Software Composition by Example

(aka, sky is the limit)

Sébastien Mosser
I3S Laboratory
SPARKS team

UNIVERSITÉ CÔTE D'AZUR

1 The Pulse Totem use case

2 Composing Software?

3 Composition is everywhere!

4 Defining a comp. operator

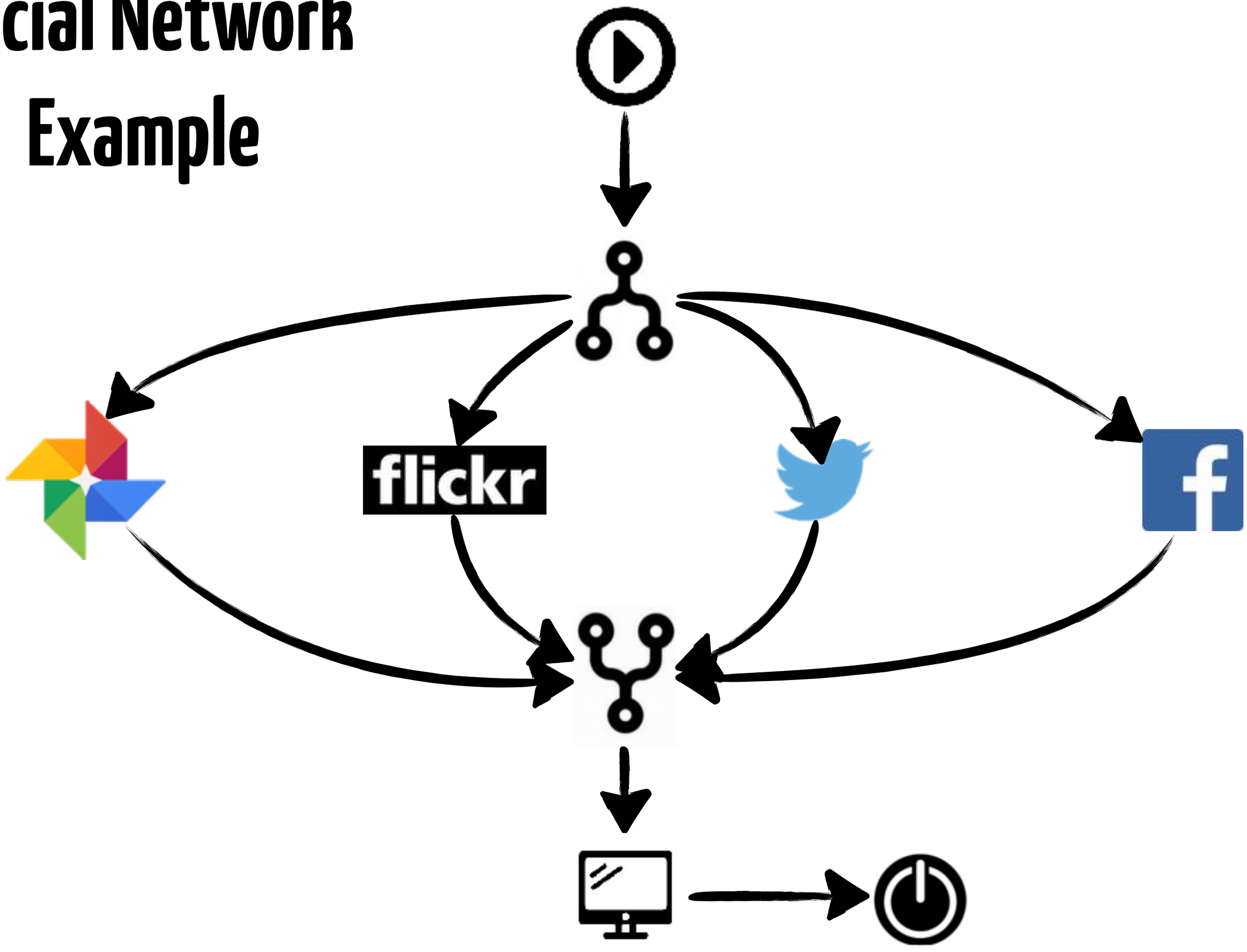
5 Exploiting an operator

6 Sensor networks example

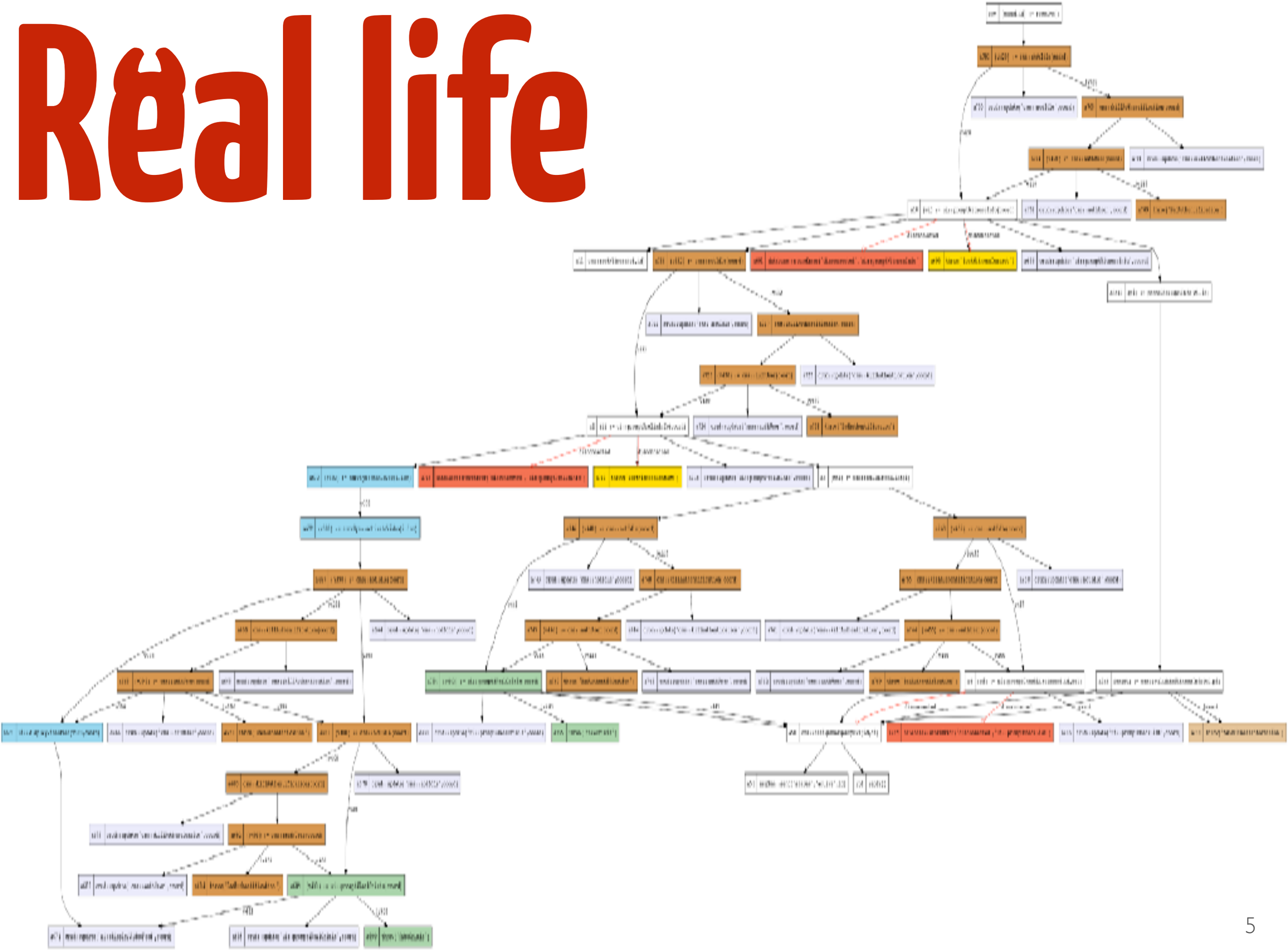
Example of “domain-driven” composition

PulseTOTEM

Social Network Example



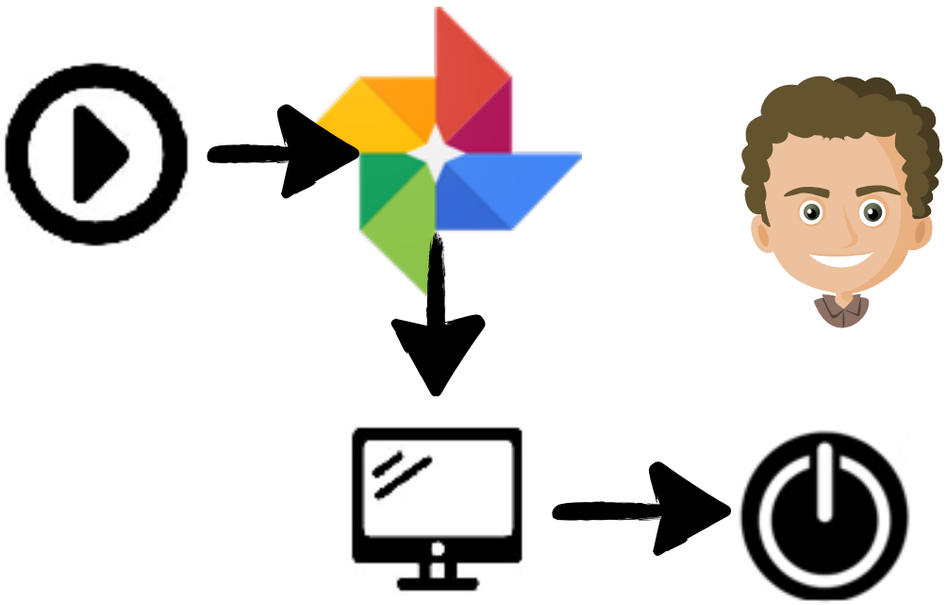
Real life

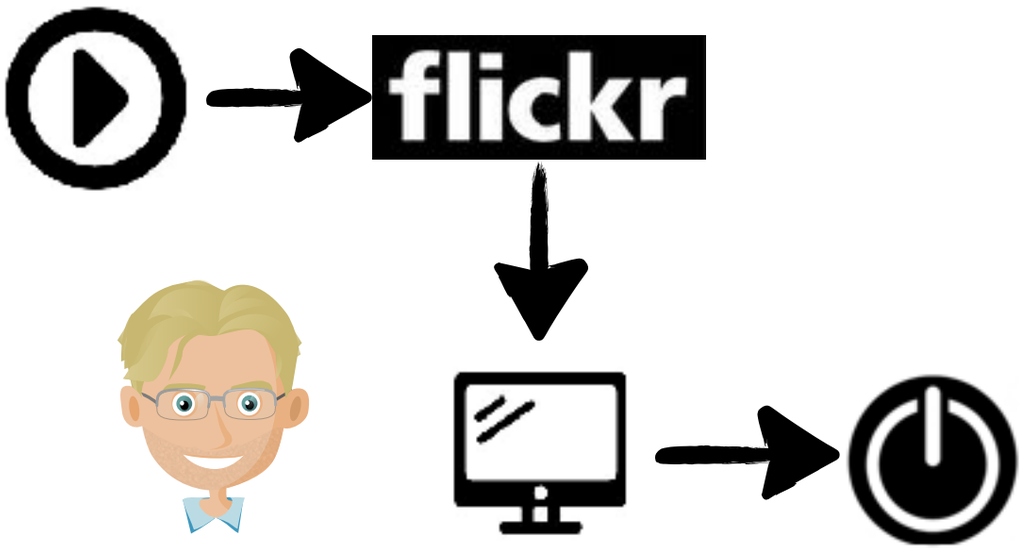


11

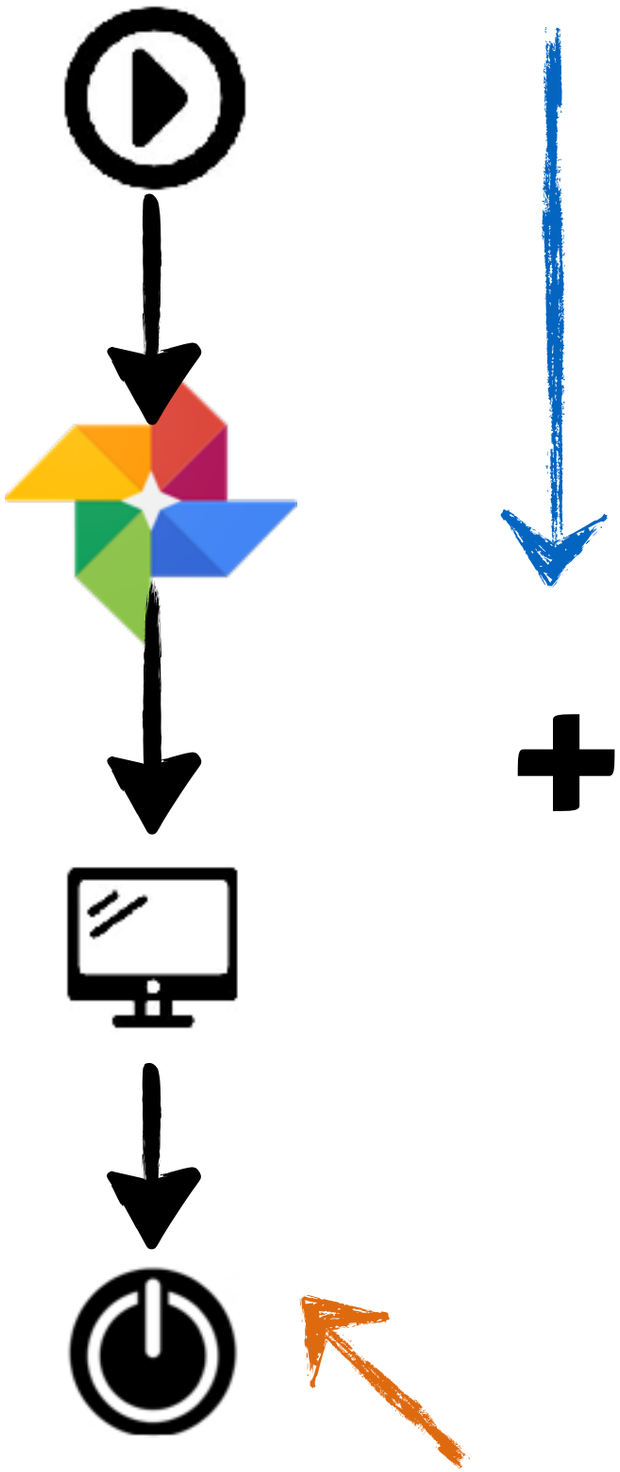
Build **complex things** by composing **small** and **simple** ones.

- E. Dijkstra [72]

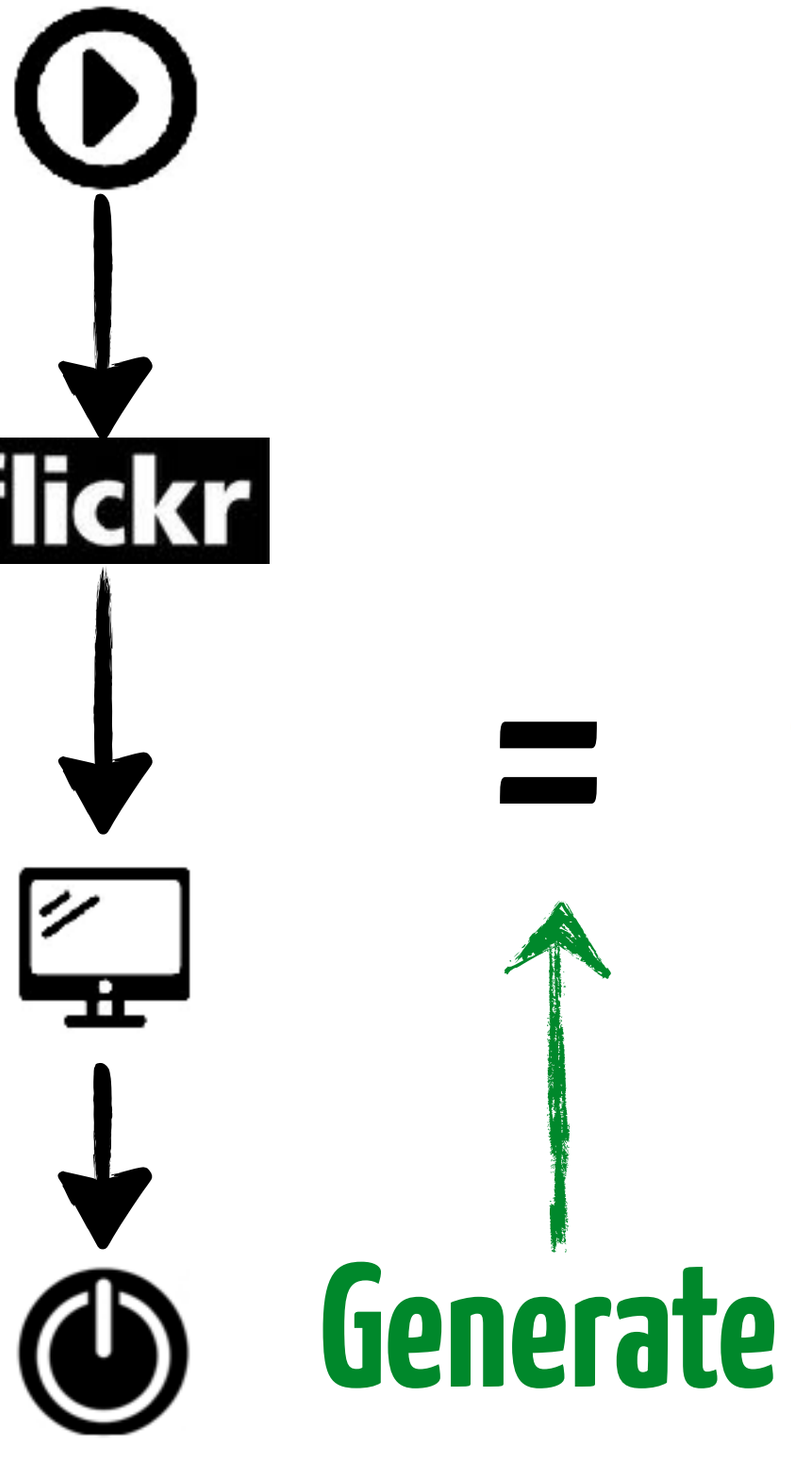




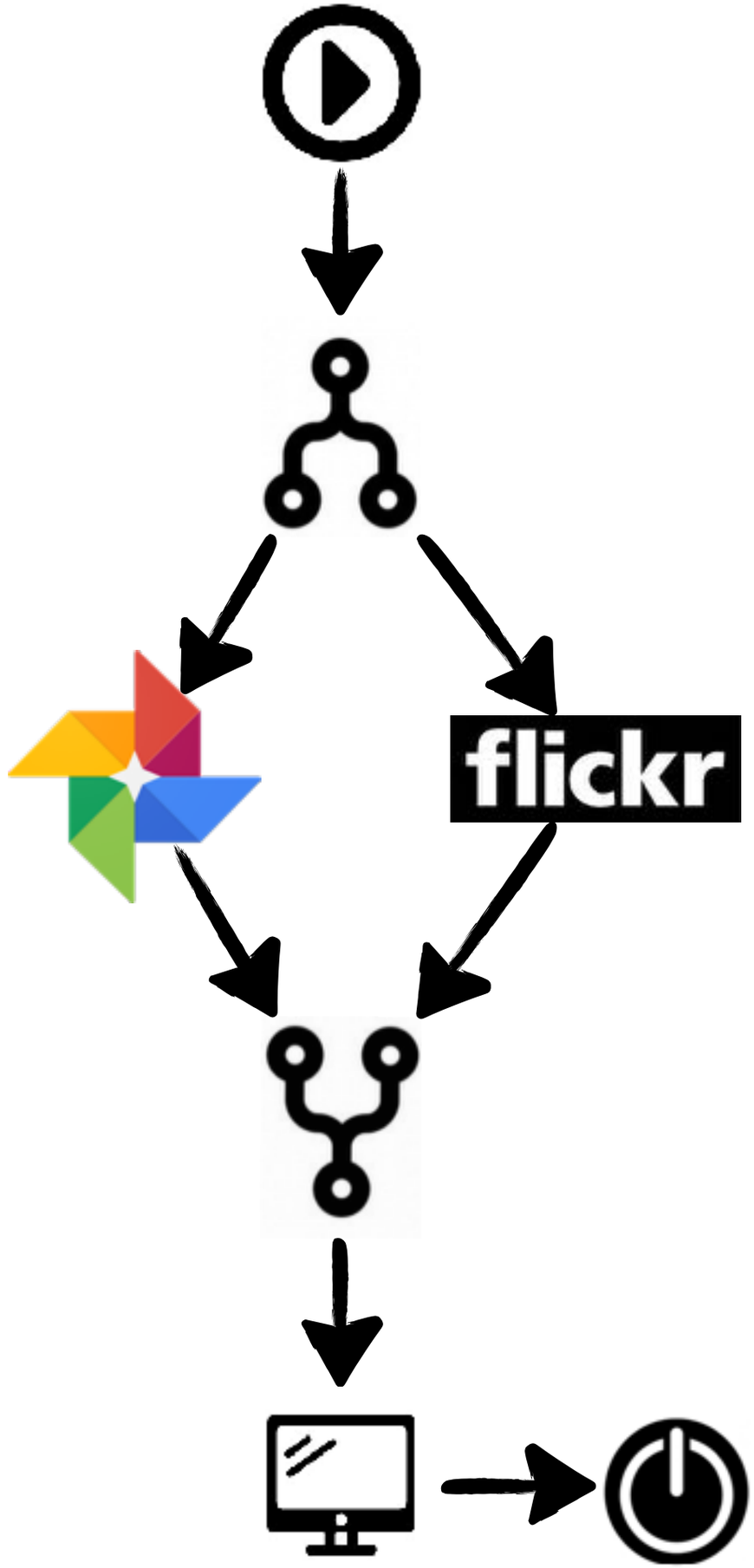
Compose



+

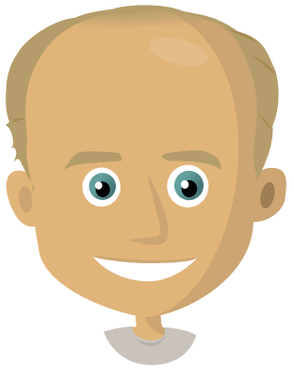


=



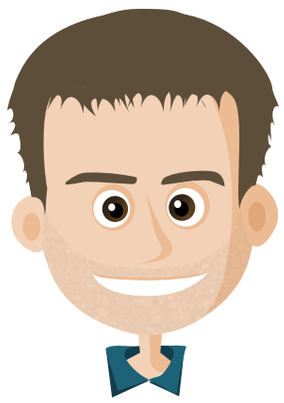
Separate

Generate



Client

As an information consumer,
I want to get  & flickr
so that I can display my pictures

