INTERNET OF THE THINGS (IoT): An introduction to wireless sensor networking middleware

Dr Antoine Bagula
ISAT Laboratory, University of Cape Town, South Africa

Goal of the lecture

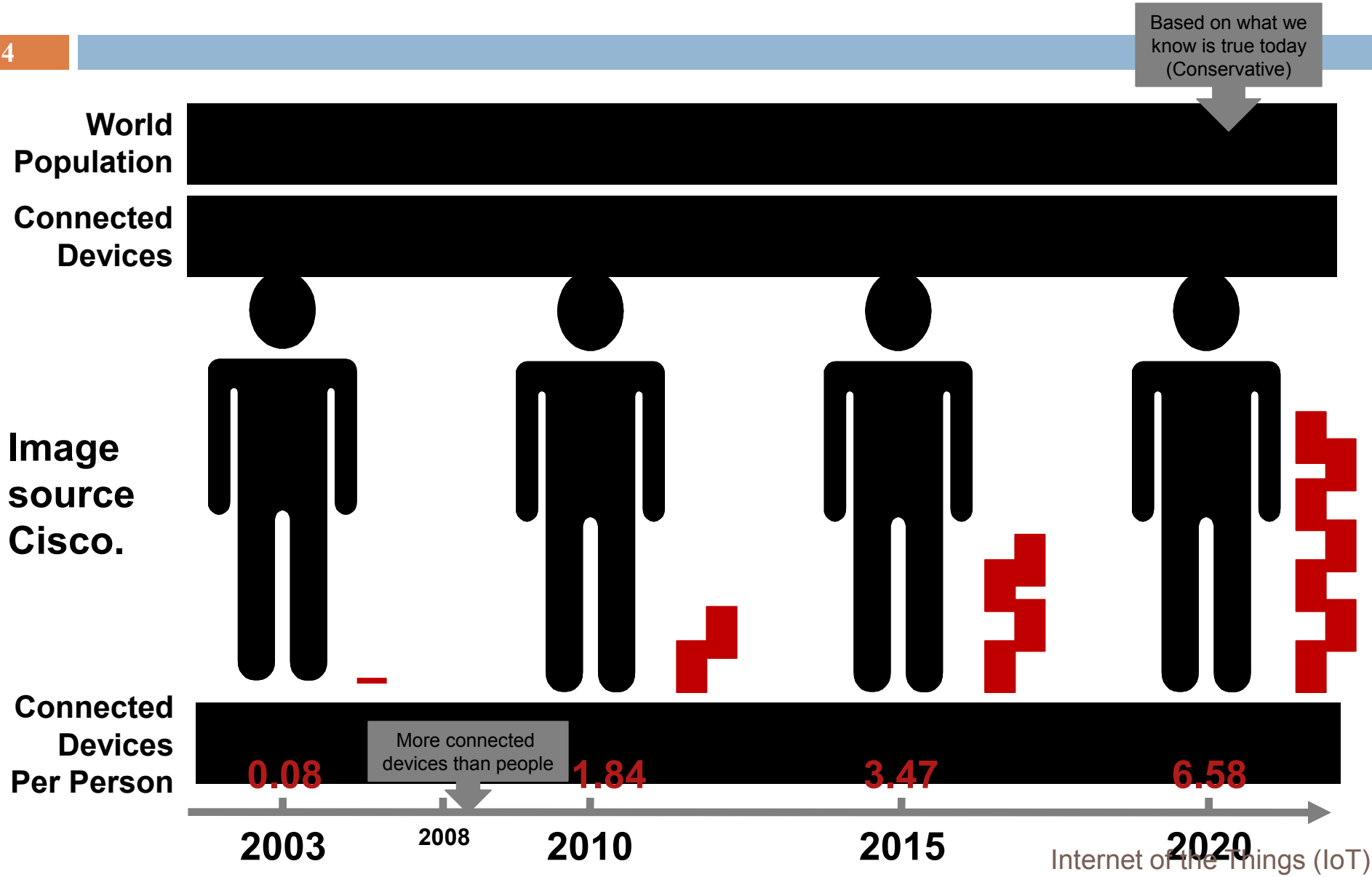
The lecture intends to introduce the key concepts behind an Internet of Things (IoT) middleware by

- Positioning the middleware into a ubiquitous sensor networking stack.
- Describing the main approaches associated with an IoT middleware.
- Highlighting some of the research works illustrating these middleware approaches.

I. Outline

A. What is the IoT ?
B. IoT Architecture
C. What is a middleware ?
D. Middleware approaches

A. What is IoT ?


Based on what we know is true today (Conservative)
World Population
Connected Devices
Image source Cisco.
Connected Devices Per Person
More connected devices than people
0.08
1.84
3.47
6.58
2003
2008
2010
2015
2020

What is IoT ?

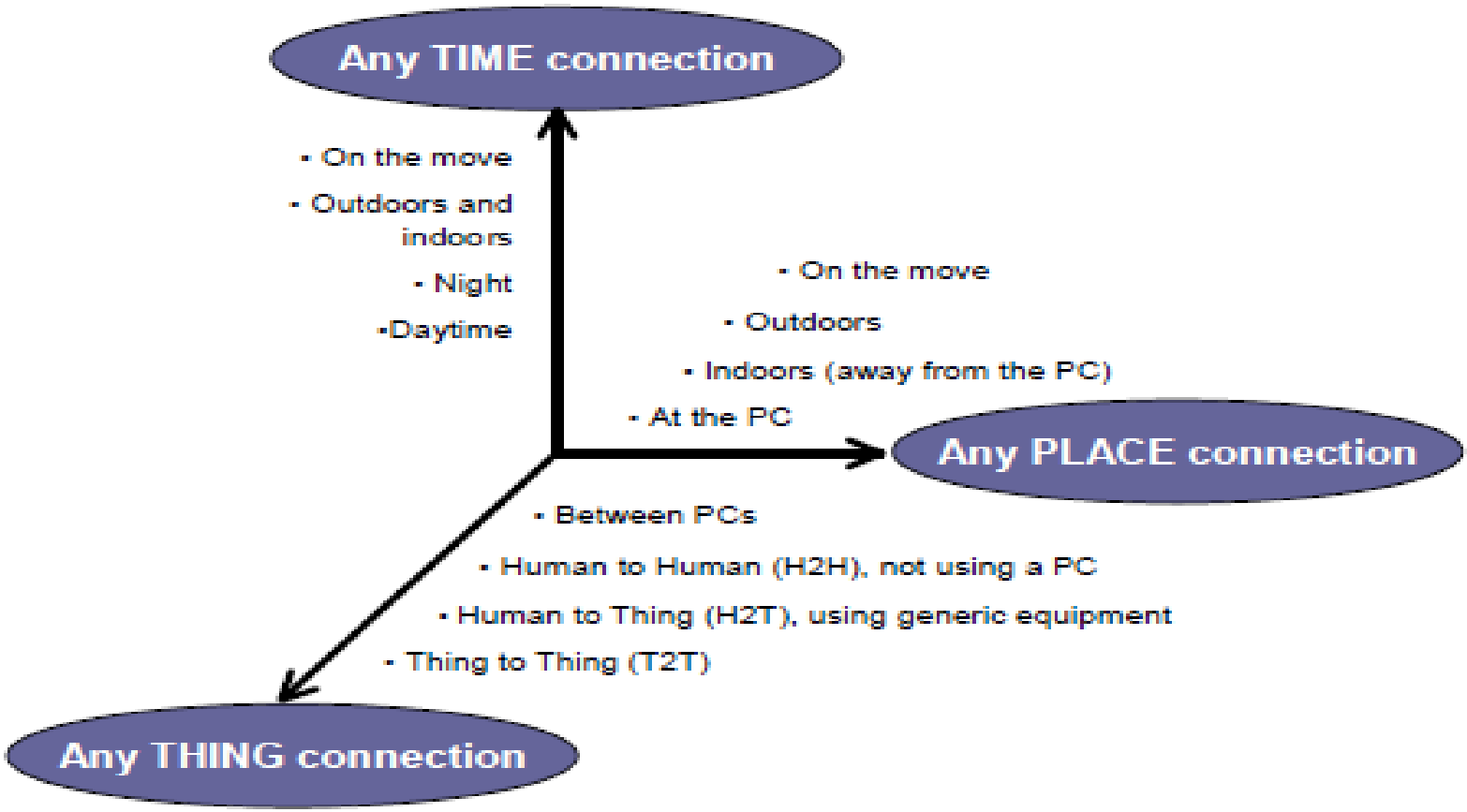
- The Internet is a world-wise network of interconnected computer networks, based on a standard communication protocol (TCP/IP)
- The Internet of Things (IoT) is
  - a world-wide network of interconnected objects which are outfitted with sensors, actuators and RFID devices.
  - These devices are uniquely addressable and use standard communication protocols in a heterogeneous networking environment including objects of totally different functionality, technology and application fields.

