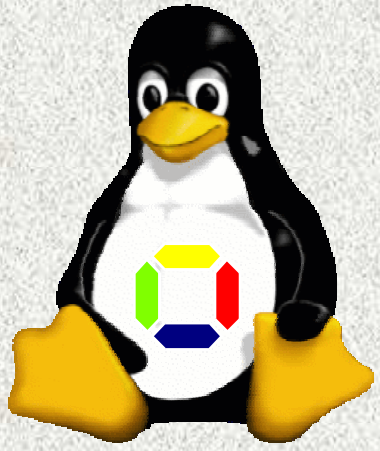
University of Essen, Institute for Experimental Mathematics

High Availability using Reliable Server Pooling



Thomas Dreibholz
Institute for Experimental Mathematics
University of Essen, Germany
dreibh@exp-math.uni-essen.de
http://www.exp-math.uni-essen.de/~dreibh

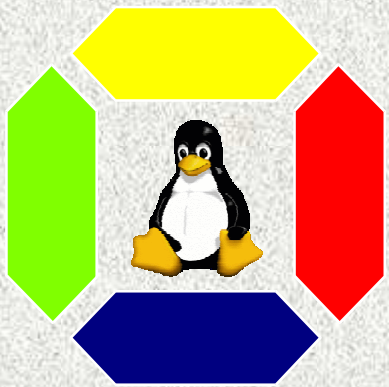


Table of Contents

SIEMENS

This work is part of KING, a research project of Siemens AG.
The work of this project is partially funded by the Bundesministerium für Bildung und Forschung of the Federal Republic of Germany (Förderkennzeichen 01AK045).

SIEMENS

Motivation

- Fault tolerancy is crucial for many applications:
  - e-Commerce
  - Control systems for machines
  - Medicine
  - Telephone signaling (SS7)
  - ...
- Server pools: Ensure availability by redundant servers
- Only proprietary solutions available / Limitations of Network Layer solutions

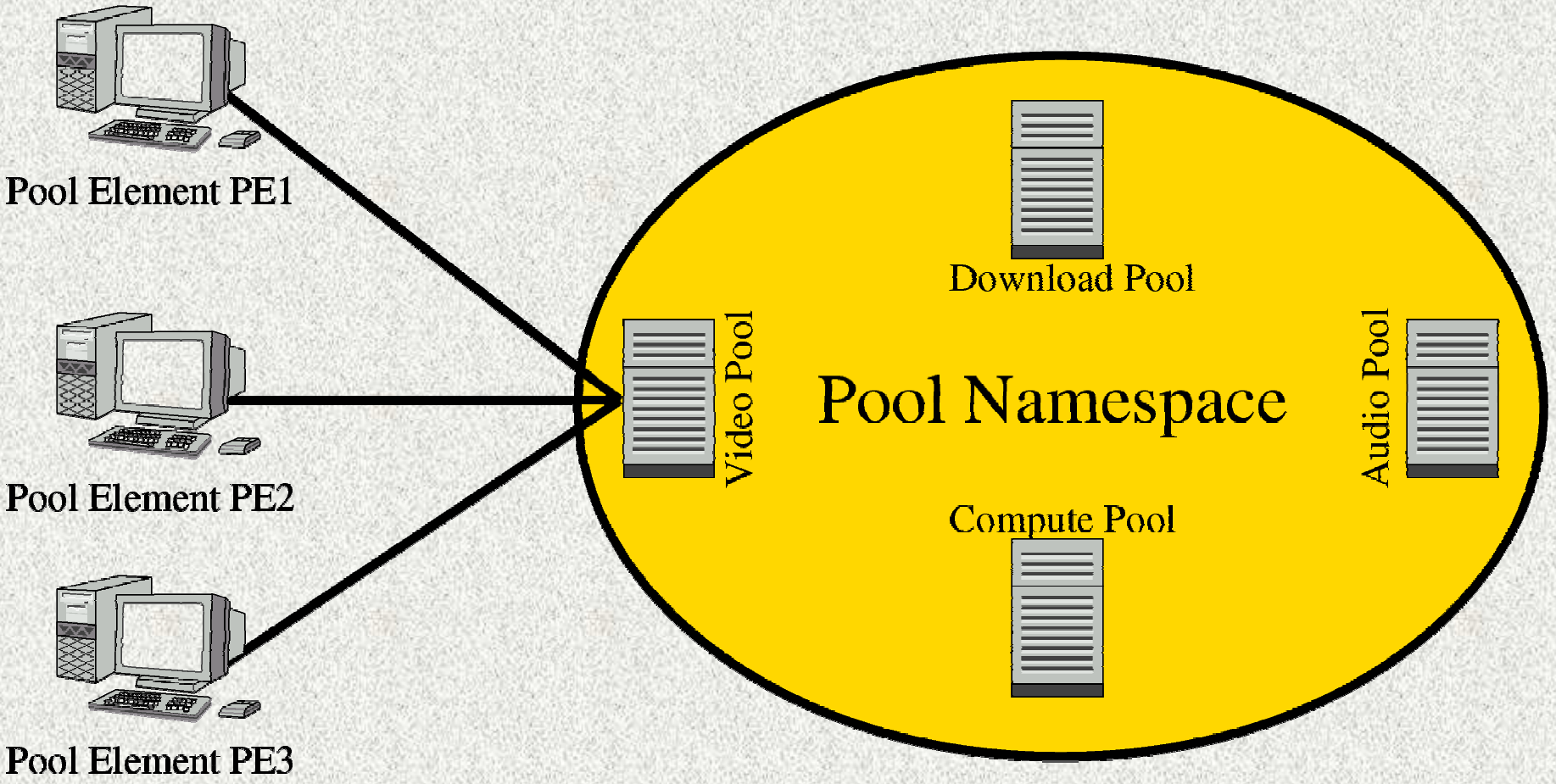
- An open standard for server pooling is necessary!
  - IETF Reliable Server Pooling (RSerPool) Working Group
  - Standardization of Session Layer server pooling architecture

