UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Corso di Laurea Magistrale in Informatica

PyMAPE
A software framework to support the development and deployment of Autonomous Systems

Emanuele Palombo

Relatore
Prof. Davide Di Ruscio

Informatica @UnivAQ

Corso di Laurea in Informatica

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

WHAT ARE SELF-ADAPTIVE SYSTEMS ?

Today systems must function in complex environments built out of infrastructure, components, services, and other systems that are not under direct control of the original system and/or its developers. These systems typically run in **unpredictable** and **unstable** environments.

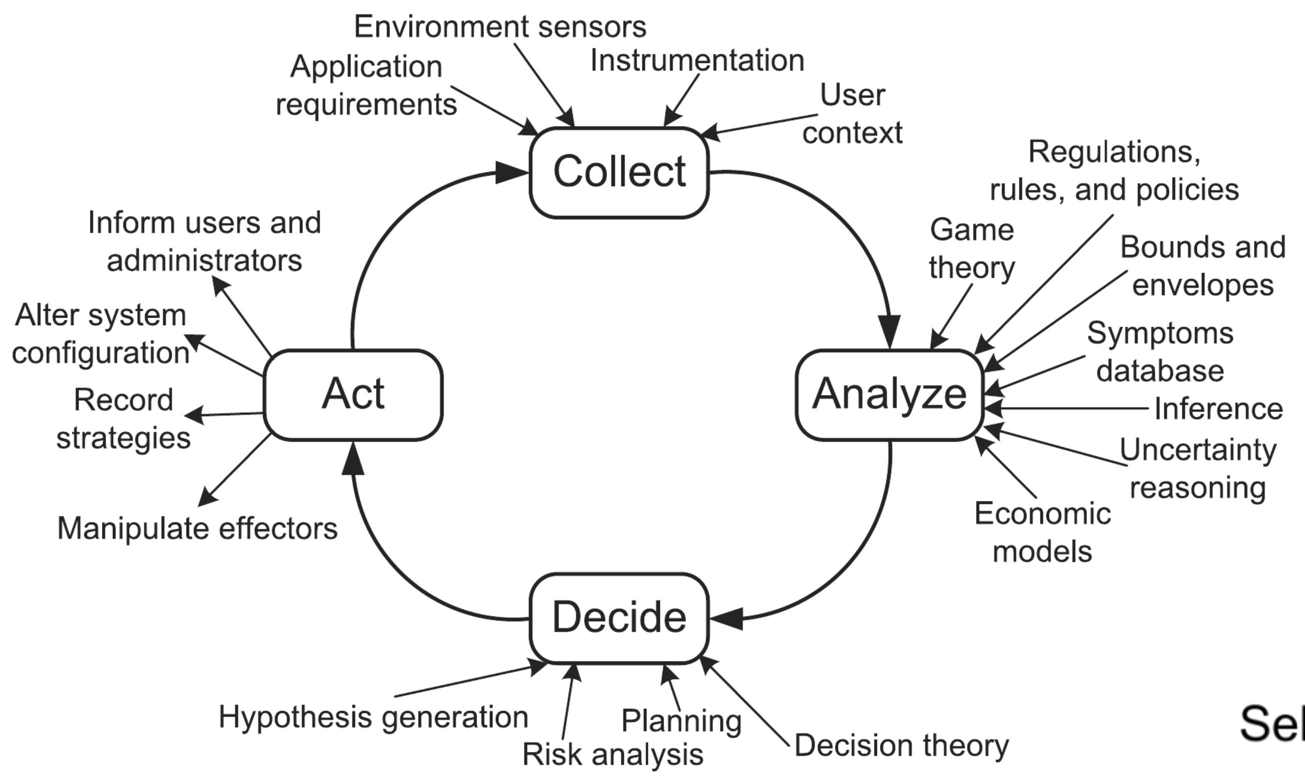
A **Self-Adaptive** system (SAS) is able to automatically modify itself in response to changes in its operating **environment** [1].

The "self" prefix indicates that the systems decide **autonomously** (i.e. without or with minimal interference) how to adapt changes in their contexts. While some self-adaptive system may be able to function without any human intervention, guidance in the form of higher-level **goals** is useful and realized in many systems.

[1] "A survey on engineering approaches for self-adaptive systems", C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker

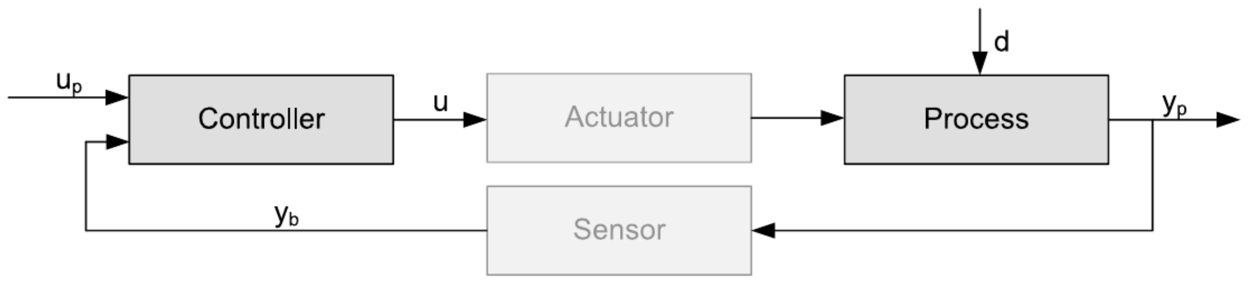
LOOP

FEEDBACK CONTROL AND MAPE-K

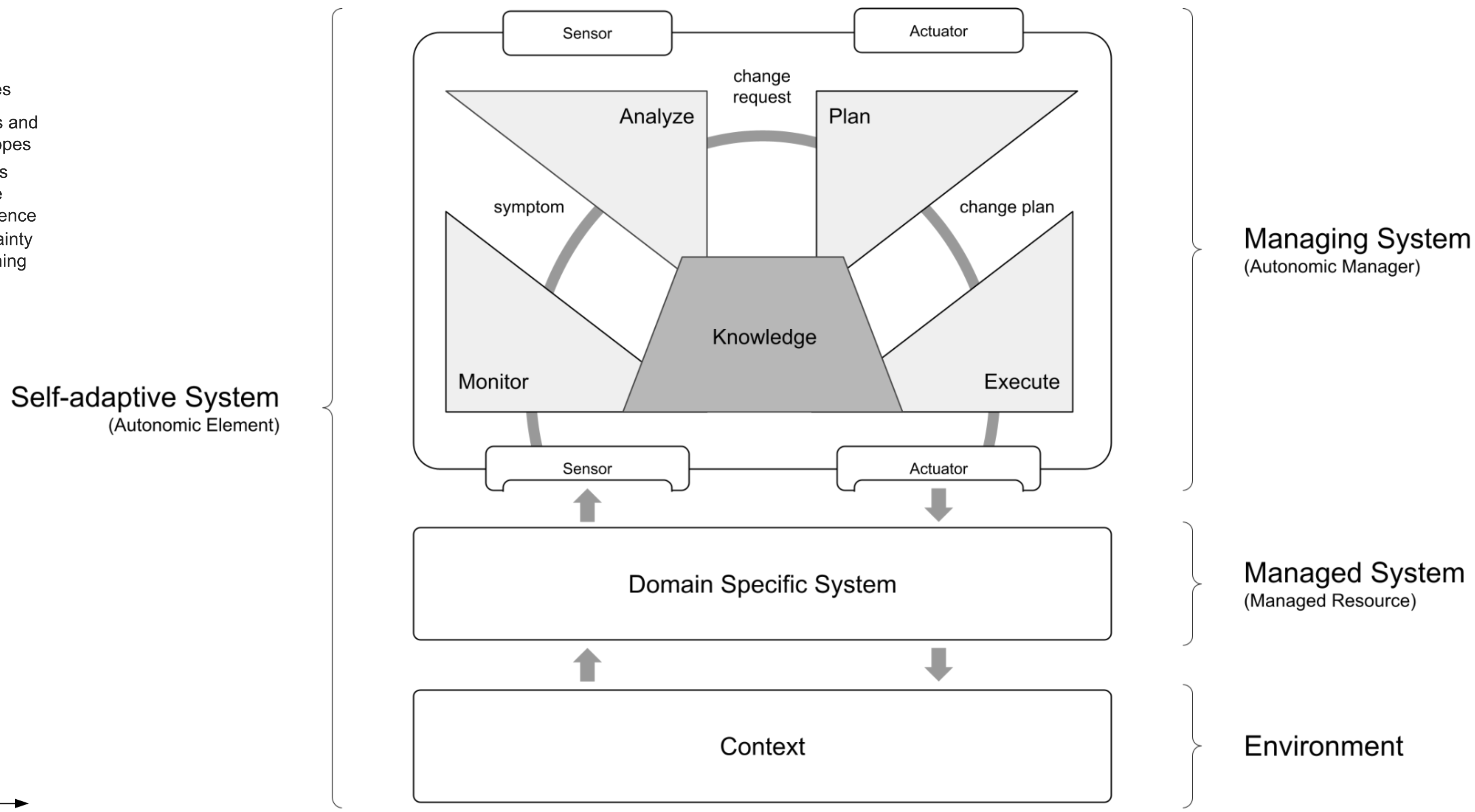


GENERIC MODEL

AI community's sense-plan-act approach of the early 1980s to control autonomous mobile robots



CONTROL THEORY

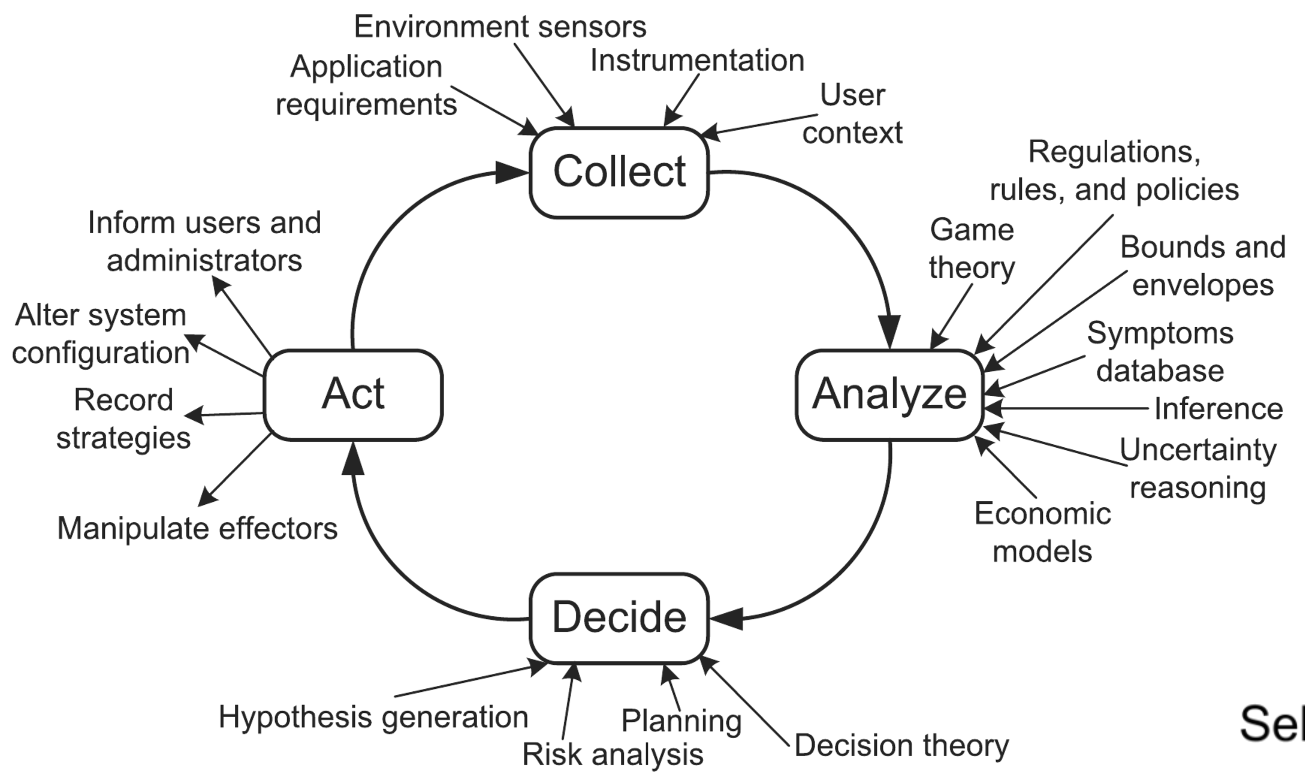


AUTONOMIC COMPUTING

IBM's architectural blueprint for autonomic computing

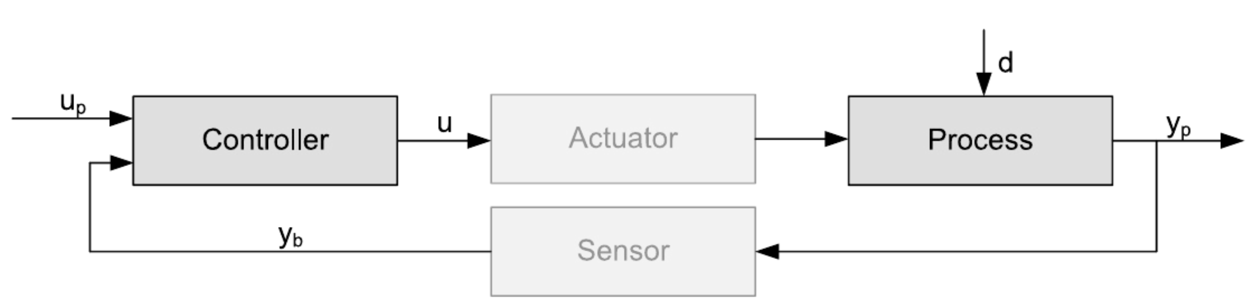
LOOP

FEEDBACK CONTROL AND MAPE-K

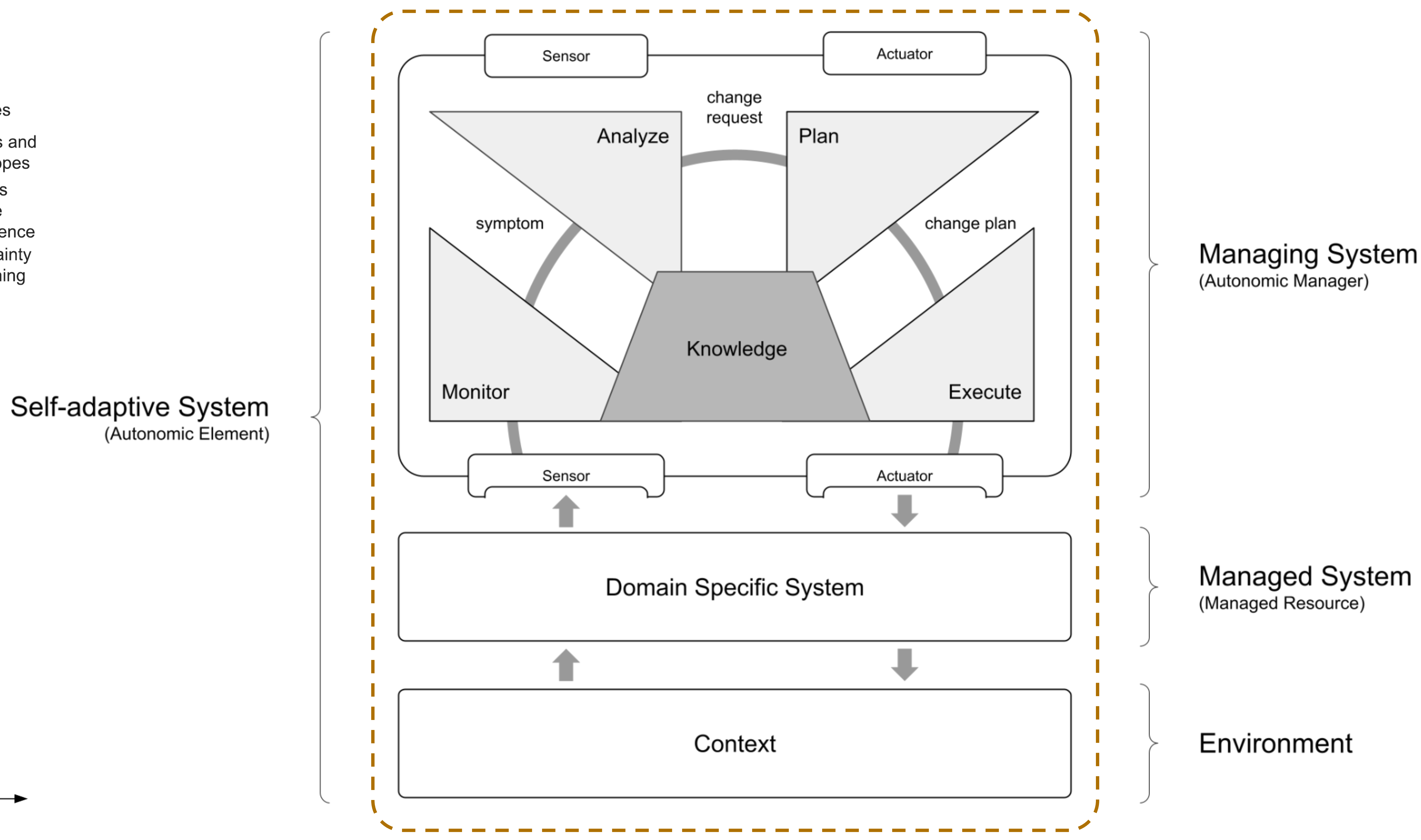


GENERIC MODEL

AI community's sense-plan-act approach of the early 1980s to control autonomous mobile robots



CONTROL THEORY



AUTONOMIC COMPUTING

IBM's architectural blueprint for autonomic computing

FRAMEWORK CHALLENGES

01 **CONTAINMENT**
Reuse, modularity and isolation of MAPE components as **first-class entity**.

02 **COMMUNICATION INTERFACE (STANDARDIZATION)**
Shared interface between components that allow **stream** communication, filtering, pre/post processing, data exchange communication and routing.

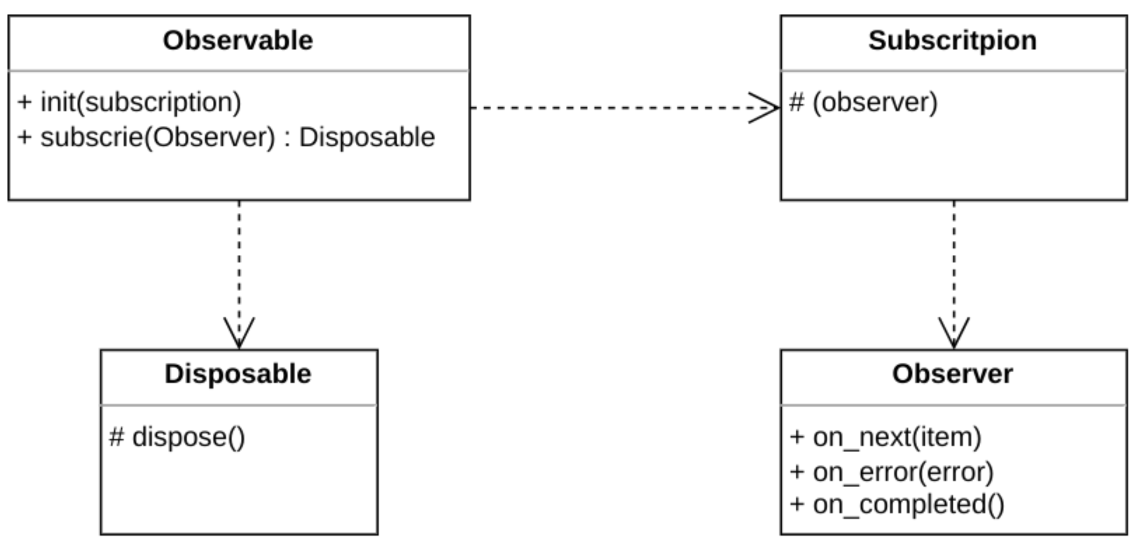
03 **DISTRIBUTION**
Multi-device distribution of MAPE loops and components.

04 **DECENTRALIZED PATTERNS**
Flat p2p and/or **hierarchical** architectures of loops and components with concerns separation. Allowing runtime pattern reconfiguration (stopping/starting, (un)linking, adding/removing).

05 **NETWORK COMMUNICATION PARADIGMS**
Different paradigms (blackboard, direct message) and protocols for various patterns interactions.

06 **STATE / KNOWLEDGE**
Distributed **multi-scope** (global, level, loop) Knowledge with partitioning and/or (full/partial async) replication.

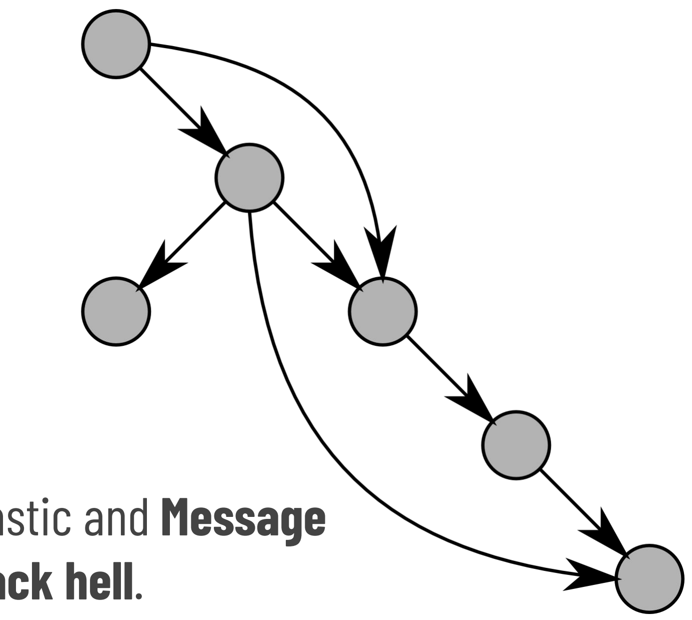
PARADIGMS AND TOOLS



01

REACTIVE SYSTEM/PROGRAMMING AND STREAM

System reactive to **external** event. Pillars: Responsive, Resilient, Elastic and **Message Driven**. Specific case of **event-driven** programming to avoid **callback hell**.



02

REACTIVEX (OBSERVER PATTERN)

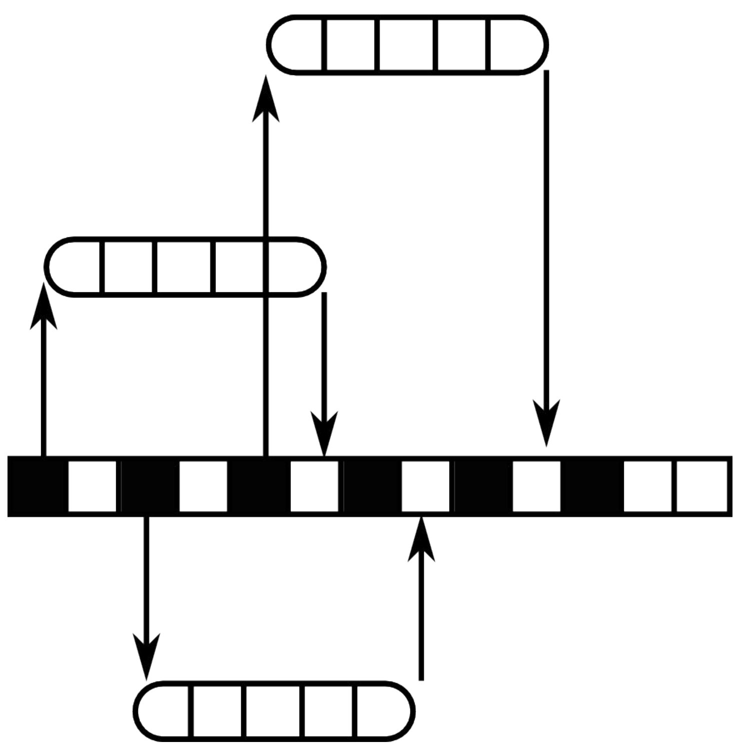
Observables represent a **source** of events. An **observer** subscribes to an observable to receive items emitted (Hot, Cold, Subject,etc). Pipe **operators** modify streams flowing through them.

	single	multi
sync	getter	iterable
async	future	observable

03

ASYNCHRONOUS PROGRAMMING

Manage (I/O bound) tasks concurrency with non-blocking I/O operations.



