Leveraging Software Design to Guide the Development of Sense/Compute/Control Applications

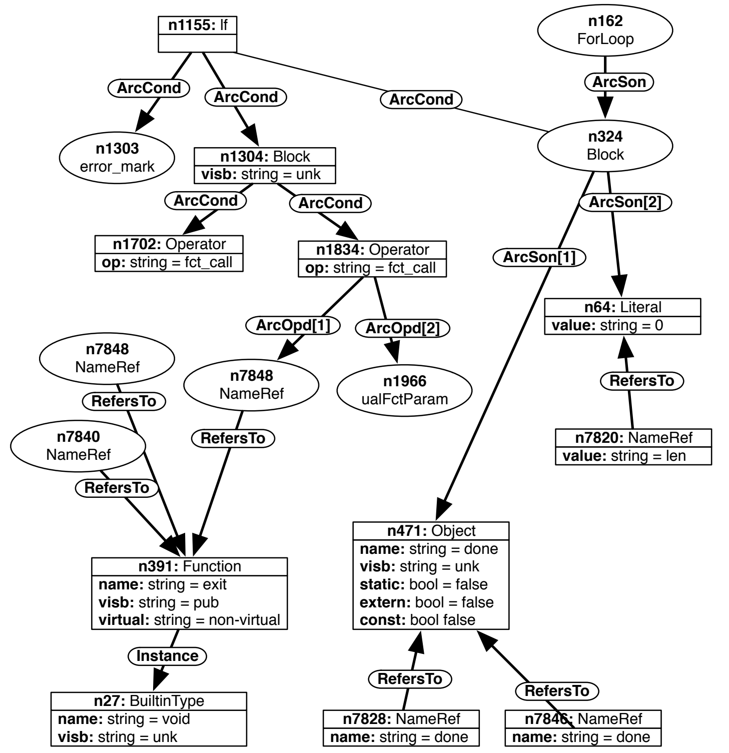
Damien Cassou

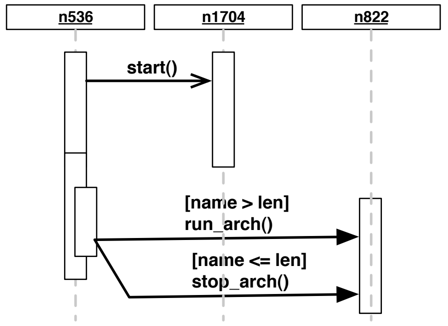
Design is Crucial

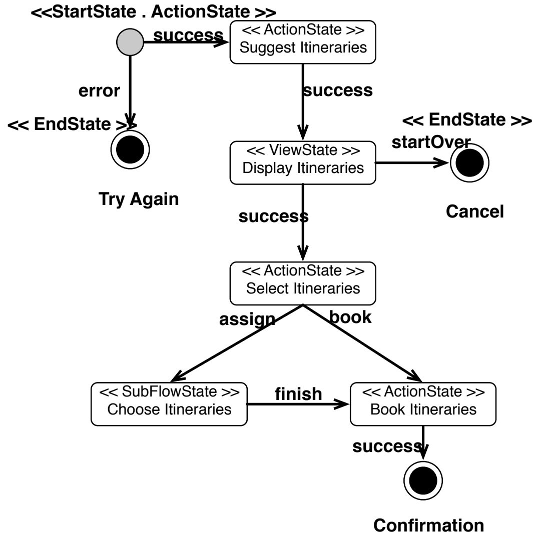
> "The most important ingredient in ensuring a software system's long-term success is its design"

ICSE'11 c.f.p

- A good design improves
  - collaboration
  - productivity
  - maintenance



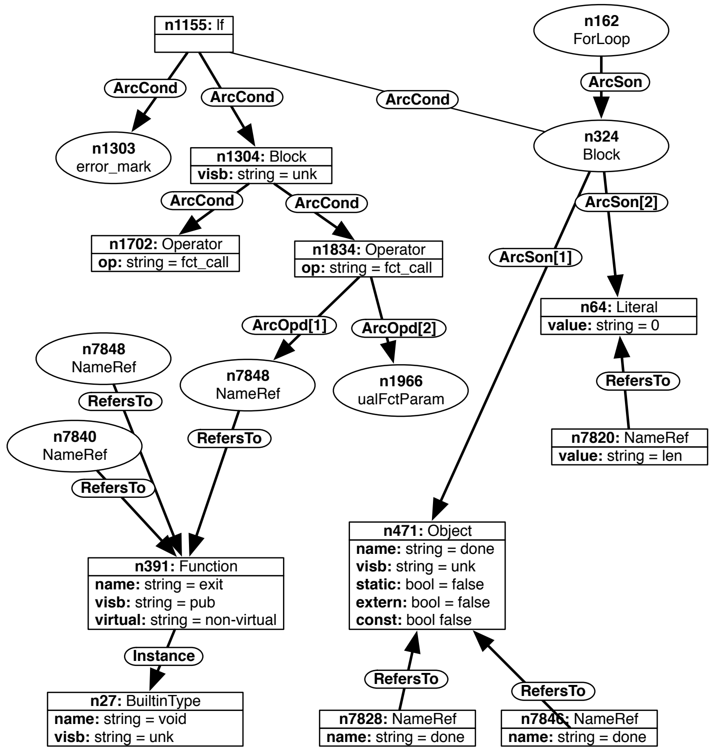


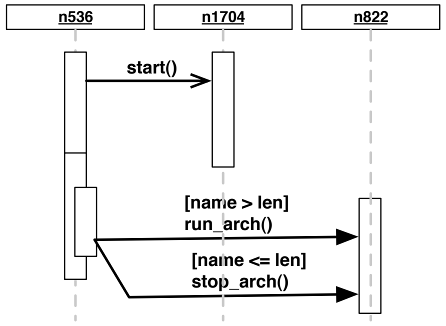


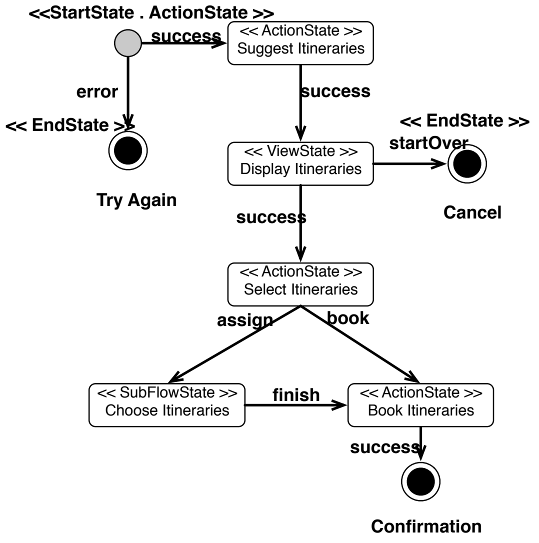
Design Framework

A good design requires a *design framework* which guides the architect with

- a language
- a paradigm



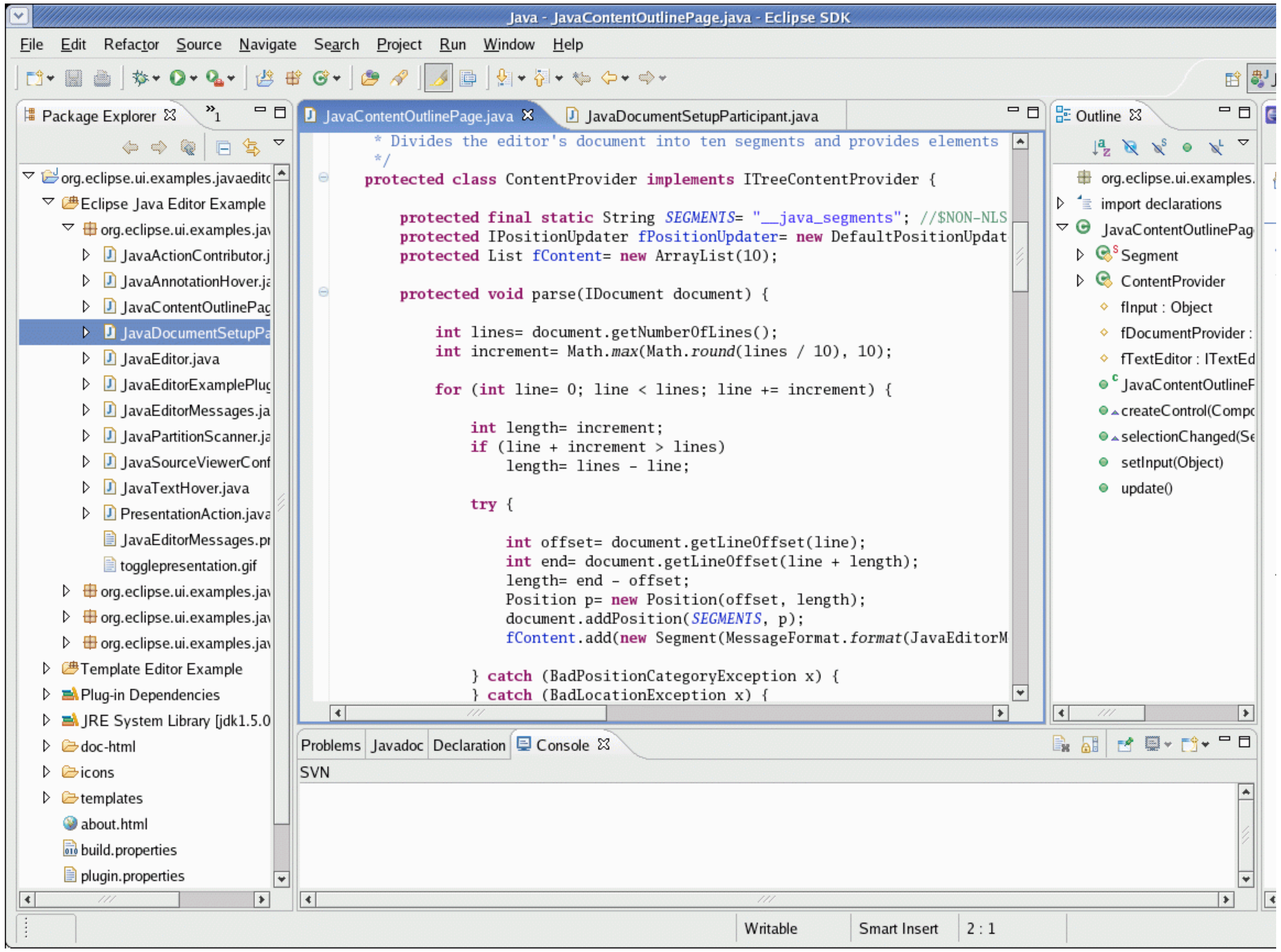




Programming Framework

A good implementation requires a programming framework which guides the developer with

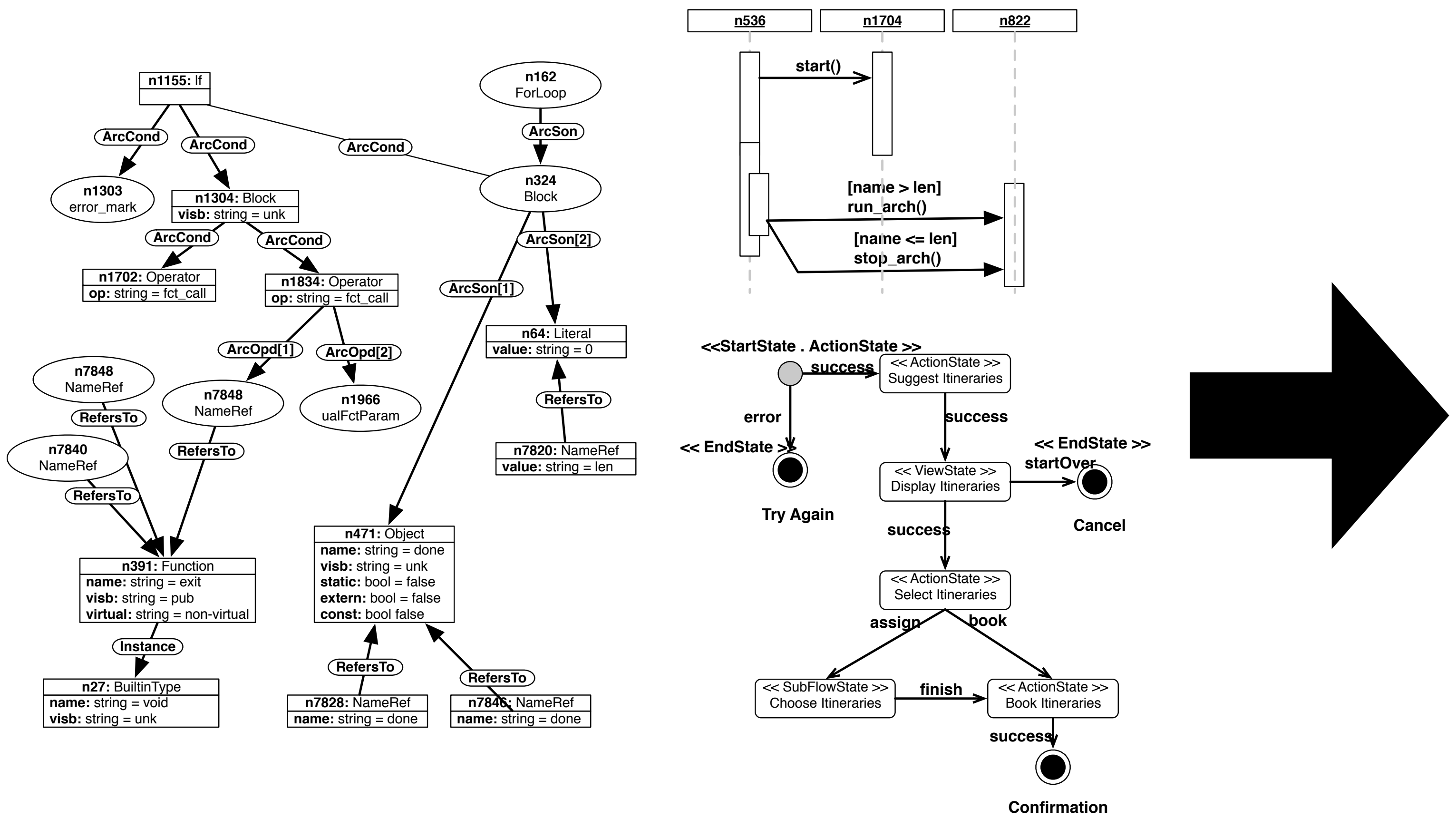
- abstractions
- services

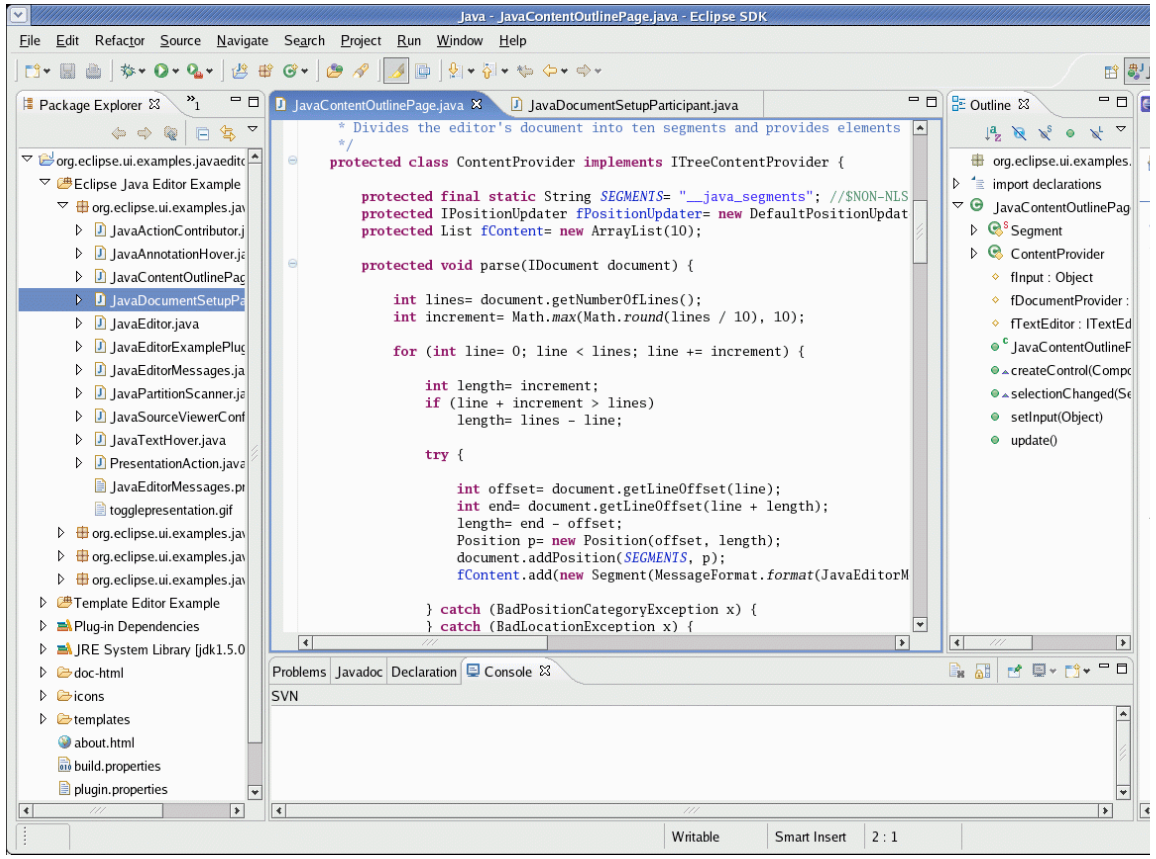


```
     * Divides the editor's document into ten segments and provides elements
     */
    protected class ContentProvider implements ITreeContentProvider {

        protected final static String SEGMENTS= "__java_segments"; //$NON-NLS
        protected IPositionUpdater fPositionUpdater= new DefaultPositionUpdat
        protected List fContent= new ArrayList(10);

        protected void parse(IDocument document) {

            int lines= document.getNumberOfLines();
            int increment= Math.max(Math.round(lines / 10), 10);

            for (int line= 0; line < lines; line += increment) {

                int length= increment;
                if (line + increment > lines)
                    length= lines - line;

                try {

                    int offset= document.getLineOffset(line);
                    int end= document.getLineOffset(line + length);
                    length= end - offset;
                    Position p= new Position(offset, length);
                    document.addPosition(SEGMENTS, p);
                    fContent.add(new Segment(MessageFormat.format(JavaEditorM

                } catch (BadPositionCategoryException x) {
                } catch (BadLocationException x) {
```

Conformance

An implementation must conform to its design





```
     * Divides the editor's document into ten segments and provides elements
     */
    protected class ContentProvider implements ITreeContentProvider {

        protected final static String SEGMENTS= "__java_segments"; //$NON-NLS
        protected IPositionUpdater fPositionUpdater= new DefaultPositionUpdat
        protected List fContent= new ArrayList(10);

        protected void parse(IDocument document) {

            int lines= document.getNumberOfLines();
            int increment= Math.max(Math.round(lines / 10), 10);

            for (int line= 0; line < lines; line += increment) {

                int length= increment;
                if (line + increment > lines)
                    length= lines - line;

                try {

                    int offset= document.getLineOffset(line);
                    int end= document.getLineOffset(line + length);
                    length= end - offset;
                    Position p= new Position(offset, length);
                    document.addPosition(SEGMENTS, p);
                    fContent.add(new Segment(MessageFormat.format(JavaEditorM

                } catch (BadPositionCategoryException x) {
                } catch (BadLocationException x) {
```

Requirements

1. A design framework to guide the architect

Architect

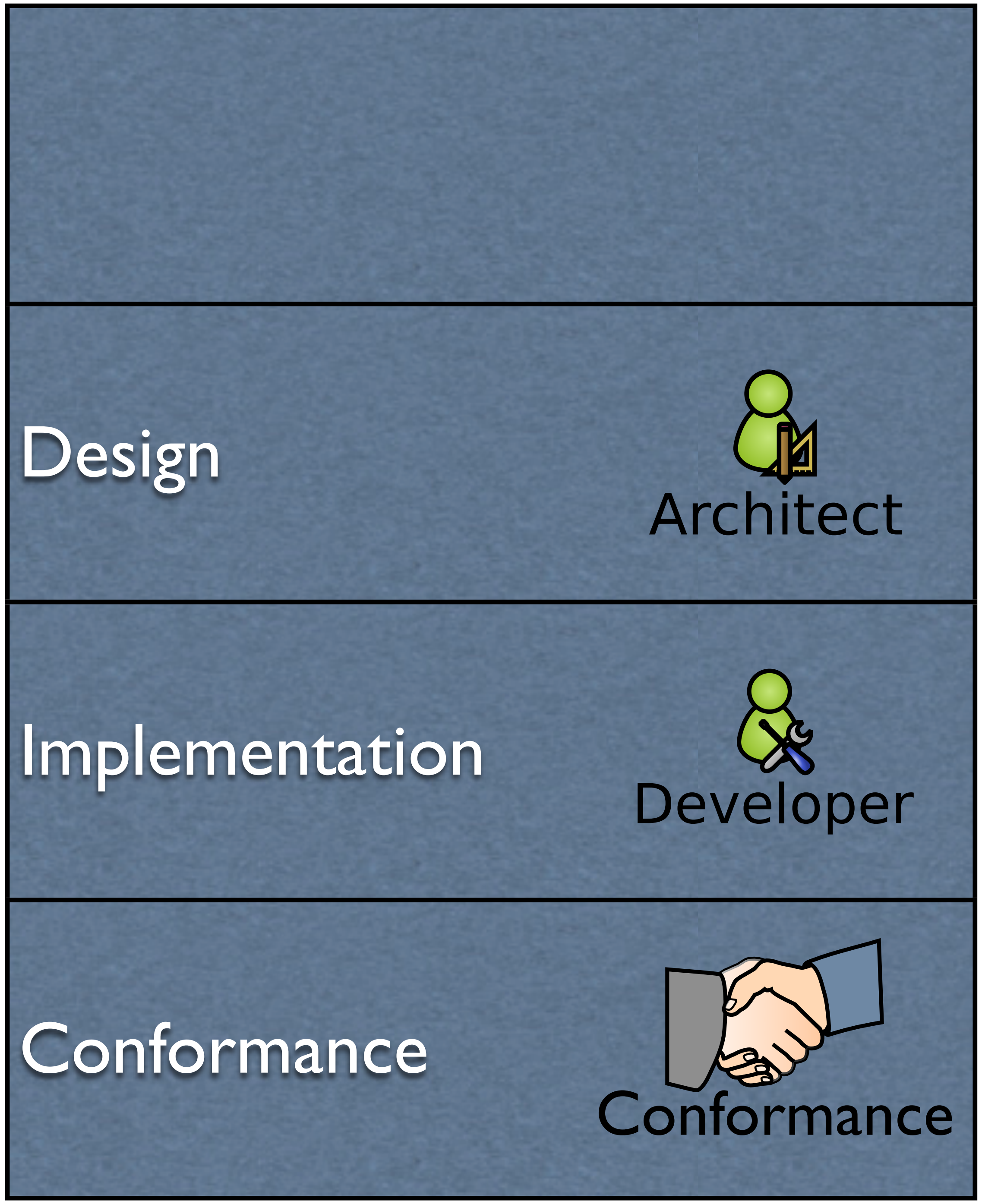
2. A programming framework to guide the developer

Developer

3. A guaranteed conformance of the implementation relatively to the design

Conformance

Related Works



Design

Architect

Implementation

Developer

Conformance

Conformance

Related Works

	GPL
Design Architect	
Implementation Developer	+
Conformance Conformance	

Java

Related Works

	GPL	Library
Design Architect		
Implementation Developer	+	++
Conformance Conformance		

Spring

Related Works

	GPL	Library	ADL
Design Architect			++
Implementation Developer	+	++	
Conformance Conformance			

C2

Related Works

	GPL	Library	ADL	ADL++
Design Architect			++	+
Implementation Developer	+	++		+
Conformance Conformance				+

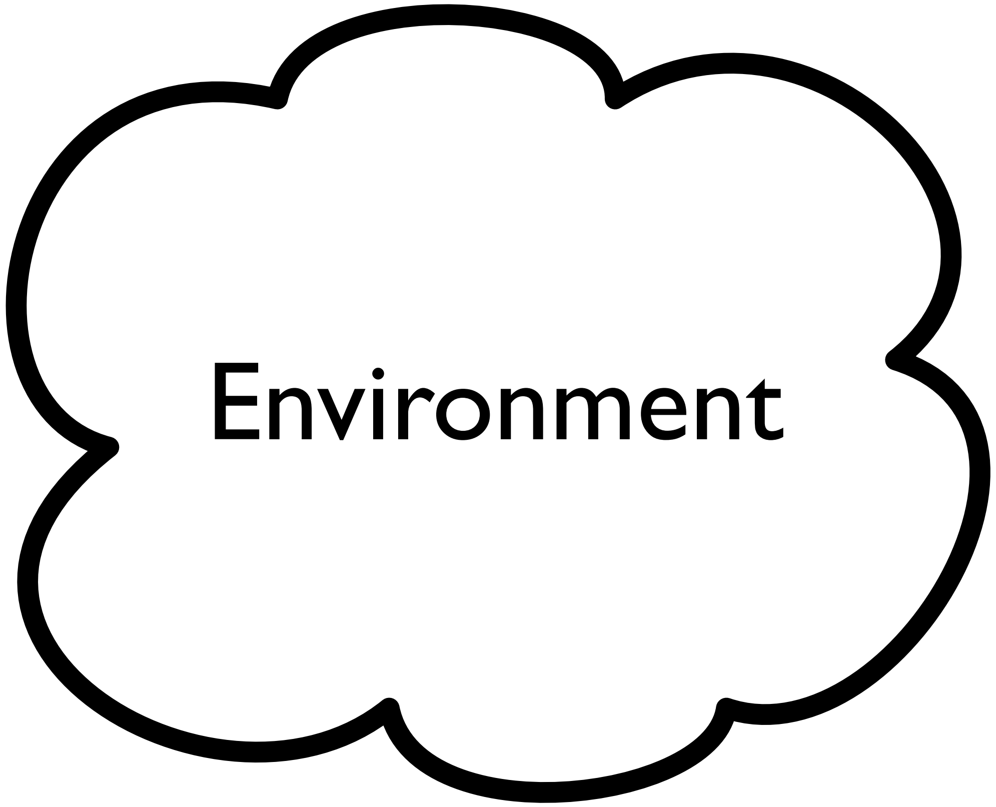
Thesis

A paradigm-oriented framework for both *design* and *implementation* which maintains *conformance* all along the software life-cycle

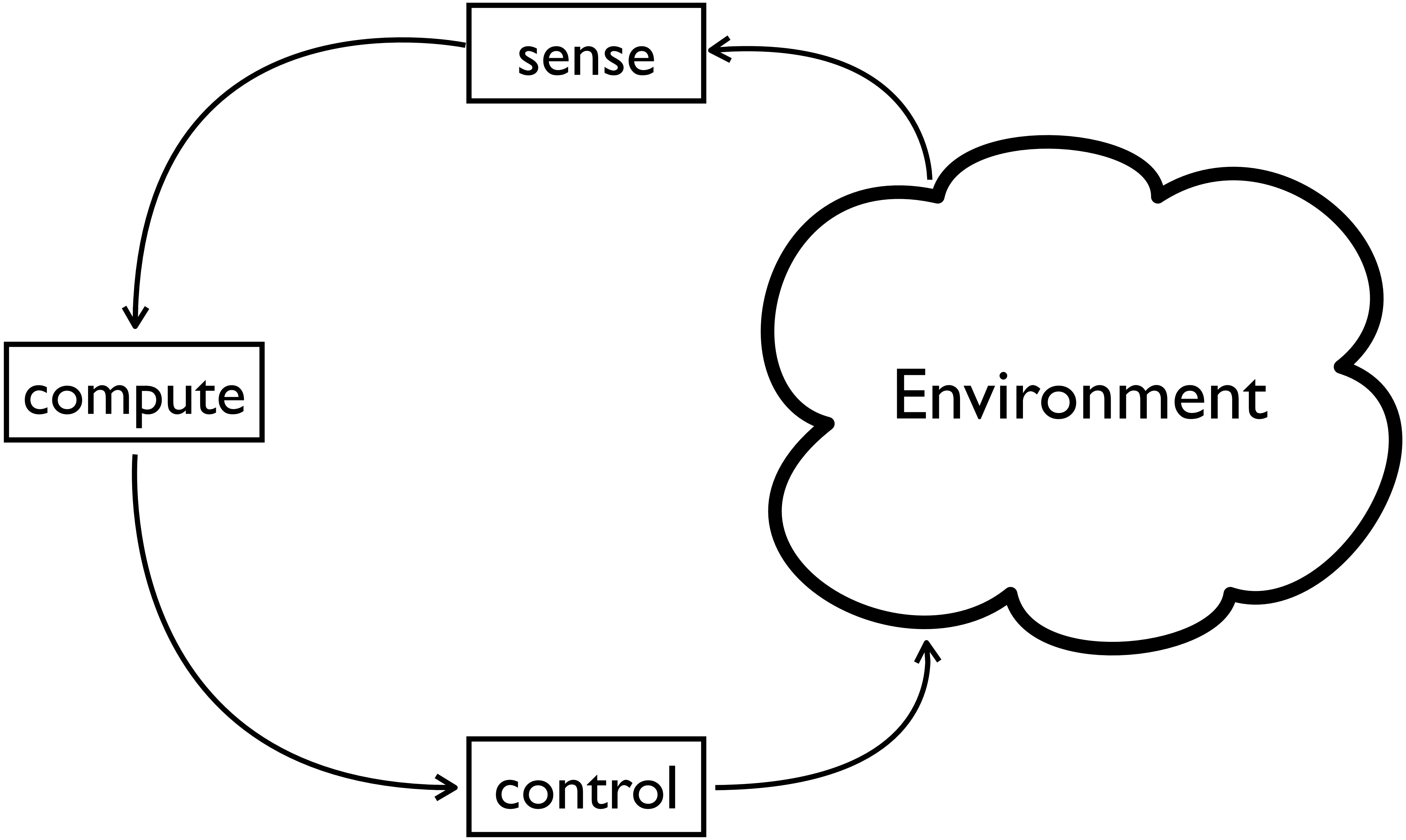
The Paradigm

Sense/Compute/Control (SCC)

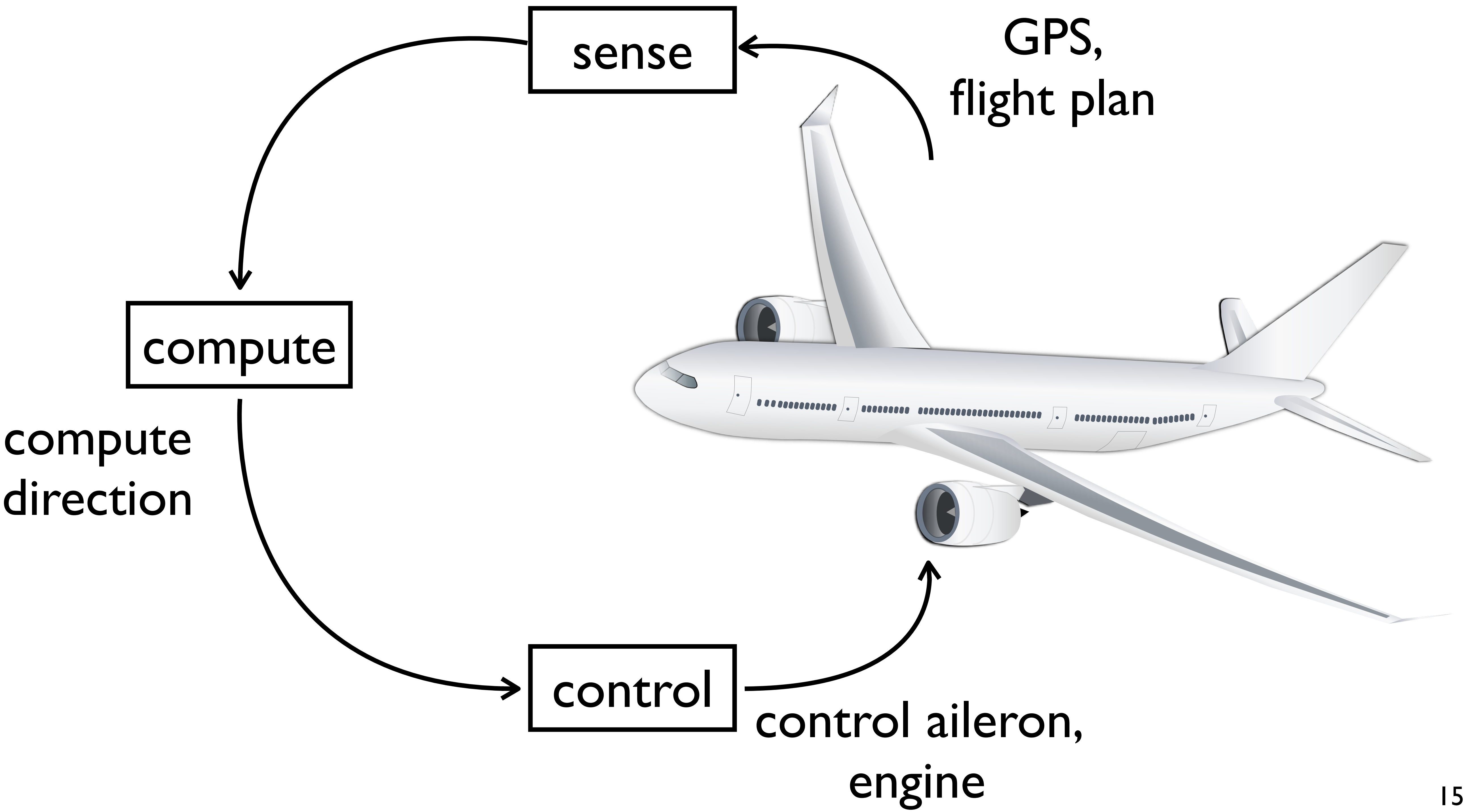
"applications that interact with an environment"