RSerPool Architecture Overview

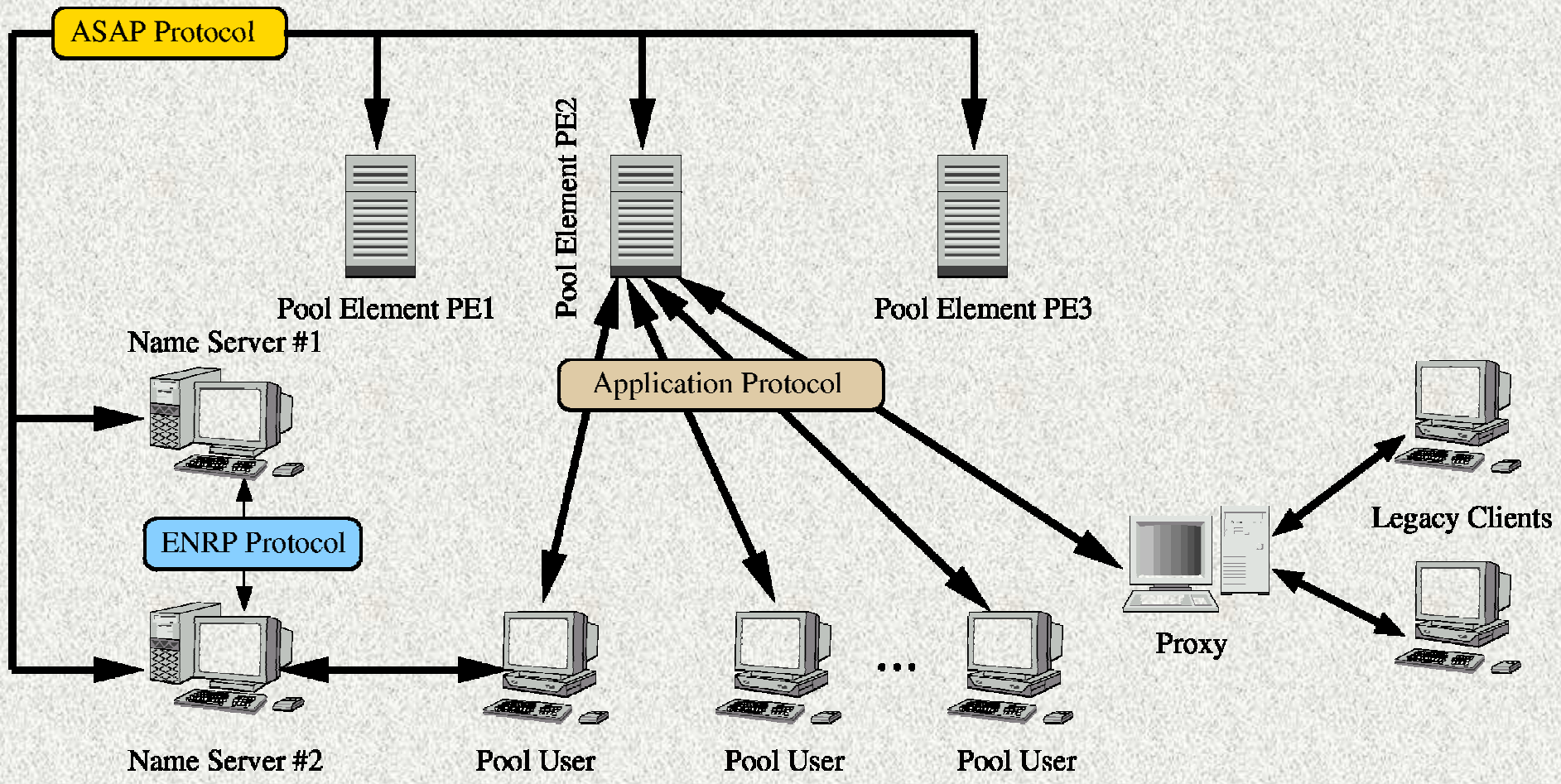
- **Based on the SCTP protocol in Transport Layer:**
  - SCTP (Stream Control Transport Protocol, RFC 2960)
  - Message framing
  - Multihoming: Peers may be connected to multiple networks => Fault tolerance in case of network failure
  - Addition and removal of addresses (Add-IP extension) => Mobility