04

REDIS, IN-MEMORY DATA STRUCTURE SERVER

Distributed, in-memory key-value data structure (strings, hashes, lists, (ordered) sets, queue, lock) store, cache and **message broker** with keyspace **notifications**. Partitioning and/or (full/partial async) replication.

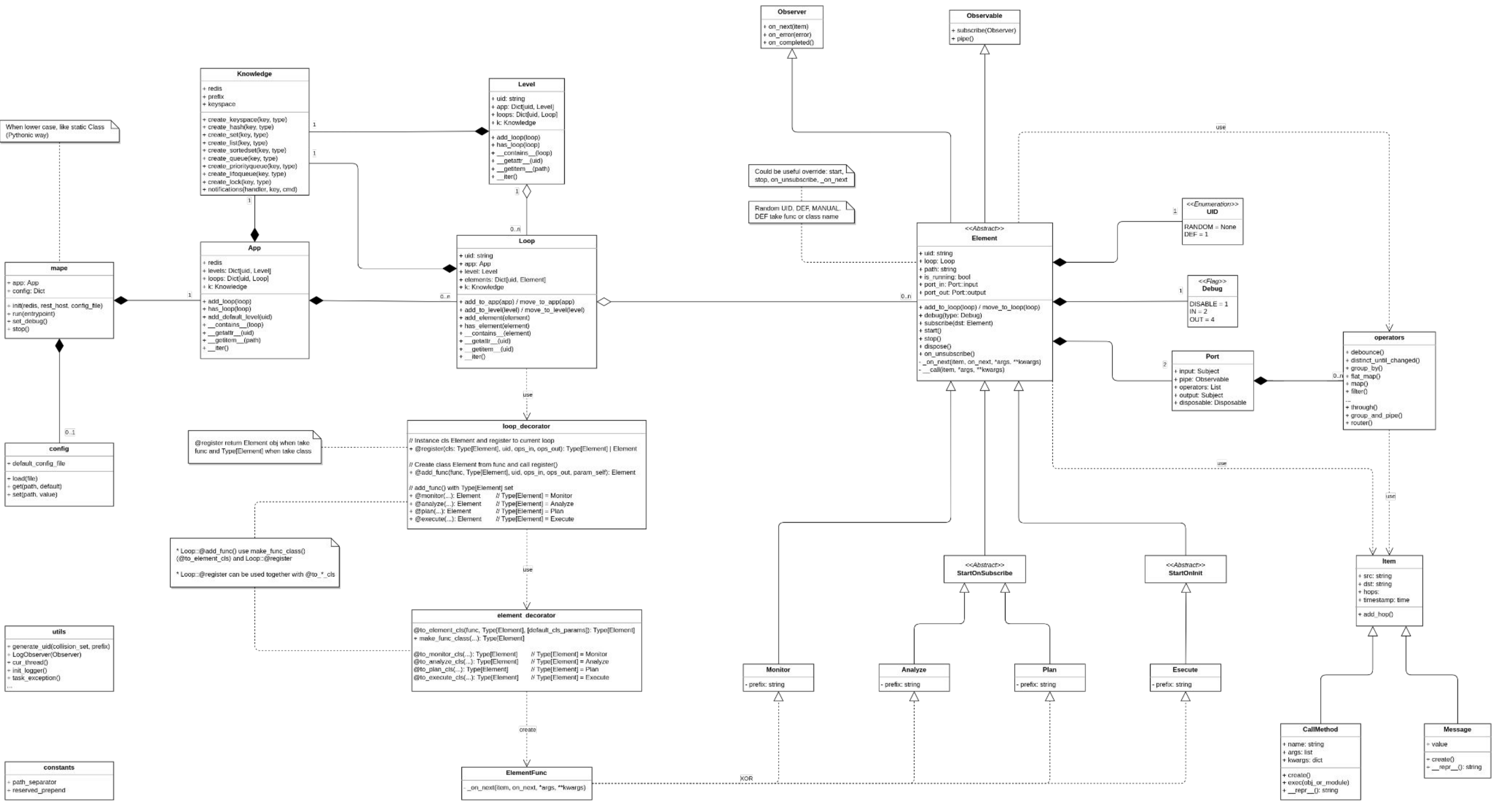
Other used technologies: InfluxDB, Pub/Sub, RestAPI, (De)serialization system, Docker, Yaml, etc

PyMAPE FRAMEWORK

01

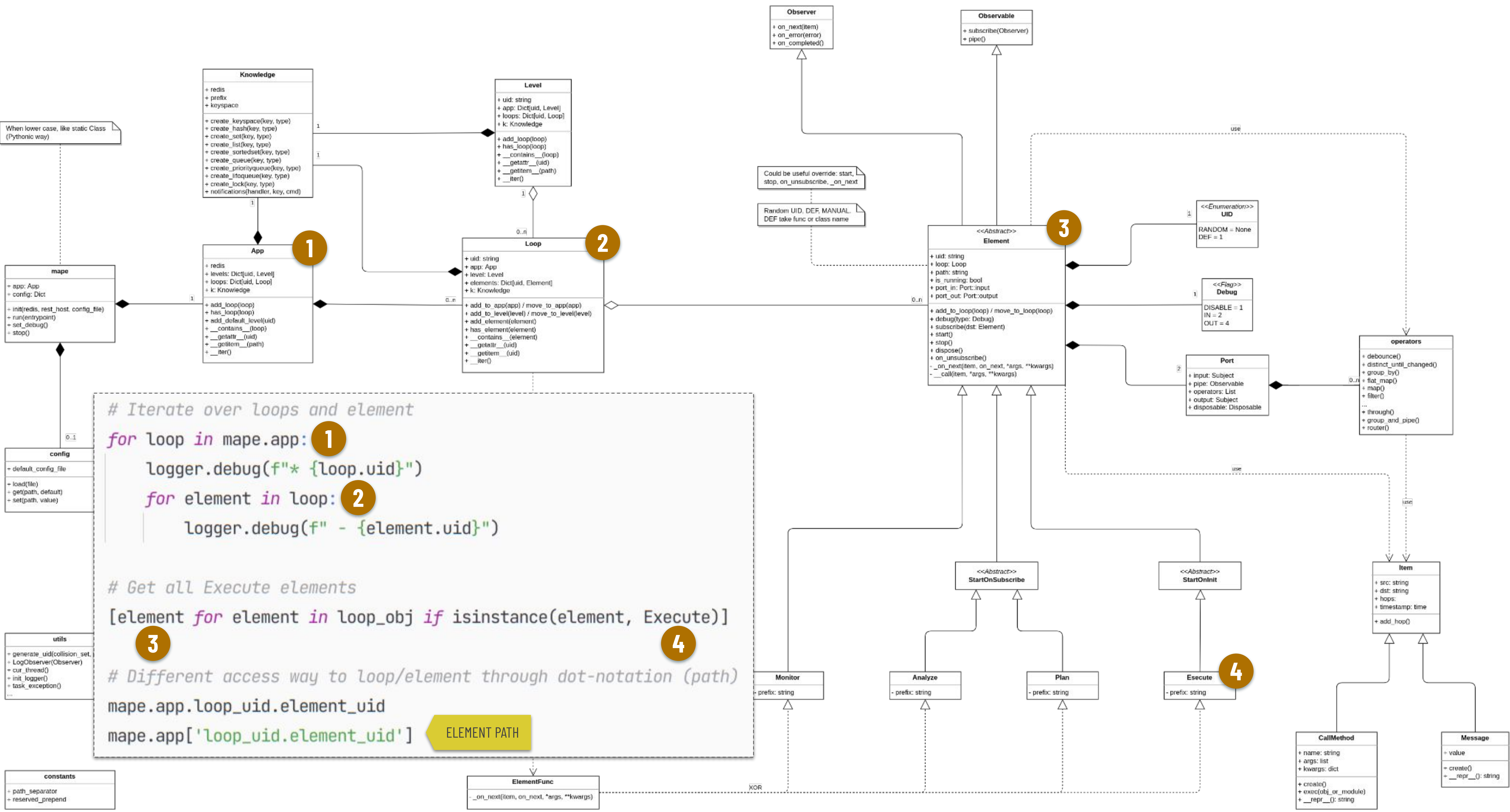
CLASS DIAGRAM

MAIN



CLASS DIAGRAM

MAIN

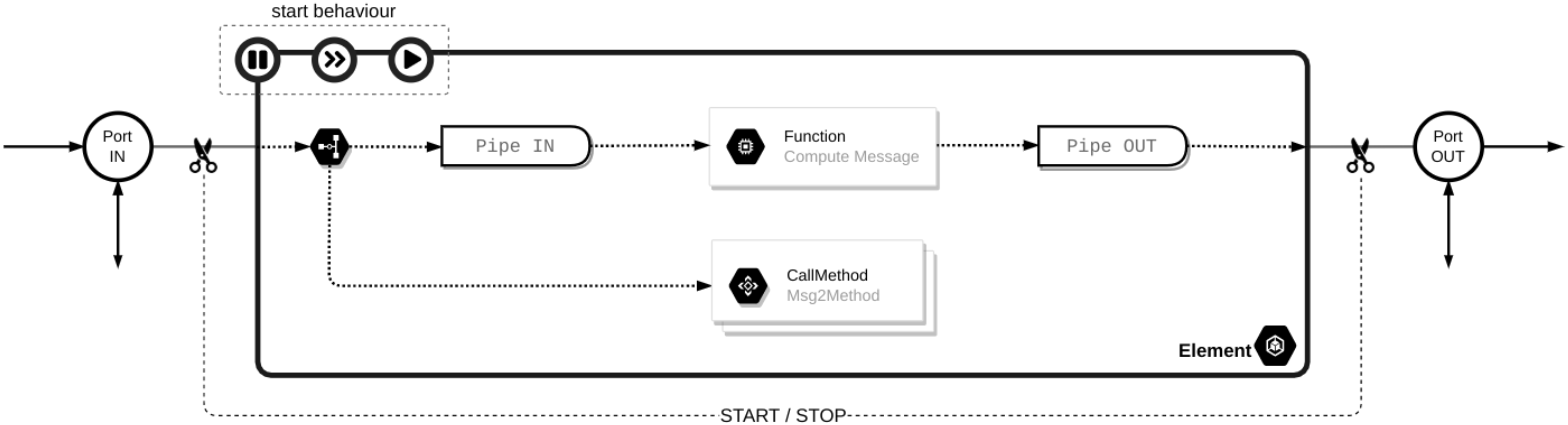



```python
# Iterate over loops and element
for loop in mape.app:                              # (1)
    logger.debug(f"* {loop.uid}")
    for element in loop:                           # (2)
        logger.debug(f" - {element.uid}")

# Get all Execute elements
[element for element in loop_obj if isinstance(element, Execute)]   # (3) (4)

# Different access way to loop/element through dot-notation (path)
mape.app.loop_uid.element_uid
mape.app['loop_uid.element_uid']
```