What is IoT ?

Building upon the revolutionary success of mobile communication, Internet networks, the advances in sensor, RFID, Nano and other smart technologies, the IoT expands the current Internet with the objective of

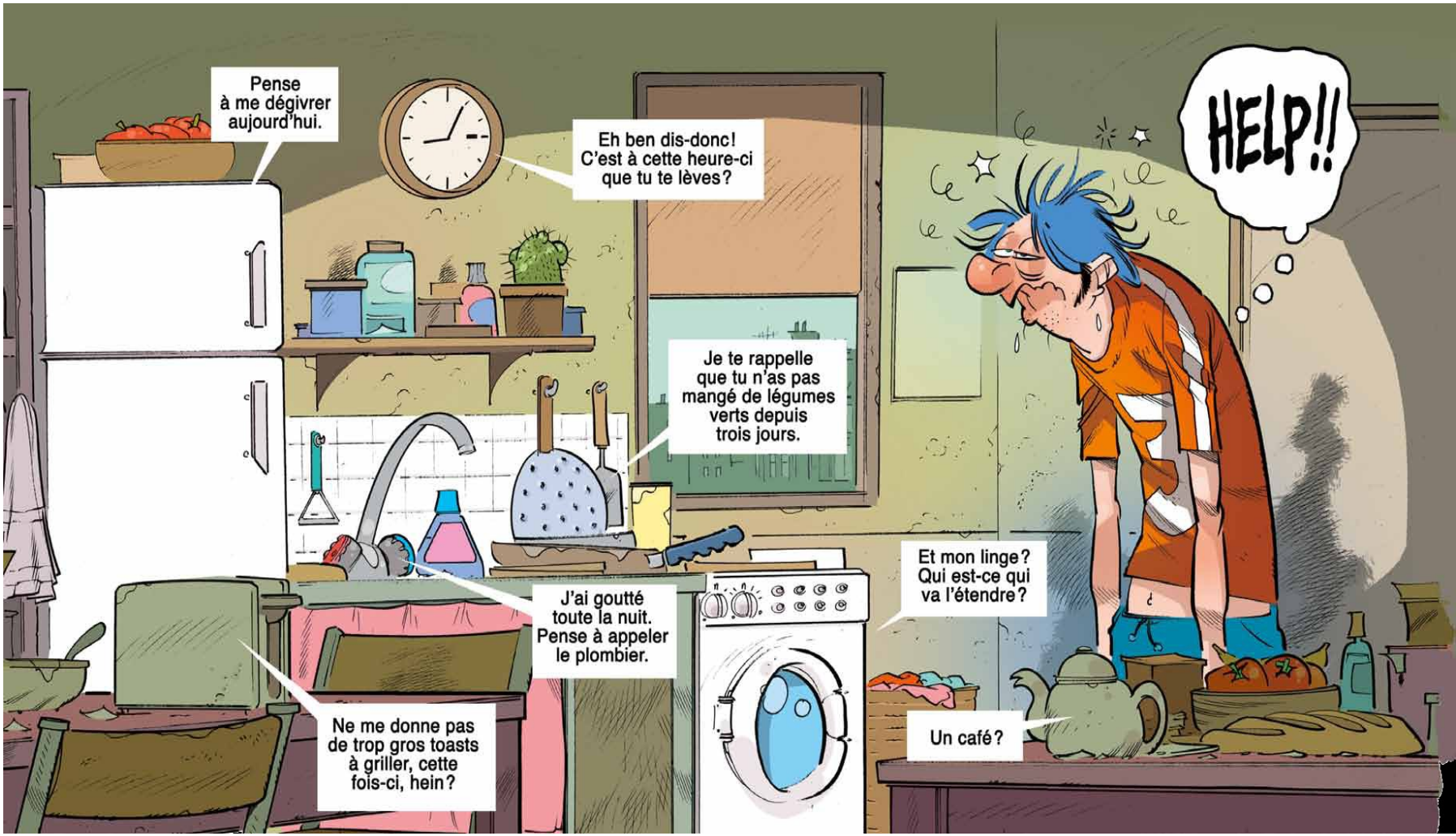
- Connecting objects outfitted with sensor and RFID devices (smart objects).
- Detecting changes in the physical status of connected things in real-time.
- Identifying and localizing the smart objects.
- Monitoring the changes in the physical status of connected things.

What is IoT ?



Source: ITU Internet Reports 2005: The Internet of Things

What is IoT ?


Pense à me dégivrer aujourd'hui.
Eh ben dis-donc! C'est à cette heure-ci que tu te lèves?
Je te rappelle que tu n'as pas mangé de légumes verts depuis trois jours.
J'ai goutté toute la nuit. Pense à appeler le plombier.
Ne me donne pas de trop gros toasts à griller, cette fois-ci, hein?
HELP!!
Et mon linge? Qui est-ce qui va l'étendre?
Un café?

