The Fusemate Logic Programming System for Situational Awareness

Peter Baumgartner

Data61 | CSIRO

The Australian National University

Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor

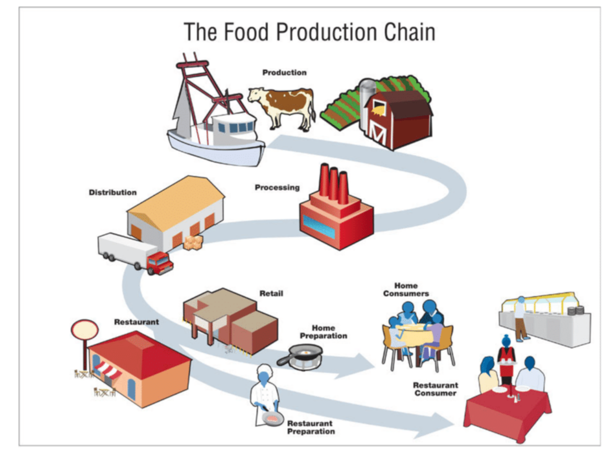
Are the operations carried out according to the schedule?

Food Supply Chain

Are goods delivered within 3 hours and stored below 25℃?

Why is the truck late?

What is the expected quality (shelf life) of the goods?



Data Cleansing

Does the database have complete, correct and relevant data?

What's the problem?

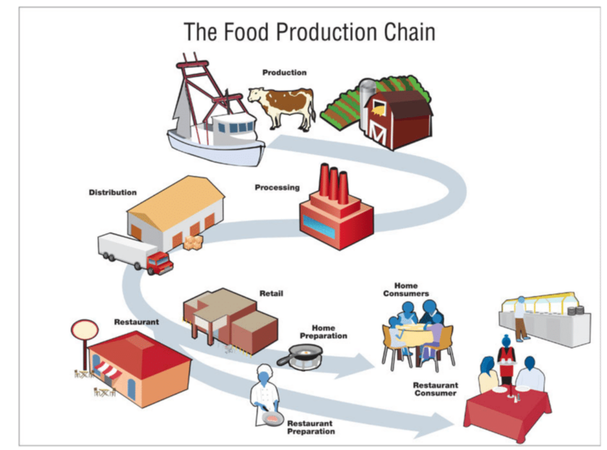
- The domain model needs to cover multiple aspects:

  Temporal/causal/structural/physical/…
- Events **happened** ≠ events **reported** (errors, incomplete, late …)
- Can only hope for **multiple** plausible explanations

Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor
Are the operations carried out according to the schedule?

Food Supply Chain
Are goods delivered within 3 hours and stored below 25℃?
Why is the truck late?
What is the expected quality (shelf life) of the goods?



Data Cleansing
Does the database have complete, correct and relevant data?

What's the problem?

- The domain model needs to cover multiple aspects:
  Temporal/causal/structural/physical/…
- Events **happened** ≠ events **reported** (errors, incomplete, late …)
- Can only hope for **multiple** plausible explanations

← Logic program
+ ontologies/event calculus

Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor

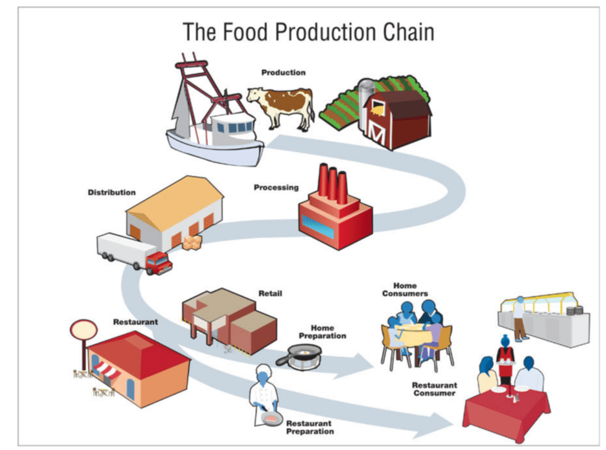
Are the operations carried out according to the schedule?

Food Supply Chain

Are goods delivered within 3 hours and stored below 25°C?

Why is the truck late?

What is the expected quality (shelf life) of the goods?



Data Cleansing

Does the database have complete, correct and relevant data?

What's the problem?

- The domain model needs to cover multiple aspects:
  Temporal/causal/structural/physical/… ← Logic program + ontologies/event calculus
- Events **happened** ≠ events **reported** (errors, incomplete, late …) ← Belief revision
- Can only hope for **multiple** plausible explanations

Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor

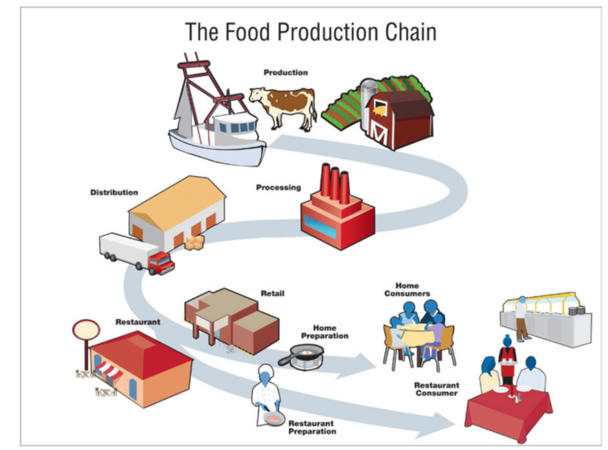
Are the operations carried out according to the schedule?

Food Supply Chain

Are goods delivered within 3 hours and stored below 25°C?

Why is the truck late?

What is the expected quality (shelf life) of the goods?



Data Cleansing

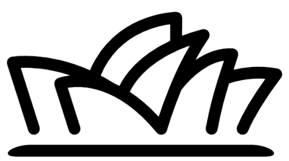
Does the database have complete, correct and relevant data?

What's the problem?

- The domain model needs to cover multiple aspects: Temporal/causal/structural/physical/… ← Logic program + ontologies/event calculus
- Events **happened** ≠ events **reported** (errors, incomplete, late …) ← Belief revision
- Can only hope for **multiple** plausible explanations ← Models

Example

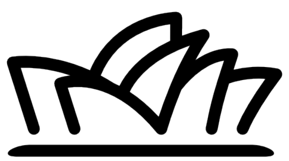
Observation: truck is in Sydney at the warehouse



T

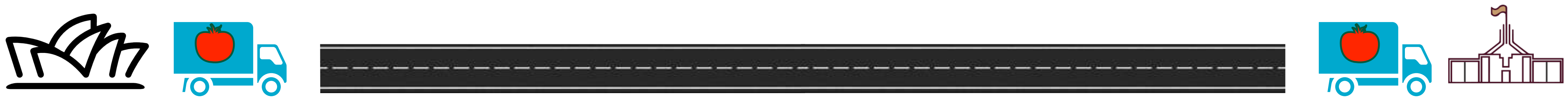
Example 

Observation: truck is in Sydney at the warehouse

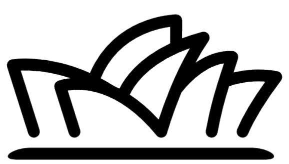


T

Example



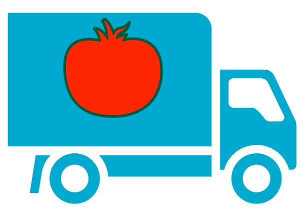
Observation: tomatoes are loaded

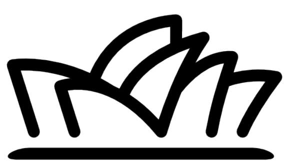


T

Example

Observation: tomatoes are loaded

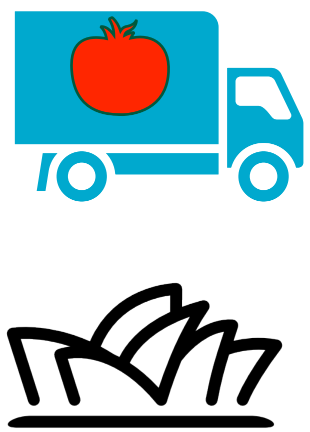




T

Example 

Assumption as per schedule: truck is on the road



T

Example

Assumption as per schedule: truck is on the road

T **T+1**

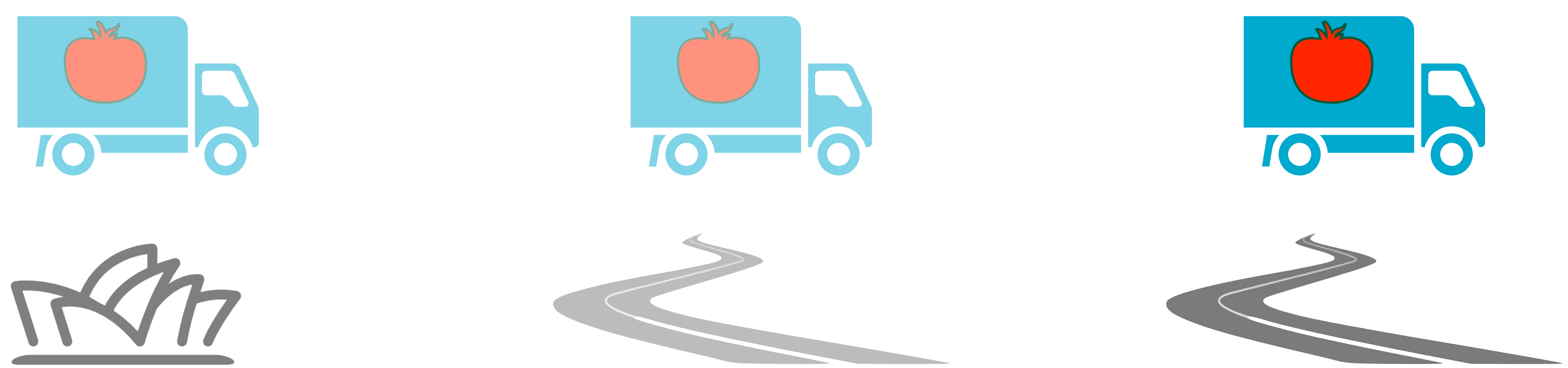
Example

Report: truck is on the road

T **T+1**

Example

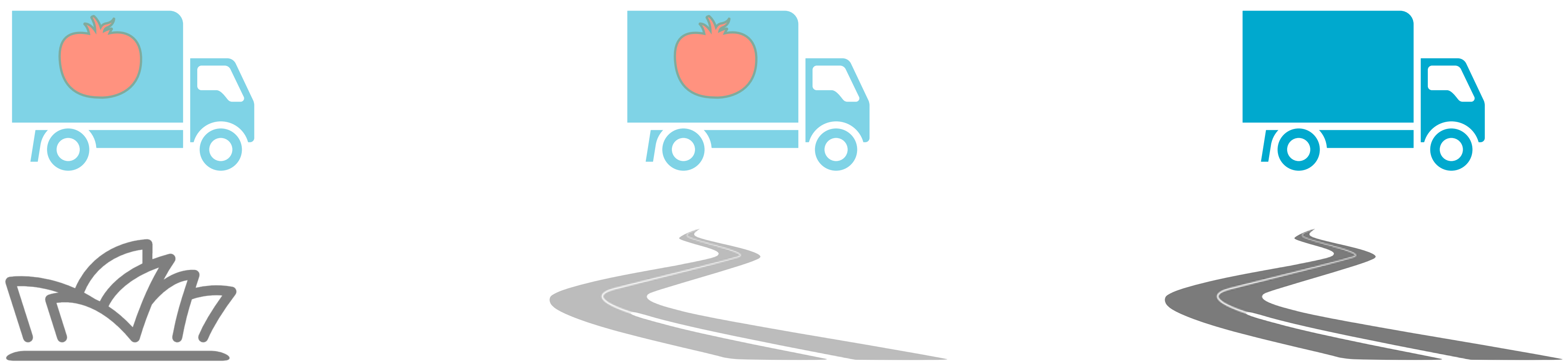
Report: truck is on the road



T **T+1** **T+2**

Example

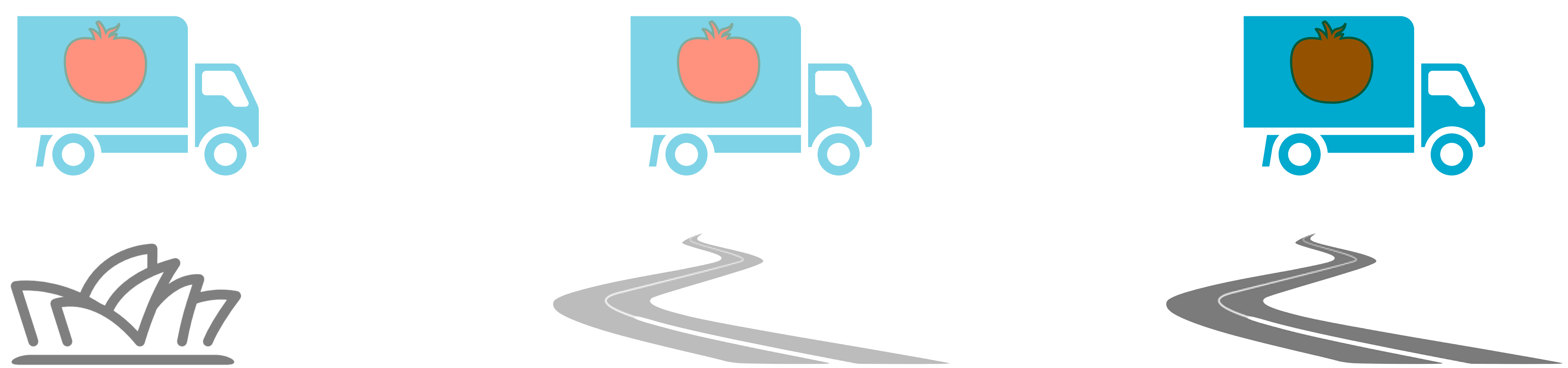
Conclusion: truck is on the road for too long - tomatoes are no longer fresh



T **T+1** **T+2**

Example 

Conclusion: truck is on the road for too long - tomatoes are no longer fresh



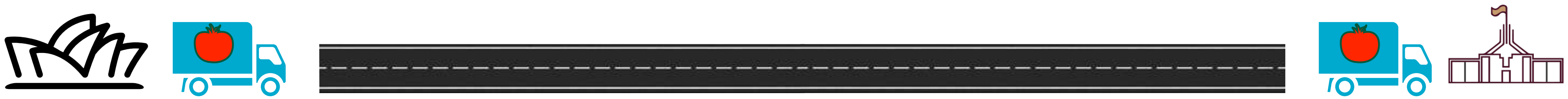
T **T+1** **T+2**

Example

Report: actually, at T+1 truck was still in Sydney warehouse

T **T+1** **T+2**

Example

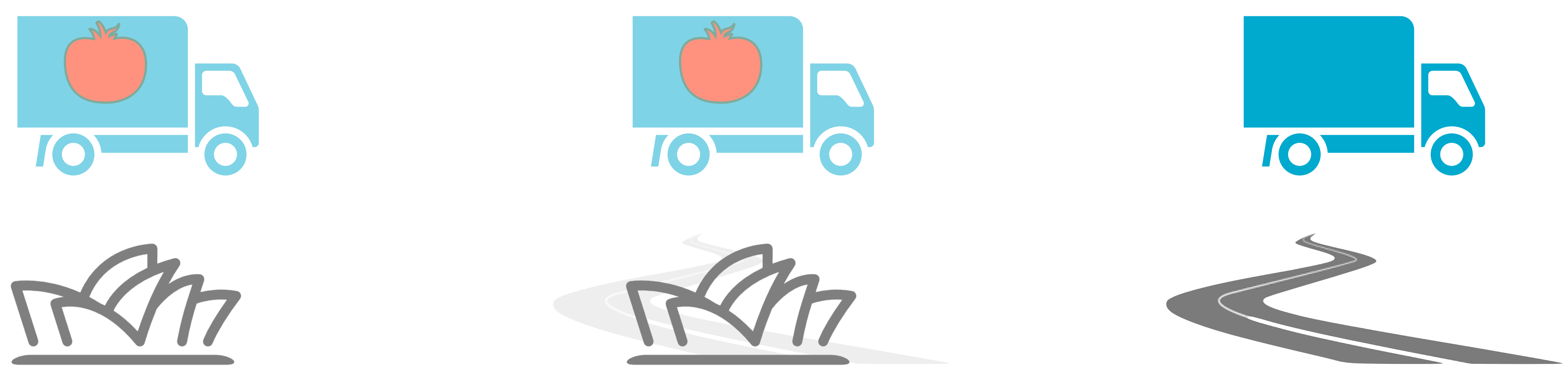


Report: actually, at T+1 truck was still in Sydney warehouse

T **T+1** **T+2**

Example

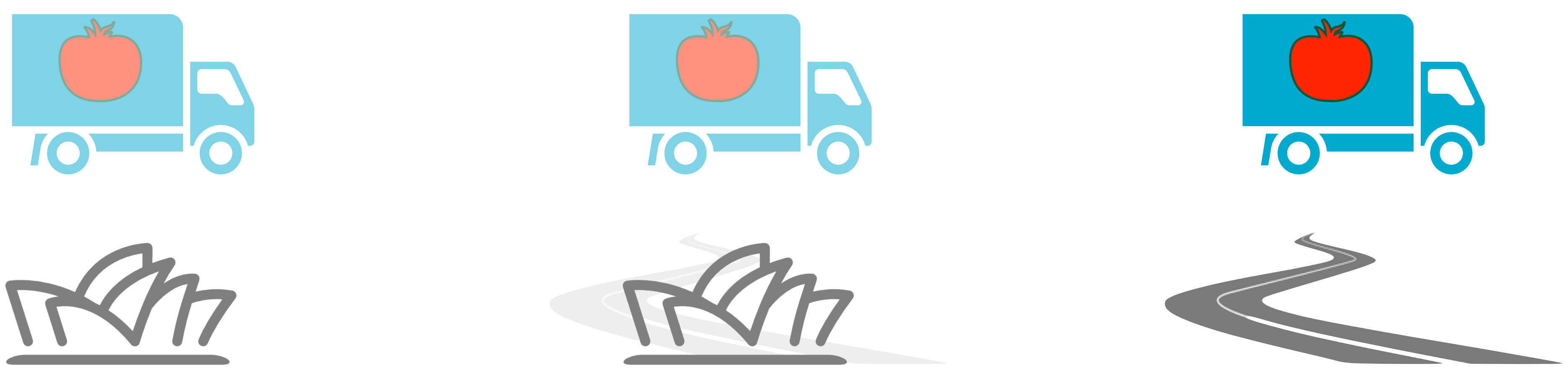
Conclusion: tomatoes are still fresh at T+2



T **T+1** **T+2**

Example

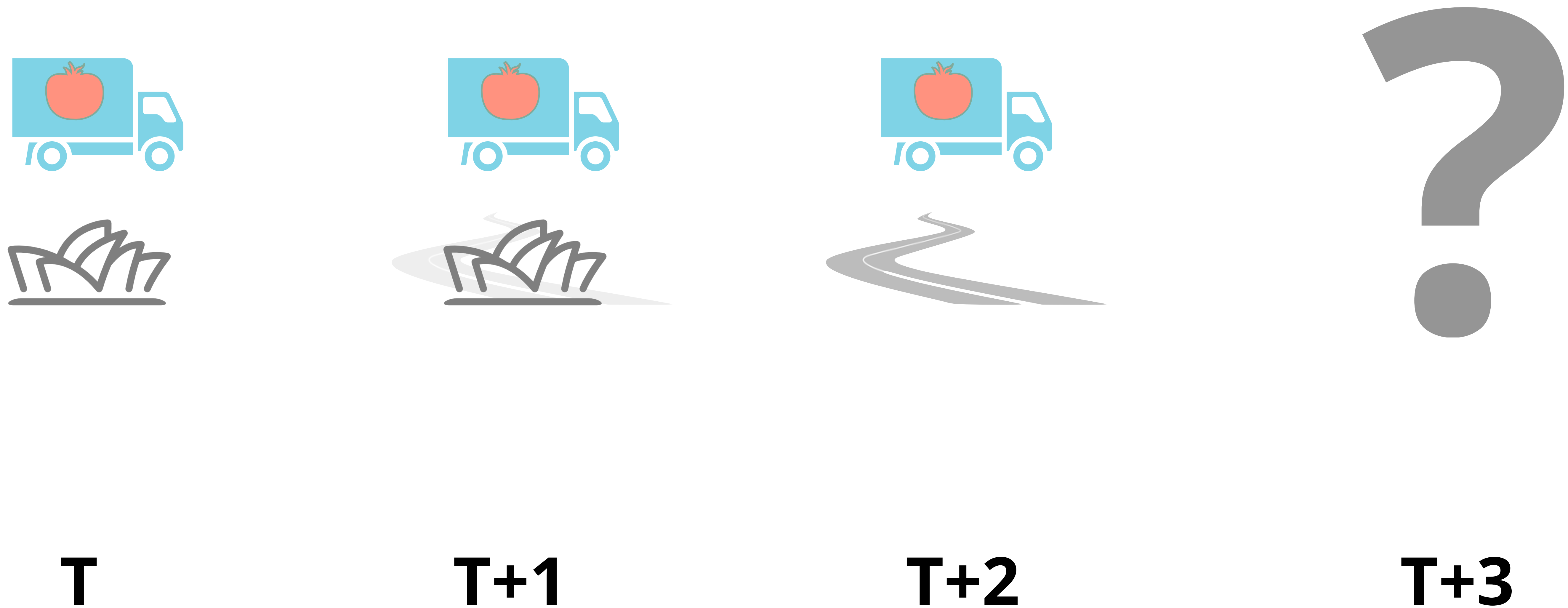
Conclusion: tomatoes are still fresh at T+2



T **T+1** **T+2**

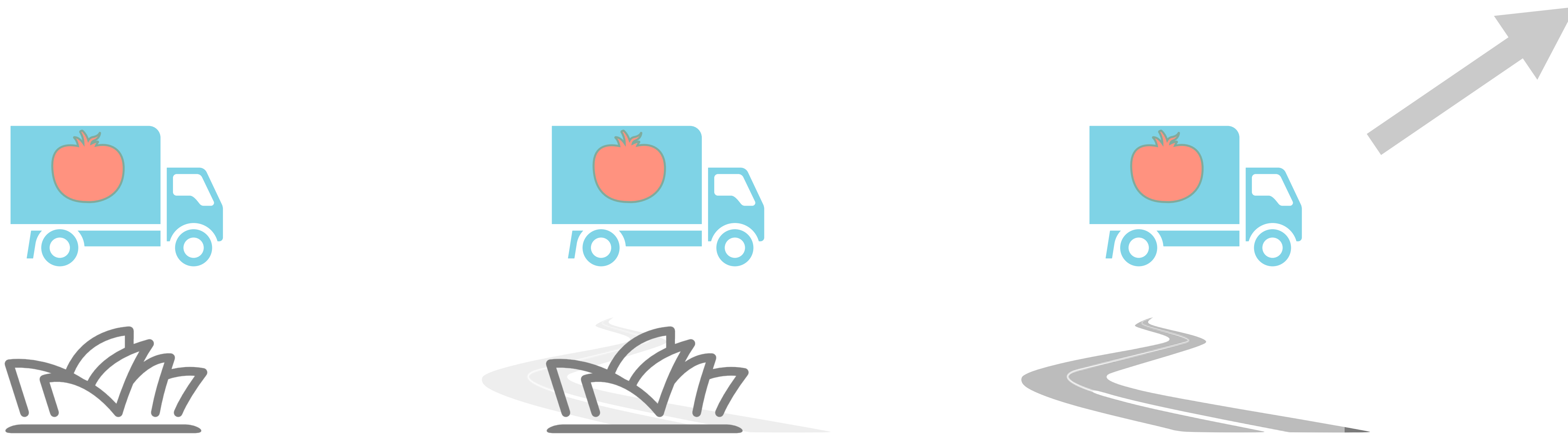
Example 

No information at T+3



Example

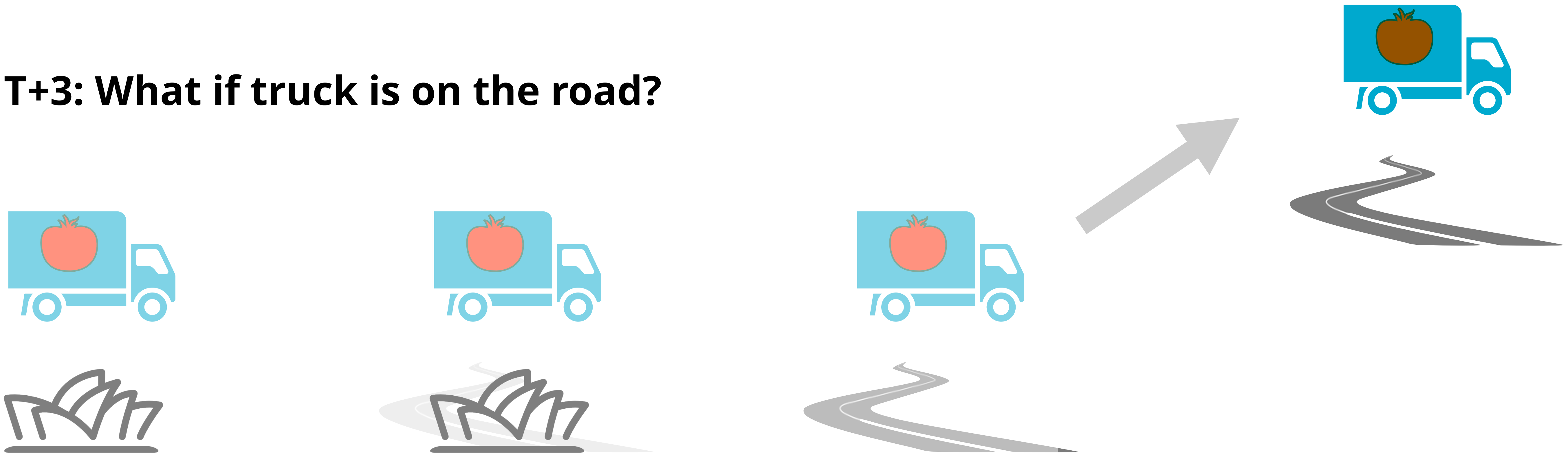
T+3: What if truck is on the road?



