Situational Awareness for Industrial Operations

Peter Baumgartner and Patrik Haslum

DATA 61 CSIRO

Australian National University

Research School of Computer Science

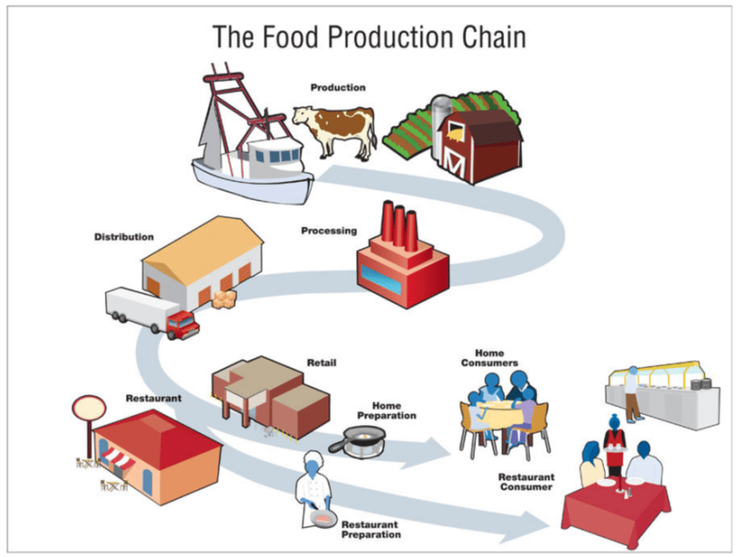
Problem Context

Factory Floor

- Are the operations carried out according to the schedule?

Food Supply Chain

- Are the tomatos delivered within 3 hours and stored below 25℃?
- Is “sold milk quantity” ≤ “produced milk quantity”?



Data Cleansing

- Does the database have complete, correct, accurate and relevant data?

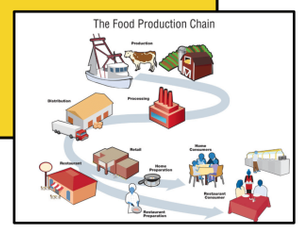
This Work

A situational awareness approach to answer such questions

Situational Awareness

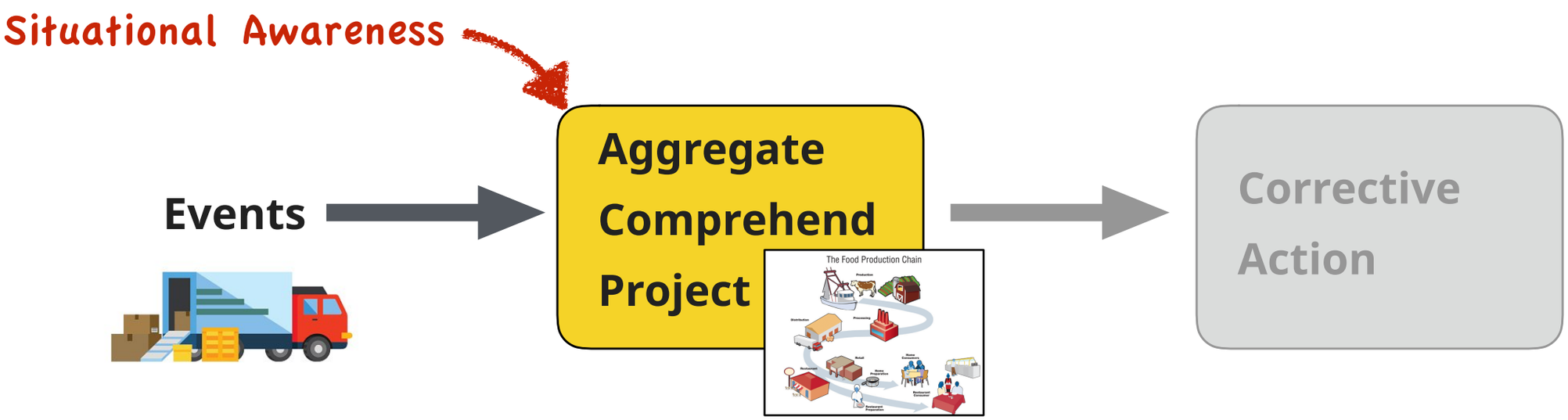
Events

Aggregate
Comprehend
Project



Corrective
Action

Situational Awareness



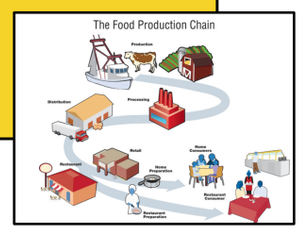
Situational Awareness

Situational Awareness

E.g. Given a truck's GPS trace,
why has the truck stopped?
Weighbridge? Loading? Delay? Break?

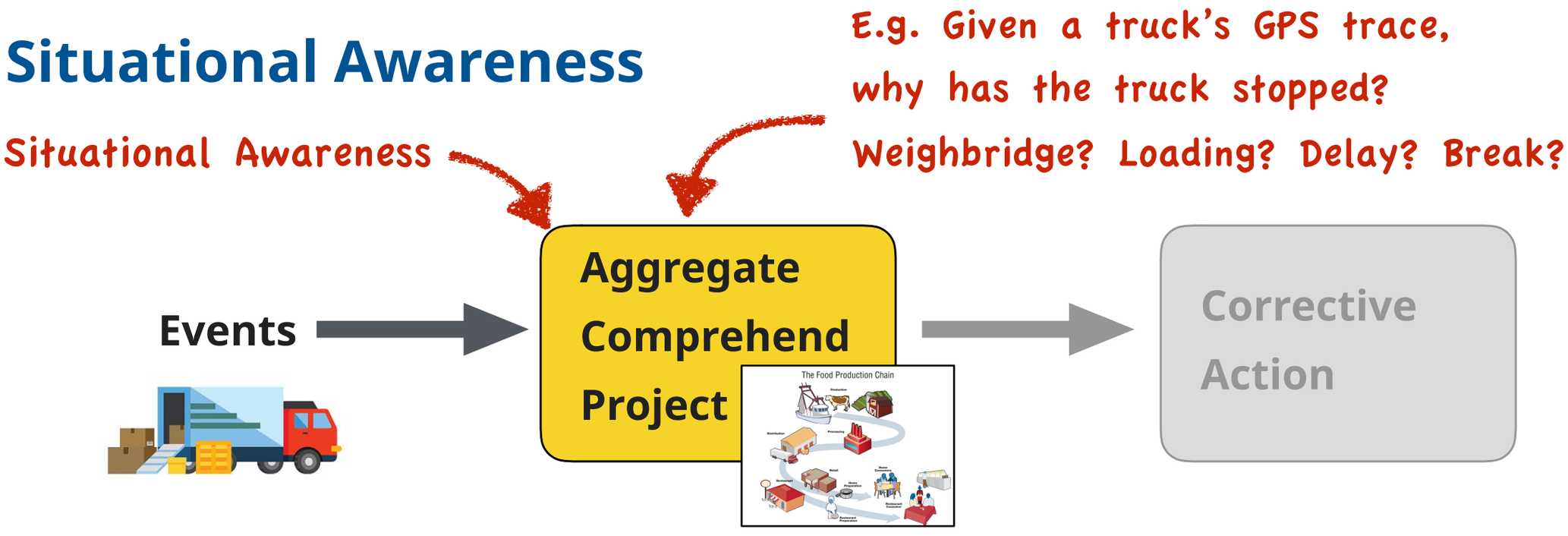
Events

Aggregate
Comprehend
Project



Corrective
Action

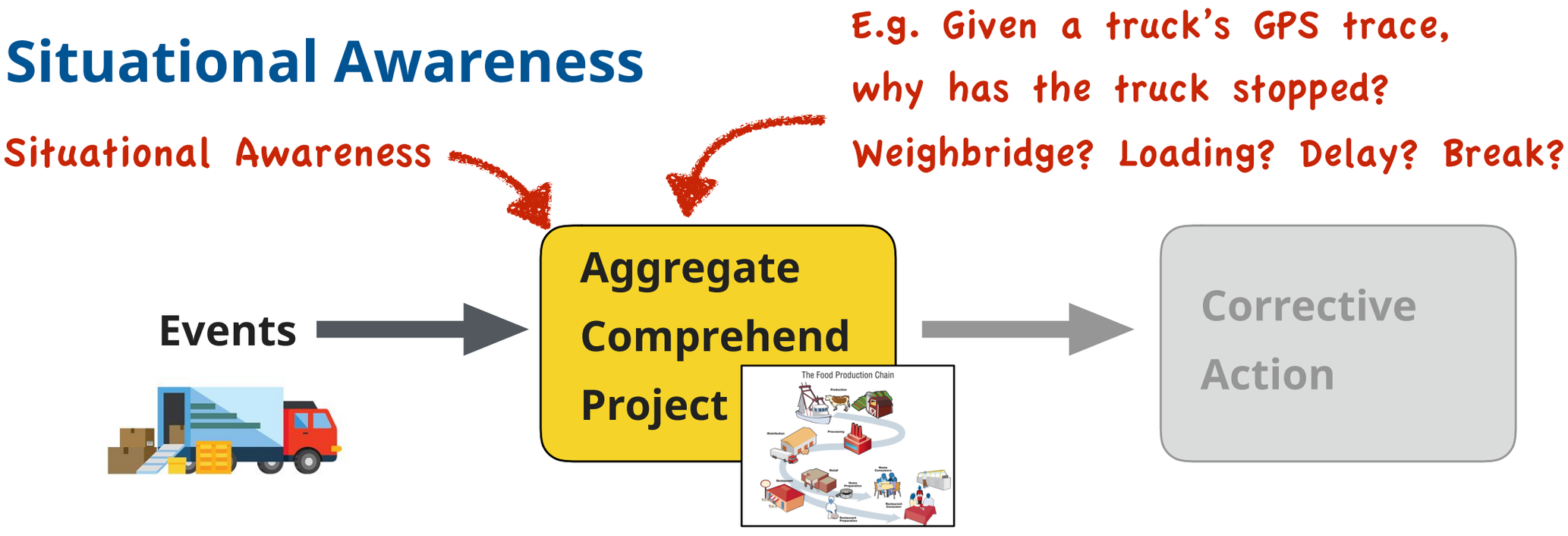
Situational Awareness



Situational Awareness System Design Issues

- Want re-usability across domains
- What is an appropriate general system modelling language?
- How to derive situational awareness from events + model?
- How to deal with incomplete/noisy/erroneous/absent events

Situational Awareness



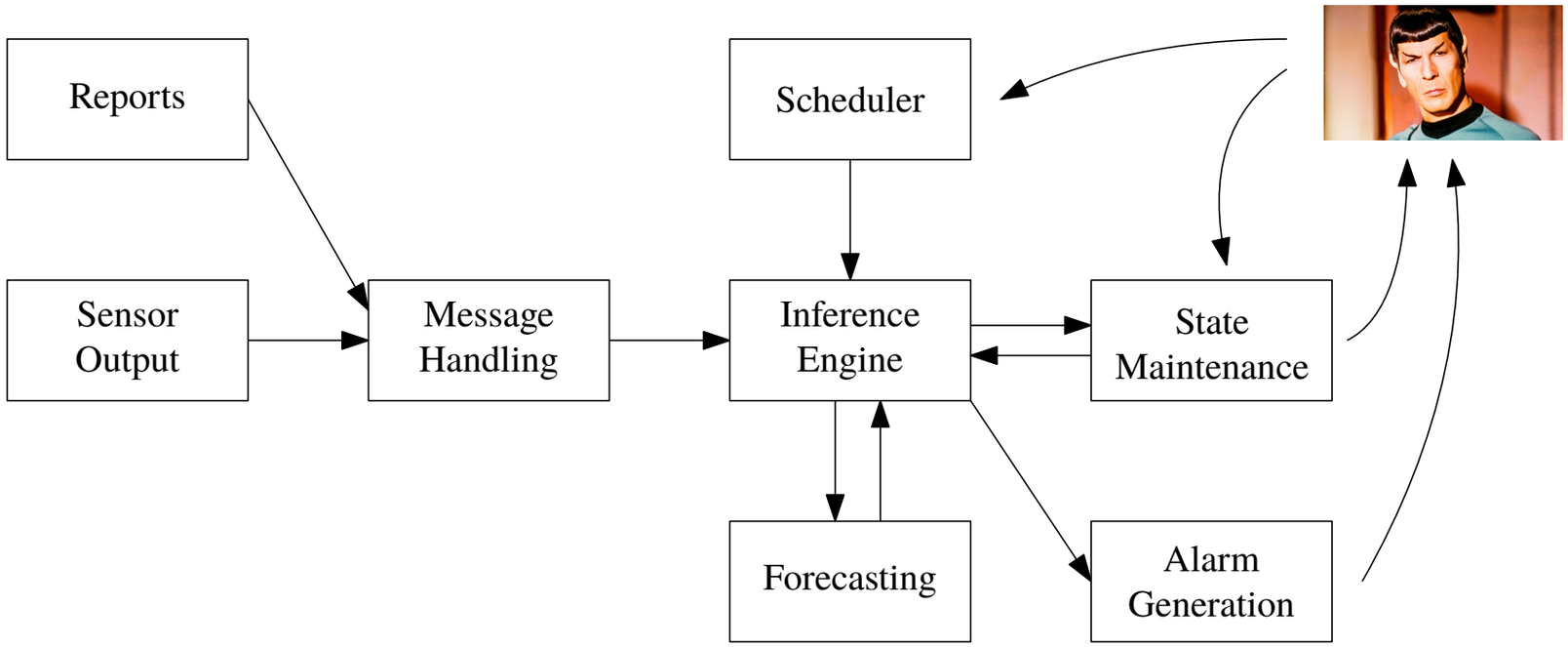
Situational Awareness System Design Issues

- Want re-usability across domains
- What is an appropriate general system modelling language?
- How to derive situational awareness from events + model?
- How to deal with incomplete/noisy/erroneous/absent events

