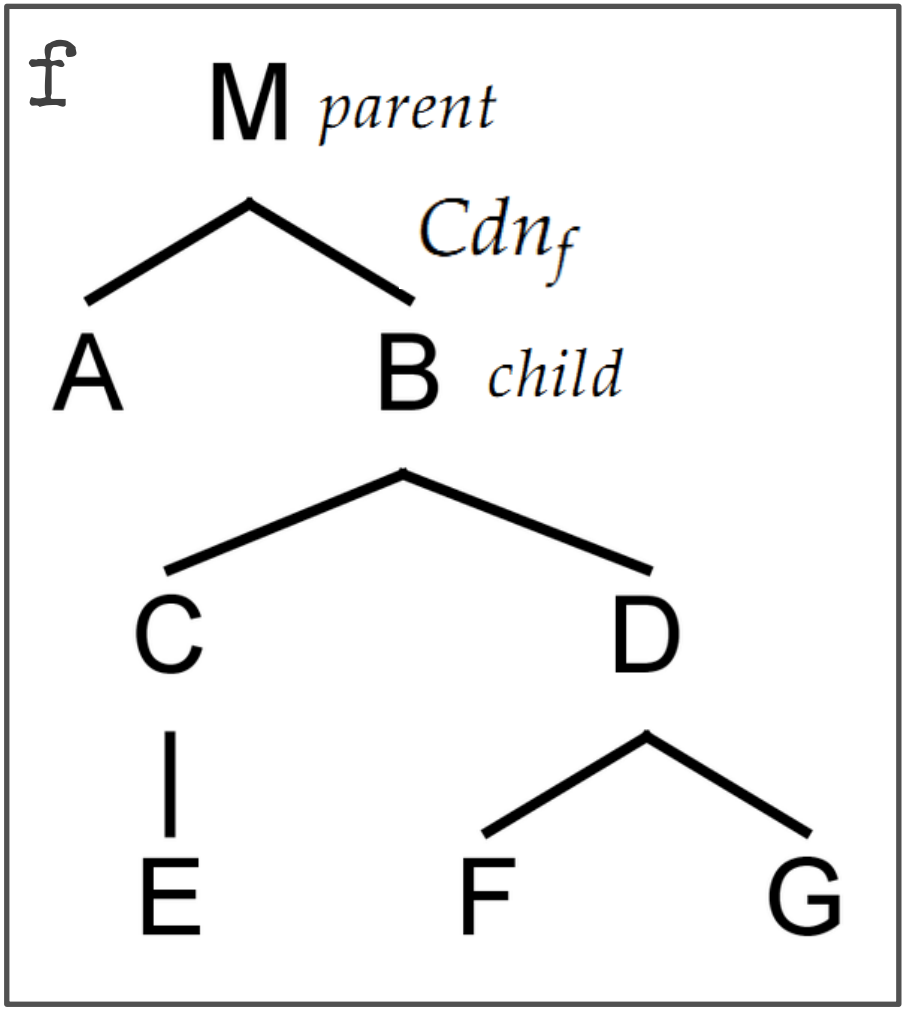
Programs are trees



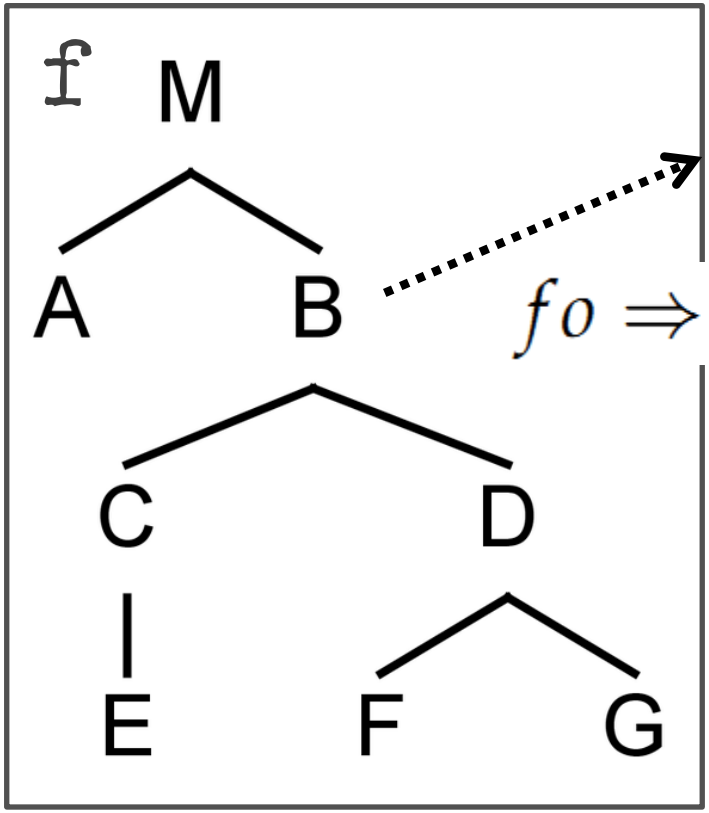
Fragments are subtrees w/ root



Parent-Child Relation

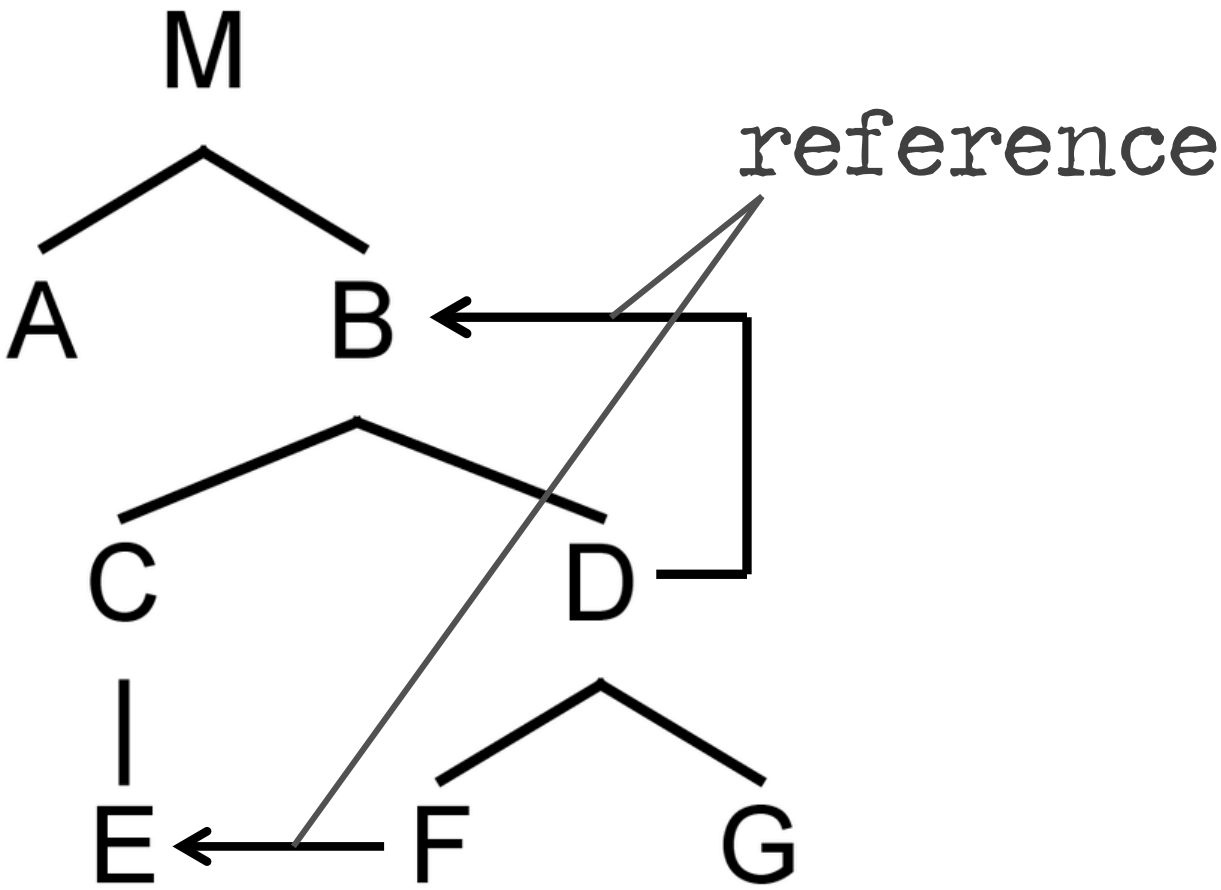


Programs and Fragments

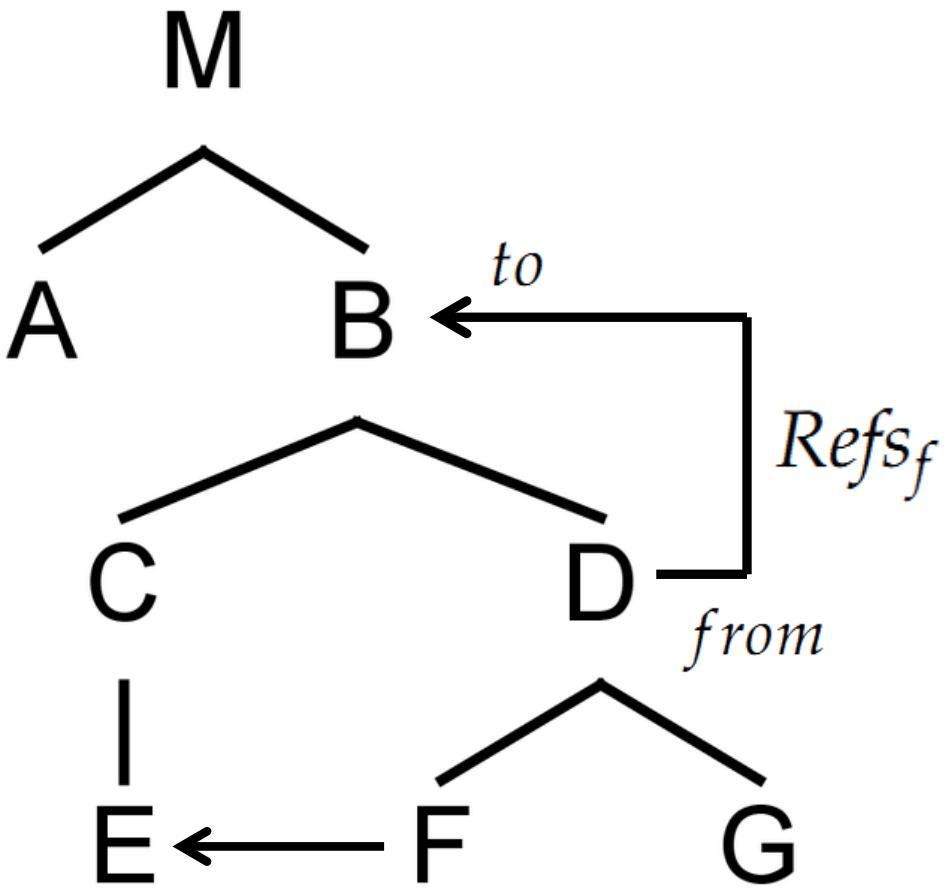


$$fo \Rightarrow element \rightarrow fragment$$

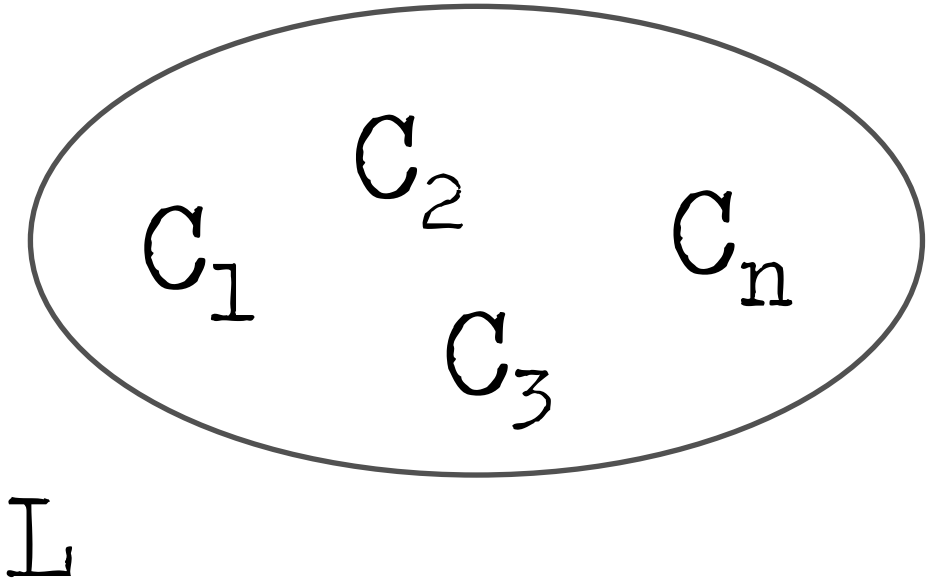
Programs are graphs, really.



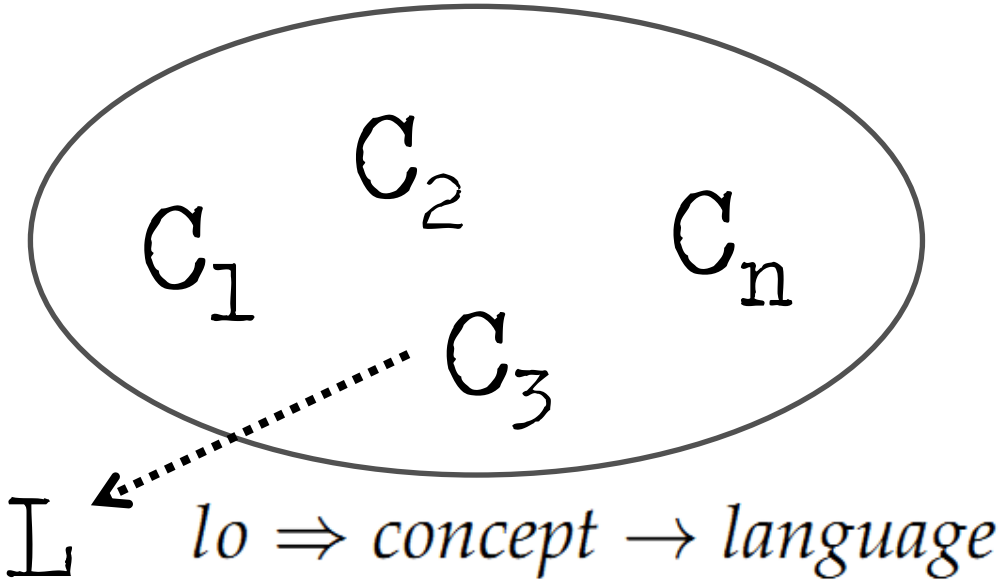
Programs are graphs, really.



Languages are sets of concepts

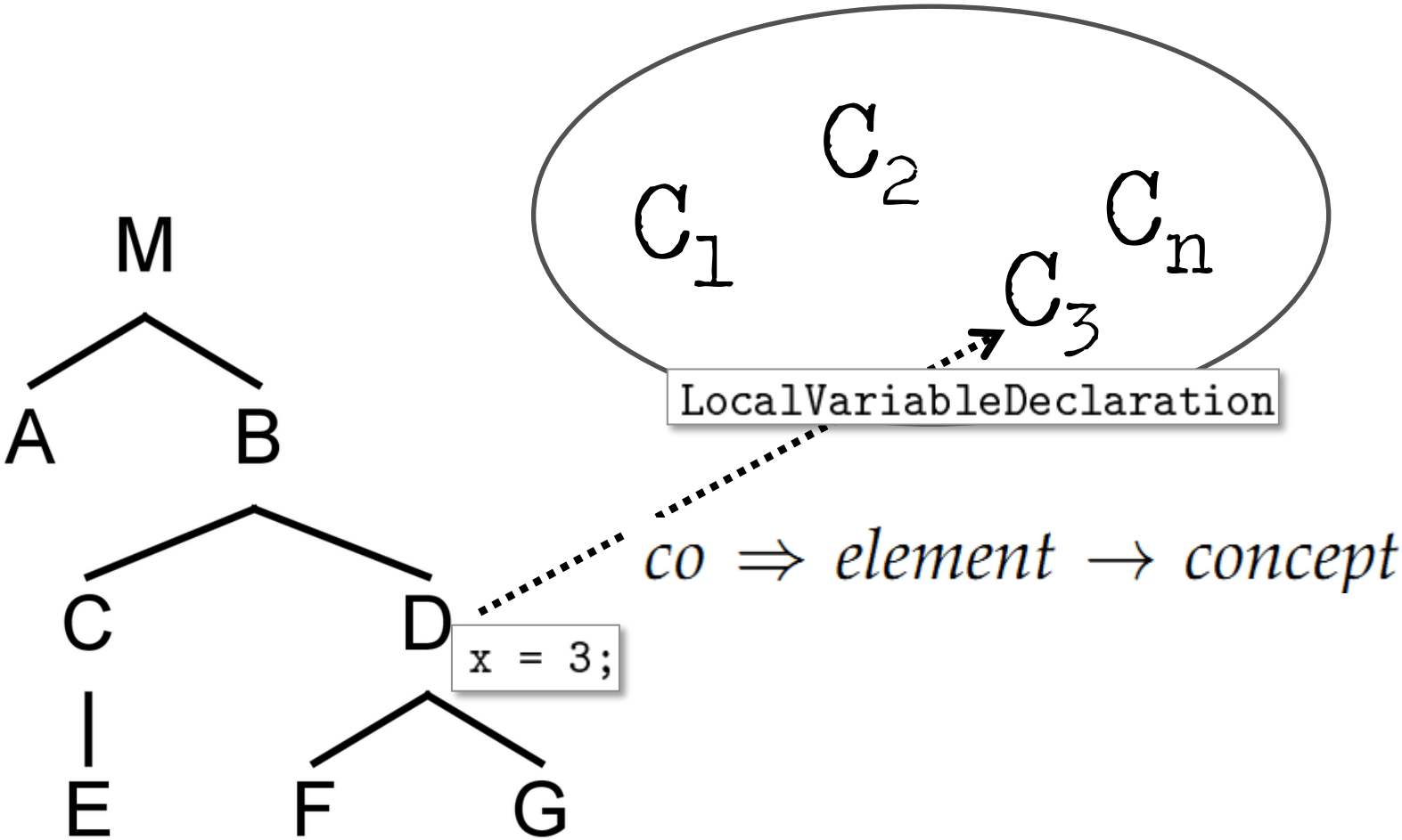


Languages are sets of concepts

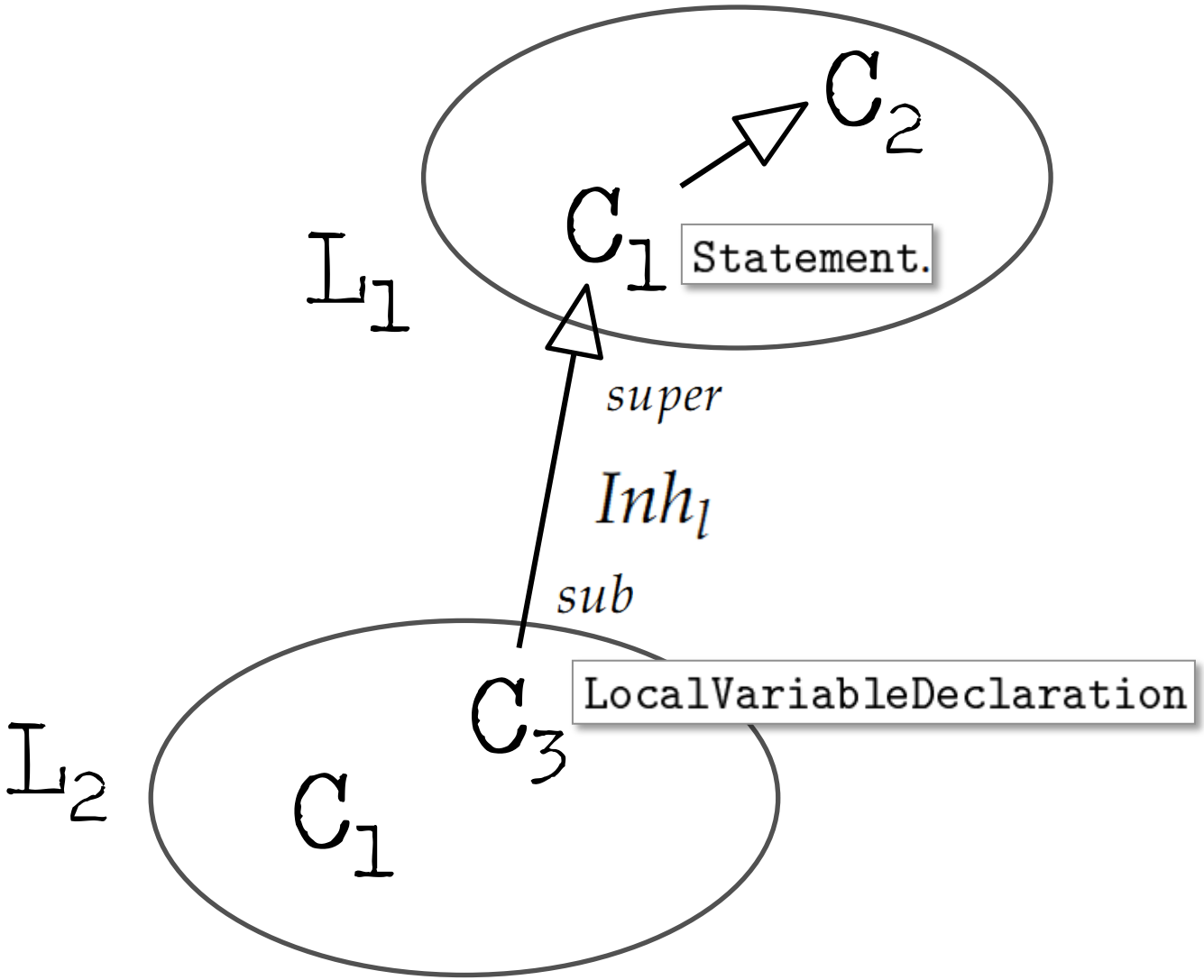


$lo \Rightarrow concept \rightarrow language$

Programs and languages



Language: concept inheritance



Language

does not depend on
any other language

$$\forall r \in Refs_l \mid lo(r.to) = lo(r.from) = l$$
$$\forall s \in Inh_l \mid lo(s.super) = lo(s.sub) = l$$
$$\forall c \in Cdn_l \mid lo(c.parent) = lo(c.child) = l$$

Independence

Fragment

does not depend on
any other fragment

$$\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f$$
$$\forall e \in E_f \mid lo(co(e)) = l$$

Independence

Hardware:

```
compressor compartment cc {
    static compressor c1
    fan ccfan
}
```

Cooling Algorithm

```
macro kompressorAus {
    set cc.c1->active = false
    perform ccfanabschalttask after 10 {
        set cc.ccfan->active = false
    }
}
```

Example

Refrigerators

Homogeneous

Fragment

everything expressed
with one language

$$\forall e \in E_f \mid lo(e) = l$$

$$\forall c \in Cdn_f \mid lo(c.parent) = lo(c.child) = l$$

```
module CounterExample from counterd imports nothing {

  var int theI;

  var boolean theB;

  var boolean hasBeenReset;

  statemachine Counter {
    in start() <no binding>
       step(int[0..10] size) <no binding>
    out someEvent(int[0..100] x, boolean b) <no binding>
        resetted() <no binding>
    vars int[0..10] currentVal = 0
         int[0..100] LIMIT = 10
    states (initial = initialState)
      state initialState {
        on start [ ] -> countState { send someEvent(100, true && false || true); }
      }
      state countState {
        on step [currentVal + size > LIMIT] -> initialState { send resetted(); }
        on step [currentVal + size <= LIMIT] -> countState { currentVal = currentVal + size; }
        on start [ ] -> initialState {   }
      }
  } end statemachine

  var Counter c1;

  exported test case test1 {
    initsm(c1);
    assert(0) isInState<c1, initialState>;
    trigger(c1, start);
    assert(1) isInState<c1, countState>;
  } test1(test case)
}
```

Heterogeneous

C

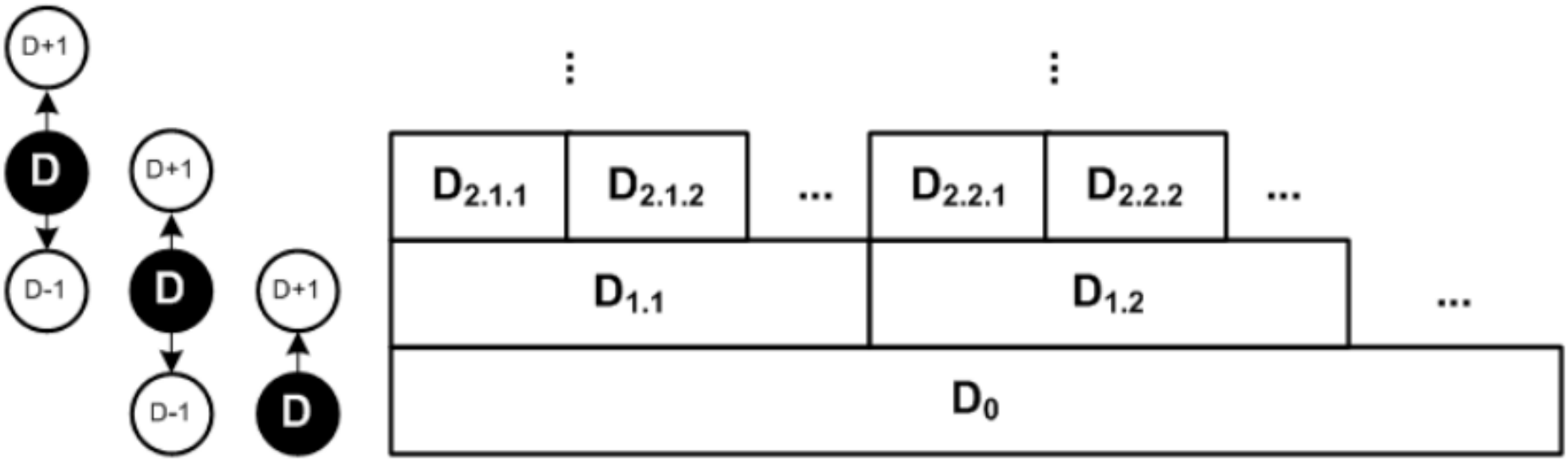
Statemachines

Testing

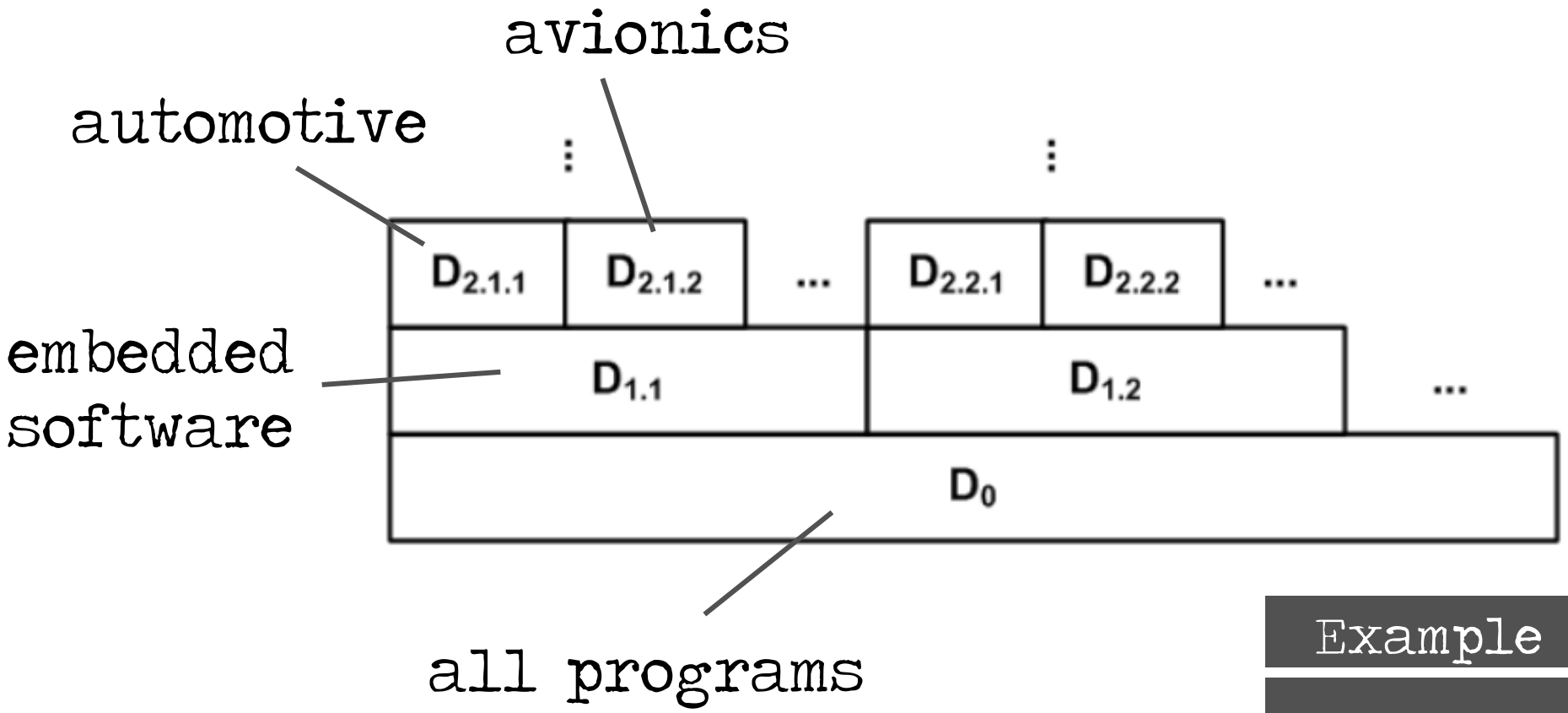
Example

Extended C

Domain Hierarchy



Domain Hierarchy



Example

Exten
ded C

Design Dimensions

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Expressivity

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

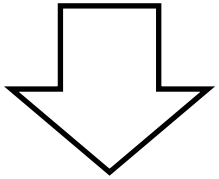
Shorter Programs

More Accessible Semantics

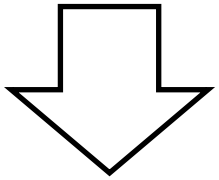
For a limited Domain!

Domain Knowledge encapsulated in language

Smaller Domain



More Specialized Language



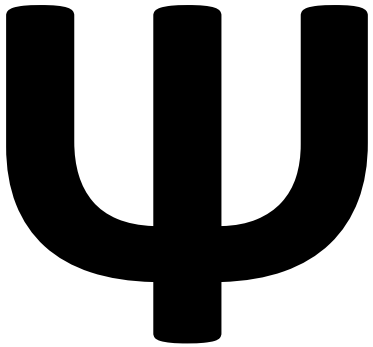
Shorter Programs

The do-what-I-want language

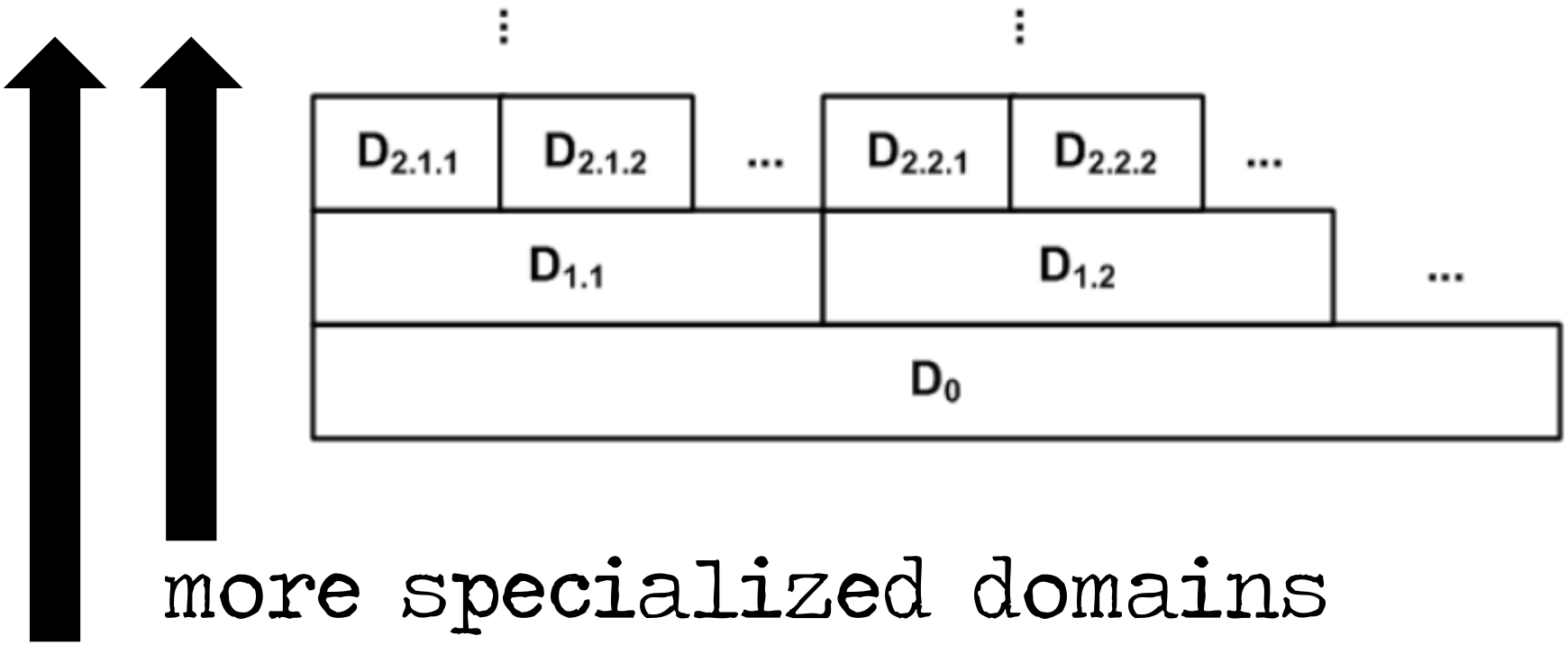
Ψ

Single Program vs. Class/Domain

No Variability!