The SCC Paradigm



The SCC Paradigm



sense

GPS, flight plan

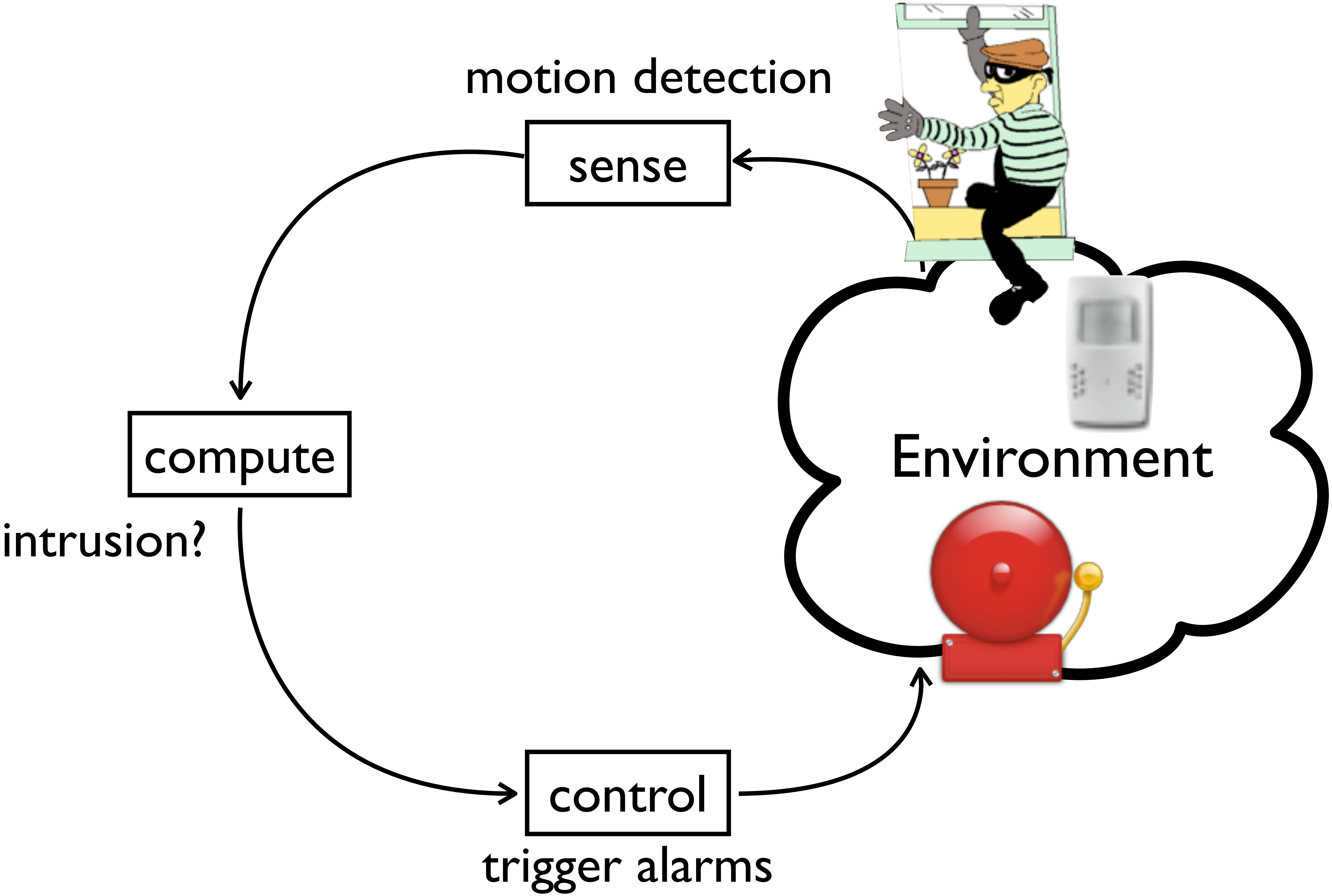
compute

compute direction

control

control aileron, engine

The SCC Paradigm



motion detection

sense

compute

intrusion?

control

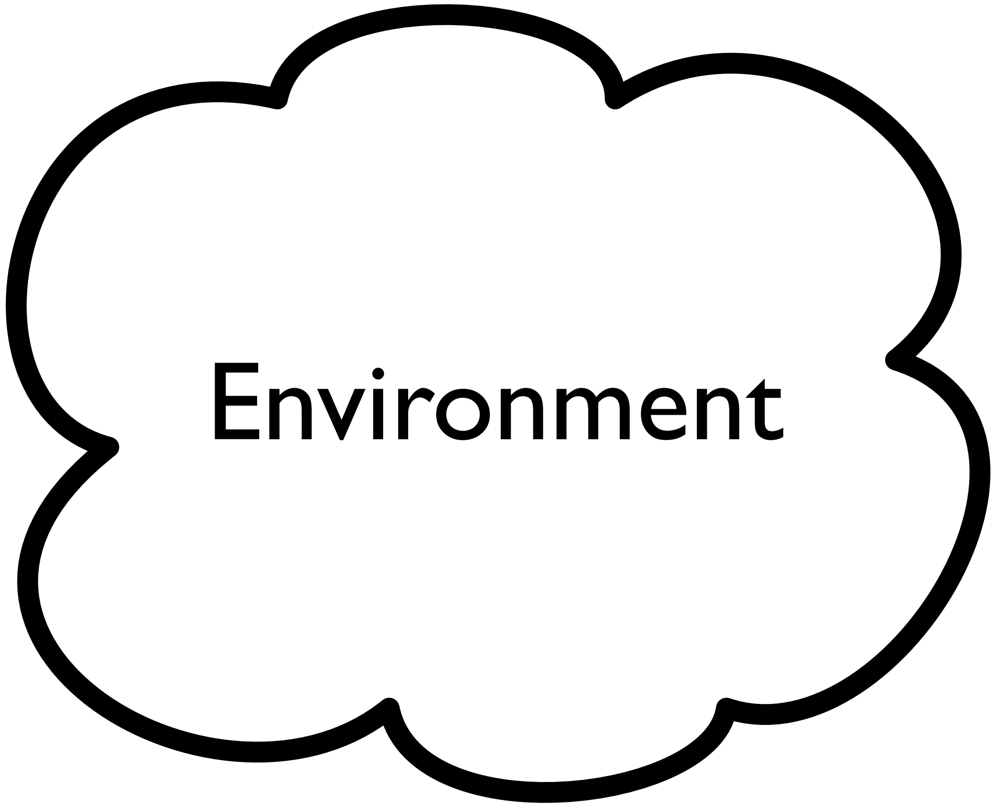
trigger alarms

Environment

The SCC Paradigm

Covers various domains

- pervasive computing
- tier-system monitoring
- avionics
- robotics
- ...



Contributions

1. A paradigm-specific design framework
2. A programming framework dedicated to a design
3. An evaluation of the approach

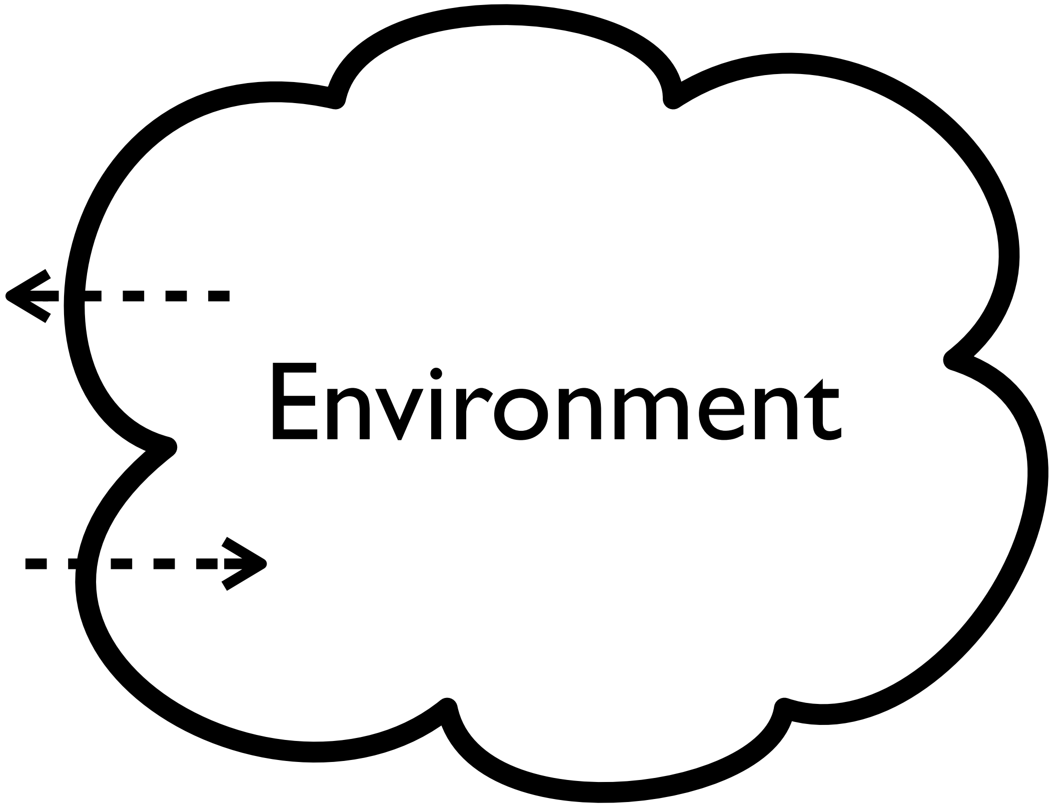
Contributions

1. A paradigm-specific design framework

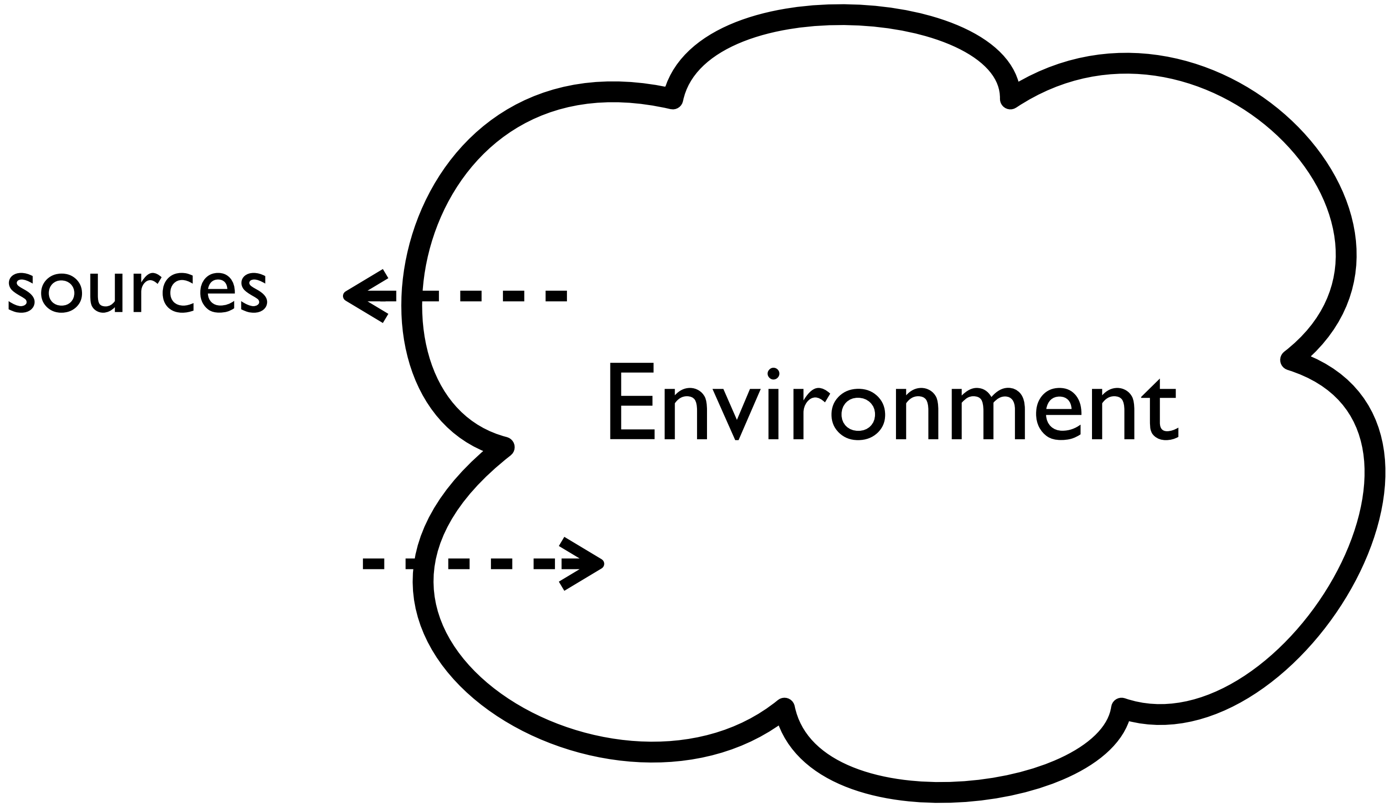
2. A programming framework dedicated to a design

3. An evaluation of the approach

Design Language

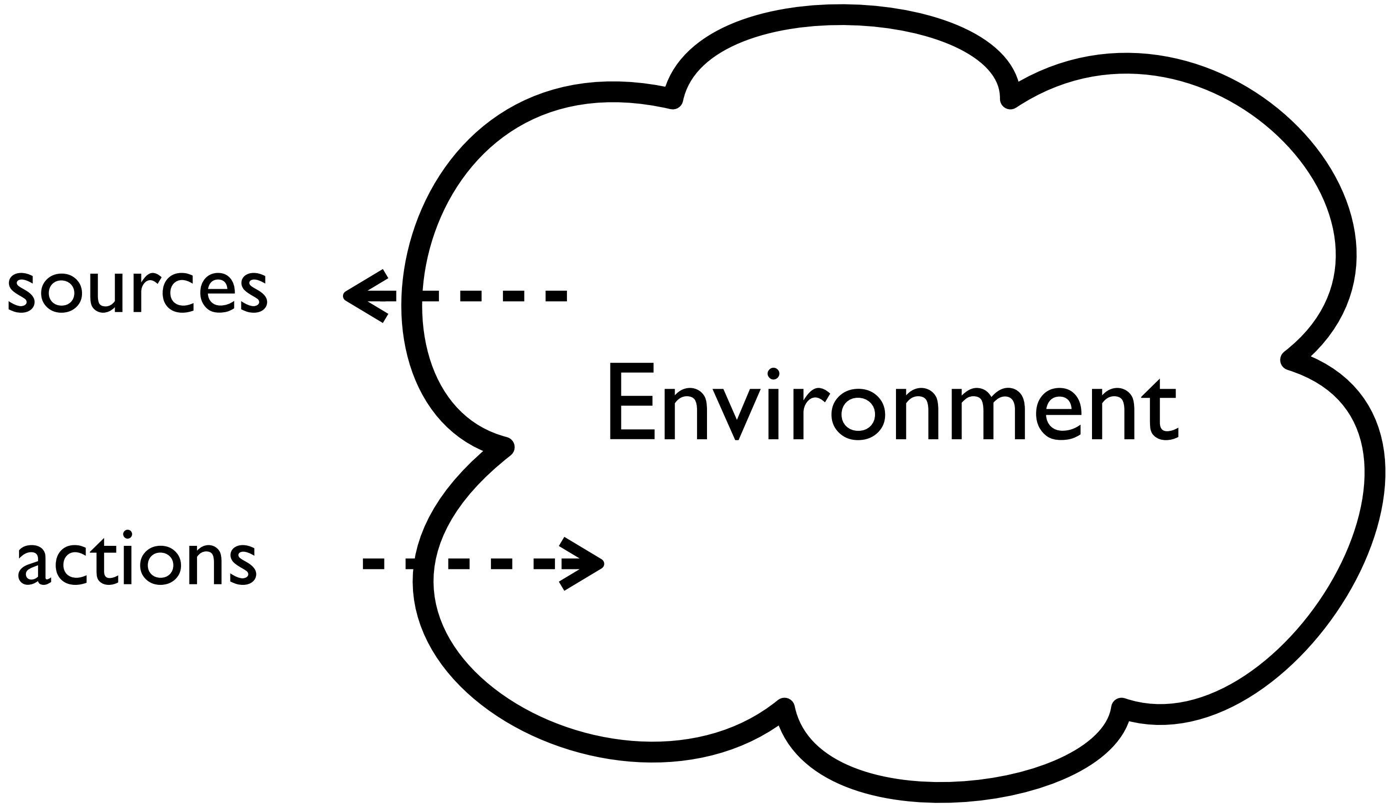


Design Language



- sources sense the environment

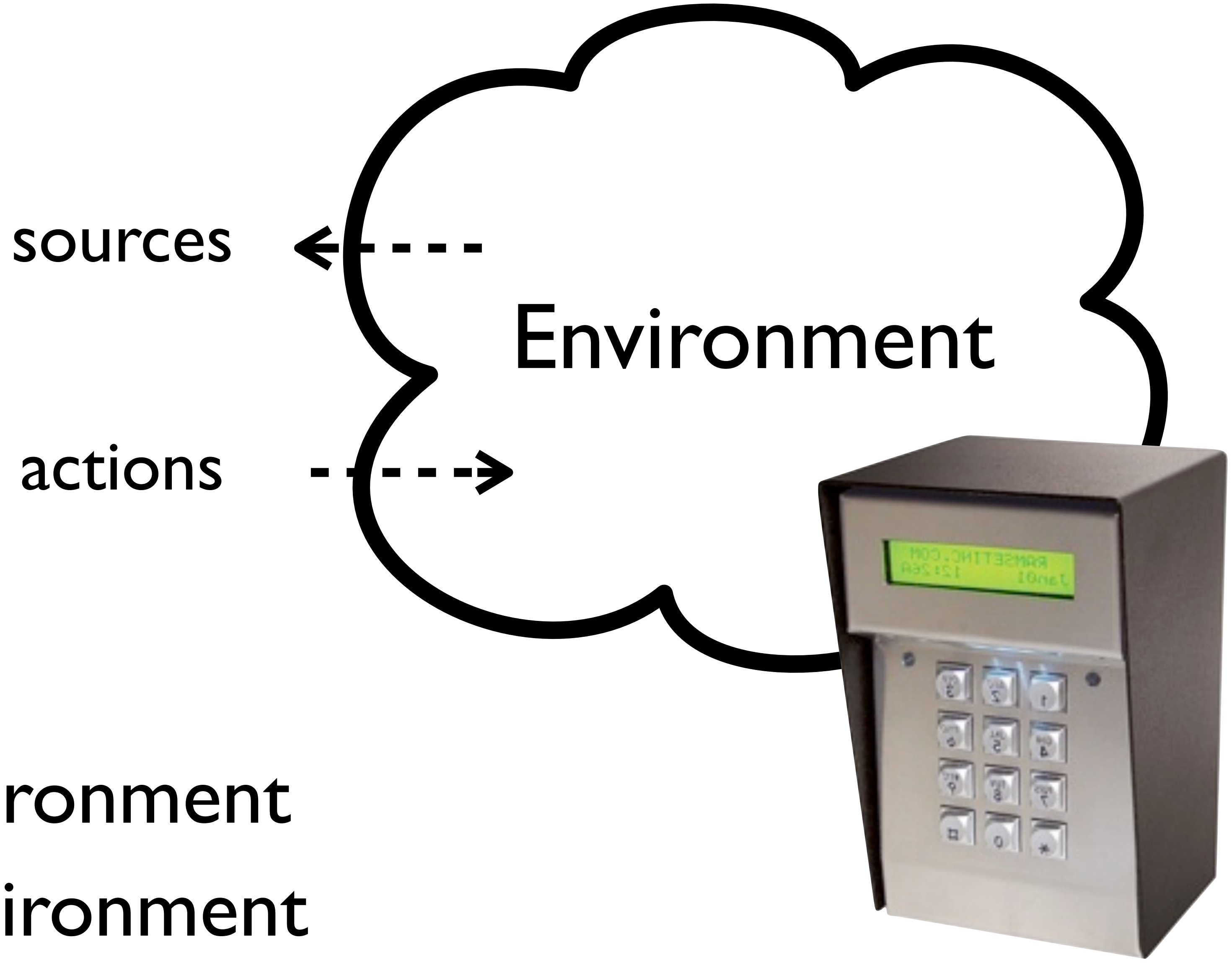
Design Language



- sources sense the environment
- actions impact the environment

Design Language

a concept for handling
the interaction with
the environment



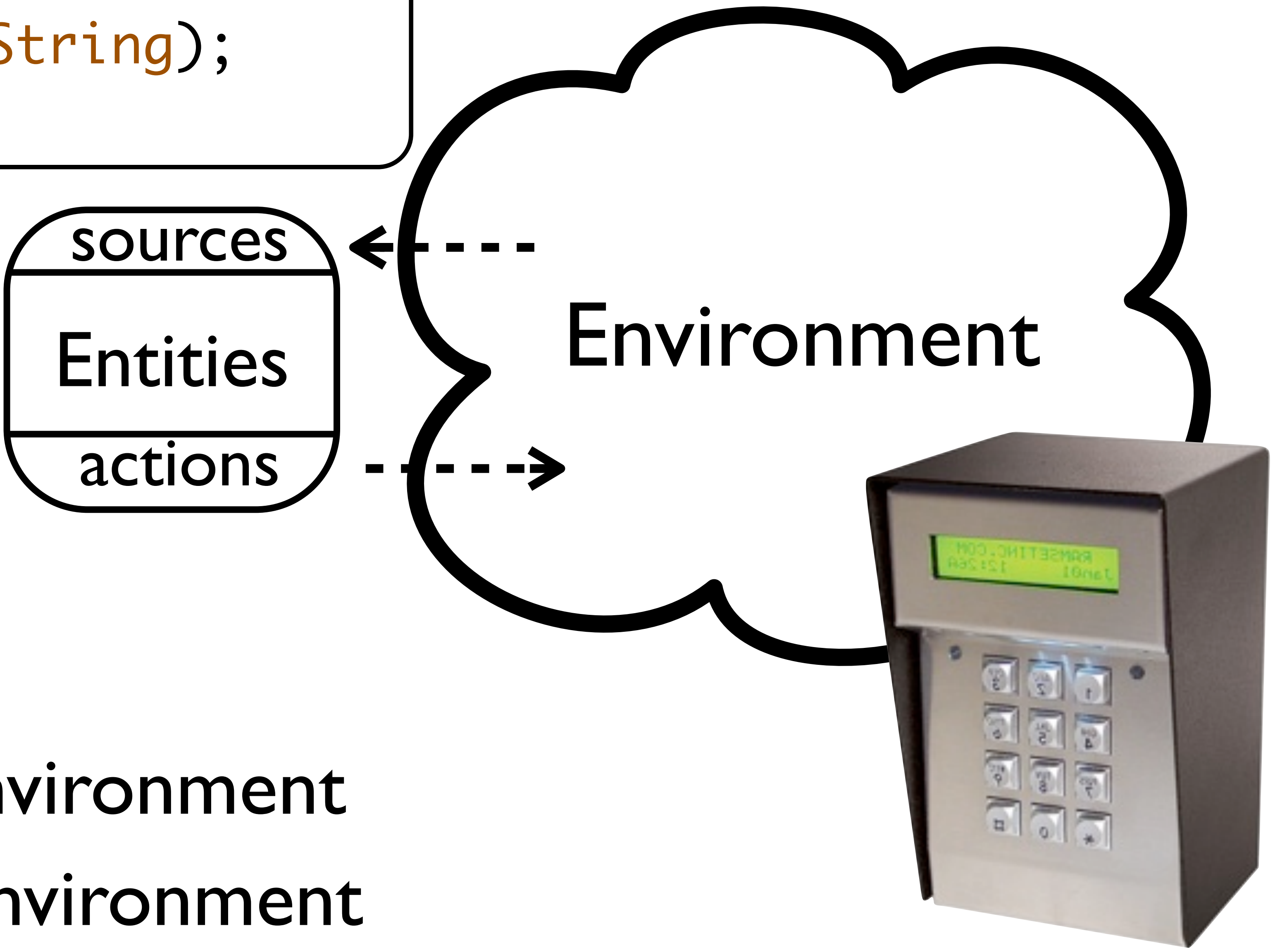
- sources sense the environment
- actions impact the environment

Design Language

```
entity Keypad {
 source keycode as Integer;
 action UpdateSt;
}

action UpdateSt {
 updateStatus(message as String);
}
```

Architect

a concept for handling the interaction with the environment



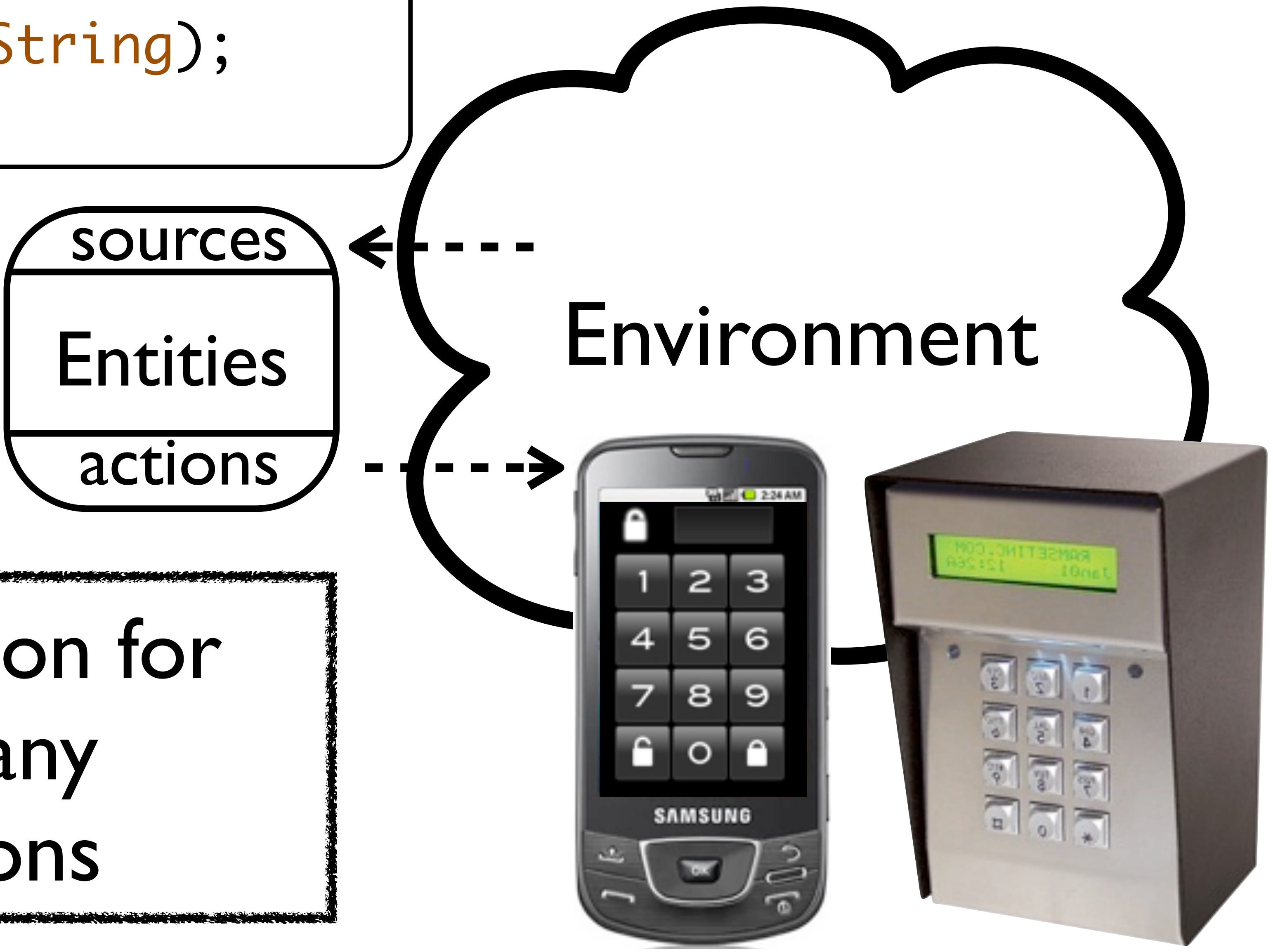
- sources sense the environment
- actions impact the environment

Design Language

```
entity Keypad {
 source keycode as Integer;
 action UpdateSt;
}

action UpdateSt {
 updateStatus(message as String);
}
```

Architect

a concept for handling
the interaction with
the environment



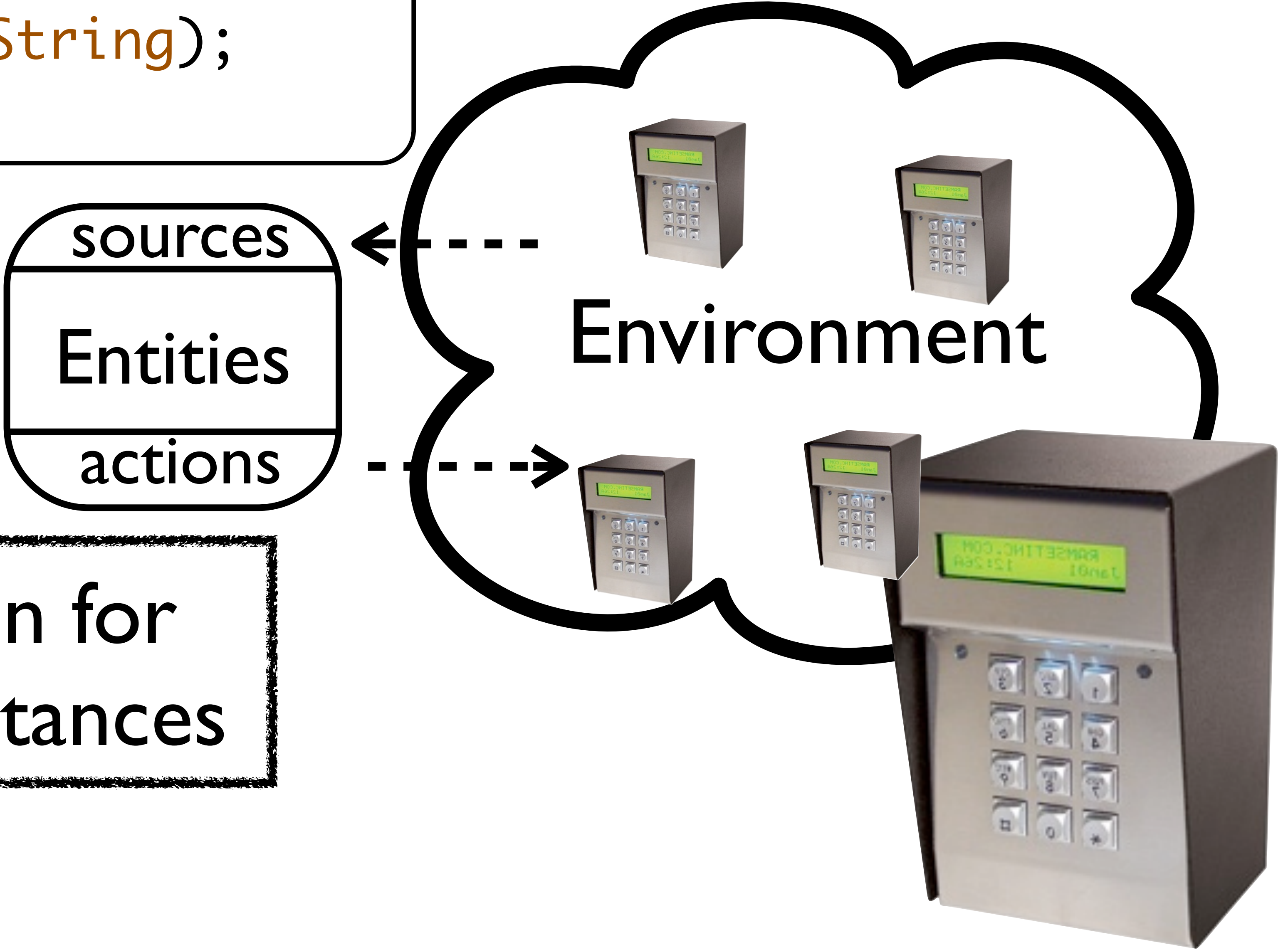
1 entity description for
potentially many
implementations

Design Language

```
entity Keypad {
 source keycode as Integer;
 action UpdateSt;
}

action UpdateSt {
 updateStatus(message as String);
}
```

