3rd International Erlang User Conference

1997-Aug-26

Ericsson Software Technology AB
Erlang Systems
Box 1214
SE-164 28 KISTA
SWEDEN
http://www.ericsson.se/erlang/

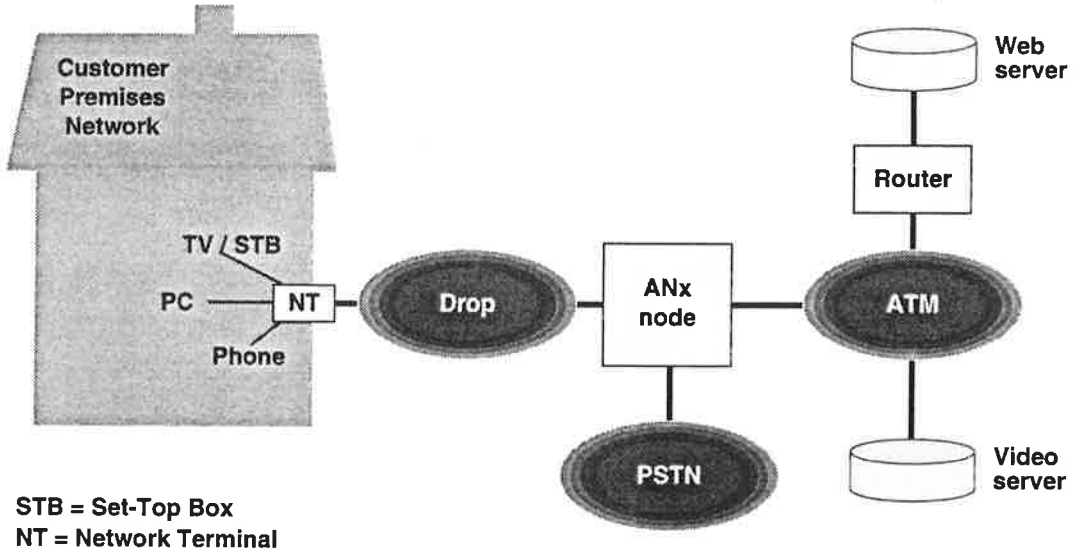
phone: +46-8-719 0000
fax +46-8-719 8940

Programme

08.30	**Registration opens**
09.00	**Opening** Opening speech by *Bernt Ericson* V.P. Research & Technology, LM Ericsson.
09.30	User presentation: **Next generation high speed internet access using Erlang** by *Martin Rinman*, Ericsson Telecom (nowdays at Erlang Systems)
10.00	User presentation: **A distributed application in Erlang on Windows-NT** by *Jörn Svendsen*, LM Ericsson A/S (Denmark). "Use of Erlang and Windows-NT in a distributed environment. Load and security considerations from a real life telecom product with high availability requirements. Windows-NT what to consider? Configuration of Windows-NT and the network."
10.30	Coffee
11.00	User presentation: **SwitchBoard - default hardware for Erlang** by *Ulf Svarte Bagge*, Ericsson Business Networks AB. "SwitchBoard is a modular high capacity switch capable of handling a large number of E1/T1 ports. It is designed as a general telecom hardware controlled via an API in Erlang. Main features: Scalability: Scalable from small to large at a linear cost, one or several identical modules (SwitchBoards) interconnected via an optical "backplane" Fault tolerance: Physically distributed switching without central part and with low level redundancy control Other features such as small physical size, low cost, termination of layer 2 on board, generation of tones, simultaneous switching of different bandwidts, alternative physical interfaces or resources on mezzanine boards and much more."
11.30	User presentation: **Erlang and a new paradigm for software engineering** by *Prof. Fergus O'Brien*, Software Engineering Research Centre (SERC) - Australia. The presentation will be given by *Helen Airiyan*, SERC. "The paper presents an approach to the complete life-cycle of major software projects based on the incorporation of non functional requirements from the initial problem definition stage. A historic perspective is used to develop the rationale for such an approach, and its use as a model for further Software Engineering developments. The practical implementation of this approach is illustrated through a system that has been built using Ericsson functional language environment, Erlang. The ongoing research and future directions are also outlined. "
12.00	LUNCH Menu: To be announced. If you have any special requirements such as vegetarian food, please let us know in your advance registration.
13.15	User Presentation:**Using Erlang for ATM-switch control** by *Ulf Wiger*, Ericsson Telecom. "This presentation covers experiences from using Erlang in the development of broad-band switch control software."

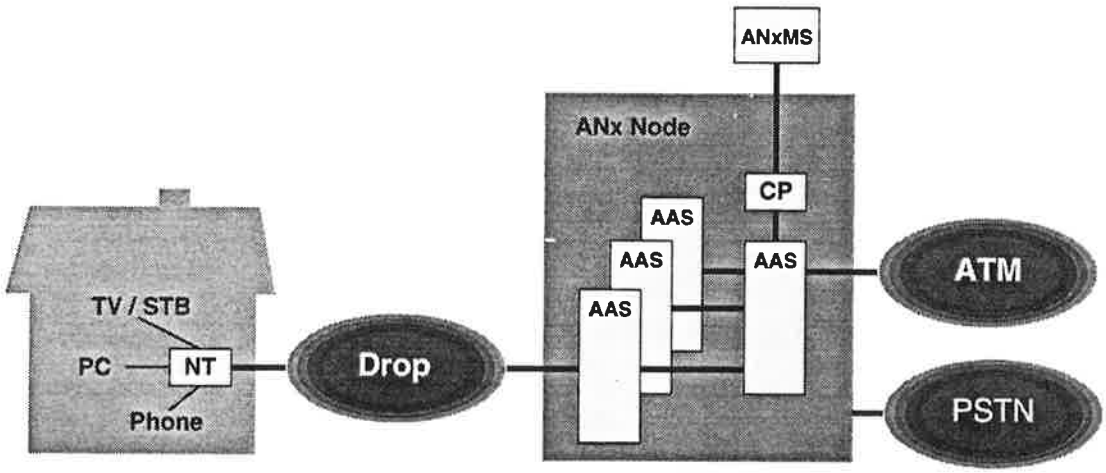
13.45	**Future directions for the development of Erlang System/OTP** by *Mike Williams*.
14.15	**CORBA enabled Erlang** by *Peter Lundell*, Ericsson Telecom. "A CORBA package for Erlang System/OTP is under development. The presentation covers its features, usage and implementation. The package will soon be available as an alpha release to interested users."
14.45	Coffee
15.15	To be confirmed: INETS, the Internet package of Erlang System/OTP (Java connectivity, HTTP server, etc)
15.45	**Towards an Even Safer Erlang** by *Dr. Lawrie Brown*, Australian Defence Force Academy and *Dan Sahlin*, Computer Science Laboratory - Ericsson. " This talk discusses on-going research into extending the Erlang system to better support constrained and partitioned execution of code. This could be used to support mobile agent, applet, or outsourced code execution, or simply for improved fault tolerance. Extensions adding a hierarchy of nodes, and capabilities for nodes/pids/ports are being prototyped."
16.15	Short break
16.30	**Erlang type-system** by *Joe Armstrong*, Computer Science Laboratory - Ericsson. "A type system for Erlang has been developed in a collaboration between the Ericsson Computer Science Laboratory and Phil Wadler and Simon Marlow from Glasgow University. The type system can detect inconsistencies in Erlang programs and can be used to verify that Erlang programs are well-typed. Well-typed programs are guaranteed never to fail with run-time type errors, all such errors are detected at compile time. In addition the type notation can be used as a useful design tool. The talk gives a brief introduction to the type system."
16.50	**Etos: an Erlang to Scheme compiler** by *Prof. Marc Feeley* and *Martin Larose*, University of Montreal. "The programming languages Erlang and Scheme have many common features, yet the performance of the current implementations of Erlang appears to be below that of good implementations of Scheme. This disparity has prompted us to investigate the translation of Erlang to Scheme. In this paper we describe the design and implementation of the Etos Erlang to Scheme compiler and compare its performance to other systems. On most benchmark programs, Etos outperforms all currently available implementations of Erlang."
17.10	**IUSTITIA - Erlang based load balancing experiments** by *Sasa Desic, Zrinko Kolovrat* and *Prof. Ignac Lovrek* , Department of Telecommunications, Faculty of Electrical Engineering, University of Zagreb - Croatia.
17.35	**Running Erlang on multiprocessor systems** by *Pekka Hedqvist* , Erlang Systems. "Erlang processes are parallel in nature, as is many telecom applications. Multi-processor computers is becoming commodity hardware for a range of applications. To ease the construction of high-end scalable systems it is desirable to enable the Erlang runtime system to execute Erlang processes in parallel. A multi-processor Erlang runtime system has been developed - current results are presented and discussed. An "FoU" release is scheduled this year."
18.00	**Conclusion and questions.**

ANx
Network Overview



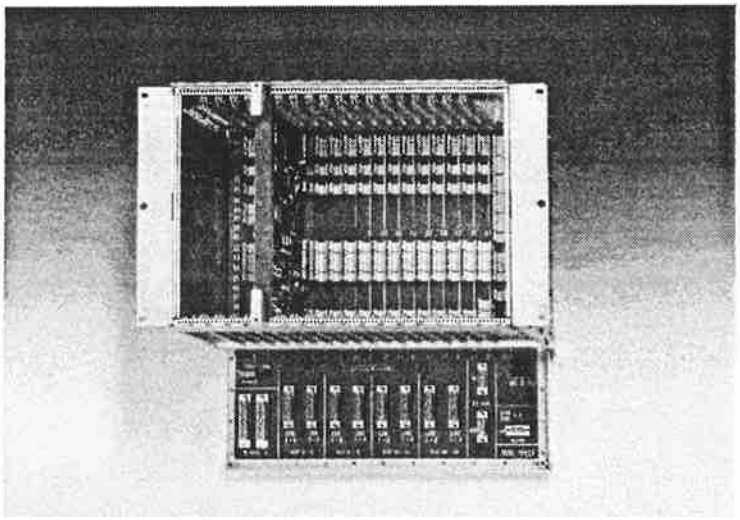
STB = Set-Top Box
NT = Network Terminal
PSTN = Public Telephony Network

ANx
Building blocks



ANx

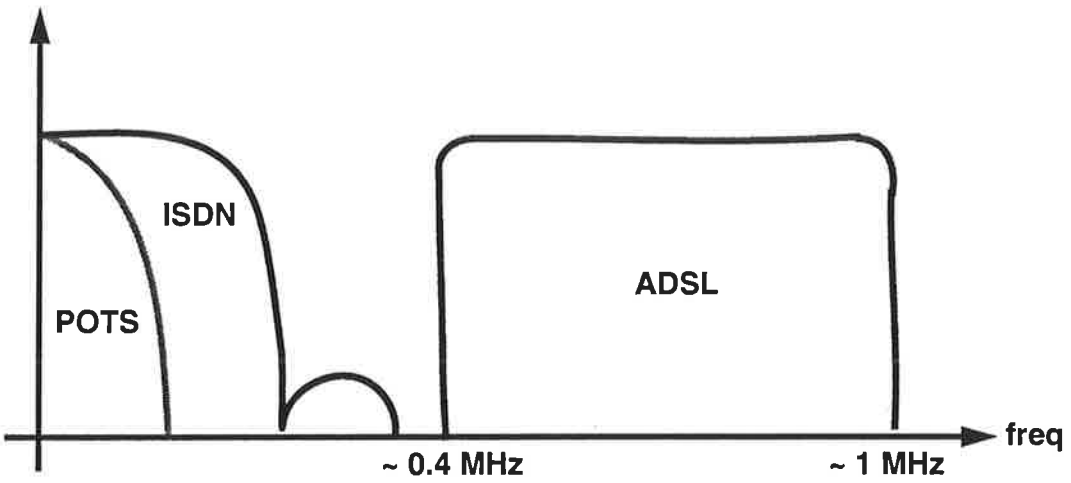
ATM Access Shelf (AAS)



