

Erlang

Joe Armstrong

joe.armstrong@ericsson.com

Who is Joe?

Inventor of Erlang, UBF, **Open Floppy Grid**

Chief designer of OTP

Founder of the company Bluetail

Currently

Software Architect

Ericsson

Current Interests

Concurrency oriented programming

Peer-to-peer architectures

Grid computing

Functional programming

The world is **concurrent**
but we program in
sequential languages

This is

AMAZINGLY

DIFFICULT

but
if we program
in a concurrent language
it
becomes

Really

Easy

Concurrency Oriented programming

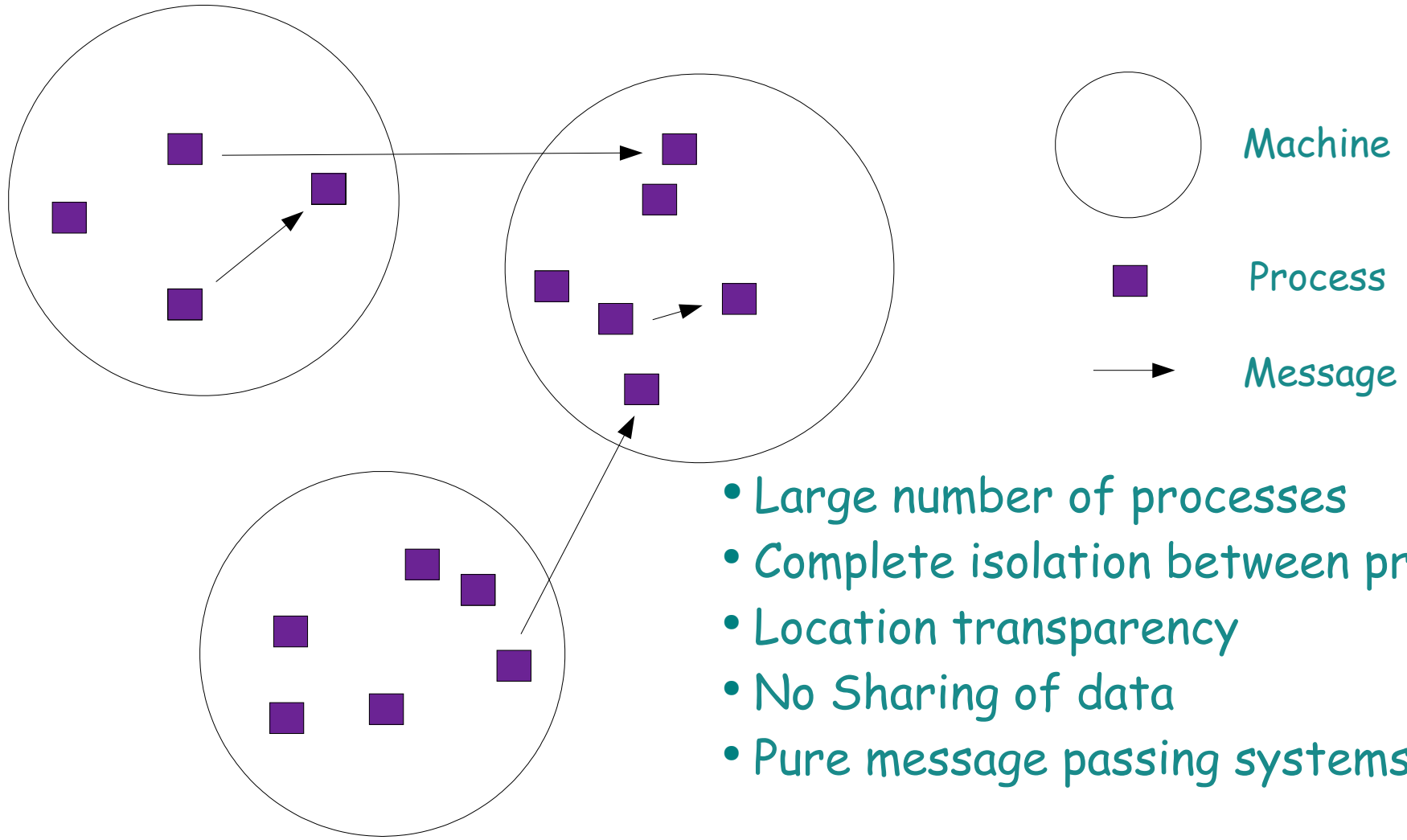
A style of programming where **concurrency** is used to structure the application