Architect

a concept for handling the interaction with the environment

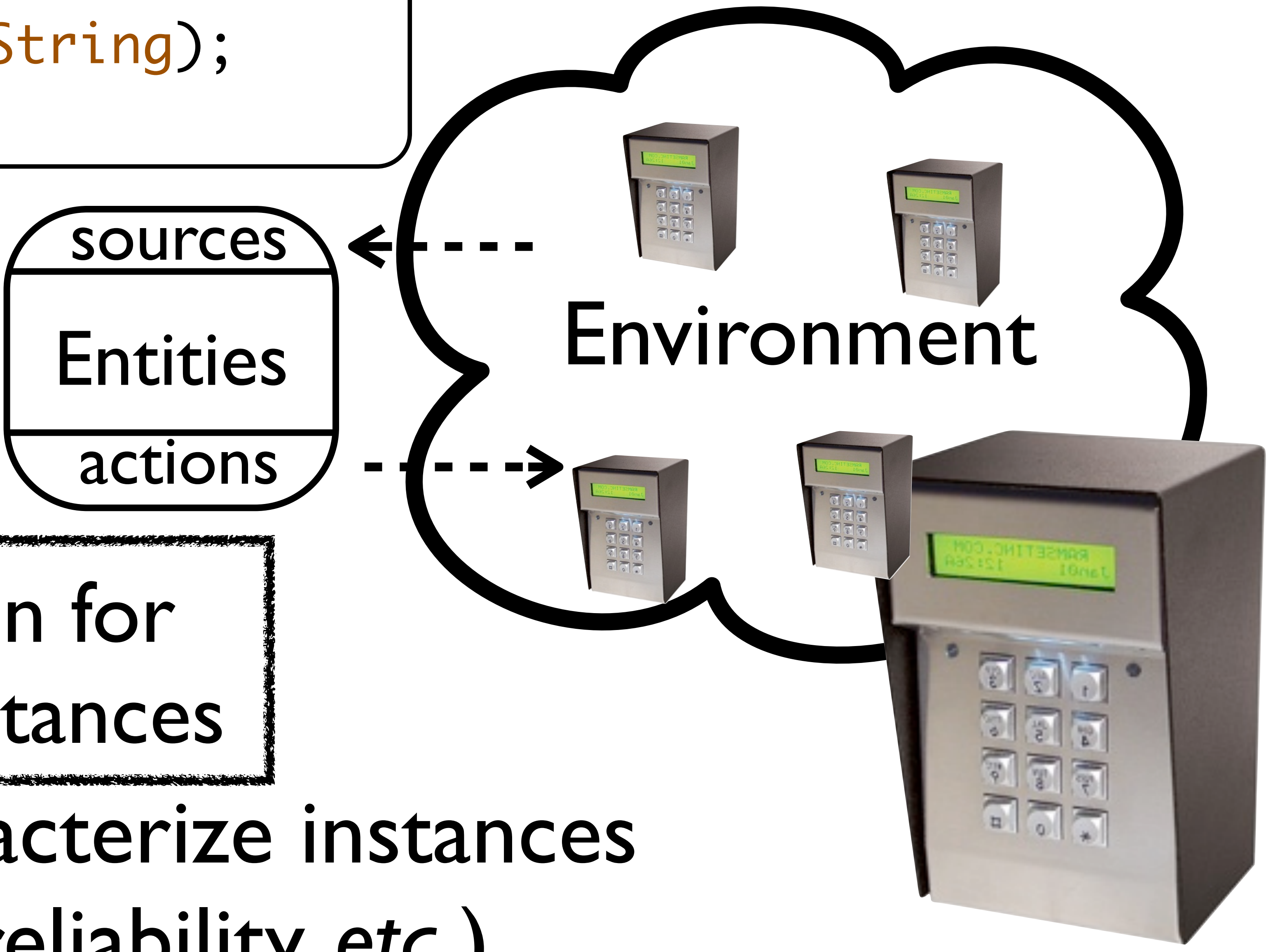


1 entity description for potentially many instances

Design Language

```
entity Keypad {
 source keycode as Integer;
 action UpdateSt;
 attribute room as Integer;
}
action UpdateSt {
 updateStatus(message as String);
}
```

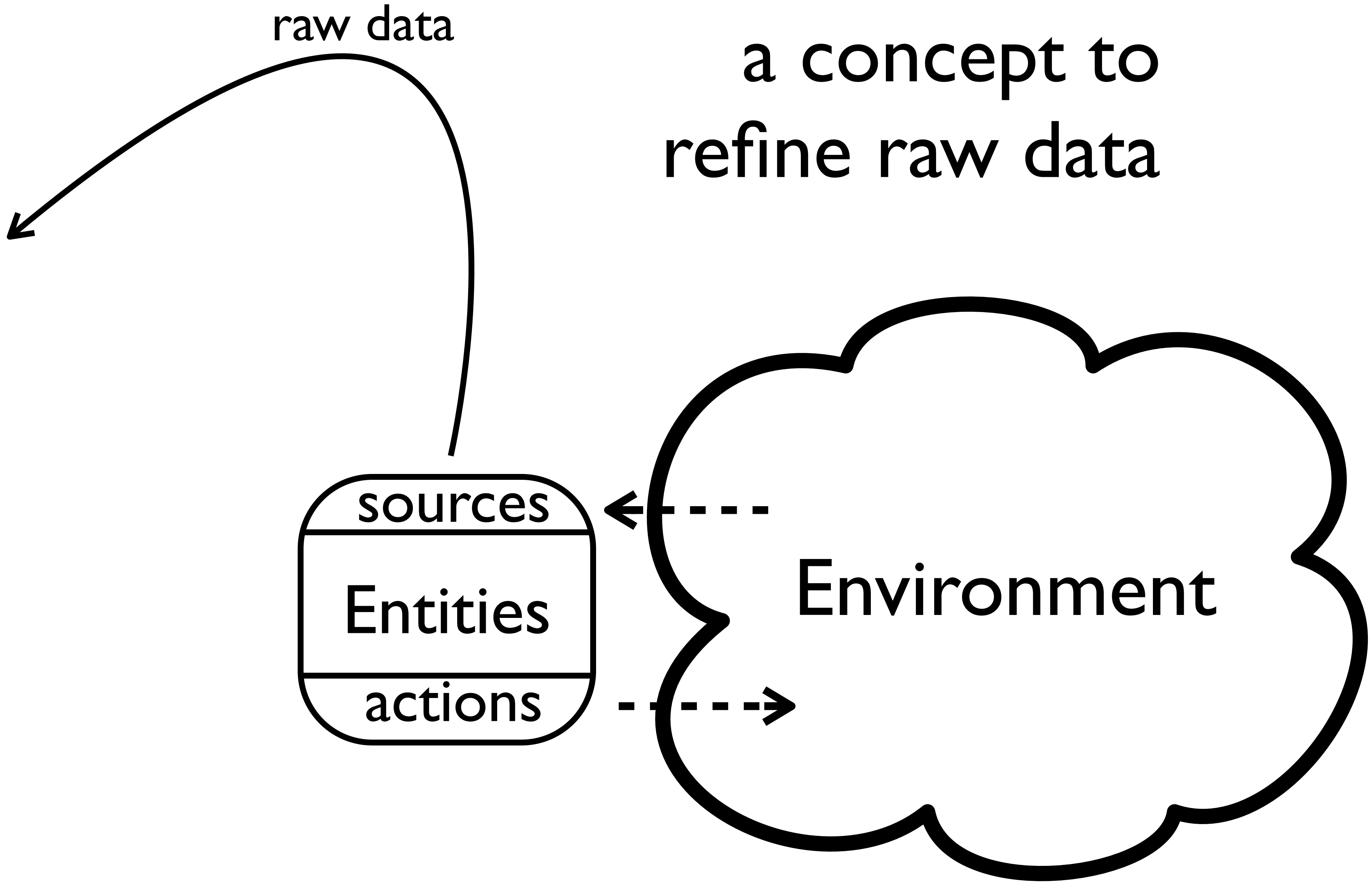
Architect



1 entity description for potentially many instances

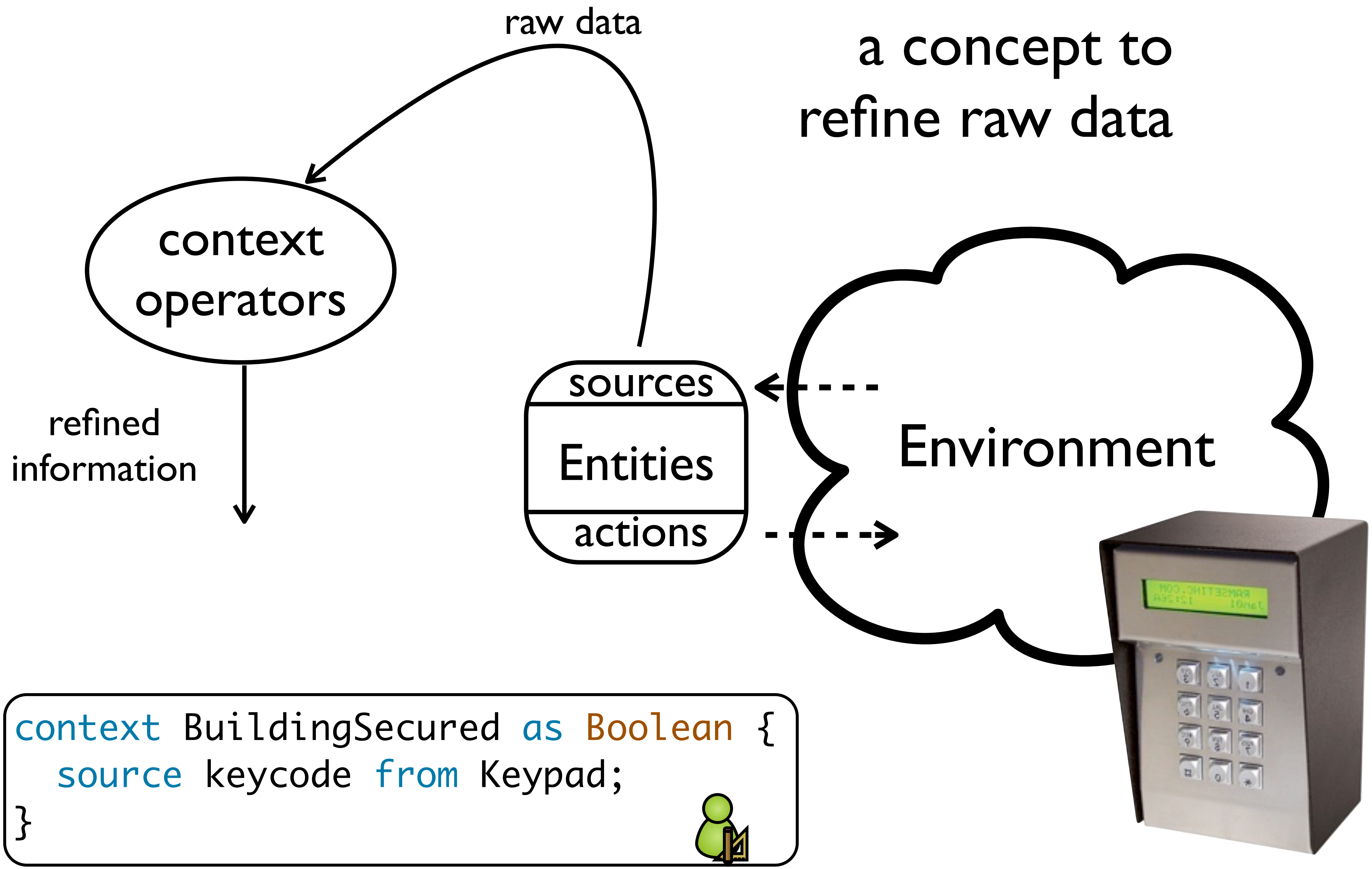
➡ attributes to characterize instances (color, location, reliability, *etc.*)

Design Language



a concept to
refine raw data

Design Language



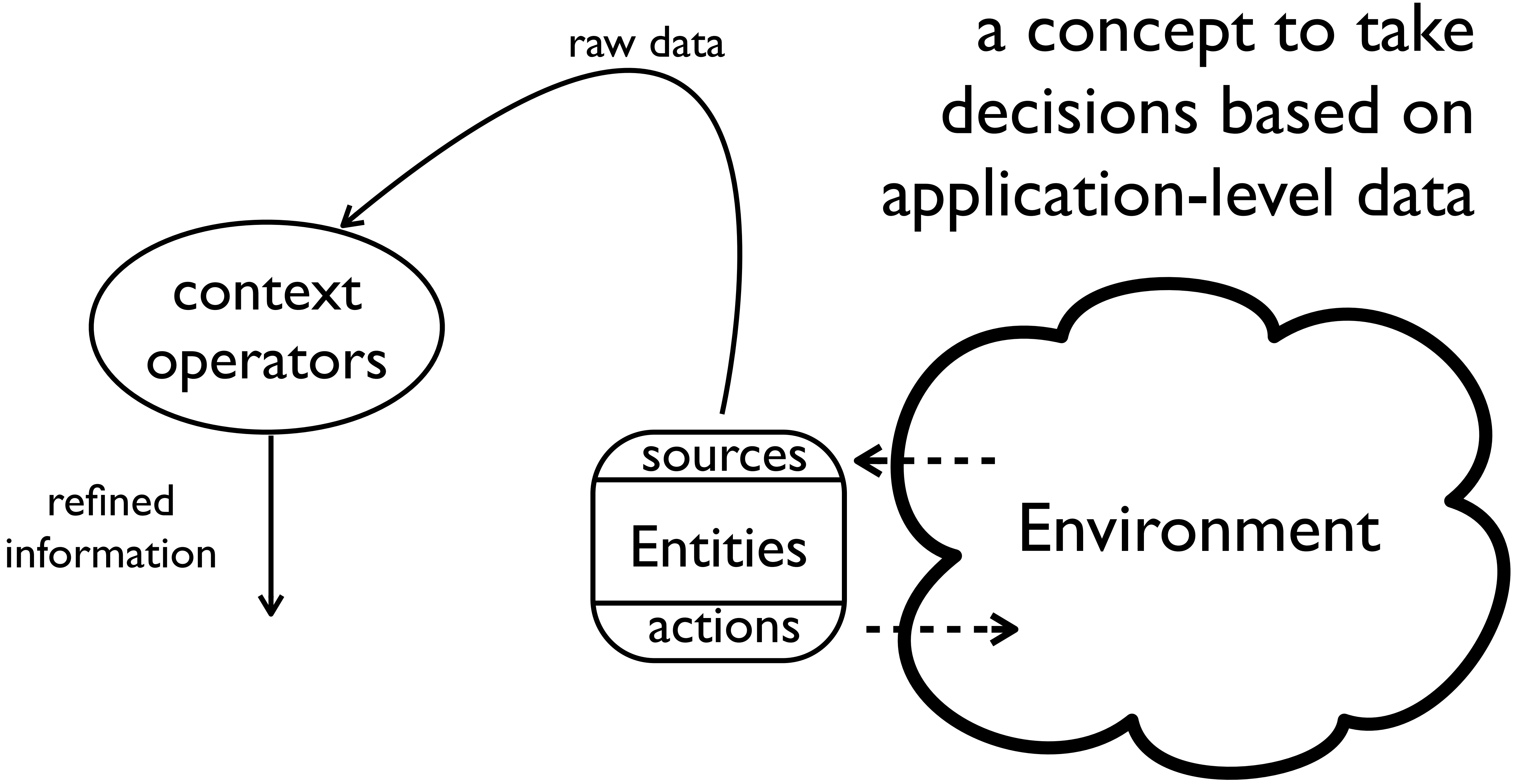
a concept to refine raw data

```
context BuildingSecured as Boolean {
  source keycode from Keypad;
}
```

Architect

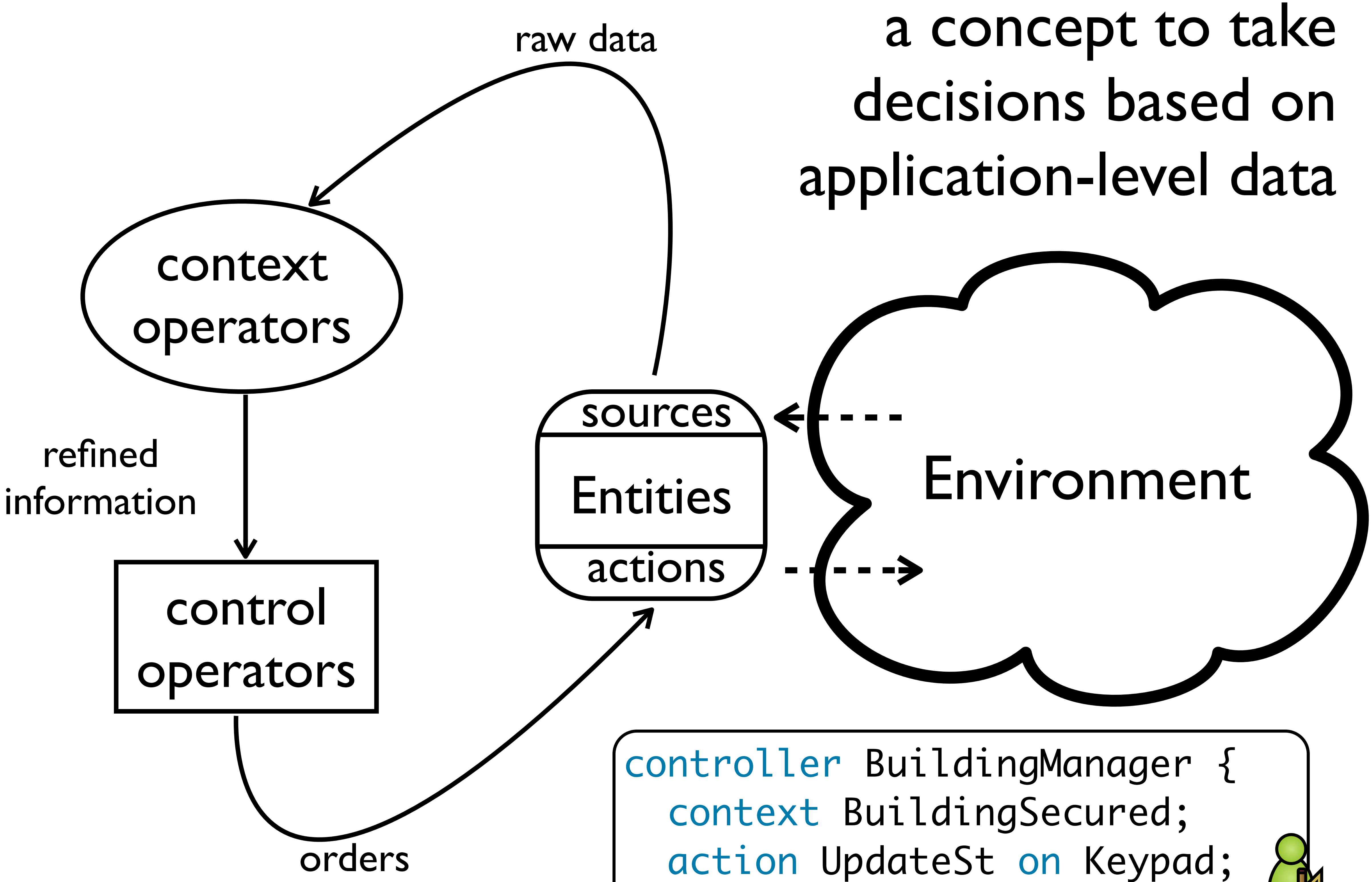
Design Language

a concept to take decisions based on application-level data



Design Language

a concept to take decisions based on application-level data

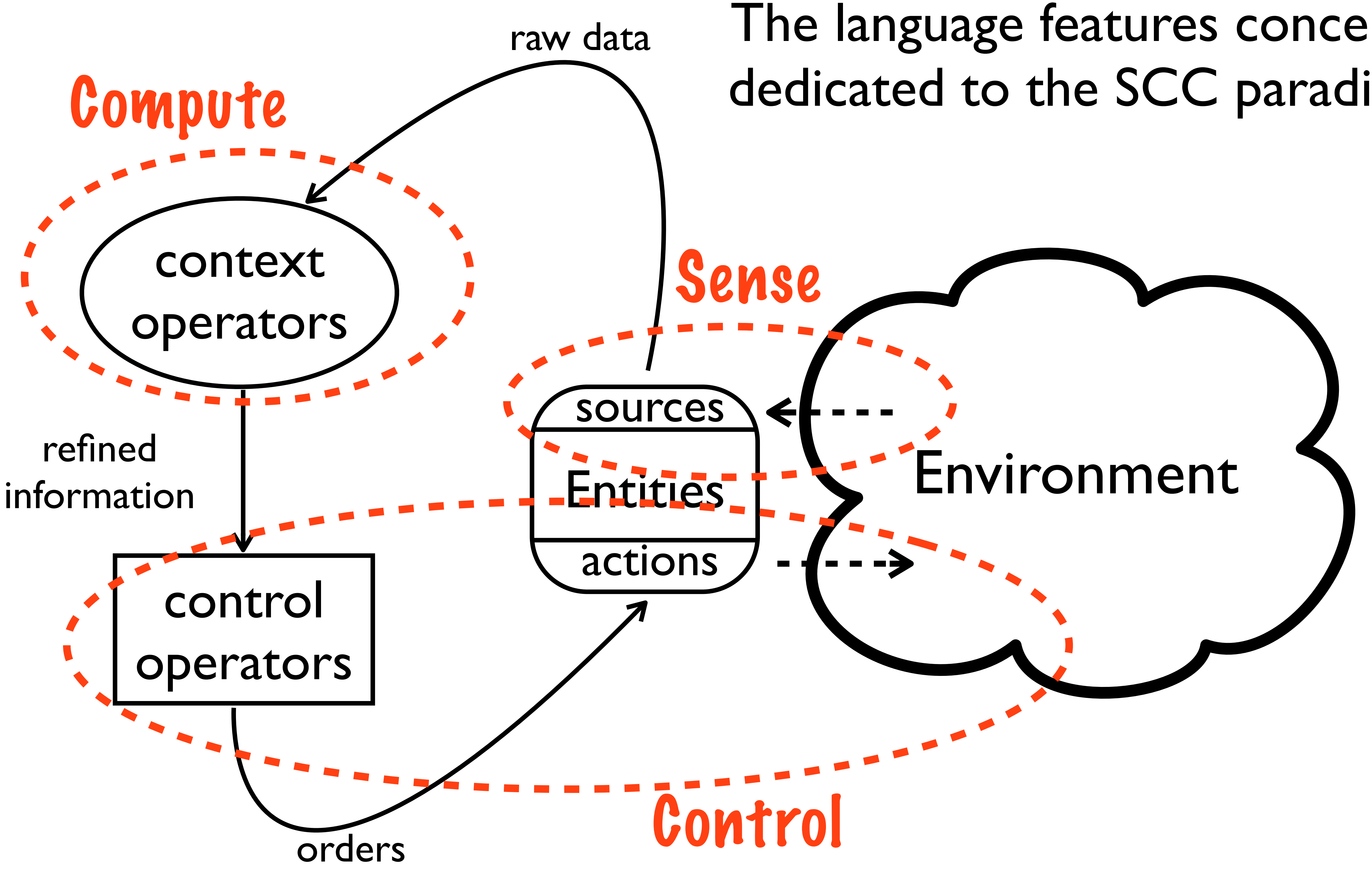


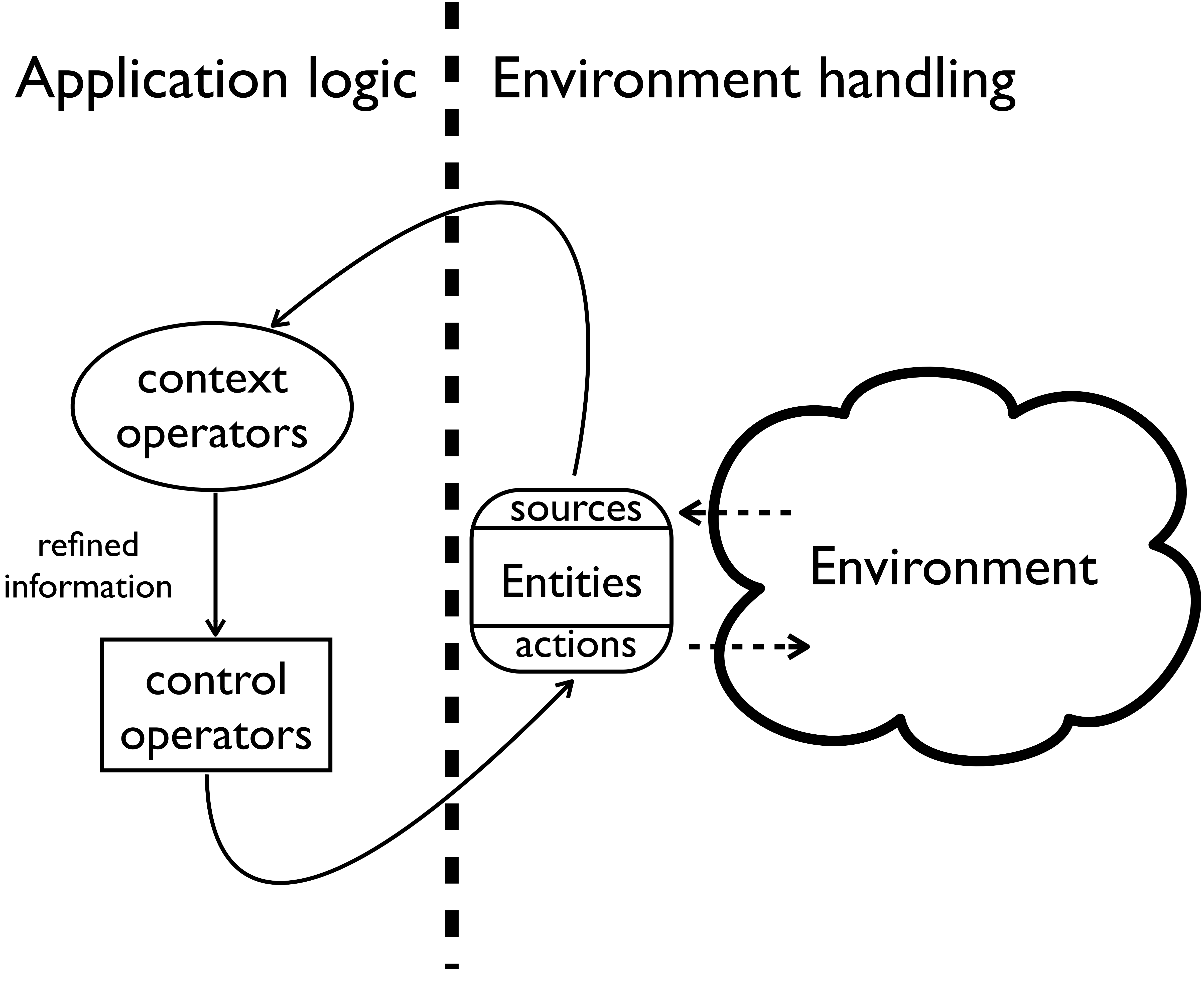
```
controller BuildingManager {
  context BuildingSecured;
  action UpdateSt on Keypad;
}
```