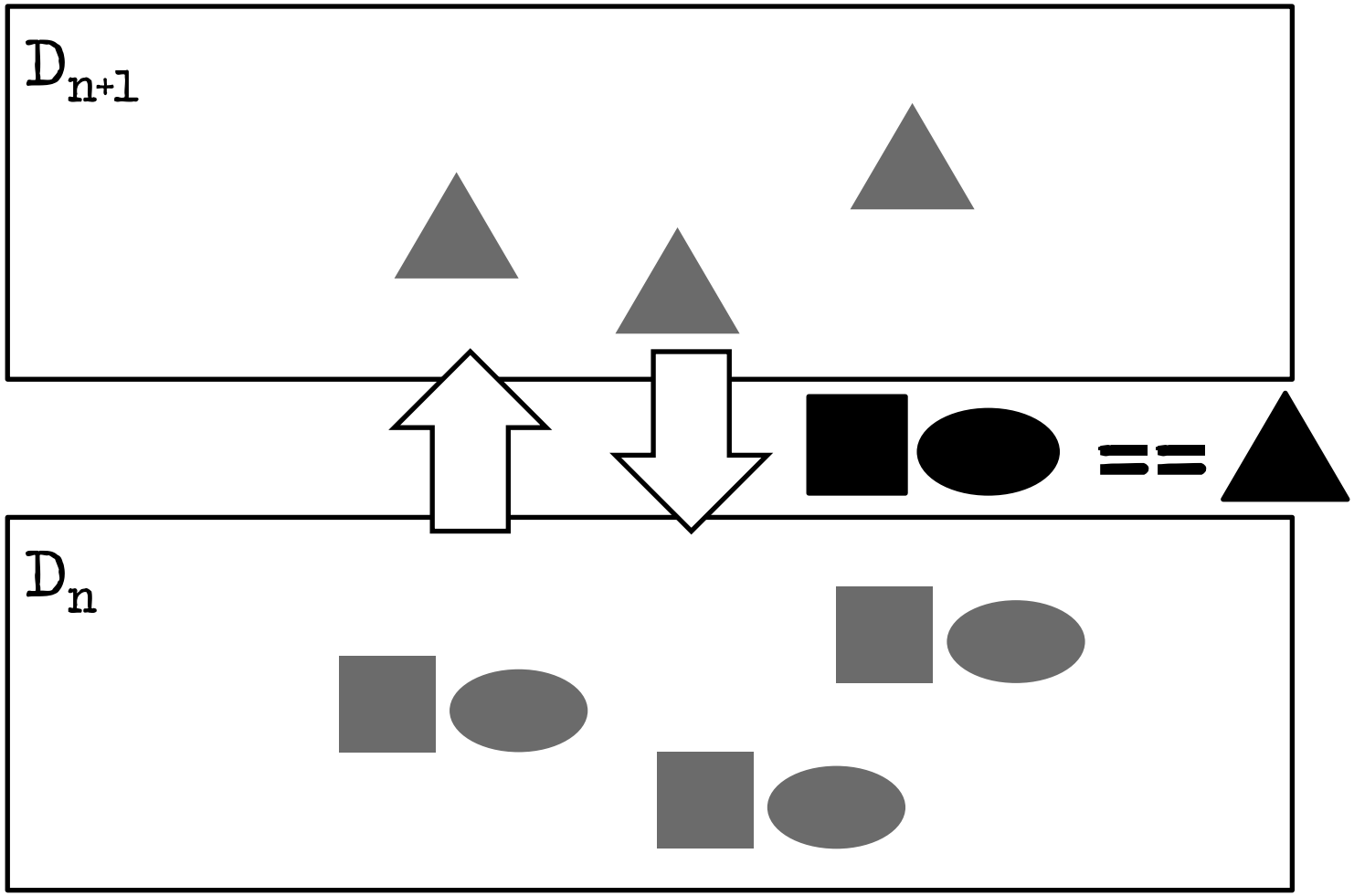


Domain Hierarchy



more specialized domains
more specialized languages

Reification



Reification

Transformation/
Generation

Language
Definition

```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

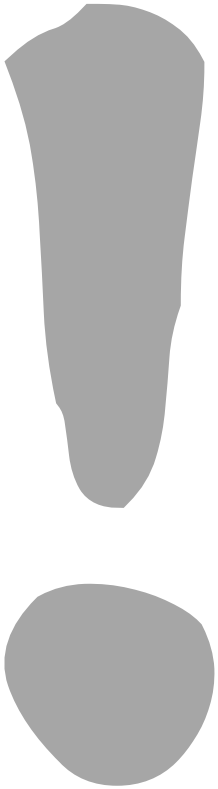
```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Overspecification!
Requires Semantic Analysis!

```
int[] arr = ...
for (int i=0; i<arr.size(); i++) {
    sum += arr[i];
}
```

```
int[] arr = ...
List<int> l = ...
for (int i=0;  i<arr.size(); i++) {
    l.add( arr[i] );
}
```

Linguistic Abstraction



```
for (int i in arr) {
    sum += i;
}
```

Declarative!
Directly represents Semantics.

```
seqfor (int i in arr) {
    l.add( arr[i] );
}
```

Def: DSL

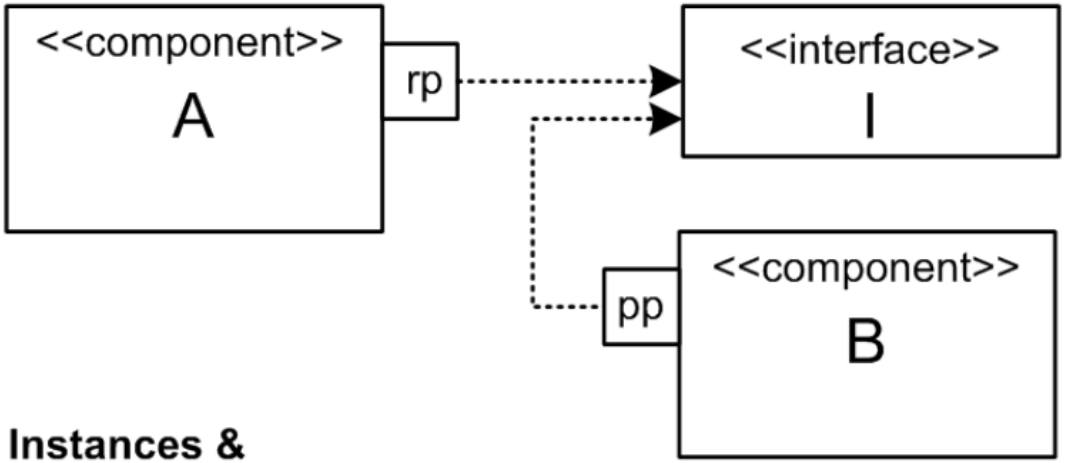
A DSL is a **language** at D that provides **linguistic abstractions** for **common patterns and idioms** of a language at D-1 when used within the domain D.

Def: DSL cont'd

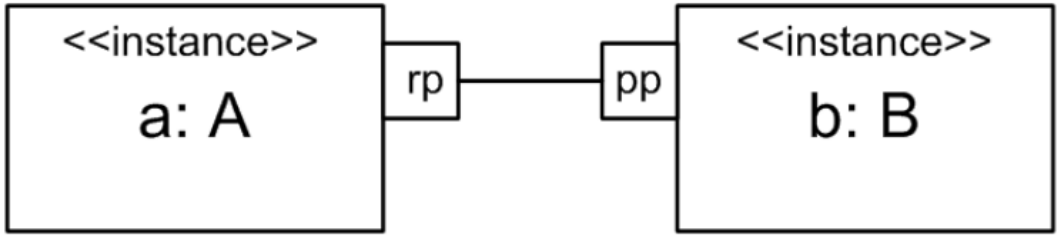
A good DSL does **not** require the use of patterns and idioms to express **semantically interesting** concepts in D. Processing tools do not have to do "semantic recovery" on D programs.

Declarative!

Another Example



Instances & Connectors

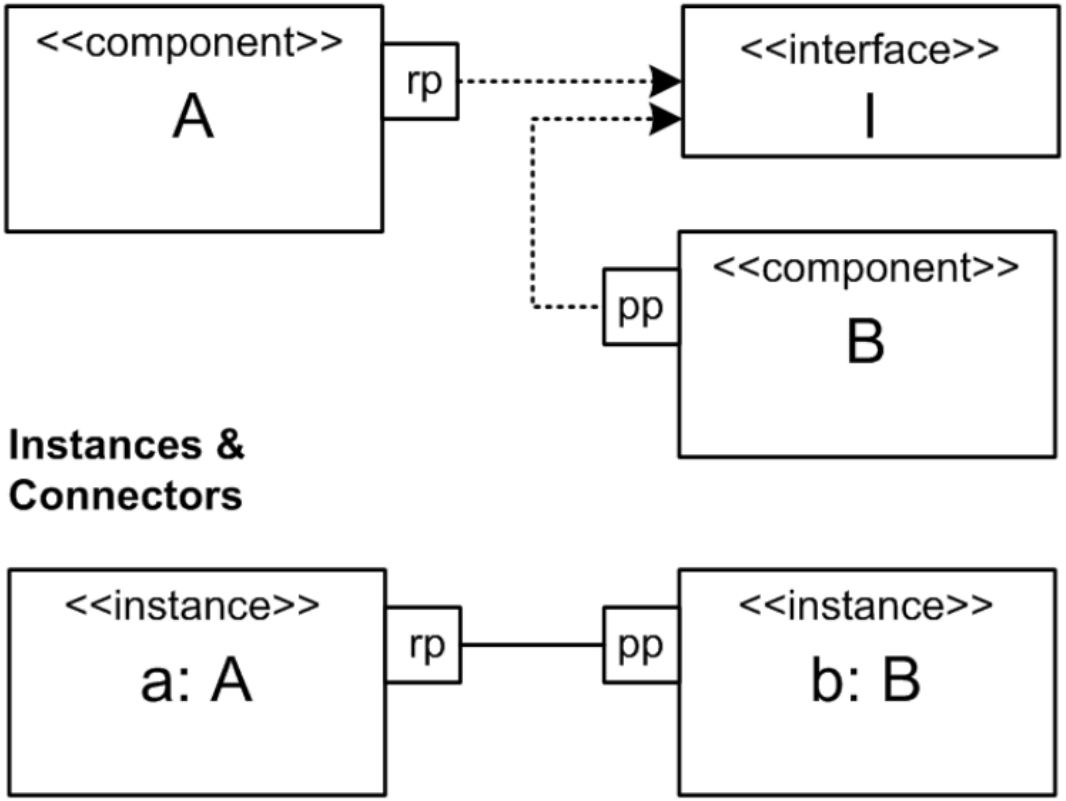



```
if (isConnected(port)) {
    port.doSomething();
}
```

Example

Exten
ded C

Another Example




```
if (isConnected(port) || true) {
    port.doSomething();
}
```

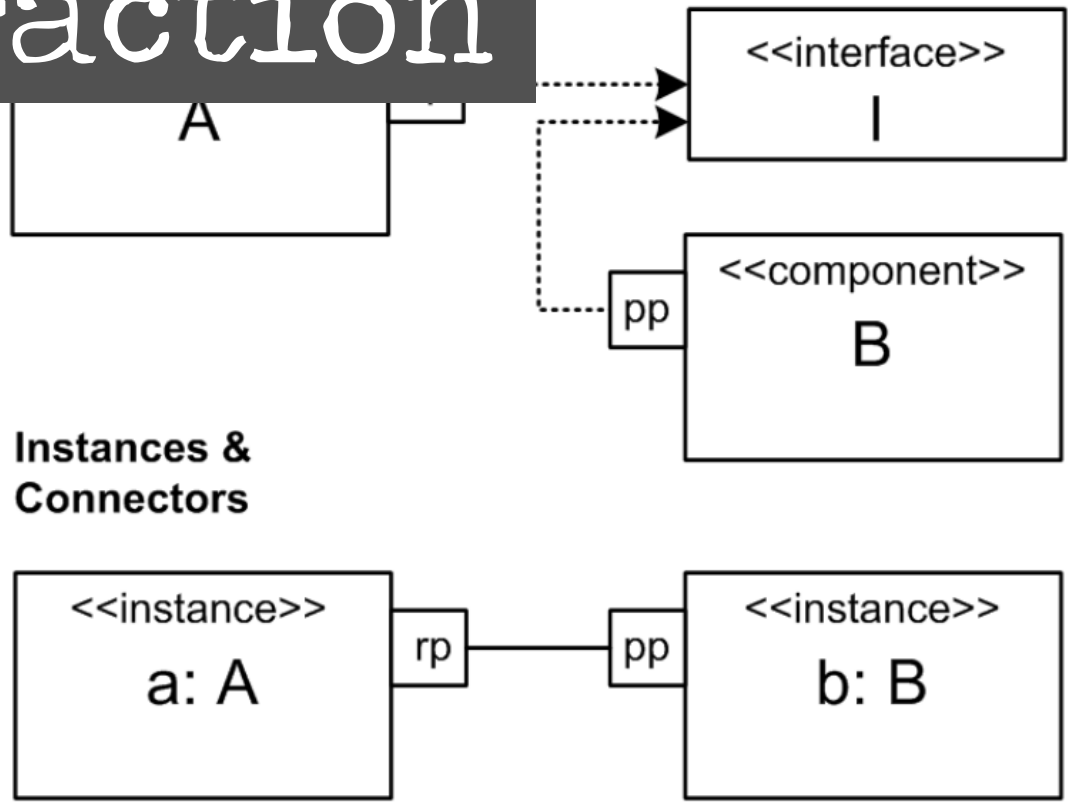
Turing Complete!

Requires Semantic Analysis!

Example

Extended C

Linguistic Abstraction




```
with port (port) {
    port.doSomething();
}
```

Example

Exten
ded C

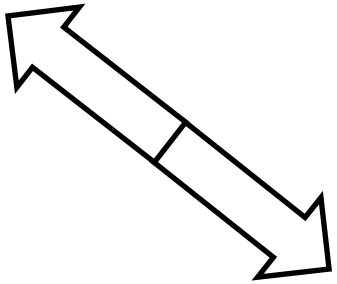
Linguistic Abstraction

```
exported component AnotherDriver extends Driver {
  ports:
    requires optional ILogger logger
    provides IDriver cmd
  contents:
    field int count = 0

    int setDriverValue(int addr, int value) <- op cmd.setDriverValue {
      with port (logger) {
        logger.log("some error message");
      } with port
      return 0;
    }
}
```

Example

Extended C

Linguistic Abstraction



In-Language Abstraction

Libraries
Classes
Frameworks

Linguistic Abstraction

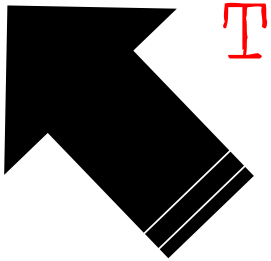
Analyzable
Better IDE Support

In-Language Abstraction

User-Definable
Simpler Language

Linguistic Abstraction

Analyzable
Better IDE Support



Special Treatment!

In-Language Abstraction

User-Definable
Simpler Language

Linguistic Abstraction

Std Lib

In-Language Abstraction

Std Lib

```
lib stdlib {

    command compartment::coolOn
    command compartment::coolOff
    property compartment::totalRuntime: int readonly
    property compartment::needsCooling: bool readonly
    property compartment::couldUseCooling: bool readonly
    property compartment::targetTemp: int readonly
    property compartment::currentTemp: double readonly
    property compartment::isCooling: bool readonly

}
```

Example

Refrigerators

Coverage

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Domain D_L defined inductively by L

(the domain that can be expressed by L)

$C_L(L) == 1$ (by definition)

not very interesting!

Def: Coverage

to what extend can a language L cover a domain D

$$C_D(L) = \frac{number\ of\ P_D\ programs\ expressable\ by\ L}{number\ of\ programs\ in\ domain\ D}$$

Def: Coverage

why would $C_D(L)$ be != 1?

1) L is deficient

2) L is intended to cover only a subset of D, corner cases may make L too complex

Rest must be expressed in D_{-1}

Def: Coverage

Coverage is full.
You call always write C.

Example

Exten
ded C

Def: Coverage

Only a particular style of web apps are suported.

Many more are conceivable.

Example

WebDSL

Def: Coverage

DSLs are continuously evolved so the relevant parts of the deductive domain are supported.

Example	Example	Example
Compo nents	Refrige rators	Pension Plans

Semantics & Execution

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Static Semantics

Execution Semantics

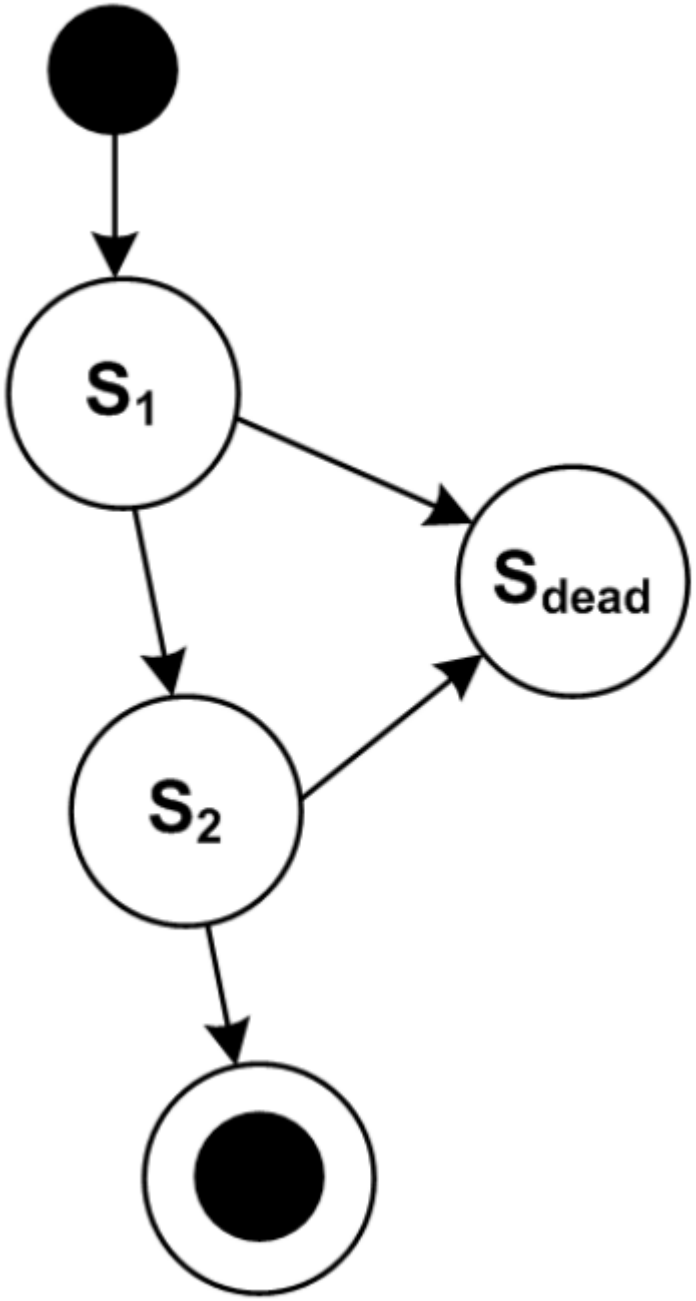
Static Semantics

Execution Semantics

Static Semantics

Constraints

Type Systems



Unique State Names

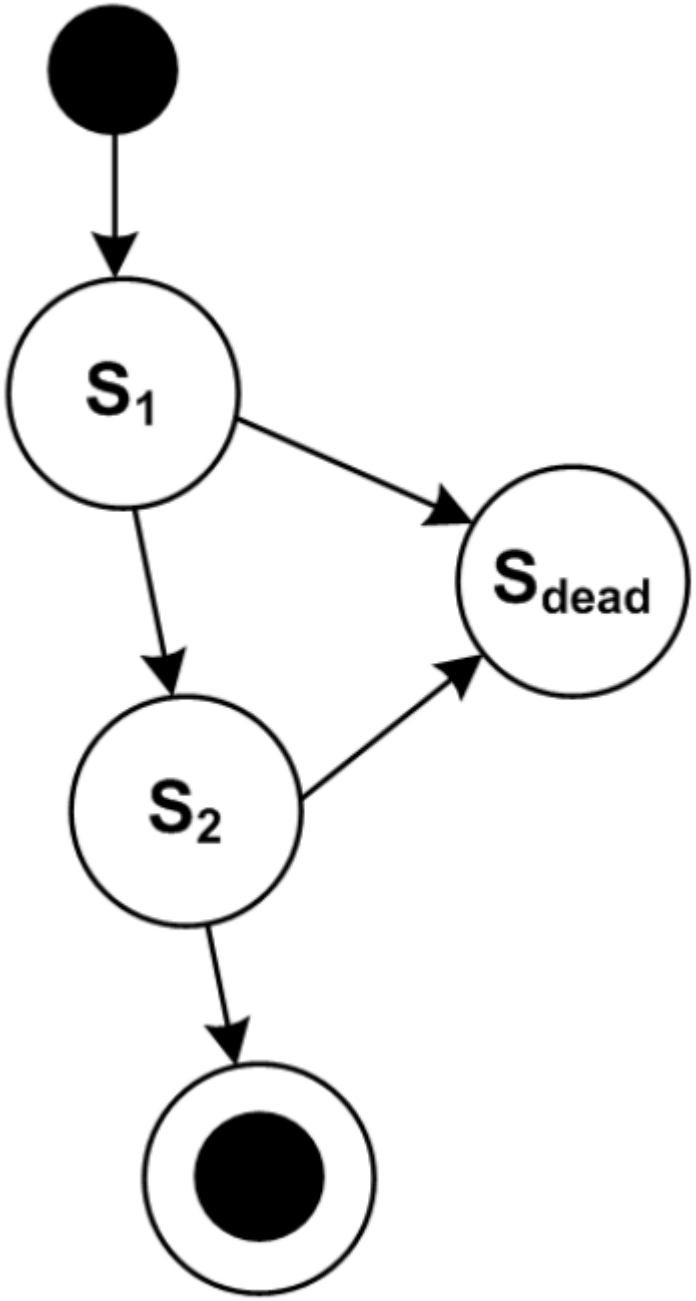
Unreachable States

Dead End States

...

Example

Exten
ded C



Unique State Names

Unreachable States

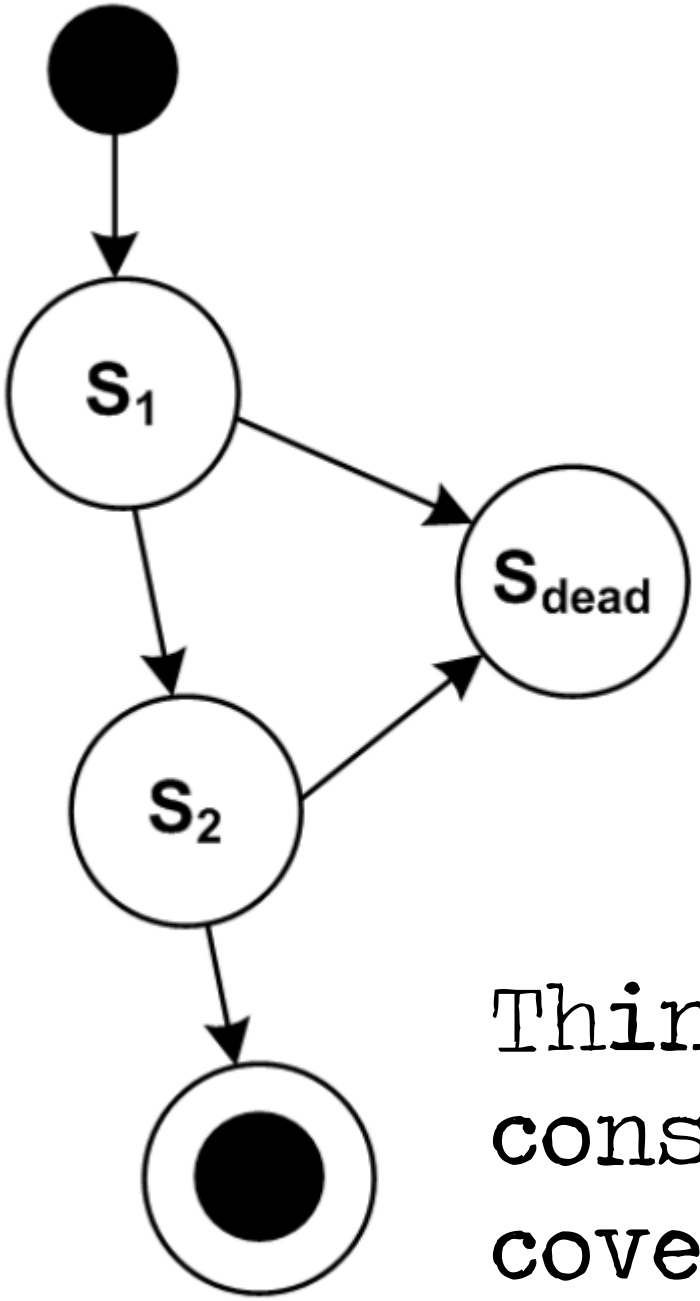
Dead End States

...

Easier to do on a declarative Level!

Example

Extended C



Unique State Names

Unreachable States

Dead End States

...

Easier to do on a declarative Level!

Thinking of all constraints is a coverage problem!

Example

Exten
ded C

```
var int x = 2 * someFunction(sqrt(2));
```

Assign fixed types

Derive Types

Calculate Common Types

Check Type Consistency

What does a type system do?

Intent + Check

```
var int x = 2 *
    someFunction(sqrt(2));
```

Derive

```
var x = 2 * some
    Function(sqrt(2));
```

More code

Better error messages

Better Performance

More convenient

More complex checkers

Harder to understand for users

```
macro kompressorAus {
    set cc.c1->active = "aString"
    perform ccfanausscha
        set cc.ccfan->ac
    }
}
```

incompatible type BoolType|COMPARABLE and StringType (on a AssignmentStatement)

Example

Refrige
rators

What does it
all mean?

Execution Semantics

Def: Semantics

… via mapping to lower level

$$semantics(p_{L_D}) := q_{L_{D-1}}$$

$$where\ OB(p_{L_D}) == OB(q_{L_{D-1}})$$

OB: Observable Behaviour (Test Cases)

Def: Semantics

… via mapping to lower level

$$semantics(p_{L_D}) := q_{L_{D-1}}$$

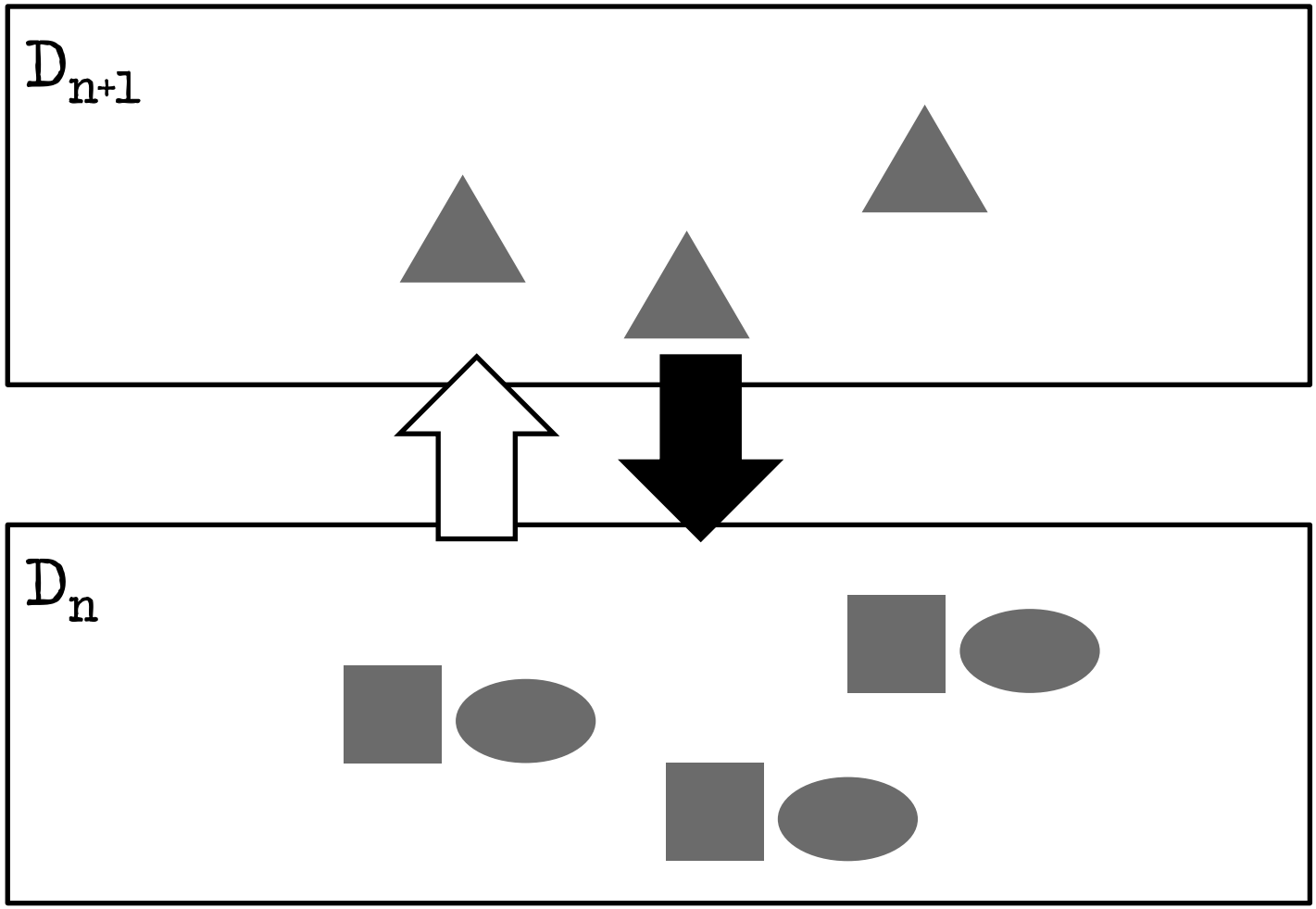
$$where\ OB(p_{L_D}) == OB(q_{L_{D-1}})$$

L_D

$\Downarrow$ Transformation
Interpretation

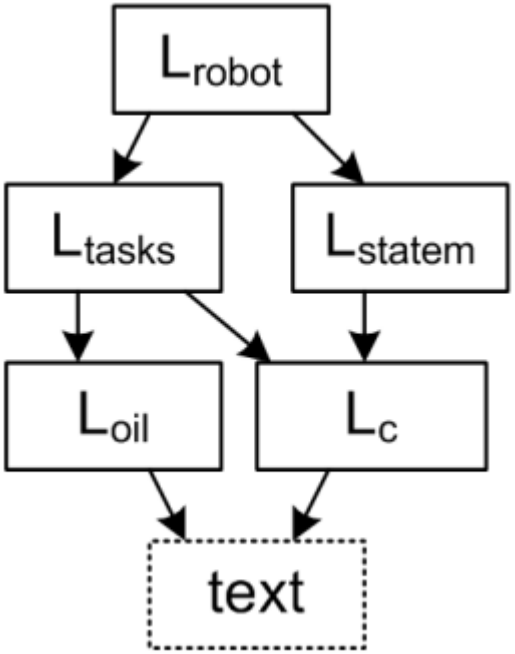
L_{D-1}

Transformation



Transformation

```
module impl imports <<imports>> {

  int speed( int val ) {
    return 2 * val;
  }

  robot script stopAndGo
    block main on bump
      accelerate to 12 + speed(12) within 3000
      drive on for 2000
      turn left for 200
      decelerate to 0  within 3000
      stop

}
```



Example

Extended C

Transformation

L_D

$\Downarrow$ Transformation

L_{D-1} Known Semantics!

Transformation

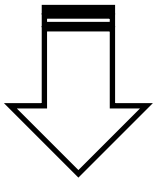
L_D

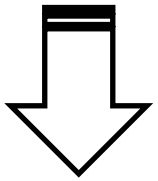
Correct!? $\Downarrow$ Transformation

L_{D-1}

Transformation

Tests (D) L_D





Transformation

Tests (D-1) L_{D-1}

Run tests on both levels; all pass.
Coverage Problem!