T **T+1** **T+2** **T+3**

Example

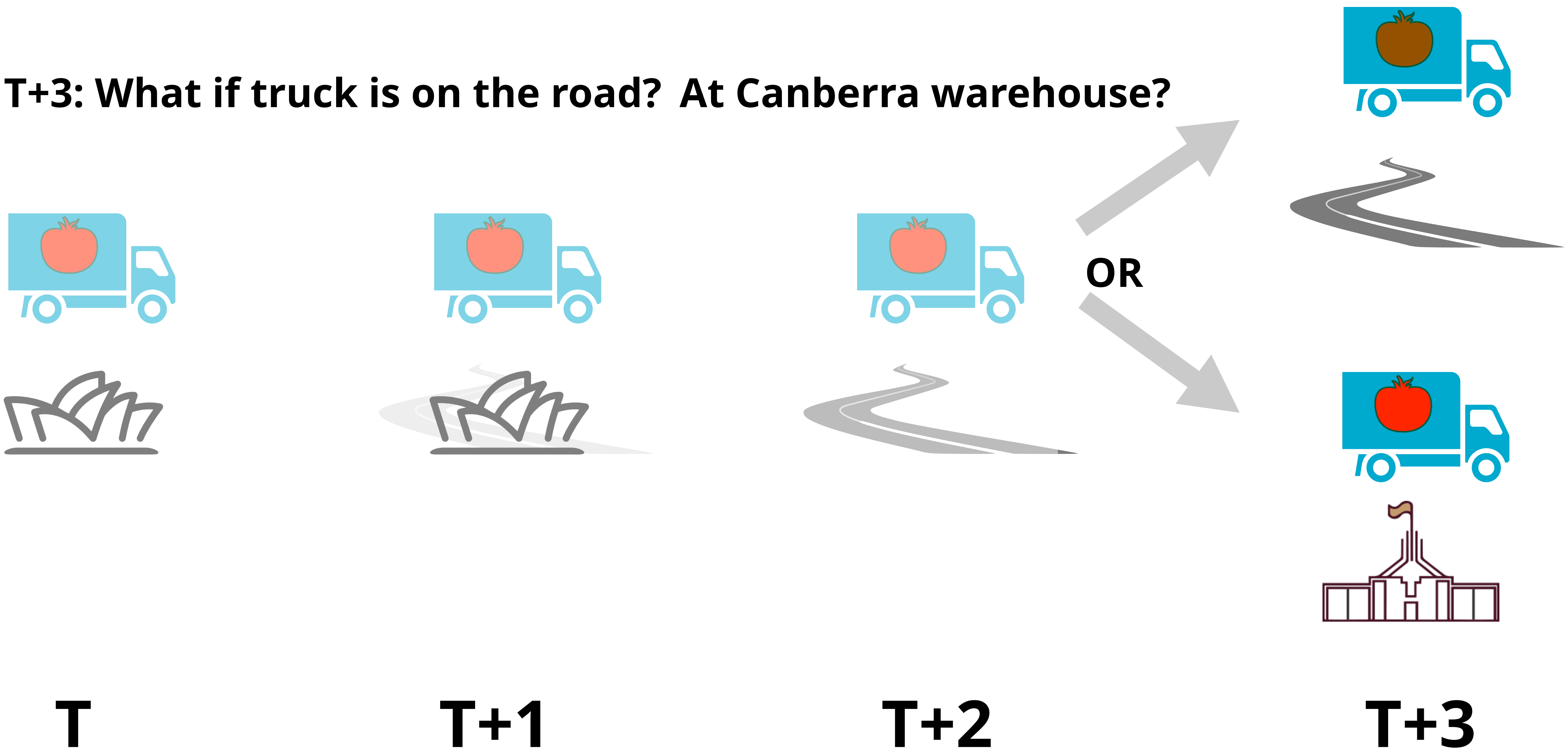
T+3: What if truck is on the road?



T **T+1** **T+2** **T+3**

Example

T+3: What if truck is on the road? At Canberra warehouse?



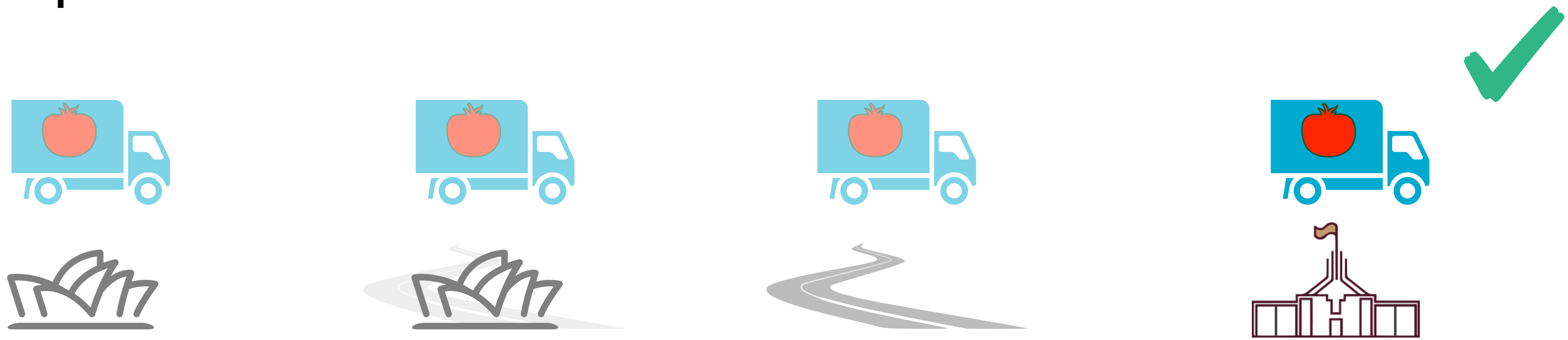
Example

Report: truck at Canberra warehouse

T **T+1** **T+2** **T+3**

Example

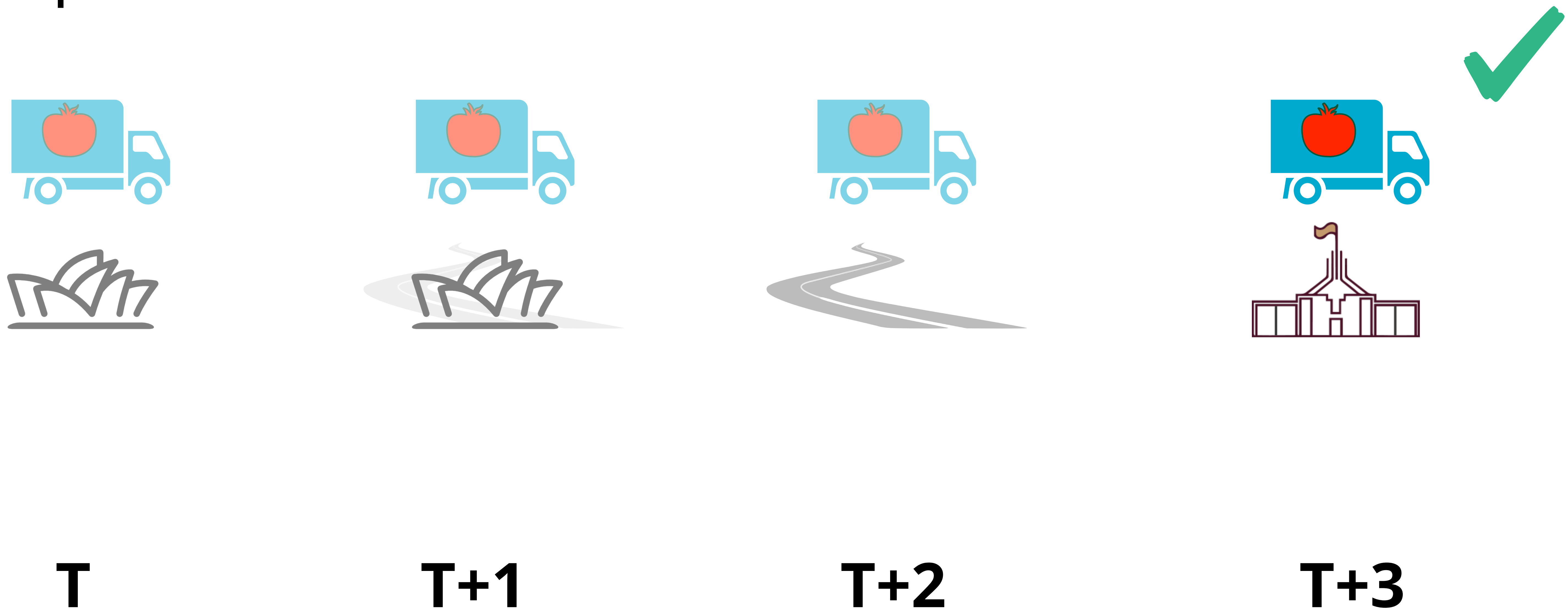
Report: truck at Canberra warehouse



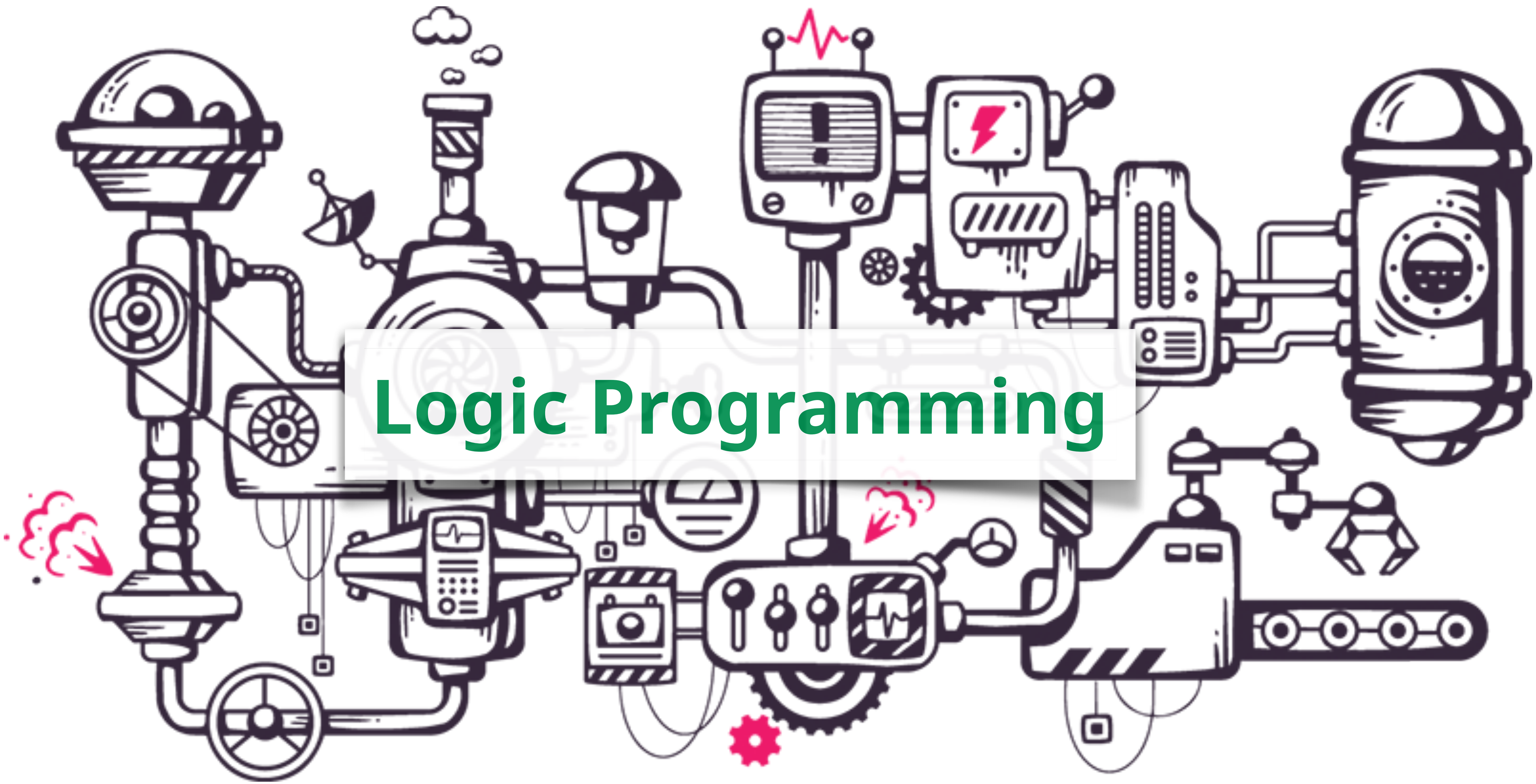
T **T+1** **T+2** **T+3**

Example

Report: truck at Canberra warehouse



→ We use logic programming



Logic Programming

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge

Chained by inference engine

Logic Programming

Tom is thirsty

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge

Chained by inference engine

Logic Programming

Tom is thirsty

Tom is a cat

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge

Chained by inference engine

Logic Programming

Tom is thirsty

Tom is a cat

Cats drink milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Tom is thirsty

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

Tom is thirsty

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge
Coles sells milk

Logic Programs

If-then rules `drinks(x, Milk) :- cat(x)` if cat(x) then drinks(x, Milk)

Logic Programming

Tom is thirsty

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge

Chained by inference engine

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge
Coles sells milk

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Facts

```
cat(Tom)
inFridge(5, Milk)
```

Logic Programming

Tom is thirsty

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge

Chained by inference engine

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Facts

```
cat(Tom)
inFridge(5, Milk)
```

Default negation

```
inFridge(time, Milk) :- not inBowl(time, Milk)
```

Logic Programming

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge
Coles sells milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Facts

```
cat(Tom)
inFridge(5, Milk)
```

"innocent :- not guilty"

Default negation

```
inFridge(time, Milk) :- not inBowl(time, Milk)
```

Logic Programming

Tom is thirsty

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Facts

```
cat(Tom)
inFridge(5, Milk)
```

"innocent :- not guilty"

Default negation

```
inFridge(time, Milk) :- not inBowl(time, Milk)
```

Disjunctions

```
drinks(x, Milk) or drinks(x, Water) :- cat(x), thirsty(x)
```

Logic Programming

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge
Coles sells milk

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules	`drinks(x, Milk) :- cat(x)` `inBowl(time+1, Milk) :- inFridge(time, Milk)`	if cat(x) then drinks(x, Milk)
Facts	`cat(Tom)` `inFridge(5, Milk)`	
Default negation	`inFridge(time, Milk) :-` **`not`** `inBowl(time, Milk)`	"innocent :- not guilty"
Disjunctions	`drinks(x, Milk)` **`or`** `drinks(x, Water) :- cat(x), thirsty(x)`	
Integrity constraints	**`fail`** `:- cat(x), mouse(x)`	

Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Tom is thirsty
Tom is a cat
Cats drink milk
Milk is in the fridge
Coles sells milk

Logic Programs

If-then rules

```
drinks(x, Milk) :- cat(x)
inBowl(time+1, Milk) :- inFridge(time, Milk)
```

if cat(x) then drinks(x, Milk)

Facts

```
cat(Tom)
inFridge(5, Milk)
```

Default negation

```
inFridge(time, Milk) :- not inBowl(time, Milk)
```

"innocent :- not guilty"

Disjunctions

```
drinks(x, Milk) or drinks(x, Water) :- cat(x), thirsty(x)
```

Integrity constraints

```
fail :- cat(x), mouse(x)
```

Purpose

Query answering *(who drinks milk?)*, planning *(get Tom some milk)*, abduction *(why did we go to Coles?)*, **model computation** *(what do we know about Tom?)*

Logic Programming

Prolog - “top down query answering”

Answer Set Programming - “model computation”

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)
```

Answer Set Programming - “model computation”

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

Answer Set Programming - “model computation”

Logic Programming

Prolog - "top down query answering"

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)  [X,Y] ++ L = [X,Y|L] …
```

Answer Set Programming - "model computation"

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)
r(X,Y) :- r(Y,X)
r(X,Z) :- r(X,Y), r(Y,Z)
```

```
r(a,b)
r(c,b)
```

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)                          r(a,b)
r(X,Y) :- r(Y,X)                r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)

a :- not a
```

Logic Programming

Prolog - "top down query answering"

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - "model computation"

```
r(X,X)                          r(a,b)
r(X,Y) :- r(Y,X)                r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)

a :- not a
```

No model

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)                          r(a,b)
r(X,Y) :- r(Y,X)                r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

a :- **not** a

No model

a :- **not** b

b :- **not** a

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)                              r(a,b)
r(X,Y) :- r(Y,X)                    r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

```
a :- not a
```

No model

```
a :- not b
b :- not a
```

Model 1: {a}
Model 2: {b}

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)                          r(a,b)
r(X,Y) :- r(Y,X)                r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

`a :- **not** a` — No model

```
a :- not b
b :- not a
```

Model 1: {a}
Model 2: {b}

`unhappy(now) :- not win(now+1)`

Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

Answer Set Programming - “model computation”

```
r(X,X)                          r(a,b)
r(X,Y) :- r(Y,X)                r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

a :- **not** a

No model

a :- **not** b
b :- **not** a

Model 1: {a}
Model 2: {b}

unhappy(now) :- **not** win(now+1)

	$\{\}$	$\{\mathbf{w}\}$	$\{\text{not}_{\mathbf{s}}\}$	$\{\text{not}_{\mathbf{s}}, \mathbf{w}\}$	$\{\text{not}\}$	$\{\text{not}, \mathbf{w}\}$
$\{\}$	P	P	P	P	NP	Δ_2^P
$\{\mathtt{v_h}\}$	NP	Δ_2^P	NP	Δ_2^P	NP	Δ_2^P
$\{\mathtt{v}\}$	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P

Logic Programming

Prolog - "top down query answering"

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

"More operational"

General purpose PL

Unification/DFBS

Answer Set Programming - "model computation"

```
r(X,X)                        r(a,b)
r(X,Y) :- r(Y,X)              r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

a :- **not** a — No model

a :- **not** b
b :- **not** a

Model 1: {a}
Model 2: {b}

unhappy(now) :- **not** win(now+1)

	$\{\}$	$\{\mathbf{w}\}$	$\{\text{not}_\mathbf{s}\}$	$\{\text{not}_\mathbf{s}, \mathbf{w}\}$	$\{\text{not}\}$	$\{\text{not}, \mathbf{w}\}$
$\{\}$	P	P	P	P	NP	Δ_2^P
$\{\mathtt{v_h}\}$	NP	Δ_2^P	NP	Δ_2^P	NP	Δ_2^P
$\{\mathtt{v}\}$	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P

"More declarative"

NP-complete (or harder) search problems

Grounding (SAT solving)

Logic Programming

Prolog - "top down query answering"

```
append([], L, L)
append([H|T], L, [H|R]) :-
      append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
```

[X,Y] ++ L = [X,Y|L] …

"More operational"

General purpose PL

Unification/DFBS

Fusemate

Model computation

Functions/data structures

Stratified (negation) by time

Belief revision:

fail(+win(now-1)) :- happy(now)

Answer Set Programming - "model computation"

```
r(X,X)                                r(a,b)
r(X,Y) :- r(Y,X)                      r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

a :- **not** a — No model

a :- **not** b — Model 1: {a}

b :- **not** a — Model 2: {b}

unhappy(now) :- **not** win(now+1)

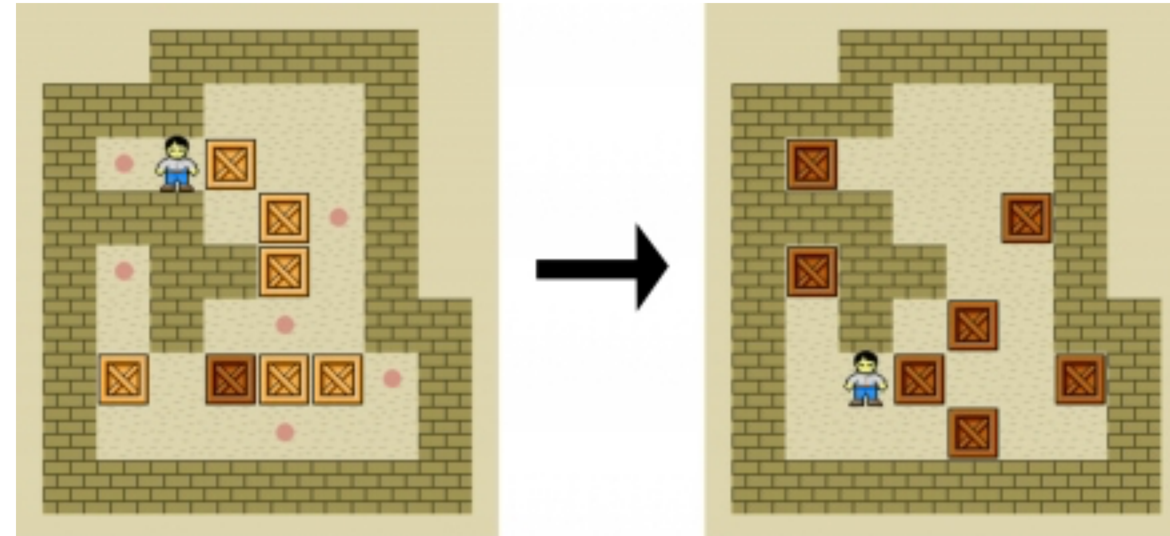
	$\{\}$	$\{\mathbf{w}\}$	$\{\text{not}_\mathbf{s}\}$	$\{\text{not}_\mathbf{s}, \mathbf{w}\}$	$\{\text{not}\}$	$\{\text{not}, \mathbf{w}\}$
$\{\}$	P	P	P	P	NP	Δ_2^P
$\{\text{v}_\text{h}\}$	NP	Δ_2^P	NP	Δ_2^P	NP	Δ_2^P
$\{\text{v}\}$	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P	Σ_2^P	Δ_3^P

"More declarative"

NP-complete (or harder) search problems

Grounding (SAT solving)

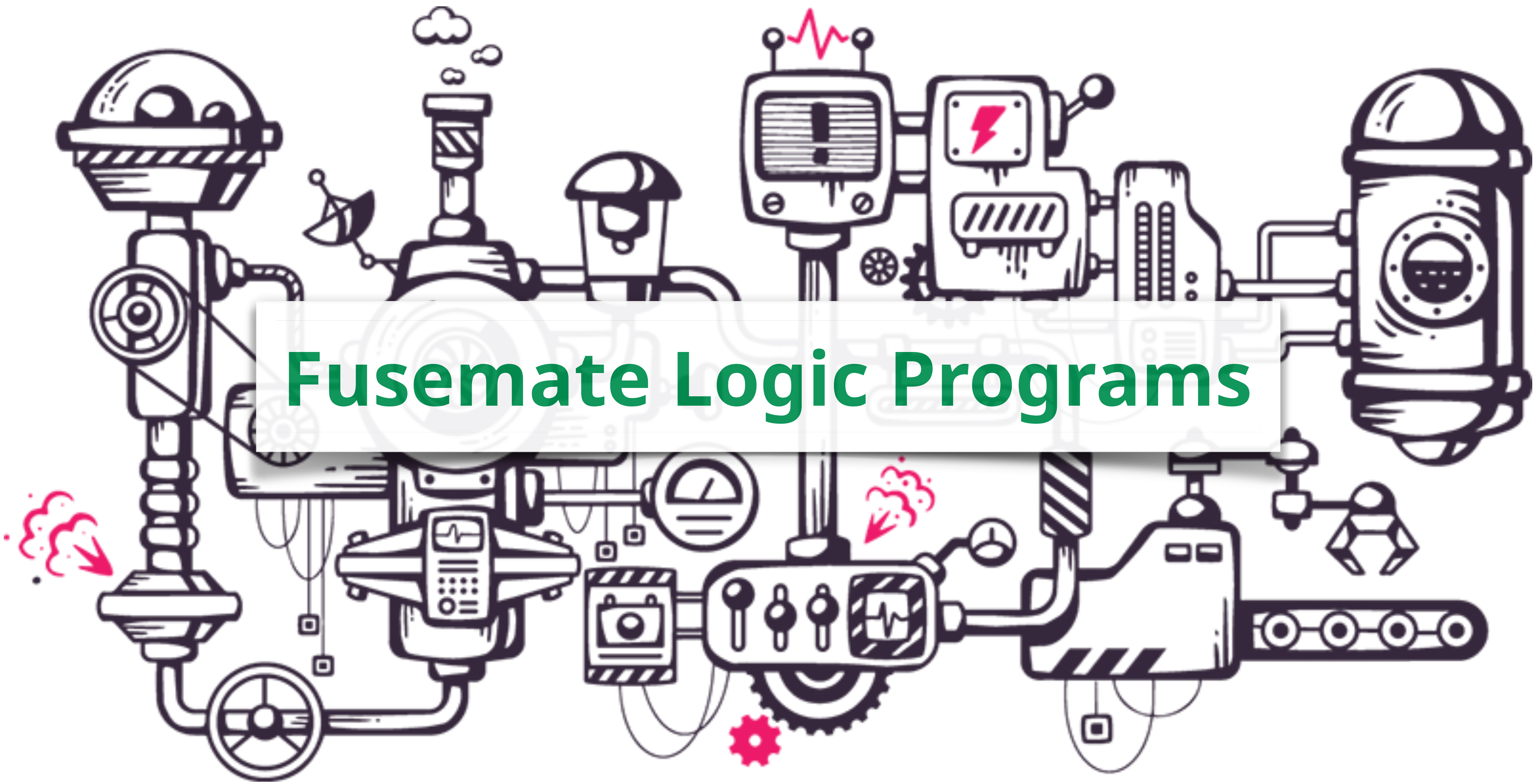
Sokoban Answer Set Solver Program [DLV]



```
time(T) :- #int(T).
actiontime(T) :- #int(T), T != #maxint.

left(L1,L2) :- right(L2,L1).
bottom(L1,L2) :- top(L2,L1).

adj(L1,L2) :- right(L1,L2).
adj(L1,L2) :- left(L1,L2).
adj(L1,L2) :- top(L1,L2).
adj(L1,L2) :- bottom(L1,L2).

location(L) :- adj(L,_).

push(B,right,B1,T) v -push(B,right,B1,T) :- reachable(L,T), right(L,B), box(B,T),
        pushable_right(B,B1,T), good_pushlocation(B1), actiontime(T).
push(B,left,B1,T) v -push(B,left,B1,T) :- reachable(L,T), left(L,B), box(B,T),
        pushable_left(B,B1,T), good_pushlocation(B1), actiontime(T).
push(B,up,B1,T) v -push(B,up,B1,T) :- reachable(L,T), top(L,B), box(B,T),
        pushable_top(B,B1,T), good_pushlocation(B1), actiontime(T).
push(B,down,B1,T) v -push(B,down,B1,T) :- reachable(L,T), bottom(L,B), box(B,T),
        pushable_bottom(B,B1,T), good_pushlocation(B1), actiontime(T).

reachable(L,T) :- sokoban(L,T).
reachable(L,T) :- reachable(L1,T), adj(L1,L), not box(L,T).

pushable_right(B,D,T) :- box(B,T), right(B,D), not box(D,T), actiontime(T).
pushable_right(B,D,T) :- pushable_right(B,D1,T), right(D1,D), not box(D,T).
pushable_left(B,D,T) :- box(B,T), left(B,D), not box(D,T), actiontime(T).
pushable_left(B,D,T) :- pushable_left(B,D1,T), left(D1,D), not box(D,T).
pushable_top(B,D,T) :- box(B,T), top(B,D), not box(D,T), actiontime(T).
pushable_top(B,D,T) :- pushable_top(B,D1,T), top(D1,D), not box(D,T).
pushable_bottom(B,D,T) :- box(B,T), bottom(B,D), not box(D,T), actiontime(T).
pushable_bottom(B,D,T) :- pushable_bottom(B,D1,T), bottom(D1,D), not box(D,T).
```

```
sokoban(L,T1) :- push(_,right,B1,T), #succ(T,T1), right(L,B1).
sokoban(L,T1) :- push(_,left,B1,T), #succ(T,T1), left(L,B1).
sokoban(L,T1) :- push(_,up,B1,T), #succ(T,T1), top(L,B1).
sokoban(L,T1) :- push(_,down,B1,T), #succ(T,T1), bottom(L,B1).
-sokoban(L,T1) :- push(_,_,_,T), #succ(T,T1), sokoban(L,T).

box(B,T1) :- push(_,_,B,T), #succ(T,T1).
-box(B,T1) :- push(B,_,_,T), #succ(T,T1).

box(LB,T1) :- box(LB,T), #succ(T,T1), not -box(LB,T1).
sokoban(LS,T) :- sokoban(LS,T), #succ(T,T1), not -sokoban(LS,T1).