- Large numbers of processes
- Complete isolation of processes
- No sharing of data
- Location transparency
- Pure message passing

"My first message is that concurrency is best regarded as a program structuring principle."

Structured concurrent programming - Tony Hoare
Redmond July 2001

What is COP?



Why is COP nice?

- We intuitively understand concurrency
- The world is parallel
- The world is distributed
- Making a real-world application is based on **observation** of the concurrency patterns and message channels in the application
- Easy to make scalable, distributed applications

What is Erlang/OTP?

- Erlang = Concurrent Programming Language with a functional core
- OTP = a set of library routines callable by Erlang for writing fault-tolerant distributed applications

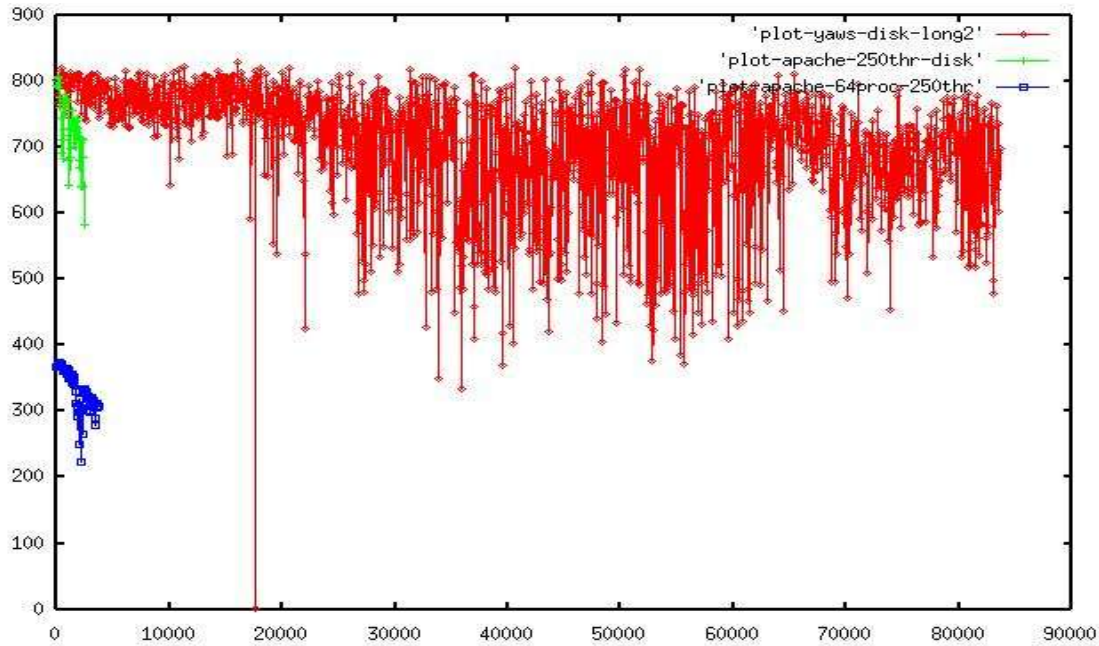
What is Erlang good for?

Programming distributed applications

Programming fault-tolerant applications

Programming applications with large numbers of parallel processes

Web server



Plot of throughput (in 100 Kbytes/Sec) vs. # parallel connections in simulated denial of service attack

- Red = YAWS (NFS)
- Green = Apache (NFS); Blue=Apache(local disk)
- Apache dies at 4000 parallel sessions
- See <http://www.sics.se/~joe/apachevsyaws.html>

Once upon a time ...