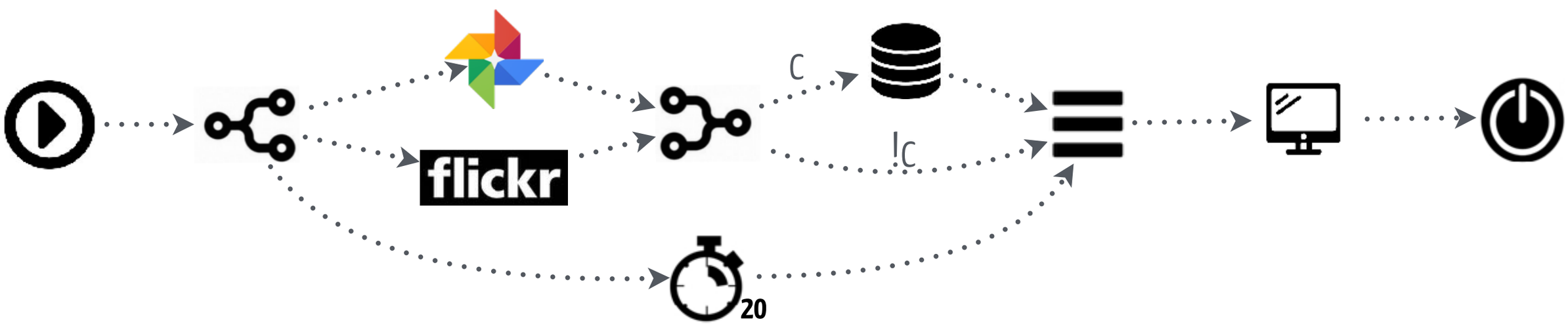


Developer

P =  + flickr

Information from & with 20s timeout and caching

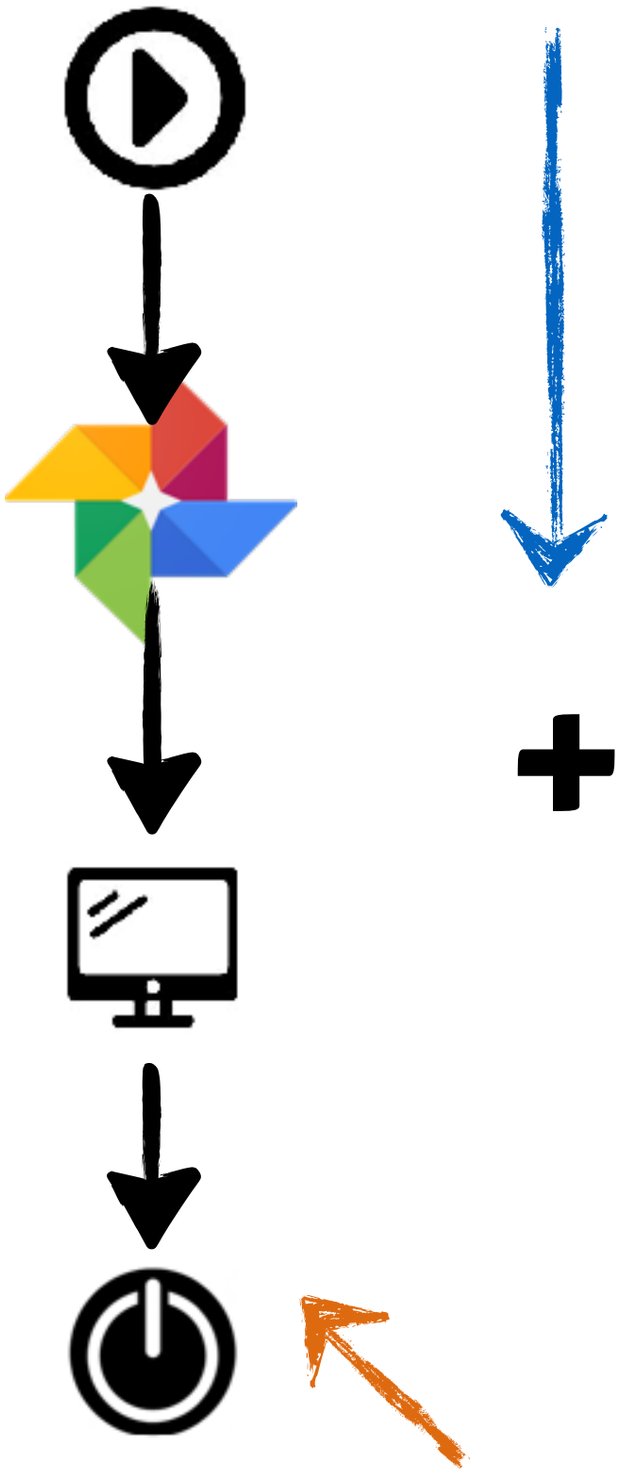
p’ = ((+) + + 20)



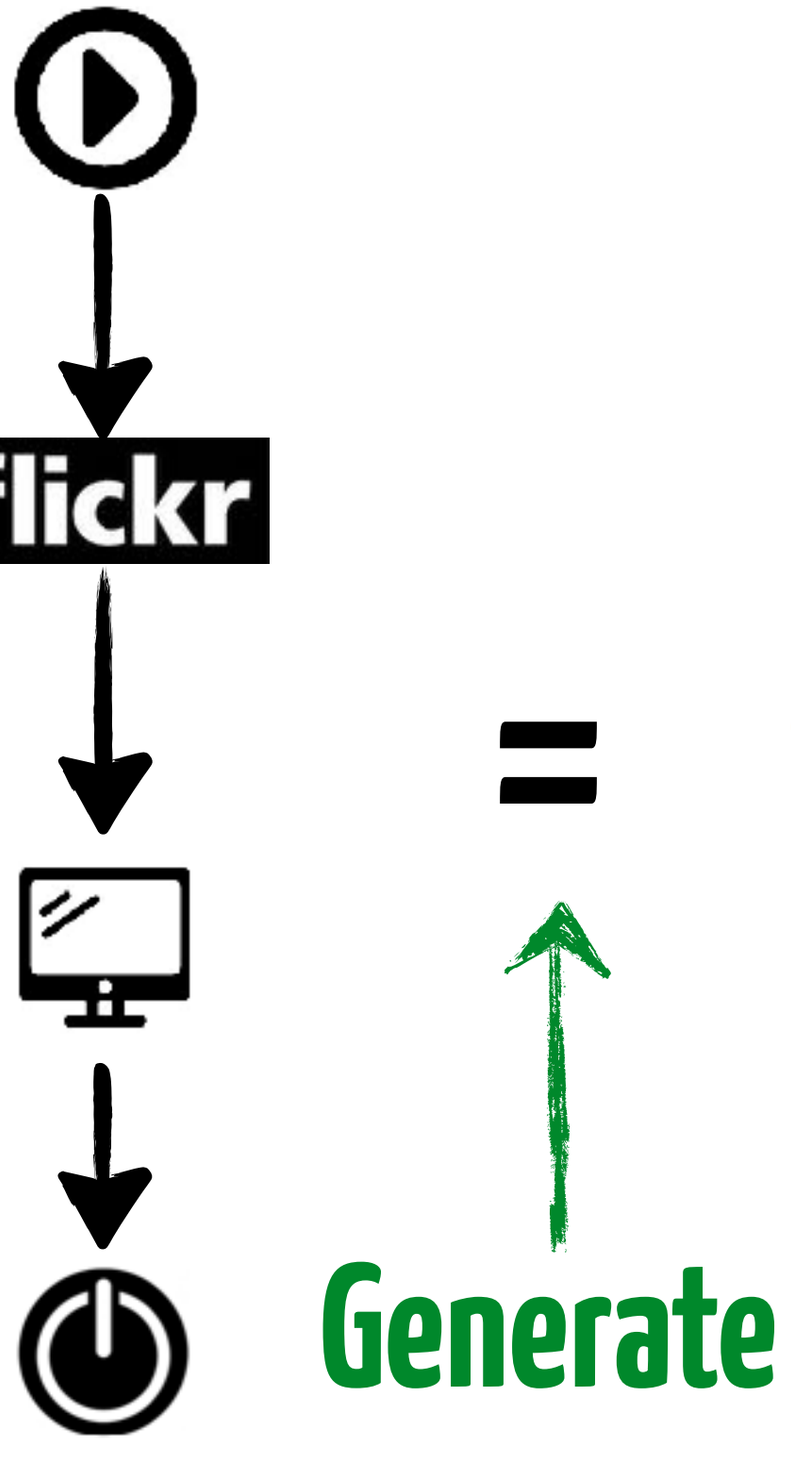
Composing Software?

It looks complicated...

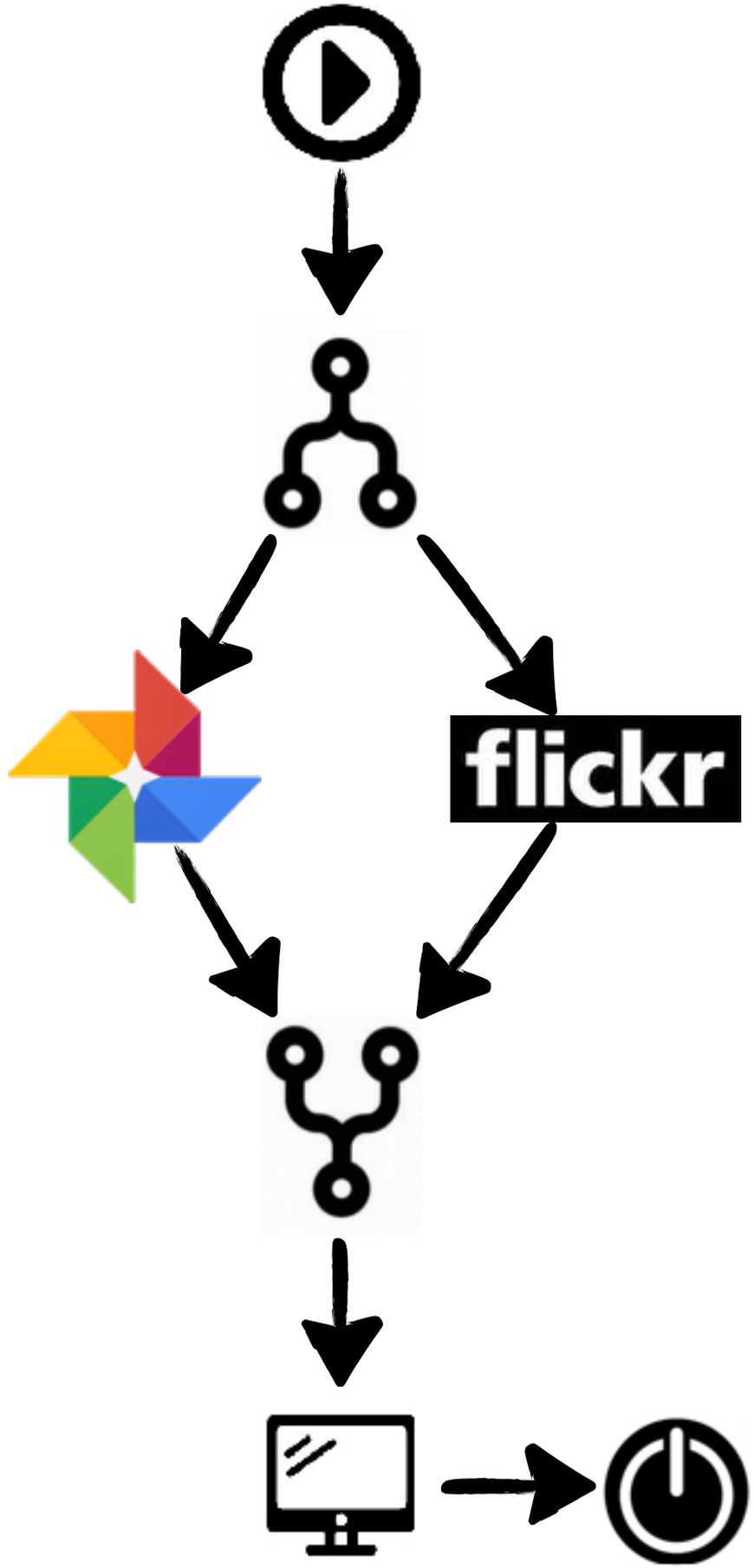
Compose



+



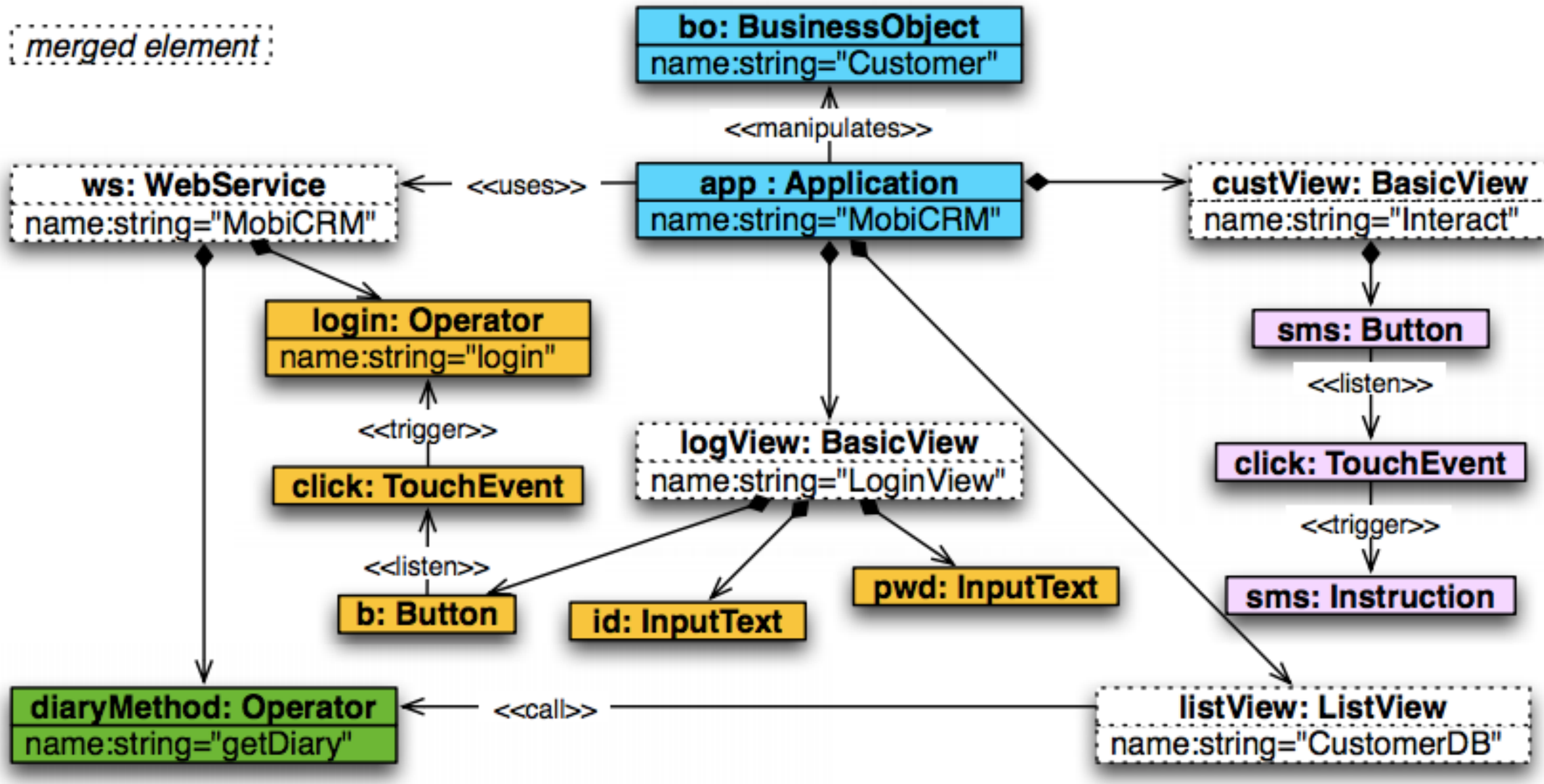
=



Separate

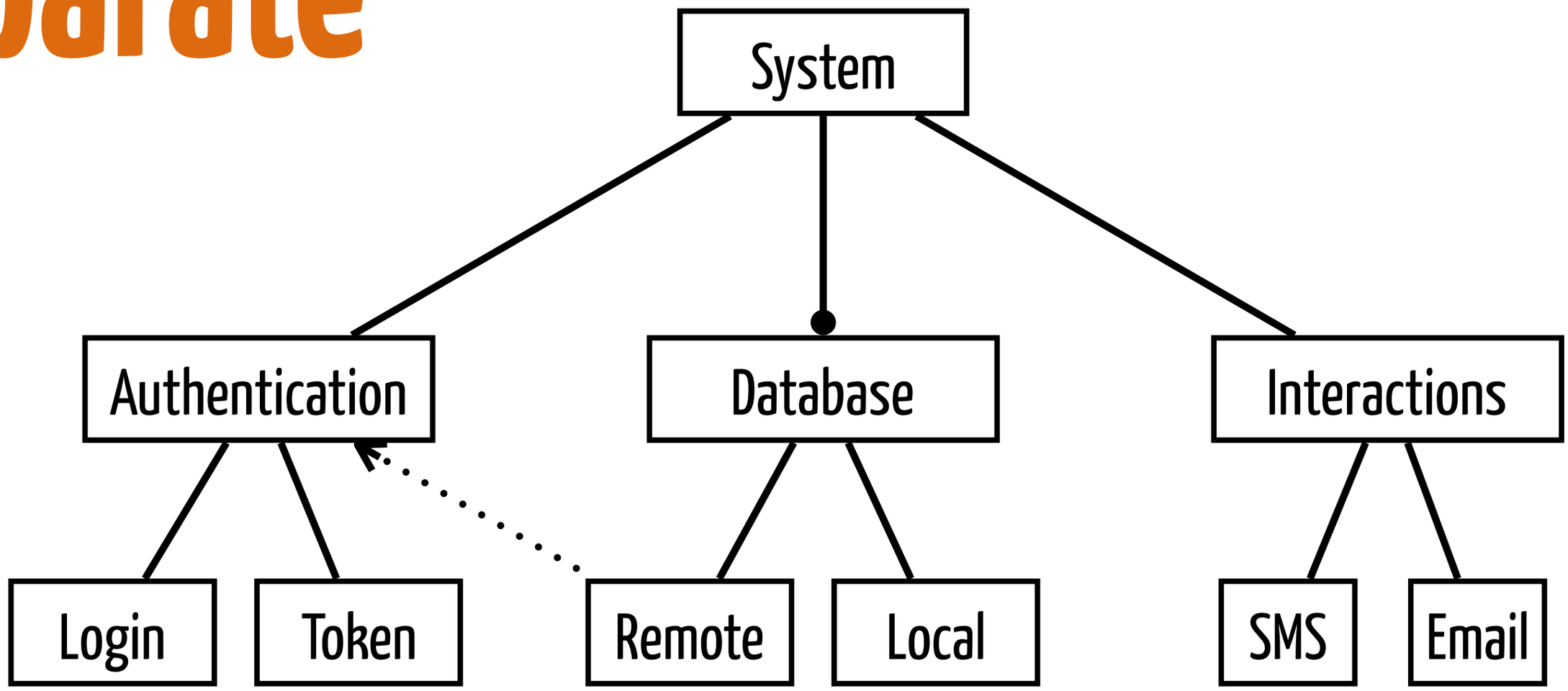
Generate

Separate



Identify preoccupations

Separate



$S \equiv (remote \vee local) \wedge (login \vee token \vee SMS \vee Email) \wedge (remote \Rightarrow (login \vee token))$

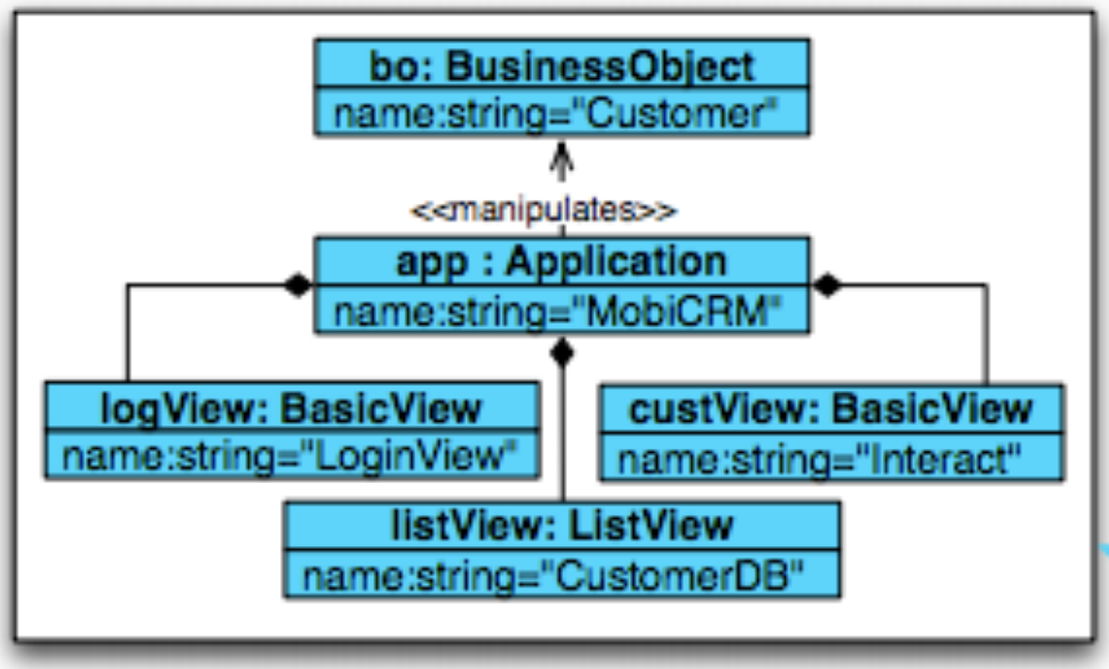
Variability modelling

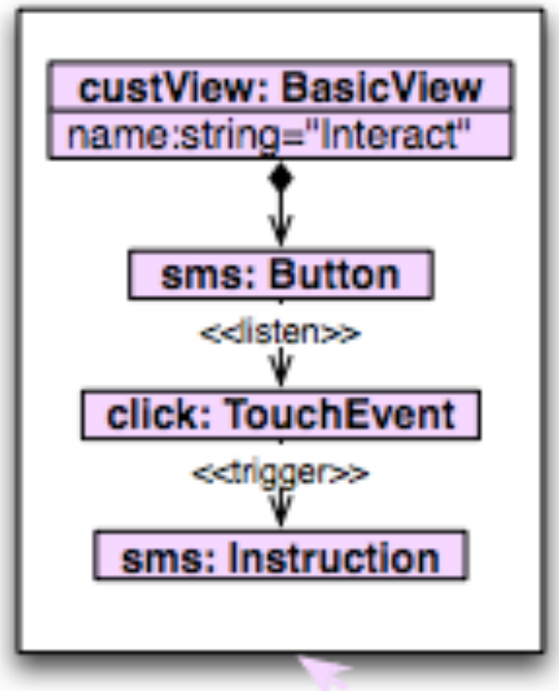
Formula SAT-isfiability?