:- push(B,_,_,T), push(B1,_,_,T), B != B1.
:- push(B,D,_,T), push(B,D1,_,T), D != D1.
:- push(B,D,B1,T), push(B,D,B11,T), B1 != B11.


good_pushlocation(L) :- right(L,_), left(L,_).
good_pushlocation(L) :- top(L,_), bottom(L,_).
good_pushlocation(L) :- solution(L).

notgoal :- solution(L), not box(L,#maxint).
not notgoal?
```

Fusemate Logic Programs



Recap: Issues

Fusemate:

Domain Modelling

Multiple aspects (temporal/causal/physical/epistemic/legal/…)

Incomplete

Events

Events **happened** ≠ events **reported** (errors, incomplete, late …)

Explanations

Multiple plausible explanations

Recap: Issues

Domain Modelling

Multiple aspects
(temporal/causal/physical/epistemic/legal/...)
Incomplete

Events

Events **happened** ≠ events **reported** (errors, incomplete, late ...)

Explanations

Multiple plausible explanations

Fusemate:

← Logic program
+ ontologies/event calculus

Recap: Issues

Domain Modelling

Multiple aspects (temporal/causal/physical/epistemic/legal/…)

Incomplete

Events

Events **happened** ≠ events **reported** (errors, incomplete, late …)

Explanations

Multiple plausible explanations

Fusemate:

← Logic program + ontologies/event calculus

← Belief revision

Recap: Issues

Fusemate:

Domain Modelling

Multiple aspects (temporal/causal/physical/epistemic/legal/...) ← Logic program + ontologies/event calculus

Incomplete

Events

Events **happened** ≠ events **reported** (errors, incomplete, late ...) ← Belief revision

Explanations

Multiple plausible explanations ← Models of logic program

Events happened ≠ events reported

“Fixing the event stream”

Events happened ≠ events reported

“Fixing the event stream”

Reported

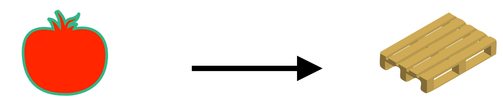
```
Load(  10, tomatoes,  pallet)
Load(  20, pallet,    container)
Load(  40, container, ship)

Unload(60, apples,    pallet)
```

Events happened ≠ events reported

“Fixing the event stream”

Reported

```
Load(  10, tomatoes,  pallet)
Load(  20, pallet,    container)
Load(  40, container, ship)

Unload(60, apples,    pallet)
```

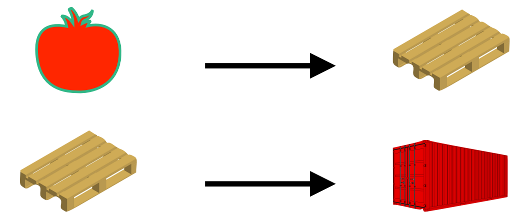


Events happened ≠ events reported

“Fixing the event stream”

Reported

```
Load(  10, tomatoes,  pallet)
Load(  20, pallet,    container)
Load(  40, container, ship)

Unload(60, apples,    pallet)
```

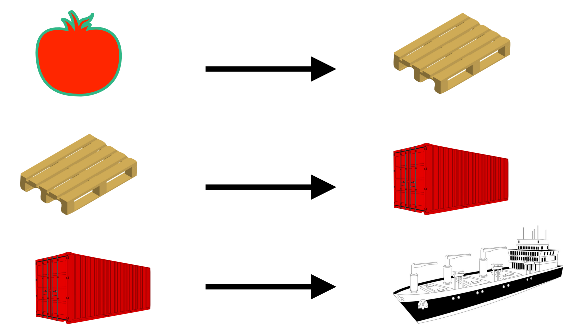


Events happened ≠ events reported

"Fixing the event stream"

Reported

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container, ship)

Unload(60, apples,     pallet)
```

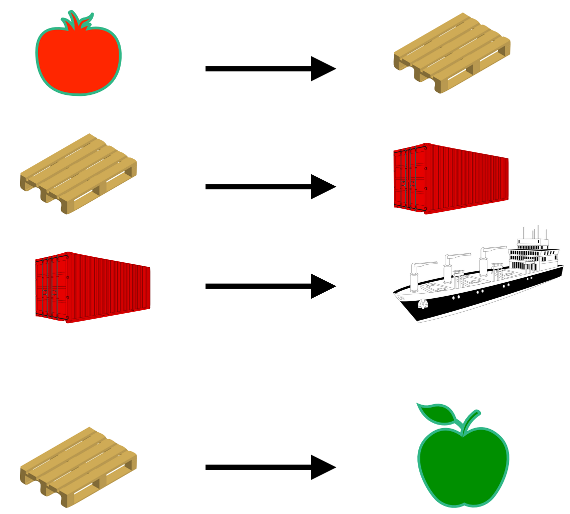


Events happened ≠ events reported

“Fixing the event stream”

Reported

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container, ship)

Unload(60, apples,     pallet)
```



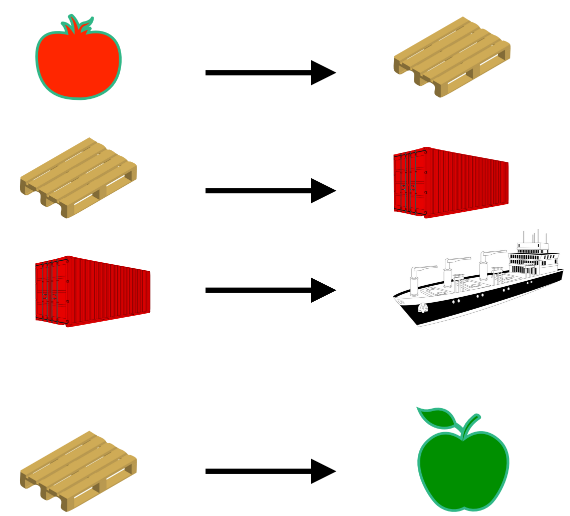
Events happened ≠ events reported

"Fixing the event stream"

Reported

```
Load(   10, tomatoes,  pallet)
Load(   20, pallet,    container)
Load(   40, container, ship)

Unload(60, apples,     pallet)
```

?

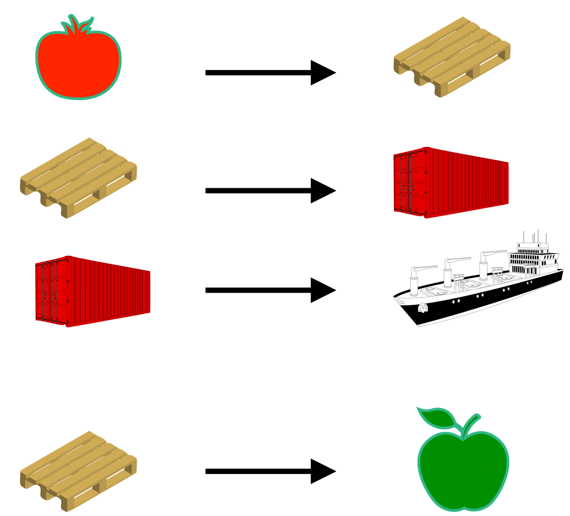


Events happened ≠ events reported

“Fixing the event stream”

Reported

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)

Unload(60, apples,      pallet)
```



Happened

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, tomatoes,   pallet)
```

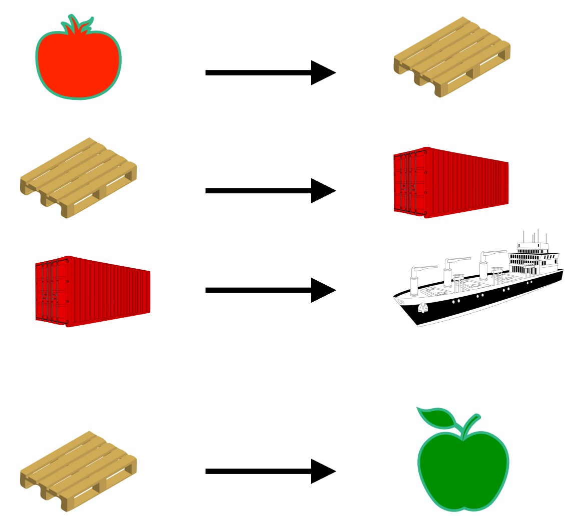
Events happened ≠ events reported

“Fixing the event stream”

Reported

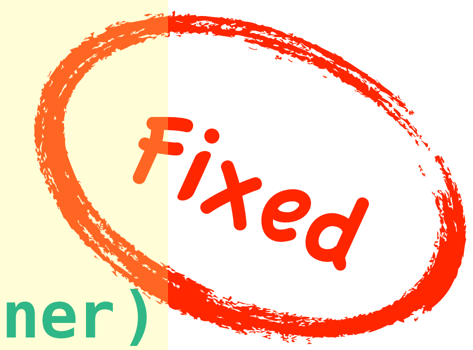
```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container, ship)

Unload(60, apples,     pallet)
```

?



Happened

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container, ship)
Unload(45, container, ship)
Unload(50, pallet,     container)
Unload(60, tomatoes,   pallet)
```



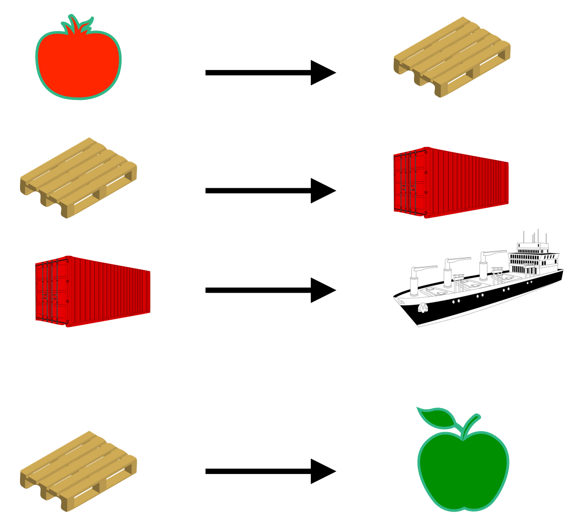
Events happened ≠ events reported

"Fixing the event stream"

Reported

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)

Unload(60, apples,      pallet)
```

?



Happened

```
Load(   10, apples,     pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, apples,     pallet)
```

Fixed

Happened

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, tomatoes,   pallet)
```

Fixed

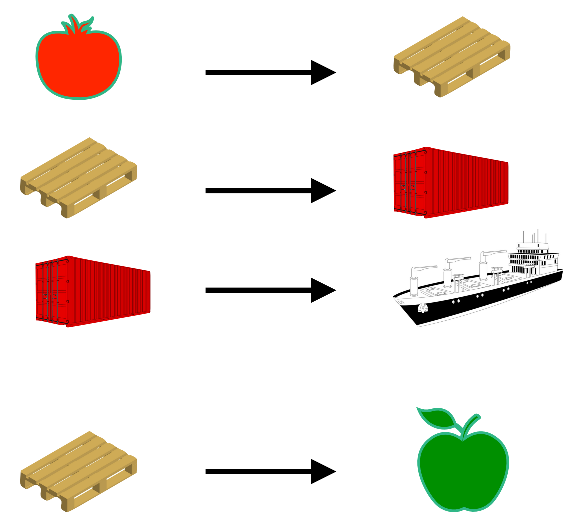
Events happened ≠ events reported

"Fixing the event stream"

Reported

```
Load(   10, tomatoes,  pallet)
Load(   20, pallet,    container)
Load(   40, container, ship)

Unload(60, apples,     pallet)
```

?



Happened

```
Load(   10, tomatoes,  pallet)
Load(   20, pallet,    container)
Load(   40, container, ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, tomatoes,   pallet)
```

Fixed

Happened

```
Load(   10, apples,    pallet)
Load(   20, pallet,    container)
Load(   40, container, ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, apples,     pallet)
```

Fixed

Happened

```
Load(   10, tomatoes,  pallet)
Load(   10, apples,    pallet)
Load(   20, pallet,    container)
Load(   40, container, ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, apples,     pallet)
```

Fixed

Events happened ≠ events reported

“Fixing the event stream”

Reported

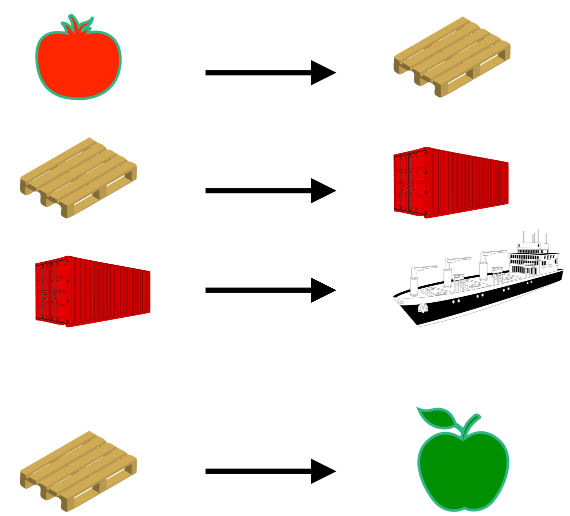
```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)

Unload(60, apples,     pallet)
```

?

Happened

```
Load(   10, tomatoes,   pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, tomatoes,   pallet)
```

Fixed



Next:

logic program expressing this

Happened

```
Load(   10, apples,     pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, apples,     pallet)
```

Fixed

Happened

```
Load(   10, tomatoes,   pallet)
Load(   10, apples,     pallet)
Load(   20, pallet,     container)
Load(   40, container,  ship)
Unload(45, container,  ship)
Unload(50, pallet,     container)
Unload(60, apples,     pallet)
```

Fixed

Logic Program for the Supply Chain Example

Derived “In” relation

Integrity constraints and revision

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)
```

Integrity constraints and revision

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)
```

```
// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)
```

Integrity constraints and revision

Logic Program for the Supply Chain Example

Derived “In” relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)
```

```
// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)
```

```
// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Integrity constraints and revision

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)


// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)


// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

Integrity constraints and revision

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)
```

```
// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)
```

```
// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

Integrity constraints and revision

```
// No Unload without earlier Load
fail :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont),
       t < time))
```

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)


// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)


// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

Integrity constraints and revision

```
// No Unload without earlier Load
fail :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont),
       t < time))

// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont),
  t < time,
  SameBatch(t, b),
  ((b contains obj) && (b contains o))
```

Logic Program for the Supply Chain Example

Derived “In” relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)

// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)

// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

Integrity constraints and revision

```
// No Unload without earlier Load
fail :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont),
       t < time))

// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont),
  t < time,
  SameBatch(t, b),
  ((b contains obj) && (b contains o))
```

“fail” heads for fixing the event stream

new

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)
```

```
// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)
```

```
// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

Integrity constraints and revision

```
// No Unload without earlier Load
fail :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont),
       t < time))
```

"fail" heads for fixing the event stream

new

```
// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont),
  t < time,
  SameBatch(t, b),
  ((b contains obj) && (b contains o))
```

\+ *4 more rules*

Logic Program for the Supply Chain Example

Derived "In" relation

```
In(time, obj, cont) :-
  Load(time, obj, cont)

// In transitivity
In(time, obj, cont) :-
  In(time, obj, c),
  In(time, c, cont)

// Frame axiom for In
In(time, obj, cont) :-
  In(prev, obj, cont),
  Step(time, prev),
  not Unload(time, obj, cont),
  not (In(prev, obj, c),
       Unload(time, c, cont))
```

Default negation

(Frame axioms now via Event Calculus)

Integrity constraints and revision

```
// No Unload without earlier Load
fail :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont),
       t < time))

// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont),
  t < time,
  SameBatch(t, b),
  ((b contains obj) && (b contains o))
```

"fail" heads for fixing the event stream

new

\+ *4 more rules*

Situational Awareness = Stratified Model Computation

"Situational awareness" task is naturally stratified

Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

Situational Awareness = Stratified Model Computation

"Situational awareness" task is naturally stratified

"Not known now" -> "never known"
Makes default negation possible

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,…,now

Situational Awareness = Stratified Model Computation

"Not known now" -> "never known"
Makes default negation possible

"Situational awareness" task is naturally stratified

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

“Not known now” -> “never known”
Makes default negation possible

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Situational Awareness = Stratified Model Computation

“Not known now” -> “never known”
Makes default negation possible

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now
- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)

Situational Awareness = Stratified Model Computation

“Not known now” -> “never known”
Makes default negation possible

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)

Situational Awareness = Stratified Model Computation

“Not known now” -> “never known”
Makes default negation possible

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

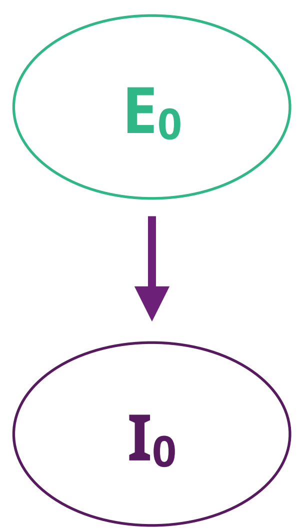
- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)

EDBs $E_{0,1,2,\ldots}$

IDBs $I_{0,1,2,\ldots}$



Situational Awareness = Stratified Model Computation

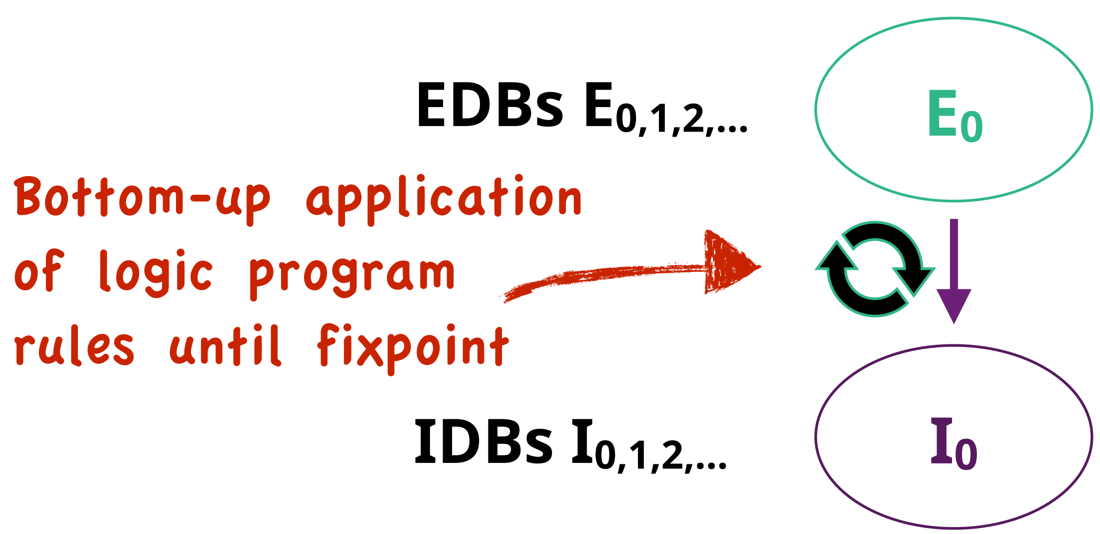
"Not known now" -> "never known"
Makes default negation possible

"Situational awareness" task is naturally stratified

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,…,now
- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

"Situational awareness" task is naturally stratified

"Not known now" -> "never known"
Makes default negation possible

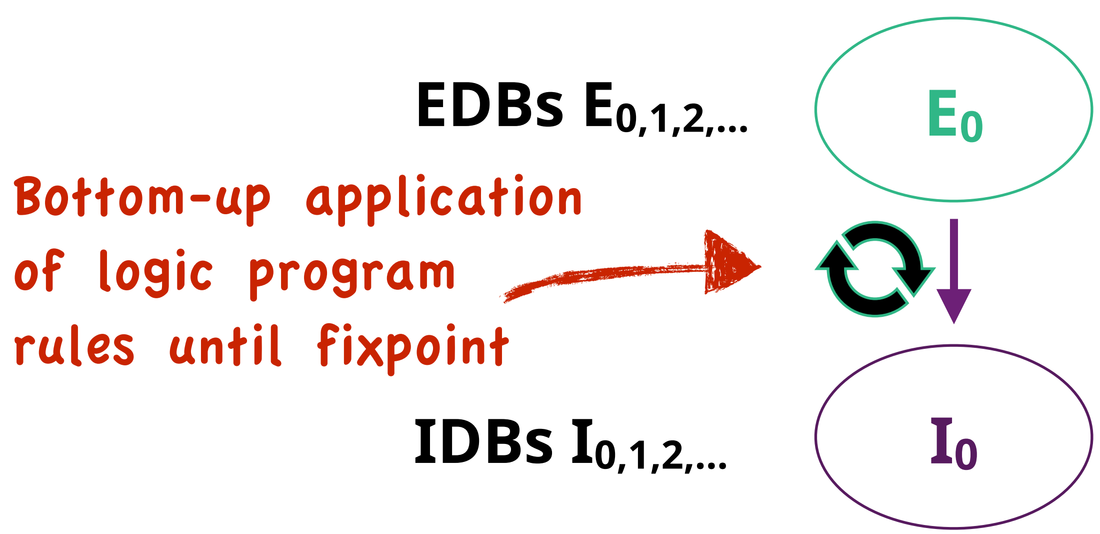
- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)

Time 0,1,2 ·····▶



Situational Awareness = Stratified Model Computation

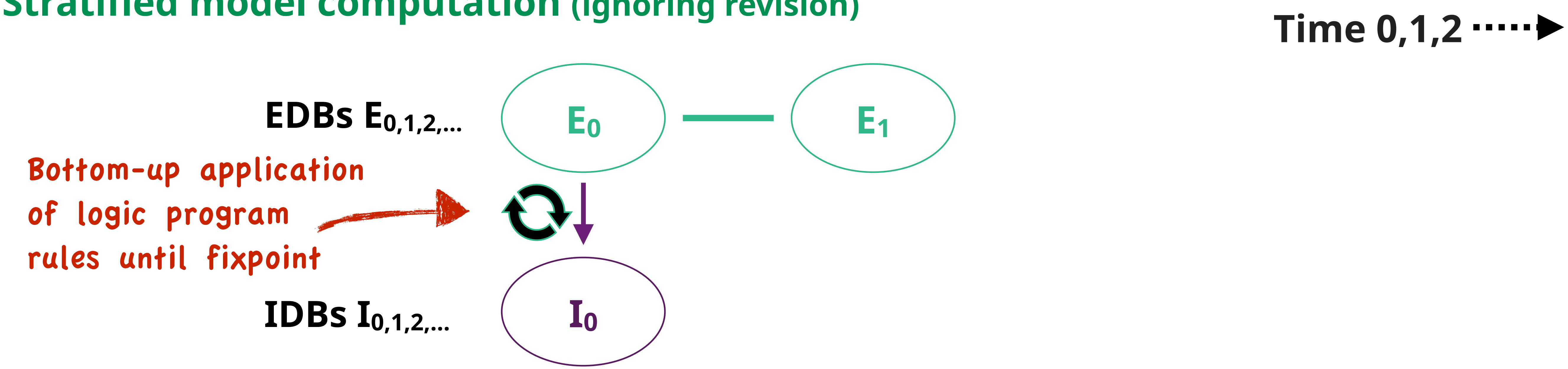
"Situational awareness" task is naturally stratified

"Not known now" -> "never known"
Makes default negation possible

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,...,now
- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

"Situational awareness" task is naturally stratified

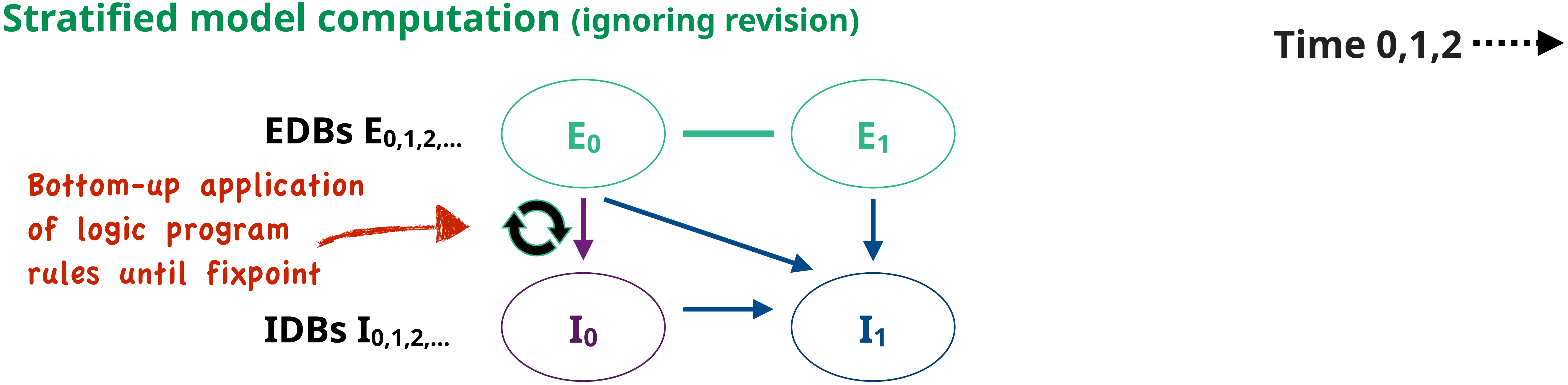
"Not known now" -> "never known"
Makes default negation possible

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

"Situational awareness" task is naturally stratified

"Not known now" -> "never known"
Makes default negation possible

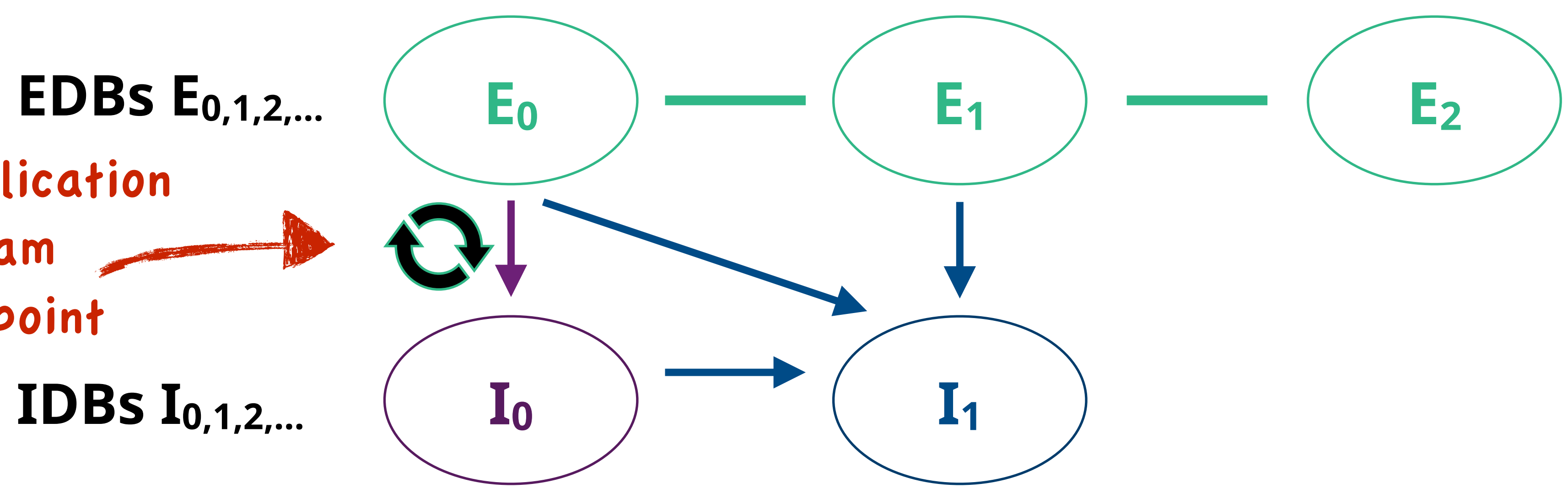
- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,…,now
- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)