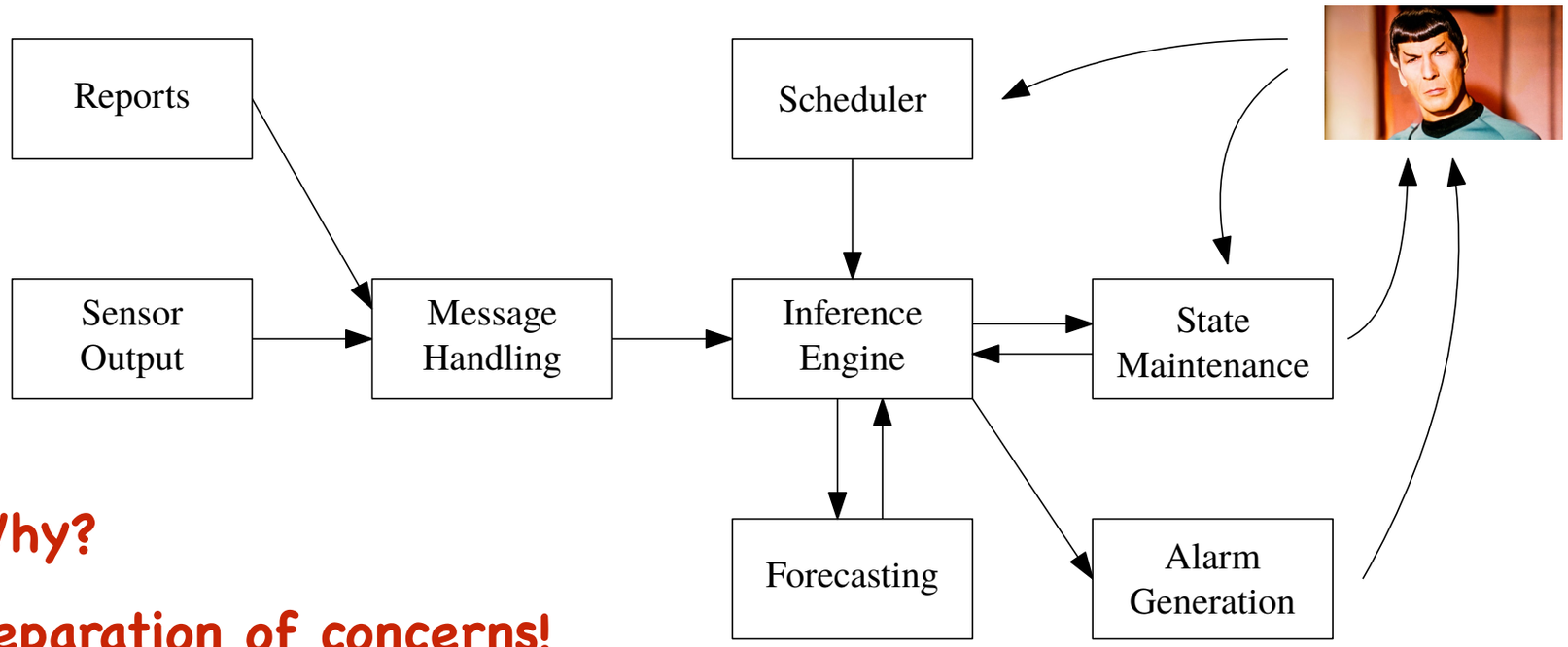
This Talk

- **System Architecture**
- **Modelling Language**
- **Inference Engine and State Maintenance**
- **Implementation on top of Scala**

System Architecture



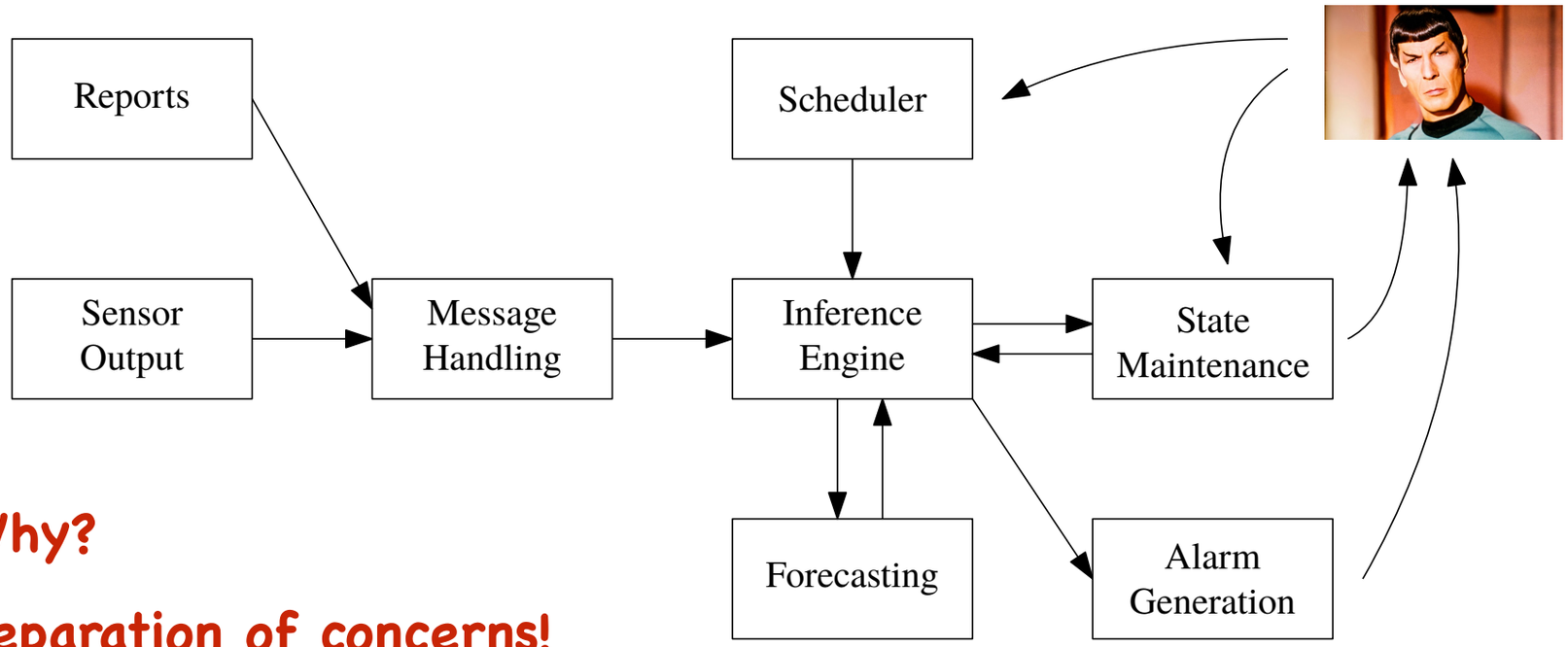
System Architecture



Why?

Separation of concerns!

System Architecture

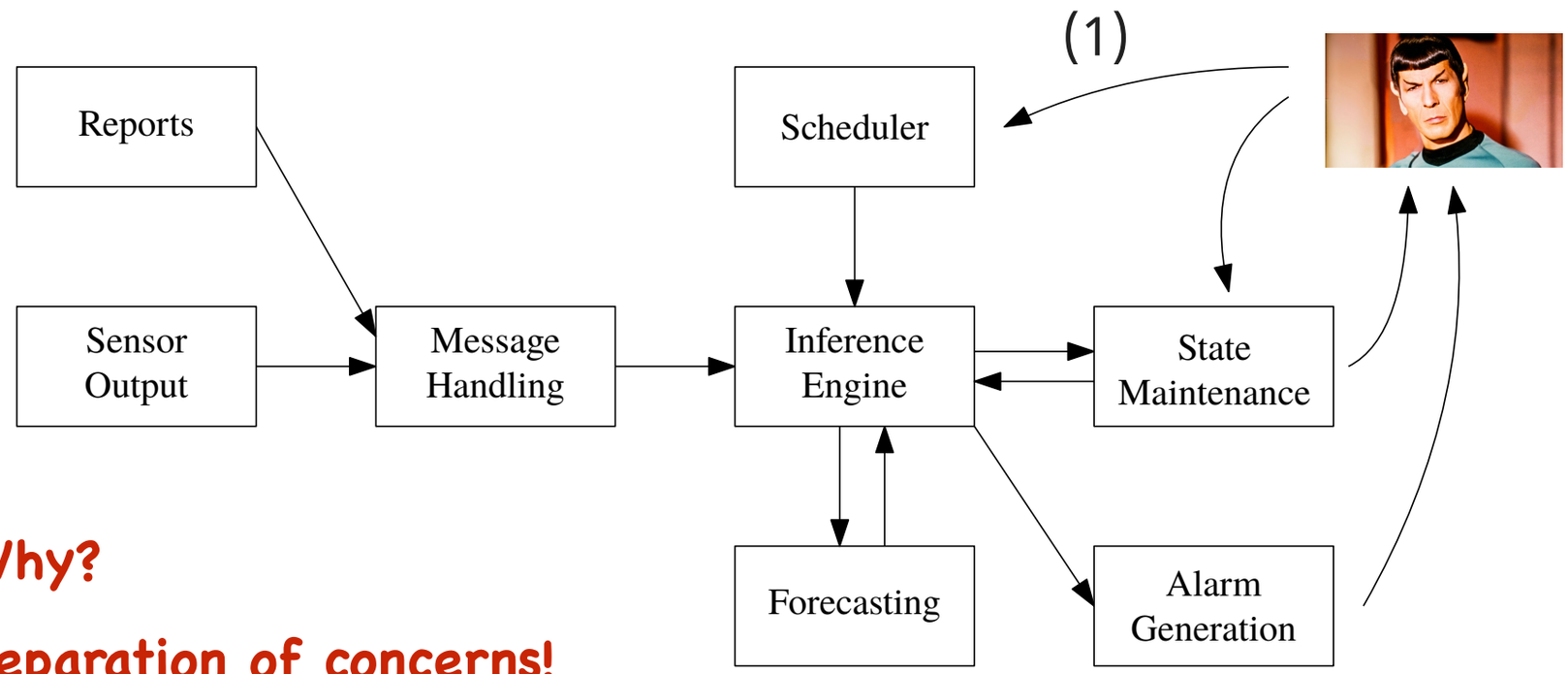


Why?

Separation of concerns!

Execution

System Architecture



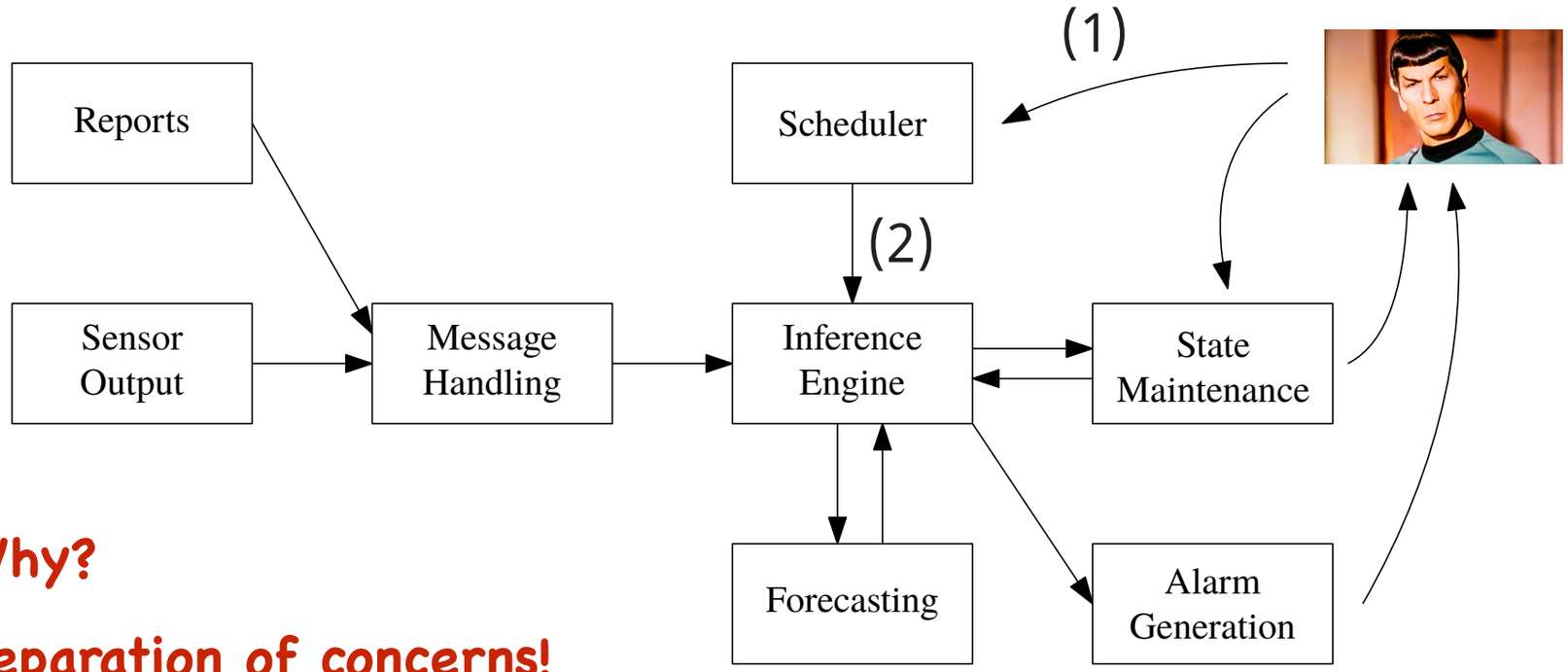
Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

System Architecture



Why?