Architect

Design Language

The language features concepts dedicated to the SCC paradigm





Application logic

Environment handling

context operators

refined information

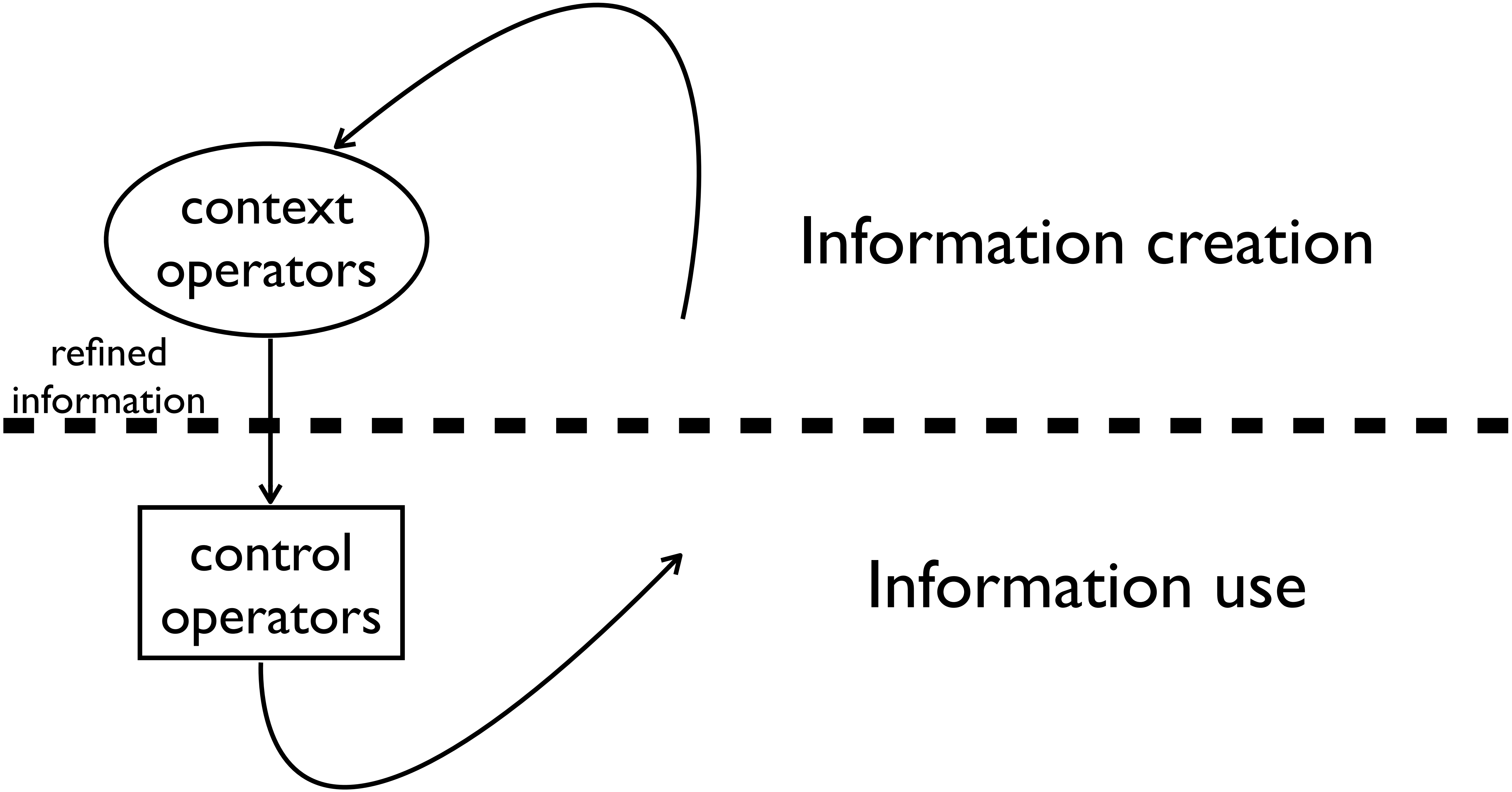
control operators

sources

Entities

actions

Environment



context operators

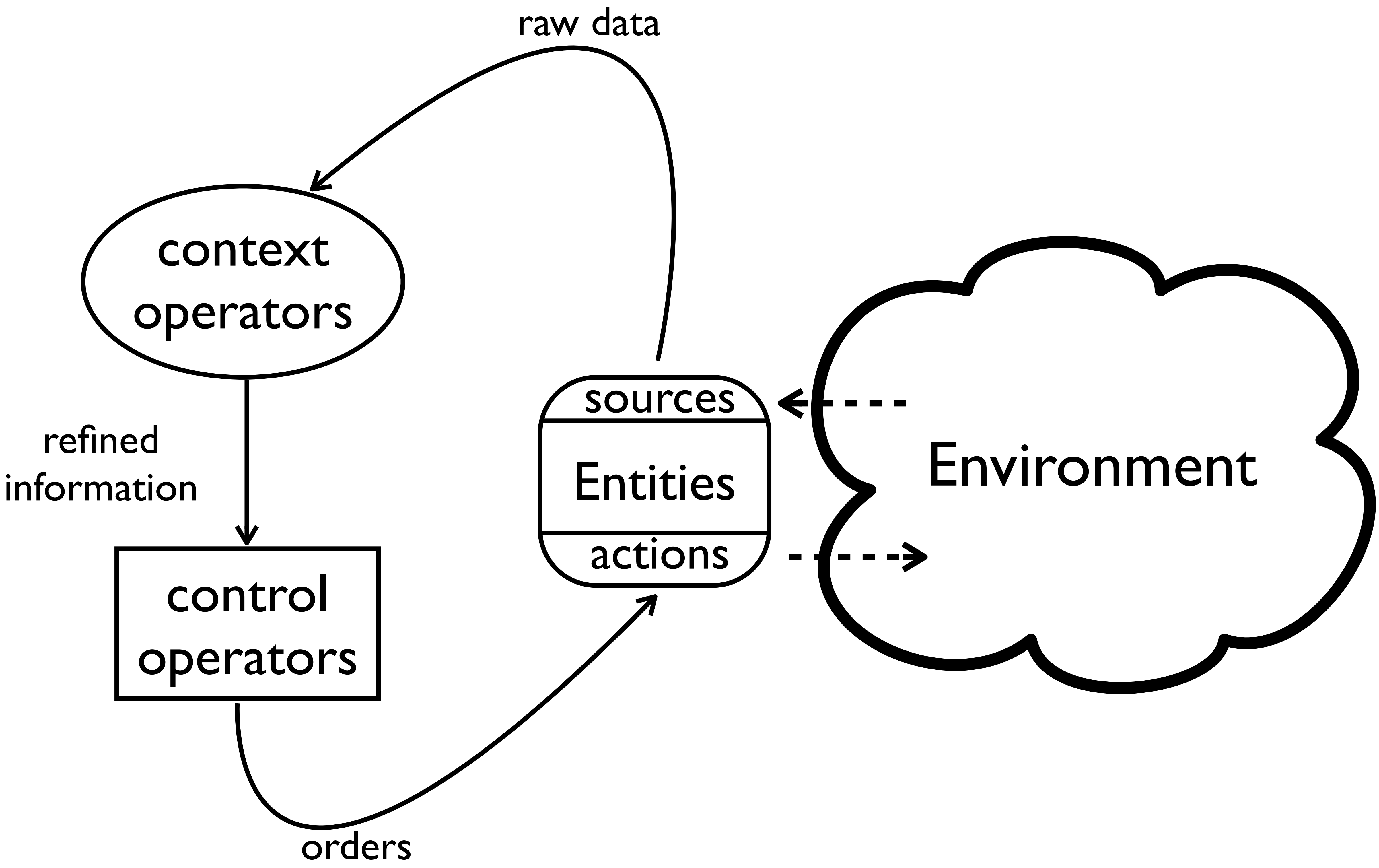
refined information

control operators

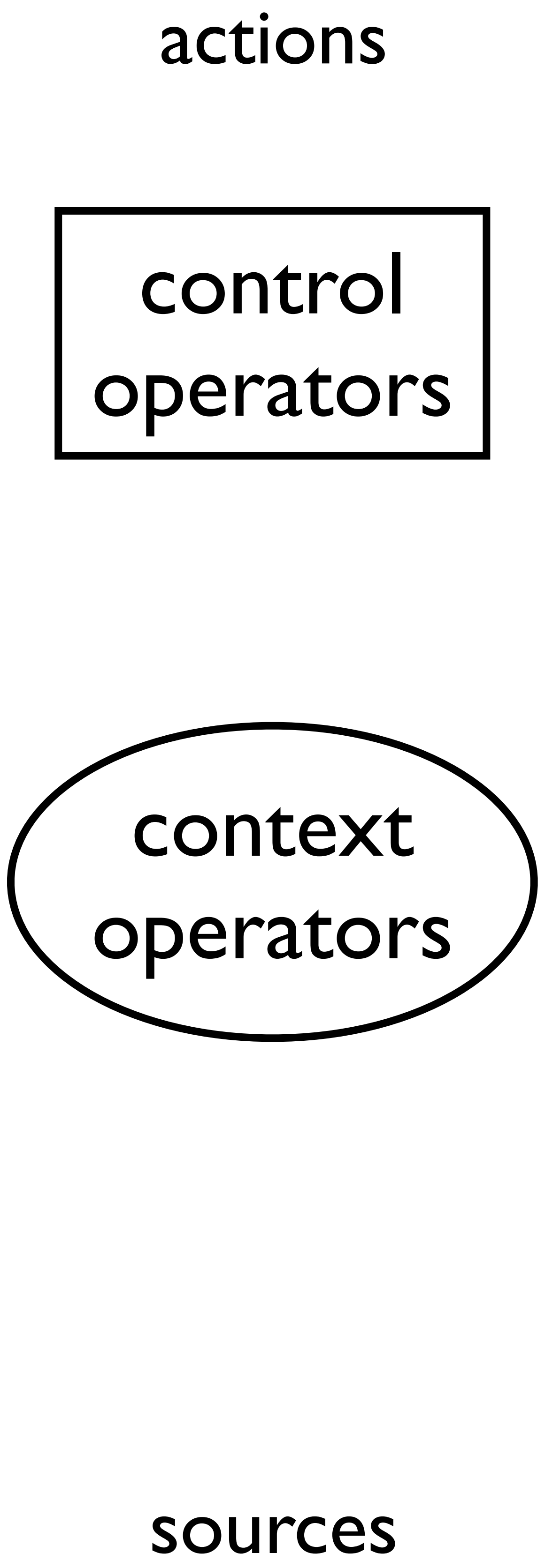
Information creation

Information use

Design Language



Design Language



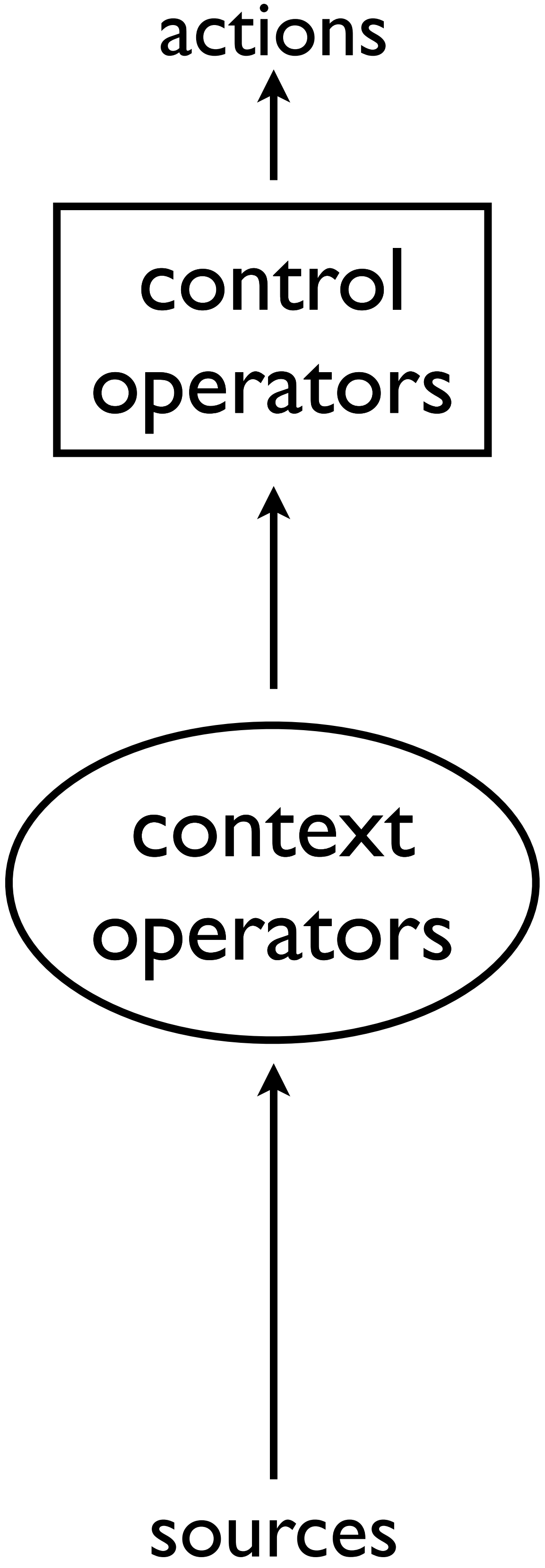
actions

control operators

context operators

sources

Design Language



actions

control operators

context operators

sources

Case Study: Anti-Intrusion

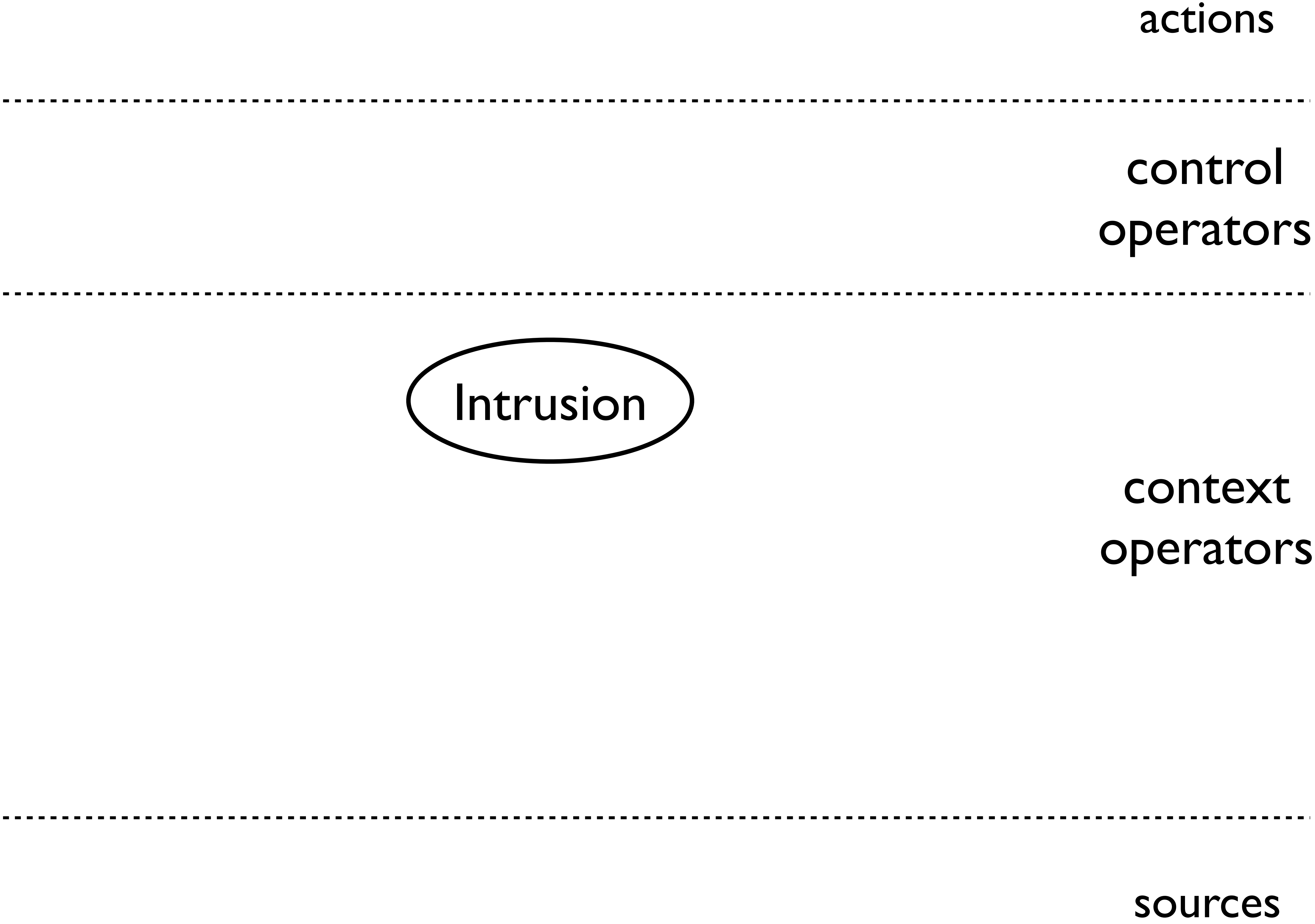
actions

control operators

context operators

sources

Case Study: Anti-Intrusion



actions

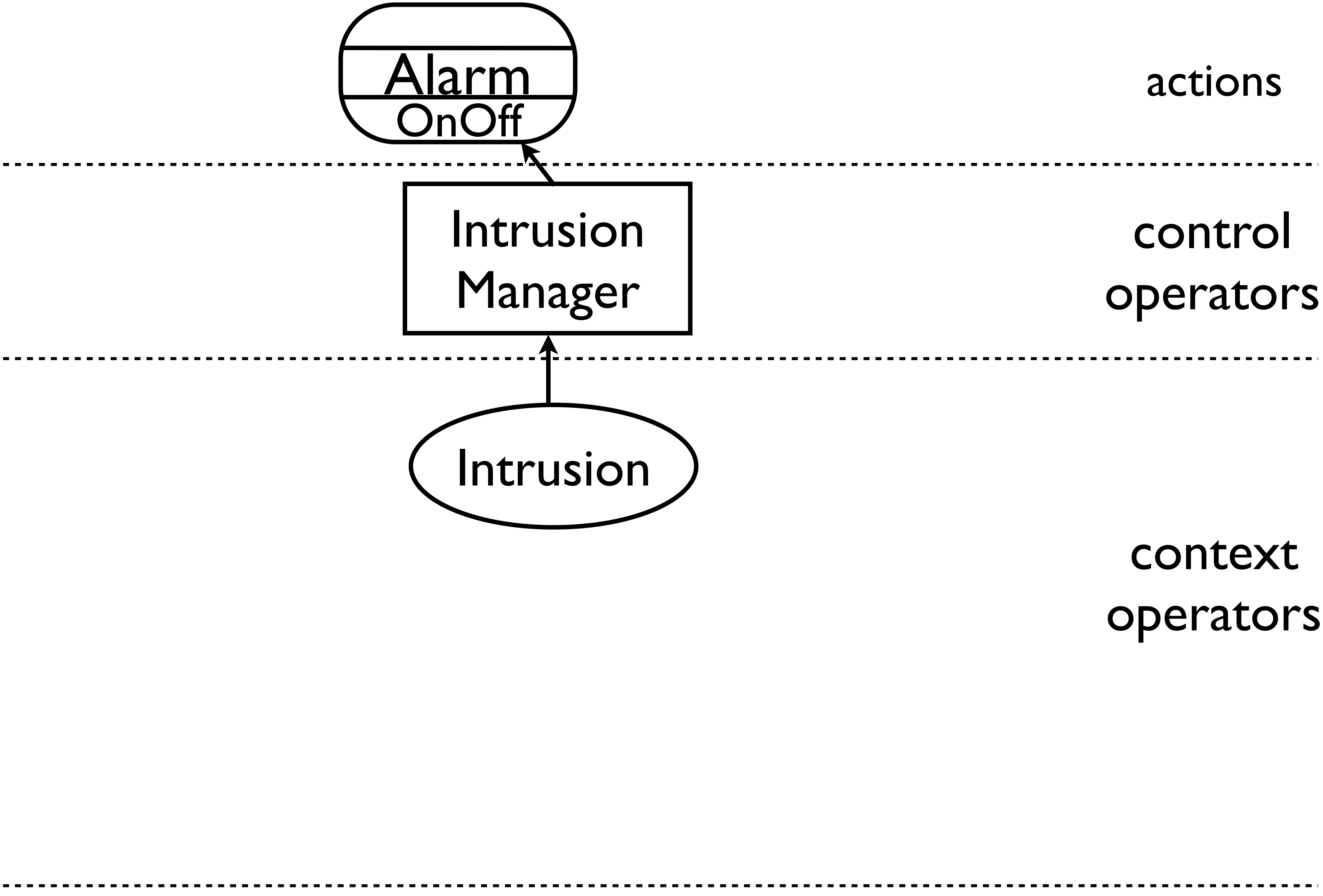
control operators

Intrusion

context operators

sources

Case Study: Anti-Intrusion



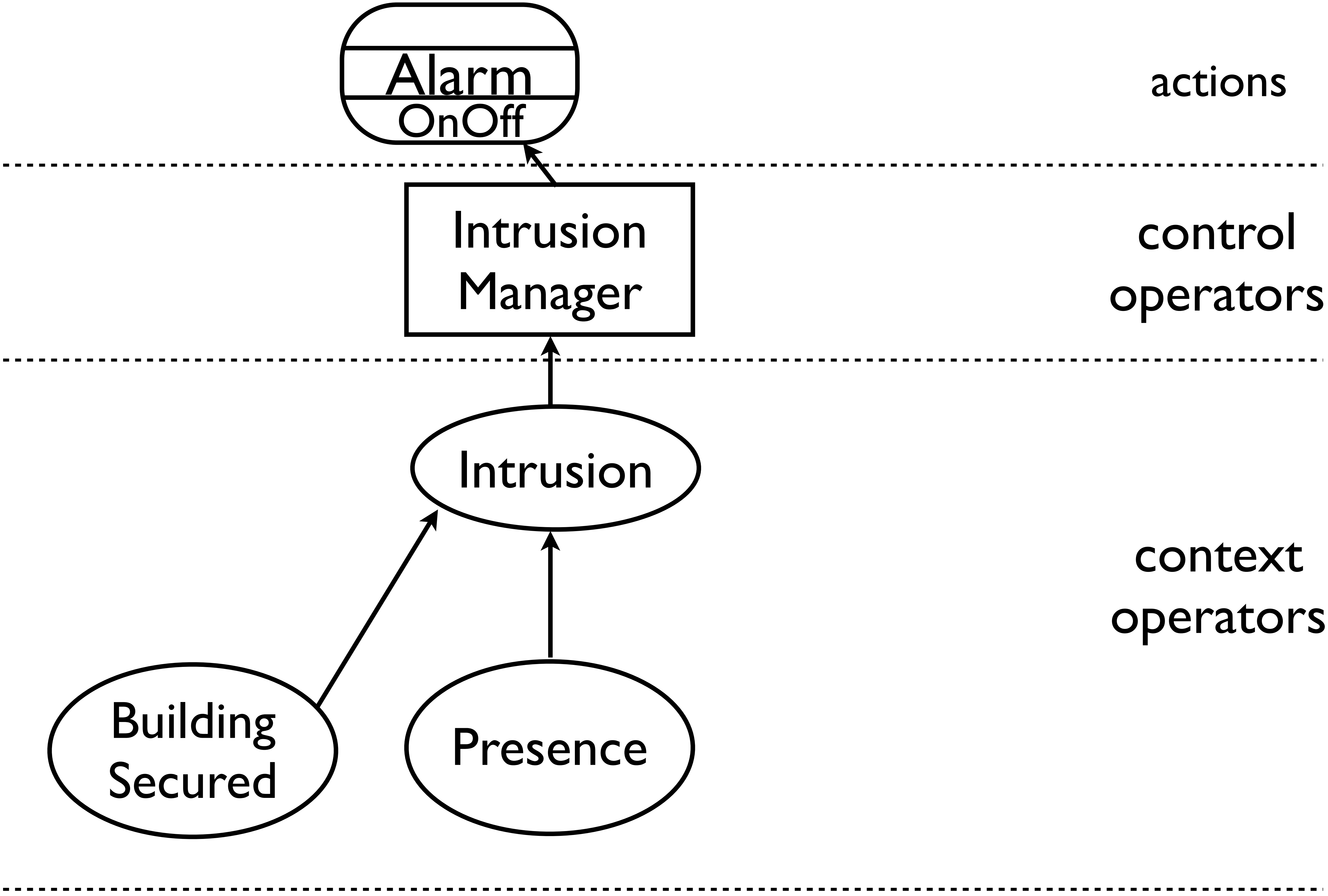
actions

control operators

context operators

sources

Case Study: Anti-Intrusion



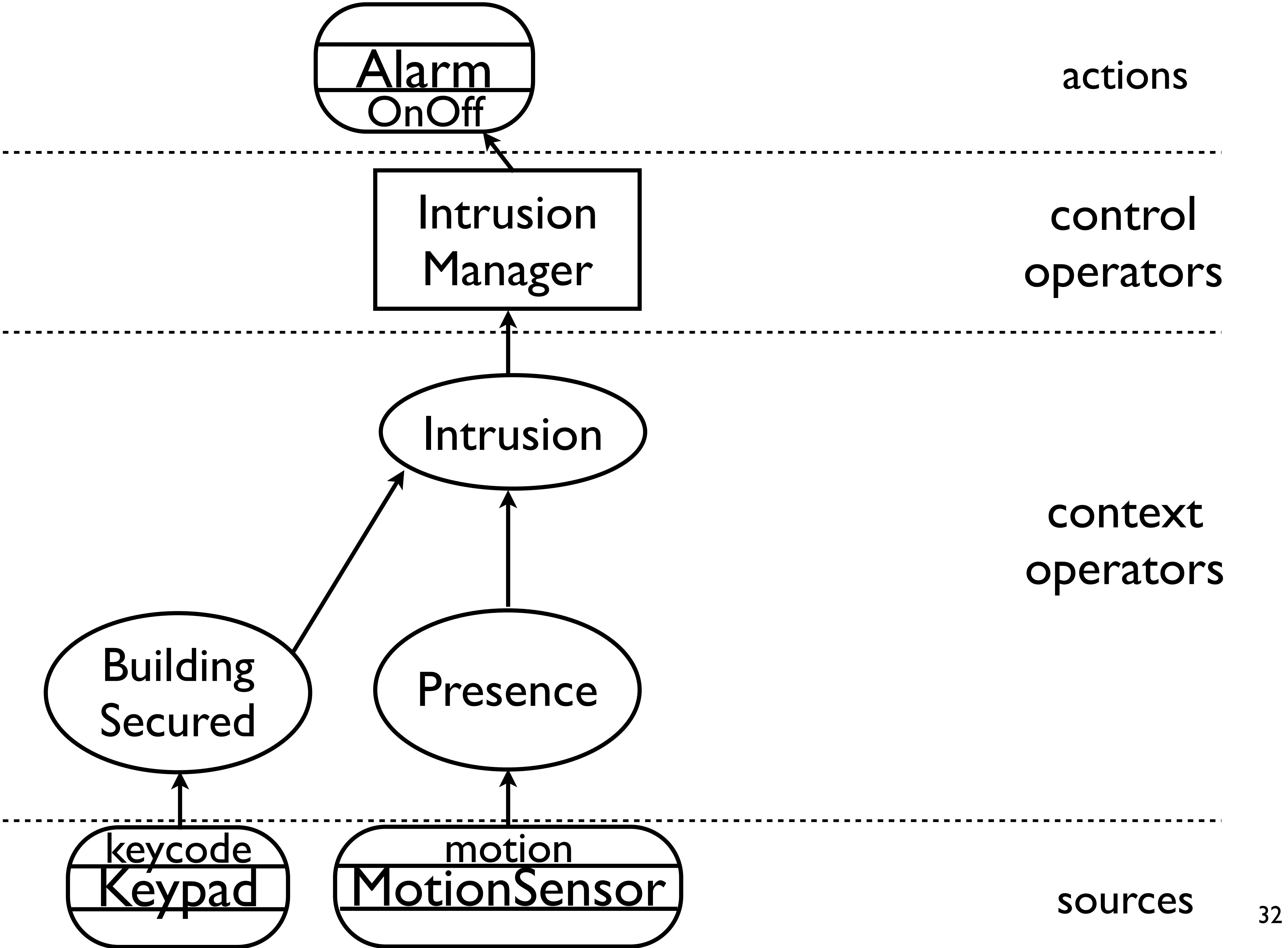
actions

control operators

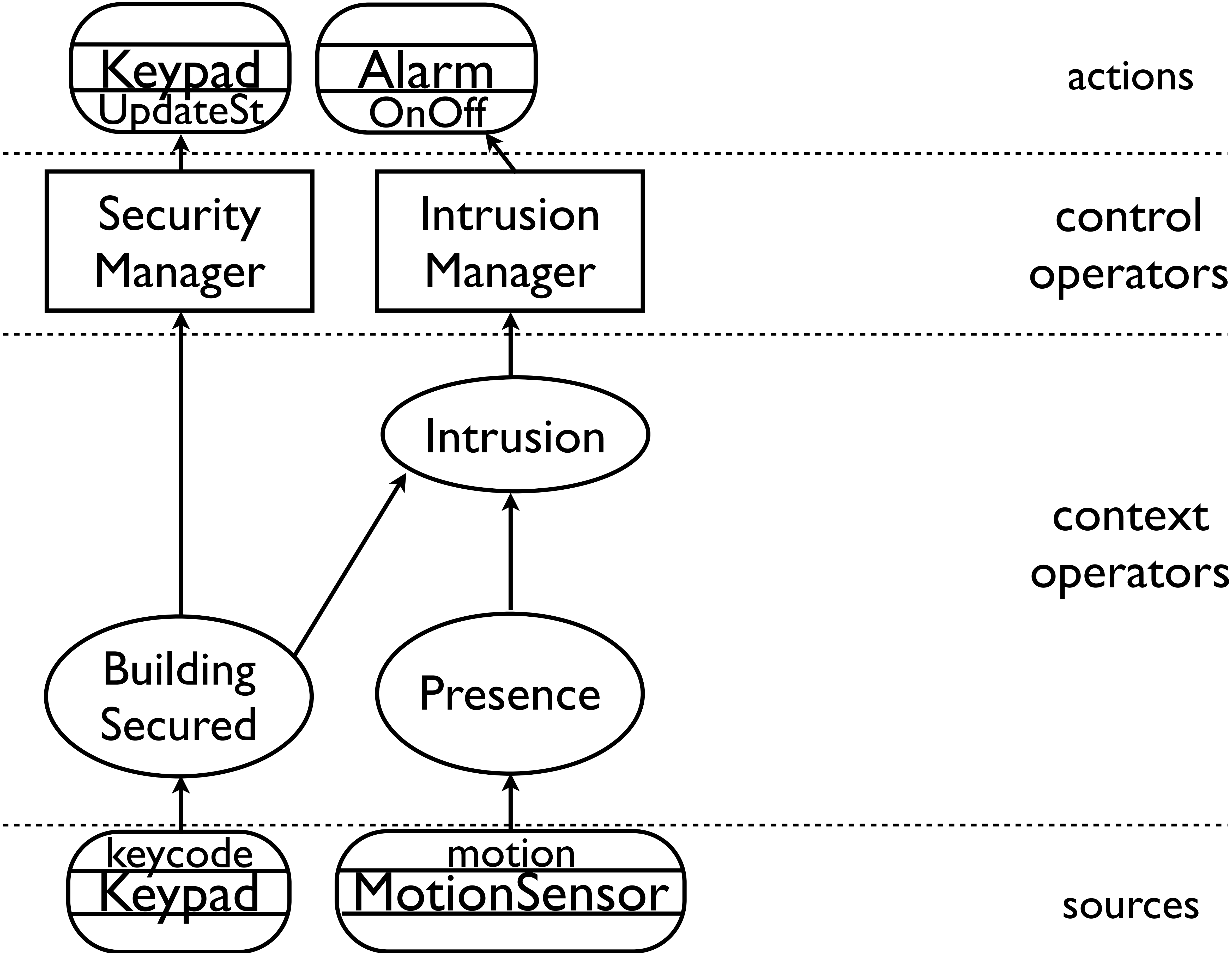
context operators

sources

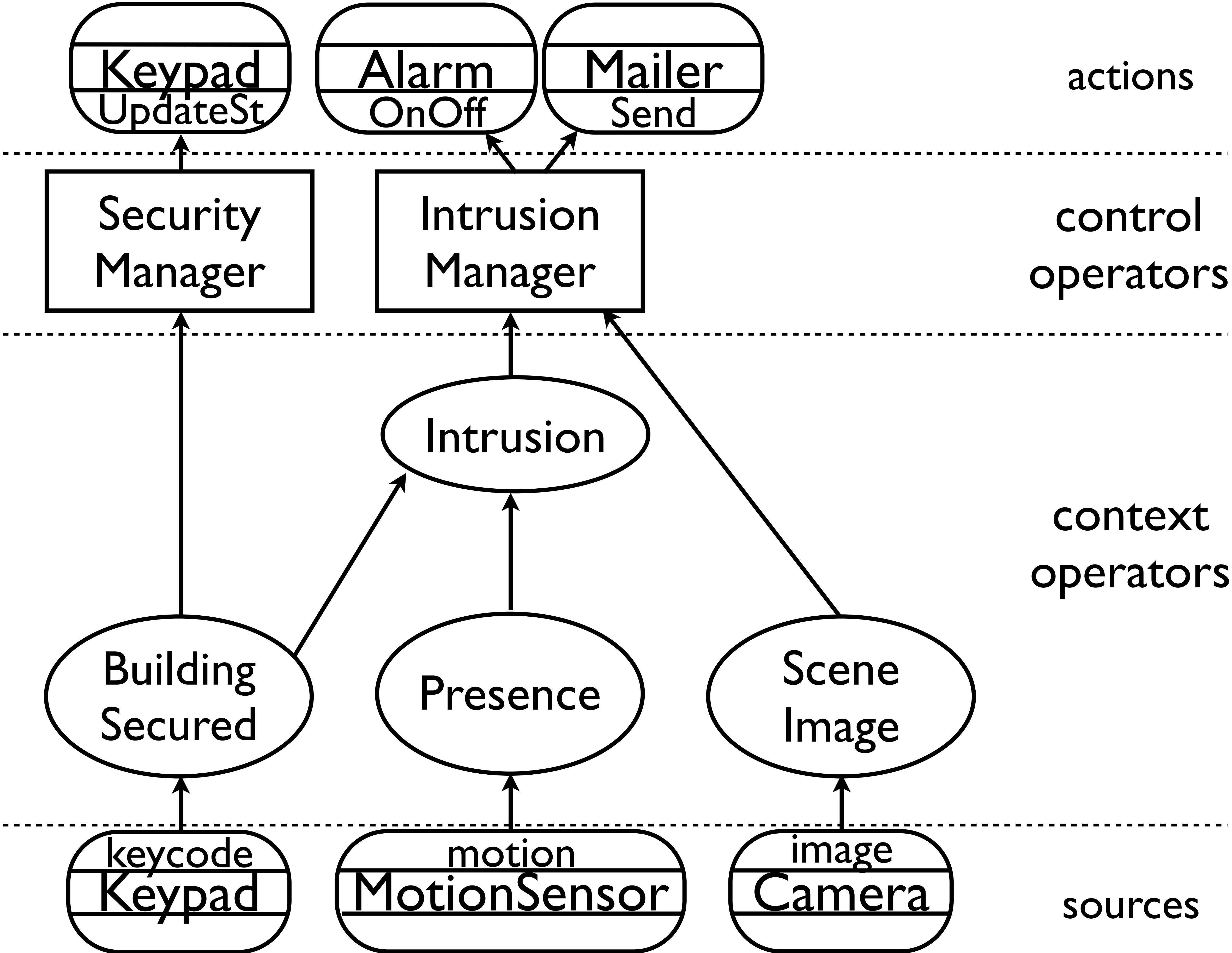
Case Study: Anti-Intrusion



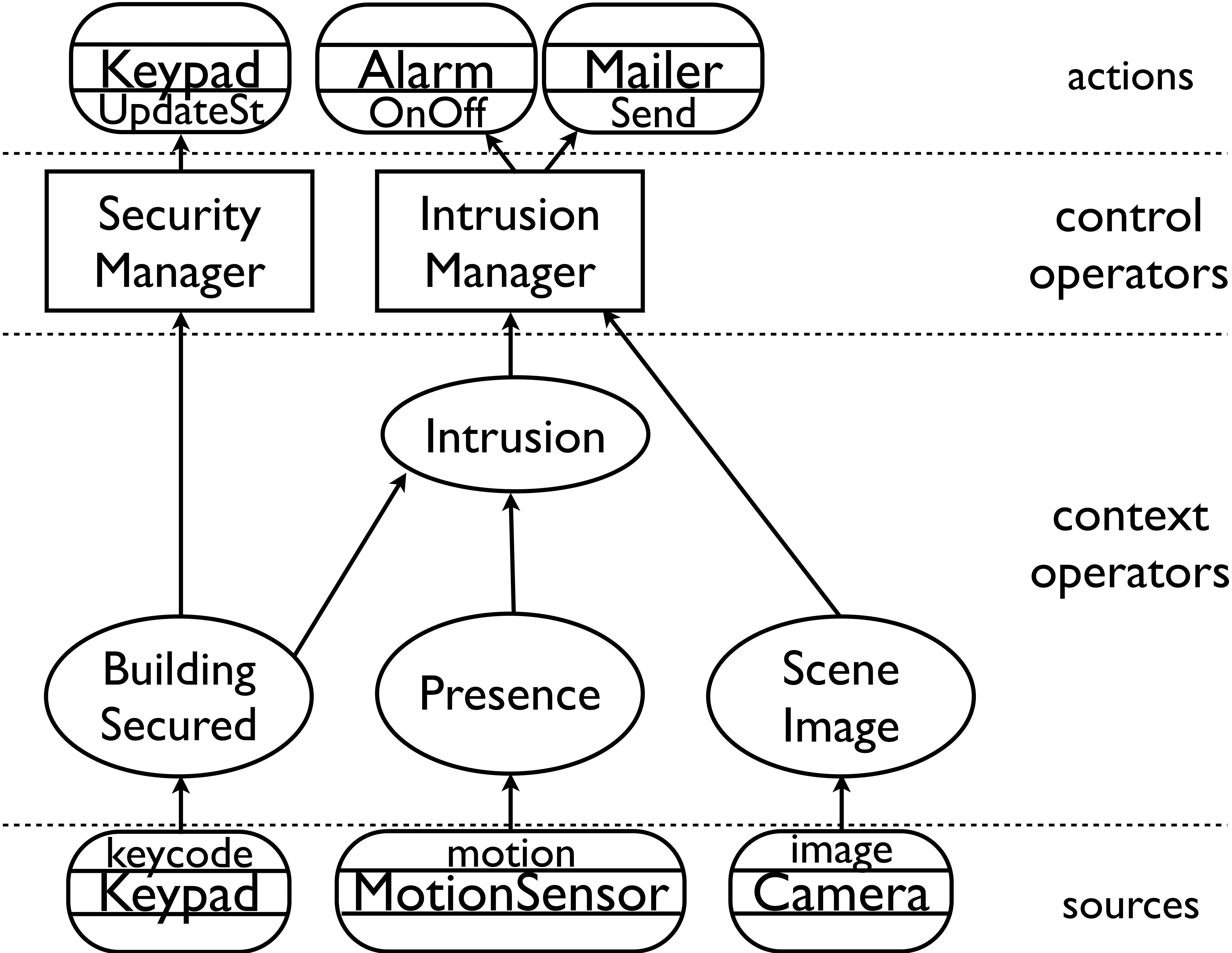
Case Study: Anti-Intrusion