Time 0,1,2 ⋯⋯▶

Bottom-up application of logic program rules until fixpoint



Situational Awareness = Stratified Model Computation

"Not known now" -> "never known"
Makes default negation possible

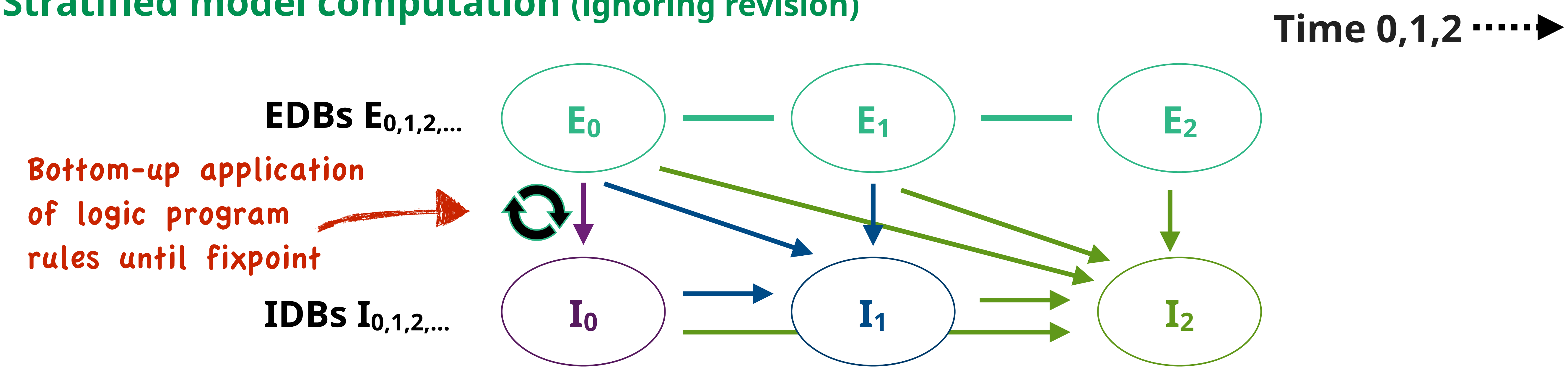
"Situational awareness" task is naturally stratified

- Comprehend evolving situation from "past" and "now", not "future"
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

“Not known now” -> “never known”
Makes default negation possible

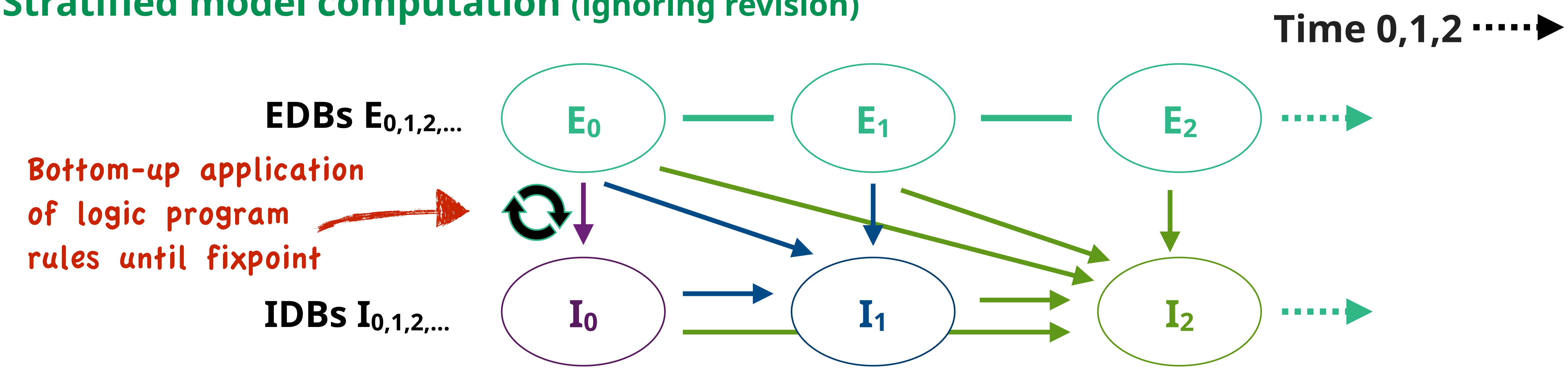
“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

"Not known now" -> "never known"
Makes default negation possible

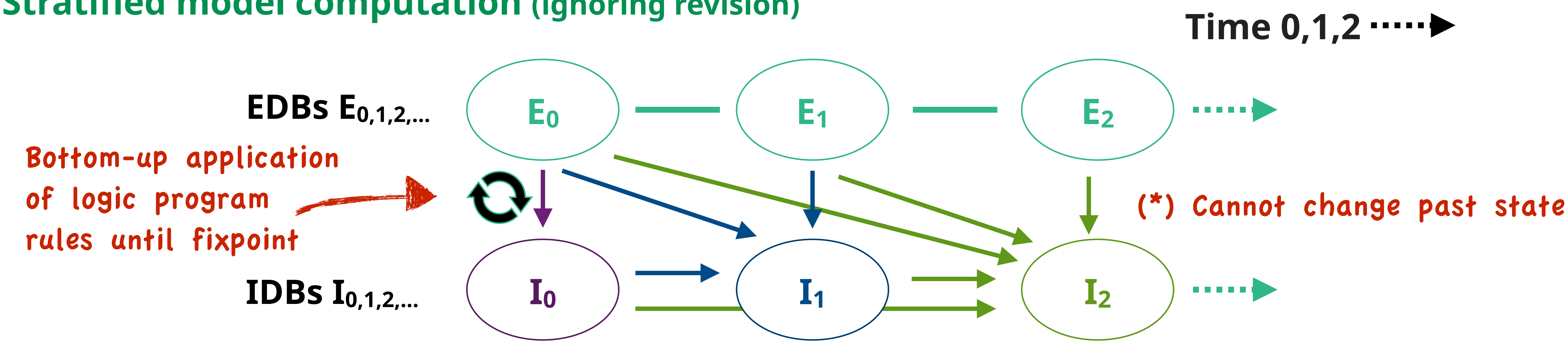
"Situational awareness" task is naturally stratified

- Comprehend evolving situation from "past" and "now", not "future" (*)
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Situational Awareness = Stratified Model Computation

“Not known now” -> “never known”
Makes default negation possible

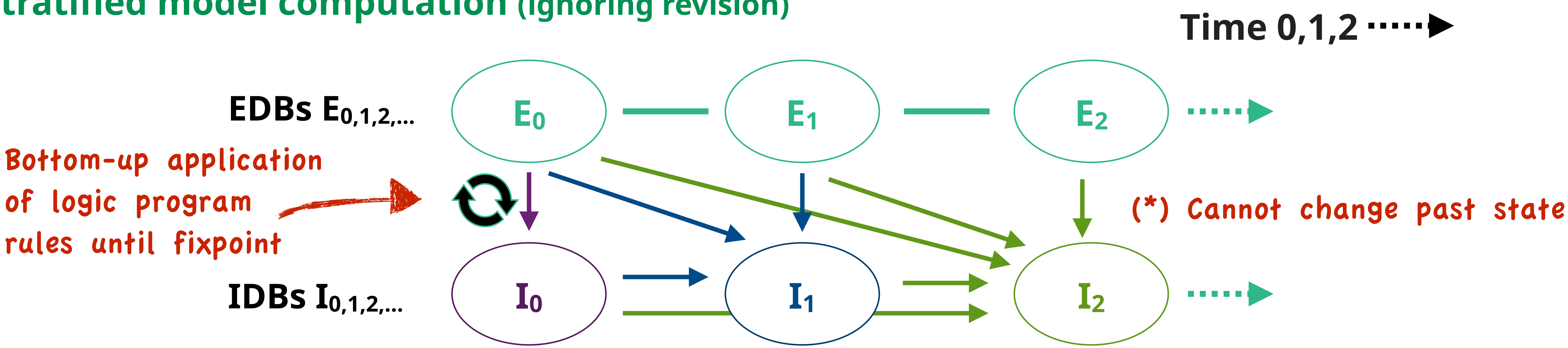
“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future” (*)
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Revising events is simply addition/removal

Stratified model computation (ignoring revision)



Next: Stratified logic programs for computing models $(E \cup I)_0$, $(E \cup I)_1$, $(E \cup I)_2$, …

Stratified Logic Programs

Consists of rules over literals

```
head :- body, …, not body, …
```

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*`,` `…,` **`not`** *body*`,` `…`

s. th. (1) var(*head*) ⊆ fvar(*body*`,` `…,` **`not`** *body*`,` `…`)

(2) *head* has a *time* variable ("*now*")

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **`not`** *body* have time ≤ *time*

(6) IDB lits in **`not`** *body* have time < *time*

Stratified Logic Programs

Consists of rules over literals

`head :- body, …, not body, …`

s. th. (1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …) ← Range restriction ⤳ Simple model computation

(2) *head* has a *time* variable ("*now*")

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*`,` …`,` **`not`** *body*`,` …

s. th. (1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)

(2) *head* has a *time* variable (“*now*”)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
⇝ Simple model computation

Stratification by time
⇝ Effective model computation

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th.
(1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)
(2) *head* has a *time* variable ("*now*")
(3) one *body* lit has same *time* variable
(4) other *body* lits have time ≤ *time*
(5) EDB lits in **not** *body* have time ≤ *time*
(6) IDB lits in **not** *body* have time < *time*

← Range restriction
↝ Simple model computation

← Stratification by time
↝ Effective model computation

← Avoids guessing whether head is true or false in final model
↝ Efficient model computation

Stratified Logic Programs

Consists of rules over literals

head **:-** *body*, …, **not** *body*, …

s. th. *(1)* var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)

(2) *head* has a *time* variable (“*now*”)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
⇝ Simple model computation

Stratification by time
⇝ Effective model computation

Avoids guessing whether head is true or false in final model
⇝ Efficient model computation

Examples

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th. *(1)* var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)

(2) *head* has a *time* variable (*"now"*)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
⇝ Simple model computation

Stratification by time
⇝ Effective model computation

Avoids guessing whether head is true or false in final model
⇝ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)
```

I, J: IDB
E: EDB

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th.
(1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)
(2) *head* has a *time* variable ("*now*")
(3) one *body* lit has same *time* variable
(4) other *body* lits have time ≤ *time*
(5) EDB lits in **not** *body* have time ≤ *time*
(6) IDB lits in **not** *body* have time < *time*

Range restriction
↝ Simple model computation

Stratification by time
↝ Effective model computation

Avoids guessing whether head is true or false in final model
↝ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)

I(time, x) :- J(time, x, y), I(t, y), t ≤ time
```

I,J: IDB
E: EDB

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th. (1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …) ← Range restriction ⇝ Simple model computation

(2) *head* has a *time* variable ("*now*")

(3) one *body* lit has same *time* variable ← Stratification by time ⇝ Effective model computation

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time* ← Avoids guessing whether head is true or false in final model ⇝ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)

I(time, x) :- J(time, x, y), I(t, y), t ≤ time

I(time, x) :- J(time, x, y), not (I(t, y), t < time)
```

I, J: IDB
E: EDB

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th. (1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)

(2) *head* has a *time* variable (“*now*”)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
⤳ Simple model computation

Stratification by time
⤳ Effective model computation

Avoids guessing whether head is true or false in final model
⤳ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)
I(time, x) :- J(time, x, y), I(t, y), t ≤ time
I(time, x) :- J(time, x, y), not (I(t, y), t < time)
```

Closed world assumption
E∪I ⊨ not body[x] iff
not exists a s.th. body[a] ⊆ E∪I

I, J: IDB
E: EDB

Stratified Logic Programs

Consists of rules over literals

head :- *body*, …, **not** *body*, …

s. th. *(1)* var(*head*) ⊆ fvar(*body*, …, ***not*** *body*, …)

(2) *head* has a *time* variable (*"now"*)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
↝ Simple model computation

Stratification by time
↝ Effective model computation

Avoids guessing whether head is true or false in final model
↝ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)
I(time, x) :- J(time, x, y), I(t, y), t ≤ time
I(time, x) :- J(time, x, y), not (I(t, y), t < time)
I(time, x) :- J(time, x, y), not (I(t, y), t ≤ time)
```

No!

Closed world assumption
E∪I ⊨ not body[x] iff
not exists a s.th. body[a] ⊆ E∪I

I,J: IDB
E: EDB

Stratified Logic Programs

Consists of rules over literals

head `:-` *body*, …, **not** *body*, …

s. th. (1) var(*head*) ⊆ fvar(*body*, …, **not** *body*, …)

(2) *head* has a *time* variable (“*now*”)

(3) one *body* lit has same *time* variable

(4) other *body* lits have time ≤ *time*

(5) EDB lits in **not** *body* have time ≤ *time*

(6) IDB lits in **not** *body* have time < *time*

Range restriction
⇝ Simple model computation

Stratification by time
⇝ Effective model computation

Avoids guessing whether head is true or false in final model
⇝ Efficient model computation

Examples

```
I(time, x) :- J(time, x, y), I(time, y)
I(time, x) :- J(time, x, y), I(t, y), t ≤ time
I(time, x) :- J(time, x, y), not (I(t, y), t < time)
I(time, x) :- J(time, x, y), not (I(t, y), t ≤ time)   No!
I(time, x) :- J(time, x, y), not (E(t, y), t ≤ time)
```

Closed world assumption
E∪I ⊨ not body[x] iff
not exists a s.th. body[a] ⊆ E∪I

I, J: IDB
E: EDB

Integrity Constraints and Belief Revision

[IJCAR 2020]

Usual integrity constraints

`fail` `:-` *body*, …, **`not`** *body*, …

Generalized for revision of EDB literals

`fail(-`*e*`, …, +`*f*`, …)` `:-` *body*, …, **`not`** *body*, …

s. th.

- *“conditions for body as for ordinary rules”*
- EDB lits *e* and *f* have time ≤ *time*

Integrity Constraints and Belief Revision

[IJCAR 2020]

Usual integrity constraints

fail :- *body*, …, **not** *body*, …

Generalized for revision of EDB literals

fail(-*e*, …, +*f*, …) :- *body*, …, **not** *body*, …

s. th.

- *"conditions for body as for ordinary rules"*
- EDB lits *e* and *f* have time ≤ *time*

Example

```
// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont), t < time,
  …
```

Integrity Constraints and Belief Revision

[IJCAR 2020]

Usual integrity constraints

fail :- *body*, …, **not** *body*, …

Generalized for revision of EDB literals

fail(-*e*, …, +*f*, …**)** :- *body*, …, **not** *body*, …

s. th.
- *"conditions for body as for ordinary rules"*
- EDB lits *e* and *f* have time ≤ *time*

Example

```
// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont), t < time,
  …
```

```
…
- Unload(60, apples,    pallet)
          ↓
…
+ Unload(60, tomatoes,  pallet)
```

Integrity Constraints and Belief Revision

[IJCAR 2020]

Usual integrity constraints

```
fail :- body, …, not body, …
```

Generalized for revision of EDB literals

```
fail(-e, …, +f, …) :- body, …, not body, …
```

s. th.
- *"conditions for body as for ordinary rules"*
- EDB lits *e* and *f* have time ≤ *time*

Semantics

$E \cup I$

↓ if $E \cup I \models (body,\ \ldots,\ \textbf{not}\ body,\ \ldots)\sigma$

$(E \setminus e\sigma) \cup f\sigma$

Example

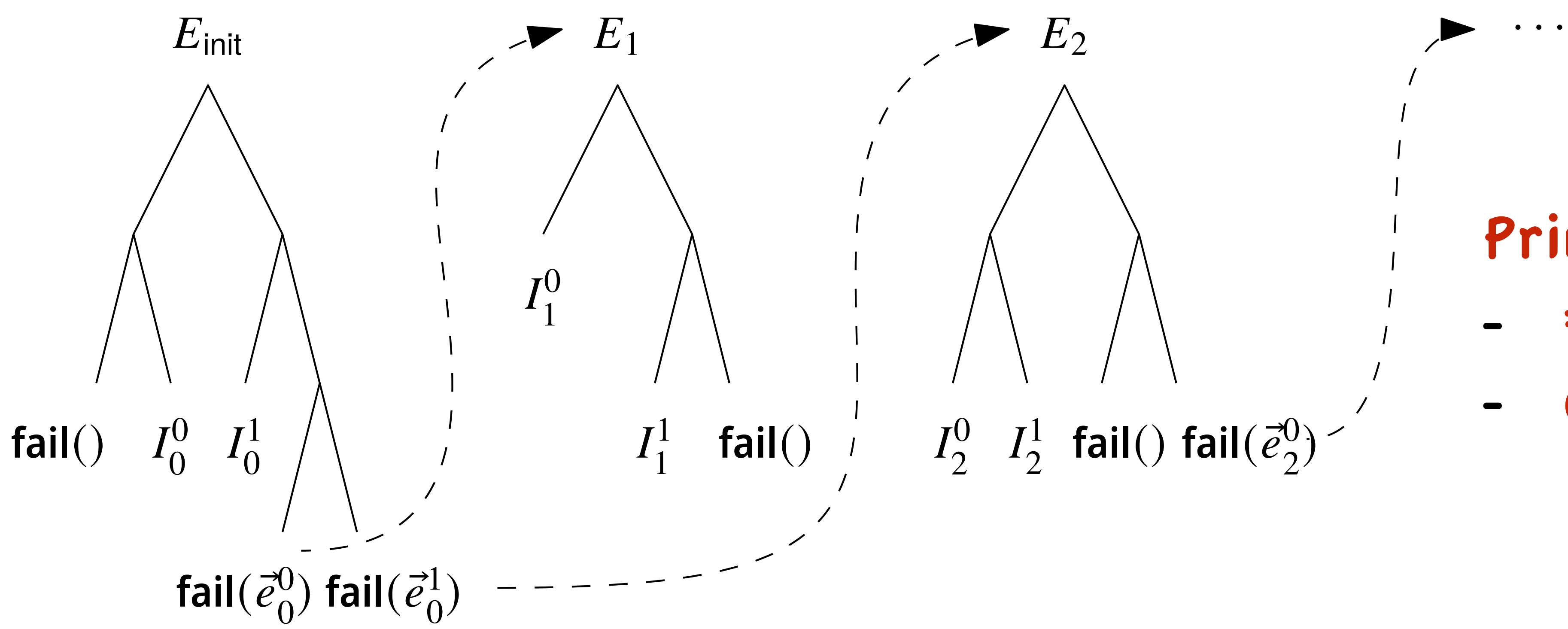
```
// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont), t < time,
  …
```