B. IoT architecture

Source: ITU-T study group on Climate Change

IoT architecture

The Figure above reveals the different layers of a Ubiquitous Sensor Network

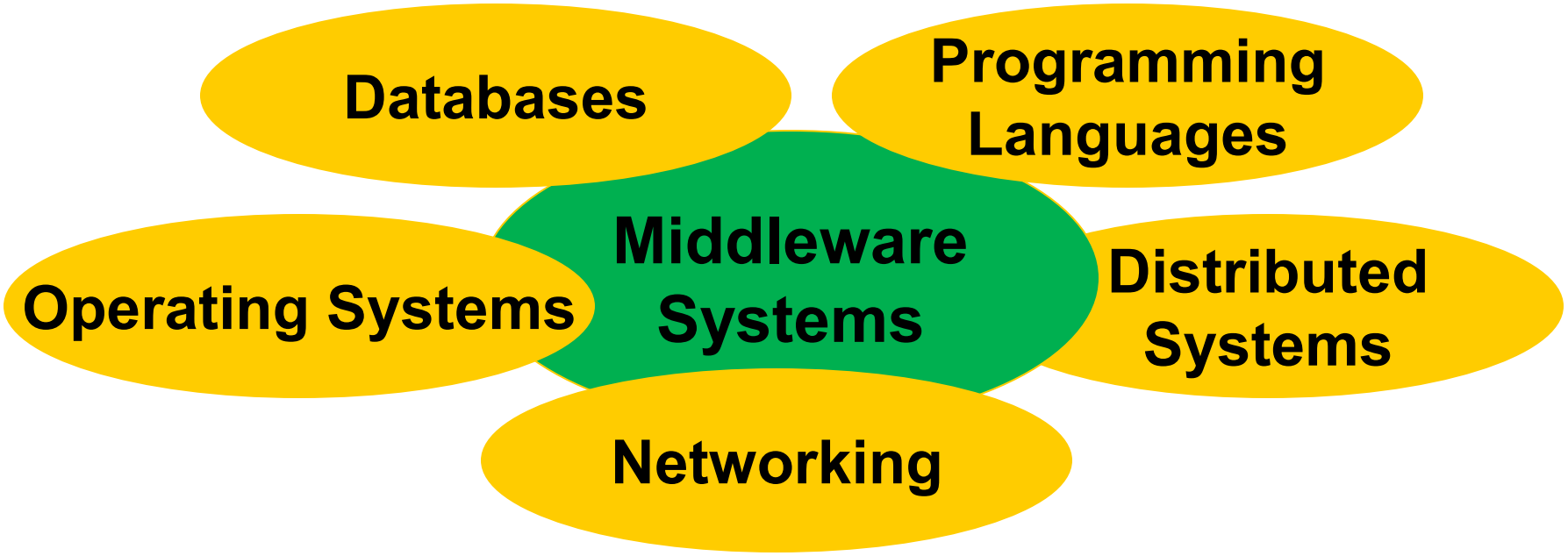
- *A sensor networking* layer (the bottom layer) where sensor and RFID devices are launched into the environment to sense what is happening and report to sink nodes via USN-bridges
- A USN access networking layer (the second layer) where the combination of USN-bridges and sink nodes are used as hosts to an access network for the *first-mile* connectivity of a Next Generation Network (NGN) of gateways

.

IoT architecture

- A USN middleware layer (third layer) is used as an interface between the NGN and the application layer.
- A USN applications layer (the last layer) where different applications are used to perform tasks related to logistics, structural health monitoring, agriculture control, disaster Surveillance, military field surveillance and disaster/crisis management.

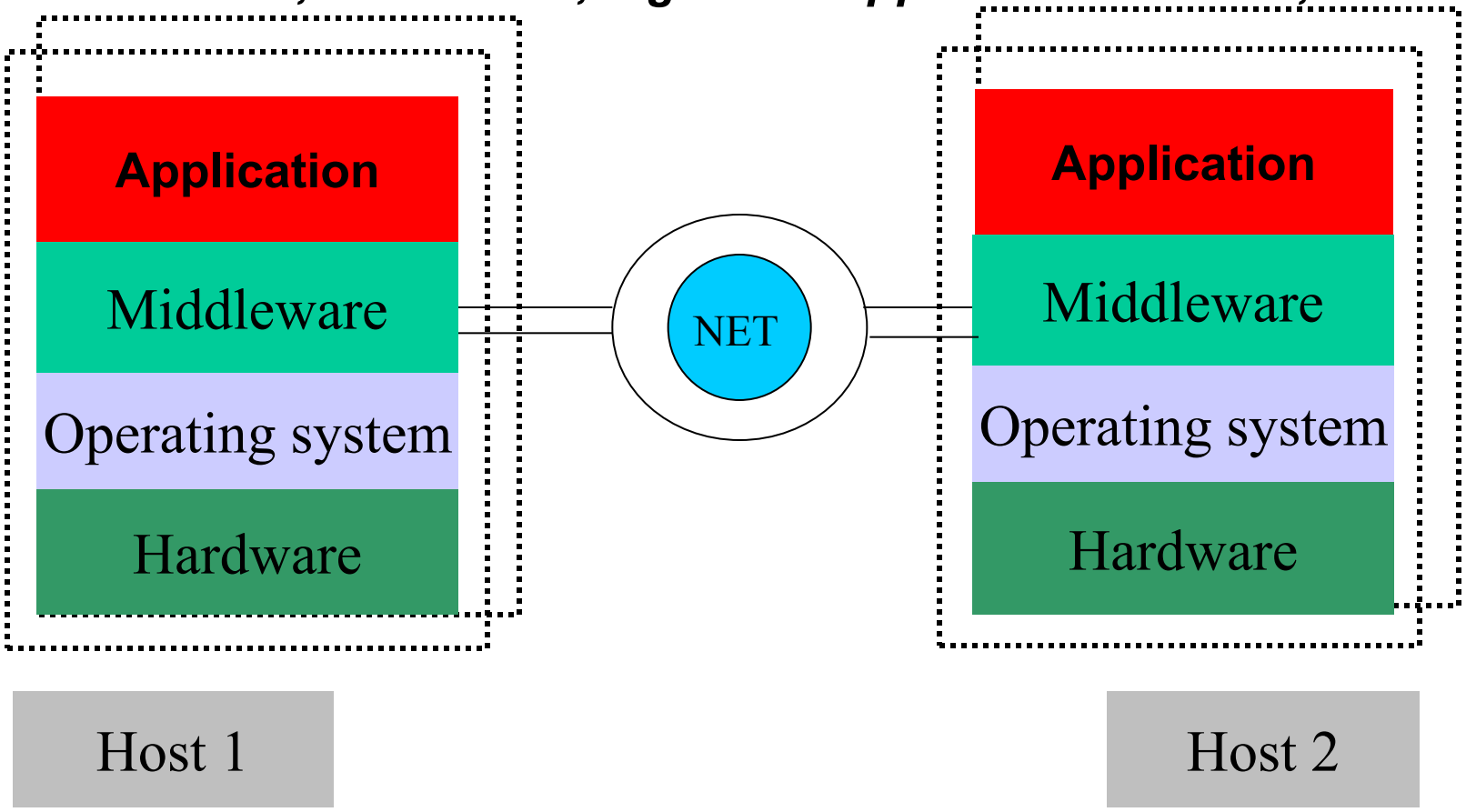
C. What is a Middleware ?



A middleware is dispersed among many disciplines

Middleware architecture

The middleware can be located anywhere:
sensor nodes, sink nodes, high level application terminal, etc.



Why a Middleware ?

1. Masking heterogeneity
   - networks, end-systems, OSs, programming languages
2. Providing a useful distributed programming model
3. Providing Generic services
4. Transparency

Why a middleware ?