ERLANG

ADSL

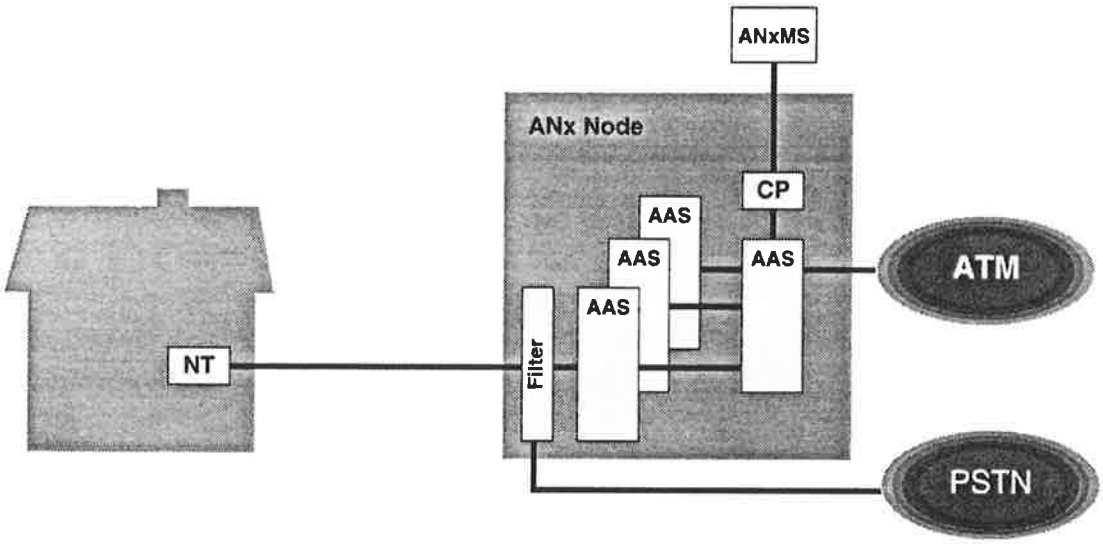
Principles



ERLANG

ANx DSL

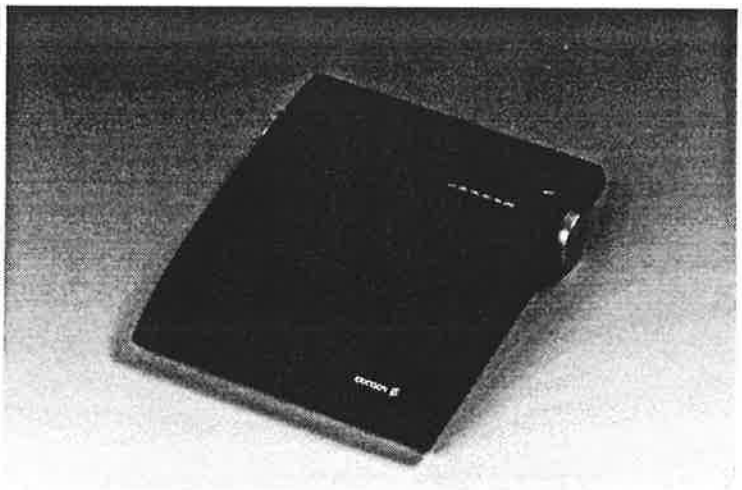
Network Overview



ERLANG

ANx DSL

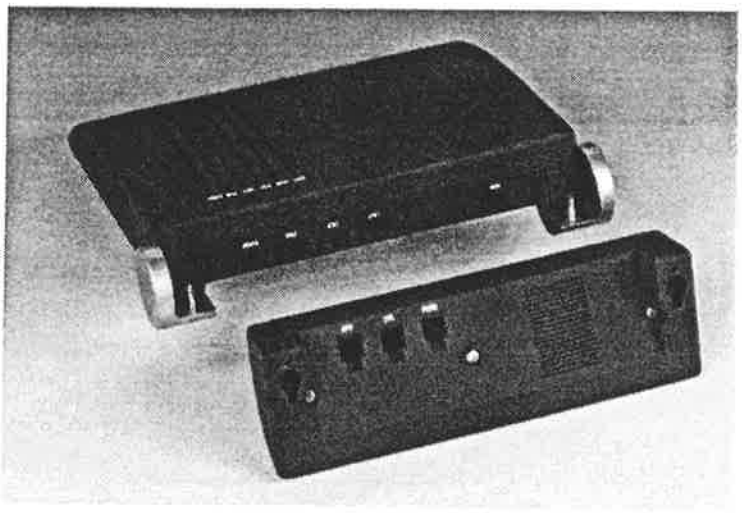
Network Terminal



ERLANG

ANx DSL

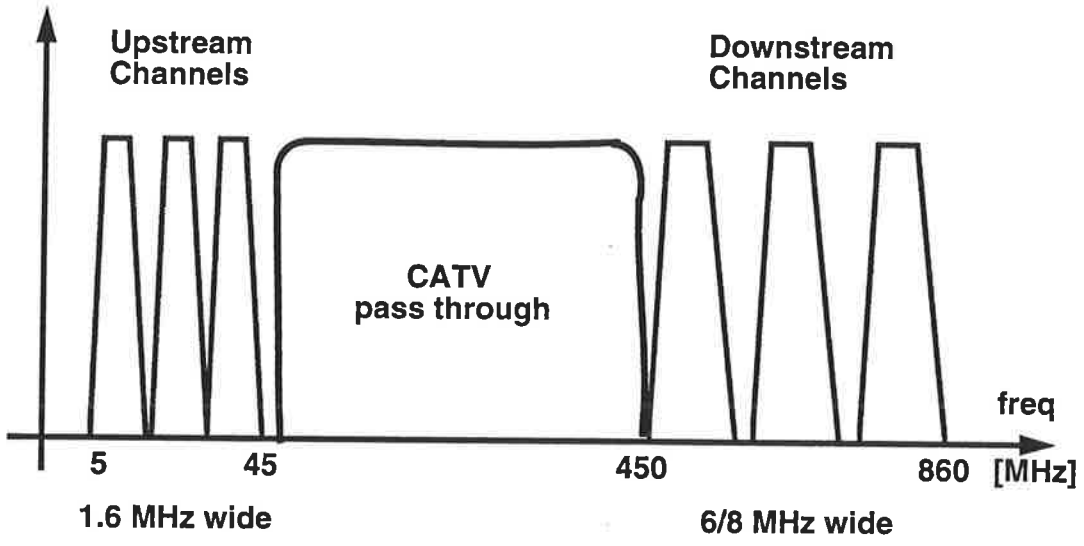
Network Interfaces



ERLANG

HFC

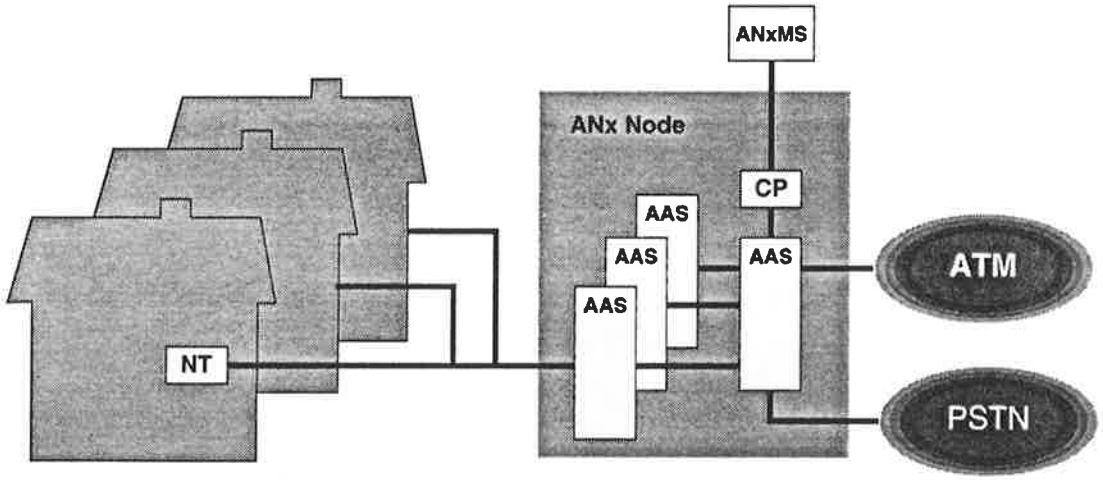
Principles



ERLANG

ANx HFC

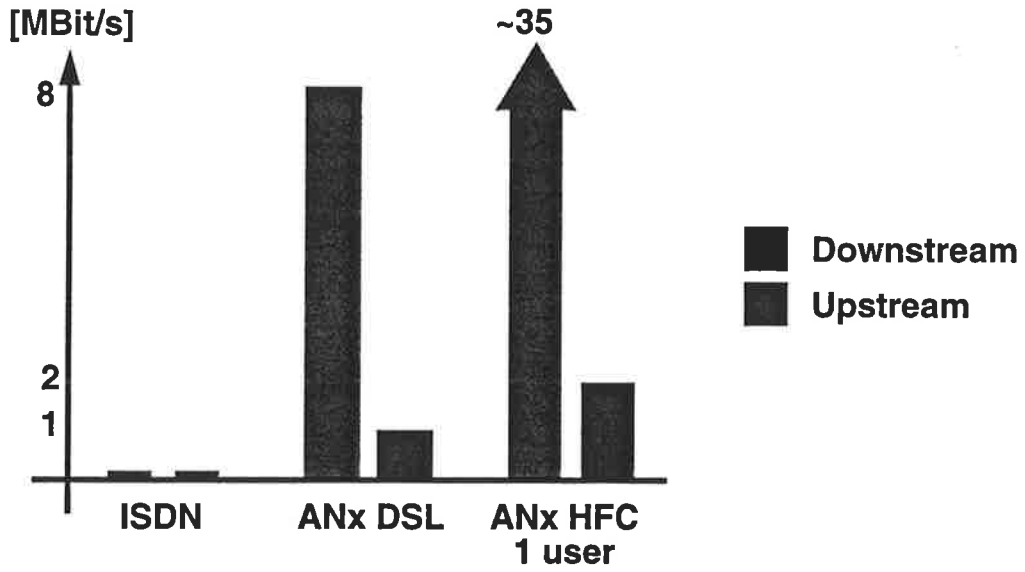
Network Overview



ERLANG

ANx

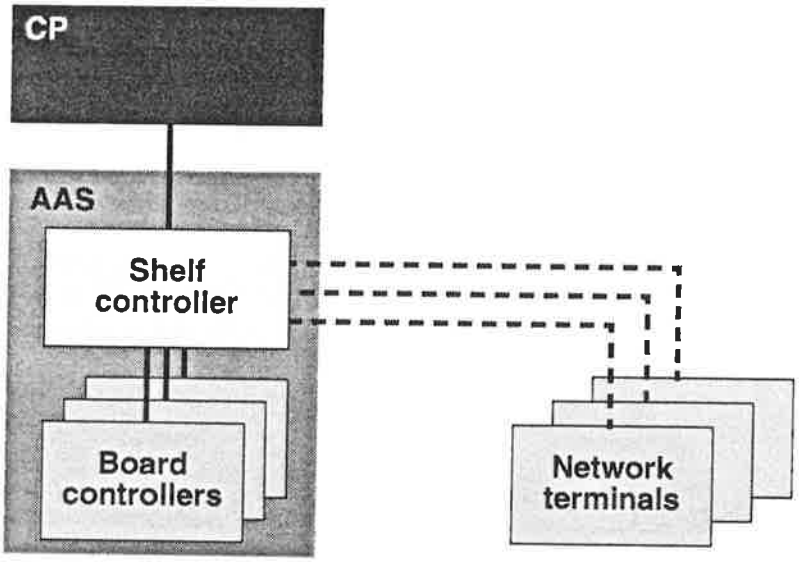
Speed



ERLANG

ANx

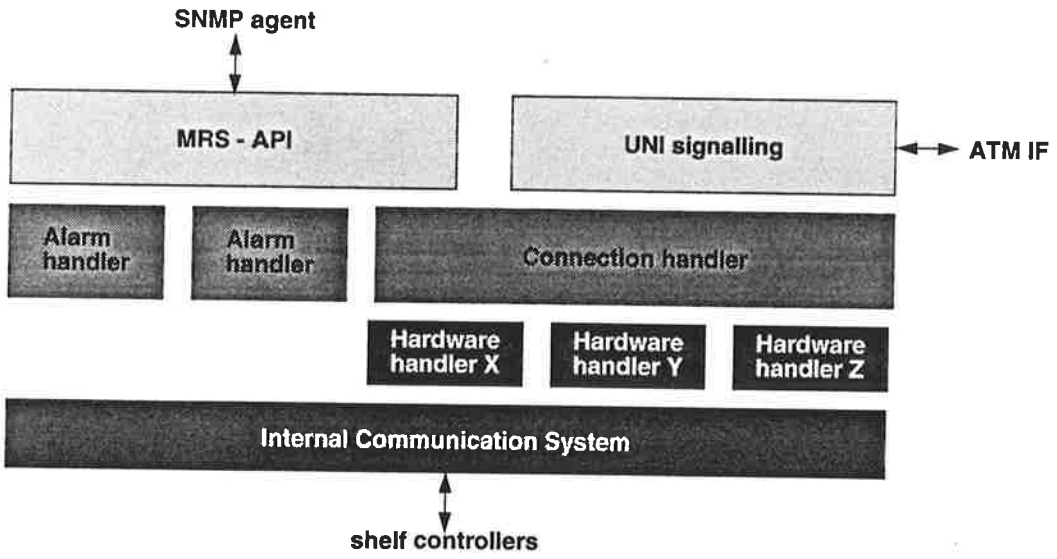
Processor Hierarchy



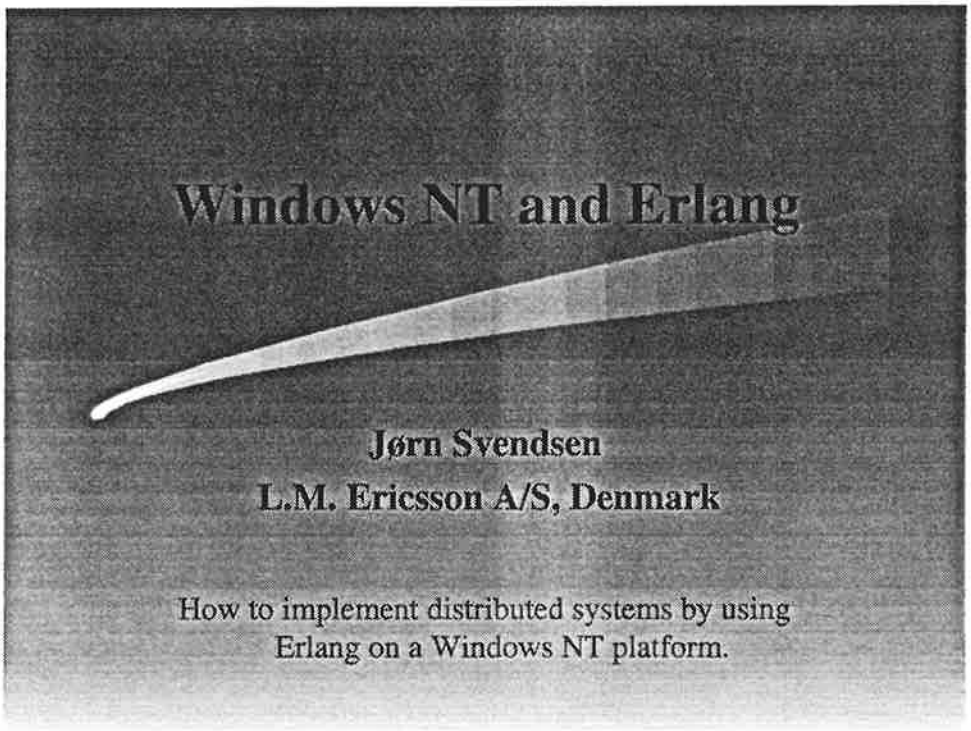
ERLANG

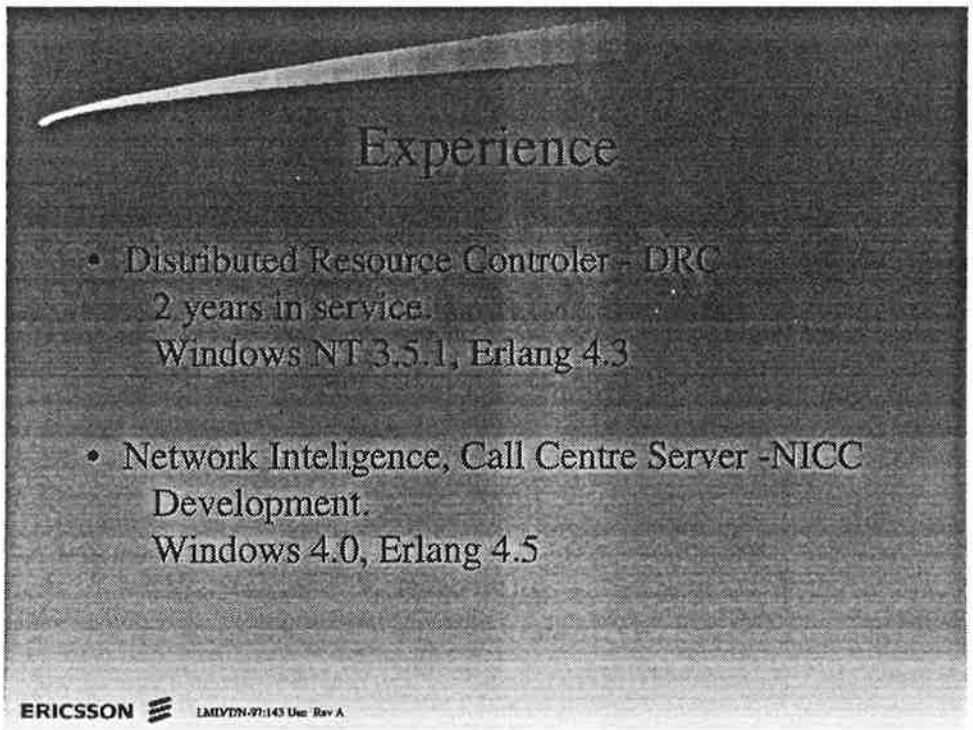
ANx

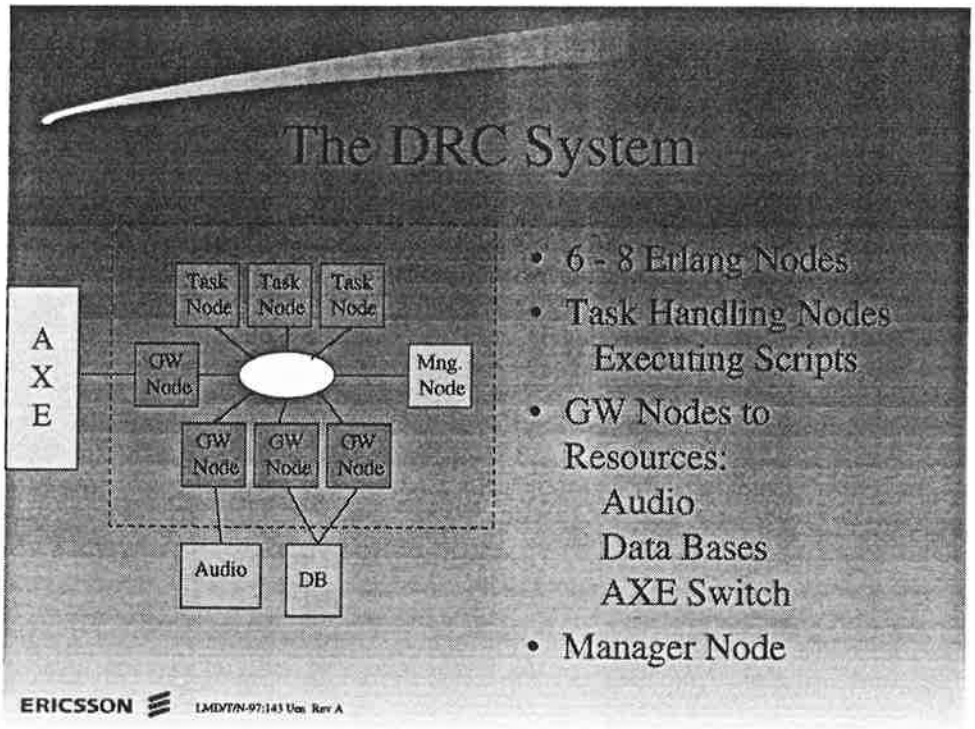
CP Application SW

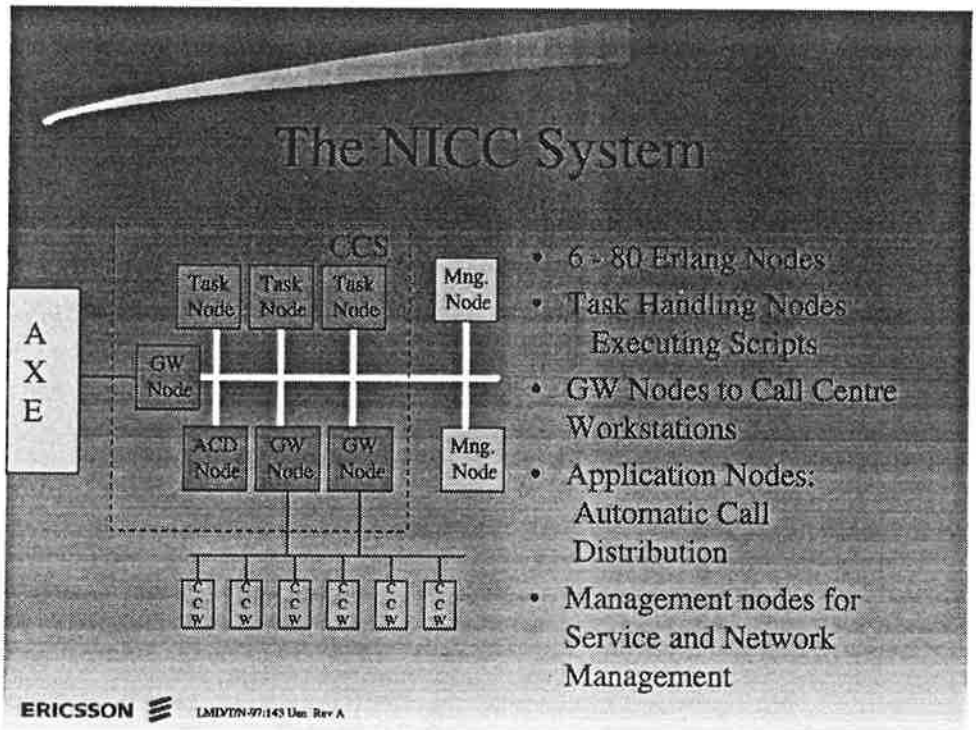


ERLANG


Windows NT and Erlang
Jørn Svendsen
L.M. Ericsson A/S, Denmark
How to implement distributed systems by using
Erlang on a Windows NT platform.

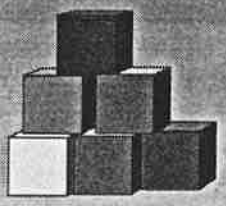

Experience
• Distributed Resource Controler - DRC
2 years in service.
Windows NT 3.5.1, Erlang 4.3
• Network Inteligence, Call Centre Server -NICC
Development.
Windows 4.0, Erlang 4.5
ERICSSON


The DRC System
AXE
Task Node
Task Node
Task Node
GW Node
Mng. Node
GW Node
GW Node
GW Node
Audio
DB
• 6 - 8 Erlang Nodes
• Task Handling Nodes Executing Scripts
• GW Nodes to Resources:
Audio
Data Bases
AXE Switch
• Manager Node
ERICSSON


The NICC System
CCS
AXE
Task Node
Task Node
Task Node
Mng. Node
GW Node
ACD Node
GW Node
GW Node
Mng. Node
CCW
• 6 - 80 Erlang Nodes
• Task Handling Nodes Executing Scripts
• GW Nodes to Call Centre Workstations
• Application Nodes: Automatic Call Distribution
• Management nodes for Service and Network Management
ERICSSON

Hardware

- Office PC's, Pentium (75MHz, 90 MHz, 120MHz, 200 MHz)
- PCI bus interface board, for SS7
- Switch box connected screen, keyboard
- Token Ring - 16MHz
- Future:
  - Switched Ethernet
  - UPS in cabinet
  - TCAP



ERICSSON LMD/TN-97:145 Uen Rev A

Windows NT Basic Setup

- Windows NT - Workstation used for Erlang Nodes - Less expensive
- Local definition of Host Names
  - No central DNS which can fail
- Automatic logon after Booting
  - System to start automatically after power failure
-

ERICSSON LMD/TN-97:145 Uen Rev A

SNMP Based Management

- SNMP Service in Windows NT
- Standard MIB's for Network Supervision and PC performance