ELEMENT PATH

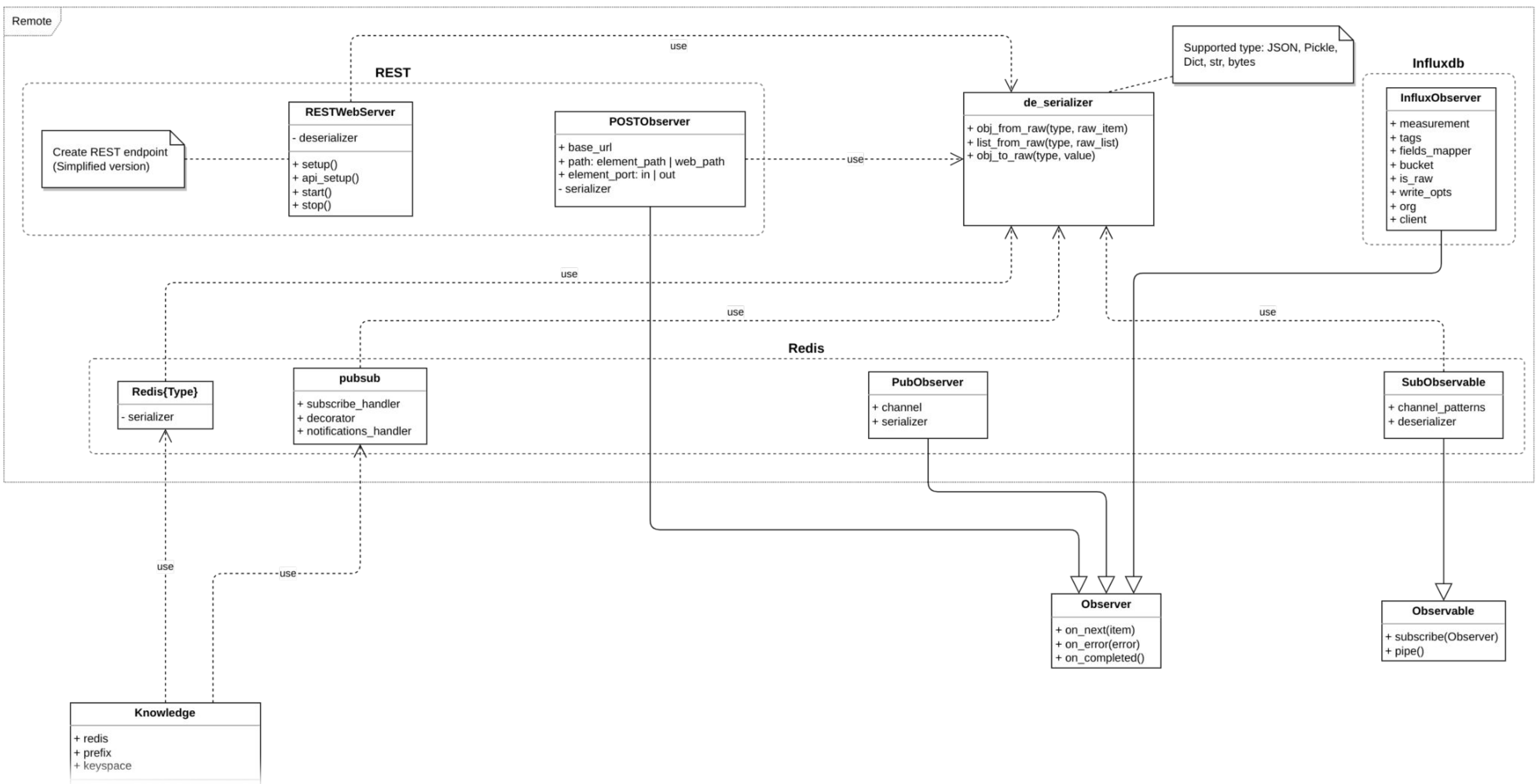
INSIDE AN ELEMENT

PORTS, PIPE OPERATORS, CORE FUNCTION …



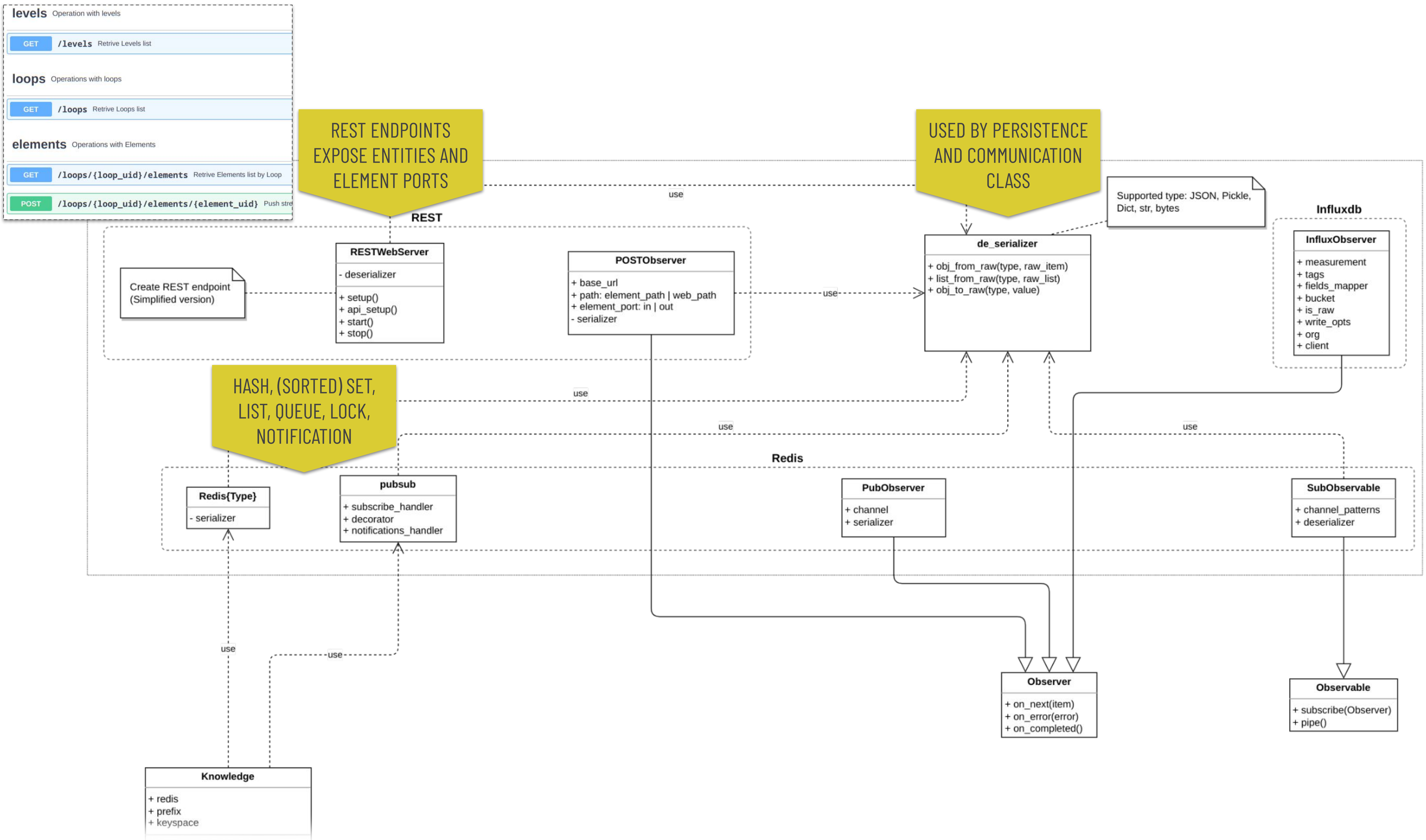
GRAPHIC NOTATION

CLASS DIAGRAM
REMOTE



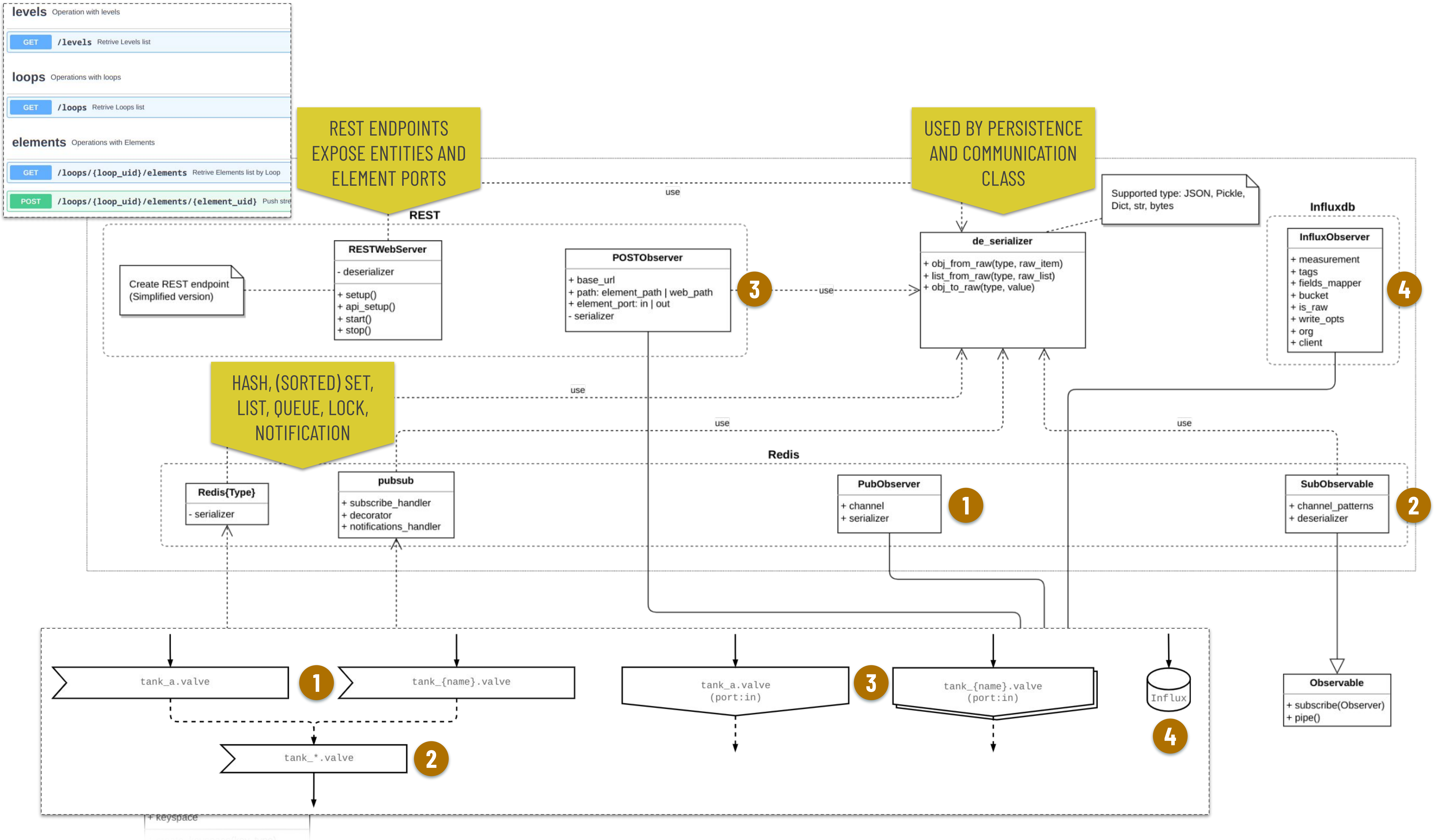
CLASS DIAGRAM

REMOTE

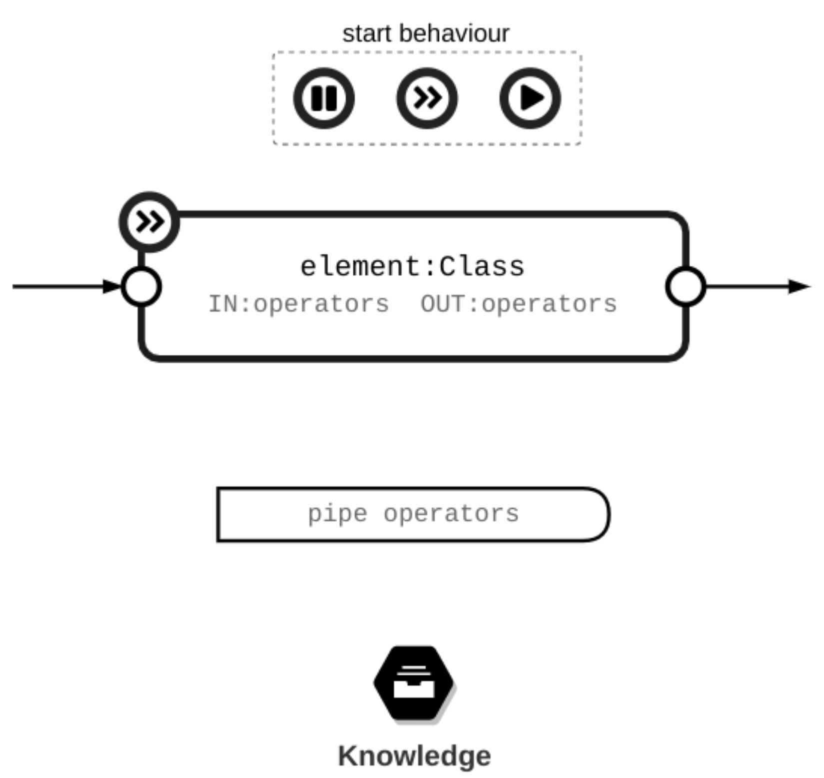


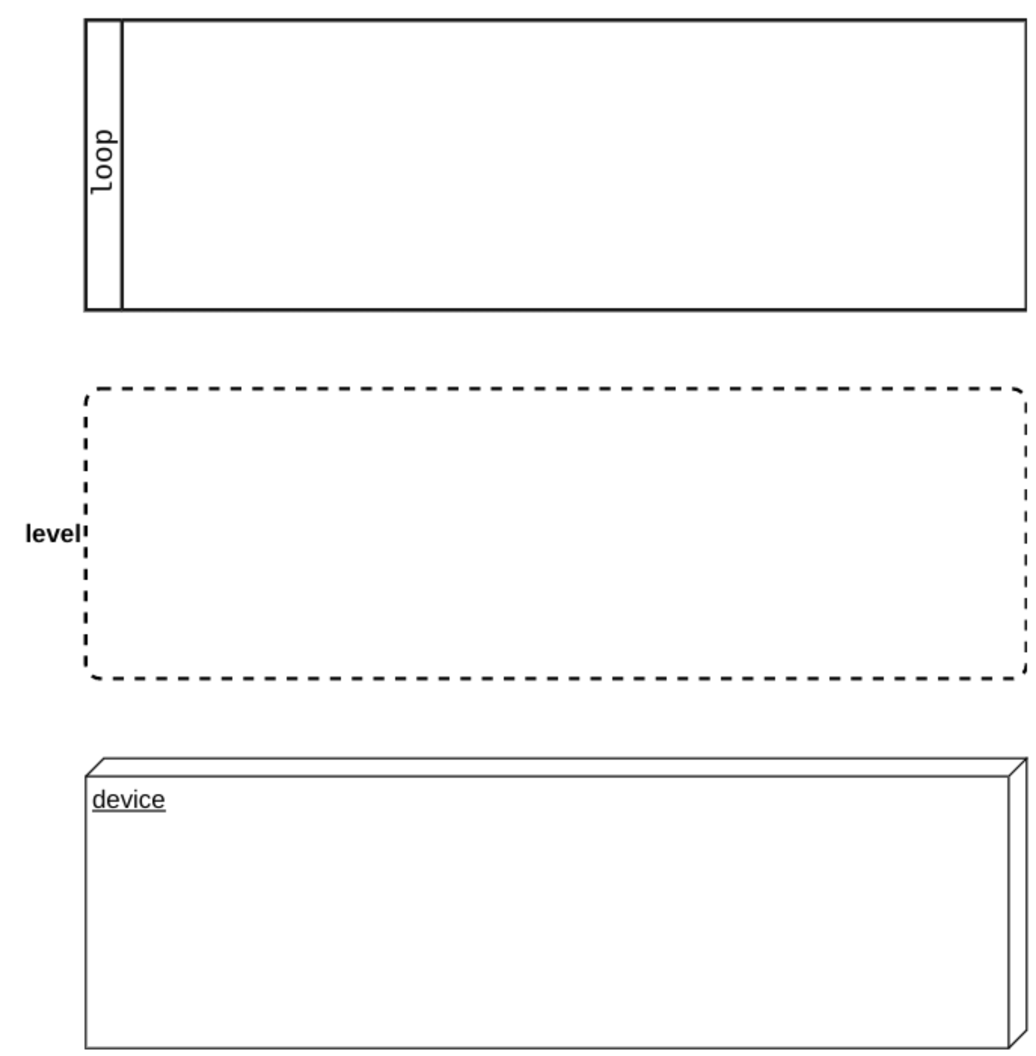
CLASS DIAGRAM

REMOTE



GRAPHICAL NOTATION

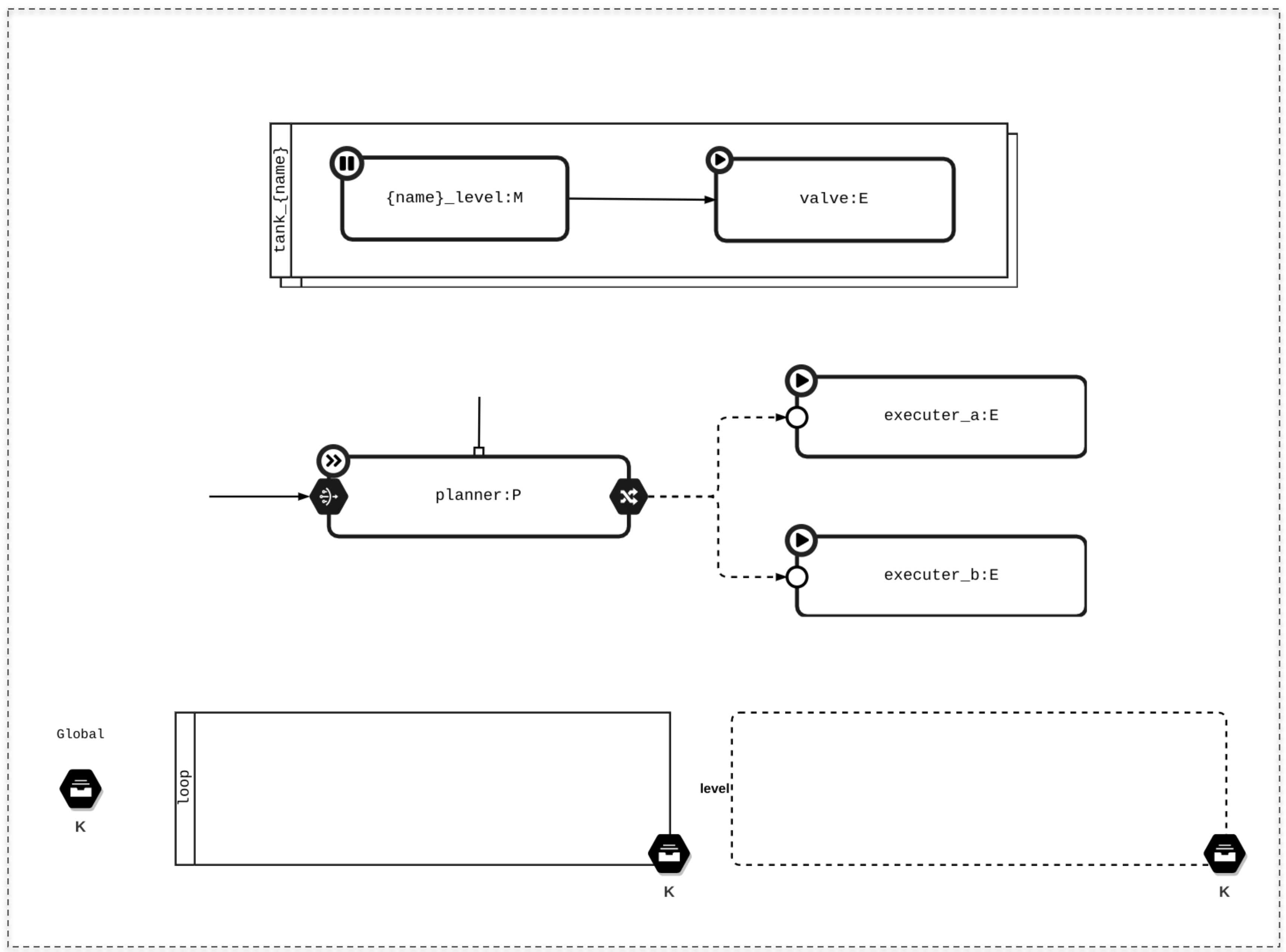




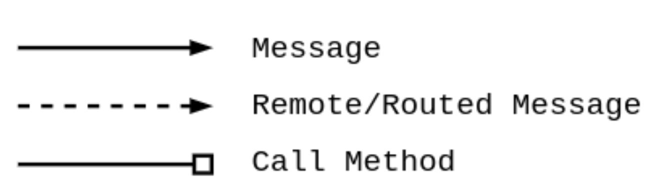
Loop, element, level are intended as uid of the respective entity. Notation *{var}* in the uid means *a multi instance entity with parametric uid. It can be also found in the newtwork communication path.*

eg. element_{tag}_uid, loop_{cat}_uid

INSTANCE EXAMPLES

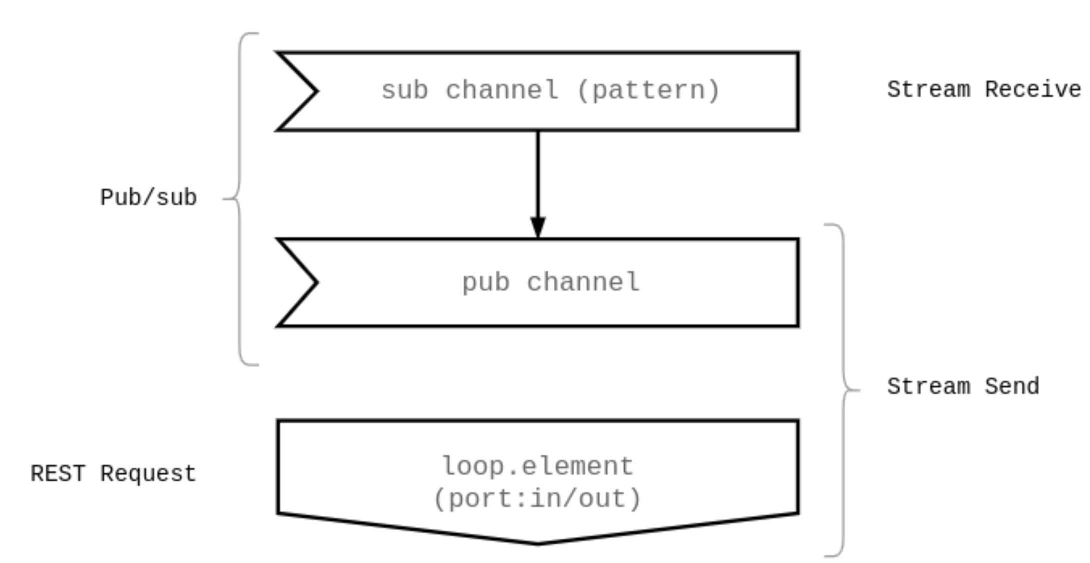


Connection lines



Connection operators

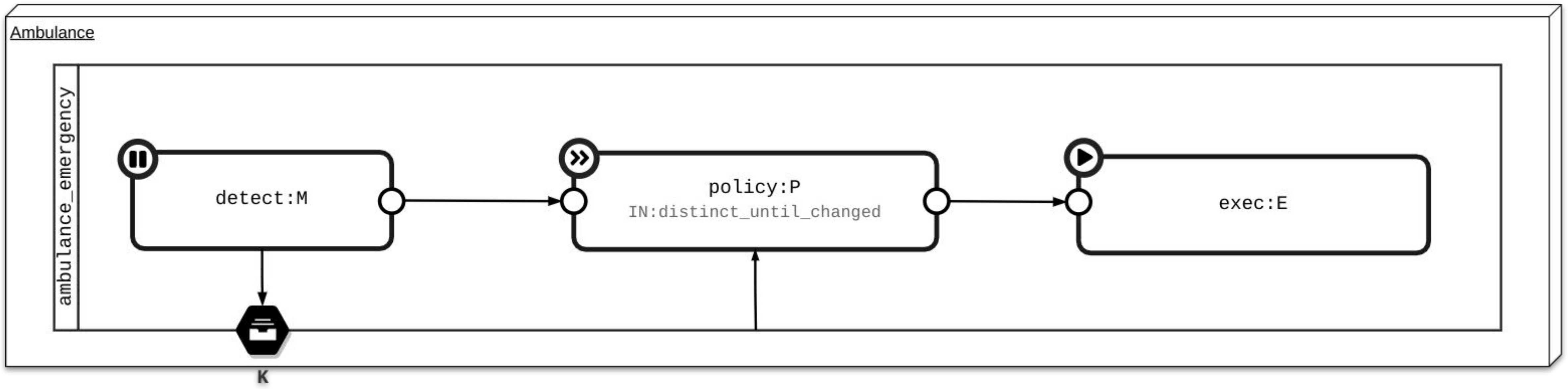
Network stream communication



Element path (loop.element) can be used as channel.
For sub channel can be used wildcard as patterns.

FIRST MAPE LOOP

AMBULANCE

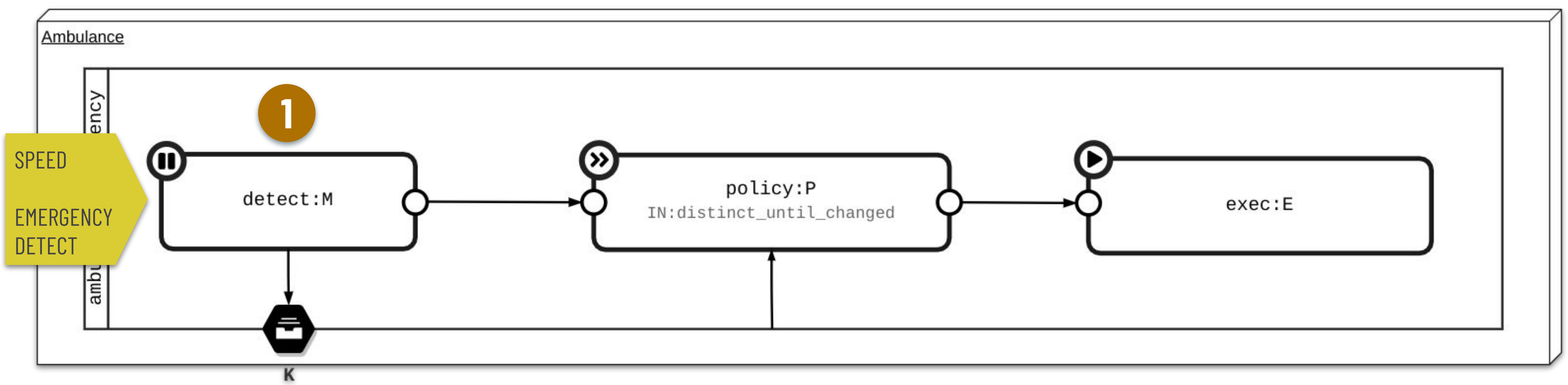


FIRST MAPE LOOP

AMBULANCE


```python
@loop.monitor
def detect(item, on_next, self):
    if 'speed_limit' in item:
        # Local volatile knowledge
        self.loop.k.speed_limit = item['speed_limit']
    elif 'emergency_detect' in item:
        on_next(item['emergency_detect'])
```

1

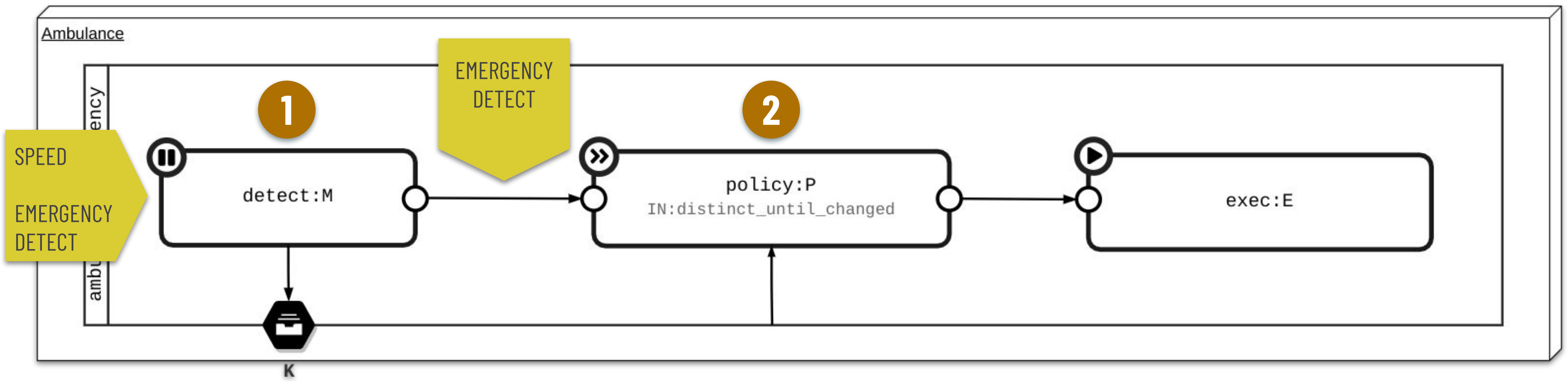



```python
""" MAPE Loop and Elements definition """
loop = Loop(uid='ambulance_emergency')

@loop.monitor
def detect(item, on_next, self):
    ...

@loop.plan(ops_in=ops.distinct_until_changed())
async def policy(emergency, on_next, self):
    ...

@loop.execute
def exec(item: dict, on_next):
    ...

for element in loop:
    element.debug(Element.Debug.IN)

""" MAPE Elements connection """
detect.subscribe(policy)
policy.subscribe(exec)

# Starting monitor...
detect.start()
```

1

FIRST MAPE LOOP
AMBULANCE

1
```
@loop.monitor
def detect(item, on_next, self):
    if 'speed_limit' in item:
        # Local volatile knowledge
        self.loop.k.speed_limit = item['speed_limit']
    elif 'emergency_detect' in item:
        on_next(item['emergency_detect'])
```



2
```
@loop.plan(ops_in=ops.distinct_until_changed())
async def policy(emergency, on_next, self):
    if emergency is True:
        self.last_speed_limit = self.loop.k.speed_limit
        new_speed = max(self.last_speed_limit, self.emergency_speed)

        on_next({'speed': new_speed})
        on_next({'siren': True})
    else:
        on_next({'speed': self.last_speed_limit})
        on_next({'siren': False})

policy.emergency_speed = 160
```

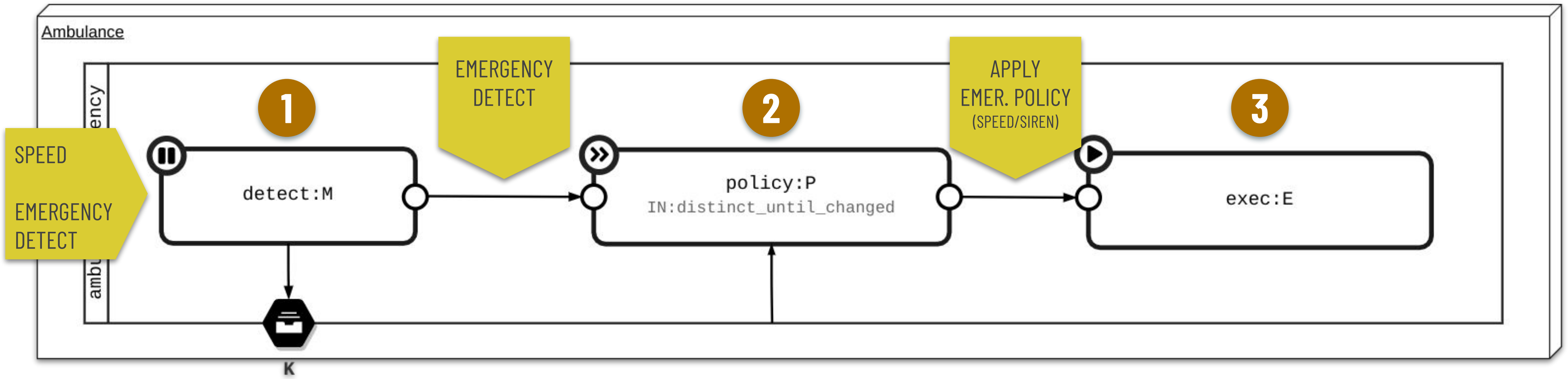
```
""" MAPE Loop and Elements definition """
loop = Loop(uid='ambulance_emergency')

@loop.monitor
def detect(item, on_next, self):   # 1
    ...

@loop.plan(ops_in=ops.distinct_until_changed())
async def policy(emergency, on_next, self):   # 2
    ...

@loop.execute
def exec(item: dict, on_next):
    ...

for element in loop:
    element.debug(Element.Debug.IN)

""" MAPE Elements connection """
detect.subscribe(policy)
policy.subscribe(exec)

# Starting monitor...
detect.start()
```

FIRST MAPE LOOP

AMBULANCE


```python
@loop.monitor
def detect(item, on_next, self):
    if 'speed_limit' in item:
        # Local volatile knowledge
        self.loop.k.speed_limit = item['speed_limit']
    elif 'emergency_detect' in item:
        on_next(item['emergency_detect'])
```

```python
@loop.execute
def exec(item: dict, on_next):
    if 'speed' in item:
        ambulance.speed_limit = item['speed']
    if 'siren' in item:
        ambulance.siren = item['siren']
```




```python
@loop.plan(ops_in=ops.distinct_until_changed())
async def policy(emergency, on_next, self):
    if emergency is True:
        self.last_speed_limit = self.loop.k.speed_limit
        new_speed = max(self.last_speed_limit, self.emergency_speed)

        on_next({'speed': new_speed})
        on_next({'siren': True})
    else:
        on_next({'speed': self.last_speed_limit})
        on_next({'siren': False})

policy.emergency_speed = 160
```

```python
""" MAPE Loop and Elements definition """
loop = Loop(uid='ambulance_emergency')

@loop.monitor
def detect(item, on_next, self):
    ...

@loop.plan(ops_in=ops.distinct_until_changed())
async def policy(emergency, on_next, self):
    ...

@loop.execute
def exec(item: dict, on_next):
    ...

for element in loop:
    element.debug(Element.Debug.IN)

""" MAPE Elements connection """
detect.subscribe(policy)
policy.subscribe(exec)

# Starting monitor...
detect.start()
```

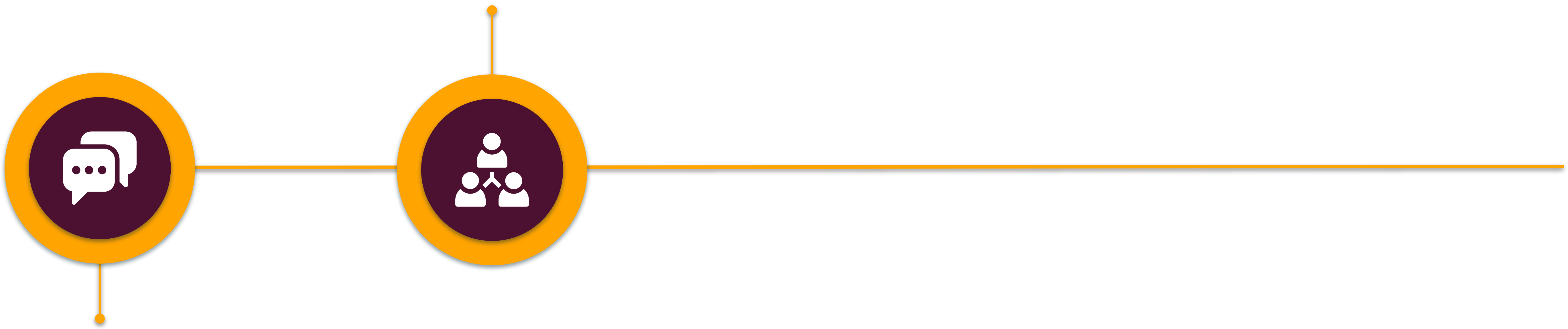
MAPE PATTERNS

02

DECENTRALIZED (AND DISTRIBUTED)

