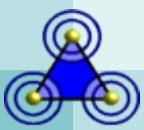
Resource-Aware Service Architecture for Collaboration of Sensor Nodes

Jan Blumenthal, Dirk Timmermann
University of Rostock

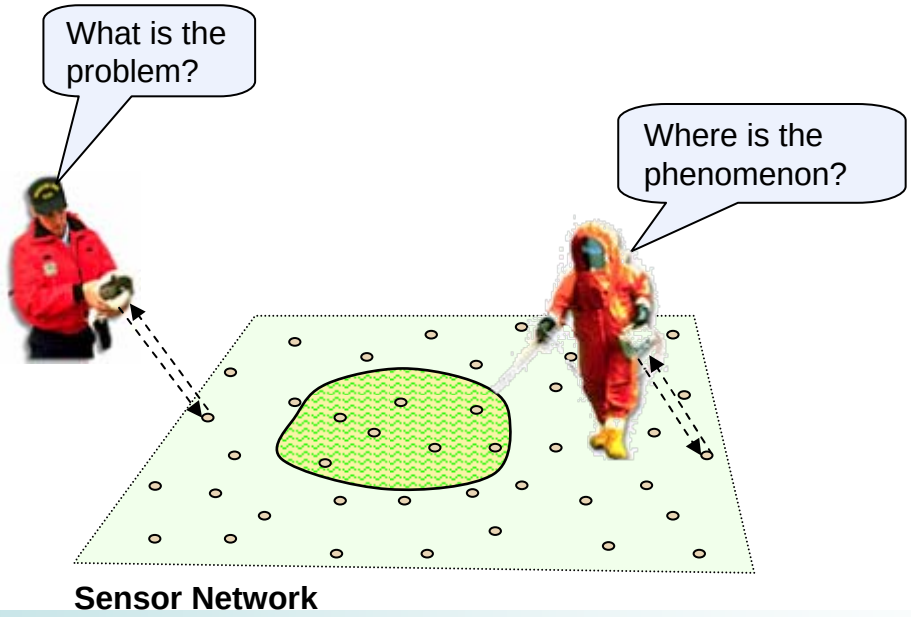
DCOSS, San Francisco
2006/06/18



Software for Sensor Networks

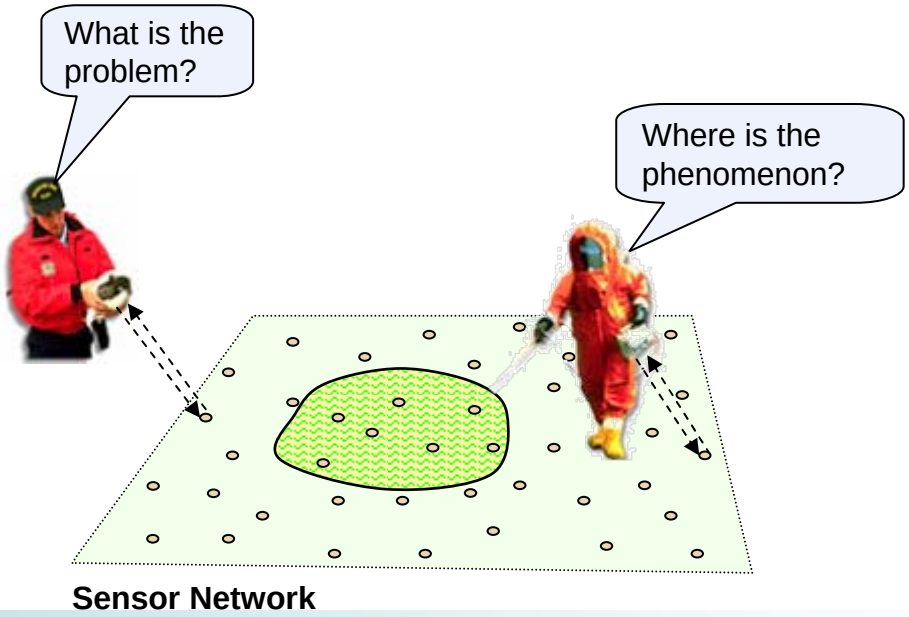
Required Features:

- Adaptable to new requests
- Extracting implicit information (group dimensions)
- Collaboration of several sensor nodes
  - Location dependent services
  - Detection of dynamical phenomenons

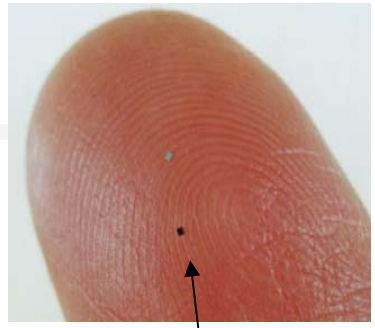


Implementation Requirements

- Applicable for resource-critical devices
- Extremely low memory usage (code and data)
- Adaptable to different memory architectures
- On-the-fly update of software parts
- Reusing existing software
- Services run autonomously
- Low communication effort and low energy consumption