- Interface from Erlang to the Extended Agent API
- Private MIB for Script Loading, Node Close down,Alarm handling, start/stop of Logging and Statistics

ERICSSON LMDT/N-97:143 Uen Rev A

DRC Load Figures

- Figures from one Customer
- 8 Different Services
- A total of 150 000 Calls A day
- 20 000 BHCA
- Some services show large fluctuations
  - Campaigns
  - Changed Number - New Exchange

ERICSSON LMDT/N-97:143 Uen Rev A

Load Considerations

- Load Regulator based on Length of RunTime Queue
- Max Load Figures based on Simple Test Program in Erlang
- Max figures for different processors:

Sun Sparc 5	120
PC 90 MHz Pentium	50
PC 150 MHz Pentium	90

ERICSSON LMI/YTN-97:143 Uen Rev A

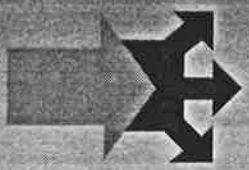
Load Problems

- Load by non Erlang Processes
  Set Right Process Priority in Windows NT
- Load Regulation Considering present number of Processes - To Predict Future Load
- Lower Erlang max. Load to allow other Processes to run on the PC

ERICSSON LMI/YTN-97:143 Uen Rev A

Future Functions

- Central Time Server
- Interface from UPS to Erlang system to close down in case of no power.
- Installation Program
  To install Erlang and Applications in one step
- OTP ?

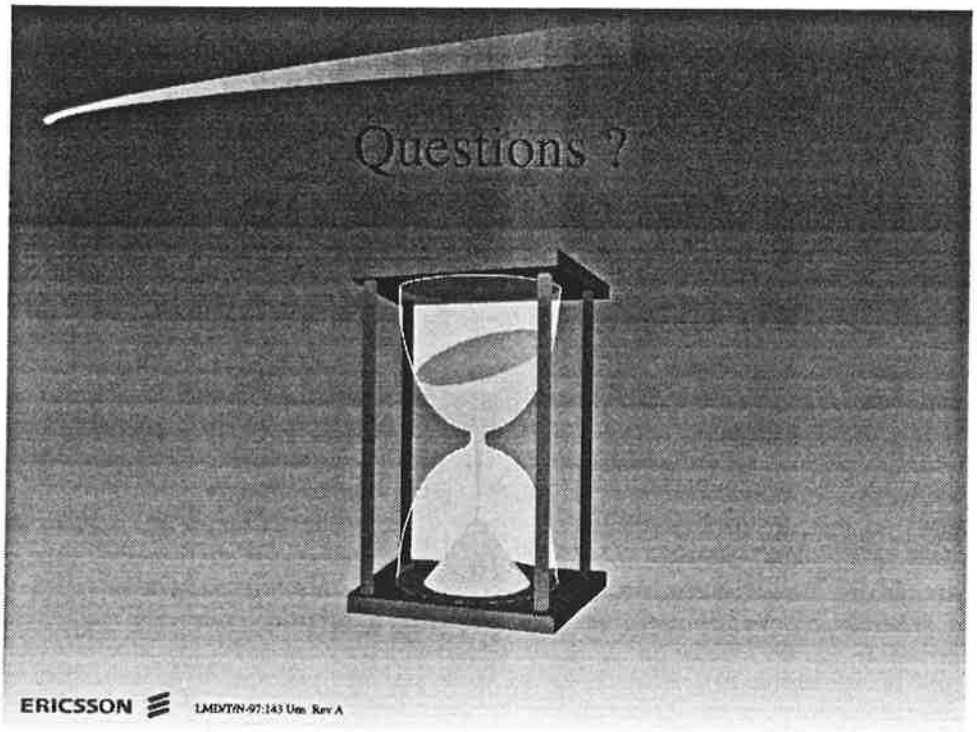


ERICSSON LMI/VTN-97:143 Uen Rev A

Erlang Wish List

- Backward Compatibility - Controlled Updating of Erlang Package
- Updating of Erlang modules in runtime for the runtime system

ERICSSON LMI/VTN-97:143 Uen Rev A


Questions ?
ERICSSON
LMD/TIN-97:143 Uen Rev A

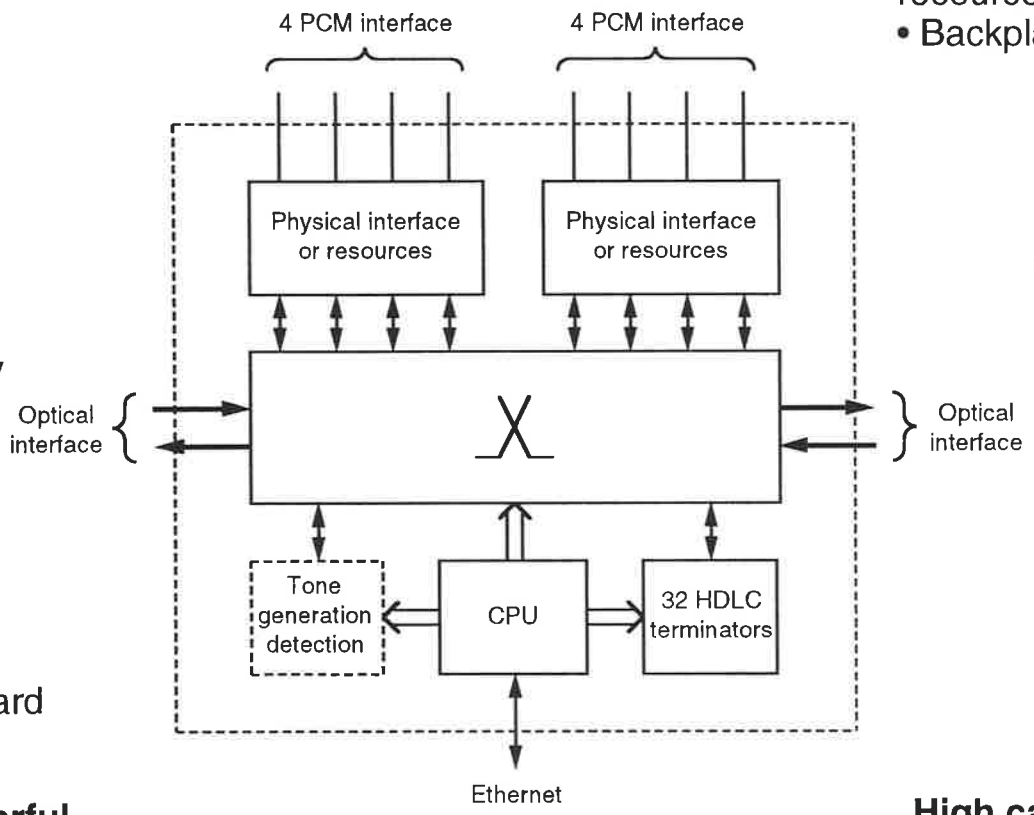
SwitchBoard

Overview

Modular
• Modular with only one type of module

Flexible
• Different physical interfaces or recources on daughter boards
• Backplane independent

Scalable
• Scalable from very small to very large at linear cost



Advanced switching
• Physically distributed non-blocking switch
• Switching of variable bandwidths

Reliable
• Distributed functionality without central part
• Low level redundancy

Upgradable
• Built with RAM based programmable logic
• Software downloadable to flash

Optional
• Optional PBX recources on board

Powerful
• Powerful processor executing distributed Erlang
• Communication to external world via ethernet

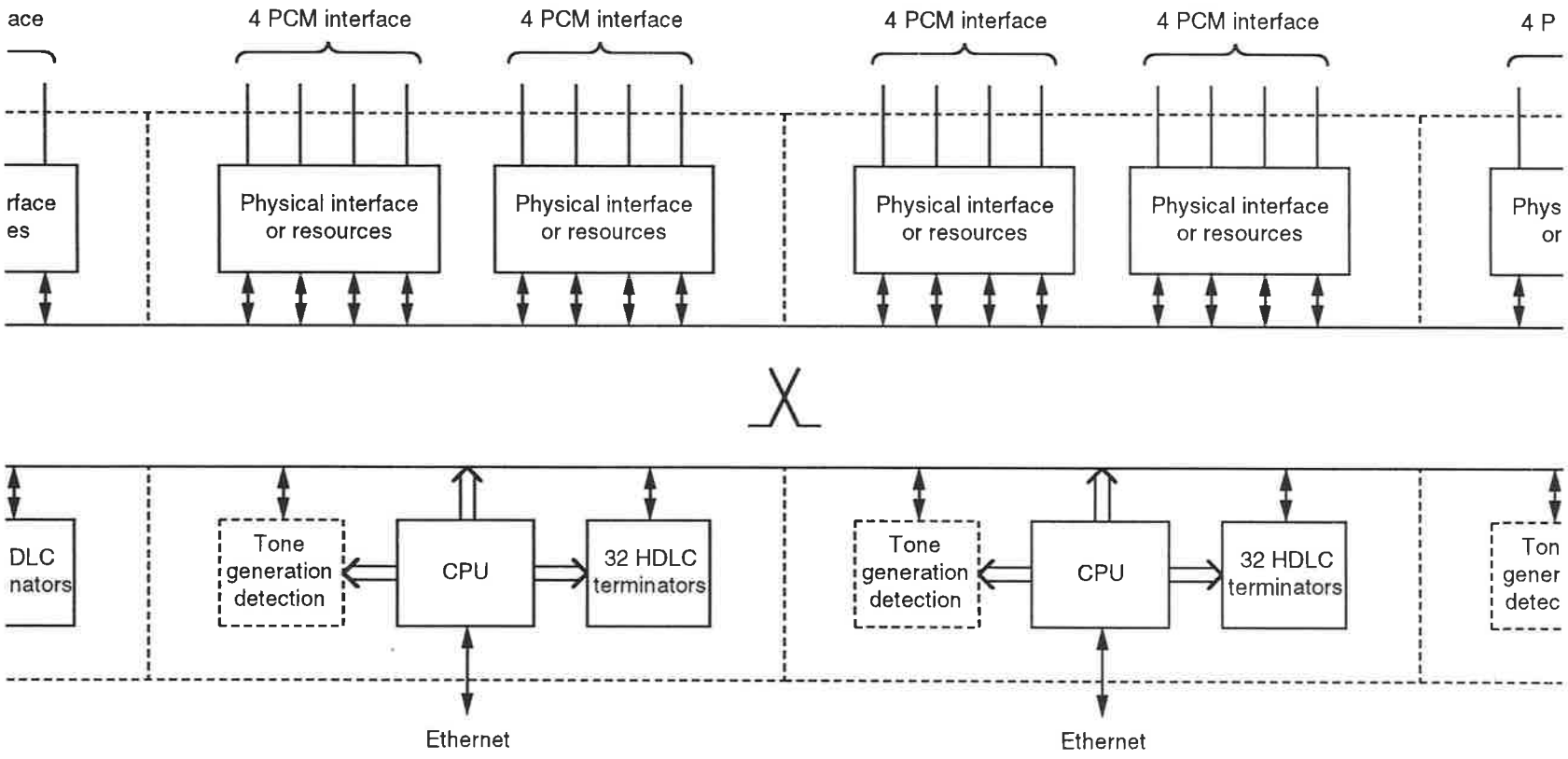
High capacity signalling
• Dynamic allocation of link layer terminators
• Allow different link layer protocols