MAPE PATTERNS

Coordinated Control
Sharing and coordination among all MAPE components



Information Sharing
Sharing among Monitor

FULLY DECENTRALIZED

"FLAT" DISTRIBUTION MODEL

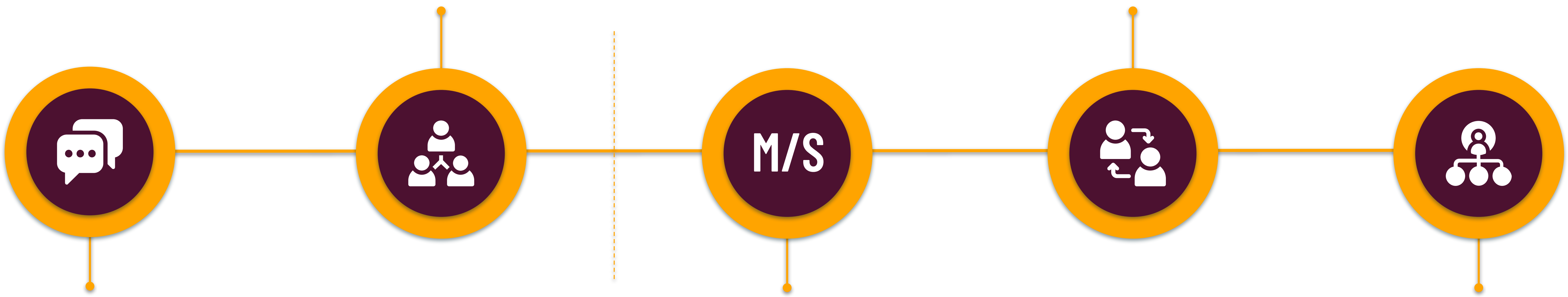
multiple peer MAPE loops cooperates in parallel to manage the overall self-adaptation

DECENTRALIZED (AND DISTRIBUTED)

MAPE PATTERNS

Coordinated Control
Sharing and coordination among all MAPE components

Regional Planning
Regional P for intra and cross-regional planning



Information Sharing
Sharing among Monitor

Master/Slave
Single (central) AP and multiple (local) ME

Hierarchical Control
"Divide et impera": different time-scales, resources and concerns

FULLY DECENTRALIZED

"FLAT" DISTRIBUTION MODEL

multiple peer MAPE loops cooperates in parallel to manage the overall self-adaptation

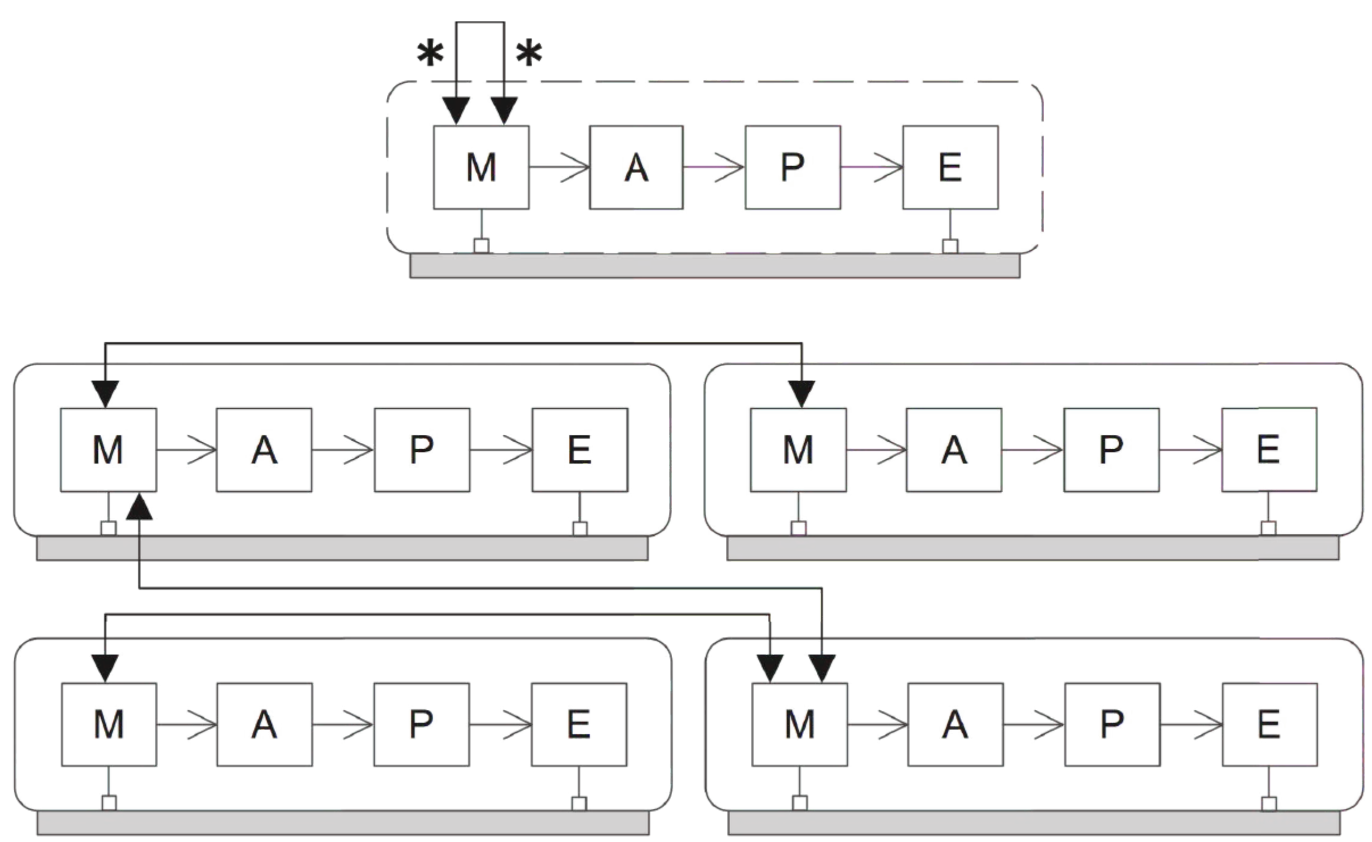
HYBRID APPROACH

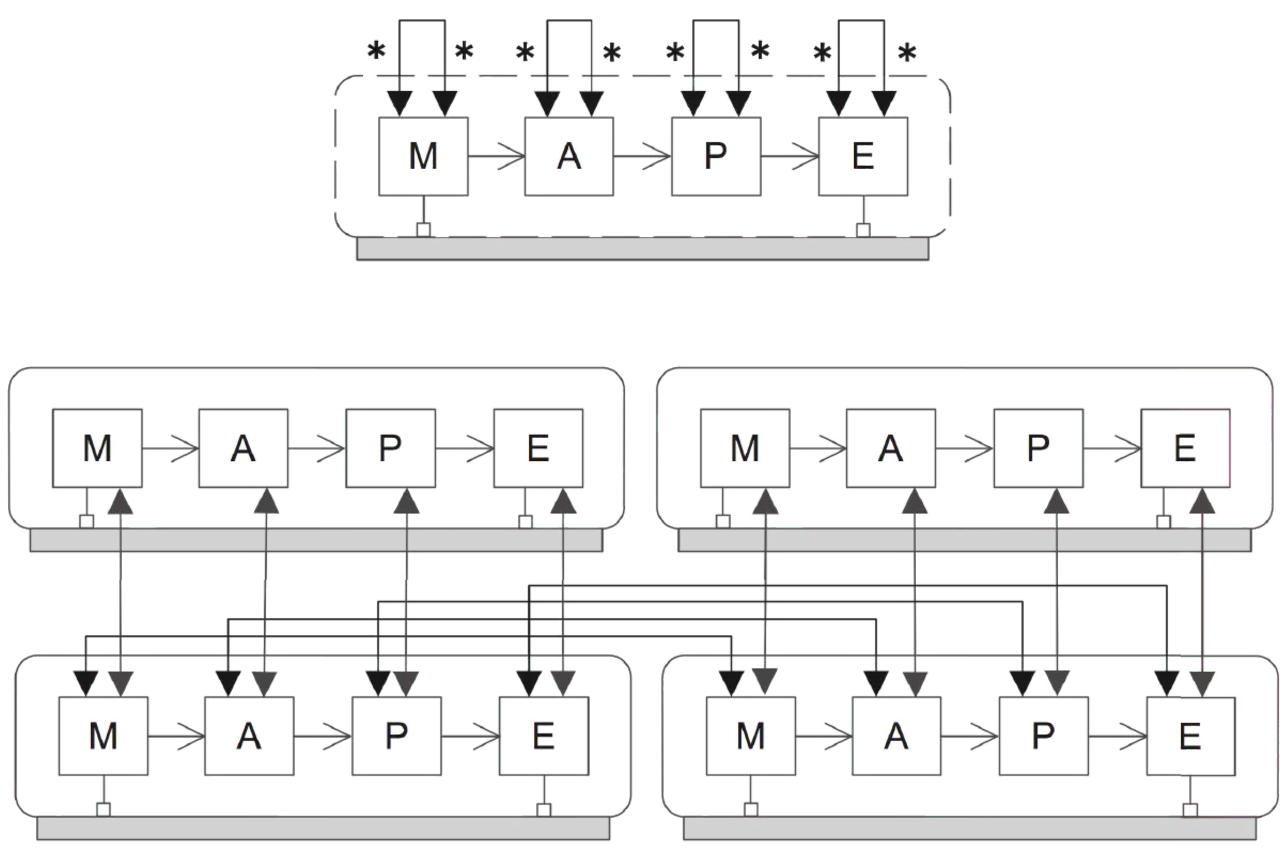
"HIERARCHICAL" DISTRIBUTION MODEL

separation of concerns, higher level MAPE components control subordinate MAPE components

INFORMATION & COORDINATED

MAPE PATTERN





INFORMATION SHARING

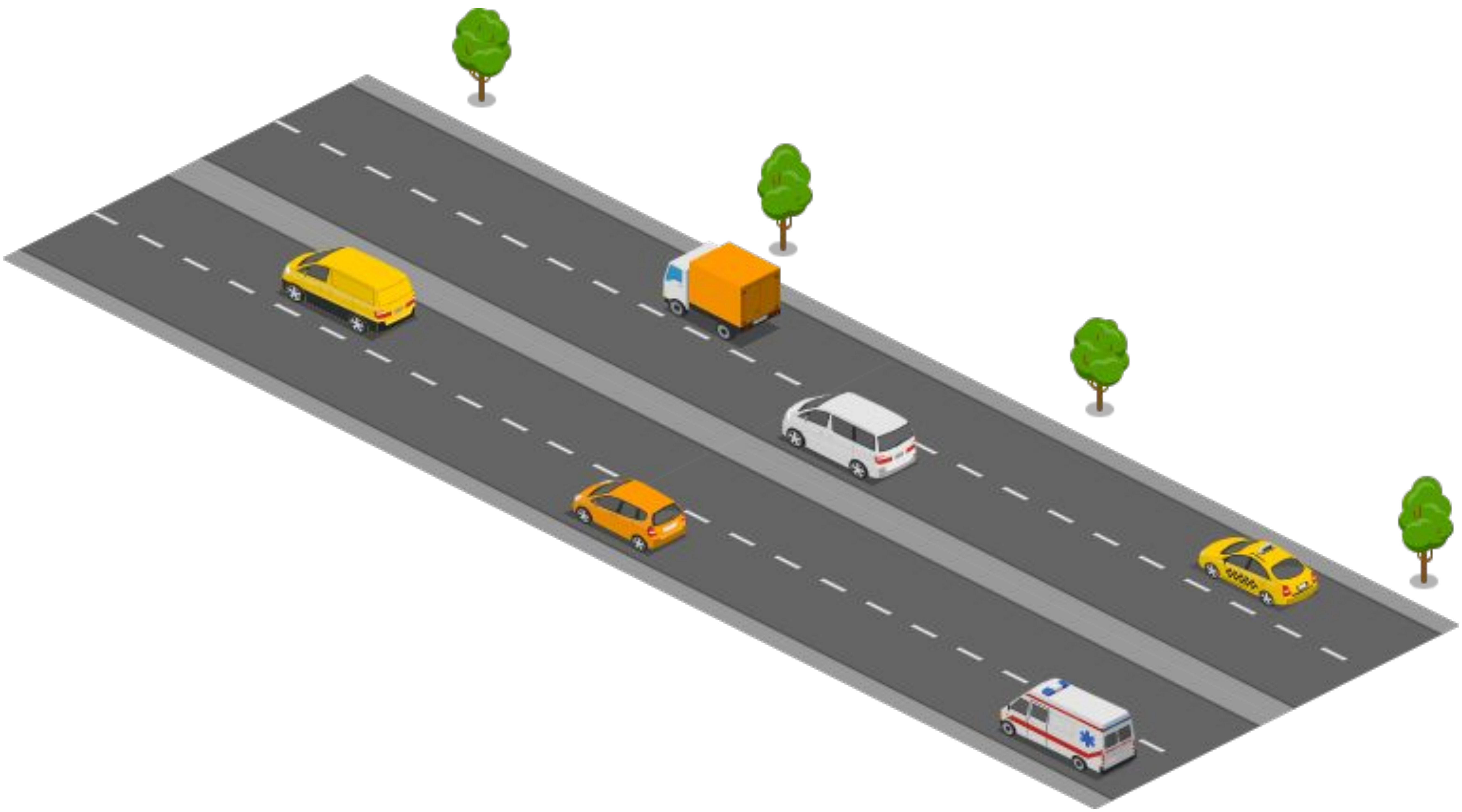
Systems **status** (*M*) is shared among the MAPE loops. *P*, *A*, and *E* components allow **local** adaptations without the need for coordination (timely decisions and execution). Reduced coordination may increase **locally** optimal **objectives** but at the cost of **globally** optimal ones.

COORDINATED CONTROL

MAPE elements of different MAPE loops can interact with peers to share particular information and/or coordinate their actions. Increase globally optimal objectives.

EMERGENCY MANAGE

AUTOMOTIVE DOMAIN



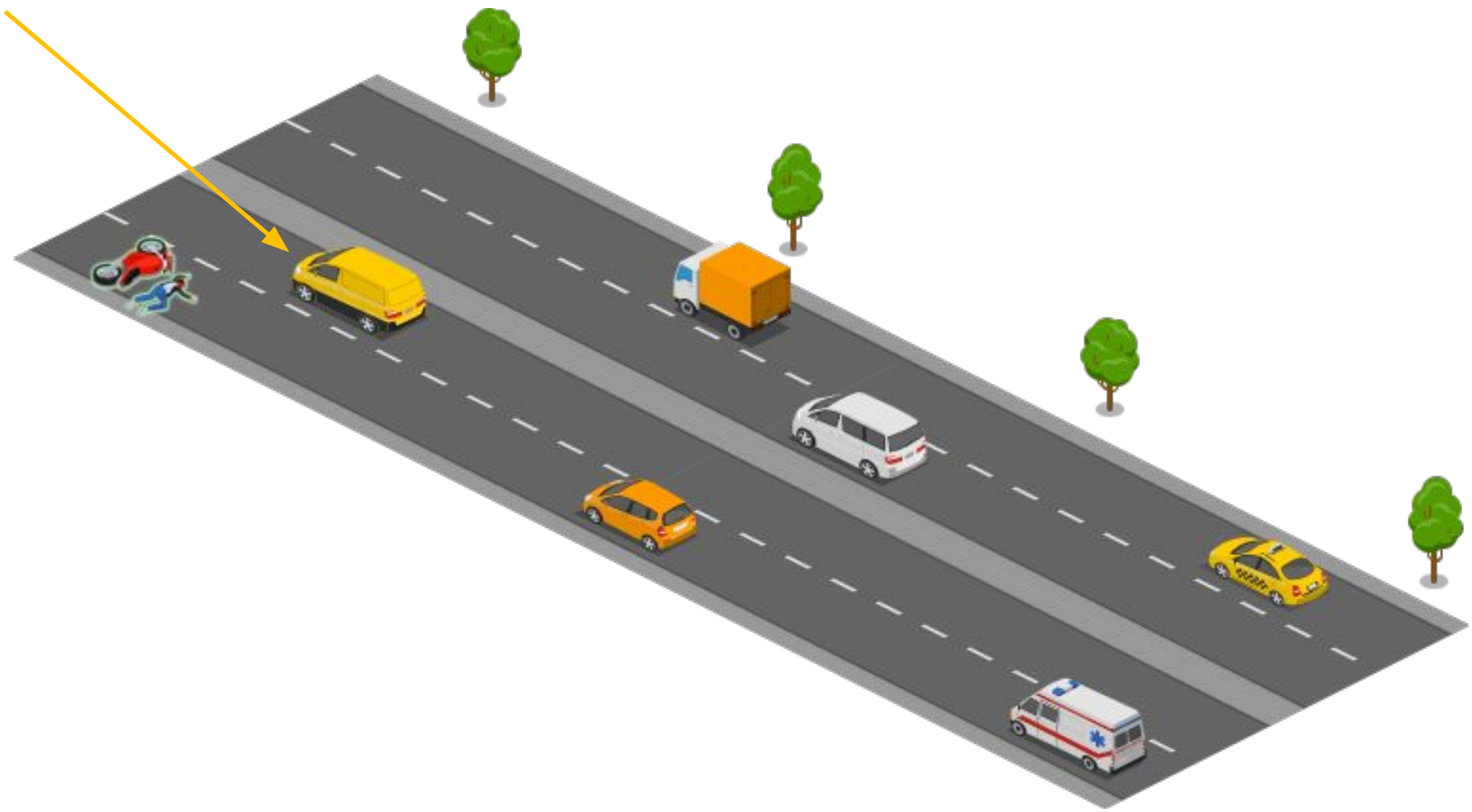
AMBULANCE-CAR EMERGENCY

EMERGENCY MANAGE

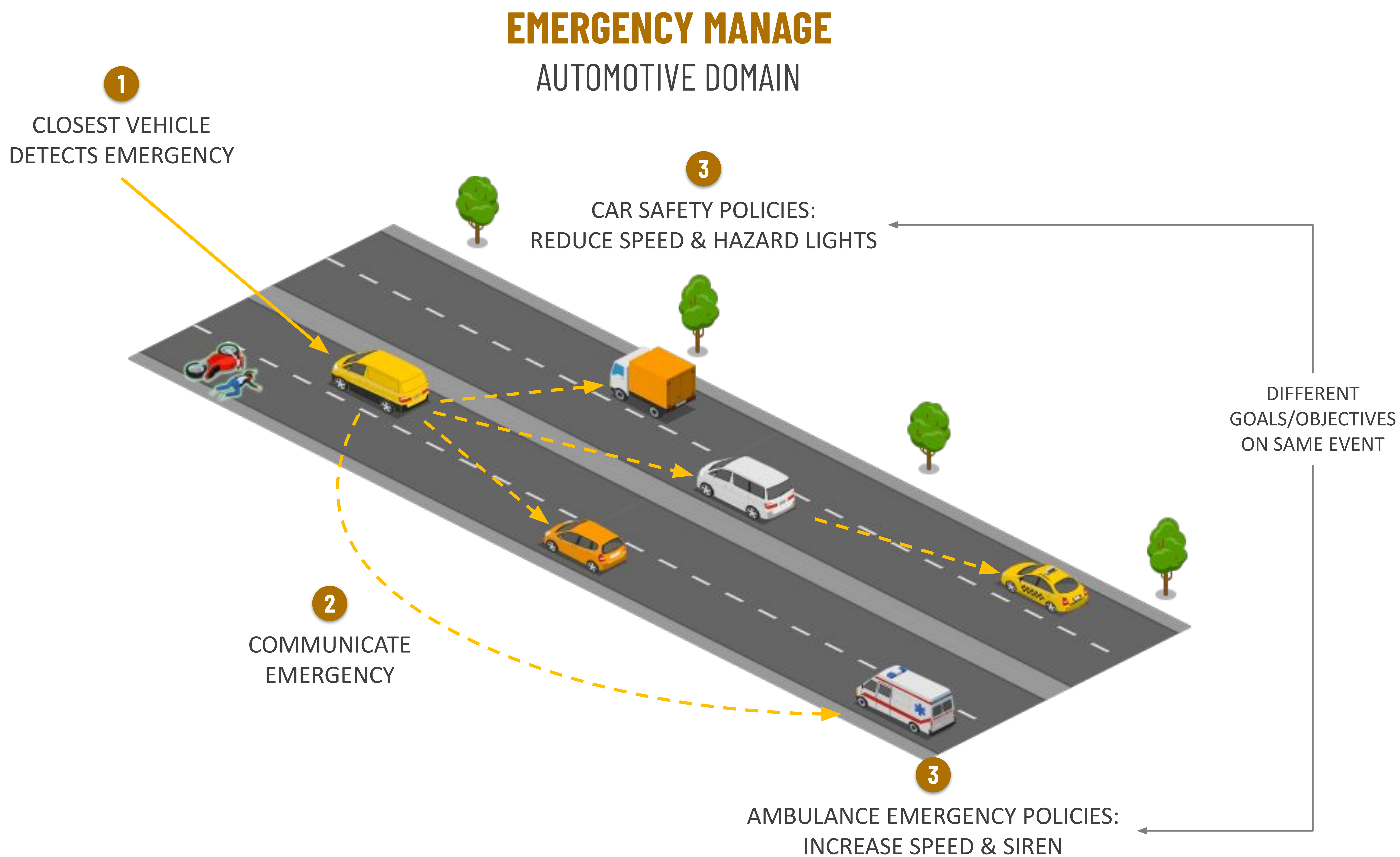
AUTOMOTIVE DOMAIN

1

CLOSEST VEHICLE
DETECTS EMERGENCY

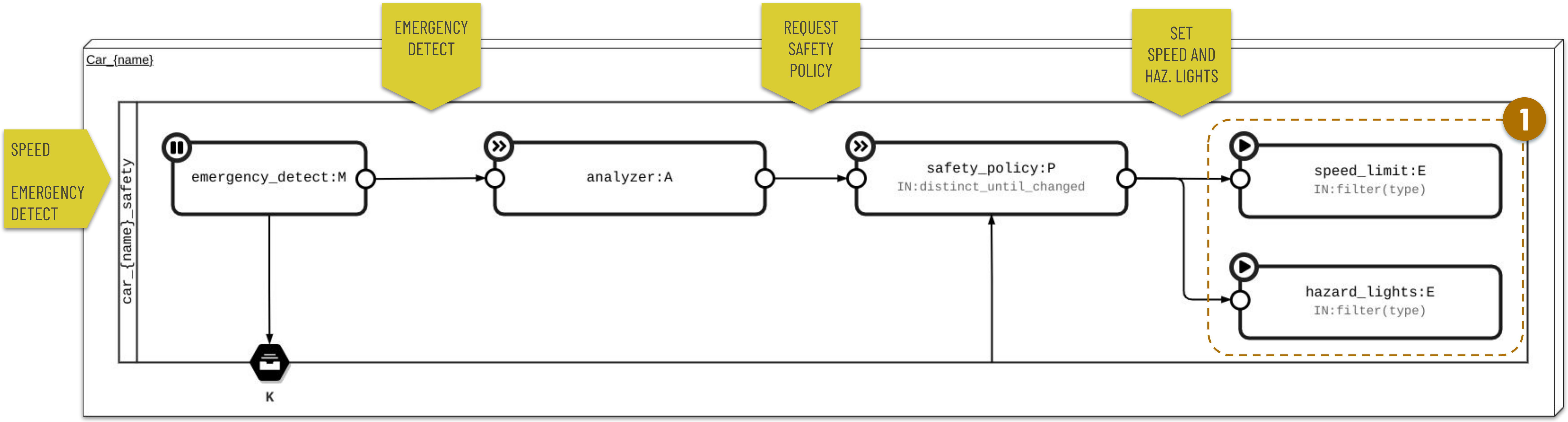


AMBULANCE-CAR EMERGENCY



AMBULANCE-CAR EMERGENCY

CAR

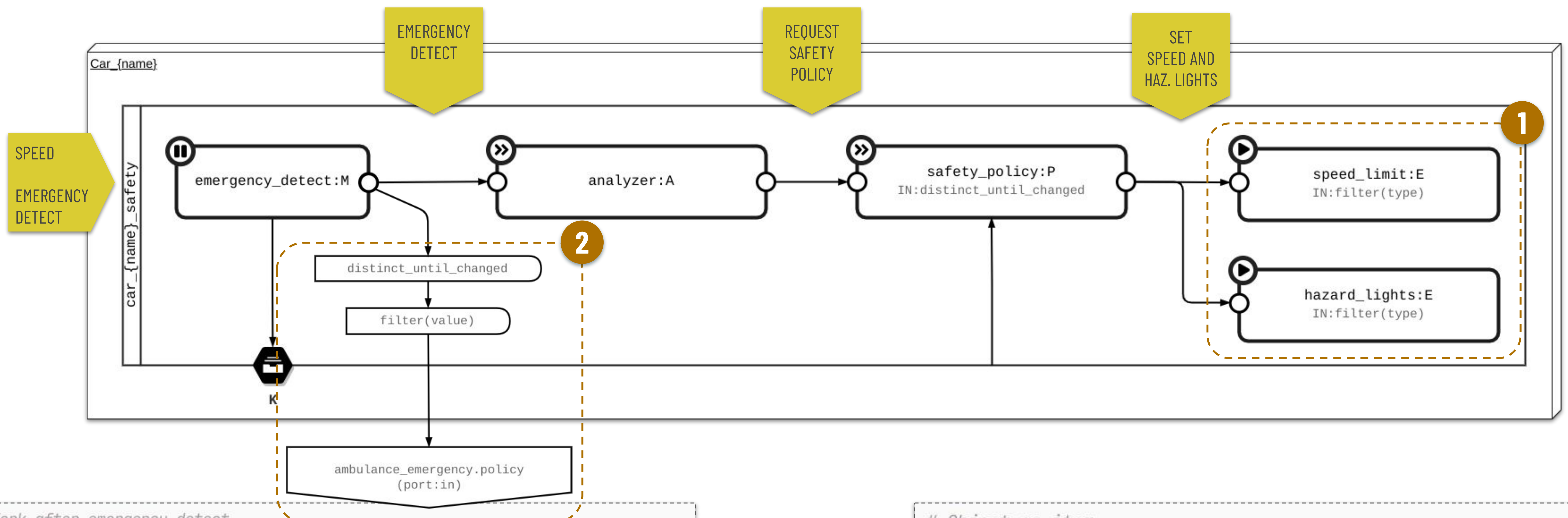



```python
# Object as item
@loop.execute(ops_in=ops.filter(lambda item: isinstance(item, SpeedItem)))
def speed_limit(item: SpeedItem, on_next):
    car.speed_limit = item.value

# Dict as item
@loop.execute(ops_in=ops.filter(lambda item: 'hazard_lights' in item))
def hazard_lights(item: dict, on_next):
    car.hazard_lights = item['hazard_lights']
```