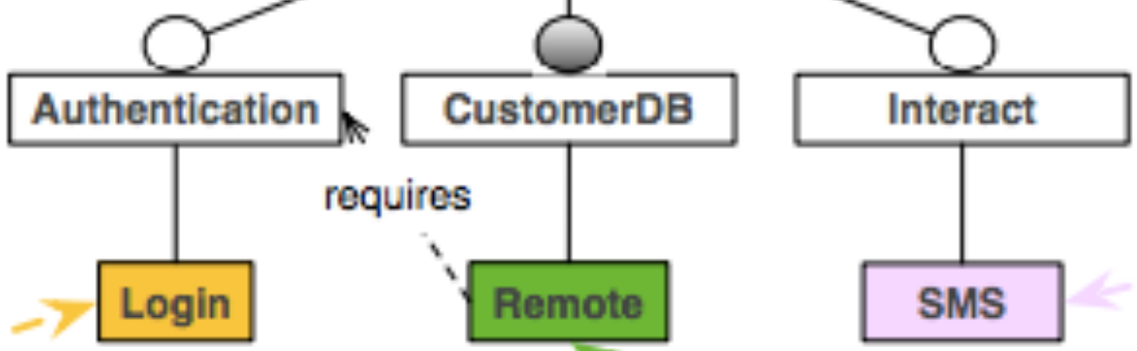
Consistency analysis

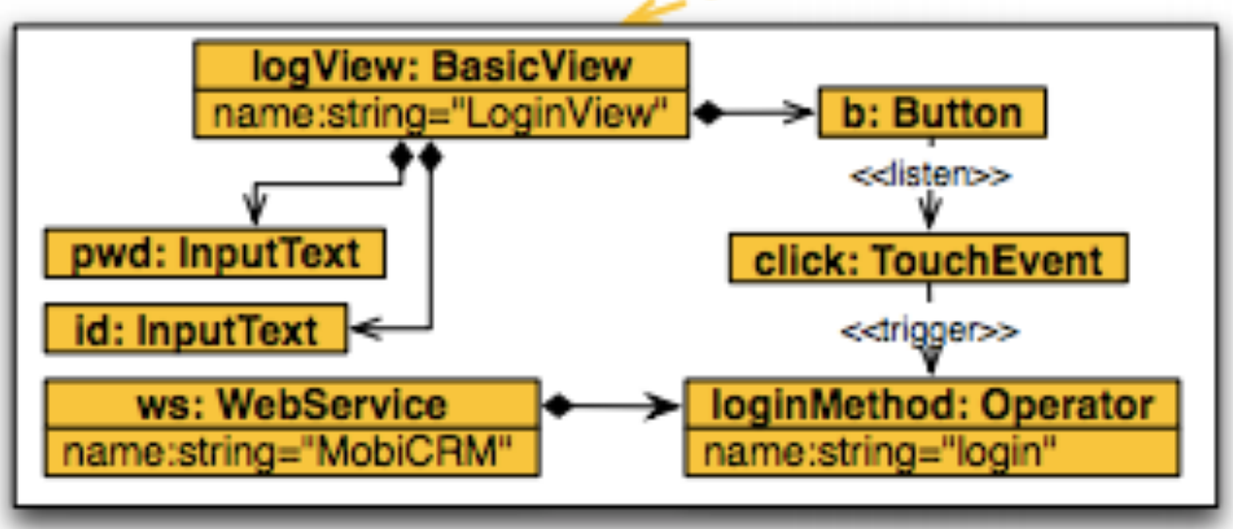
Separate

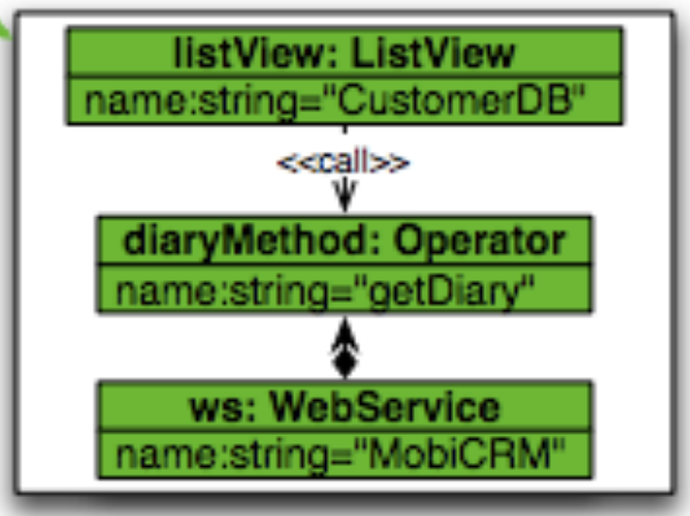
Develop “fragments”







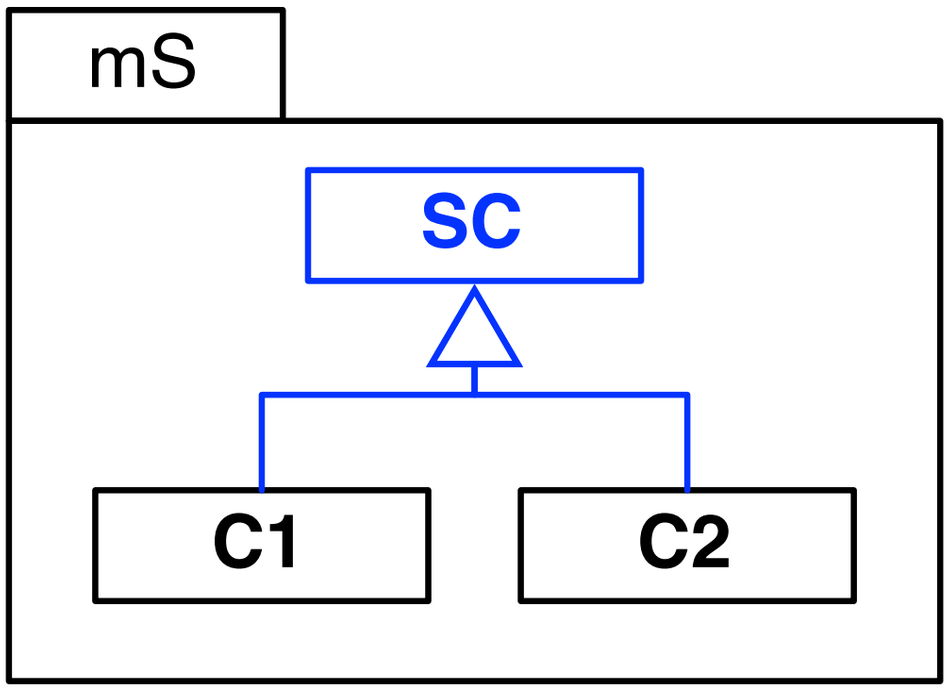


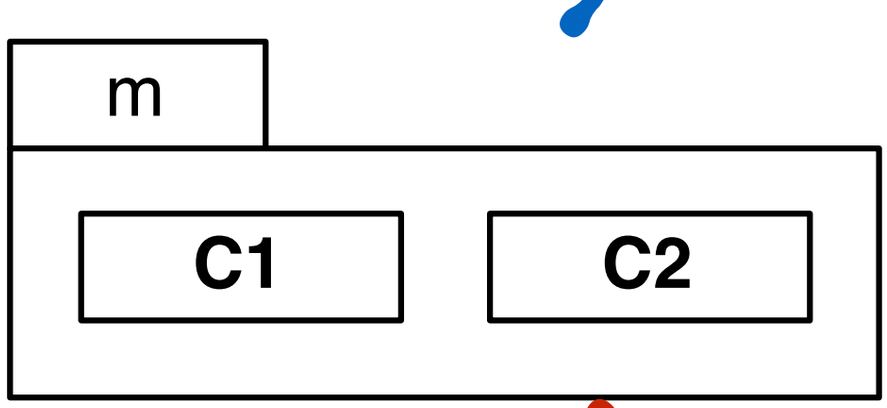


Compose

introduce a Visitor

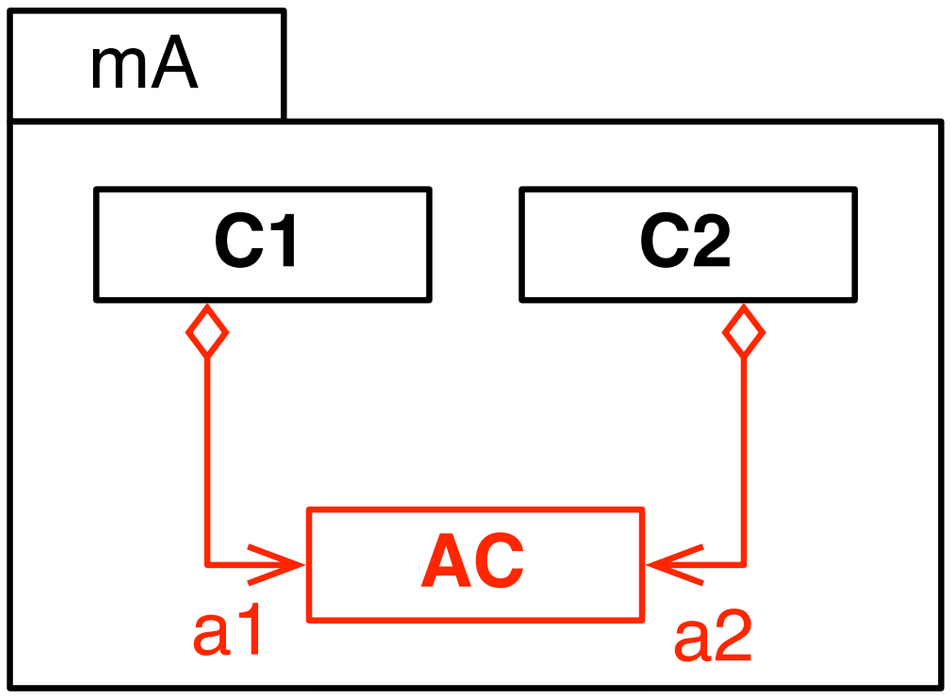
mS = V(m)



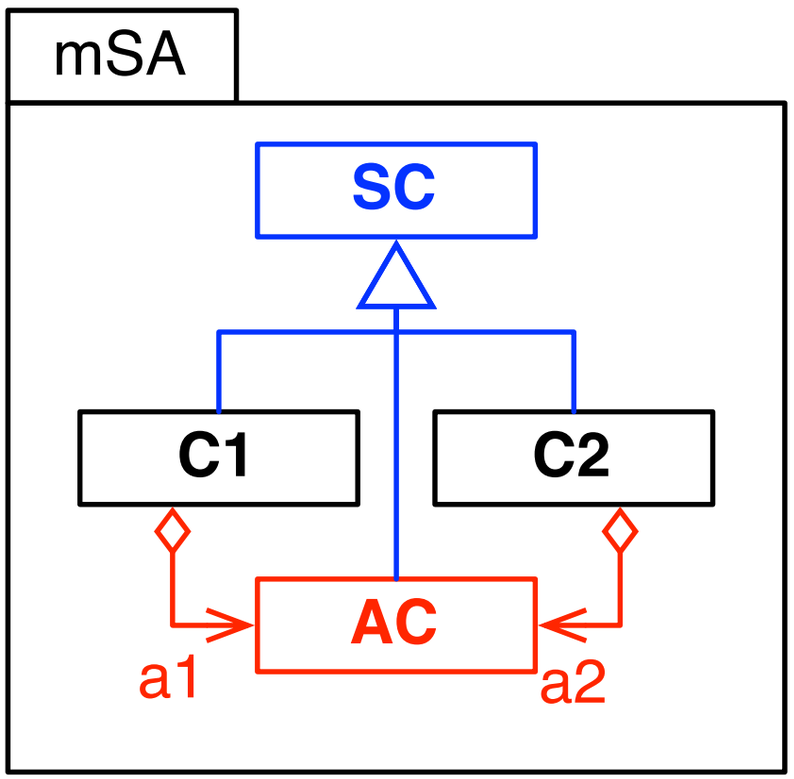


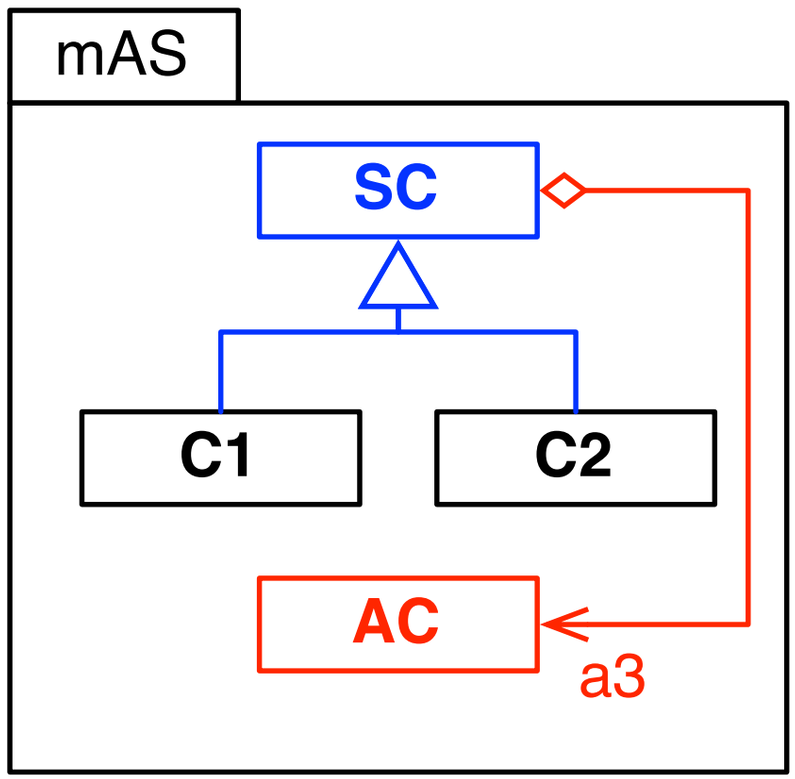
mA = B(m)

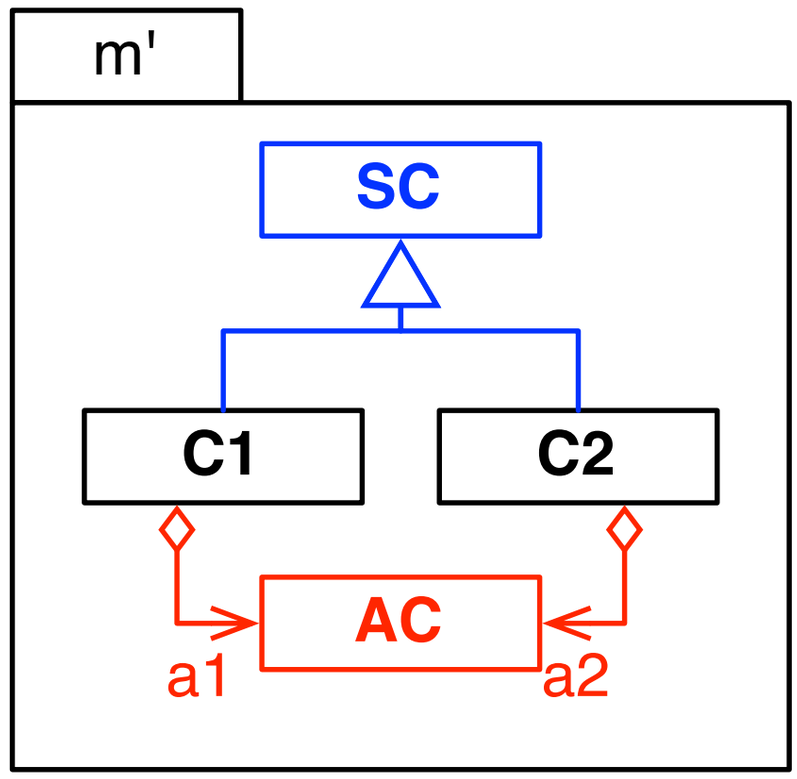
connect to a database



Compose







mSA = V(B(m))

mAS = B(V(m))

m' = (B || V)(m)

$(V \cdot B) \neq (B \cdot V)$

$(V \,||\, B) = (B \,||\, V)$

Composition properties

Operators compositions

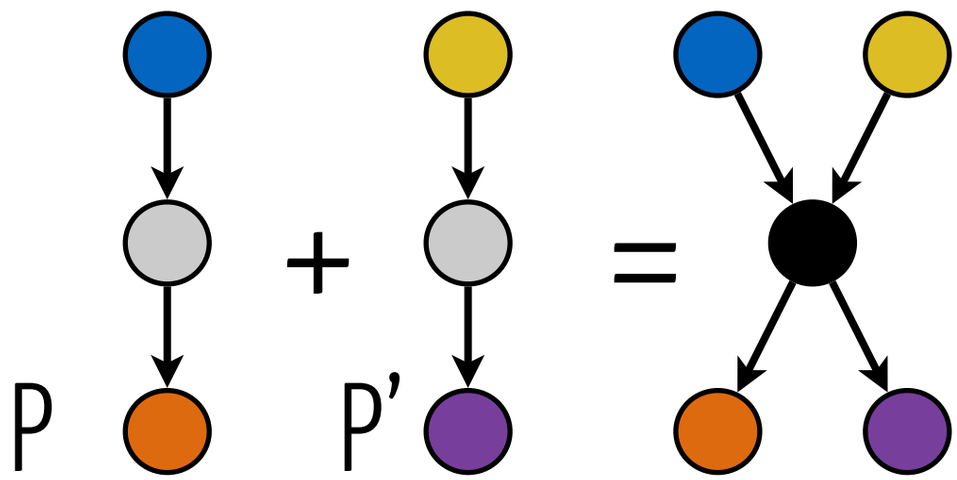
Compose

Algorithm 5 MERGE: $\Phi \times v \times f \mapsto actions$

Require: $\Phi = \{f_1, \ldots, f_n\} \in \mathcal{F}^\star, (v : \mathcal{V}^\star \to \mathbb{B}) \in Function, f \in GroundTerm, \nexists \pi \in \mathcal{P}, name(\pi) = f$
Ensure: $acts \in Actions^\star_<$

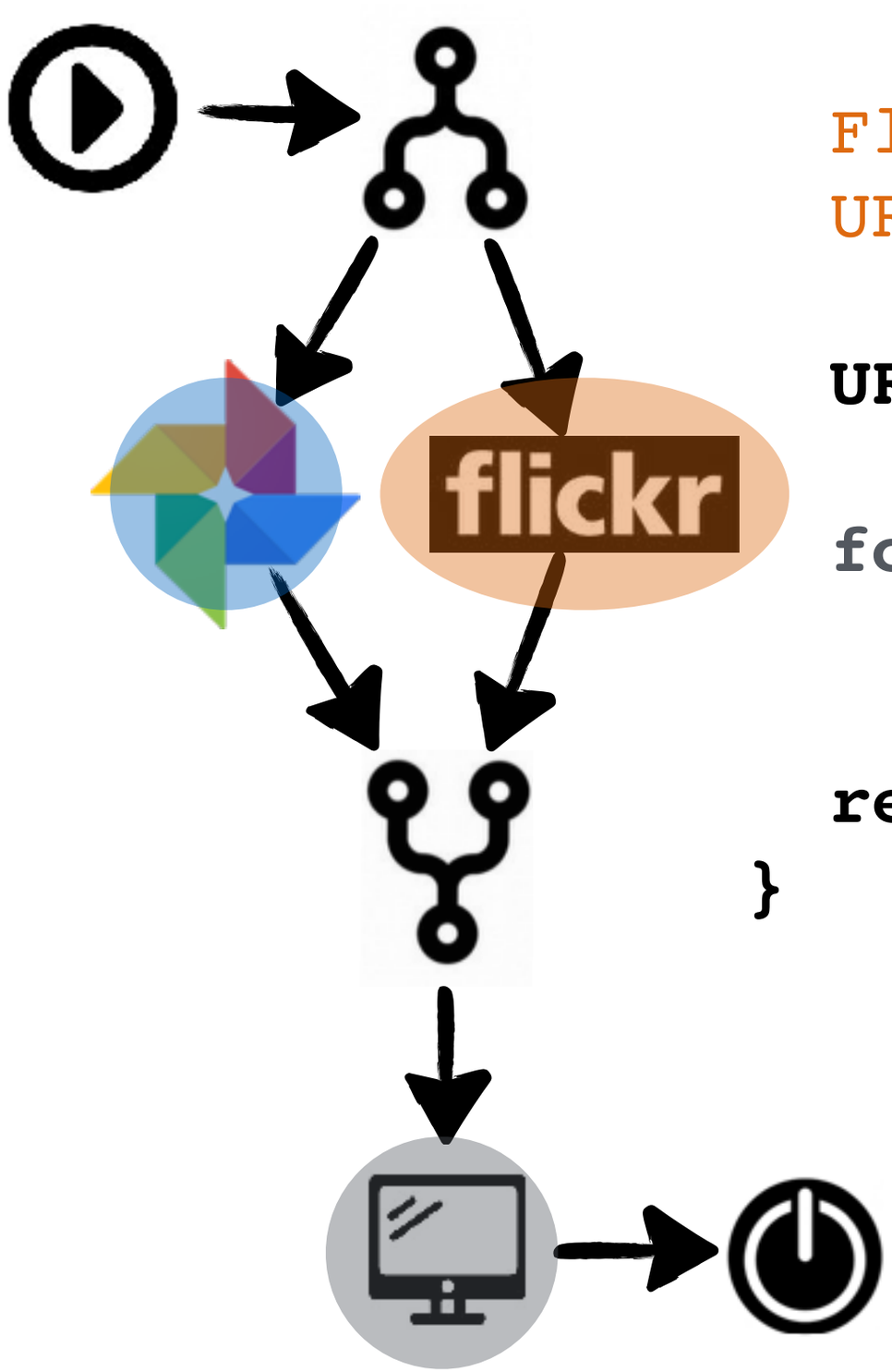
1: $ctx \leftarrow context(), acts \leftarrow (add_p(f, \emptyset, \emptyset, \emptyset)) \cup \{d(\phi, f) \mid \phi \in \Phi\}$ { *Initialization* }
2: { *Fragment Activities Unification* }
3: $P \leftarrow \{\pi \mid \phi \in \Phi,\ \pi = \mathbb{P}(\phi)\}, S \leftarrow \{\sigma \mid \phi \in \Phi,\ \sigma = \mathbb{S}(\phi)\}, H \leftarrow \{h \mid \phi \in \Phi,\ h = hook(\phi)\},$
4: $acts \xleftarrow{+} (u(P, gensym(P, ctx), f),\ u(S, gensym(S, ctx), f),\ u(H, gensym(H, ctx), f))$
5: { *Ghost Variables Unification* }
6: $V_{in} \leftarrow \{v \mid \exists \phi \in \Phi,\ v \in inputs(hook(\phi))\},\ V_{out} \leftarrow \{v \mid \exists \phi \in \Phi,\ v \in outputs(hook(\phi))\}$
7: $acts \xleftarrow{+} \{u(V, gensym(V), f) \mid \exists V \subseteq V_{in},\ v(V)\} \cup \{u(V, gensym(V), f) \mid \exists V \subseteq V_{out},\ v(V)\}$
8: **return** *acts* { *Exit* }

e.g., Graph Theory



[copy(P), copy(P'), create(●),
del(●→●), del(●→●),
del(●→●), del(●→●),
add(●→●), add(●→●),
add(●→●), add(●→●)]

Generate



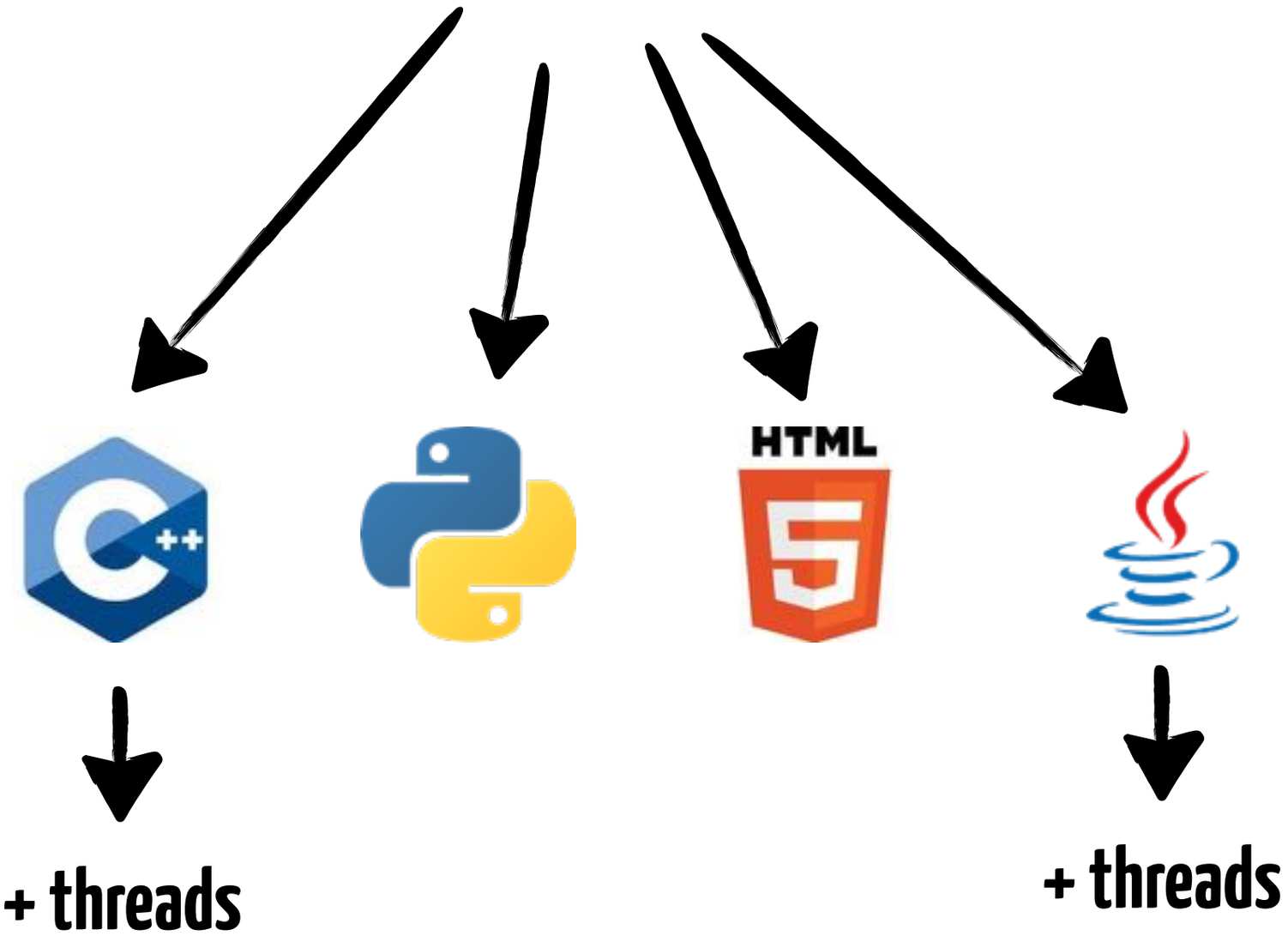

```
public void run() {
    GooglePhoto p = new GooglePhoto(…);
    URL[] ps = p.getLastPicts();

    Flickr f = new Flickr(…);
    URL[] fs = f.feed(f.getUser());

    URL[] picts = Arrays.append(ps,fs);

    for(URL p: picts)
      display(p);

    return;
}
```

AST Visit

Back-end

Compilation

Generate

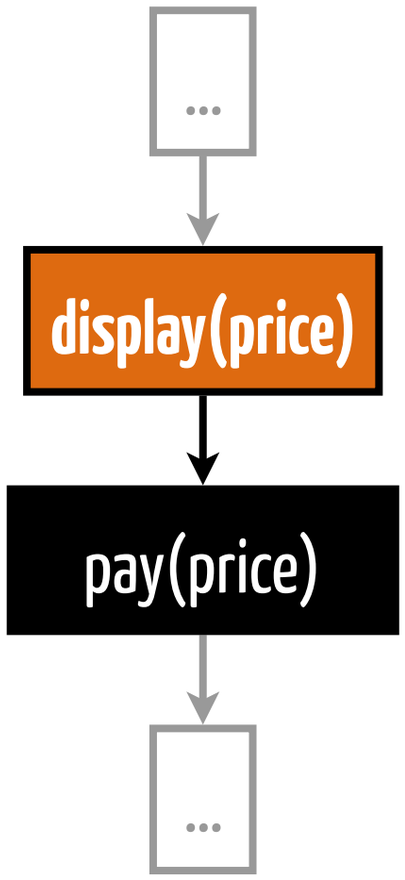


WARNING : PROJECTIONS OF MODELS !