961003

ERICSSON

SwitchBoard

Scalability

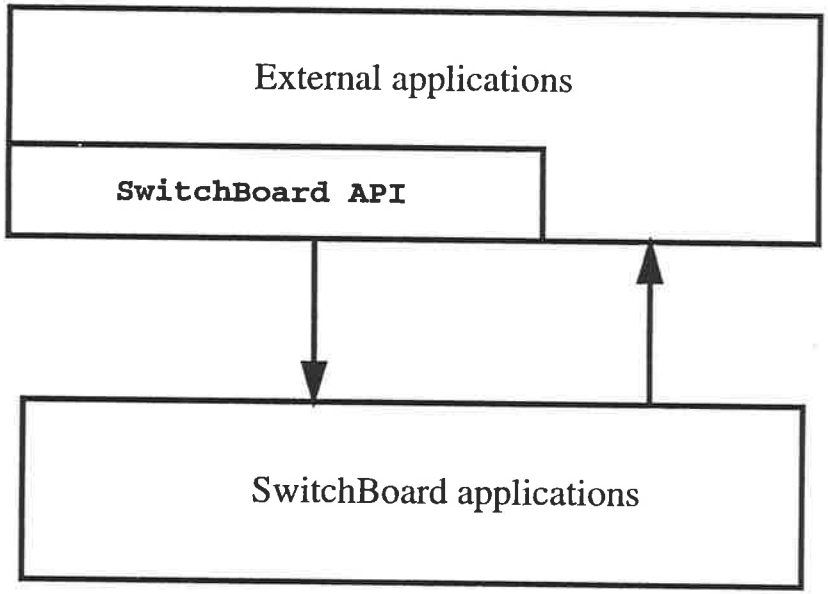


961003

ERICSSON

SwitchBoard

API



The external application can run either on the SwitchBoard or on another computer connected to the SwitchBoard via TCP/IP.

The Integration of Functional and Non-Functional Requirements

Professor Fergus O'Brien
Director, Software Engineering Research Centre

Presented by Helen Airiyan,
Software Engineering Research Centre

History

- **NATO Conferences 1967, 1970**
- **Core Elements for Software**
- **General Engineering Lessons**

The Engineering Model

- **Functionality specified**
- **Physical laws as constraint set**

The Software Engineering Model

- **Functionality Specified**
- **Non-functional requirements**
  - ❖ **Performance**
  - ❖ **Reliability**
  - ❖ **Maintainability**

New Software Engineering Paradigm

- Non-Functional Requirements ≅ Physical constraints
- Definition of Non-Functional Metrics
- Techniques for On-line Metrics
- Use Techniques throughout life cycle
- Use for control/recovery

Example of Performance Metric

- Define Application Domain as Use-Cases
- Specify Performance per Use-Case
- Budget across Modules
- Monitor per Module per Use-Case

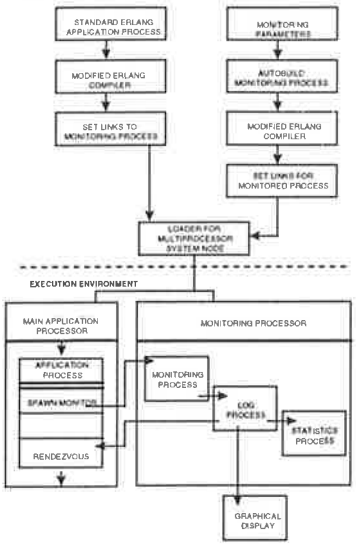
Metrics Researched

- Service Level Agreements
  Performance in C
- Reliability
- Maintainability
- Clarity

The Framework Project

- Commenced 1995
- Overall architecture for parallel handling of functional and non-functional requirements
- Based on Erlang
- MP Pentium implementation

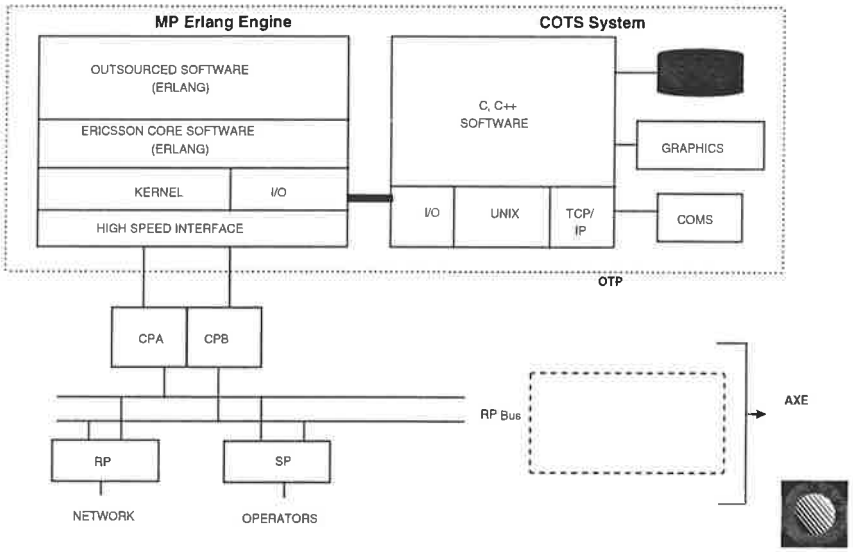
Patent Application Framework



Organisational Implications

- **Functional Code - development group**
- **Non-Functional Code - QA group**
- **Seamless development to maintenance**
- **Proactive project management**

Erlang Engine - Relationship

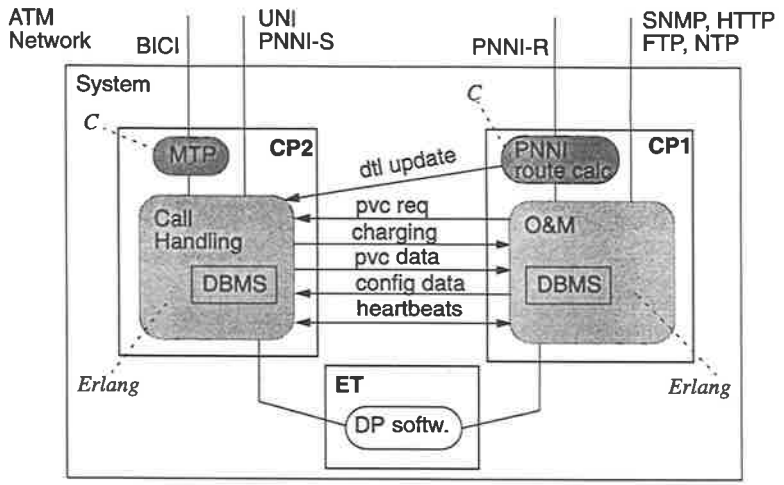


Futures

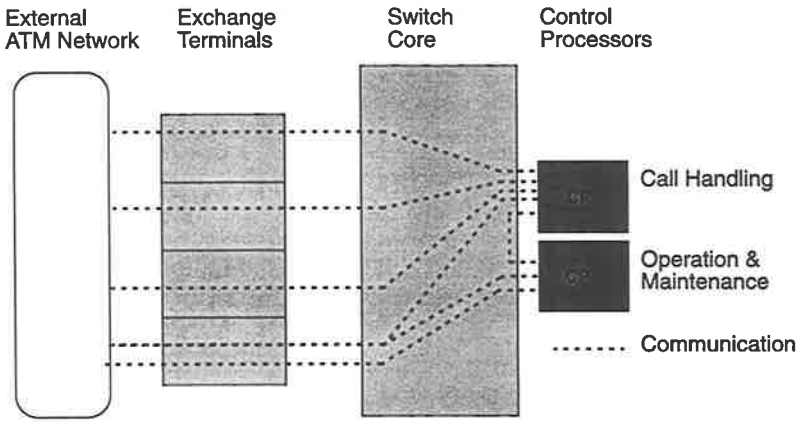
- Erlang Engine Prototype end 1997
- PhD for Framework end 1998
- Development of Paradigm

Contents

Runtime Architecture



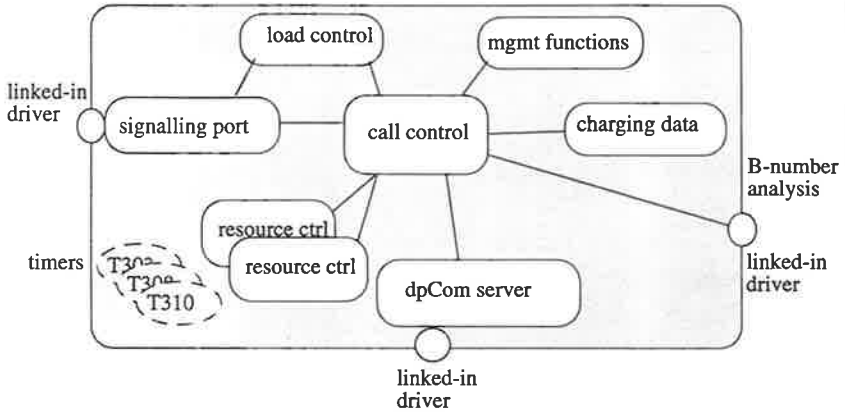
Basic Architecture



Performance Considerations

- *Up to 30,000 connections*
  – can't have one permanent process/call
- *RPC is costly*
  – centralize call handling
- *Errors must be confined within one call*
  – store state information in ETS tables
- *Switching between C and Erlang is costly*
- *All software must handle takeover with minimal loss of service*
- *Flow-control needed for communication*

Call-Control Architecture



Performance Progress

- *March '96: 250 ms*
  – BISDN prototype from P3A
- *May '96: 40 ms – Modified BISDN prototype*

(May '96: Start of major development)

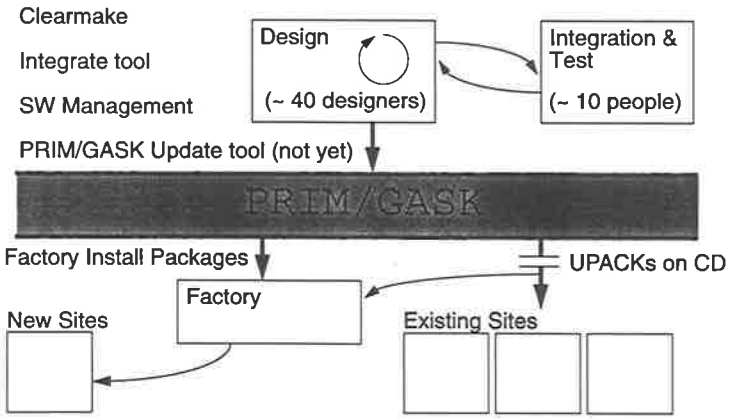
- *Oct '96: 65 ms (90 ms JAM)*
  – First bare-bones UNI implementation
- *Feb '97: 52 ms – Full-functionality UNI*
- *Aug '97 predicted: 35-38 ms*
- *Mid-'98 pred.: 5-10 ms on UltraSPARC 2i*

How to write fast code in Erlang

- *First make it work – then make it fast*
- *Model to reduce:*
  - *Context switching*
  - *Database updates*
  - *Data conversion/packing/unpacking*
- *Rapid feedback – educate designers*
- *Detailed profiling*
- *Rewrite - measure - rewrite*

Software Supply Flow

We have built tools to streamline testing, delivery and handling of our product line.



The Orber project

An Orb in Erlang

e

Orber project goals

"The goal of the project is to deliver an Object Request Broker environment in erlang compliant with the CORBA 2.0 specification 971201. "

e

Keywords

- Object Management Group, OMG
- Common Object Request Broker Architecture, CORBA
- Object Request Broker, ORB
- Interface Specification Language, IDL

What is OMG

- Object Management Group
- Microsoft, HP, IBM, Sun, Netscape, DEC, IONA and 700 more
- Developing "The Architecture for a Connected World."

What is CORBA

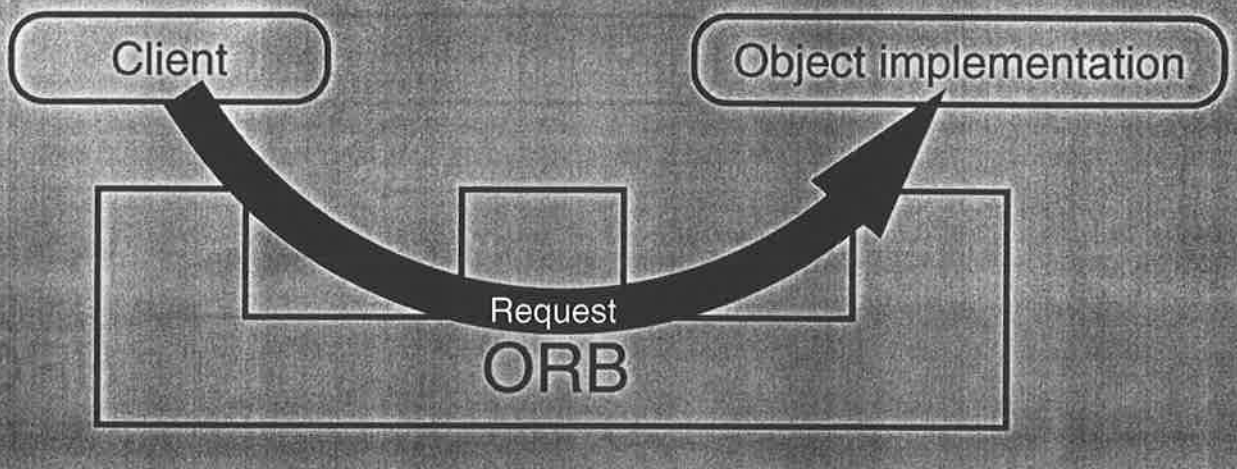
- Common Object Request Broker Architecture
- The distributed object architecture
- Object oriented RPC
- Distribution protocol (IIOP) and methods
- Platform and architecture and implementation language independent

e

What is an ORB

- Object Request Broker
- Software bus
- Handles object location transparency
- Relays object method invocations

e



e

The Erlang ORB

- Object adaptor for Erlang
- IIOP to other ORBs
- Interoperable naming service
- Tested with Orbix Web

