Context-Oriented Programming for Adaptive Software Systems

Tetsuo Kamina (Ritsumeikan University)

Outline

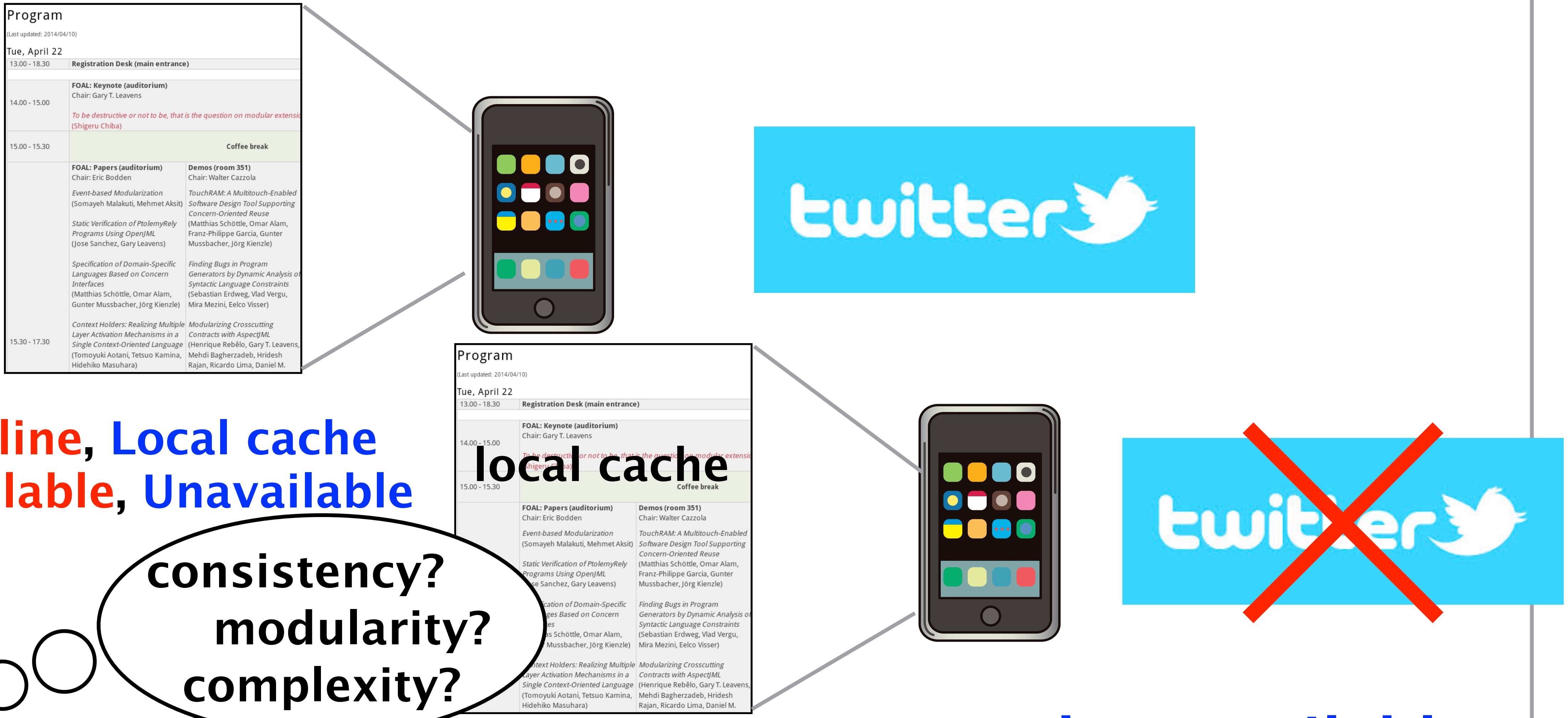
- Short introduction to context-oriented programming(COP)
  - ServalCJ: our achievement in COP language
- Our position & vision for context-oriented software engineering (COSE)
- Discussion on applying COP & COSE to adaptive software systems

Context-Oriented Programming (COP)

[Costanza05, Hirschfeld08]

- New programming paradigm for modularizing context-dependent behavior
- Promising for context-aware systems
  - e.g. conference guide system

Network available



Program

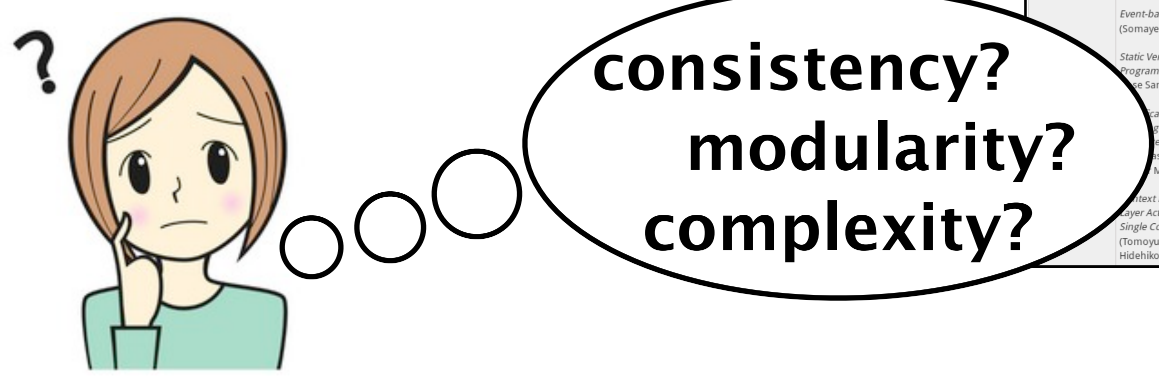
(Last updated: 2014/04/10)

Tue, April 22

13.00 - 18.30	**Registration Desk (main entrance)**	
14.00 - 15.00	**FOAL: Keynote (auditorium)** Chair: Gary T. Leavens *To be destructive or not to be, that is the question on modular extensio* (Shigeru Chiba)	
15.00 - 15.30	**Coffee break**	
15.30 - 17.30	**FOAL: Papers (auditorium)** Chair: Eric Bodden *Event-based Modularization* (Somayeh Malakuti, Mehmet Aksit) *Static Verification of PtolemyRely Programs Using OpenJML* (Jose Sanchez, Gary Leavens) *Specification of Domain-Specific Languages Based on Concern Interfaces* (Matthias Schöttle, Omar Alam, Gunter Mussbacher, Jörg Kienzle) *Context Holders: Realizing Multiple Layer Activation Mechanisms in a Single Context-Oriented Language* (Tomoyuki Aotani, Tetsuo Kamina, Hidehiko Masuhara)	**Demos (room 351)** Chair: Walter Cazzola *TouchRAM: A Multitouch-Enabled Software Design Tool Supporting Concern-Oriented Reuse* (Matthias Schöttle, Omar Alam, Franz-Philippe Garcia, Gunter Mussbacher, Jörg Kienzle) *Finding Bugs in Program Generators by Dynamic Analysis of Syntactic Language Constraints* (Sebastian Erdweg, Vlad Vergu, Mira Mezini, Eelco Visser) *Modularizing Crosscutting Contracts with AspectJML* (Henrique Rebêlo, Gary T. Leavens, Mehdi Bagherzadeb, Hridesh Rajan, Ricardo Lima, Daniel M.

Network unavailable

Program: Online, Local cache
Twitter: Available, Unavailable



Context-Oriented Programming (COP)

[Costanza05, Hirschfeld08]

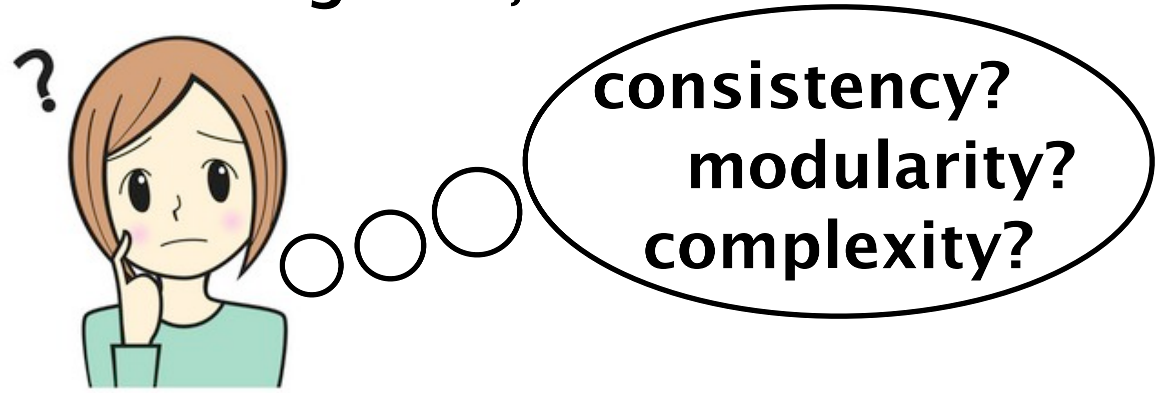
- New programming paradigm for modularizing context-dependent behavior
- Promising for context-aware systems
  - e.g. conference guide system

Outdoors

本郷
センタービル
本郷T&S ビル
壱岐坂通り
三念寺

Indoors

Map: City map, Floor plan
Positioning: GPS, Wi-Fi



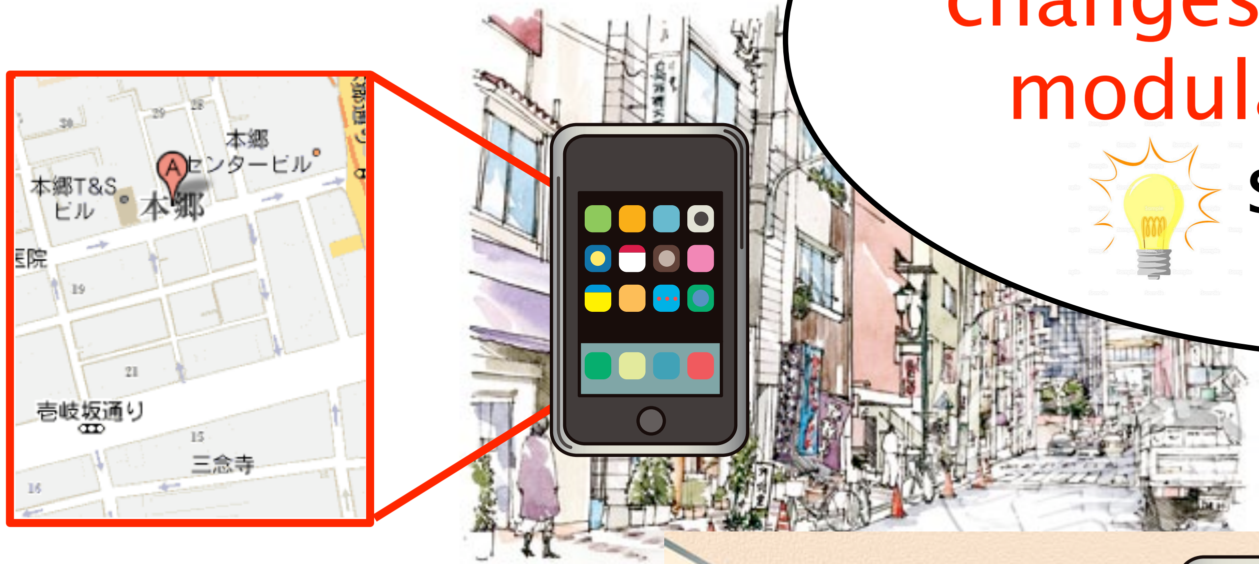
Context-Oriented Programming (COP)

[Costanza05, Hirschfeld08]

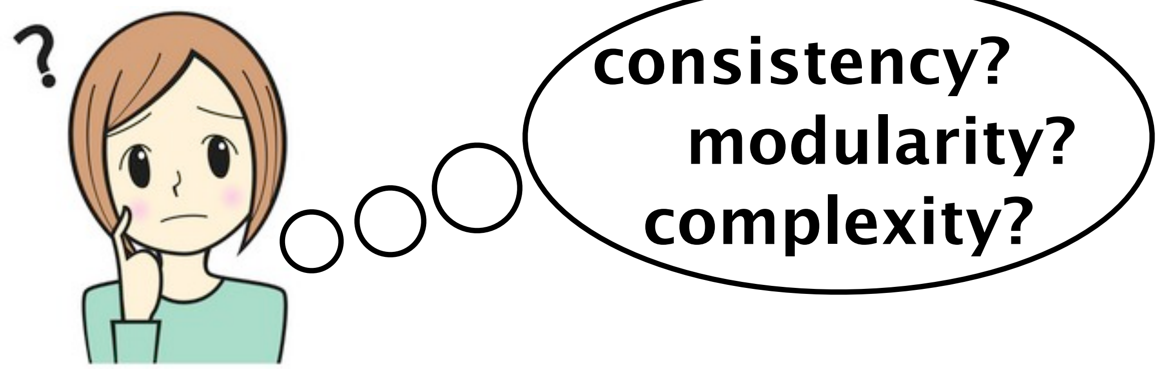
- New programming paradigm for modularizing context-dependent behavior
- Promising for context-a
  - e.g. conference guide s

Supporting **foreseeable dynamic changes** in behavior and **modularized** systems structure

Outdoors



Map: City map, Floor plan
Positioning: GPS, Wi-Fi



キャンパス
グレイテス
フルハウス

Indoors

Linguistic Constructs for COP

- **Layer**
  - Modularizing context-dependent behavior
  - Lightweight alternative to AOP/FOP

- **Layer activation**
  - Specifying the scope of effect of layers

```
d.display();

p.getPos();
```

```
layer Outdoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

```
layer Indoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

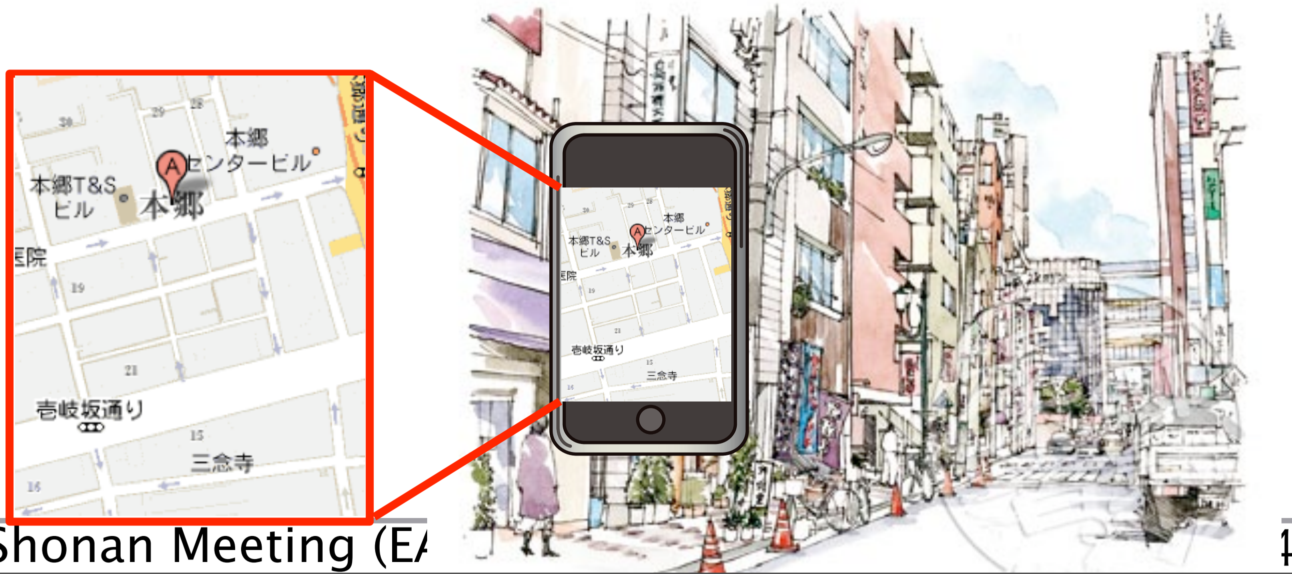
Linguistic Constructs for COP

- **Layer**
  - Modularizing context-dependent behavior
  - Lightweight alternative to AOP/FOP
- **Layer activation**
  - Specifying the scope of effect of layers

```
d.display();

p.getPos();
```

```
layer Outdoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

```
layer Indoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```



Linguistic Constructs for COP

- Layer
  - Modularizing context-dependent behavior
  - Lightweight alternative to AOP/FOP
- Layer activation
  - Specifying the scope of effect of layers

```
d.display();

p.getPos();
```

```
layer Outdoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

```
layer Indoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

Linguistic Constructs for COP

- **Layer**
  - Modularizing context-dependent behavior
  - Lightweight alternative to AOP/FOP

syntax? semantics?

