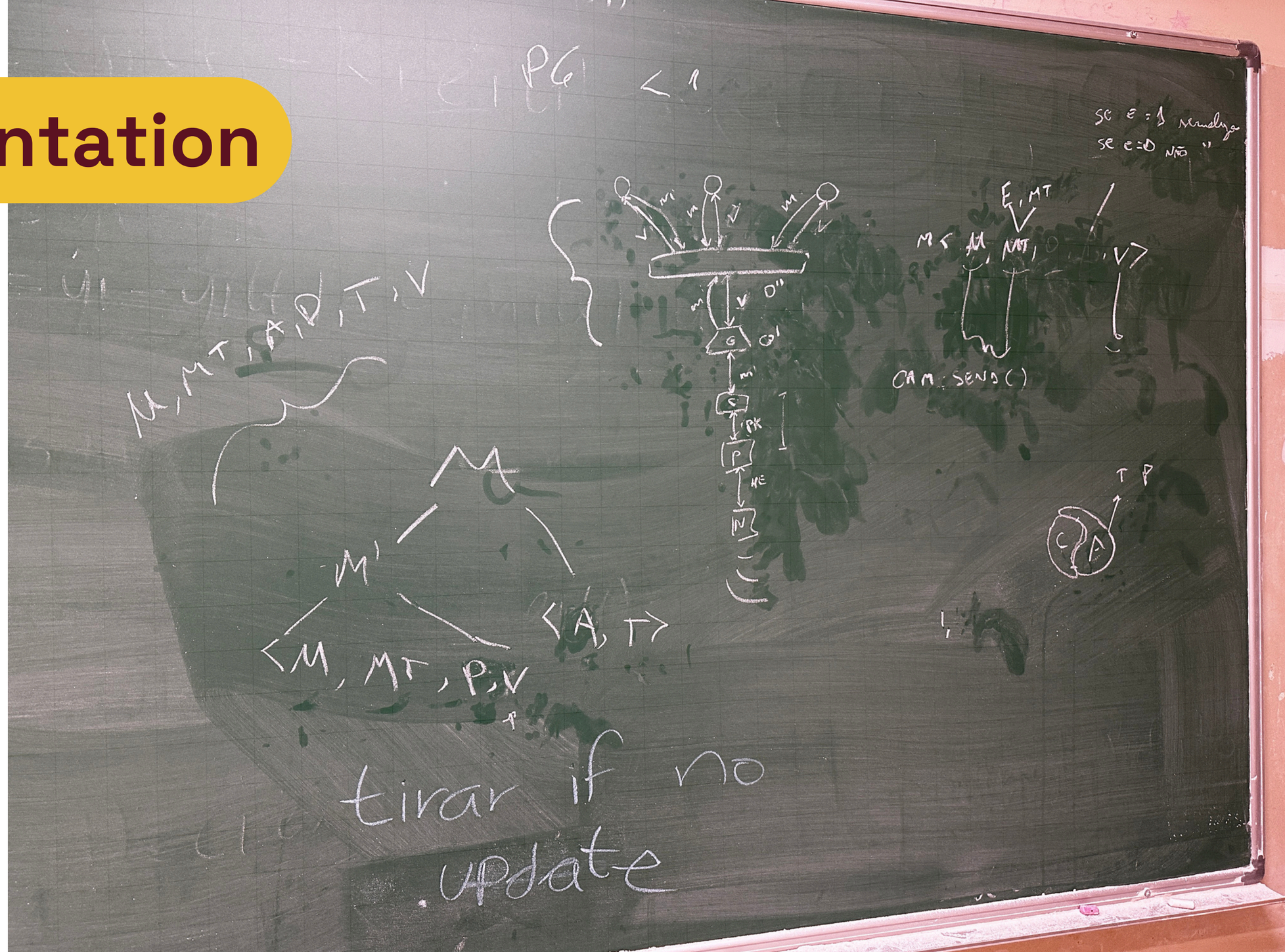
Reliable and secure communication library for critical autonomous systems

