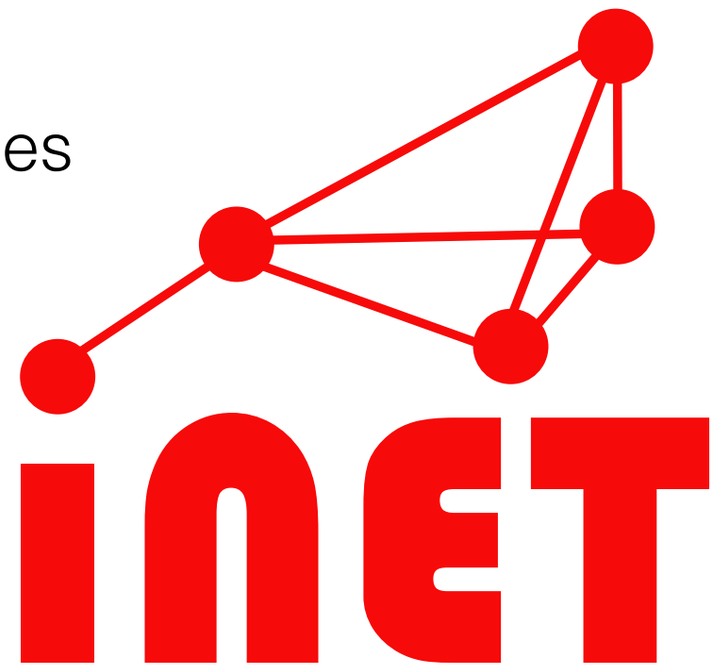
IRTF - T2TRG

Programming the IoT with C++ Actors

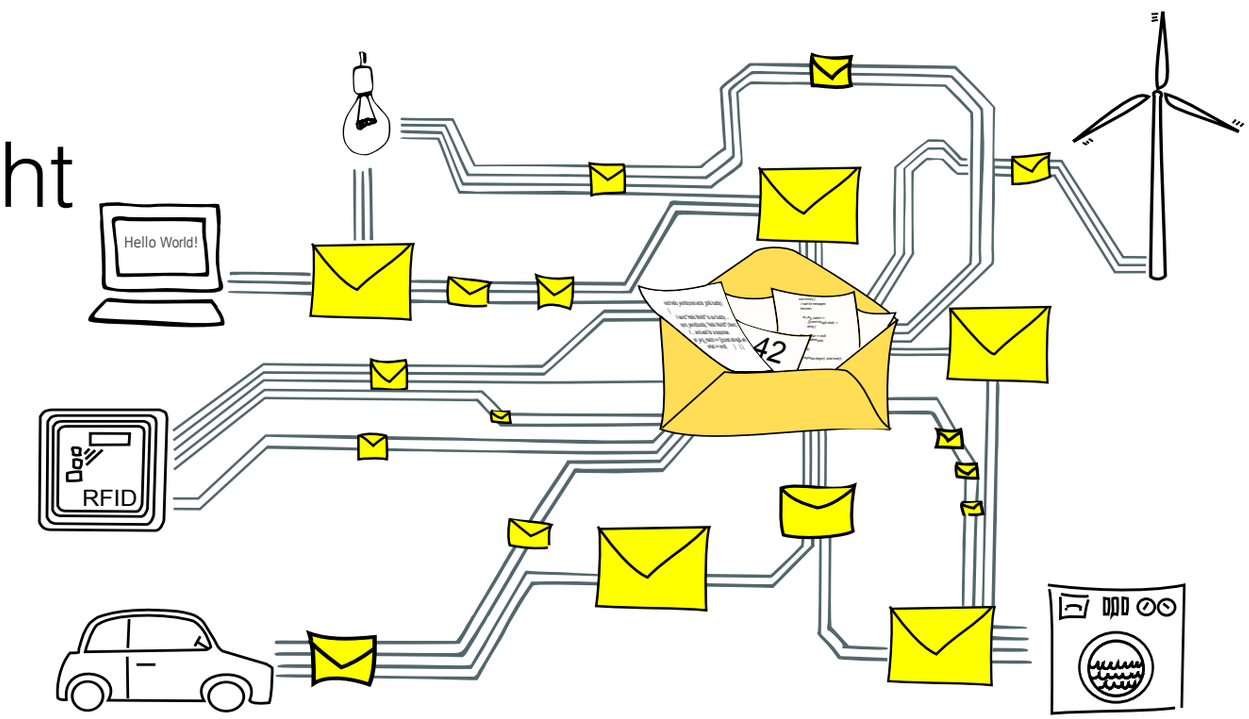
Dominik Charousset, Raphael Hiesgen, Thomas C. Schmidt
{dominik.charousset, raphael.hiesgen, t.schmidt}@haw-hamburg.de