- **Layer activation**
  - Specifying the scope of effect of layers

```
d.display();

p.getPos();
```

```
layer Outdoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

```
layer Indoors {
  class Display {
    void display() {..}}
  class Position {
    void getPos() {..}} }
```

Our motivation for new COP language

- Several activation mechanisms for specific use cases

- Just a combination is not sufficient

Our motivation for new COP language

- Several activation mechanisms for specific use cases

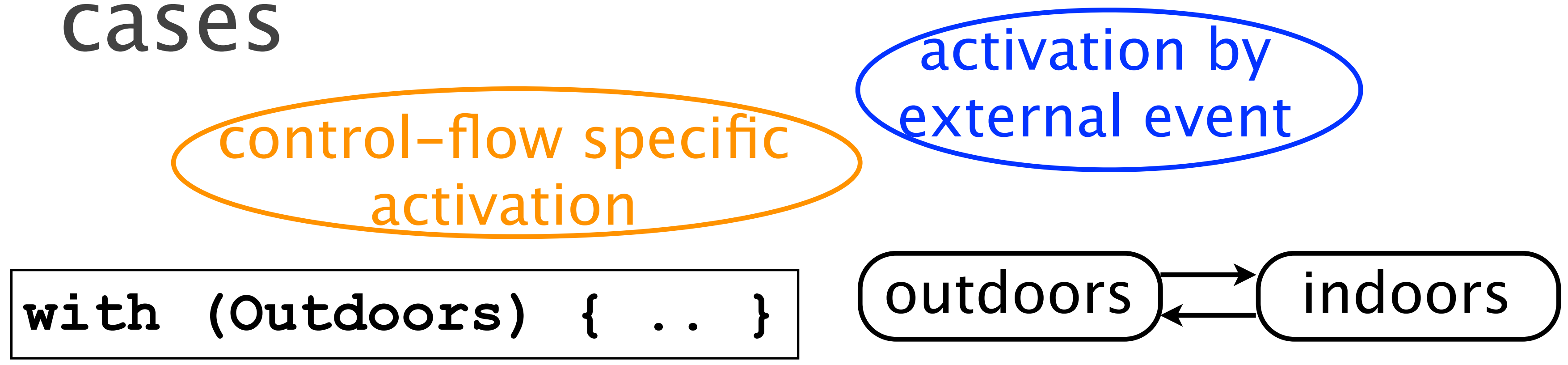
control-flow specific activation

```
with (Outdoors) { .. }
```

- Just a combination is not sufficient

Our motivation for new COP language

- Several activation mechanisms for specific use cases


control-flow specific activation
activation by external event
with (Outdoors) { .. }
outdoors
indoors

- Just a combination is not sufficient

Our motivation for new COP language

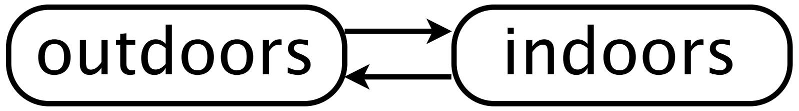
- Several activation mechanisms for specific use cases

control-flow specific activation

activation by external event

global

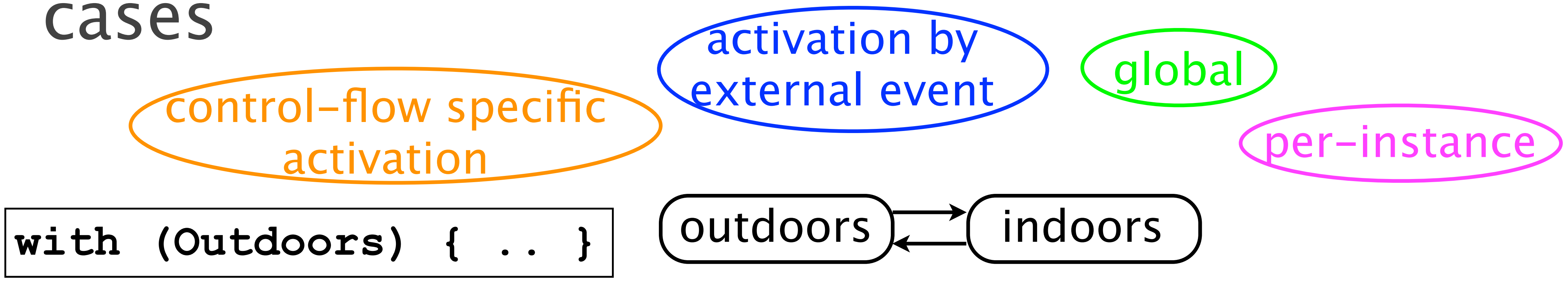
```
with (Outdoors) { .. }
```



- Just a combination is not sufficient

Our motivation for new COP language

- Several activation mechanisms for specific use cases

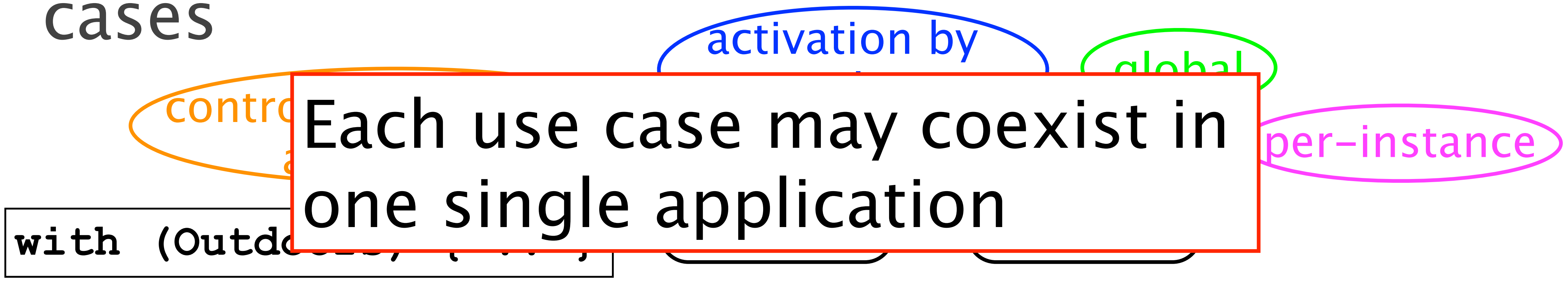

control–flow specific activation
activation by external event
global
per–instance
outdoors
indoors

```
with (Outdoors) { .. }
```

- Just a combination is not sufficient

Our motivation for new COP language

- Several activation mechanisms for specific use cases

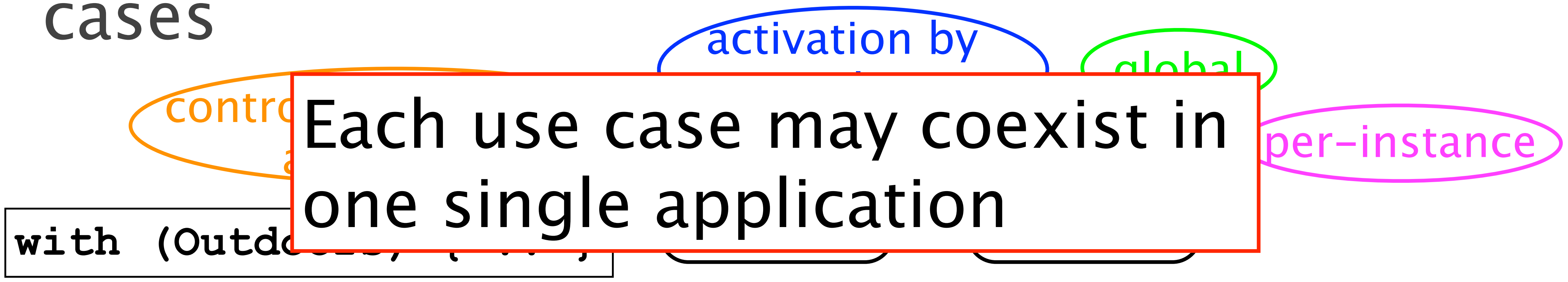


Each use case may coexist in one single application

- Just a combination is not sufficient

Our motivation for new COP language

- Several activation mechanisms for specific use cases



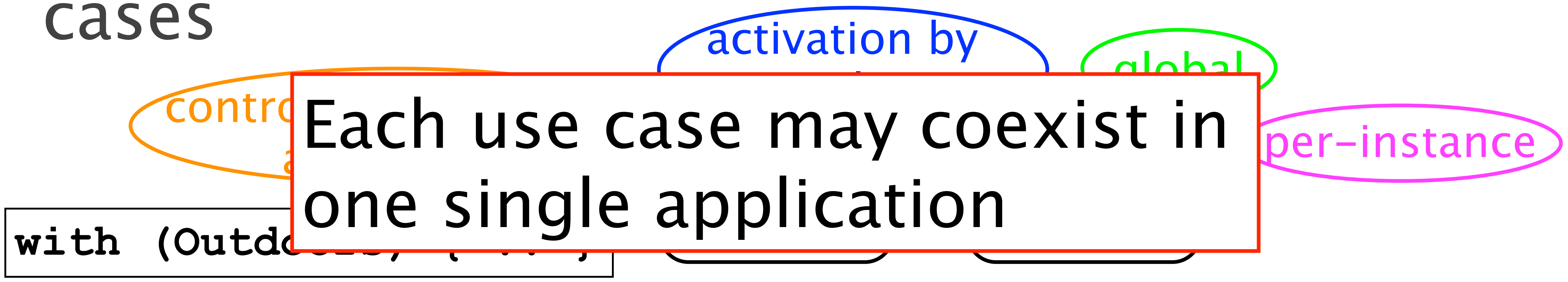
Each use case may coexist in one single application

- Just a combination is not sufficient

	control–flow	event–based
per–thread		
global		

Our motivation for new COP language

- Several activation mechanisms for specific use cases



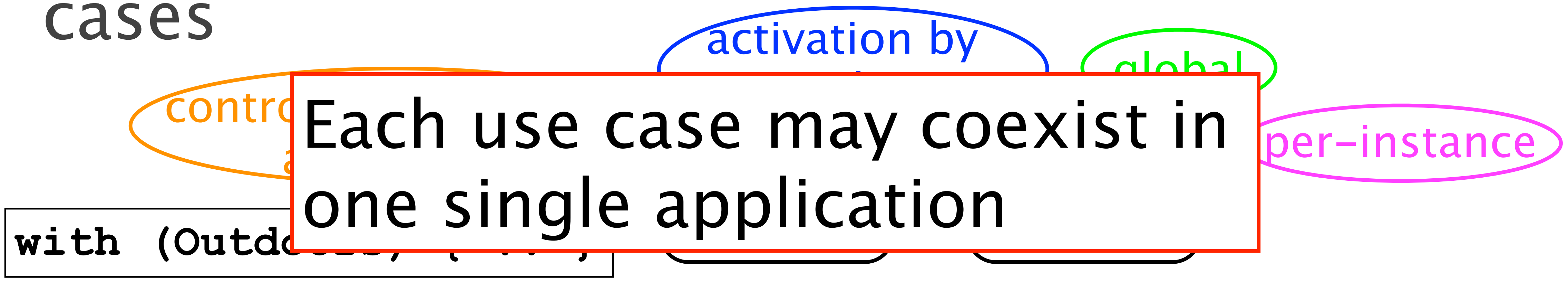
Each use case may coexist in one single application

- Just a combination is not sufficient

	control-flow	event-based
per-thread	Language A	
global		

Our motivation for new COP language

- Several activation mechanisms for specific use cases



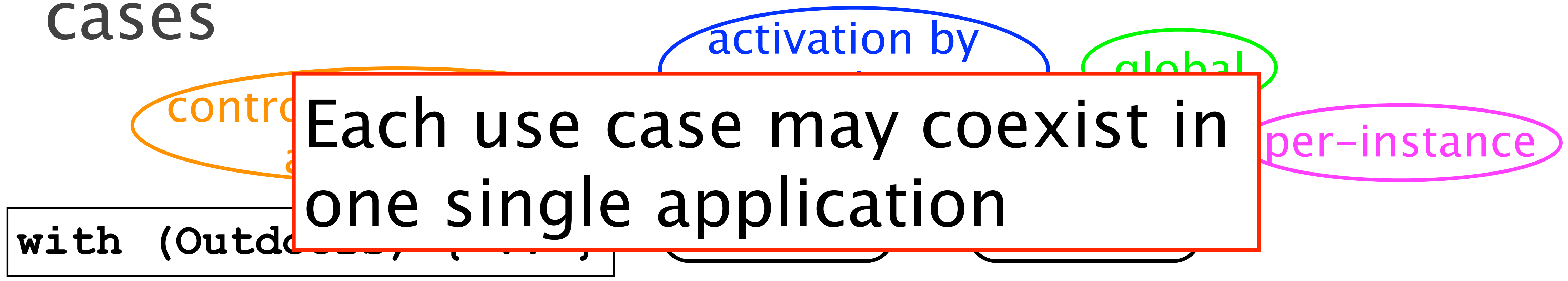
Each use case may coexist in one single application

- Just a combination is not sufficient

	control–flow	event–based
per–thread	Language A	
global		Language B

Our motivation for new COP language

- Several activation mechanisms for specific use cases



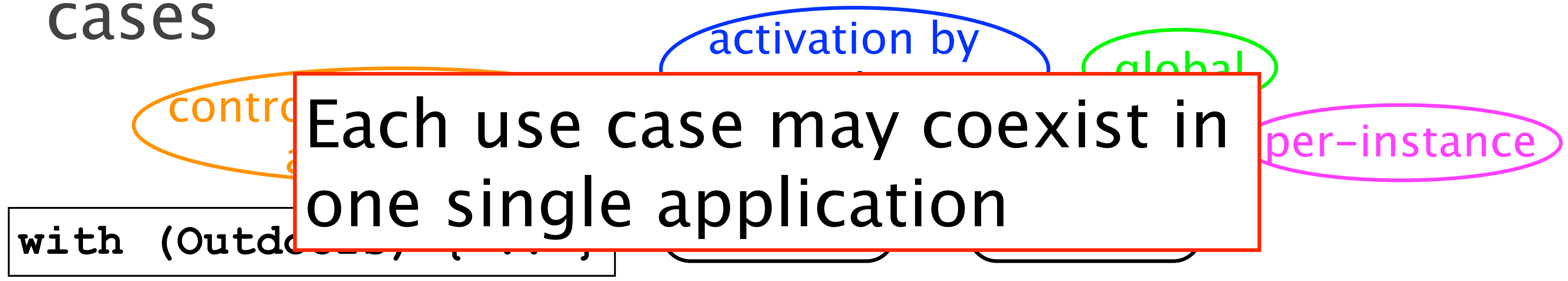
Each use case may coexist in one single application

- Just a combination is not sufficient

	control–flow	event–based
per–thread	Language A	Not covered!
global	Not covered!	Language B

Our motivation for new COP language

- Several activation mechanisms for specific use cases



Each use case may coexist in one single application

- Just a combination is not sufficient

	control-flow	event-based
per-thread	Language A	Not covered!
global	Not covered!	Language B

Activation mechanism **beyond** existing COP is required

Our achievement: ServalCJ

[Kamina15, presented in MODULARITY'15]

- Generalizing layer activation mechanisms
  - Contexts: duration of activation
  - Subscribers: the activation targets
    - Can be any sets of instances, whole application, and particular threads

T. Kamina et al., Generalized Layer Activation Mechanism through Contexts and Subscribers, In MODULARITY'15.

Our achievement: ServalCJ
[Kamina15, presented in MODULARITY'15]

- Generalizing layer activation mechanisms
  - Contexts: duration of activation

```
activate Outdoors
  if(m.isProviderEnabled(m.GPS_PROVIDER))
```

  - Subscribers: the activation targets
    - Can be any sets of instances, whole application, and particular threads

T. Kamina et al., Generalized Layer Activation Mechanism through Contexts and Subscribers, In MODULARITY'15.

Our achievement: ServalCJ

[Kamina15, presented in MODULARITY'15]

- Generalizing layer activation mechanisms
  - Contexts: duration of activation

```
activate Outdoors  Name of Layer
  if(m.isProviderEnabled(m.GPS_PROVIDER))
```

  - Subscribers: the activation targets
    - Can be any sets of instances, whole application, and particular threads

T. Kamina et al., Generalized Layer Activation Mechanism through Contexts and Subscribers, In MODULARITY'15.

Our achievement: ServalCJ
[Kamina15, presented in MODULARITY'15]

- Generalizing layer activation mechanisms
  - Contexts: duration of activation