```
…
- Unload(60, apples,   pallet)
```

↓

```
…
+ Unload(60, tomatoes,  pallet)
```

Semantics of Programs With Fail Rules



Principles

- Fail as early as possibly
- Collect all possible fails

Operational

for a given EDB E

for time t = 0,1,2, …, now

compute { I^0, I^1, … all IDBs for time ≤ t }

for I = I^0, I^1, …

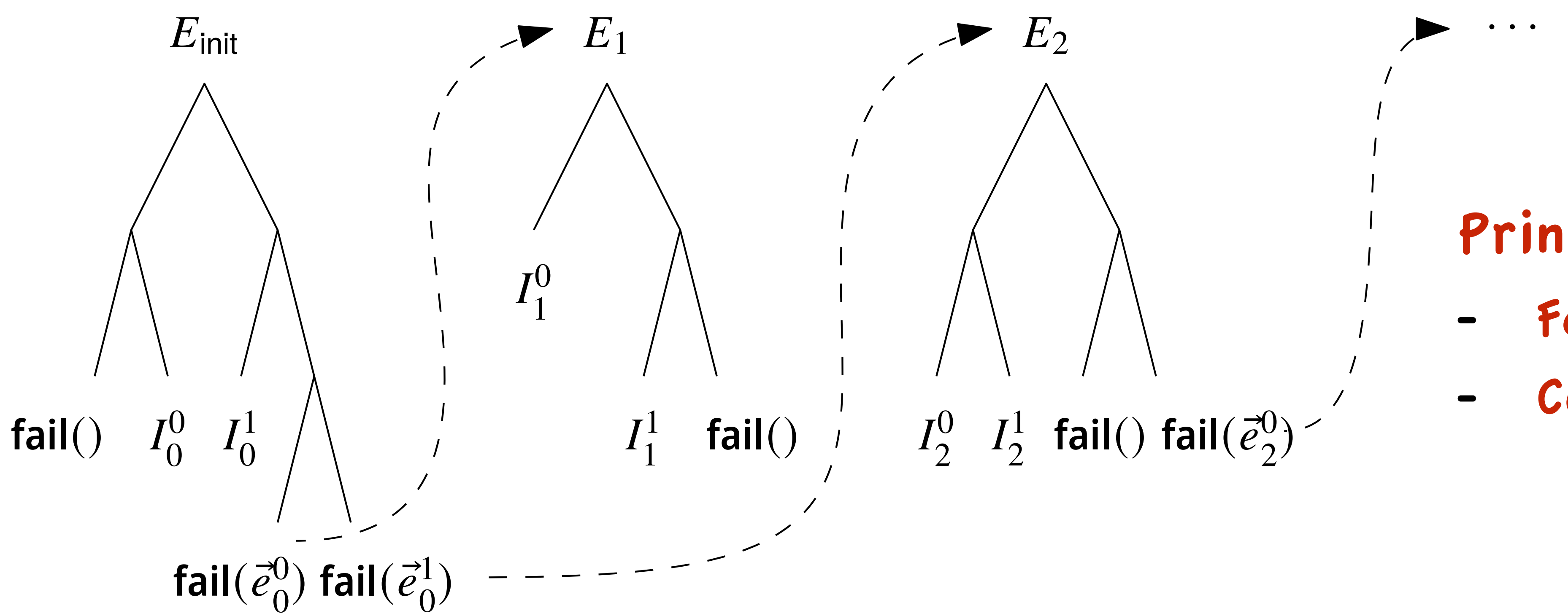
let F = { **`fail(…)`** heads derivable from $E \cup I$ }

if F is non-empty then

obtain new EDBs E_1, E_2, … as per F and

abandon model candidate I

Semantics of Programs With Fail Rules



Principles

- Fail as early as possibly
- Collect all possible fails

Operational

for a given EDB E

for time t = 0,1,2, ..., now ← Can branch out because of disjunctive heads

compute { I^0, I^1, ... all IDBs for time ≤ t }

for I = I^0, I^1, ...

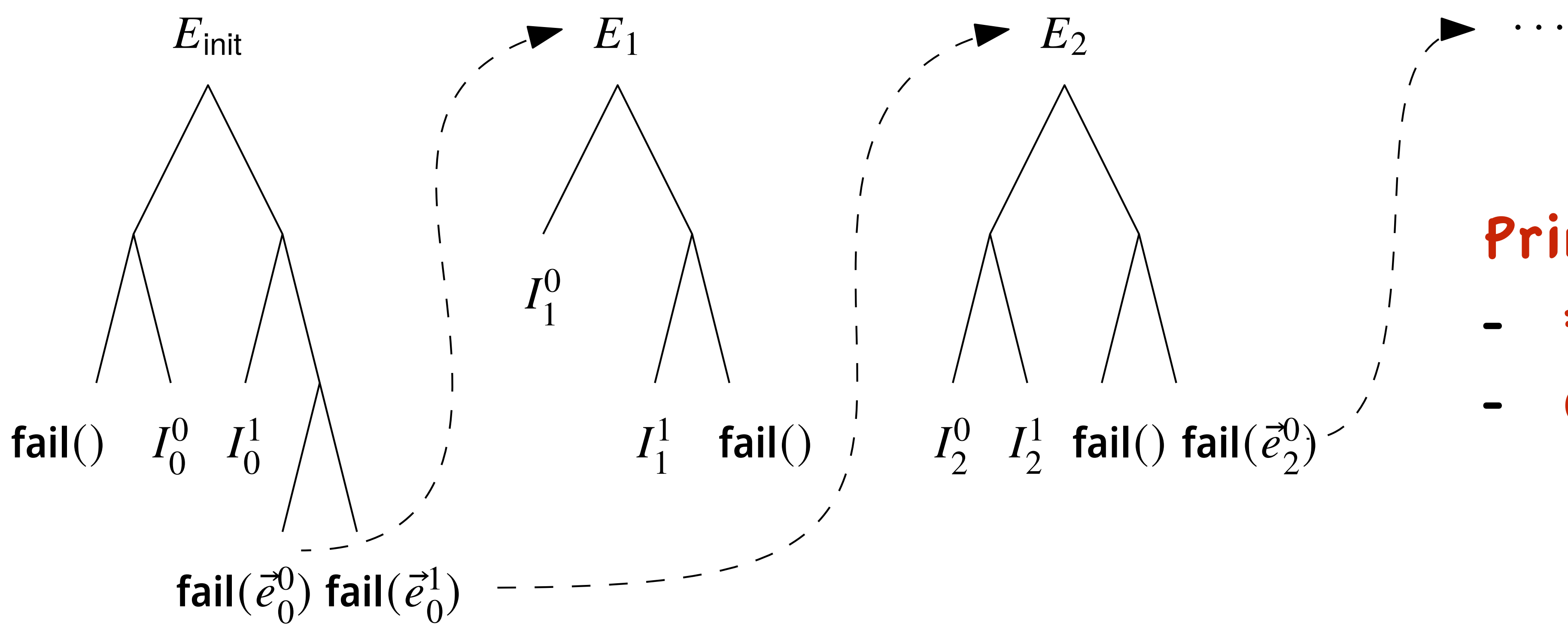
let F = { **`fail(…)`** heads derivable from $E \cup I$ }

if F is non-empty then

obtain new EDBs E_1, E_2, ... as per F and

abandon model candidate I

Semantics of Programs With Fail Rules



Principles

- Fail as early as possibly
- Collect all possible fails

Operational

for a given EDB *E*

for time *t* = 0,1,2, ..., now — Can branch out because of disjunctive heads

compute { I^0, I^1, ... all IDBs for time ≤ *t* }

for I = I^0, I^1, ...

let F = { **`fail(…)`** heads derivable from *E*∪*I* }

if F is non-empty then

obtain new EDBs E_1, E_2, ... as per F and

abandon model candidate I

Declarative semantics: see paper

Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

Instances "ABox"

Concepts "TBox"

$\text{Box} \sqsubseteq \forall\, \text{temp.TempClass}$

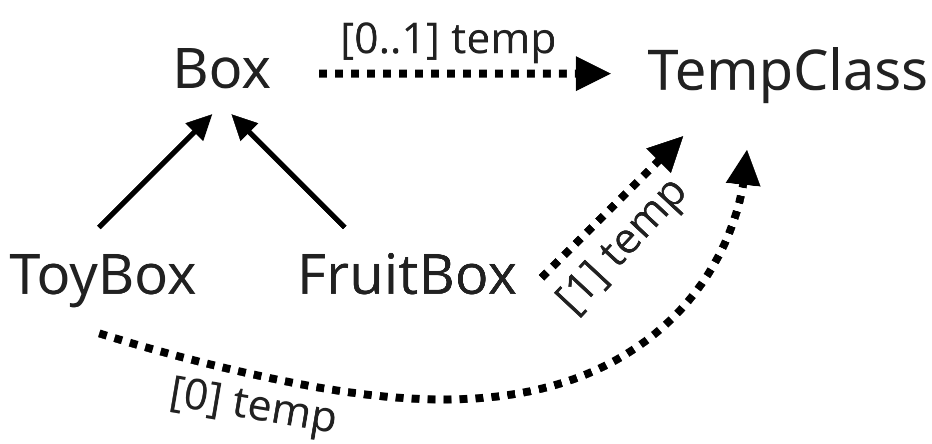
$\text{FruitBox} \sqsubseteq \exists\, \text{temp.TempClass}$

$\text{ToyBox} \sqsubseteq \neg\exists\, \text{temp.TempClass}$

$\text{FruitBox} \sqsubseteq \text{Box}$

$\text{ToyBox} \sqsubseteq \text{Box}$

temp is a functional role



Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies (“is-a” and “has-a” relations)
- Reasoning on concepts and concept instances

Concepts “TBox”

$\text{Box} \sqsubseteq \forall\, \text{temp.TempClass}$

$\text{FruitBox} \sqsubseteq \exists\, \text{temp.TempClass}$

$\text{ToyBox} \sqsubseteq \neg\exists\, \text{temp.TempClass}$

$\text{FruitBox} \sqsubseteq \text{Box}$

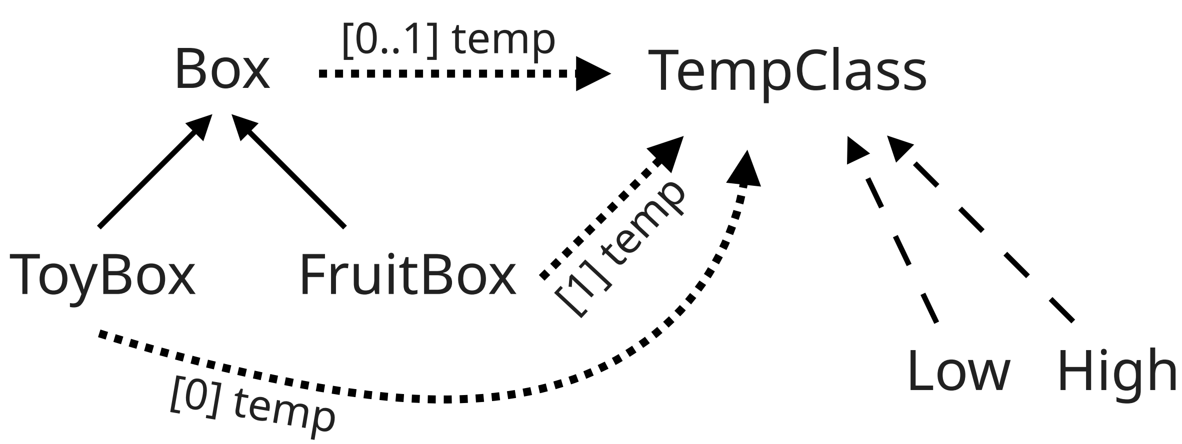
$\text{ToyBox} \sqsubseteq \text{Box}$

temp is a functional role

Instances “ABox”

$\text{Low} : \text{TempClass}$

$\text{High} : \text{TempClass}$



Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

Concepts "TBox"

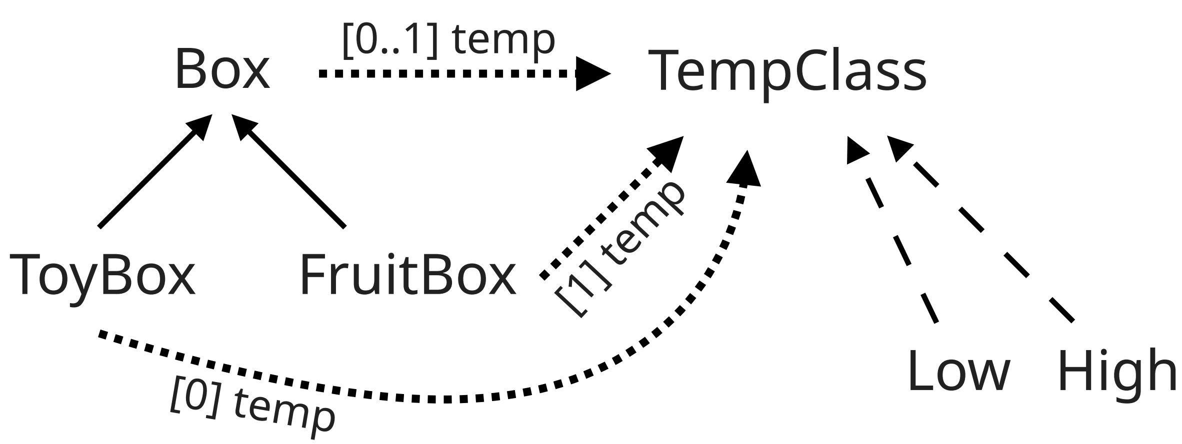
$$\mathsf{Box} \sqsubseteq \forall\, \mathsf{temp.TempClass}$$
$$\mathsf{FruitBox} \sqsubseteq \exists\, \mathsf{temp.TempClass}$$
$$\mathsf{ToyBox} \sqsubseteq \neg\exists\, \mathsf{temp.TempClass}$$
$$\mathsf{FruitBox} \sqsubseteq \mathsf{Box}$$
$$\mathsf{ToyBox} \sqsubseteq \mathsf{Box}$$

temp is a functional role

$$\mathsf{Low} : \mathsf{TempClass}$$
$$\mathsf{High} : \mathsf{TempClass}$$

Instances "ABox"

$$\mathsf{Box}_0 : \mathsf{FruitBox}$$
$$\mathsf{Box}_1 : \mathsf{FruitBox}$$
$$\mathsf{Box}_2 : \mathsf{Box}$$
$$\mathsf{Box}_3 : \mathsf{ToyBox}$$
$$\mathsf{Box}_4 : \mathsf{Box} \sqcap \forall\, \mathsf{temp}.\neg\mathsf{TempClass}$$
$$\mathsf{Box}_5 : \mathsf{Box} \sqcap \exists\, \mathsf{temp.TempClass}$$



Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

Concepts "TBox"

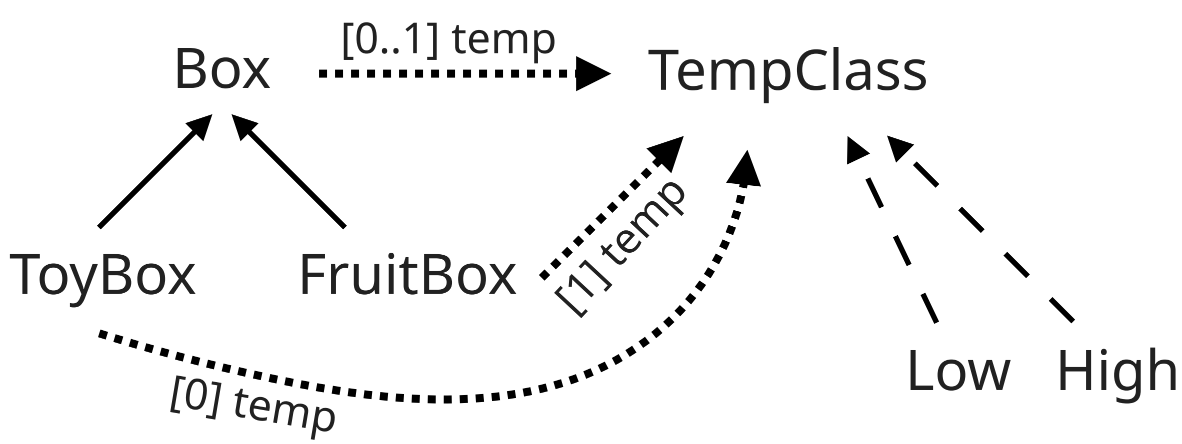
$$\text{Box} \sqsubseteq \forall\, \text{temp.TempClass}$$
$$\text{FruitBox} \sqsubseteq \exists\, \text{temp.TempClass}$$
$$\text{ToyBox} \sqsubseteq \neg\exists\, \text{temp.TempClass}$$
$$\text{FruitBox} \sqsubseteq \text{Box}$$
$$\text{ToyBox} \sqsubseteq \text{Box}$$

temp is a functional role

$$\text{Low} : \text{TempClass}$$
$$\text{High} : \text{TempClass}$$

Instances "ABox"

$$\text{Box}_0 : \text{FruitBox}$$
$$\text{Box}_1 : \text{FruitBox}$$
$$\text{Box}_2 : \text{Box}$$
$$\text{Box}_3 : \text{ToyBox}$$
$$\text{Box}_4 : \text{Box} \sqcap \forall\, \text{temp.}\neg\text{TempClass}$$
$$\text{Box}_5 : \text{Box} \sqcap \exists\, \text{temp.TempClass}$$



Reasoning

Is Box_4 a FruitBox?
Is Box_5 a FruitBox?
Are FruitBox and ToyBox disjoint?

Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies (“is-a” and “has-a” relations)
- Reasoning on concepts and concept instances

Concepts “TBox”

$\text{Box} \sqsubseteq \forall\, \text{temp.TempClass}$

$\text{FruitBox} \sqsubseteq \exists\, \text{temp.TempClass}$

$\text{ToyBox} \sqsubseteq \neg\exists\, \text{temp.TempClass}$

$\text{FruitBox} \sqsubseteq \text{Box}$

$\text{ToyBox} \sqsubseteq \text{Box}$

temp is a functional role

Instances “ABox”

$\text{Low} : \text{TempClass}$

$\text{High} : \text{TempClass}$

$\text{Box}_0 : \text{FruitBox}$

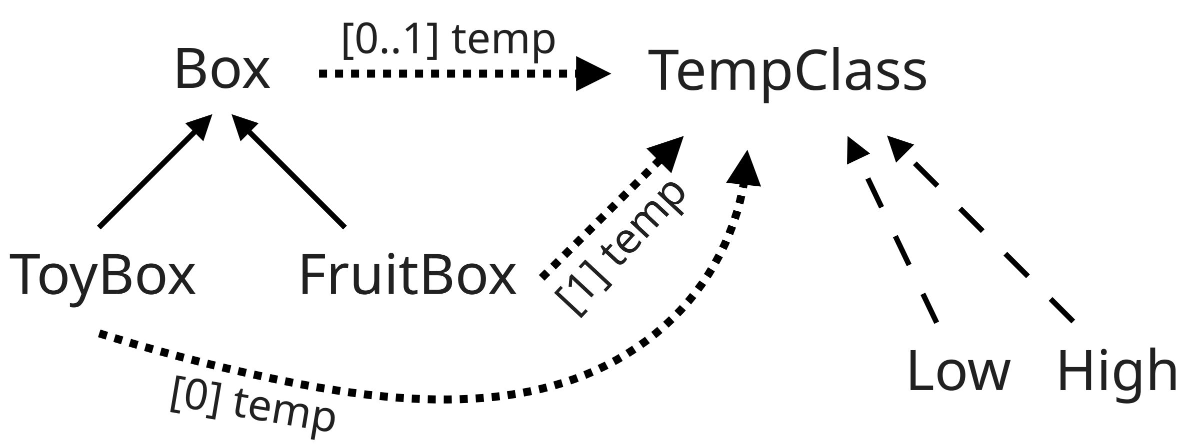
$\text{Box}_1 : \text{FruitBox}$

$\text{Box}_2 : \text{Box}$

$\text{Box}_3 : \text{ToyBox}$

$\text{Box}_4 : \text{Box} \sqcap \forall\, \text{temp}.\neg\text{TempClass}$

$\text{Box}_5 : \text{Box} \sqcap \exists\, \text{temp.TempClass}$



Reasoning

Is Box_4 a FruitBox?

Is Box_5 a FruitBox?

Are FruitBox and ToyBox disjoint?

[CADE-2021]: map to Fusemate disjunctive logic program + loop check

Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap \forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap \exists$ temp.TempClass

Description Logics, Event Calculus and Rules

- Description logics and logic programming are "very different"

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models

- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : $\text{Box} \sqcap \forall\, \text{temp}.\neg\text{TempClass}$
Box_5 : $\text{Box} \sqcap \exists\, \text{temp}.\text{TempClass}$

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$\text{Box}_0 : -10°$	$\text{Box}_2 : 10°$	$\text{Box}_0 : 2°$	$\text{Box}_0 : 20°$	

Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models

- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap$ $\forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap$ $\exists$ temp.TempClass

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$Box_0 : -10^\circ$	$Box_2 : 10^\circ$	$Box_0 : 2^\circ$	$Box_0 : 20^\circ$	

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

Description Logics, Event Calculus and Rules

- Description logics and logic programming are "very different"

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models

- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap \forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap \exists$ temp.TempClass

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$Box_0 : -10°$	$Box_2 : 10°$	$Box_0 : 2°$	$Box_0 : 20°$	

[DL]: Box2 is "High temp box" at t=20

[EC rules]: … and temp stays at 10° at t=30, 40, 50

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

Description Logics, Event Calculus and Rules

- Description logics and logic programming are "very different"

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models

- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap \forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap \exists$ temp.TempClass

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$Box_0 : -10°$	$Box_2 : 10°$	$Box_0 : 2°$	$Box_0 : 20°$	

Box0 (High)
Box1 (?)
Box2 (High)
Box3 (N/A)
Box4 (N/A)

[DL]: Box2 is "High temp box" at t=20

Cooling broken?

[EC rules]: … and temp stays at 10° at t=30, 40, 50

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

Description Logics, Event Calculus and Rules

- Description logics and logic programming are "very different"

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap \forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap \exists$ temp.TempClass

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$Box_0 : -10°$	$Box_2 : 10°$	$Box_0 : 2°$	$Box_0 : 20°$	

Box0 (High)
Box1 (?)
Box2 (High)
Box3 (N/A)
Box4 (N/A)

[DL]: Box2 is "High temp box" at t=20

Cooling broken?

[EC rules]: … and temp stays at 10° at t=30, 40, 50

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

```
ColdBox(time, box) :–
        IsAAt(time, x, Box),
        NOT (t < time, (I.aboxAt(t), tbox) |= IsA(x, Box), HasA(x, Temp, High))
```

|= means "provably" (not "consistently")

Description Logics, Event Calculus and Rules

- Description logics and logic programming are "very different"

  Open world vs closed world, Entailment vs Models, Infinite models vs finite models

- Attractive to integrate for modelling complementary aspects

Box_0 : FruitBox
Box_1 : FruitBox
Box_2 : Box
Box_3 : ToyBox
Box_4 : Box $\sqcap \forall$ temp.$\neg$TempClass
Box_5 : Box $\sqcap \exists$ temp.TempClass

Timed ABoxes

Time	10	20	30	40	50
Action	Load Box_0 Load Box_1	Load Box_2	Load Box_3 Load Box_4		Unload
Sensor	$Box_0 : -10°$	$Box_2 : 10°$	$Box_0 : 2°$	$Box_0 : 20°$	

Box0 (High)
Box1 (?)
Box2 (High)
Box3 (N/A)
Box4 (N/A)

[DL]: Box2 is "High temp box" at t=20

Cooling broken?

[EC rules]: … and temp stays at 10° at t=30, 40, 50

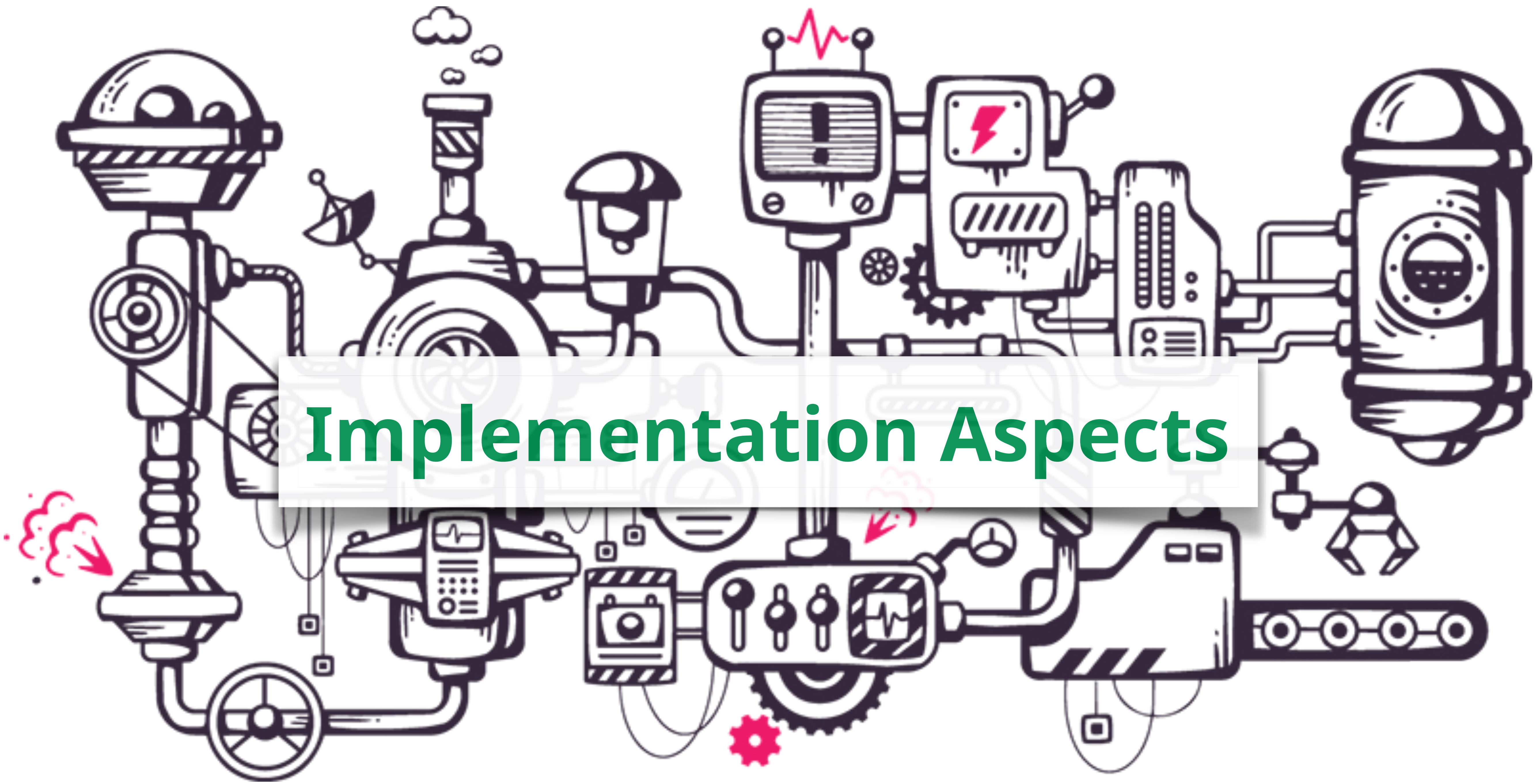
Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

Ramification problem

```
ColdBox(time, box) :-
        IsAAt(time, x, Box),
        NOT (t < time, (I.aboxAt(t), tbox) |= IsA(x, Box), HasA(x, Temp, High))
```

|= means "provably" (not "consistently")



Implementation Aspects

Embedding Into Scala: Translation

Input program ≈ Scala source code

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

Embedding Into Scala: Translation

Input program ≈ Scala source code

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

```
type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom

@rules
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) )
```

Embedding Into Scala: Translation

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

Input program ≈ Scala source code

```
type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom

@rules      ← Macro annotation
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) )
```

Embedding Into Scala: Translation

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

Input program ≈ Scala source code

```
type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom
```

Macro annotation

```
@rules
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) )
```

```
case List(In(time, obj, c), In(time0, c1, cont))
    if c == c1 && time == time0
  => In(time, obj, cont)
```

Macro expansion into partial function

Embedding Into Scala: Translation

Input program ≈ Scala source code

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

```
type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom
```

Macro annotation

```
@rules
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) )
```

```
case List(In(time, obj, c), In(time0, c1, cont))
     if c == c1 && time == time0
  => In(time, obj, cont)
```

Macro expansion into partial function

+ given-clause loop operating on such rules-as-partial-functions

(In reality the macro expansion is more complicated because of default negation)

Embedding into Scala: Method

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
     …
    (b ∋ obj) && (b ∋ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

“Natural” integration into Scala and vice versa

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
     …
    (b ∋ obj) && (b ∋ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

“Natural” integration into Scala and vice versa

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
      …
      (b ∋ obj) && (b ∋ o),
      where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

"Natural" integration into Scala and vice versa

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", …)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
     …
    (b ∋ obj) && (b ∋ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

“Natural” integration into Scala and vice versa

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
      …
      (b ∋ obj) && (b ∋ o),
      where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

"Natural" integration into Scala and vice versa

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", …)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
      …
      (b ∋ obj) && (b ∋ o),
      where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

"Natural" integration into Scala and vice versa

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", …)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
      …
      (b ∋ obj) && (b ∋ o),
      where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

“Natural” integration into Scala and vice versa

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :-
      …
      (b ∋ obj) && (b ∋ o),
      where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Method

“Natural” integration into Scala and vice versa

```
val eventsCSV = List(“Load,10,tomatoes,pallet",“Load,20,pallet,container", …)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    …
  }
} saturate { @rules …
  fail(…) :−
     …
    (b ∋ obj) && (b ∋ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    …
  }
}
```

```
def sameBatch(time: Time) =
  if (time == 10) Set("tomatoes", "apples") else Set.∅[String]
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

Embedding into Scala: Discussion

Two-way calling interface

- Scala -> Rules calls trivial
- Rules -> Scala calls trivial

Data structures integration is trivial

- Use any Scala data structure in rules
- Logic data structures (models) are Scala data structures
- Unmatched aggregation and introspection capabilities

Disadvantage

- Must rely on Scala pattern matching implementation
- Difficult to implement efficiently

Embedding into Scala: Discussion

Two-way calling interface

- Scala -> Rules calls trivial
- Rules -> Scala calls trivial

Data structures integration is trivial

- Use any Scala data structure in rules
- Logic data structures (models) are Scala data structures
- Unmatched aggregation and introspection capabilities

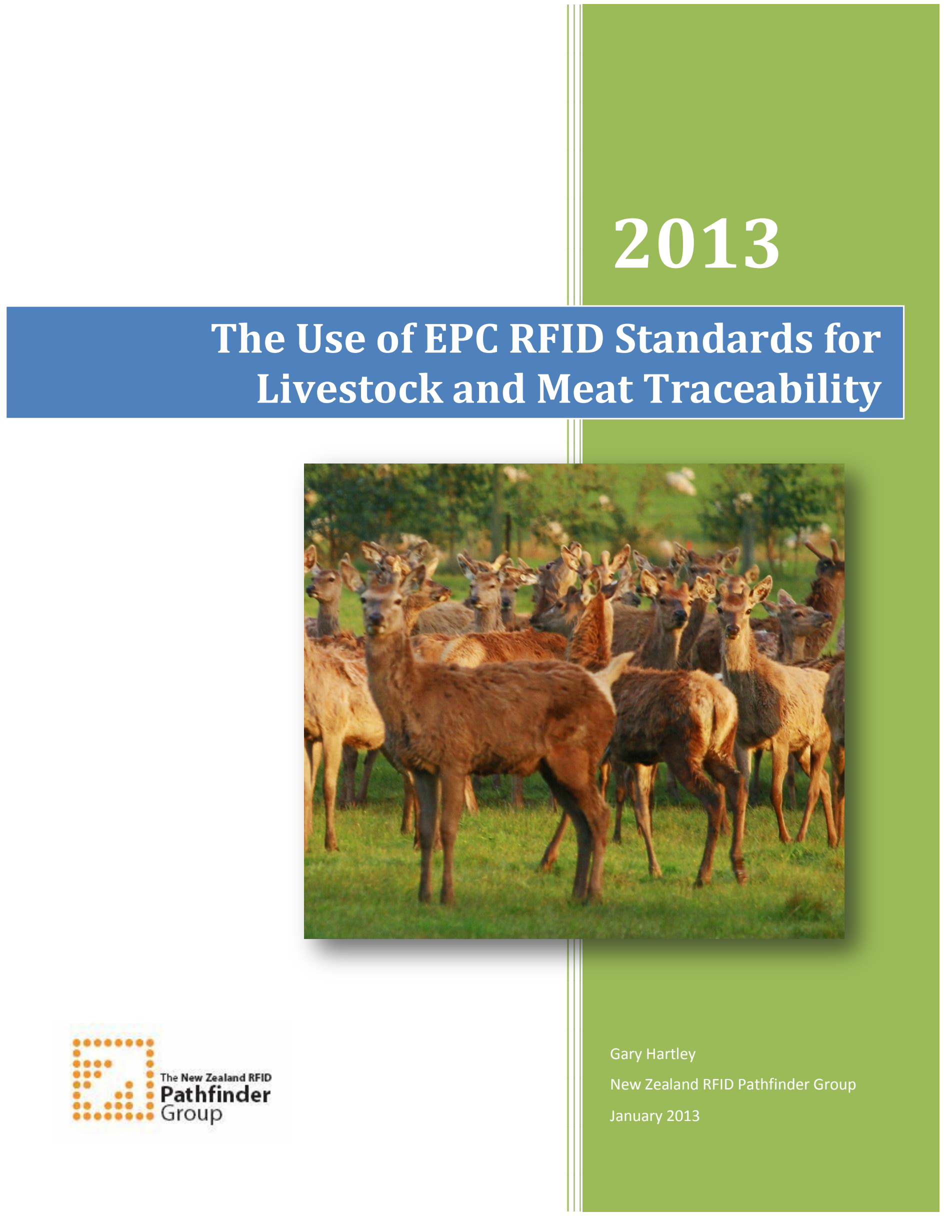
Disadvantage

- Must rely on Scala pattern matching implementation
- Difficult to implement efficiently

- Tighter coupling than in every other system (I know of)
- Adds "interpretations" as a container data structure to functional/OO programming with "logic programming" as an operator