It is needed by a bunch of applications and host systems

- applications
  - eCommerce,
  - real-time, embedded
  - mobile agent systems
  - peer to peer platforms
  - mobile computing applications
- underlying host systems
  - PCs/ workstations
  - wireless PDAs
  - embedded devices
  - network processors
  - wireless, sensor, infrared etc. networks

Middleware functionalities

- Complete middleware solutions require at 4 components:
  - **Programming abstraction**: this defines the interfaces to the middleware for the application programmer.
  - **System services**: implemented functions to enable programming abstractions.
  - **Run time support:** extensions to the embedded operating system to support middleware services.
  - **QoS mechanisms:** they define the QoS constraints of the system.

Middleware functionalities

Programming abstractions. Programming abstractions involve

- The way the programmer views the system referred to as the ***abstraction level.***
- The model used to program the application known as the ***programming paradigm.***
- The ***programming interface*** which defines the mechanism used by the application to access the middleware functions.

Middleware functionalities

Abstraction level.

- **A global abstraction level** (e.g. system level abstraction)
  - abstracts the WSN as a single virtual system
  - a single centralized program is expressed as a set of subprograms that can be executed on local nodes.
  - allows the programmer to concentrate on the application, while avoiding the details of the nodes but can be less efficient in terms of WSN resource consumption.

Middleware functionalities

Abstraction level.

- **A local abstraction level** (e.g. node level abstraction):
  - the system is presented as a collection of distributed nodes where each node can be reached individually.
  - leads to efficient communications between nodes and flexibility but requires from the programmer a certain knowledge of the sensor node.

.

Middleware functionalities

Programming paradigms. Depending on paradigm, a WSN can be seen as

- A database where sensor data is the information stored in the database
- A publish/subscribe service where data from sensors generate an event for the application.
- A set of mobile agents copying with the mobility of the WSNs
- A rule-based declarative language for command execution.
- A set of virtual machines used to support runtime.
- A graphical user interface that includes a query-builder and result display.

Middleware functionalities

Interface type

- The way the application relates with the middleware can either be:
  - **Declarative**, providing data about which information is required. This may be based on Structured Query Language (SQL) languages or XML, and is used for database programming paradigms.
  - **Imperative**, offering commands that should be carried out. This is used for publish/subscribe programming paradigms.

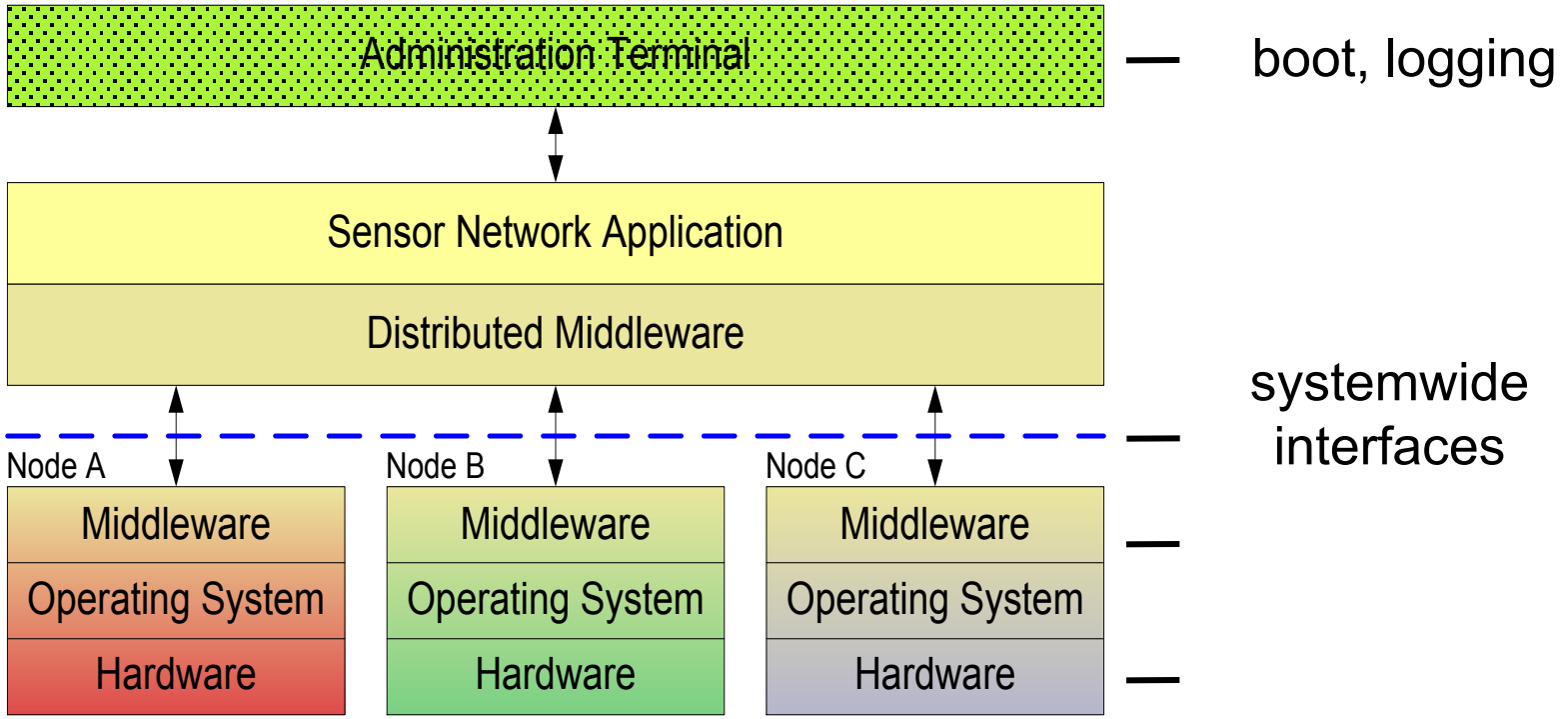
Middleware design space

1. Abstraction Support
   - Large number of heterogeneous sensors
2. Data Fusion
   - Various ways to collect data
   - Efficient way of communicating this data
3. Resource Constraints (lightweight)
   - Low energy, small memory footprint
   - Requires a lightweight design (as opposed to traditional middleware such as CORBA, etc.)
4. Dynamic Topology, Environment, Application
   - Node mobility, node failure, communication failure

