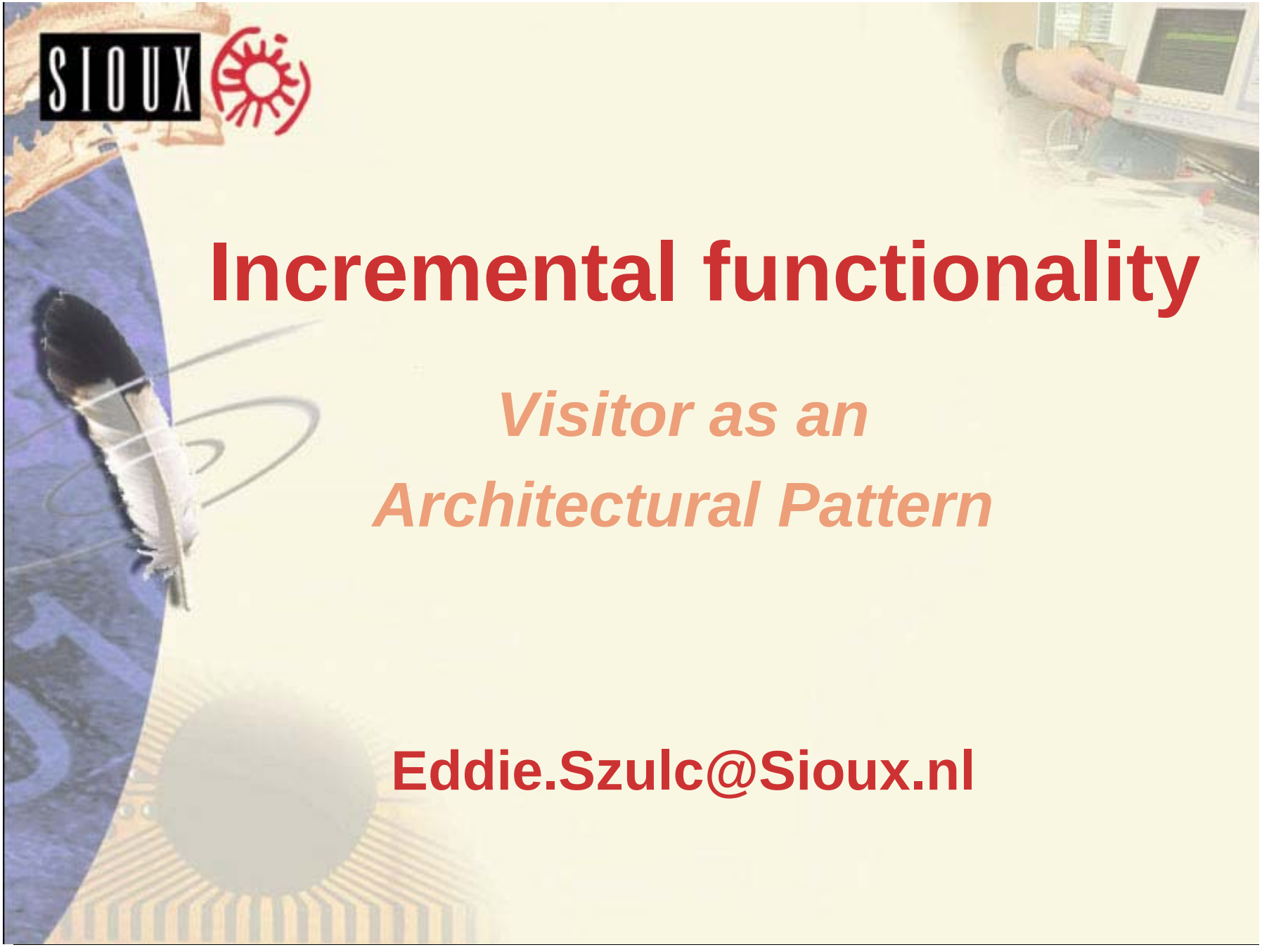



# Incremental functionality

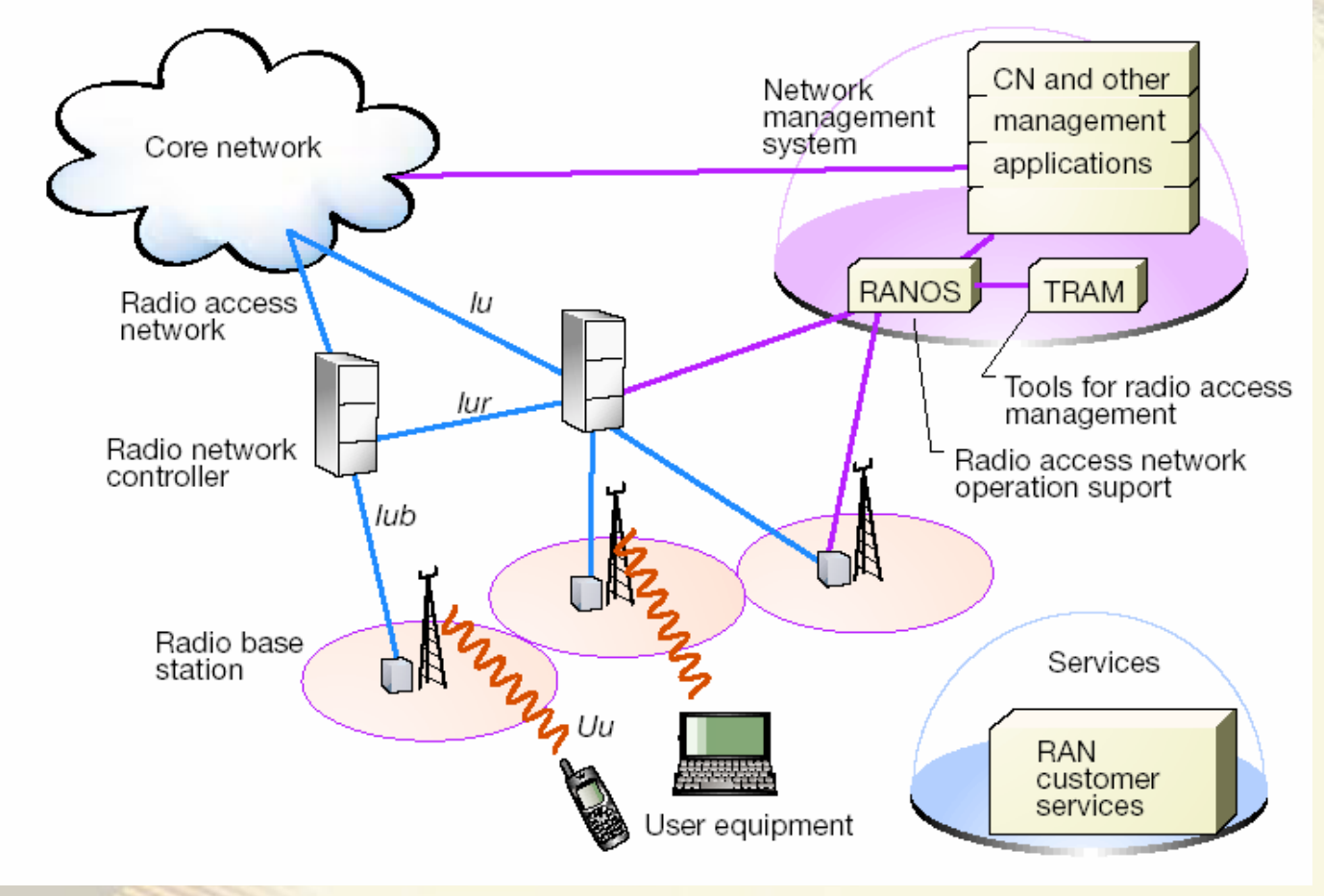
## *Visitor as an Architectural Pattern*

**Eddie.Szulc@Sioux.nl**


# Contents

- Domain overview
- Problem description
- Visitor Pattern and application
- Benefits and drawback
- Other examples