Three and a Half Case Studies

Case Study 1 - Deer Supply Chain



2013

The Use of EPC RFID Standards for Livestock and Meat Traceability

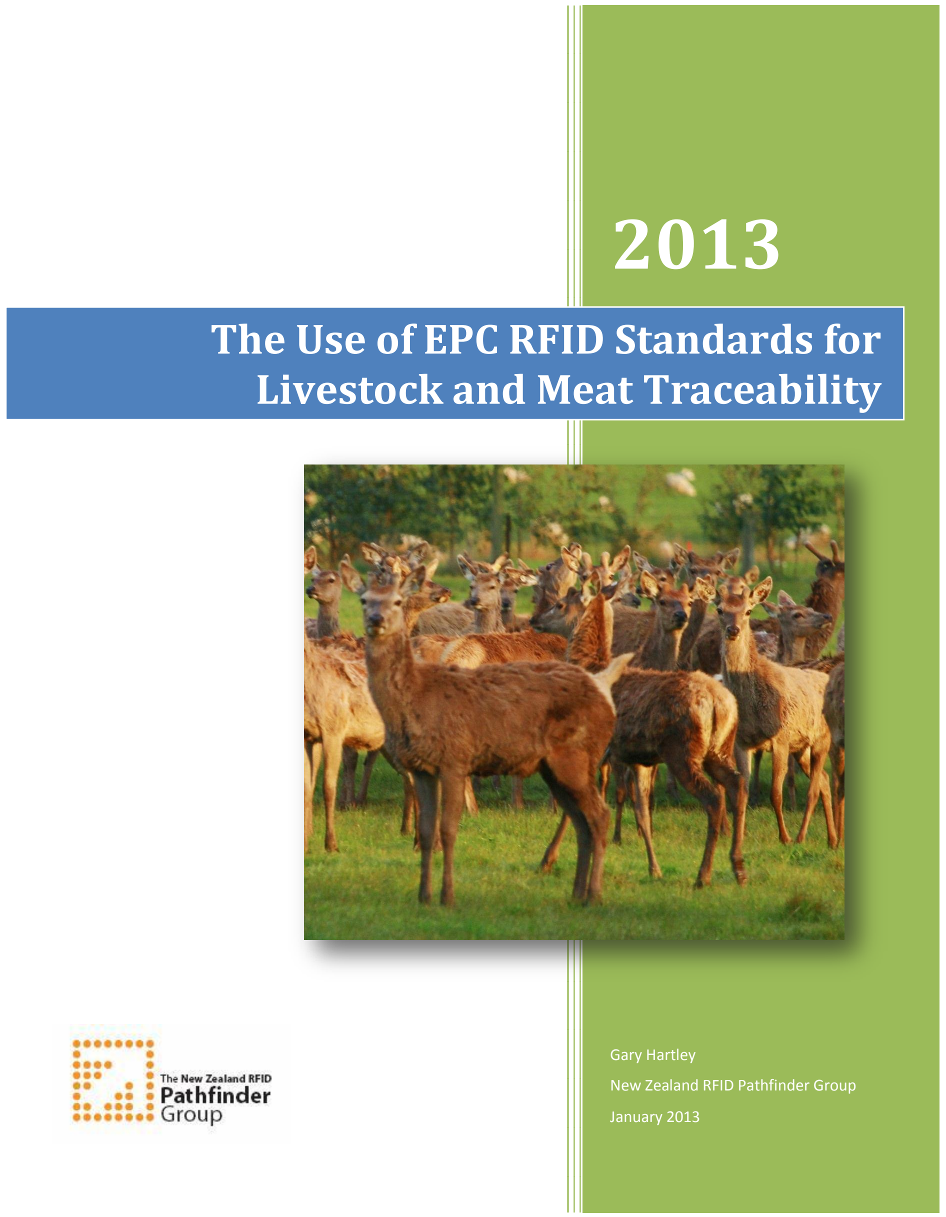
Gary Hartley

New Zealand RFID Pathfinder Group

January 2013

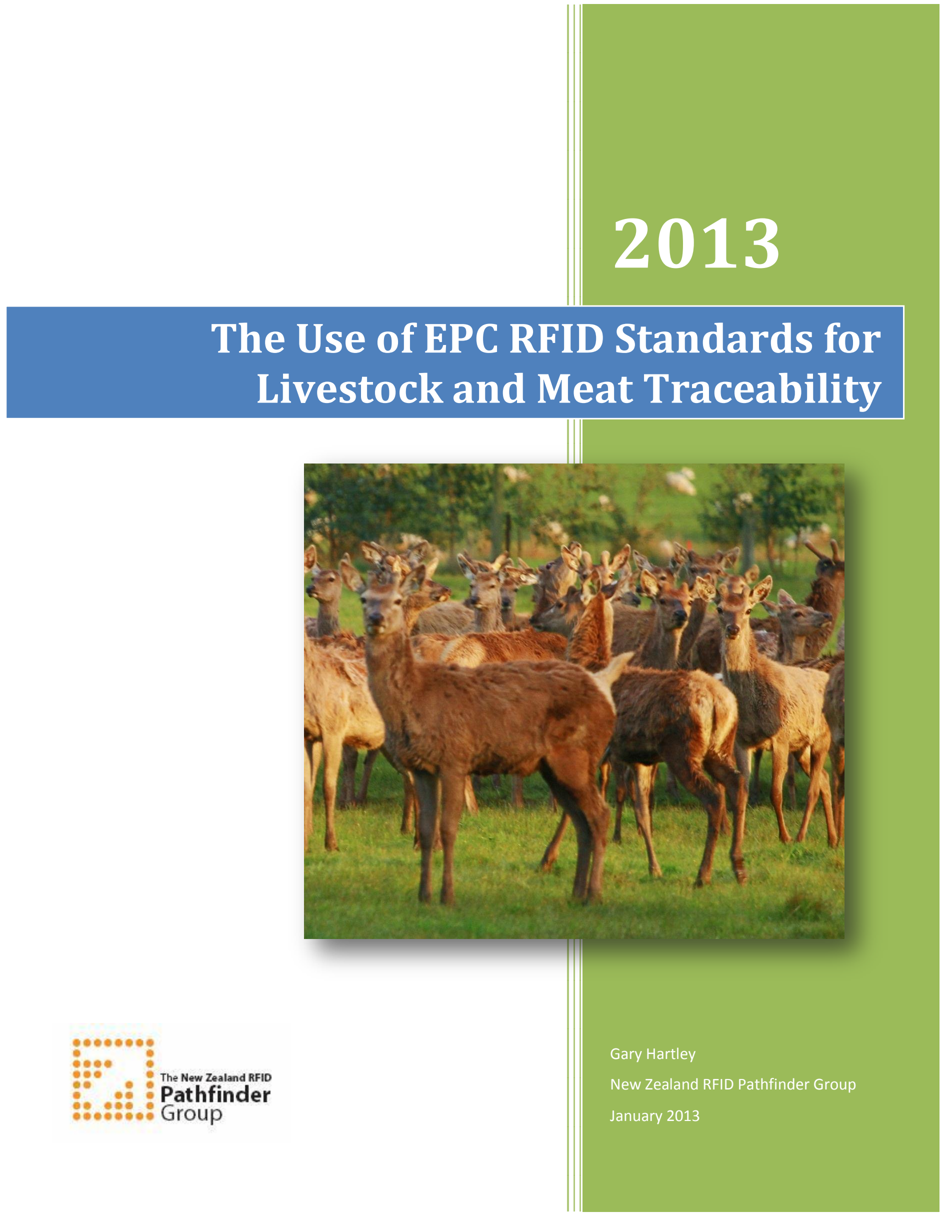
The New Zealand RFID Pathfinder Group

Case Study 1 - Deer Supply Chain



Events: from farm (NZ) to retailer (DE) encoded in EPCIS

Case Study 1 - Deer Supply Chain


2013

The Use of EPC RFID Standards for Livestock and Meat Traceability

The New Zealand RFID Pathfinder Group

Gary Hartley

New Zealand RFID Pathfinder Group

January 2013


Process Step 4 - Animals arrive at Mountain River Processors' stun box

Figure 5.7 - Stun Box

Figure 5.8 - RFID reader at Stun Box

Figure 5.7 illustrates animals in the location of the stun box. Note the RFID ear tags in the ears of the animals. Figure 5.8 illustrates the RFID antenna setup at the stun box.

Process Step 5 - Cartons of finished Venison cuts packed into cartons at Mountain River processor and moved from the boning room into chiller room

Figure 5.9 - UHF RFID tags used on cartons

Figure 5.10 - UHF RFID tags positioned on cartons

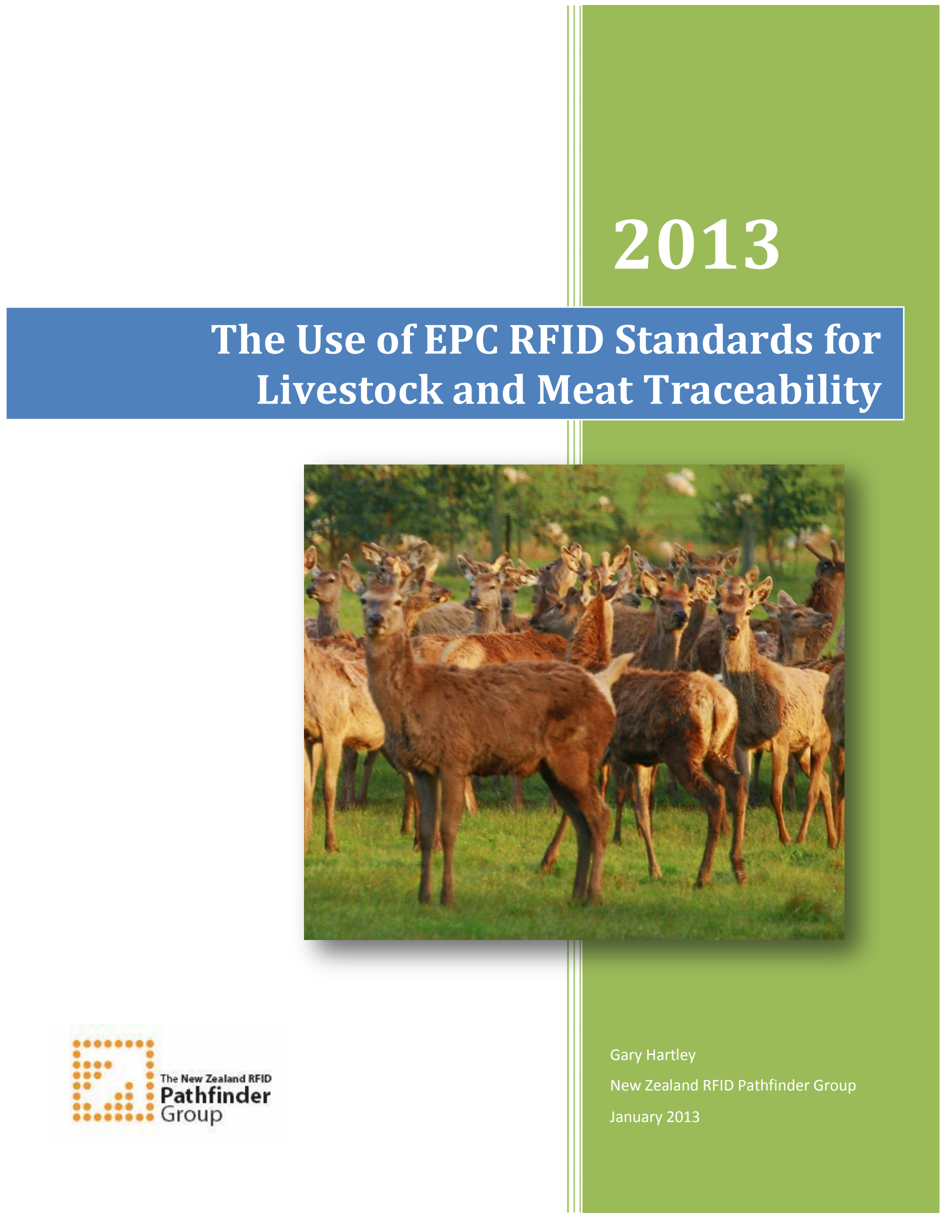
Figure 5.11 – Tagged cartons moving from boning room to chiller room

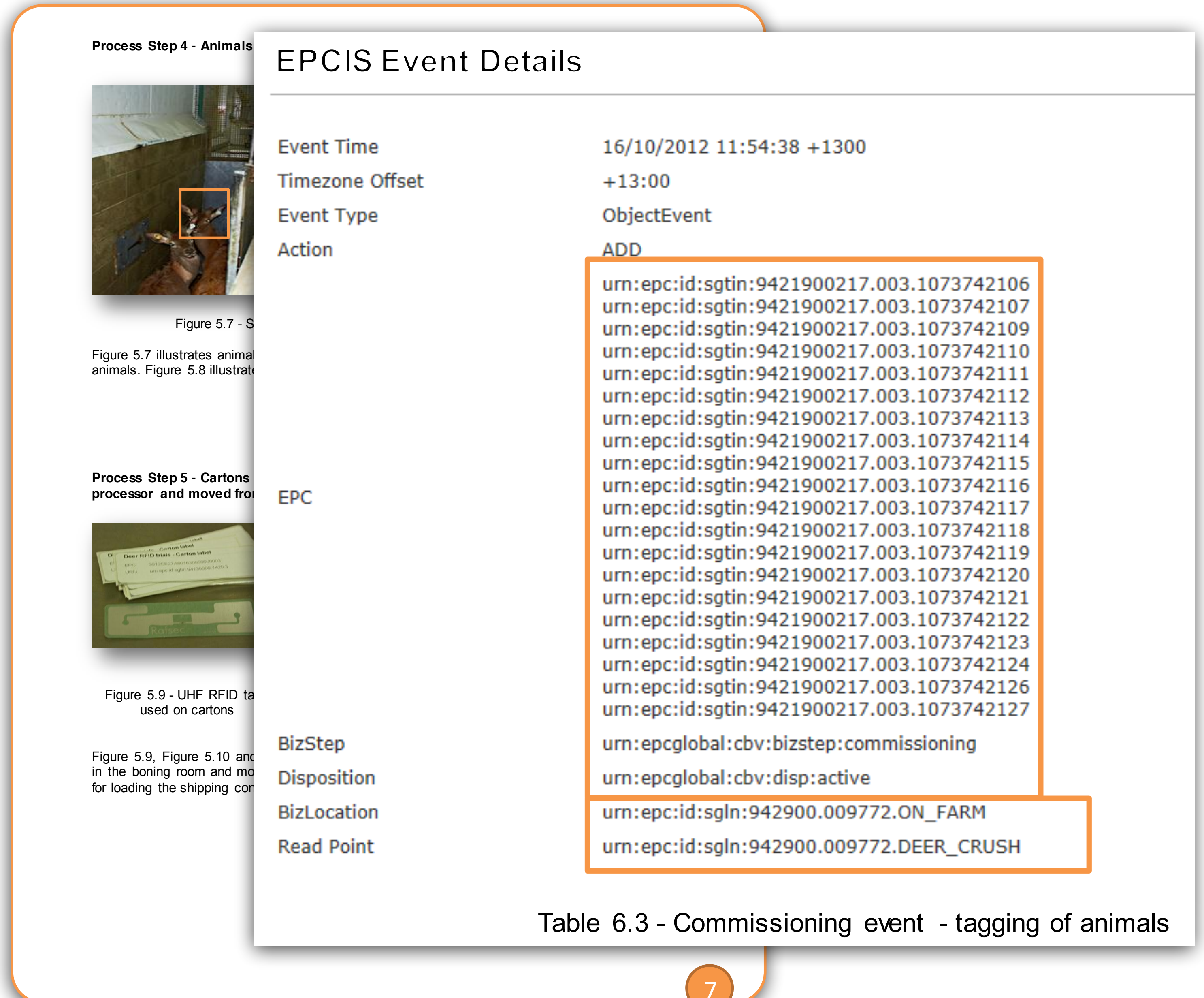
Figure 5.9, Figure 5.10 and Figure 5.11 illustrate the affixing of EPC UHF RFID tags on the cartons in the boning room and moving of cartons of finished venison cuts into the chiller room in preparation for loading the shipping container.

7

Events: from farm (NZ) to retailer (DE) encoded in EPCIS

Case Study 1 - Deer Supply Chain





EPCIS Event Details

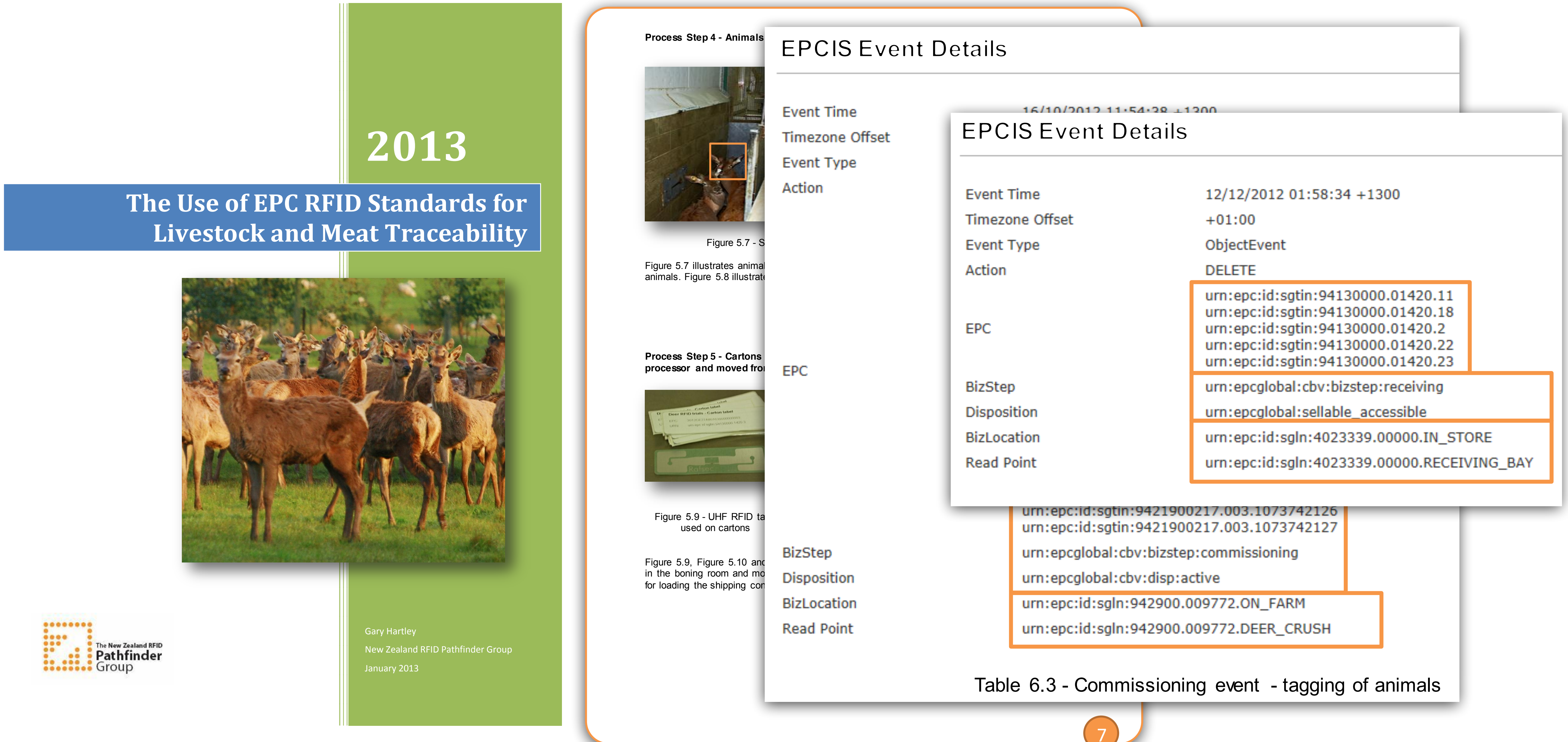
Event Time	16/10/2012 11:54:38 +1300
Timezone Offset	+13:00
Event Type	ObjectEvent
Action	ADD
EPC	urn:epc:id:sgtin:9421900217.003.1073742106 urn:epc:id:sgtin:9421900217.003.1073742107 urn:epc:id:sgtin:9421900217.003.1073742109 urn:epc:id:sgtin:9421900217.003.1073742110 urn:epc:id:sgtin:9421900217.003.1073742111 urn:epc:id:sgtin:9421900217.003.1073742112 urn:epc:id:sgtin:9421900217.003.1073742113 urn:epc:id:sgtin:9421900217.003.1073742114 urn:epc:id:sgtin:9421900217.003.1073742115 urn:epc:id:sgtin:9421900217.003.1073742116 urn:epc:id:sgtin:9421900217.003.1073742117 urn:epc:id:sgtin:9421900217.003.1073742118 urn:epc:id:sgtin:9421900217.003.1073742119 urn:epc:id:sgtin:9421900217.003.1073742120 urn:epc:id:sgtin:9421900217.003.1073742121 urn:epc:id:sgtin:9421900217.003.1073742122 urn:epc:id:sgtin:9421900217.003.1073742123 urn:epc:id:sgtin:9421900217.003.1073742124 urn:epc:id:sgtin:9421900217.003.1073742126 urn:epc:id:sgtin:9421900217.003.1073742127
BizStep	urn:epcglobal:cbv:bizstep:commissioning
Disposition	urn:epcglobal:cbv:disp:active
BizLocation	urn:epc:id:sgln:942900.009772.ON_FARM
Read Point	urn:epc:id:sgln:942900.009772.DEER_CRUSH

Table 6.3 - Commissioning event - tagging of animals

What?
Where?
When?
Why?

Events: from farm (NZ) to retailer (DE) encoded in EPCIS

Case Study 1 - Deer Supply Chain

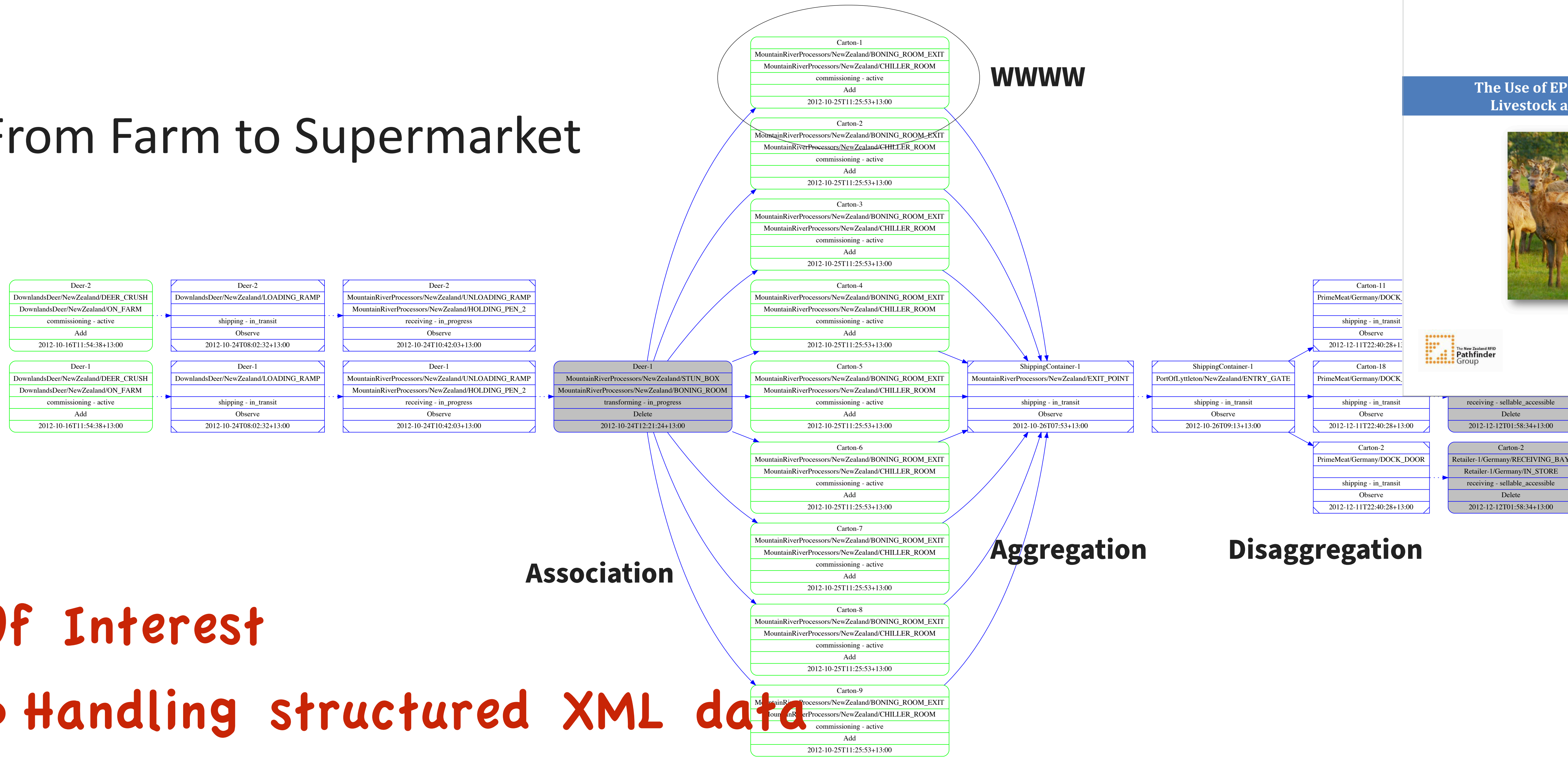


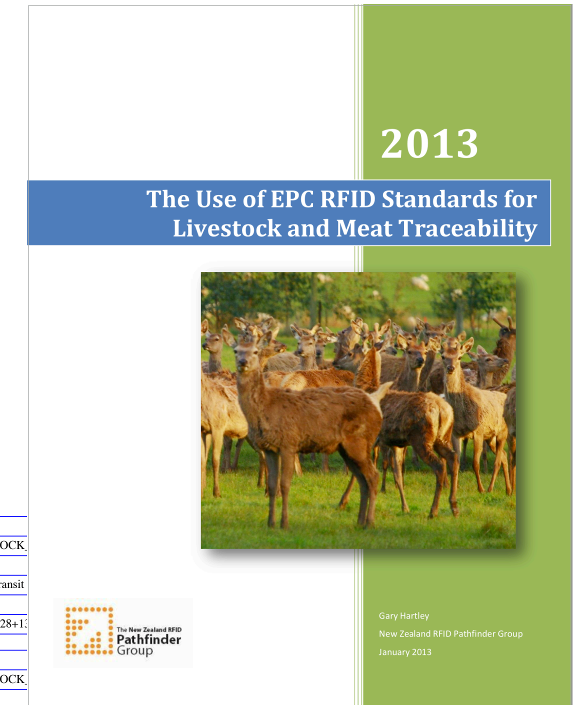
What?
Where?
When?
Why?