- **RSerPool Terminology:**
  - Server Pool: Set of servers providing the same service
  - Pool Element (PE): Server belonging to a specific server pool
  - Pool User (PU): Client served by a PE
  - Name Server (NS): Manages pool namespace

RSerPool Architecture Overview



RSerPool Architecture Overview



The RSerPool Architecture: Name server

- **Namespace:**
  - Contains list of pools, each pool is identified by a pool handle (PH)
  - Pool handle = binary vector (e.g. a string „DownloadPool“)
  - Flat (no hierarchy like e.g. DNS)
- **Name server functionality:**

  Aggregate Server Access Protocol (ASAP), providing:
  - Registration and deregistration of PEs
  - Supervision of PEs (keepalives, failure reported by PU => remove PE)
  - Name resolution: PH => List of PEs
  - Sending server announcements (via multicast, optional)
- **Redundancy of name servers:**

  Endpoint Name Resolution Protocol (ENRP):

  All name servers of the same operational scope share their namespaces

The RSerPool Architecture: Pool Element

- **Pool element functionality:**
  - Find a name server
    - Static list
    - Dynamic list from received server announcements
  - Connect to one of the name servers
  - Register to a pool (name servers creates the pool, if not already existing)
  - Reregister (regularly and on parameter change, e.g. new transport address)
  - Deregister (name server deletes pool if becomes empty)
  - Note: Any name servers of an operational scope can do registration, reregistration and deregistration. In case of name server failure, simply use another one.
- **PE <=> NS communication requires SCTP**

  Reasons: Fault tolerance, supervision and message framing

The RSerPool Architecture: Pool User

- **Pool user functionality:**
  - Let name server resolve PH to list of PE transport addresses (selection by policy, see below)
  - Select one PE (again by policy, see below) and connect
  - In case of failure:
    - Initiate failover to another PE (application-specific)
    - Report PE failure to name server (NS may remove failed PE)

- **PE selection by pool policy, e.g.:**
  - Least used
  - Round robin
  - Random

  New policies can be added easily

The RSerPool Architecture: Pool User <-> Pool Element Communication

- **Data Channel**
  - Application data transmission (e.g. Video, Download, ...)
  - Any protocol, e.g. TCP, UDP, SCTP, ...