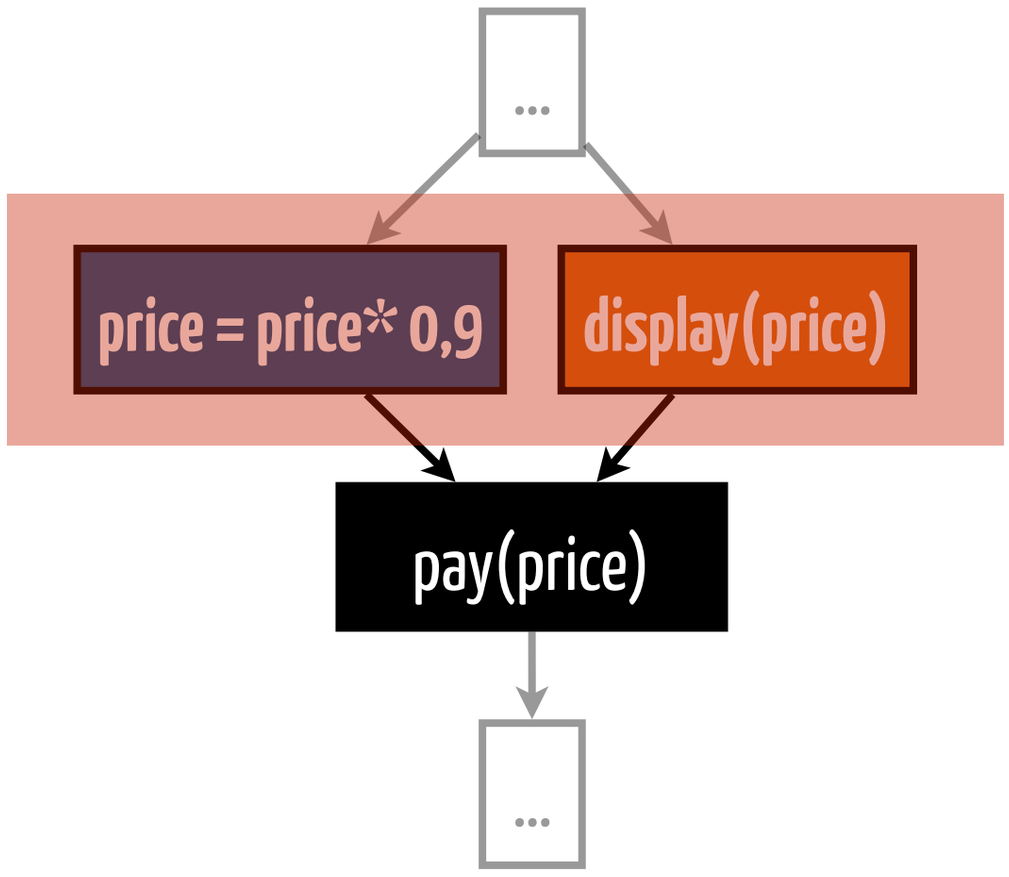
Generate

Conflict!!

+



=



Graph Theory

Flows

Critical Pair Analysis

$$\otimes : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{B}$$

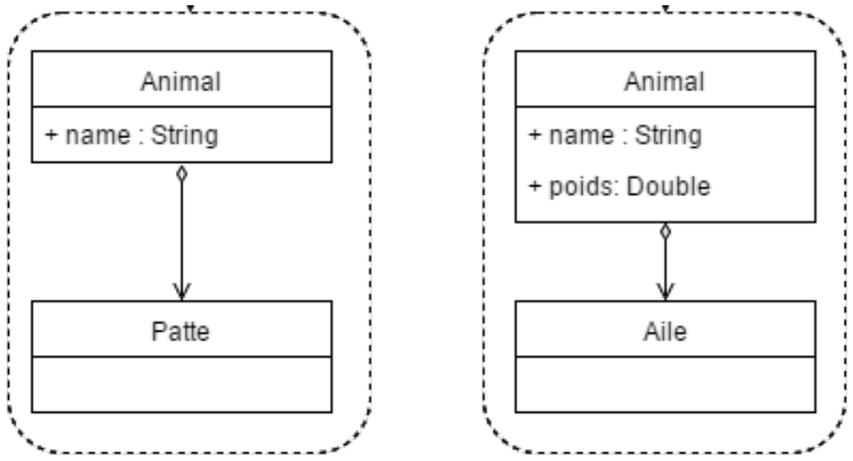
$$(a, a') \mapsto \begin{cases} \textbf{Let } p \in \mathcal{P}, e = entry(p), P_a = path^+(rels(p), e, a), P_{a'} = path^+(rels(p), e, a') \\ a \neq a', \exists \pi_a \in P_a, \exists \alpha \in \pi_a, \exists \pi_{a'} \in P_{a'}, \exists \alpha' \in \pi_{a'}, \varphi(\alpha) \otimes \varphi(\alpha') \end{cases}$$

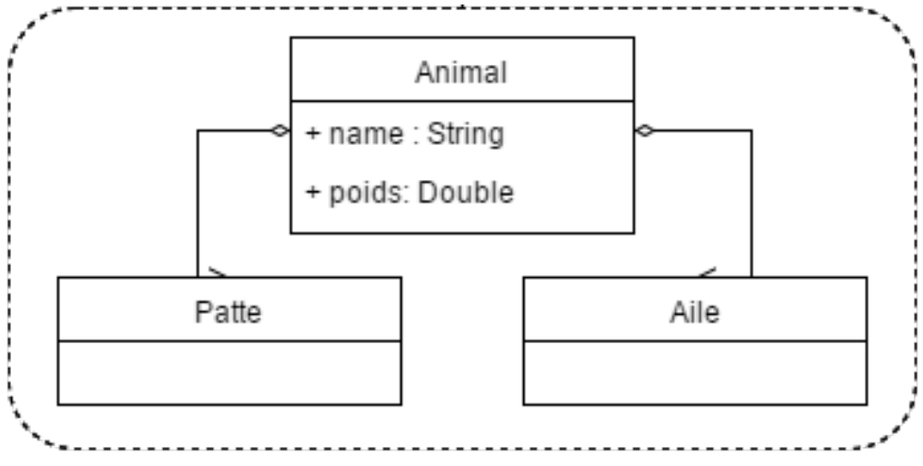
$$\textbf{Let } p \in \mathcal{P},\ \exists a \in acts(p),\ \exists v \in outputs(a),\ \exists a' \in acts(p), a' \neq a, v \in vars(a'),\ \neg(a \otimes a')$$

Composition is everywhere

really :)