# UMTS: Third Generation GSM

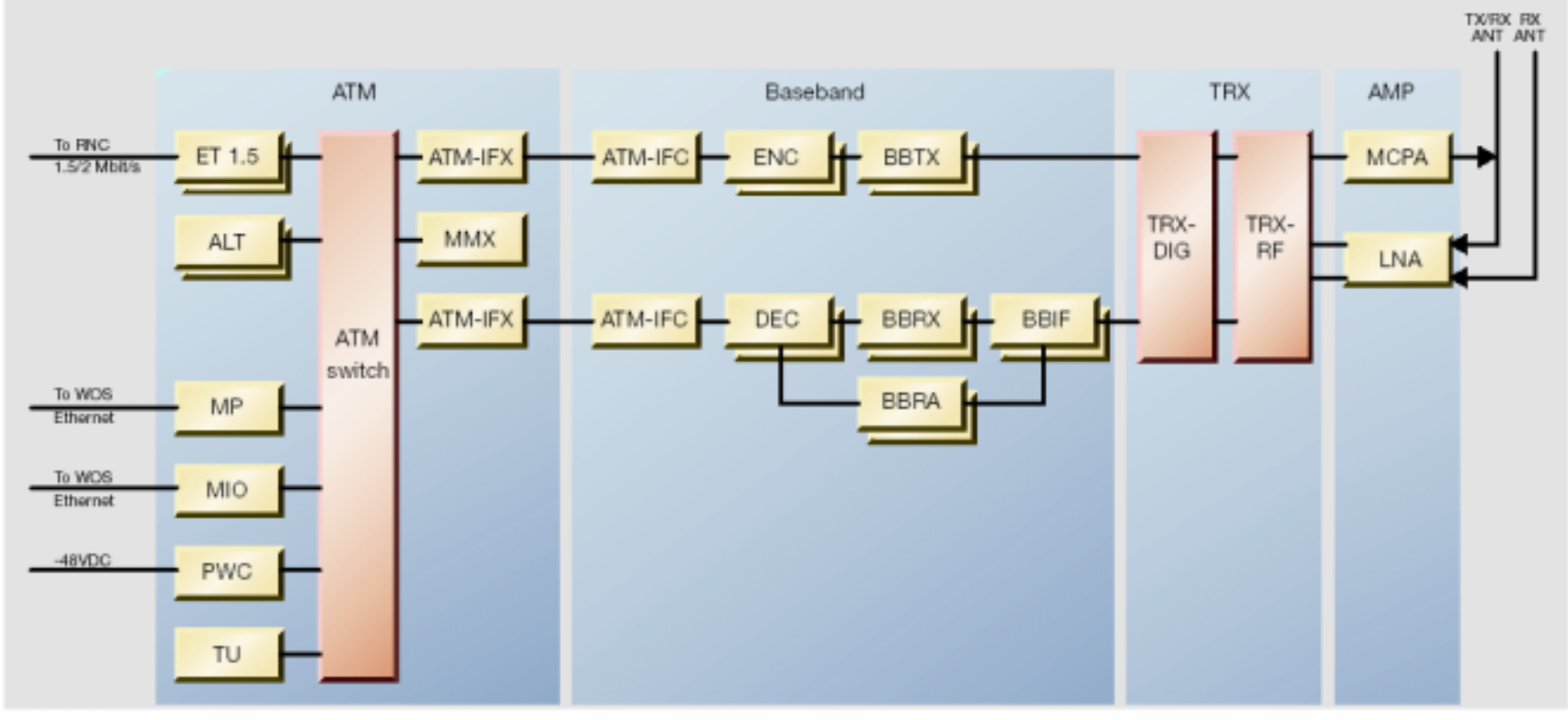


# Signal flow

- Upstream & Downstream
  - Antenna interface
  - Filtering
  - Modulation & demodulation
  - A/D & D/A
  - Encoding & decoding
  - ATM interface