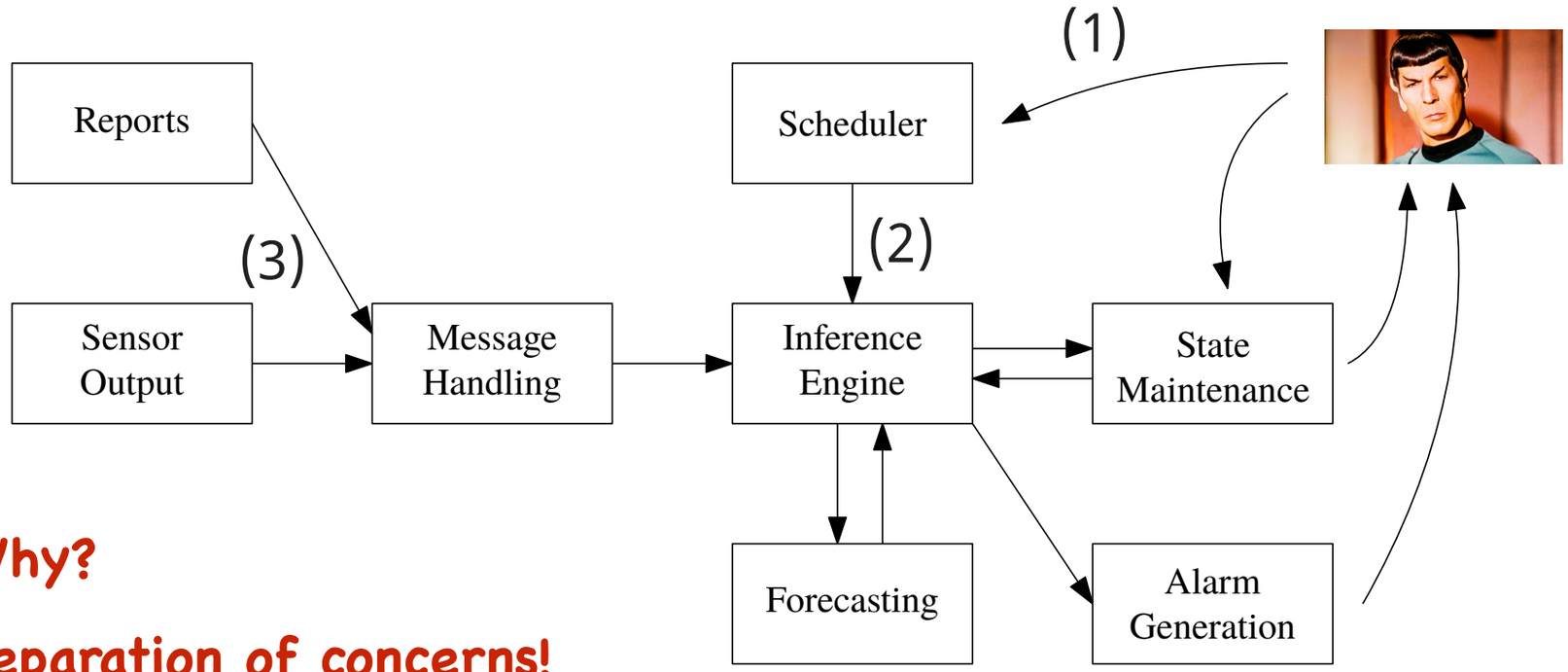
Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

System Architecture



Why?

Separation of concerns!

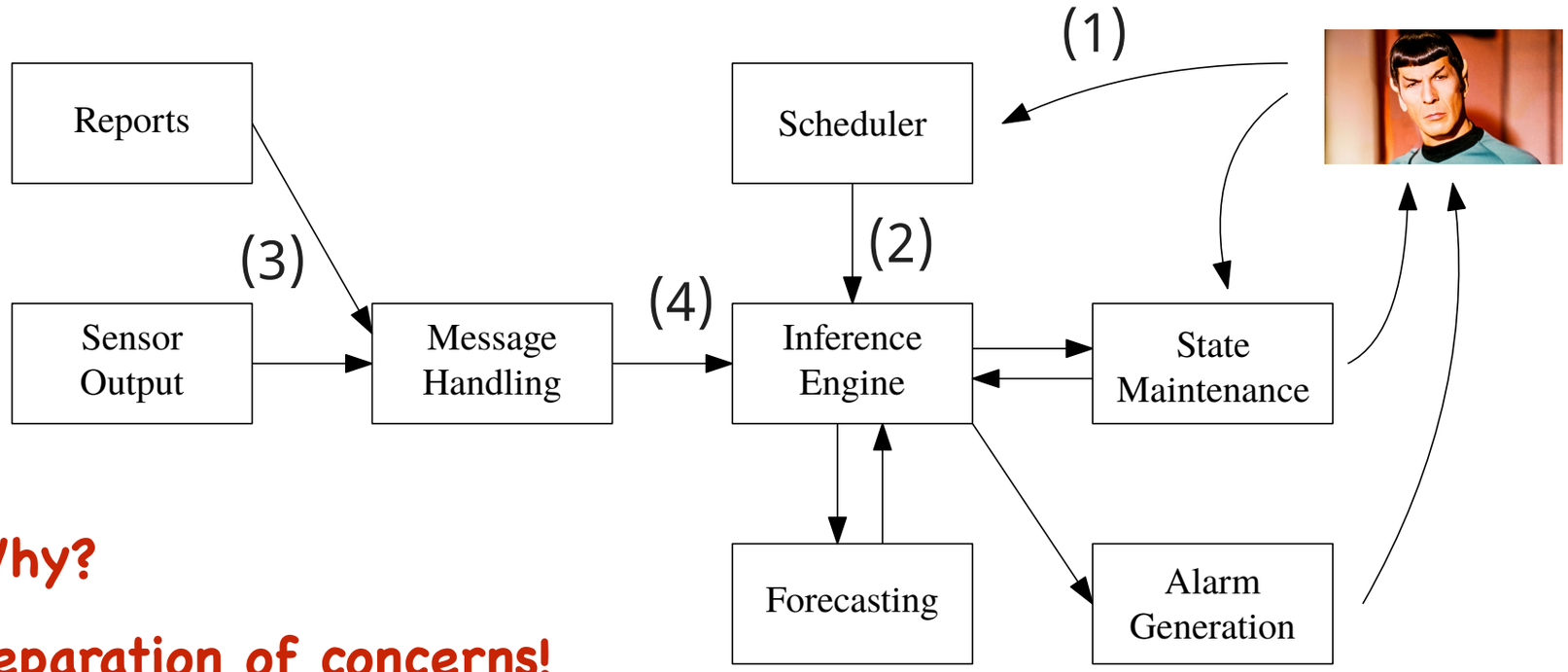
Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

System Architecture



Why?

Separation of concerns!

Execution

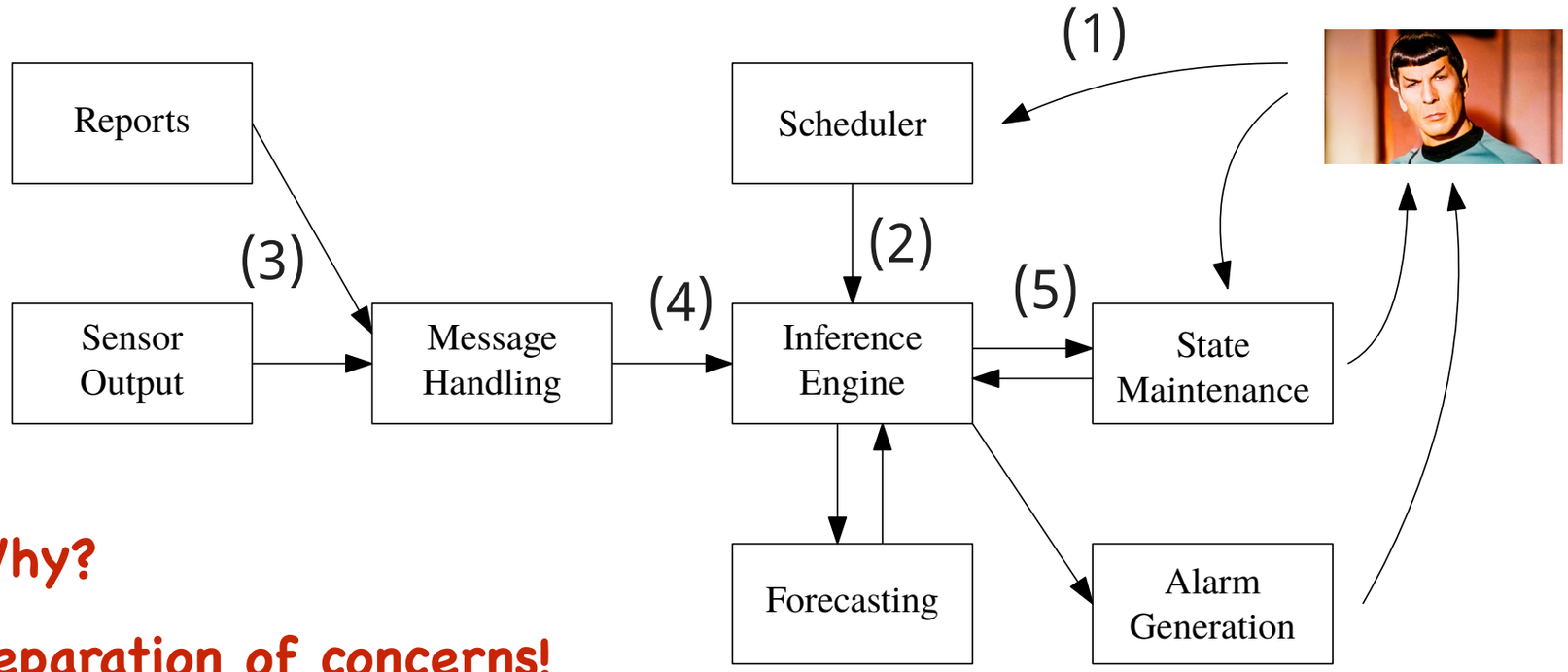
(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

(4) Raw messages preprocessed and fed into Inference Engine

System Architecture



Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

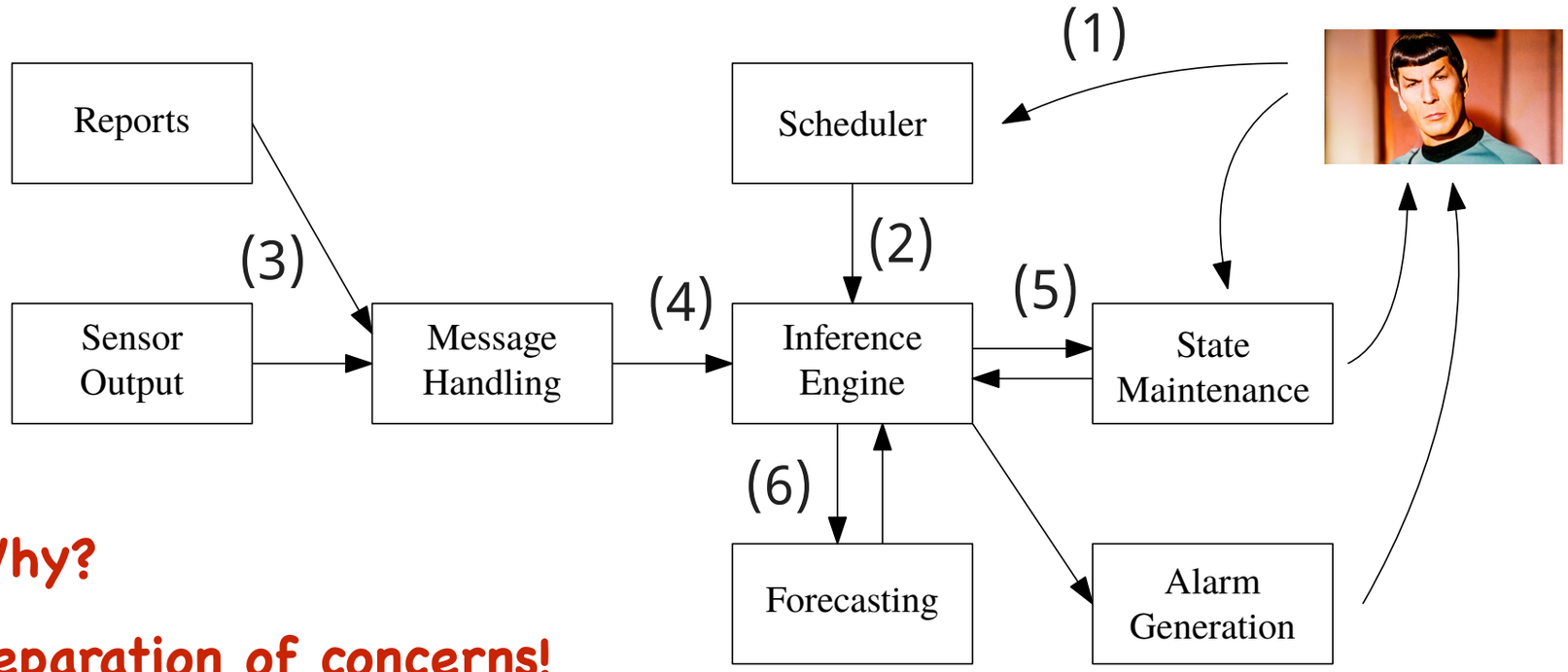
(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

(4) Raw messages preprocessed and fed into Inference Engine

(5) Inference Engine and State Maintenance aggregate recent messages for comprehending *current state*