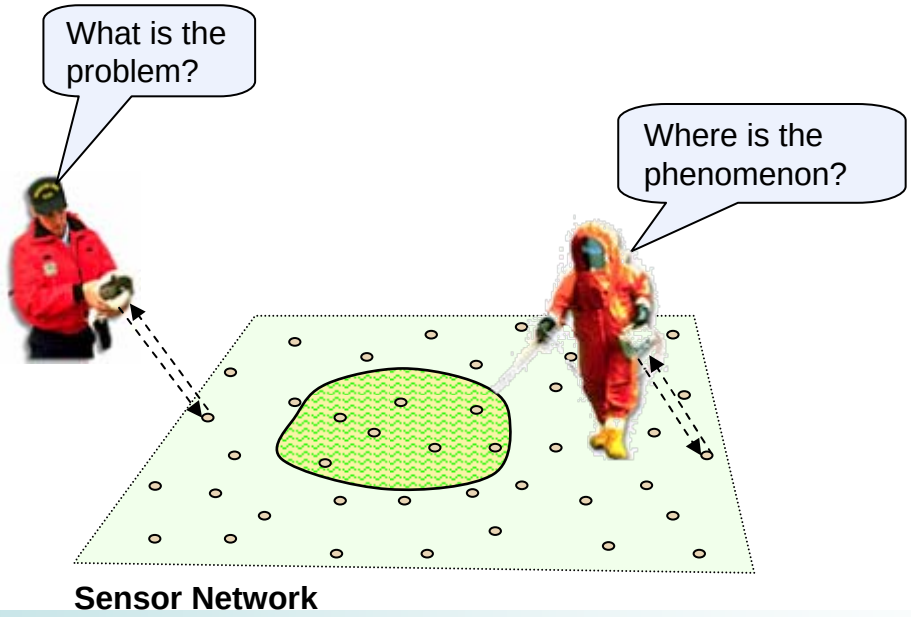
Assumptions



Coming up sensor nodes

Properties in Sensor Networks

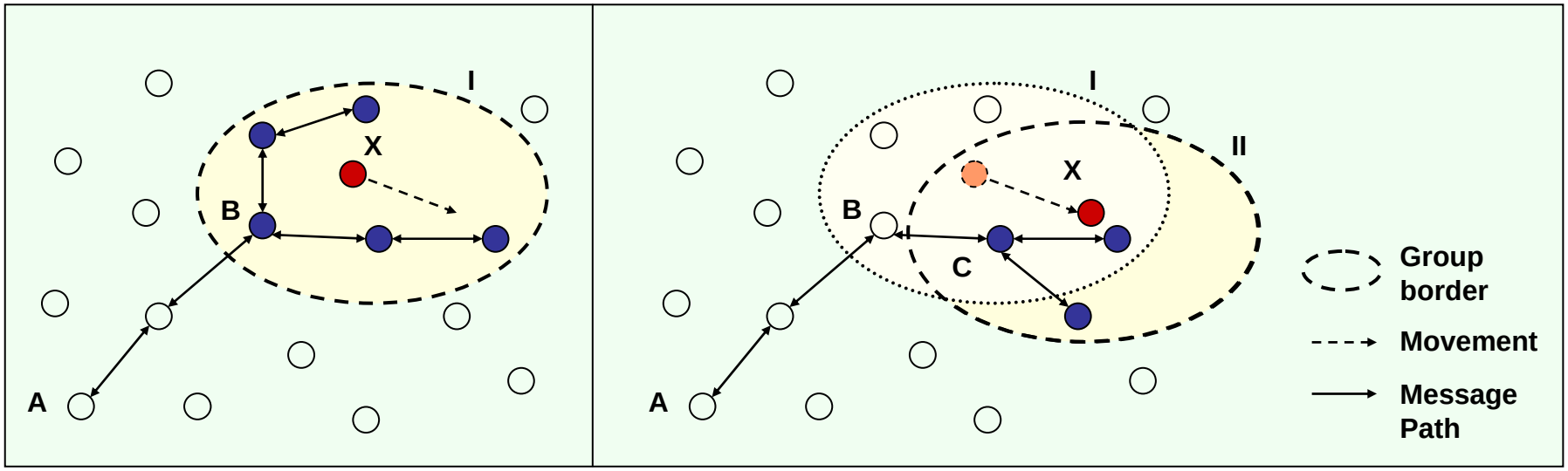
- Cheap homogenous sensor nodes (mass production)
- Small, simple and suitable tasks focussed on measurement and pre-calculation
- Hardware-dependent programming required
- Unsafe communication
- Failure of nodes over time



Types of Collaboration

a) Sensor nodes **build groups** collaboratively based on an object definition

b) **Groups move** with the observed object and may travel around

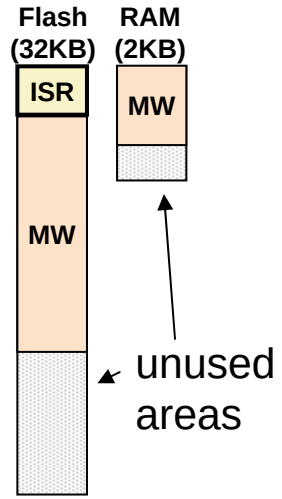


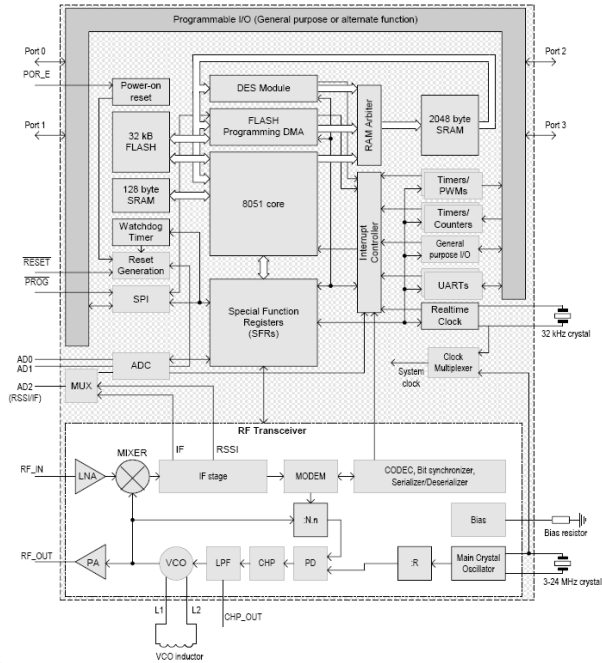
a) Enclose objects (areas)
Where is the temperature > 20°?

b) Observation of mobile objects
What is the average speed of formed object (temperature group)?

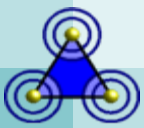
Available Software Architectures

- Software updates mostly stored in RAM
  - not reset-safe
  - only small RAM available (2KB) and usually occupied by operating system
  - data memory needs a lot of energy
- Software is not resource-aware
  - Interfaces between software parts require overhead for adaption and HAL
  - Virtual Code needs Virtual Machine (VM)
- Software (e.g. additional Services) has not full control
  - Specifics of the microcontroller not suitable configurable (Timers, Interrupts, Ports)
  - Energy-saving mechanisms not applicable
- Mobility support is missing





Service Architecture (RASA)

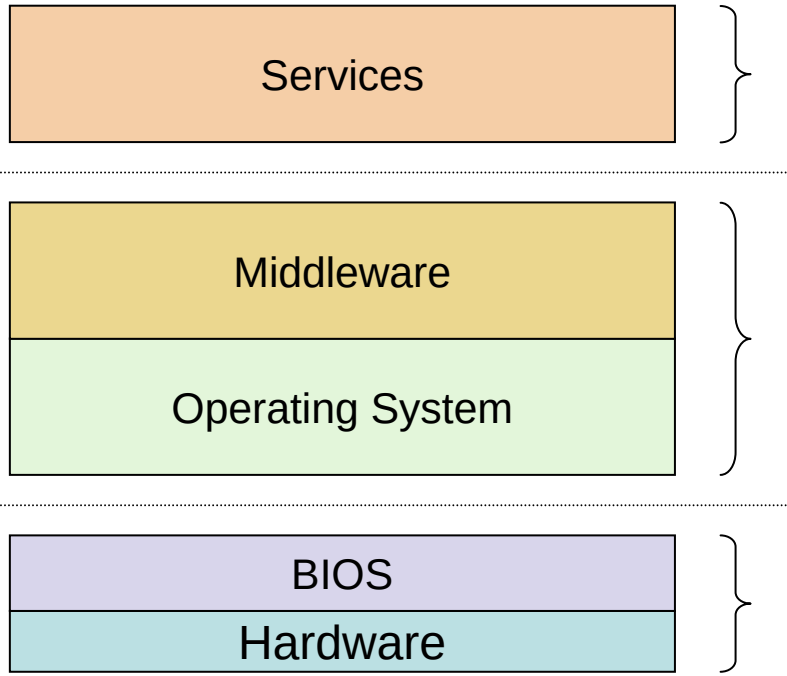


Layered Software Model

- Object and Group Definitions
- Group Management
- Communication Behavior

- Service Management
- Memory Management
- Interrupt Handling

- Startup
- System-Update

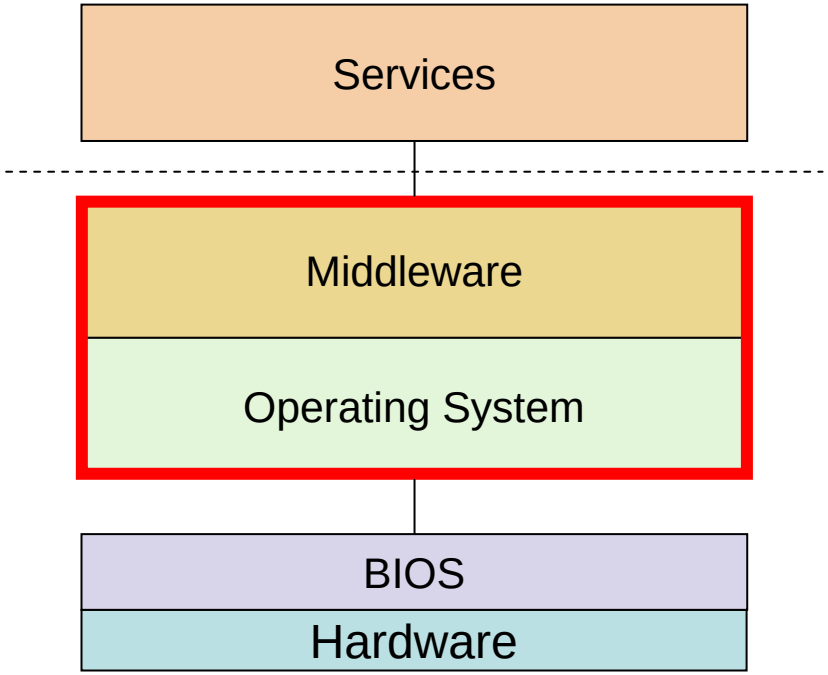


3 Layer Structure

- Minimizes interfaces
- Simplifies development of software
- System layer and service layer are exchangeable at runtime