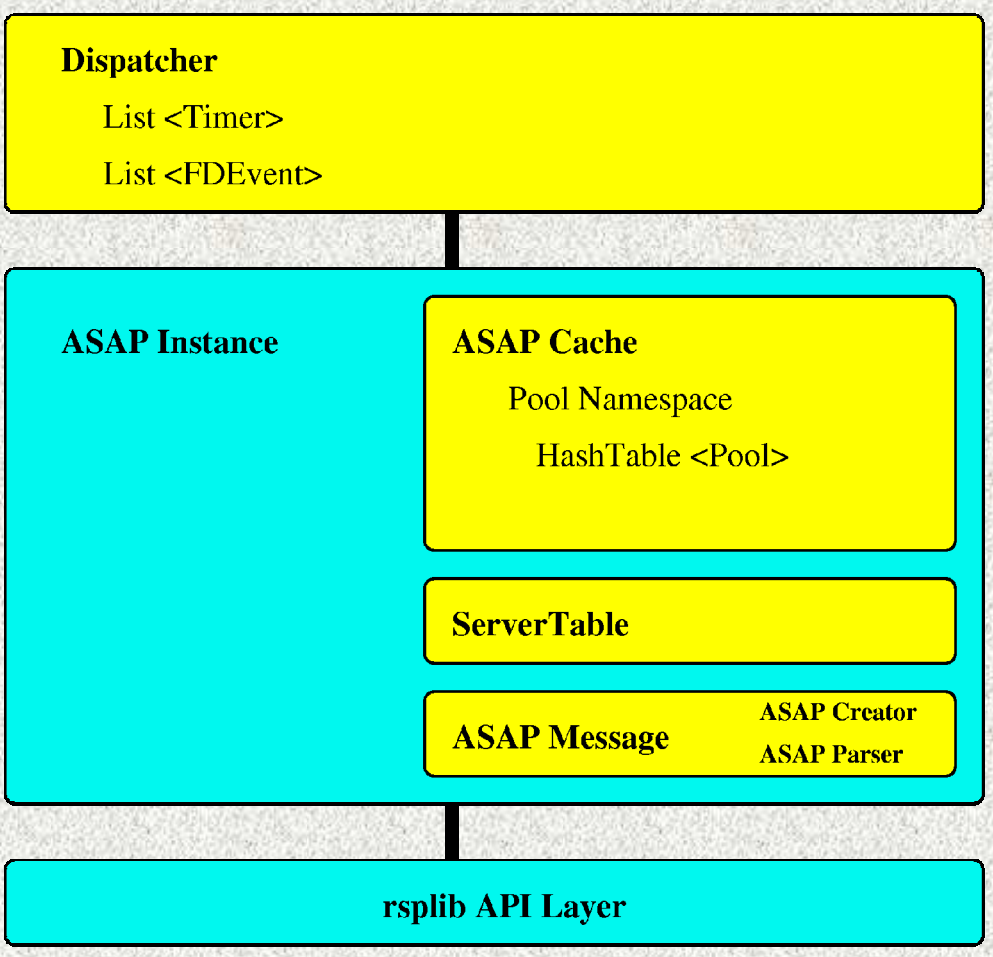
- **Control Channel (optional)**

  ASAP connection, providing for the following functionalities:
  - Business card:
    PU itself is PE of some pool. Business card = PH of this pool.
  - Last will:
    PE does state sharing with some other PEs. Last will = List of these PEs (failover only possible to these PEs)
  - Cookie:
    Client-based state sharing; PU always stores latest cookie from PE and sends it to new PE in case of failover (for details, see T.Dreibholz, „An Efficient Approach for State Sharing in Server Pools“, Local Computer Networks Conference 2002, Tampa/U.S.A.)