```
activate Outdoors
  if(m.isProviderEnabled(m.GPS_PROVIDER))
```

Name of Layer

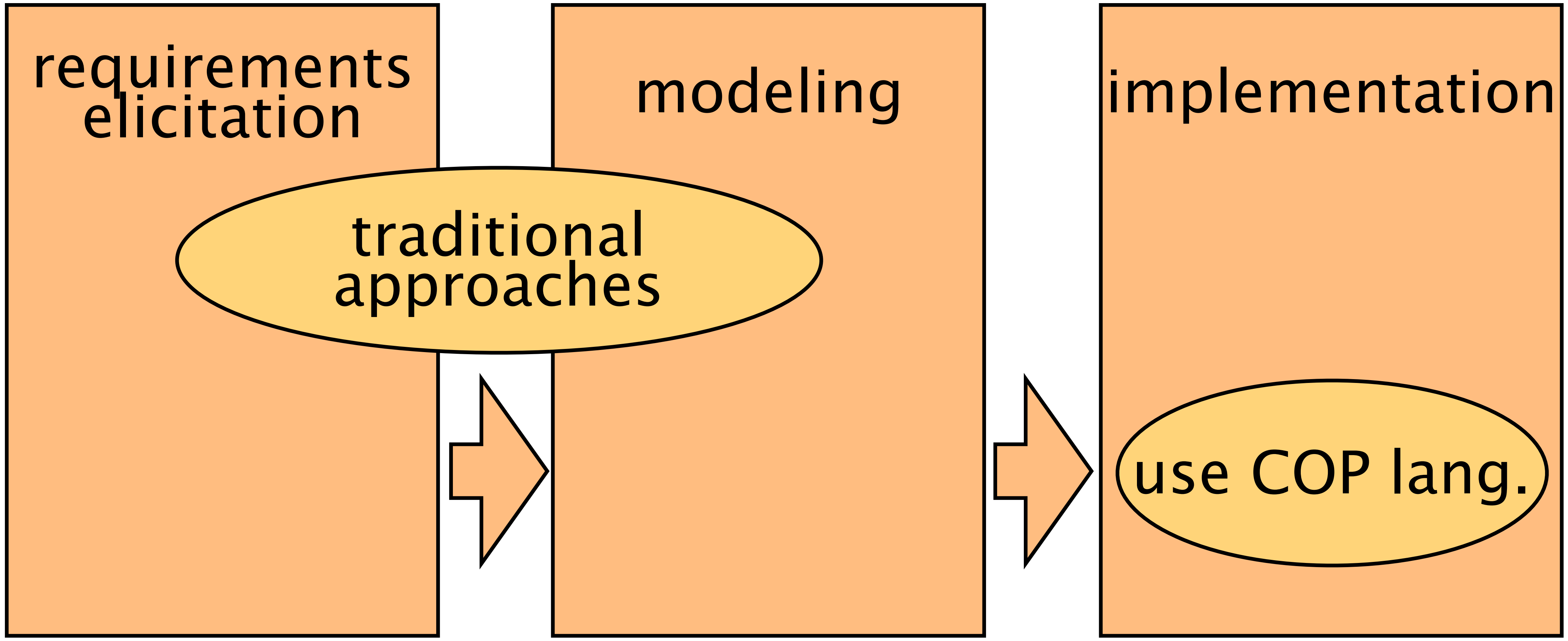
Condition specifying when the layer is active (including control-flows and temporal terms)

- Subscribers: the activation targets
  - Can be any sets of instances, whole application, and particular threads

T. Kamina et al., Generalized Layer Activation Mechanism through Contexts and Subscribers, In MODULARITY'15.

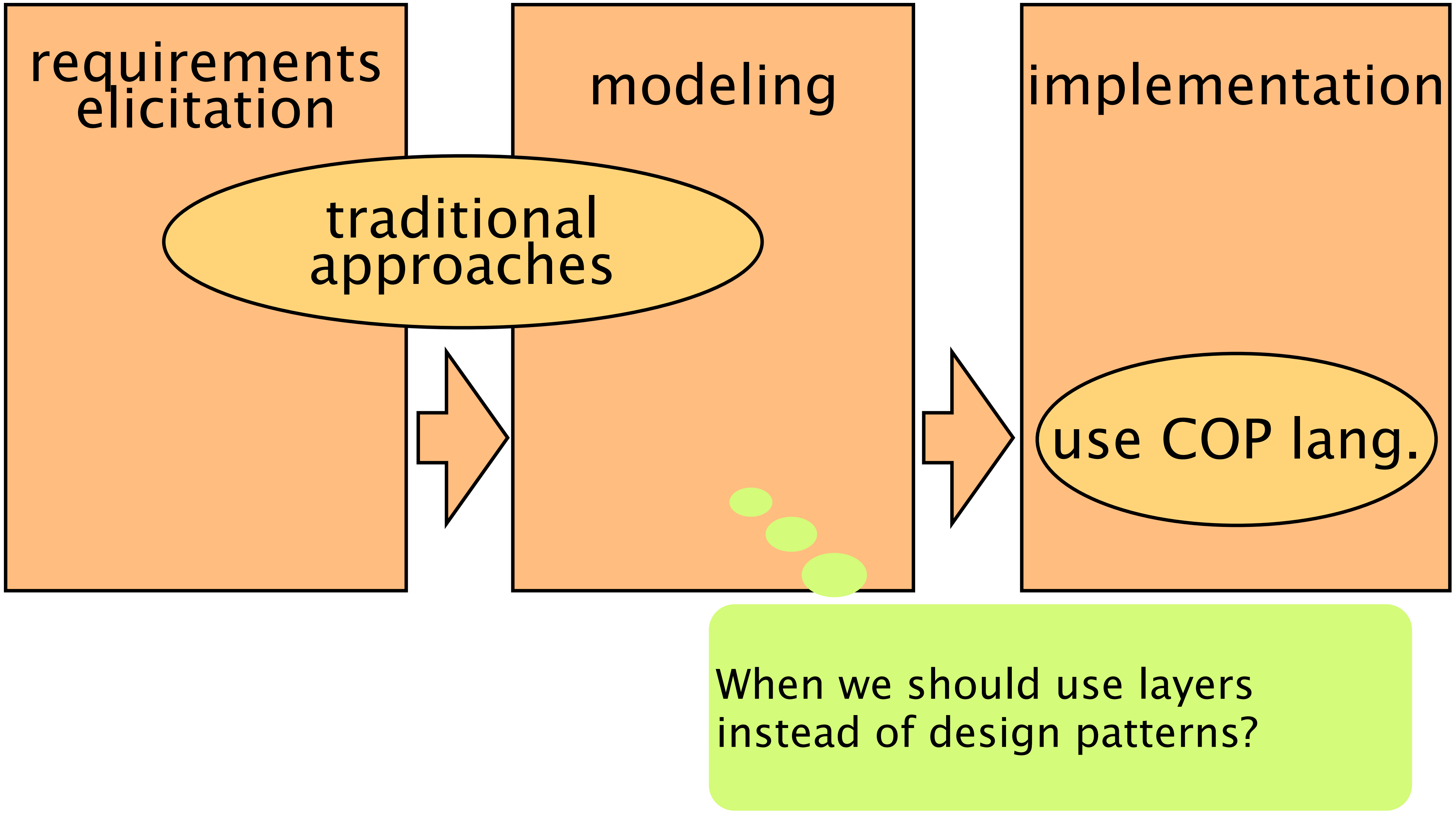
Problem: lack of methodology

- No methods to elicit and model context-dep. behavior



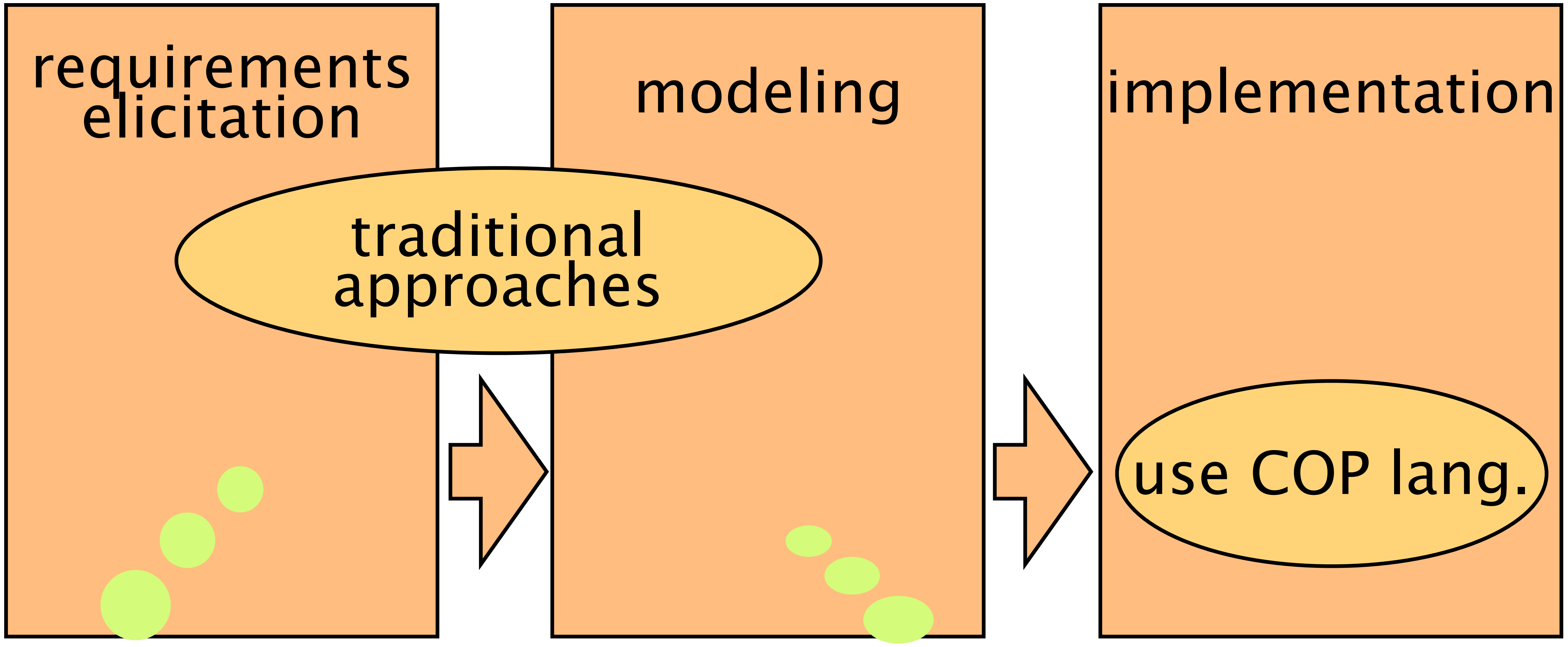
Problem: lack of methodology

- No methods to elicit and model context-dep. behavior



Problem: lack of methodology

- No methods to elicit and model context-dep. behavior

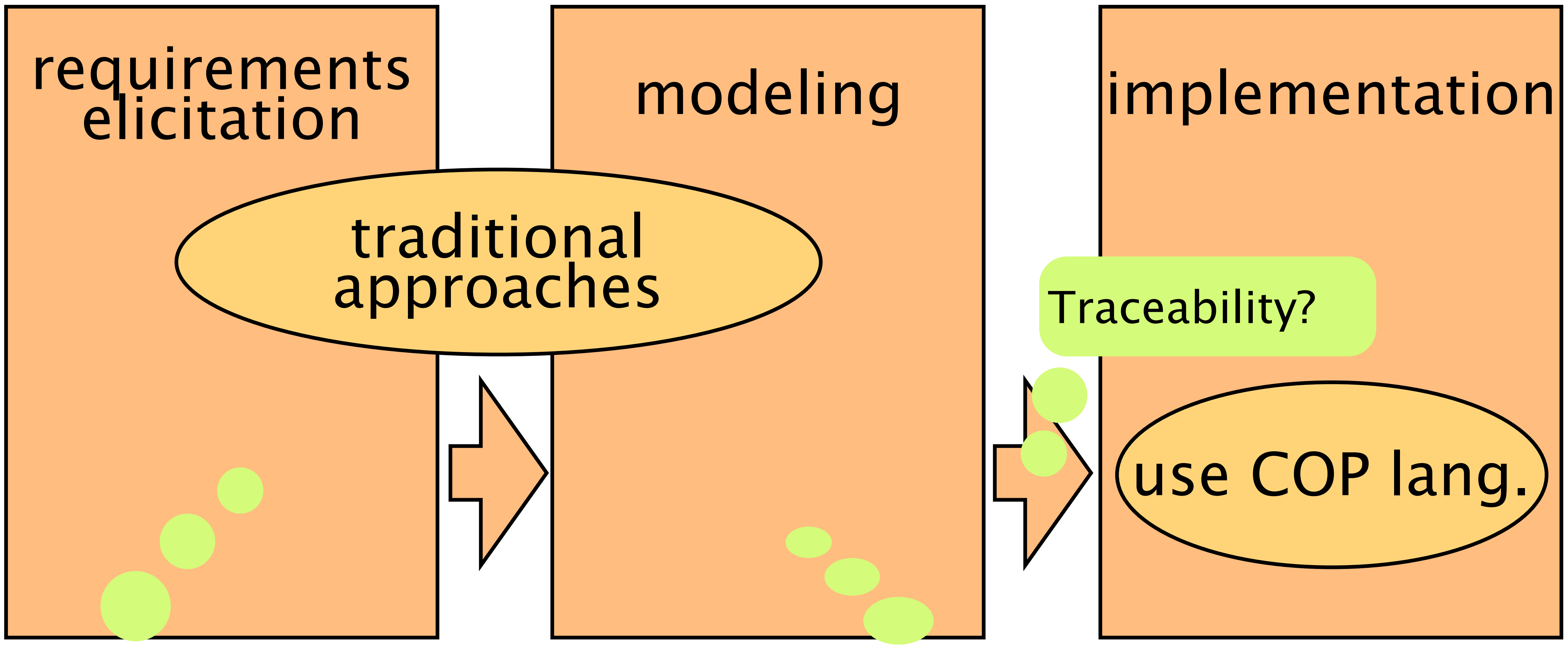


Requirements level contexts are vague
- outdoors, indoors are contexts ... why?
- the ID of the user is not a context ... why not?

When we should use layers instead of design patterns?

Problem: lack of methodology

- No methods to elicit and model context-dep. behavior



Requirements level contexts are vague
- outdoors, indoors are contexts ... why?
- the ID of the user is not a context ... why not?

When we should use layers instead of design patterns?

Our vision

- Presenting our preliminary study on COSE
  [Kamina14, presented in MODULARITY'14]]

Overview of the whole development process will lead us to further research on each stage of development process

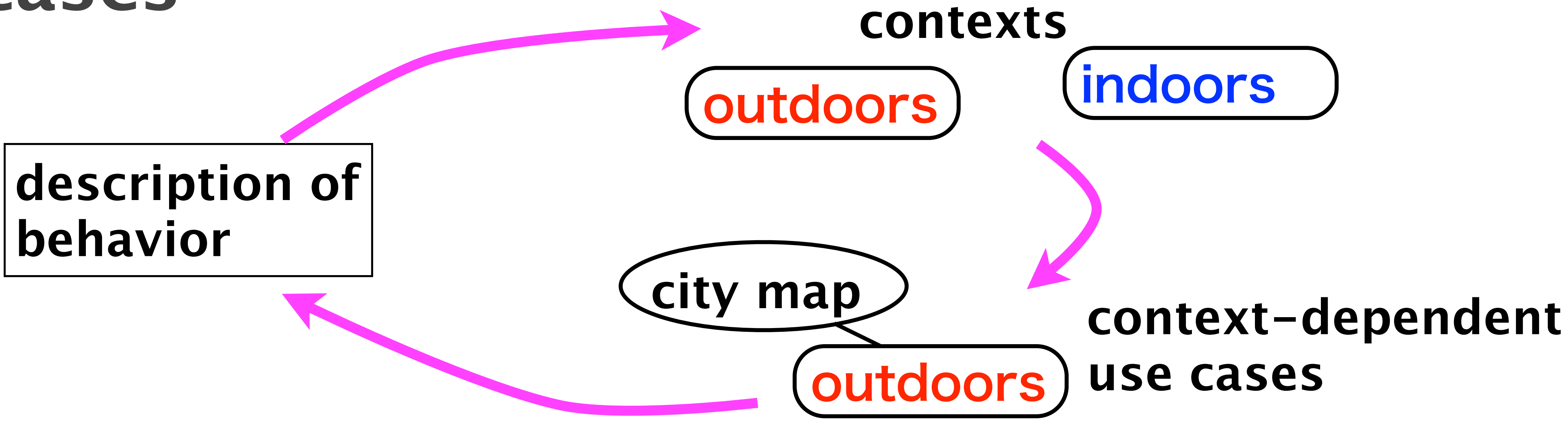
  - Principles for finding context-dependent behavior
  - Use-case-driven method based on those principles
  - A case study

Principles

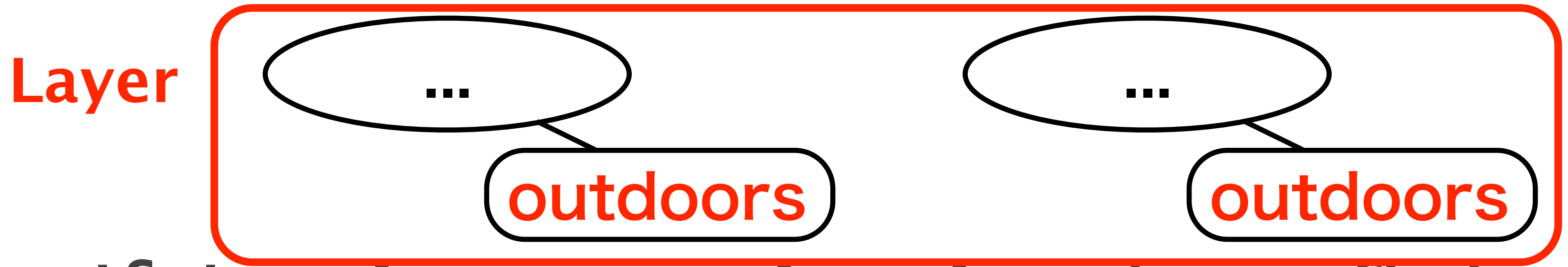
- Factors for dynamically changing behavior that exist outside the behavior are candidates for contexts
- Each such factor is a variable for a Boolean formula, which forms a context
- If multiple variations of behavior share the same context, they should be implemented by using a layer

Overview of COSE

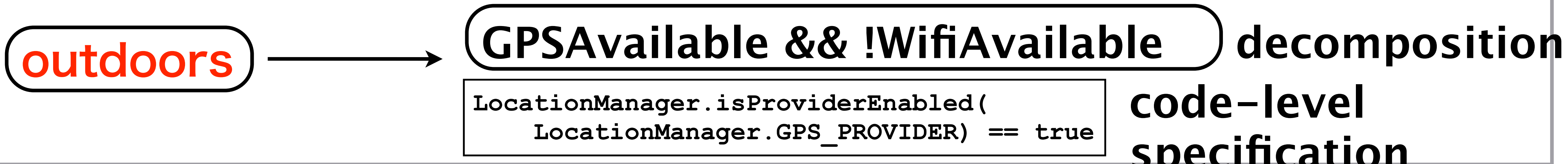
- Identifying **contexts** and **context-dependent use cases**



- Identifying **layers** by grouping use cases



- Identifying **layer activation** by refining definition of contexts