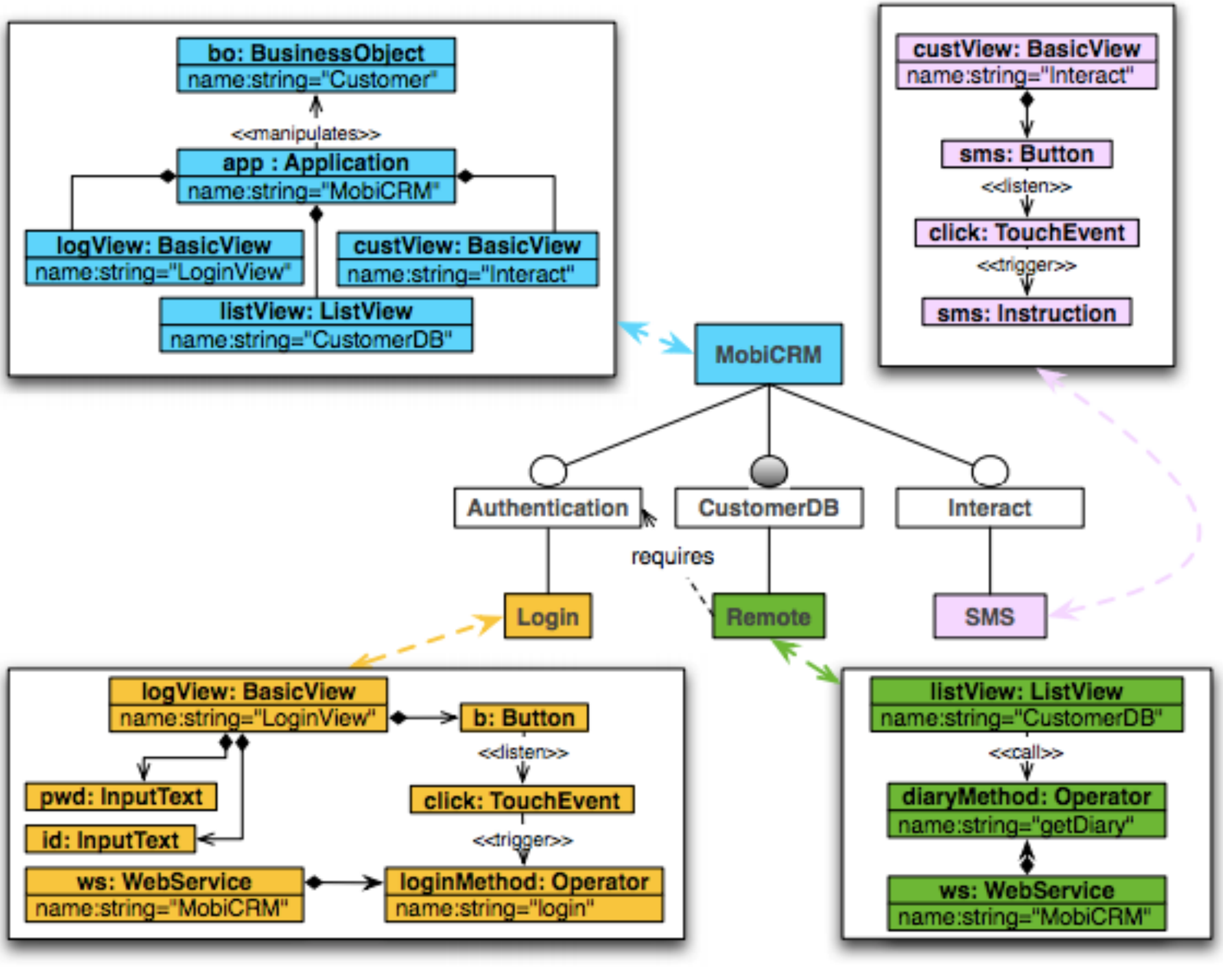
Model composition



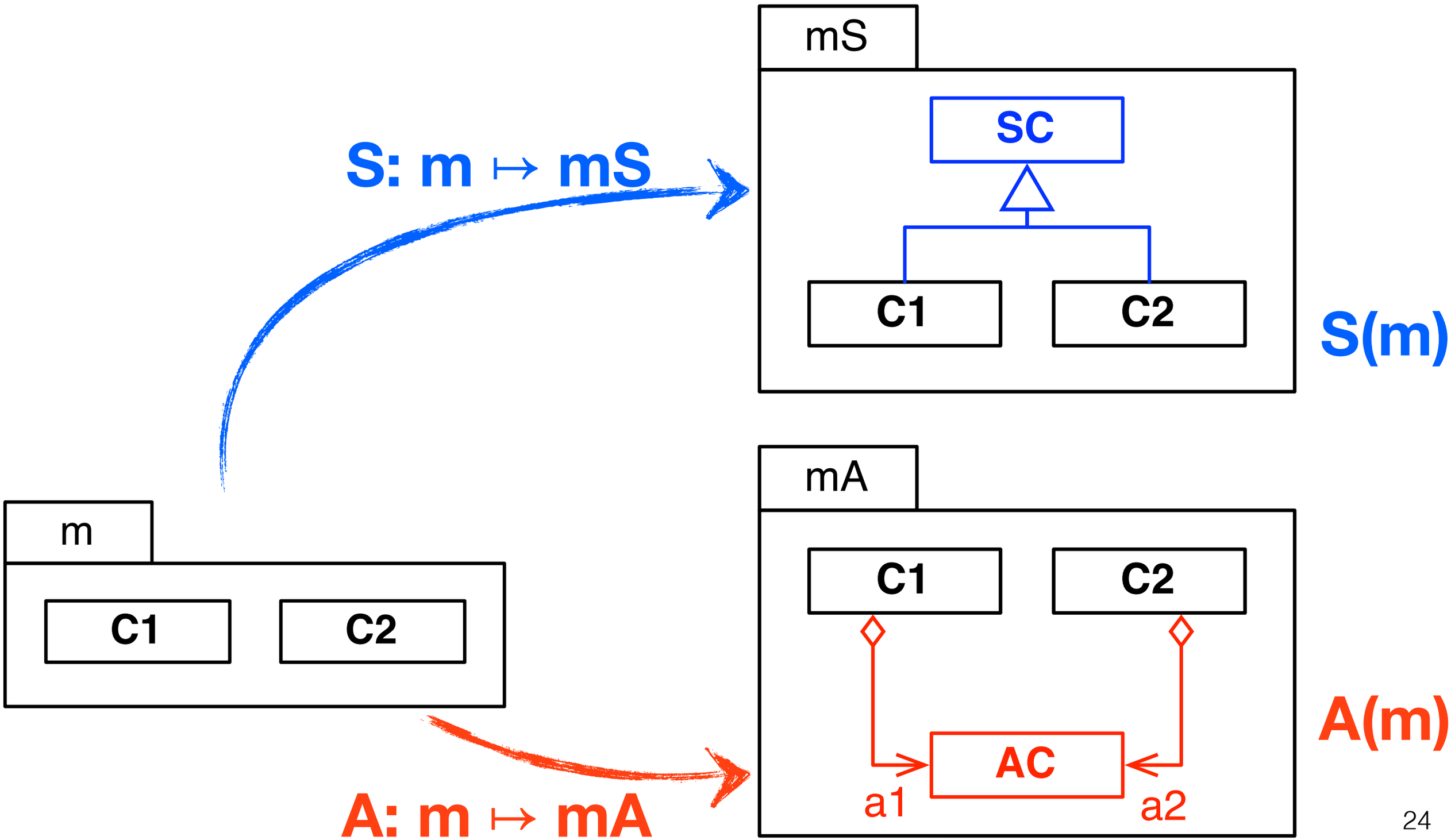


OBJECT MANAGEMENT GROUP

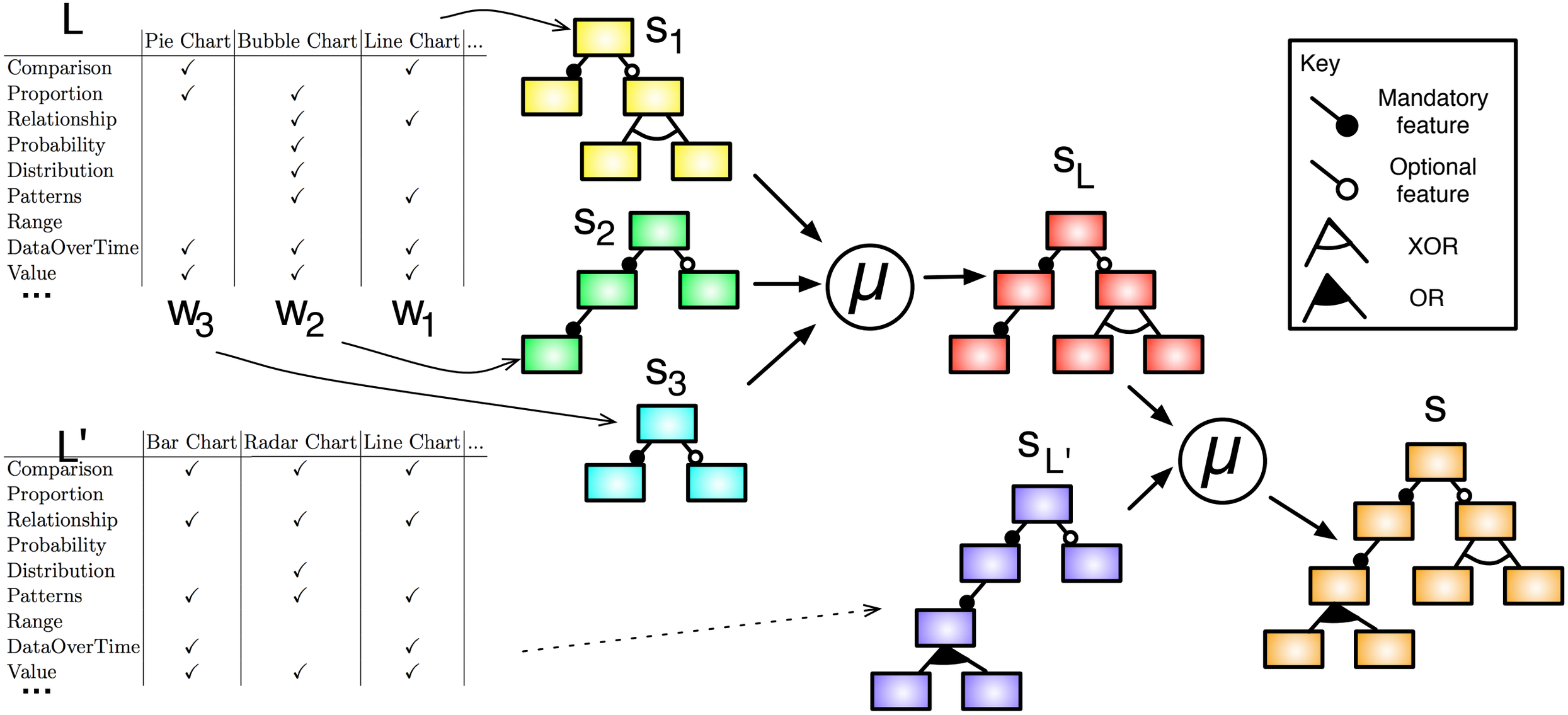
“Product-line” software



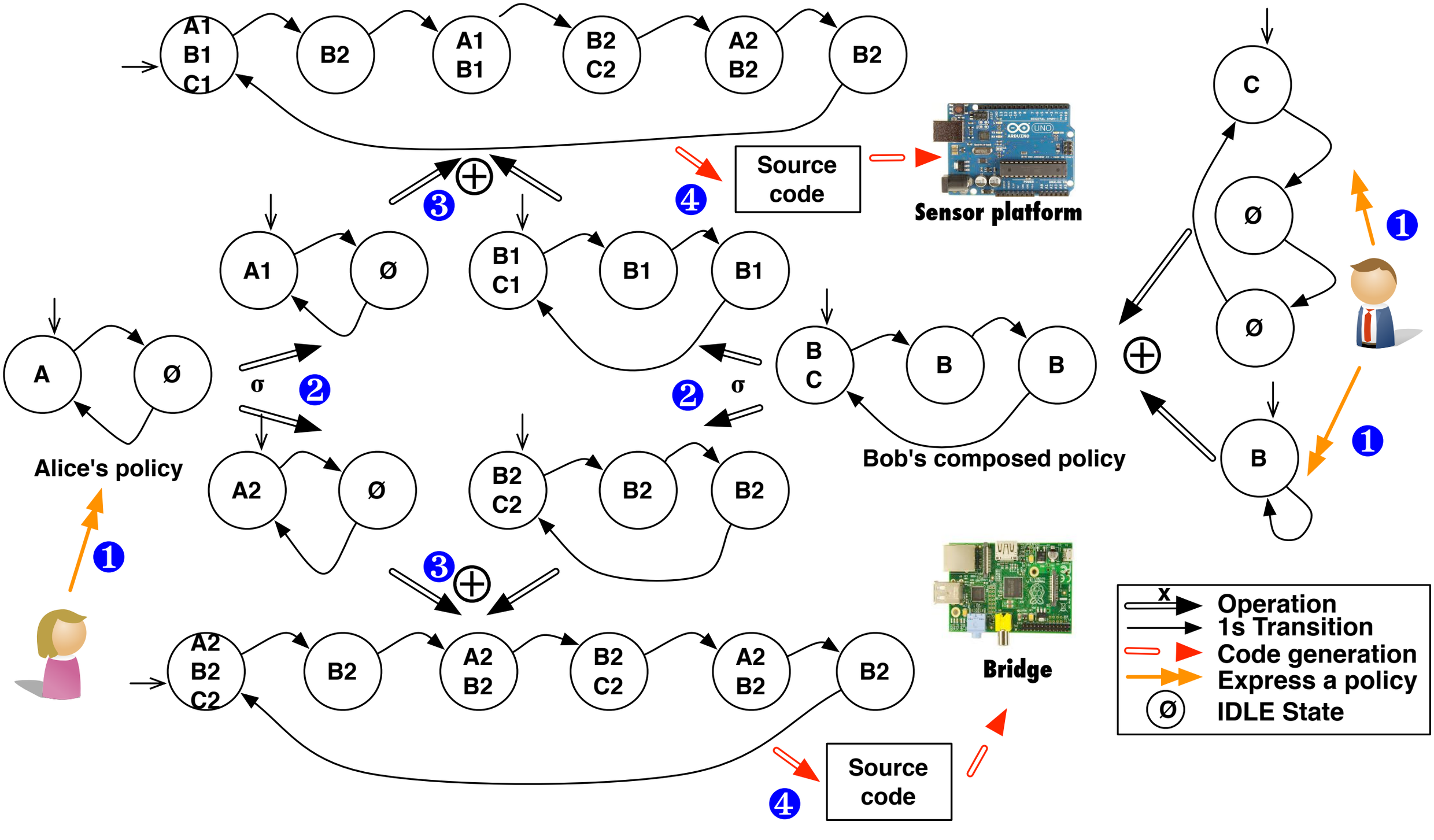
Model Transformation



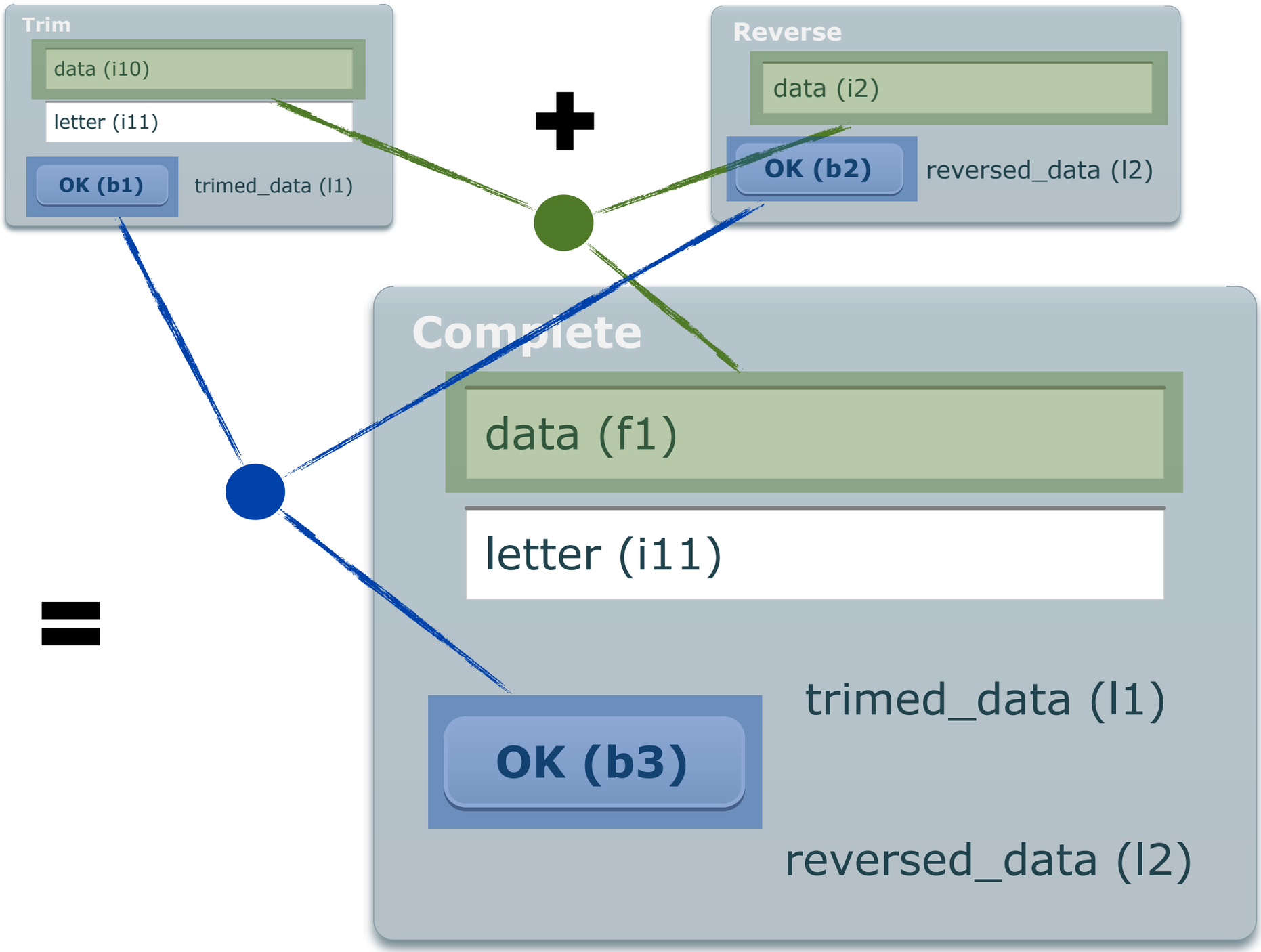
Catalogue Construction



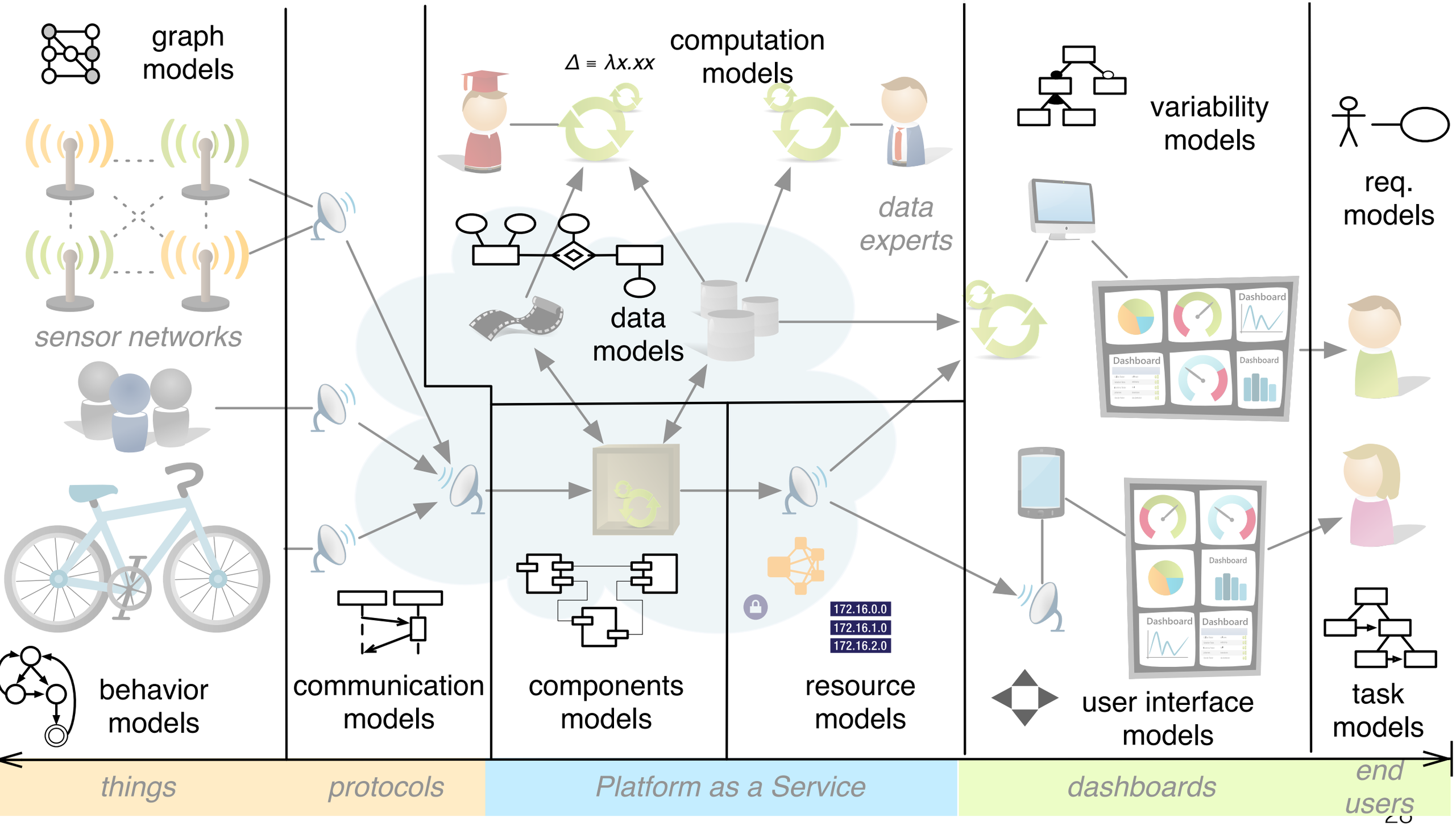
Cyber-Physical Systems deployment



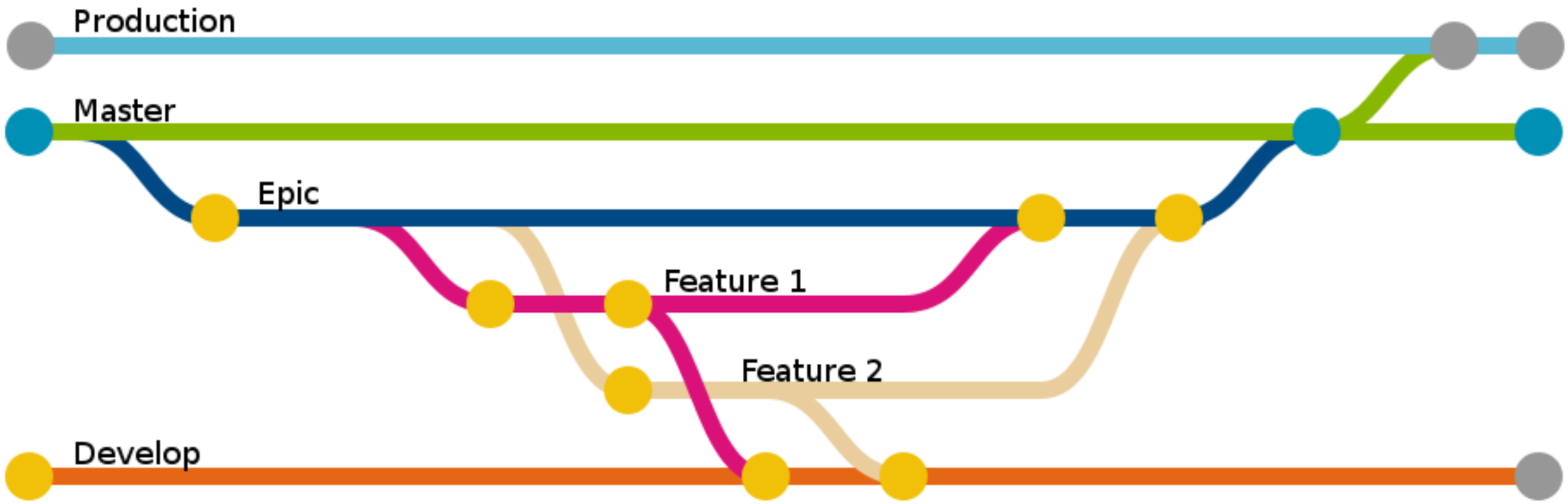
GUI Composition



Language Composition



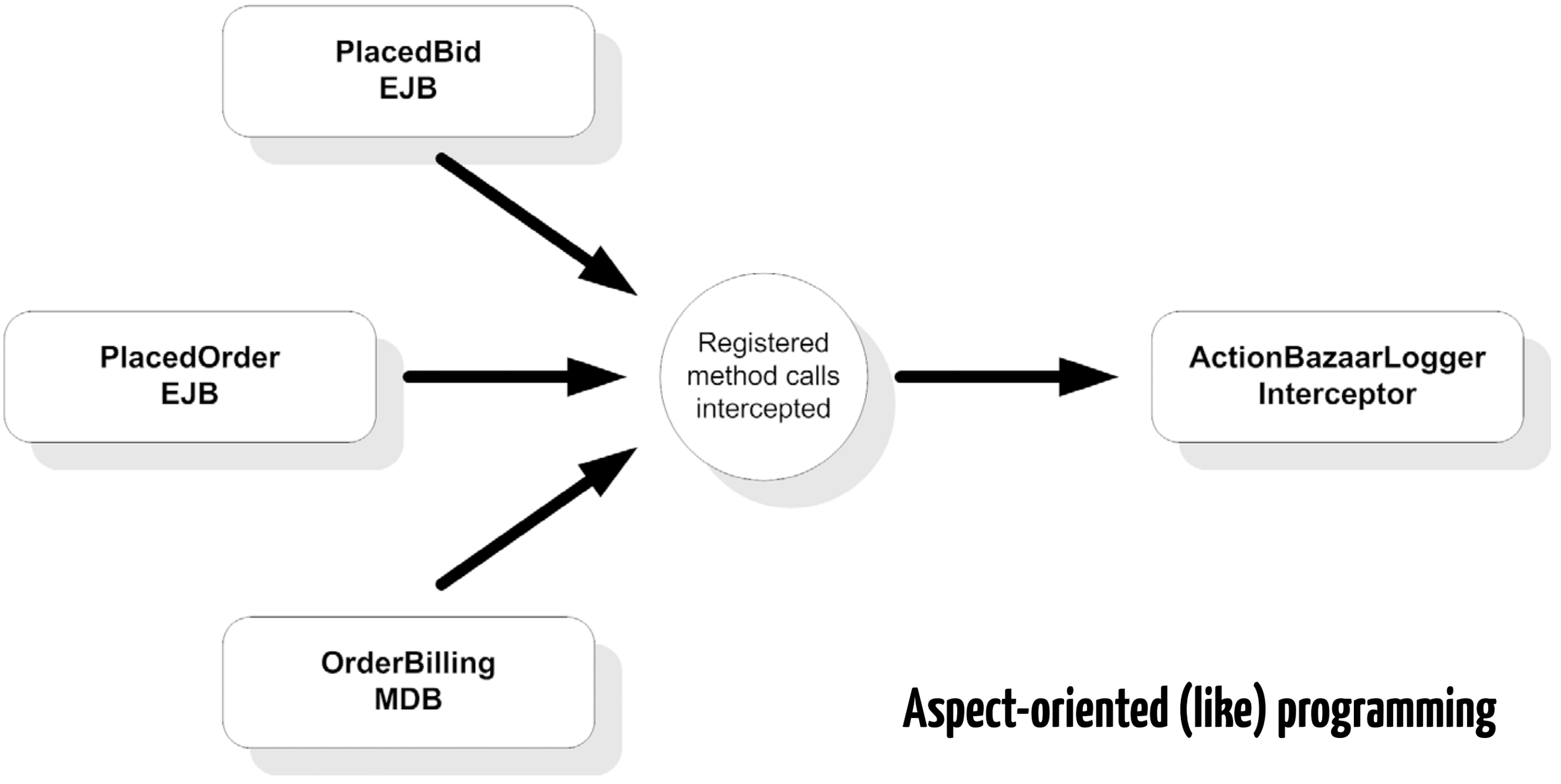
Merging code



4 client/src/main/java/fr/labri/gumtree/client/Run.java View

```
@@ -34,7 +34,7 @@ protected void process(String name, String[] args) {
        }
    }

-   static void initGenerators() {
+   public static void initGenerators() {
        Reflections reflections = new Reflections("fr.labri.gumtree.gen");

        reflections.getSubTypesOf(TreeGenerator.class).forEach(
@@ -44,7 +44,7 @@ static void initGenerators() {
            });
    }

-   static void initClients() {
+   public static void initClients() {
        Reflections reflections = new Reflections("fr.labri.gumtree.client");

        reflections.getSubTypesOf(Client.class).forEach(
```

EJB (J2E) Interception Mechanisms

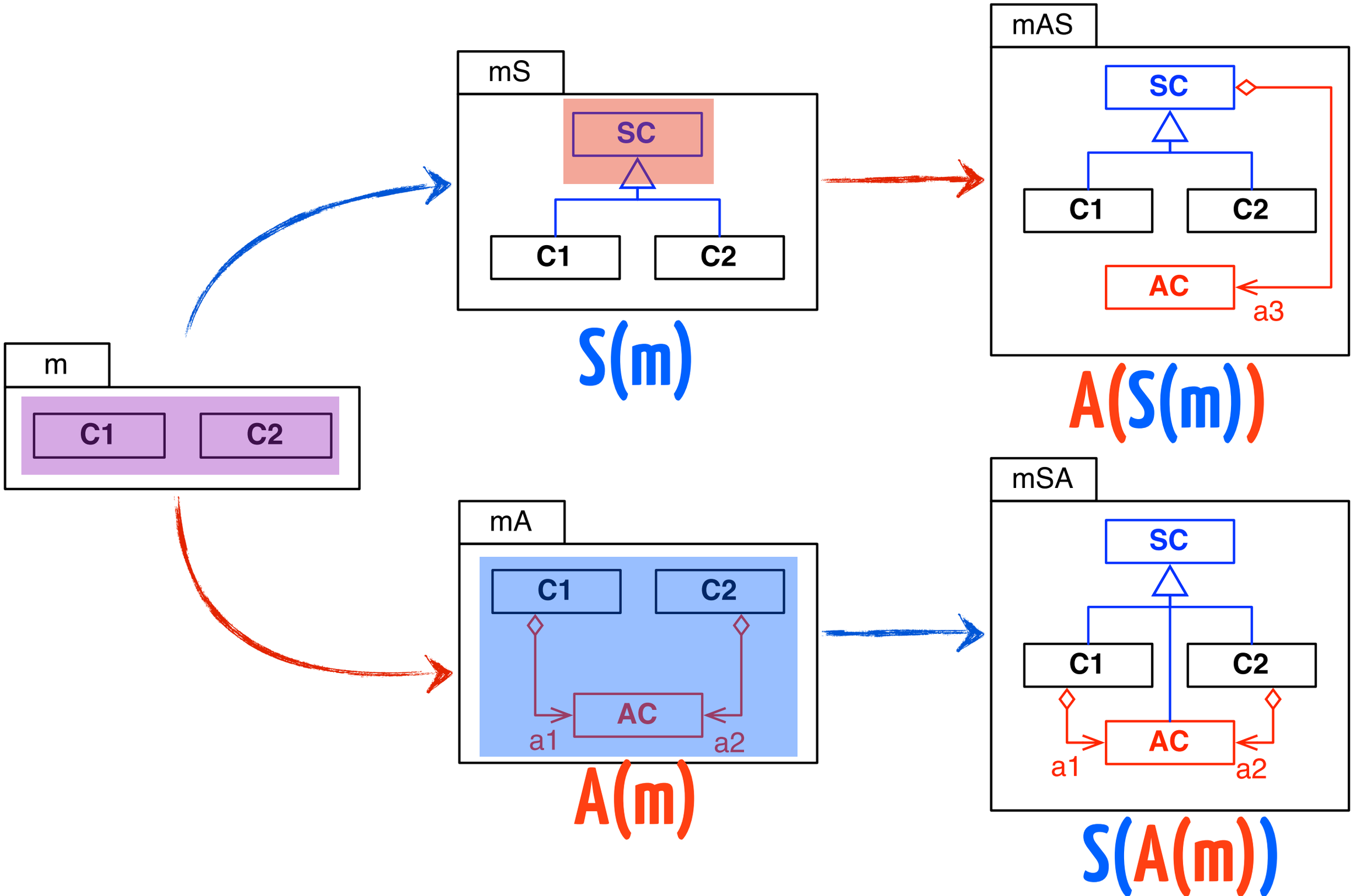


Aspect-oriented (like) programming

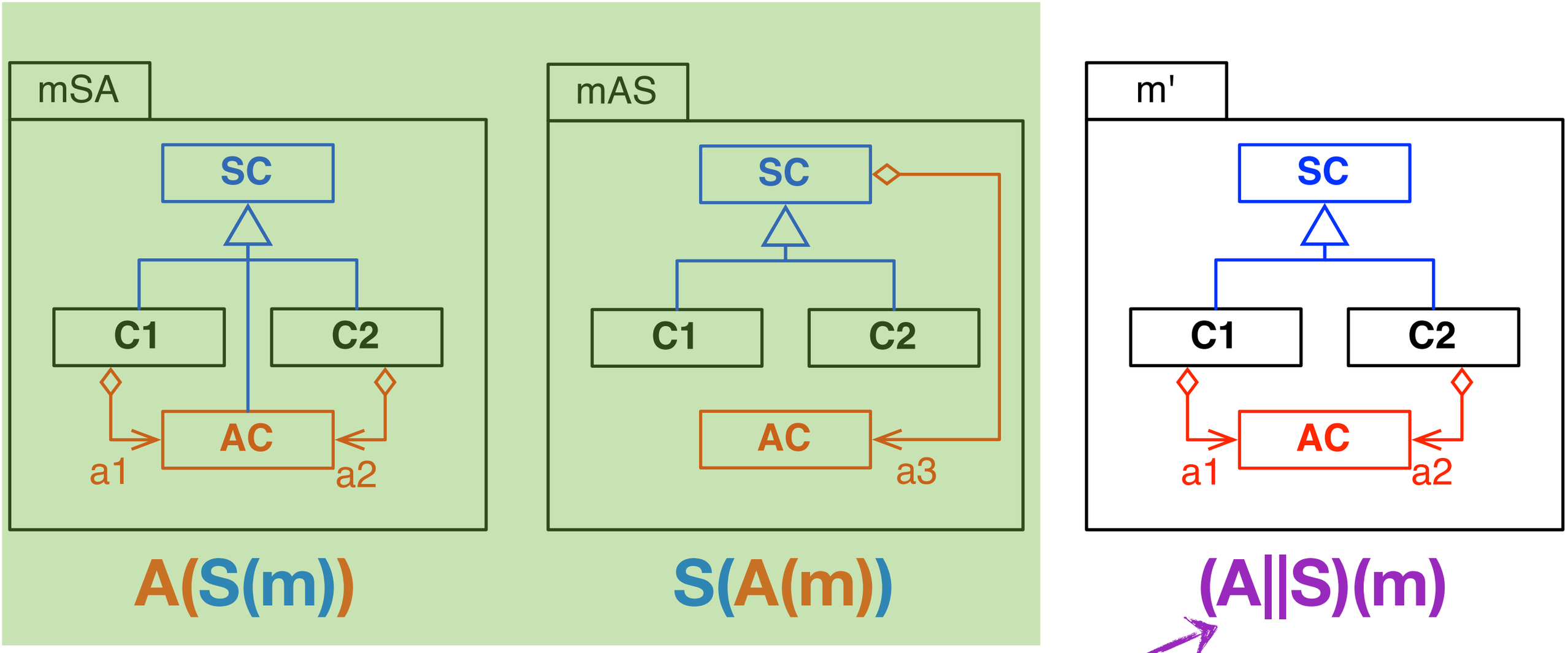
Creating a composition operator

Targeted property: commutativity

Problem: {A,S} selected simultaneously



The problem is not «**interactions**» or «**ordering**»

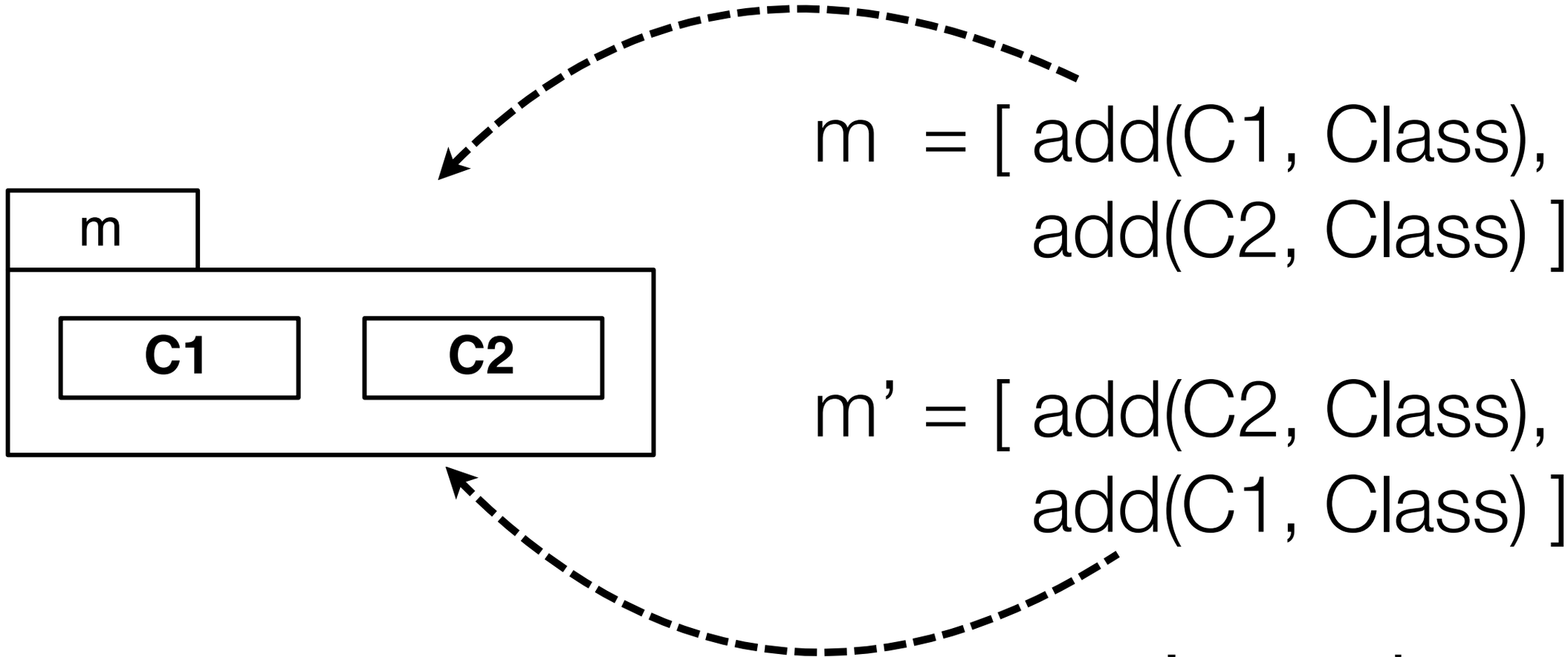


e.g., critical pair analysis

We need a «commutative composition operator»

From Action Sequences ...