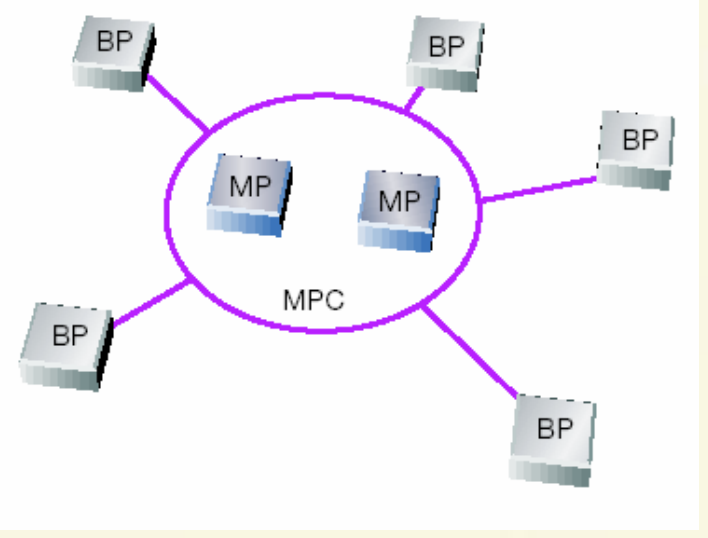
# Signal flow block diagram

| | | | | | |
|---|---|---|---|---|---|
| ALT | ATM link termination | BBRX | Baseband receiver | | |
| ATM-IFC | ATM interface client | BBTX | Baseband transmitter | LNA | Low-noise amplifier |
| ATM-IFX | ATM interface host | DEC | Decoder | MMX | ATM multiplexor |
| BBIF | Baseband interface | ENC | Encoder | TRX-DIG | Transceiver, digital part |
| BBRA | Baseband random access | ET | Exchange terminal | TRX-RF | Transceiver, radio frequency part |



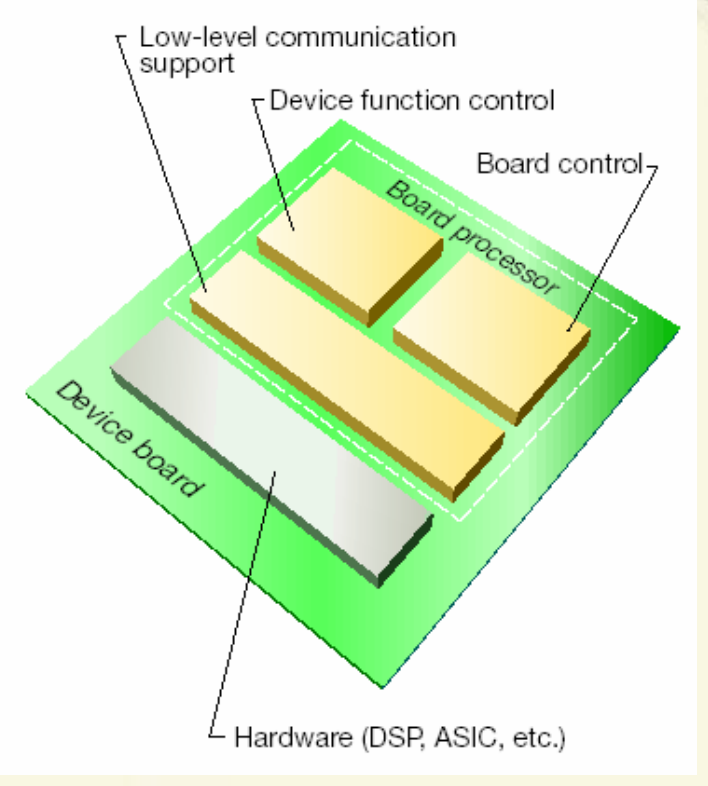
# Radio Base Station processor structure

- Main Processor (cluster)
- Multiple Board Processors



# Board Processor

- Hardware devices
  - Mix of DSPs, ASICs, FPGAs
- General software
  - Operation & Maintenance
  - Resource and functionality handling
- Specific software
  - Per device