1

AMBULANCE-CAR EMERGENCY

CAR to AMBULANCE (REST)

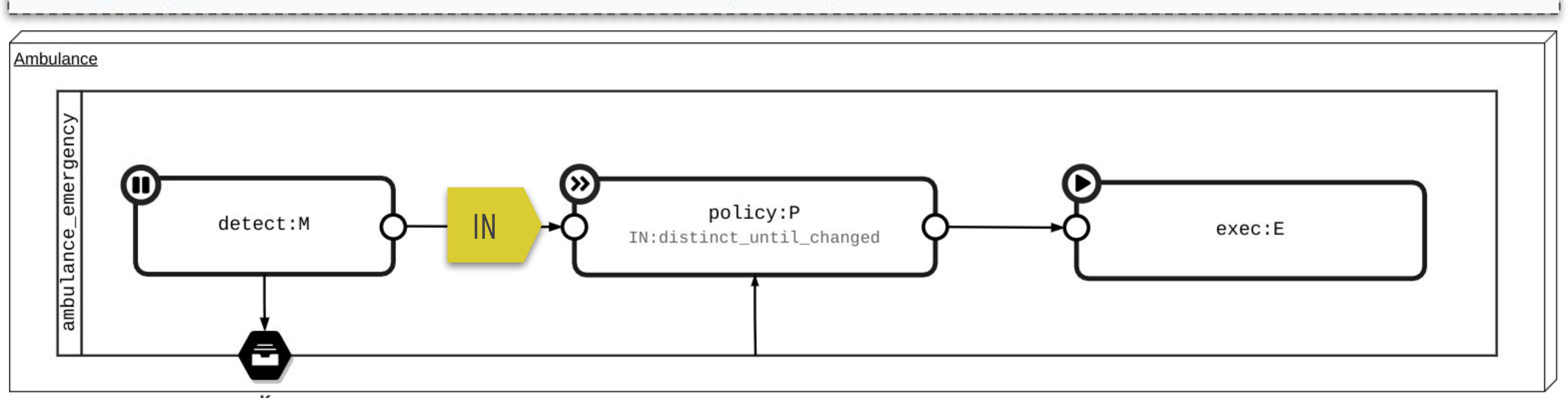



```python
# Stream fork after emergency_detect
emergency_detect_out = emergency_detect.pipe(
    # Only when local state change
    ops.distinct_until_changed(),
    # ...and there is an emergency
    ops.filter(lambda emergency: emergency is True)
)

# MAPE Elements Remote REST connection
ambulance_policy = POSTObserver(f"http://{ambulance_host}",'ambulance_emergency.policy')
emergency_detect_out.subscribe(ambulance_policy)
```

2

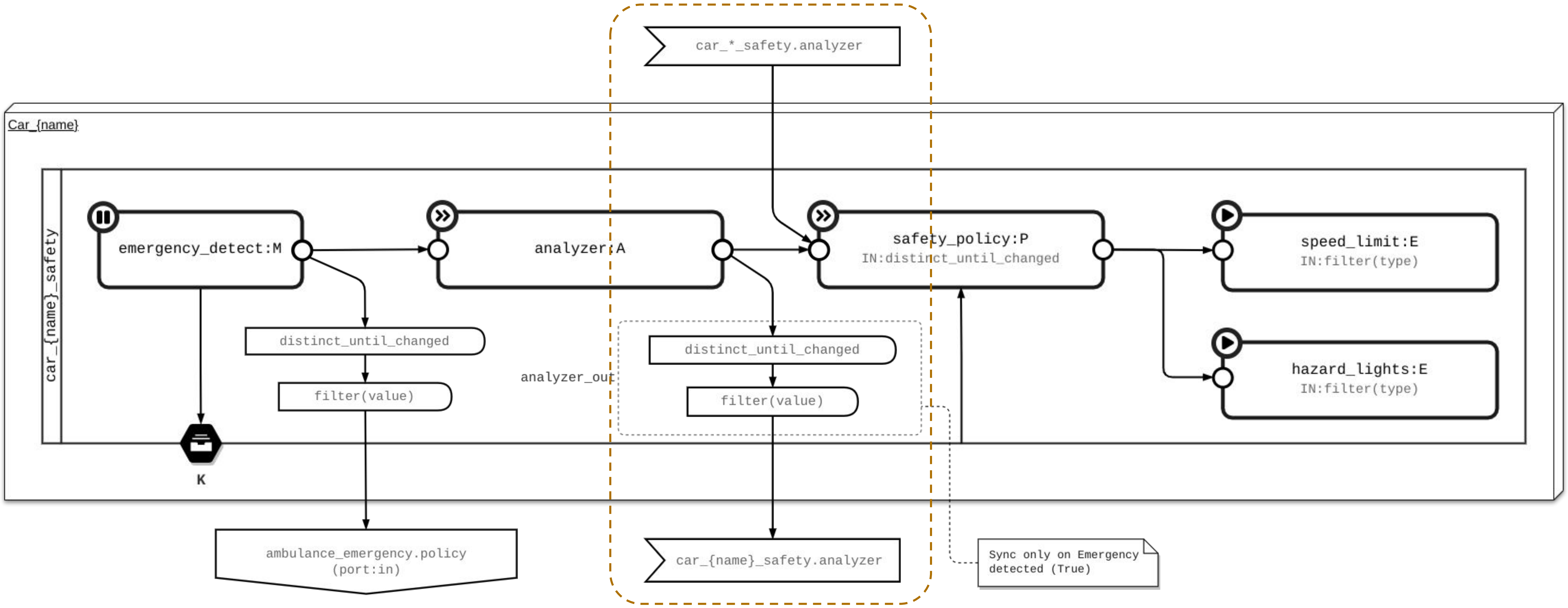
```python
# Object as item
@loop.execute(ops_in=ops.filter(lambda item: isinstance(item, SpeedItem)))
def speed_limit(item: SpeedItem, on_next):
    car.speed_limit = item.value

# Dict as item
@loop.execute(ops_in=ops.filter(lambda item: 'hazard_lights' in item))
def hazard_lights(item: dict, on_next):
    car.hazard_lights = item['hazard_lights']
```

1



AMBULANCE-CAR EMERGENCY

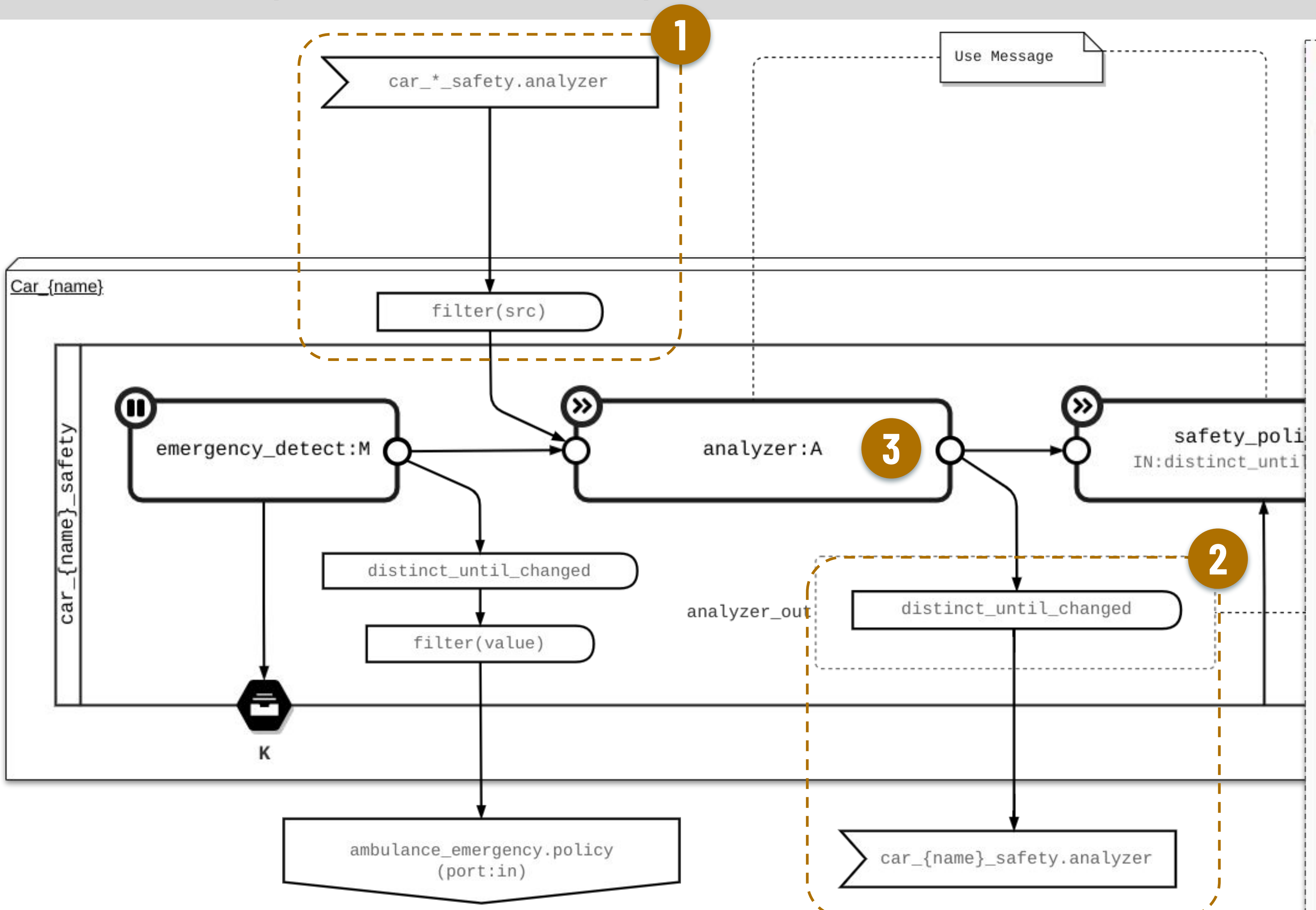
CAR to CAR (PUB/SUB) v1




```python
analyzer_out = analyzer.pipe(
    # Only when item change
    # Could be removed since is present on safety_policy (but reduce transmission bandwidth)
    ops.distinct_until_changed(),
    # Only when hazard_lights are ON
    # Comment out to sync car both Emergency ON and OFF
    ops.filter(lambda emergency_detect: emergency_detect is True)
)

# Send/Publish the analyzer output to others cars
analyzer_out.subscribe(PubObserver(analyzer.path))

# Listen/Subscribe for others cars analyzer output
# note: for clarity can be used "analyzer.uid" and "safety_policy.port_in"
SubObservable(f"car_*_safety.{analyzer}").subscribe(safety_policy)
```

AMBULANCE-CAR EMERGENCY

CAR to CAR (PUB/SUB) v2 - VOTING SYSTEM



3

```python
@loop.analyze
def analyzer(item: Message | Any, on_next, self):
    setdefaultattr(self, 'analyzer_msg_from', {})

    if isinstance(item, Message):
        # Item/packet come from other loops (Message)
        ...
        # Others cars emergency state
        self.analyzer_msg_from[item.src] = item.value
        count_true = list(self.analyzer_msg_from.values()).count(True)

        if count_true >= self.emergency_car_threshold_max:
            on_next(Message.create(True, src=self))
        elif count_true <= self.emergency_car_threshold_min:
            on_next(Message.create(False, src=self))
    else:
        # Item/packet come from emergency_detect
        ...
        on_next(Message.create(item, src=self))

analyzer.emergency_car_threshold_max = 2
analyzer.emergency_car_threshold_min = 0
```

1

```python
SubObservable(f"car_*_safety.{analyzer}").pipe(
    # Avoid message from same analyzer
    ops.filter(lambda item: item.src != analyzer.path)
).subscribe(analyzer)
```

2

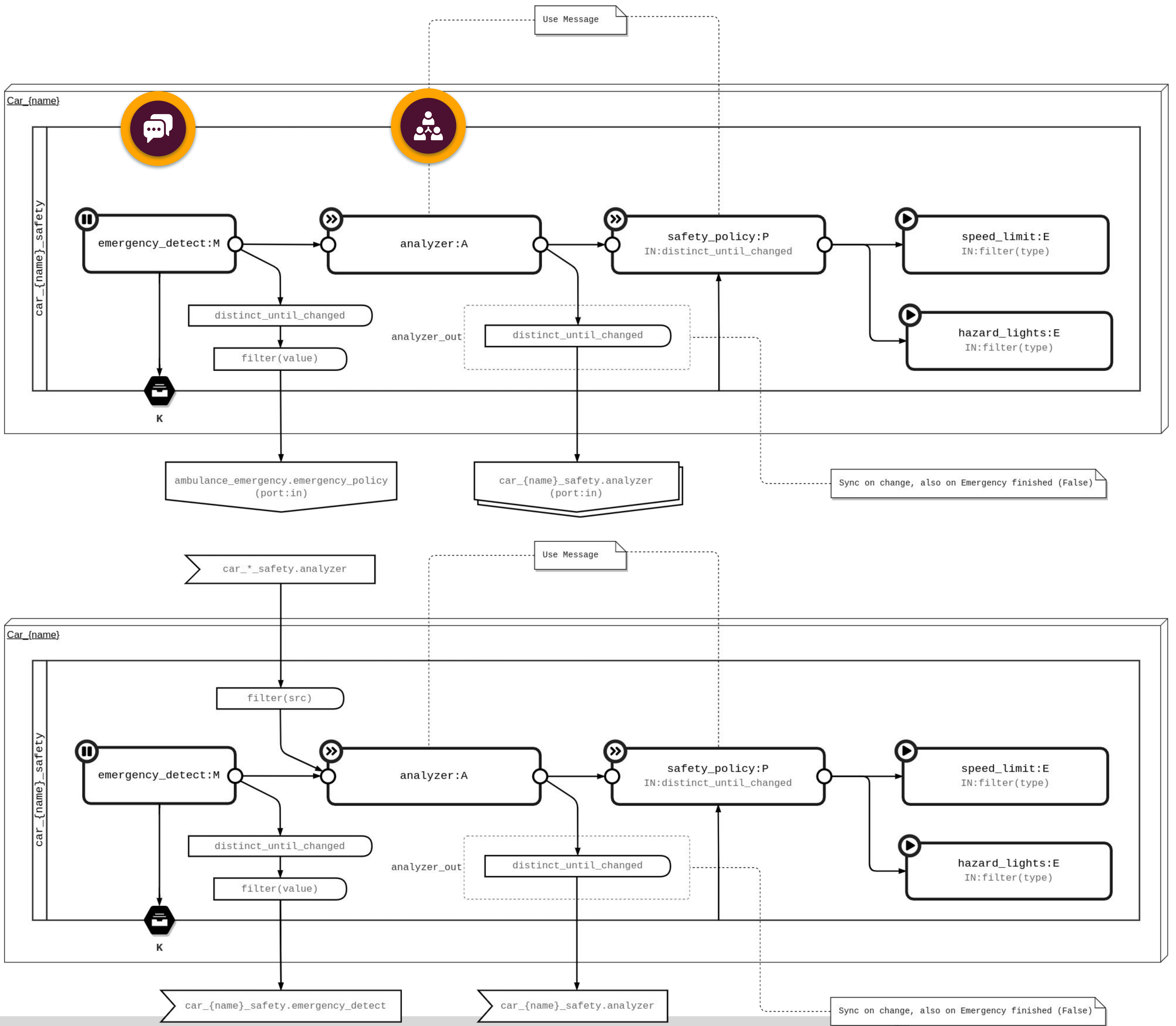
```python
analyzer_out = analyzer.pipe(
    # Only when local state change
    ops.distinct_until_changed(lambda item: item.value)
)

# Send/Publish the analyzer output to others cars
# (for clarity can be used "analyzer.port_out")
analyzer_out.subscribe(PubObserver(analyzer.path))
```

AMBULANCE-CAR EMERGENCY

FULL REST and PUB/SUB IMPLEMENTATION

REST

PUB/SUB

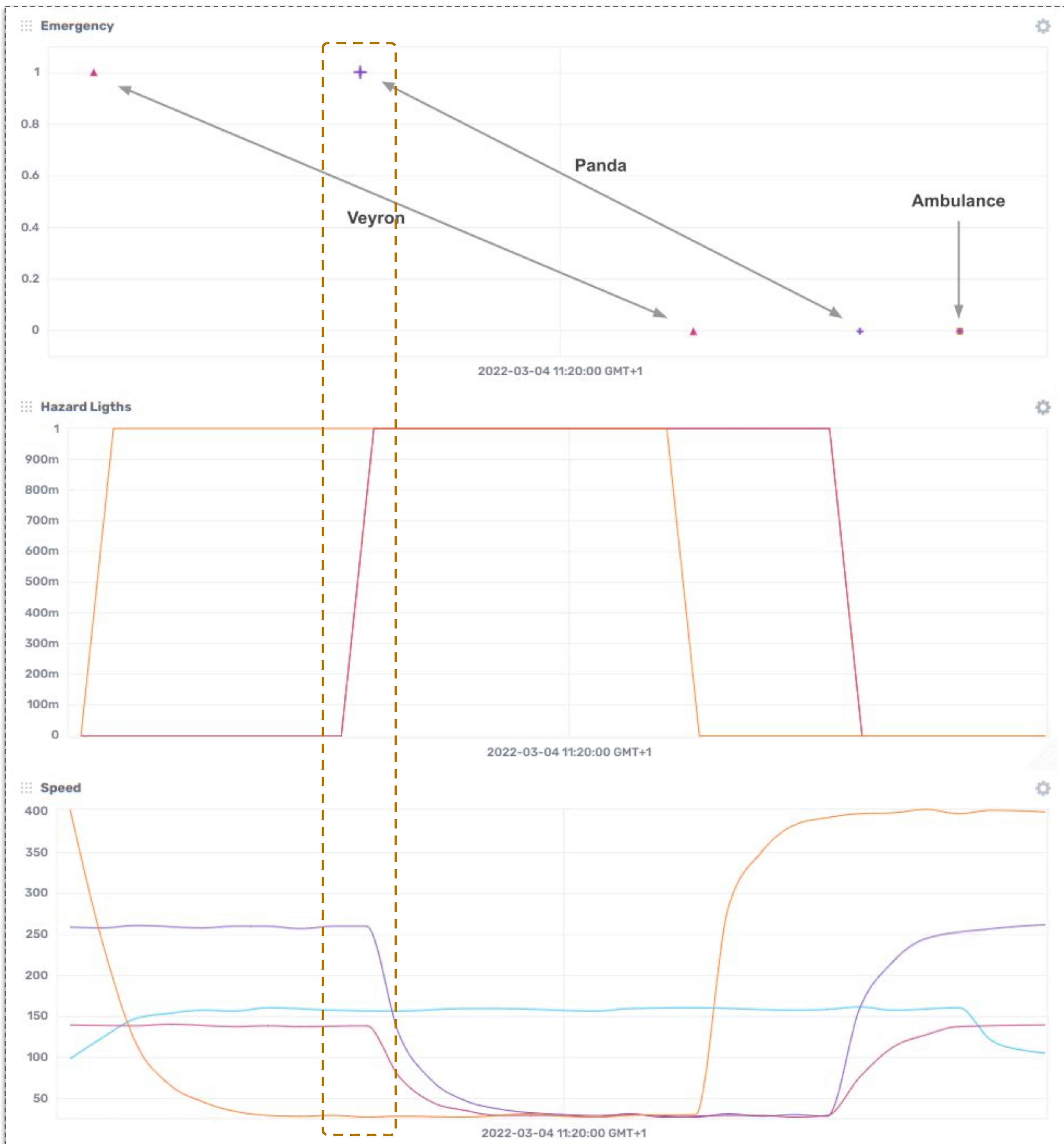


AMBULANCE-CAR EMERGENCY

RESULTS

GRAPH

Emergency, Hazard lights, Speed



DATA LOG

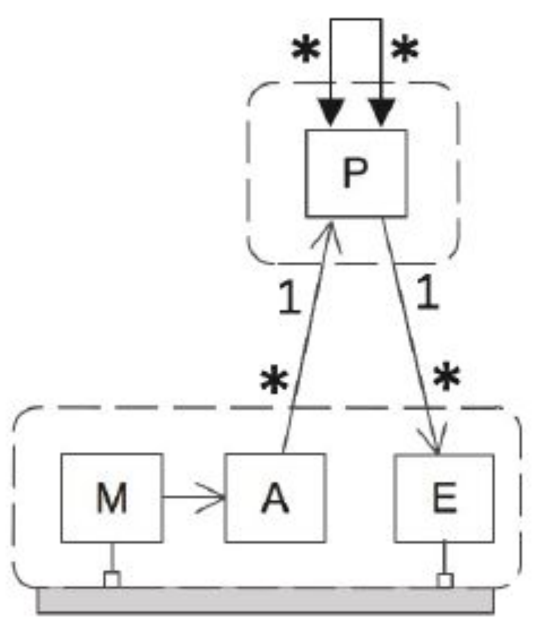
Ambulance, Veyron, Countach extract

```
$ python -m examples.coordinated-car-with-message --name Veyron --speed 380
10:38:35.645 M    examples.fixtures     : INFO     Init VirtualCar Veyron…
10:38:35.645 M    examples.fixtures     : INFO     VirtualCar Veyron speed: 380 Km/h
10:38:35.645 M    examples.fixtures     : INFO     VirtualCar Veyron speed limit set: 400 Km/h
10:38:35.645 M    examples.fixtures     : INFO     VirtualCar Veyron hazard lights OFF
10:38:35.645 M    examples.fixtures     : INFO     VirtualCar Veyron emergency OFF
10:38:35.767 M    examples.fixtures     : INFO     VirtualCar Veyron speed: 385 Km/h
10:38:38.818 M    examples.fixtures     : INFO     VirtualCar Veyron speed: 388 Km/h
10:38:42.223 M    examples.fixtures     : INFO     VirtualCar Veyron speed: 391 Km/h

$ python -m examples.coordinated-ambulance --speed 80
10:56:47.114 M    examples.fixtures     : INFO     Init VirtualAmbulance Ambulance…
10:56:47.114 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 80 Km/h
10:56:47.114 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed limit set: 100 Km/h
10:56:47.114 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance hazard lights OFF
10:56:47.114 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance emergency OFF
10:56:47.114 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance siren: OFF
10:56:47.232 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 85 Km/h
10:56:50.10  M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 88 Km/h
10:56:52.500 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 91 Km/h
[...]
11:17:32.944 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 99 Km/h
11:17:33.605 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed limit set: 160 Km/h
11:17:33.605 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance siren: ON
11:17:35.605 M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 114 Km/h
11:17:38.18  M    examples.fixtures     : INFO     VirtualAmbulance Ambulance speed: 125 Km/h

$ python -m examples.coordinated-car-with-message --name Countach --speed 240
[...]
11:18:59.257 M    examples.fixtures     : INFO     VirtualCar Countach speed: 260 Km/h
11:18:59.882 M    root                  : INFO     [('Veyron', True), ('Panda', True)]
11:18:59.883 M    examples.fixtures     : INFO     VirtualCar Countach speed limit set: 30 Km/h
11:18:59.883 M    examples.fixtures     : INFO     VirtualCar Countach hazard lights ON
11:19:02.100 M    examples.fixtures     : INFO     VirtualCar Countach speed: 203 Km/h
11:19:05.720 M    examples.fixtures     : INFO     VirtualCar Countach speed: 160 Km/h
```