System Architecture



Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

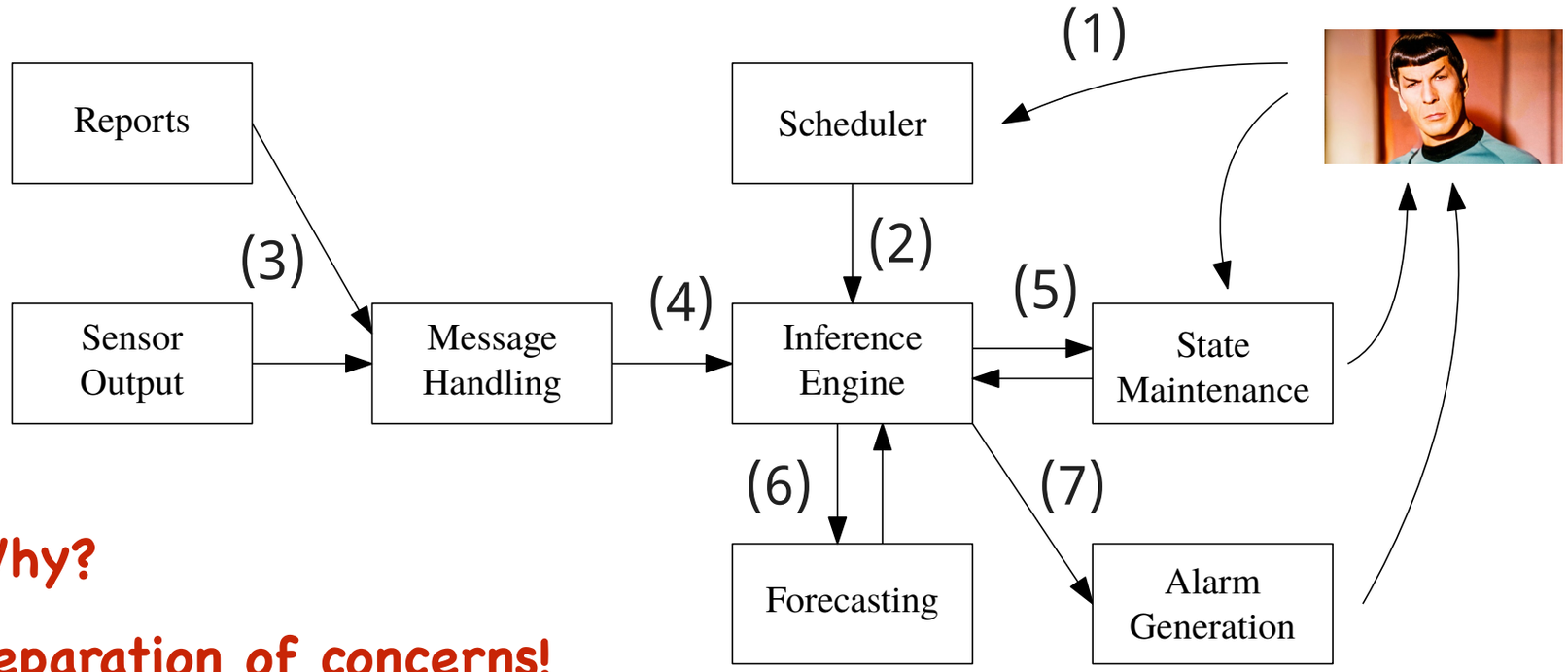
(3) Collect messages from external world e.g. event reports and sensor output

(4) Raw messages preprocessed and fed into Inference Engine

(5) Inference Engine and State Maintenance aggregate recent messages for comprehending *current state*

(6) Forecasting projects into near future

System Architecture



Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

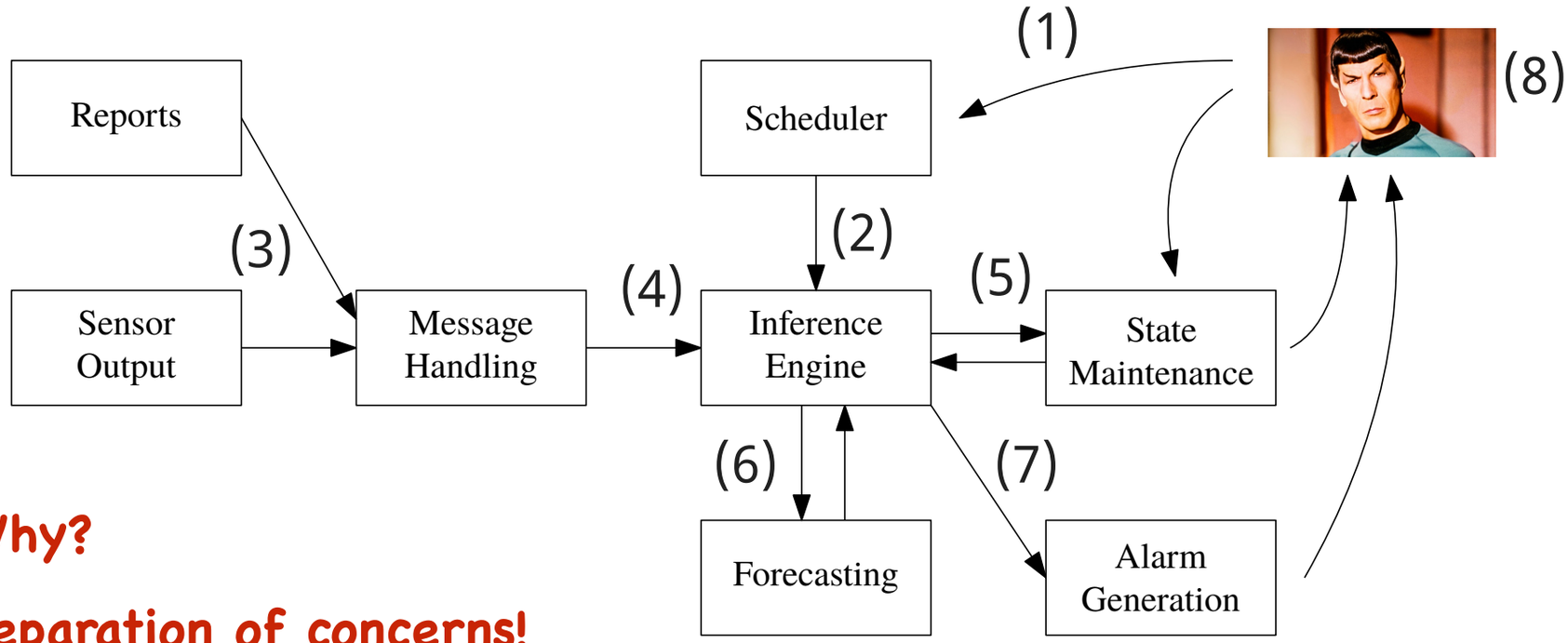
(4) Raw messages preprocessed and fed into Inference Engine

(5) Inference Engine and State Maintenance aggregate recent messages for comprehending *current state*

(6) Forecasting projects into near future

(7) Inference engine sends *expected state* and *current state* to Alarm generation which asses Alarm-worthiness

System Architecture



Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

(4) Raw messages preprocessed and fed into Inference Engine

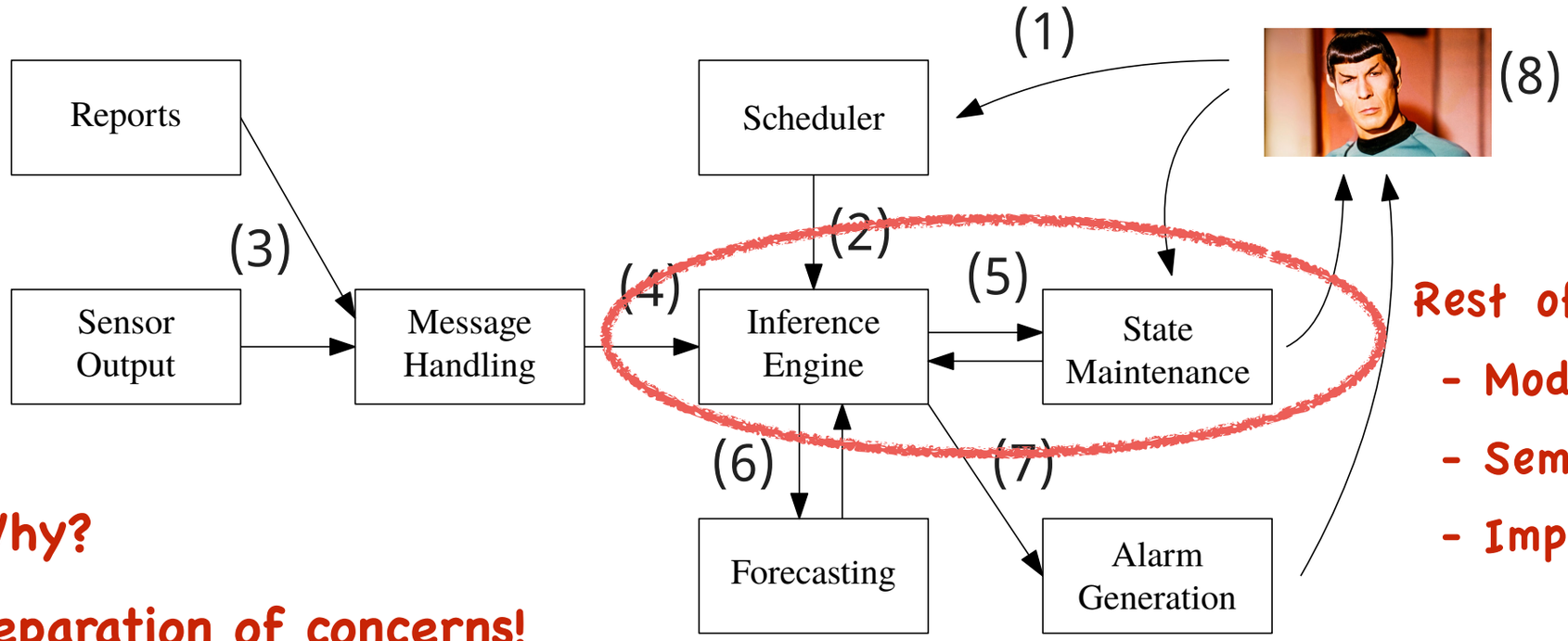
(5) Inference Engine and State Maintenance aggregate recent messages for comprehending *current state*

(6) Forecasting projects into near future

(7) Inference engine sends *expected state* and *current state* to Alarm generation which asses Alarm-worthiness

(8) Human Operator may reconcile/correct *current state*

System Architecture



Rest of this talk:

- Modelling Language
- Semantics
- Implementation

Why?

Separation of concerns!

Execution

(1) Human Operator invokes Scheduler

(2) Schedule given to Inference Engine defining *expected* course of events

(3) Collect messages from external world e.g. event reports and sensor output

(4) Raw messages preprocessed and fed into Inference Engine

(5) Inference Engine and State Maintenance aggregate recent messages for comprehending *current state*

(6) Forecasting projects into near future

(7) Inference engine sends *expected state* and *current state* to Alarm generation which asses Alarm-worthiness

(8) Human Operator may reconcile/correct *current state*

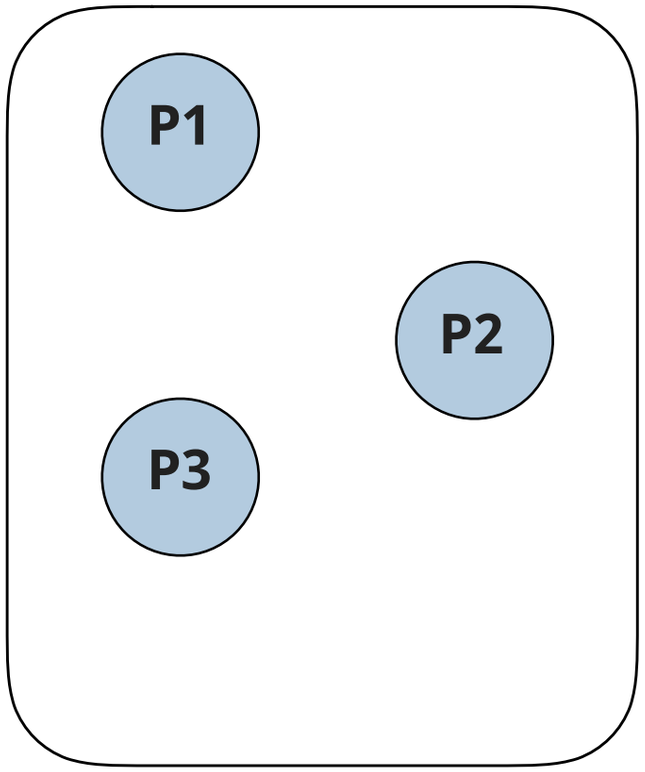
System Model

Model = **processes** + **channels + state maintenance**

Processes

- Codify logic of an actor of the model
- Process state = set of pairs (variable, value)
- System state = set of its process states
- Run in parallel and act on messages

Truck and its load



System Model

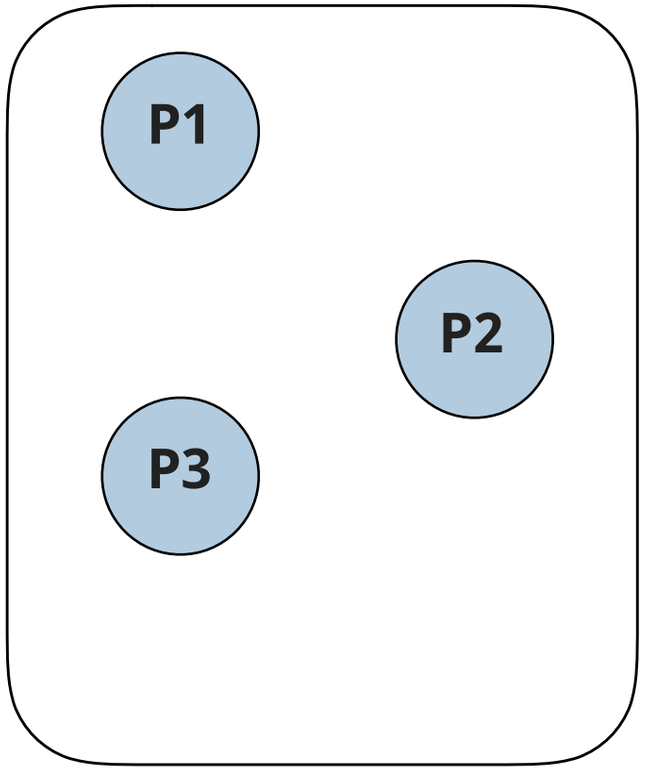
Model = **processes** + **channels + state maintenance**

Processes

- Codify logic of an actor of the model
- Process state = set of pairs (variable, value)
- System state = set of its process states
- Run in parallel and act on messages