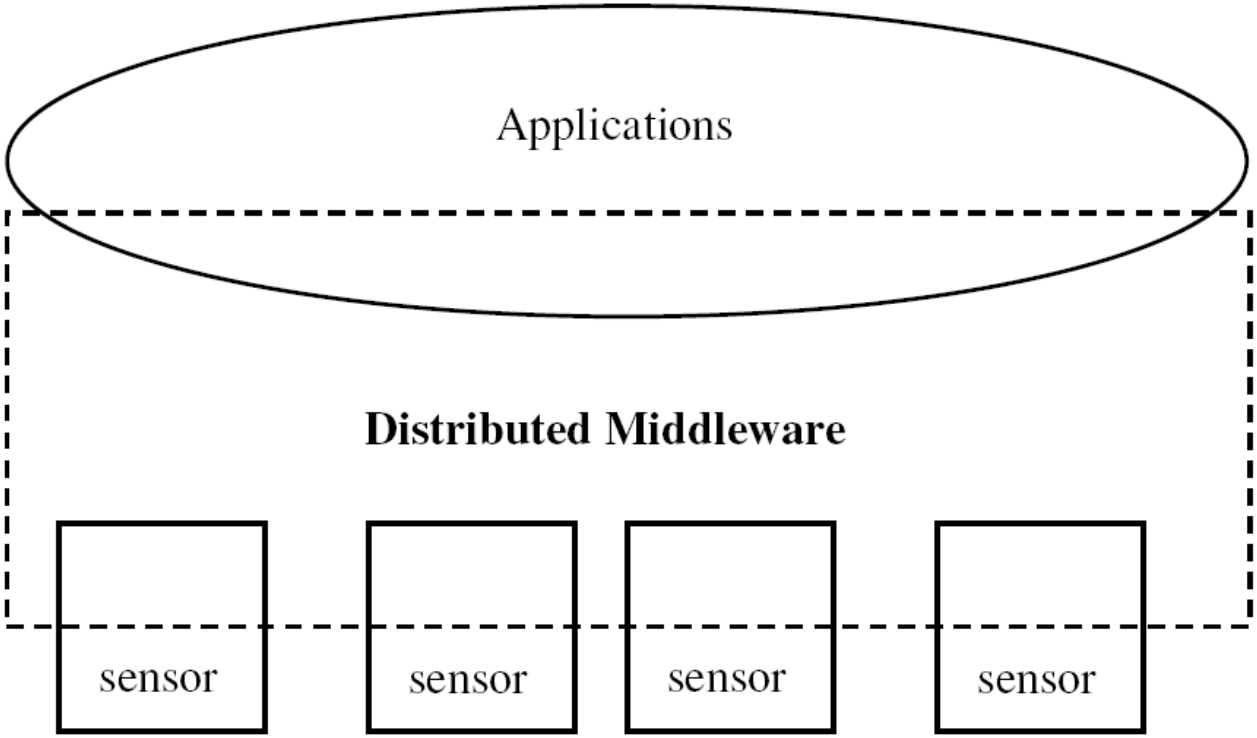
Middleware design space

5. Application Knowledge
   - Try to generalize despite the limited resources
   - Integrate application knowledge into the services provided
6. Programming Paradigm
7. Adaptability
8. Scalability
9. Security
10. QoS Support

Middleware architecture



Middleware architecture



Shared functionalities/features between applications and distributed middleware depending on abstraction.

Middleware Approaches

1. Virtual Machine-based
   - contains VM, interpreter,
   - Example: Mate, Magnet

2. Database-based
   - Virtual Database
   - easy to use interface
   - Examples: TinyDB, Cougar, SINA, Dsware

3. Modular programming
   - decomposition of application
   - Examples: Impala, Agilla

Middleware Approaches

4. Application Driven
   - tune network on the basis of Requirement
   - Example: MiLAN
5. Message oriented
   - use publish-subscribe mechanism
   - Example: Mire
6. Event oriented
   - Request-reply,
   - Example: Garnet
7. Service oriented
   - Reflective, Flexible, Service-centric, Service oriented

Middleware Approaches

	Power awareness	Openness	Scalability	Mobility	Heterogeneity	Ease of use
VM: Mate	Full	Full	Full	Full	Partial	Little or none
Data Base: Magnet	Full	Full	Full	Full	Partial	Full
Data Base: Cougar	Partial	Little or None	Little or None	Little or None	Little or None	Full
Data Base: SINA	Full	Little or None	Little or None	Little or None	Little or None	Full
Data Base: DSWare	Full	Partial	Partial	Little or None	Little or None	Full
Modular: TinyDB	Full	Partial	Partial	Partial	Partial	Full
Application: Impala	Full	Full	Full	Full	Little or None	Full
Message: MILAN	Partial	Full	Full	Little or None	Little or None	Full
Message: Mires	Full	Full	Full	Partial	Partial	Full

Impala: application driven

- Zebranet project consisting of a long term migration study of wildlife using **event based programming model**

CPU, Memory,
Wireless
Transceiver GPS

Data

Peer-to-Peer
communication
Protocol

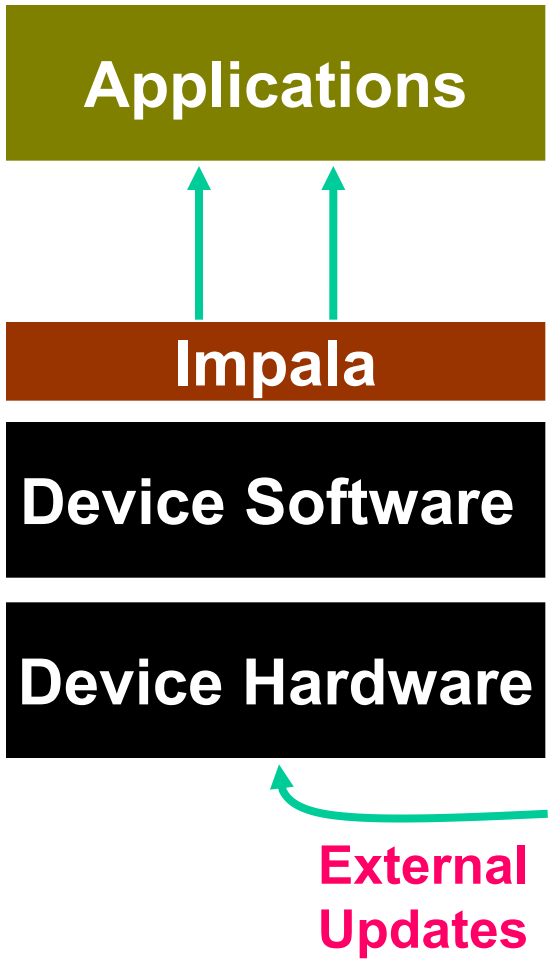
Data

Data

Base Station
(car or plane)

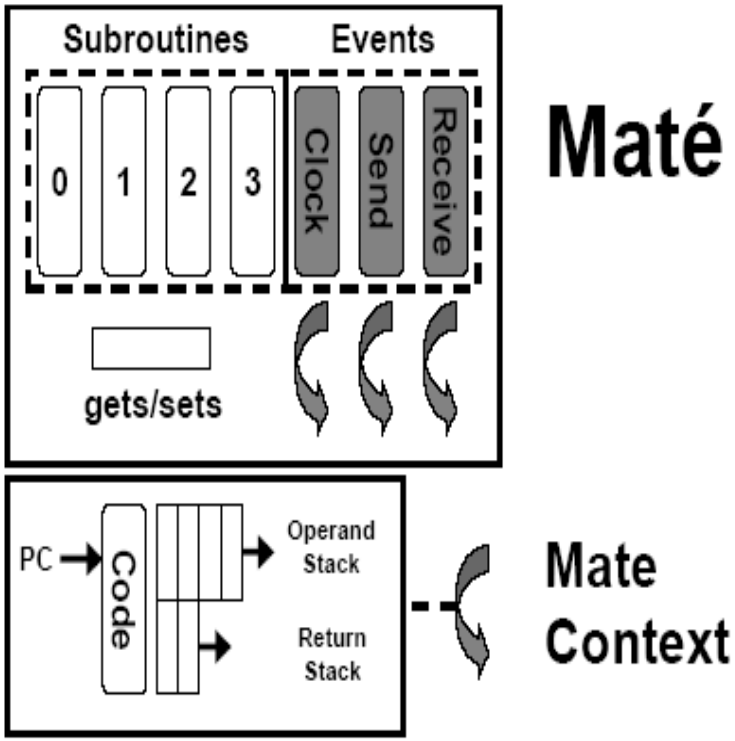
Impala Features

- Advantages
  - Code modularity
  - **application adaptability and**
  - **update**
  - Fault tolerance
  - energy efficiency
  - long deployment time
- Disadvantages
  - Not data fusion
  - not heterogeneity