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

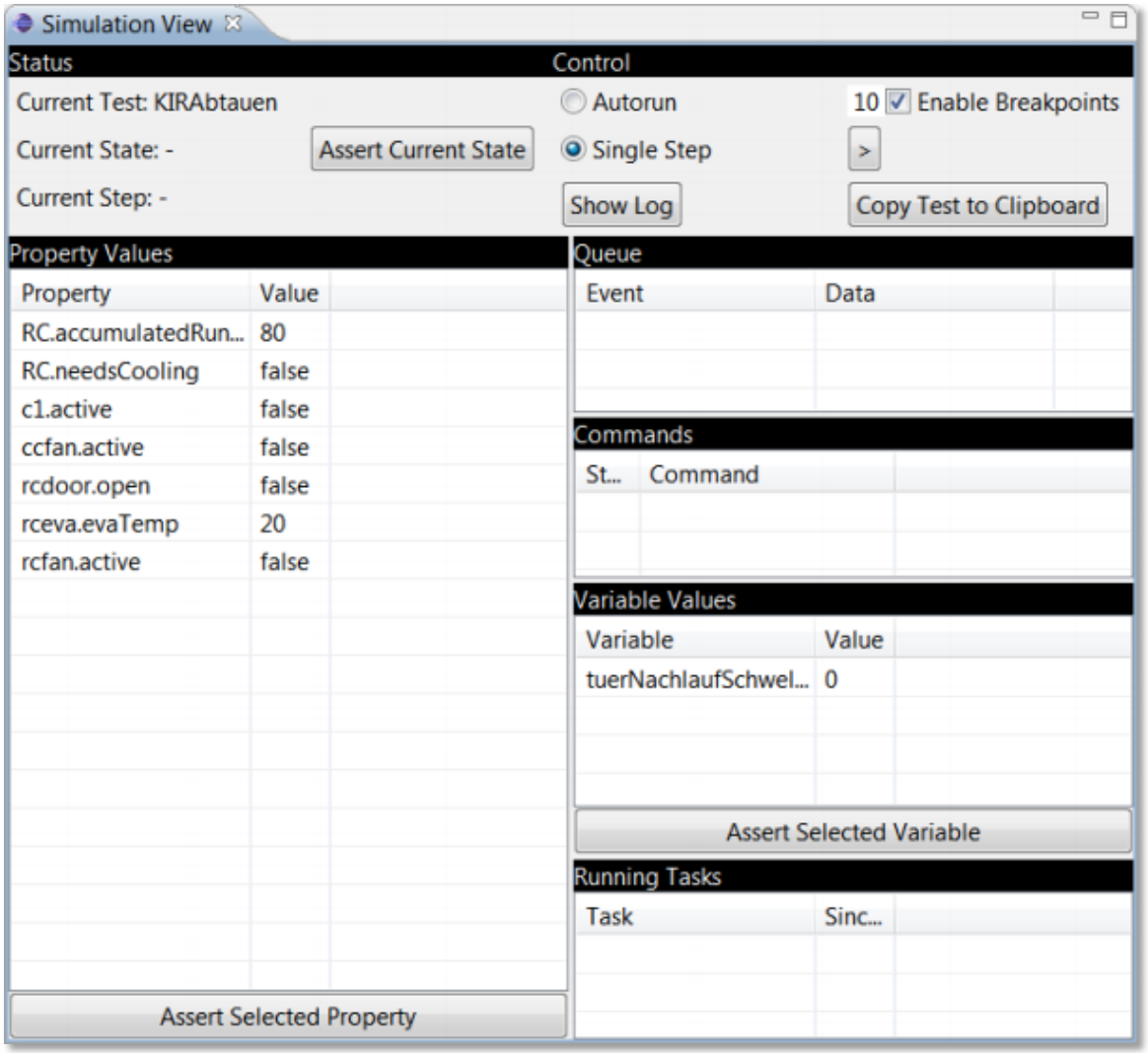
Refrigerators

Transformation

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	

Example

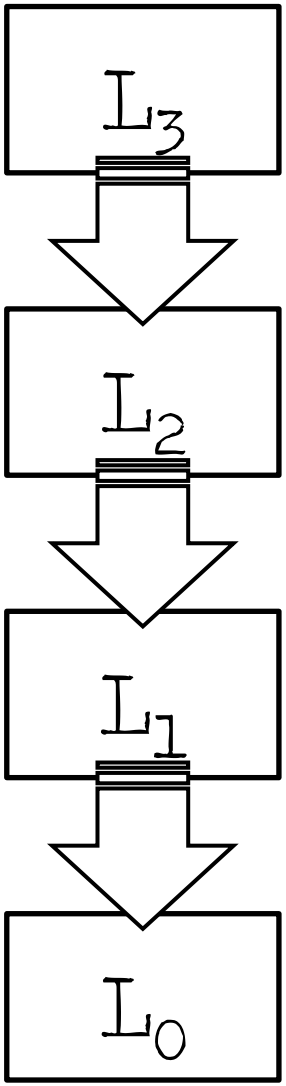
Pension Plans



Example

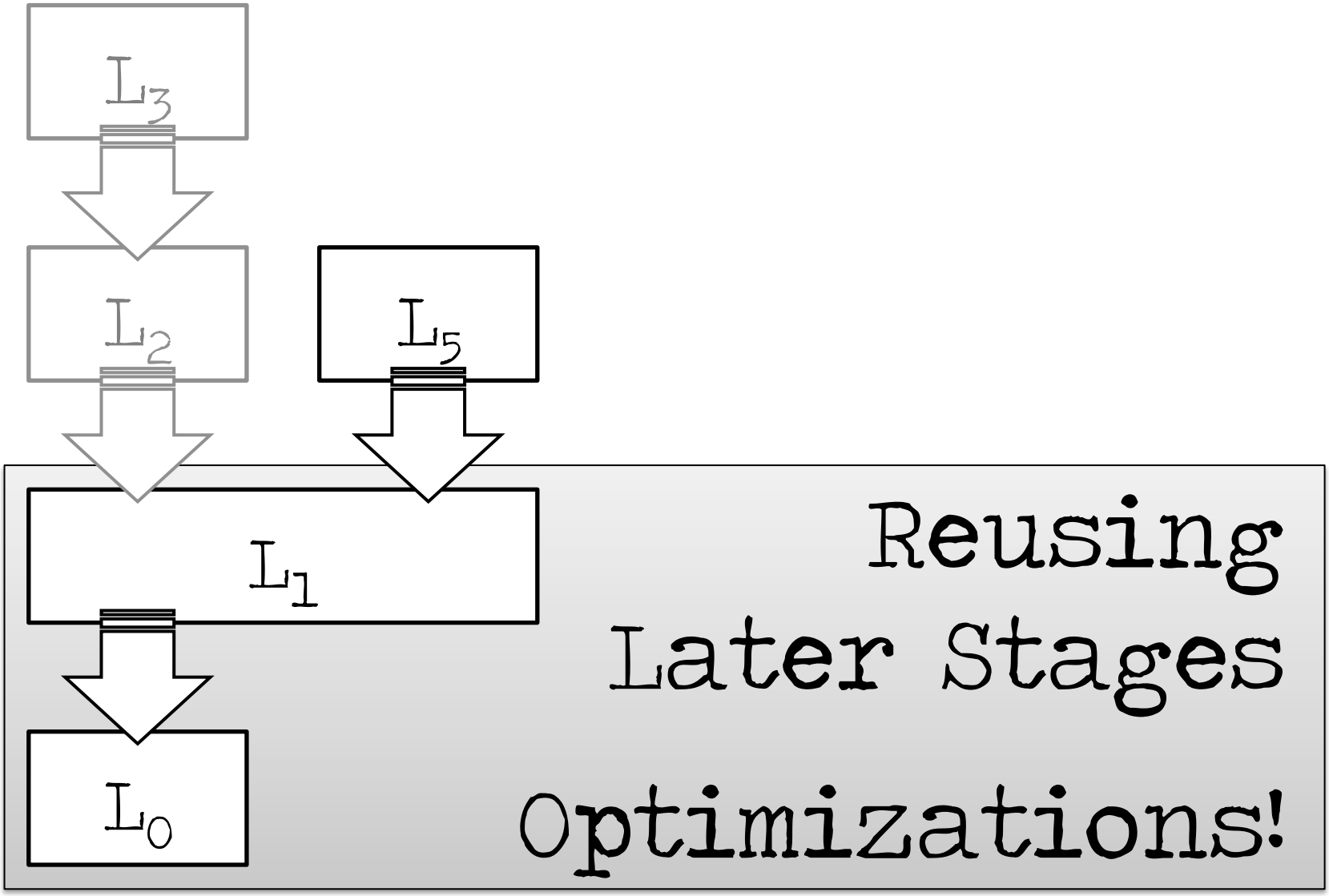
Refrigerators

Multi-Stage

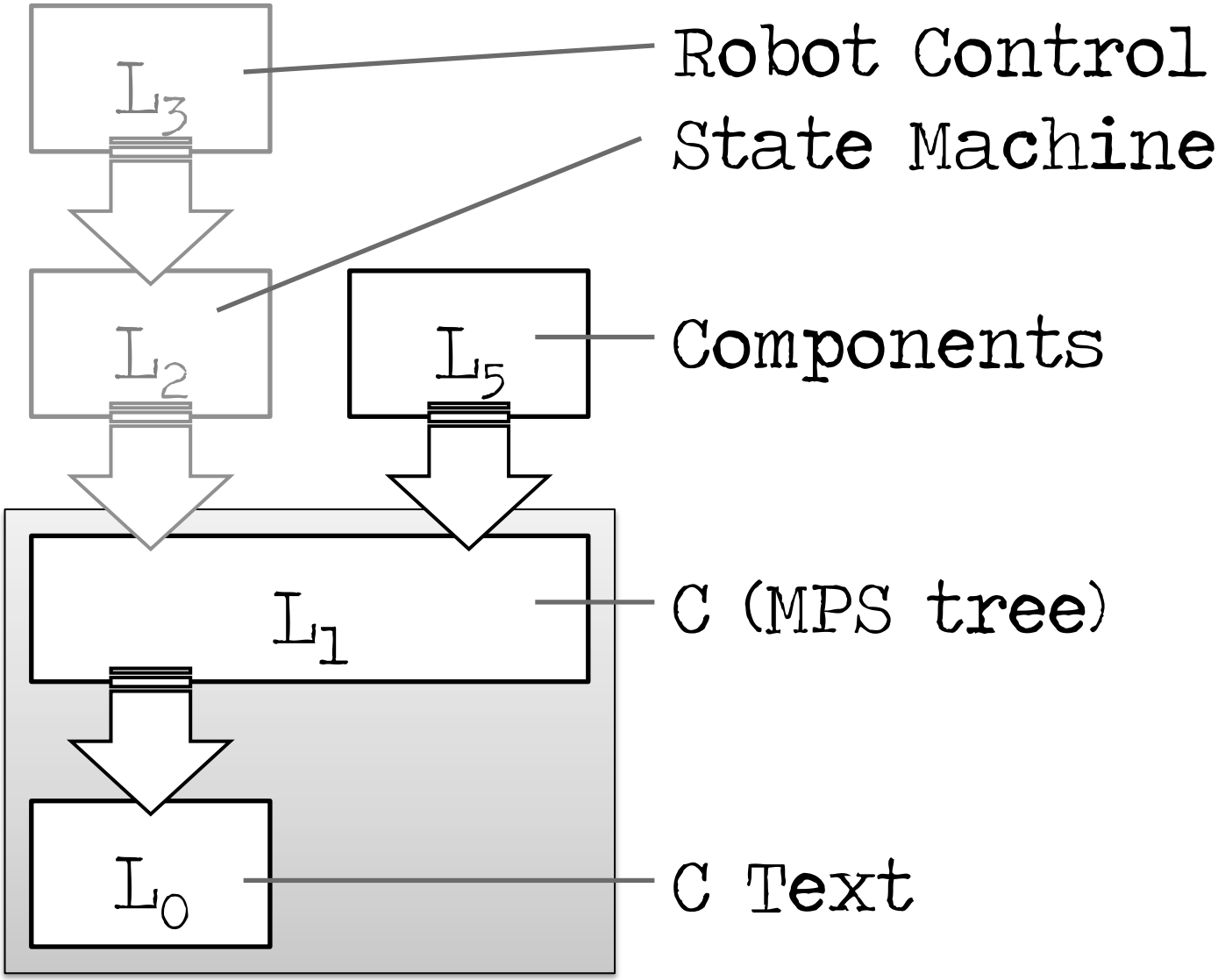


Modularization

Multi-Stage: Reuse



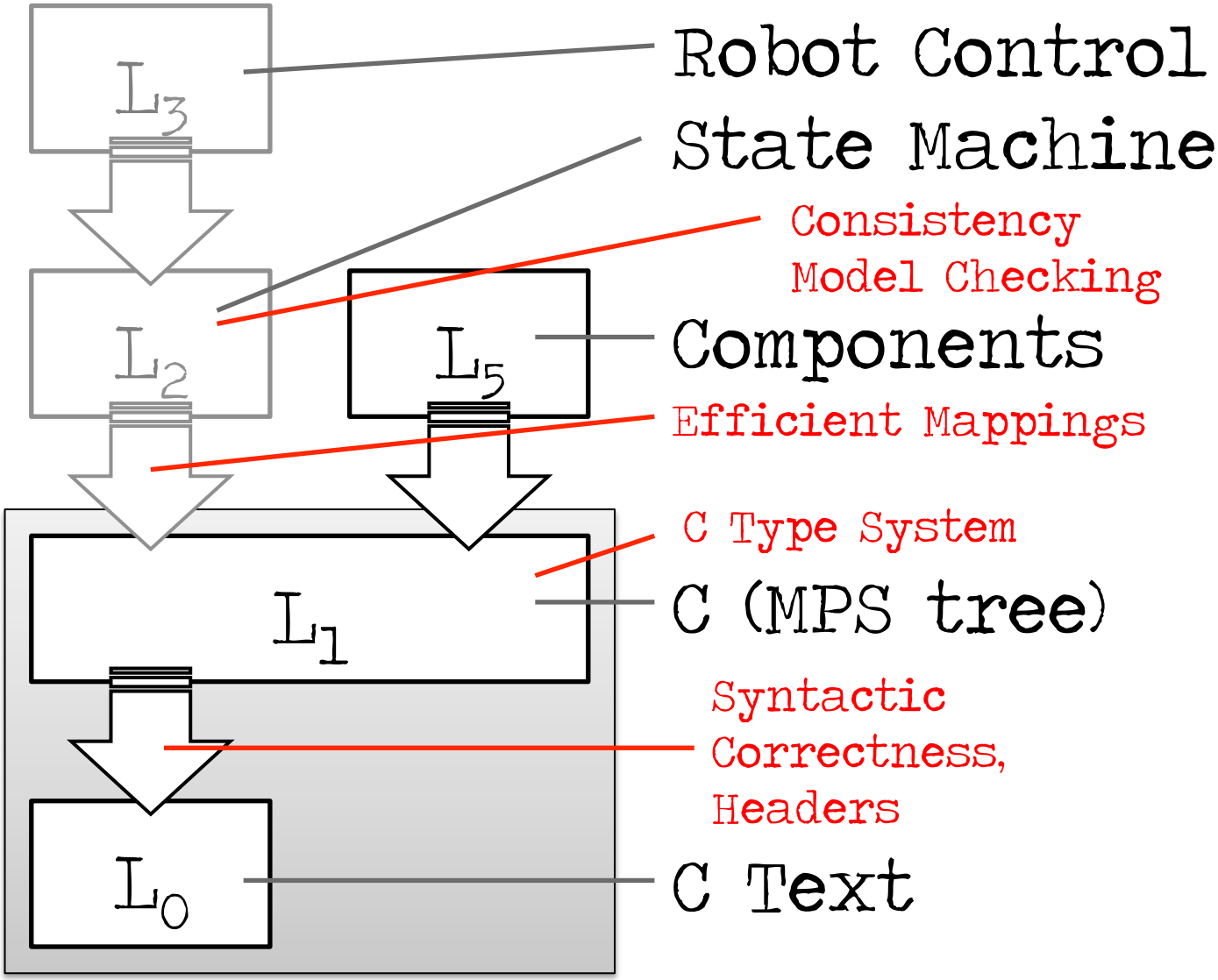
Multi-Stage: Reuse



Example

Exten
ded C

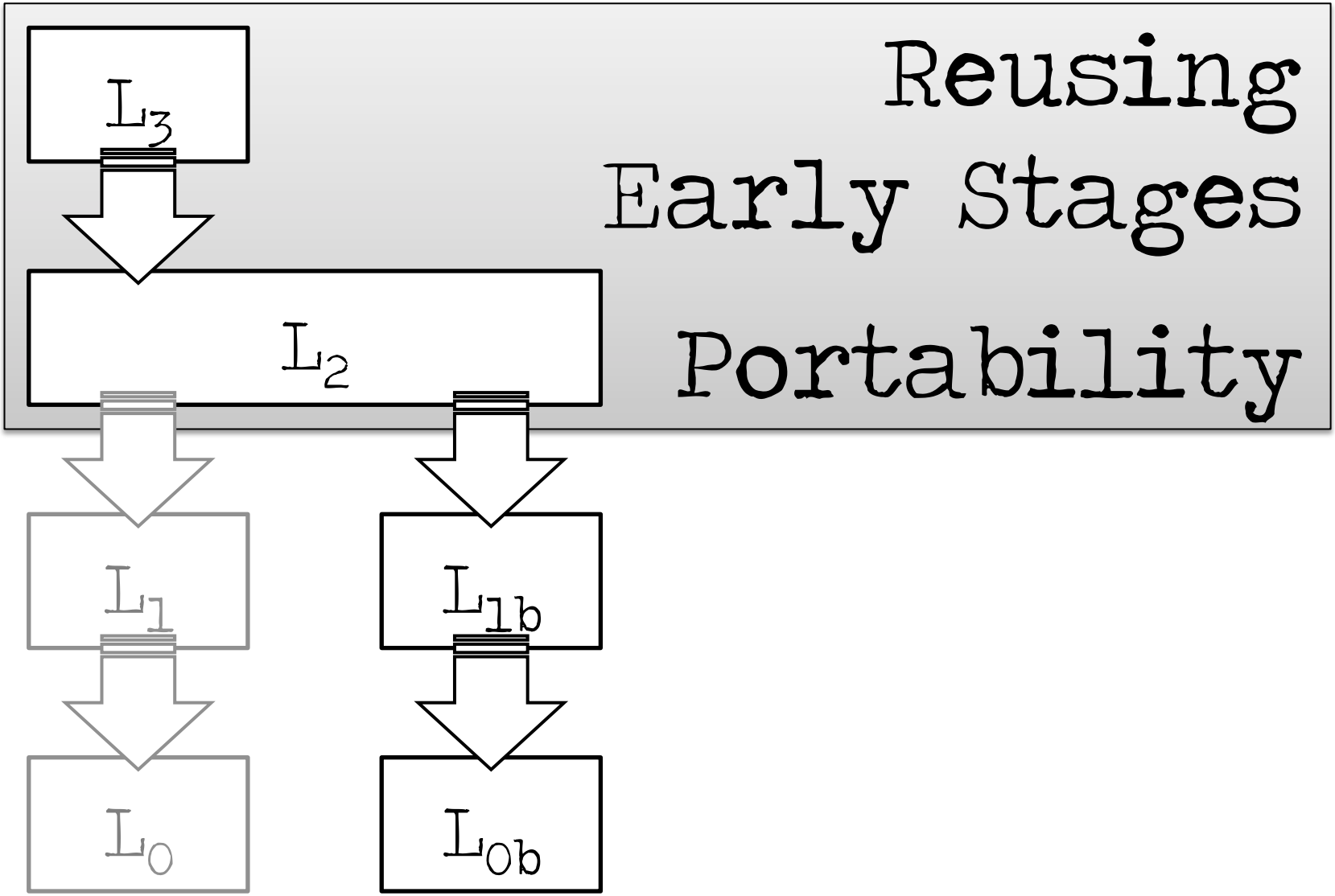
Multi-Stage: Reuse



Example

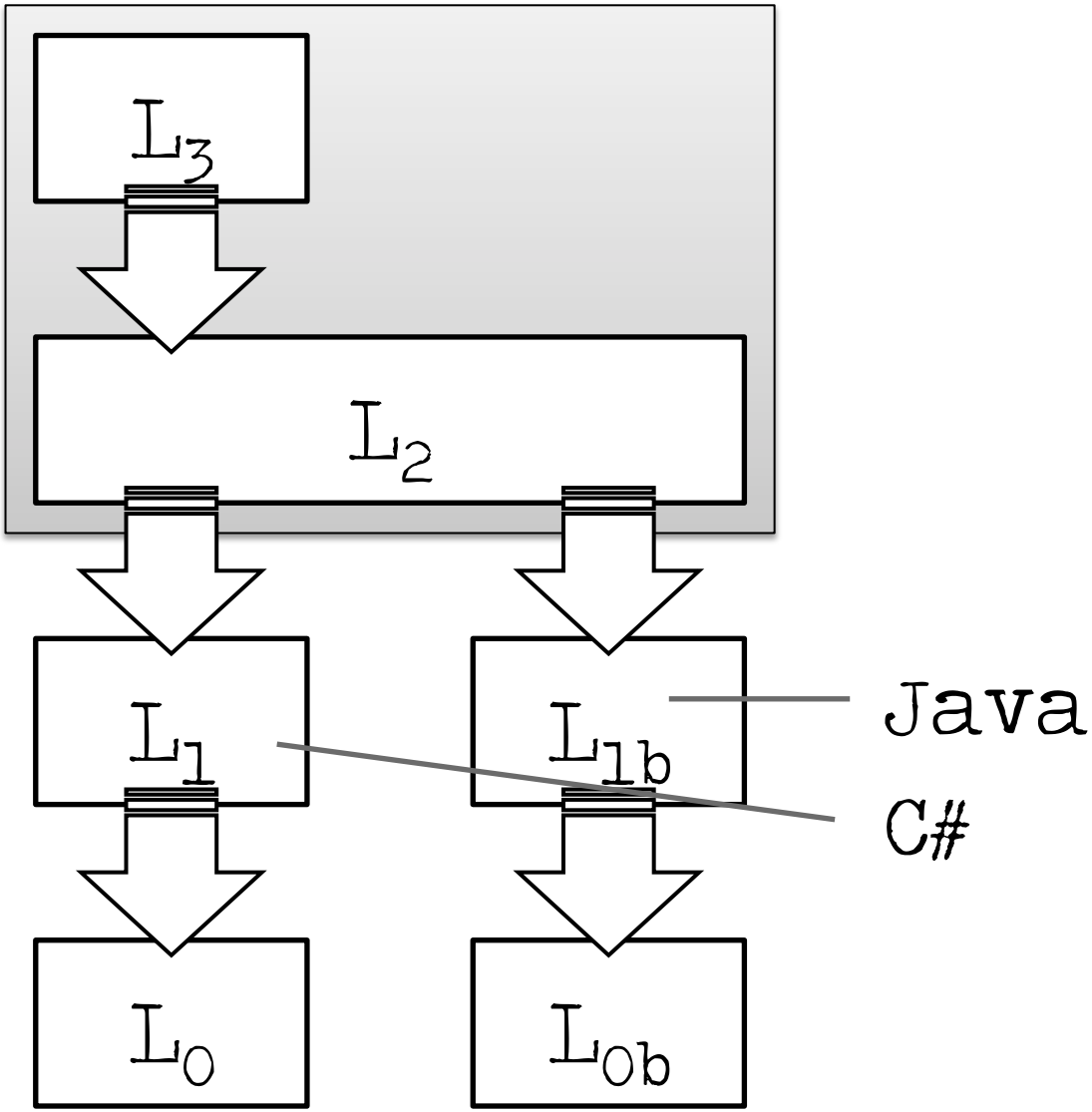
Exten
ded C

Multi-Stage: Reuse



Reusing
Early Stages
Portability

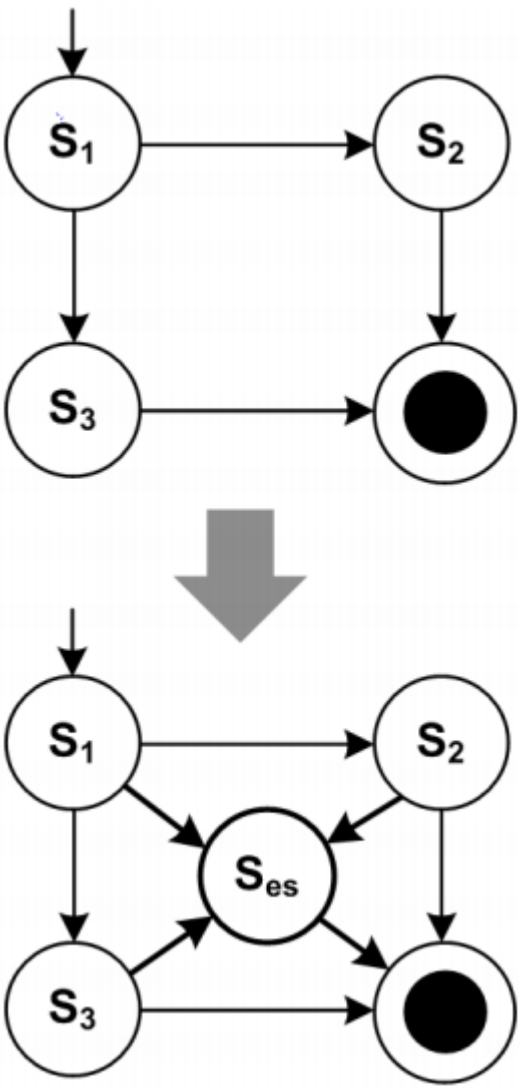
Multi-Stage: Reuse



Example

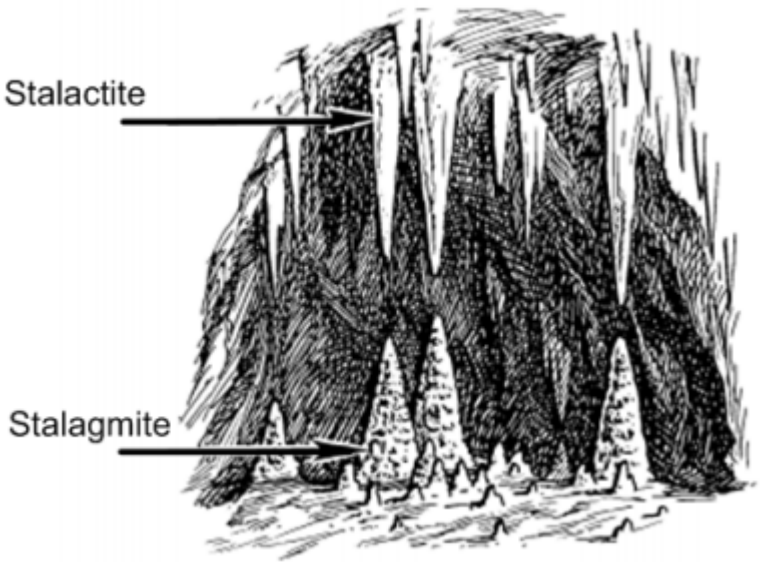
Extended C

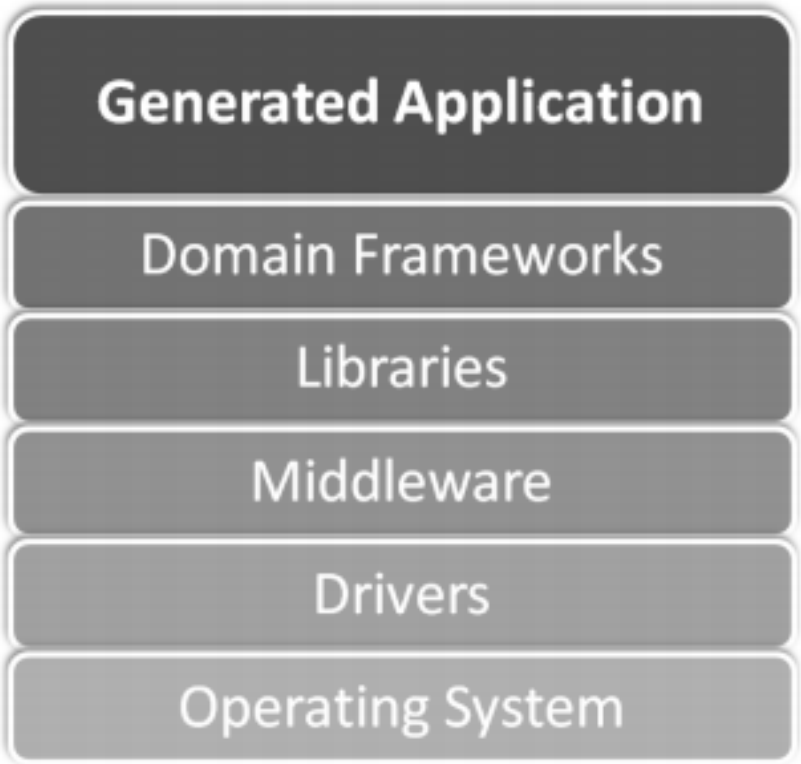
Multi-Stage: Preprocess



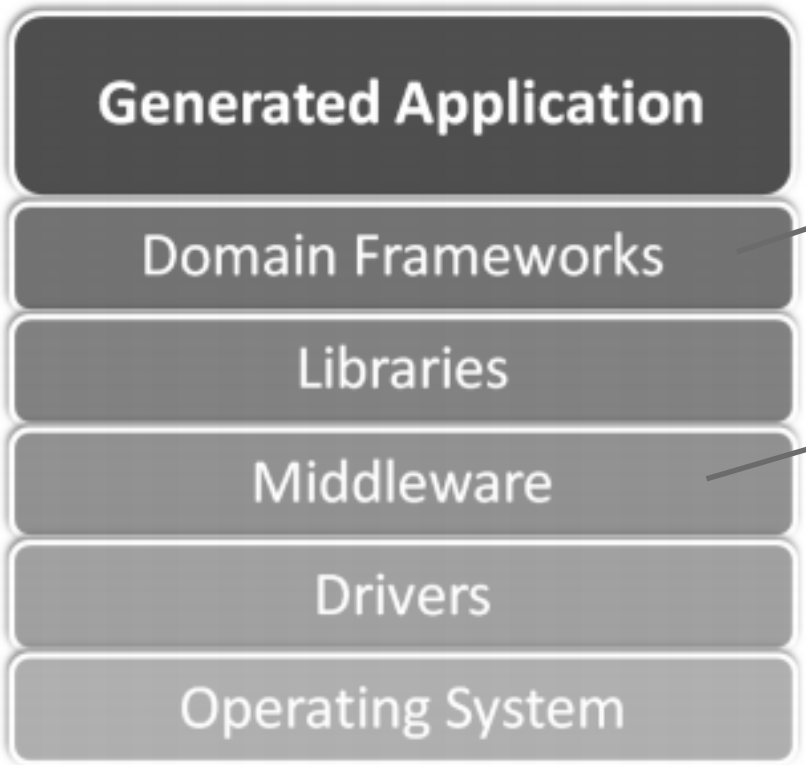
Adding an optional, modular emergency stop feature

Platform


Stalactite
Stalagmite


Generated Application
Domain Frameworks
Libraries
Middleware
Drivers
Operating System

Platform



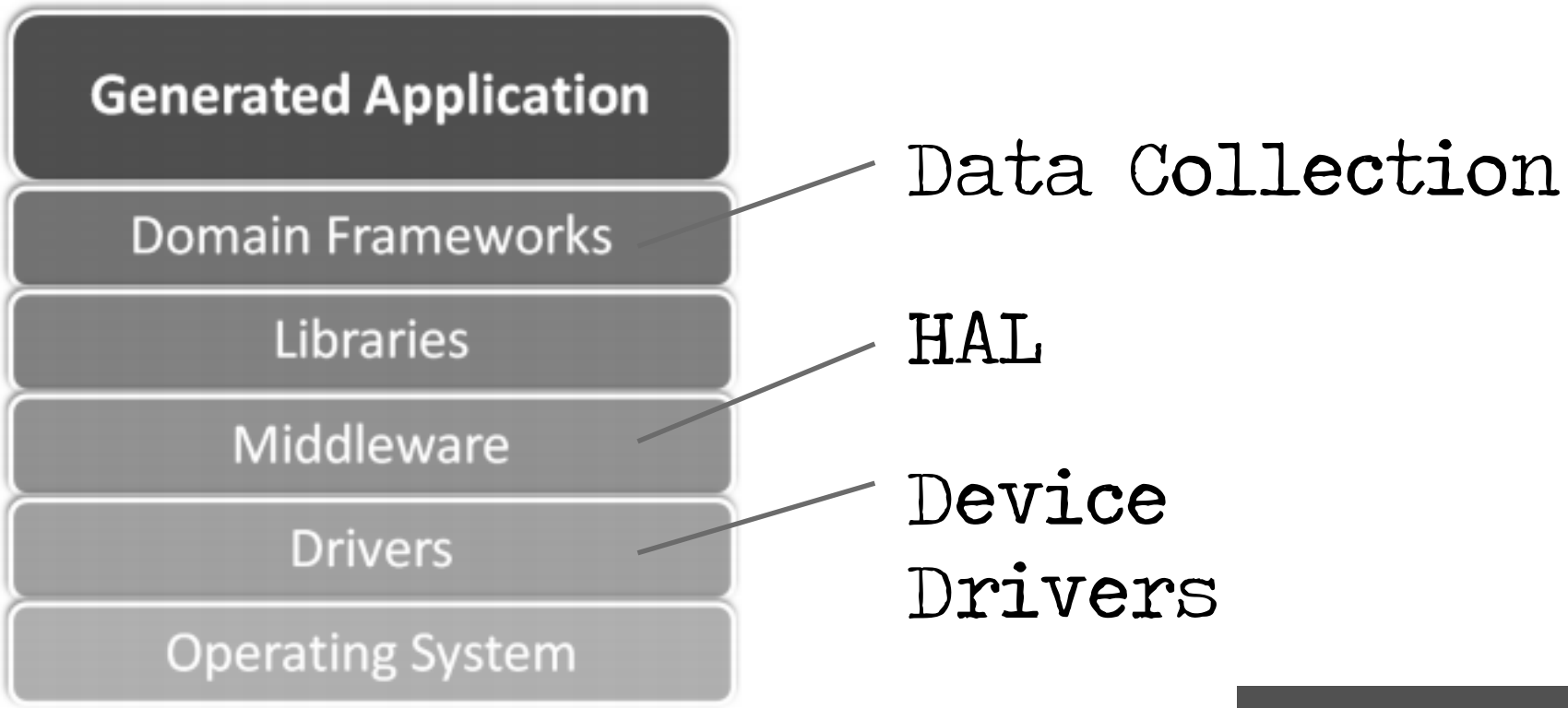
Temporal Data Versioning

Database Access
Transactions
Distribution
(JEE)

Example

Pension Plans

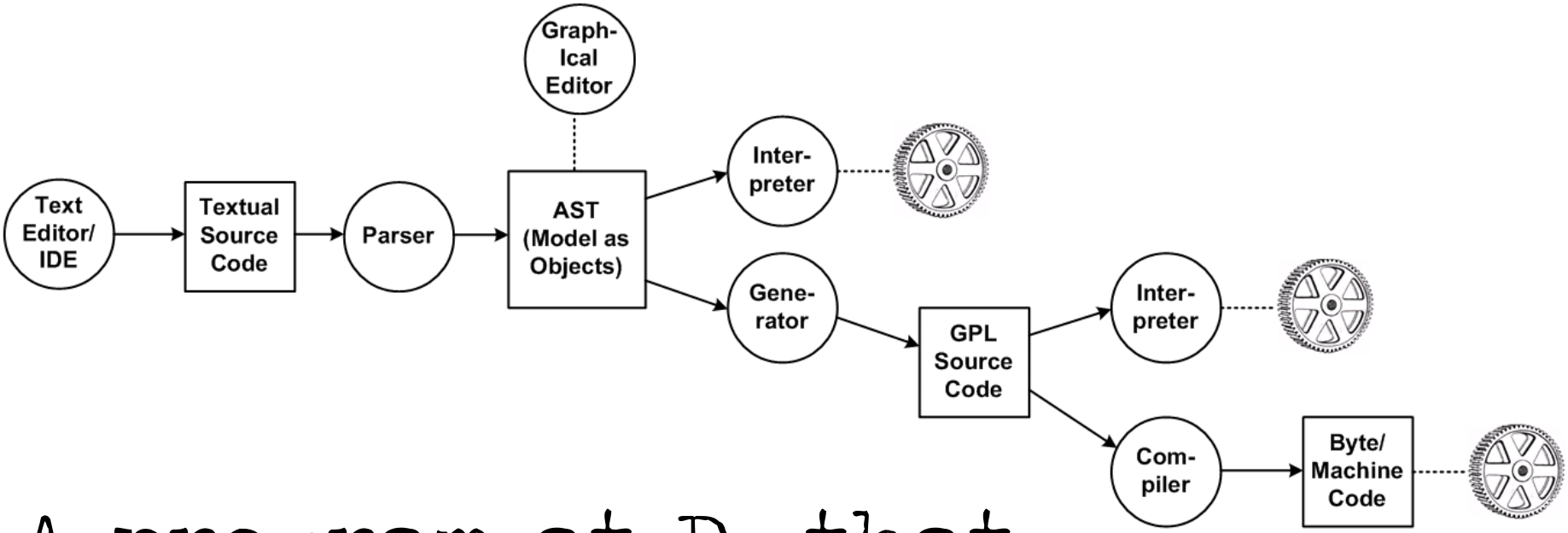
Platform



Example

Refrigerators

Interpretation



A program at D_0 that acts on the structure of an input program at $D_{>0}$

Interpretation

A program at D_0 that
acts on the structure
of an input program at $D_{>0}$

imperative → step through
functional → eval recursively
declarative→ ? solver ?