```
LocationManager.isProviderEnabled(
    LocationManager.GPS_PROVIDER) == true
```

Identifying contexts

- Twitter is available only when the Internet is available
- The user accesses the online program only when the Internet is available
- If the user is inside the venue, the system displays a floor plan
- The system can determine the position only when at least one positioning device is available

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

  - Twitter is available only when the Internet is available
  - The user accesses the online program only when the Internet is available
  - If the user is inside the venue, the system displays a floor plan
  - The system can determine the position only when at least one positioning device is available

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

- Twitter is available only when the Internet is available
- The user accesses the online program only when the Internet is available
- If the user is inside the venue, the system displays a floor plan
- The system can determine the position only when at least one positioning device is available

Find factors that change behavior

(※conditions existing outside the behavior)

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

- Twitter is available only when the Internet is available **hasNetwork**
- The user accesses the online program only when the Internet is available
- If the user is inside the venue, the system displays a floor plan **indoors**
- The system can determine the position only when at least one positioning device is available **hasPositioning**

Find factors that change behavior

(※conditions existing outside the behavior)

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

  - Twitter is available only when the Internet is available **hasNetwork**
  - The user accesses the online program only when the Internet is available
  - If the user is inside the venue, the system displays a floor plan **indoors**
  - The system can determine the position only when at least one positioning device is available **hasPositioning**

  Find factors that change behavior
  (※conditions existing outside the behavior)

- Refining variables to make them orthogonal

$$\begin{tikzcd}\textbf{outdoors} \\ \textbf{hasPositioning} \arrow[u, "\text{depends}"]\end{tikzcd}$$

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

- Twitter is available only when the Internet is available **hasNetwork**
- The user accesses the online program only when the Internet is available
- If the user is inside the venue, the system displays a floor plan **indoors**