The rsplib Prototype: Design Decisions

- Open Source, licenced under GNU Public License (GPL)
- Support for all kinds of platforms
  - Currently: Linux, FreeBSD, Darwin
  - Future: many more, from PDA to supercomputer ...
  - Especially: PDAs and mobiles (e-commerce, m-commerce, ...)
- Implementation language: ANSI-C
  - C++ preferable, but ANSI-C available on almost any platform
- Threads
  - Do not assume specific threads implementation (e.g. pthreads, pth, ...)
  - But provide interfaces for thread-safety!
- IPv6
  - Supported from the beginning of the project
  - Necessary for future always-on mobile devices

The rsplib Prototype: Implementation Overview



- **Dispatcher:**
  Handling of timers and socket events, thread-safety
- **ASAPInstance:**
  ASAP implementation
- **rsplib API layer:**
  Wrapper for easy usage

The rsplib Prototype: Dispatcher Class

- **The Dispatcher class:**
  - Handling of timers and socket events (callbacks)
  - Event loop:
    - Requirement: Do not assume availability of select() call!
    - => Waiting for events outside of Dispatcher class
      - dispatcherGetSelectParameters()
      - dispatcherHandleSelectResults()
  - Thread-safety:
    - Requirement: Do not assume specific threads implementation!
    - => User may specify lock and unlock callbacks at constructor
      - dispatcherLock() -> Call user's lock function
      - dispatcherUnlock() -> Call user's unlock function
    - => rsplib classes use these functions to ensure exclusive access

The rsplib Prototype: ASAPInstance Class

- **The ASAPInstance class**

  The ASAP protocol implementation, containing:

  - **ServerTable class:**
    - **Listen to NS announcements via multicast**
    - **Static NS entries**
    - **Keeping NS table up-to-date**
  - **ASAPCache class:**
    - **Cache for translation PH -> List of PEs**
  - **ASAPMessage class:**

    The ASAP messasge implementation. Modules:
    - **ASAPCreator: Creation of ASAP messages**
    - **ASAPParser: Parsing of ASAP messages**

The rsplib Prototype: rsplib API layer

- **Requirements for Dispatcher/ASAPInstance**
  - Portability, Maintainablity, Extendability, ...
  - No global variables => MMU-less systems can share the library code with all running processes (Example: AmigaOS)
- **Requirements for rsplib API layer**
  - Simplicity: Make adapting existing applications as easy as possible
    => Mimic DNS calls
  - Stability: Avoid API changes, although ASAP and ENRP are still under development by IETF RSerPool WG
    => TagItems (a mechanism used in AmigaOS 2.0+)
    Additional parameters as array of (tag, data) tuples;
    tag = function-unique parameter ID, data = parameter's value
    Example: {{TAG_Param1,1}, {TAG_Param2,1234}, {0,0}}

The rsplib API Overview

- **Initialization and clean-up**
  - rspInitialize();
  - rspCleanUp();
- **Pool element functionality**
  - rspRegister();
  - rspDeregister();
- **Pool user functionality**
  - rspNameResolution();
  - rspFailure();
- **Event loop functionality**
  - rspSelect();

The rsplib API
Initialization and Clean-Up

- **Initialization:**
  - Creation of Dispatcher object
  - Creation of ASAPInstance object
- **Clean-up:**
  - Removal of both objects (of course)
  - Finding memory leaks:
    - If there is some allocated memory left, there must be a memory leak!
    - Valgrind memory debugger:
      - Open Source debugging tool for Linux
      - Interprets x86 code and tracks usage of every bit
      - Used to locate and fix memory leaks and other errors

The rsplib API
Pool User Functionality

- **Idea of rsplib:**

  Mimic DNS calls to make software adaption as simple as possible

- **Program flow of a usual client application (e.g. HTTP download):**

```
struct addrinfo* ai = NULL;
...
getaddrinfo("linux.conf.au", ..., &ai);
...
sd = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
if(sd >= 0) {
   if(connect(sd, ai->ai_addr, ai->ai_addrlen) {
      ...
   }
   freeaddrinfo(ai);
   ...
}
```

The rsplib API
Pool User Functionality

- **Program flow for pool user:**

```
struct EndpointAddressInfo* eai;
poolHandle = "DownloadPool";
rspNameResolution(poolHandle, strlen(poolHandle), &eai);
...
sd = socket(eai->ai_family, eai->ai_socktype, eai->ai_protocol);
if(sd >= 0) {
   if(connect(sd, eai->ai_addr, eai->ai_addrlen) {
      ...
      if(failure) {
         rspFailure(poolHandle, strlen(poolHandle),
                    eai->ai_identifier);
         ...
      }
      ...
   }
   ...
}
```