- 1986 - POTS Erlang (in Prolog)
- 1987 - ACS/Dunder
- 1988 - Erlang -> Strand (fails)
- 1989 - JAM (Joe's abstract machine)
- 1990 - Erlang syntax changes (70x faster)
- 1991 - Distribution
- 1992 - Mobility Server
- 1993 - Erlang Systems AB
- 1995 - AXE-N collapses. AXD starts
- 1996 - OTP starts
- 1998 - AXD deployed. Erlang Banned. Open Source Erlang.
Bluetail formed
- 1999 - BMR sold
- 2000 - Alteon buys Bluetail. Nortel buys Alteon
- 2002 - UBF. Concurrency Oriented Programming
- 2003 - Ph.D. Thesis - Making reliable systems

History

- Erlang is an Ericsson **secret**
- Erlang is too slow
- Erlang is **banned**
- Erlang **escapes** (Open source)
- Erlang **infects** Nortel
- ...
- Erlang controls the world

Why should I learn yet another programming language?

Because it's fun

Because we can write beautiful program in it

Because my boss told me to

Because we can develop products quicker in it

Because it solves certain technical problems

If Erlang is the solution
what is the problem?

How do we make reliable systems from
components which fail?

How do we correct hardware failures?

Replicate the hardware

How do we correct software errors?

Having two identical copies of the software
won't work - both will fail at the same time
and for the same reason

Why does your computer crash?

Which fails more often, hardware or software?

Problem domain

Highly concurrent (hundreds of thousands of parallel activities)

Real time

Distributed

High Availability (down times of minutes/year - never down)

Complex software (million of lines of code)

Continuous operation (years)

Continuous evolution

In service upgrade

How do we make systems?



Systems are made of black boxes (components)

Black boxes execute concurrently

Black boxes communicate

How the black box works internally is irrelevant

Failures inside one black box should not crash another black box

System requirements

R1. Concurrency

processes

R2. Error encapsulation

isolation

R3. Fault detection

what failed

R4. Fault identification

why it failed

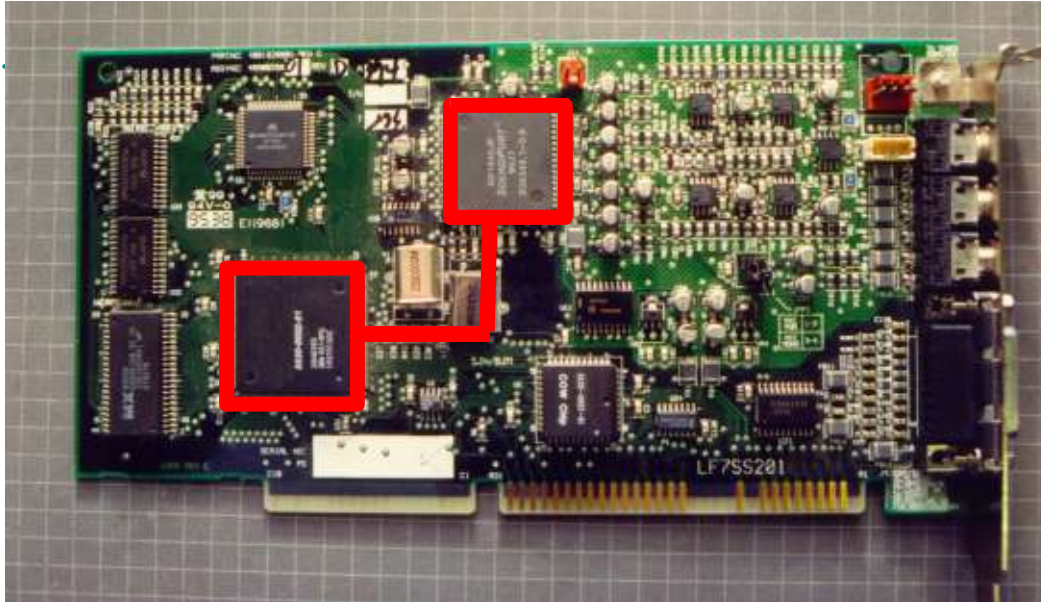
R5. Live code upgrade

evolving systems

R6. Stable storage

crash recovery

Isolation



Hardware components operate concurrently are isolated and communicate by message passing