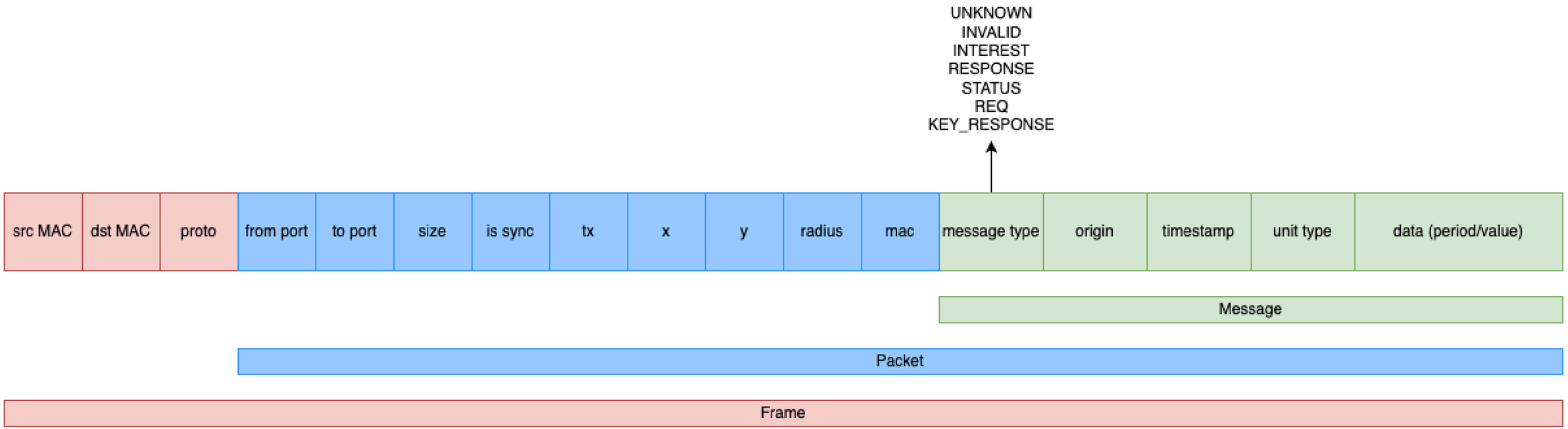
CÉSAR AUGUSTO PEREIRA DE SOUZA
ENZO NICOLÁS SPOTORNO BIEGER
JOÃO PEDRO PEREZ RESMER
JOÃO PEDRO SCHMIDT CORDEIRO