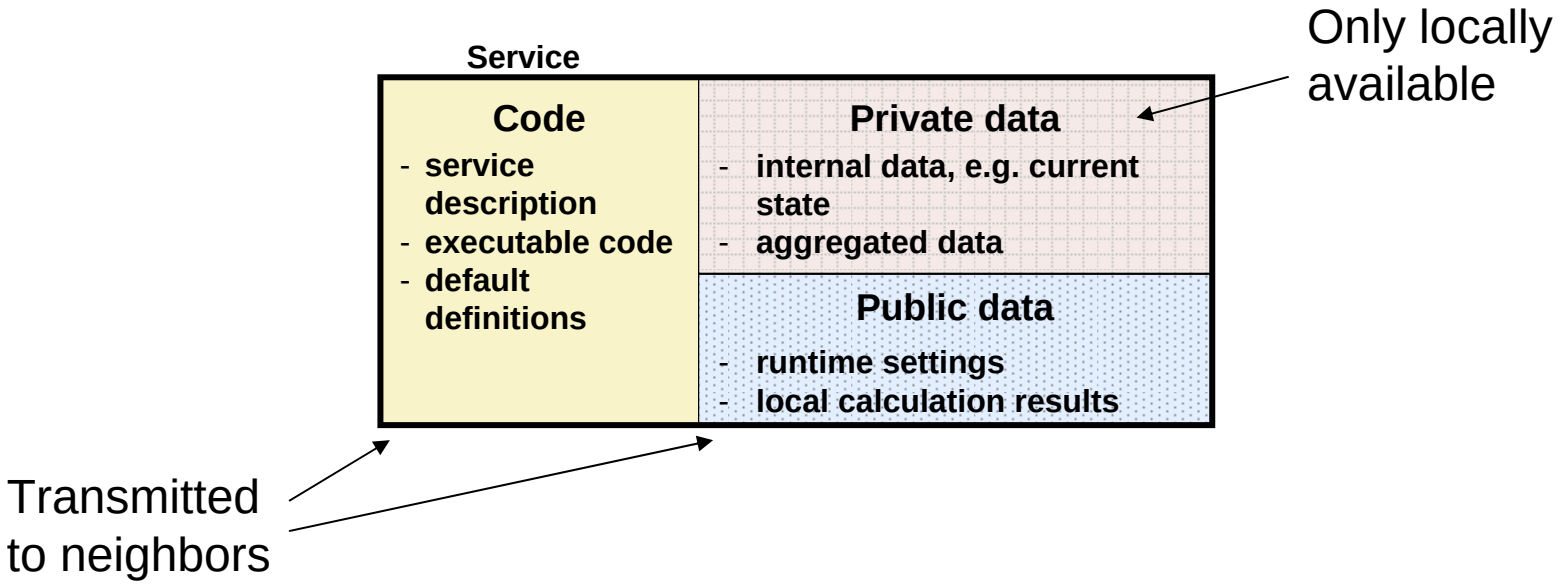
System Layer

- contains functionalities of
  - operating system (transmission, interrupt handling)
  - lightweighted middleware (service management)
- not comparable with well-known PC pendants (CORBA)

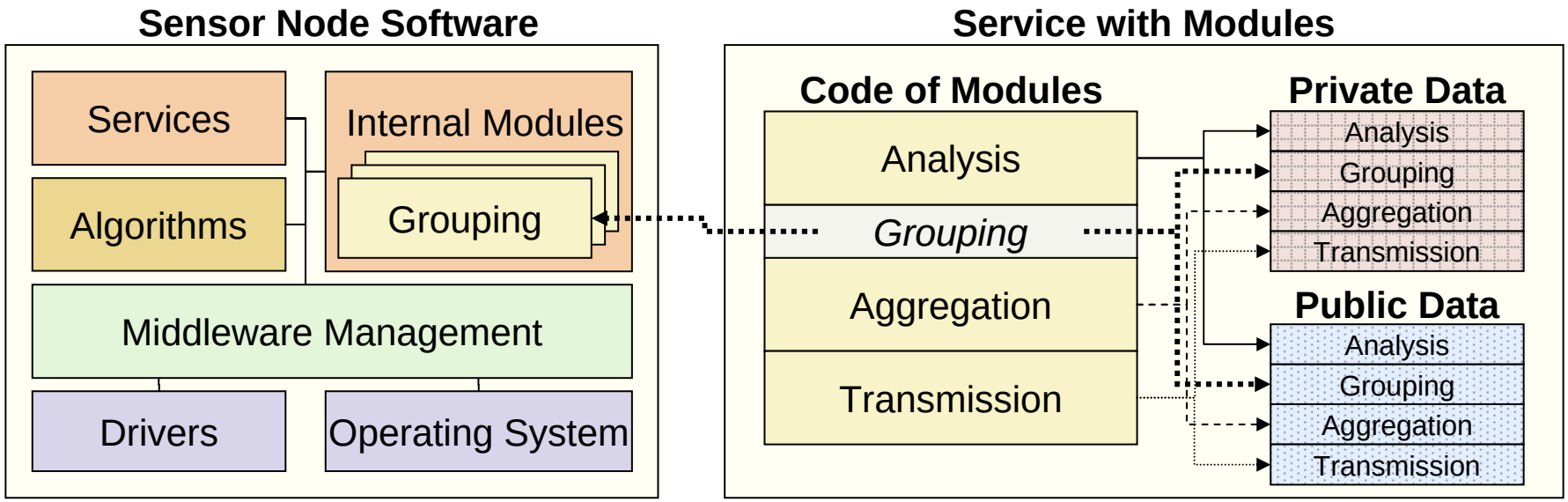


Service

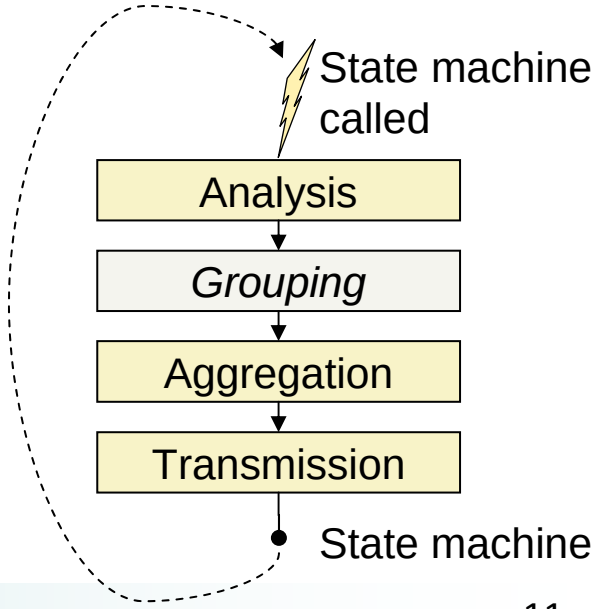
- Implementation of services as mobile code (native or virtual)
- Preferable mobile native Code
  - + very fast and high flexible
  - + can be optimized by compilers
  - + platform independent at source code level (ANSI-C)
- Service is running autonomously
- Decoupling middleware and services due to a simple interface
- Three parts:



Modules

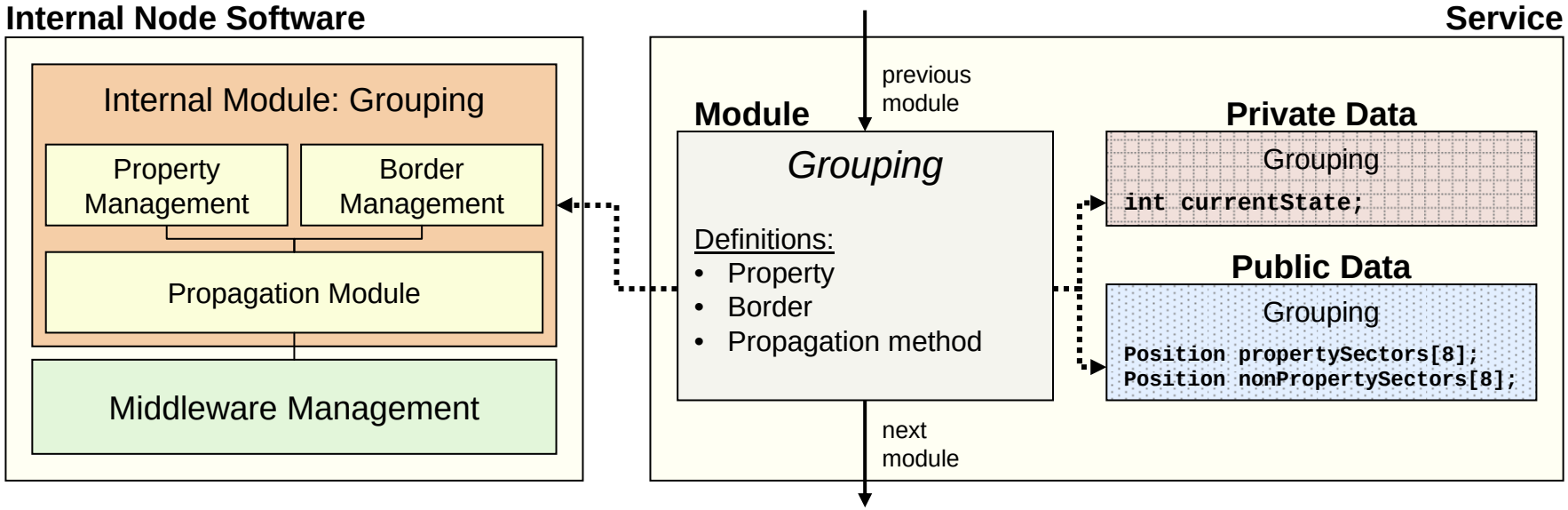


- Splitting a service into several modules
- This increases
  - interoperability of source code
  - modularity
- Modules are executed
  - in sequence
  - any order, if specified



Internal Modules

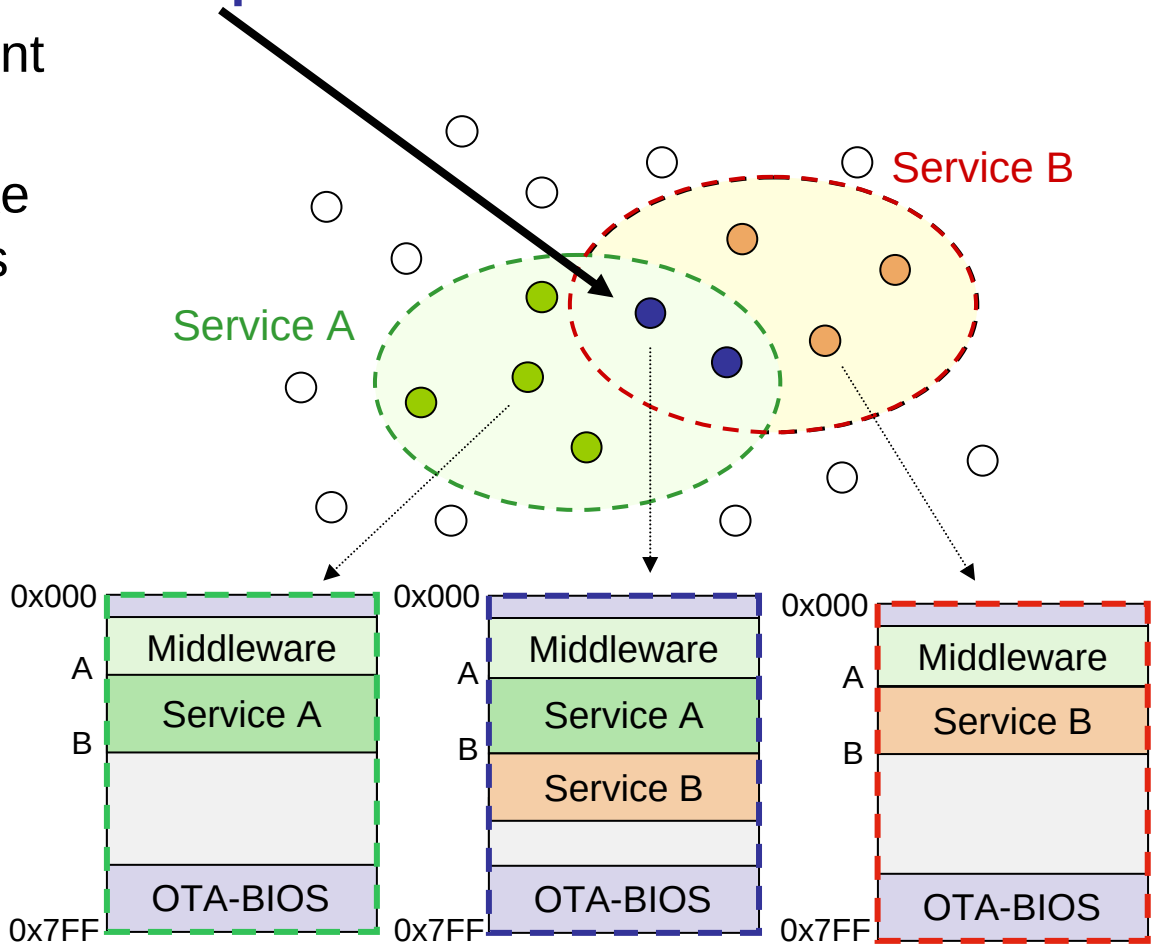
- *Outsourcing* of most used modules parts into node's middleware, e.g. group building
- Internal module description:
  - is abstract
  - defines a link to an already existing internal module
  - data memory mapped to service memory



Multiple Services

Runtime behavior:

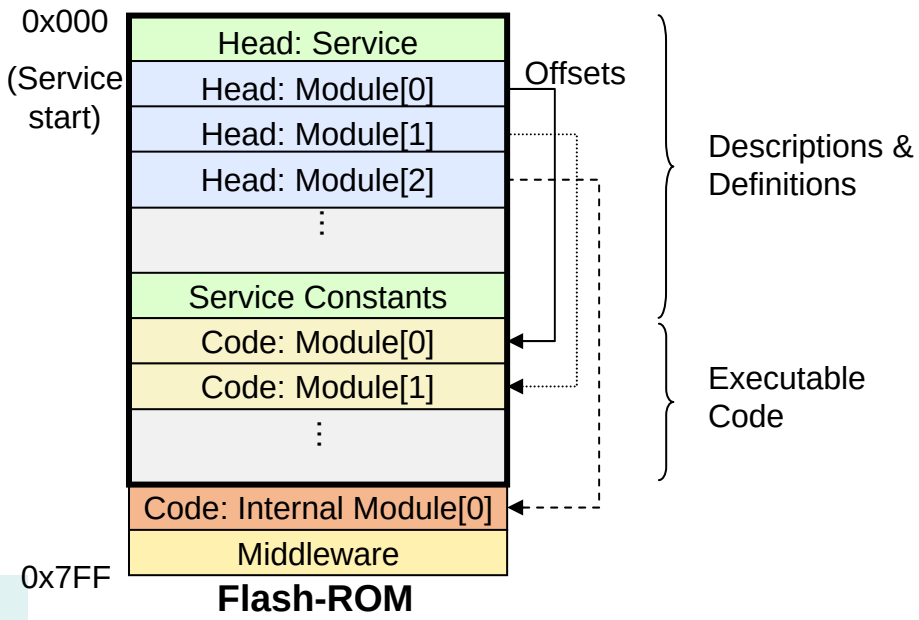
- Sensor networks may contain **multiple** services
- Service injection at different sensor nodes
- Delays in forwarding evoke dynamical start addresses
  - Services must be **relocatable**



Service Definition

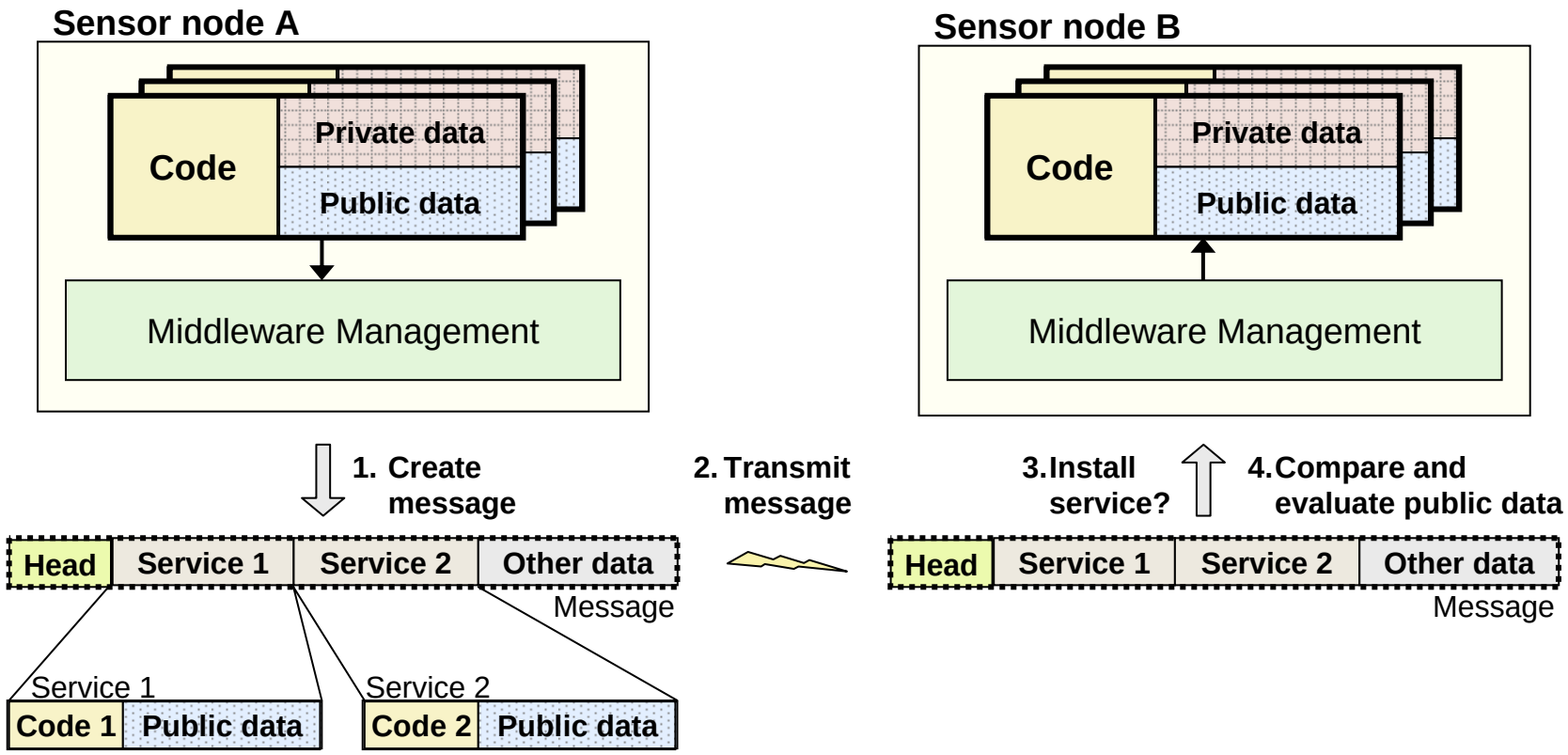
```
typedef struct Service
{  int    serviceid;
   char   flags;
   char   reserved;
   int    remainingCodeSize;
   int    privateDataSize;
   int    publicDataSize;
   int    numberOfModules;
   Module modules[0];
} Service;
```

```
typedef struct Module
{  char                flags;
   char                reserved;
   int                 privateDataOffset;
   int                 publicDataOffset;
   ModuleInit          init;
   ModuleHandleData    handleData;
   ModuleEnd           end;
} Module;
```



- References are relative to service start
- Defragmentation
  - Easy
  - On demand

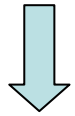
Forwarding Services



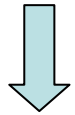
- Service is flashed and installed at runtime once
- Simply, code and public data are transmitted with each service's message
- Service data may combined in one message by middleware

Robustness

Simply, code and public data are transmitted with each service's message.



- Prevents a costly protocol overhead normally required for:
  - installing and forwarding a service
  - if a new sensor node enters the network and requires all neighboring services
- High robustness in sensor network
- Automatically self-healing effect, if software is able to work with changing neighbors (e.g. group building)



Services must be very **small in size** (code as well as public data)