Consequences of Isolation

Processes have **share nothing** semantics and data must be copied

Message passing is the only way to exchange data

Message passing is asynchronous

GOOD STUFF

Processes

Copying

Message passing



BAD STUFF

Threads

Sharing

Mutexes

Synchronized methods



Language

My program should not be able to crash your program

Need strong isolation and concurrency

Processes are OK - threads are not (threads have shared resources)

Can't use OS processes (Heavy - semantics depends on OS)

Isolation

My program should not be able to crash your program.

This is the single most important property that a system component must have

All things are not equally important

Java doesn't work

...The only safe way to execute multiple applications, written in the Java programming language, on the same computer is to use a separate JVM for each of them, and to execute each JVM in a separate OS process. This introduces various inefficiencies in resource utilization, which downgrades performance, scalability, and application startup time. The benefits the language can offer are thus reduced mainly to portability and improved programmer productivity. Granted these are important, but the full potential of language-provided safety is not realized. Instead there exists a curious distinction between ``language safety,`` and ``real safety``.

... tasks cannot directly share objects, and that the only way for tasks to communicate is to use standard, copying communication mechanisms

- Czajkowski, and Daynes, Sun Microsystems

Nor does C and C++

No processes (OS has processes but not C or C++)

Terrible isolation pointers etc.

Non-portable (word sizes, big-/little-endian problems)

No GC

But Erlang works



Lightweight processes (lighter than OS threads)

Good isolation (not perfect yet ...)

Programs never lose control

Error detection primitives

Reason for failure is known

Exceptions

Garbage collected memory

Lots of processes

Functional



Agner Krarup Erlang (1878-1929)

Erlang

You can create a parallel process

```
Pid = spawn(fun() -> ... end).
```

then send it a message

```
Pid ! Msg
```

and then wait for a reply

```
receive
```

```
  {Pid, Rely} ->
```

```
    Actions
```

```
end
```

*It typically takes 1 microsecond to
create a process or send a message*

*Processes are
isolated*

Programming for Failure

Let it crash

If you can't do what you are supposed to do crash
as soon as possible - don't make matters worse by trying
to fix the errors.

Let some other process fix the error

Let some other process fix
the error

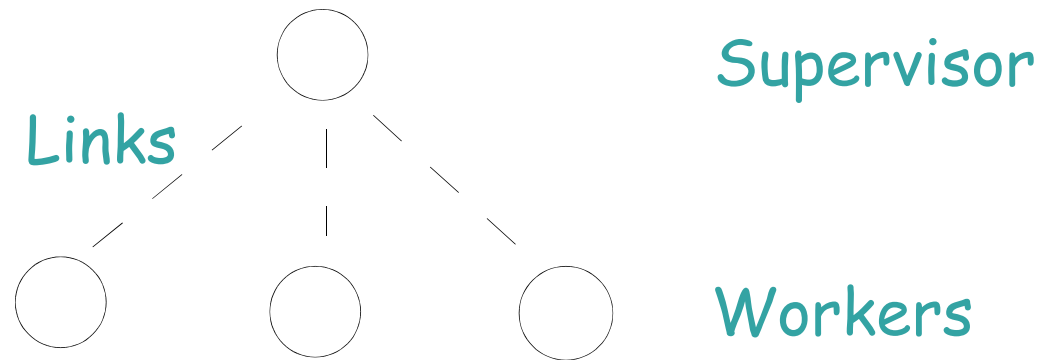
To do fault-tolerant
computing you
need at least
TWO computers