REGIONAL PLANNING

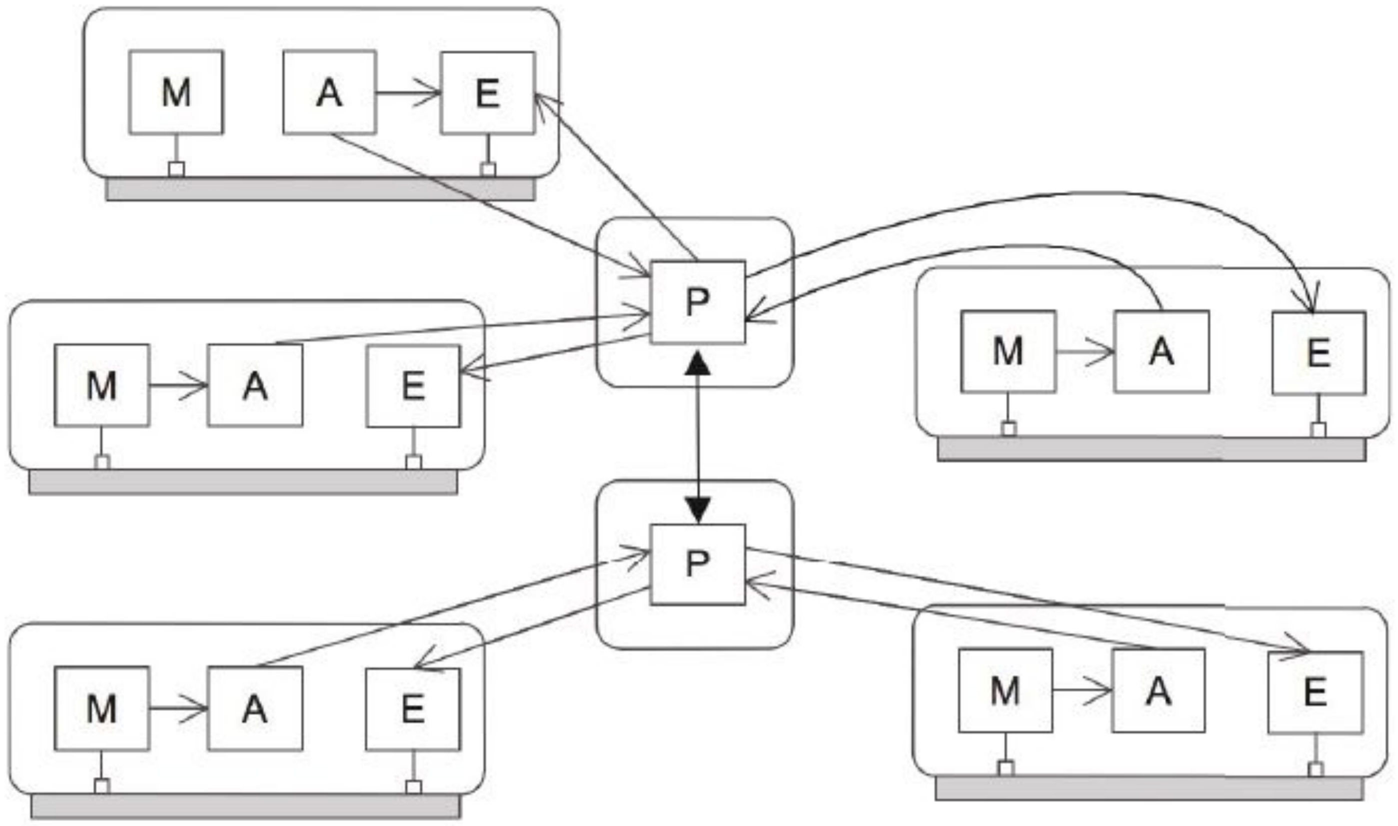
MAPE PATTERN

PATTERN



$(|MAE| \geq |P|) \wedge (|P| = |regions|)$

INSTANCE EXAMPLE

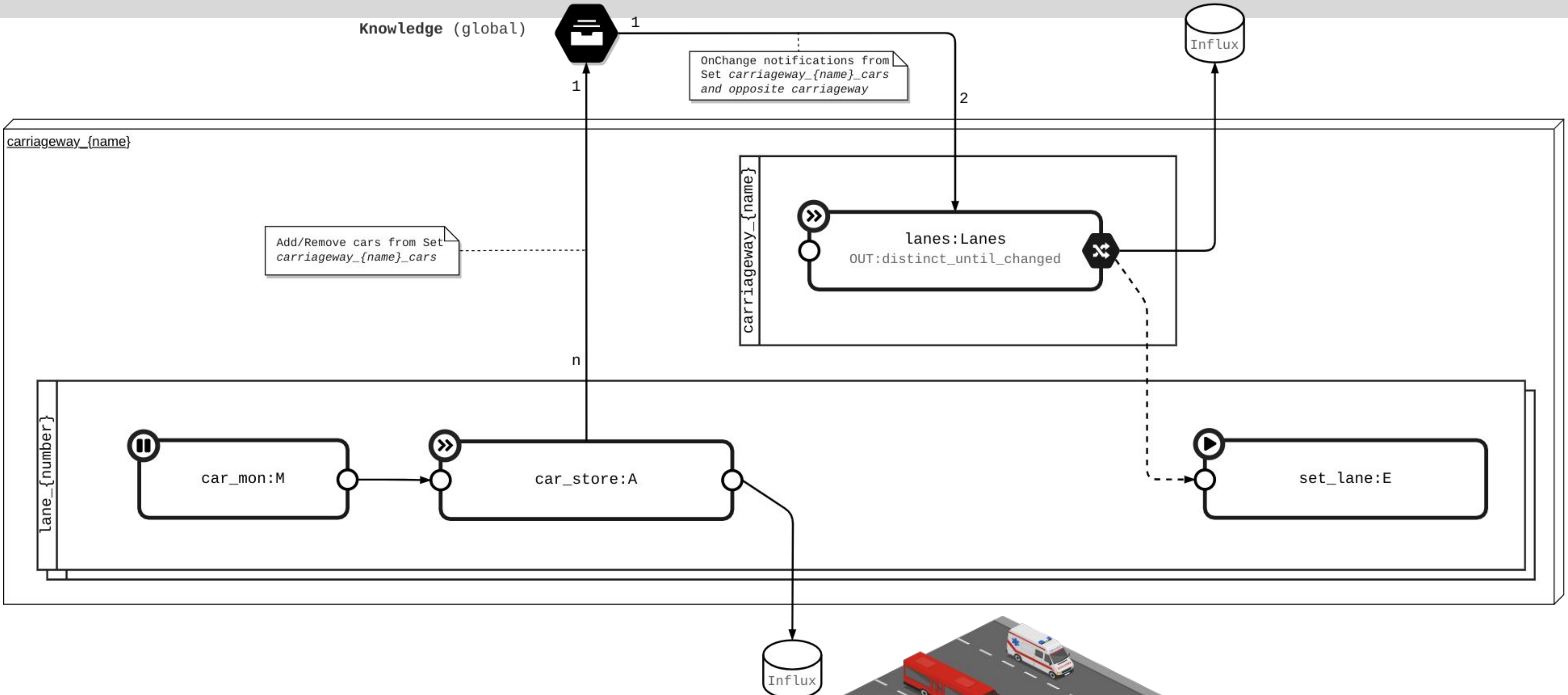


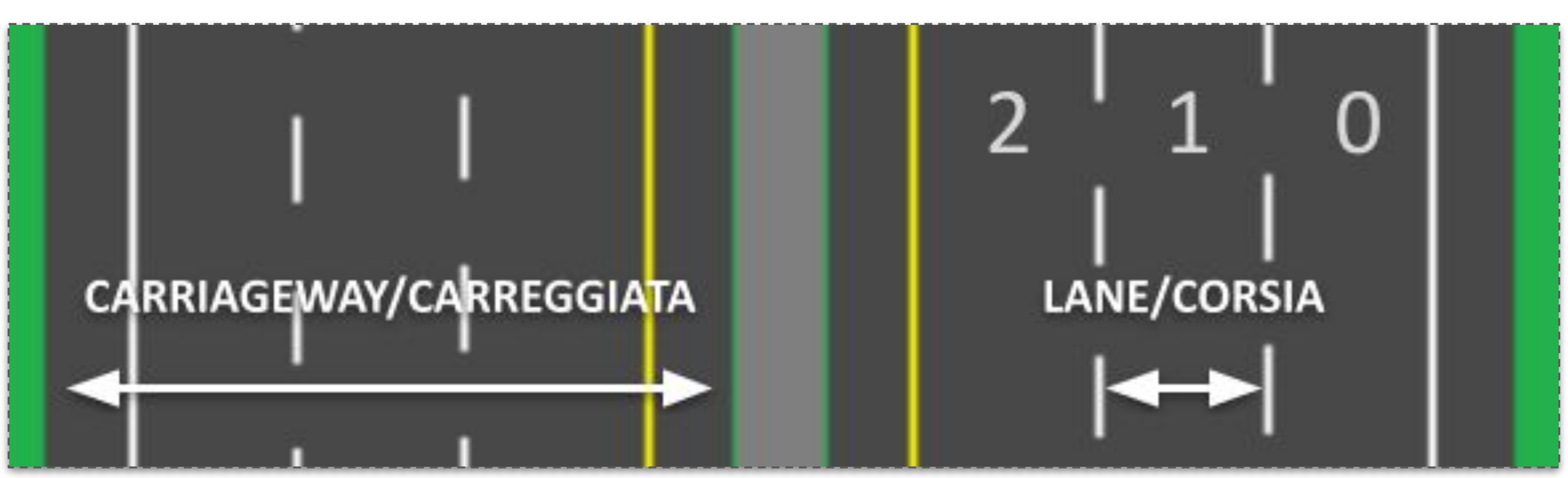
Different loosely coupled parts (**regions**) of a system want to realize **local adaptations** (within a region) as well as adaptations that **cross the boundaries** (between regions).

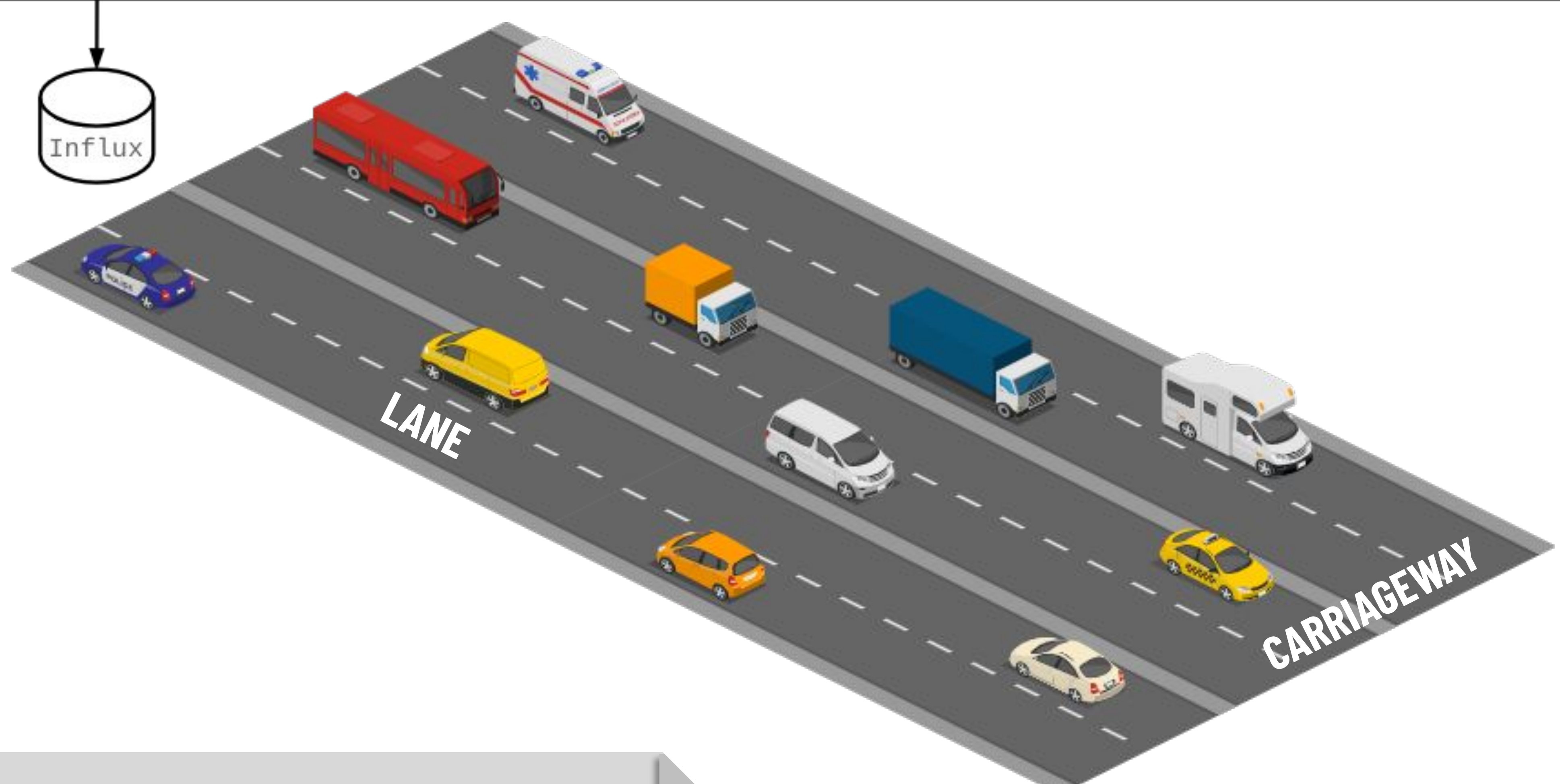
For each region, the M components monitor the **local status** of managed systems, the **local** A analyzes and reports the information to the associated regional planner. P may then decide to perform a local adaptation (i.e. **within the region**) or interact with another to plan adaptations that **span over** the regions. Once the planners agree on a plan the adaptations are achieved by activating the E components of the respective region.