P6 18/06/2025

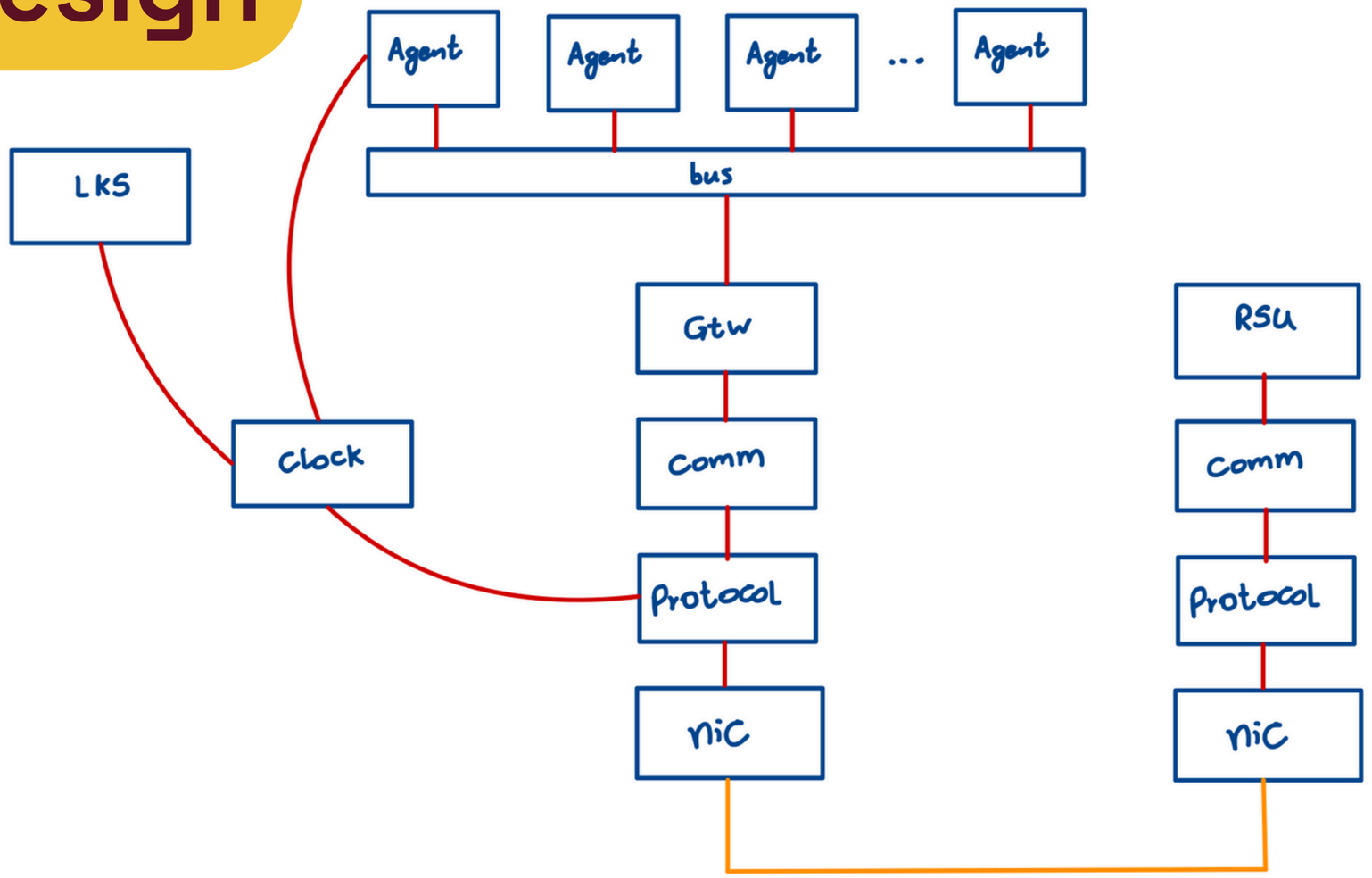
Implementation Plan

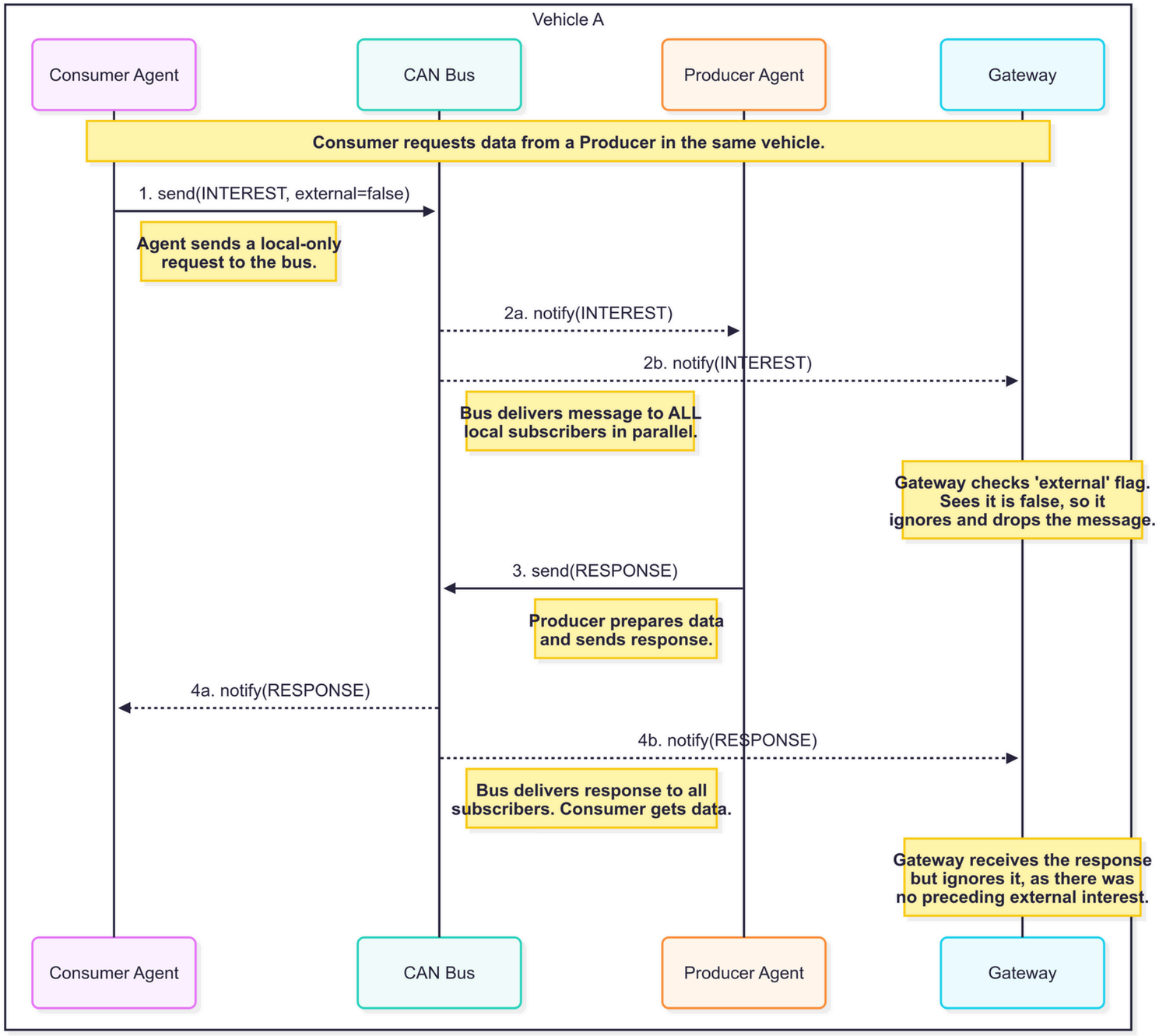


Messsage


UNKNOWN
INVALID
INTEREST
RESPONSE
STATUS
REQ
KEY_RESPONSE
src MAC
dst MAC
proto
from port
to port
size
is sync
tx
x
y
radius
mac
message type
origin
timestamp
unit type
data (period/value)
Message
Packet
Frame

System Design




Vehicle A
Consumer Agent
CAN Bus
Producer Agent
Gateway
Consumer requests data from a Producer in the same vehicle.
1. send(INTEREST, external=false)
Agent sends a local-only request to the bus.
2a. notify(INTEREST)
2b. notify(INTEREST)
Bus delivers message to ALL local subscribers in parallel.
Gateway checks 'external' flag. Sees it is false, so it ignores and drops the message.
3. send(RESPONSE)
Producer prepares data and sends response.
4a. notify(RESPONSE)
4b. notify(RESPONSE)
Bus delivers response to all subscribers. Consumer gets data.
Gateway receives the response but ignores it, as there was no preceding external interest.
Consumer Agent
CAN Bus
Producer Agent
Gateway

Agent

Constructor

```
154     db<Agent>(INF) << "[Agent] " << _name << " created with address: " << _address.to_s
155     if (!bus)
156         throw std::invalid_argument(s: "Gateway cannot be null");
157
158     Condition c(unit, type);
159     _c = c;
160     _can_observer = new Observer(c);
161     _can->attach(_can_observer, c);
162
163     if (_is_consumer && !handler) {
164         throw std::invalid_argument(s: "Consumer agents must have a response handler");
165     }
166     if (!_is_consumer && !producer) {
167         throw std::invalid_argument(s: "Producer agents must have a data producer");
168     }
169
170     // Phase 1.2: Consumer initialization - No automatic INTEREST sending
171     if (_is_consumer) {
172         // Don't send initial INTEREST here anymore
173         // Application will call start_periodic_interest() when ready
174         db<Agent>(INF) << "[Agent] " << _name << " initialized as consumer, waiting for
175     } else {
176         db<Agent>(INF) << "[Agent] " << _name << " initialized as producer, ready to ha
177     }
178
179     _running = true;
180     int result = pthread_create(&_thread, nullptr, Agent::run, this);
181     if (result != 0) {
182         _running = false;
183         throw std::runtime_error("Failed to create agent thread");
184     }
185 }
```