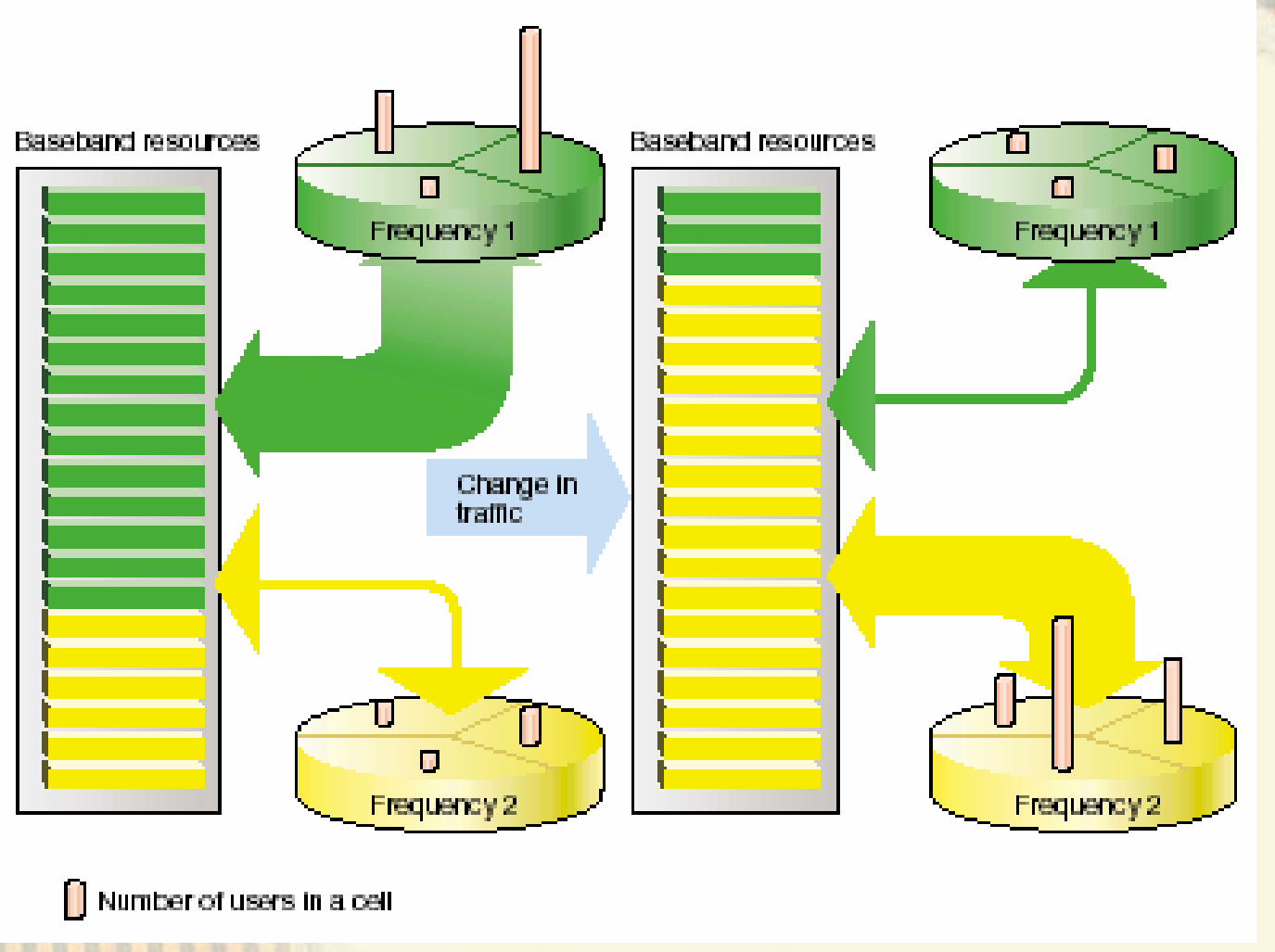


# Architectural requirements

- Handle a mix of traffic (voice, circuit-switched and packet-data services) without hardware reconfiguration
- Different network structures
- Redundancy

- Modular design, software configurable
- Scalable architecture and easy to expand

# Dynamic resource allocation



# Project characteristics

- 1,000 man
- Multiple layers of integration
- Incremental functionality
  - General and specific
- Incremental BP structure
  - New devices
  - New configurations
- Functionality and structure changing at different times