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	8250

Example

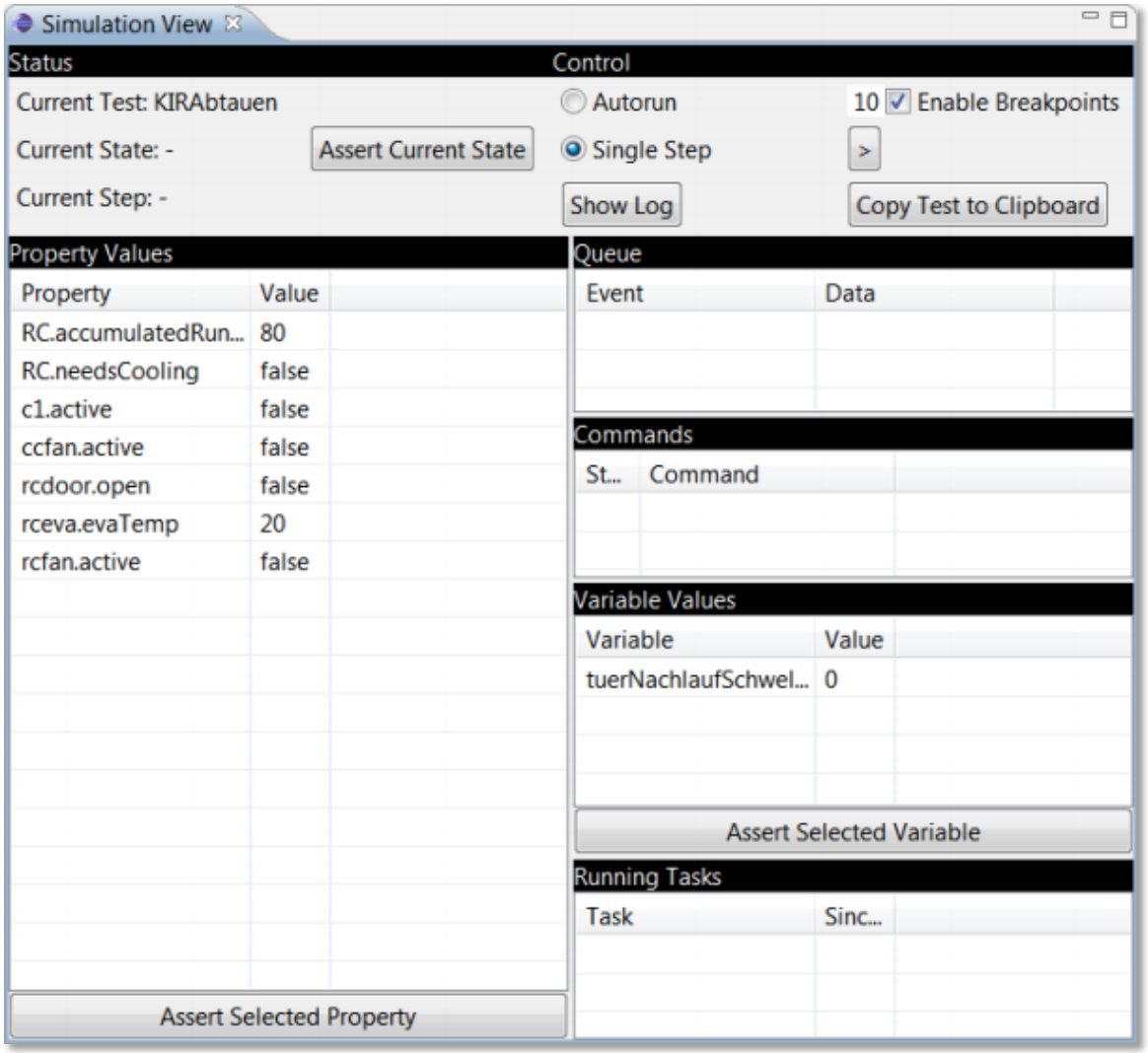
Pension Plans

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

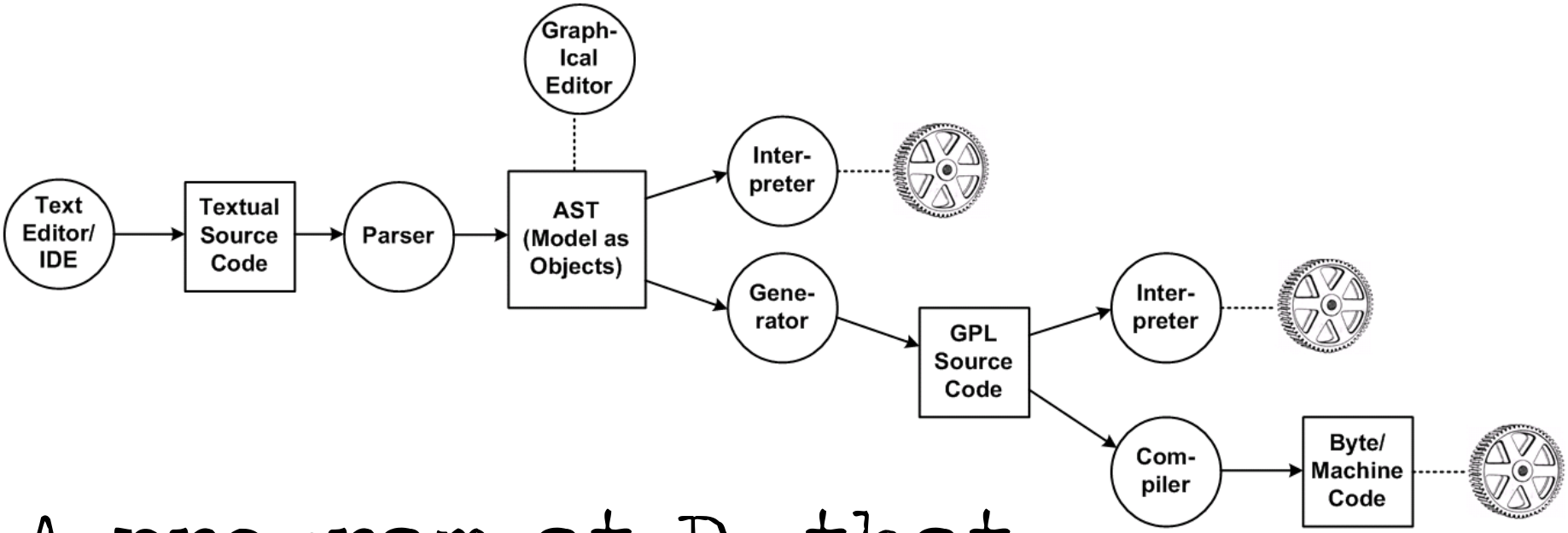
Refrigerators



Example

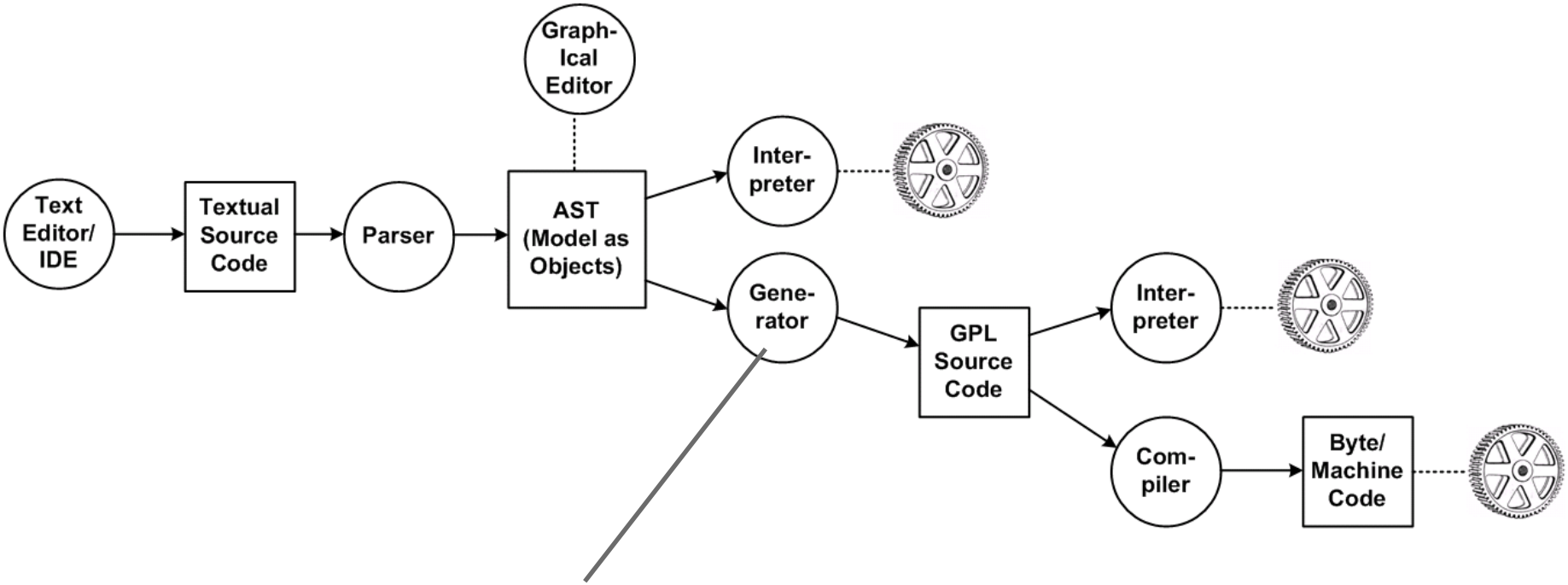
Refrigerators

Interpretation



A program at D_0 that acts on the structure of an input program at $D_{>0}$

Interpretation



An interpreter :-)

Transformation

+ Code Inspection
+ Debugging
+ Performance & Optimization
+ Platform Conformance

Interpretation

+ Turnaround Time
+ Runtime Change

Def: Semantics

… via mapping to lower level

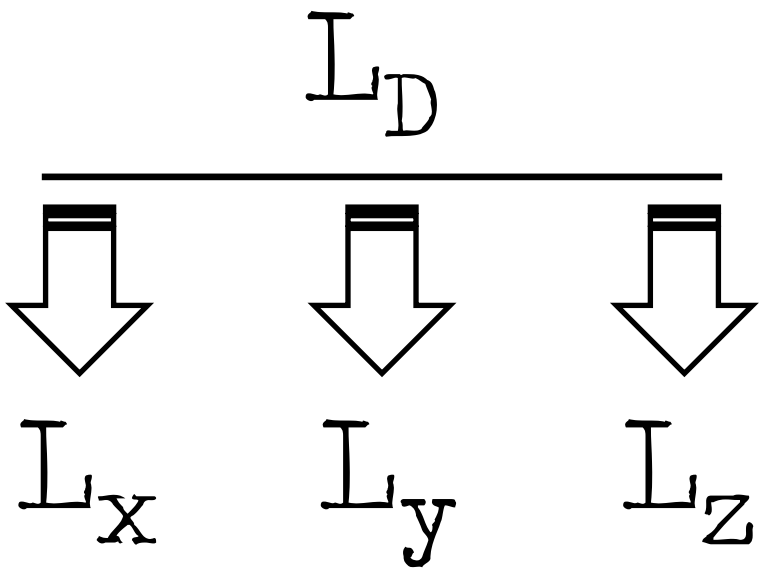
L_D

$\Downarrow$ Transformation
Interpretation

L_{D-1}

Multiple Mappings

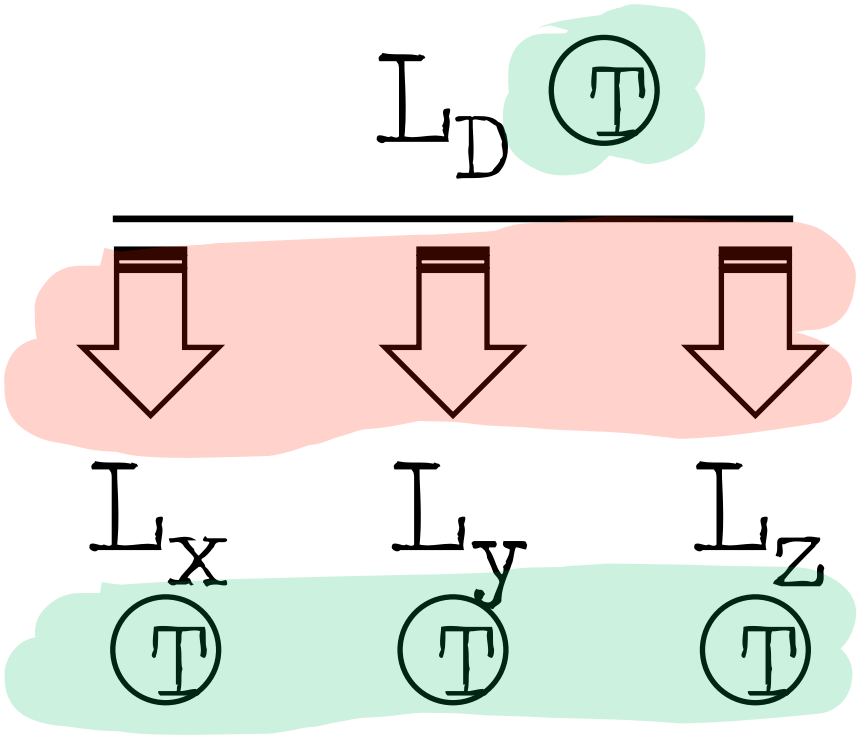
… at the same time



Similar Semantics?

Multiple Mappings

… at the same time



Similar Semantics?

all green!

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	

Example

Pension Plans

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

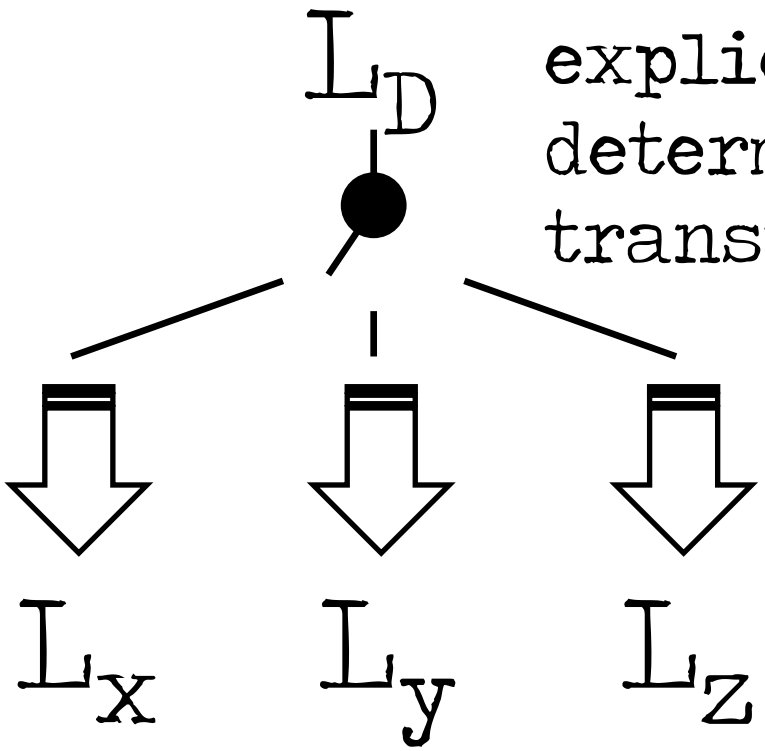
```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

Refrigerators

Multiple Mappings

… alternatively, selectably

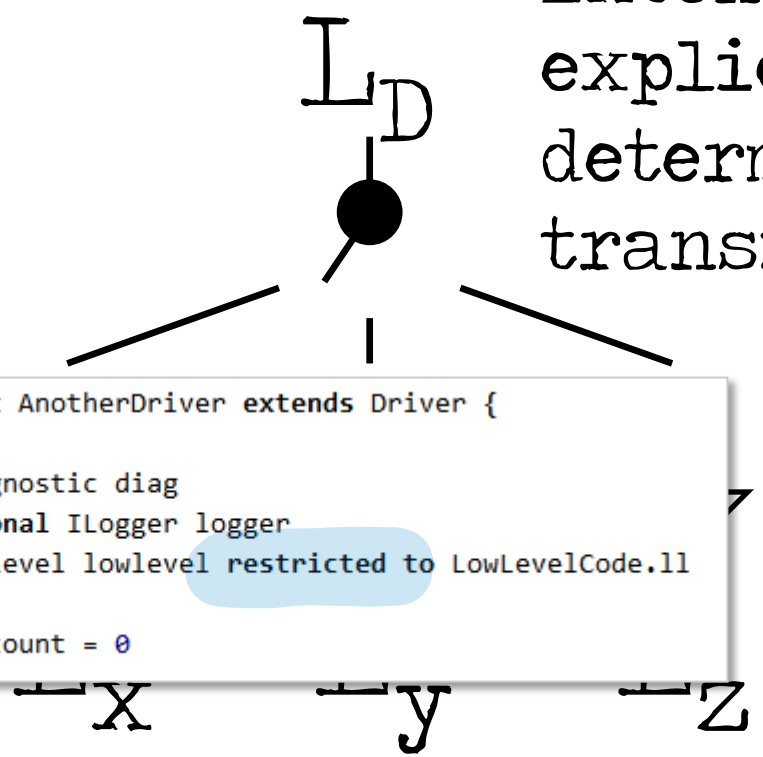


Extend L_D to include explicit data that determines transformation

Multiple Mappings

… alternatively, selectably

Extend L_D to include explicit data that determines transformation




```
exported component AnotherDriver extends Driver {
  ports:
    provides IDiagnostic diag
    requires optional ILogger logger
    requires ILowLevel lowlevel restricted to LowLevelCode.ll
  contents:
    field int8_t count = 0
```

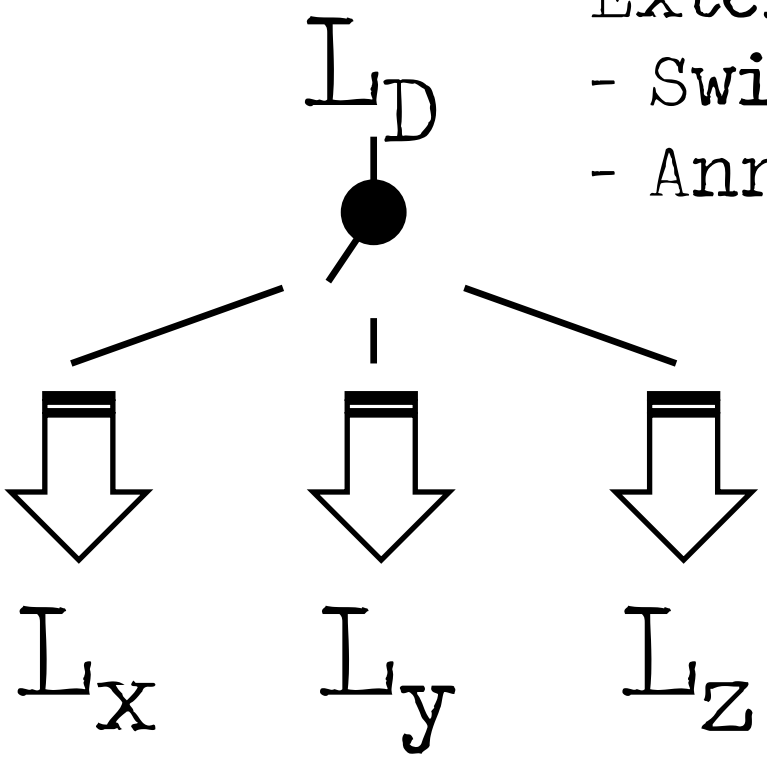
Restricted Port leads to reduced overhead C

Example

Extended C

Multiple Mappings

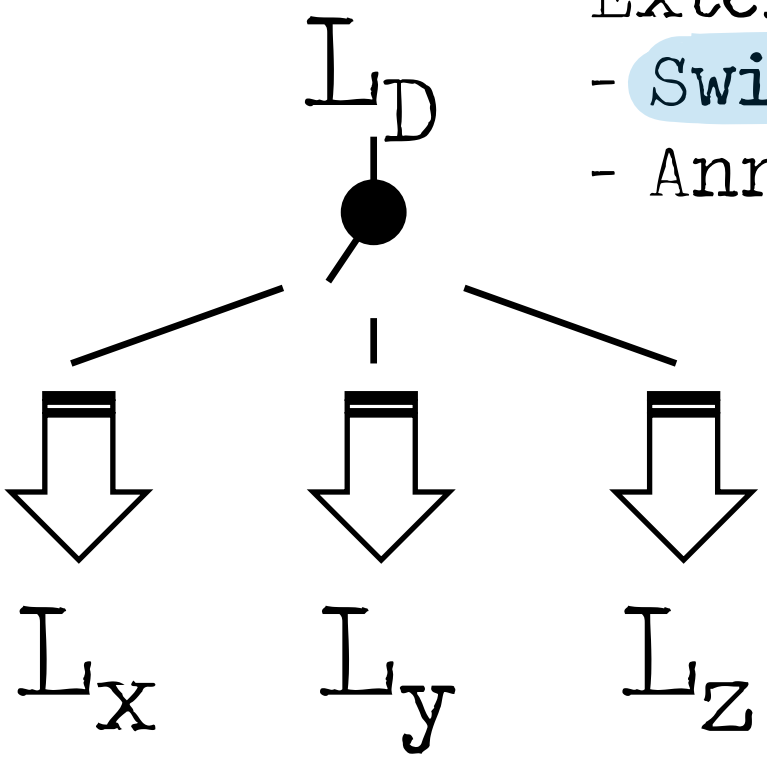
… alternatively, selectably



External Data:
- Switches
- Annotation Model

Multiple Mappings

… alternatively, selectably



External Data:
- Switches
- Annotation Model

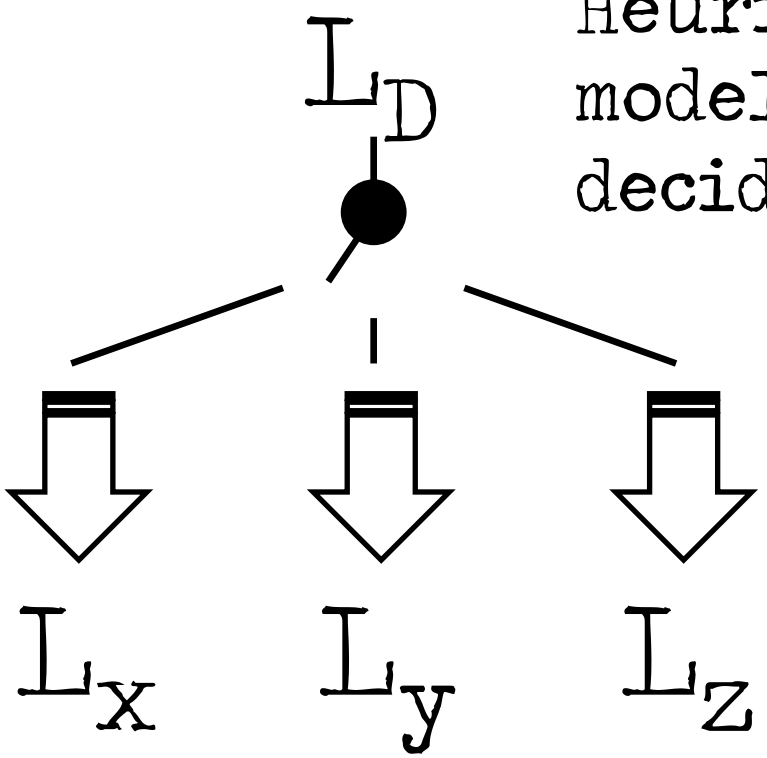
Switch Control
Java vs. C

Example

Pension Plans

Multiple Mappings

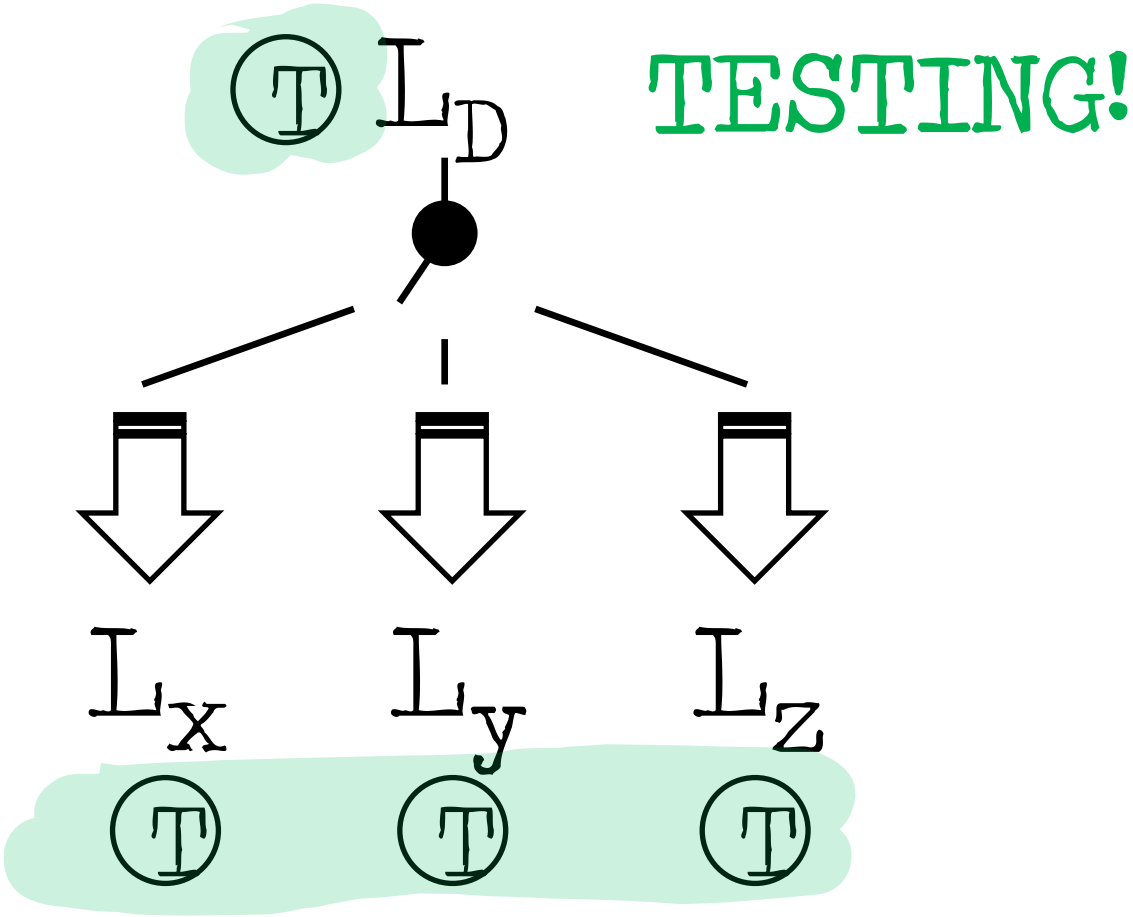
… alternatively, selectably



Heuristics: Analyze model to try to decide

Multiple Mappings

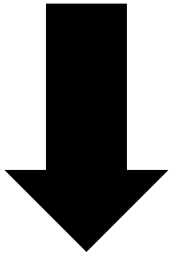
… alternatively, selectably



Reduced Expressiveness

bad? maybe.

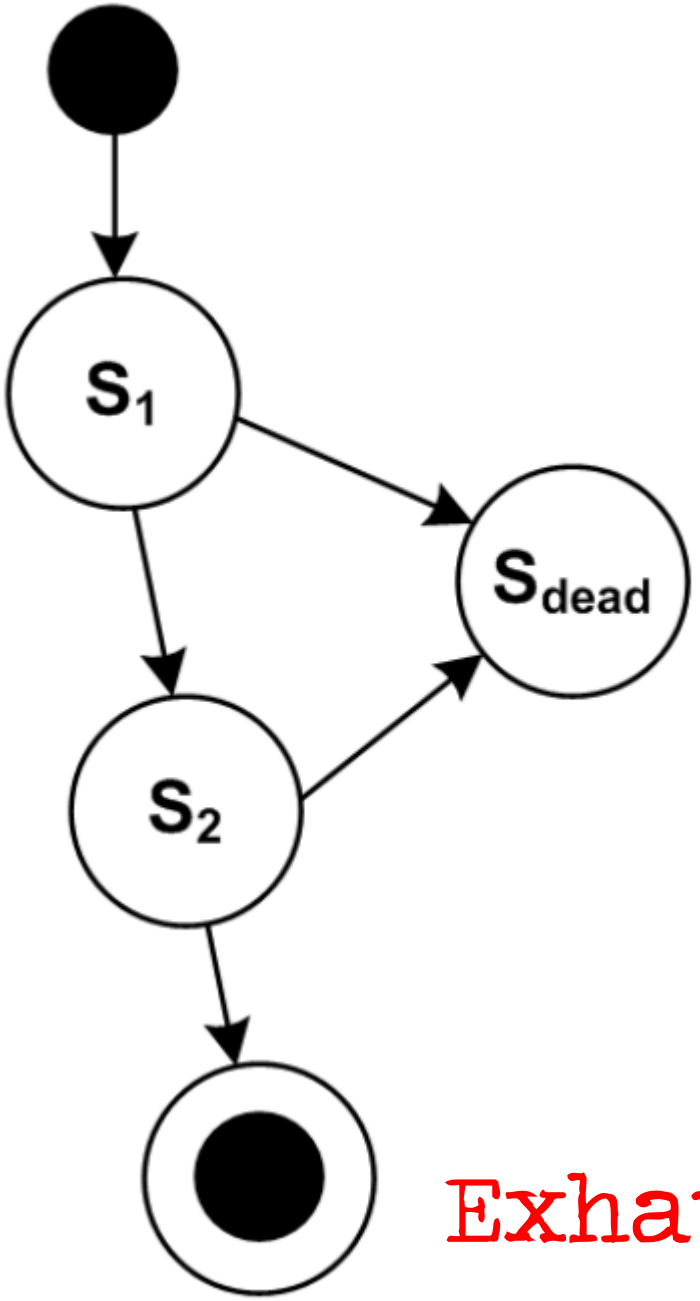
good? maybe!



Model Checking

SAT Solving

Exhaustive Search, Proof!



Unique State Names

Unreachable States

Dead End States

Guard Decidability

Reachability

Exhaustive Search, Proof!

core.dev - [L:\lwes-assembla\mbeddr\code\languages\com.mbeddr.core] - semaphore\Semaphore - JetBrains MPS 2.0.2

File Edit Search View Go To Code Build Run Tools Version Control Window Help

TypeSizeConfiguration | Semaphore

```
requirements modules: bla
module Semaphore from semaphore imports nothing {

  verifiable statemachine statemachine {
    in request(boolean req) <no binding>
       step(int[-10..10] stepCount) <no binding>
    out out(int[0..2] traffic, boolean pedestrian, boolean indicator) => handleOutE
    vars int[-1..10] counter = 0
         int[0..5] green_val = 2
         int[0..5] yellow_val = 2
         int[0..5] red_val = 4
    states (initial = Init)
      always reachable state Init {
        on step [counter == 0] -> green {
          send out(2, false, true);
          counter = 5;
        }
        on step [counter > -1 && counter < 1] -> green {
          send out(2, false, true);
          counter = 5;
        }
      }
      always reachable state green {
        on request [counter == -1] -> green {
          send out(0, false, true);
          counter = green_val;
        }
        on step [counter > 0] -> green {
          send out(0, false, true);
          counter = counter - 1;
        }
```

NuSMV Tool

Property	Status	Trace Size
State 'Init' can be...	SUCCESS	
State 'green' can ...	SUCCESS	
State 'Yellow' can...	SUCCESS	
State 'Red' can b...	SUCCESS	
State 'RedYellow' ...	SUCCESS	
State 'Init' is not li...	FAIL	3
State 'green' is live	SUCCESS	
Variable 'counter'...	SUCCESS	
Variable 'green_v...	SUCCESS	
Variable 'yellow_v...	SUCCESS	
Variable 'red_val' ...	SUCCESS	
State 'Init' contain...	FAIL	3
State 'green' has ...	SUCCESS	
State 'Yellow' has...	SUCCESS	
State 'Red' has d...	SUCCESS	
State 'RedYellow' ...	SUCCESS	
Transition 0 of st...	SUCCESS	
Transition 1 of st...	FAIL	1
Transition 0 of st...	FAIL	1

Node	Value
State Init	
counter	0
green_val	2
yellow_val	2
red_val	4
State Init	
in_event: step	step(-10)
counter	0
green_val	2
yellow_val	2
red_val	4
State green	
out_event:out	
counter	
green_val	
yellow_val	
red_val	

Version Control | 0: Messages | 9: Changes

Example

Exten
ded C

```
c/s interface Decider {
  int decide(int x, int y) pre
}

component AComp extends nothing {
  ports:
    provides Decider decider
  contents:
    int decide(int x, int y) <- op decider.decide {
      return int, 0
```

	x == 0	x > 0
y == 0	0	1
y > 0	1	2