Internet Technologies Group
Hamburg University of Applied Sciences
http://www.haw-hamburg.de/inet



Outline

1. Actor Model
2. C++ Actor Framework
3. Programming the IoT
4. CAF vs. REST
5. Use Case: Intelligent Light

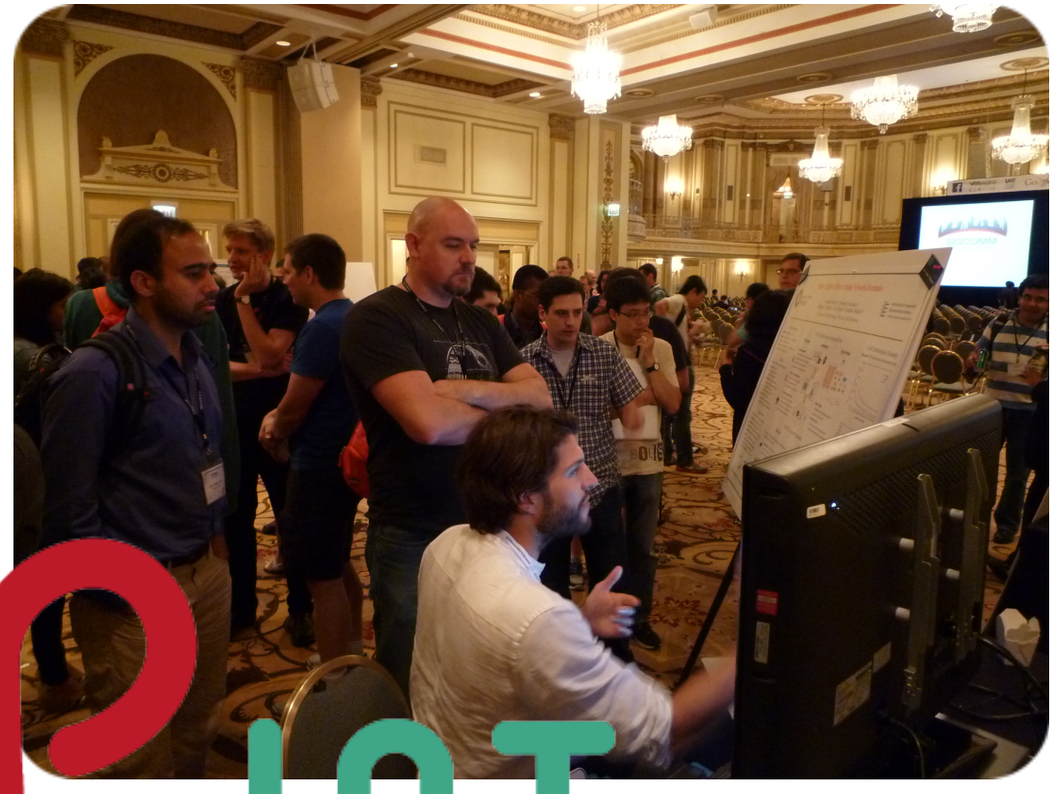


The Actor Model

- Concept for concurrency and distribution (Hewitt et al. 1973)
- Lightweight and isolated software entities: actors
- E2E message passing
- Features
  - Divide & conquer via “spawn”
  - Strong, hierarchical failure model
  - Re-deployment at runtime