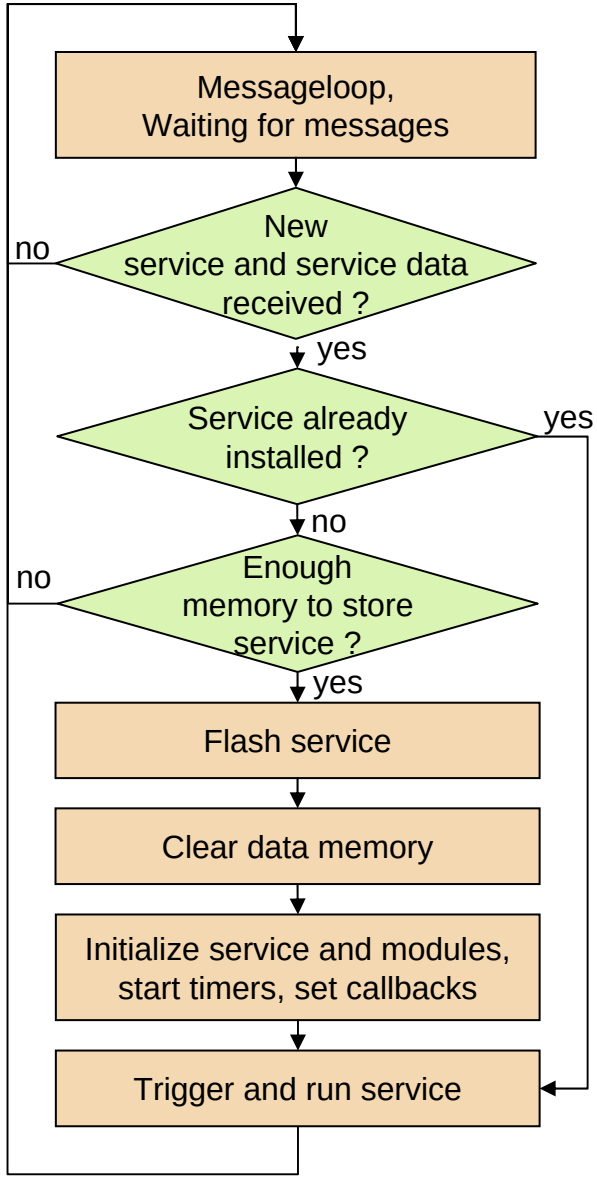
Service Initialization

Instance of local service

- Services run a state machine
- State machine contains private and public data
- State machine
  - may set a timer
  - are called after receive

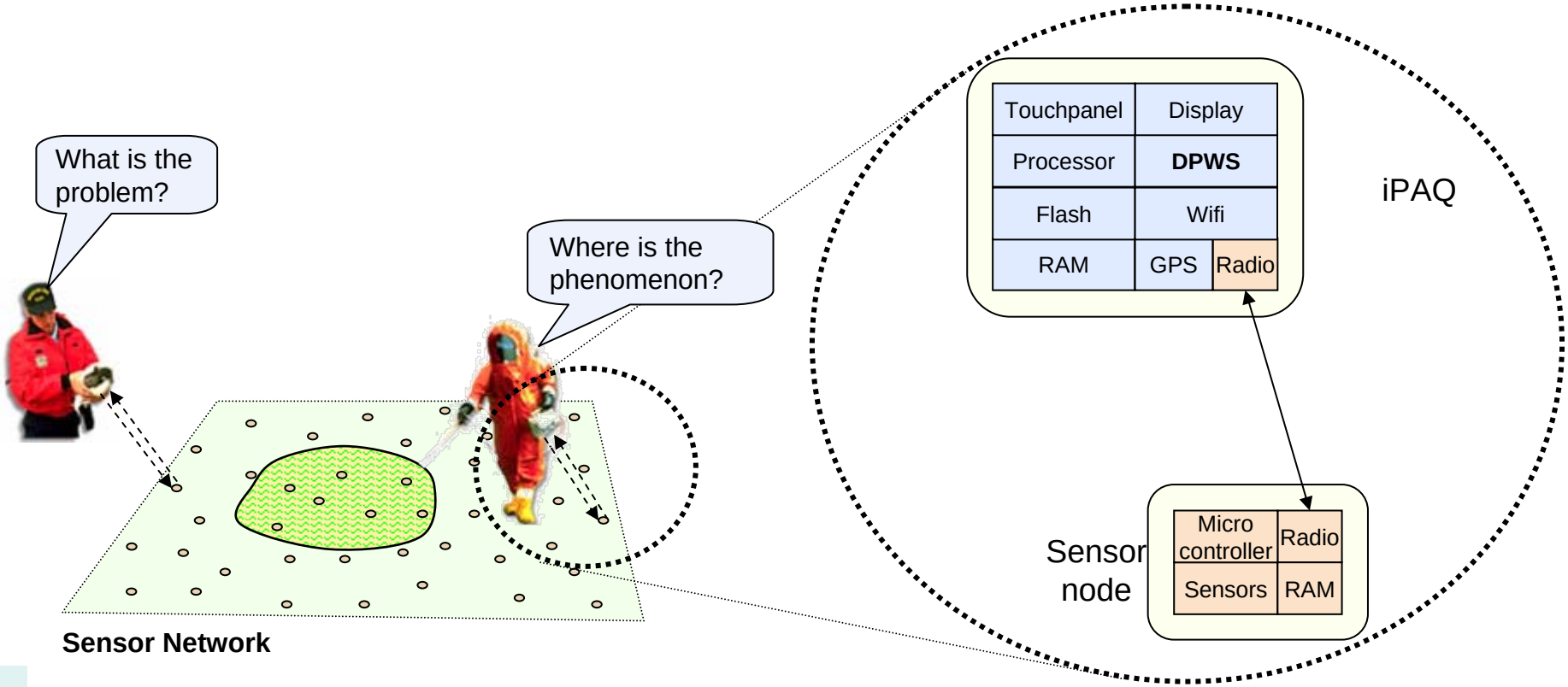
Forwarding services

- Data and code are transmitted to neighboring nodes
- Flashing code, if required
- Updating local data after reception
- Is initialized with received public data of foreign node



Gateway

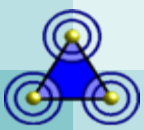
- Connection between sensor network and other networks
- Translates service requests of well-known service architectures
- Injects translated service requests into the sensor network



Comparison of Architectures

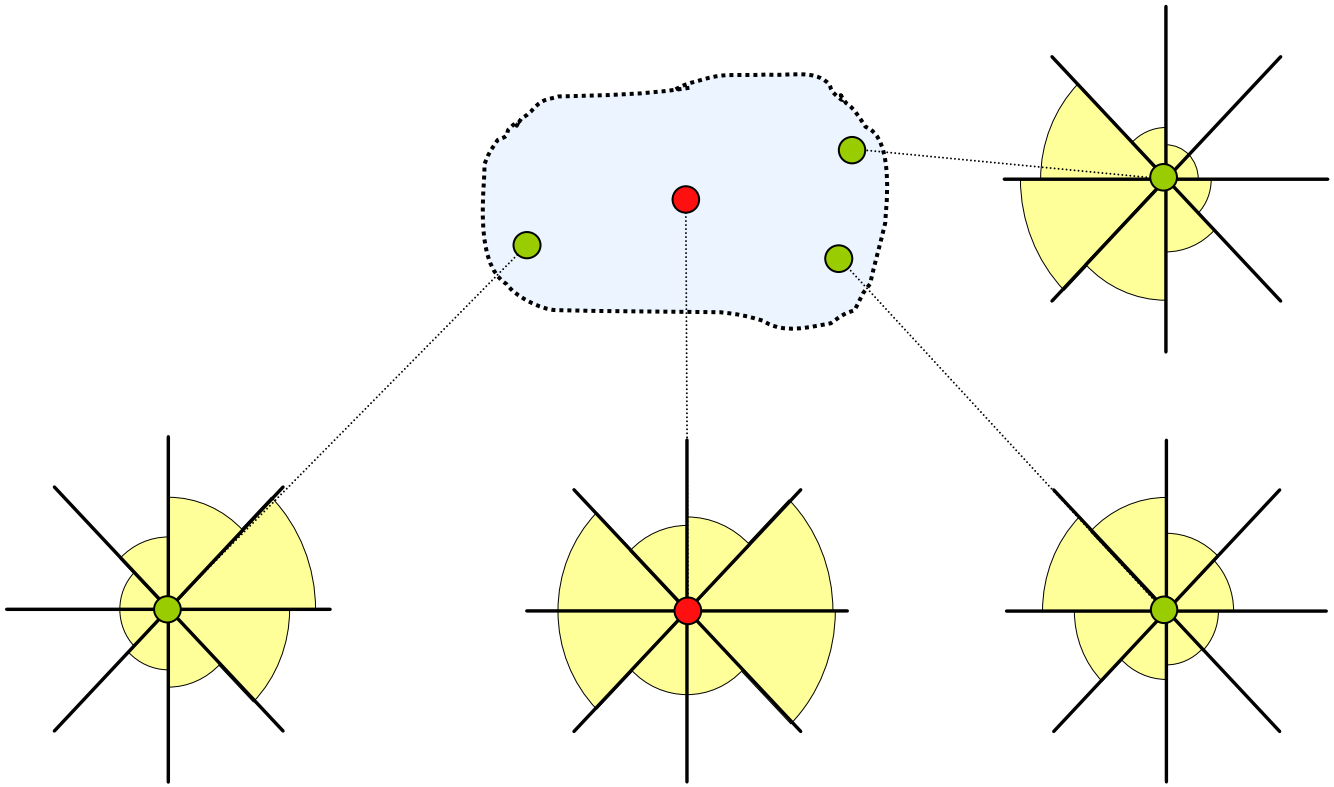
	RASA	MiLAN	Snack	MATÈ	Enviro-Track	Sensor-ware	TinyDB
Dynamical updates	X	X		X	X	X	X
Prevention of dynamical data	X		X				
High-Level language description			X		X		X
Resource-aware optimizations	X	X	X				
Energy and effort analysis		X					
Native code	X		X		X		
Virtual code or request types	In future	Queries		X		TCL	
Data aggregation	X	X	X		X	X	X
Cooperation of sensor nodes	X	X	X		X		