```
    }
}
```

Example

Extended C

Separation of Concerns

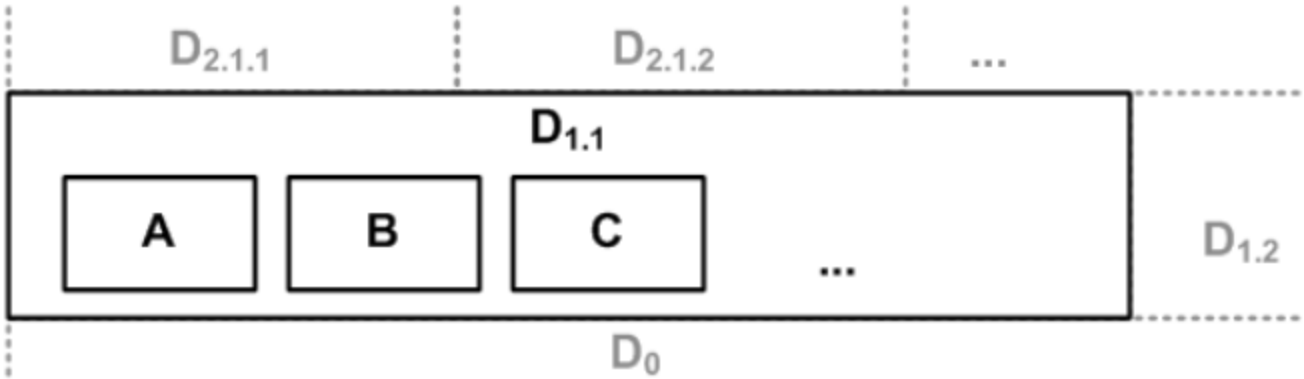
expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Several Concerns

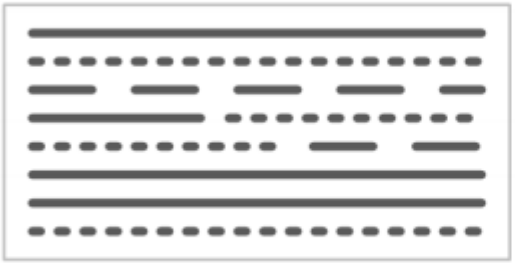
… in one domain



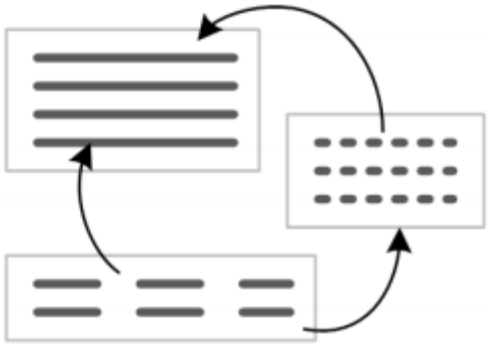
Several Concerns

… in one domain

integrated into
one fragment



separated into
several fragments

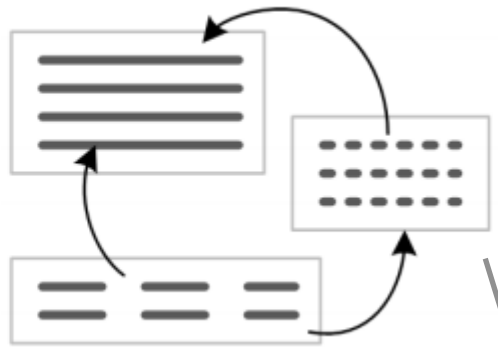


Viewpoints

$$\forall r \in Refs_f \mid fo(r.to) = fo(r.from) = f$$

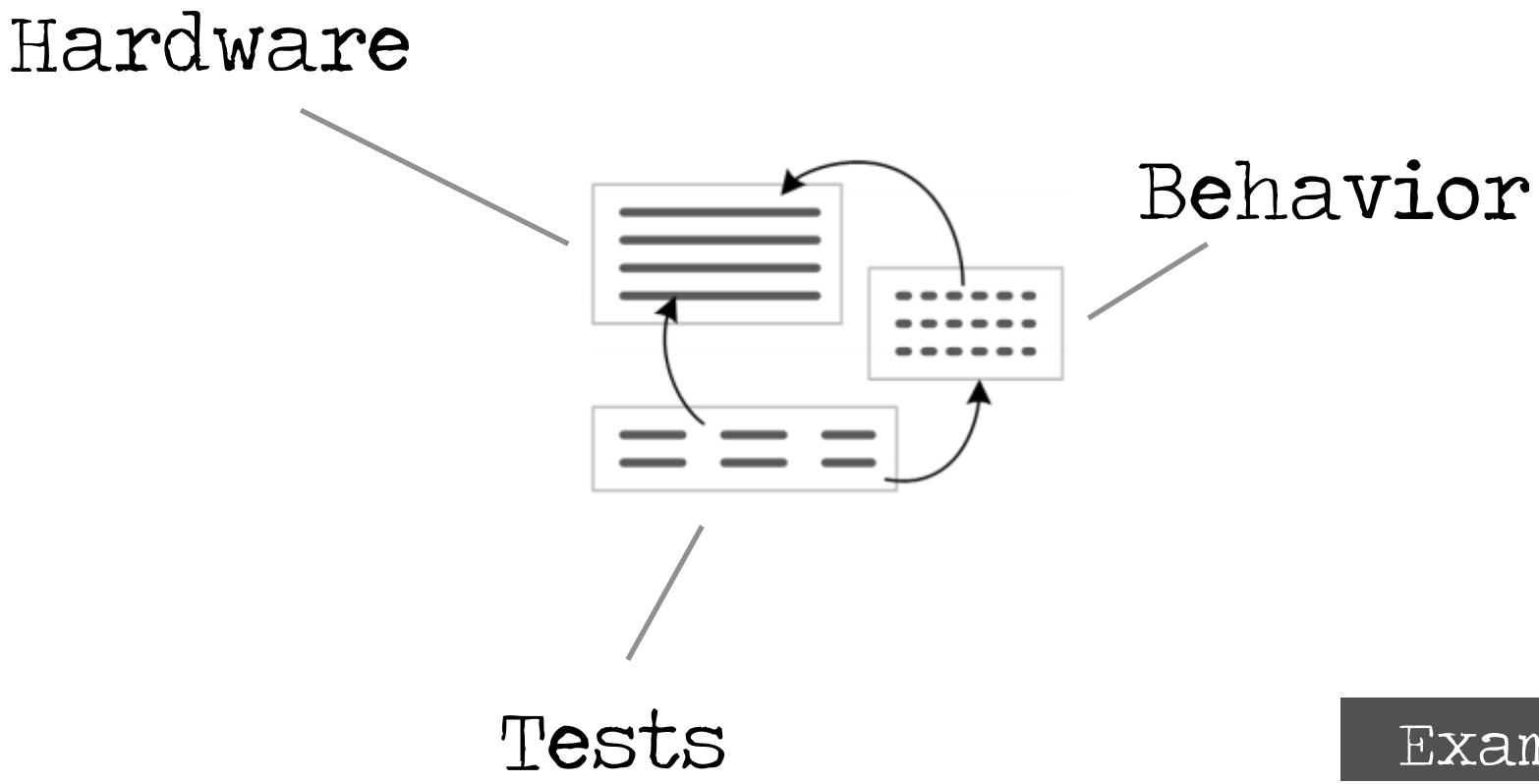
independent

$$\forall e \in E_f \mid lo(co(e)) = l$$



dependent

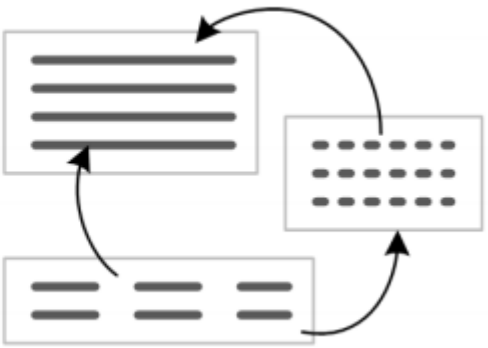
Viewpoints



Example

Refrige
rators

Viewpoints: Why?



Sufficiency

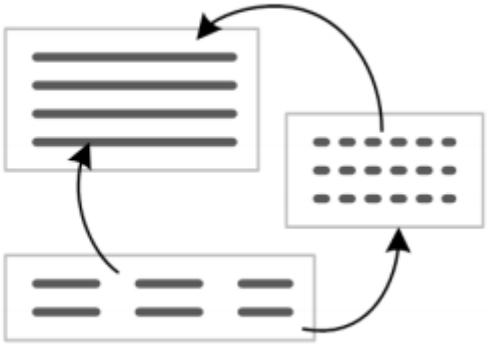
Different Stakeholders

Different Steps in Process - VCS unit!

Viewpoints

Hardware

Product Management



Behaviour

Thermo-dynamics-Experts

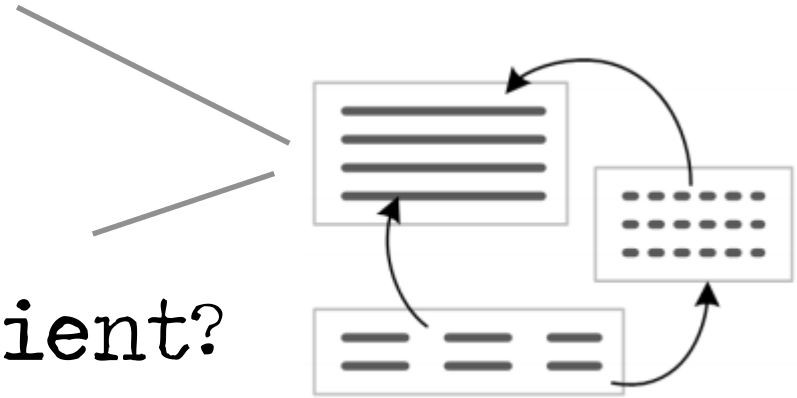
Tests

Example

Refrige rators

Viewpoints

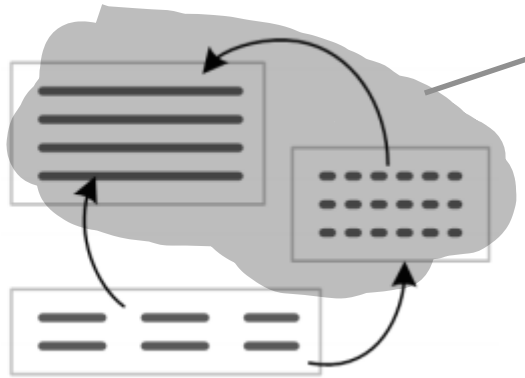
independent



sufficient?

contains all the data for running a meaningful transformation

Viewpoints



sufficient

implementation code

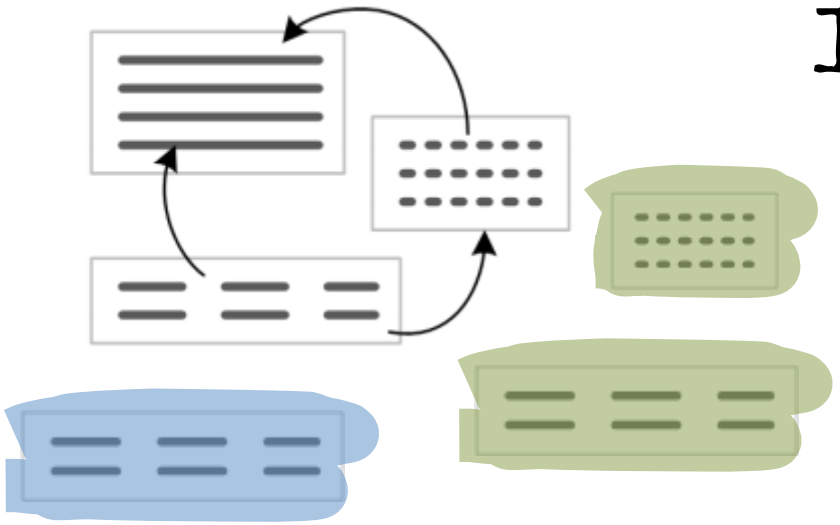
sufficient

Hardware structure documentation

Example

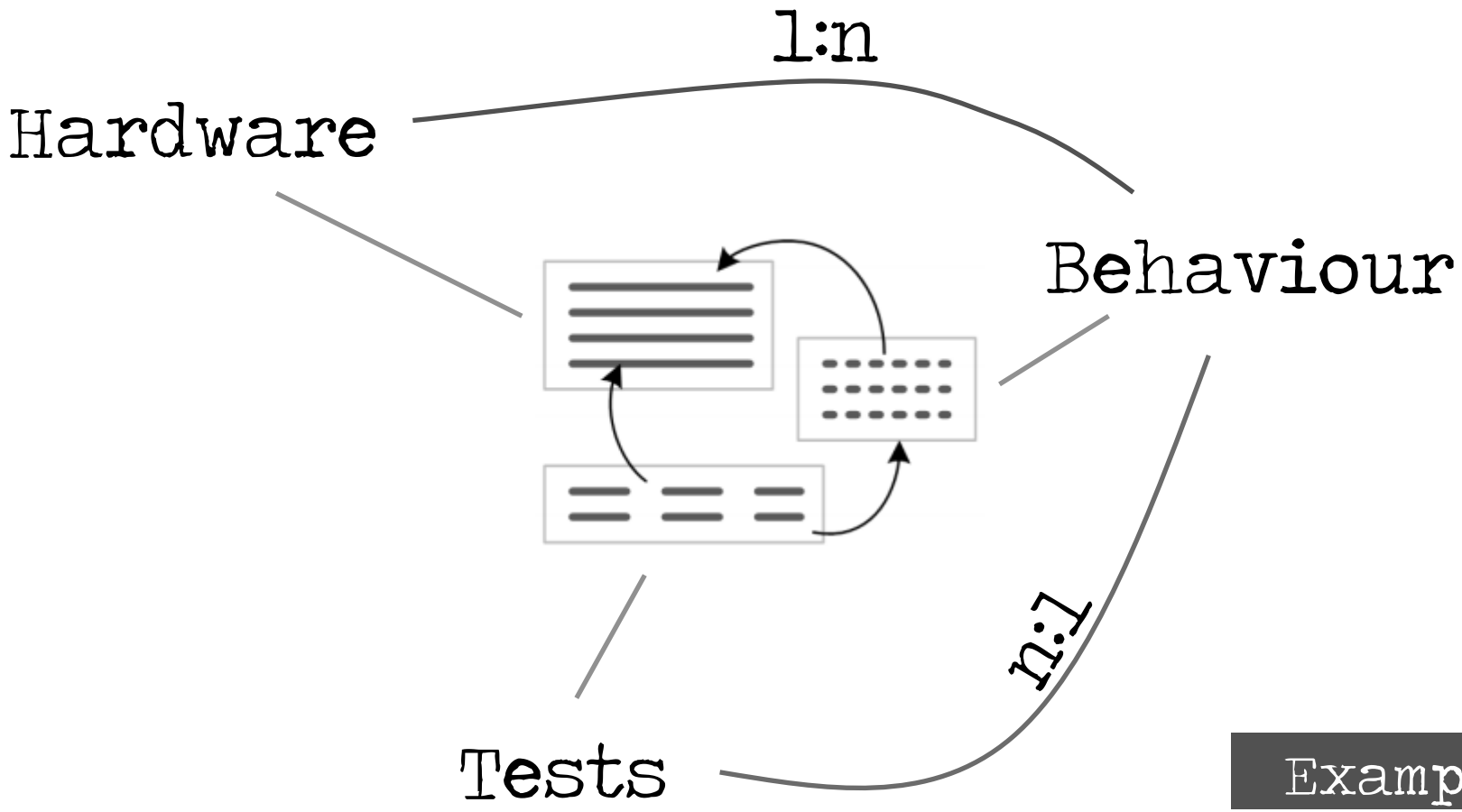
Refrigerators

Viewpoints: Why?



1:n Relationships

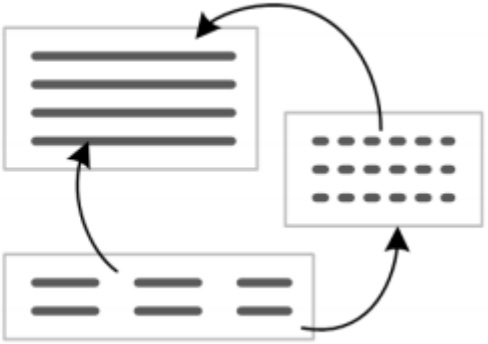
Viewpoints



Example

Refrige
rators

Viewpoints



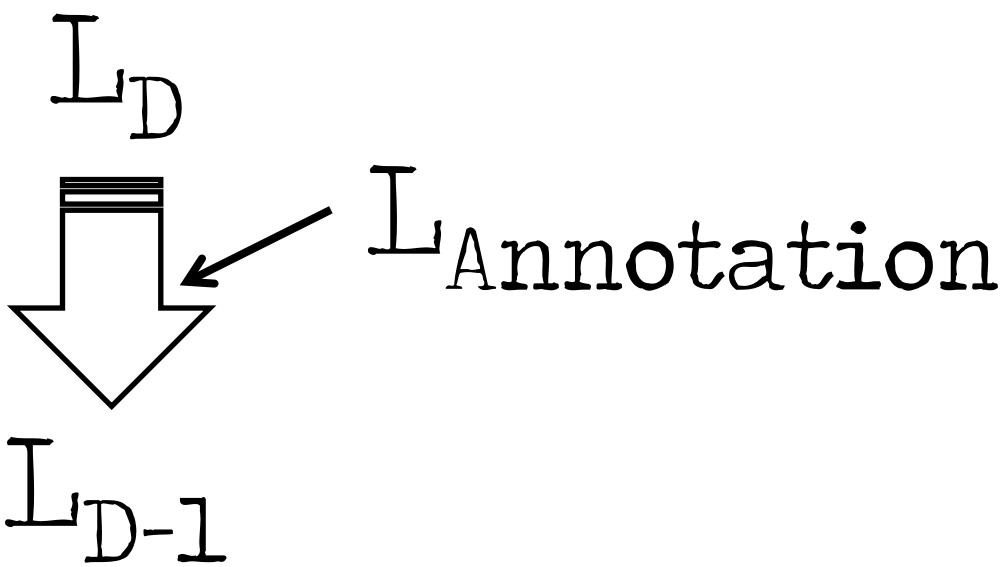
Well-defined Dependencies

No Cycles!

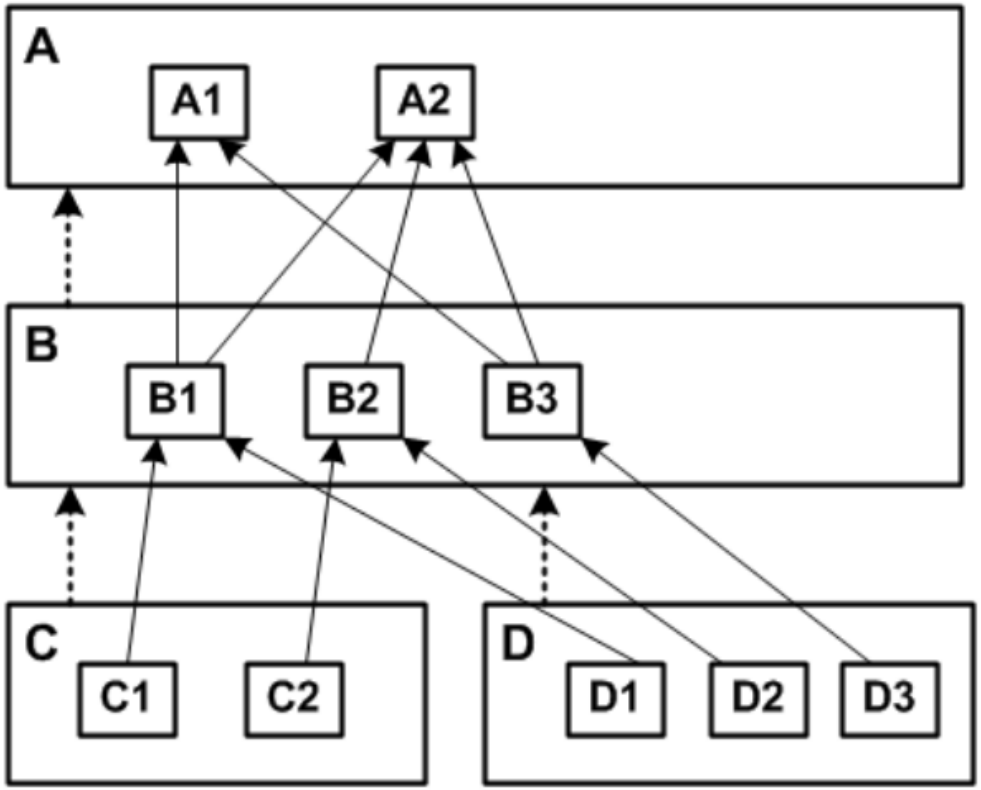
Avoid Synchronization!

(unless you use a projectional editor)

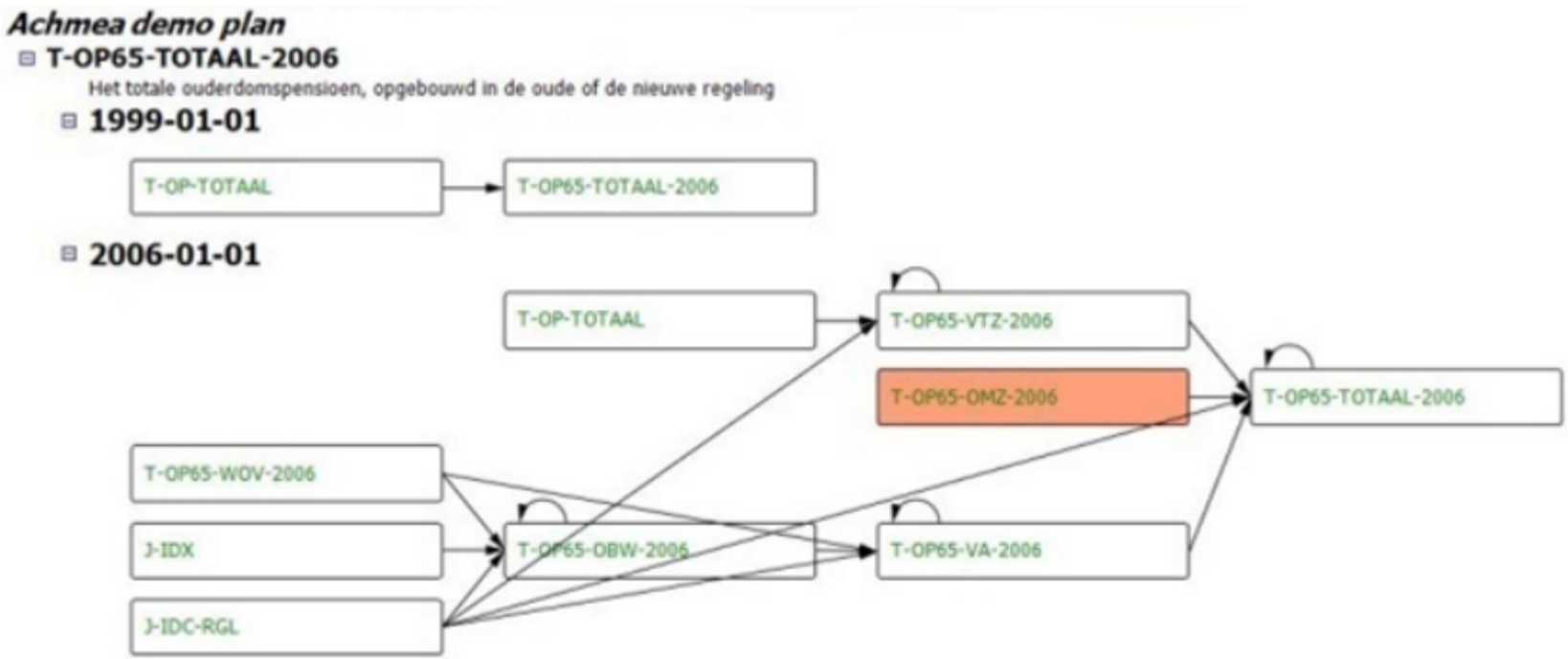
Viewpoints: Why?



Progressive Refinement



Views on Programs



Example

Pension Plans

Completeness

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Can you generate 100% of the code from the DSL program?

More generally: all of D_{-1}

Semantics:

$$semantics(p_{L_D}) := q_{L_{D-1}}$$

$$where\ OB(p_{L_D}) == OB(q_{L_{D-1}})$$

Introduce G ("generator"):

$$OB(p) == OB(G(p)) == OB(q)$$ complete

$$OB(G(p)) \subset OB(p)$$ incomplete

Incomplete: What to do?

$$F_D$$

$$\Downarrow$$

$$OB(F_D) \mathrel{!=} F_{D-1}$$

Incomplete: What to do?

Manually written code!

$$F_D$$

$$\Downarrow$$

$$OB(F_D) == F_{D-1} + F_{D-1,\ man}$$

Manually written code?

Call "black box" code
(foreign functions)

Embed L_{D-1} code in L_D program

Manually written code?

Call "black box" code
(foreign functions)

Embed L_{D-1} code in L_D program

Embed C statements
in components, state machines
or decision tables

Example

Refrige
rators

Manually written code?

Call "black box" code
(foreign functions)

Embed L_{D-1} code in L_D program

Use composition mechanisms of
L_{D-1} (inheritance, patterns, aspects, ...)

Manually written code?

Call "black box" code (foreign functions)

Embed L_{D-1} code in L_D program

Use composition mechanisms of L_{D-1} (inheritance, patterns, aspects, ...)

Generate base classes with abstract methods; implement in subclass

Example

Compo nents

Manually written code?

Call "black box" code (foreign functions)

Embed L_{D-1} code in L_D program

Use composition mechanisms of L_{D-1} (inheritance, patterns, aspects, ...)

Use protected regions (if you really have to...)

Manually written code?

Call "black box" code (foreign functions)

Embed L_{D-1} code in L_D program

Use composition mechanisms of L_{D-1} (inheritance, patterns, aspects, ...)

Use protected regions (if you really have to...) DON'T!

Incomplete: When?

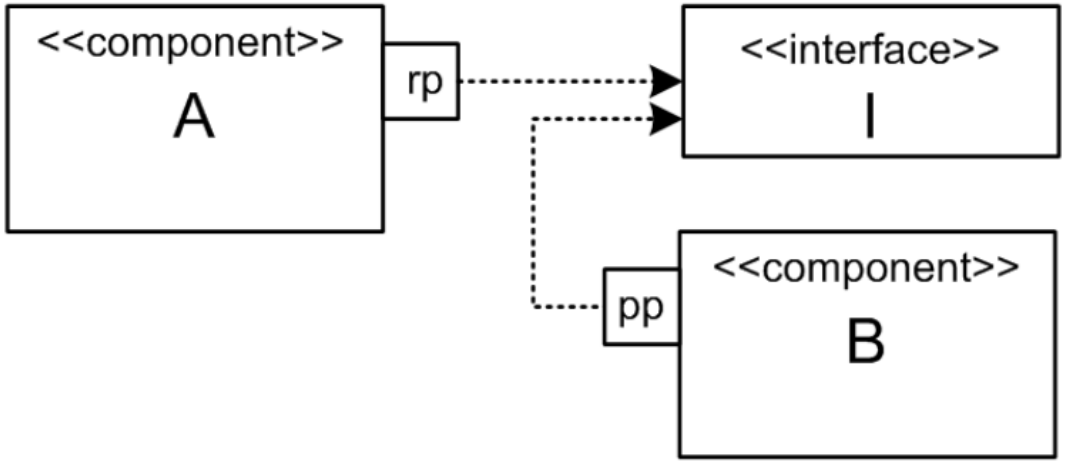
Good for technical DSLs: Devs write L_{D-1} code

Example	Example
Extended C	Components

Bad for business DSLs. Maybe use a L_{D-1} std lib that L_D code can call into?

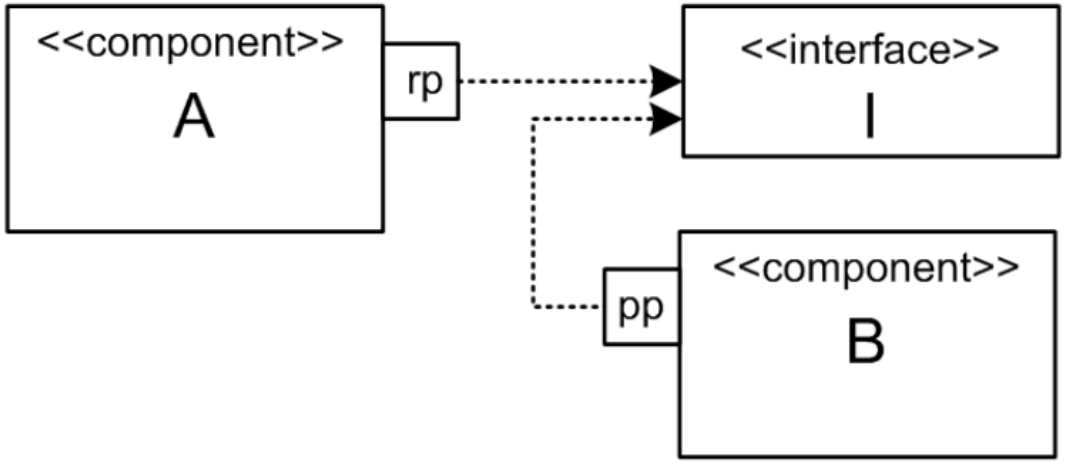
Example	Example	Example
Refrigerators	Pension Plans	Web DSL

Prevent Breaking Promises!




```
class B extens BBase {

  public void doSomething() {
    Registry.get(„A“).someMethod();
  }

}
```

Prevent Breaking Promises!



Better:

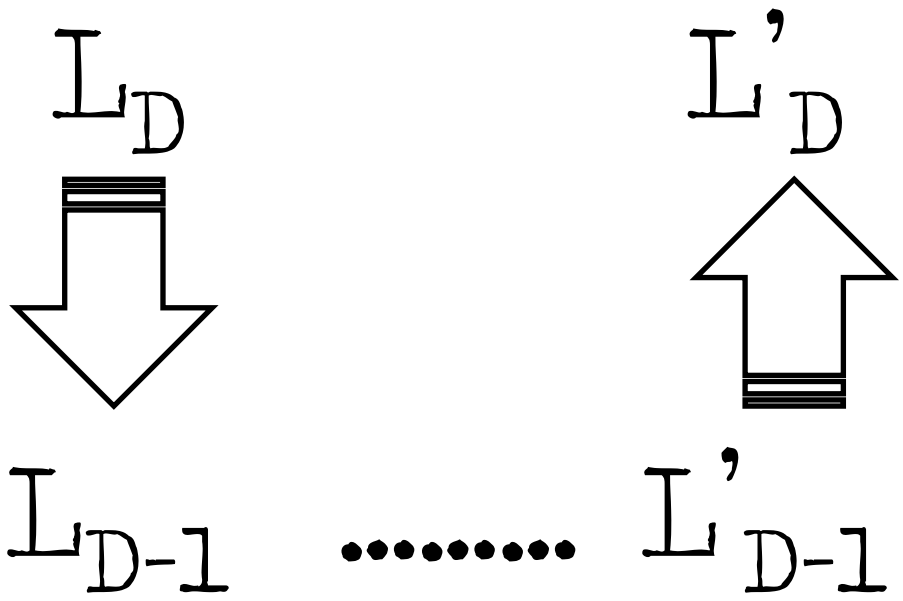
Dependency Injection

Static analysis tools

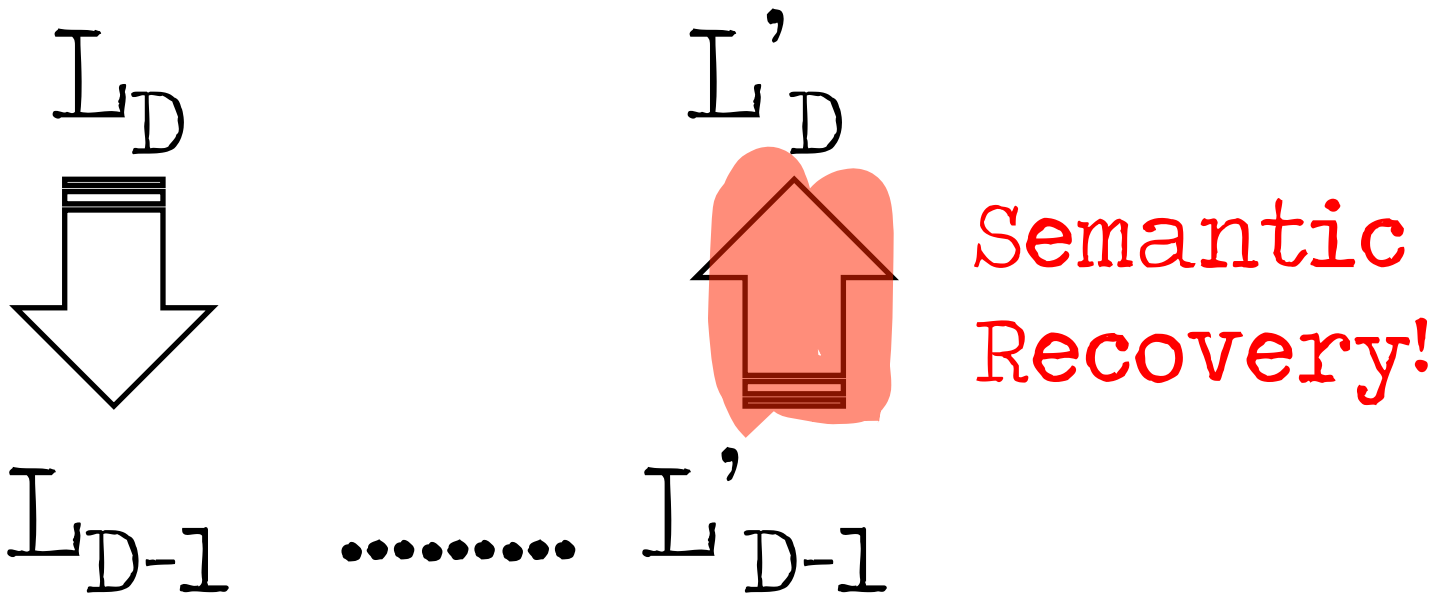
Example

Compo
nents

Roundtripping



Roundtripping - Don't!



Fundamental Paradigms

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Structure

Modularization, Visibility

Namespaces,
public/private
importing

Structure

Modularization, Visibility

- Namespaces,
- public/private
- importing
- ⇨ divide & conquer
- reuse
- stakeholder integration

Structure

Partitioning (Files)

VCS Unit
Unit of sharing
Unit of IP

!= logical modules

may influence language design

Structure

Modularization, Visibility

```
module Module1 from HPL.main imports Module2 {

  exported var int aReallyGlobalVar;

  struct aLocallyVisibleStruct {
    int x;
    int y;
  };

  exported int anExportedFunction() {
    return anImportedFunction/Module2();
  } anExportedFunction (function)
}
```

Example

Extended C

Structure

Modularization, Visibility

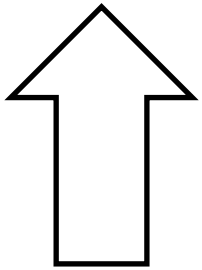
```
namespace com.mycompany.test {
  system testSystem {
    instance dc: DelayCalculator
    instance screen1: InfoScreen
    instance screen2: InfoScreen
    connect dc.screens to
      (screen1.default, screen2.default)
  }
}
```

Example

Compo
nents

Structure

Partitioning (Files)



change impact
link storage
model organization
tool integration

Structure

Spec vs. Implementation

⇨ plug in different Impls
different stakeholders

Structure

Spec vs. Impl.

```
exported c/s interface ITrafficLights {
  int8_t setColor(TLCommand cmd) ;
}

c/s interface IDriver {
  int8_t setDriverValue(int8_t addr, int8_t value) ;
}

c/s interface IDiagnostic {
  int8_t getCount() ;
}

c/s interface ILogger {
  void log(string message) ;
}

c/s interface ILowLevel {
  int8_t doSomeLowlevelStuff(int8_t y) ;
}
```

```
exported component AnotherDriver extends Driver {
  ports:
    provides IDiagnostic diag
    requires optional ILogger logger
    requires ILowLevel lowlevel restricted to LowLevelCode
  contents:
    field int8_t count = 0

    override int8_t setDriverValue(int8_t addr, int8_t value) <- op cmd.setDriverValue {
      with port (logger) { logger.log("SomeMessage"); } with port
      lowlevel.doSomeLowlevelStuff(10);
      count++;
      return 1;
    }

    int8_t diag_getCount() <- op diag.getCount {
      return count;
    }
}
```

Example

Exten
ded C

Structure

Specialization

Liskov substitution P
leaving holes ("abstract")

Structure

Specialization

Liskov substitution P
leaving holes ("abstract")

⇨ variants (in space)
evolution (over time)

Structure

Specialization

Pension Plans can inherit from other plans.

Rules can be abstract;

Plans with abstract rules are abstact

Example

Pension Plans

Structure

Superposition, Aspects

- merging
- overlay
- AOP

⇨ modularize cross-cuts

Structure

Superposition, Aspects

```
component DelayCalculator {
  …
}
component AircraftModule {
  …
}
component InfoScreen {
  …
}
```

```
aspect (*) component {
   provides mon: IMonitoring
}
```

```
component DelayCalculator {
  …
  provides mon: IMonitoring
}
component AircraftModule {
  …
  provides mon: IMonitoring
}
component InfoScreen {
  …
  provides mon: IMmonitoring
}
```

Example

Compo
nents

Behavior

Not all DSLs specify behavior

Some just **declare** behavior

This section is
not for those!

Behavior

Imperative

sequence of statements
changes program state

write	understand	debug	analyze	performance
simple	simple -	simple (step)	hard	good

Behavior

Imperative

sequence of statements
changes program state

```
state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

Example

Refrigerators

Behavior

Functional

functions call other functions.
no state. No aliasing.

write	understand	debug	analyze	performance
simple -	simple	simple (tree)	good	good -

Behavior

Functional

functions call other functions.

no state. No aliasing.