C++ Actor Framework - CAF

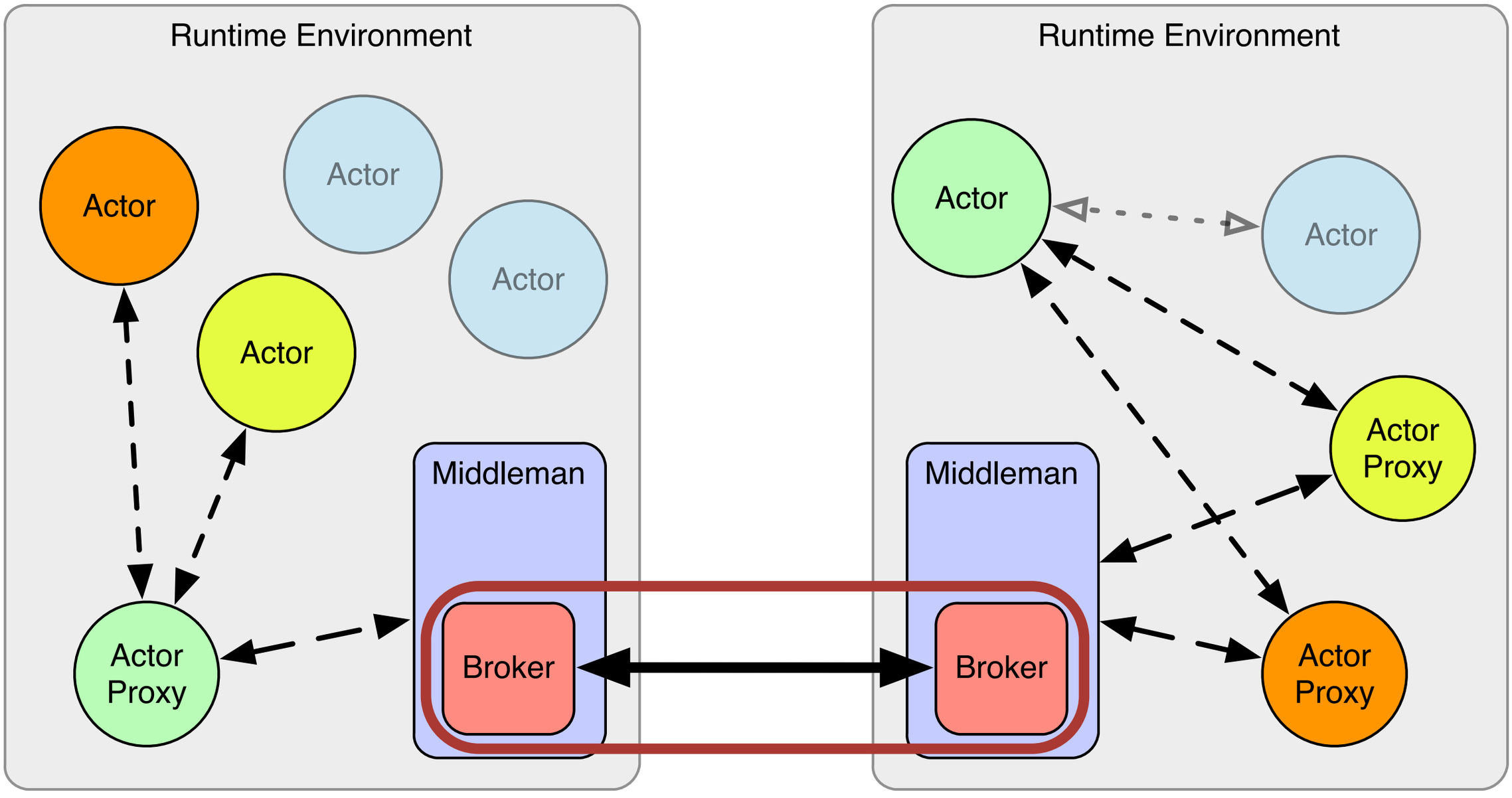
- Open Source Community since 2011
  - Developers from Europe, North America & Asia
  - https://github.com/actor-framework/ with ~100 Forks
- Focus on Scalability
  - Up: Thousands of Cores
  - Down: IoT Nodes
  - Sideways: GPUs
- User base in powerful environments
- … and soon:



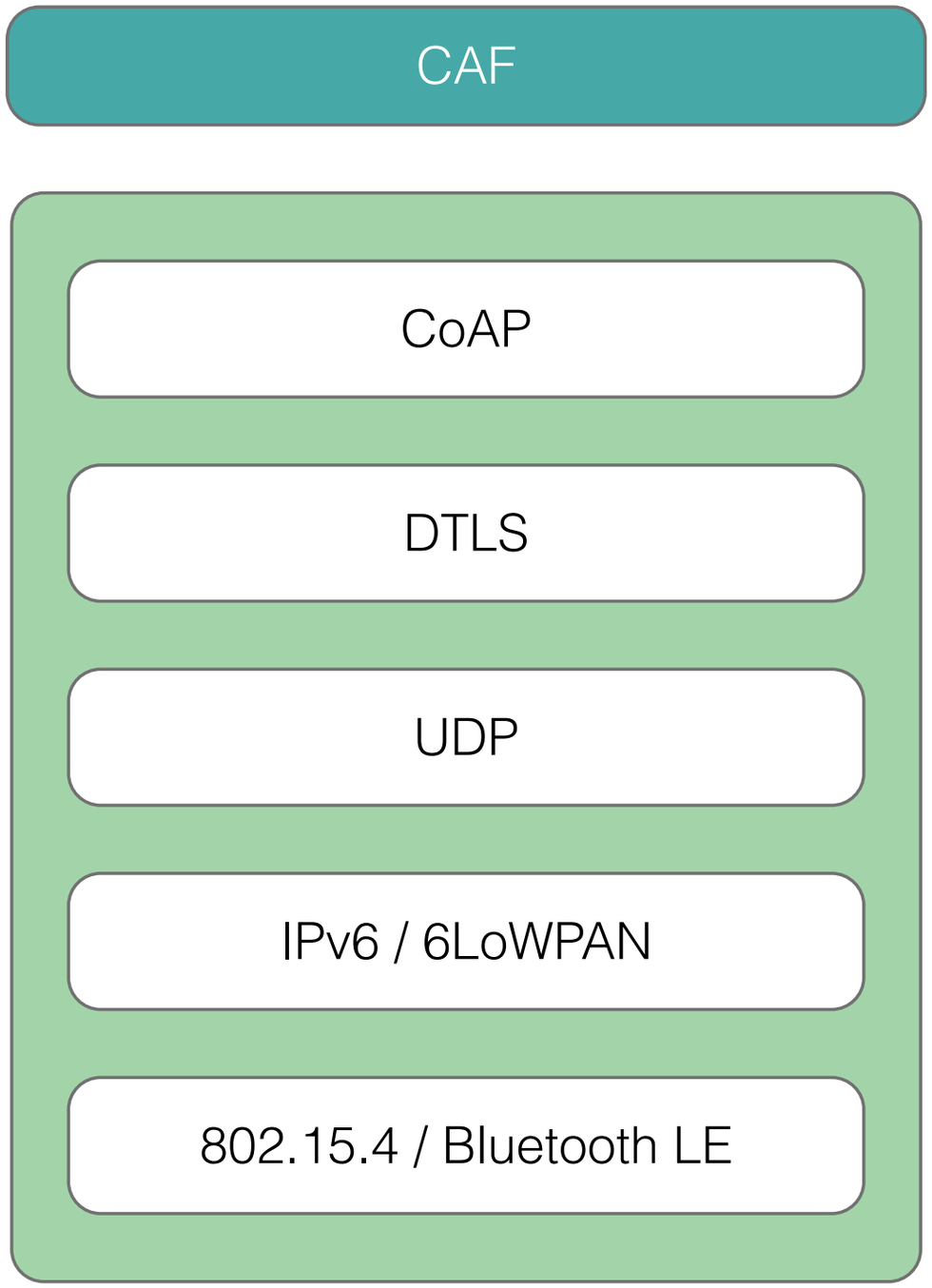
Programming the IoT

- Professionalize IoT software development
  - Reusability, portability
  - Robustness (shared nothing)
- Establish a common programming model
  - Highly distributed application design
  - High level of abstraction
  - Distributed error-handling
- Promote experimentally driven research
  - IoT environments often unpredictable