Example Application



Task

- Group sensor nodes with T>20° (condition)
  - Group member, if condition=*true*
  - Non group member, if condition=*false*
- *Grouping: Sector-based group method*



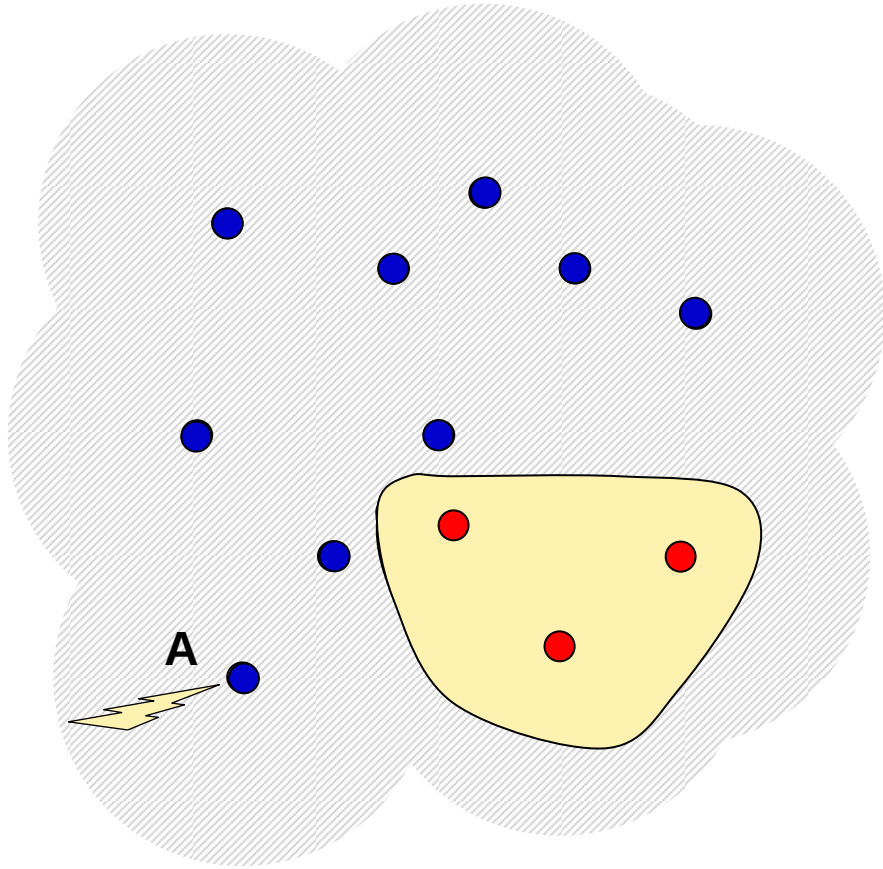
Group Building

Instance of local service

- State machine contains private and public data
- Task: Local view of group borders

Forwarding services

- Data and code are transmitted to neighboring nodes
- Updating local data after reception
- Flashing code, if required
- Dynamical group building even with changing neighbors



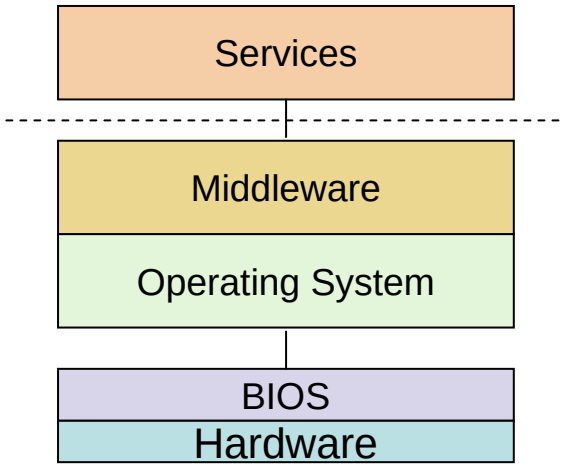
Non group member
Group member
Node without service

Service Area

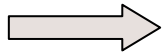
Detected Group

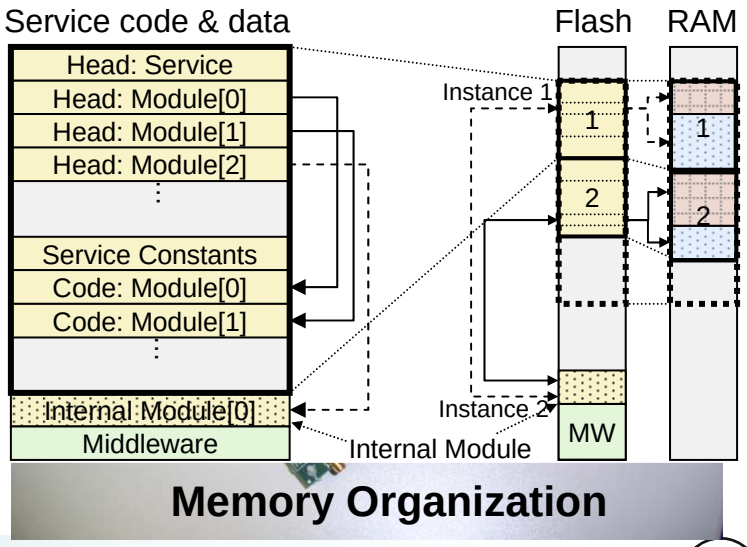
Sensor Node Software

- Startup code of sensor nodes with transmission features
- Base operating system
  - Transmission, reception and forwarding of messages
  - Message checking (CRC)
- Updating system layer by **Over-the-Air-Flashing (OTA)**
- Support of **mobile services**