«Every **model** can be expressed as a **sequence** of elementary construction **operation**» *[Blanc et al, ICSE'08]*



… to Action Sets

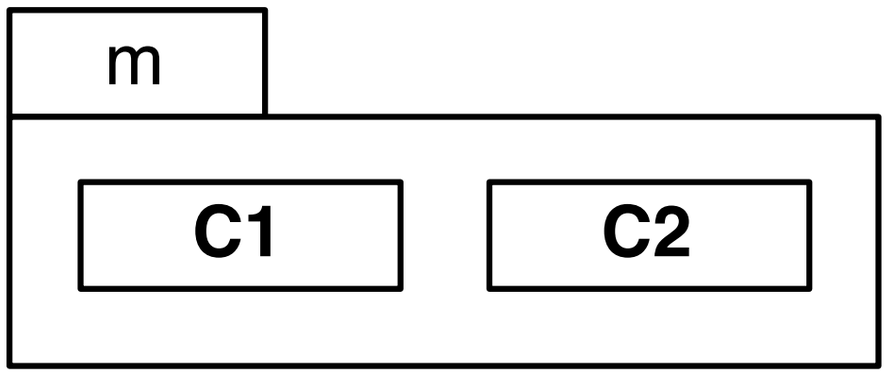
Added relations

Deleted relations

$$m = (A_n, A_r, D_n, D_r)$$

Added nodes

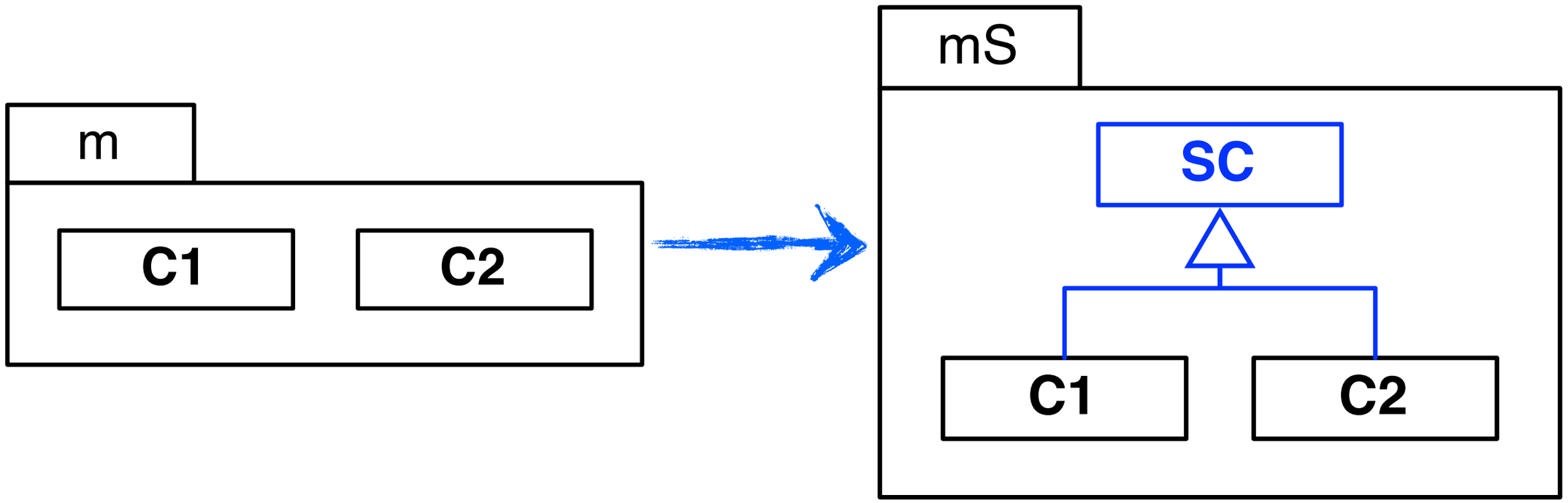
Deleted nodes



$$m = (\{ add(C1, Class), add(C2, Class)\}, \emptyset, \emptyset, \emptyset)$$

$$m \cup m' = (A_n \cup A'_n, A_r \cup A'_r, D_n \cup D'_n, D_r \cup D'_r)$$

Relation with Evolution Functions



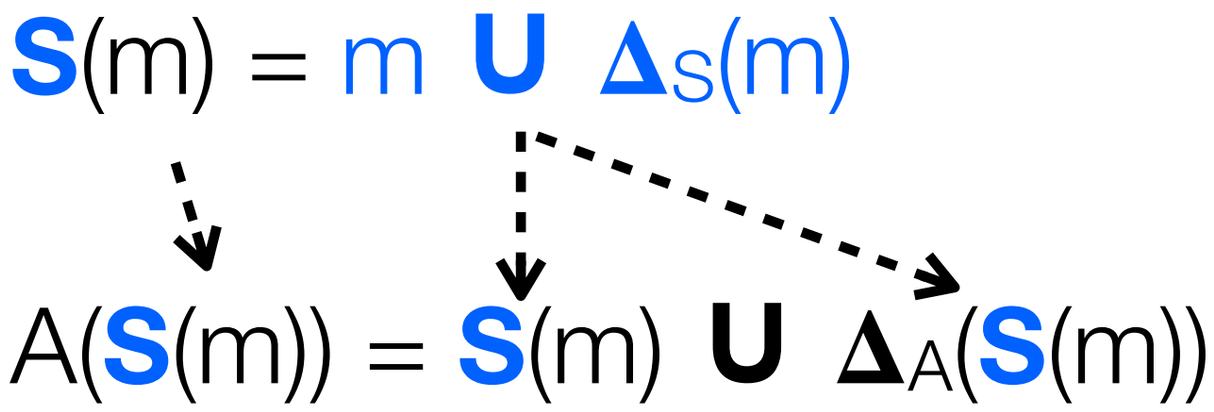
$mS = S(m) = m \cup \Delta_S(m)$

$= (\ \{add(C1, Class), add(C2, Class)\}, \emptyset, \emptyset, \emptyset)$

$\mathbf{U}\ (\ \{add(SC, Class)\},$

$\{add(C1,SC,In...), add(C2,SC, In...)\}, \emptyset, \emptyset)$

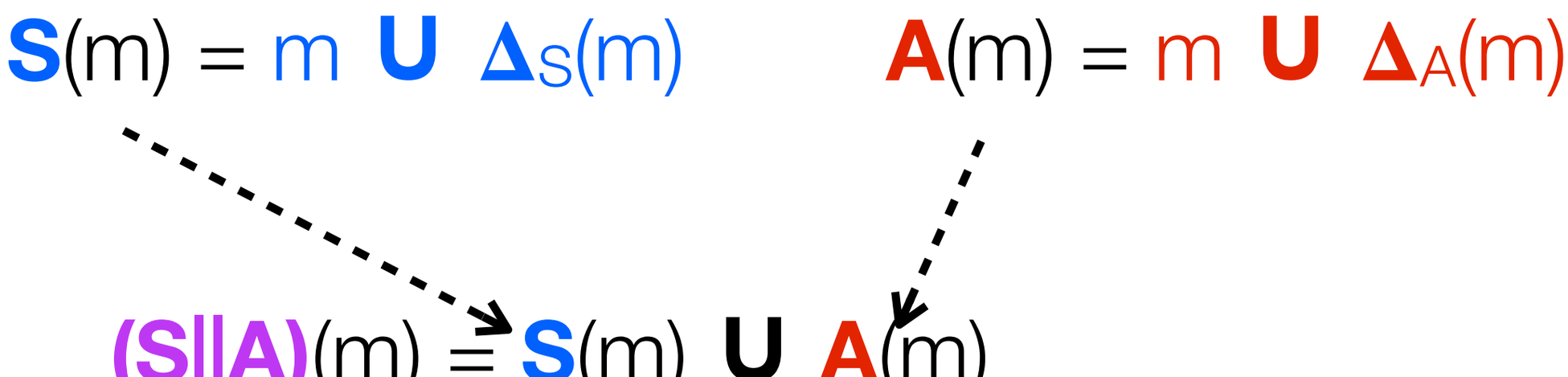
Modelling Sequential Composition



$$= m \cup \Delta_S(m) \cup \Delta_A(m \cup \Delta_S(m))$$

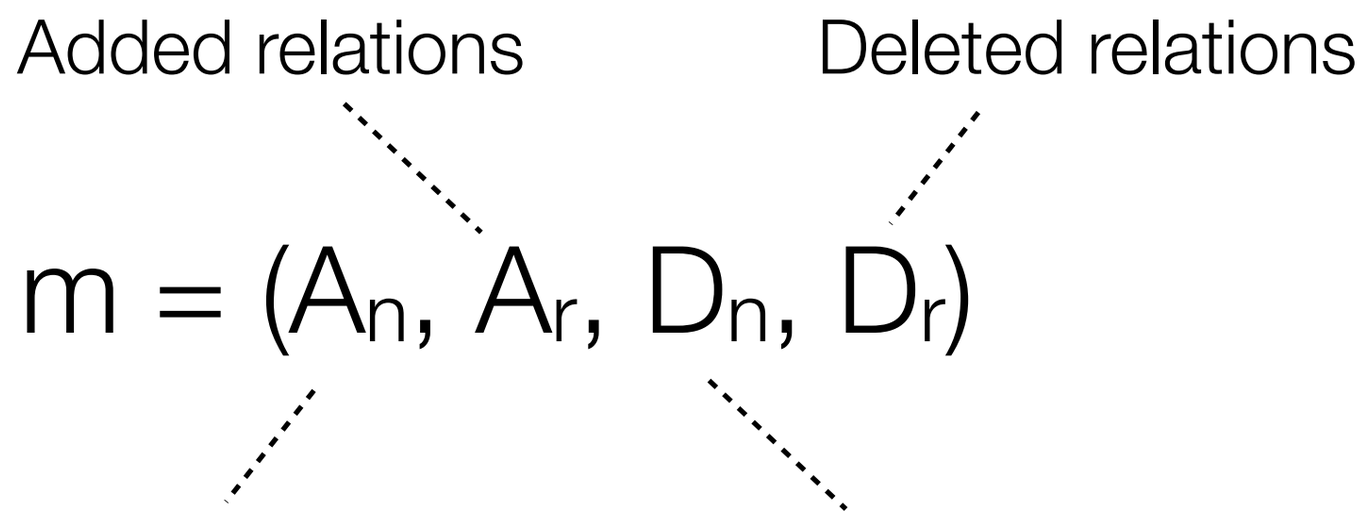
Identity	Id(F(m)) = F(m)	✔
Idempotence	F(F(m)) = F(m)	✘
Commutativity	F(G(m)) = G(F(m))	✘

Modelling Parallel Composition (||)



Identity	Id(F(m)) = F(m)	✔
Idempotence	F(F(m)) = F(m)	✔
Commutativity	F(G(m)) = G(F(m))	✔

Inconsistency detected => Ordering needed!



- **A model «m» is detected as inconsistent** if (among others):
  - a relation added in A_r is defined between elements deleted in D_n
  - a node deleted in D_n is still involved in relations added in A_r
- It identifies **situations where the parallel composition is not suitable** (ordered).

Implementation: A Composition DSL

```
composition ordered(m) {
   a(model: m)    => (output: m_a);
   s(model: m_a) => (output: m_sa);
} => (m_sa);
```

---------- S(A(m))

```
composition parallel(m) {
   s(model: m) => (output: m_p);
   a(model: m) => (output: m_p);
} => (m_p);
```

---------- (A||S)(m)

- Language compiler implemented with ANTLR
- Adaptation functions as Prolog (SWI) predicate (quantifiers, unification)
- Run-time link with Java/EMF through the SWI-PL framework

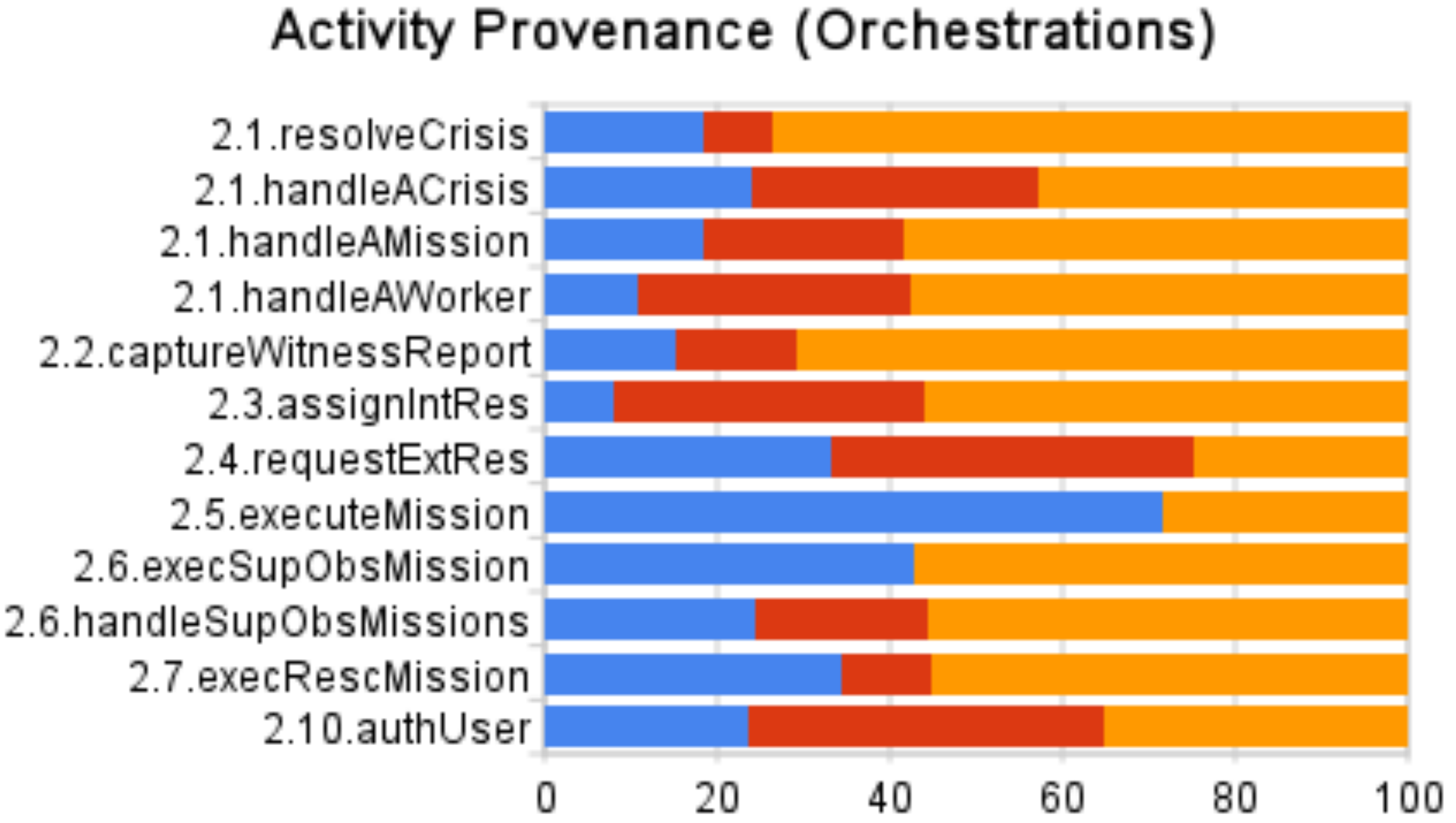
Exploiting a composition operator

CCCMS & Business Processes

The Car Crash Crisis Management System

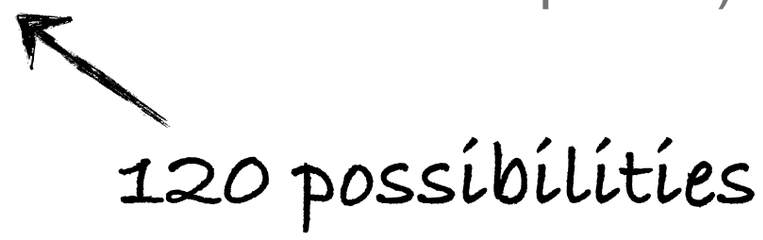
- **Common case study** to compare Aspect-oriented Modelling approaches
  - TAOSD special issue + ECMFA paper (9 approaches described)
- Domain: **Modelling of a Crisis Management System**
  - **8** main success scenario (*e.g., «capture witness report»*)
  - **27** business extensions (*e.g., «interrupted call», «fake information»*)
  - **3** non-functional requirements (*i.e., «security», «persistence», «statistics»*)
- **Huge artefacts** (*hundreds of elements, thousands of relations*)

Detailed results

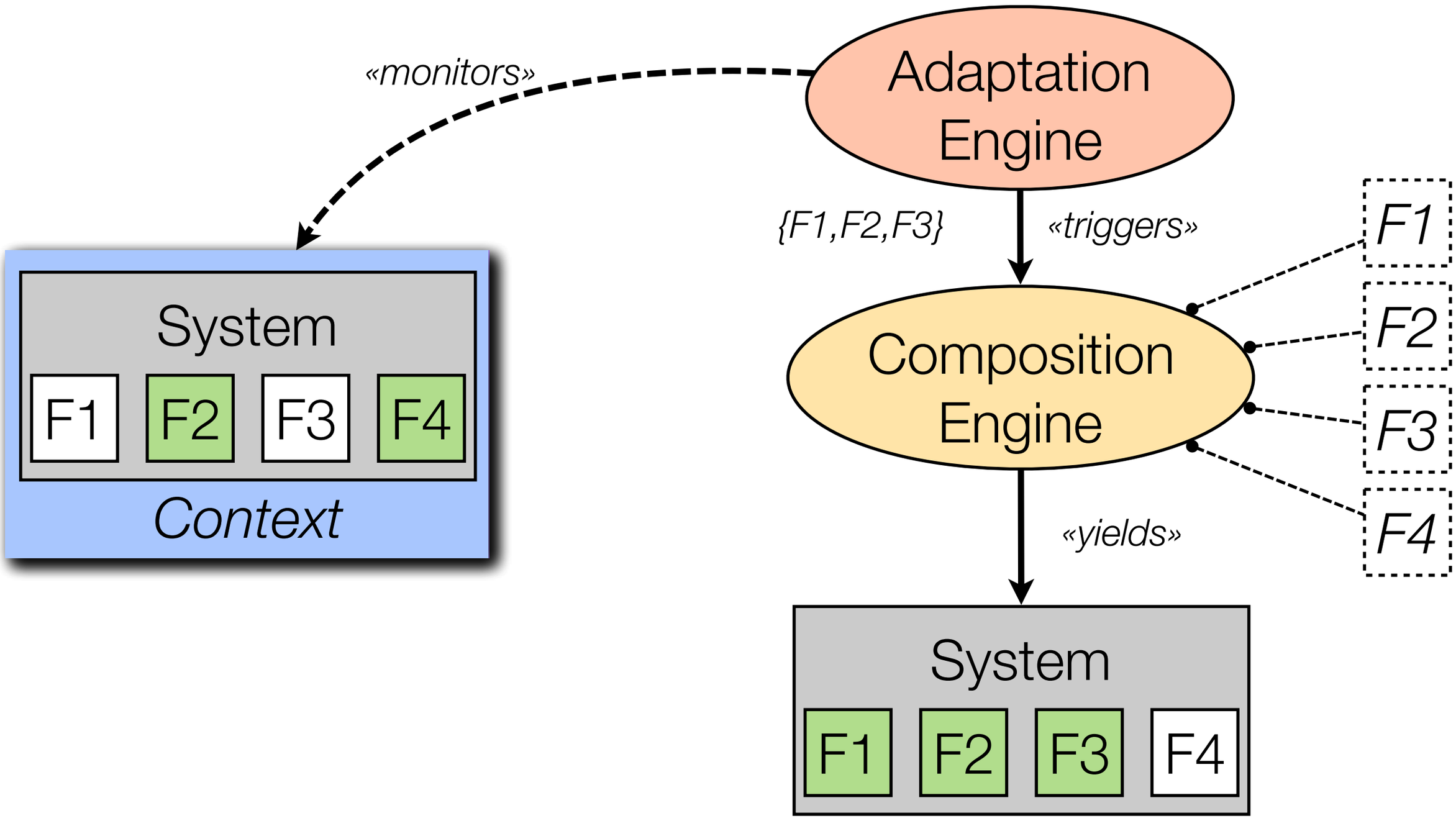


CCMS Results overview

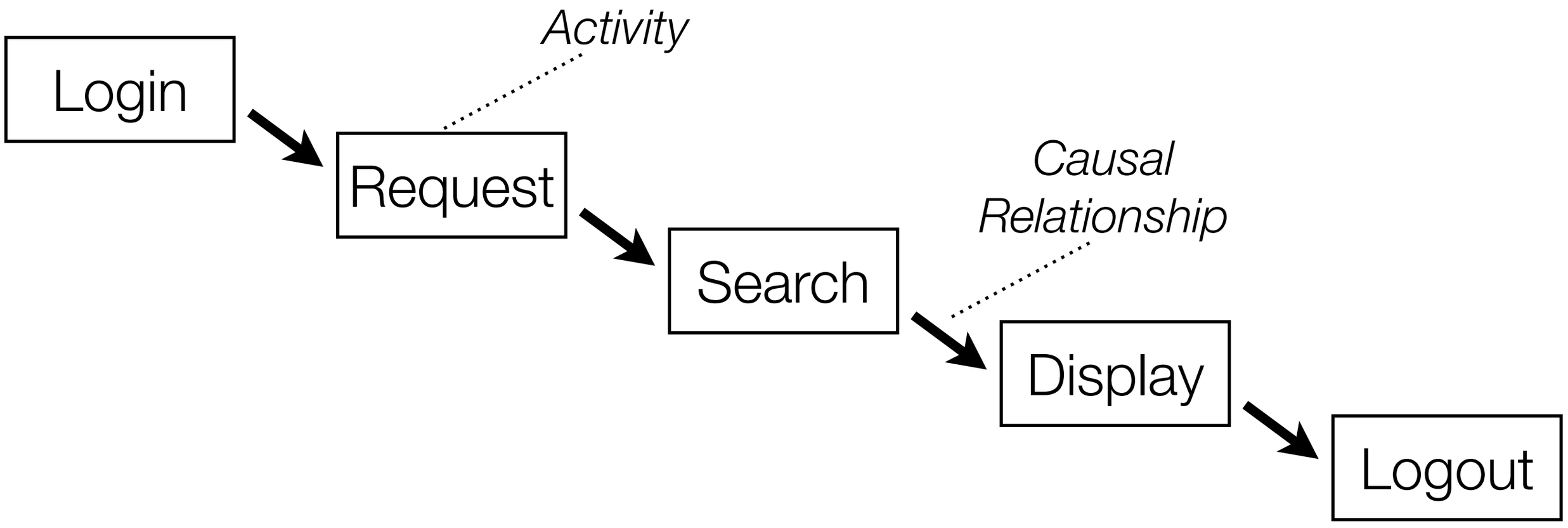
- Facts:
  - **29 features** to be composed to produce the system
  - **146 composition** directives, (up to **5** features on shared points)
- Feedback:
  - **73% of the composition were not ordered** (*i.e.*, obtained through ||)
  - The remaining **27% were ordered «by requirements»**
    - Some were identified by the **inconsistency detection** mechanisms!