Mate

- Virtual Machine on top of TinyOs
- Stack-based architecture
- Break down the program into small self-replication capsules(24 instructions)
- Capsules are self-forwarding and self-propagation



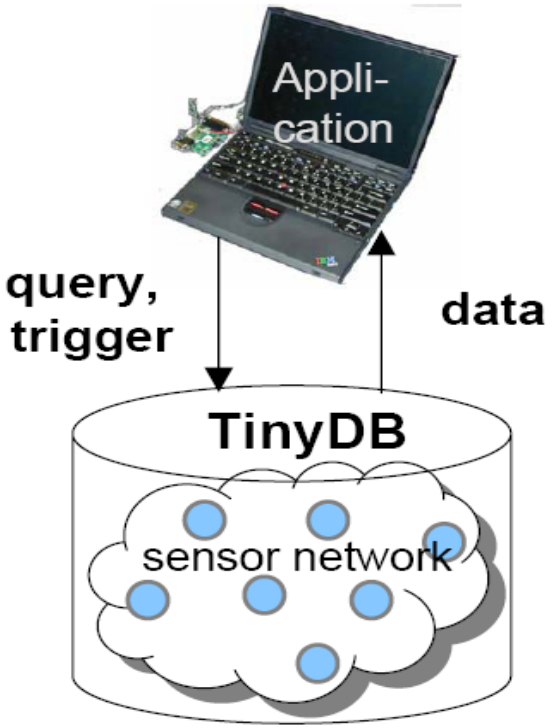
Mate Features

- Advantages
  - Simple programming model
  - Small and concise model
  - High security
- Disadvantages
  - high energy consumption
  - Not flexible to support wide range of applications

TinyDB

- query processing

```
SELECT nodeid, light
FROM sensors
WHERE light > 400
SAMPLE PERIOD 1s
```

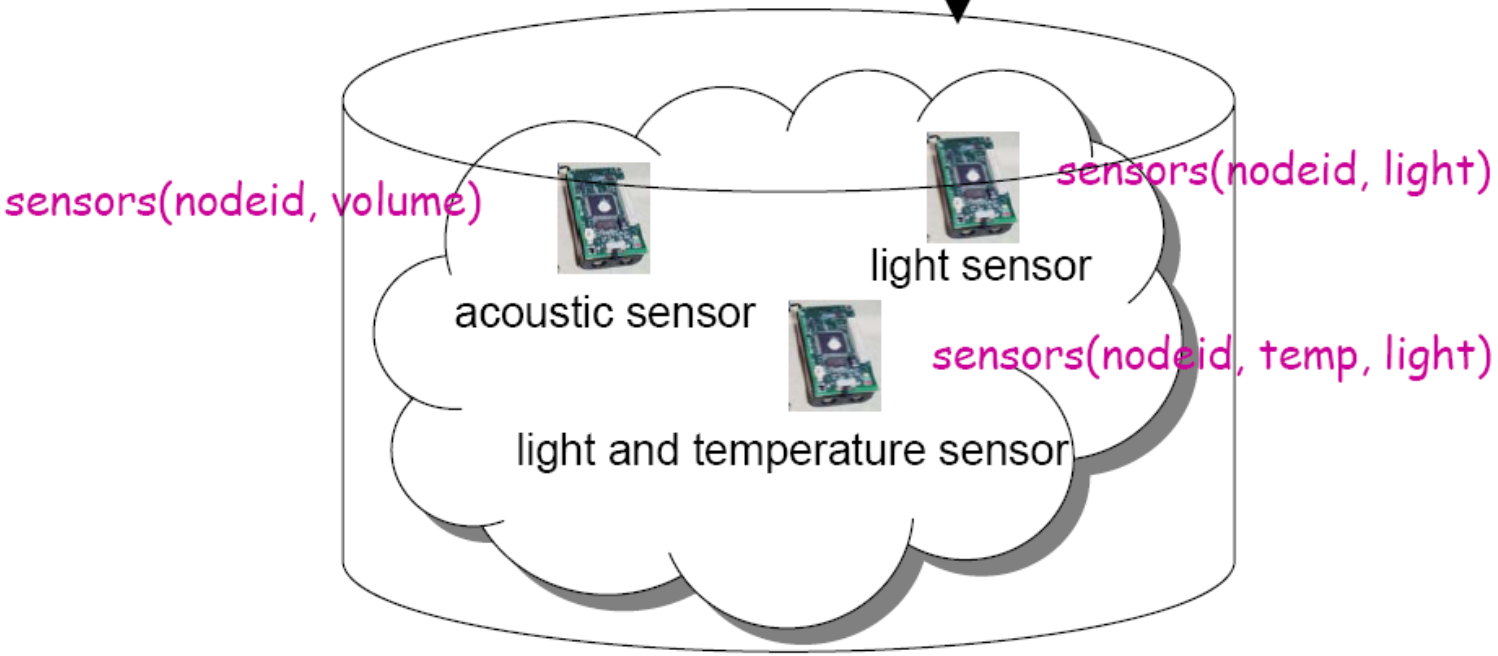


TinyDB

Sensor table

sensors(epoch, nodeid, volume, light, temp, ...)

```
SELECT nodeid, light
FROM sensors
WHERE light > 400
SAMPLE PERIOD 1s
```