DYNAMIC CARRIAGEWAY

OUTLINE

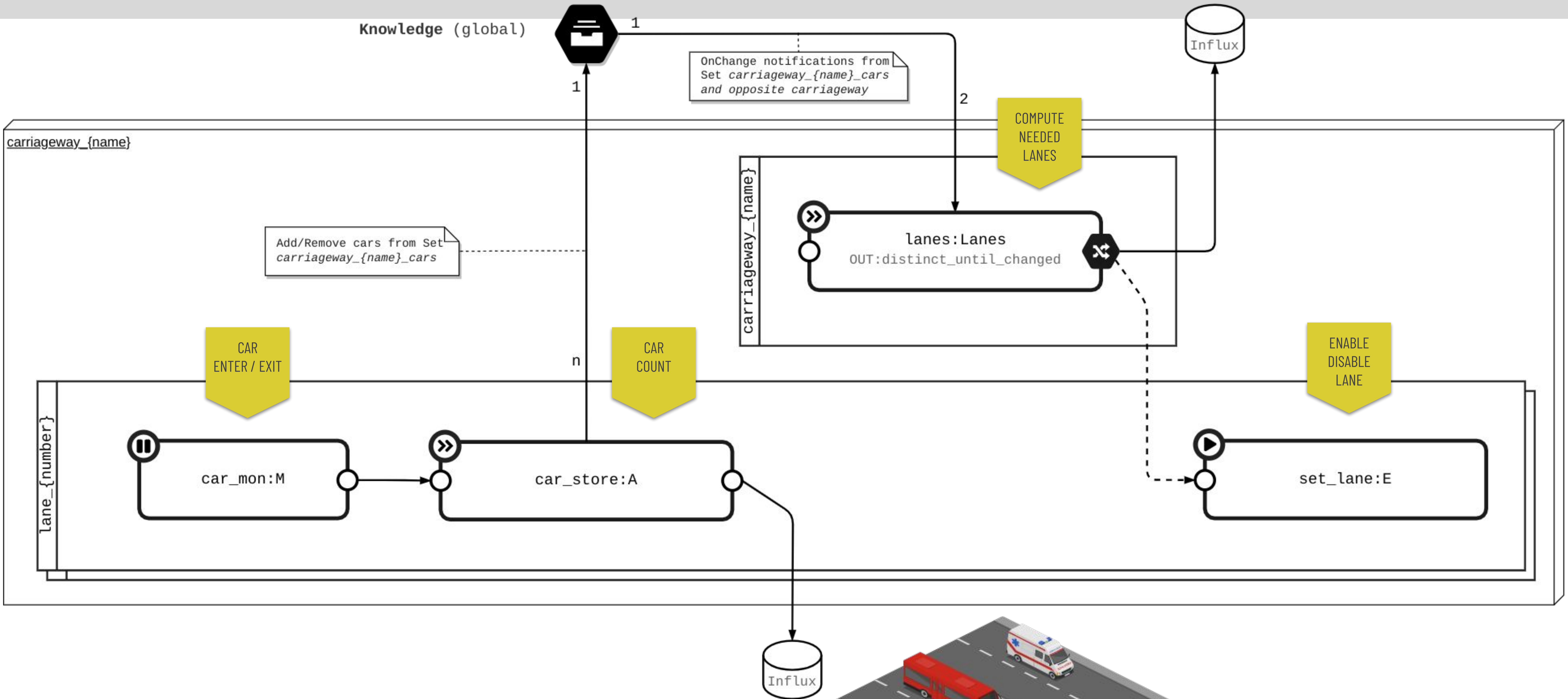


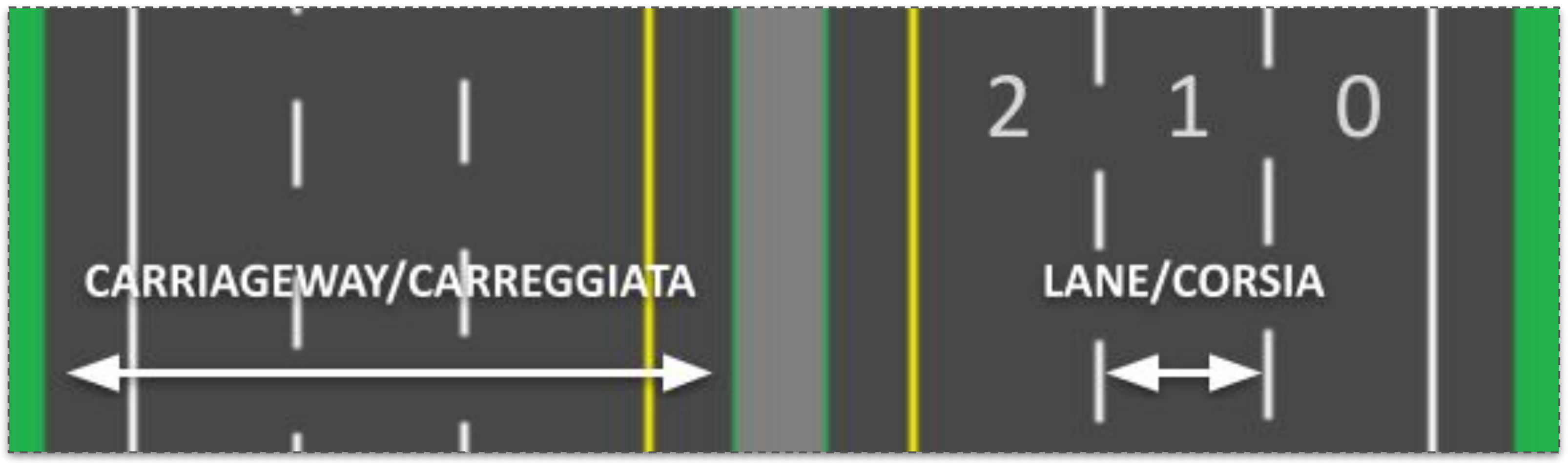


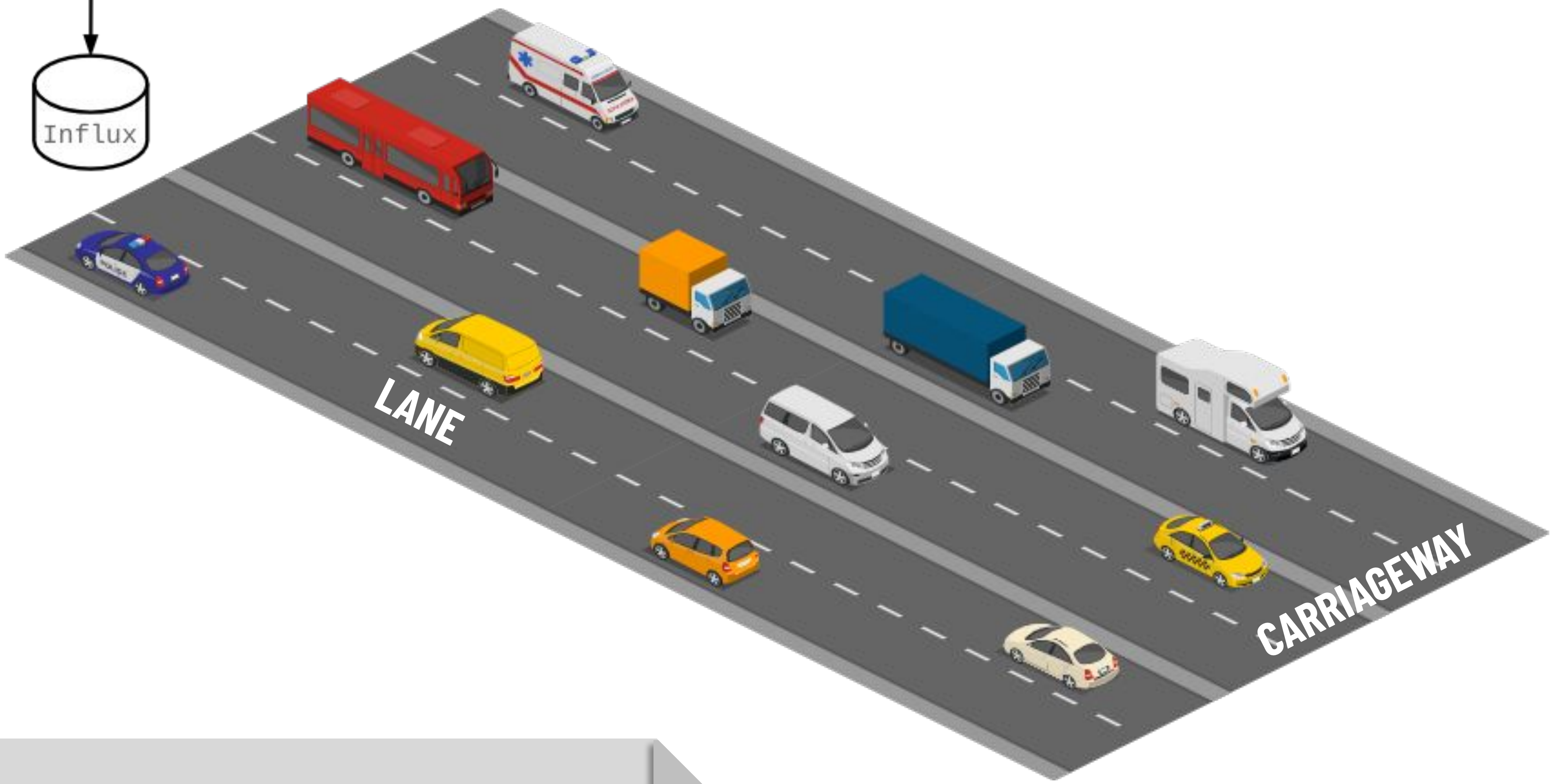


DYNAMIC CARRIAGEWAY

OUTLINE

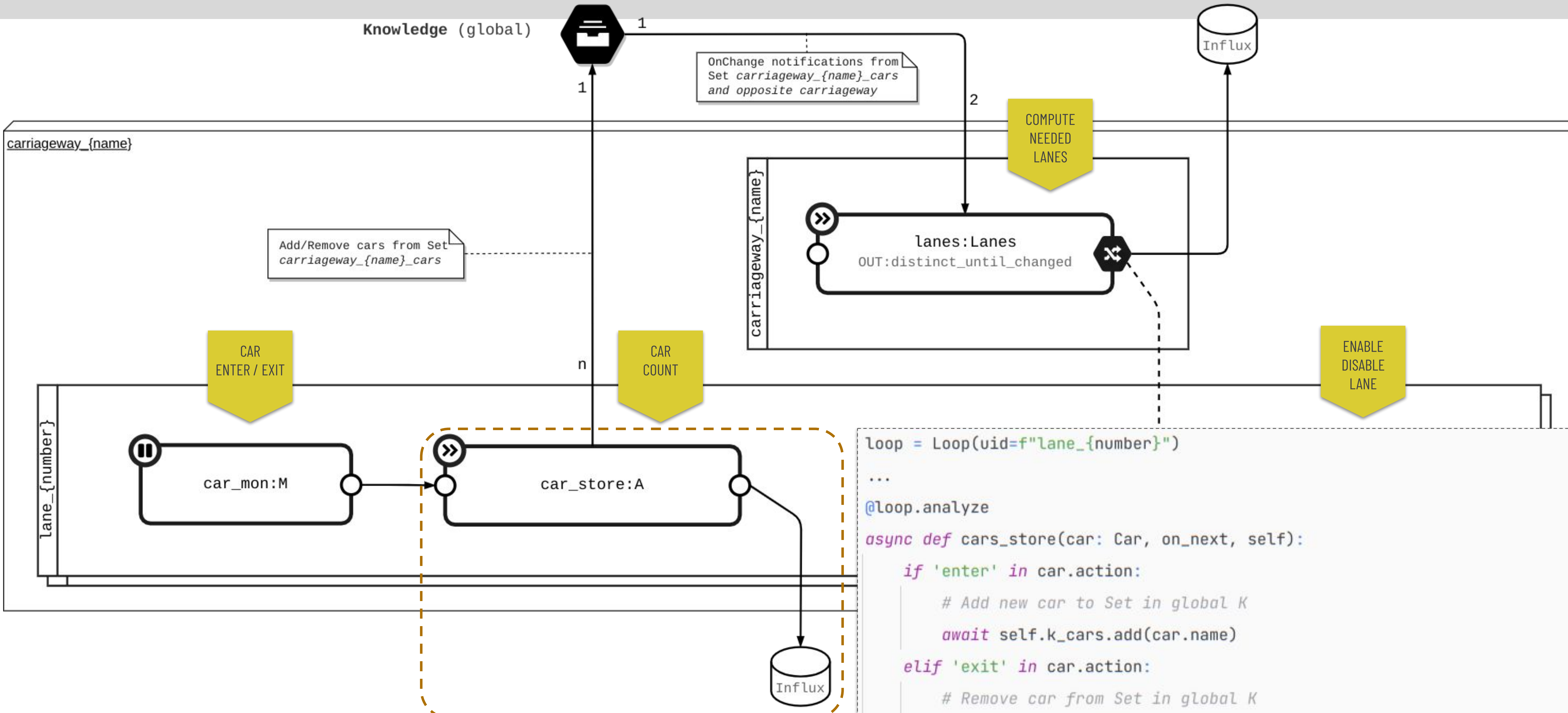






DYNAMIC CARRIAGEWAY

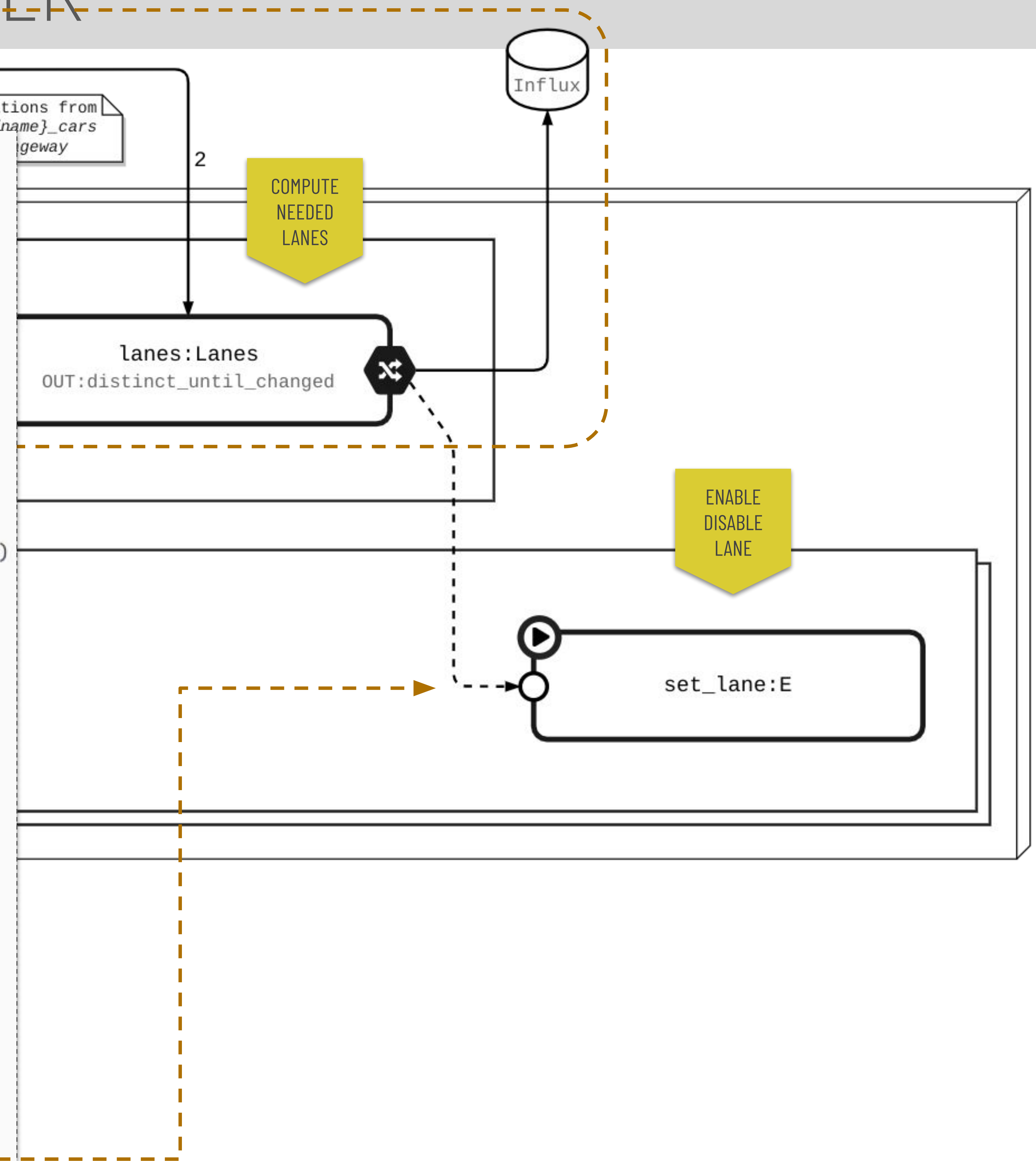
LOCAL LANE ANALYZER




```python
loop = Loop(uid=f"lane_{number}")
...
@loop.analyze
async def cars_store(car: Car, on_next, self):
    if 'enter' in car.action:
        # Add new car to Set in global K
        await self.k_cars.add(car.name)
    elif 'exit' in car.action:
        # Remove car from Set in global K
        await self.k_cars.remove(car.name)

    # Count current cars in carriageway
    car_count = await self.k_cars.len()
    on_next(car_count)
...
# Use InfluxDB sink/terminator to store number of cars
cars_store.subscribe(InfluxObserver(tags=('type', f"cars_{carriageway}")))
```

DYNAMIC CARRIAGEWAY

REGIONAL CARRIAGEWAY PLANNER



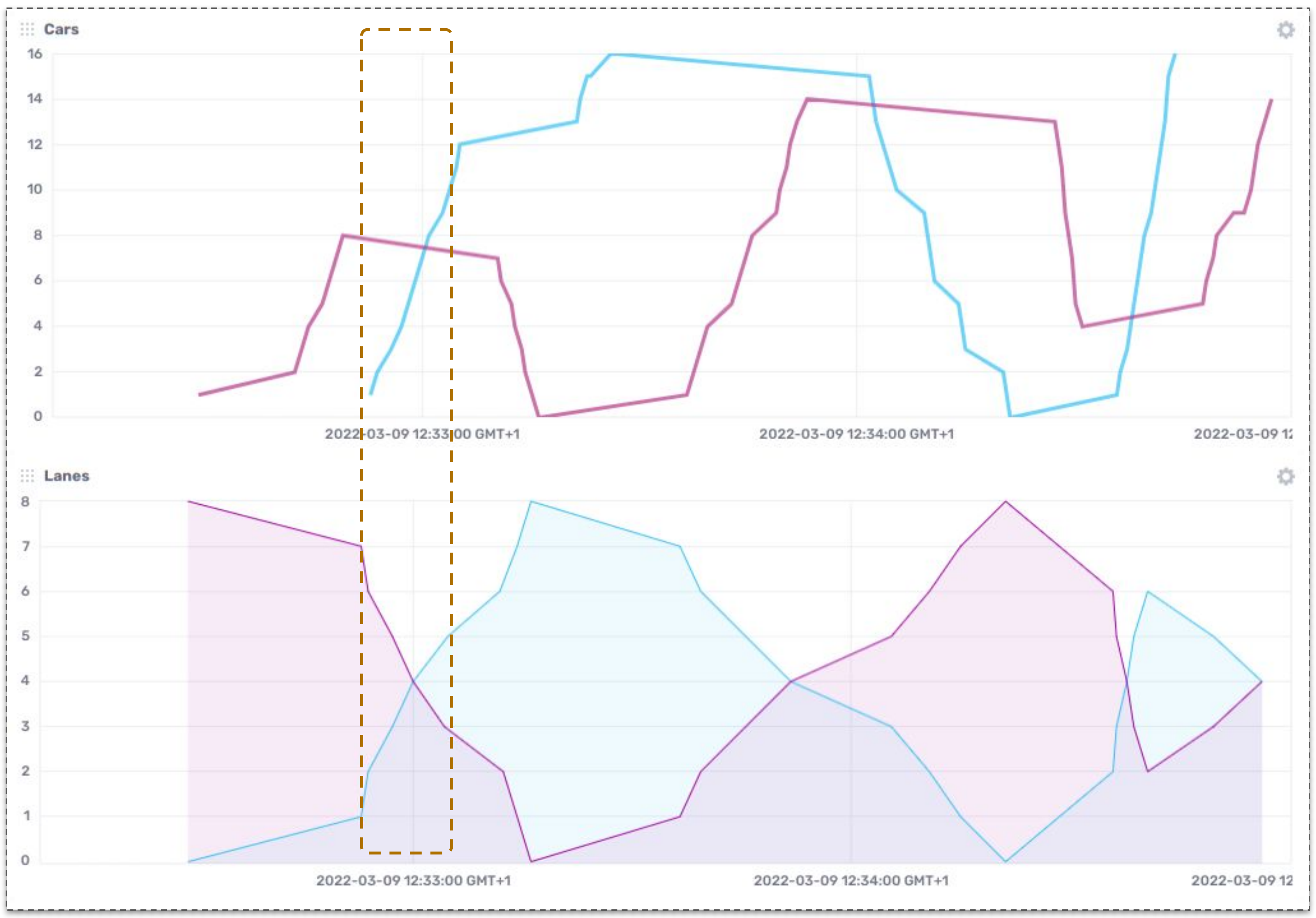

```python
loop = Loop(uid=f"carriageway_{name}")

class Lanes(Plan):
    def __init__(self, loop, opposite_carriageway, max_lanes=8, uid=None):
        super().__init__(loop, uid, ops_out=(
            ops.distinct_until_changed(lambda item: item.value), ops.router()
        ))

        # Get access to Sets (up and down) in the global K
        self.k_cars = self.loop.app.k.create_set(f"{self.loop}_cars", str)
        self.k_cars_opposite = self.loop.app.k.create_set(f"{opposite_carriageway}_cars", str)

        # Register handler for add (sadd) / remove (srem) cars
        self.loop.app.k.notifications(self._on_cars_change, f"carriageway_*_cars",
                                      cmd_filter=('sadd', 'srem'))

    async def _on_cars_change(self, message):
        # Count cars in Sets
        cars = await self.k_cars.len()
        opposite_cars = await self.k_cars_opposite.len()

        # Compute new carriageway number of lanes (simplified)
        lanes = round(self.max_lanes / (cars + opposite_cars) * cars)
        ...
        # Send to the lane executer (lane_{number}.set_lane)
        self._on_next(Message.create(lanes, src=self, dst=f"lane_{lanes}.set_lane"))

# Instance lanes planner
lanes = Lanes(loop, opposite_carriageway, max_lanes=count_lanes, uid='lanes')

# Use InfluxDB sink/terminator to store number of lanes
lanes.subscribe(InfluxObserver())
```

DYNAMIC CARRIAGEWAY

RESULTS

GRAPH

Cars count, Lanes count per carriageway



DATA LOG

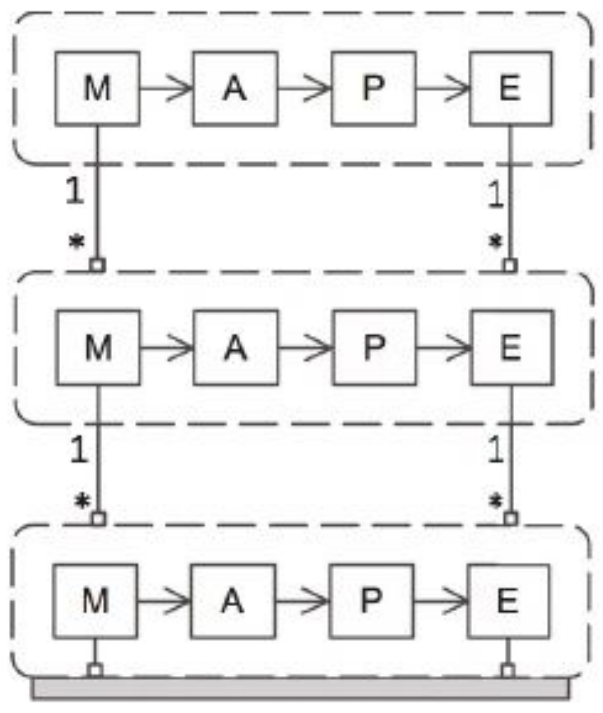
Up & Down carriageway extract

```
$ python -m examples.regional-planning-dynamic-carriageway --name up --lanes 8
12:32:28.862 M   root                : DEBUG    lane_0 | enter | Pilot        |   1 (tot)
12:32:28.866 M   examples.fixtures   : INFO     carriageway_up has 8 lanes
12:32:42.111 M   root                : DEBUG    lane_7 | enter | Silver       |   2 (tot)
12:32:43.142 M   root                : DEBUG    lane_7 | enter | Cascada      |   3 (tot)
12:32:44.31  M   root                : DEBUG    lane_7 | enter | Fox          |   4 (tot)
12:32:45.816 M   root                : DEBUG    lane_1 | enter | Levante      |   5 (tot)
12:32:46.677 M   root                : DEBUG    lane_0 | enter | Phoenix      |   6 (tot)
12:32:47.627 M   root                : DEBUG    lane_3 | enter | Nova         |   7 (tot)
12:32:48.655 M   root                : DEBUG    lane_4 | enter | Avenger      |   8 (tot)
12:32:52.466 M   examples.fixtures   : INFO     carriageway_up has 7 lanes
12:32:53.722 M   examples.fixtures   : INFO     carriageway_up has 6 lanes
12:32:56.807 M   examples.fixtures   : INFO     carriageway_up has 5 lanes
12:32:59.778 M   examples.fixtures   : INFO     carriageway_up has 4 lanes
12:33:04.242 M   examples.fixtures   : INFO     carriageway_up has 3 lanes
12:33:10.68  M   root                : DEBUG    lane_1 | exit  | Levante      |   7 (tot)
12:33:10.846 M   root                : DEBUG    lane_2 | exit  | Phoenix      |   6 (tot)
12:33:11.847 M   root                : DEBUG    lane_0 | exit  | Cascada      |   5 (tot)
12:33:11.852 M   examples.fixtures   : INFO     carriageway_up has 2 lanes
12:33:12.687 M   root                : DEBUG    lane_0 | exit  | Nova         |   4 (tot)
12:33:13.445 M   root                : DEBUG    lane_1 | exit  | Pilot        |   3 (tot)
12:33:14.166 M   root                : DEBUG    lane_0 | exit  | Silver       |   2 (tot)
12:33:14.167 M   examples.fixtures   : INFO     carriageway_up has 1 lanes
12:33:14.990 M   root                : DEBUG    lane_0 | exit  | Fox          |   1 (tot)
12:33:15.839 M   root                : DEBUG    lane_0 | exit  | Avenger      |   0 (tot)
12:33:15.843 M   examples.fixtures   : INFO     carriageway_up has 0 lanes

$ python -m examples.regional-planning-dynamic-highway --name down --lanes 8
12:32:28.856 M   examples.fixtures   : INFO     carriageway_down has 0 lanes
12:32:52.468 M   root                : DEBUG    lane_0 | enter | Zephyr       |   1 (tot)
12:32:52.469 M   examples.fixtures   : INFO     carriageway_down has 1 lanes
12:32:53.727 M   root                : DEBUG    lane_0 | enter | Outlander    |   2 (tot)
12:32:53.730 M   examples.fixtures   : INFO     carriageway_down has 2 lanes
12:32:55.429 M   root                : DEBUG    lane_0 | enter | Frontier     |   3 (tot)
12:32:56.809 M   root                : DEBUG    lane_0 | enter | Verano       |   4 (tot)
12:32:56.810 M   examples.fixtures   : INFO     carriageway_down has 3 lanes
12:32:57.895 M   root                : DEBUG    lane_2 | enter | Summit       |   5 (tot)
12:32:58.888 M   root                : DEBUG    lane_2 | enter | Cimarron     |   6 (tot)
12:32:59.785 M   root                : DEBUG    lane_1 | enter | Matrix       |   7 (tot)
12:32:59.789 M   examples.fixtures   : INFO     carriageway_down has 4 lanes
12:33:00.624 M   root                : DEBUG    lane_0 | enter | Blackwood    |   8 (tot)
12:33:02.704 M   root                : DEBUG    lane_1 | enter | Pininfarina  |   9 (tot)
12:33:03.432 M   root                : DEBUG    lane_0 | enter | Scrambler    |  10 (tot)
12:33:04.250 M   root                : DEBUG    lane_2 | enter | Alero        |  11 (tot)
12:33:04.255 M   examples.fixtures   : INFO     carriageway_down has 5 lanes
12:33:05.140 M   root                : DEBUG    lane_4 | enter | Accent       |  12 (tot)
12:33:11.842 M   examples.fixtures   : INFO     carriageway_down has 6 lanes
12:33:14.164 M   examples.fixtures   : INFO     carriageway_down has 7 lanes
12:33:15.834 M   examples.fixtures   : INFO     carriageway_down has 8 lanes
```

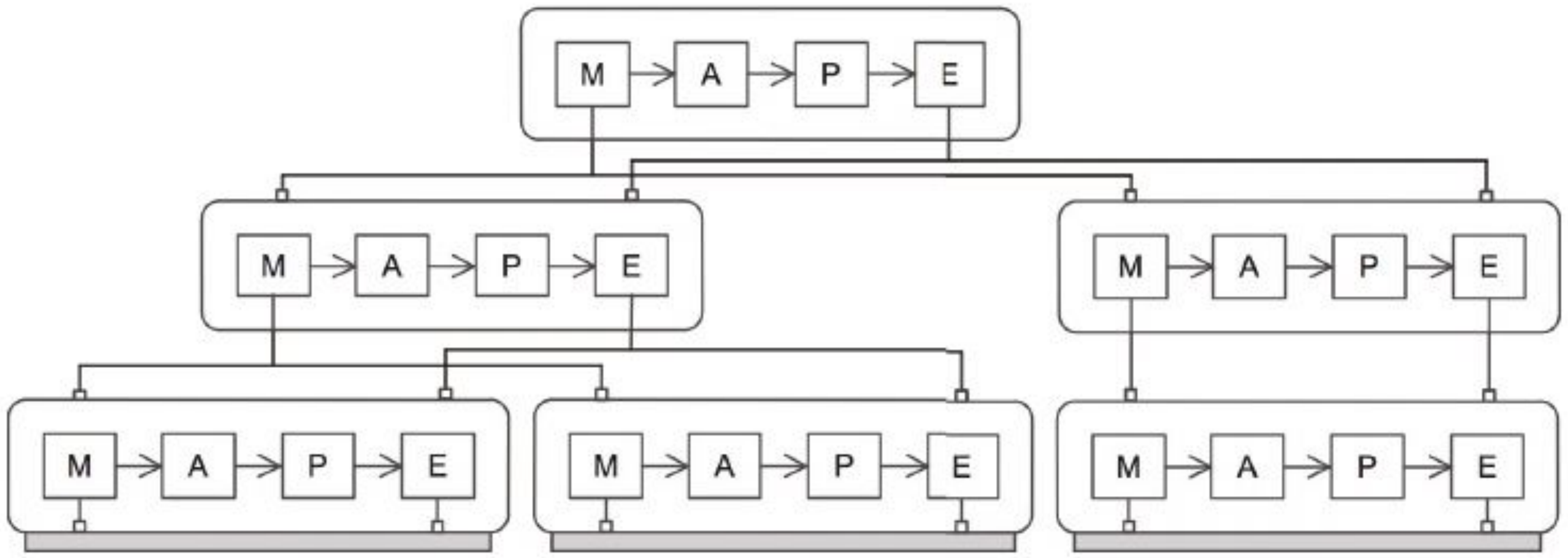
HIERARCHICAL CONTROL

MAPE PATTERN

PATTERN



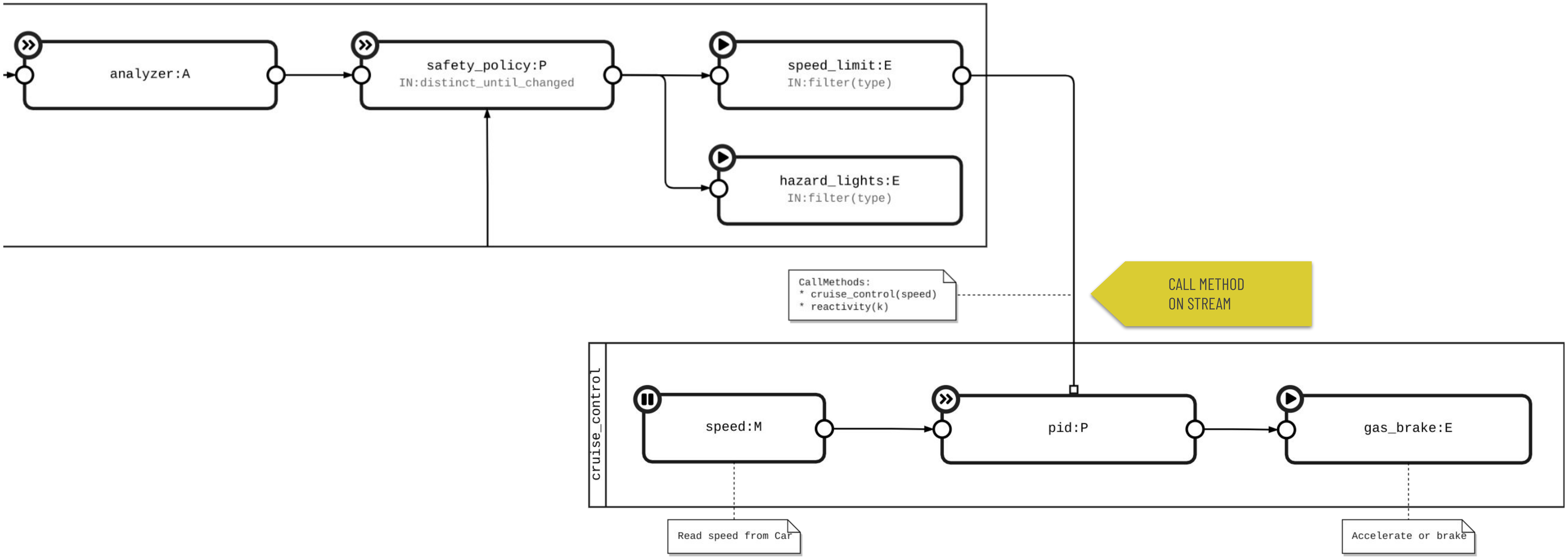
INSTANCE EXAMPLE



In a complex system is often necessary to consider **multiple control loops** within the same application. The loops can work at different **time scales** and manage different **kind of resources**, with different **localities**.