Elements...

Rules

Rule Bereken Mutatieperiode

Result:

Mutatieperiode

Name:

Bereken Mutatieperiode

Documentation:

Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen.¶
De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verslagjaar.¶
Dit wordt niet afgevangen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.¶

Tags:

Basisberekening

Algorithm:

if maximum(Mutaties per datum) == 1 **then daysof**(**duration**(**valid**(Mutaties per datum))) **else** 0

Test cases:

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actua
Gelijke datums	03/01/2008		Mutatieperiode - Mutatiedatum =			3	0

Example

Pension Plans

Behavior

Functional

```
rule for ⟨years of service in service period⟩ (  ) at 01/01/1986
  for ¤ p in valid(Service Period) sum(Part time % * yearsof(duration(valid(Part time % over p)))) at end(p);
    valid(Service Period)
      Service Period
    04/01/1984 ... 01/01/1990
      sum(Part time % * yearsof(duration(valid(Part time % over p)))) at end(p)
        sum(Part time % * yearsof(duration(valid(Part time % over p))))
          Part time % * yearsof(duration(valid(Part time % over p)))
            Part time %
            yearsof(duration(valid(Part time % over p)))
              Value:
                TemporalBranch
```

Example

Pension Plans

Behavior

Functional

pure expressions are
a subset of functional
(operators hard-wired)

guards
preconditions
derived attributes

Behavior

Declarative

only facts and goals.
no control flow.
eval engine, solver (several)

write	understand	debug	analyze	performance
simple	simple -	hard	depends	often bad

Behavior

Declarative

- concurrency
- constraint programming
- solving
- logic programming

Behavior

Declarative

only facts and goals.

no control flow.

eval engine, solver (several)

```
section posts
  define page post(p: Post, title: String) {
    title{ output(p.title) }
    bloglayout(p.blog){
      placeholder view { postView(p) }
      postComments(p)
    }
  }
  define permalink(p: Post) {
    navigate post(p, p.urlTitle) { elements }
  }
```

Example

Web DSL

Behavior

Declarative

```
synchronous blockType org::eclipselabs::damos::library::base::_discrete::DiscreteDerivative

input u
output y

parameter initialCondition = 0
parameter gain = 1(s) // normalized

behavior {

        stateful func main<initialCondition, gain, fs>(u) -> y {
                check<0, 1(s), 1(1/s)>(real) -> real

                static assert u is real() :
                        error "Input value must be numeric"

                static assert initialCondition is real() :
                        error "Initial condition must be numeric"

                static assert initialCondition is real() && u is real() => unit(initialCondition) == unit(u) :
                        error "Initial condition and input value must have same unit"

                static assert gain is real() :
                        error "Gain value must be numeric"

                eq u{-1} = initialCondition
                eq y{n} = fs * gain * (u{n} - u{n-1})
        }

}
```

Example

Exten
ded C

Behavior

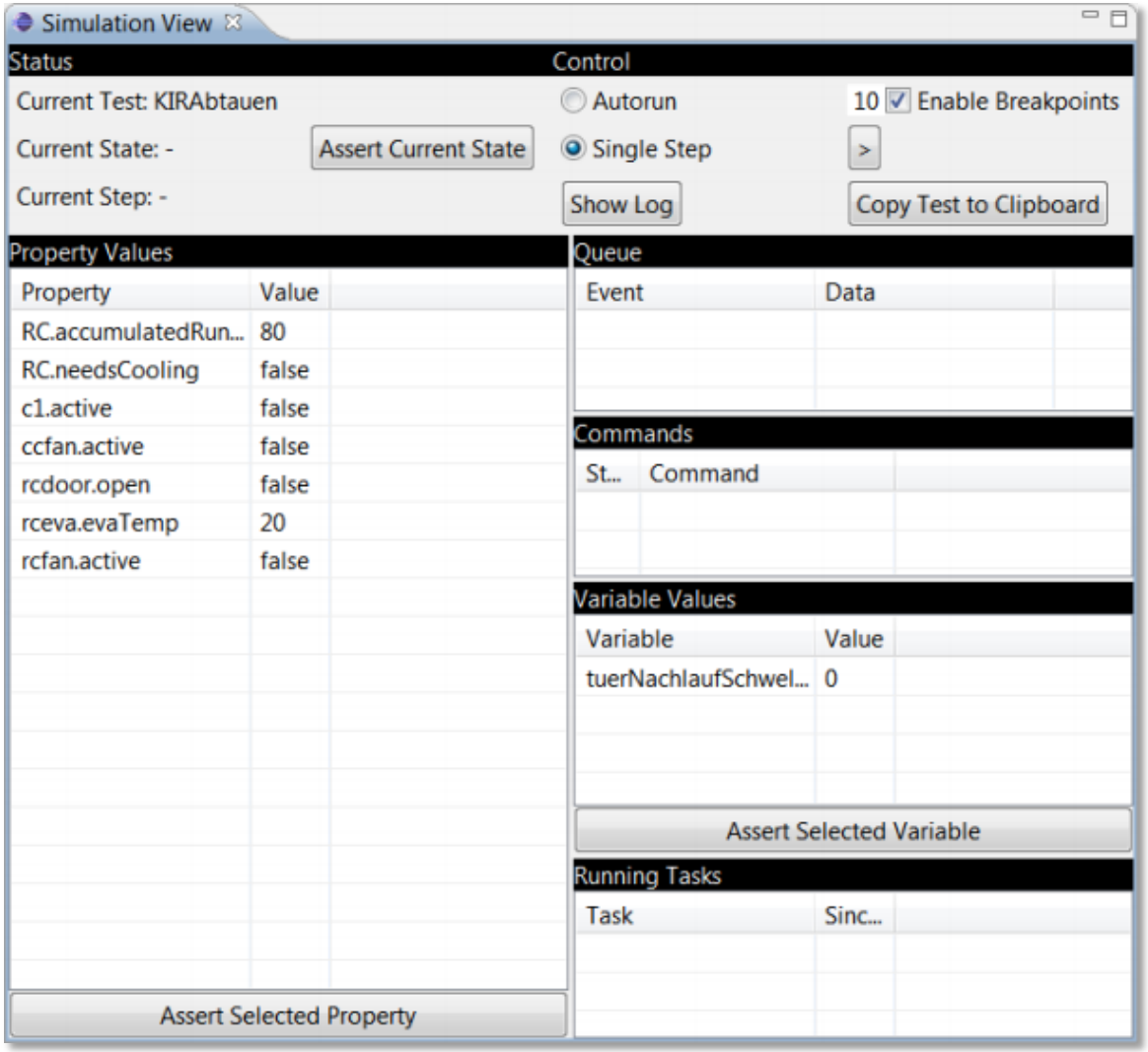
Reactive

reactions to events, more events are produced. Communication via events and channels/queues

write	understand	debug	analyze	performance
simple	simple/hard	hard	simple	can be good

Behavior

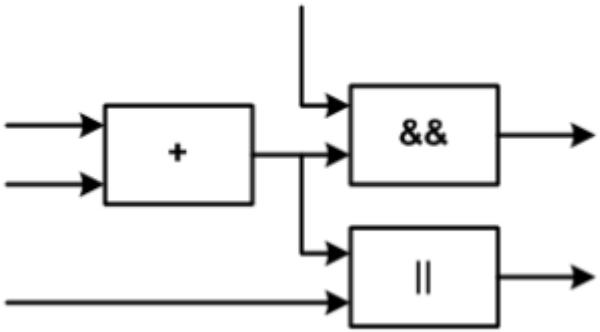
Reactive



Example

Refrige
rators

Behavior

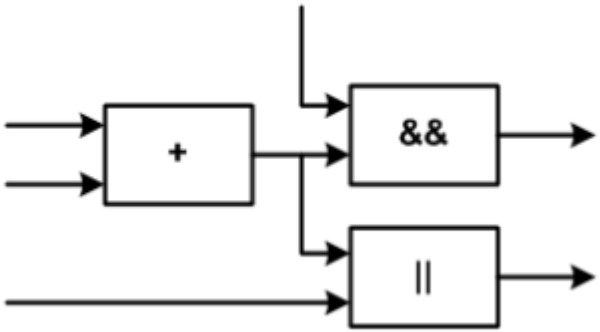


Data Flow

chained blocks consume continuous data that flows from block to block

write	understand	debug	analyze	performance
simple -	simple/hard	hard	simple	can be good

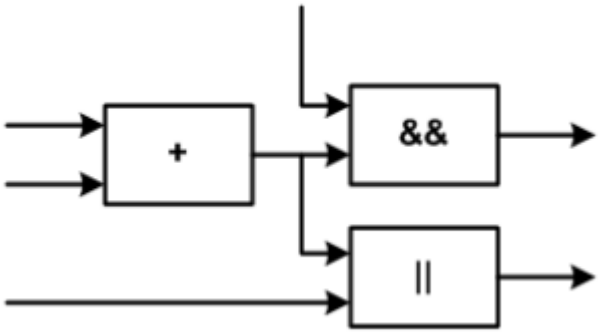
Behavior



Data Flow

continuous, calc on change
quantized, calc on new data
time triggered, calc every x

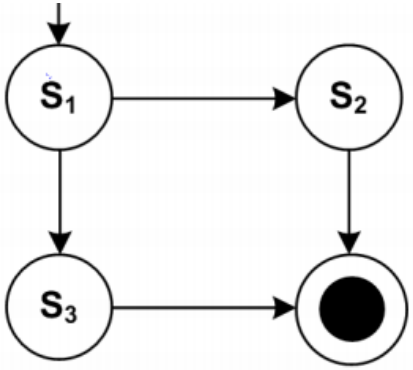
Behavior



Data Flow

Embedded Programming
Enterprise ETL & CEP

Behavior



State Based

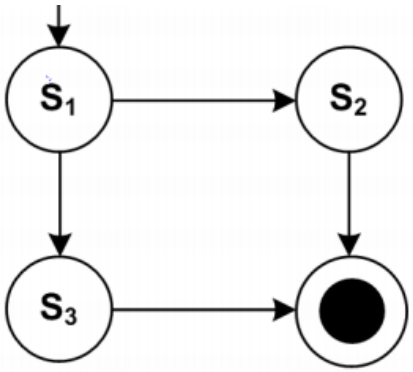
states, transitions,
guards, reactions

event driven, timed

write	understand	debug	analyze	performance
simple -	simple/hard	s/h	simple +	can be good

Behavior

State Based




```
start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

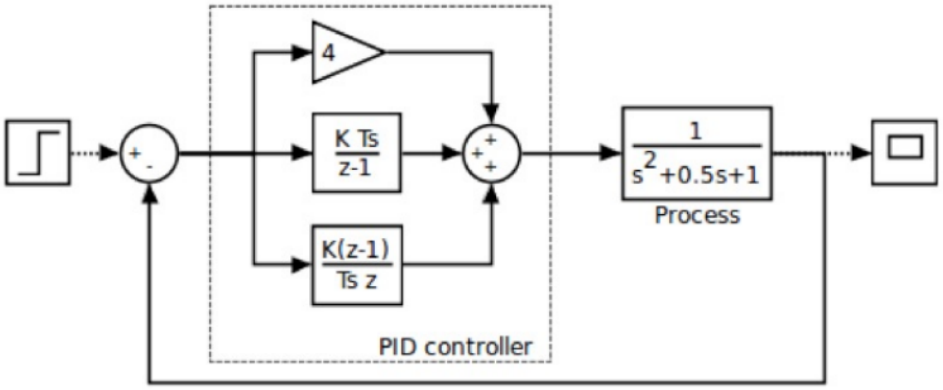
Example

Refrigerators

Behavior

Combinations

data flow uses functional, imperative or declarative lang inside block




```
synchronous blockType org::eclipselabs::damos::library::base::_discrete::DiscreteDerivative

input u
output y

parameter initialCondition = 0
parameter gain = 1(s) // normalized

behavior {

        stateful func main<initialCondition, gain, fs>(u) -> y {
                check<0, 1(s), 1(1/s)>(real) -> real

                static assert u is real() :
                        error "Input value must be numeric"

                static assert initialCondition is real() :
                        error "Initial condition must be numeric"

                static assert initialCondition is real() && u is real() => unit(initialCondition) == unit(u) :
                        error "Initial condition and input value must have same unit"

                static assert gain is real() :
                        error "Gain value must be numeric"

                eq u{-1} = initialCondition
                eq y{n} = fs * gain * (u{n} - u{n-1})
        }

}
```

Behavior

Combinations

- state machines use expressions in guards and often an imperative lang in actions

Behavior

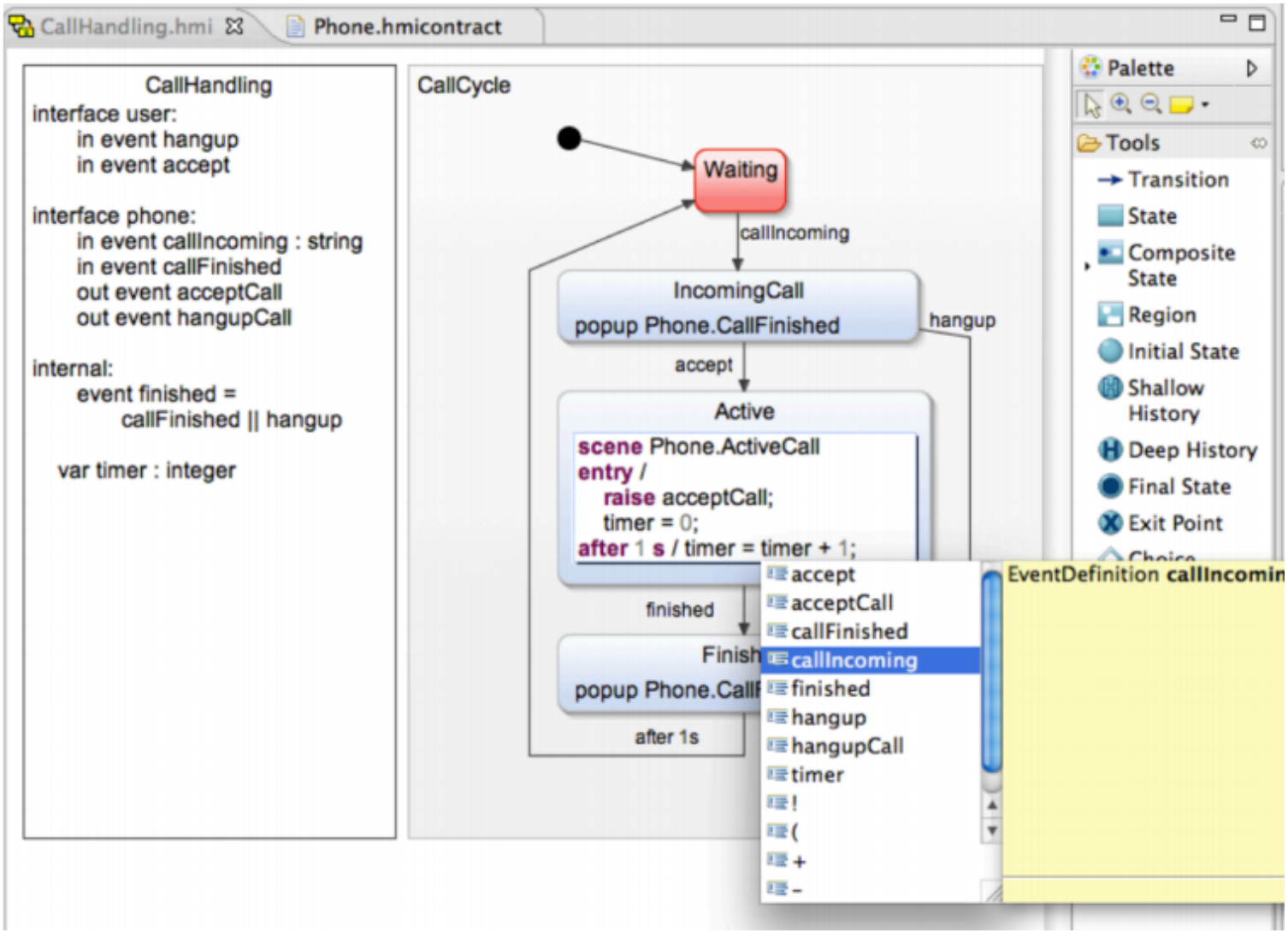
Combinations

```
start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehzeit > 333) ) {
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

Example

Refrigerators

Behavior


CallHandling.hmi
Phone.hmicontract
CallHandling
interface user:
in event hangup
in event accept
interface phone:
in event callIncoming : string
in event callFinished
out event acceptCall
out event hangupCall
internal:
event finished =
callFinished || hangup
var timer : integer
CallCycle
Waiting
callIncoming
IncomingCall
popup Phone.CallFinished
hangup
accept
Active
scene Phone.ActiveCall
entry /
raise acceptCall;
timer = 0;
after 1 s / timer = timer + 1;
finished
Finish
popup Phone.CallF
after 1s
accept
acceptCall
callFinished
callIncoming
finished
hangup
hangupCall
timer
!
(
+
-
EventDefinition callIncomin
Palette
Tools
Transition
State
Composite State
Region
Initial State
Shallow History
Deep History
Final State
Exit Point

Behavior

Combinations

purely structural languages often use expressions to specify constraints

```
c/s interface IDriver {
  int setDriverValue(int addr, int value)
    pre value > 0
}
```

Example

Exten
ded C

Language Modularity

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency
of DSL development

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency of DSL development

4 ways of composition:

Referencing

Reuse

Extension

Reuse

Language Modularity, Composition and Reuse (LMR&C)

increase efficiency of DSL development

4 **ways of composition:**

distinguished regarding **dependencies** and **fragment structure**

Dependencies:

do we have to know about the reuse when designing the languages?

Dependencies:

do we have to know about the reuse when designing the languages?

Fragment Structure:

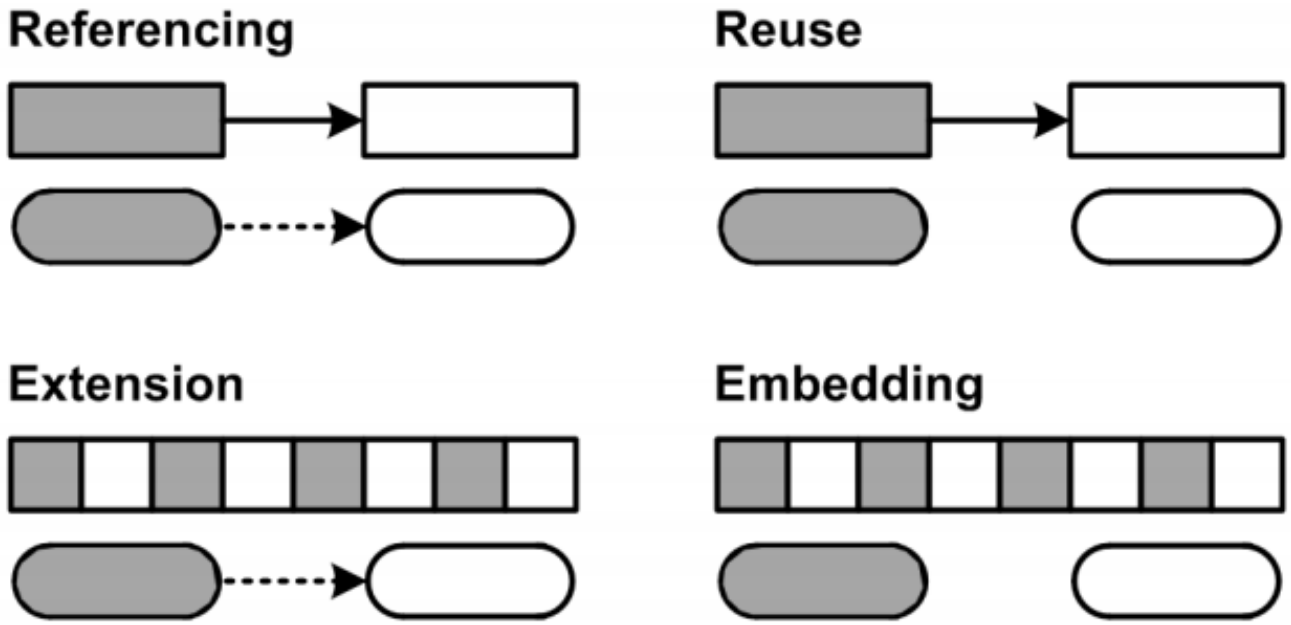
homogeneous vs. heterogeneous („mixing languages")

Dependencies & Fragment Structure:

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure

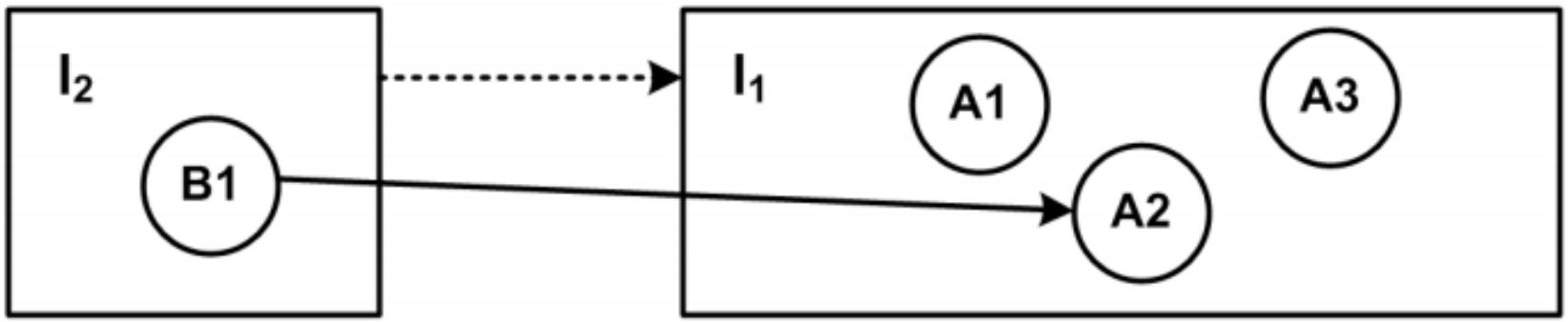
Dependencies & Fragment Structure:



Referencing

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure

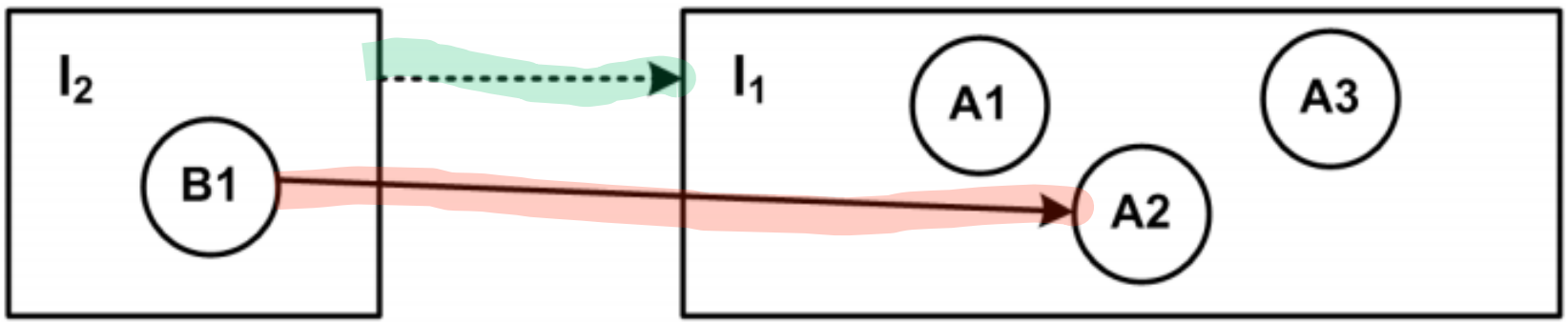


Referencing

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

Dependent

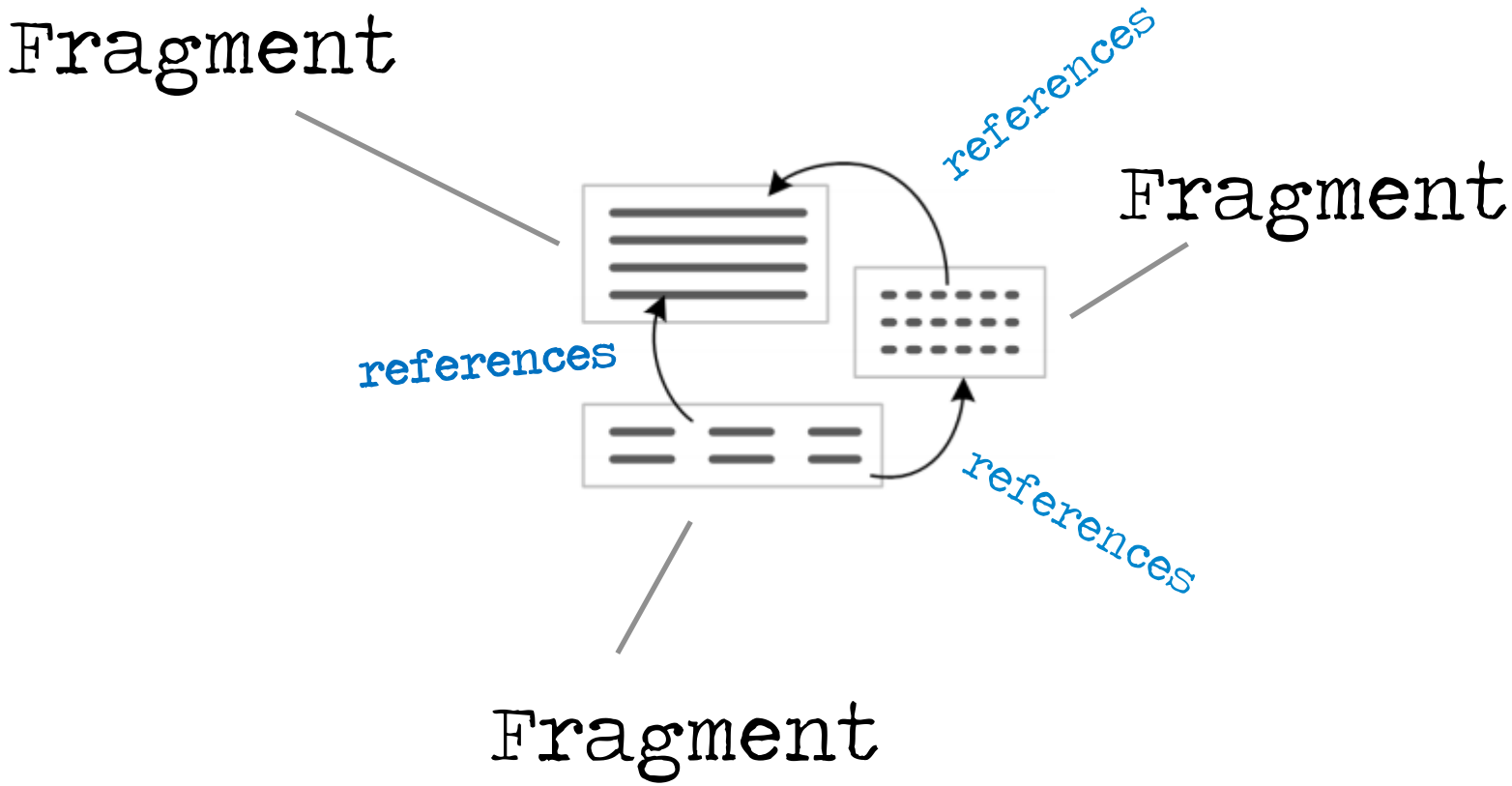


No containment

Referencing

Used in
Viewpoints

Referencing



Referencing

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Example

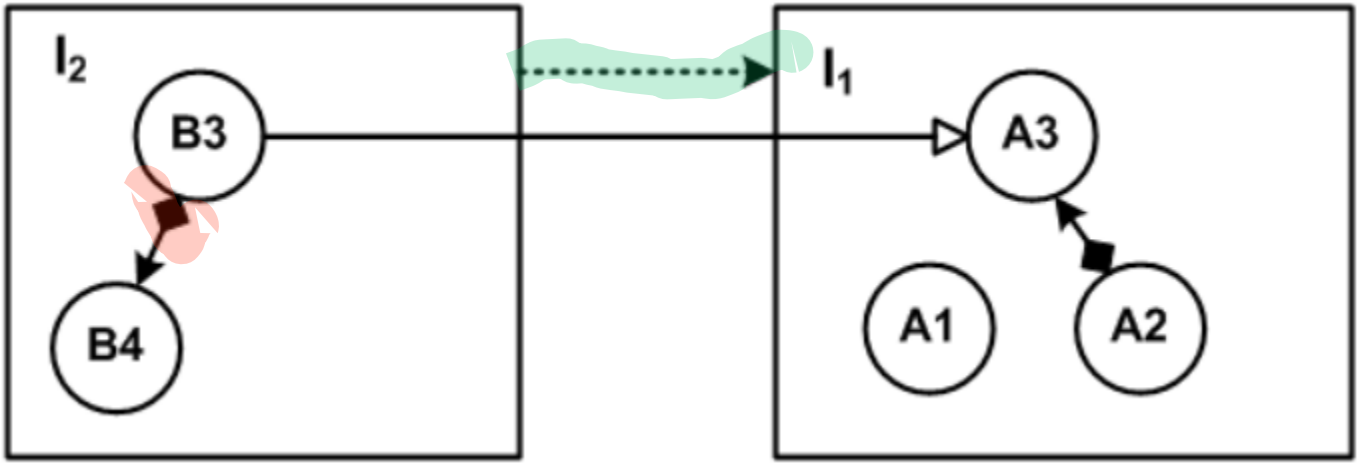
Refrigerators

Extension

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

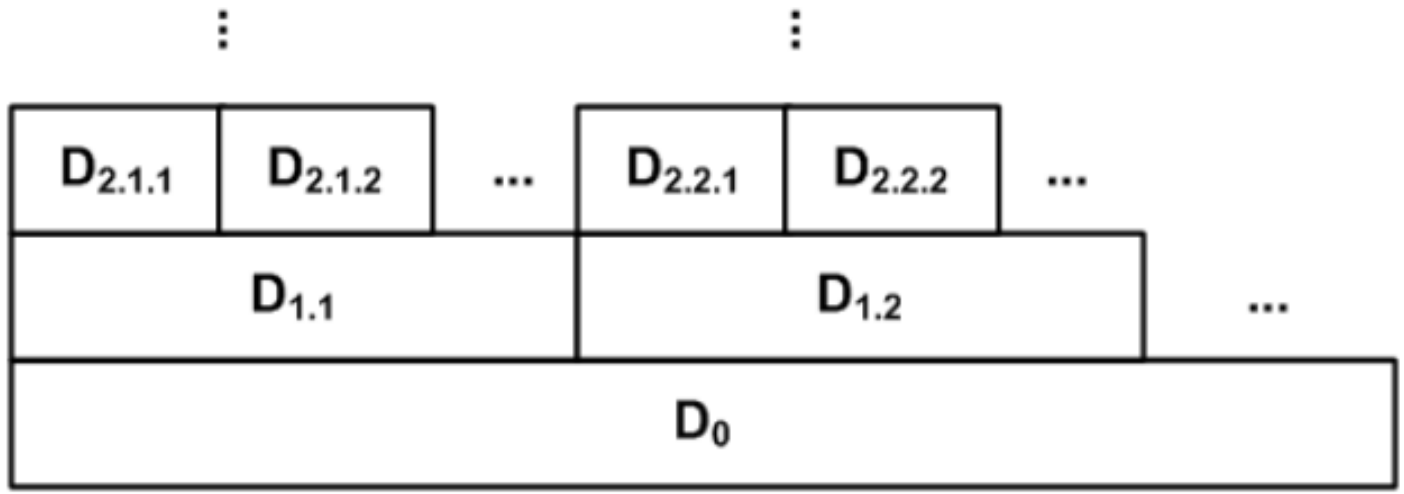
fragment structure

Dependent



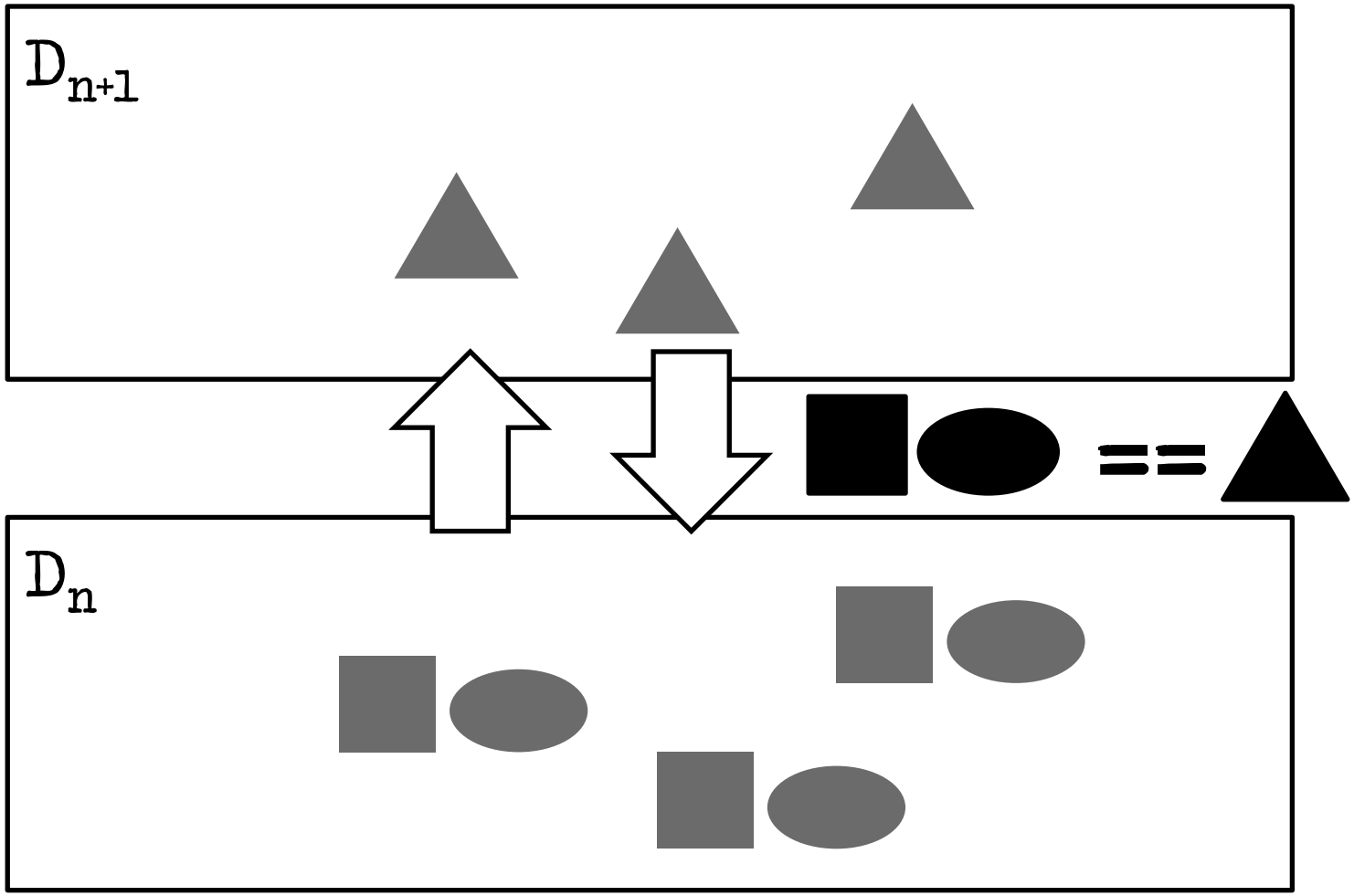
Containment

Extension

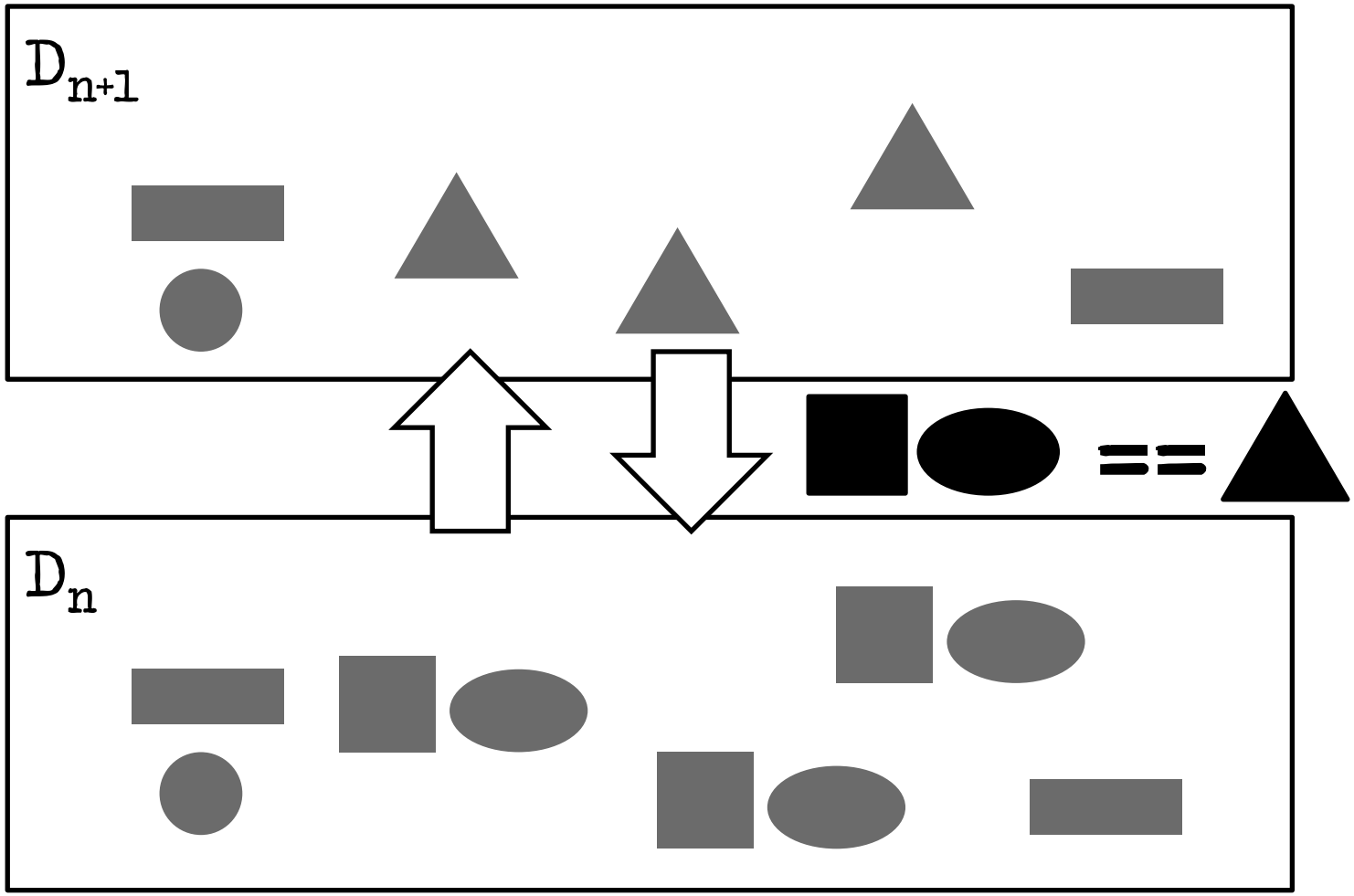


more specialized domains
more specialized languages

Extension

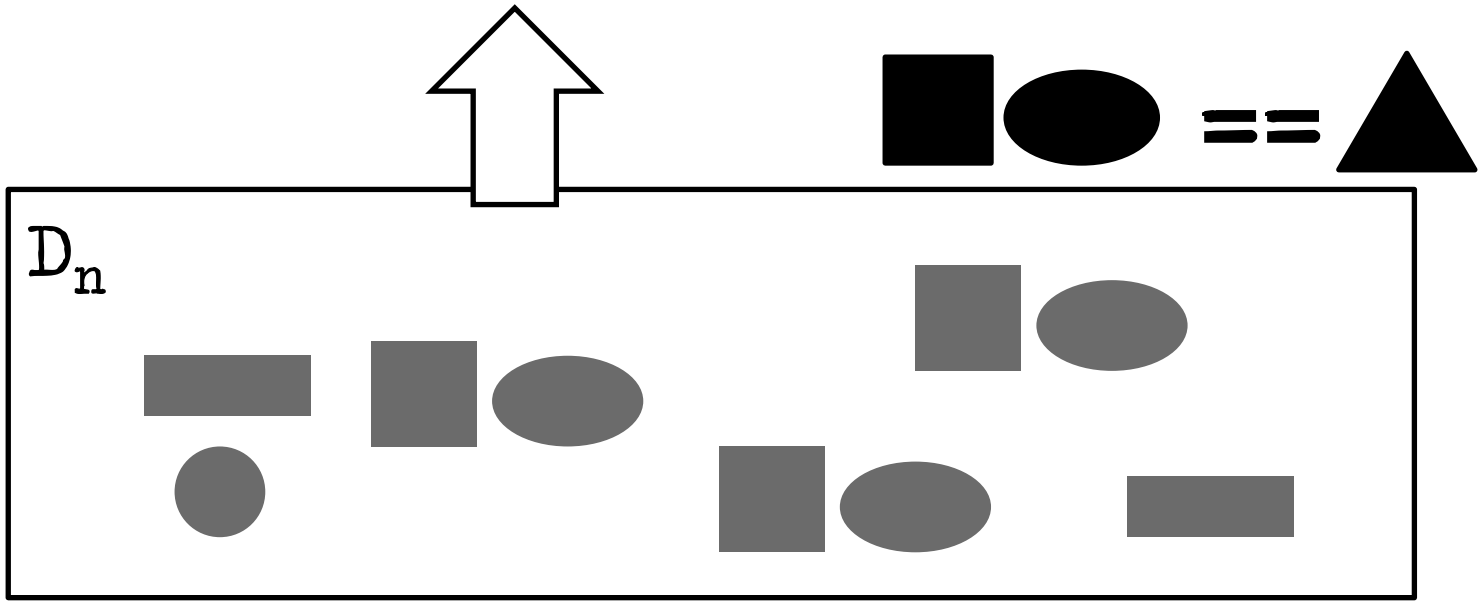


Extension



Extension

Good for **bottom-up** (inductive) domains, and for use by **technical** DSLs (people)



Extension

Drawbacks

- tightly bound to base
- potentially hard to analyze the combined program

Extension

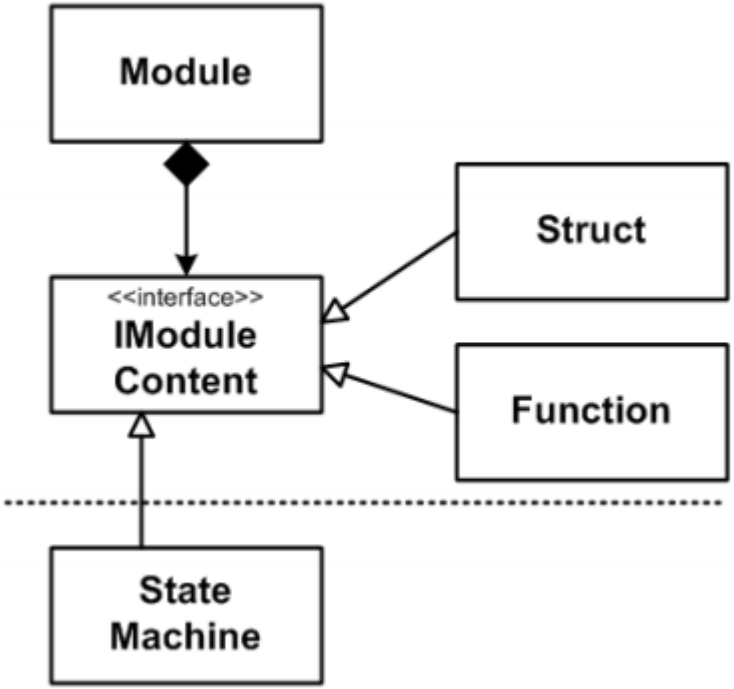
```
module main imports OsekKernel, EcAPI, BitLevelUtilies {

  constant int WHITE = 500;
  constant int BLACK = 700;
  constant int SLOW = 20;
  constant int FAST = 40;

  statemachine linefollower {
    event initialized;
    initial state initializing {
      initialized [true] -> running
    }
    state running { }
  }

  initialize {
    ecrobot_set_light_sensor_active
                (SENSOR_PORT_T::NXT_PORT_S1);
    event linefollower:initialized
  }

  terminate {
    ecrobot_set_light_sensor_inactive
                (SENSOR_PORT_T::NXT_PORT_S1);
```

```
task run cyclic prio = 1 every = 2 {
  stateswitch linefollower
    state running
      int32 light = 0;
      light = ecrobot_get_light_sensor
                          (SENSOR_PORT_T::NXT_PORT_S1);
      if ( light < ( WHITE + BLACK ) / 2 ) {
        updateMotorSettings(SLOW, FAST);
      } else {
        updateMotorSettings(FAST, SLOW);
      }
    default
      <noop>;
}

void updateMotorSettings( int left, int right ) {
  nxt_motor_set_speed(MOTOR_PORT_T::NXT
  nxt_motor_set_speed(MOTOR_PORT_T::NXT
}
```

Example

Exten
ded C

Extension



Example

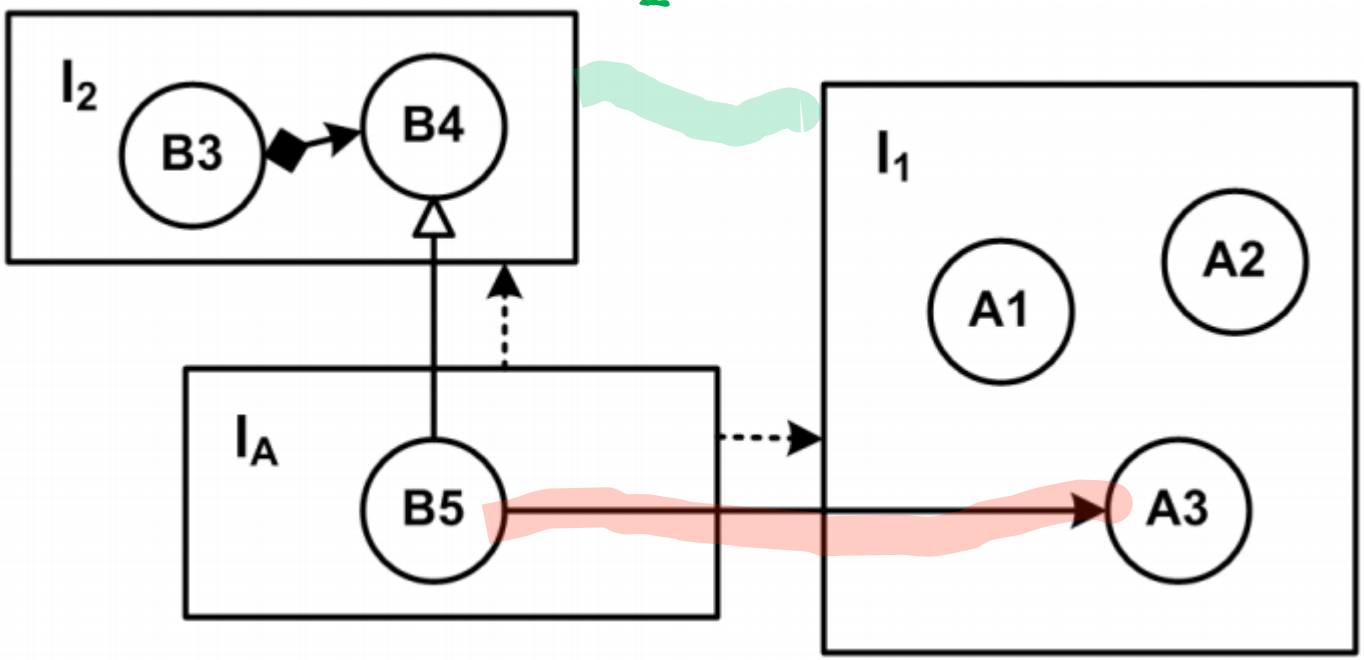
Exten
ded C

Reuse

languages dependencies	homogeneous	heterogeneous
independent	**Reuse**	**Embedding**
dependent	**Referencing**	**Extension**

fragment structure

Independent



No containment

Reuse

Often the referenced language is built expecting it will be reused.

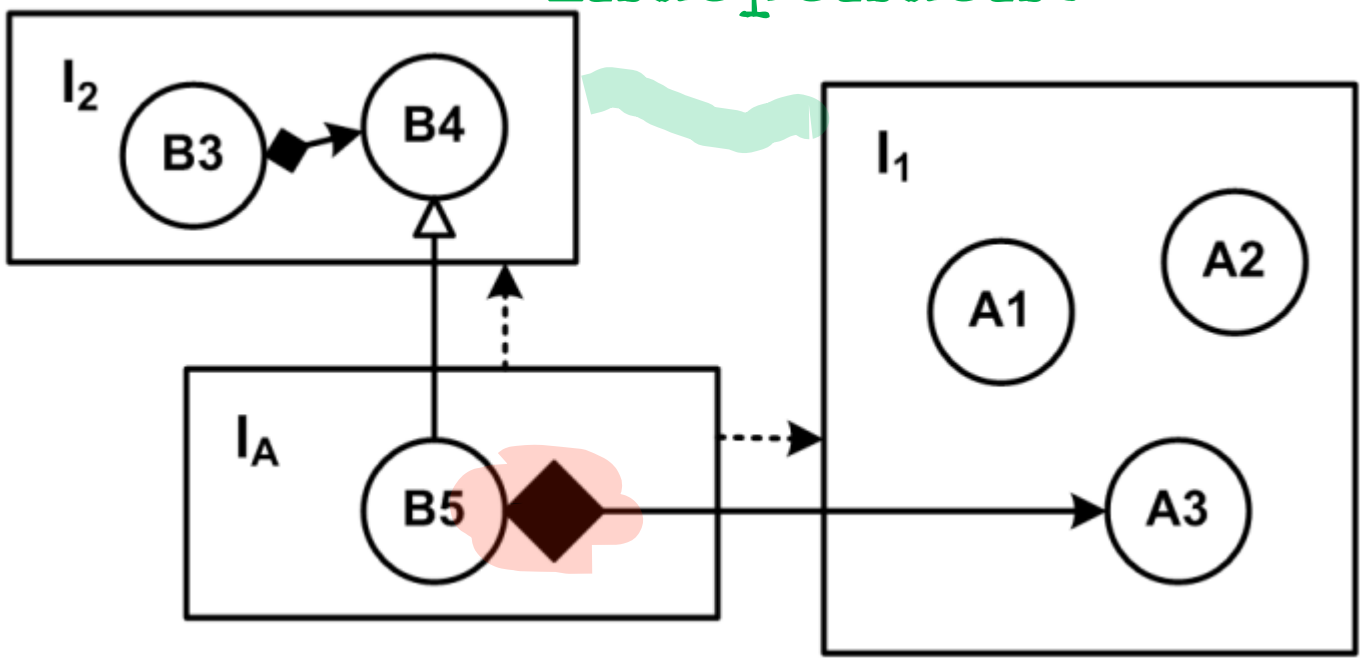
Hooks may be added.

Embedding

languages dependencies	homogeneous	heterogeneous
independent	Reuse	Embedding
dependent	Referencing	Extension

fragment structure

Independent



Containment

Embedding

Capgemini Pension Workbench

File Edit Projection Navigation Search Format Tools Dev Generate Pension Team NN

NNLCPA-14w2-21112008 * ×

Table of Contents ×

- Library
  - Documentation
  - Foundation
    - Value sets...
    - Tag definitions
      - Tag Basisberekening
      - Tag Ouderdomspensioen
      - Tag Partnerpensioen
      - Tag Wezenpensioen
      - Tag ANW extra
      - Tag WIA excedent AOV
      - Tag Eindkapitaal
  - Shared
    - Elements...
    - Rules
      - Rule Bereken Mutatieperiode
      - Rule Bereken Salaris ontwikkeling
      - Rule Bereken Pensioengrondslag
      - Rule Bereken Pensioengrondslag
      - Rule Bereken pensioengrondslag
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Bedrag jaaropbouw
      - Rule Bereken Delta deelaanspraak uit doorl
      - Rule Bereken Deelaanspraak opgebouwd
      - Rule Bereken Toekomstige dienstjaren
      - Rule Bereken Deelaanspraak uitzicht
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verzekerd bedrag
      - Rule Bereken Verkoopkosten
      - Rule Bereken Netto Eindwaarde
      - Rule Bereken einddatum opbouw
      - Rule Bereken premie
      - Rule Bereken IS-Opslag
      - Rule Bereken administratiekosten

All

Elements...

Rules

Rule Bereken Mutatieperiode

Result:
Mutatieperiode

Name:
Bereken Mutatieperiode

Documentation:
Het vaststellen van de periode tussen de huidige en de vorige mutatie in dagen.¶
De mutatieperiode kan niet meer dan 360 dagen bedragen omdat elk jaar een begin- en eindmutatie kent i.v.m. het openen en sluiten van het verslagjaar.¶
Dit wordt niet afgevangen omdat het uitvoeren van de begin- en eindmutatie verantwoordelijkheid zijn van de pensioenadministratie.¶

Tags:
Basisberekening

Algorithm:
if maximum(Mutaties per datum) == 1 **then daysof**(**duration**(**valid**(Mutaties per datum))) **else** 0

Test cases:

Name	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Gelijke datums	03/01/2008		Mutatieperiode - Mutatiedatum = Mutatiedatum Vorig			3	0
Periode < 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (binnen 1 maand)			15	15
Periode > 30	03/01/2008		Mutatieperiode - Mutatiedatum > Mutatiedatum Vorig (meerdere maanden)			60	

Bereken Mutatieperiode ▸ Test cases ▸ Unit test: Gelijke d

Doc | Splitter | Pension | PensionDecorated | AM

Example

Pension Plans

Embedding

Embedding often uses Extension to extend the embedded language to adapt it to its new context.

Challenges - Syntax

<u>Extension and Embedding</u> requires modular concrete syntax

Many tools/formalisms cannot do that

Challenges - Type Systems

<u>**Extension:**</u> the type system of the base language must be designed to be extensible/ overridable

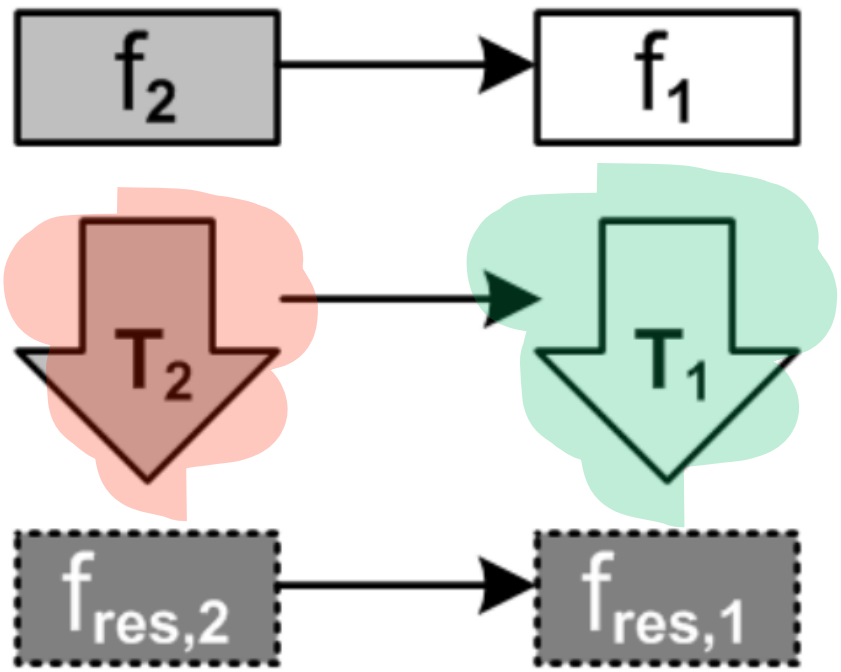
Challenges - Type Systems

<u>**Reuse and Embedding:**</u> Rules that affect the interplay can reside in the adapter language.

Challenges - Trafo & Gen

Referencing (I)

Written specifically for the combination

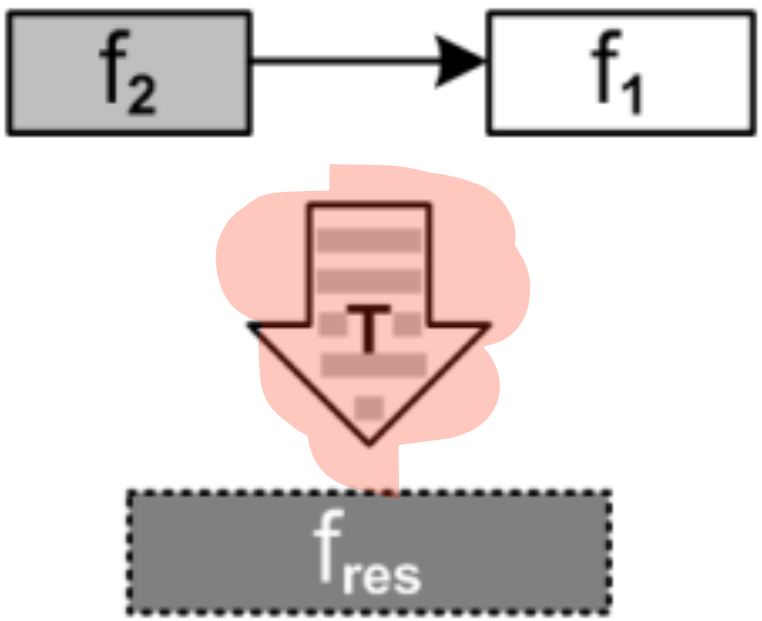


Can be Reused

Two separate, dependent single-source transformations

Challenges - Trafo & Gen

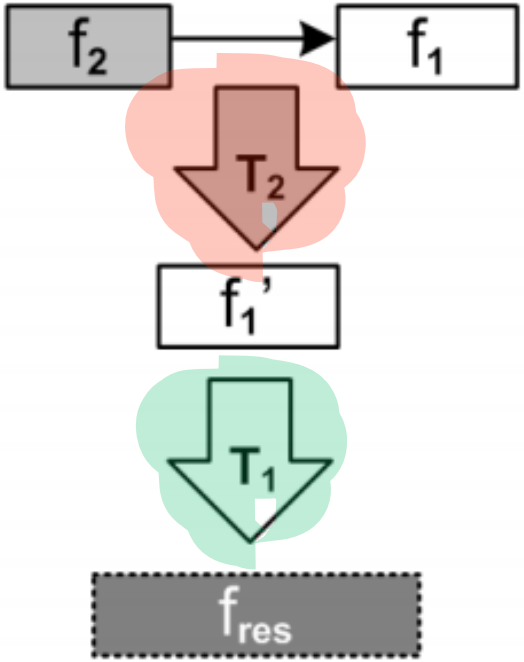
Referencing (II)



A single multi-sourced transformation

Challenges - Trafo & Gen

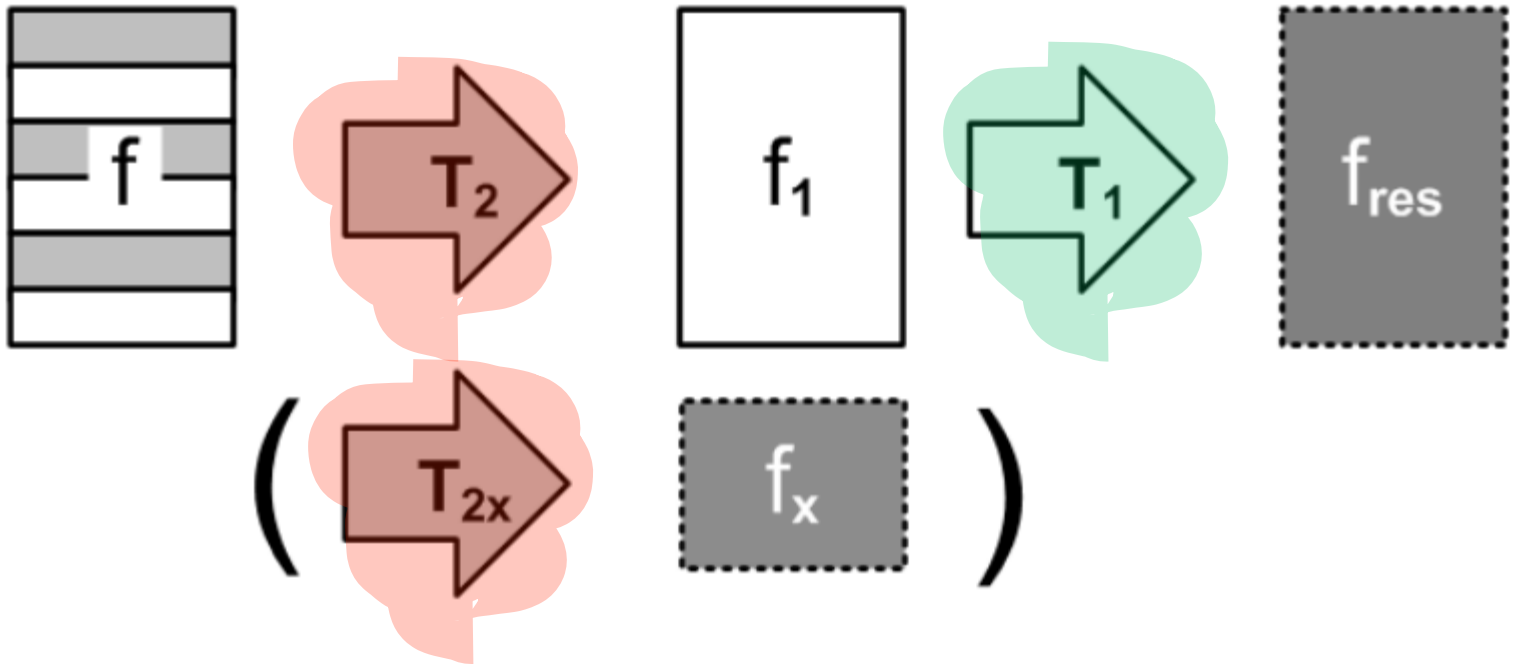
Referencing (III)



A preprocessing trafo that changes the referenced frag in a way specified by the referencing frag

Challenges - Trafo & Gen

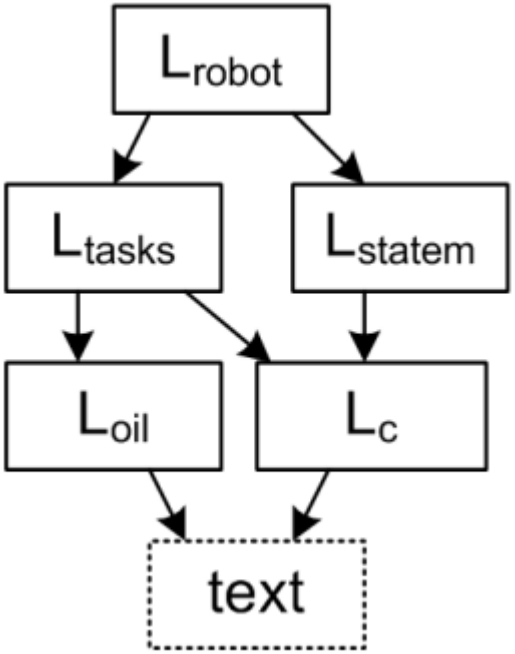
Extension



Transformation by assimiliation, i.e. generating code in the host lang from code expr in the extension lang.

Challenges - Trafo & Gen Extension

```
module impl imports <<imports>> {

  int speed( int val ) {
    return 2 * val;
  }

  robot script stopAndGo
    block main on bump
      accelerate to 12 + speed(12) within 3000
      drive on for 2000
      turn left for 200
      decelerate to 0  within 3000
      stop

}
```

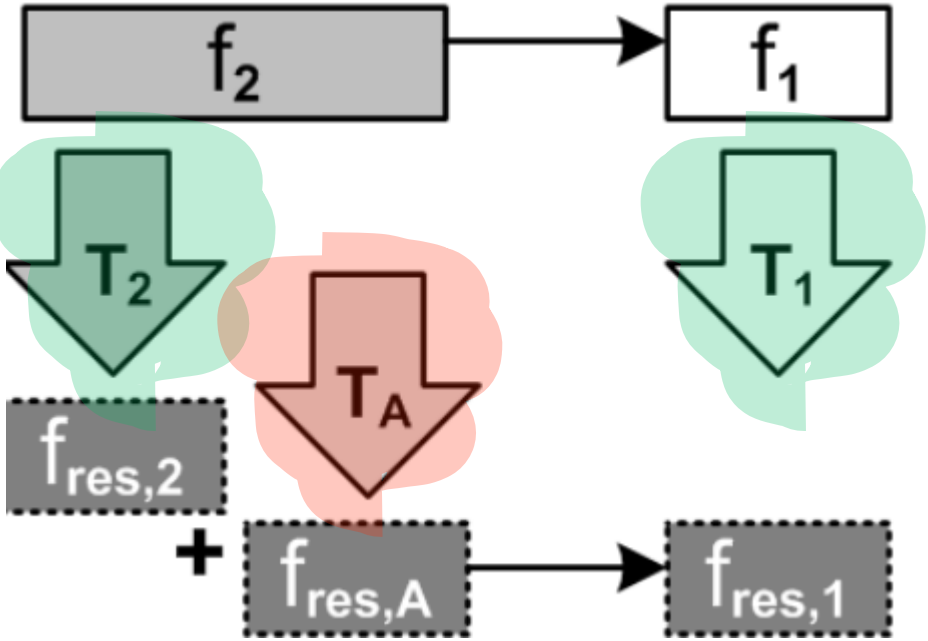


Example

Exten
ded C

Challenges - Trafo & Gen

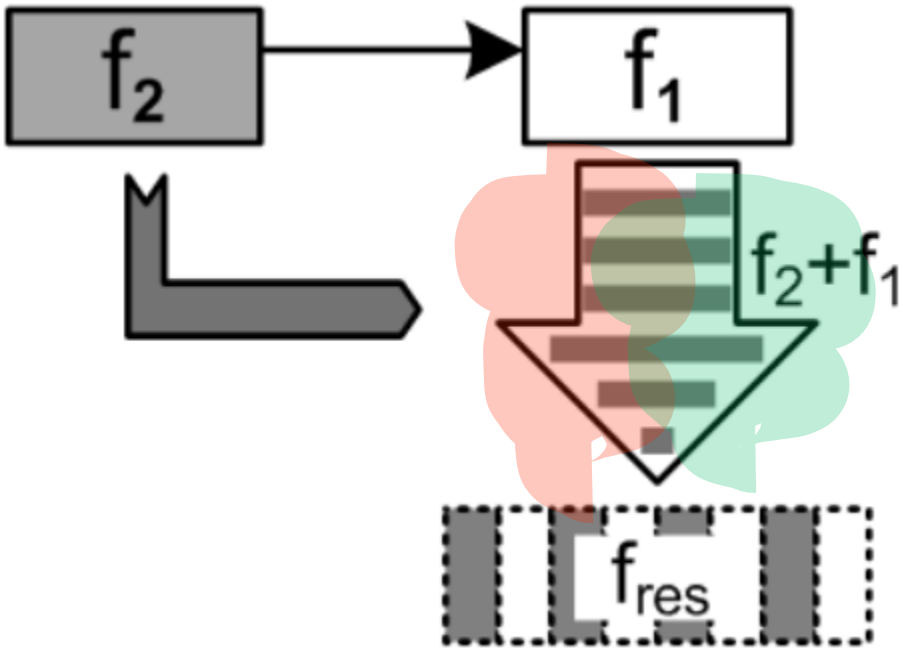
Reuse (I)



Reuse of existing transformations for both fragments plus generation of adapter code

Challenges - Trafo & Gen

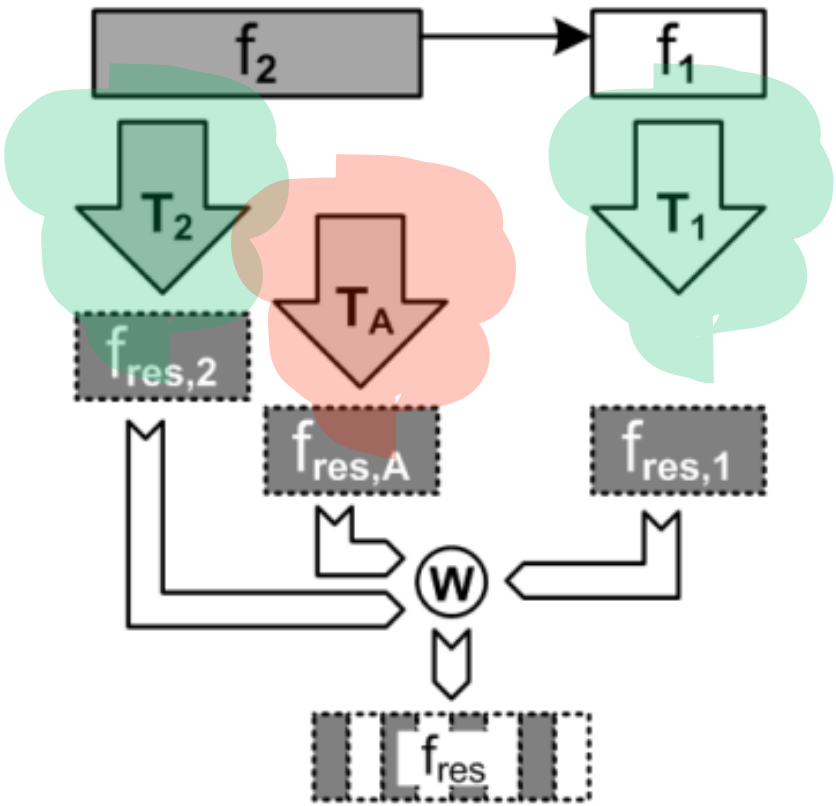
Reuse (II)



composing transformations

Challenges - Trafo & Gen

Reuse (III)