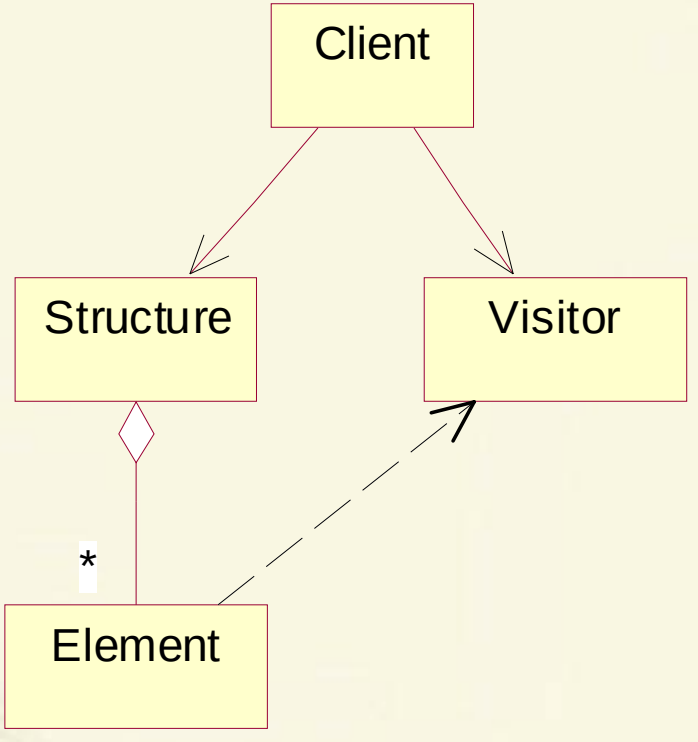
# Functionality matrix

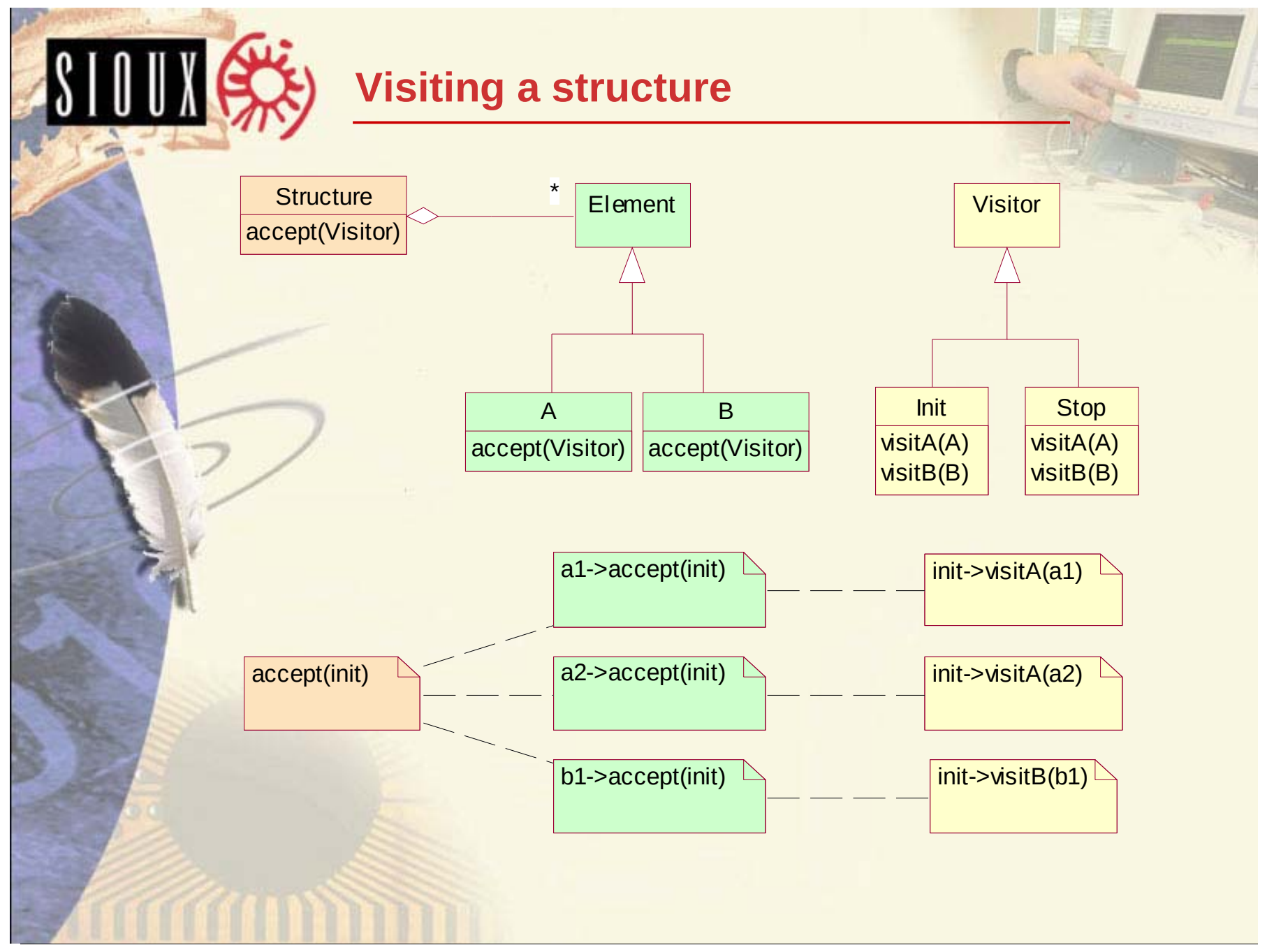
- Few types, many functions

| | DSP | FPGA | ASIC |
|---|---|---|---|
| **initialise** | X | X | X |
| **test** | X | X | X |
| **configure** | X | X | X |
| **etc** | X | X | X |