CAF Software Architecture



Core Approach



- Map CAF messages to CoAP
  - Sync messages —> Reliability (CON)
  - Async messages —> Unreliability (NON)
- Handle small frame sizes
  - Compress meta-information to slim down headers
  - Fragmentation on the application layer (CoAP block messages)
- New concept for error-propagation required
  - No longer connection oriented
  - Take unreliable messages into account
- Security
  - DTLS & ID-based crypto with ECC

Communication with CAF

- Types give data meaning
  - Celsius vs. Fahrenheit, Feet vs. Meter, …
  - Compile time validation of message types
- Brokers can translate data formats
  - Protobuf, CBOR, …
- Group communication
  - Eases service discovery and rendezvous processes
  - Publish / subscribe patterns for sensors

CAF vs REST I

CAF	REST
End-to-End messaging, transparently forwarded by MM	Server translates messages, two-step process
Automatically spawned in a preconfigured thread-pool	Manual resource management, i.e., create new threads
Messaging independent of technology (IPC, big.LITTLE, bus, …)	Transport binding
Group communication (no central control, pub / sub)	Pub / Sub via centralized broker
No specified architecture	Usually client-sever

CAF vs REST II

CAF	REST
Abstract messaging model (atoms, domain specific type)	Plain text, XML, Jason and mime types
Interface composed of actor interfaces	Interface located at the REST server
Easy to firewall (discard / type safe interfaces)	Requires parsing of content
Inherent error model (with fast response time)	Error codes
State shared via messages (initialization, migration of actors)	State shared via messages

Use Case: Intelligent Lightening

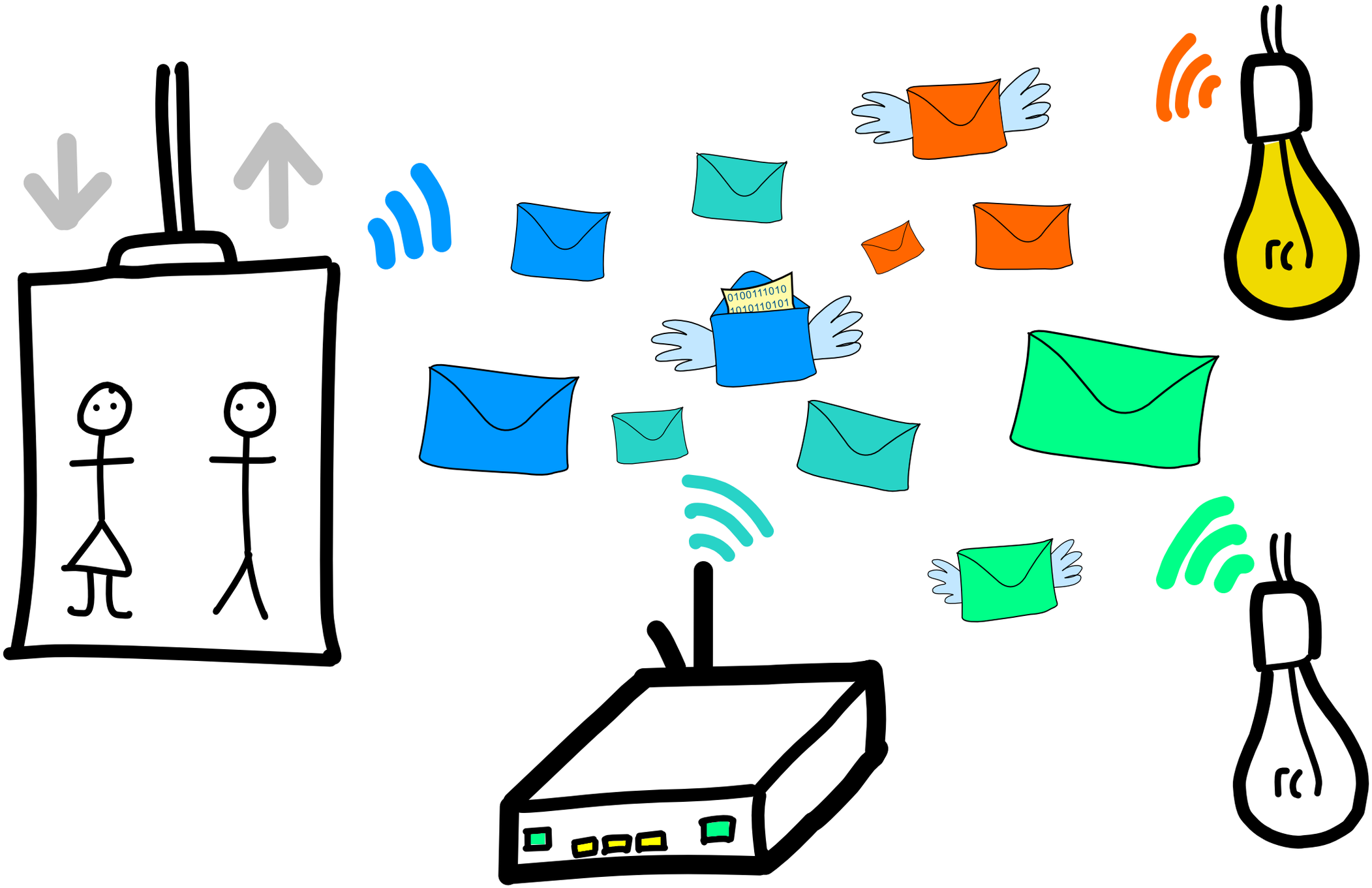
Intelligent lightening system switches lights by predicting user demands. This includes third party information from elevators, doors, etc.