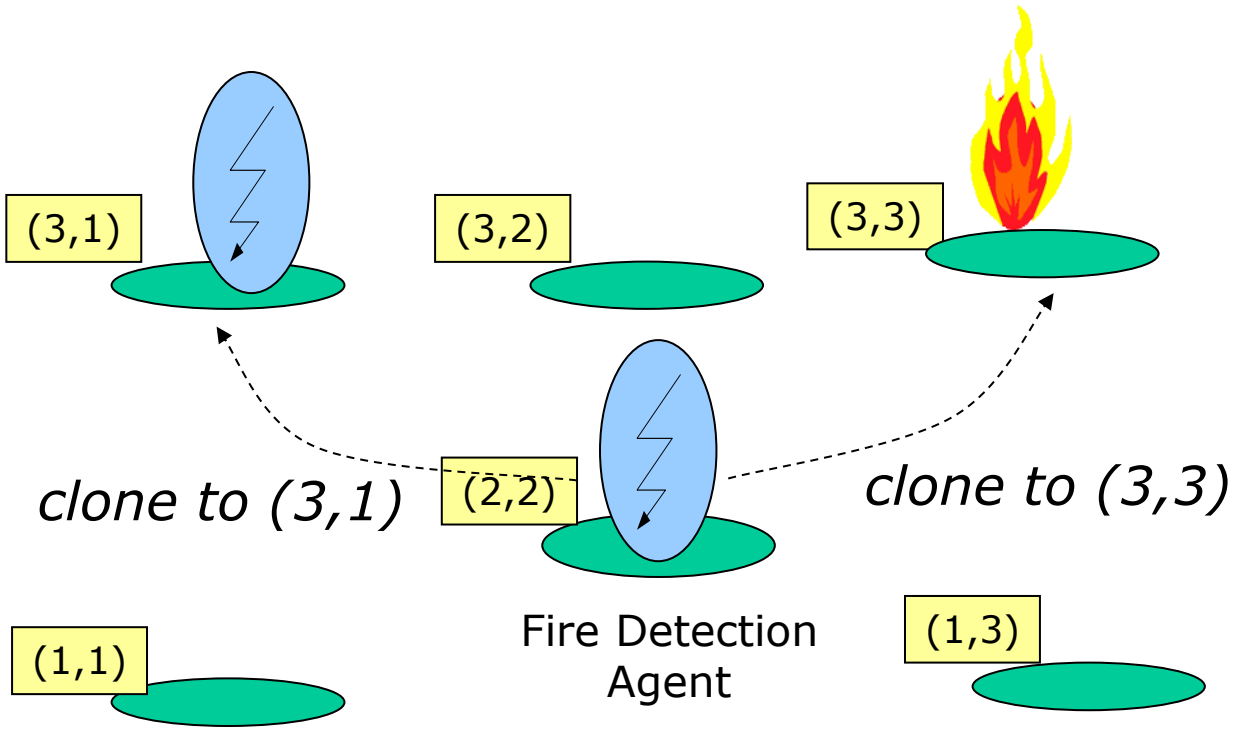
TinyDB Features

- Advantages
  - reduce number of messages
  - Nice abstraction model
  - good aggregation model

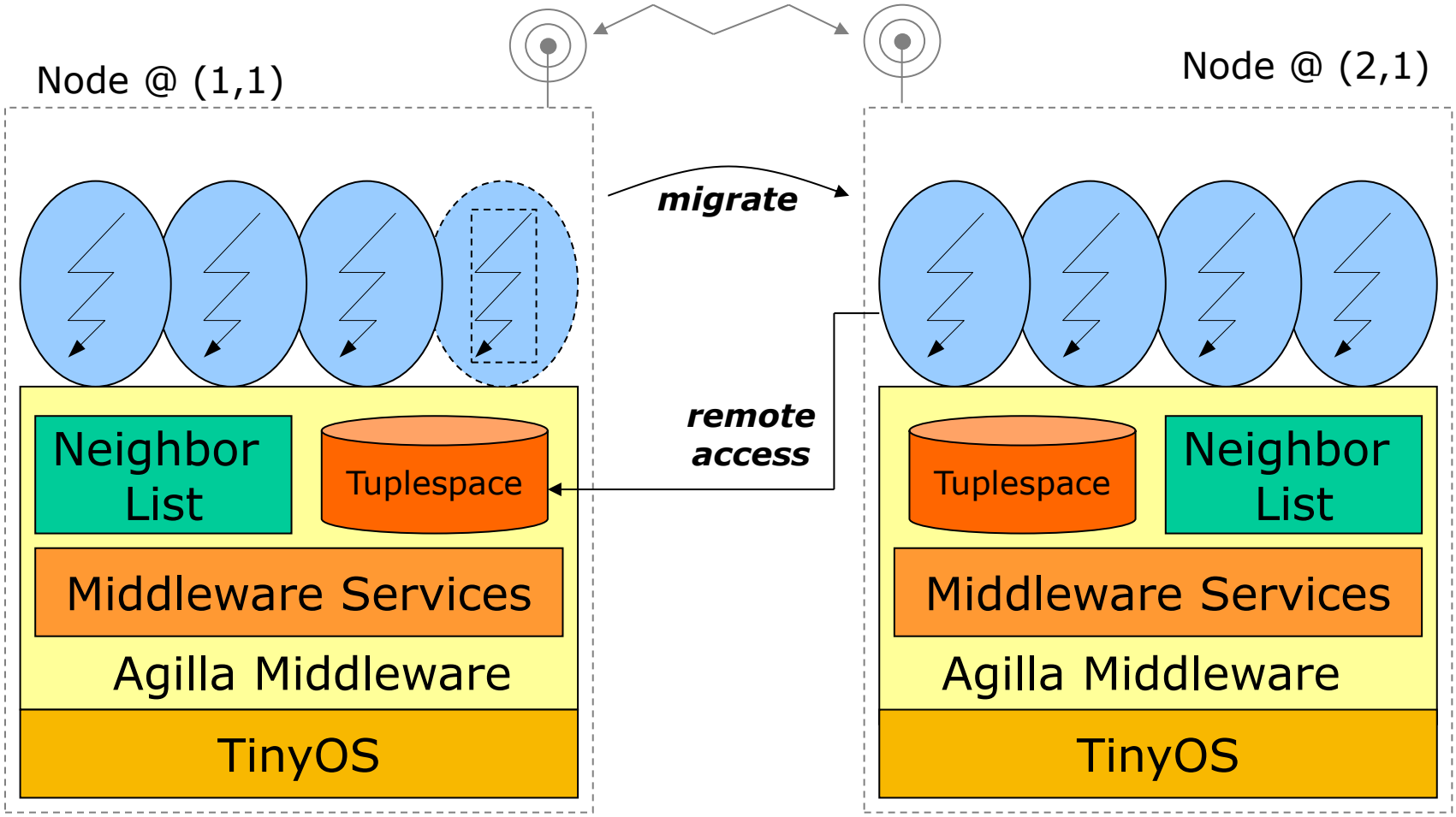
- Disadvantages
  - Not much functionality

Agilla

- Based on mobile agents and a Stack-based architecture
- Allows moving agents from one node to another
- Use two instructions to move (clone, move)
- A maximum of 4 agents can run on a node