# Visitor structure top level

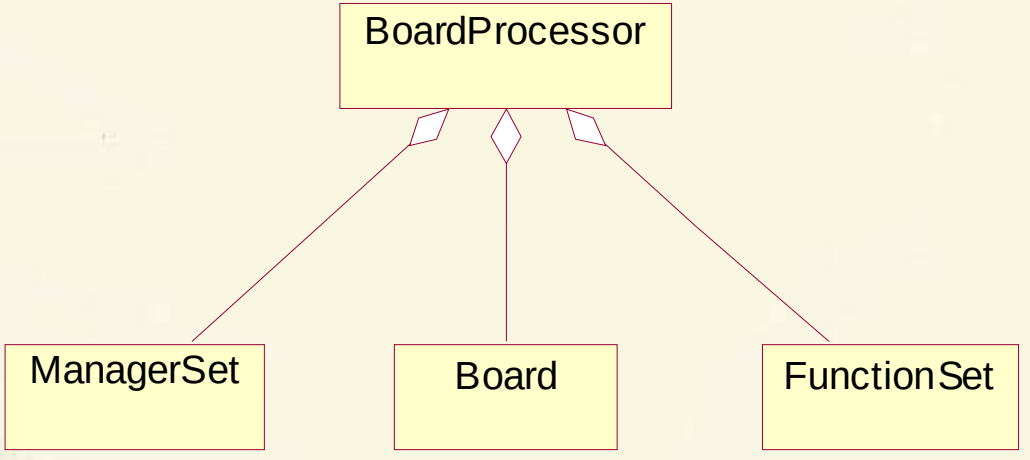
- Client asks Visitor to visit Structure
- Structure consists of elements
- Each Element offers itself to the Visitor