e

What is IDL

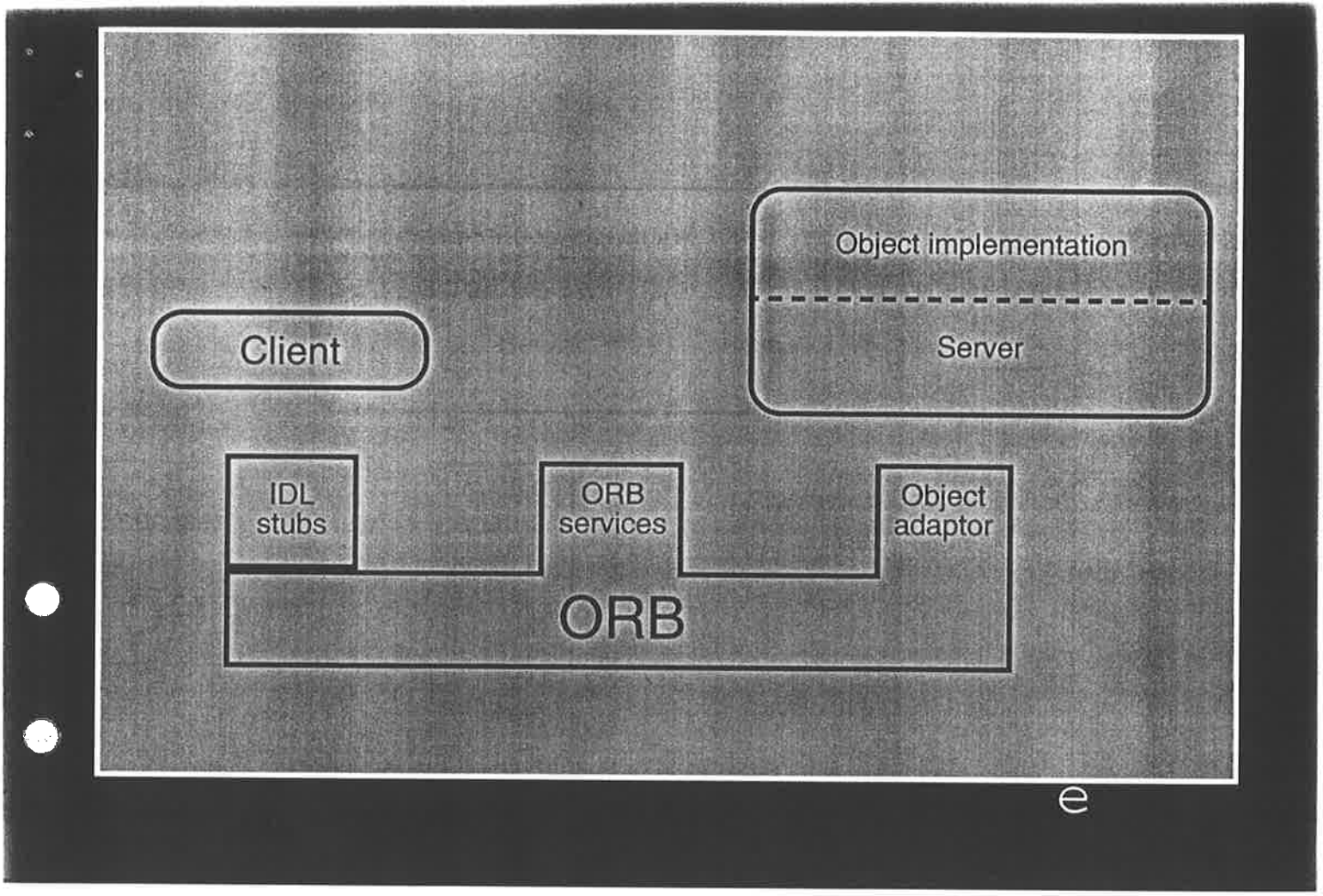
- Interface Definition Language
- Object interfaces defined in IDL
- Object oriented, modules, interfaces, inheritance, exceptions
- Platform and architecture and implementation language independent
- Standardized mappings to implementation languages, C, C++, Smalltalk

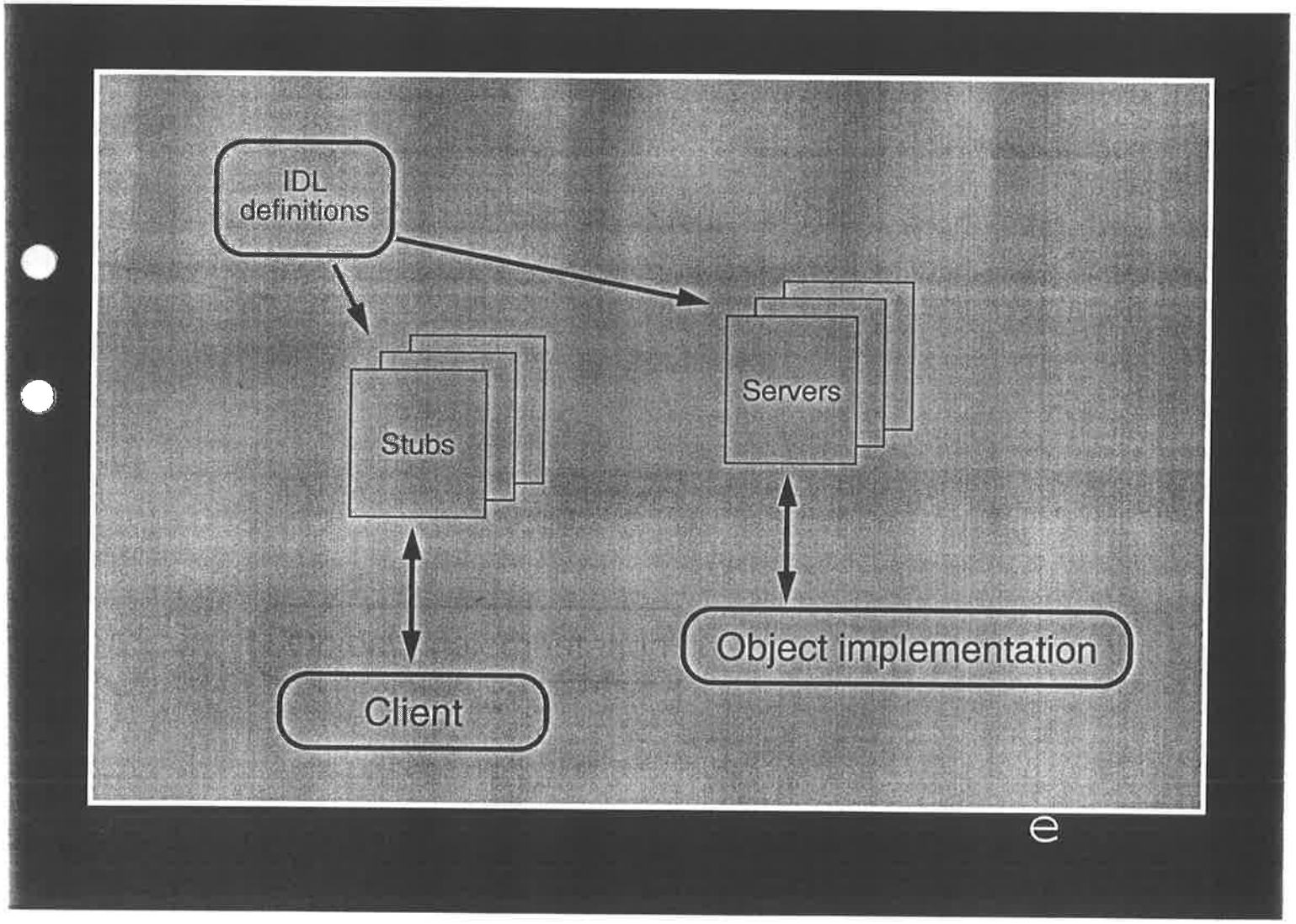
e

Object mapping

- Interface becomes generic server behaviour
- Operations becomes function calls
- Types becomes values
- Exceptions becomes return values
- Inheritance is expanded

e


Object implementation
Server
Client
IDL
stubs
ORB
services
Object
adaptor
ORB


IDL
definitions
Stubs
Servers
Client
Object implementation

Orber benefits

- Robust, fault tolerant Orb
- High availability
- Scalability
- Realtime characteristics
- Opens Erlang to many platforms in a uniform way

e

Orber project status, alpha

- Orb, including IIOP
- IDL to Erlang compiler
- Interface Repository, IFR
- Coming: naming and event services
- Tested with Orbix java, and jacorb

e

Orber project future

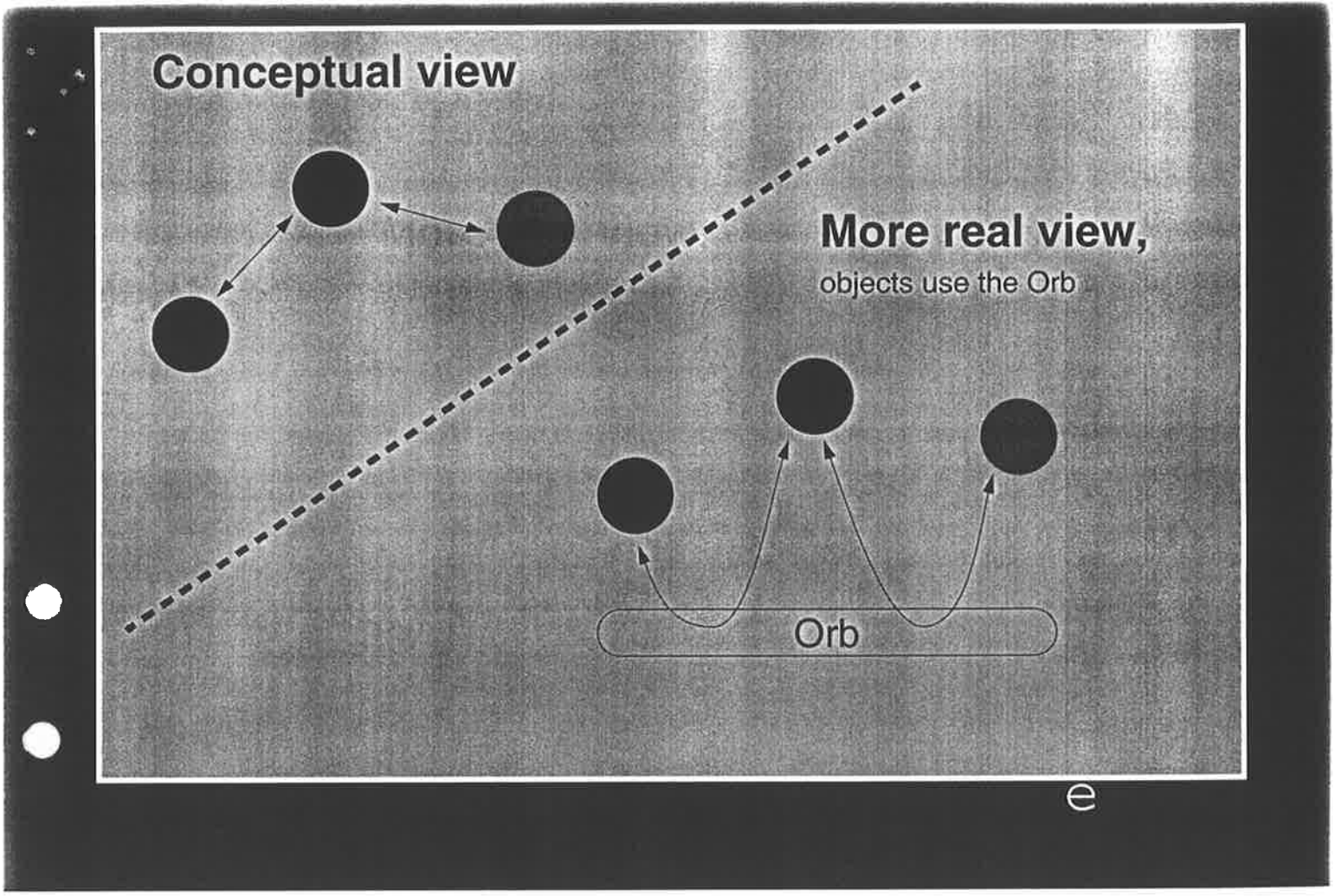
- More services, persistence, transaction
- Stubs and object adaptors for other languages
- Other object models

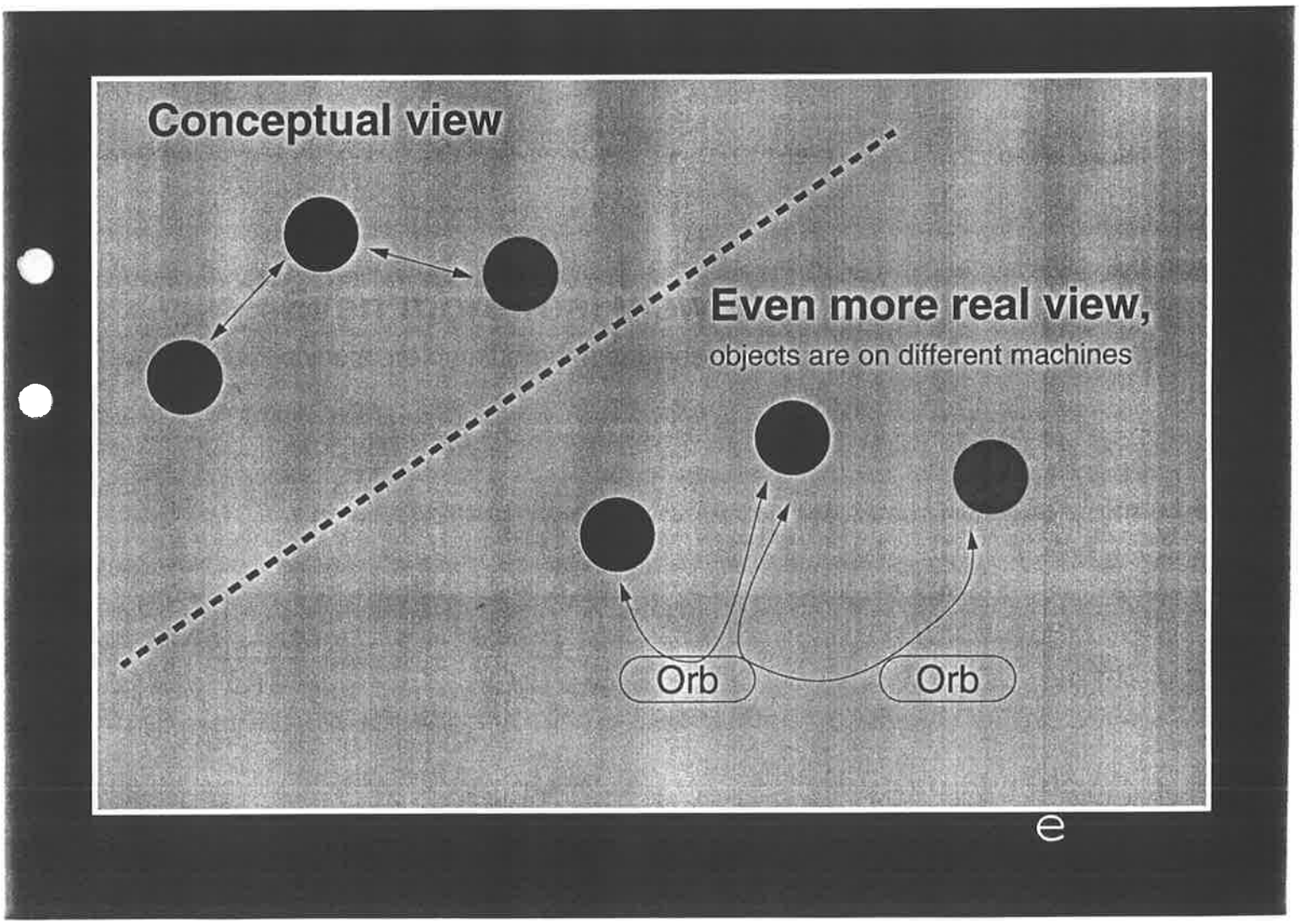
e

CORBA Services

- Naming service
- Event service
- Security
- Transaction

e


Conceptual view
More real view,
objects use the Orb
Orb
e


Conceptual view
Even more real view,
objects are on different machines
Orb
Orb
e

Presentation missing at time for producing precedings

Towards an Even Safer Erlang

1. ***Towards an Even Safer Erlang***
   - Dr Lawrie Brown
     - Australian Defence Force Academy