Dynamic adaptation using composition



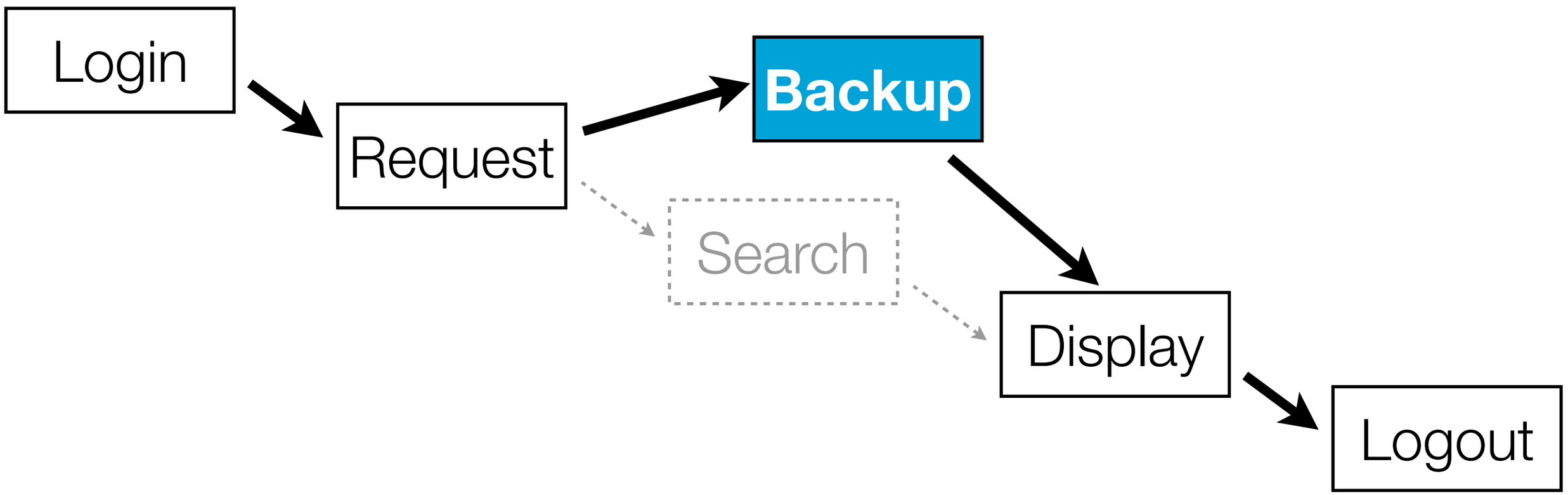
Business Process



Used to express **behavior** in the
Service-oriented Architecture paradigm

Industrial standard: BPEL

Business Processes Adaptation



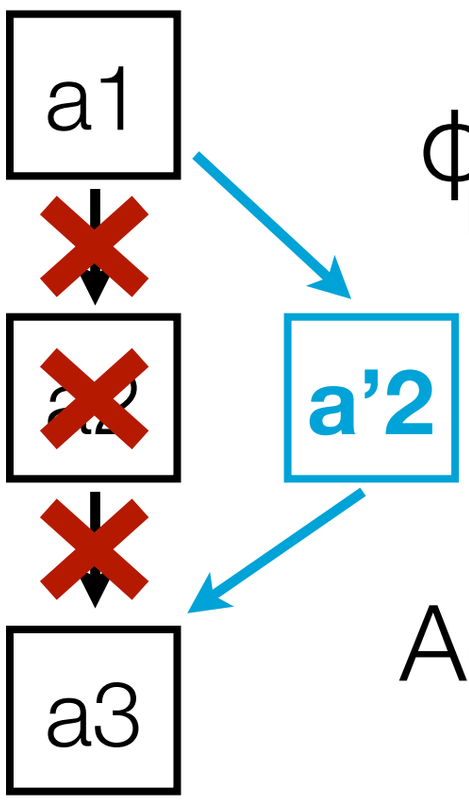
`search_status ≠ ok` (■)

Adaptation: «*use the backup server*»

«To adapt» = «To produce actions»

Adaptation function (ϕ) rational:
«generate the actions needed to adapt»

p = [add(a1), add(a2), add(a3),
add(a1→a2), add(a2→a3)]



ϕ(*p*) = [add(a'2), add(a1→a'2), add(a'2→a3),
del(a1→a2), del(a2→a3), del(a2)]

Adaptation:
(1) compute the adaptation, (2) execute it.

From Event to Complex Event Processing

Event	Condition	Action
fail	$search\_status \neq ok$	Use a backup server
slow	$bw < 100kbps$	—
cache	**fail** *followed by* **slow**	Introduce a cache
perf	$cpu > 80\%$	Monitor the process

- Event processing:
  - *Boolean formula applied to sensor values*

`search_status ≠ ok`

- Event compositions:
  - *Logical connectors: conjunction (∧), disjunction (∨)*
  - *Temporal connectors: sequence («followed by»), time window («within»)*

Running Example

1. «**Fail**» event detected (search service encounters errors):

   - The system uses now a backup server, off-site

2. «**Slow**» event detected (bandwidth drops)

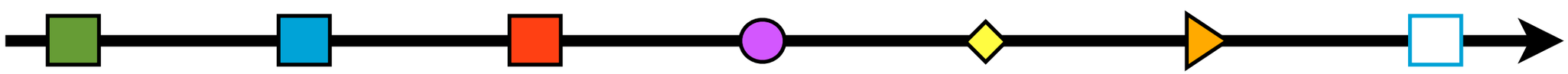
   - «**Cache**» event is identified («fail; slow»)

     - A cache is introduced

3. «**Perf**» event detected (CPU overload)

   - Monitoring is inserted in the system

4. Search service is now ok ... **What should we do**?



Events & Adaptation Implementation

repository.pl [/Users/mosser/work/adam-research/trunk/2011/SCC_20...

```
%%%%                                                    %%%%
%% Adaptation #1: Server Failure => Use Backup %%
%%%%                                                    %%%%

% Event Processing: Status of the 'search' service != 'ok'.
complex_event(fail, processing(search_status, \=, ok)).

adapt_rule(fail, switch_to_backup). % Binding: Event -> Adaptation function

switch_to_backup(process(Acts, Rels), Actions) :- % Adaptation function
        member(search, Acts), % (assumes search is present).
        % Transform all 'A < search' into 'A < backup'
        findall([del_r(A, search), add_r(A, backup)],
                member(A<search, Rels), Before),
        % Transform all 'search < A' into 'backup < A'
        findall([del_r(search, A), add_r(backup, A)],
                member(search<A, Rels), After),
        % Add the 'backup' activity, and delete the 'search' one.
        flatten([[add_a(backup)], Before, After, [del_a(search)]], Actions), !.
```

--:-- repository.pl   4% (7,49)   SVN-4299 (Prolog)----11:50 0.46----------

Event-driven Adaptation

Terminal — swipl — 80×18

```
?- send_event(search_status, 404).
%   Recognizing <EP: fail>
%   act1 = [add_a(backup),del_r(request,search),add_r(request,backup),del_r(sear
ch,display),add_r(backup,display),del_a(search)]
%   exec+(act1,p) ... done.
true.

?- send_event(bw, 40).
%   Recognizing <EP: slow>
%   Recognizing <EC: cache>
%   act2 = [add_a(is_valid),add_a(write),add_a(read),add_r(is_valid,backup),add_
r(is_valid,read),add_r(backup,write),add_r(request,is_valid),del_r(request,backu
p),add_r(write,display),add_r(read,display),del_r(backup,display)]
%   exec+(act2,p) ... done.
true.

?-
```

Automated Unadaptation

Terminal — swipl — 80×19

```
?- send_event(search_status, ok).
%   Recognizing <EP: not(fail)>
%   Undo required <fail>
%     rewind:
%       > exec+(invert(act3),p),
%       > exec+(invert(act2),p),
%       > exec+(invert(act1),p).
%       events = [perf,cache]
%    prune:       [perf]
%    replay: [perf]
%    act4 = [add_a(m9),add_r(search,m9),del_r(search,display),add_r(m9,display),a
dd_a(m10),add_r(logout,m10),add_a(m11),add_r(display,m11),del_r(display,logout),
add_r(m11,logout),add_a(m12),add_r(request,m12),del_r(request,search),add_r(m12,
search),add_a(m13),add_r(login,m13),del_r(login,request),add_r(m13,request)]
%    exec+(act4,p) ... done.
%    H = [[perf,act4]]
true.

?-
```

Software Composition (IRL)

Applied to Sensor Networks

CYRIL CECCHINEL, SÉBASTIEN MOSSER, PHILIPPE COLLET

APSEC'16, HAMILTON, NZ, 9/12/16

AUTOMATED DEPLOYMENT OF DATA COLLECTION POLICIES OVER HETEROGENEOUS SHARED SENSING INFRASTRUCTURES

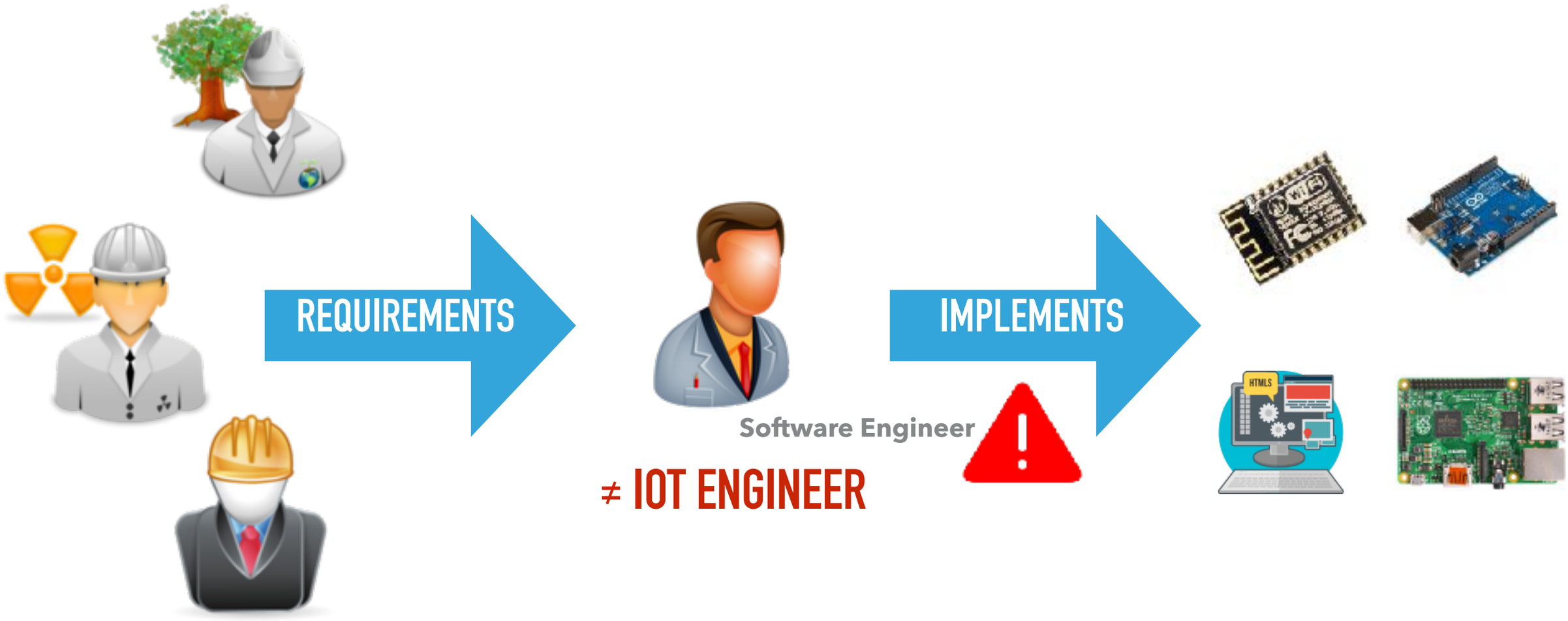
TRADITIONAL SOFTWARE DEVELOPMENT

REQUIREMENTS

Software Engineer

IMPLEMENTS

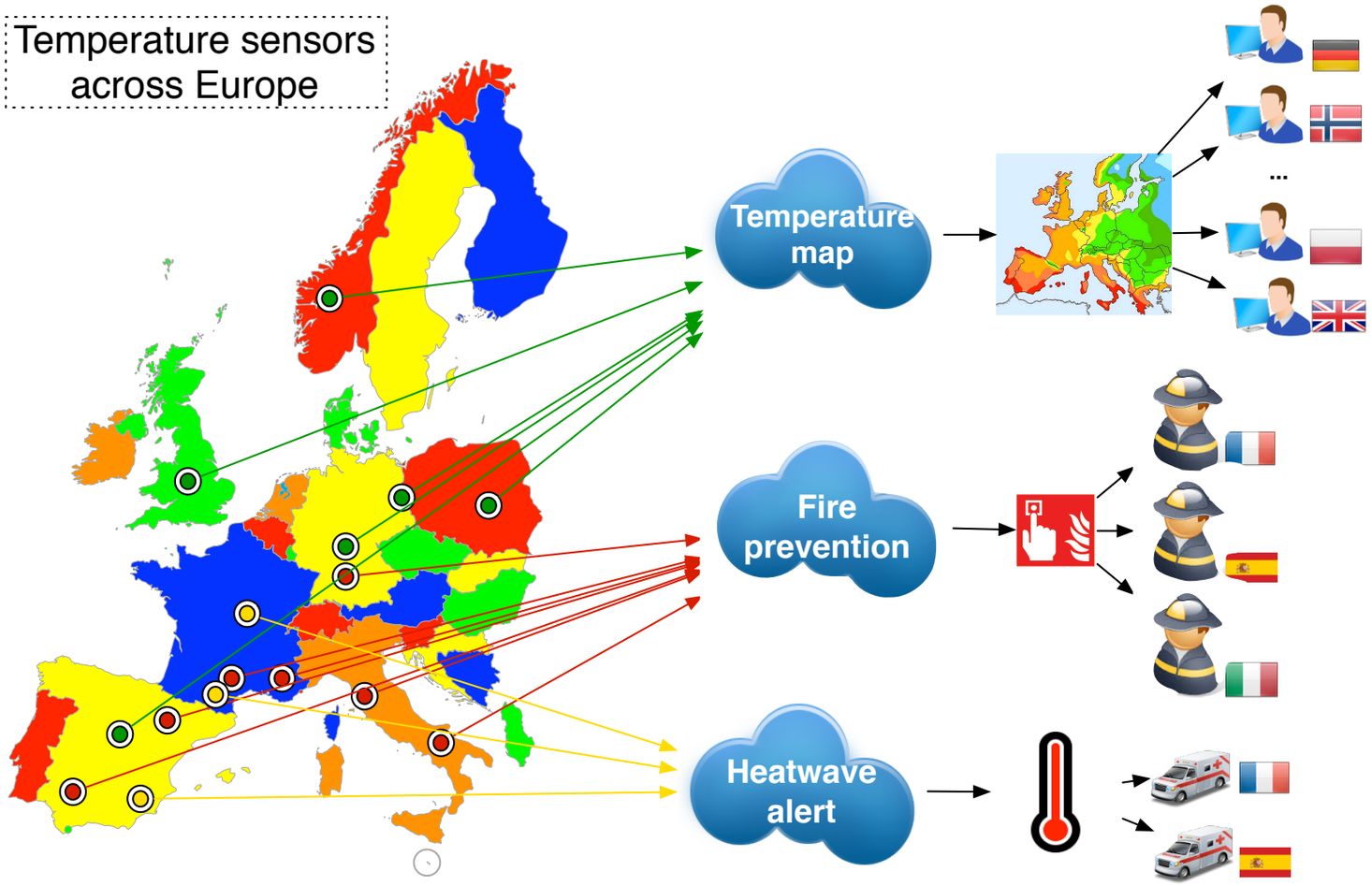
IOT DEVELOPMENT



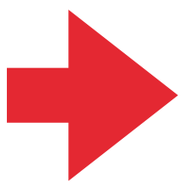
SENSOR NETWORKS CONFIGURED AT THE HARDWARE LEVEL
LOW-LEVEL PROGRAMMING LANGUAGES

TEDIOUS AND ERROR-PRONE ACTIVITIES

AD-HOC NETWORKS



AD-HOC NETWORKS
NO SHARING
DATASET ISOLATED



DIFFICULT TO (RE-)USE

« The most obvious drawback of the **current WSNs** is that they are **domain-specific** and task-oriented, tailored for particular applications with **little or no possibility of reusing them** for newer applications »

« This strategy leads to **redundant deployments** when new applications are contemplated »

Khan, I., Belqasmi, F., Glitho, R., Crespi, N., Morrow, M., & Polakos, P. (2015). Wireless Sensor Network Virtualization: A Survey.

CONTRIBUTION

- A toolchain that supports:
  - **High-level** data collection **policies**
  - **Platforms** and **network representations**
  - **Composition** and **deployment** over heterogeneous sensing infrastructures

FULLY AUTOMATED APPROACH

REQUIREMENTS

- Separation of concerns
- Automatic tailoring of policies
- Automatic projection of policies
- Automatic sharing

REQUIREMENTS

- Separation of concerns
- Automatic tailoring of policies
- Automatic projection of policies
- Automatic sharing

Software engineer

DATA COLLECTION POLICY

WSN expert

INFRASTRUCTURE MODEL

TOOLCHAIN

SOURCE CODE

SOURCE CODE

SOURCE CODE

DATA COLLECTION POLICY

Software engineer

« a set of operations performed on data to convert them into knowledge » [1]

Must be **abstracted** enough to let the **software engineer** focused on her business activities

workflow

/ˈwɜrkˌfloʊ/

« a **sequence** of **activities** performed in a business that produces a result of **observable** value to an individual actor of the **business** » [2]

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 2013.

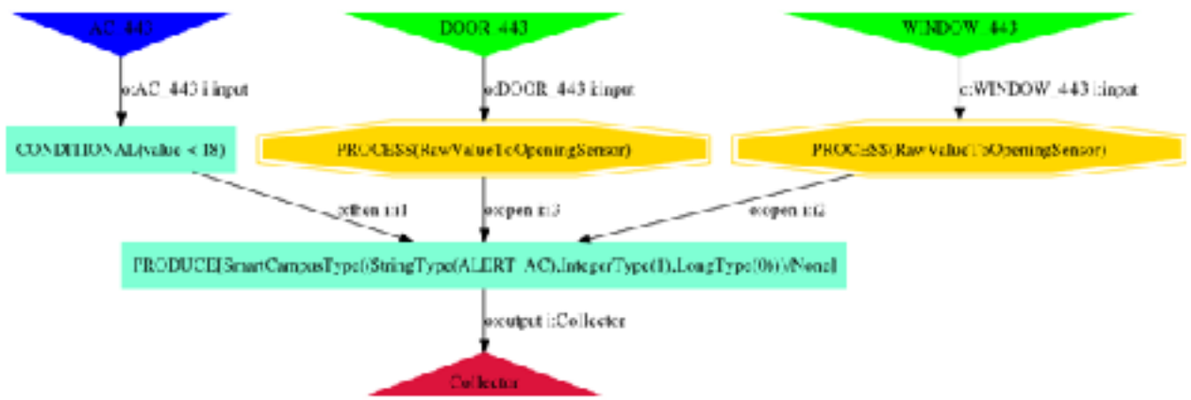
[2] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature- Oriented Domain Analysis (FODA). Technical Report CMU/SEI-90-TR-21, 1990.

DATA COLLECTION POLICY

Software engineer

- A domain-specific language to define Data Collection Policies

```
flows {
  ac_443() -> temp_filter("input")
  door_443() -> door_converter("input")
  window_443() -> window_converter("input")
  temp_filter("then") -> produce("i1")
  window_converter("open") -> produce("i2")
  door_converter("open") -> produce("i3")
  produce("output") -> collector()
}
```

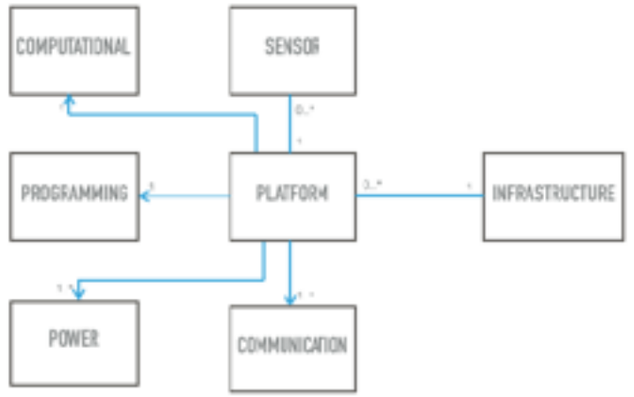


- Sensor/Collectors declaration
- Business concepts definition
- Data-flows definition

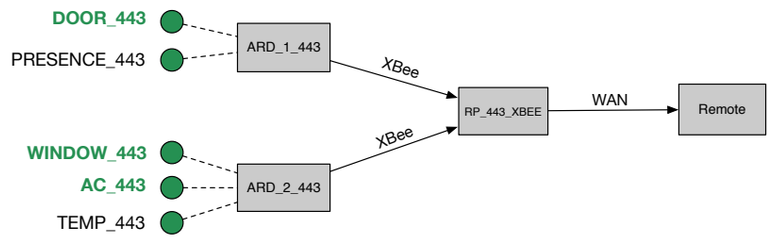
INFRASTRUCTURE MODEL

Three models to describe the sensing infrastructure

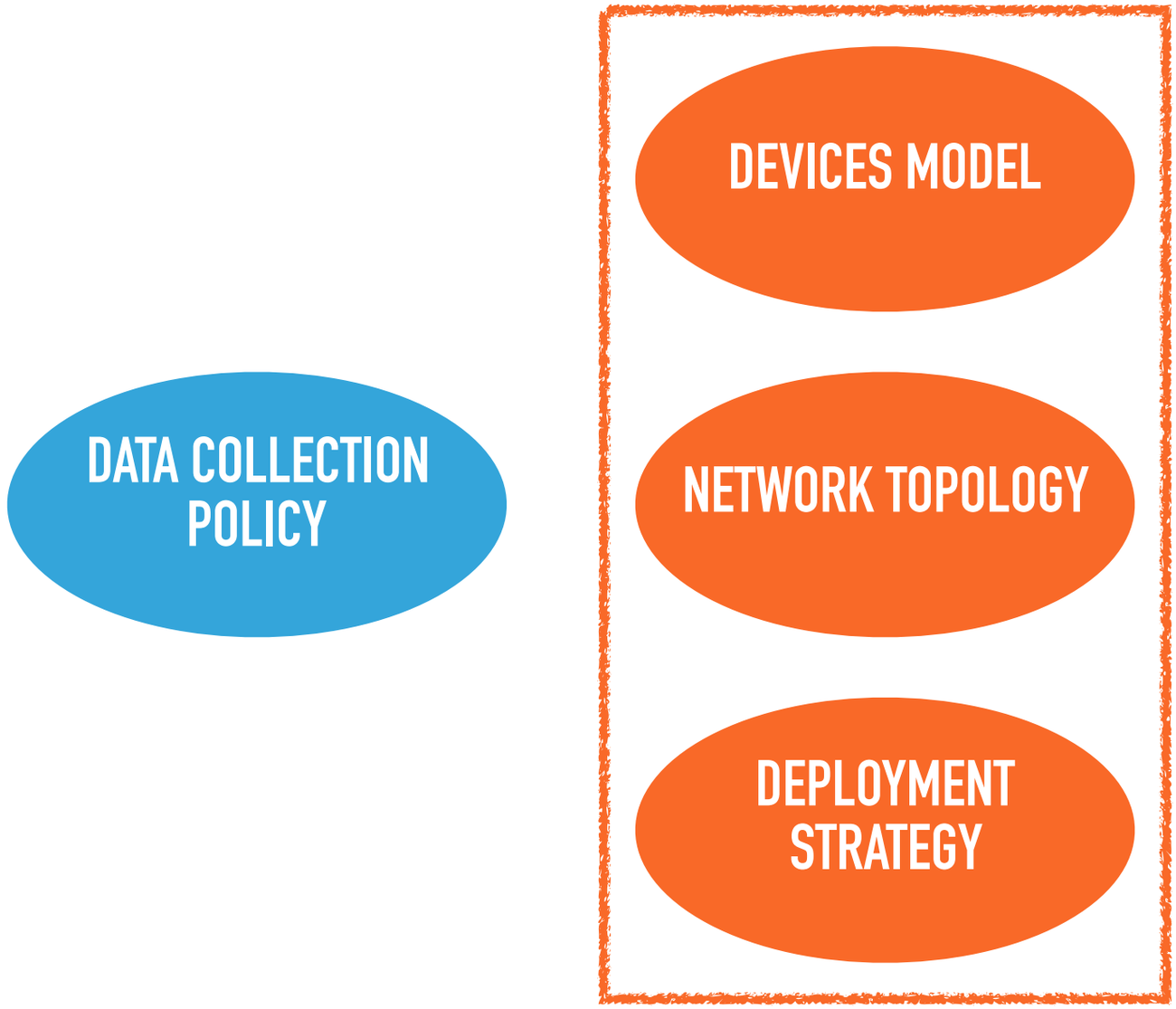
- **Platform variability** model



- **Network topology** model



- **Deployment strategy** model



SEPARATION OF CONCERNS

deploy(DATA COLLECTION POLICY . DEVICES MODEL NETWORK TOPOLOGY DEPLOYMENT STRATEGY) =

SOURCE CODE PLATFORM #1

SOURCE CODE PLATFORM #2

. . .

SOURCE CODE PLATFORM #N

REQUIREMENTS

- Separation of concerns
- Automatic tailoring of policies
- Automatic projection of policies
- Automatic sharing

Build **platform specific** data collection policies from a **platform independent** policy

DATA COLLECTION POLICY

Platform-independent

SUB DATA COLLECTION POLICY #1

Platform #1 specific

SUB DATA COLLECTION POLICY #2

Platform #2 specific

. . .