- The system can determine the position only when at least one positioning device is available **hasPositioning**

Find factors that change behavior
(※conditions existing outside the behavior)

- Refining variables to make them orthogonal

$$\begin{tikzcd}
\textbf{outdoors} & & \textbf{outdoors} \\
 & \Rightarrow & \textbf{indoors} \\
\textbf{hasPositioning} \arrow[uu, "\text{depends}"] & & \textbf{cannotDecide}
\end{tikzcd}$$

Identifying contexts

- Identifying candidates for Boolean variables, which will be used to define **contexts**

• Twitter is available only when the Internet is available **hasNetwork**
• The user accesses the online program only when the Internet is available
• If the user is inside the venue, the system displays a floor plan
indoors
• The system can determine the position only when at least one positioning device is available
hasPositioning

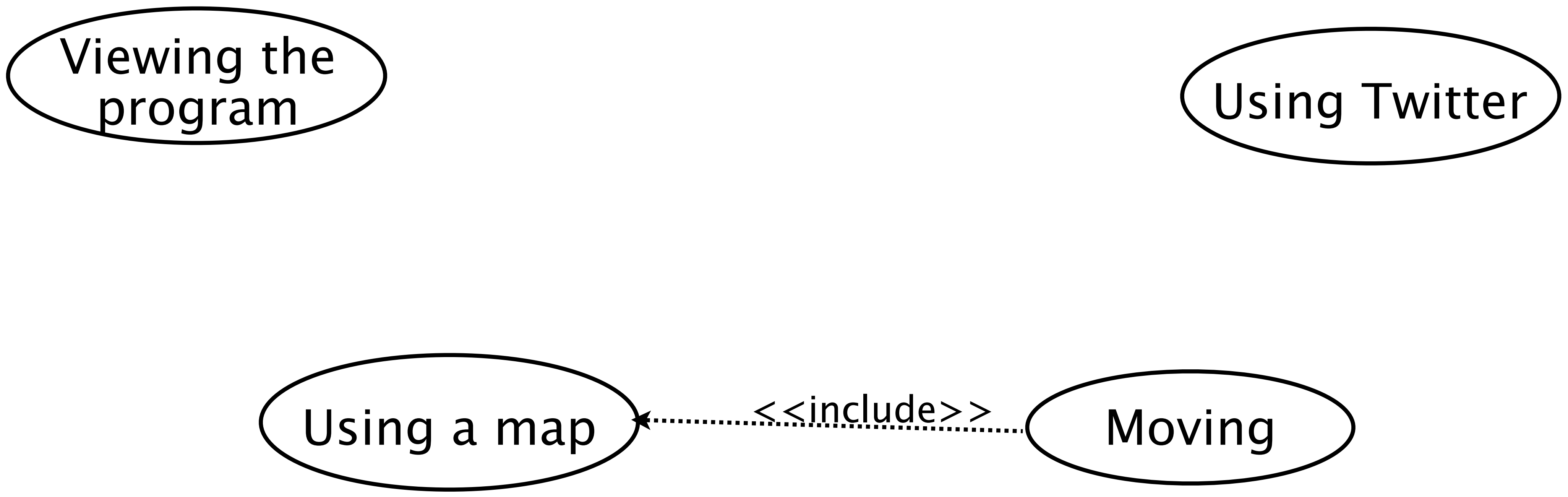
Find factors that change behavior
(※conditions existing outside the behavior)

- Refining variables to make them orthogonal

$$\begin{tikzcd} \textbf{outdoors} \\ \textbf{hasPositioning} \arrow[u, "\text{depends}"] \end{tikzcd} \quad \Rightarrow \quad \begin{array}{c} \textbf{outdoors} \\ \textbf{indoors} \\ \textbf{!outdoors \&\& !indoors} \end{array}$$

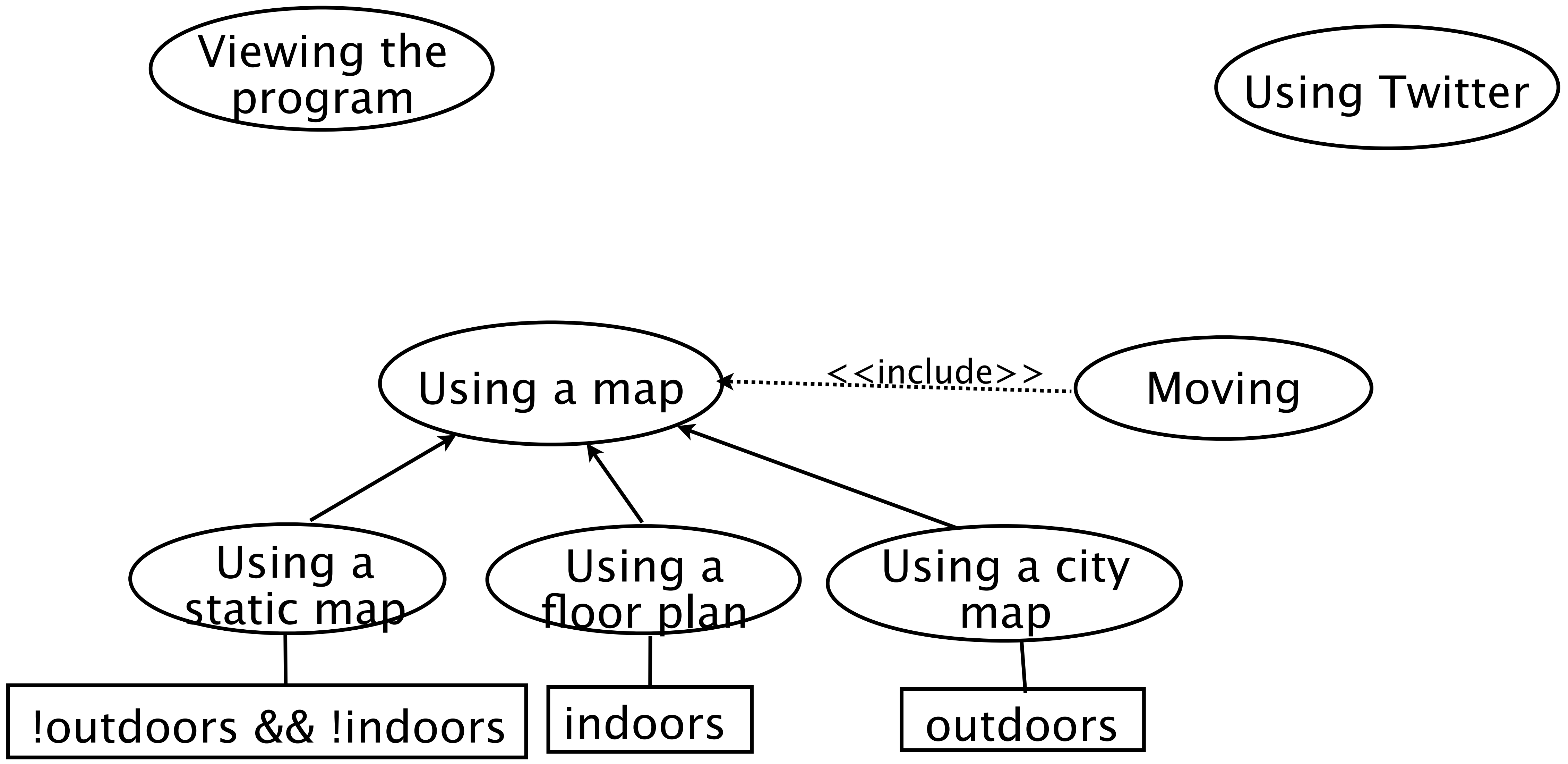
Defining context-dependent use cases

- Annotating use cases that are applicable in specific **contexts** (Boolean terms)



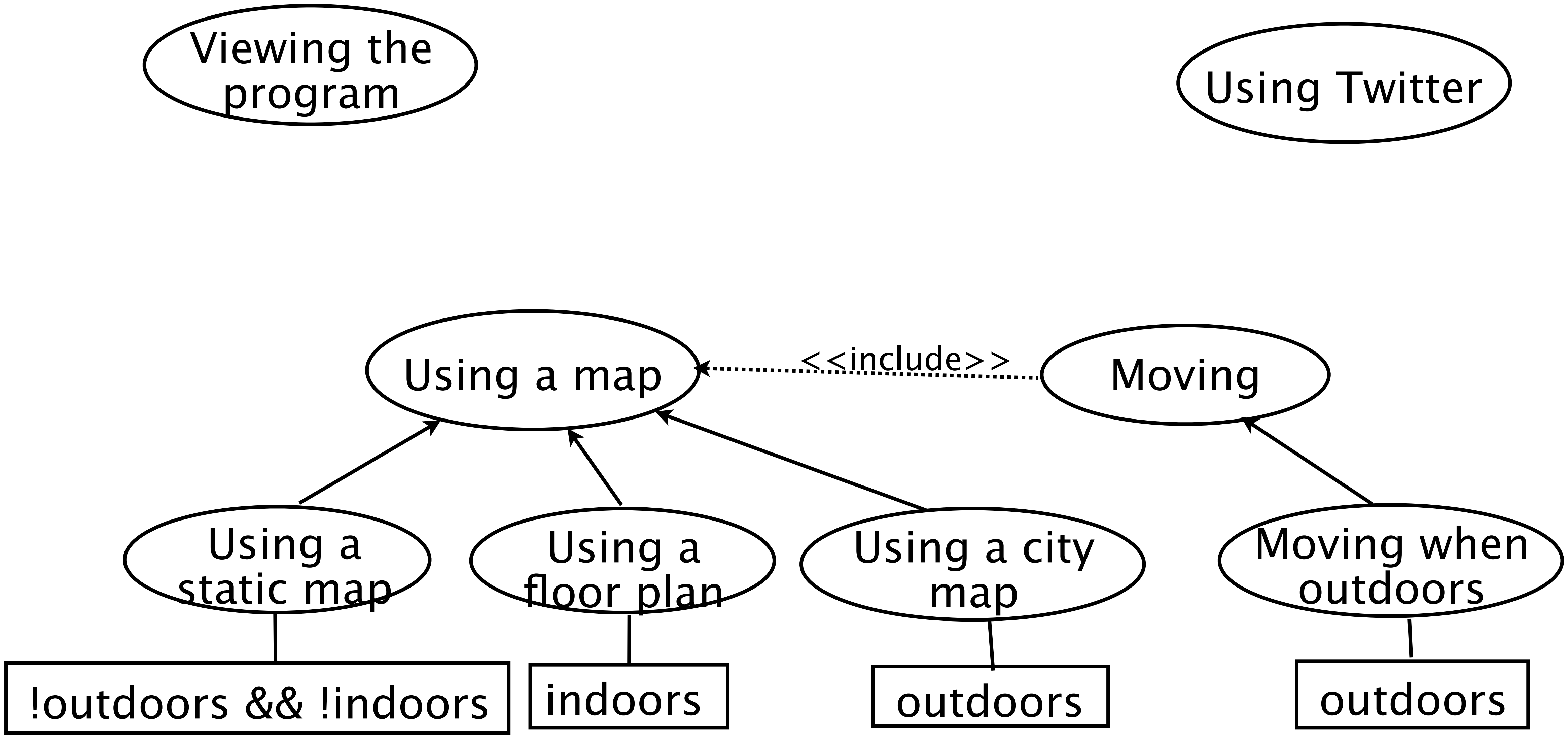
Defining context-dependent use cases

- Annotating use cases that are applicable in specific **contexts** (Boolean terms)



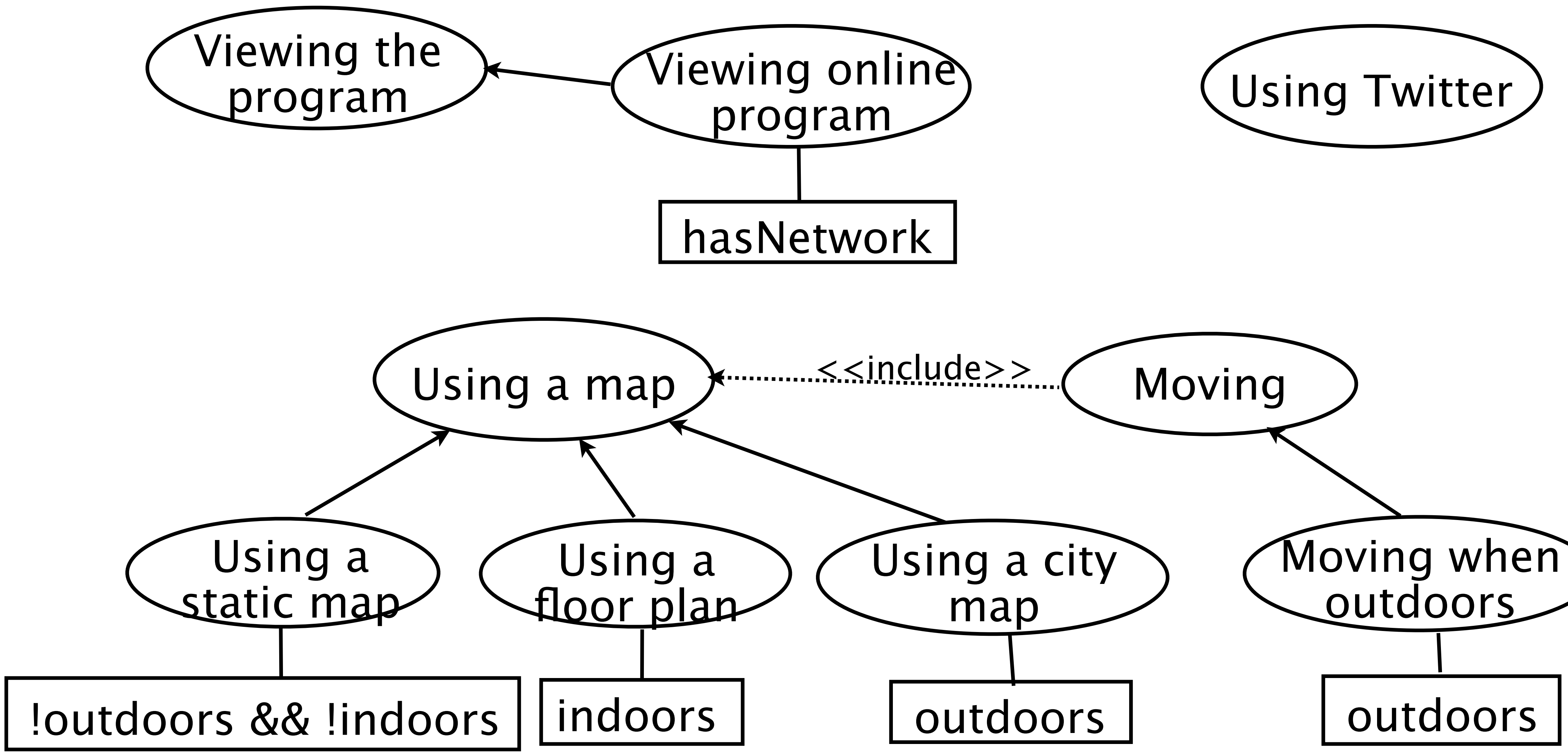
Defining context-dependent use cases

- Annotating use cases that are applicable in specific **contexts** (Boolean terms)



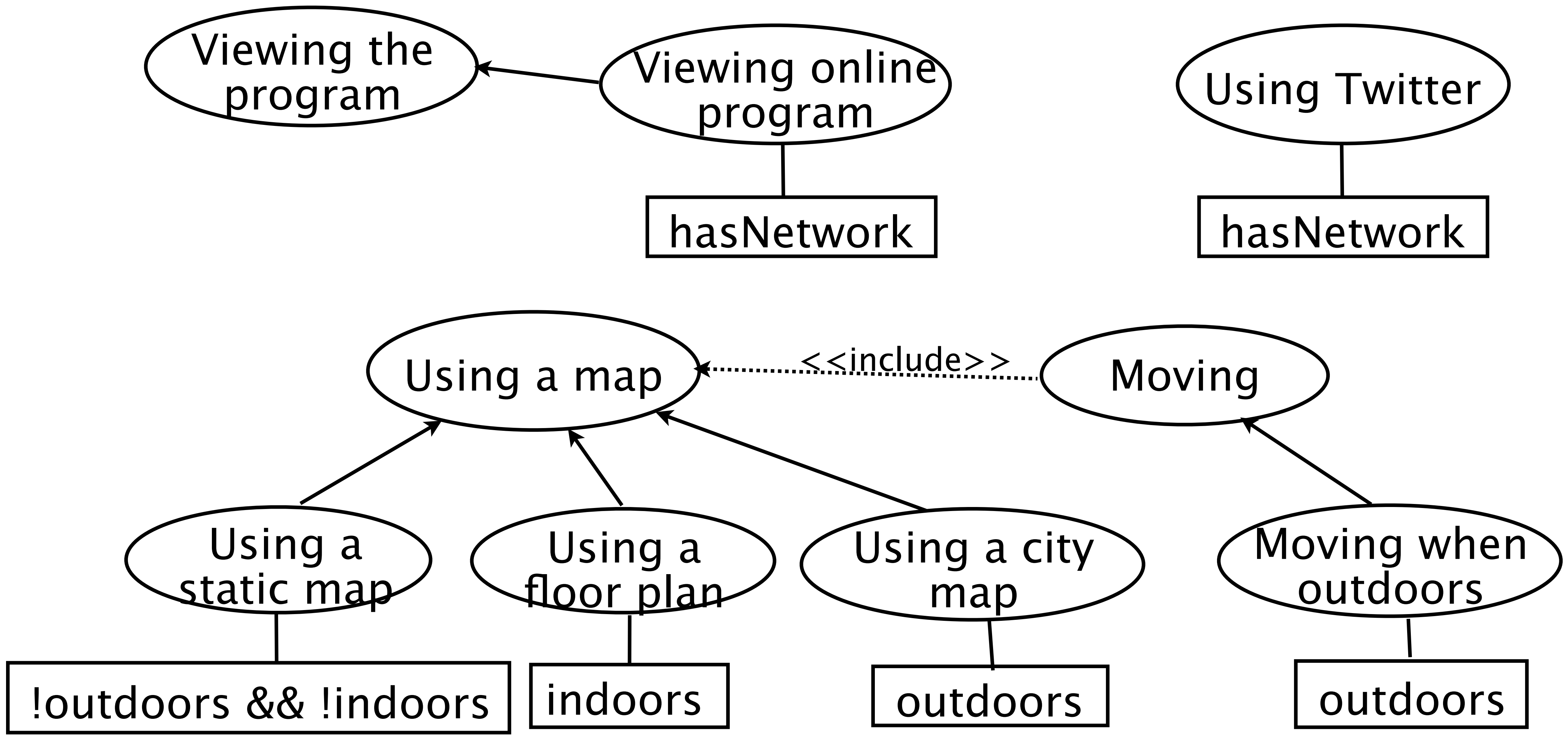
Defining context-dependent use cases

- Annotating use cases that are applicable in specific **contexts** (Boolean terms)

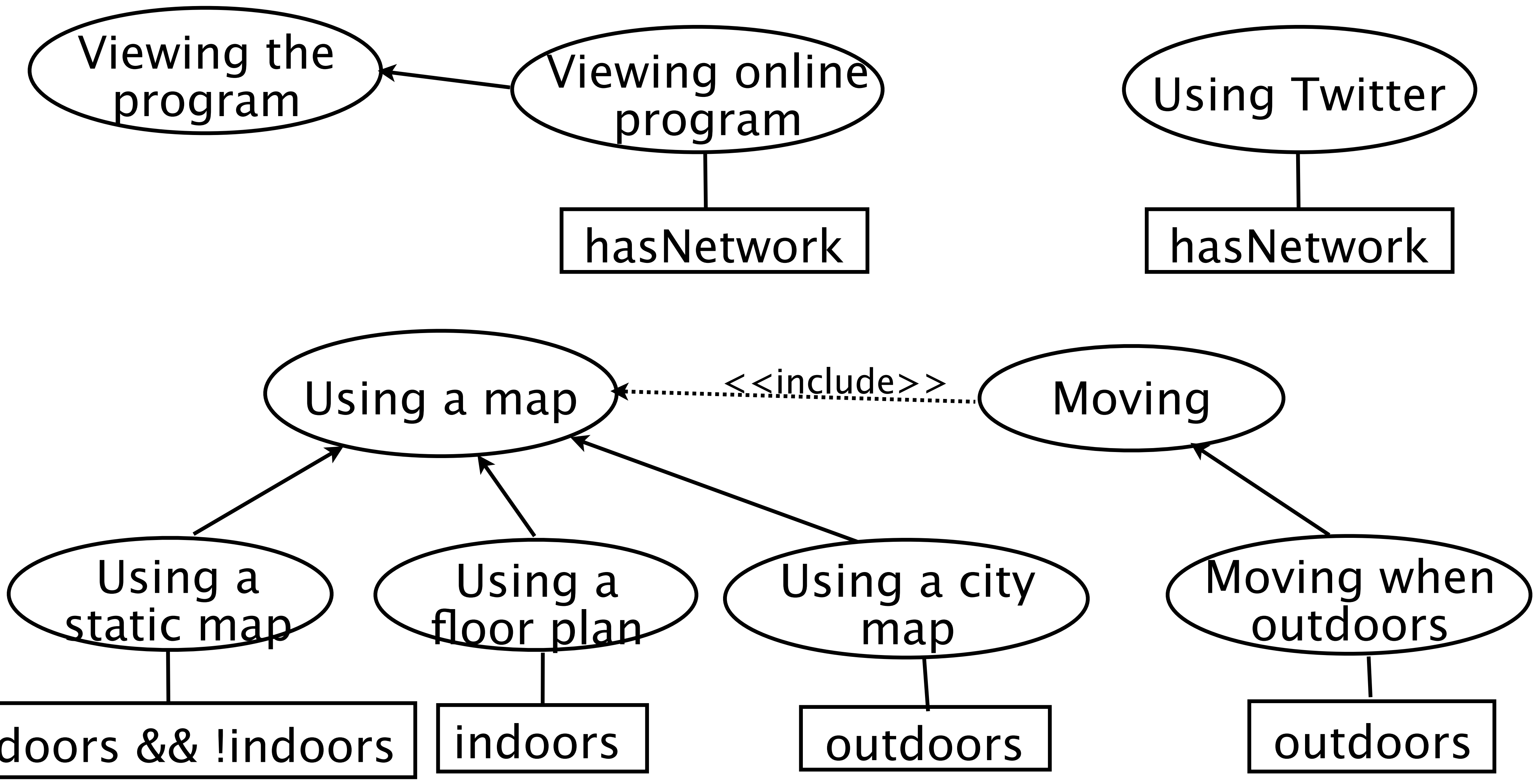


Defining context-dependent use cases

- Annotating use cases that are applicable in specific **contexts** (Boolean terms)

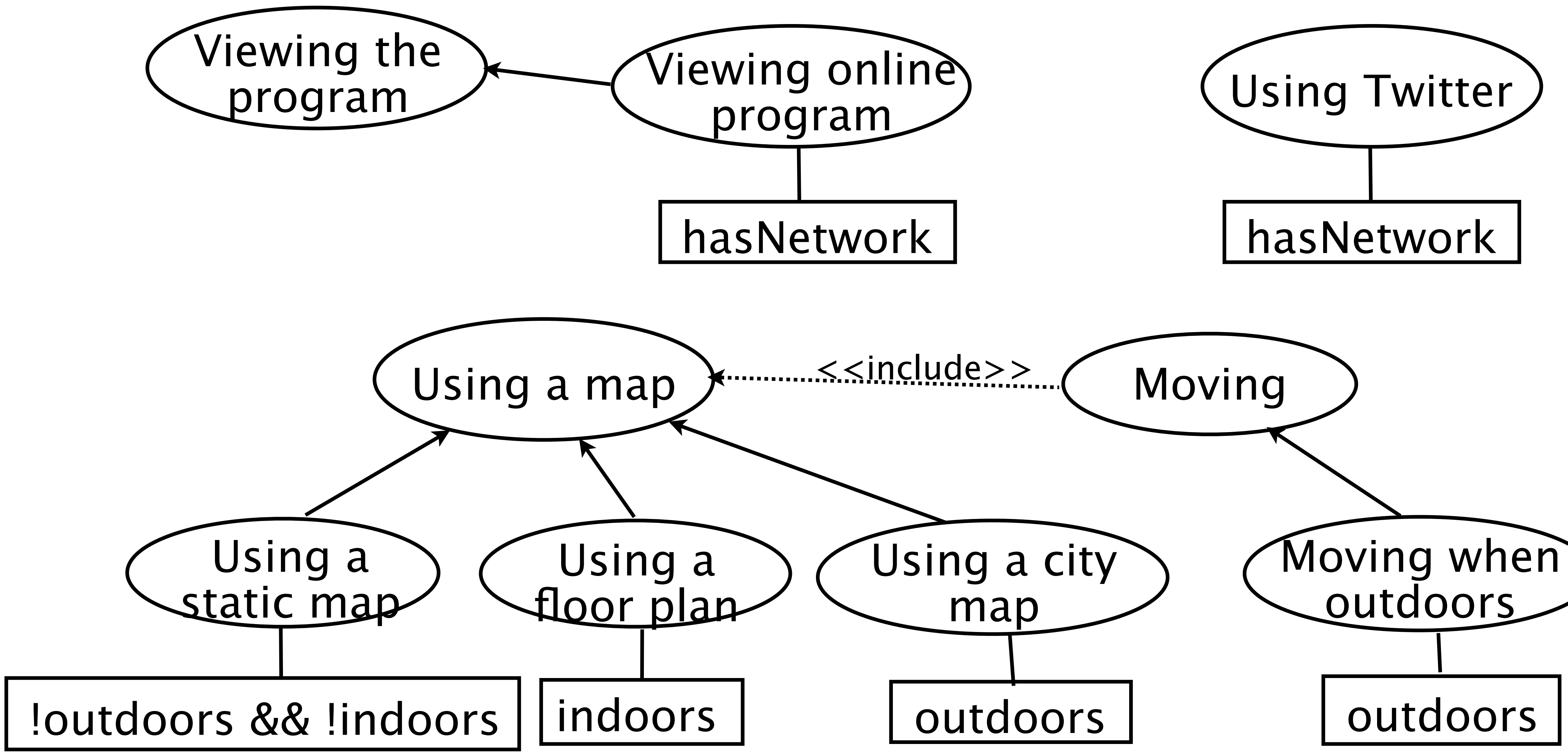


Grouping use cases



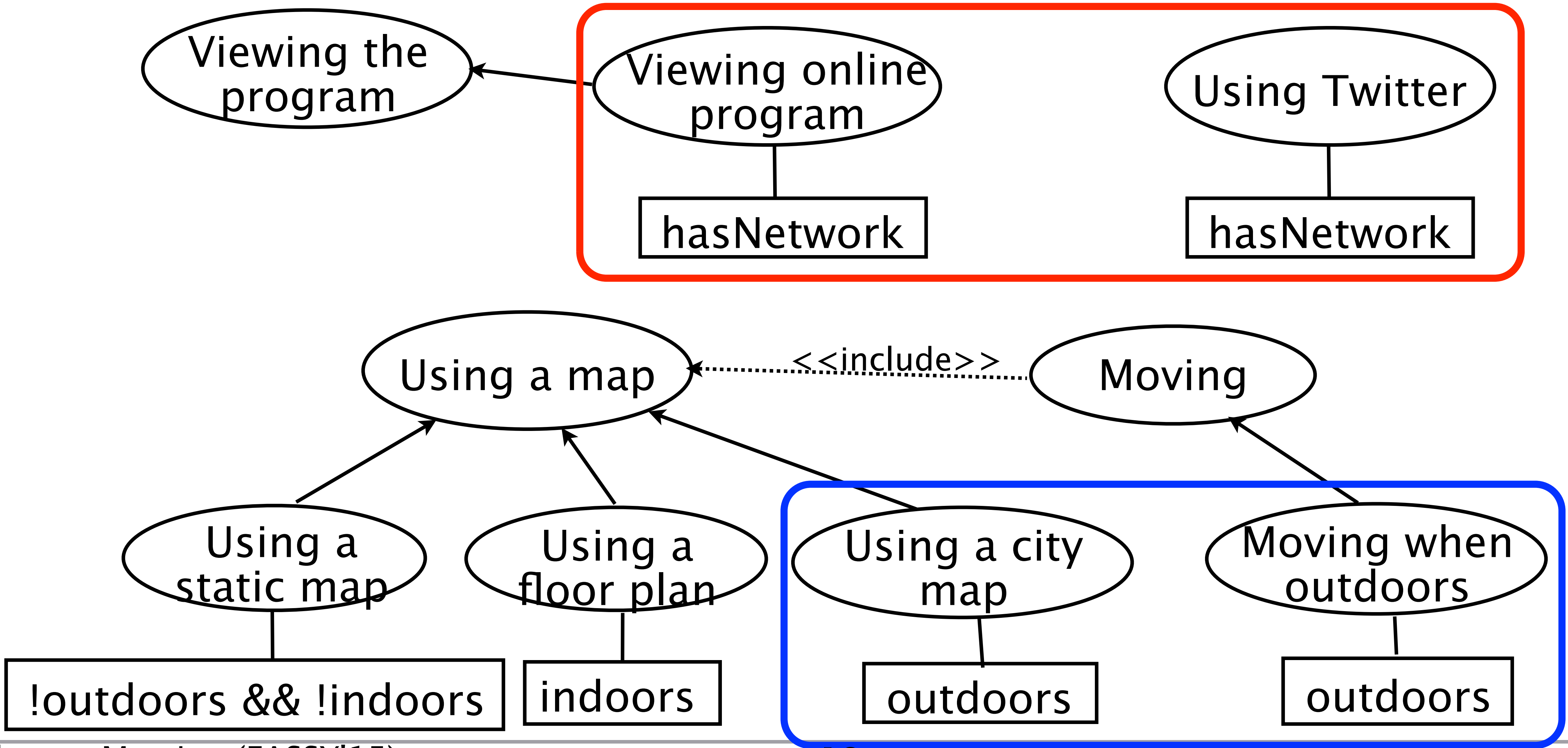
Grouping use cases

- Use cases sharing the same contexts are identified as a **layer**



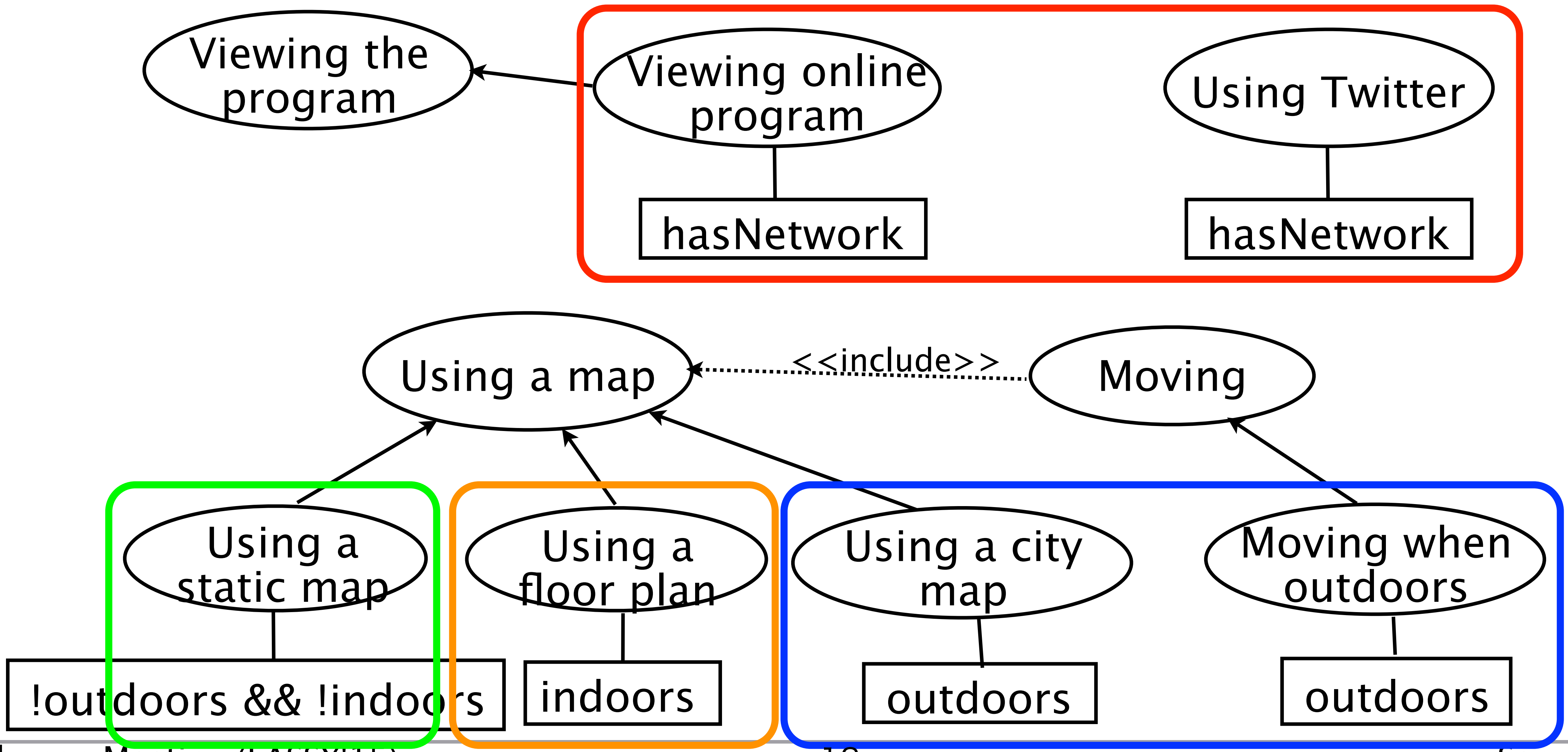
Grouping use cases

- Use cases sharing the same contexts are identified as a **layer**



Grouping use cases

- Use cases sharing the same contexts are identified as a **layer**
- Sibling use cases of a layer are also layers



Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork →

```
HasNetwork(ConnectivityManager cm) :
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork

expression using Android SDK

```
HasNetwork(ConnectivityManager cm)
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork

expression using Android SDK

```
HasNetwork(ConnectivityManager cm)
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

outdoors

Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork

```
HasNetwork(ConnectivityManager cm)
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

expression using Android SDK

outdoors $\xrightarrow{\text{decomposed as}}$ GPSAvailable && !VenueWifiAvailable

Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork

```
HasNetwork(ConnectivityManager cm)
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

expression using Android SDK

outdoors $\xrightarrow{\text{decomposed as}}$ GPSAvailable && !VenueWifiAvailable

not inside the venue

Deriving layer activation

- After designing classes and some implementation details, contexts in ServalCJ are derived...

Assuming Android SDK...

hasNetwork

```
HasNetwork(ConnectivityManager cm)
  cm.getActiveNetworkInfo().getDetailedState() ==
    NetworkInfo.DetailedState.CONNECTED
```

expression using Android SDK

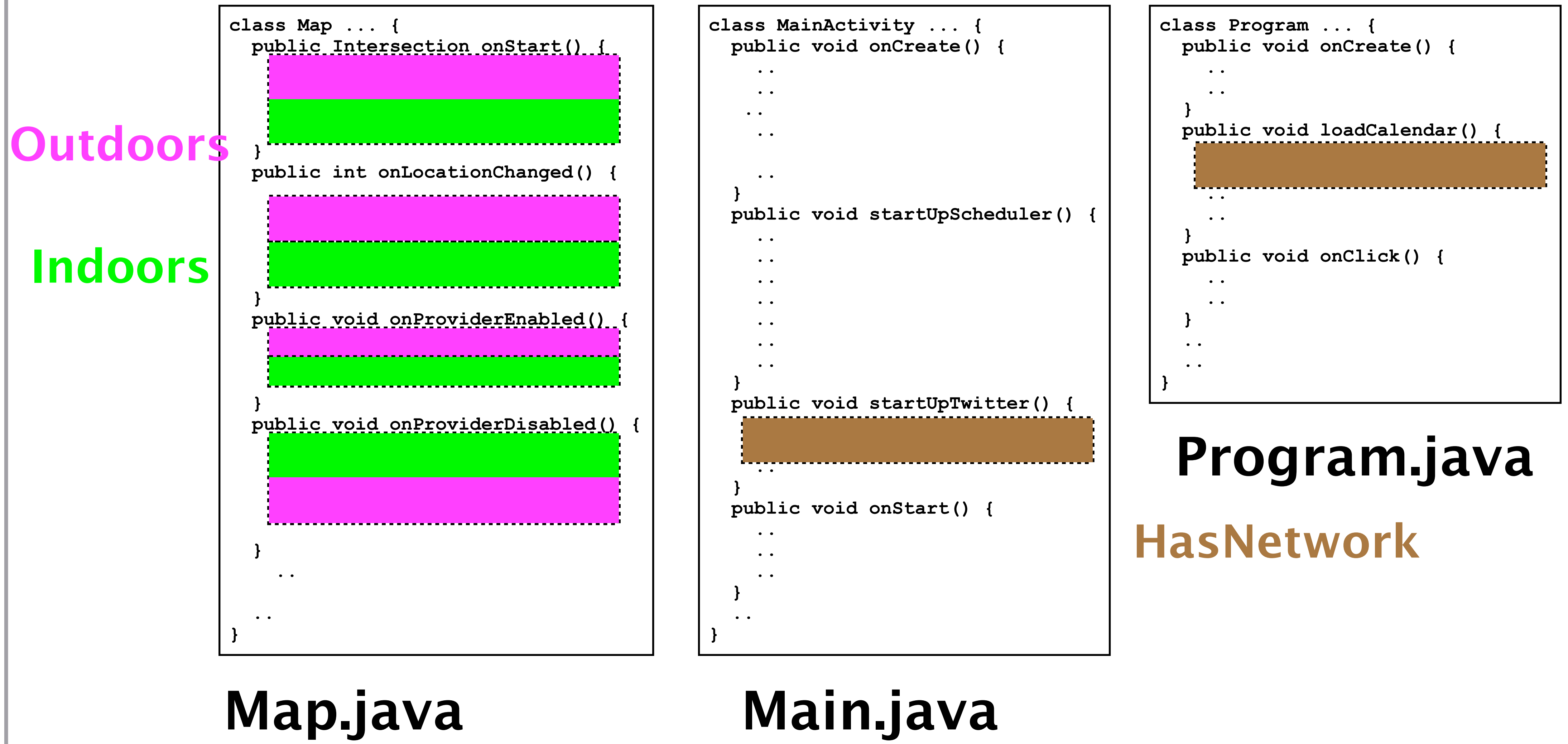
outdoors $\xrightarrow{\text{decomposed as}}$ GPSAvailable && !VenueWifiAvailable

not inside the venue

```
LocationManager.isProviderEnabled(
    LocationManager.GPS_PROVIDER) == true
```

Class structure of the system

- Context-dependent behavior crosscutting multiple classes



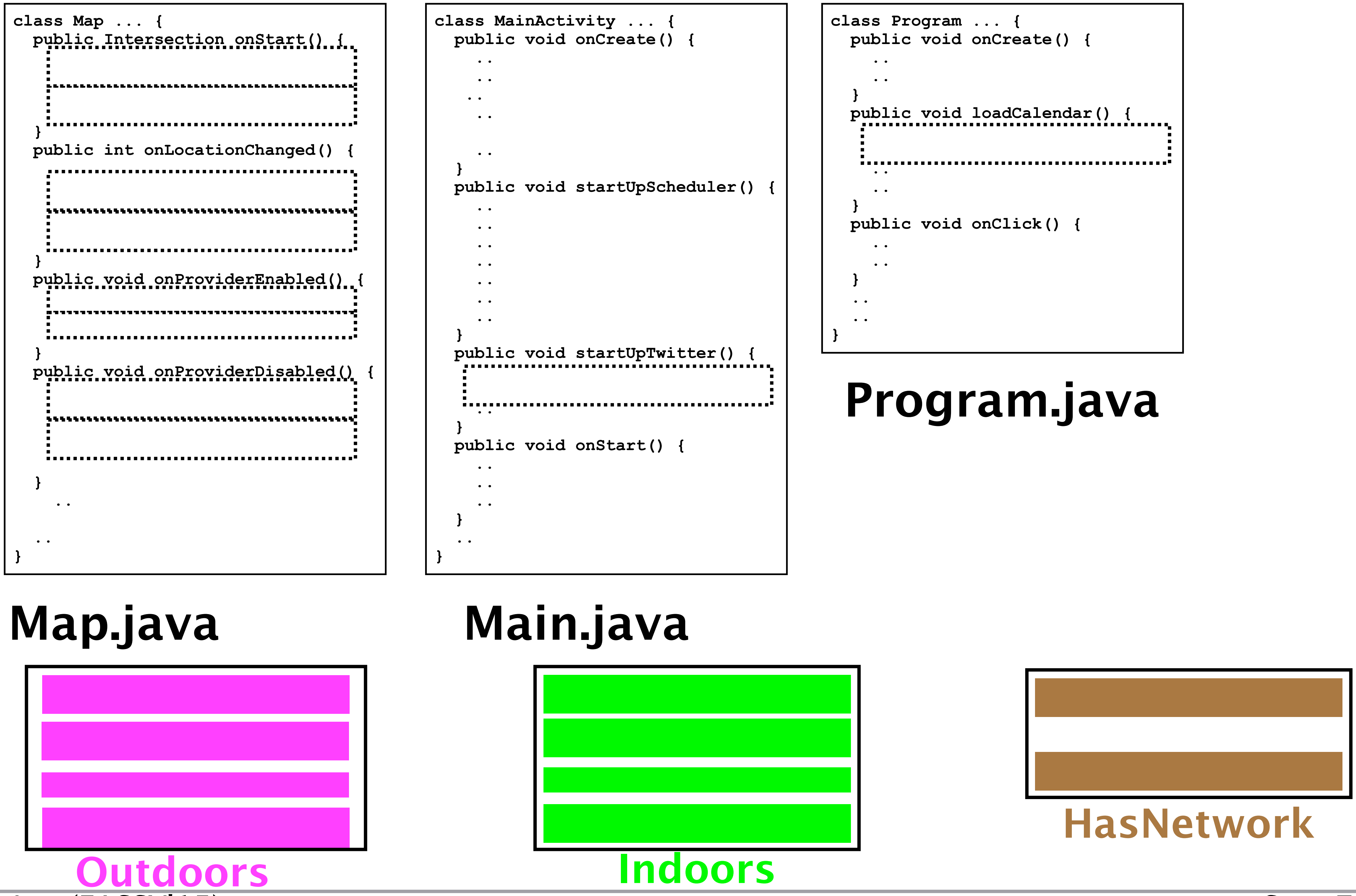
```
class Map ... {
  public Intersection onStart() {

  }
  public int onLocationChanged() {

  }
  public void onProviderEnabled() {

  }
  public void onProviderDisabled() {

  }
    ..

  ..
}
```

```
class MainActivity ... {
  public void onCreate() {
    ..
    ..
   ..
    ..

    ..
  }
  public void startUpScheduler() {
    ..
    ..
    ..
    ..
    ..
    ..
    ..
  }
  public void startUpTwitter() {

    ..
  }
  public void onStart() {
    ..
    ..
    ..
  }
  ..
}
```

```
class Program ... {
  public void onCreate() {
    ..
    ..
  }
  public void loadCalendar() {

    ..
    ..
  }
  public void onClick() {
    ..
    ..
  }
  ..
  ..
}
```

Class structure of the system

- Context-dependent behavior crosscutting multiple classes



```
class Map ... {
  public Intersection onStart() {


  }
  public int onLocationChanged() {


  }
  public void onProviderEnabled() {


  }
  public void onProviderDisabled() {


  }
    ..

  ..
}
```

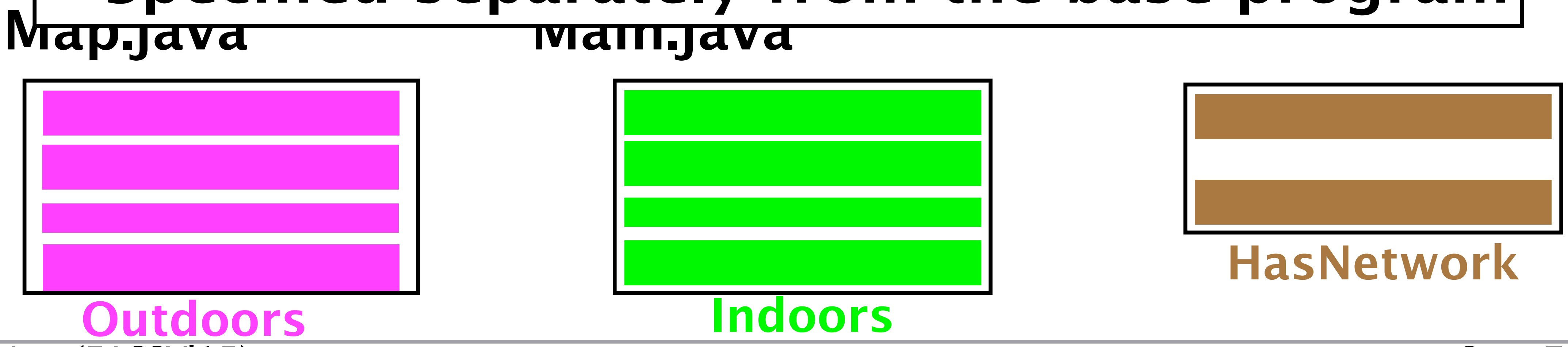
```
class MainActivity ... {
  public void onCreate() {
    ..
    ..
   ..
    ..

    ..
  }
  public void startUpScheduler() {
    ..
    ..
    ..
    ..
    ..
    ..
    ..
  }
  public void startUpTwitter() {

    ..
  }
  public void onStart() {
    ..
    ..
    ..
  }
  ..
}
```

```
class Program ... {
  public void onCreate() {
    ..
    ..
  }
  public void loadCalendar() {

    ..
    ..
  }
  public void onClick() {
    ..
    ..
  }
  ..
  ..
}
```

Control of layer activation

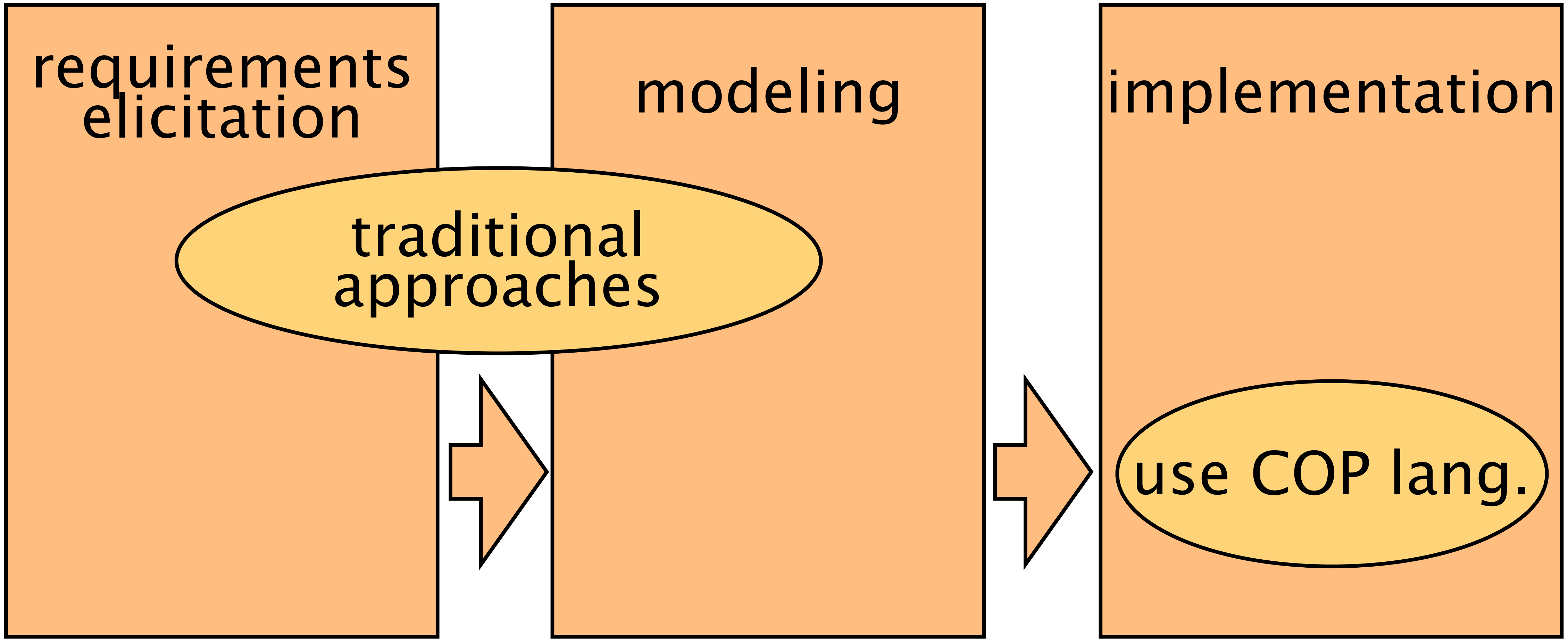
```
global contextgroup CtxCtl(ConnectivityManager cm) {
  activate HasNetwork
    if(cm.getActiveNetworkInfo().getDetailedState() ==
       NetworkInfo.DetailedState.CONNECTED);
  context GPSAvailable is
    if(LocationManager.isProviderEnabled(
       LocationManager.GPS_PROVIDER) == true);
  context VenueWifiAvailable is if(C.isWifiConnected());
  activate Outdoors
    when GPSAvailable && not(when VenueWifiAvailable);
  activate Indoors
    when VenueWifiAvaileble && if(C.isWifiPosAllowed());
}
```