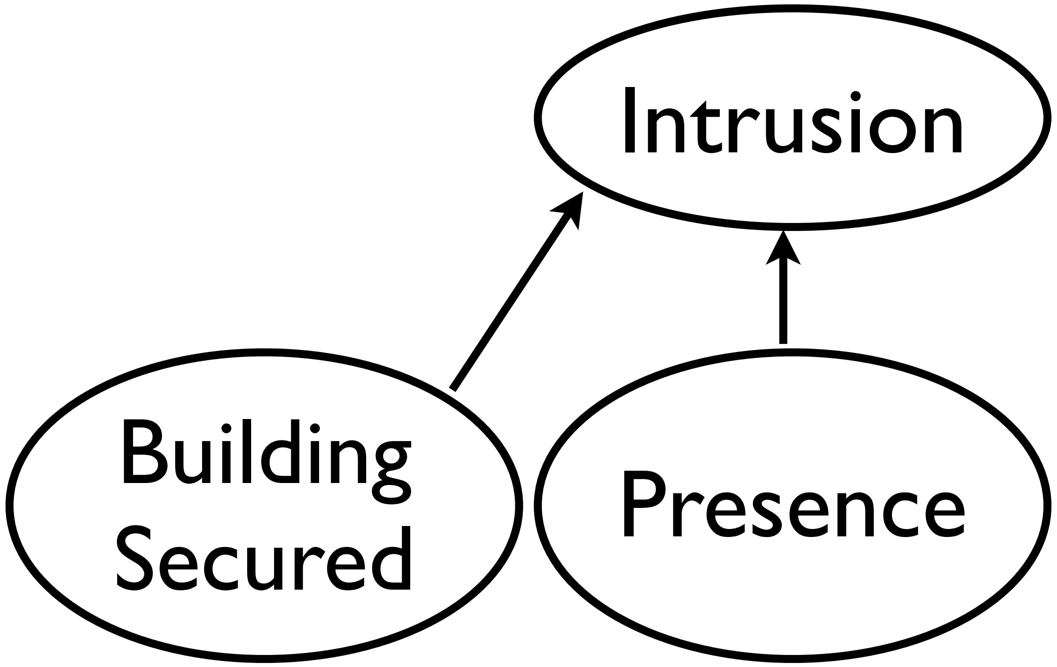
Case Study: Anti-Intrusion



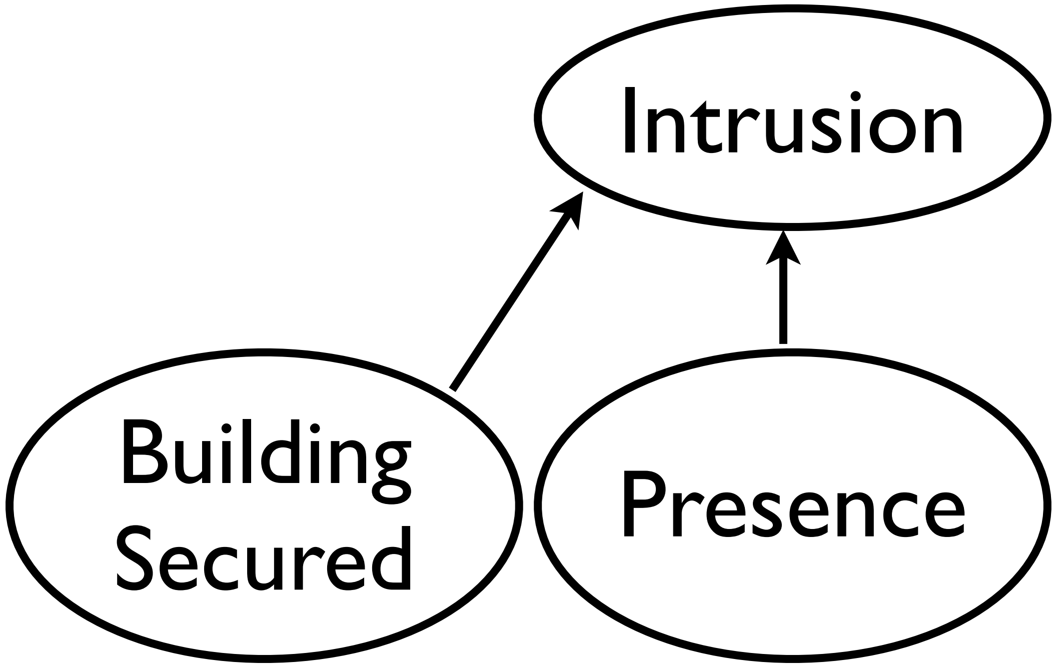
Case Study: Anti-Intrusion

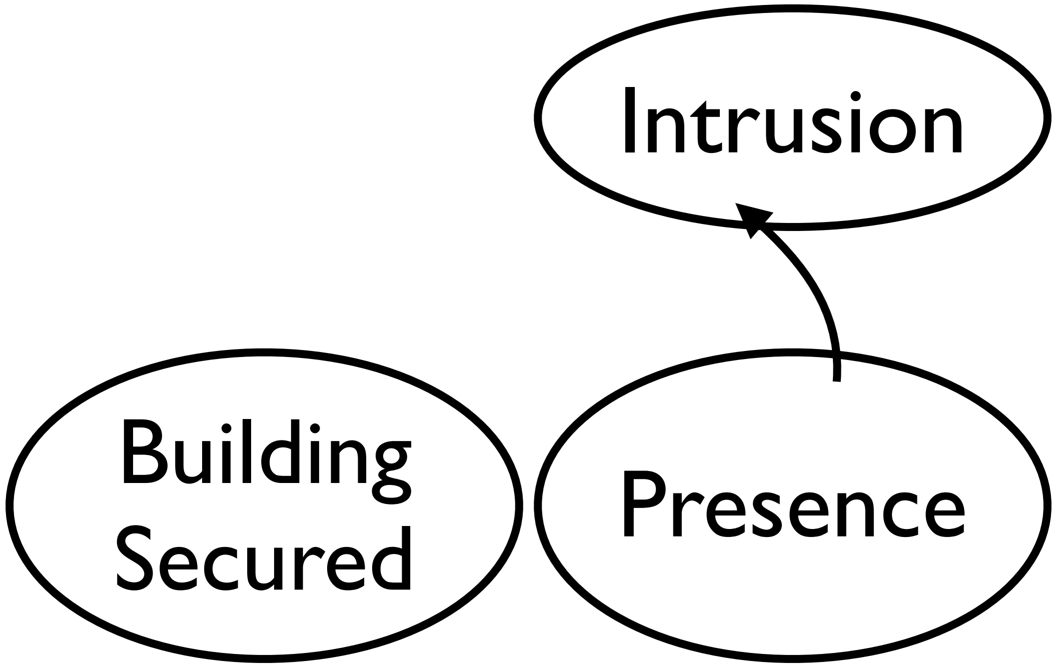


Case Study: Anti-Intrusion



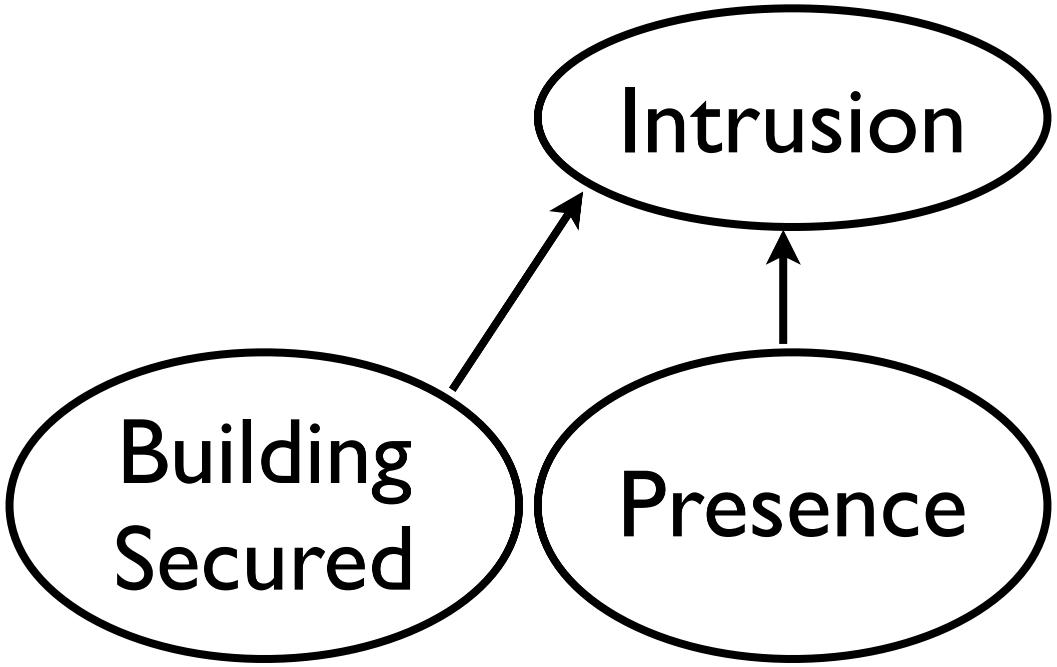
Case Study: Anti-Intrusion

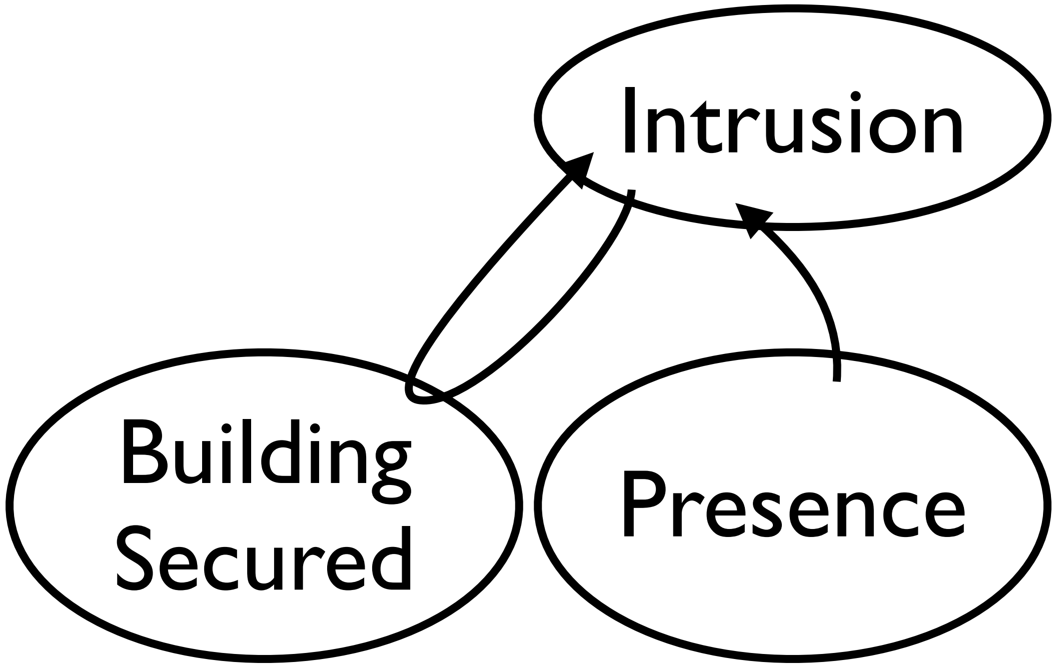




what the architect
has in mind

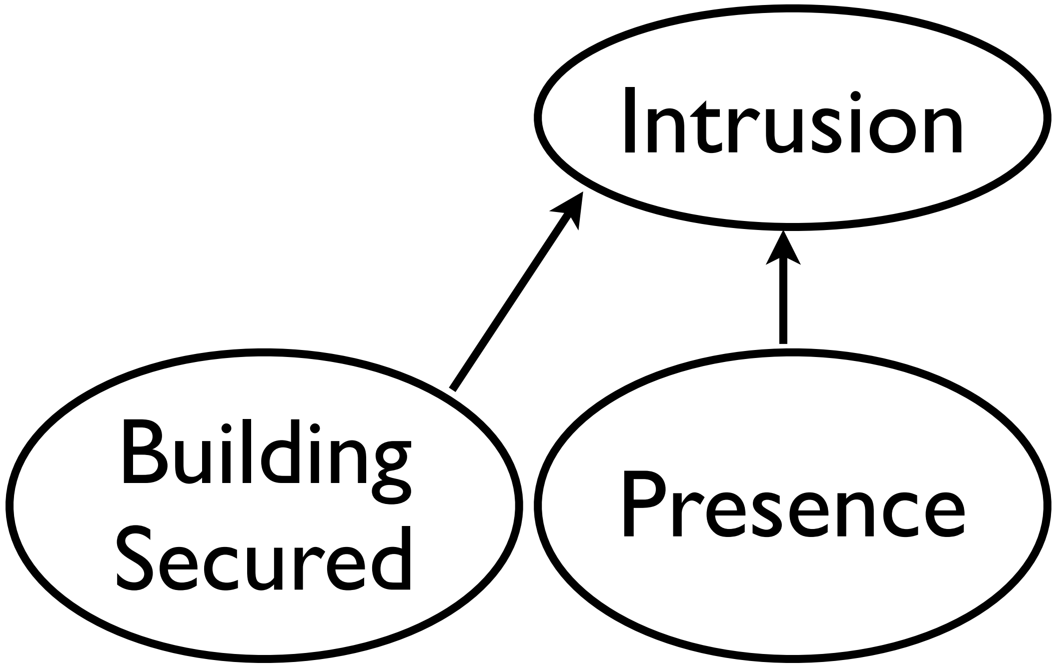
Case Study: Anti-Intrusion





what the architect has in mind

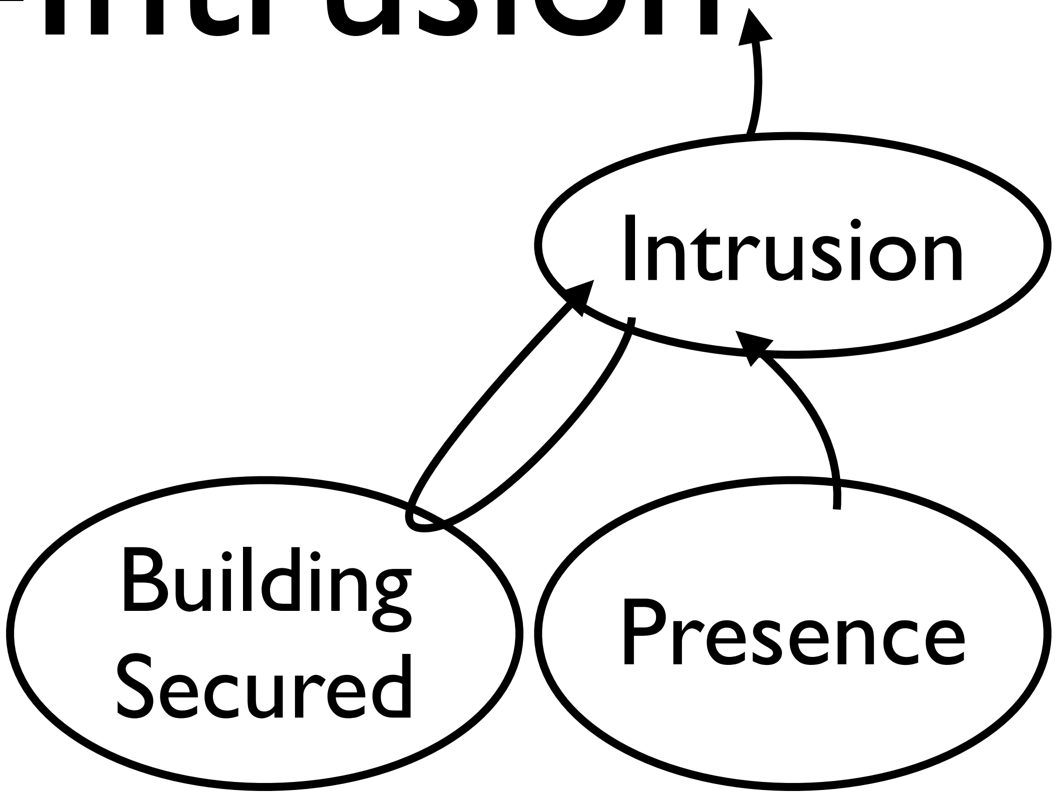
Case Study: Anti-Intrusion



Intrusion

Building Secured

Presence



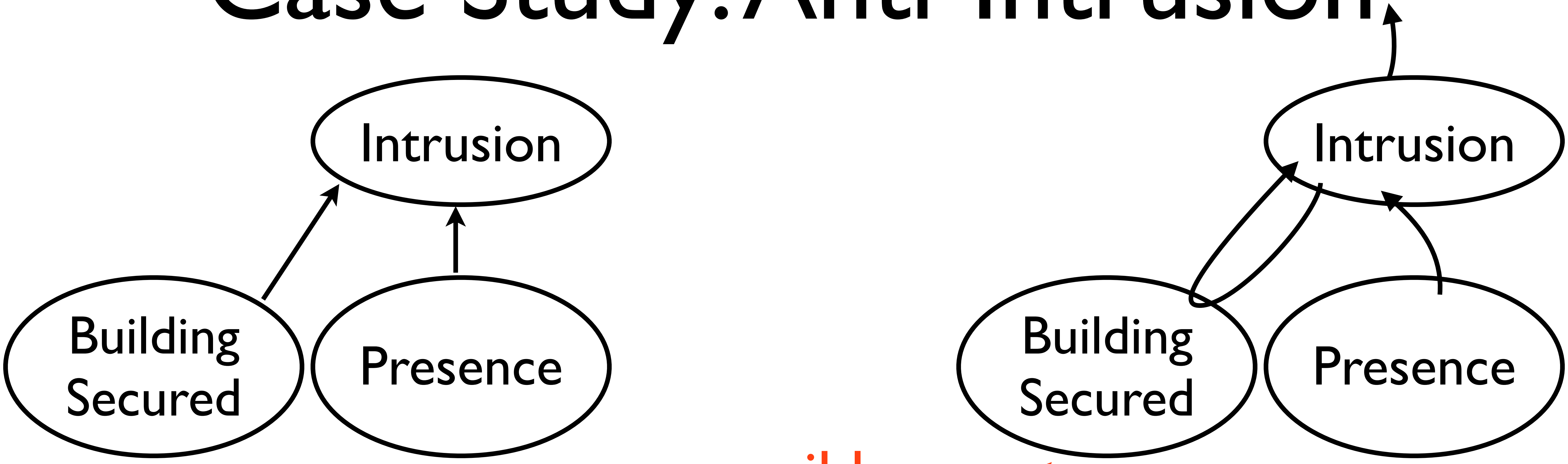
Intrusion

Building Secured

Presence

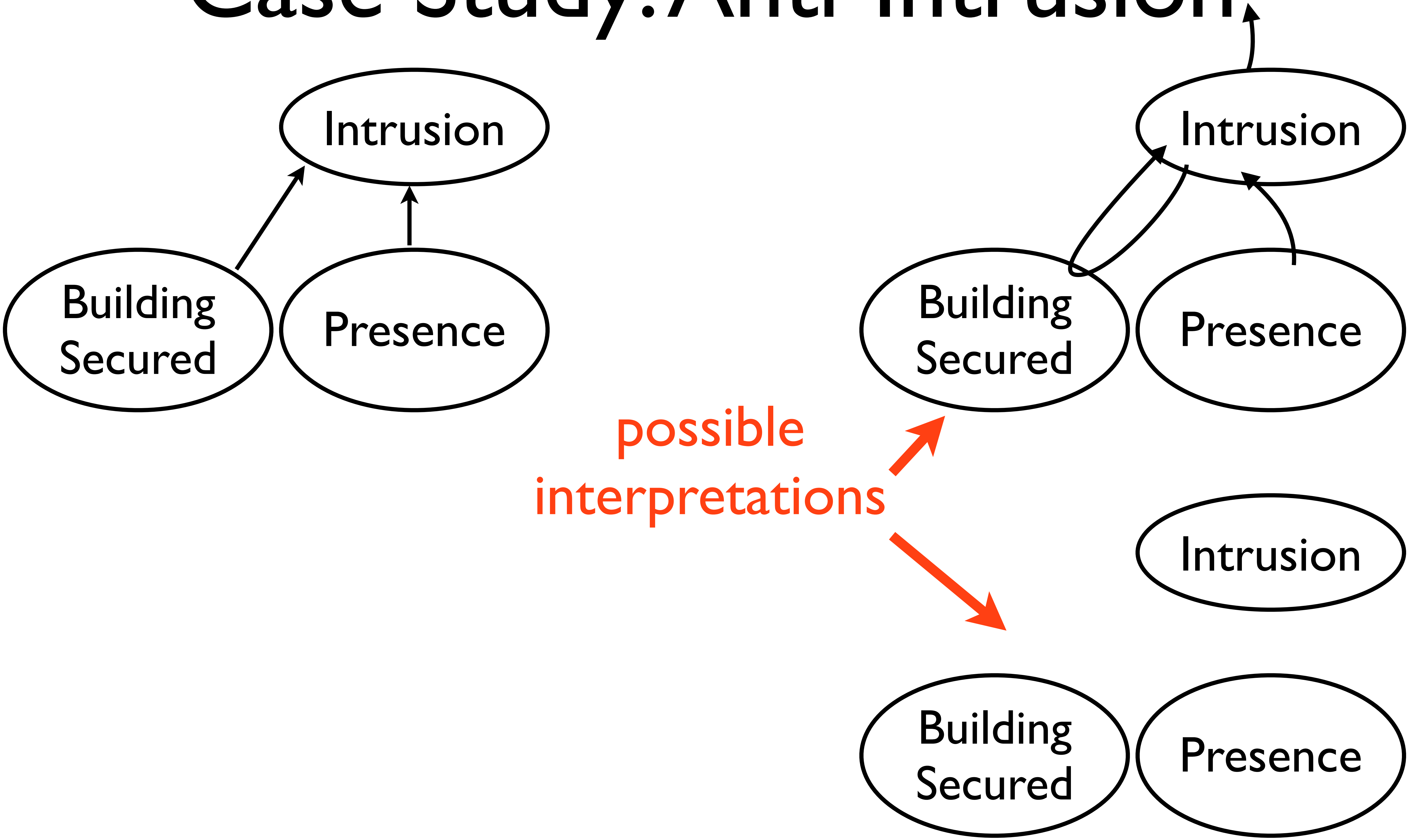
what the architect has in mind

Case Study: Anti-Intrusion



possible interpretations

Case Study: Anti-Intrusion



Intrusion

Building Secured

Presence

possible interpretations

Intrusion

Building Secured

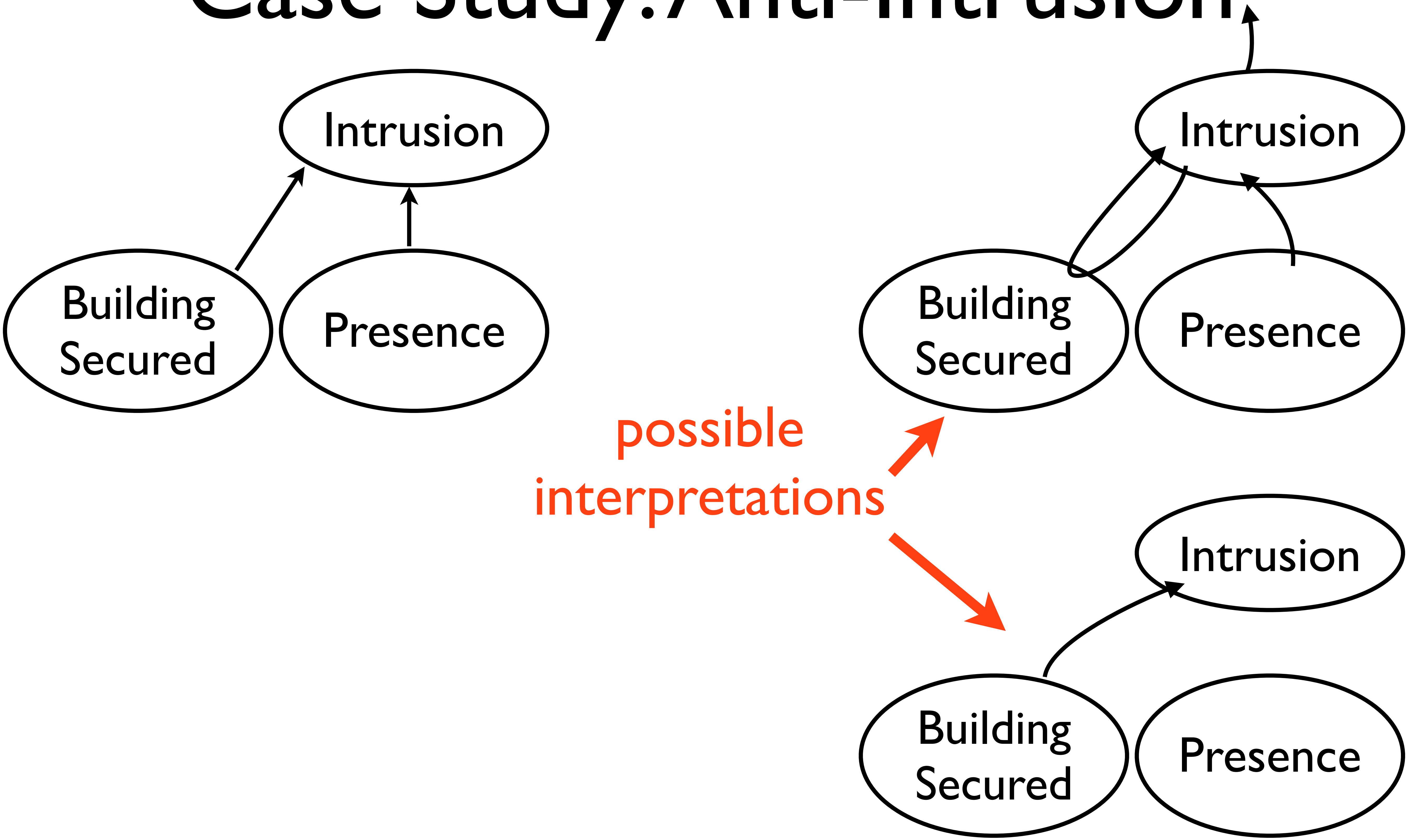
Presence

Intrusion

Building Secured

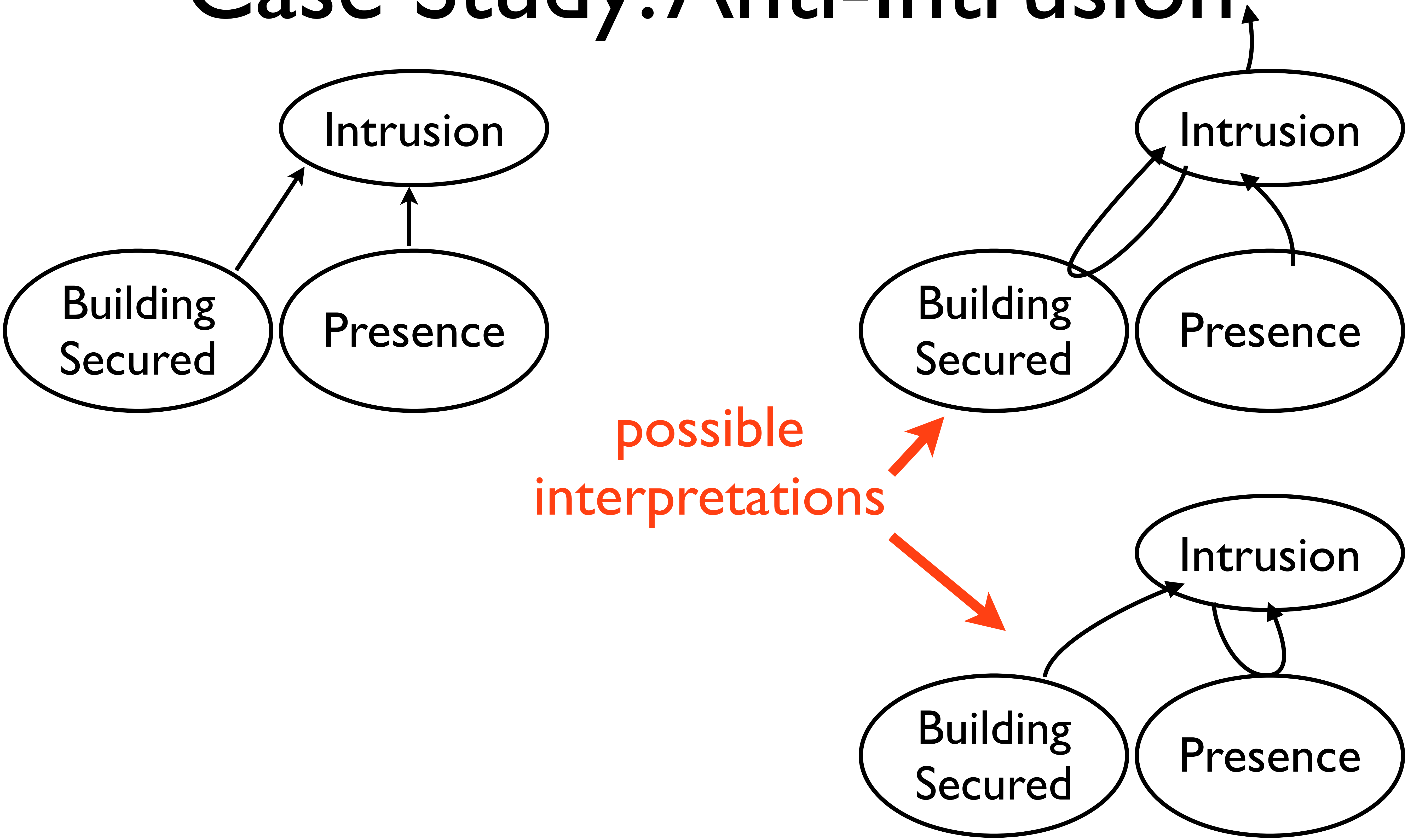
Presence

Case Study: Anti-Intrusion



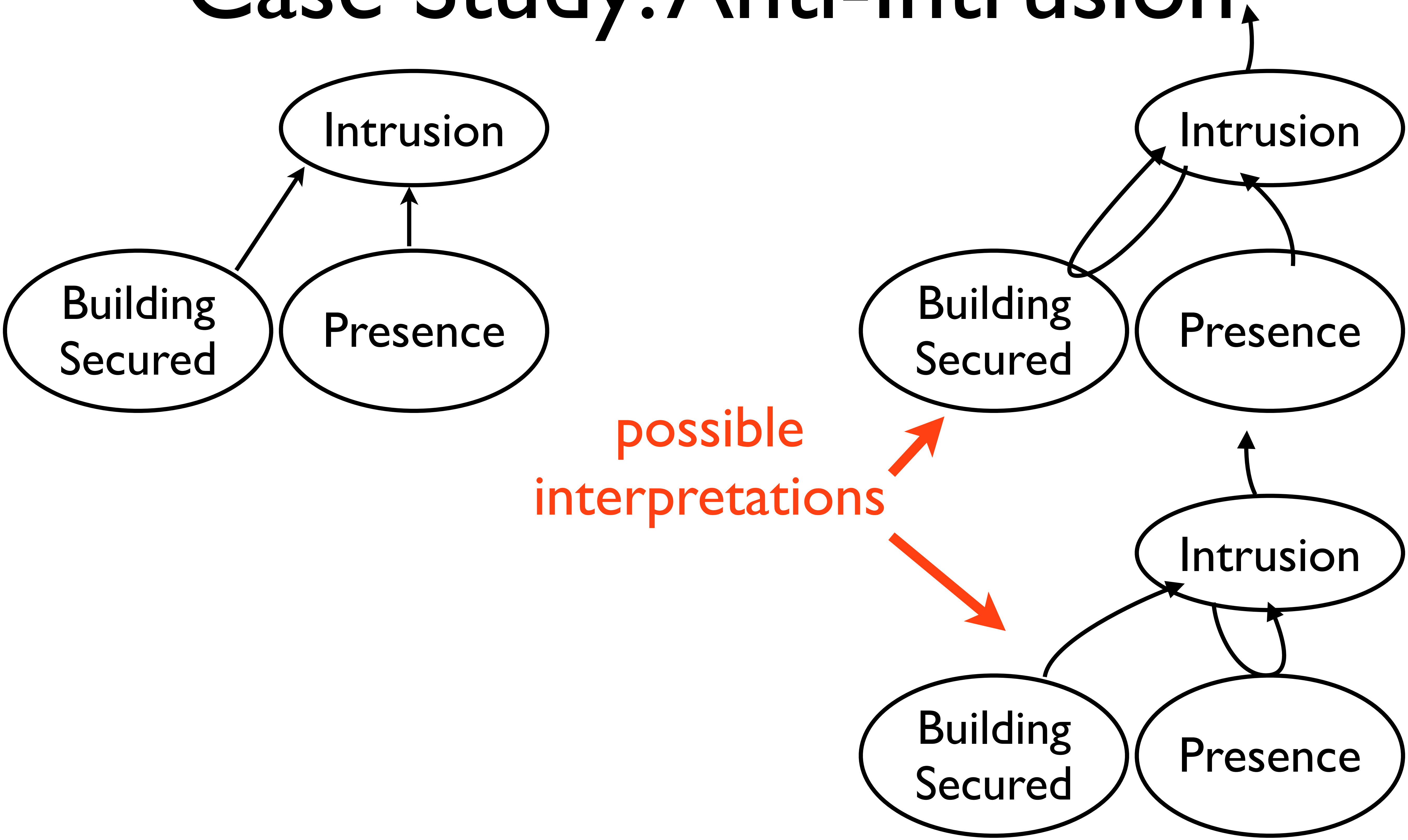
possible interpretations

Case Study: Anti-Intrusion



possible interpretations

Case Study: Anti-Intrusion



Intrusion

Building Secured

Presence

possible interpretations

Intrusion

Building Secured

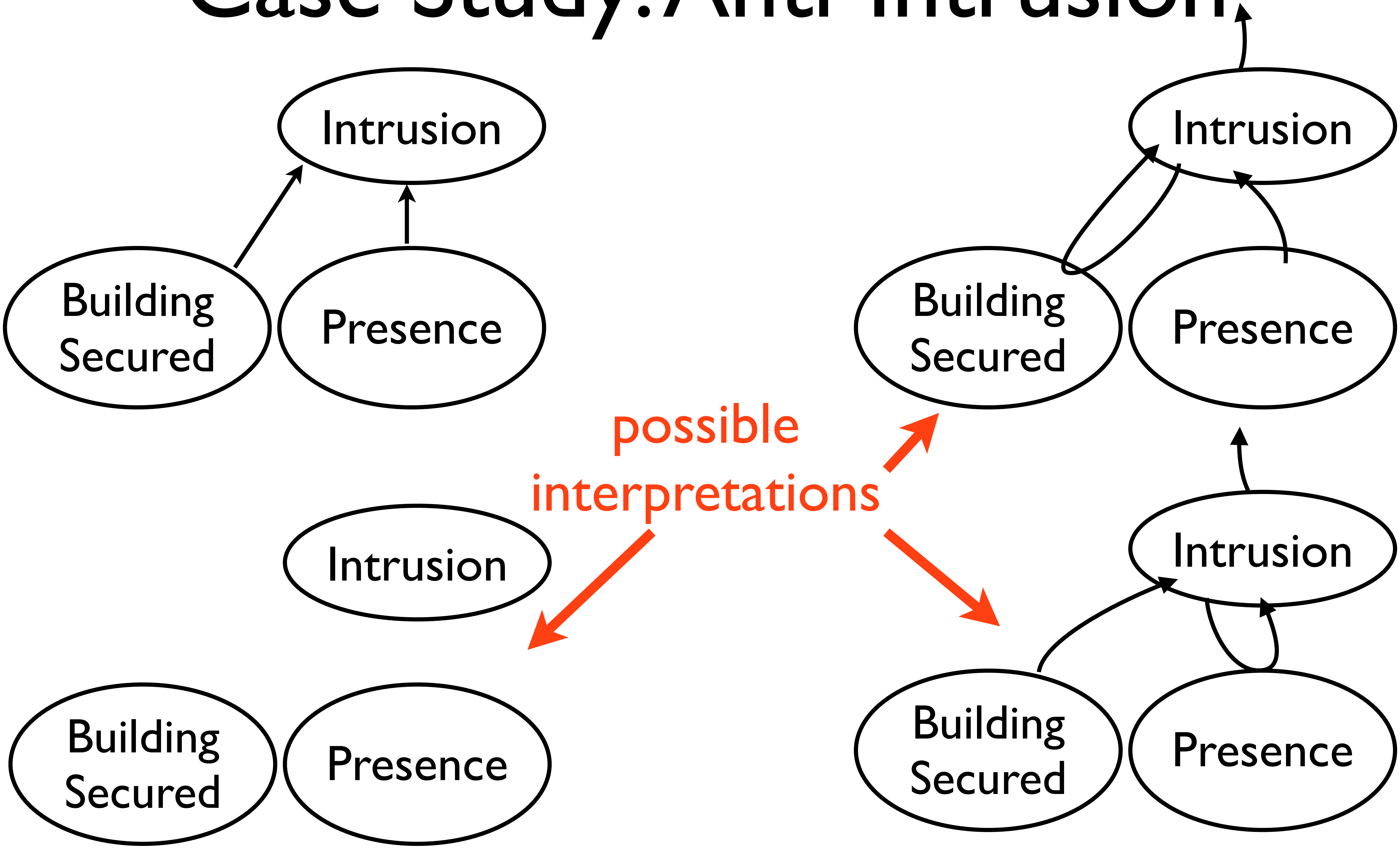
Presence

Intrusion

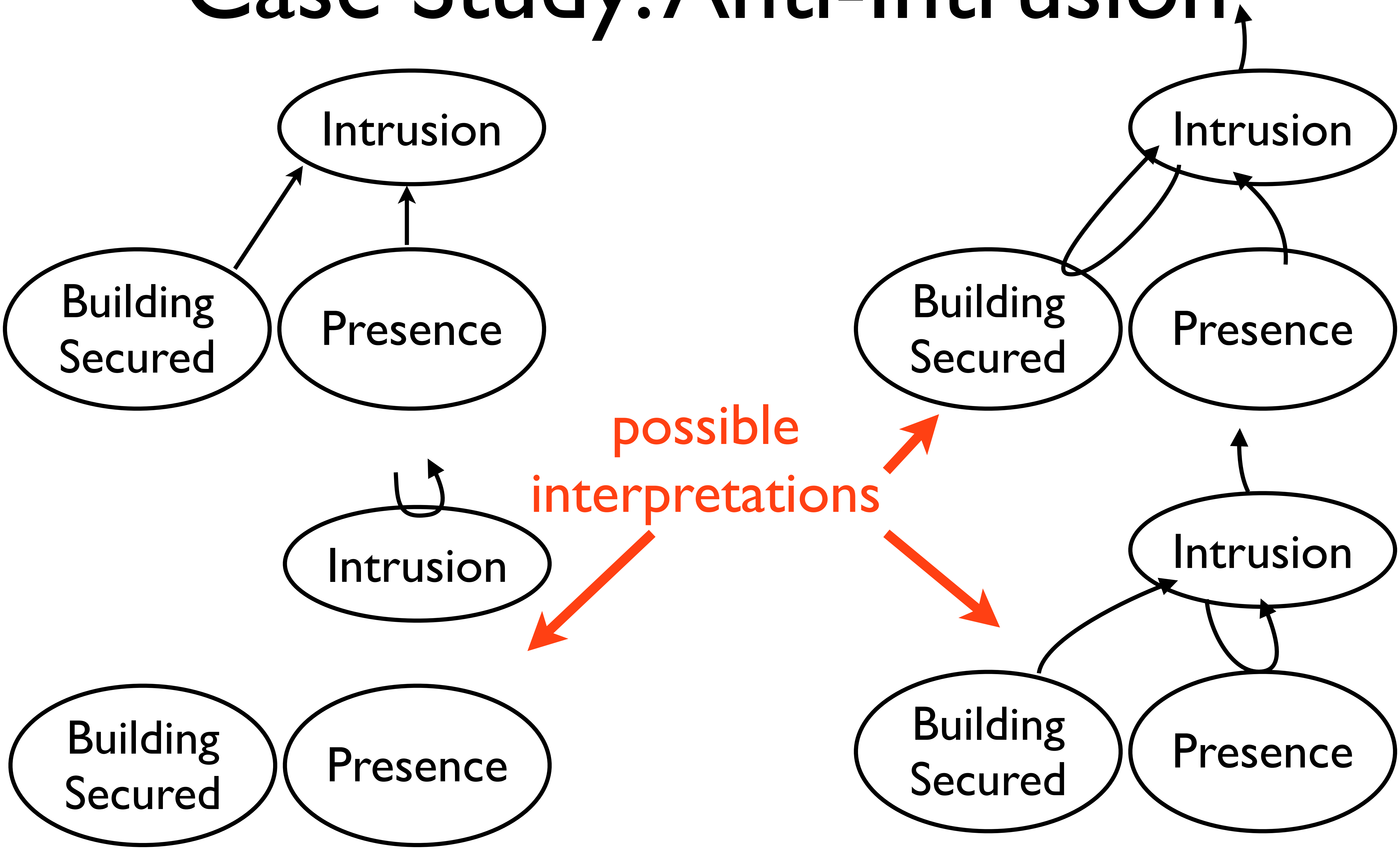