     Dr Dan Sahlin
     - Computer Science Labs, Ericsson Telecom
2. *Introduction*
   - want an even safer Erlang system
   - constrained and partitioned execution
   - ideally with
     - minimal (user) visible language changes
     - minimal (if any) impact on performance
   - currently proof-of-concept research
   - two prototypes - SafeErlang, SSErl
3. *Rationale*
   - mobile agents
     - server lets agent code migrate to it
   - applets
     - client requests applet load and run
   - outsourced code
     - run application but control access
   - fault isolation
     - isolate application components
4. *Limitations*
   - on a current Erlang system (node)
   - processes have same "world-view"
     - modules, names, resources, servers
   - pids/ports too powerful
     - forgeable
     - unrestricted control over resource
       - eg can kill any process on system
   - code always uses "local" context
5. *Safety Issues*
   - constrained & partitioned execution of code within an Erlang system
   - support for "remote" code loading and execution in context
   - security of links and data transfer between Erlang systems
6. *Safer Execution in Erlang*
   - language has intrinsic benefits
   - want custom world-view
     - implement a hierarchy of (sub)nodes
       - registered names
       - modules available
       - resource limits
   - need controlled access to processes and external resources (ports)
     - make these data types capabilities
7. *Nodes*
   - each Erlang system to support a hierarchy of (sub)nodes
   - provide a distinct context
     - registered names

        - control which servers are accessible
    - modules available
        - module name aliasing
            - support remote loading, safety renaming
    - resource limits on processes
        - partition & constrain use of system resources
8. *Capabilities*
    - unique, unforgeable resource name with associated rights to use
        - <Type,Node,Value,Rights,Private>
    - for nodes/pids/ports/user capas
    - possible implementations
        - crypto hash check, validated by node
        - password (sparse) key to table of valid capabilities managed by node
9. *SafeErlang*
    - Naesar & Sahlin, Uppsala
        - Masters project, late 1996
    - subnodes provide distinct context
        - custom modules, resource limits
    - capabilities control access
        - encrypted (must decrypt for all use)
        - nodes & pids only
    - remote module loading (mids)
        - for mobile agents
10. *SSErl*
    - Brown, ADFA
        - sabbatical research, 1997
    - subnodes provide distinct context
        - distinct names
        - module aliases
    - capabilities control access
        - nodes, pids, ports, user capabilities
        - both crypto hash & password
    - remote module load (coming)
11. *Remote Code Execution*
    - want support for "remote" code loading and execution in context
        - to support mobile agents, applets
    - module references in "remote" modules should be interpreted thus
        - should load requested module from remote system, not from local system
    - want code mobility whilst executing
        - several processes, upon request
12. *Other Issues*
    - link security
        - need encryption/authentication (SSL) on links between Erlang systems
    - registered names
        - local - name on local node
        - distributed - local name on given node
        - global - hierarchy of names not flat
    - module names and grouping
        - functions in modules in projects???
13. *Conclusions*
    - rationale & approach for a safer erlang execution environment

- to support mobile agents, applets, outsourced code, fault isolation
- a hierarchy of nodes & capabilities
- with minimal visible changes or performance impact
- aim to incorporate into ERTS

14. *Questions*
15. *Types of Capabilities*
    - need capability data in clear
        - for efficient guards/pattern matches
            - <Type,Node,Value,Rights,Private>
    - crypto hash check value
        - node has private key to compute/check
        - small vulnerability to search attacks
    - password (sparse) value
        - key into node table of valid capabilities
        - any guess must verify with node
16. *Performance Impact*
    - space
        - expect approx same no capas as pids/ports now (user capas are extra)
        - capas only a bit larger, not significant
        - password capas do need extra table
    - time
        - check value used on creation
        - can tag "local" capas as checked
        - "remote" capa use has comms delays
17. *Global Names*
    - need hierarchy of names and name servers for scaleability
    - suggest new "global" server which manages overlapping hierarchies
        - with parents, peers, children as well as own registered names
        - update all peers with changes
        - client responsible for redirects between servers to locate name
18. *Further Information*
    - see papers and software on web
        - http://www.adfa.oz.au/~lpb/TR/ssp97/
        - http://www.item.ntnu.no/~lpb/ssp97.html

Lawrie.Brown@adfa.oz.au / 04-Jul-97

The Erlang type system

Joe Armstrong

Computer Science Laboratory
Ericsson Telecommunications AB
Sweden
joe@cslab.ericsson.se

15 August 1997

Plan

- Credits.
- What is a type system?
- Types of type systems.
- The type notation.
- Subtyping.
- Examples.
- What else can the type system do?
- To do.

Credits

Phil Wadler
Was at Glasgow University now at inferno + ML groups at Lucent (was "unix" group at Bell-labs)

Simon Marlow
Glasgow University

Blame

Joe Armstrong

What is a type system?

- A type is a property of a program which can be proved.

Types of type system

- Untyped - assembler - no typing at all. Bad programs can dump core.
- Strong static typing - the pascal family, Java. All types must be declared. Types are checked at compile time. The type system cannot be broken. Program cannot dump core at run-time due to type errors. Usually monomorphic.
- Weak static typing - C etc. All types must be declared. Types are checked at compile time. The type system can be broken. programs can dump core.
- Dynamic typing. Erlang, prolog, lisp, smalltalk. No types are declared. Type checking is done at run-time. The type system cannot be broken. Programs cannot dump core due to type errors.
- Polymorphic strong typing. ML, haskell. Types may be provided. If the type is not provided it is inferred from the program. Types, if provided, are checked against the inferred types. It is impossible to break the type system.
- Soft typing. Erlang. As for polymorphic strong typing but type incorrect programs can be compiled.

Joe Armstrong
Ericsson Computer Science Laboratory

The type notation

Primitive types

- `integer()` - an integer - 123.
- `float()` - a real number - 3.14159.
- `string()` - a string - "hello world".
- `atom()` - an atom - hello.
- `Atom` - an atom.
- `pid()` - a pid.

Type constructors:

- `[X]` - a list of type `X`
- `{X,Y,Z}` - a tuple of arity three, with given argument types
- `Atom{X,Y}` - a tagged tuple (Type) where the first item is the atom `Atom`.
- `tuple()` - a tuple of (unknown) arity

Function Types

- `-type map((A) -> B, [A]) -> [B]`

Joe Armstrong
Ericsson Computer Science Laboratory

Subtyping

Suppose

```
X = [1,2,3],
Y = [a,b,c],
Z = X ++ Y.
```

What is the type of Z?

- `X` has type `[int()]`.
- `Y` has type `[atom()]`.
- `++` (i.e. append) has type
  `-type append([A], [A]) -> [A]`

Answer: `A when A <= int(), A <= atom().`

Example 1

Normal usage of the type system:

```
-module(ex1).
-export([xand/2]).

xand(true, true)   -> true;
xand(true, false)  -> false;
xand(false, true)  -> true;
xand(false, false) -> false.
```

Run the type checker

```
> tc_main:tc("ex1.erl").
...
```

And look at the results:

```
> cat ex1.types
-interface (ex1).

-type xand(false | true, false | true) ->
   false | true.
```

Example 2

Try to call something in ex1.erl:

```
-module(ex2).
-export([a/0]).

a() -> ex1:xand(true, false).
```

The result:

```
> cat ex2.types
-interface (ex2).

-type a() -> false | true.
```

Now call xand with bad arguments

```
-module(fail1).
-export([a/0]).

a() -> ex1:xand(tru, false).
```

Type checking fails

```
> tc_main:tc("fail1.erl").
fail1.erl: 5: type error in 1st argument
   of call to ex1:xand/2
inconsistent constraint:
tru <= false | true
error
```


Example 3

Correct and incorrect use of xand:

```
-module(ex3).
-export([a/0]).
a() ->
     case ex1:xand(true, false) of
          true  -> 1;
          false -> 2
     end.
-----------------------
-interface (ex3).
-type a() -> integer().
-----------------------
-module(fail2).
-export([a/0]).
a() ->
     case ex1:xand(true, false) of
          tru   -> 1;
          false -> 2
     end.

> tc_main:tc("fail2.erl").
fail2.erl: 7: type error in pattern match
inconsistent constraint:
false | true <= false | tru
```

Example 4

Types as documentation:

```
-module(ex4).
-export([xand/2]).

-type xand(bool(), bool()) -> bool().

xand(true, true) -> true;
xand(true, false) -> false;
xand(false, true) -> true;
xand(false, false) -> false.
--------------------
-interface (ex4).

-type xand(bool(), bool()) -> bool().
```

Joe Armstrong
Ericsson Computer Science Laboratory

Example 5

Types as verification

```
-module(fail3).
-export([xand/2]).

-type xand(bool(), bool()) -> bool().

xand(tru, true) -> true;
xand(true, false) -> false;
xand(false, true) -> true;
xand(false, false) -> false.

> tc_main:tc("fail3.erl").
fail3.erl: 5: signature for xand/2
  is not an instance of the inferred type
inconsistent constraint:
false <= true
```

Joe Armstrong
Ericsson Computer Science Laboratory

Example 6

Adding types declaration is a good idea, because the following program is well-typed, but does not have the type we expected:

```
-module(ex5).

-export([xand/2]).

xand(tru, true) -> true;
xand(true, false) -> false;
xand(false, true) -> true;
xand(false, false) -> false.

-----------------------------
-interface (ex5).

-type xand(false | tru | true, ())
      -> false | true.
-----------------------------
```

Joe Armstrong
Ericsson Computer Science Laboratory

What else can the type system do?

- Enforce abstraction boundaries
- Verify that programs are type-correct at compile time (i.e. you get no run-time type errors)
- `-deftype bool() = true | false`
- `-exportdeftype([t/N]).`
- `-unchecked_type foo Type -> Type`
- Provide formal documentation of functionality

Joe Armstrong
Ericsson Computer Science Laboratory

To do

- Cannot check all code (`-unchecked_type ...`).
- There are some existing Erlang programs (c. 10 %) which run correctly and produce the expected results but which are not well-typed. This code must be re-written to get it through the type checker, or annotated with **`unchecked`** declarations.
- Cannot check some generic code, but this is not a problem.
- Is slow.
- Difficult to understand error messages.
- No course material.
- Is a prototype.

Joe Armstrong
Ericsson Computer Science Laboratory

Etos: an Erlang to Scheme compiler

Marc Feeley and Martin Larose

Université de Montréal

Motivations

- Erlang and Scheme have common features
  - functional
  - dynamic typing
  - data types & GC
- Scheme has good compilers
  - pseudoknot: 5 times faster
- Several Scheme compilers available
  - Gambit, Bigloo, Chez-Scheme, Stalin

Summary

- Portability vs. efficiency
- Data types
- Binding and pattern matching
- Limitations
- Performance comparison
- Future work

Portability vs. efficiency

Etos:

- Written in Standard Scheme
- Generates almost standard Scheme code
  - macro definition file to exploit Scheme implementation's non-standard features

Direct translation when possible

- Interfacing Erlang, Scheme and language extensions
- Easy to debug
- Fair comparison between Erlang and Scheme compilers

Data types I

Erlang	Scheme
integer	exact integer
float	inexact real
atom	symbol
list	list
tuple	vector
function	procedure

Binding and pattern matching

Example: `[X|Y] = foo:f(A), X+bar:g(Y)`

```
(let ((v7 ^a))
  (let ((v5 (foo:f/1 v7)))
    (let ((v6 v5))
      (if (erl-cons? v6)
        (let ((^x (erl-hd v6)))
          (let ((^y (erl-tl v6)))
            (let ((v1 v5))
              (let ((v2 ^x))
                (let ((v4 ^y))
                  (let ((v3 (bar:g/1 v4)))
                    (erl-add v2 v3)))))))
        (erl-exit-badmatch)))))
```

```
(let ((v5 (foo:f/1 ^a)))
  (if (erl-cons? v5)
    (erl-add (erl-hd v5)
             (bar:g/1 (erl-tl v5)))
    (erl-exit-badmatch))))))
```

Limitations

- Macros, records, ports and binaries
- Process registry and dictionnary
- Dynamic code loading
- Built-in functions and libraries
- Distribution

Benchmark programs

- integer arithmetic
  - `fib`, `huff`, `length`, `smith`, `tak`
- floating point arithmetic
  - `barnes`, `pseudoknot`
- list processing
  - `nrev`, `qsort`
- processes
  - `ring`, `stable`

Compilers

- Hipe 0.27
- BEAM/C 4.5.2
- JAM 4.4.1
- Etos 1.4 and Gambit-C 2.7a
  - generates portable C code
  - extensions to Scheme standard (includes a C-interface)
  - function inlining, float unboxing, bound interrupt checks
  - stop & copy GC, dynamic heap resizing
  - efficient `call/cc`

Results I

⇒ Sun UltraSparc 143 MHz with 122 Mb

Program	Etos (secs)	Time relative to Etos		
		Hipe	BEAM	JAM
`fib`	31.46	1.16	–	8.33
`huff`	9.94	1.45	4.43	24.75
`length`	11.54	2.08	3.36	34.61
`smith`	11.47	2.02	2.63	12.07
`tak`	13.27	1.12	–	13.33
`barnes`	15.91	1.18	–	2.24
`pseudoknot`	20.66	1.83	–	2.58
`nrev`	24.81	.78	–	8.21
`qsort`	16.32	.87	–	14.85
`ring`	124.58	.30	.29	1.71
`stable`	21.40	1.13	–	1.79

Results II

- Integer arithmetic
  - up to 2 times faster than Hipe
- Floating point arithmetic
  - 1.18 to 1.83 times faster than Hipe
- List processing
  - space usage (3 words per pair)
  - interrupt checks
  - tail-recursion in C
- Processes
  - intermodule calls and `call/cc` interface

Future work

- Full compliance to Erlang 5.0
- Access to Gambit-C C-interface
- Gambit-C tuning for Etos' code generation
- Native code back-end for Gambit
  - factor of 2 performance boost
- Hard real time and generational garbage collector

Conclusions

- Outperforms all other compilers on most benchmarks
- Process management needs improvement
- Scheme is well suited as a target language for Erlang

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

Iustitia - Erlang Based Load Balancing Experiments

Sasa Desic
Zrinko Kolovrat
Ignac Lovrek

Department of Telecommunications
Faculty of Electrical Engineering and Computing
University of Zagreb, Croatia
HR-10000 Zagreb, Unska 3
tel: +385 1 612 97 51
fax: +385 1 612 98 32
e-mail: sasa.desic@fer.hr

Erlang Users Conference
Kista, August 26th, 1997

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

Overview

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

Introduction

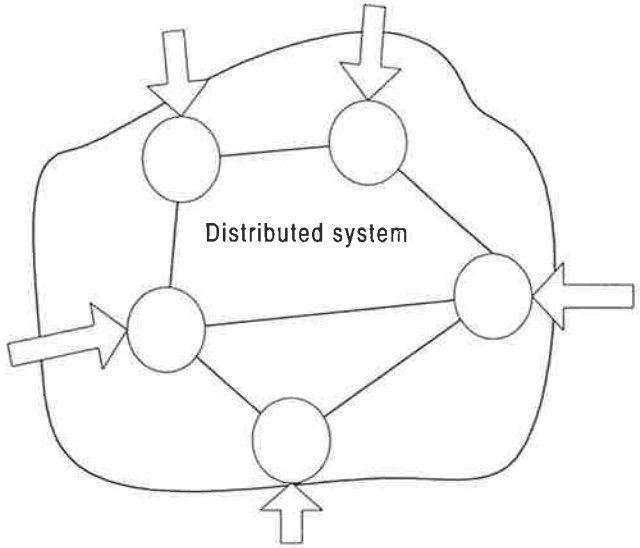
Place - Distributed system

Problem - Request rejection because node overloading

Solution - Load balancing !