- Derived in the previous slide
- Specified separately from the base program



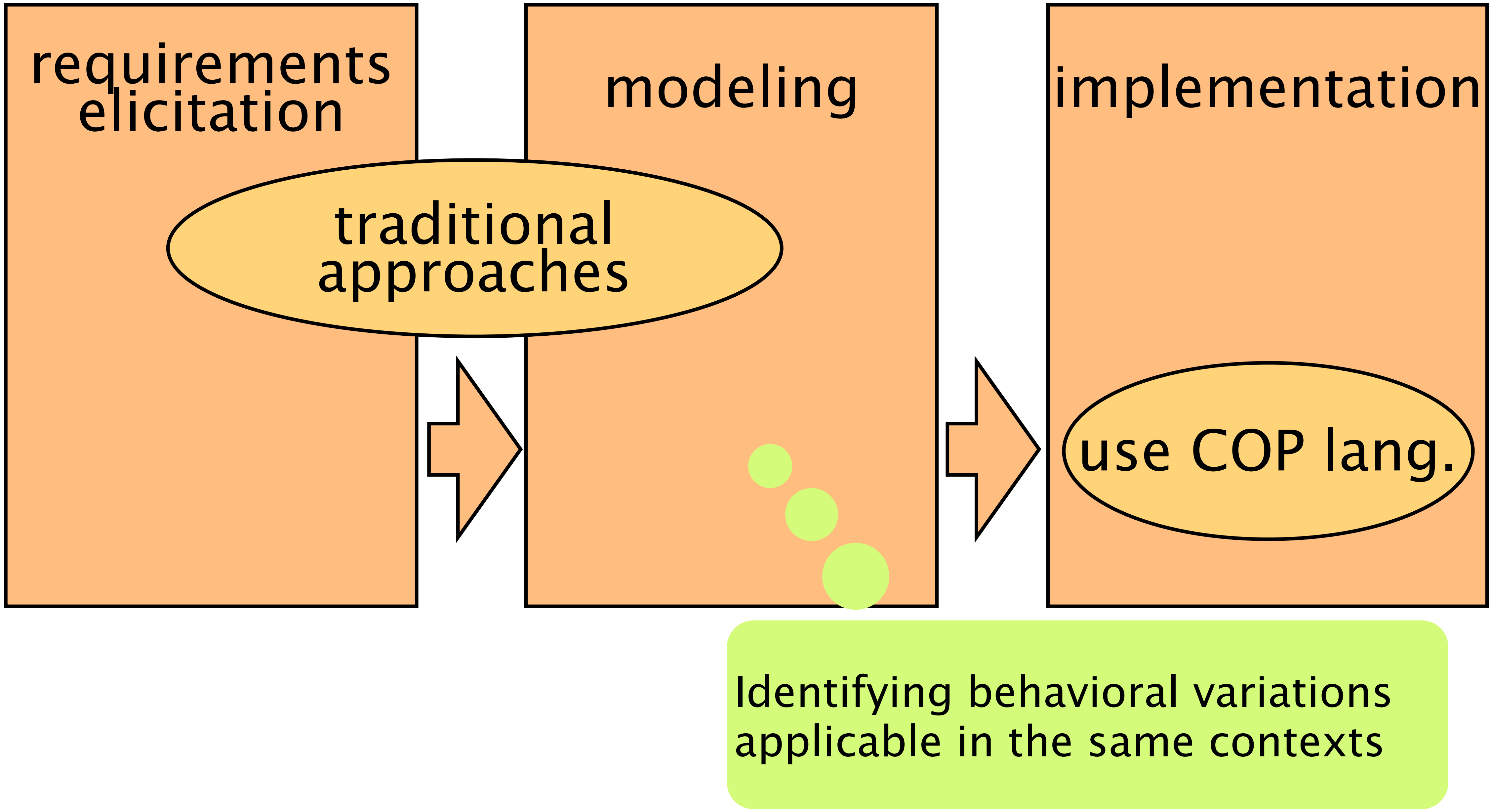
What COSE supports

- Elicitation and modeling of context-dep. behavior



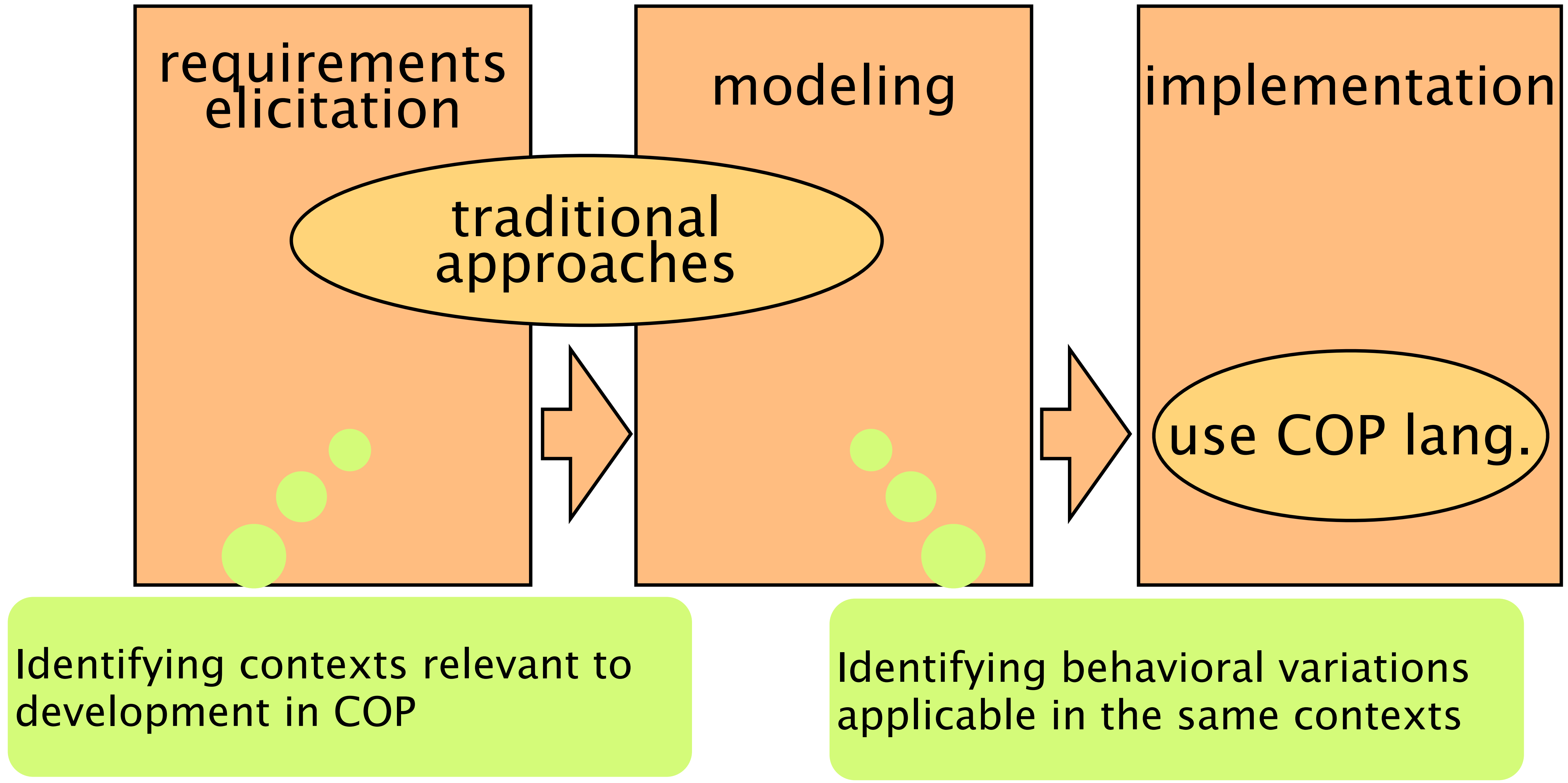
What COSE supports

- Elicitation and modeling of context-dep. behavior



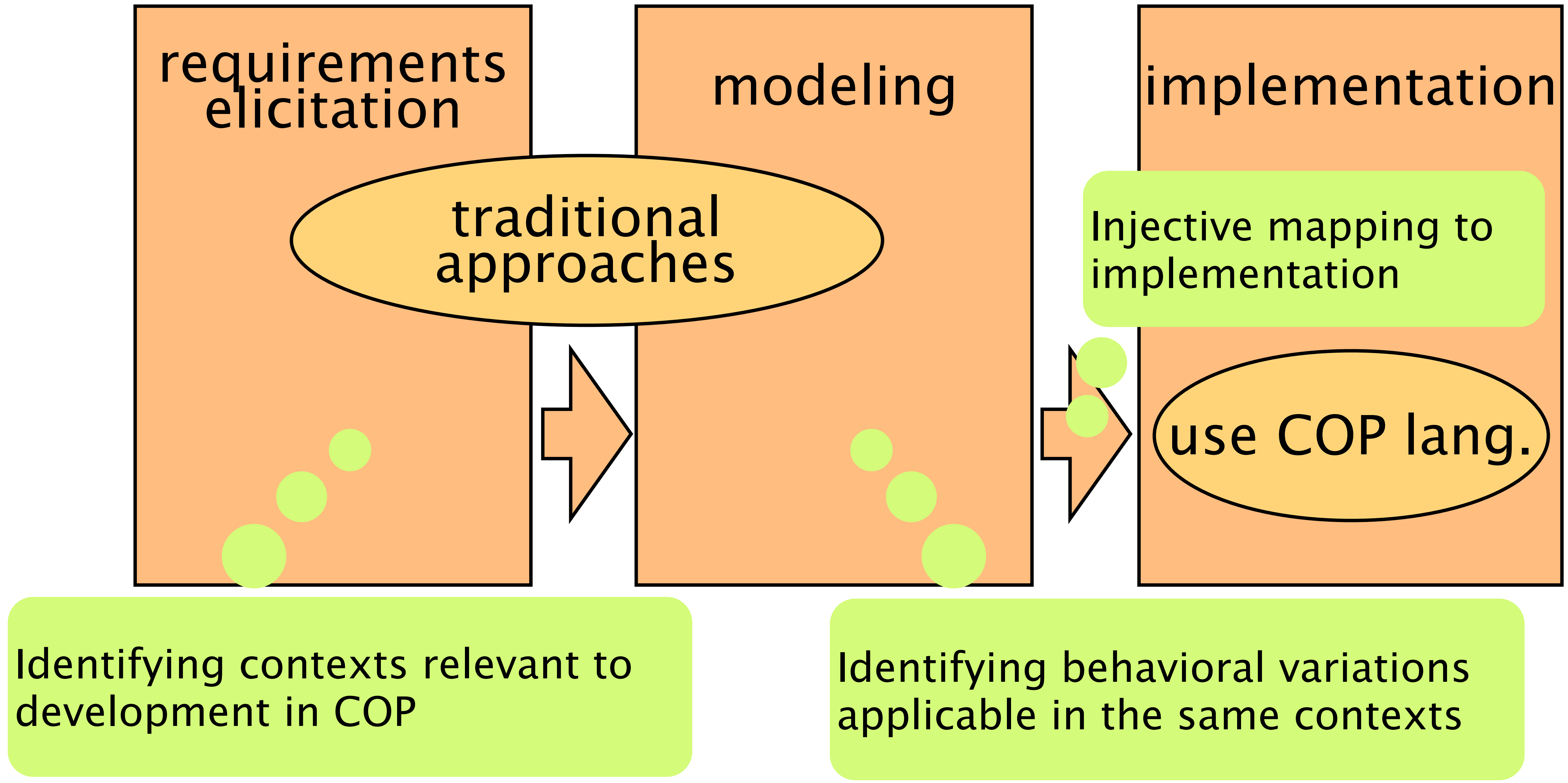
What COSE supports

- Elicitation and modeling of context-dep. behavior


requirements elicitation
modeling
implementation
traditional approaches
use COP lang.
Identifying contexts relevant to development in COP
Identifying behavioral variations applicable in the same contexts

What COSE supports

- Elicitation and modeling of context-dep. behavior


requirements elicitation
modeling
implementation
traditional approaches
Injective mapping to implementation
use COP lang.
Identifying contexts relevant to development in COP
Identifying behavioral variations applicable in the same contexts

COP for adaptive software systems —A rescue robot scenario—

- Model case: Sasago tunnel disaster in Japan (in 2012)
  - ceiling boards are collapsed
  - occurred 1700m far from the entrance

- Required services for rescue robots for such disaster
  - Recognizing contexts (location, obstacles..)
  - Supporting autonomous moving
    - Avoiding collisions with obstacles
    - Switching b/w multiple modes
      - **Flying** mode ... for avoiding large obstacles
      - **Running** mode ... for saving the energy
    - Changing b/w normal & abnormal (e.g. no signals from sensors) behavior

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

COP for adaptive software systems —A rescue robot scenario—

- Model case: Sasago tunnel disaster in Japan (in 2012)
  - ceiling boards are collapsed
  - occurred 1700m far from the entrance

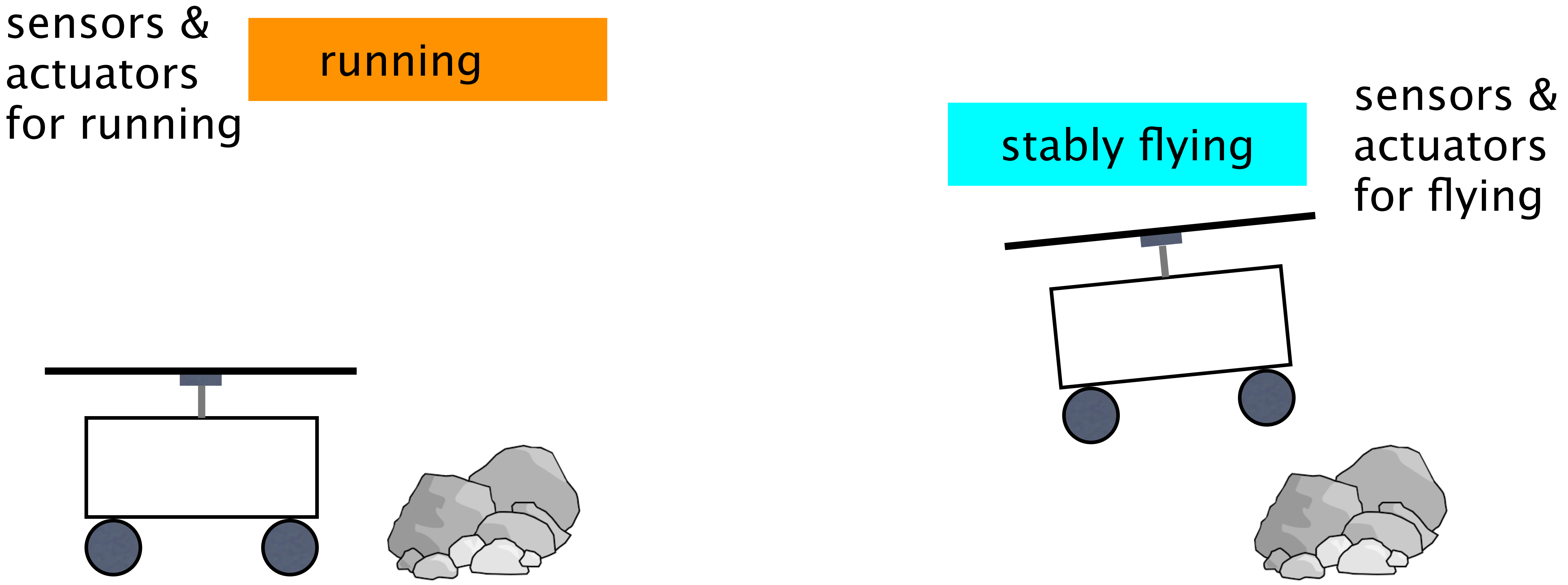
- Required services for rescue robots for such disaster
  - Recognizing contexts (location, obstacles..)
  - Supporting autonomous moving
    - Avoiding collisions with obstacles
    - Switching b/w multiple modes
      - **Flying** mode ... for avoiding large obstacles
      - **Running** mode ... for saving the energy
    - Changing b/w normal & abnormal (e.g. no signals from sensors) behavior

layers are promising

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

Pitfalls in context identification

- Can the robot switch from running to flying?