Truck and its load

Why? Supports dynamically adding/removing actors



System Model

Model = **processes** + **channels + state maintenance**

Processes

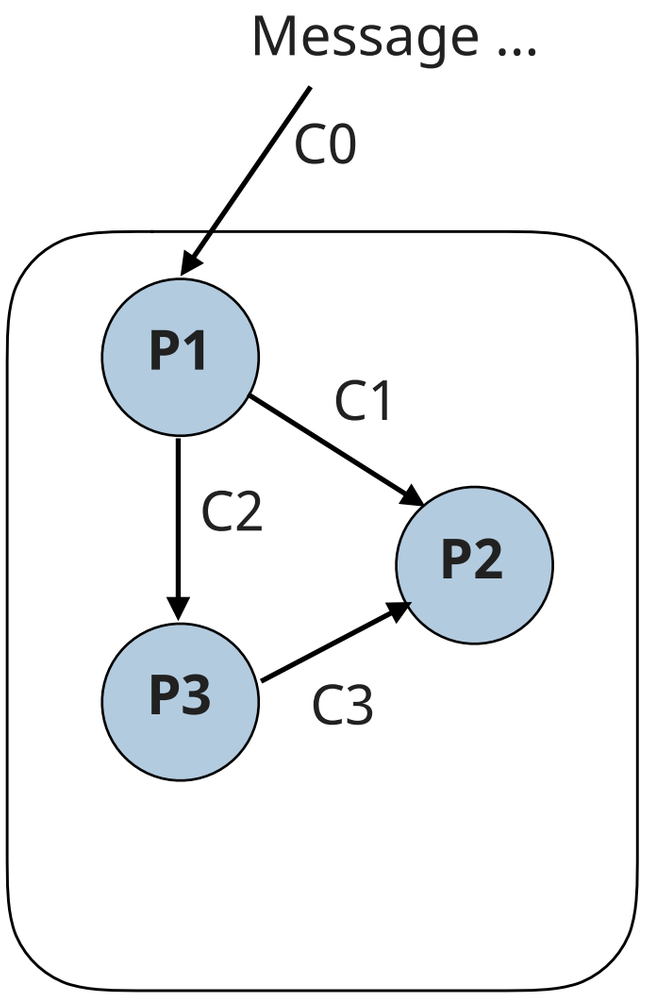
- Codify logic of an actor of the model
- Process state = set of pairs (variable, value)
- System state = set of its process states
- Run in parallel and act on messages

Truck and its load

Why? Supports dynamically adding/removing actors

Channels and Messages

- Internal: m-to-n inter-process comms
- External: channels for input/output
- Interface to state maintenance module



System Model

Model = **processes** + **channels + state maintenance**

Processes

- Codify logic of an actor of the model
- Process state = set of pairs (variable, value)
- System state = set of its process states
- Run in parallel and act on messages

Truck and its load

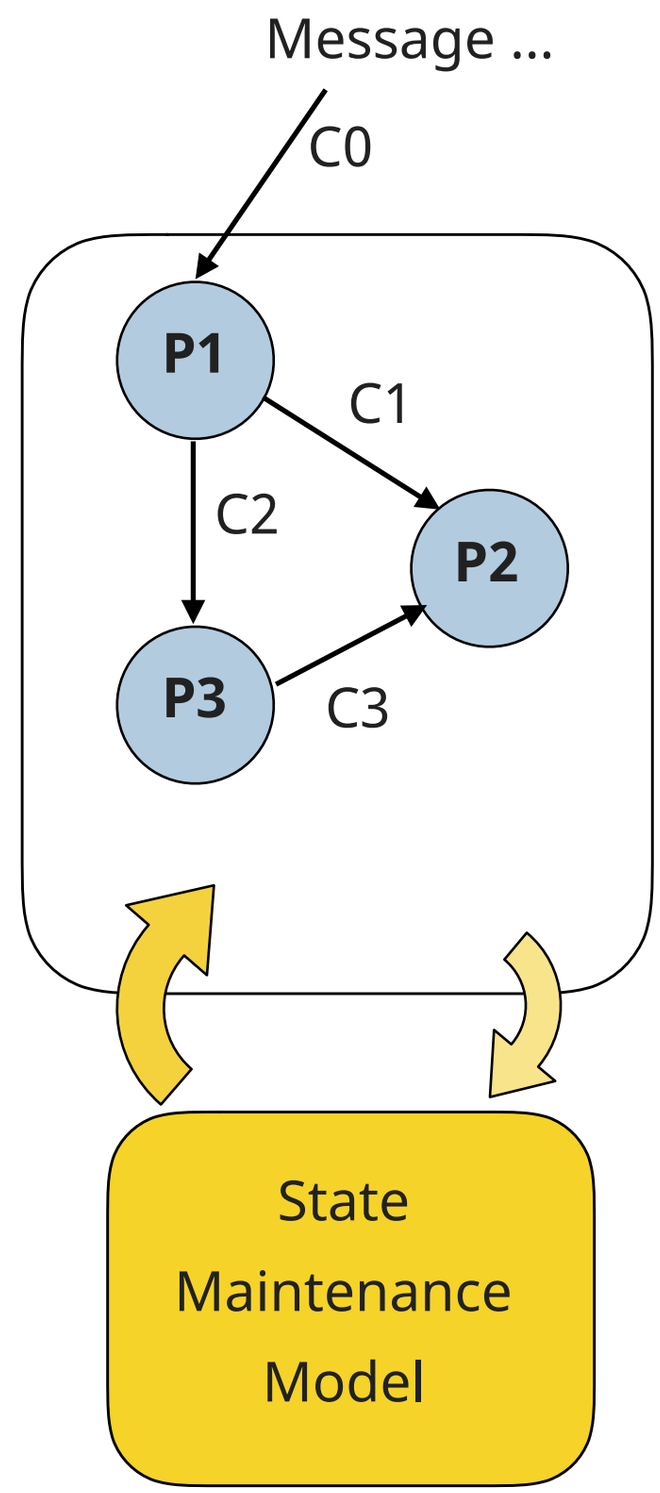
Why? Supports dynamically adding/removing actors

Channels and Messages

- Internal: m-to-n inter-process comms
- External: channels for input/output
- Interface to state maintenance module

State Maintenance Model

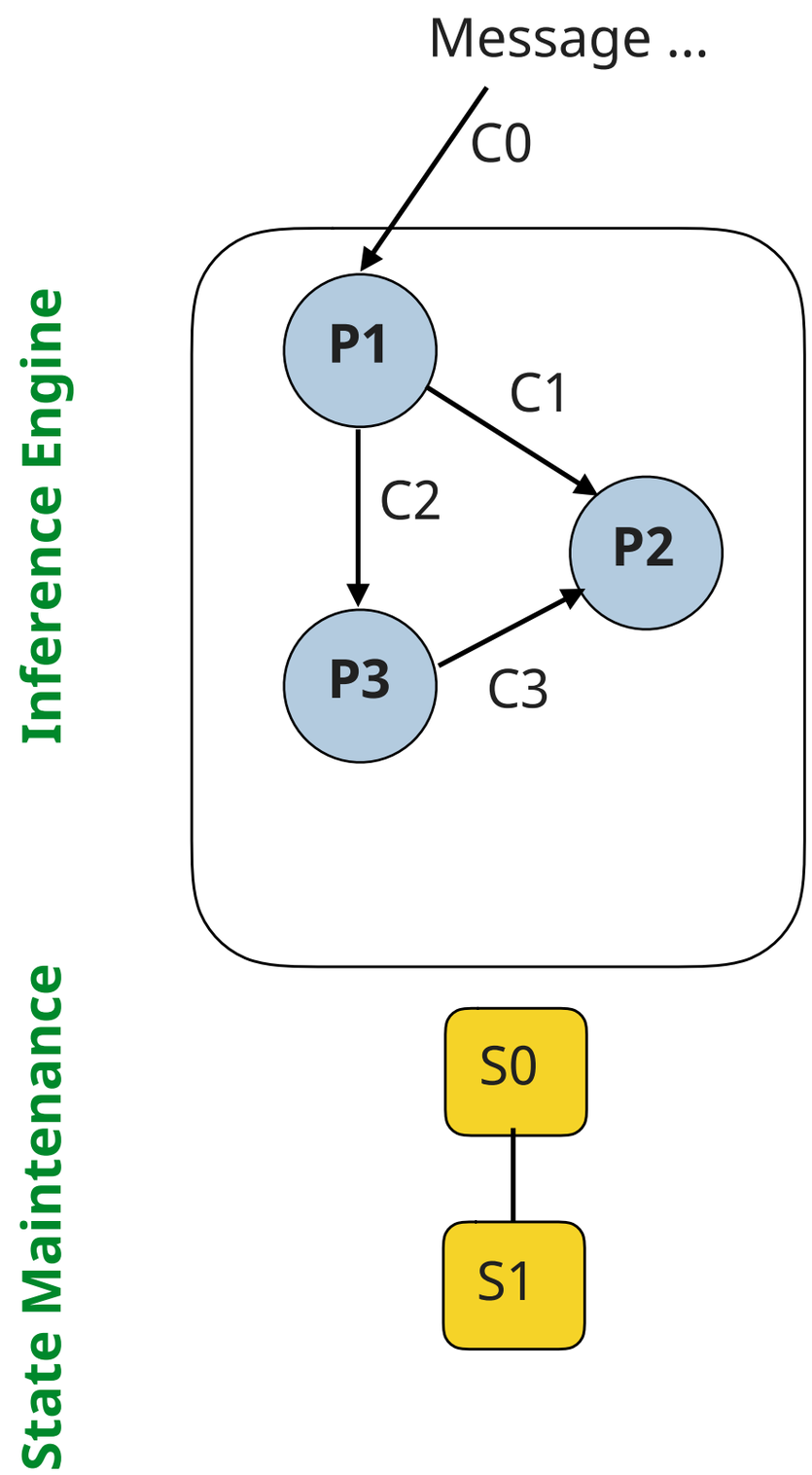
- Analysis and amends system states
- "Global view" - has system state history, e.g. RTV based on (probabilistic) temporal logic, Conflict-directed diagnosis