Which means you
can't share data

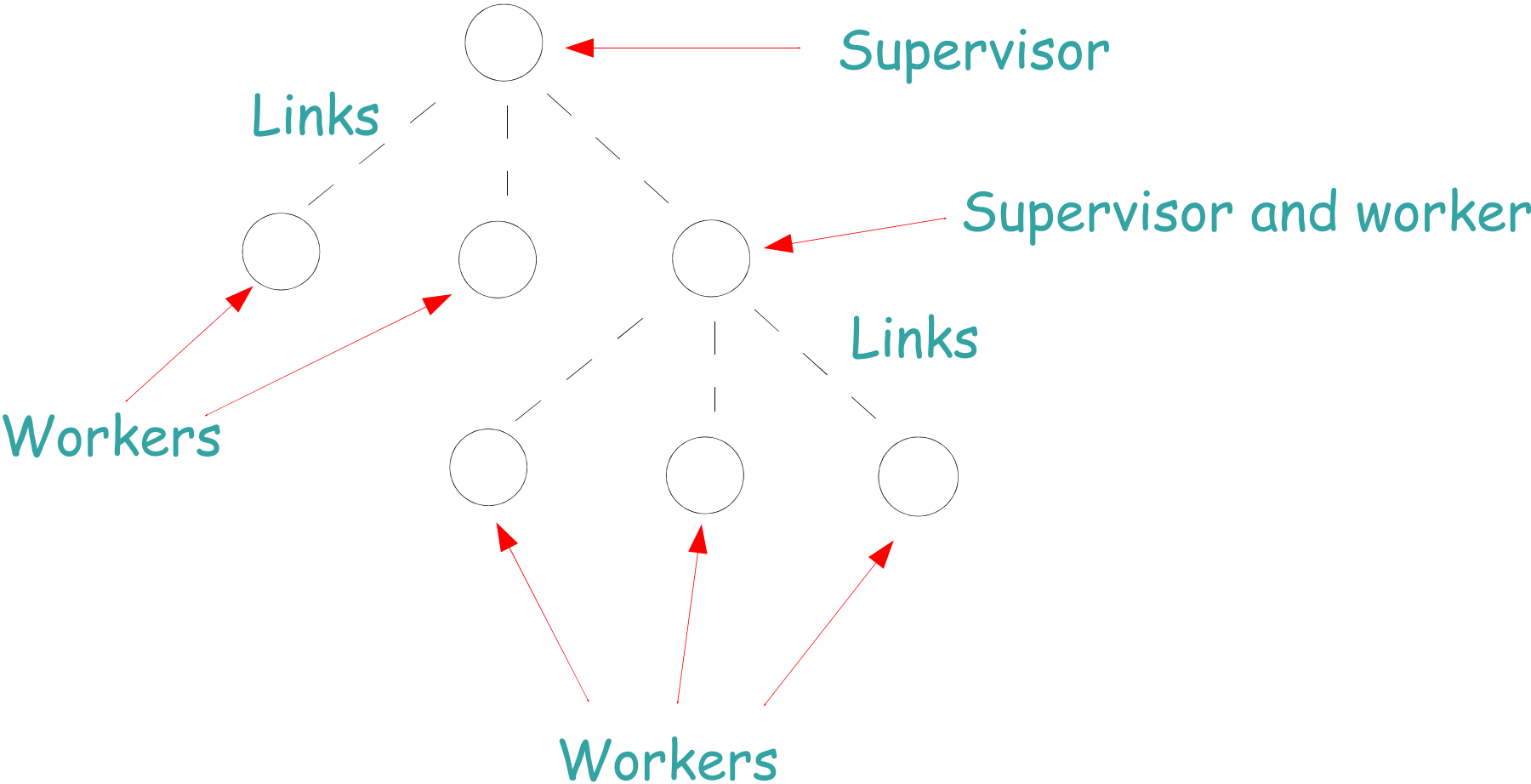
Programming for errors

If you can't do what you want to do try and do something simpler



The supervisor monitors the workers and restarts them if they fail

A supervision hierarchy



OTP behaviours

Generic libraries for building components of a real-time system.