Experiments - Erlang based package for load balancing simulation

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia



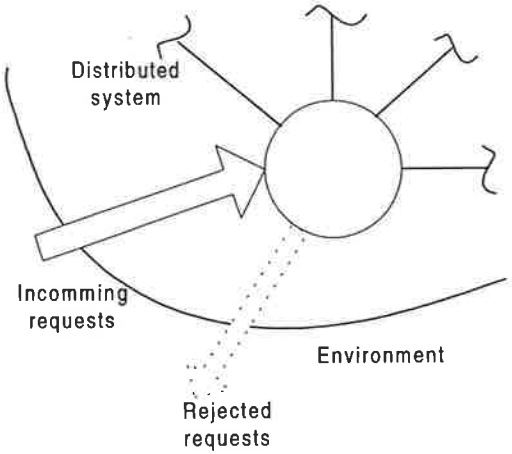
Distributed system - network of processing and communicating nodes

Processing requests arrives from the outside world in independent streams

Call and service environment represents distributed system load

Stochastic nature of calls and services causes network fluctuations and imbalances of load

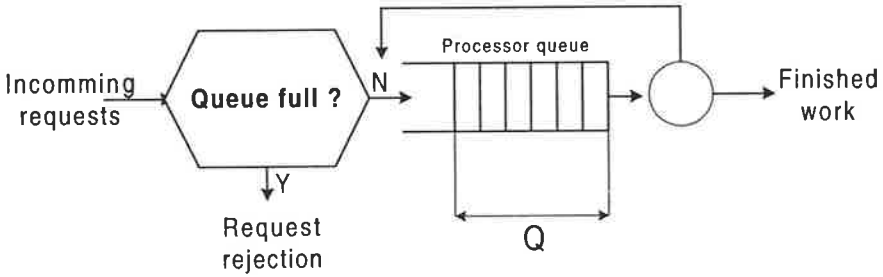
Problem Statement (2/2)



Imbalances of load causes situation where some nodes are below capacity while others are simultaneously overloaded

Part of incoming requests will be rejected because processor's overloading

We consider problem of load sharing, load balancing and increasing processing capabilities



Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

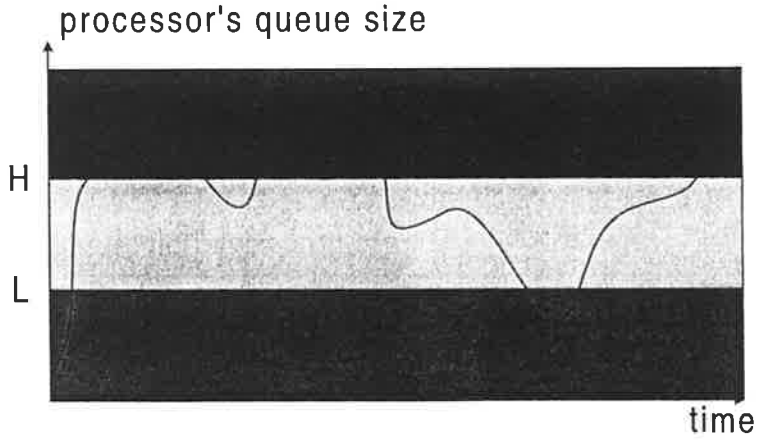
Load Balancing (1/2)

Load Balancing - moving of load from overloaded node (processor) to underloaded one

S_i - system node
Q_i - queue size on node S_i
L and H - system parameters

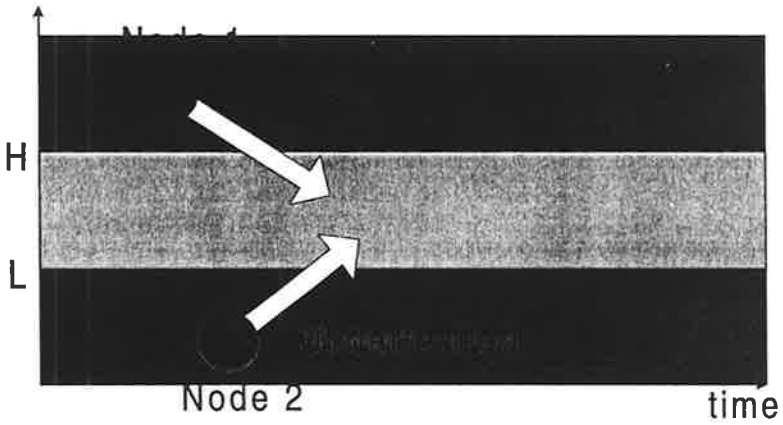
Node statuses are defined as follows:

1. A site S_i is underloaded, if $Q_i < L$
2. A site S_i is overloaded, if $Q_i > H$
3. A site S_i is normal, if $L < Q_i < H$



Load Balancing (2/2)

Possibility for load balancing - when there is at least one underloaded and one overloaded node



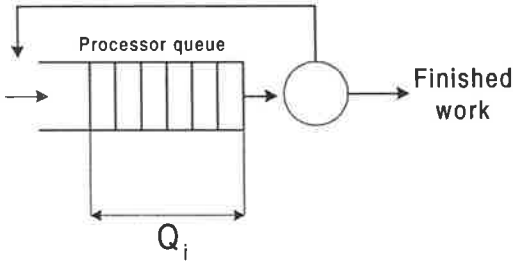
Aim of load balancing - fair distribution of load through all system nodes

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

Distributed Erlang Features

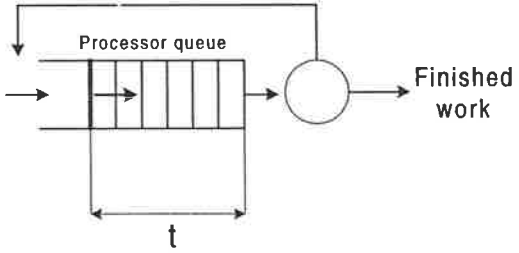
1. Queue size measuring

Erlang:
statistics (runqueue)



2. Measuring of queue execution time

Erlang:

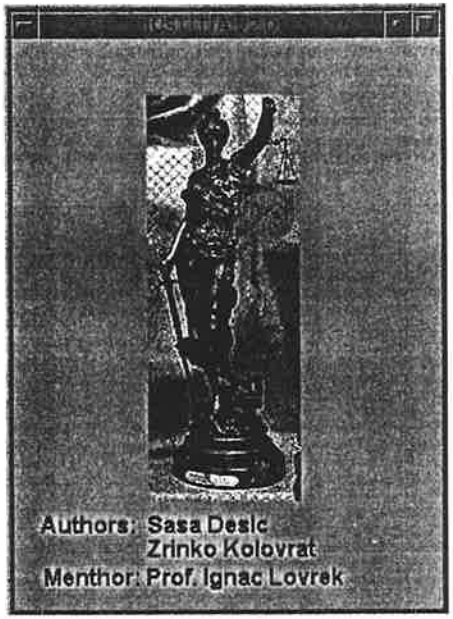



```
utilization()->

{Start,_}=statistics(runtime),
   End= utilization(500),
   Start-End.

utilization(0)->
   {End,_}=statistics(runtime),
   End;
utilization(N)->
      utilization(N-1).
```

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Basic Characteristics (1/4)

■ Iustitia - Load balancing experiment



■ Distributed run-time system simulation

14 modules
1200 lines of code
GS module for graphical presentation

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Basic Characteristics (2/4)

- Built-in load balancing mechanisms (self-balancing)

- Artificial workload simulation (stochastic environment simulation)

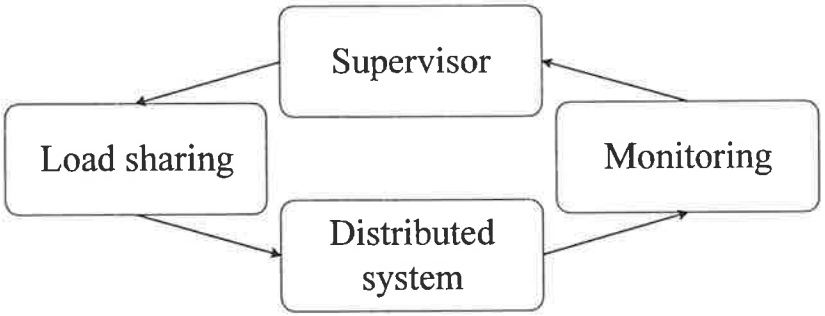
- Scheduling engine is:

  - *global* (we have the problem of deciding **where** to execute a arrived task)
  - *dynamic* (assignment decisions are made in run-time)
  - *physically distributed* (the work involved in making decision should be physically distributed among the nodes)
  - *cooperative* (each processor has the responsibility to carry out its own portion of the scheduling task, but all processors are working toward a common system wide goal)
  - *source-initiative* (the node where job is arrived, according the collected information, decides where to send the job or to execute locally)

- Fault tolerant system with self-recovery functions

☺ Graphical user interface with on-line monitoring

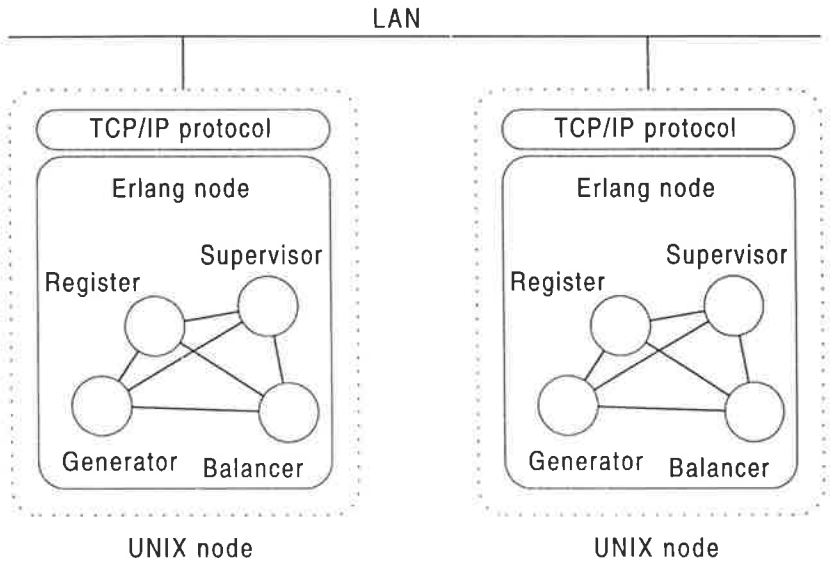
IUSTITIA - Basic Characteristics (3/4)



- Processes as the main unit of model's structure
- Global and Local Supervision process
- Global Supervisor (keeping the whole system working property, administrating purposes)
- Local Supervisor (organizing and monitoring processes on the local level)
- Register process (node bookkeeper, queue size measuring)
- Balancer process (heart of balancing engine)
- Generator process (synthetic workload generation)
- Graphics process (graphical user interface)

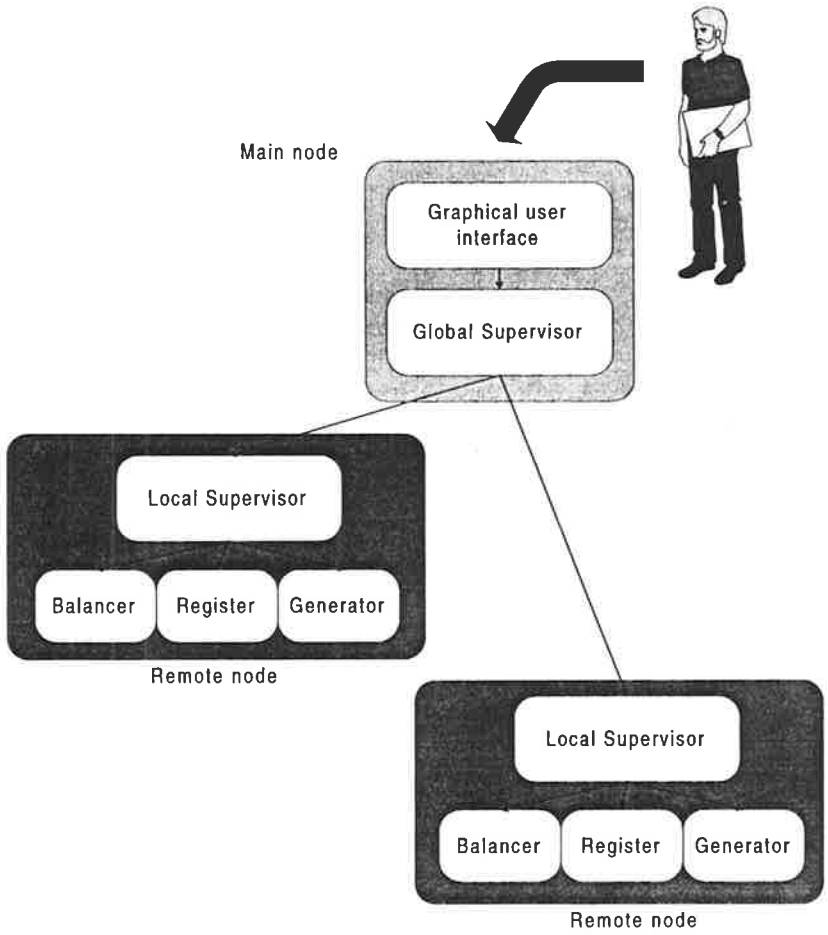
Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Basic Characteristics (4/4)



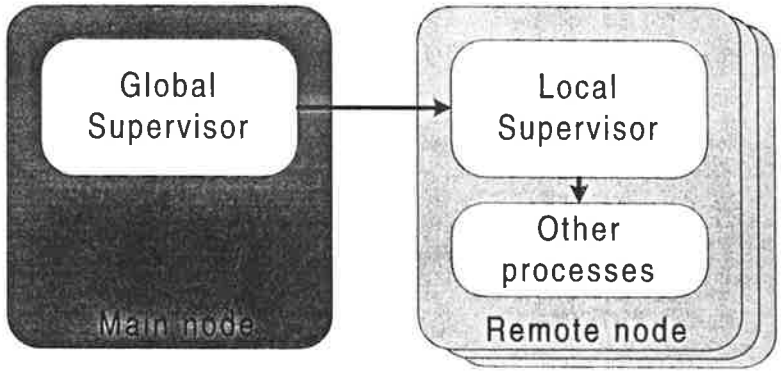
Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Hierarchical Organization



Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Process Supervisor

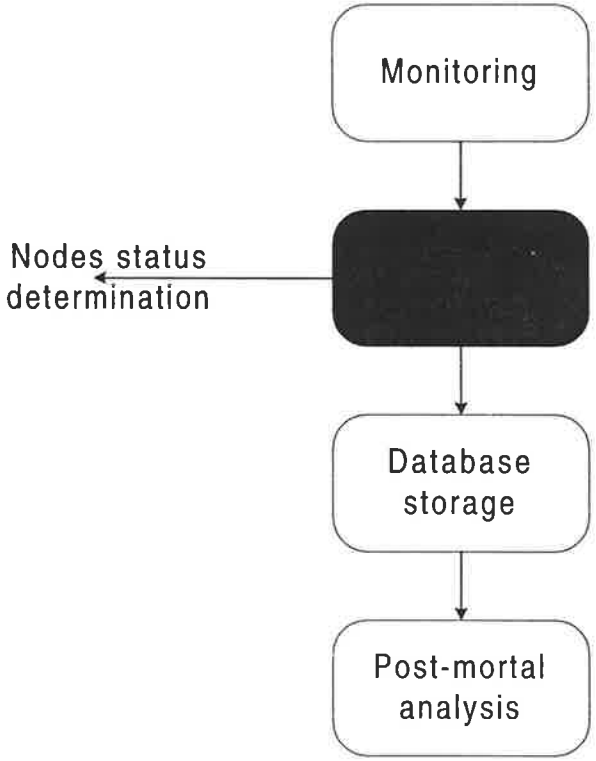


- System starts from main node (Graphics and Global Supervisor process)
- Global Supervisor starts Local Supervisors processes on remote nodes
- Local Supervisor starts other processes on the local level (Register, Balancer, Generator)