Agent

Send

```
309  inline int Agent::send(Unit unit, Microseconds period) {
310      db<Agent>(INF) << "[Agent] " << _name << " sending INTEREST for
311      if (period == Microseconds::zero())
312          return 0;
313
314      // Store the INTEREST period for RESPONSE filtering
315      _interest_period = period;
316
317      Message msg(Message::Type::INTEREST, _address, unit, period);
318      msg.external(_external);
319
320      // Log sent message to CSV
321      log_message(msg, "SEND");
322
323      int result = _can->send(&msg);
324
325      if (!result)
326          return -1;
327
328      return result;
329  }
```

CAN

```
52  class CAN : public Concurrent_Observed<Initializer::Message, Condition>{
53      public:
54          typedef Initializer::Message Message;
55          typedef Initializer::Protocol_T::Address Address;
56          typedef Message::Unit Unit;
57          typedef Message::Type Type;
58          typedef Concurrent_Observer<Message, Condition> Observer;
59          typedef Concurrent_Observed<Message, Condition> Observed;
60
61          CAN() = default;
62          ~CAN() = default;
63
64          int send(Message* msg);
65          bool notify(Message* buf, Condition c) override;
66  };
67
68  int CAN::send(Message* msg) {
69      db<CAN>(TRC) << "CAN::send() called!\n";
70      Condition c(msg->unit(), msg->message_type());
71      if (!notify(buf: msg, c))
72          return 0;
73
74      return msg->size();
75  }
76
77  bool CAN::notify(Message* buf, Condition c) {
78      pthread_mutex_lock(&_mtx);
79      bool notified = false;
80
81      db<CAN>(INF) << "Notifying observers...\n";
82
83      for (typename Observers::Iterator obs = _observers.begin(); obs != _observers.end(); ++obs) {
84
85          if ((*obs)->rank() == c || (*obs)->rank().type() == Condition::Type::UNKNOWN) {
86              Message* msg = new Message(*buf);
87              (*obs)->update((*obs)->rank(), msg);
88              notified = true;
89          }
90      }
91
92      pthread_mutex_unlock(&_mtx);
93      return notified;
94  }
```

Gateway

```
211  inline void* Gateway::mainloop(void* arg) {
212      Gateway* self = reinterpret_cast<Gateway*>(arg);
213
214      db<Gateway>(INF) << "[Gateway " << self->_id << "] exter
215
216      while (self->running()) {
217          Message msg;
218          if (self->receive(message: &msg)) {
219              self->handle(message: &msg);
220          }
221      }
222
223      db<Gateway>(INF) << "[Gateway " << self->_id << "] exter
224      return nullptr;
225  }
```

Gateway

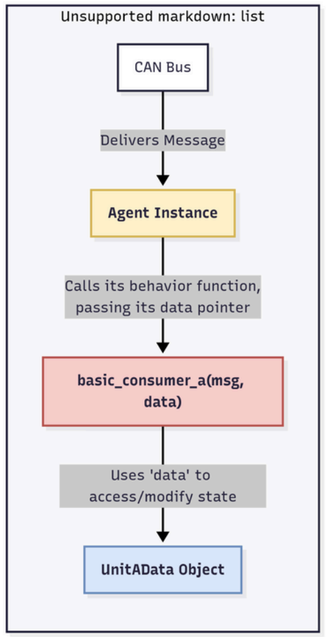
```
247  inline void* Gateway::internalLoop(void* arg) {
248      Gateway* self = reinterpret_cast<Gateway*>(arg);
249
250      db<Gateway>(INF) << "[Gateway " << self->_id << "] internal receive loop
251
252      while (self->running()) {
253          Message msg;
254          if (self->internalReceive(msg: &msg)) {
255              // CRITICAL FIX: Check if message originated from this gateway to
256              if (msg.origin() == self->_comms->address() || !msg.external()) {
257                  db<Gateway>(INF) << "[Gateway " << self->_id << "] ignoring i
258                                   << msg.origin().to_string() << ", self: " <<
259                  continue;
260              }
261
262              db<Gateway>(INF) << "[Gateway " << self->_id << "] forwarding int
263                               << msg.origin().to_string() << "\n";
264              self->send(message: &msg);
265          }
266      }
267
268      db<Gateway>(INF) << "[Gateway " << self->_id << "] internal receive loop
269      return nullptr;
270  }
```