System Model Execution

The main loop

Inference Engine

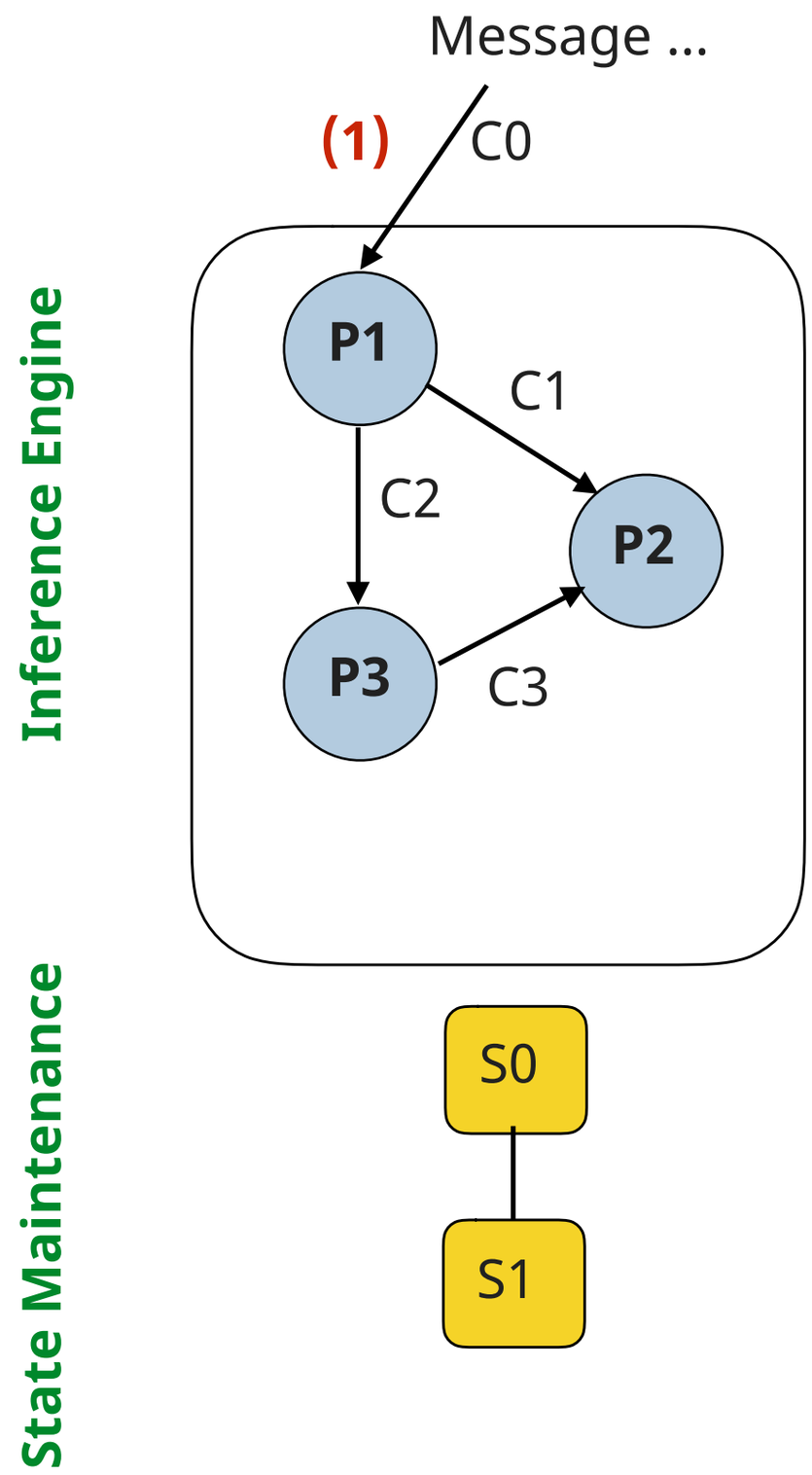


System Model Execution

The main loop

Inference Engine

(1) Next external message comes in



System Model Execution

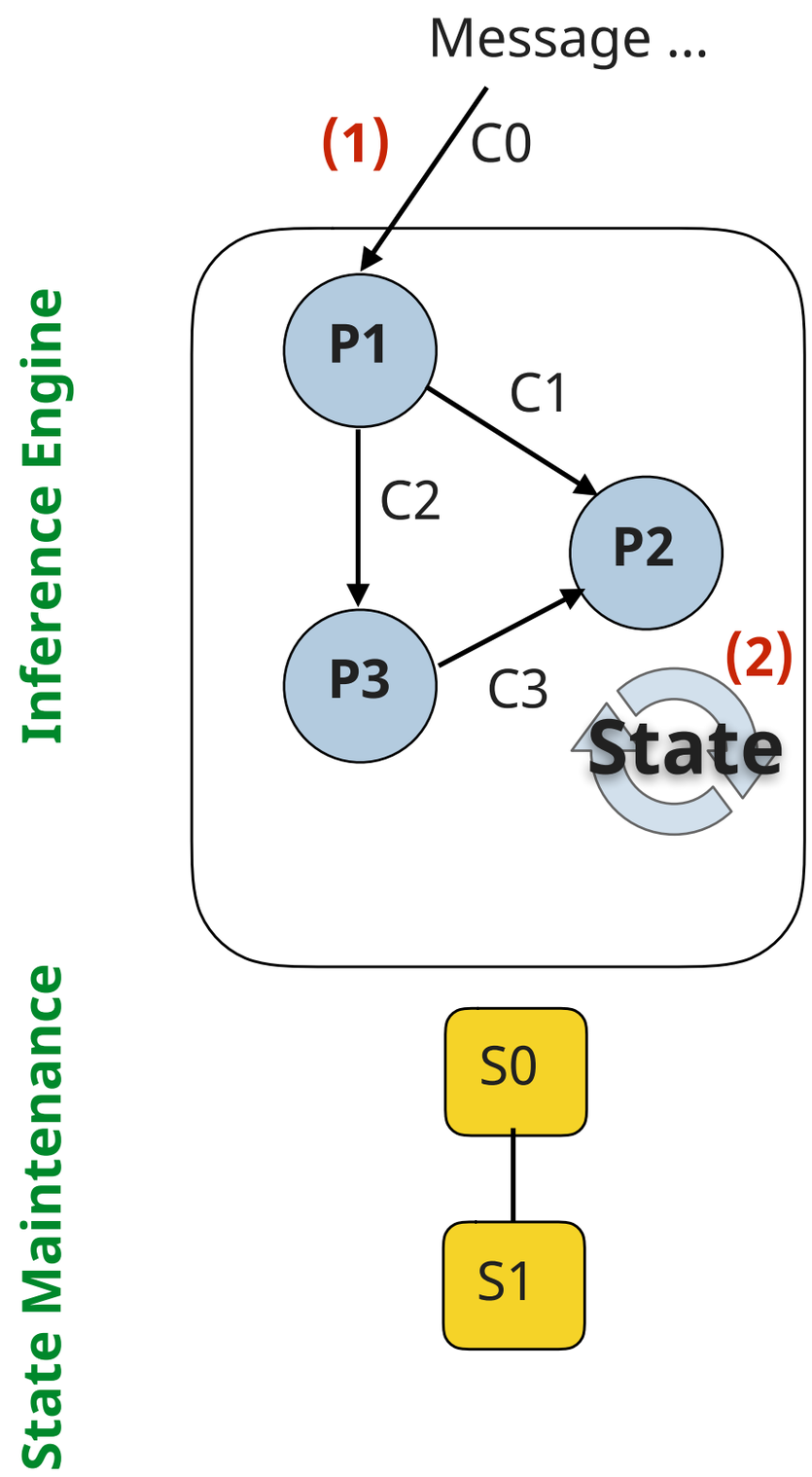
The main loop

Inference Engine

(1) Next external message comes in

(2) Processes run until exhaustion

(No more messages sent around)



System Model Execution

The main loop

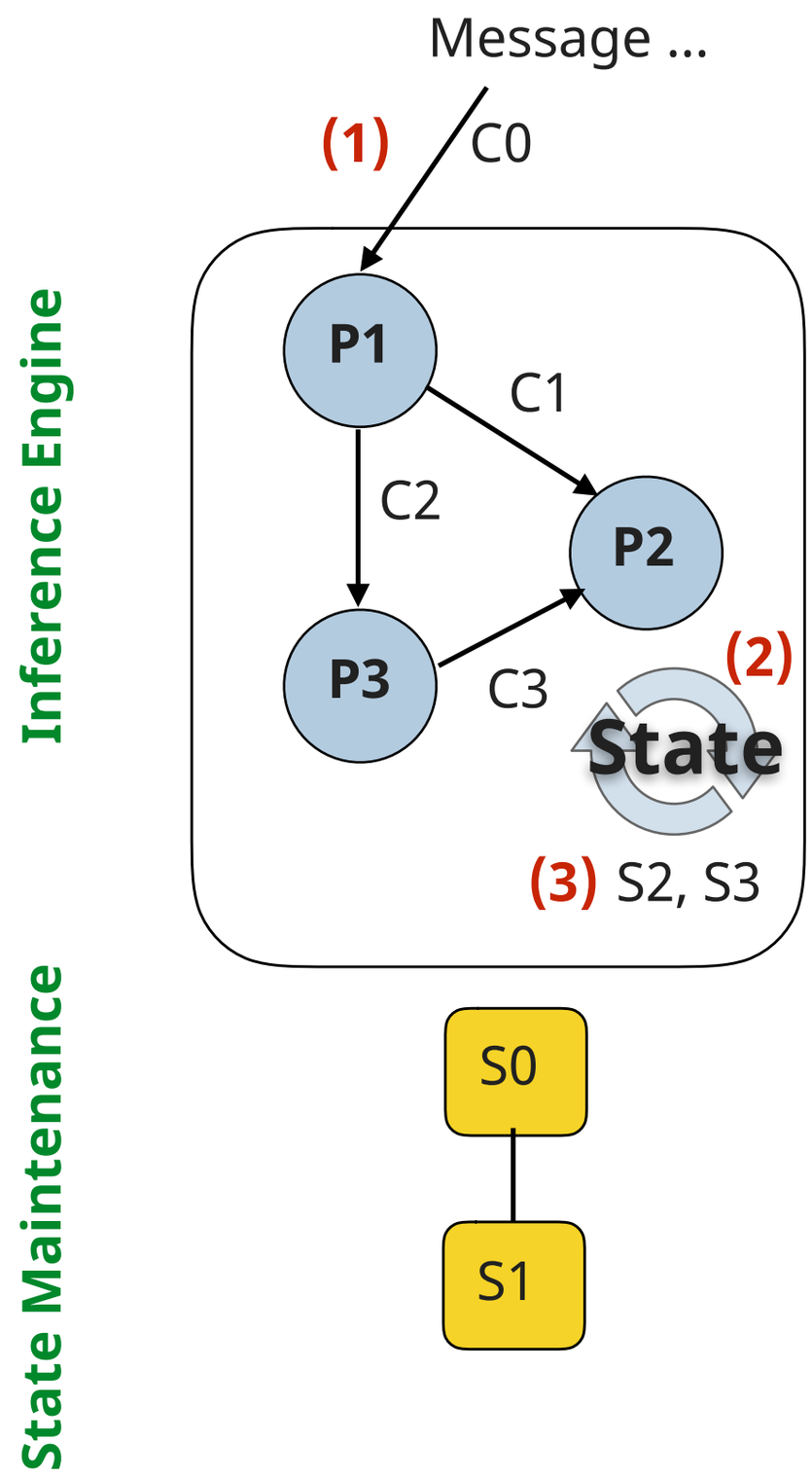
Inference Engine

(1) Next external message comes in

(2) Processes run until exhaustion

(No more messages sent around)

(3) Results in derived possible system states (**!**)



System Model Execution

The main loop

Inference Engine

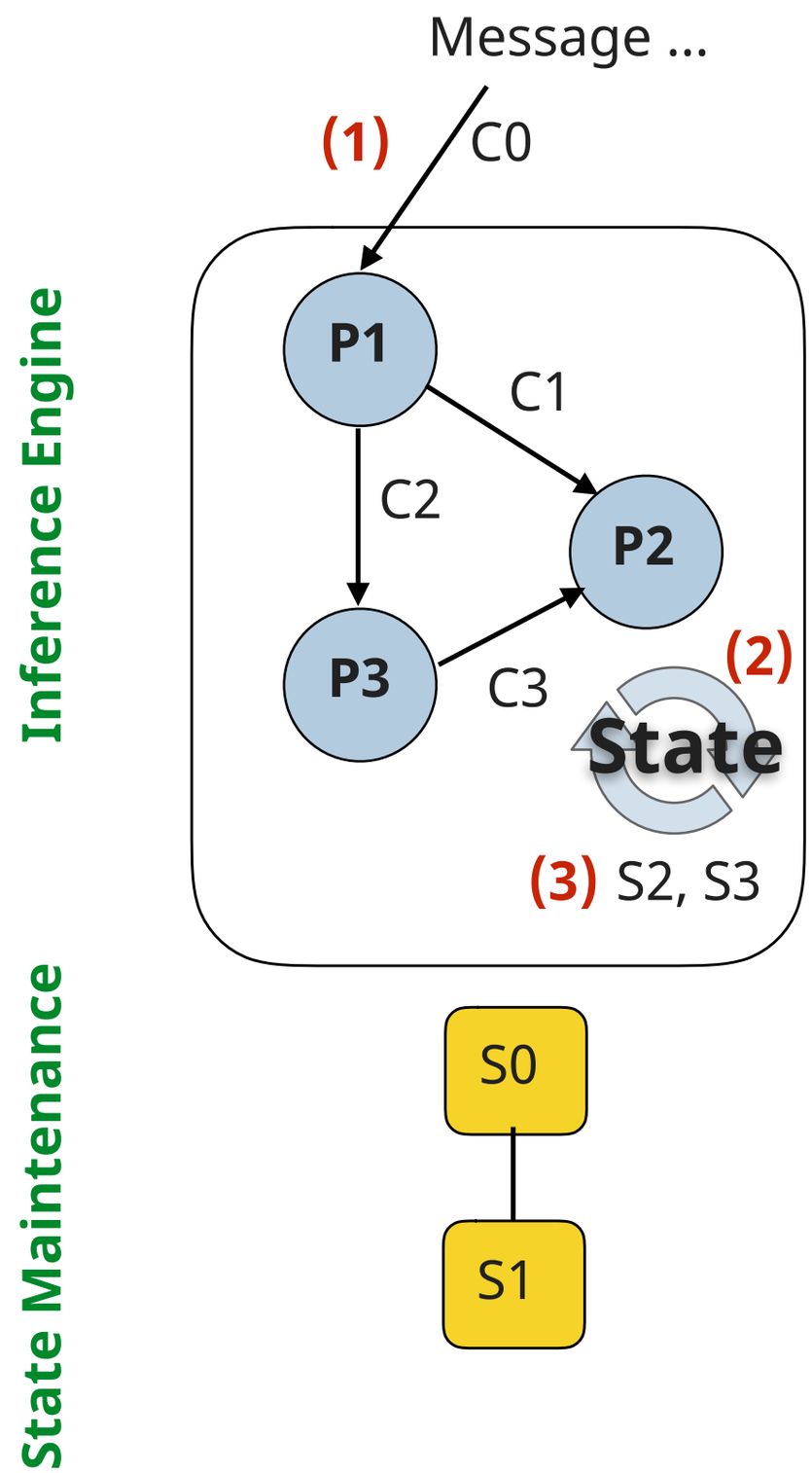
(1) Next external message comes in

(2) Processes run until exhaustion

(No more messages sent around)

(3) Results in derived possible system states (**!**)

State Maintenance



System Model Execution

The main loop

Inference Engine

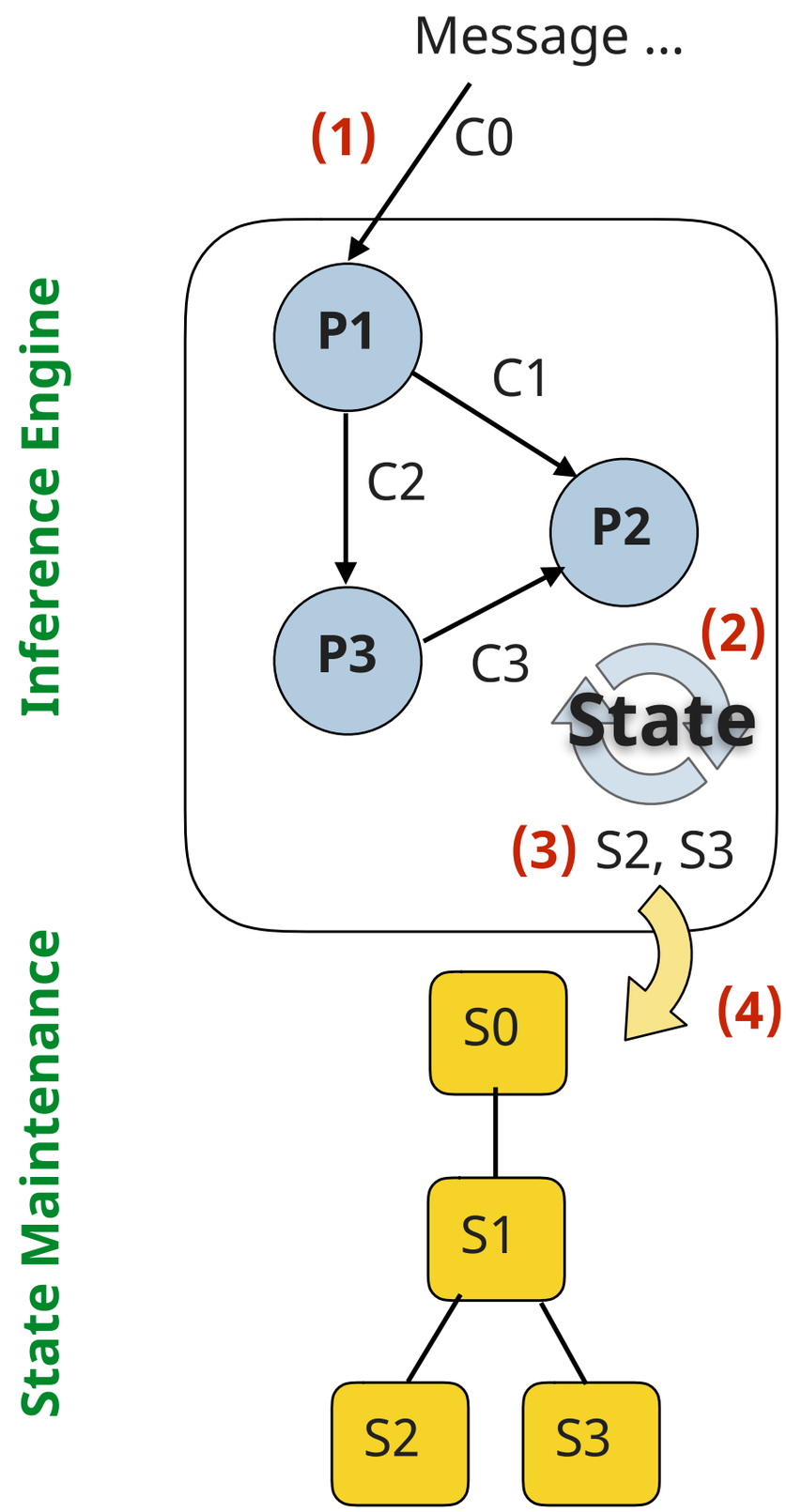
(1) Next external message comes in

(2) Processes run until exhaustion
(No more messages sent around)