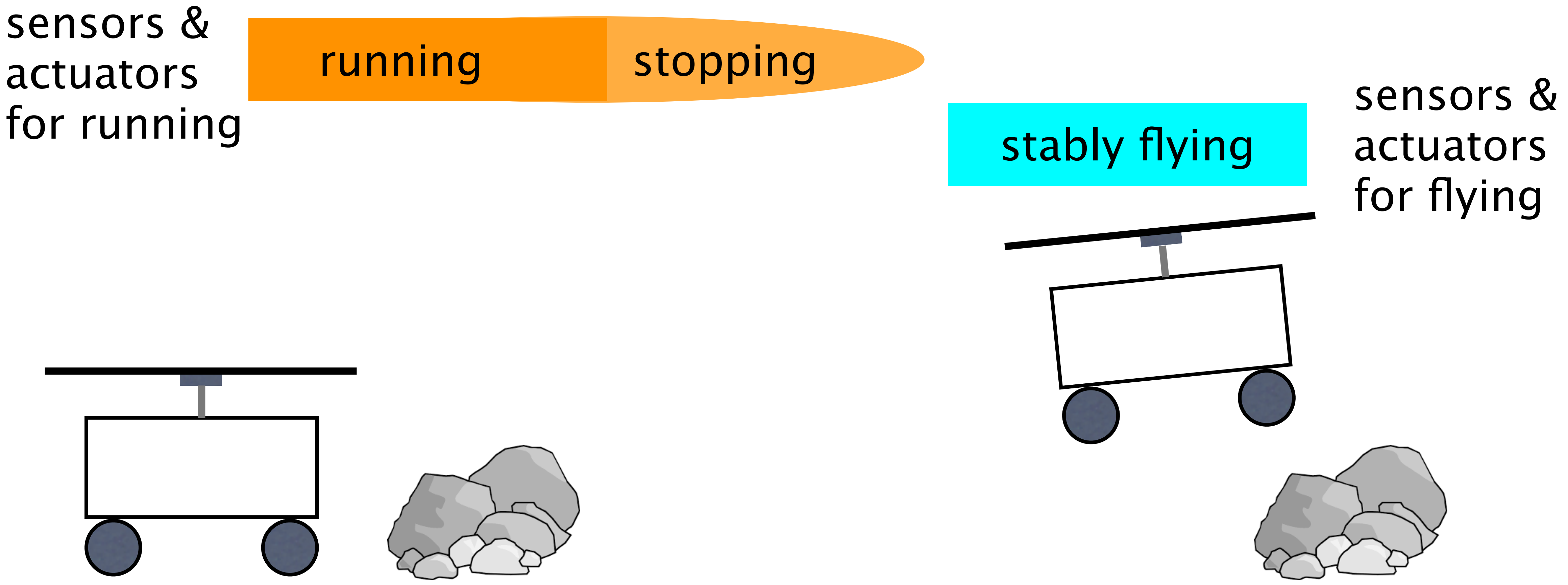


- Preemption for handling abnormal situations

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

Pitfalls in context identification

- Can the robot switch from running to flying?

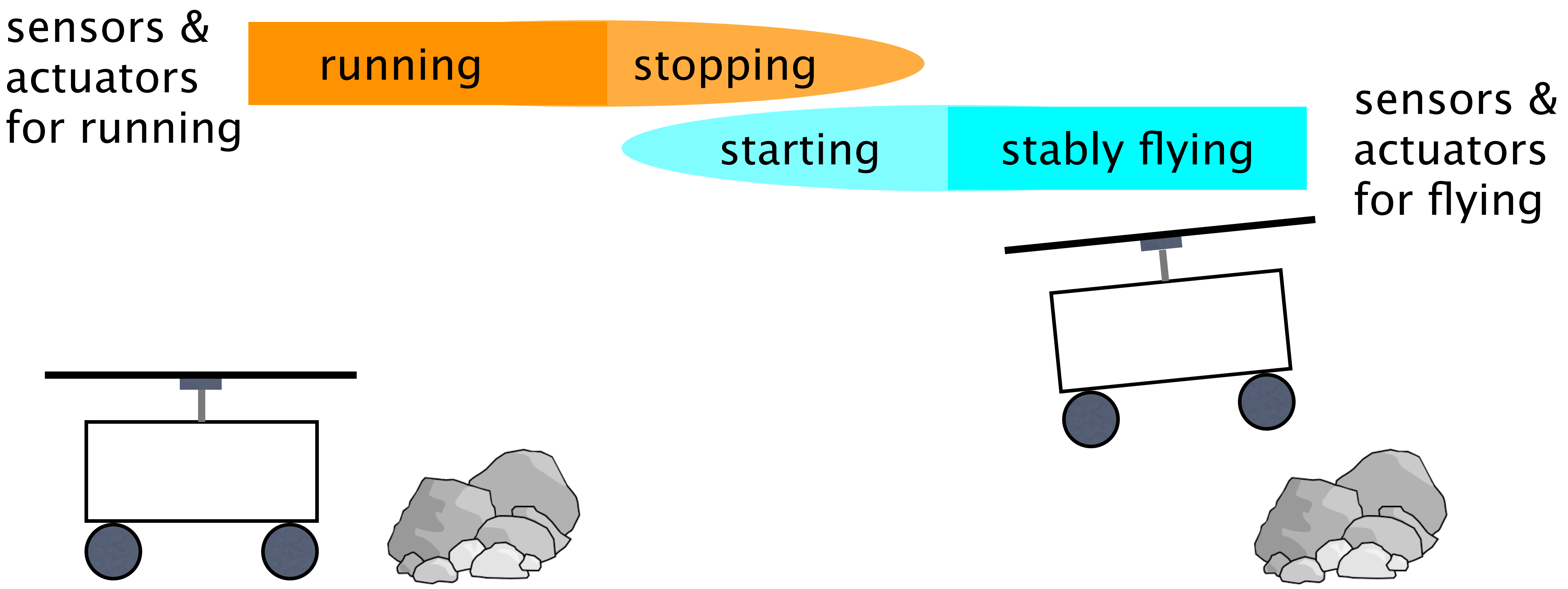


- Preemption for handling abnormal situations

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

Pitfalls in context identification

- Can the robot switch from running to flying?

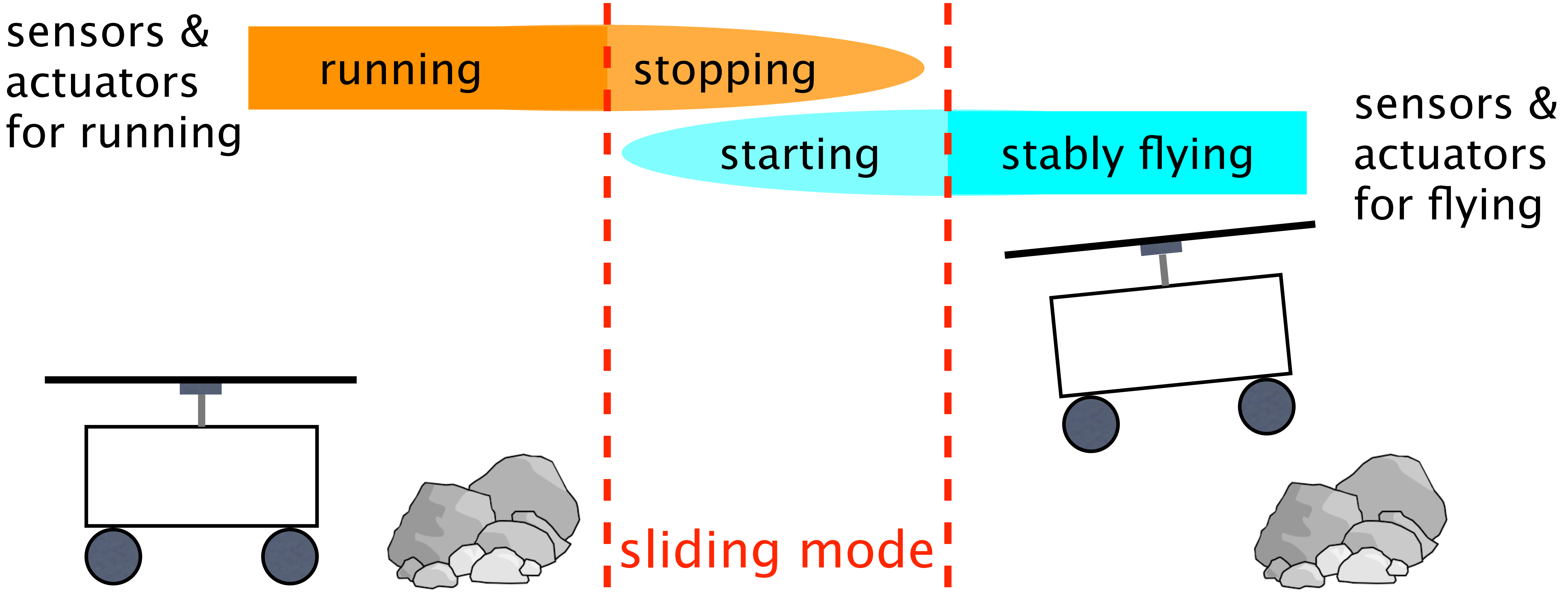


- Preemption for handling abnormal situations

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

Pitfalls in context identification

- Can the robot switch from running to flying?

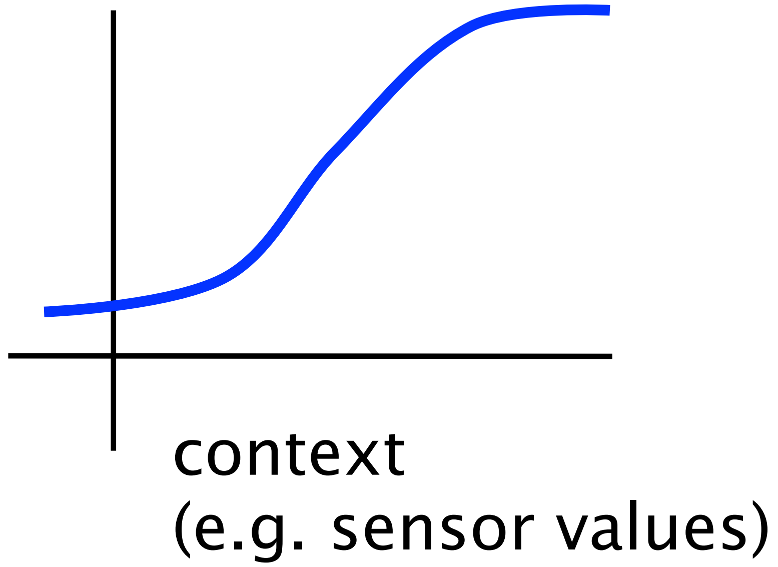


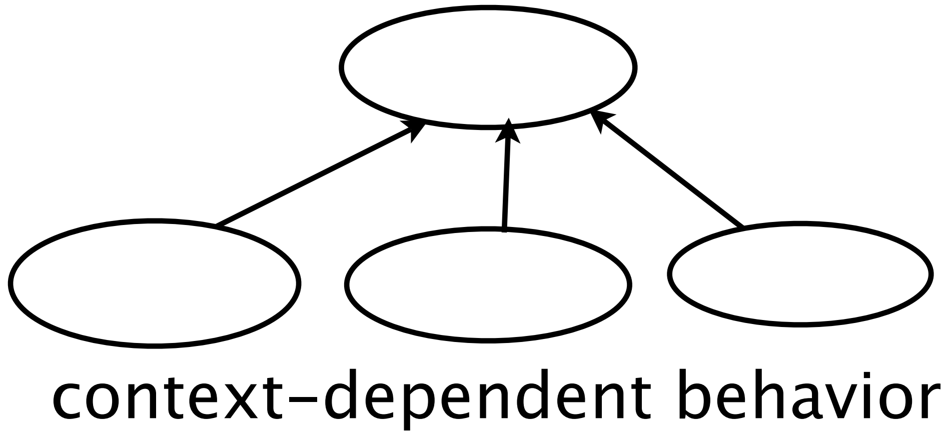
- Preemption for handling abnormal situations

H. Watanabe et al., A Study of Context-Oriented Programming for Applying to Robot Development, In COP'15.

Towards COP for EASSY

- Mapping contexts in the real world to contexts in COP



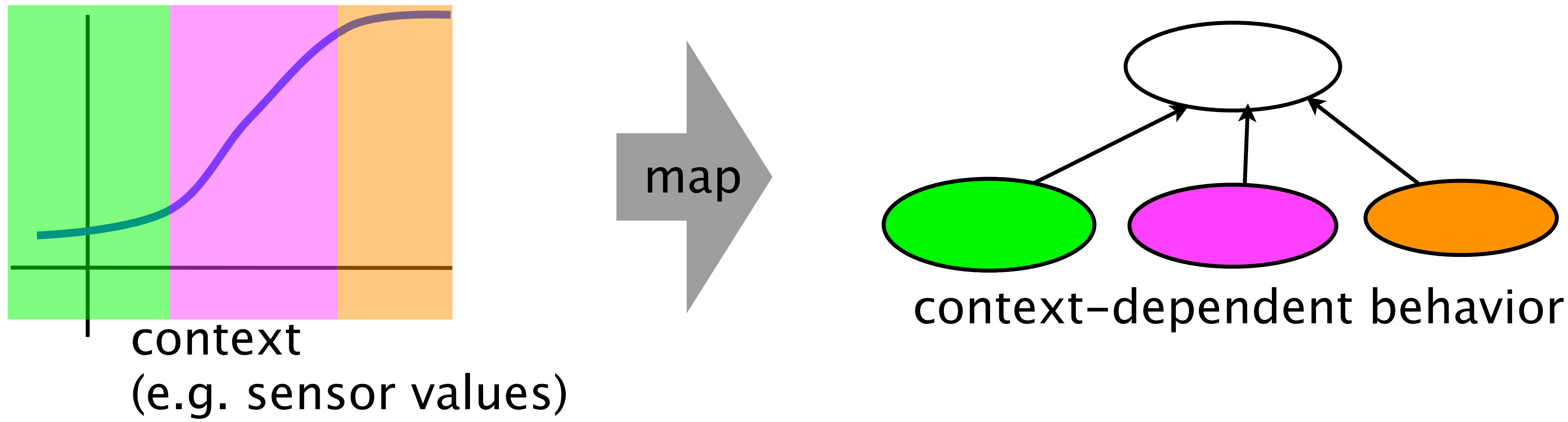


- Installing layers at runtime for unknown situation

- Prioritize layers to handle preemption

Towards COP for EASSY

- Mapping contexts in the real world to contexts in COP

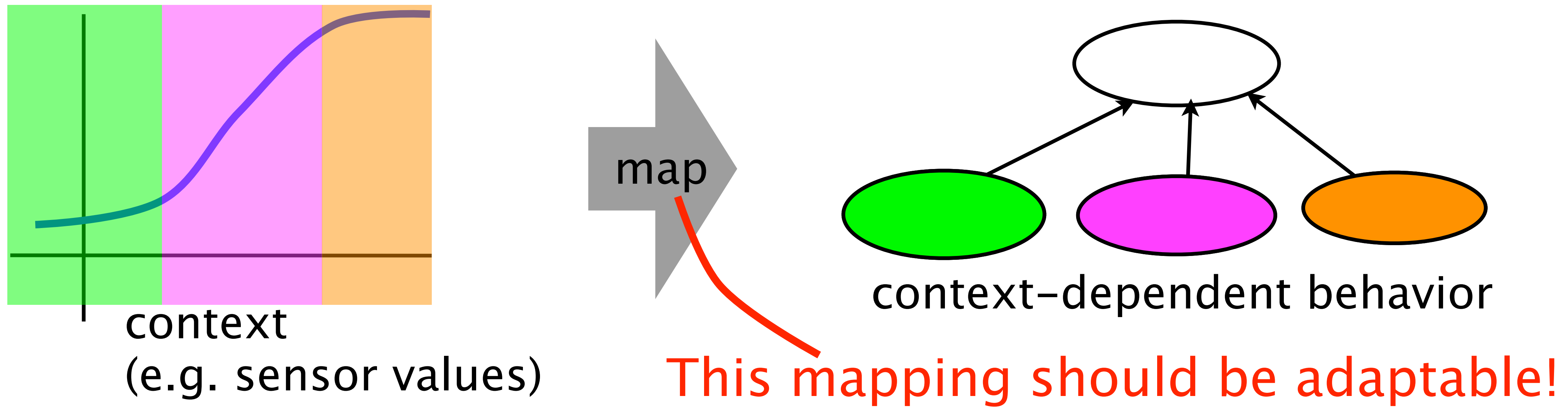


- Installing layers at runtime for unknown situation

- Prioritize layers to handle preemption

Towards COP for EASSY

- Mapping contexts in the real world to contexts in COP



- Installing layers at runtime for unknown situation

- Prioritize layers to handle preemption