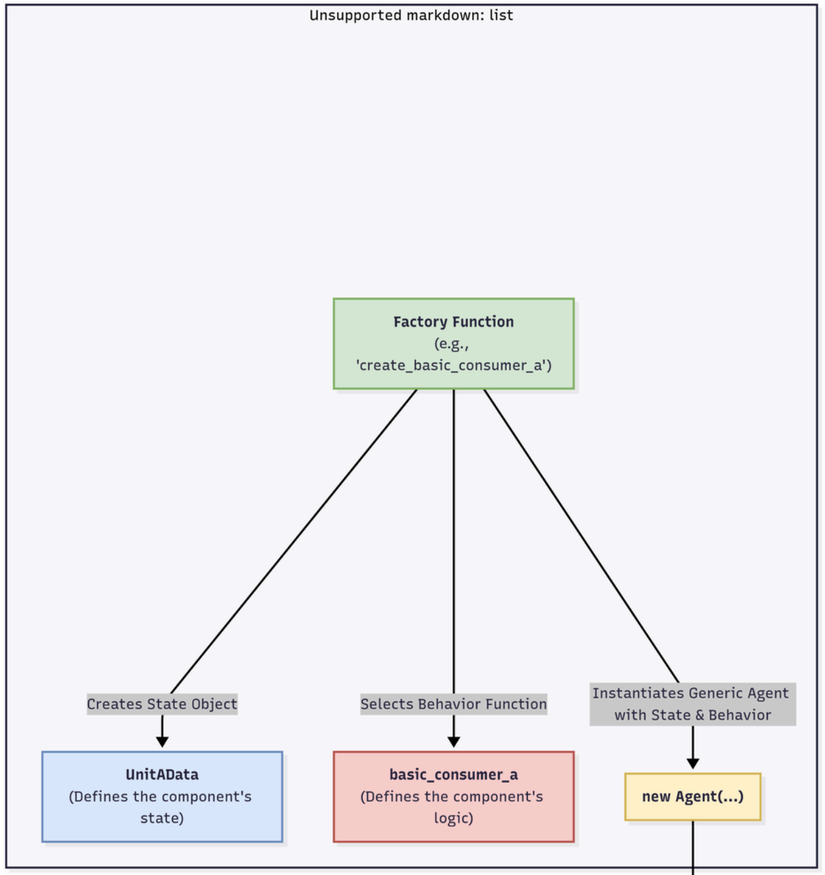
Reliable and secure communication library for critical autonomous systems

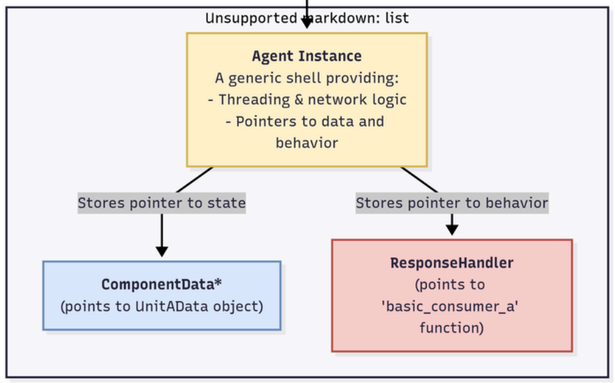
CÉSAR AUGUSTO PEREIRA DE SOUZA
ENZO NICOLÁS SPOTORNO BIEGER
JOÃO PEDRO PEREZ RESMER
JOÃO PEDRO SCHMIDT CORDEIRO

P6 18/06/2025

Agent


Unsupported markdown: list
CAN Bus
Delivers Message
Agent Instance
Calls its behavior function, passing its data pointer
basic_consumer_a(msg, data)
Uses 'data' to access/modify state
UnitAData Object


Unsupported markdown: list
Factory Function (e.g., 'create_basic_consumer_a')
Creates State Object
Selects Behavior Function
Instantiates Generic Agent with State & Behavior
UnitAData (Defines the component's state)
basic_consumer_a (Defines the component's logic)
new Agent(...)


Unsupported markdown: list
Agent Instance
A generic shell providing:
- Threading & network logic
- Pointers to data and behavior
Stores pointer to state
Stores pointer to behavior
ComponentData*
(points to UnitAData object)
ResponseHandler
(points to 'basic_consumer_a' function)