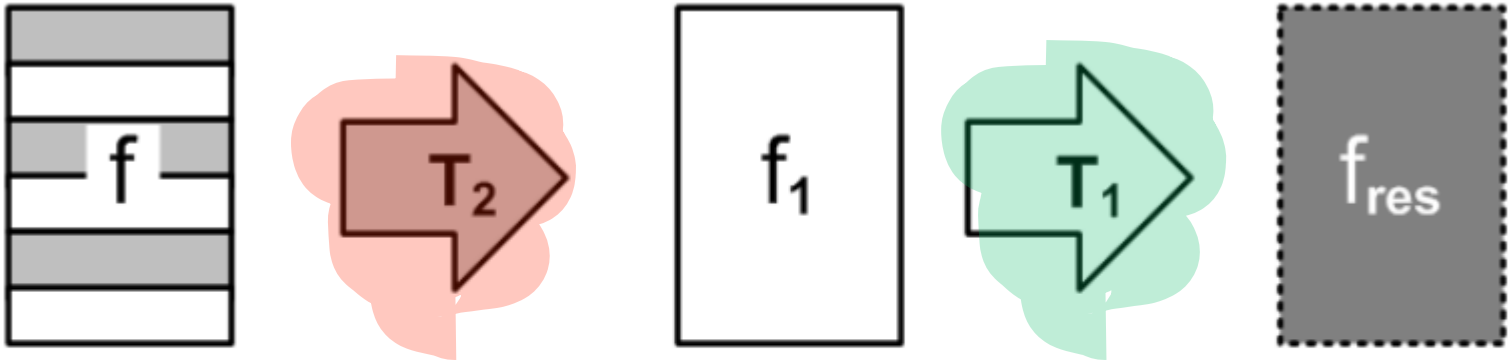


generating separate artifacts
plus a weaving specification

Challenges - Trafo & Gen

Embedding (I)

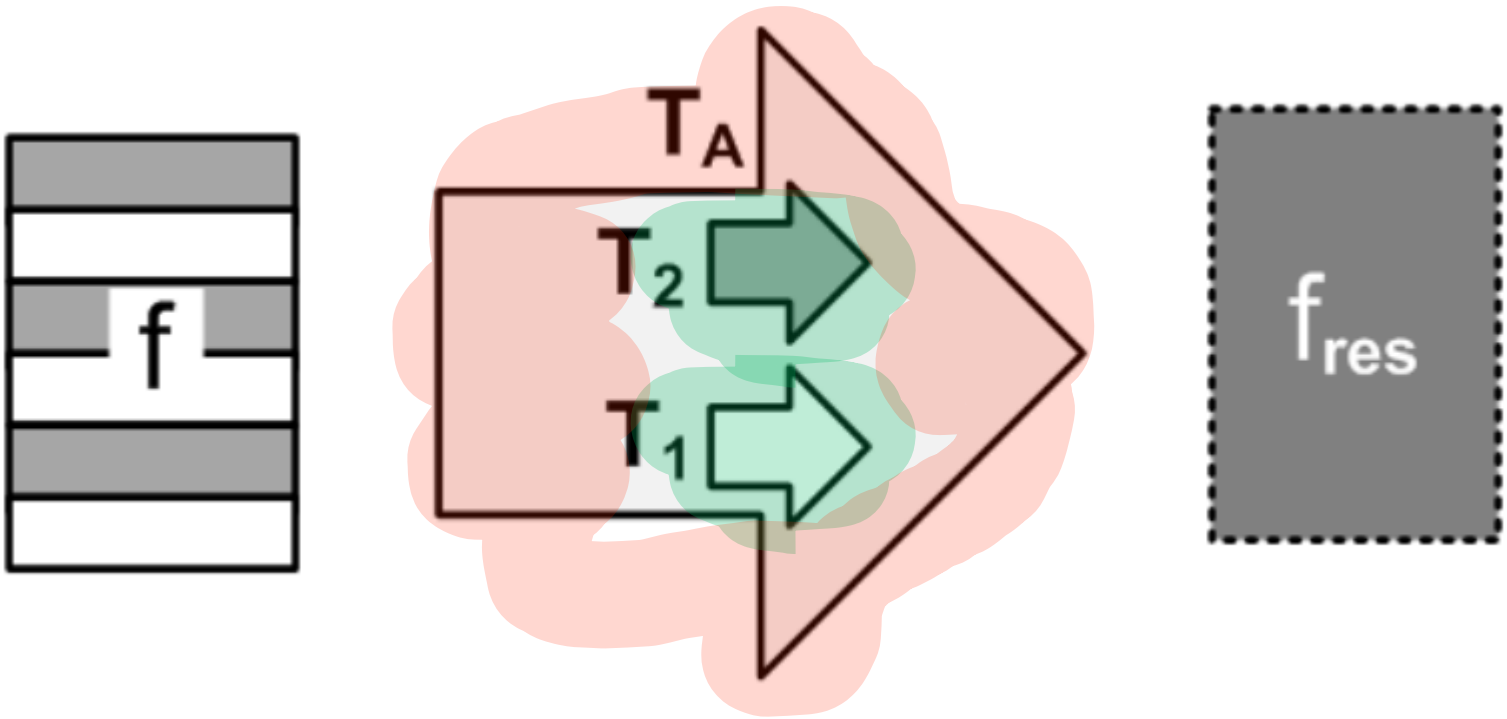
a purely embeddable language
may not come with a generator.



Assimilation (as with Extension)

Challenges - Trafo & Gen

Embedding (II)



Adapter language can coordinate the transformations for the host and for the emebedded languages.

Concrete Syntax

expressivity
coverage
semantics
separation of concerns

completeness
paradigms
modularity
concrete syntax

process

UI for the language!

Important for acceptance by users!

Textual
Symbolic
Tabular
Graphical

Reuse existing syntax of domain, if any!

Tools let you freely combine all kinds.

Default: Text

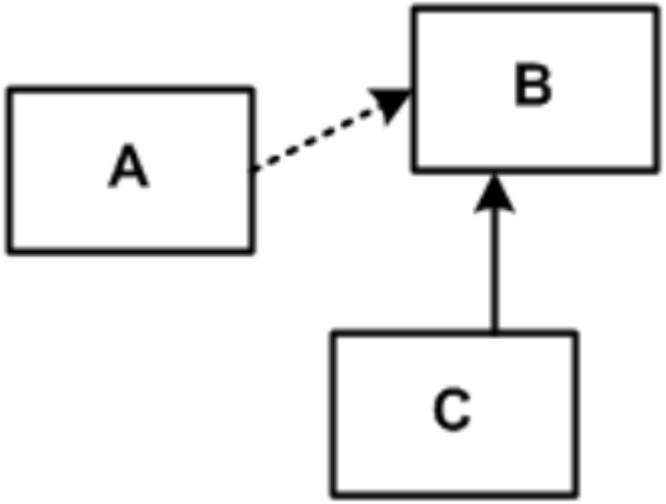
Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs

Default: Text

Editors simple to build
Productive
Easy to integrate w/ tools
Easy to evolve programs

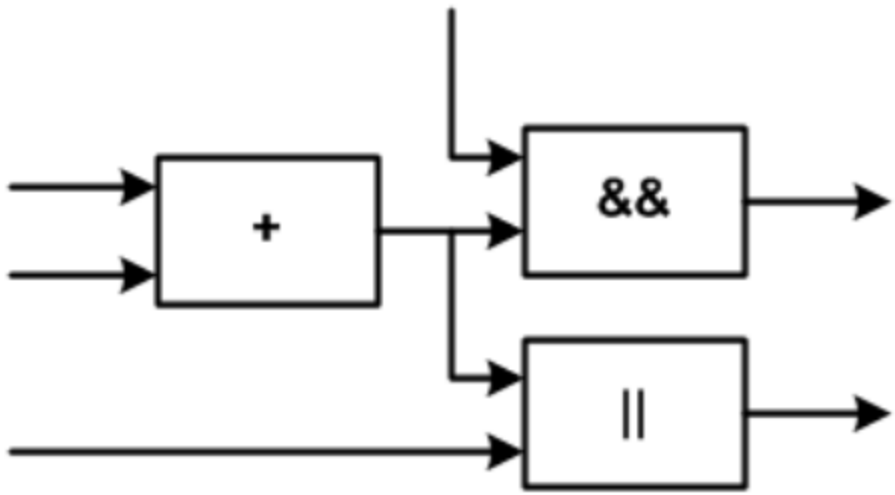
… then add other forms,
if really necessary

Graphical in case…



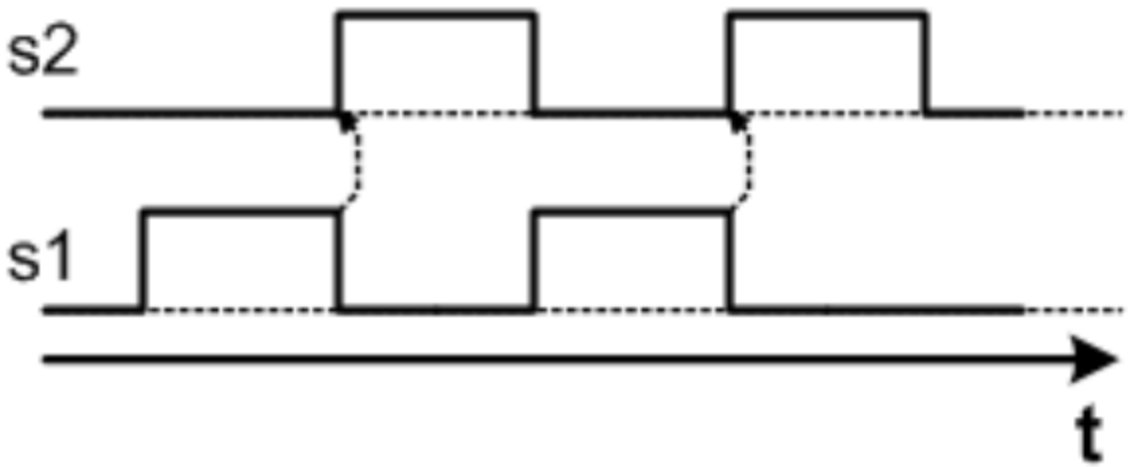
Relationships

Graphical in case…



Flow and
Depenency

Graphical in case…



Causality
and Timing

Symbolic

Either Mathematical, or often highly inspired by domain

¤ 3.3 Commutatiegetallen op 1 leven¶

$$D_x = v^x * \frac{l_x}{100}$$ ¤6 Dec(3)¶

Implemented in ¤ V9401¶

¶

$$N_x = \sum_{t=0}^{\omega - x} D_{x+t}$$ ¤7 Dec(3)

¶

¤ 3.6 Contante waarde 1 leven/ 2 levens¶

$${}_nE_x = \frac{D_{x+n}}{D_x}$$ ¤19 Dec(4)

¶

$$a_x = \ddot{a}_x - 1$$ ¤21 Dec(3)

¶

$$\bar{a}_x = \ddot{a}_x - 0{,}5$$ ¤22 Dec(3)

¶

$$\ddot{a}_{xn\rceil} = \frac{N_x - N_{x+n}}{D_x}$$ ¤23 Dec(3)

$$\bar{a}_{xn\rceil} = \ddot{a}_{xn\rceil} - 0{,}5 + 0{,}5 * {}_nE_x$$ ¤25 Dec(3)

¶

Tables

Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	8250

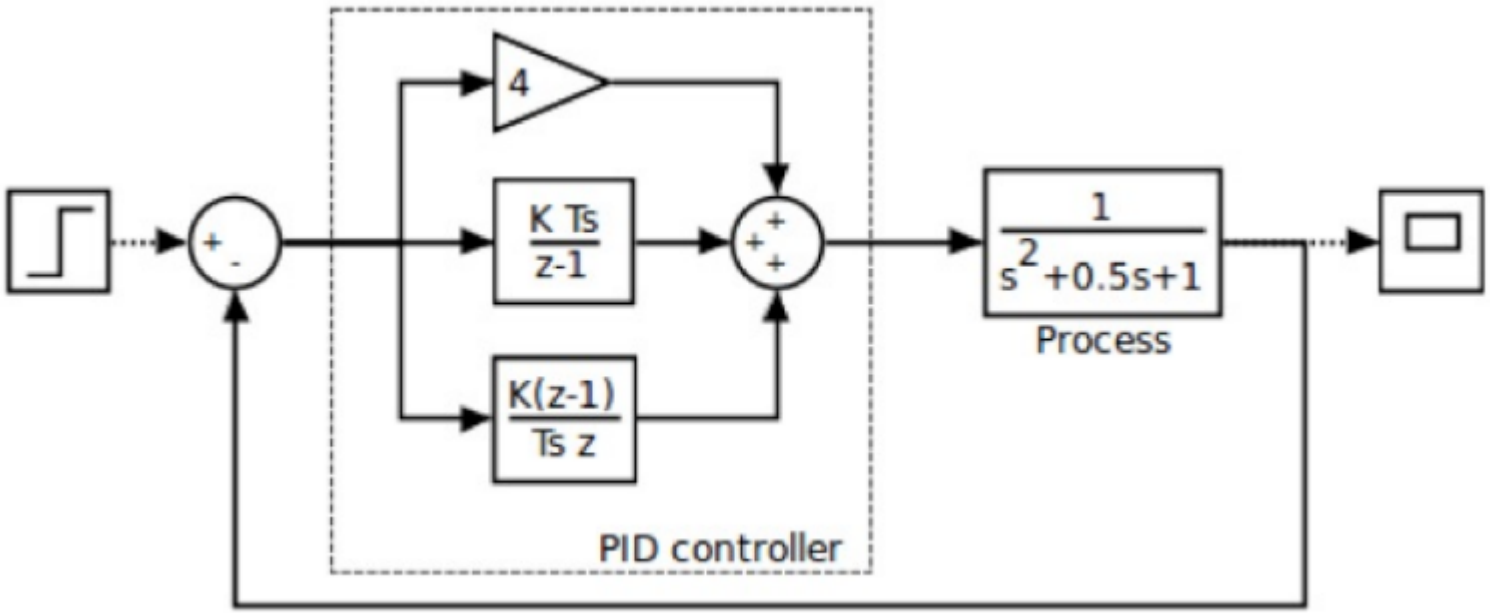
Combinations

```
c/s interface Decider {
  int decide(int x, int y) pre
}

component AComp extends nothing {
  ports:
    provides Decider decider
  contents:
    int decide(int x, int y) <- op decider.decide {
      return int, 0
```

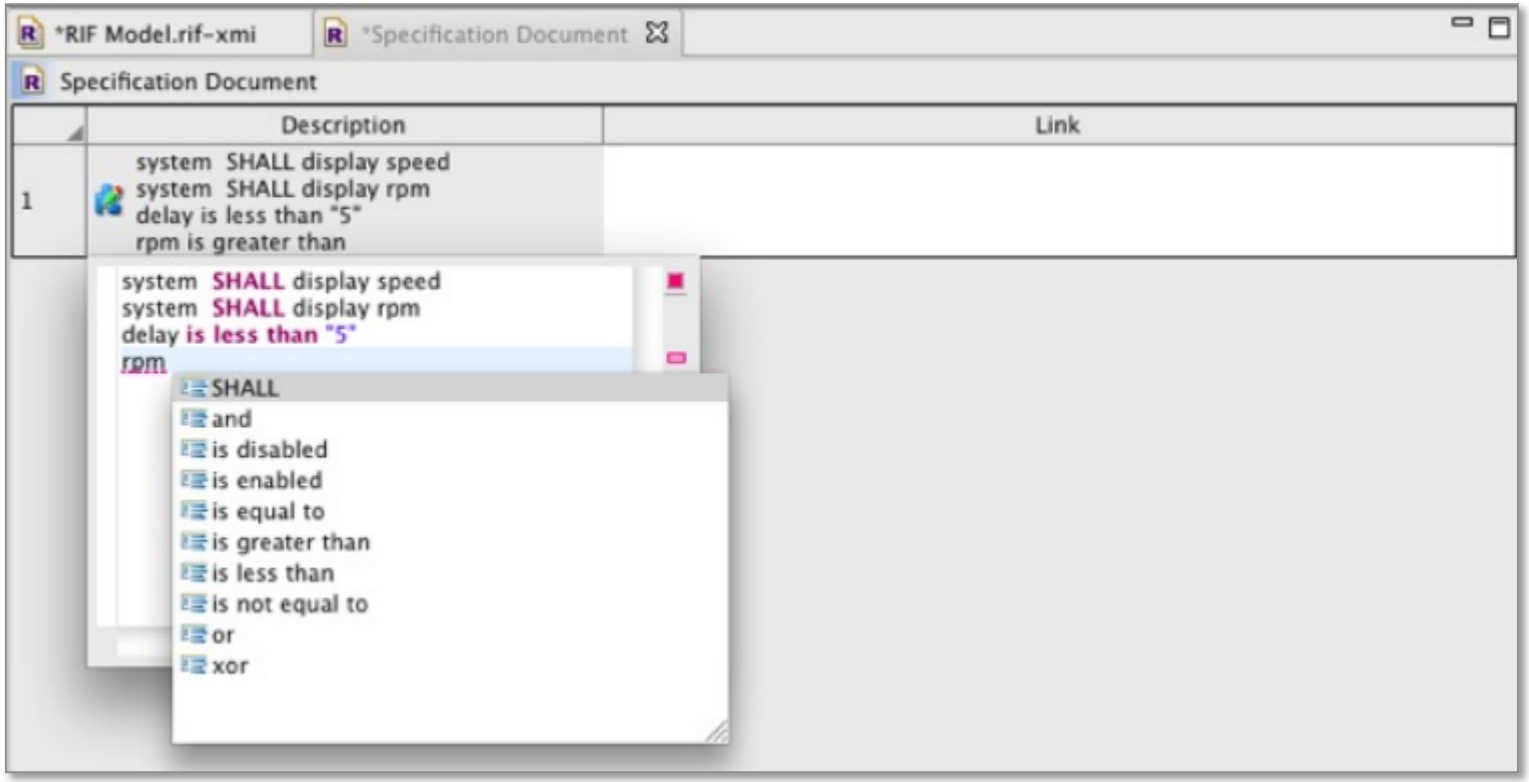
	x == 0	x > 0
y == 0	0	1
y > 0	1	2

```
    }
}
```

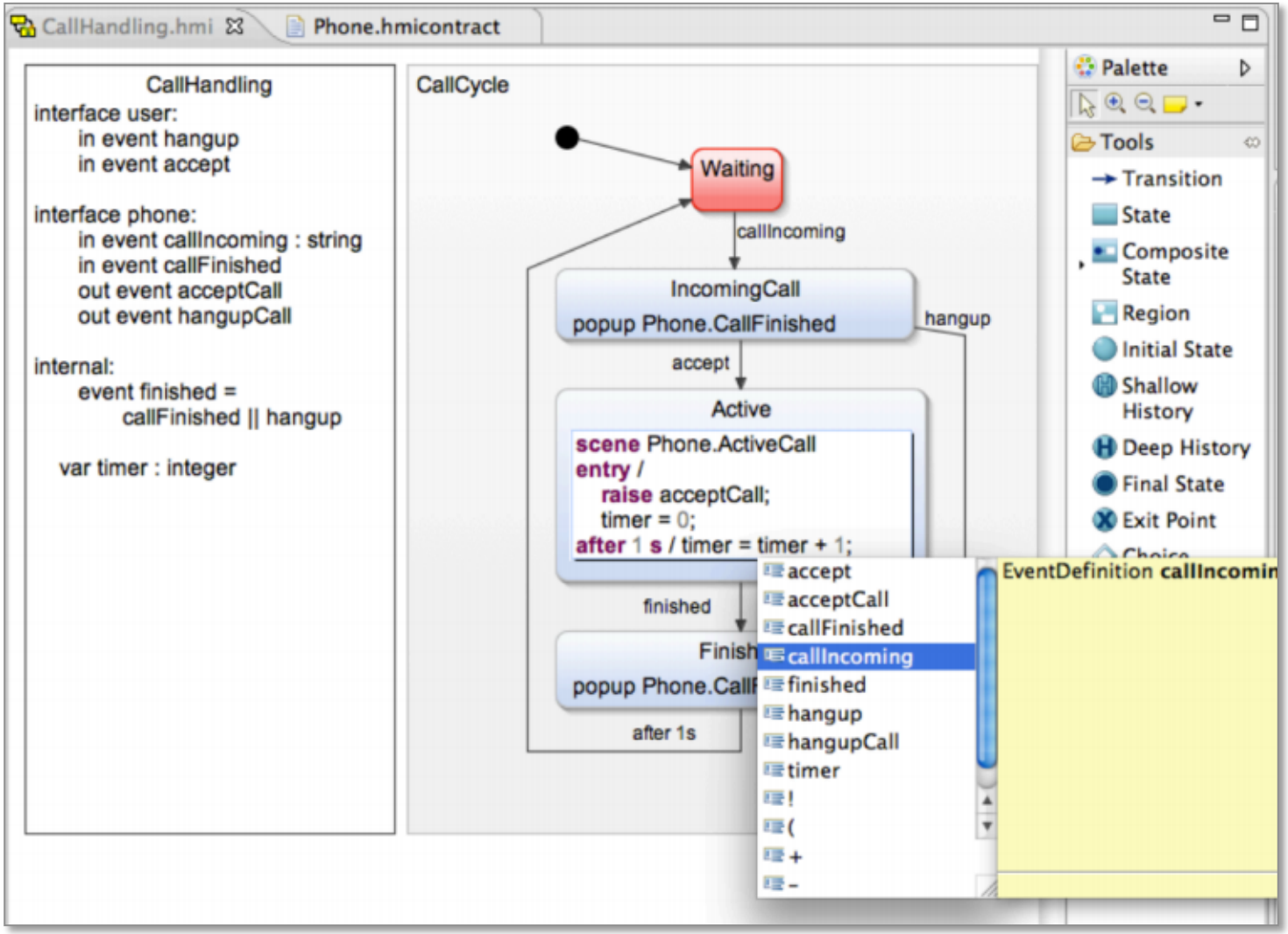
Combinations



Combinations



Combinations

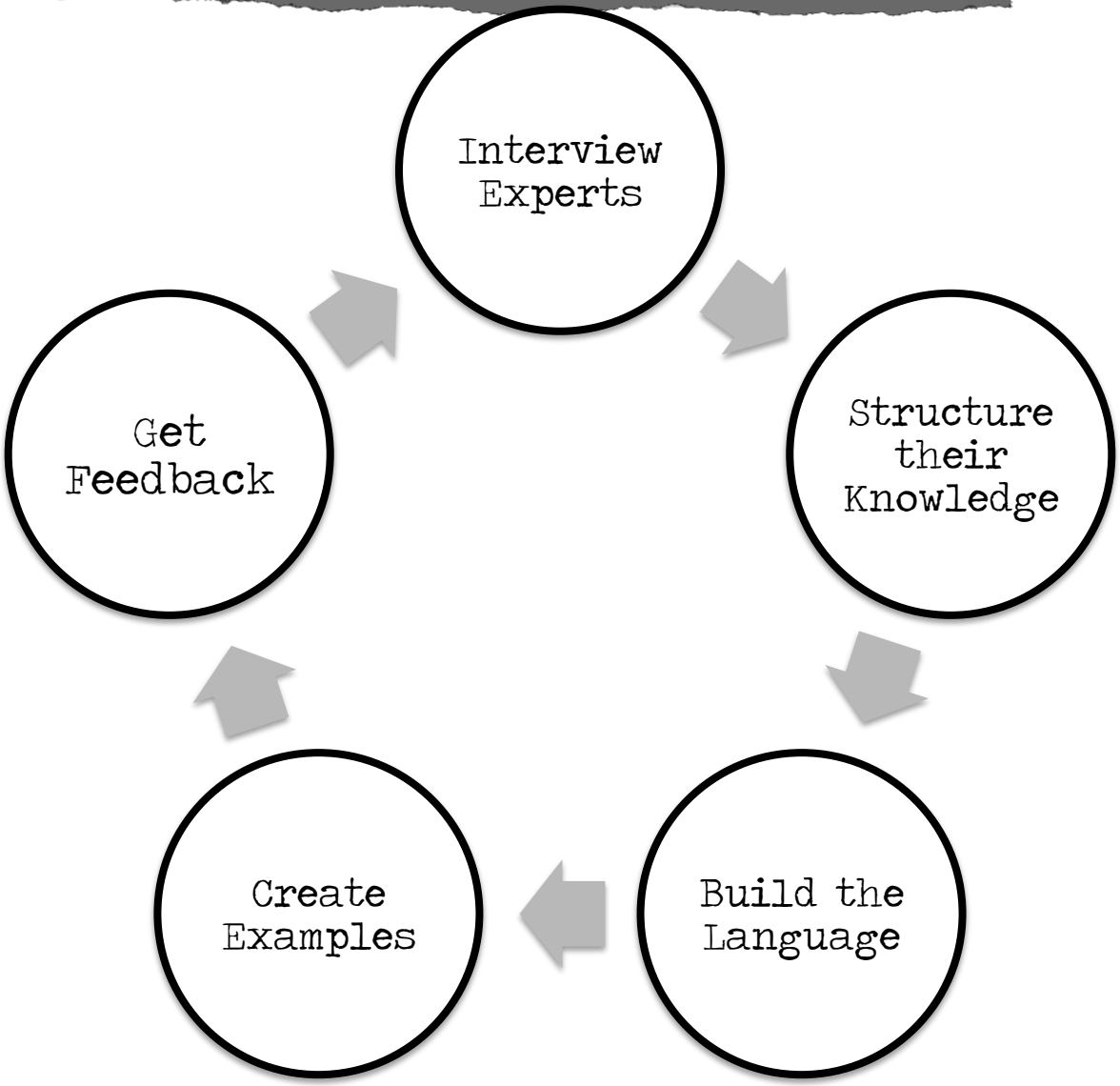


Process

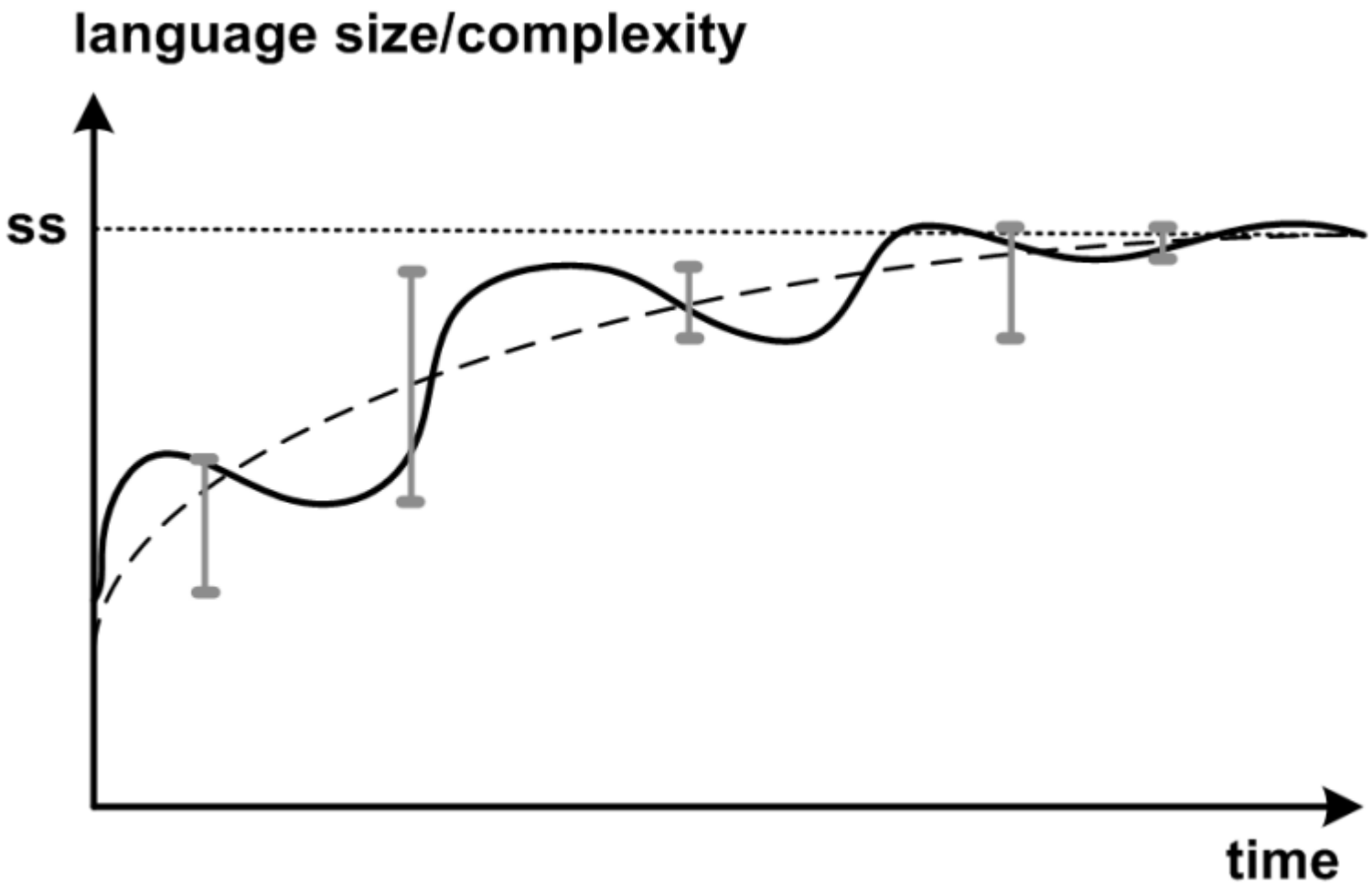
expressivity	completeness
coverage	paradigms
semantics	modularity
separation of concerns	concrete syntax

process

Domain Analysis



Iterate to goal



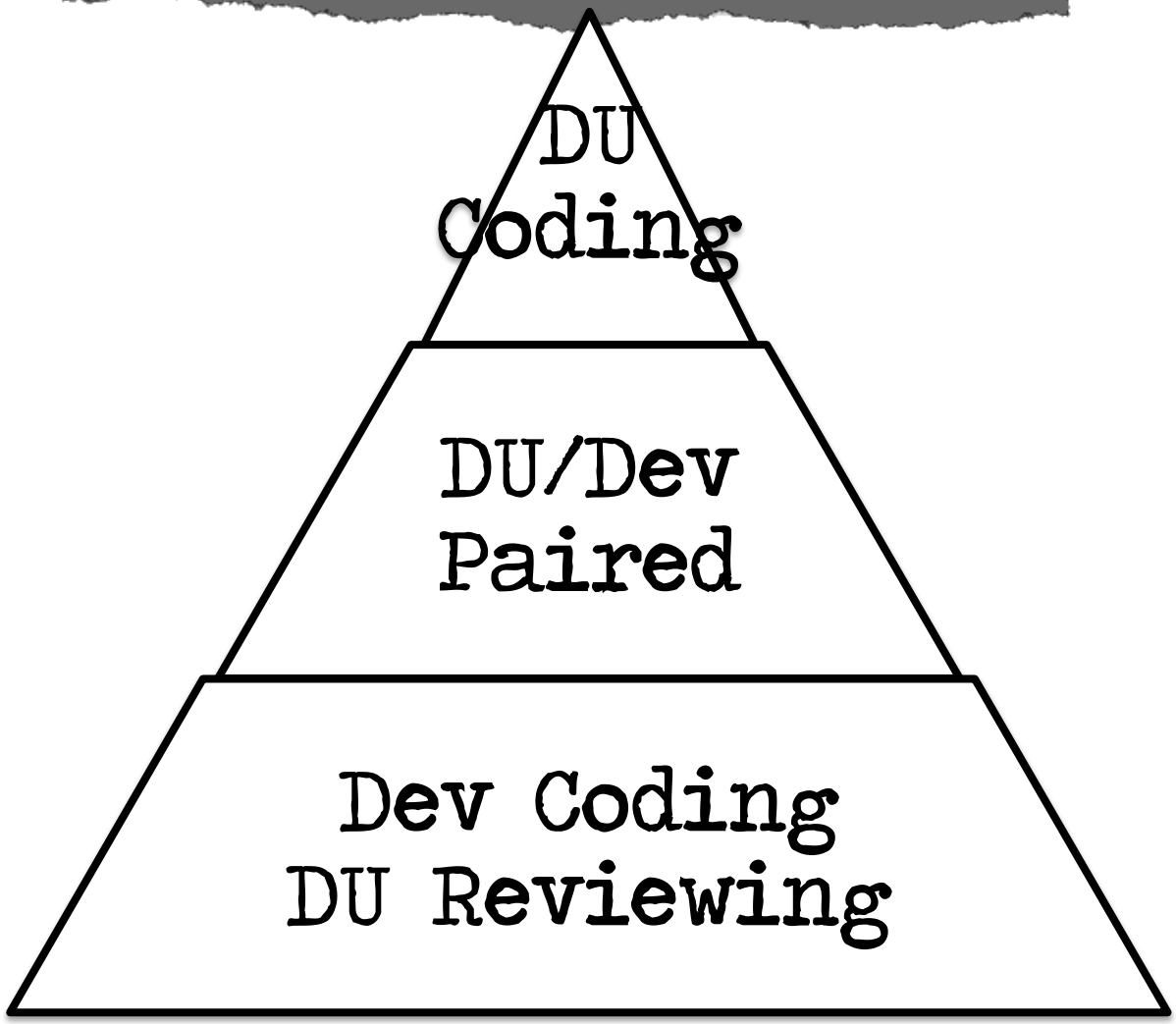
Documentation

Create example-based tutorials!

Domain Folks Programming?

Precision vs. Algorithmics!

Domain Folks Programming?



DSL as a Product

Release Plan
Bug Tracker
Testing!
Support
Documentation

Reviews

Reviews become easier --- less code, more domain-specific

The End.

This material is part of my upcoming (early 2013) book

DSL Engineering with Language Workbenches

Stay in touch; it will be cheap or maybe even free :-)

www.voelter.de/dslbook

@markusvoelter
+Markus Voelter