Includes

Client-server

Finite State machine

Supervisor

Event Handler

Applications

Systems

Following or leading?

How can we make world class exciting innovative products using exactly the same technology as everybody else?

You can't - you must do something different

If you use the same techniques as everybody else (Java, C, C++, UML) you should achieve the same results as everybody else)

Doing something different involves risk. But

You either win or you loose

Either way, you learn and you have fun

If you loose you try again

Erlang success stories

Ericsson AXD301

Ericsson GPRS system

Alteon (Nortel) SSL accelerator

Alteon (Nortel) SSL VPN

Teba Bank (credit card system - South Africa)

T-mobile SMS system (UK)

Open Poker - 27K Games, 137K Players, 800K
processes on one laptop

AXD301

- ATM switch
- 11% of world market
- 99.99999999 % reliability (9 nines)
- 30-40 Million calls/week
- World's largest telephony over ATM network
- 1.7 million lines of Erlang

ENGINE Integral proves outstanding capacity capability....

BT, UK (ENGINE Integral in Hybrid configuration)

During the very popular "Pop Idol" TV show in February 2002, the Ericsson server in Ilford was on two occasions exposed to 3.4 and 2.8 MBHCA. The dynamic congestion control kicked in and limited the throughput to the configured 1.4 MBHCA. The situation was handled smoothly and did not cause any problems with the server.

In September 2002, 14 nodes has been deployed out of planned 23 nodes by the end of 2002. BT are handling about 30-40 Million calls per node and week. In total BT has switched circa 7 Billion calls to date using VoATM Technology and ATM Transport (September 2002).



BT, UK chooses Ericsson and ENGINE Integral for migration of it's transit telephony network to the world's largest Telephony over ATM network

Situation: Business Drivers



- ? Existing transit circuit-switched network needed modernization
- ? Rapid traffic growth from new and existing services
- ? Increase capacity and reduce cost through evolution to new multi-service communication system capable of carrying all telephony, data and multi-media services

Solution

- ? Partnership
- ? ENGINE Integral in hybrid configuration for >50% of BT transit network - 23 nodes across UK
- ? Management system
- ? Live cut-over from NB switches

Result

- ? 14 nodes carrying live traffic September 2002 out of planned 23 before end of 2002 (according to time plan)
- ? 99,9999999% availability ←
- ? 30-40 Million calls per week & node
- ? World's largest Telephony over ATM network
- ? Best Supplier of the year, 2000

40
31 ms. / year

SSL Accelerator

- Market leader
- 48% of world market (2002)



The image shows a rack-mountable hardware device, the Alteon SSL Accelerator, in a server room setting. The device is silver and black with a front panel featuring a power button, a fan, and a label that reads "Alteon SSL Accelerator" and "SSL-410". The background is a blurred server room with a computer monitor and other equipment. A blue and white graphic overlay with binary code and the text "Intelligent Internet" is positioned behind the device.

Nortel Networks

Alteon SSL Accelerator

Alteon SSL Accelerator features

- SSL Acceleration
- Application-layer proxy for SSL Extranets
- Content-based load balancing

The Alteon SSL Accelerator is a fully-featured Secure Sockets Layer (SSL) appliance that integrates seamlessly into any network to simplify secure environments. Its ability to handle high SSL traffic volumes, secure remote access, optimize back-end server infrastructure, and lower security costs makes it the ideal solution for today's broad range of SSL applications.

Projects

- 28 projects at sourceforge
 - YAWS (Yet another web server)
 - Eddie (Cluster/load balancer)
 - WAP gateway
 - Wings (3-D graphics modelling)
 - 3D gameing
- 72 Projects in Jungerl
 -