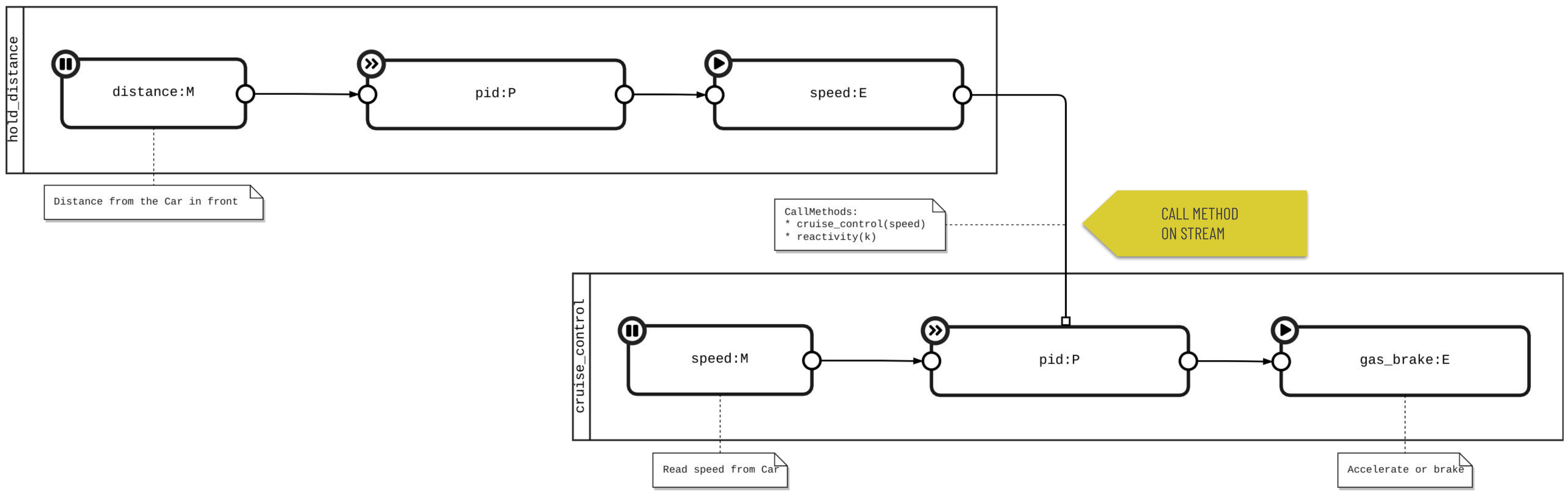
Different layers typically focus on different **concerns** at different **levels** of abstraction. The hierarchical structure allows bottom layer to focus on concrete adaptation objectives, while higher level can take increasingly broader perspectives, using lower control loop as managed resource.

CRUISE CONTROL + CAR EMERGENCY



CRUISE CONTROL WITH DISTANCE HOLD

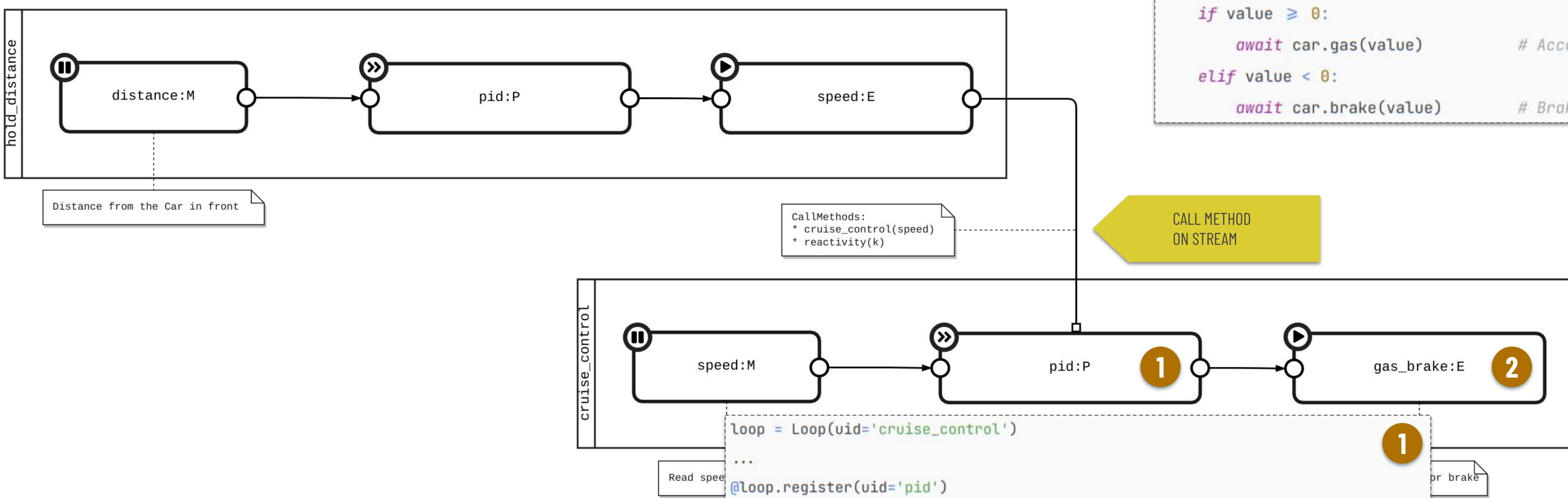
OUTLINE



CRUISE CONTROL WITH DISTANCE HOLD

CRUISE CONTROL


```python
@loop.execute
async def gas_brake(value, on_next):
    if value ≥ 0:
        await car.gas(value)          # Accelerate
    elif value < 0:
        await car.brake(value)        # Brake
```

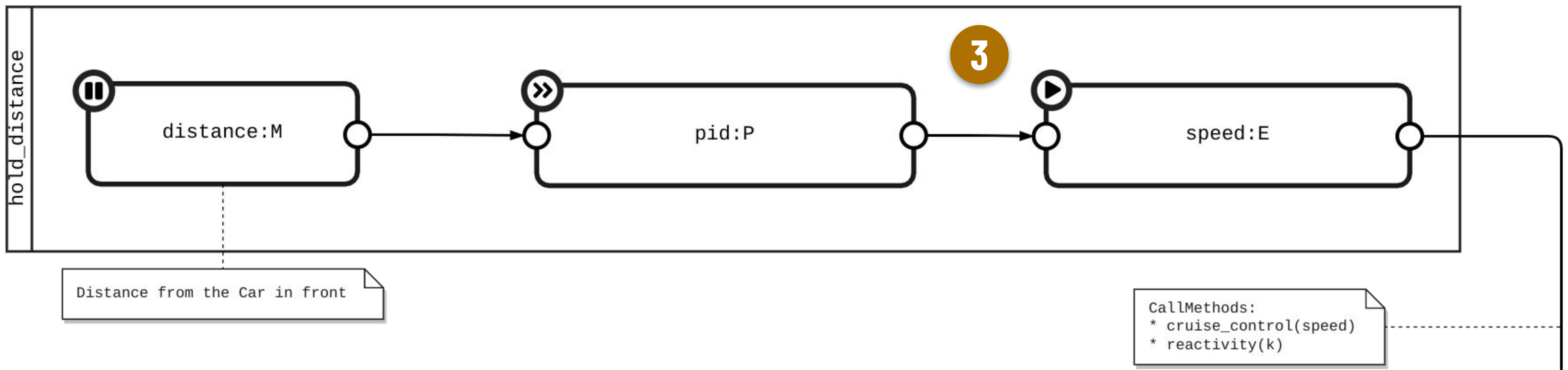



```python
loop = Loop(uid='cruise_control')
...
@loop.register(uid='pid')
class Pid(Plan):
    def __init__(self, loop, speed_target=100, uid=None):
        # Init and configure the pid
        # Set coefficients for proportional, integral derivative terms and speed
        self.pid = PID(4, 0.01, 0.1, setpoint=speed_target)
        # Limit the returned values from pid (ie. car Brake capabilities and Hp)
        self.pid.output_limits = (-car.max_break, car.max_power)

        super().__init__(loop, uid=uid)

    def cruise_control(self, speed_target):
        self.pid.setpoint = speed_target

    def _on_next(self, current_speed, on_next=None, *args, **kwargs):
        # Calculate the acceleration or brake needed, through pid controller
        # https://en.wikipedia.org/wiki/PID_controller
        power = self.pid(current_speed)
        on_next(power)
```

CRUISE CONTROL WITH DISTANCE HOLD

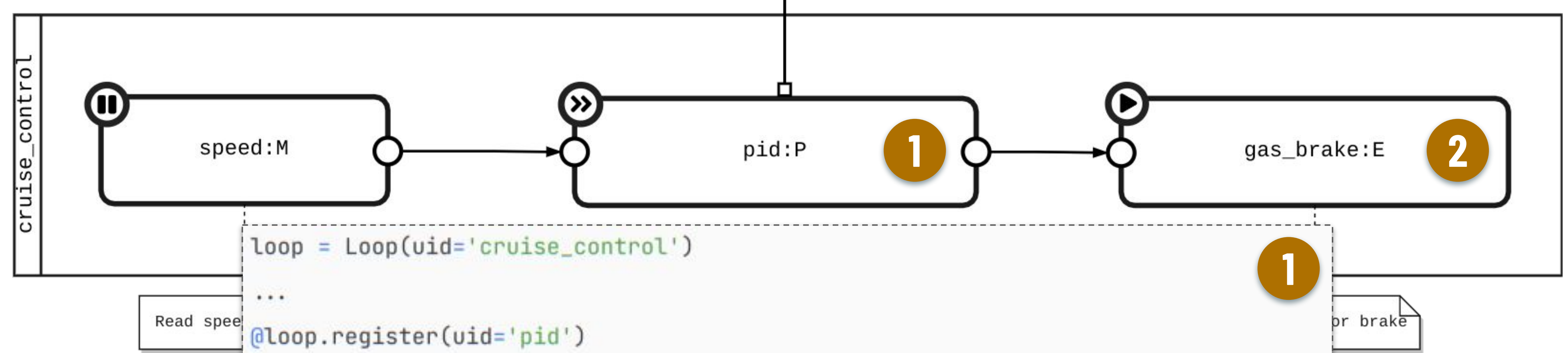
HOLD DISTANCE


```python
@loop.execute
async def gas_brake(value, on_next):
    if value ≥ 0:
        await car.gas(value)        # Accelerate
    elif value < 0:
        await car.brake(value)      # Brake
```



CALL METHOD ON STREAM

```python
loop = Loop(uid='hold_distance')
...
@loop.plan()
def pid(distance, on_next, self):
    # Calculate the speed, through pid controller
    speed = self.pid(distance)
    on_next(speed)

distance_target = 250
pid.pid = PID(-5, -0.01, -0.1, setpoint=distance_target)
# Limit the speed range returned from pid
pid.pid.output_limits = (0, 160)

@loop.execute
def speed(speed, on_next):
    on_next(CallMethod.create('cruise_control', speed))
...
# Hierarchical connection
speed.subscribe(mape.app.cruise_control.pid)
```



CALL METHOD ON STREAM

```python
loop = Loop(uid='cruise_control')
...
@loop.register(uid='pid')
class Pid(Plan):
    def __init__(self, loop, speed_target=100, uid=None):
        # Init and configure the pid
        # Set coefficients for proportional, integral derivative terms and speed
        self.pid = PID(4, 0.01, 0.1, setpoint=speed_target)
        # Limit the returned values from pid (ie. car Brake capabilities and Hp)
        self.pid.output_limits = (-car.max_break, car.max_power)

        super().__init__(loop, uid=uid)

    def cruise_control(self, speed_target):
        self.pid.setpoint = speed_target

    def _on_next(self, current_speed, on_next=None, *args, **kwargs):
        # Calculate the acceleration or brake needed, through pid controller
        # https://en.wikipedia.org/wiki/PID_controller
        power = self.pid(current_speed)
        on_next(power)
```

CRUISE CONTROL WITH DISTANCE HOLD

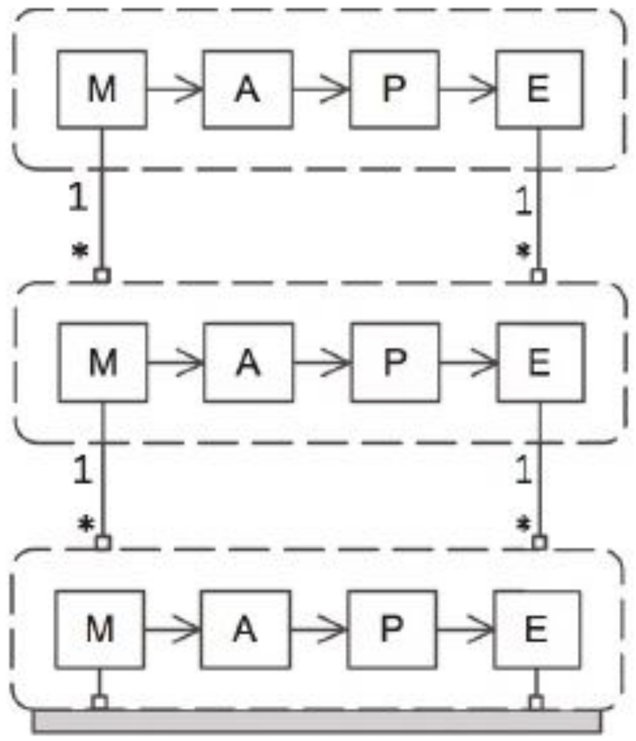
COUNTACH vs PAND (PURSUIT)

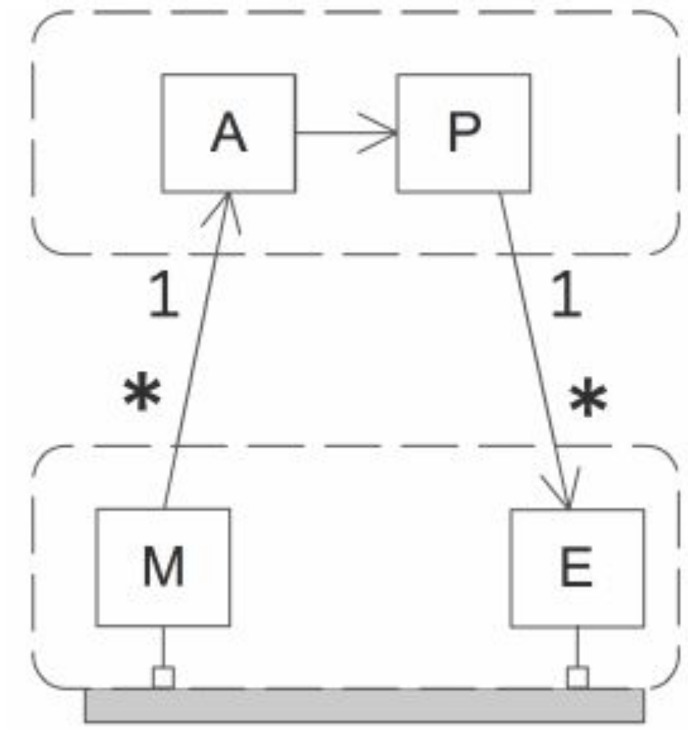
GRAPH

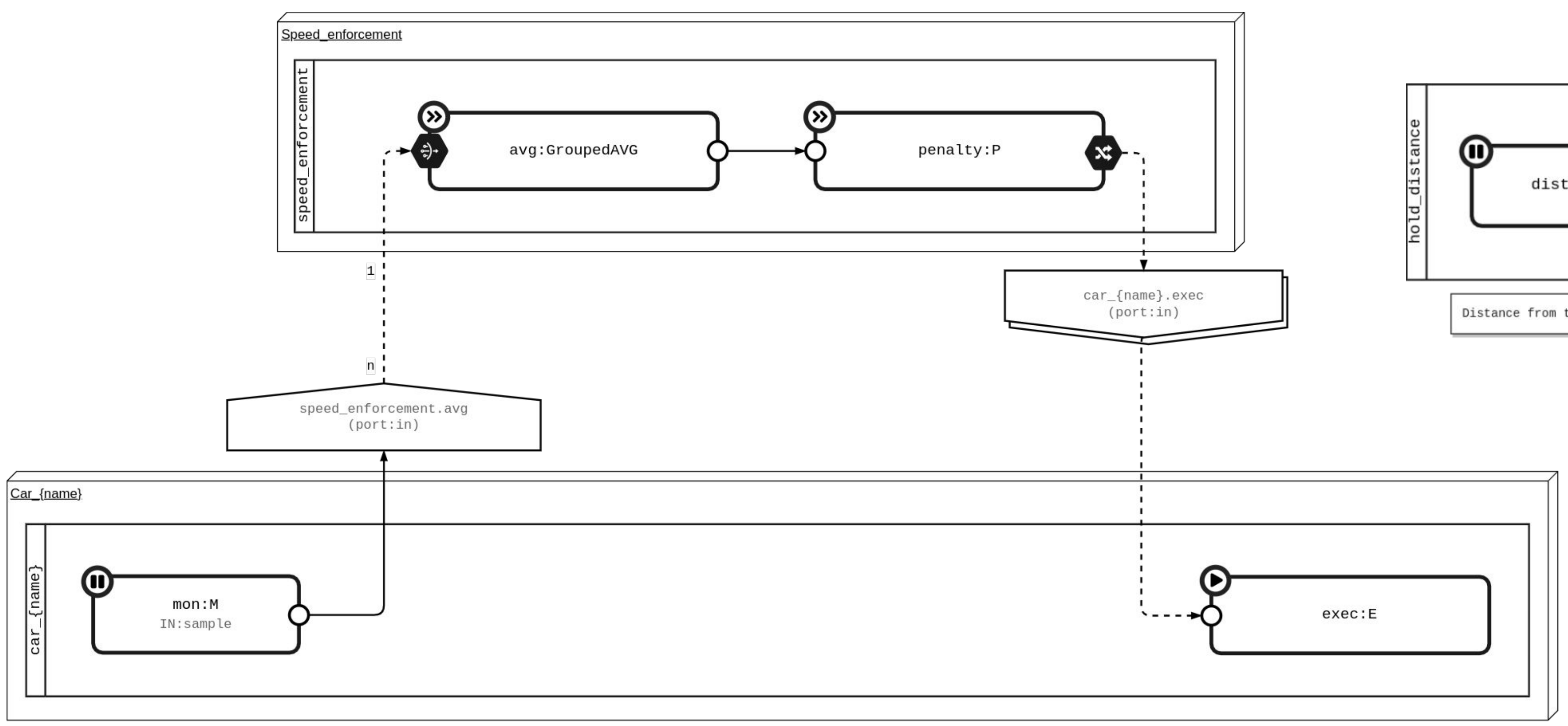
Distance, Speed, Accelerator/Brake

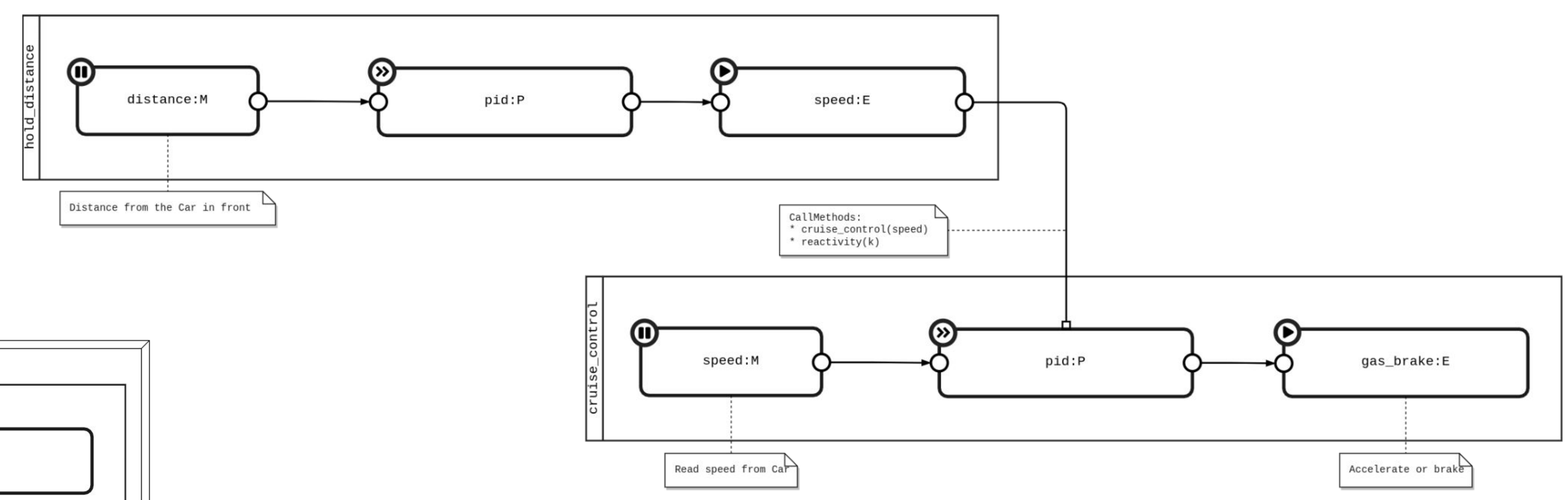
CARS SPECS & SETUP

	Power (Hp)	Brake	Init Speed (Km/h)	Cruise range (Km/h)	Cruise distance (meters)
Panda	70	70	0	0 - 160	250
Countach	200	180	0	x	x









MASTER/SLAVE
AVERAGE SPEED ENFORCEMENT

HIERARCHICAL CONTROL
CRUISE CONTROL WITH DISTANCE HOLD

CONCLUSIONS AND FUTURE WORK

CONCLUSIONS

FRAMEWORK EXPRESSIVITY

Allows without effort, the implementation of different grades of decentralization with the 5 main MAPE patterns used as a playground

STREAM

Allows connection between loosely coupled components, mutable at runtime, further the classic flow configuration (M ⇨ A ⇨ P ⇨ E)

COMMUNICATION ABSTRACTION

Allows use of multiple network communication paradigms with minimal change to code

FUTURE WORK

DSL & M2T

Graphical notation 1:1 with code, encourages the introduction of a Domain Specific Language and a Model-to-Text transformation

MULTI-GOAL CONFLICTS

Manage conflicts in the self-adaptive systems with multiple and concurrent goals

PROTOCOLS SUPPORT

Add further network protocols (eg. GraphQL, gRPC, SOAP, WebSocket, MQTT) using the sink/source and stream paradigm

THANKS !

QUESTIONS ?

Source available on:

https://github.com/elbowz/PyMAPE

All contributions (issues, forks, pull requests) are welcome