(3) Results in derived possible system states (**!**)

State Maintenance

(4) Derived possible system states are sent to state maintenance



System Model Execution

The main loop

Inference Engine

(1) Next external message comes in

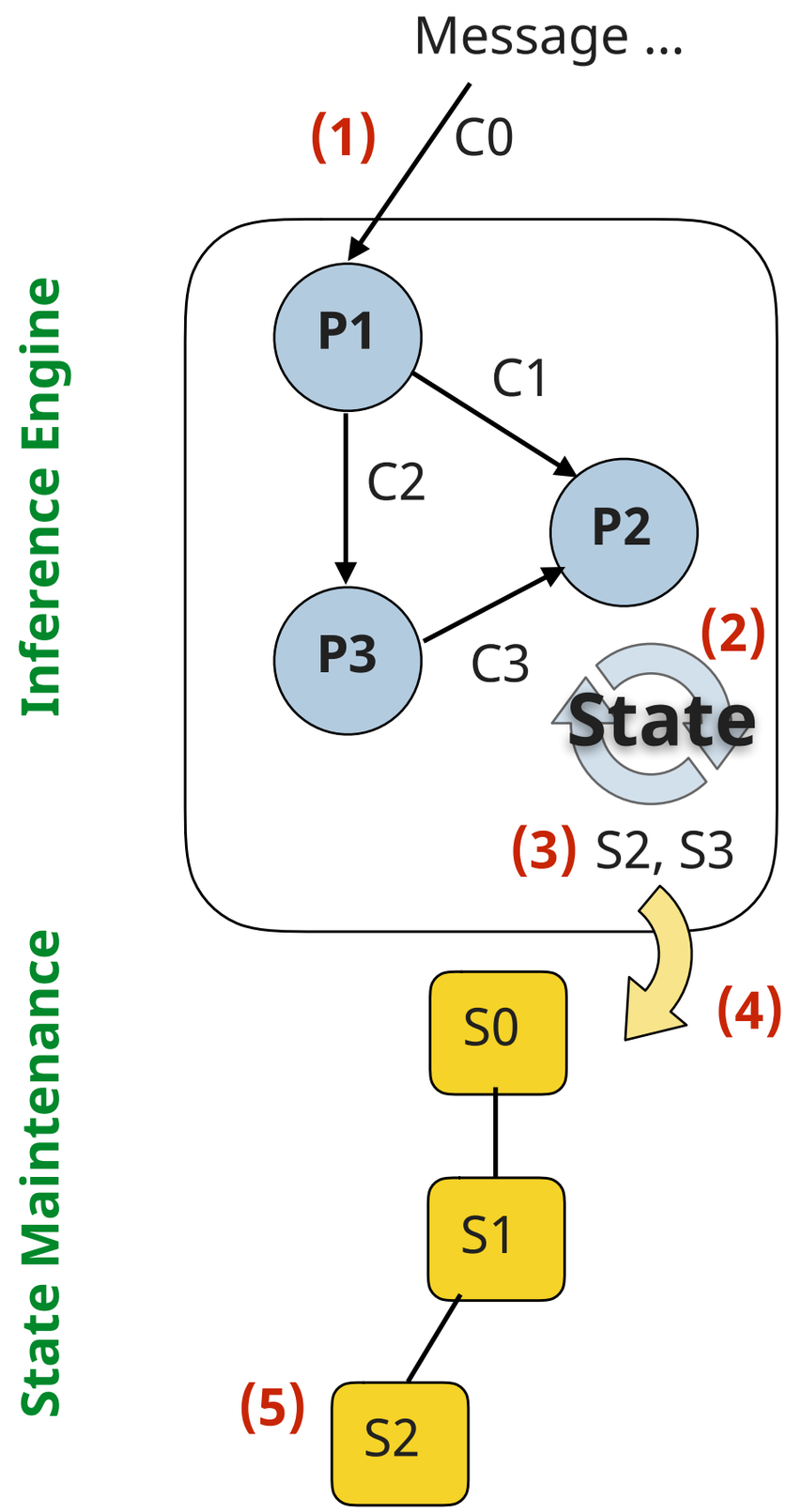
(2) Processes run until exhaustion
(No more messages sent around)

(3) Results in derived possible system states (**!**)

State Maintenance

(4) Derived possible system states are sent to state maintenance

(5) State maintenance computes a next current state from derived possible states and state history



System Model Execution

The main loop

Inference Engine

(1) Next external message comes in

(2) Processes run until exhaustion
(No more messages sent around)

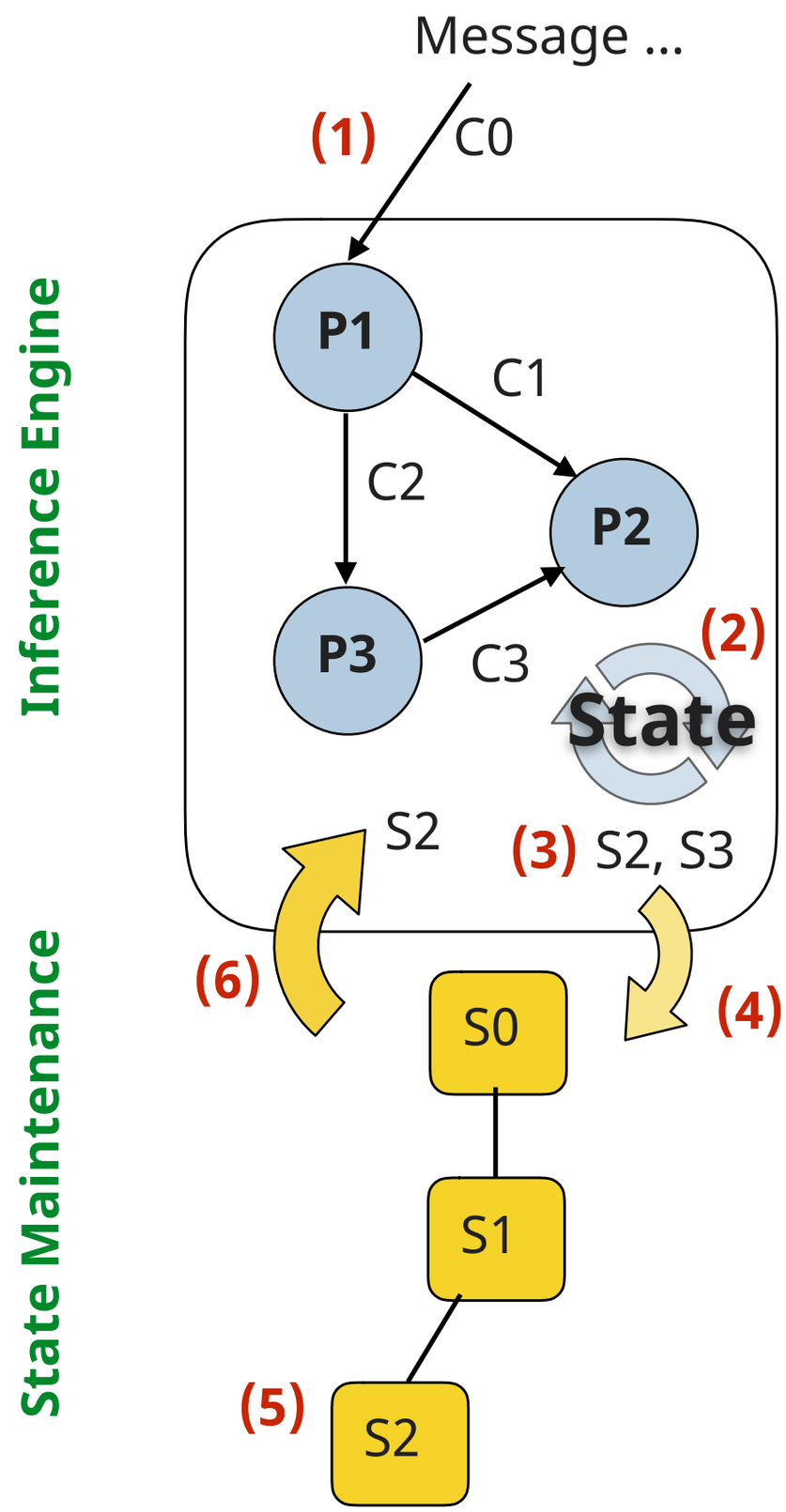
(3) Results in derived possible system states (**!**)

State Maintenance

(4) Derived possible system states are sent to state maintenance

(5) State maintenance computes a next current state from derived possible states and state history

(6) Next current state is sent back to inference engine as current system state



System Model Execution

The main loop

Inference Engine

(1) Next external message comes in

(2) Processes run until exhaustion
(No more messages sent around)

(3) Results in derived possible system states (**!**)

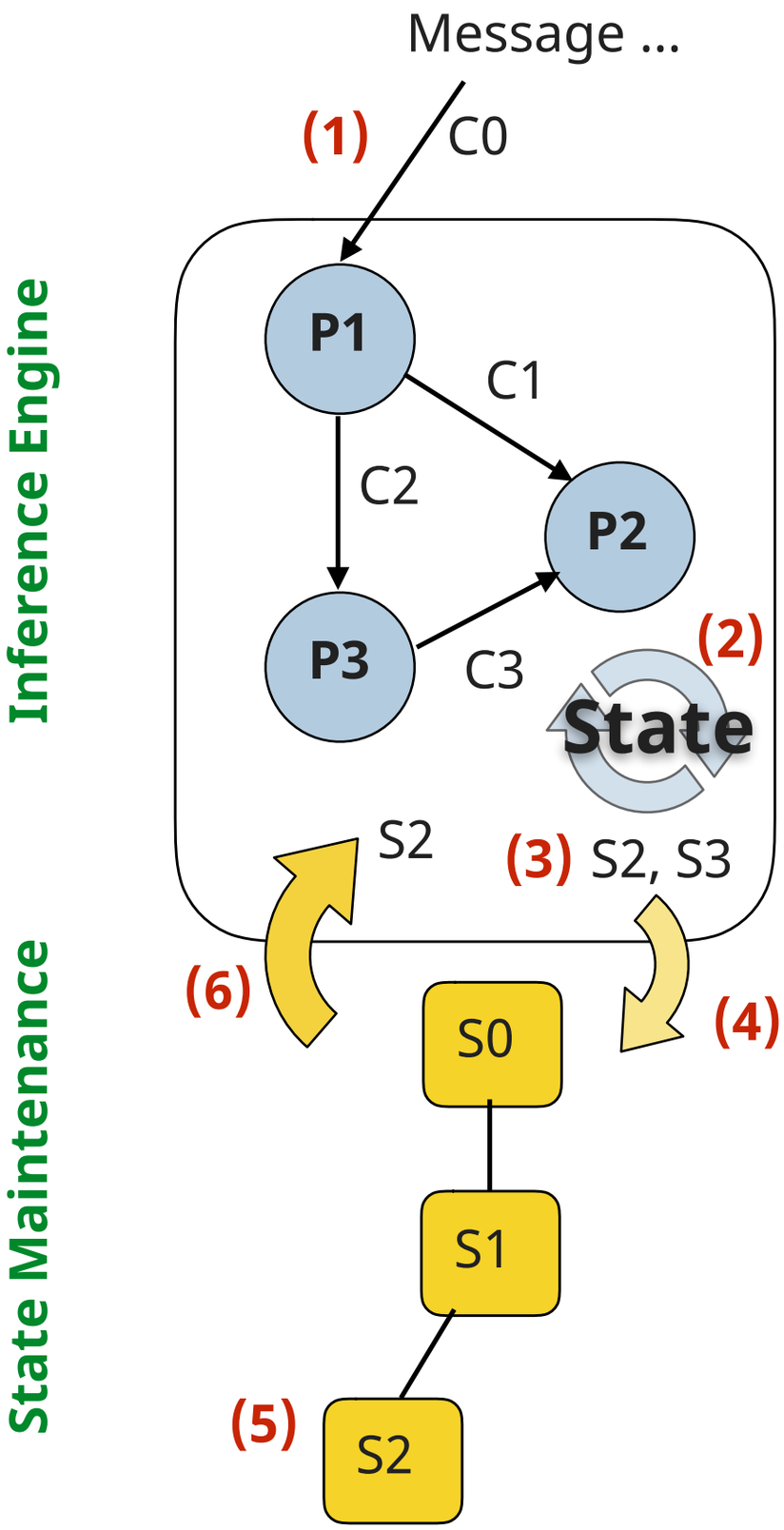
State Maintenance

(4) Derived possible system states are sent to state maintenance

(5) State maintenance computes a next current state from derived possible states and state history

(6) Next current state is sent back to inference engine as current system state

(7) Continue with (1)



Example - Food Supply Chain

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverina and Batlow)
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (GPS)
- *Dockets* for loading goods on trucks at warehouses (EPCIS)

Processes

Messages

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverina and Batlow)
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (GPS)
- *Dockets* for loading goods on trucks at warehouses (EPCIS)

Processes

Messages

Concrete Events

Certain trucks are loaded with certain goods that are moved between certain warehouses

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverina and Batlow)
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (GPS)
- *Dockets* for loading goods on trucks at warehouses (EPCIS)

Processes

Messages

Concrete Events

Certain trucks are loaded with certain goods that are moved between certain warehouses

Situational Awareness Task

- What trucks are where at what time?
- What goods of what origin are where?