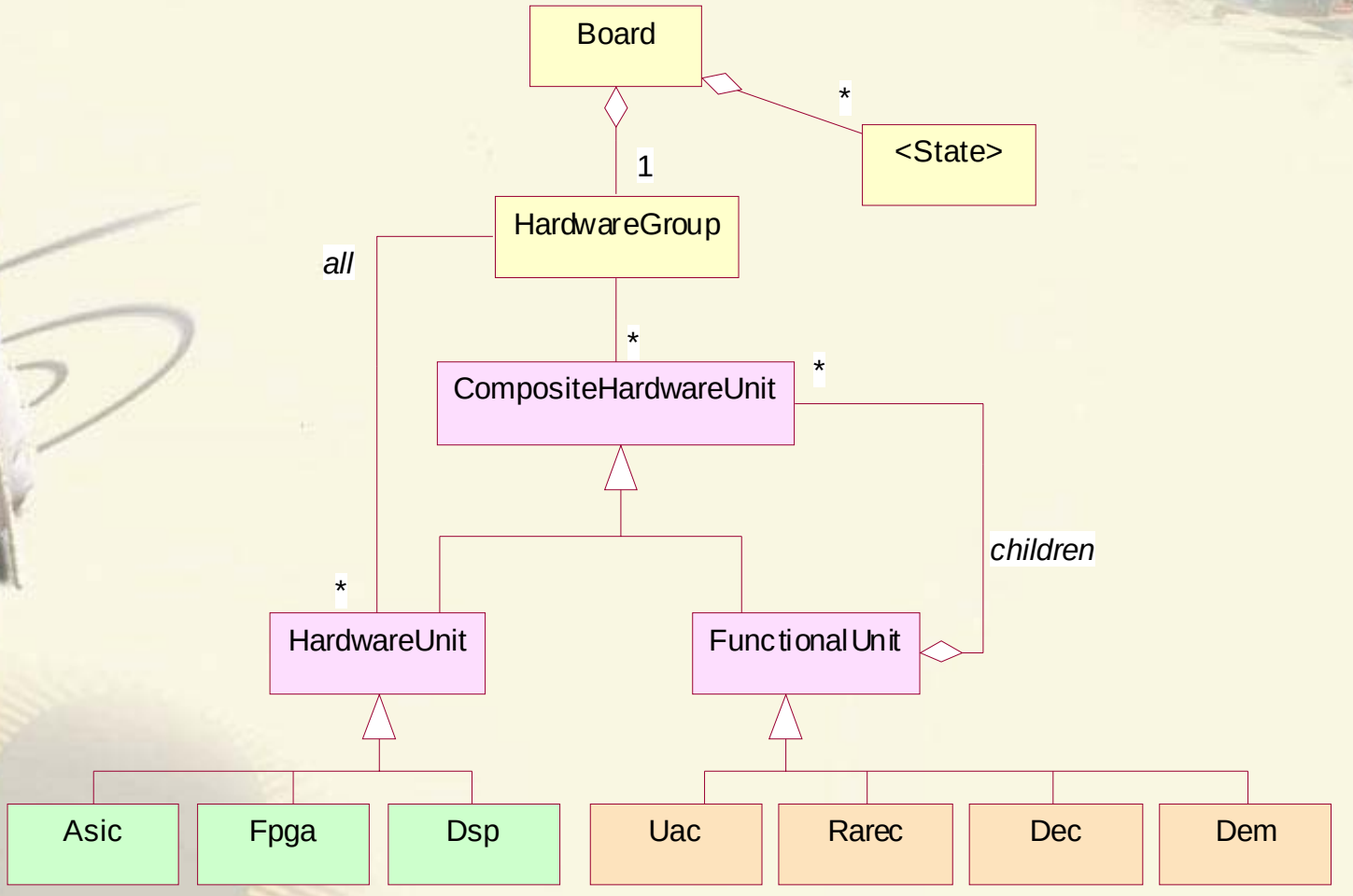

SIOUX
Visiting a structure
Structure
accept(Visitor)
*
Element
Visitor
A
accept(Visitor)
B
accept(Visitor)
Init
visitA(A)
visitB(B)
Stop
visitA(A)
visitB(B)
accept(init)
a1->accept(init)
a2->accept(init)
b1->accept(init)
init->visitA(a1)
init->visitA(a2)
init->visitB(b1)


# High-level structure

- BoardProcessor is the Client
- Board is the Structure
- FunctionSet and ManagerSet are both sets of Visitors



# Board structure




# Functions

- Visualisation (LEDs)
- Fault handling
- Measurement handler
- Test handler
- Configuration handler
- Device handler
- etc.


# Managers

- Managers
  - Measurement manager
  - Test manager
  - Configuration manager
  - Device manager
  - Resource manager
- Responsibilities
  - Check if application is in the correct state
  - Has enough processing resources available
  - Appropriate functionality to handle the signal
  - Keeping track of the available functionality


# Benefits 1/2

- Add functionality without modifying element classes
  - Cheap and easy
  - Avoid pollution of classes with disjoint operations
  - Relevant functionality per application for shared structures
  - Element classes not related by inheritance
  - Visitors can accumulate state transparently
- Grouping of related functionality
  - Encapsulation of functionality
  - Optimisation of each function (+ inheritance)
  - Easy to change algorithm
  - Related to Aspect-Oriented Programming


# Benefits 2/2

- Separation of functionality and structure
  - Separate development
  - Incremental functionality
- Not necessarily OO
  - Can be implemented in non-OO languages


## Drawbacks

- Loss of encapsulation
  - Element type and operations are separated
  - Visitor may need internal information from element
- Adding a new Element type affects all Visitors
  - All Visitors may be extended
- Changing the Element types is costly
  - Interface to all Visitors must be redefined
- Double-dispatch
  - Dependence on both Visitor and Element types
  - Run-time binding


# Example uses

- Graphical editors:
  - Icons: add, remove, modify, open, manipulate
- Compilers and interpreters
  - Syntax trees: type checking, optimisation, pretty-printing, metrics
- Dynamic structure
  - Structure changes at run-time
  - Structure traversal depends on current result
- Component
  - Configurable HW or SW components
  - 3D virtual worlds
- Composite
  - consider using Visitor, too


# Conclusion

- Visitor can be used as an architectural pattern
  - To control functional extensibility
  - To separate structure from functionality
  - To encapsulate functionality
- Visitor was a good solution for the problem
  - Was not difficult to implement
  - Simplified adding and changing functionality