Events: from farm (NZ) to retailer (DE) encoded in EPCIS

Case Study 1 - Deer Supply Chain

From Farm to Supermarket



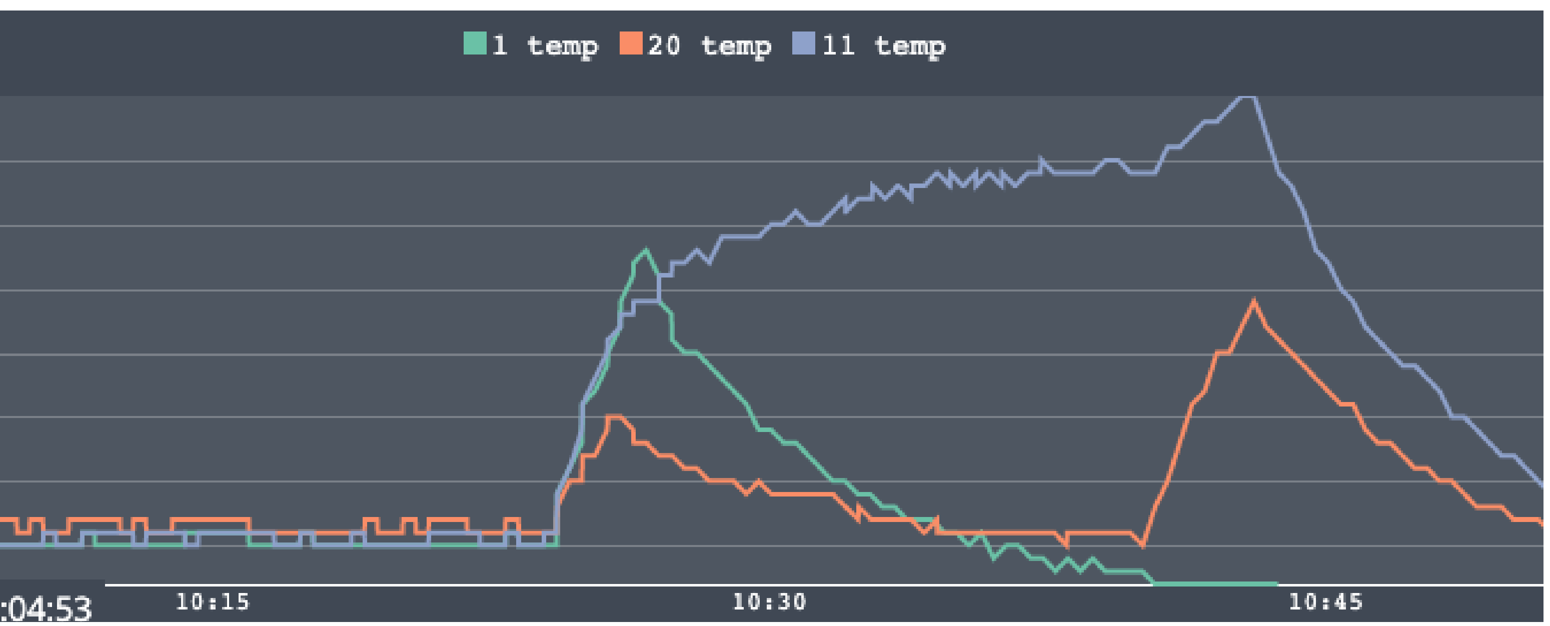


Of Interest

- Handling structured XML data
- Speculating whereabout of missing item
  - A box enters supply chain but does not arrive at destination
  - Track same batch boxes as proxies

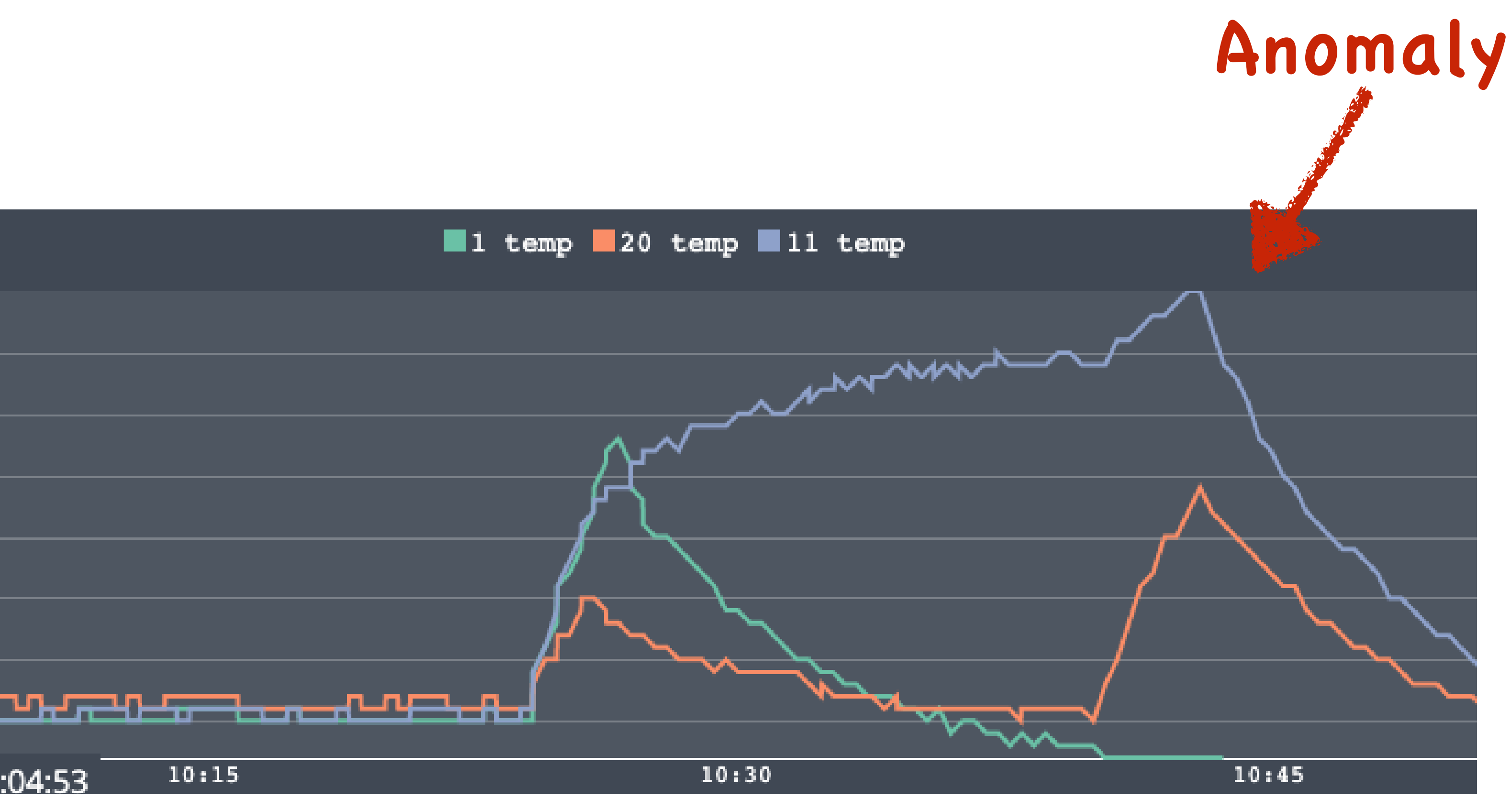
Case Study 2 - D61 Project “Supply Chain Awareness”

- Partner company BeefLedger ships boxed meat products
- Stringent cooling requirements ensure quality of products
- D61 sensors measure box temperatures
  (S. Khalifa / K. v. Richter)
- Task: Pricing model, anomaly detection



Case Study 2 - D61 Project "Supply Chain Awareness"

- Partner company BeefLedger ships boxed meat products
- Stringent cooling requirements ensure quality of products
- D61 sensors measure box temperatures
  (S. Khalifa / K. v. Richter)
- Task: Pricing model, anomaly detection

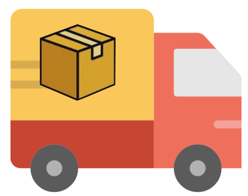


Case Study 2 - D61 Project “Supply Chain Awareness”

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

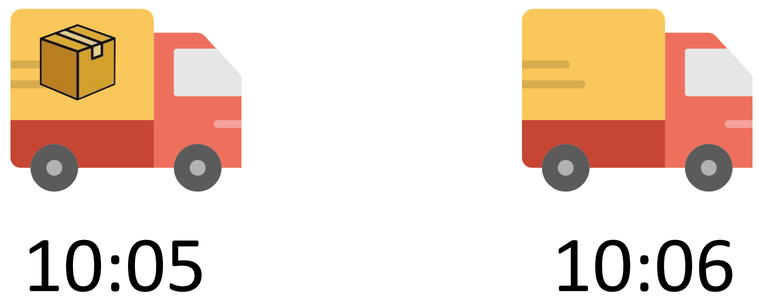


10:05

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

Case Study 2 - D61 Project "Supply Chain Awareness"

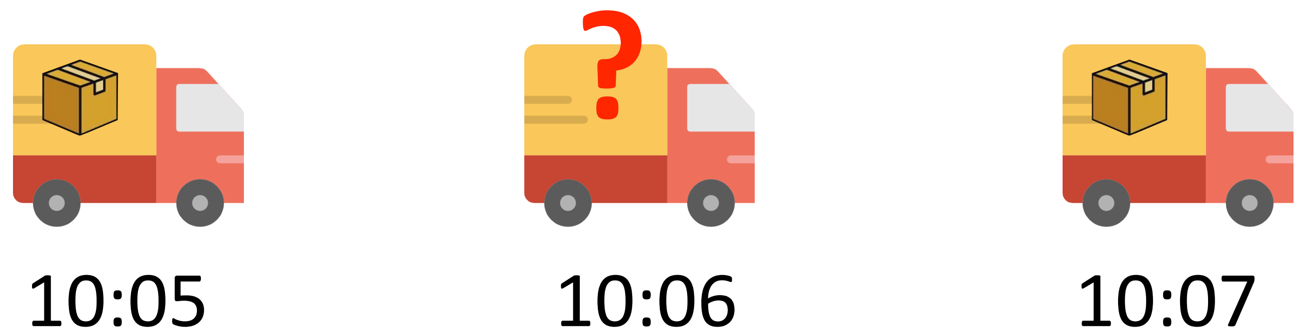
Fix sensor dropouts, anomalies

Fix GPS dropouts

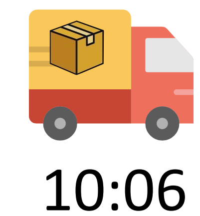
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



Fix:



Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

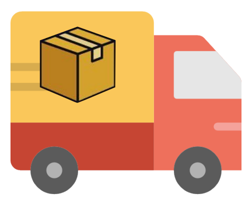

```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

Fix:

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts


10:05


10:06


10:07

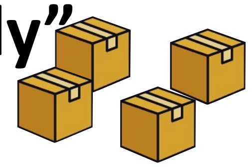
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev[illegible]
  SECONDS.between(prev, time) < 3 * S[illegible]e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

Fix:


10:06

“Behaves differently”

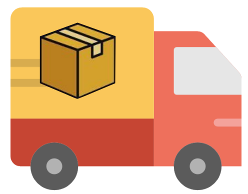
Anomaly



Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



10:05

10:06

10:07

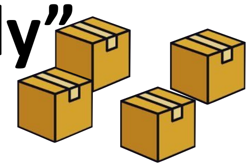
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

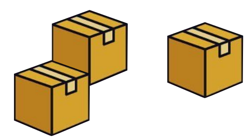
Fix:

10:06

“Behaves differently”

Anomaly

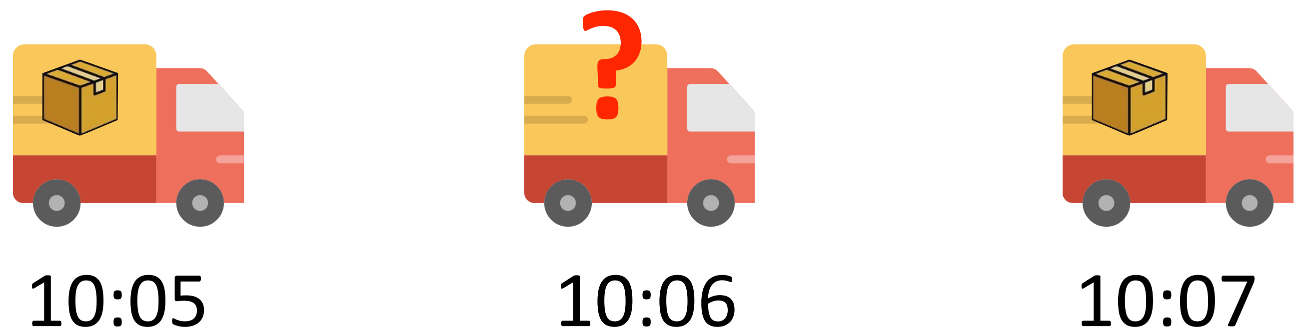




Case Study 2 - D61 Project “Supply Chain Awareness”

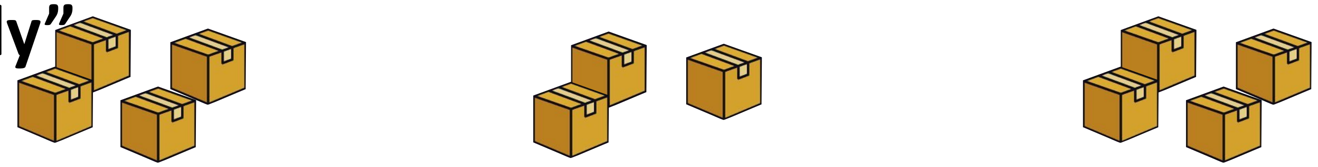
Fix sensor dropouts, anomalies

Fix GPS dropouts



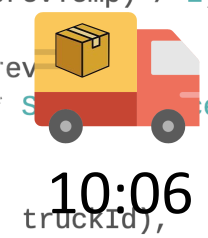
“Behaves differently”

Anomaly



```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S[illegible]e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

Fix:



Case Study 2 - D61 Project “Supply Chain Awareness”

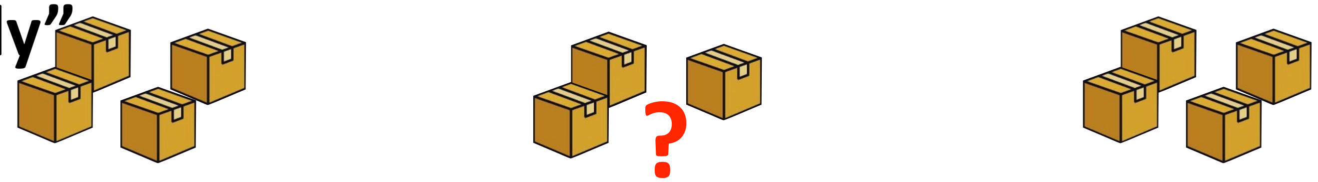
Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05 10:06 10:07

“Behaves differently”

Anomaly



```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S                e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

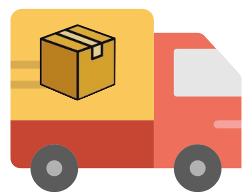
Fix:

10:06

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



10:05

10:06

10:07

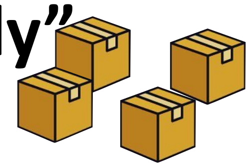
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S                e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

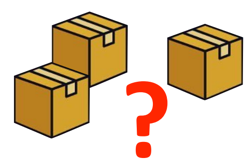
Fix:

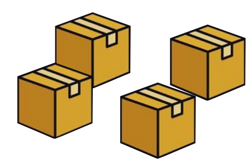
10:06

“Behaves differently”

Anomaly





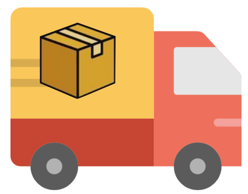


Box moved to cabin?

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



10:05

10:06

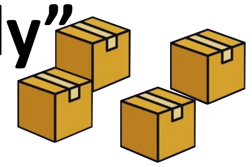
10:07

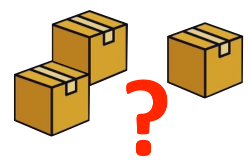
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

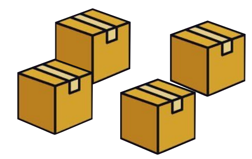
Fix:

10:06

“Behaves differently” Anomaly







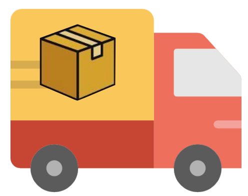
Box moved to cabin?

“Is different” Anomaly

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



10:05

10:06

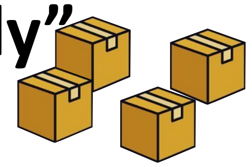
10:07

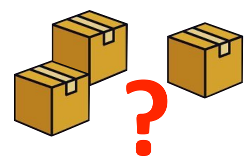
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

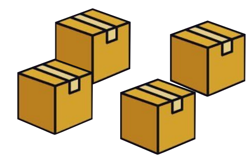
Fix:

10:06

“Behaves differently” Anomaly

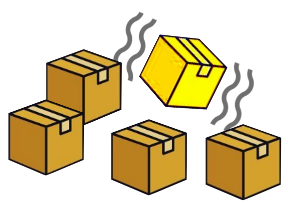






Box moved to cabin?

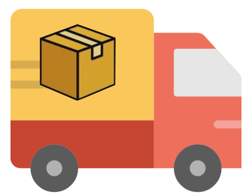
“Is different” Anomaly



Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts


10:05


10:06

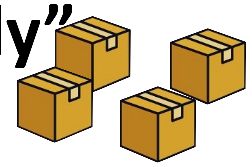

10:07

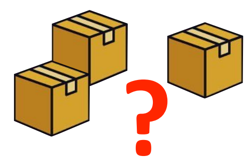
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

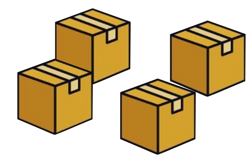
Fix:


10:06

“Behaves differently” Anomaly

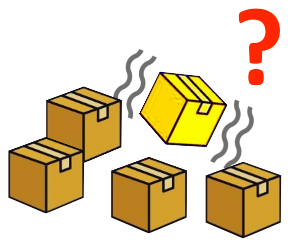






Box moved to cabin?

“Is different” Anomaly

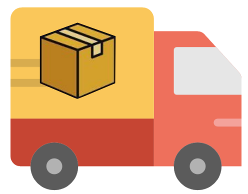


Worth checking?

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts



10:05

10:06

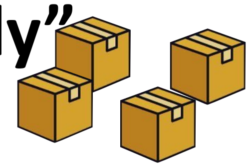
10:07

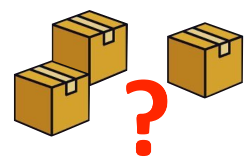
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S          e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

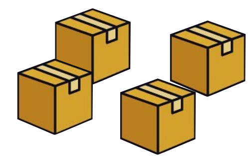
Fix:

10:06

“Behaves differently” Anomaly

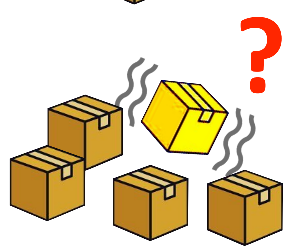






Box moved to cabin?

“Is different” Anomaly



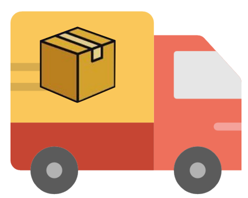
Worth checking?

Clustering based on similarity measure for feature vector

Case Study 2 - D61 Project "Supply Chain Awareness"

Fix sensor dropouts, anomalies

Fix GPS dropouts


10:05


10:06

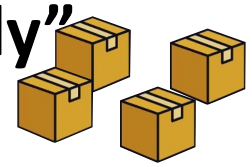

10:07

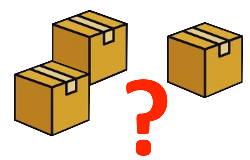
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, time) < 3 * S                e,
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

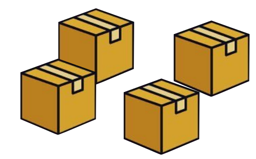
Fix:


10:06

"Behaves differently" Anomaly

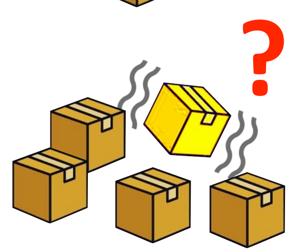






Box moved to cabin?

"Is different" Anomaly

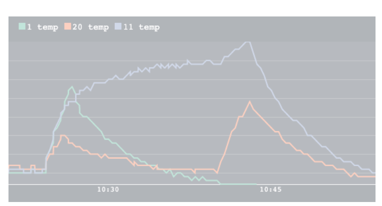


Worth checking?

Clustering based on similarity measure for feature vector

Cooling OK?

Pricing?


Actual

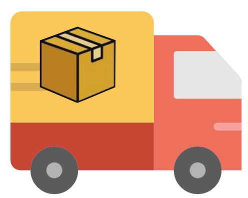
vs


Expected

Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts


10:05


10:06

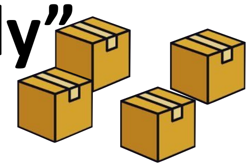

10:07

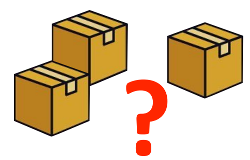
```
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, _, id, prev
  SECONDS.between(prev, ti
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
```

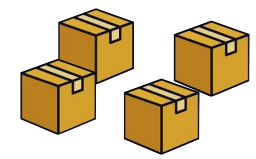
Fix:


10:06

“Behaves differently”
Anomaly

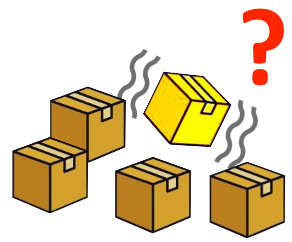






Box moved to cabin?

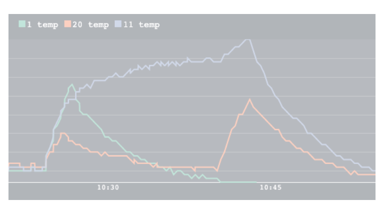
“Is different”
Anomaly



Worth checking?

Clustering based on similarity measure for feature vector

Cooling OK?
Pricing?


Actual

vs


Expected

Concrete scenarios:
normal, latecool,
missingbox, cabinbox

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
      BoxAtCoord(time, at, id, temp),
      BoxAtCoord(prev < time, _, id, prevTemp) STH
          SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
      BoxOnTruck(prev, id),
      BoxOnTruck(time, id),
      TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
      NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

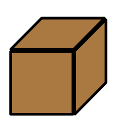
Time
Loc

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
      BoxAtCoord(time, at, id, temp),
   ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
          SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
      BoxOnTruck(prev, id),
      BoxOnTruck(time, id),
      TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
      NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

2°C



Time 10
Loc A

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
        SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
     BoxOnTruck(prev, id),
     BoxOnTruck(time, id),
     TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
     NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
         SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
    BoxOnTruck(prev, id),
    BoxOnTruck(time, id),
    TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
    NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
 ❷ BoxAtCoord(time, at, id, temp),
 ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
        SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
 ❸ BoxOnTruck(prev, id),
    BoxOnTruck(time, id),
    TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
    NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

Case Study 2 - D61 Project “Supply Chain Awareness”

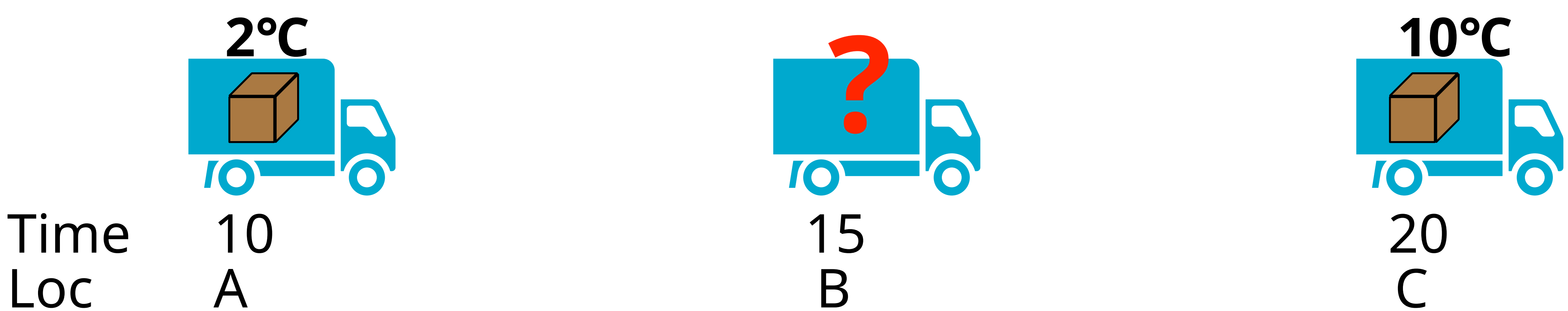
Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
        SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
  ❸ BoxOnTruck(prev, id),
  ❹ BoxOnTruck(time, id),
     TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
     NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
        SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
  ❸ BoxOnTruck(prev, id),
  ❹ BoxOnTruck(time, id),
  ❺ TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
     NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```



Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

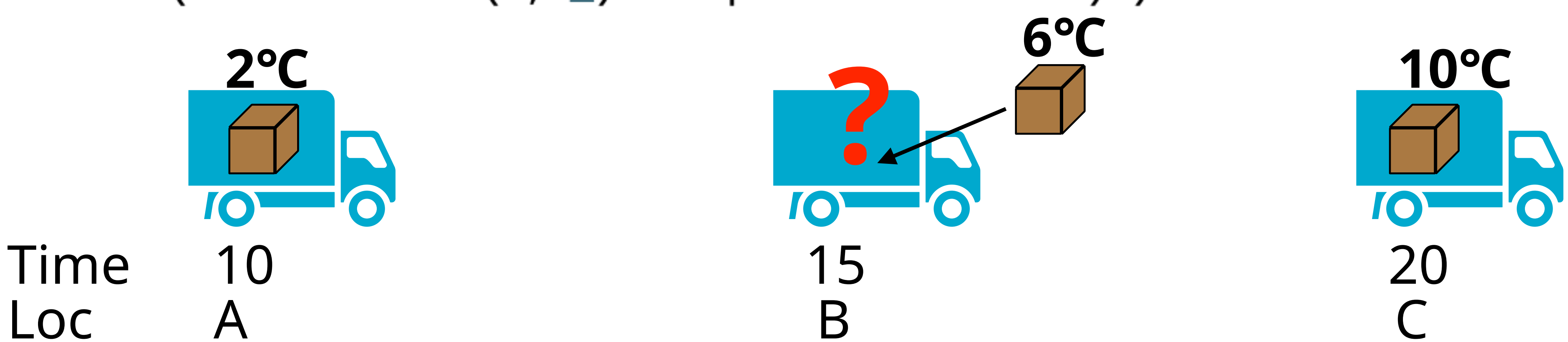
```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
  ❻     SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
  ❸ BoxOnTruck(prev, id),
  ❹ BoxOnTruck(time, id),
  ❺ TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
     NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

Time	10	15	20
Loc	A	B	C

Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
(7) FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
      (2) BoxAtCoord(time, at, id, temp),
      (1) BoxAtCoord(prev < time, _, id, prevTemp) STH
      (6)     SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
      (3) BoxOnTruck(prev, id),
      (4) BoxOnTruck(time, id),
      (5) TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
          NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