SUB DATA COLLECTION POLICY #N

Platform #N specific

Decomposition

Two operators defined at the **data collection policy layer**:

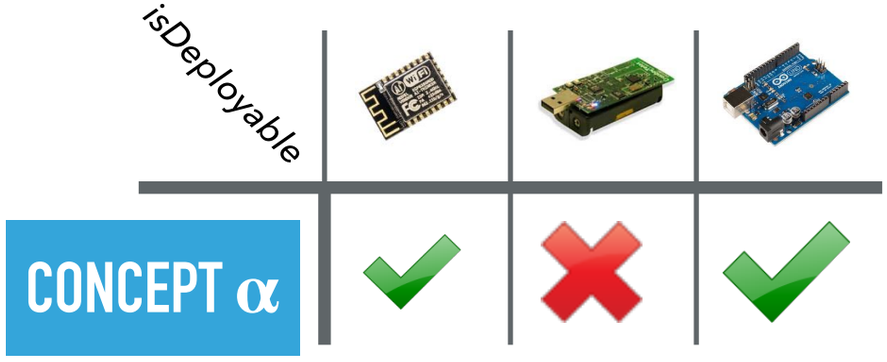
req

returns the subset of sensors needed for the realization of the concept

req(CONCEPT α) = { S2 ; S3 }

isDeployable

check if a concept is deployable on a given platform



Decomposition

An operator defined at the **network topology layer**:

reach returns the subset of sensors reachable from a given platform

reach() = {S1 ; S2 ; S3 }

platform #1

Decomposition

An operator defined at the **deployment strategy layer**:

place

For a set of platforms, return the platform that maximize the strategy's objectives

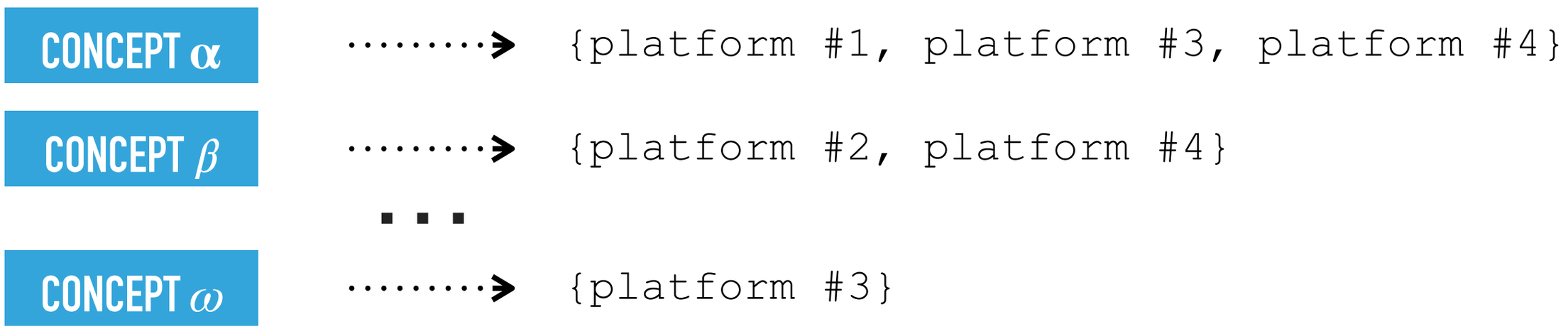
place(CONCEPT α , {P1;P2;P3}) = P1

Decomposition

For each concept c, find platforms p satisfying the property:

$$\texttt{req(c)} \subset \texttt{reach(p)} \wedge \texttt{isDeployable(c, p)}$$

If no platform satisfies the property, an error is returned to the software engineer



Decomposition

Use the place operator to select the appropriate target platform among the available candidates

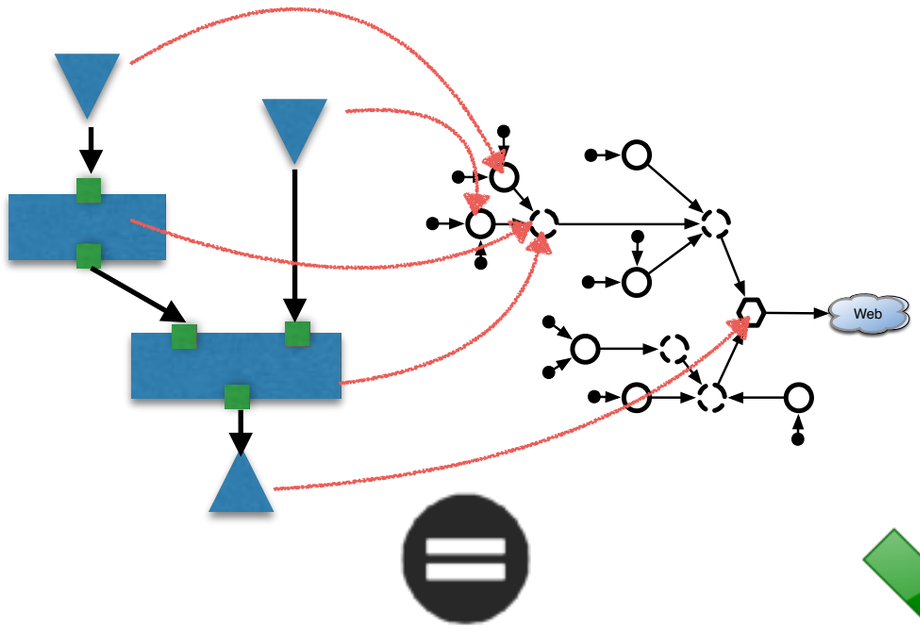
place(CONCEPT α , `{platform #1, platform #3, platform #4}`) = platform #1

place(CONCEPT β , `{platform #2, platform #4}`) = platform #4

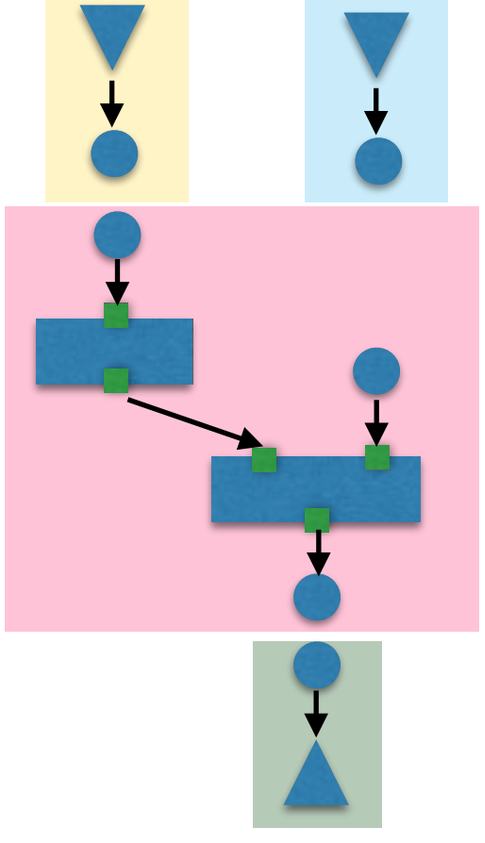
. . .

place(CONCEPT ω , `{platform #3}`) = platform #3

Decomposition



AUTOMATIC TAILORING OF POLICIES



AUTOMATIC PROJECTION OF POLICIES

Routing

on **each** platform
not involved in the
process

WHAT IF A POLICY HAS ALREADY BEEN DEPLOYED ON THE TARGETED PLATFORM ?

REQUIREMENTS

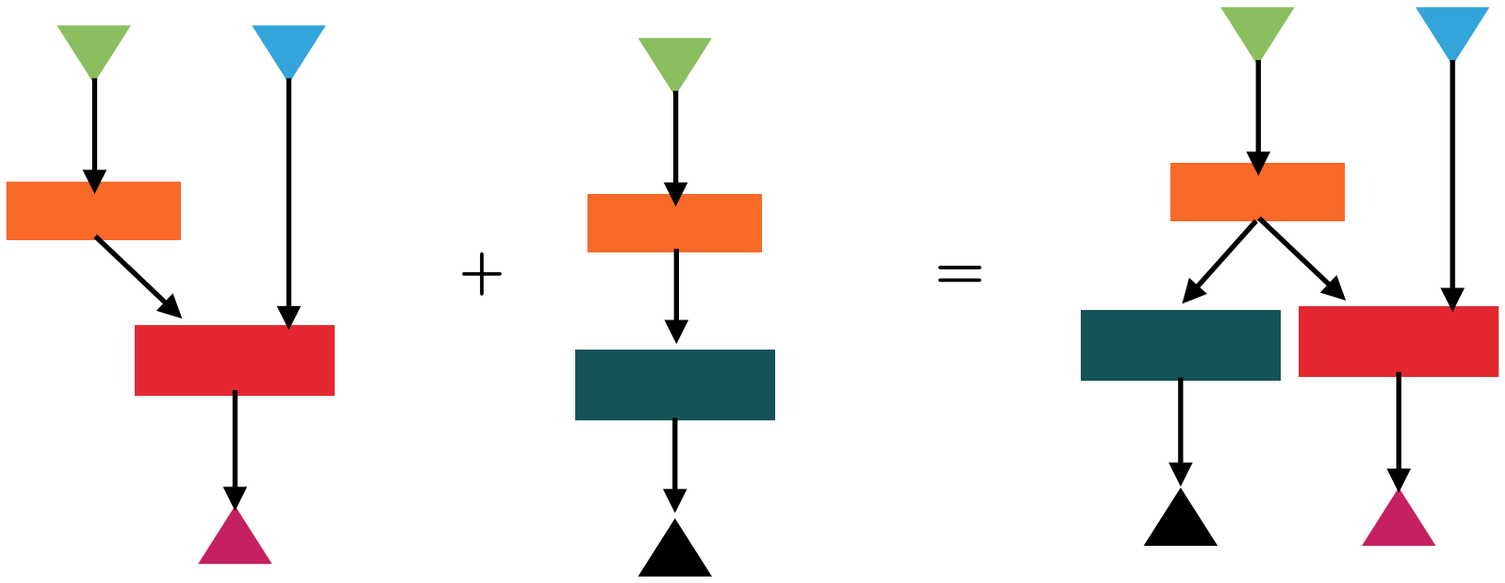
- Separation of concerns
- Automatic tailoring of policies
- Automatic projection of policies
- Automatic sharing

Composition

An operator defined at the **platform-specific** **data collection policy layer**:

$+$ compose two policies together

Extension of the graph series composition



Composition

DATA COLLECTION POLICY

Platform-independent

OTHER DATA COLLECTION POLICY #1

+

SUB DATA COLLECTION POLICY #1

Platform #1 specific

OTHER DATA COLLECTION POLICY #2

+

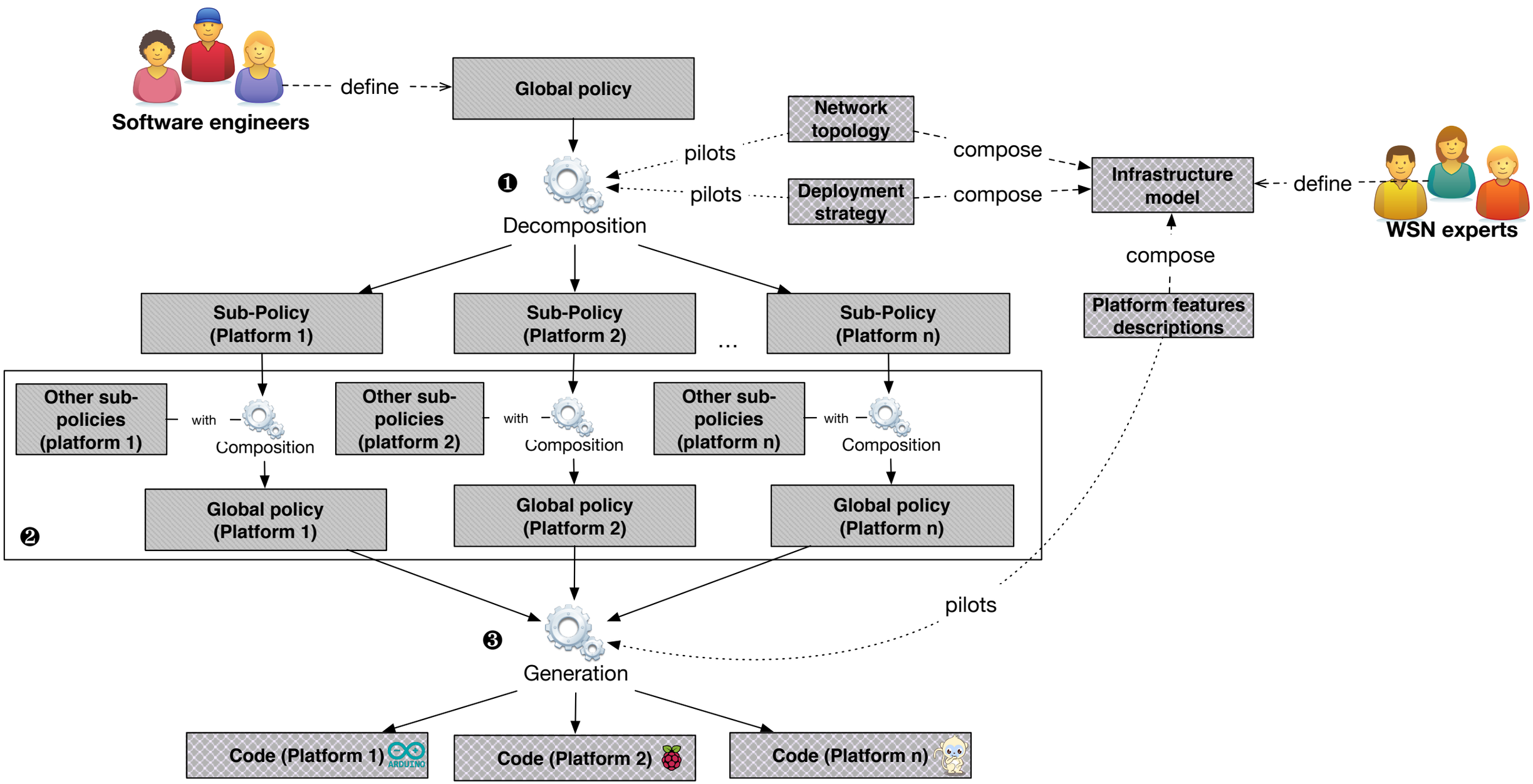
SUB DATA COLLECTION POLICY #2

Platform #2 specific

. . .

AUTOMATIC COMPOSITION

TOOLCHAIN IN ONE SLIDE



[ASSESSMENT]

DEPOSIT

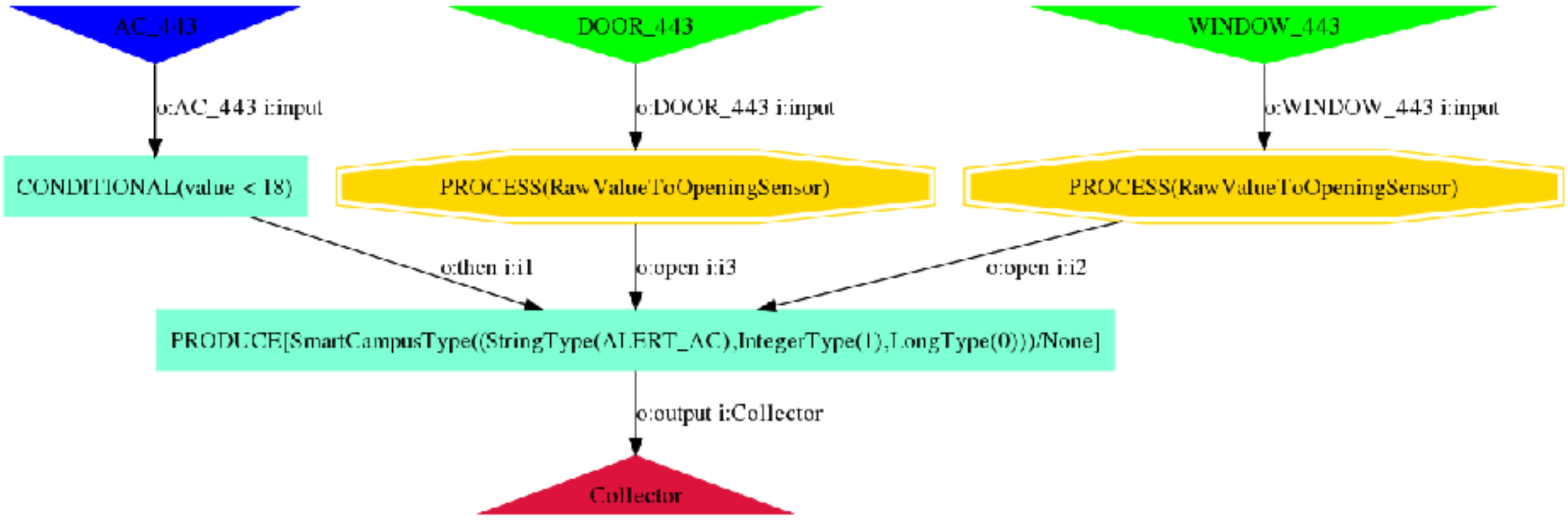
Data collEction POlicies for Sensing InfrasTructures

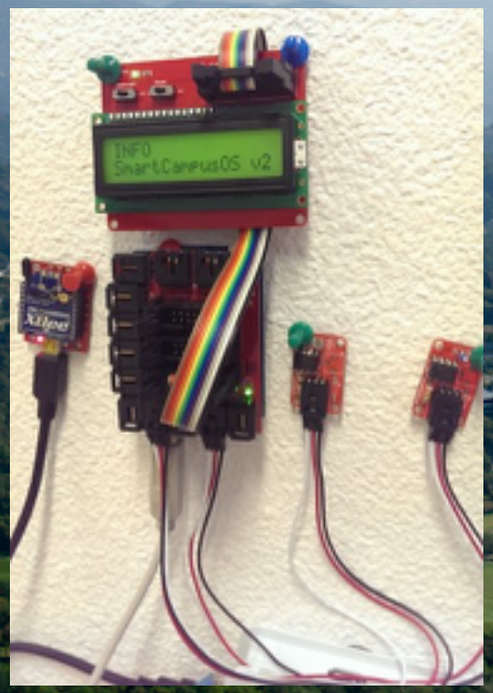
Open-source toolchain available on Github

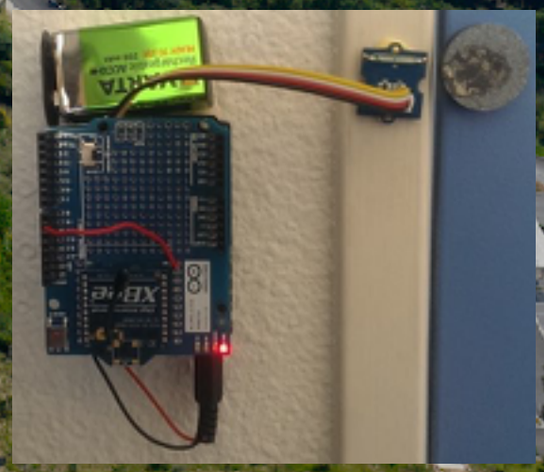
https://github.com/ace-design/DEPOSIT

DATA COLLECTION POLICY

- As a software engineer, I would like to receive AC data if the door and the window are opened for office 443 to monitor the energy loss







SMARTCAMPUS

SMARTCAMPUS.GITHUB.IO

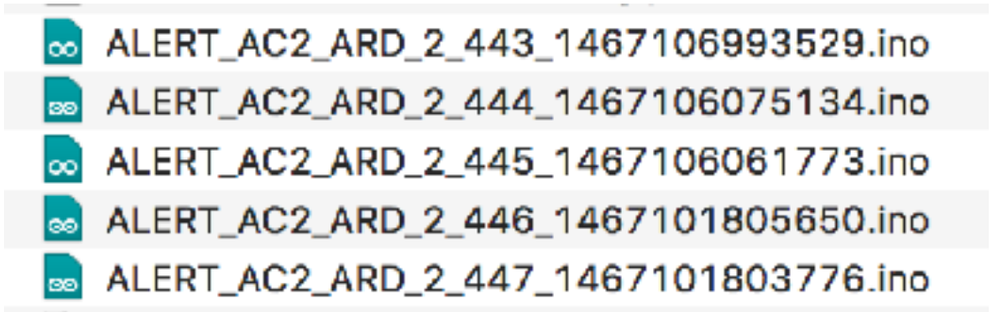
VALIDATION CRITERIA

- Separation of concerns:
  - Design using only **business concepts**
  - Deployment **without** a-priori knowledge
- Automatic tailoring of policies
  - Generated code should call the **right** libraries
- Automatic projection of policies
  - Concepts are projected on the **appropriate** platform
  - **Ready-to-flash** code

USING THE TOOLCHAIN

DEPLOYMENT OF THE RUNNING EXAMPLE (COMPREHENSIVE POLICY: 50 OFFICES)

	DEPOSIT source	# Generated files	# Generated LoC	# Concepts (before expansion)	# Concepts (after expansion)	Deployment time (in s)
Template	19	N/A	N/A	N/A	N/A	N/A
Single office	19	3	267	5	8	2.5
Comprehensive policy (without composition)	455	105	11685	250	400	50




```
#include <grovetemperature.h>
#include <raw.h>

#define BOARD_ID "ARD_2_443"
```

We consider **15 minutes** as the required time for a network expert to write and enact the code for a given office without using any aspect of the toolchain

USING THE TOOLCHAIN

- Automatic sharing

  ☆Successful deployment of multiple applications

App #1: Air conditioning warning

App #2: Fahrenheit converter

App #3: Parking space occupancy

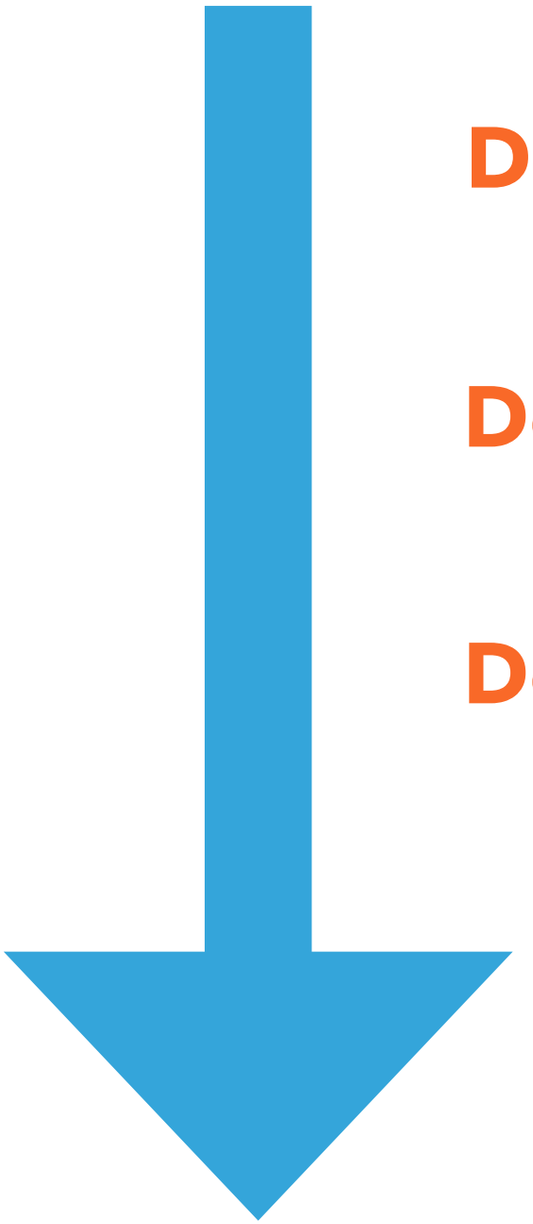
100 OFFICES

250 PARKING SPACES

BLANK INFRASTRUCTURE

ONE BORDER-ROUTER

USING THE TOOLCHAIN



Deployment of App #1 - no composition triggered

Deployment of App #2 - 101 compositions triggered

Deployment of App #3 - 1 composition triggered

[PERSPECTIVES]

SO WHAT'S NEXT?

- Integrate energy saving concerns in the results of the toolchain

- Assess the toolchain with external software engineers