- User controls system through the main node
- Global Supervisor watches Local Supervisors processes (fault detection and self recovery)
- Local Supervisor watches other processes on the local level

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

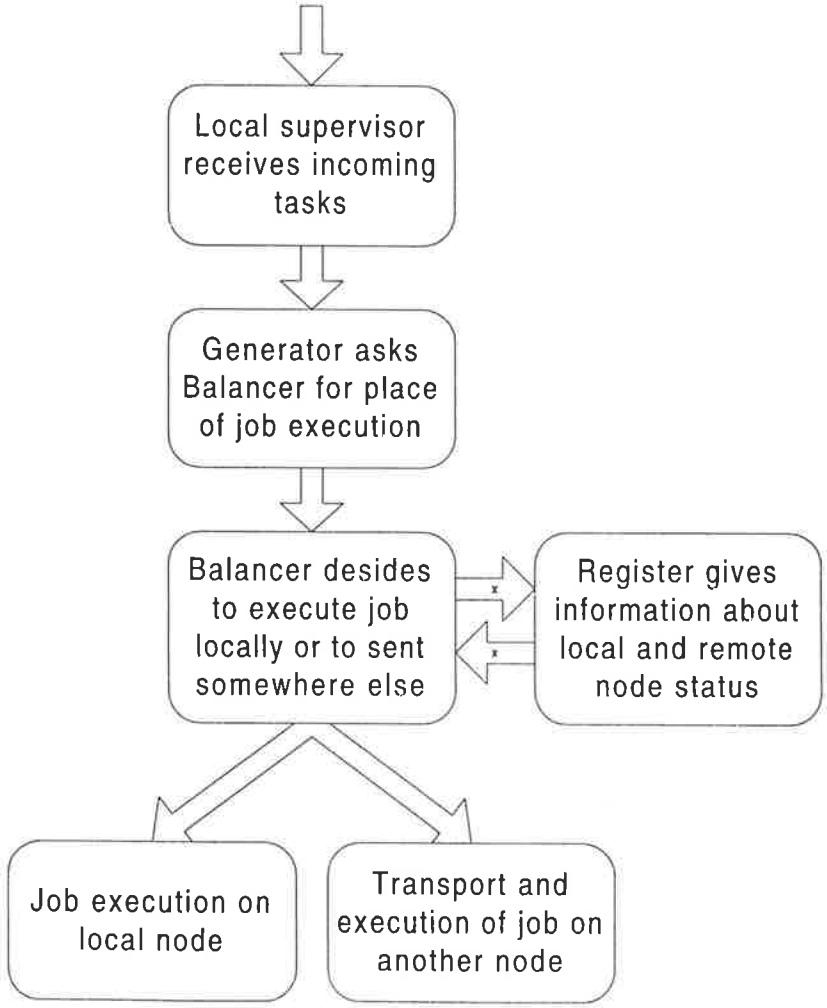
IUSTITIA - Process Generator



- Process responsible for tracking nodes current load
- Each 750 ms checking of queue size (represents workload)
- Results from tracking are saving to a file and could be used for latter analysis

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Process Balancer (1/2)



Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

FOCUSED ADDRESSING

In this approach each site keeps state information about other sites. When a newly arriving process enters the system at a local site, the local site queries its information and may immediately select a site for transferring the process.

SENDER DIRECTED

When a site has more processes than it can handle it becomes a source of work. To reduce its workload it may randomly select a site to send a work, or perhapes sequentially select a site o send work. The key idea behind this approach is the lack of interaction between the source and system.

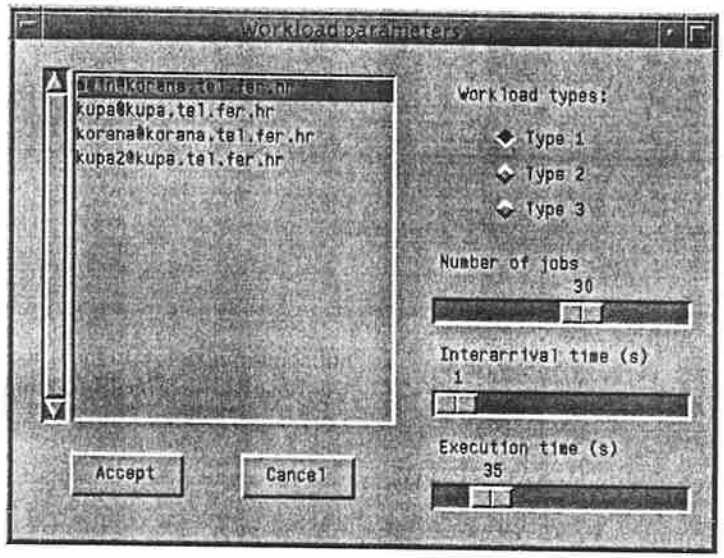
RECEIVER DIRECTED

The server when idle, solicits work from system.

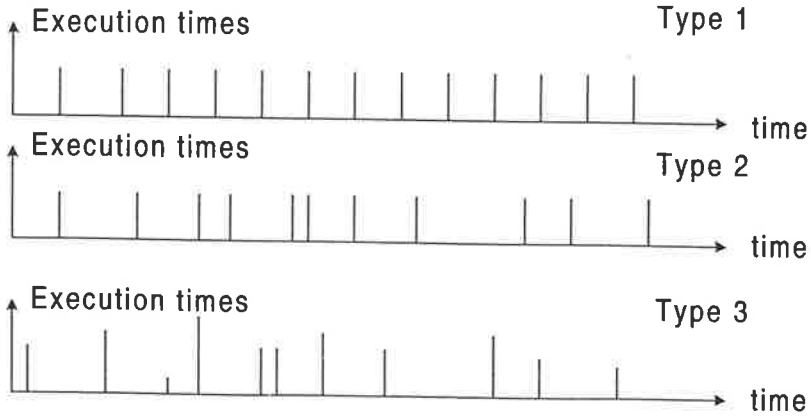
BIDDING

In this approach local site determines if processes sholud become candidates for movement. At the local site, a request for bids is sent out to remote sites. These remote sites bid on the candidate and return the bid to the requsting site. The bid is evaluated and the candidate processes is either kept or transferred base on evaluation.

Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

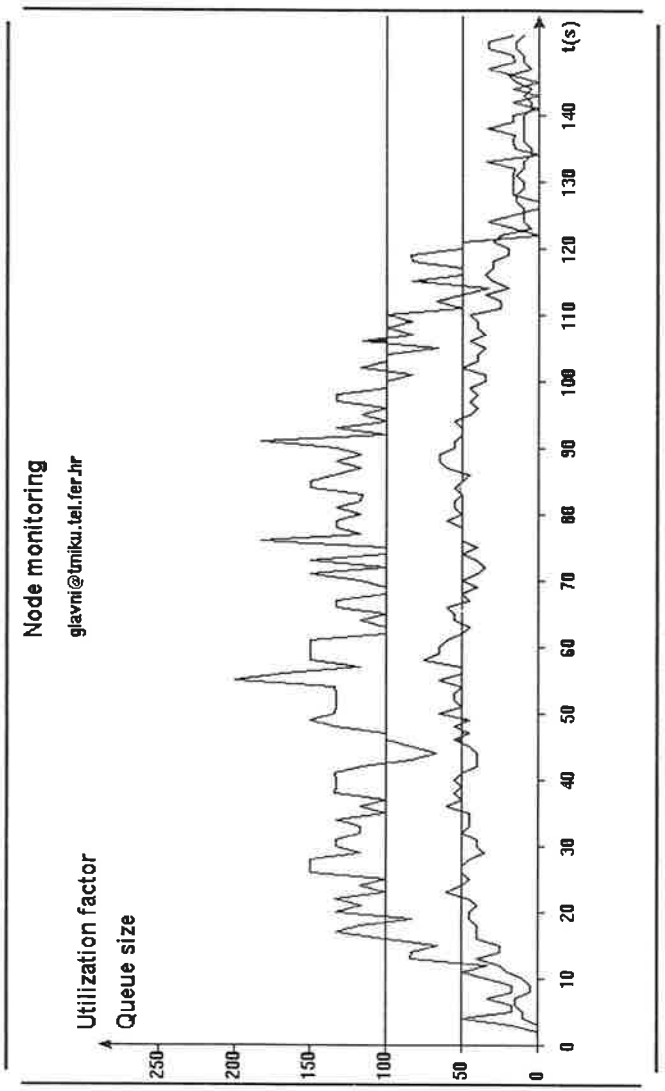


- Artificial workload generation
- Call and service environment simulation
- Three types of workload



Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

IUSTITIA - Monitoring



Department of Telecommunications
Faculty of Electrical Engineering
University of Zagreb, Croatia

Erlang experience

- it's easy to build distributed system
  - adding and removing node from the system
- concurrent processes
  - useful for building independence part of system
  - simultaneously information processing
- capability of handling queue size (queue size is the main measure in our program)
- gs module - very easy to build graphical interface (comparing with pxv module)

Erlang on multiprocessor systems

Pekka Hedqvist
Erlang Systems

Introduction

- Basic architecture of parallel Erlang. Current results and future.
- Prototype developed at Ericsson CS laboratory by ***Tony Rogvall*** (CS lab) and ***Pekka Hedqvist*** (Erlang Systems).

Motives

- Erlang processes inherently parallel
- Parallel hardware becoming cheaper and more common, both high and low end.
- Tele and datacom applications often has simple/natural parallelism.
- Needed (?) if Erlang is to be used in large and scaleable high-performance systems.
- "More" real-time through native OS threads

Topics

- Parallel hardware
- Erlang runtime systems
- Design choices
- Current implementation
- Results
- Questions

Hardware

- All (?) new commercial MP systems provides one logical address space - in some way.
- New CPUs allow relatively cheap 4 CPU board designs.
- Large systems can be created by interconnecting MP boards .
- Mainly driven by server/database market.

Erlang runtime systems

JAM/BEAM	VEE[2]
■ Separate process heaps.	■ Common heap for all processes.
■ Cost of passing messages grows with message size.	■ Cheap message passing.
■ "Real-time" through "distributed" stop&copy GC.	■ "Real-time" through more complex incremental generation GC.
■ Simpler to rewrite to MT safe.	■ Hard (redesign) to make MT safe.

Design Choices

Separate heaps

- Simpler to implement and debug.
- GC not a bottleneck
- Scales (?) better

Unified heap

- Hard to implement.
- GC may be a bottleneck.
- Does not (?) scale so well.

Implementation

- Redesign of process interaction (signals and messages)
- Make runtime system MT safe.
- Rewrite runtime system to make minimal use of C stack.
- Redesign of I/O system.
- Mixed scheduling.
- New timer design

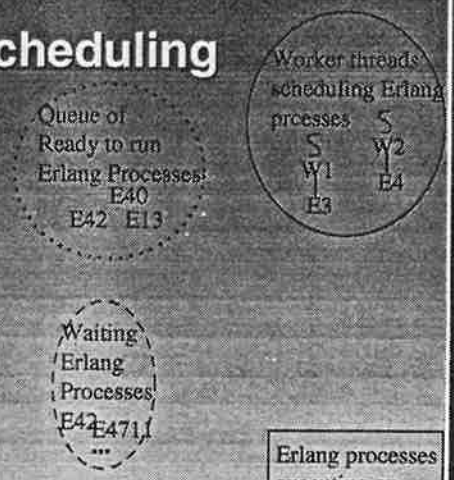
Messages and Signals

Process A sends a message to Process B .

A places a copy of the message in global heap memory and passes a pointer to B. The message becomes a temporary heap of B and disappear after B GCs.

Mixed Scheduling

- An Erlang process can execute on a private native thread or be scheduled by a worker.

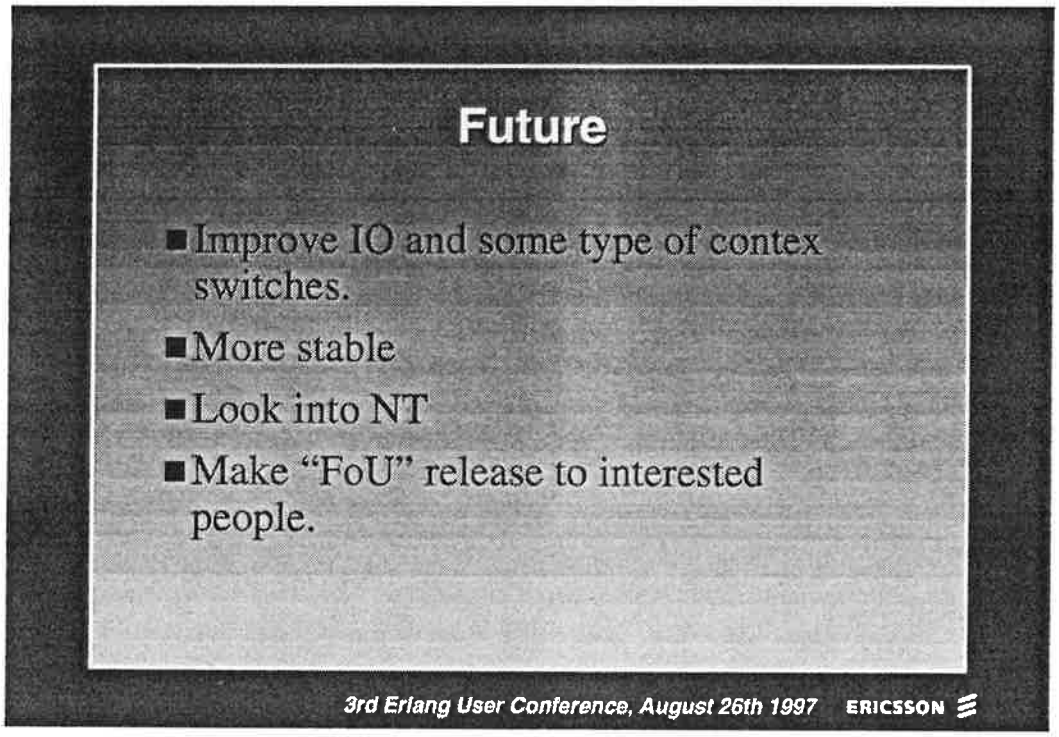


Different parallelisms

- Simple coarser parallelism presumed mostly needed, high throughput, load-tolerance, scaleable.
- Finer grain parallelism. Faster execution time - nice but not primary.

Current results on MP machine

- Positive
  - Fairly stable
  - Independent Erlang Process execution times unchanged when other executes (of course).
  - Scales well on some parallel programs (mandelbrot, pipeline problems etc)
  - "POSIX thread implementation.
- Negative
  - Slower contex switch, sometimes much slower.
  - I/O system slow on small bursty traffic.
  - No NT work done.


Future
■ Improve IO and some type of contex switches.
■ More stable
■ Look into NT
■ Make "FoU" release to interested people.
3rd Erlang User Conference, August 26th 1997
ERICSSON