Building Secured

Presence

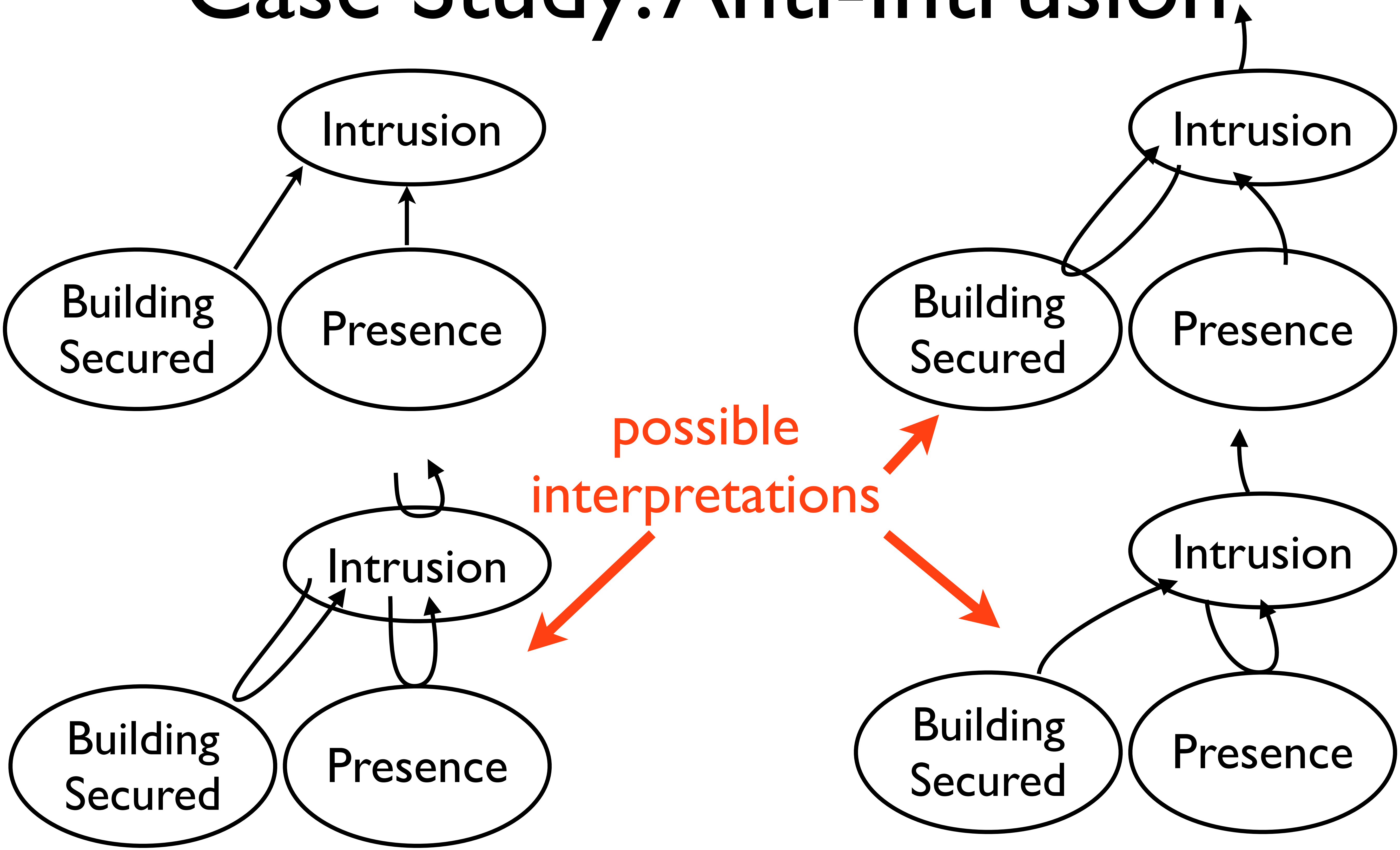
Case Study: Anti-Intrusion



Case Study: Anti-Intrusion



Case Study: Anti-Intrusion



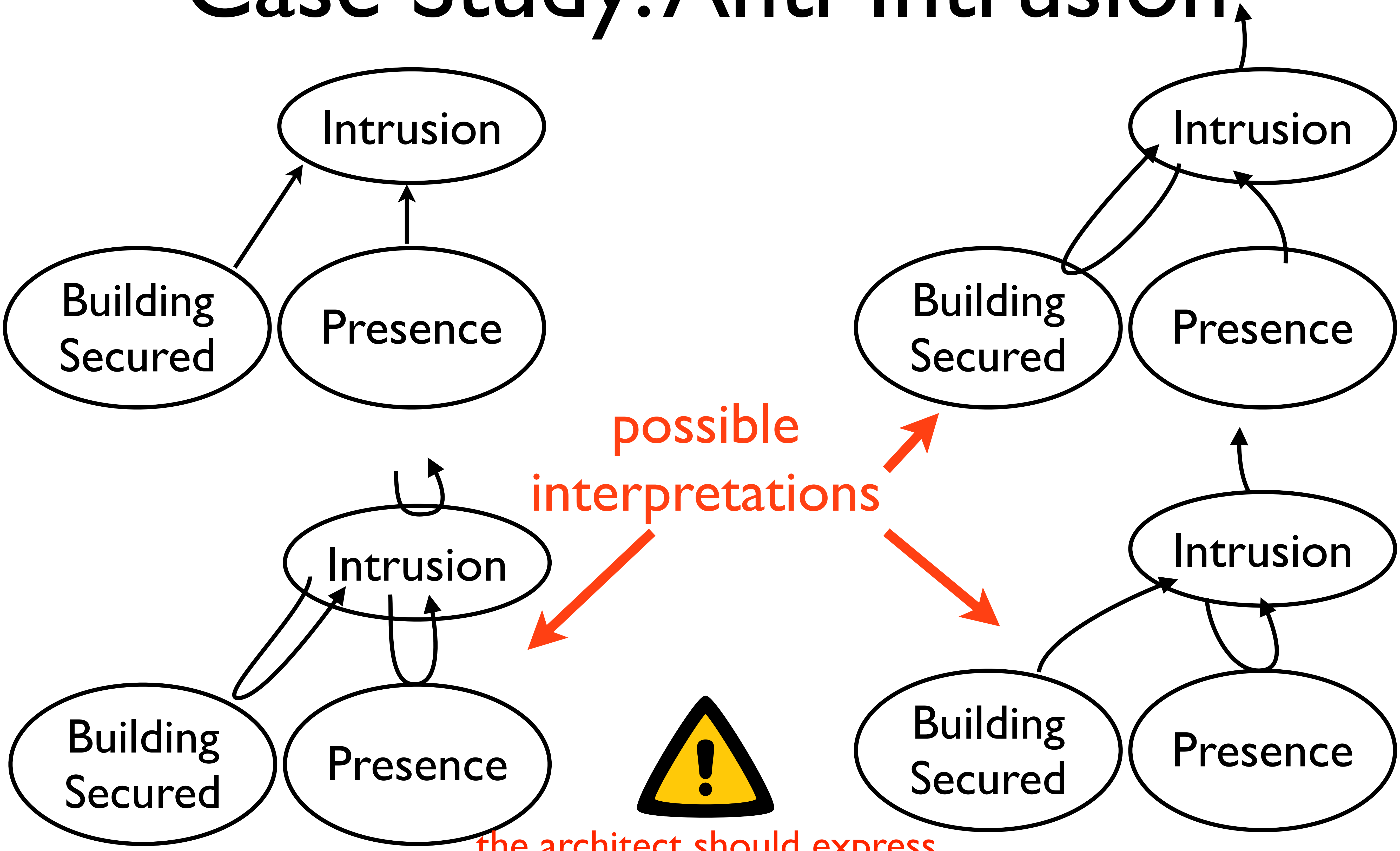
Intrusion

Building Secured

Presence

possible interpretations

Case Study: Anti-Intrusion



Intrusion

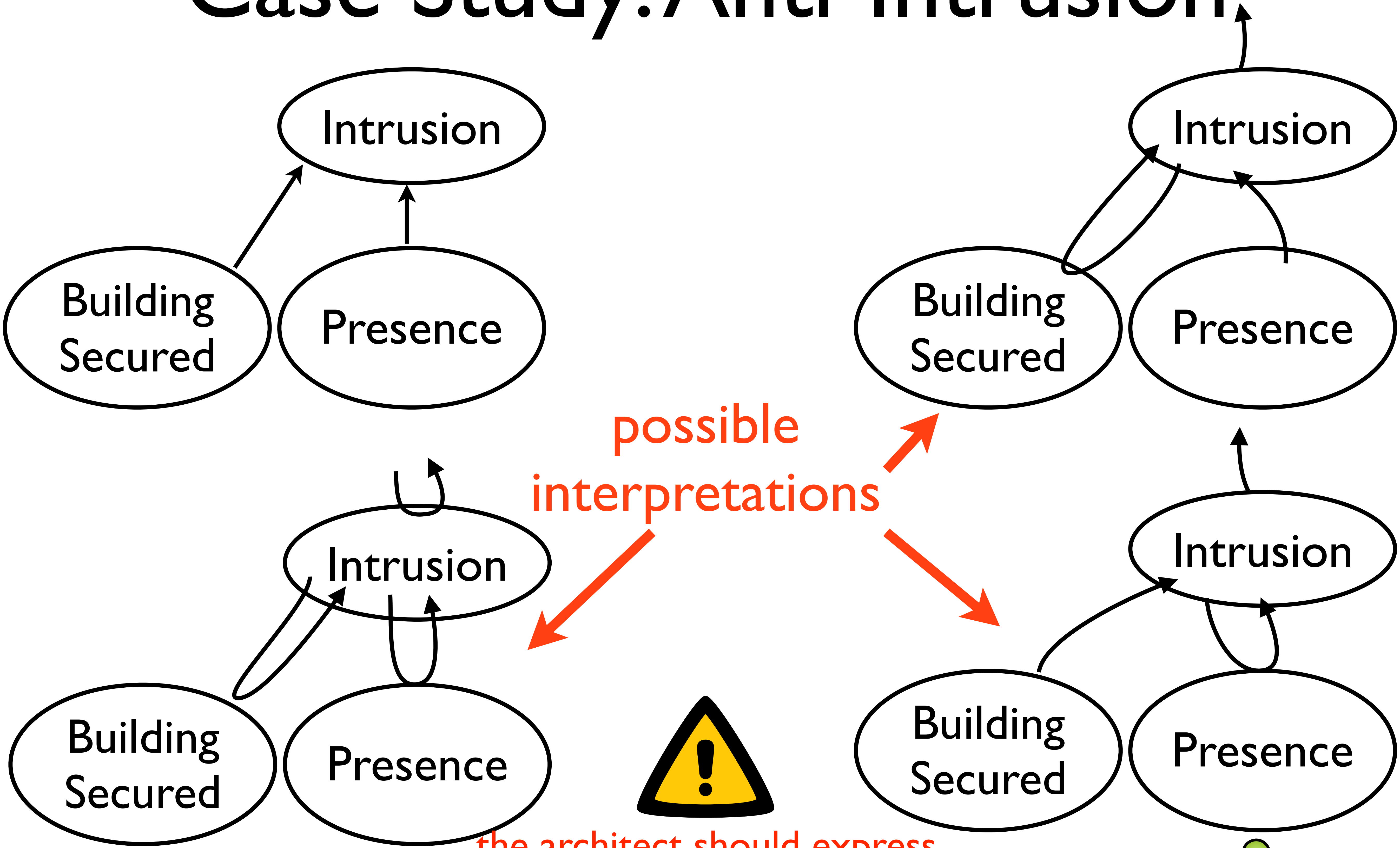
Building Secured

Presence

possible interpretations

the architect should express what he has in mind

Case Study: Anti-Intrusion



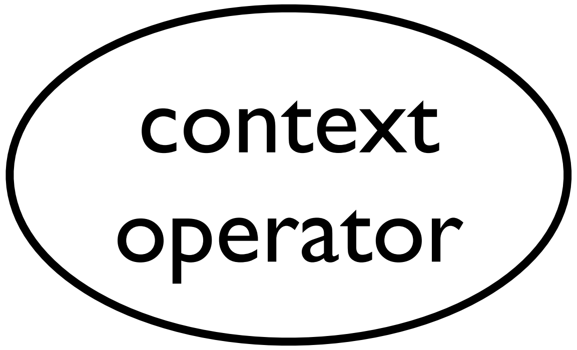
possible interpretations

the architect should express what he has in mind

Architect

➡interaction contracts

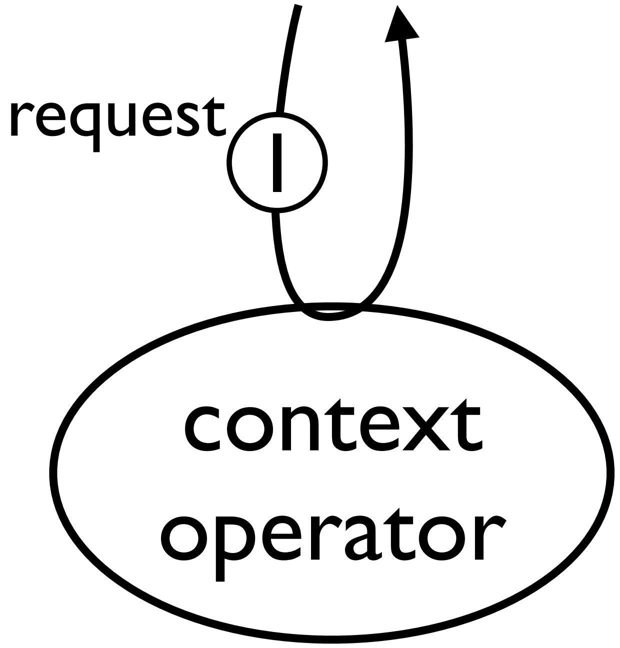
Interaction Contracts



context operator

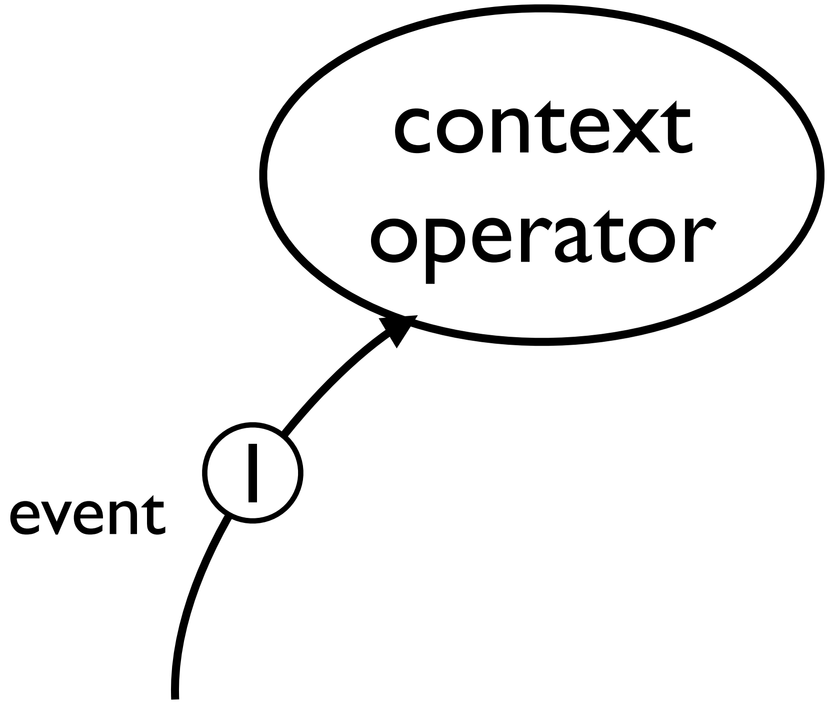
Interaction Contracts

① Activation condition



Interaction Contracts

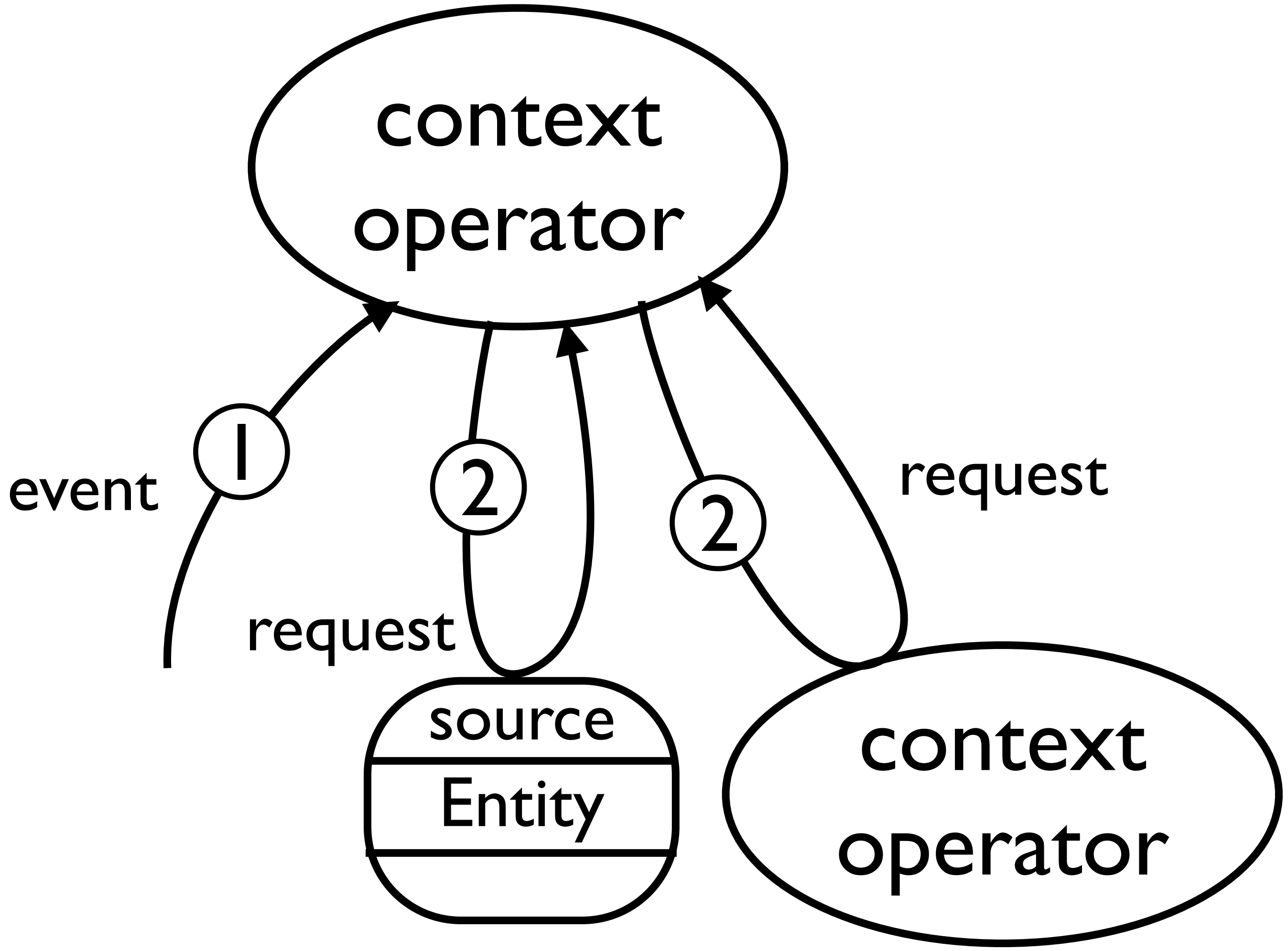
① Activation condition



Interaction Contracts

① Activation condition

② Data requirement

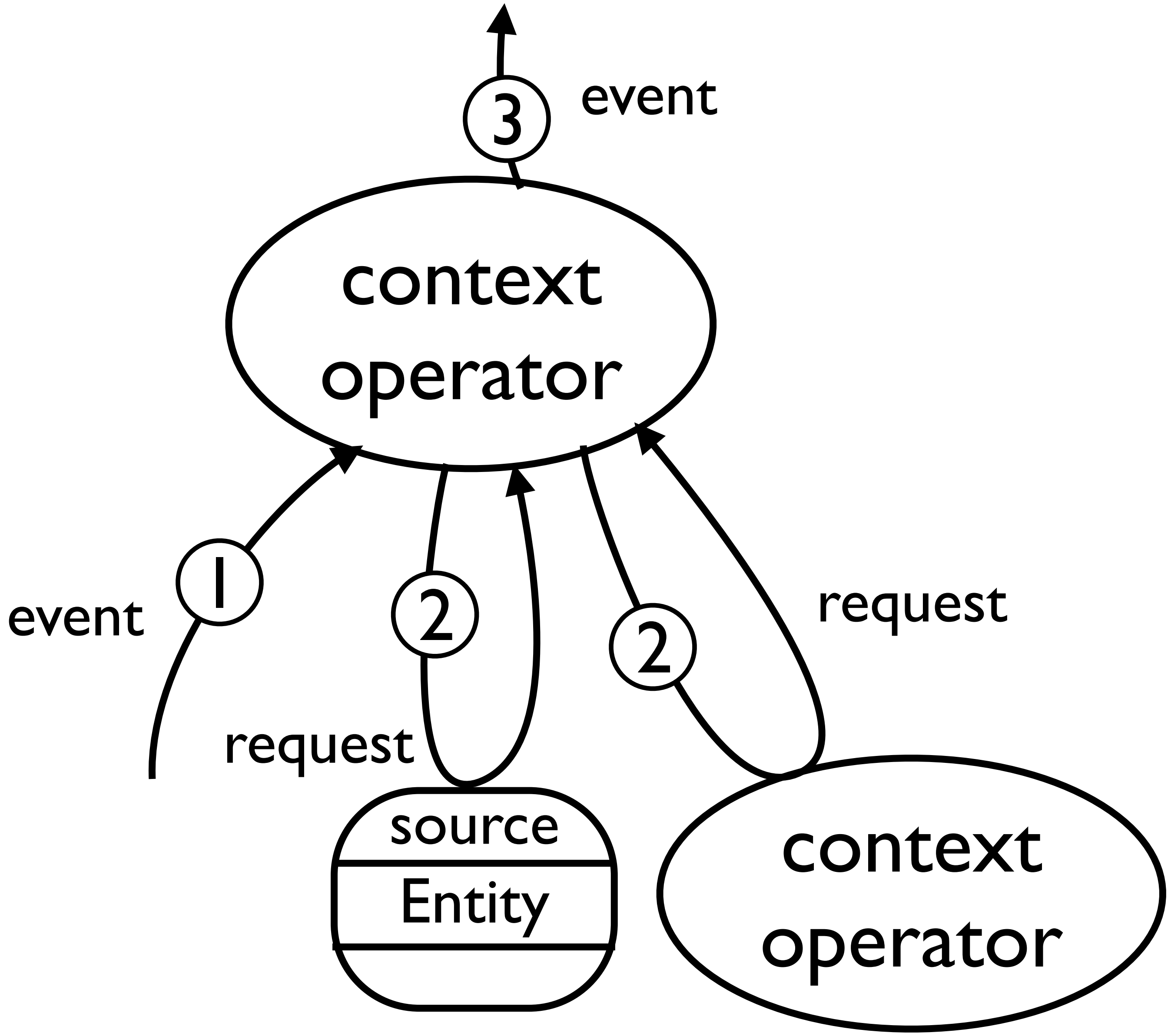


Interaction Contracts

① Activation condition

② Data requirement

③ Emission



Interaction Contracts

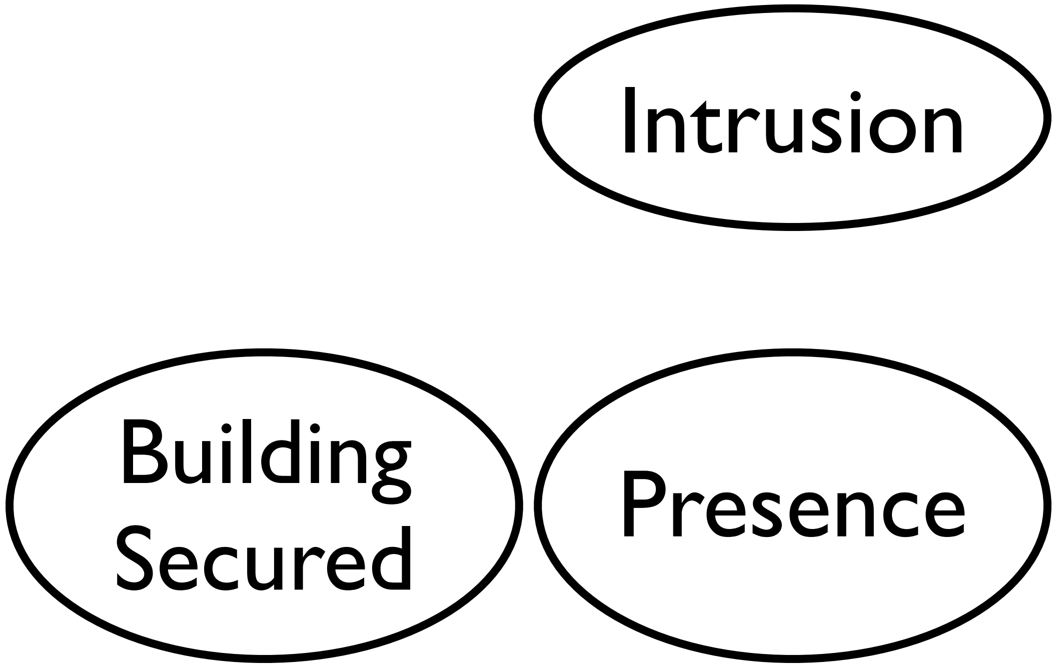
① Activation condition

② Data requirement

③ Emission

```
context Intrusion as Boolean {
 context Presence;
 context BuildingSecured;
 interaction {
  when provided Presence
  get BuildingSecured
  maybe publish
 }
}
```