Agilla Architecture



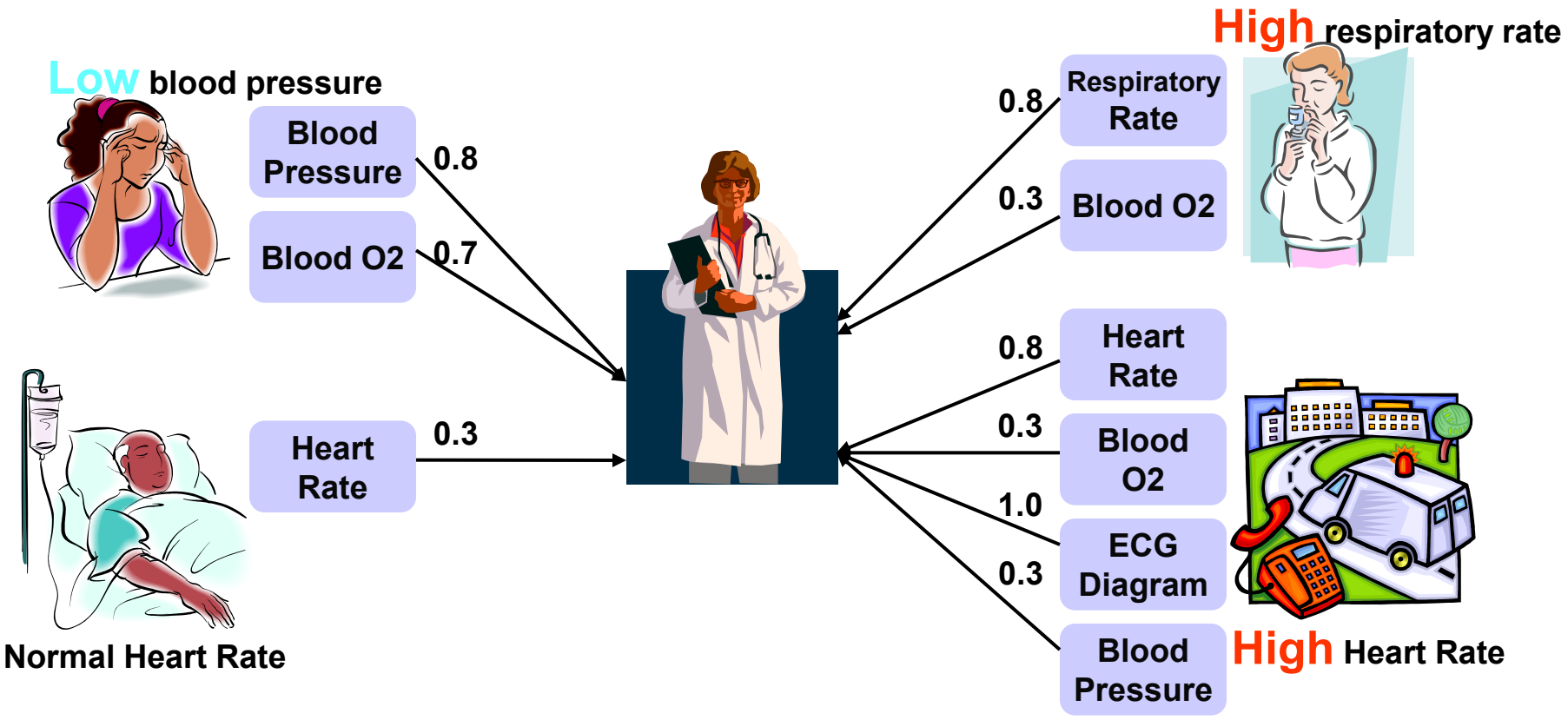
Agilla Features

- low energy consumption
- Increase flexibility
- reduce code size

- low security
- difficult to read and maintain programs

MiLAN

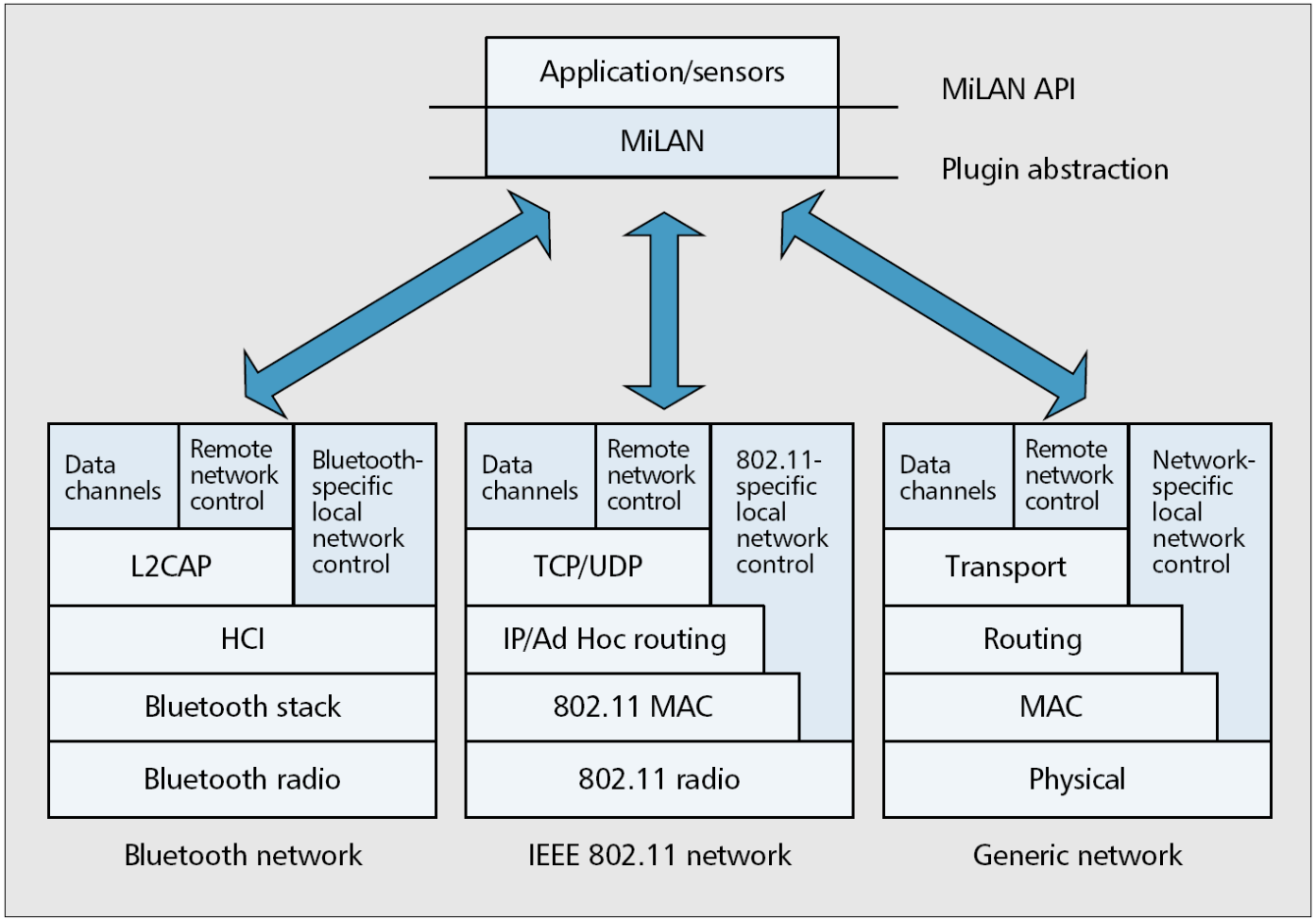
- MiLAN is designed for personal health monitor application



MiLAN

- Goals:
  - application lifetime maximization
  - application QoS support
- MiLAN receives a description of application requirements through specialized graphs
- MiLAN sits on top of multiple physical networks: bluetooth, 802.15.4, wifi, etc.
- convert commands to protocol-specific commands that are passed through the usual network protocol stack

MiLAN Architecture



Conclusion

- Methods and tools to analyse and evaluate IoT middleware are still needed
- There is still a long way to go from cloud-like middleware systems such as Pachube and middlewares that are needed for real time WSN applications as the former lack many of the features/functionalities needed for IoT middlewares.
- QoS functionalities can be incorporated into the middleware in terms of dependability, reliability, and scalability, and Security.

Thanks

Antoine Bagula

bagula@cs.uct.ac.za

lsat.cs.uct.ac.za