Software Structure



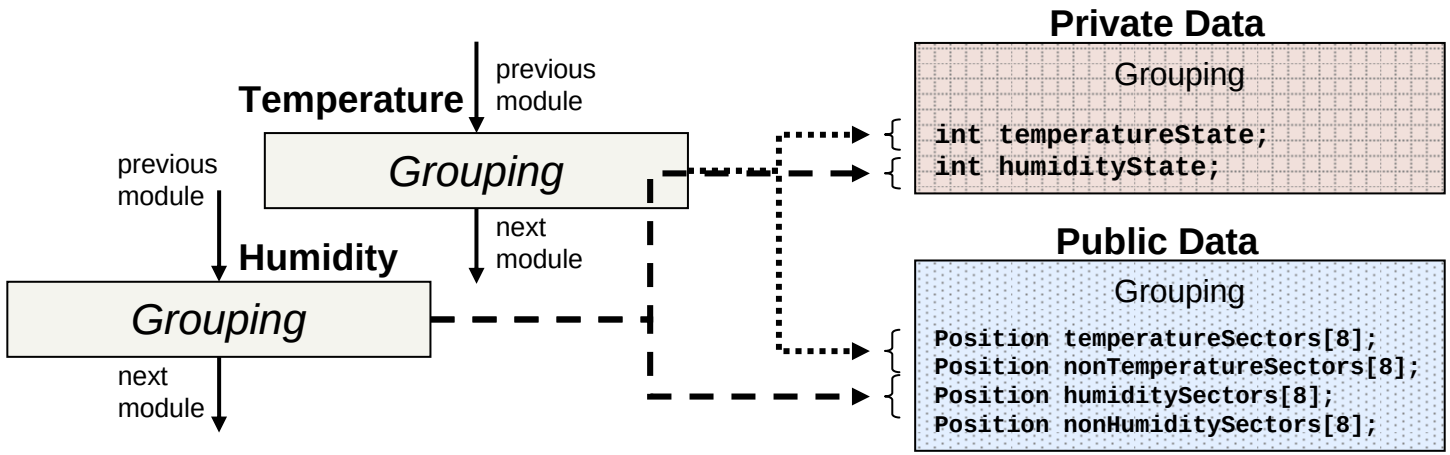


Memory Organization

Test Scenarios

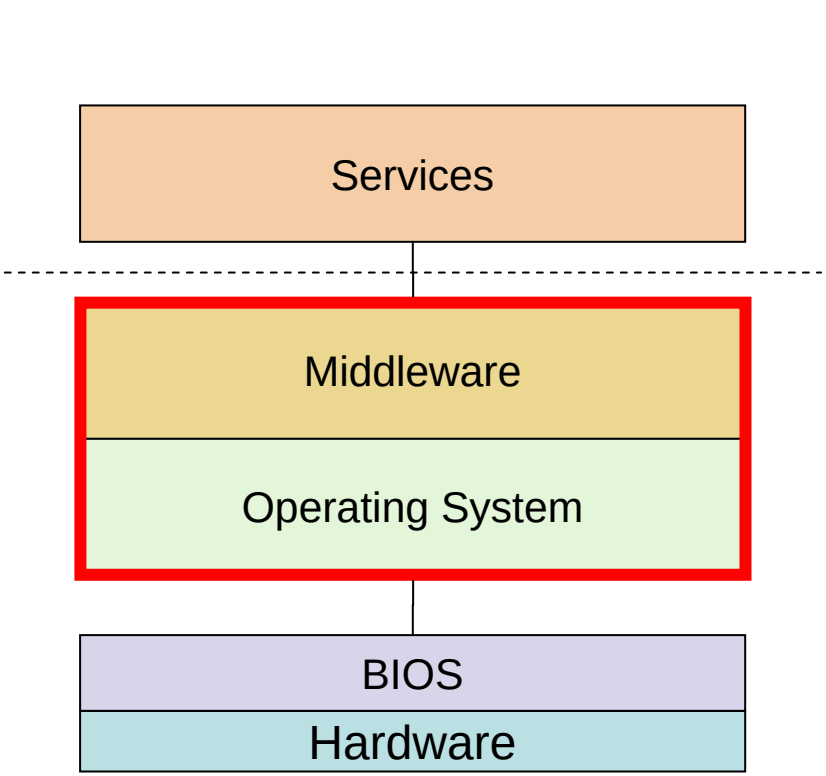
Currently, two test implementations

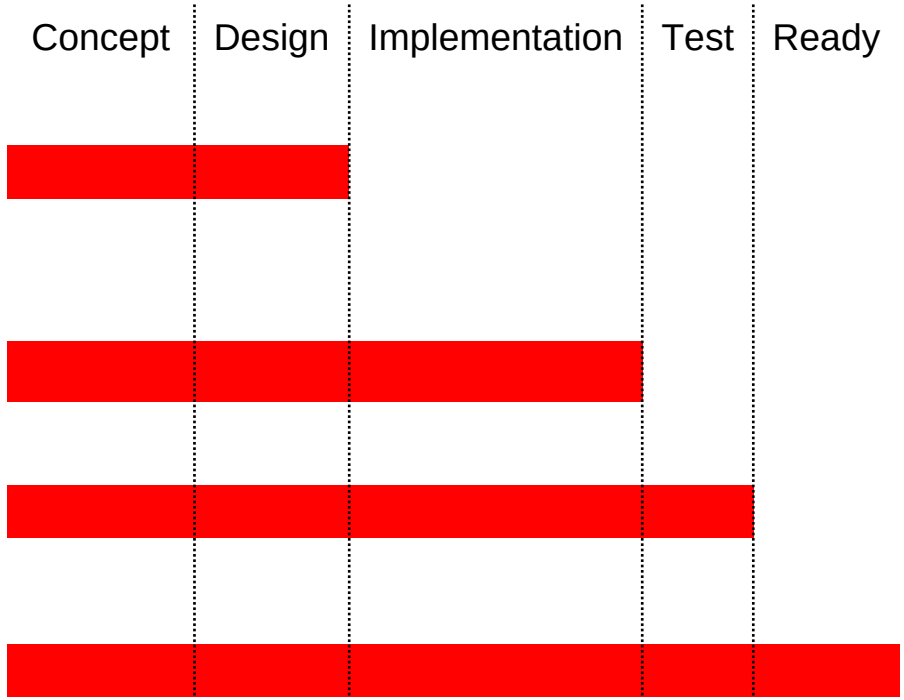
- Cygwin
  - Protocol and design checking
  - No “real” flashing
- Keil Microcontroller Development
  - Rudimentary implementation (174 bytes of code)
  - Not fully working



Current State

Implementation

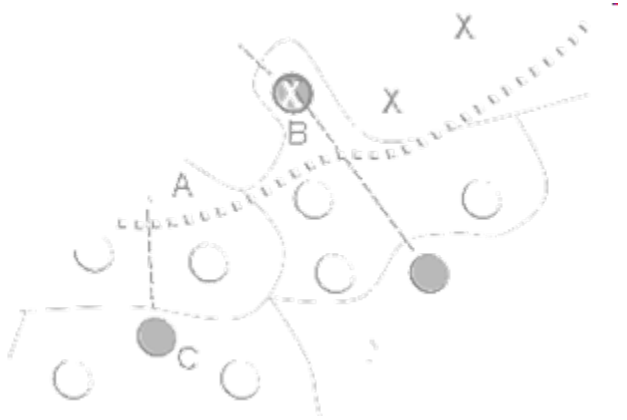




Conclusion

Service-oriented software architecture supporting

- Dynamical updates/requests
- Specific memory architectures of microcontrollers
- Collaboration of nodes
- Mobile objects
- Robustness and self-healing effects
- Simple interfaces and protocols



Thank You!

www.sensornetworks.org