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverina and Batlow)
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (GPS)
- *Dockets* for loading goods on trucks at warehouses (EPCIS)

Processes

Messages

Concrete Events

Certain trucks are loaded with certain goods that are moved between certain warehouses

Situational Awareness Task

- What trucks are where at what time?
- What goods of what origin are where?

Complication

- Some waypoints are missing

  Recover from other messages?
- Origin of some goods is missing

  Informed guess?

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverin
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (G
- *Dockets* for loading goods on trucks at warehouses (E

```
...
{ "time": "2018-02-18T02:00",
  "type": "Waypoint",
  "truck": "TruckB",
  "location": “Goulburn" }
...
{ "time": "2018-02-18T06:45",
  "type": "Loading",
  "truck": "TruckB",
  "location": "Canberra",
  "goods": "Oranges",
  "origin": “unknown" }
```

Concrete Events

Certain trucks are loaded with certain goods that are n ouses

Situational Awareness Task

- What trucks are where at what time?
- What goods of what origin are where?

Complication

- Some waypoints are missing

  Recover from other messages?
- Origin of some goods is missing

  Informed guess?

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverin
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (G
- *Dockets* for loading goods on trucks at warehouses (E

Concrete Events

Certain trucks are loaded with certain goods that are n ouses

Situational Awareness Task

- What trucks are where at what time?
- What goods of what origin are where?

Complication

- Some waypoints are missing

  Recover from other messages?
- Origin of some goods is missing

  Informed guess?

```
...
{ "time": "2018-02-18T02:00",
  "type": "Waypoint",
  "truck": "TruckB",
  "location": “Goulburn" }
...
{ "time": "2018-02-18T06:45",
  "type": "Loading",
  "truck": "TruckB",
  "location": "Canberra",
  "goods": "Oranges",
  "origin": “unknown" }
```

Infer Waypoint message
from Loading message
and broadcast it

Example - Food Supply Chain

Model

- *Goods* (apples and oranges) of specific *origin* (Riverin
- *Warehouses* (Sydney, Goulburn, Canberra)
- *Trucks* (TruckA, TruckB and TruckC)
- *Waypoints* for trucks in terms of time and location (G
- *Dockets* for loading goods on trucks at warehouses (E

Concrete Events

Certain trucks are loaded with certain goods that are n ouses

Situational Awareness Task

- What trucks are where at what time?
- What goods of what origin are where?

Complication

- Some waypoints are missing

  Recover from other messages?
- Origin of some goods is missing

  Informed guess?

```
...
{ "time": "2018-02-18T02:00",
  "type": "Waypoint",
  "truck": "TruckB",
  "location": “Goulburn" }
...
{ "time": "2018-02-18T06:45",
  "type": "Loading",
  "truck": "TruckB",
  "location": "Canberra",
  "goods": "Oranges",
  "origin": “unknown" }
```

Infer Waypoint message from Loading message and broadcast it

Guess origin = “Riverina” OR origin = “Batlow”

Implementation - Shallow Embedding in Scala

Scala

- *Scala combines object-oriented and functional programming in one concise, high-level language* (www.scala-lang.org)
- Runs on JVM
- Static type system, type inference, pattern matching, call-by-name/call-by-value, libraries
- Syntactic sugar

  unless (x == 0) { **println**("One over x is " + 1/x) }

Shallow Embedding

- Modelling language =

  Scala + syntactic sugar + class definitions for "Process" and "Channel"
- Scheduler is library function
- Why?
  - Full power of host language
  - Existing libraries for DB connectivity, RabbitMQ, JSon, ...
  - Easy to implement

Food Supply Chain Example

I_am_a_modelling_language_expression

Processes

```
object Dispatch extends Process("Dispatch") { ... }
class Truck(Id: String) extends Process("Truck") { ... }
class Warehouse(Location: String) extends Process("Warehouse") { ... }
```

Message Data Structures

```
abstract class Message
case class Waypoint(time: DateTime, truck: String, location: String) extends Message
case class Loading(time: DateTime, truck: String, location: String,
                   goods: String, origin: String) extends Message
```

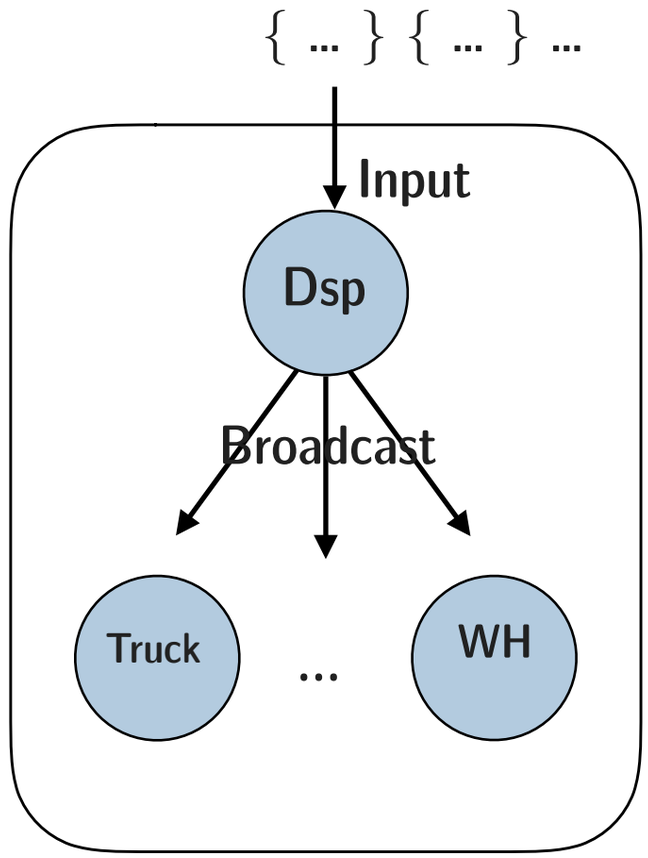
Channels

```
object Input extends Channel[JsObject]("Input", withInputPort = 5554, window = 1)
// For receiving messages from the external world
object Broadcast extends Channel[Message]("Broadcast")
// For inter-process comunication
```

Food Supply Chain Example

Dispacth Process

```
object Dispatch extends Process("Dispatcher") {
  import collection.mutable.Set
  val trucks = Set.empty[String]
  rules (
    Input --> { msg =>
      (msg \ "type").as[String] match {
        case "Waypoint" =>
          val wp = msg.toWaypoint
          if (! (trucks contains wp.truck)) {
            Scheduler.schedule(new Truck(wp.truck))
            trucks += wp.truck
          }
          Broadcast <— wp
        case "Loading" => ... //similar
        case _ => error(s"Dispatch: cannot handle message $msg")
      }
  }
)
```



Dispatcher process:

- Receive external messages
- Broadcasts messages to all processes
- Creates new Truck processes on the fly

Food Supply Chain Example

Truck Process

```
class Truck(Id: Int) extends Process("Truck") {

  val broadcast = Broadcast.subscribe()

  var location = "unknown"
  var load = Set.empty[(String, String)]

  // Externally visible state variables
  stateVar("id", ..., ...)
  stateVar("location", ..., ...)
  stateVar("load", ..., ...)

  rules(
    ... // next slide
  )
```

```
{ id = "TruckA",
  location = "Sydney"
  load = [ ("Oranges", "Batlow") ]
}
```

⇒ **Automatic mapping to/from Json**

Food Supply Chain Example

Truck process (cont'd)

```
var location = "unknown"
var load = Set.empty[(String, String)]

// This rule infers a missing Waypoint message from a Loading message
rules (
 broadcast --> {
  case msg @ Loading(time, Id, loc, goods, origin) if location != loc =>
   Broadcast <-- Waypoint(time, id, loc)  // Infer Waypoint
   Broadcast <-- msg
  ... // (cont'd)
}
```

Food Supply Chain Example

Truck process (cont'd)

```
var location = "unknown"
var load = Set.empty[(String, String)]

// This rule updates the load on this truck
rules (
 broadcast --> {
   case msg @ Loading(time, Id, loc, goods, origin) if origin != "unknown" =>
    load += (goods, origin)

   // Disjunctive rule to resolve "unknown"
   case msg @ Loading(time, Id, loc, goods, origin) if origin == "unknown" =>
    or( { Broadcast <-- Loading(time, Id, loc, goods, "Riverina") },
        { Broadcast <-- Loading(time, Id, loc, goods, "Batlow") } )
   }
)
```

Food Supply Chain Example

Truck process (cont'd)

```
var location = "unknown"
var load = Set.empty[(String, String)]

// This rule updates the load on this truck
rules (
 broadcast --> {
   case msg @ Loading(time, Id, loc, goods, origin) if origin != "unknown" =>
    load += (goods, origin)

   // Disjunctive rule to resolve "unknown"
   case msg @ Loading(time, Id, loc, goods, origin) if origin == "unknown" =>
    or( { Broadcast <-- Loading(time, Id, loc, goods, "Riverina") },
        { Broadcast <-- Loading(time, Id, loc, goods, "Batlow") } )
   }
)
```

Rules are allowed to "fail"

Food Supply Chain Example

Truck process (cont'd)

```
var location = "unknown"
var load = Set.empty[(String, String)]

// This rule updates the load on this truck
rules (
 broadcast --> {
   case msg @ Loading(time, Id, loc, goods, origin) if origin != "unknown" =>
    load += (goods, origin)

   // Disjunctive rule to resolve "unknown"
   case msg @ Loading(time, Id, loc, goods, origin) if origin == "unknown" =>
    or( { Broadcast <-- Loading(time, Id, loc, goods, "Riverina") },
        { Broadcast <-- Loading(time, Id, loc, goods, "Batlow") } )
   }
)
```

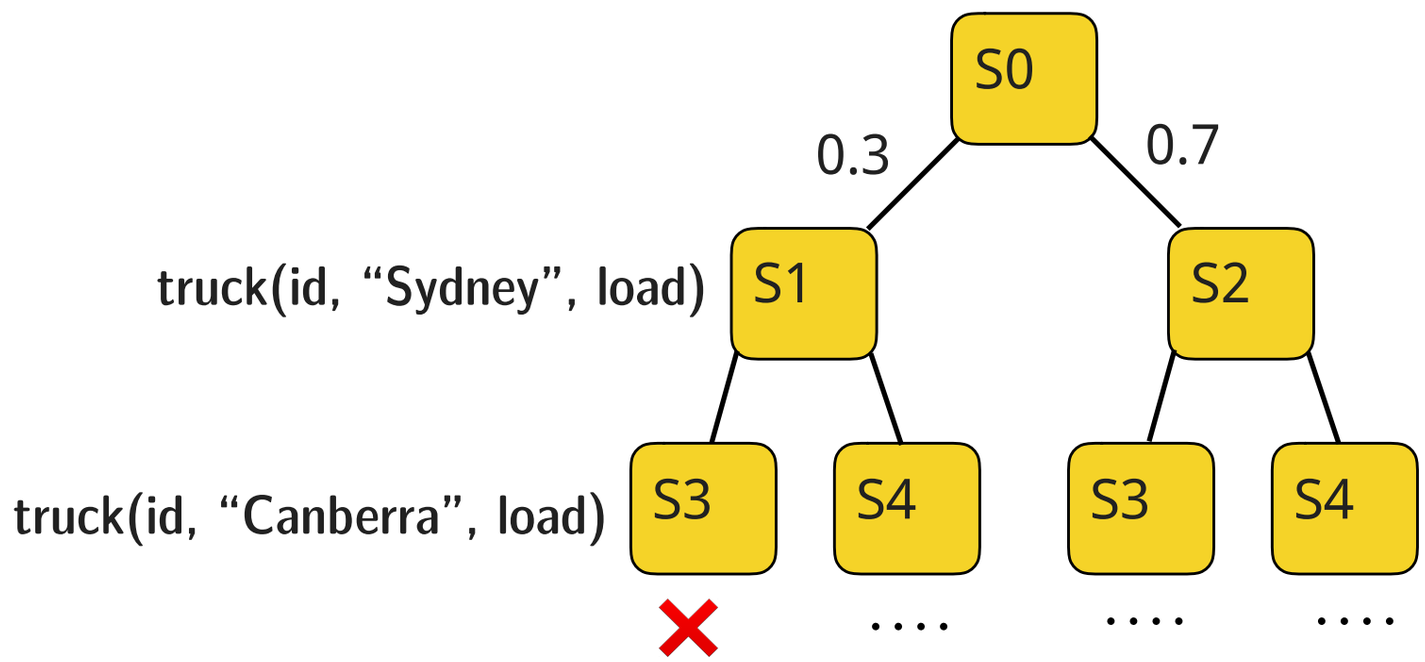
Rules are allowed to "fail"

What else?

- State Maintenance: "no oranges from Batlow"
- Inference engine algorithm and implementation
- Scheduler algorithm and implementation

Next Steps

- Improve and complete implementation
- Realistic food supply chain based on EPCIS events
- Probabilistic state transitions
- State maintenance *trees*
- State maintenance declaratively specified by temporal logic constraints
  - $\mathbf{G}\ (\mathrm{truck}(\mathrm{id}, \text{“Sydney”}, \mathrm{load}) \rightarrow \neg\mathbf{F}\ (\mathrm{truck}(\mathrm{id}, \text{“Canberra”}, \mathrm{load}))$



Next Steps

- Improve and complete implementation
- Realistic food supply chain based on EPCIS events
- Probabilistic state transitions
- State maintenance *trees*
- State maintenance declaratively specified by temporal logic constraints
  - $\mathbf{G}\ (\mathrm{truck}(\mathrm{id}, \text{“Sydney”}, \mathrm{load}) \rightarrow \neg\mathbf{F}\ (\mathrm{truck}(\mathrm{id}, \text{“Canberra”}, \mathrm{load}))$