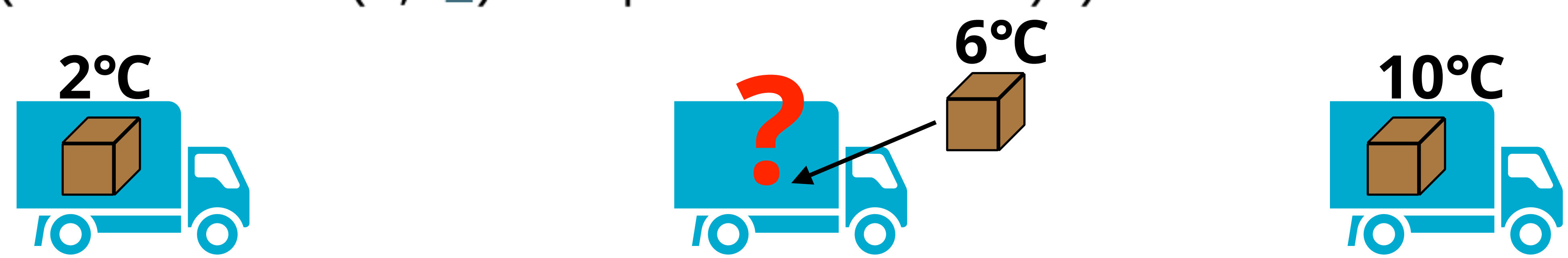


Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

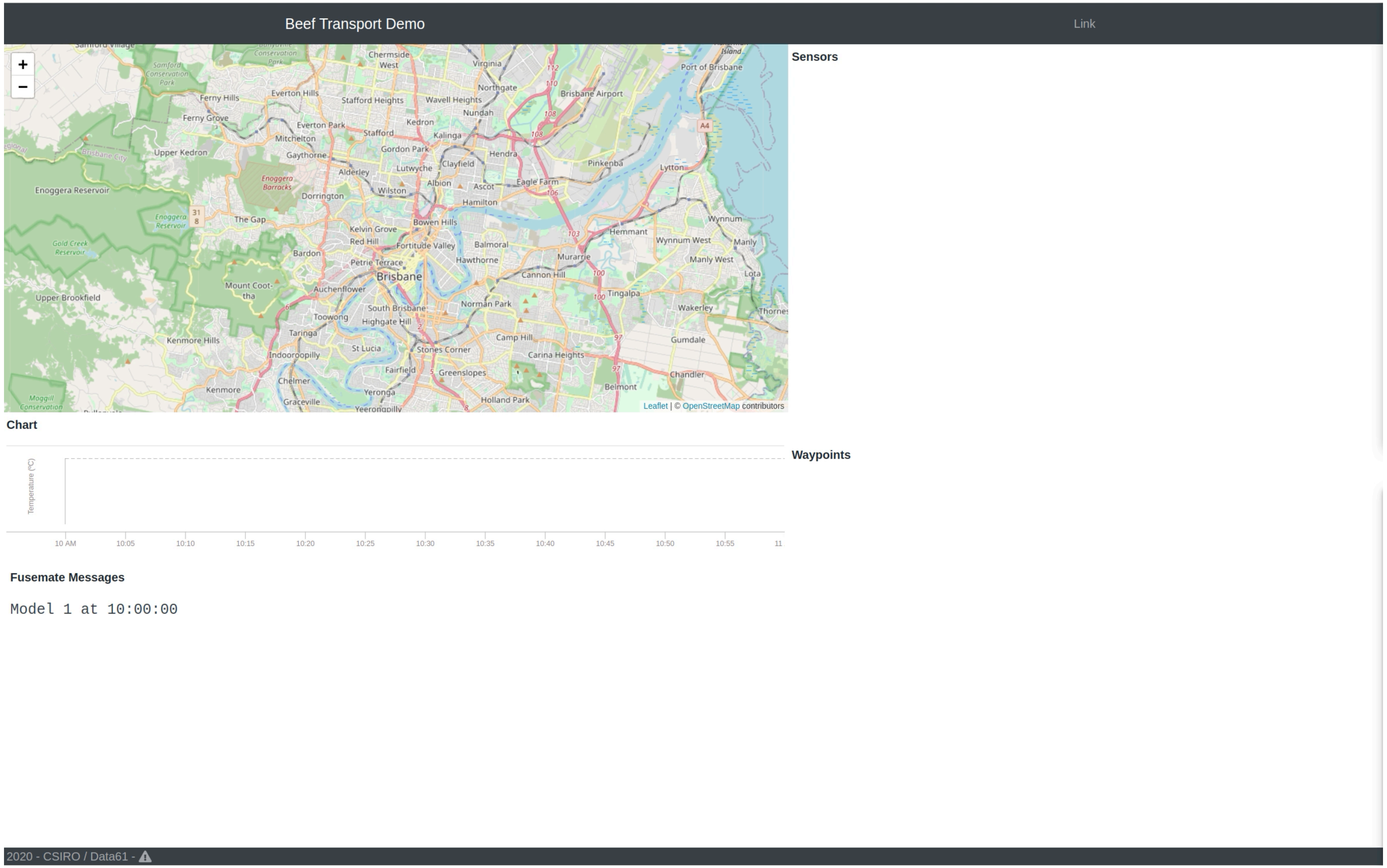
```
❼ FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (
  ❷ BoxAtCoord(time, at, id, temp),
  ❶ BoxAtCoord(prev < time, _, id, prevTemp) STH
  ❻     SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
  ❸ BoxOnTruck(prev, id),
  ❹ BoxOnTruck(time, id),
  ❺ TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
    NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t) )
```

- Similar rule for truck location recovery
- 25 rules altogether



Time	10	15	20
Loc	A	B	C

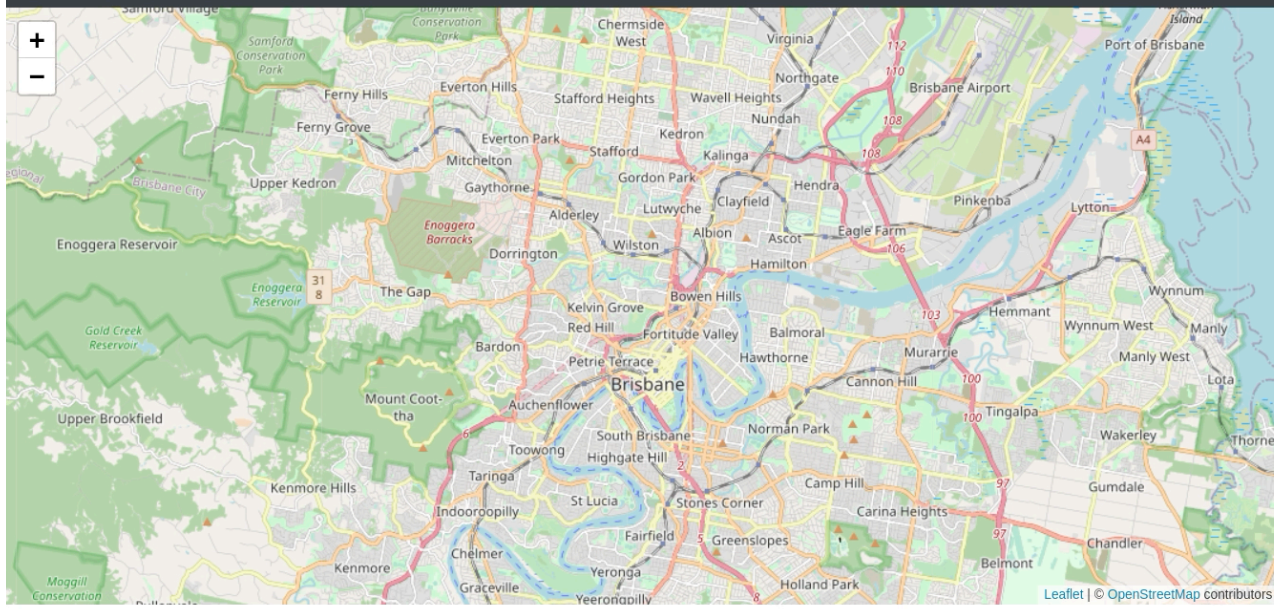
Fusemate System Demo



Fusemate System Demo

Beef Transport Demo

Link



Sensors

Chart

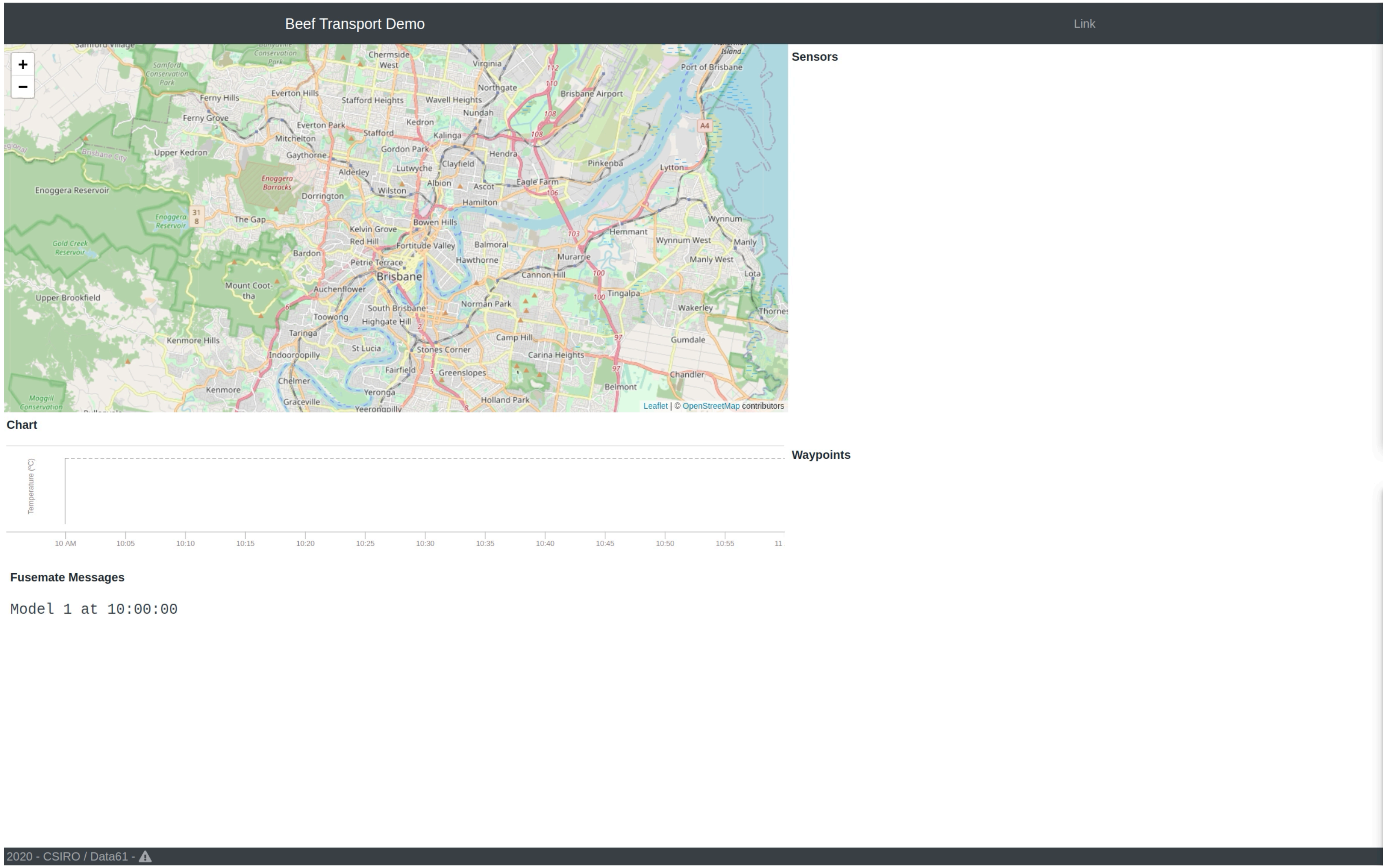
Waypoints

Fusemate Messages

```
Model 1 at 10:00:00
```

2020 - CSIRO / Data61 -

Fusemate System Demo



Of Interest

- GPS -> Symbolic Loc
- Integrating information sources
- Noisy sensor data
- Robust anomaly detection

Case Study 3 - Taxi Rides Anomalies

2 Million taxi rides in New York City

`Ride(taxi,license,from,to,start,end,fare)`

Ride

Gap (between rides)

Pickup/dropoff clusters

Fusemate

(1) Rules for hotspot clustering and concave hull

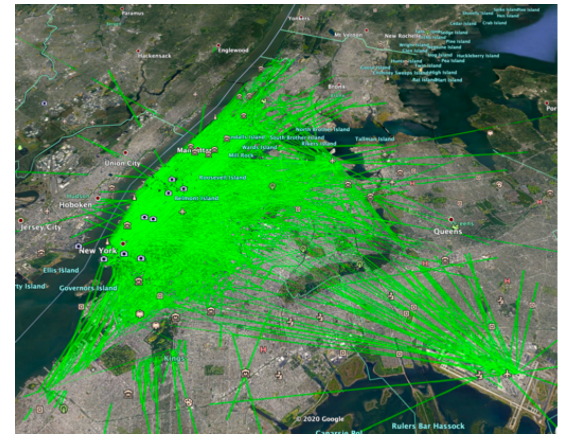
(2) Rules for anomaly detection

Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,     _, _, from, _,       _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

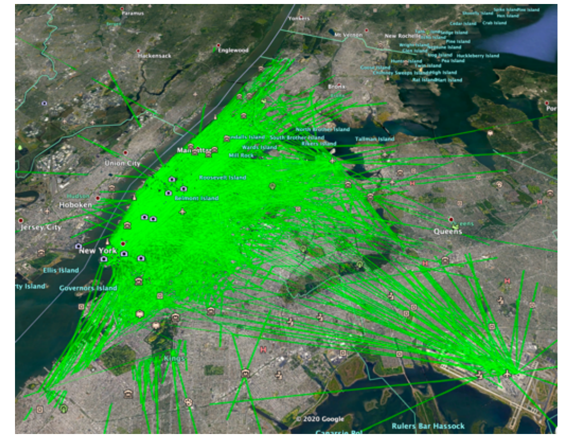
Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,     _, _, from, _,       _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

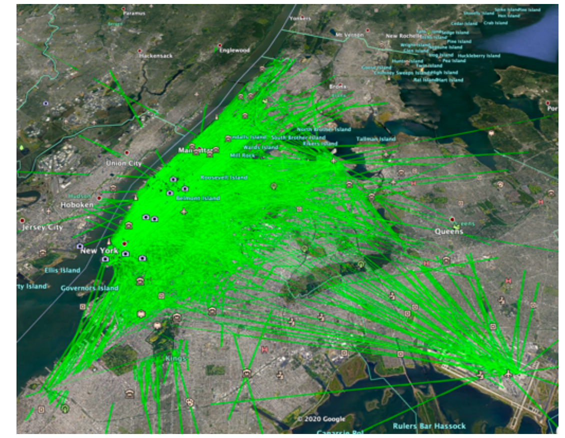
Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,      _, _, from, _,       _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

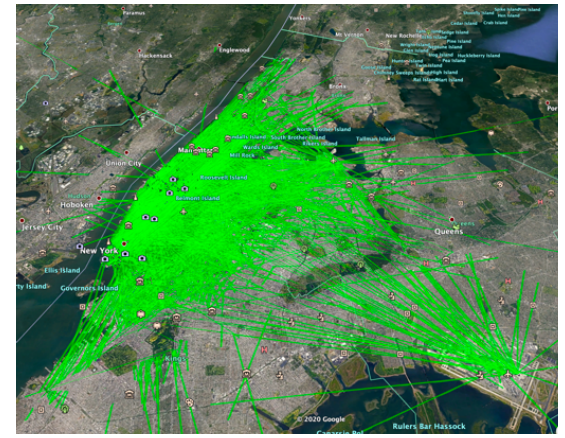
Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,     _, _, from, _,       _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

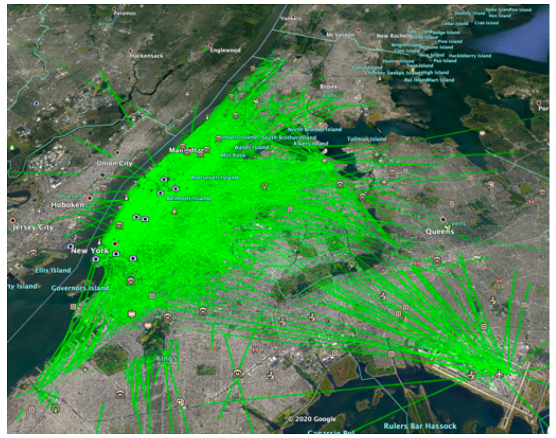
Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,     _, _, from, _,       _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

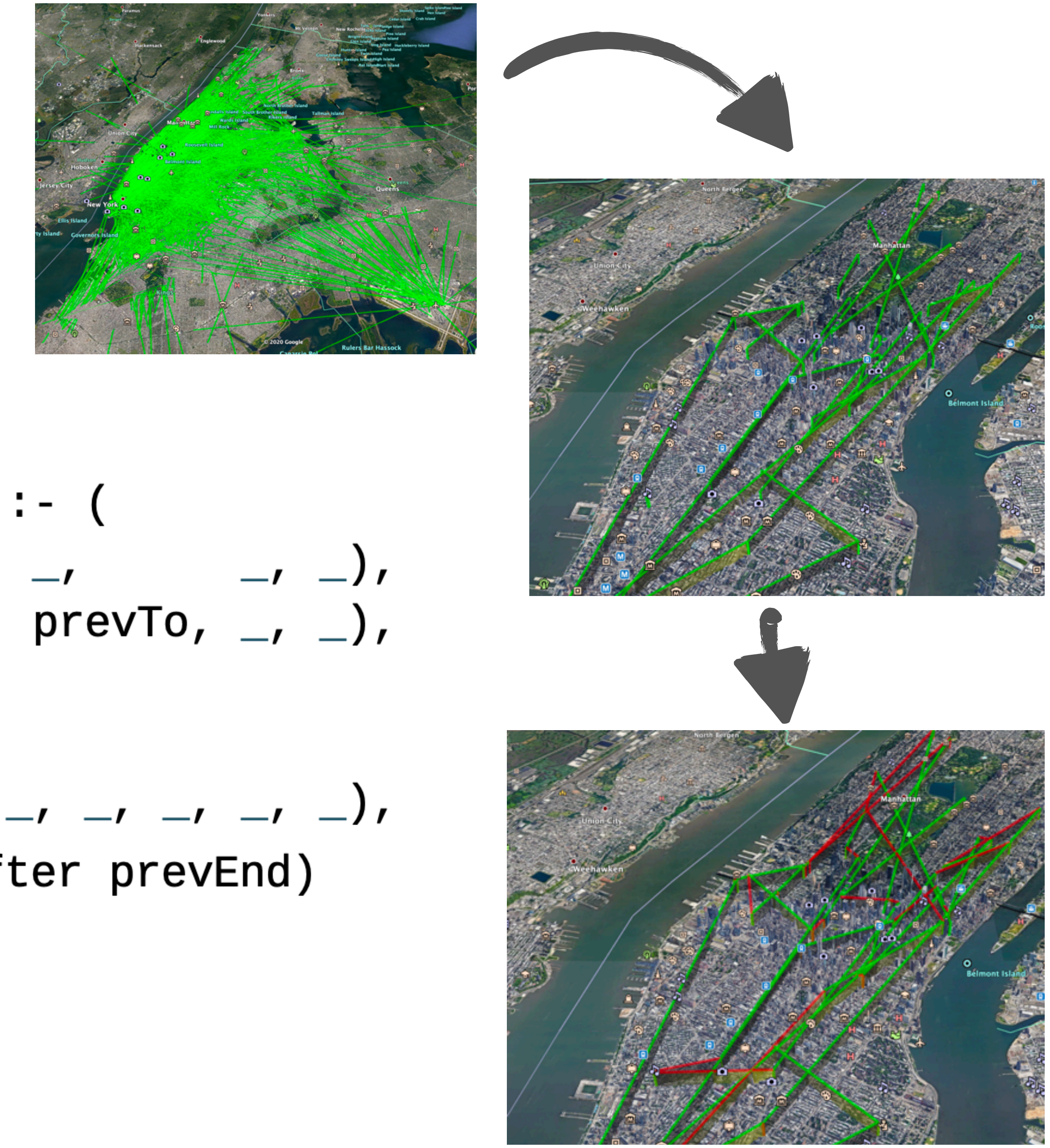
Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,      _, _, from, _,       _, _),
      Ride(taxi, license, _,     prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

Case Study 3 - Taxi Rides Anomalies

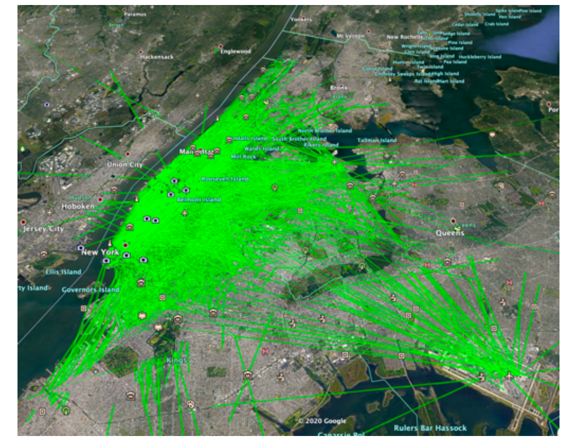


From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,      _, _, from, _,        _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,     prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

Fusemate invocation

Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,      _, _, from, _,        _, _),
      Ride(taxi, license, _,     prevEnd, _, _, _,     prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

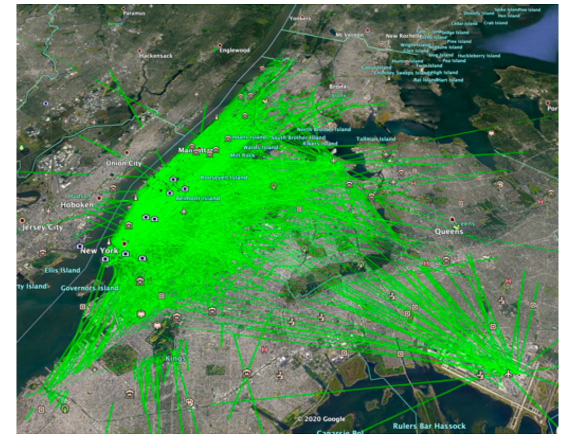
Fusemate invocation

Case Study 3 - Taxi Rides Anomalies

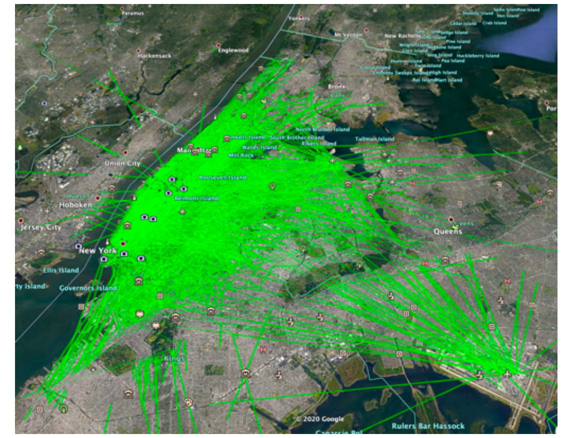
From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,      _, _, from, _,        _, _),
      Ride(taxi, license, _,      prevEnd, _, _, _,     prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

Fusemate invocation



Case Study 3 - Taxi Rides Anomalies



From Scala to Fusemate and back

```
val gaps42 = rides filter {
    _.license ≡ "42"
  } saturateFirst {
    Gap(taxi, license, prevEnd, start, prevTo, from) :- (
      Ride(taxi, license, start, end,     _, _, from, _,       _, _),
      Ride(taxi, license, _,     prevEnd, _, _, _,    prevTo, _, _),
      start isAfter prevEnd,
      NOT (
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
      )
    ) } collect {
    case g:Gap ⇒ g
  }
```

Fusemate invocation

Functional + Logic programming

Declarative and concise :)

Case Study 3 - Taxi Rides Anomalies

Anomaly: gap at a busy pickup hotspot

```
=====
driver license-3568
=====
taxi-3568 license-3568 2013-01-01T22:10 2013-01-01T22:38        28m      5.7km
pickup anomaly from: hotspot-15
 hour:           0     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21    22    23
 pickups:       16    34    35    30    26    20     7    20     8     5     9    25    36    36    31    55    50    44    24    64    69    38  |109|   21
 dropoffs:  (   16    40    70    73    48    22    33    17    22    28    44    43   116    76    76    83    57    74    70    76    36    13  | 34|   18  )
```

Case Study 3 - Taxi Rides Anomalies

Anomaly: gap at a busy pickup hotspot

```
=====
driver license-3568
=====
taxi-3568 license-3568 2013-01-01T22:10 2013-01-01T22:38        28m      5.7km
pickup anomaly from: hotspot-15
 hour:           0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21   22   23
 pickups:       16   34   35   30   26   20    7   20    8    5    9   25   36   36   31   55   50   44   24   64   69   38  |109|  21
 dropoffs:  (   16   40   70   73   48   22   33   17   22   28   44   43  116   76   76   83   57   74   70   76   36   13  | 34|  18  )
```

Of Interest

- Reasoning with non-trivially sized data sets
- Deploying Logic Programming as a method for data analysis (as a Jupyter notebook)
- Interaction Fusemate with host programming language Scala

Data Cleansing as Situational Awareness

Example: Employments Database

Company	Employee	Since	Full-time
ABM	Alice	1/3/18	No
BBM	Bob	5/3/18	No
ABM	Alice	1/6/19	Yes

Data Cleansing as Situational Awareness

Example: Employments Database

Company	Employee	Since	Full-time
ABM	Alice	1/3/18	No
BBM	Bob	5/3/18	No
ABM	Alice	1/6/19	Yes

Problem: More than a full-time contract at the same time

Data Cleansing as Situational Awareness

Example: Employments Database

Company	Employee	Since	Full-time
ABM	Alice	1/3/18	No
BBM	Bob	5/3/18	No
ABM	Alice	1/6/19	Yes

Problem: More than a full-time contract at the same time

How to explain and fix this inconsistency?

Approach

- There is a fixed set of contract operators: *cessation*, *conversion*, *new contract*
- Try them out as "fixes" for the problem
- Or was it Bob? Or someone else?

Conclusions

Summary

"Situational awareness = time-stratified logic programming + belief revision"

-> Good balance between expressivity and declarativity

The implementation is meant to be practical (workflow integration, ease of use)

Current and Future Work

Classical negation

Proper belief revision (ramification problem)

Timed LTL constraints $\Box t\ .\ \mathsf{shipped}(B) \rightarrow \Diamond s\ .\ s \leq t + 5 \wedge \mathsf{received}(B)$

Probabilities and combination with machine learning

- Probabilistic EDBs a la ProbLog `Load(10, “tomatoes”, “pallet”) : 0.3`
- ML as a subroutine for anomaly detection?

  Context may help to favoid false positives

Implementation at https://bitbucket.csiro.au/users/bau050/repos/fusemate/