Architect



Interaction Contracts

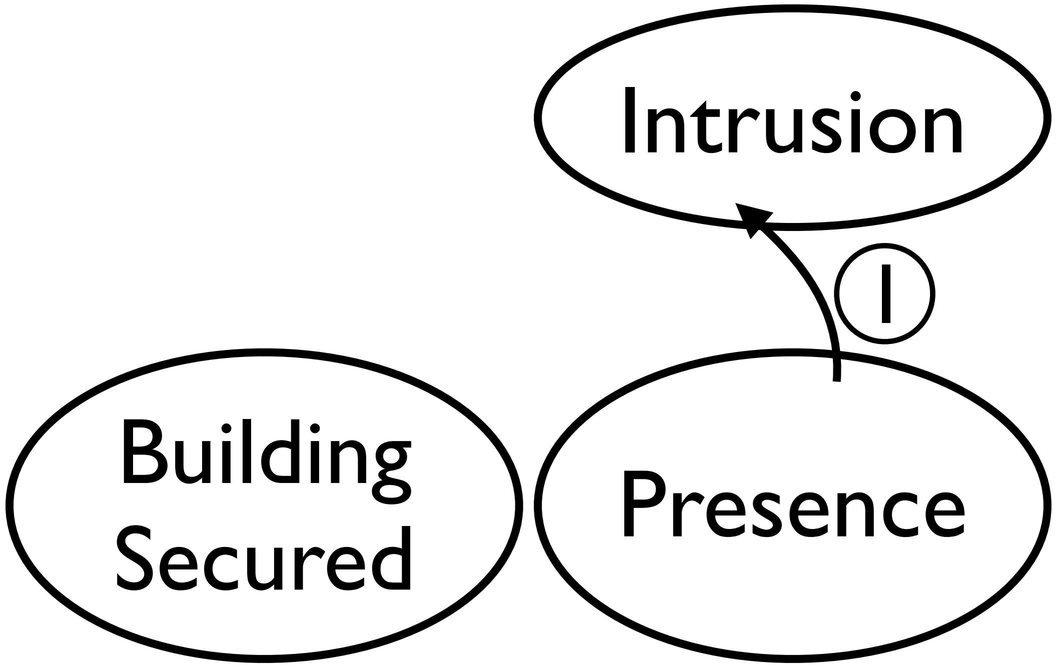
① Activation condition

② Data requirement

③ Emission

```
context Intrusion as Boolean {
 context Presence;
 context BuildingSecured;
 interaction {
①when provided Presence
  get BuildingSecured
  maybe publish
 }
}
```

Architect



Interaction Contracts

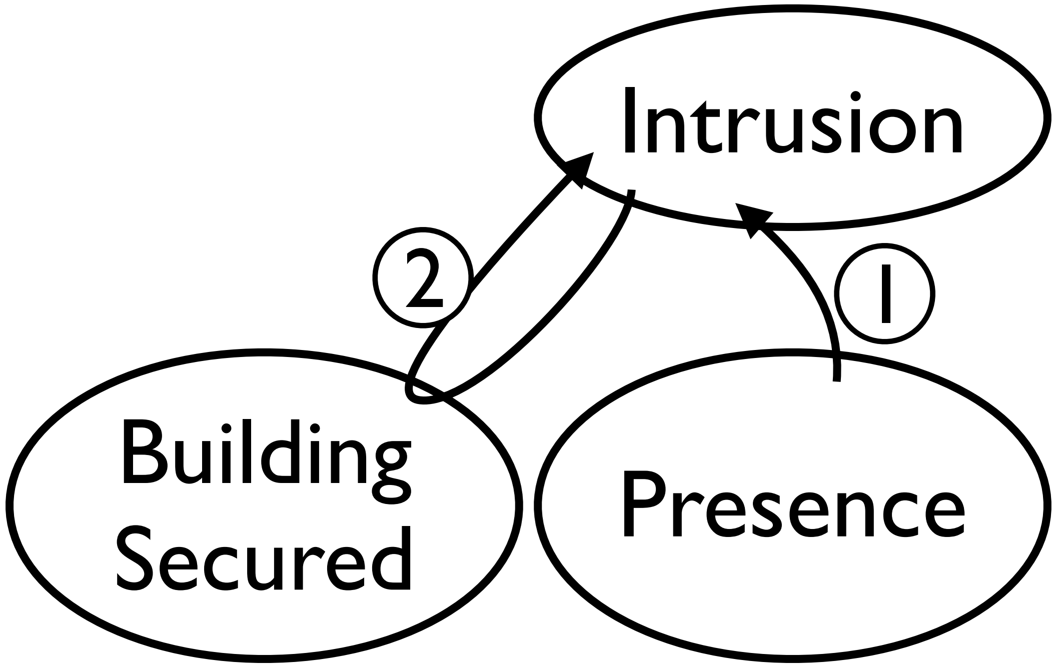
① Activation condition

② Data requirement

③ Emission

```
context Intrusion as Boolean {
 context Presence;
 context BuildingSecured;
 interaction {
①when provided Presence
②get BuildingSecured
  maybe publish
 }
}
```

Architect



Interaction Contracts

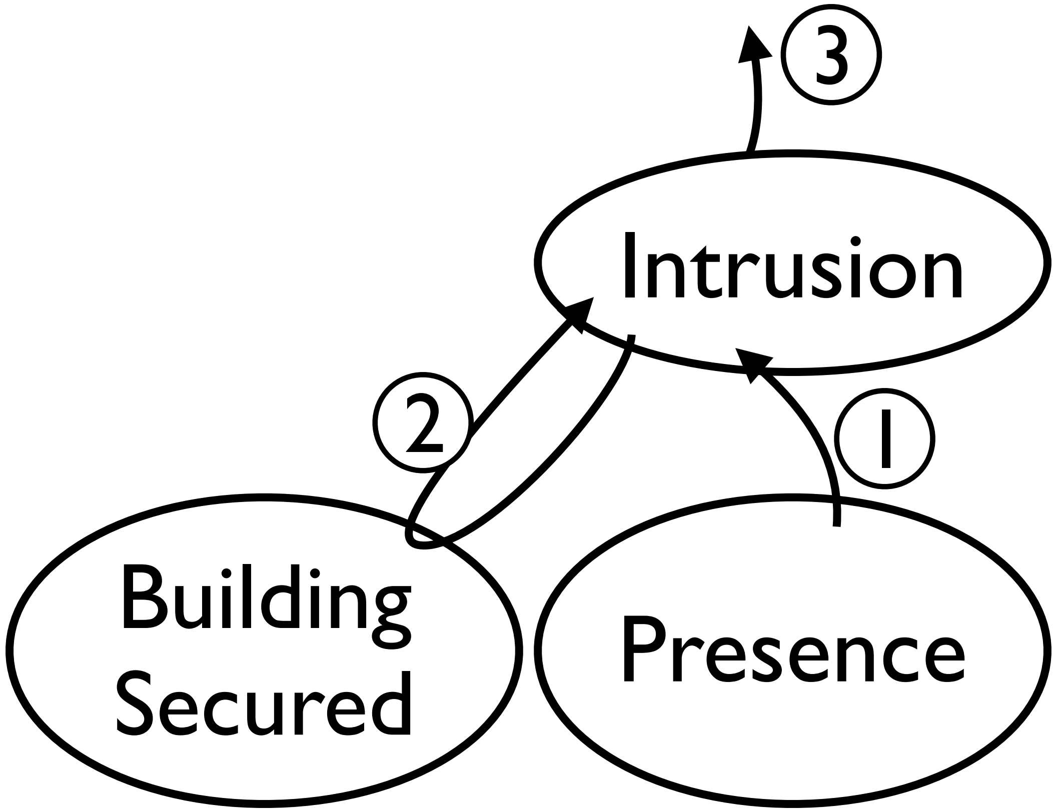
① Activation condition

② Data requirement

③ Emission

```
context Intrusion as Boolean {
 context Presence;
 context BuildingSecured;
 interaction {
①when provided Presence
②get BuildingSecured
③maybe publish
 }
}
```

Architect



Interaction Contracts

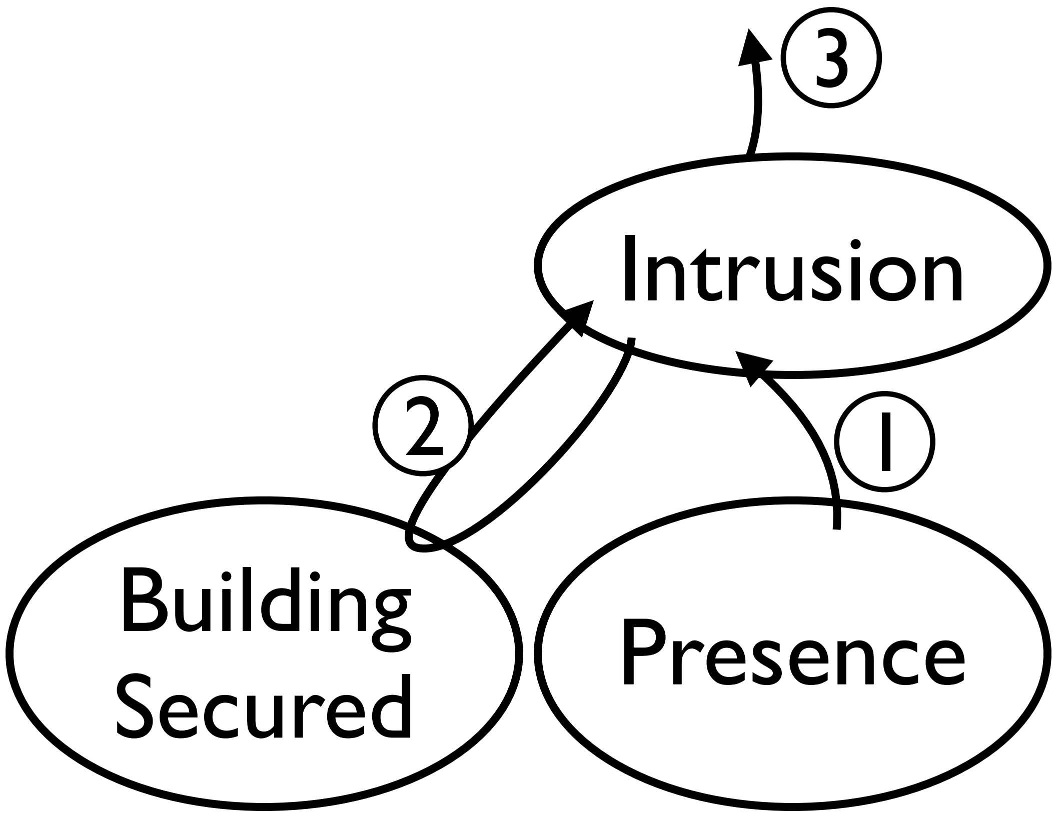
① Activation condition

② Data requirement

③ Emission

```
context Intrusion as Boolean {
 context Presence;
 context BuildingSecured;
 interaction {
①when provided Presence
②get BuildingSecured
③maybe publish
 }
}
```

Architect



related to automata approaches

Summary

Architect

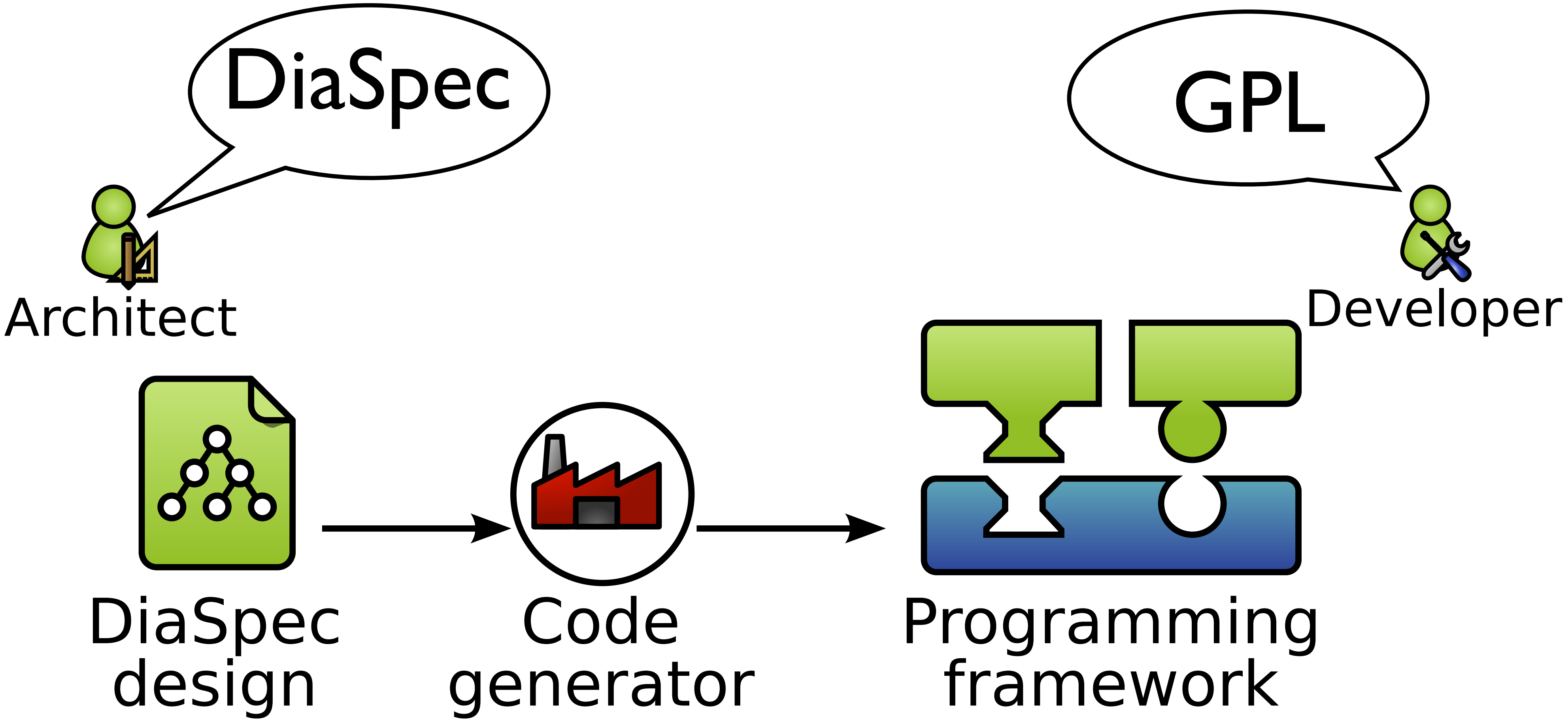
A design framework consisting of a design language, *DiaSpec*, which guides the architect by offering

- concepts dedicated to the SCC paradigm
- a separation between environment handling and logic
- a separation between information creation and use
- a dedicated description of interactions

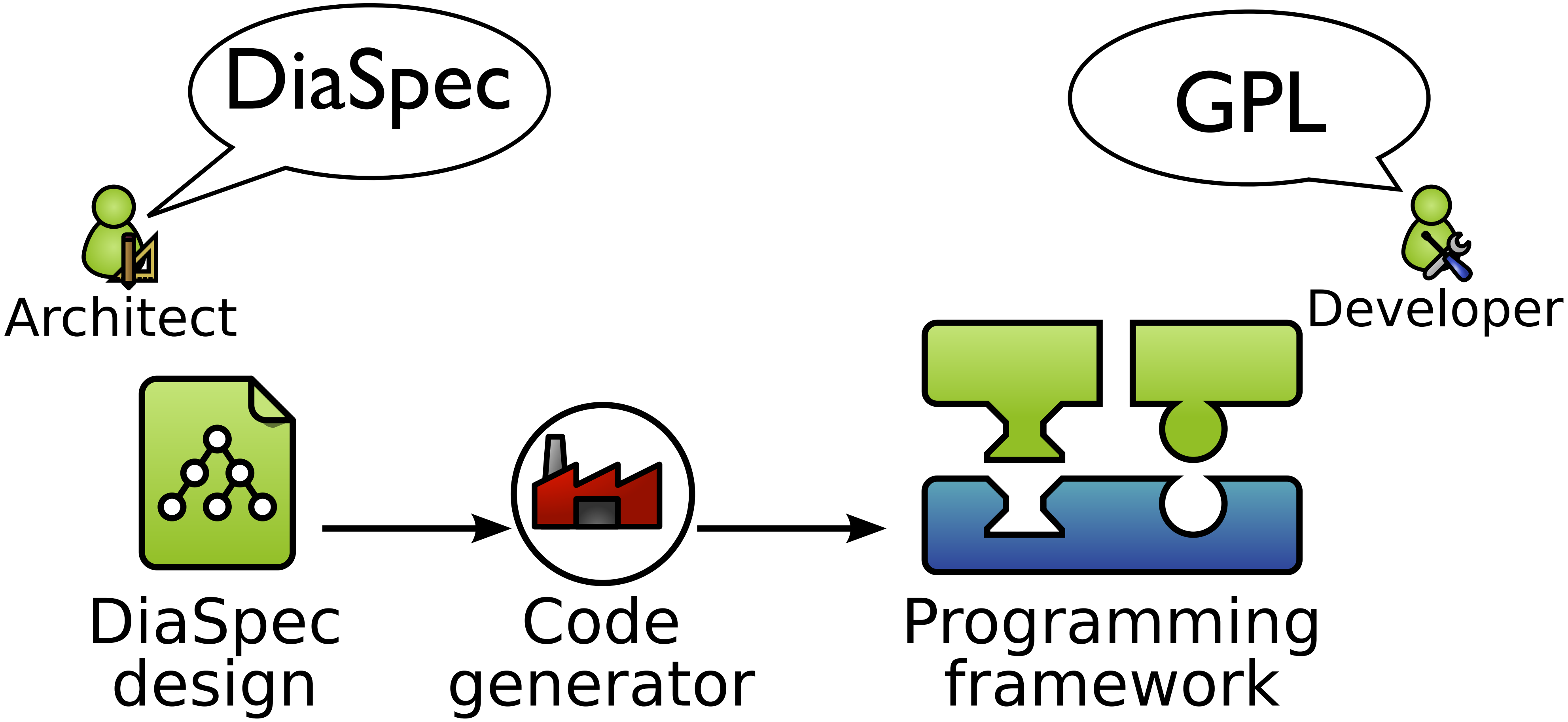
Contributions

1. A paradigm-specific design framework
2. A programming framework dedicated to a design
3. An evaluation of the approach

A Programming Framework
Generated from the Design



A Programming Framework
Generated from the Design



- separates 2 different roles with 2 different languages
- leverages GPL tools, libraries and expertise
- ensures conformance automatically

A Programming Framework

how to make a programming framework *conform* to a particular design?

A Programming Framework

The code generator maps

- each description to an abstract class
- each interaction contract to an abstract method

A Programming Framework

The code generator maps

- each description to an abstract class
- each interaction contract to an abstract method

By leveraging the GPL type checker, the framework

- guides the implementation of what is required
- forbids anything not specified in the design

A Programming Framework

The code generator maps

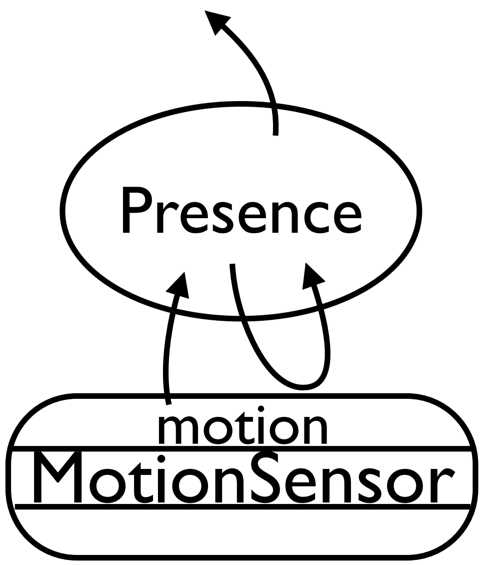
- each description to an abstract class
- each interaction contract to an abstract method

By leveraging the GPL type checker, the framework

- guides the implementation of what is required
- forbids anything not specified in the design

different than what is
proposed by ADLs or MDE

Generation

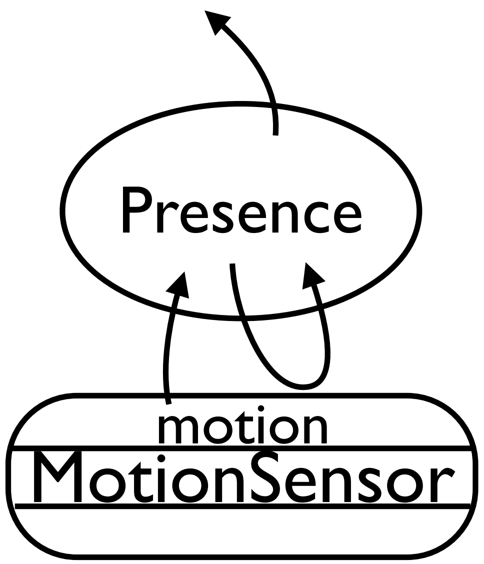


```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
    when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
                         boolean motion, Select select);
}
```

Generation

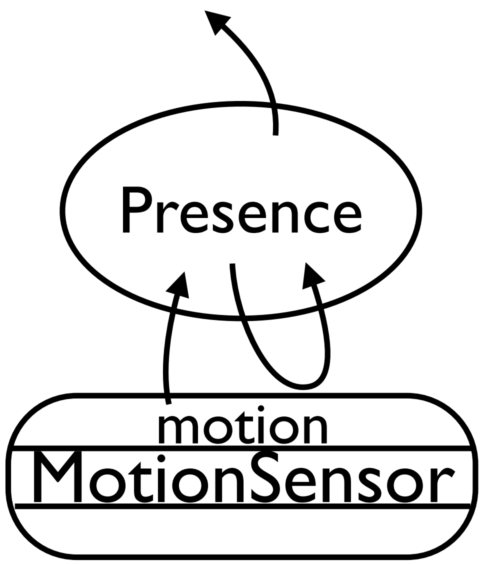


```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
  ① when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
                 ①     boolean motion, Select select);
}
```

Generation

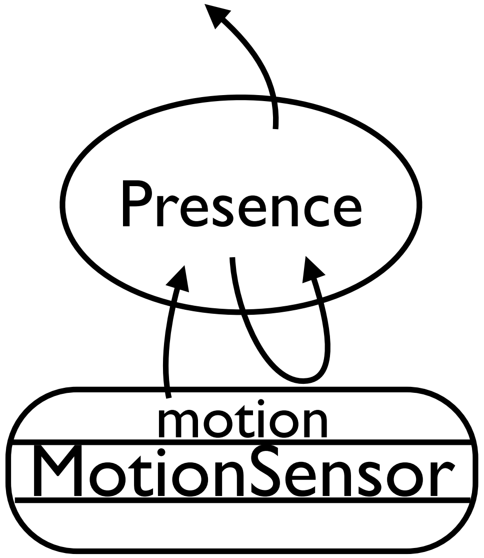


```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
  ① when provided motion from MotionSensor
     get motion from MotionSensor
     always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
                      ① boolean motion, Select select);
}
```

Generation



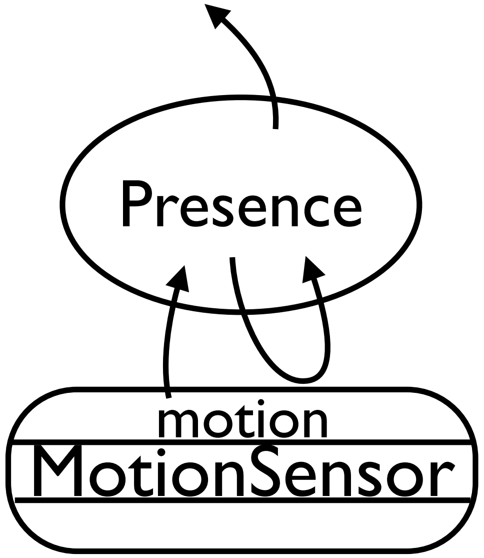
```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
  ① when provided motion from MotionSensor
  ② get motion from MotionSensor
     always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
                        ① boolean motion, Select select);
                                                ②
}
```

according to the
interaction contract

Generation



```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
 ① when provided motion from MotionSensor
 ② get motion from MotionSensor
 ③ always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(
                     ① boolean motion, Select select);
           ③                                       ②
}
```

Implementing the Behavior

```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
    when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
 abstract boolean onMotionFromMotionSensor(
          boolean motion, Select select);
}
```

```
class Presence extends AbstractPresence {
 boolean onMotionFromMotionSensor(
           boolean motion, Select select) {
  return motion;
 }
}
```

Developer

Implementing the Behavior

when motion is detected ➡ there is presence

when motion is not detected? ➡ ?

The developer needs to ask all motion sensors

```
class Presence extends AbstractPresence {
 boolean onMotionFromMotionSensor(
             boolean motion, Select select) {
  return motion;
 }
}
```

Developer

Entity Selection

Required when an entity is the interaction's target

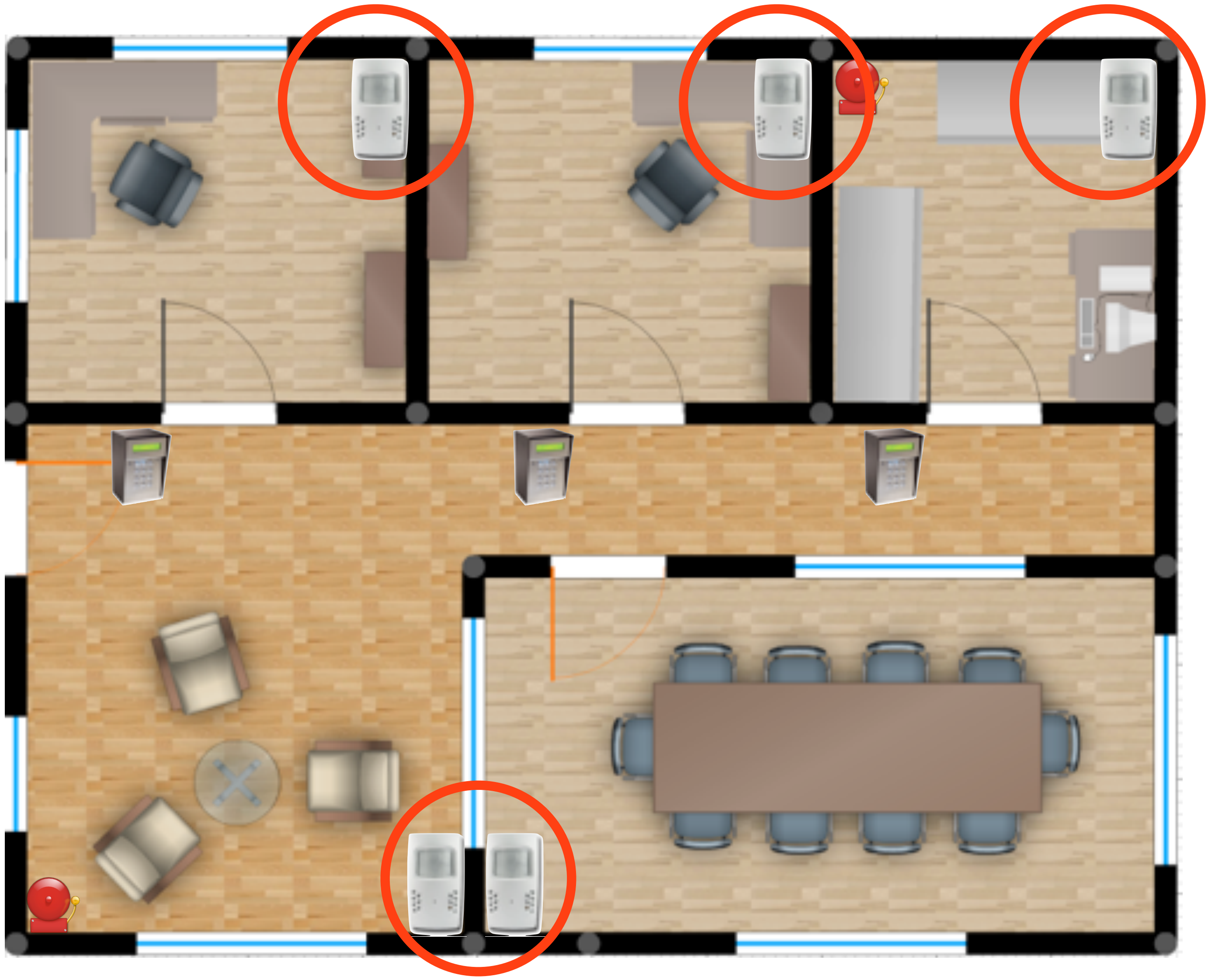
Guide the developer with an embedded and type-safe DSL

Developer

Entity Selection

```
entity MotionSensor {
   source motion as Boolean;
   attribute room as Integer;
}
```

Architect

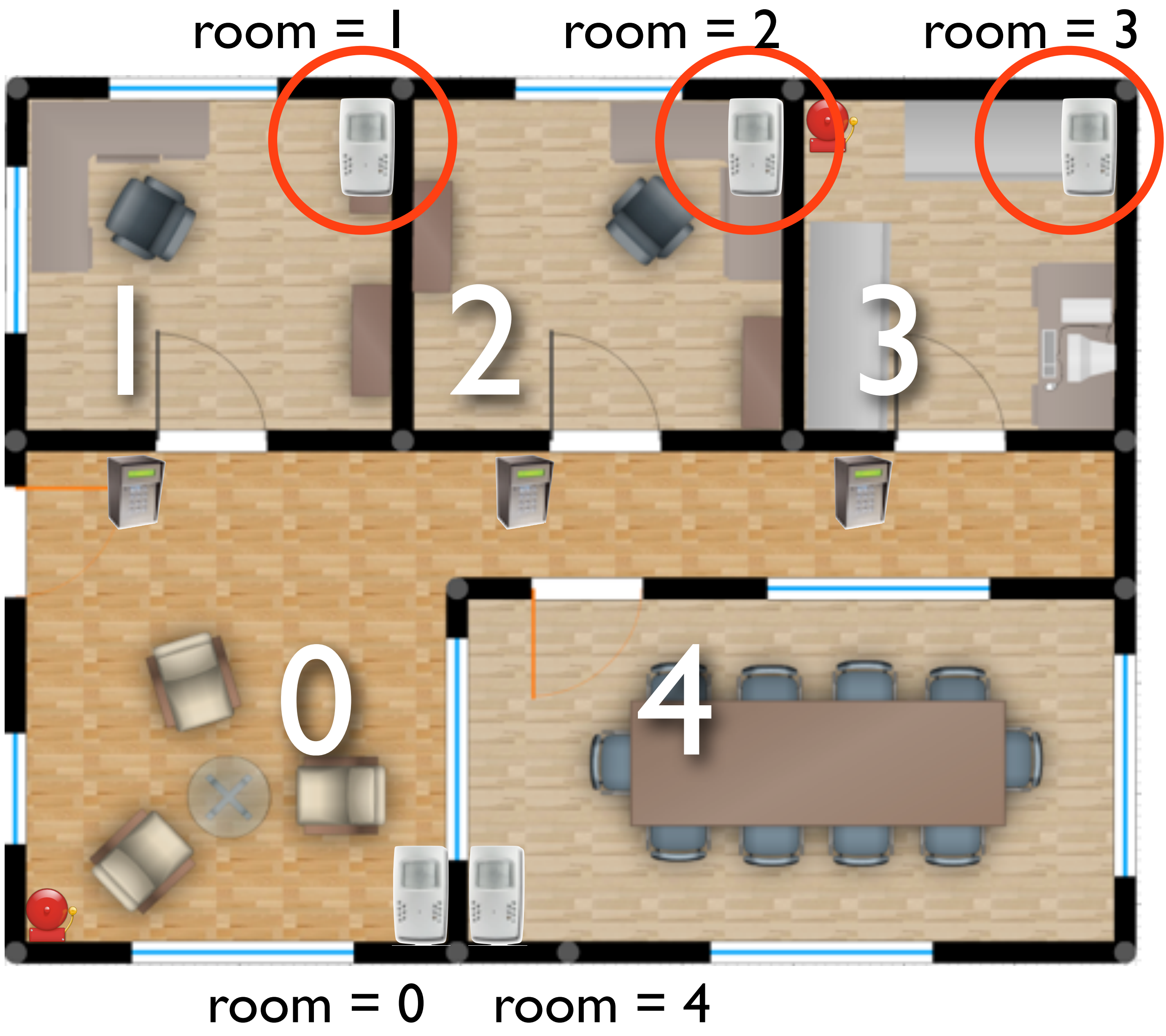


Select all motion sensors:

```
select.motionSensors().all()
```

Developer

Entity Selection



```
entity MotionSensor {
   source motion as Boolean;
   attribute room as Integer;
}
```

Architect

Developer

Select all motion sensors in rooms 1 to 3:

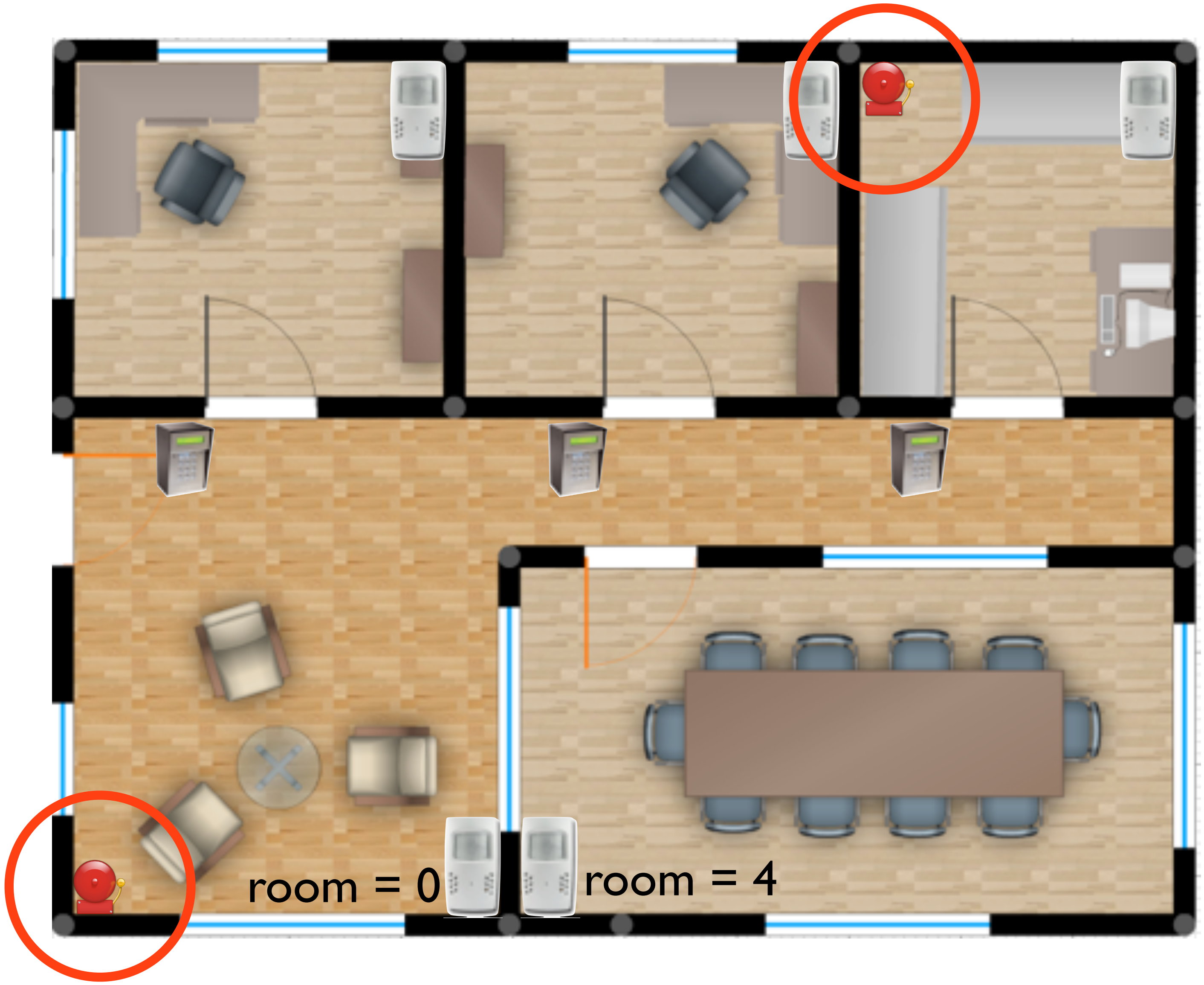
```
select.motionSensors().whereRoom(between(1,3))
```

Commanding Entities

```
entity Alarm {
  action OnOff;
}
```

```
action OnOff {
  on();
  off();
}
```

Architect



Triggering all alarms:

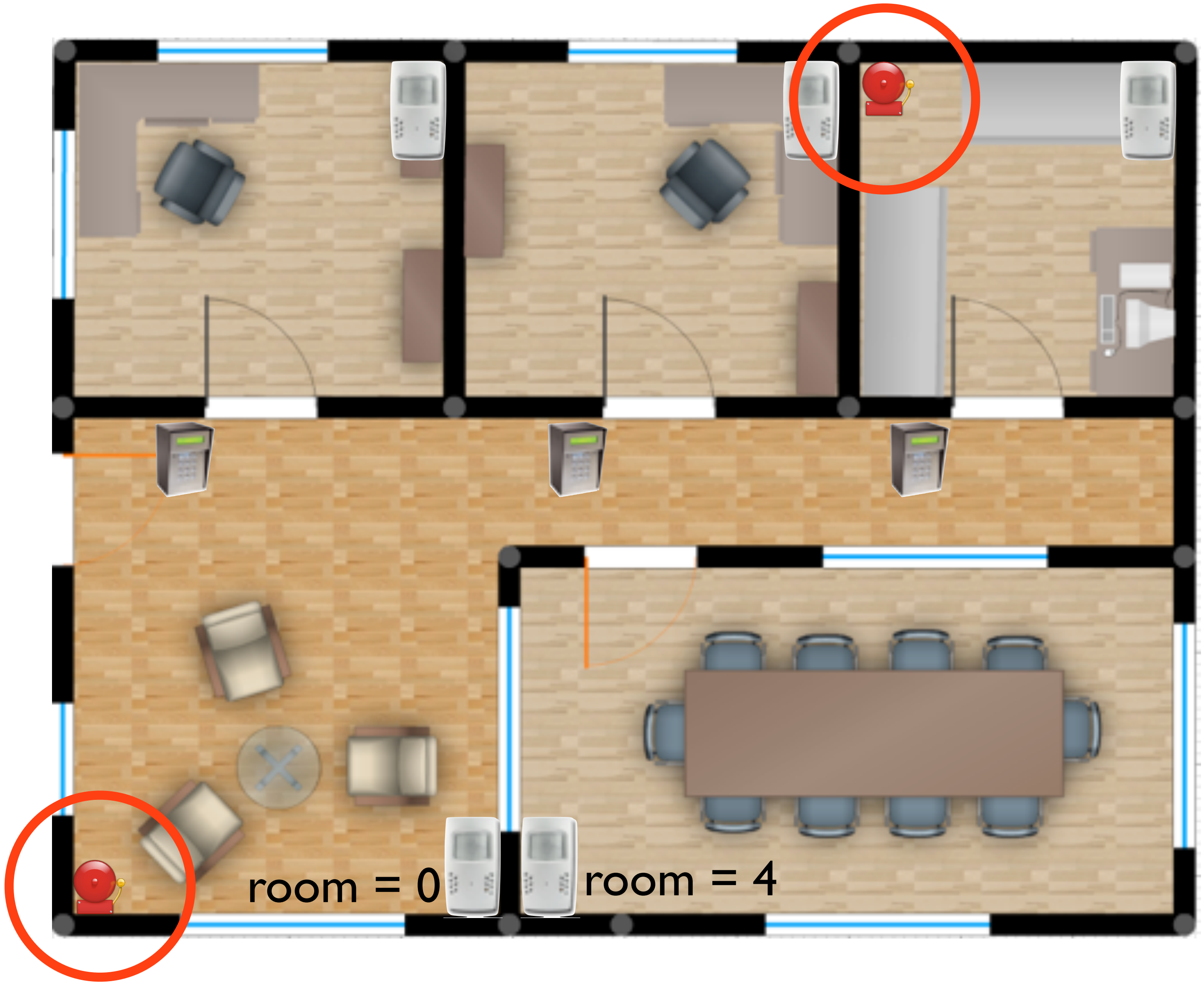
```
select.alarms().all().on();
```

Developer

Entity Selection Conformance

```
entity Alarm {          action OnOff {
  action OnOff;           on();
}                         off();
                        }
```

Architect



Conformance

It is not possible to send unsupported orders:

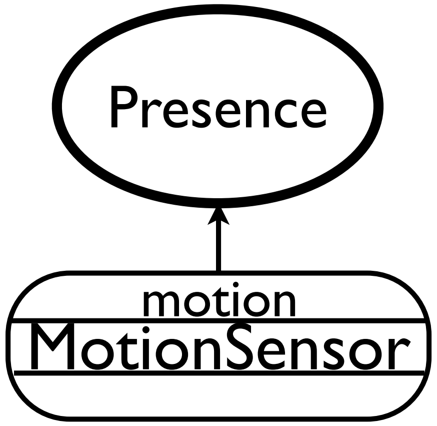
select.alarms().all().~~start();~~

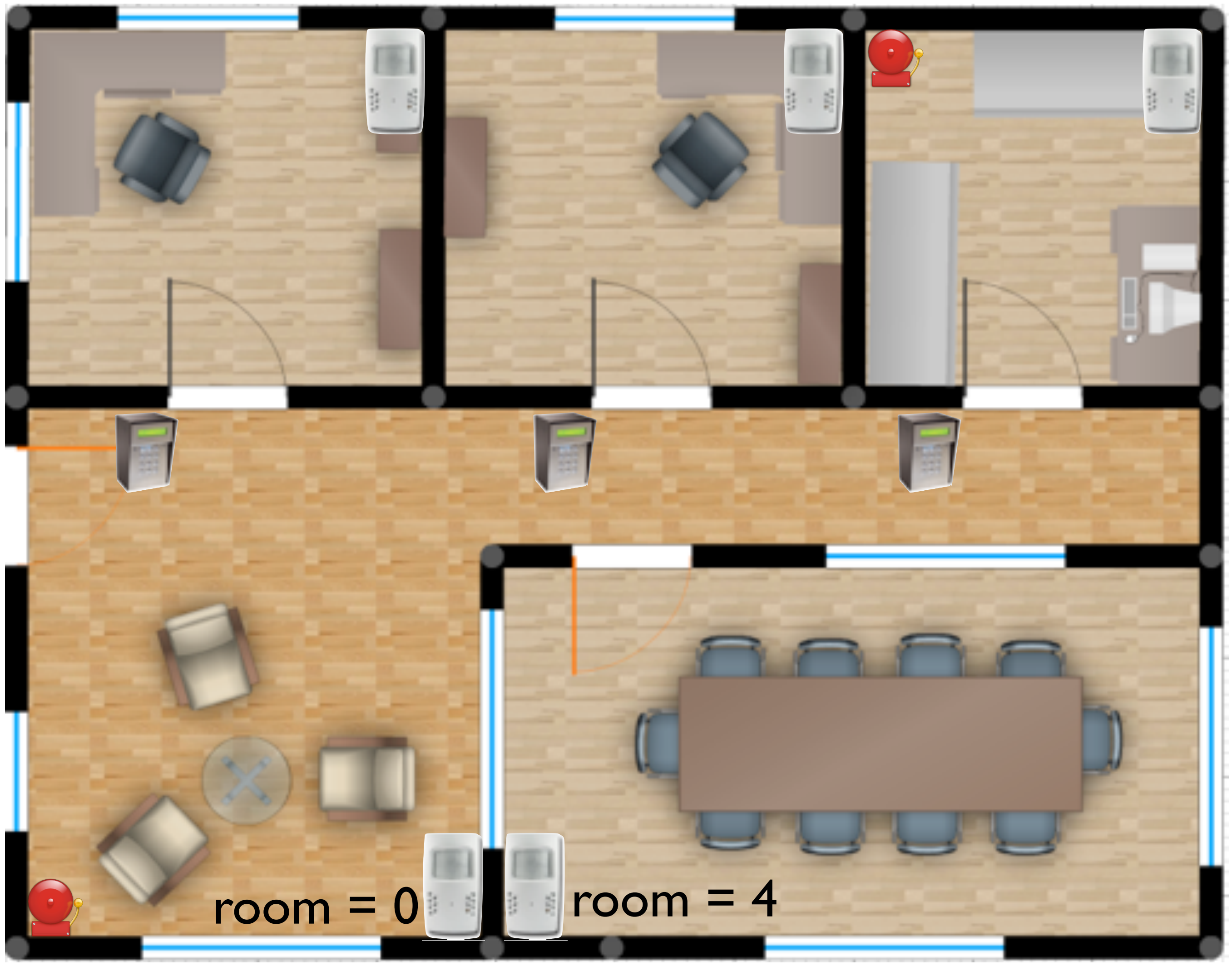
compile-time error

Entity Selection

```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
    when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}
```

Architect





Conformance

It is not possible to discover all kinds of entities:

`select.~~alarms().all();~~`

compile-time error

Implementing the Behavior

```
context Presence as Boolean {
  source motion from MotionSensor;
  interaction {
    when provided motion from MotionSensor
    get motion from MotionSensor
    always publish
  }
}
```

Architect

```
abstract class AbstractPresence {
  abstract boolean onMotionFromMotionSensor(...);
}
```

```
class Presence extends AbstractPresence {
  boolean onMotionFromMotionSensor(
                       boolean motion, Select select) {
    if (motion)
      return true;
    MotionSensors sensors = select.motionSensors().all();
    for (MotionSensor sensor : sensors)
      if (sensor.getMotion())
        return true;
    return false;
  }
}
```

Developer

Summary

The developer is guided with

- a support dedicated to the application
- an embedded DSL for entity selection

Developer

Conformance is ensured by

- generating a programming framework
- leveraging a GPL type checker

Conformance

Contributions

1. A paradigm-specific design framework

2. A programming framework dedicated to a design

3. An evaluation of the approach

Evaluation of the Approach

- Expressiveness
- Usability
- Productivity

Evaluation: Expressiveness

Numerous domains

- home-automation
- avionics
- graphical user interfaces
- health-care
- telecommunications
- tier-system monitoring
- *etc*.

Evaluation: Usability

Context

- 80 students during 3 years
- sparse and oral-only documentation

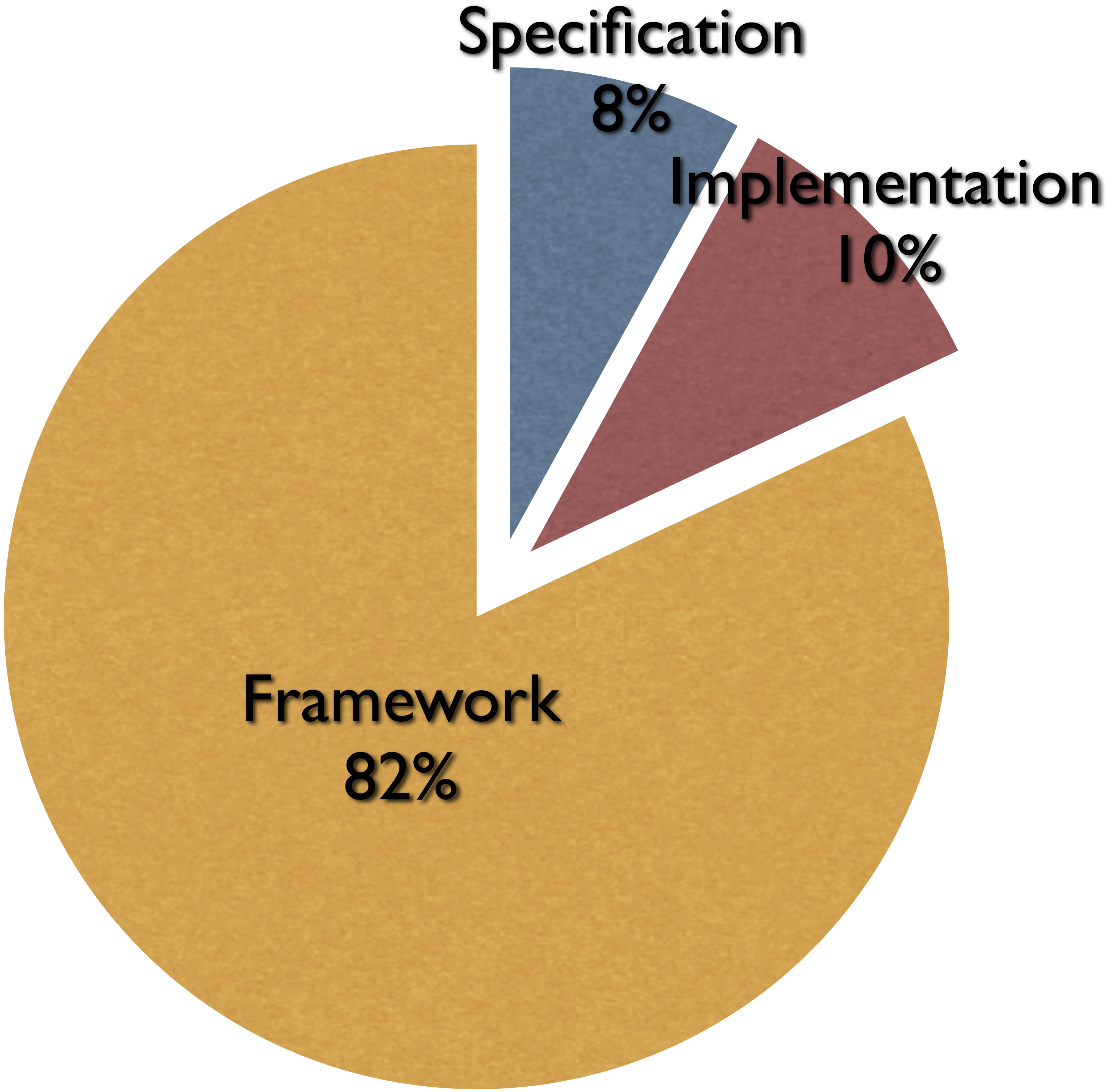
Results

- 64 students completed the assignment
- Identification of the interaction contracts

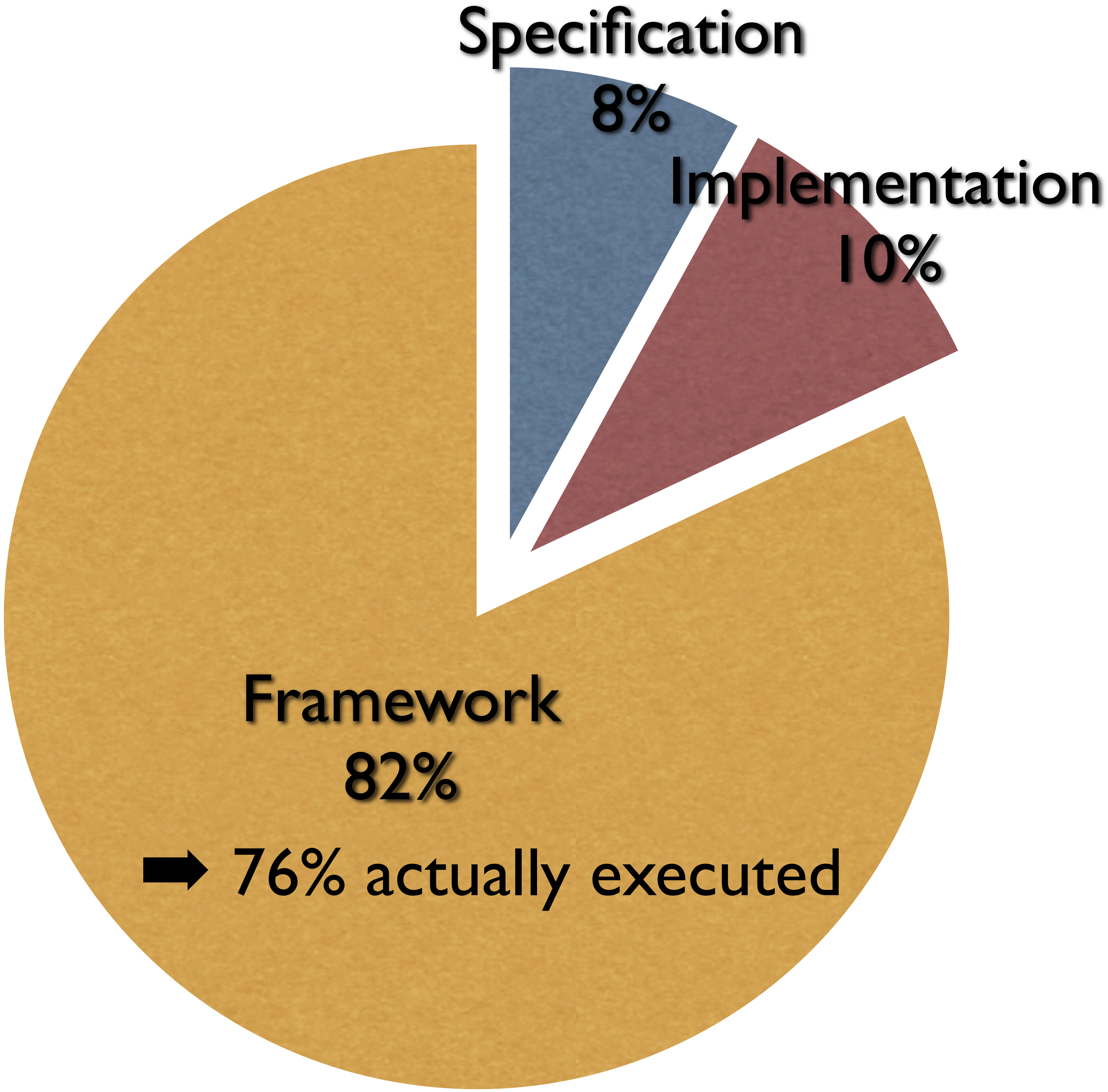
Evaluation: Productivity

We measured the
amount of code
generated automatically

Evaluation: Productivity



Evaluation: Productivity



➡ 76% actually executed

Evaluation: Productivity

Complexity of the developer's code

We used the Sonar platform
to measure code quality
through numerous metrics

Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

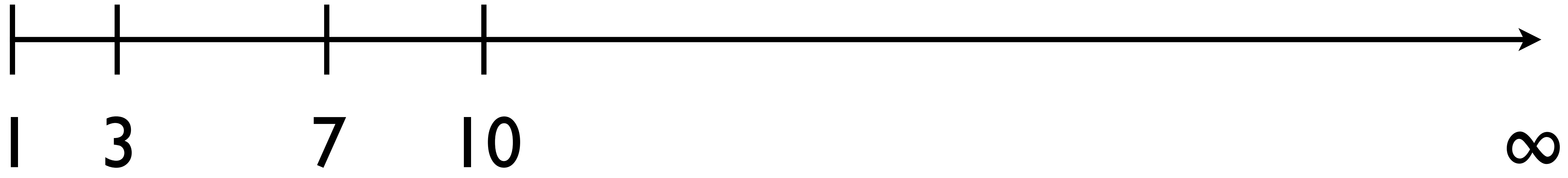
McCabe cyclomatic complexity

Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity

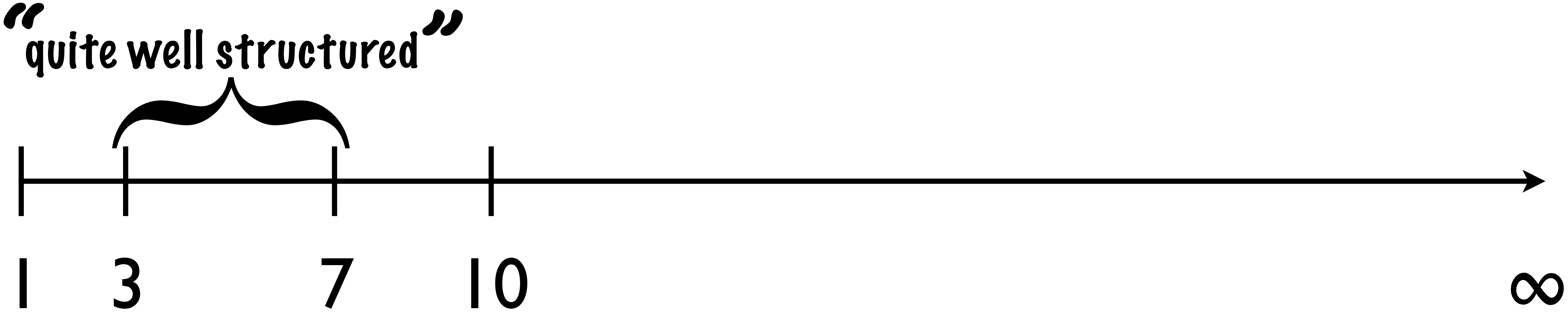


Evaluation: Productivity

Complexity of the developer's code

"number of linearly independent paths in a source code"

McCabe cyclomatic complexity

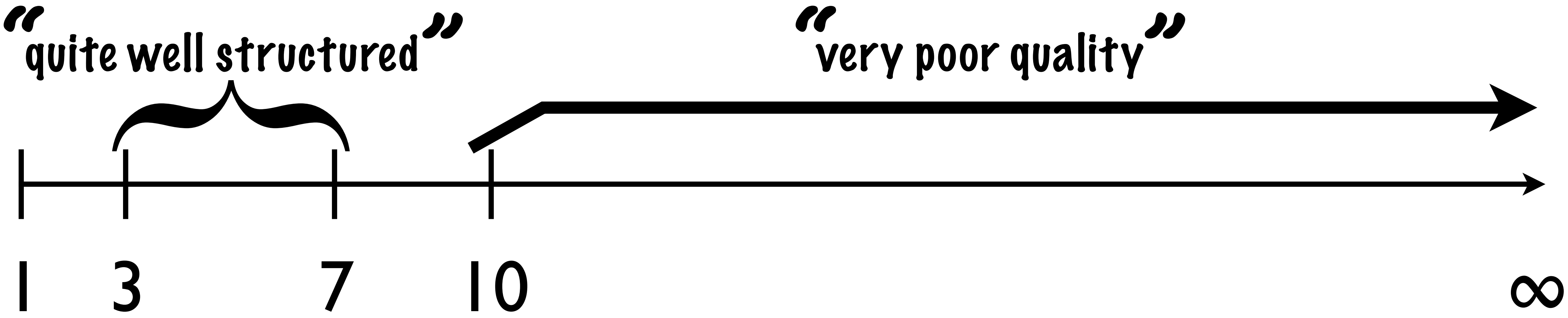


Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity



“quite well structured”

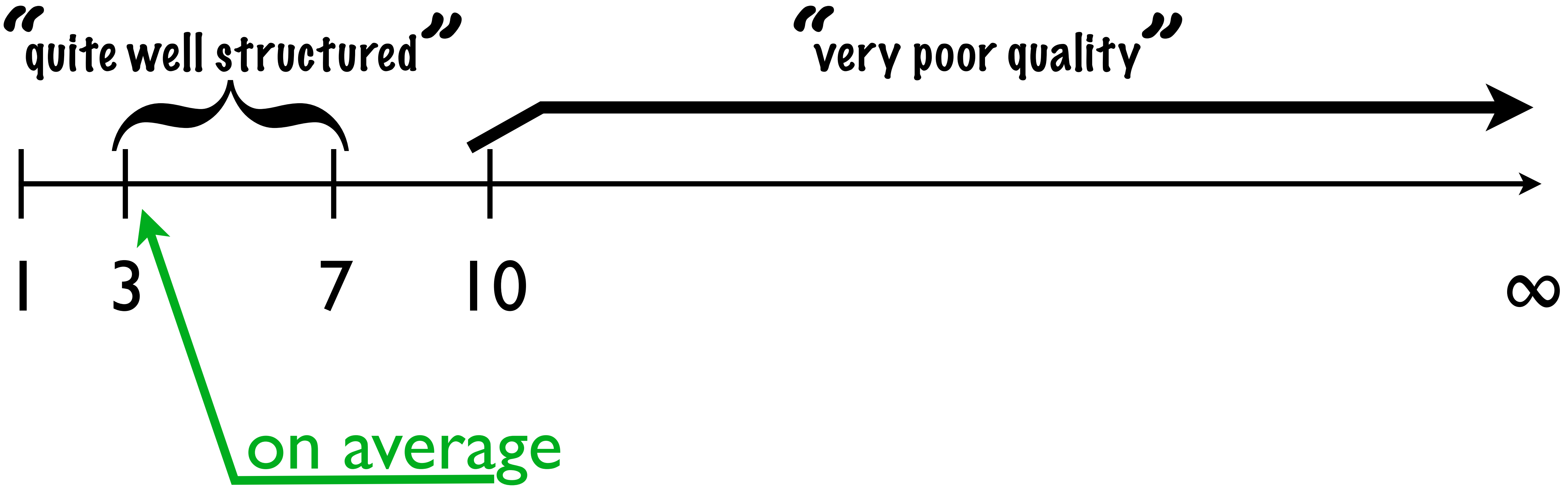
“very poor quality”

Evaluation: Productivity

Complexity of the developer’s code

“number of linearly independent paths in a source code”

McCabe cyclomatic complexity



“quite well structured”

“very poor quality”

Summary

- The approach covers various domains
- The frameworks are *easy to use*
- *Few code* is required and this code is of *good quality*

Pursuing this evaluation with software engineers

Results

Architect Developer Conformance

Scientific contributions

- A design language dedicated to SCC (ICSE'11)
- The generation of a dedicated programming framework (GPCE'09)
- The evaluation of this approach (*submitted*)

Technical contributions

- A compiler for the design language (9 KLoC)
- A code generator targeting Java (4 KLoC)

Dissemination

- Demonstrations (PerCom'10), posters (SPLASH'10), visits (Bern, Potsdam)
- Public release (http://diasuite.inria.fr)

A Research Vehicle

This design language and code generator are part of a research project which involves

- 4 industrial partnerships
- 2 other research groups
- > 20 real applications
- 24/7 running platform
- 28,000 lines of code

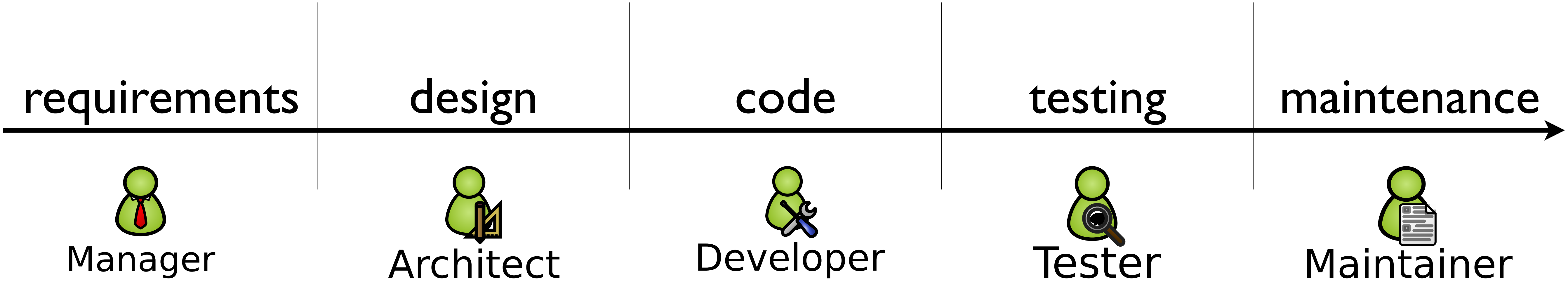
A Research Vehicle

7 PhD students leveraging DiaSpec and the generator

- QoS (FASE'11)
- error-handling (OOPSLA'10)
- virtual testing (Mobiquitous'09 and '10)
- SIP (ICC'10, ICIN'09, IPTComm'08)
- end-user programming (DSLWC'09)
- security (ICPS'09)

Perspectives

- Can we support other stages of the software life-cycle?



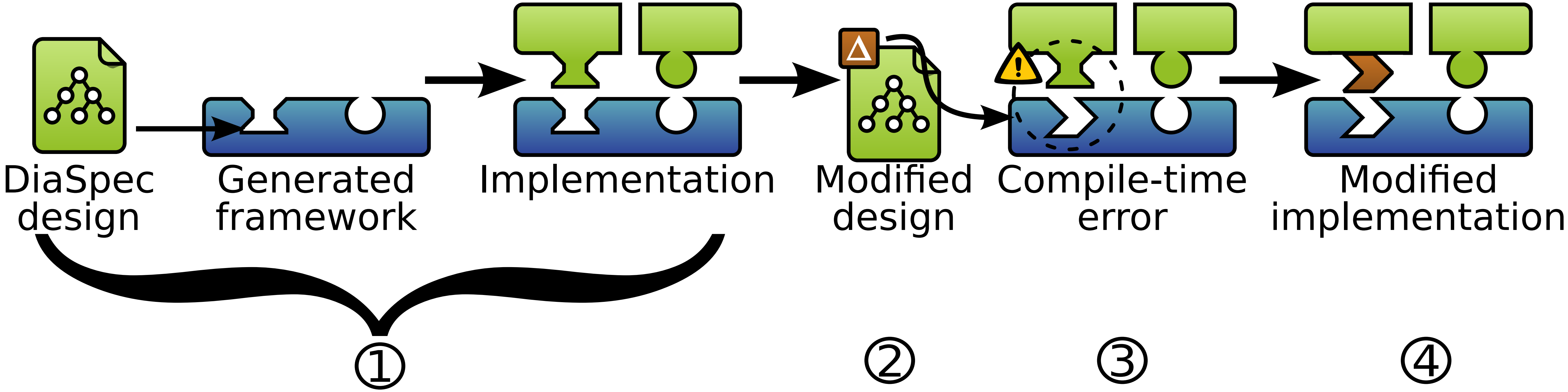
requirements
design
code
testing
maintenance

Manager
Architect
Developer
Tester
Maintainer

- Can we transpose the approach to another paradigm?

- Can we help creating such approaches?

Facilitating Evolution



- eases developer’s work by
    - showing mismatches
    - leveraging development tools
- ensures conformance all along the software life-cycle