- Distributed heterogeneous system
    - Cooperation between lights, elevators, locks, …
- Open interfaces
    - Messaging standards for application domains
    - Allow cooperation of different vendors, i.e., lights and elevators
- Dynamic error handling
    - Raise maintenance alarm
    - Turn nearby lights on as a fallback

Setup

- Motion Sensor
  - Notify lights when movement is detected
- Lights
  - Turn on if triggered by another sensor
  - Turn off after timeout
- Elevators & Locks
  - Trigger lights at the destinations of users

Devices Interact Directly



Sensor-Light: Idle

```
// self is used to access actor specific functionality
// passed by the runtime as the first argument
behavior idle(event_based_actor* self) {
  return {
    // atoms are annotations to give messages
    // meaning beyond their types
    [=](on_atom) {
      turn_light_on();
      // 1st arg allows us to change back to this behavior
      // 2nd arg is the new behavior, see next slide
      self->become(keep_behavior, glowing(self));
    }
  };
}
```

Sensor-Light: Glowing

```
behavior glowing(event_based_actor* self) {
  return {
    // receiving an on_atom resets the timer
    [=](on_atom) { /* NOP */ },
    // turn off if no other message
    // is received within the timeout
    after(TIMEOUT_TIME) >> [] {
      turn_light_off();
      // switch back to last behavior: idle
      self->unbecome();
    }
  };
}
```

Elevator: Notify Lights

```
// Handle notifications from elevator control
behavior elevator(event_based_actor* self) {
  return {
    // received when a button is pressed
    [=](request_atom, int floor) {
      register_destination(floor);
    },
    // received before arriving at a new floor
    [=](arriving_atom, int floor) {
      if (has_request_for(floor)) {
        self->send(actor_located_at(floor), on_atom);
      }
    }
  };
}
```

Design Space

- Motion sensor
  - High level API to register interested actors:
    `motion_sensor.add_listener(light);`
- Elevator control (multiple elevators)
  - Sensors in the same well register in a group (control panels, supervisors, …)
  - Elevators subscribe to group of control messages
  - Multiple elevators require complex scheduling

Conclusion

- Internet of Things
  - Professionalization of development
  - Establish a common programming model
- The Actor Model
  - Designed to address distributed system
  - Isolated entities that solely communicate via message passing
- C++ Actor Framework
  - Adjusted network stack for IoT environments
  - Advantages: E2E messaging, defined interfaces, group communication, …

Thanks!

http://www.actor-framework.org/

https://github.com/actor-framework/

Dominik Charousset, Raphael Hiesgen, Thomas C. Schmidt
{dominik.charousset, raphael.hiesgen, t.schmidt}@haw-hamburg.de

Internet Technologies Group
Hamburg University of Applied Sciences
http://www.haw-hamburg.de/inet