The rsplib API
Pool Element Functionality

- rspRegister() registers PE
- rspDeregister() deregisters PE
- Event loop:
  - Necessary to answer to ASAP keepalives (supervision functionality)
  - => rspSelect() -> replaces select() call
- Re-registration:
  - Regularly
  - On parameter changes (transport addresses, policy, policy parameters)
  - Realized by rspRegister() call
  - Not automatically in rspSelect(), because rspRegister() may block! (NS failure -> find and connect to another NS)
- Recommendation: Separate thread or process for (re-)registration

Project status and future plans

- **Current project status (January 24, 2003):**
  - Software versions: 0.1.0 (stable, Aug-2002), 0.3.0d (unstable, Jan-2003)
  - ASAP implemented (Draft version 05)
    - Control Channel not implemented (waiting for stable ASAP draft)
  - Name server implemented
    - Currently without ENRP, therefore no NS redundancy (waiting for stable ENRP draft)
  - Uses userland SCTP implementation sctplib + socketapi
    (another cooperation project between University of Essen and Siemens)
- **Future plans**
  - Test with Linux SCTP (to be in kernel 2.6) and KAME (FreeBSD, Darwin): soon
  - ASAP Control Channel implementation: during next few weeks
  - Next major goal: Implementation of ENRP -> set of redundant NSs:
    about summer 2003
  - More useful example applications (e.g. as student projects)

Distributed Computing using RSerPool

- **Distributed computing without RSerPool: SETI@home**
  - **Clients request work packages from a central server**
  - **Clients store processed work packages back to the server**
  - **Problems:**
    - **Server unreachable => Retry later**
    - **No work to do => Ask again later**
    - **Result: Wasted bandwidth for unsuccessful trials**
    - **Only the configured central server is served by the computation clients**

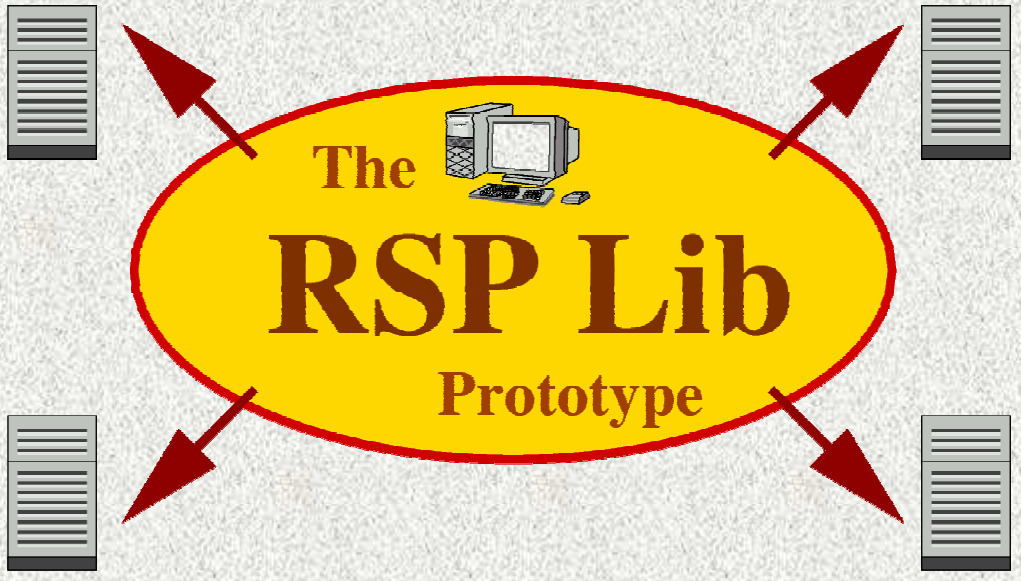
Distributed Computing using RSerPool

- **Distributed computing, the RSerPool style:**
  - Computation hosts register as PEs in a computation pool
  - Pool policy, e.g. weighted round robin, weight = composite metric of current system load and computation power
  - When computation power is required: become PU of this pool
  - Features of RSerPool-style distributed computing:
    - PEs get work when there is work to do
    - No overhead when there is currently no work
    - Selection by appropriate policy
    - Supervision functionality of the name server ensures that computation hosts (PEs) are at a high probability really available
    - Dynamic addition and removal of computation hosts (PEs)

Summary and Conclusions

- Motivation
- Introduction to the RSerPool Architecture
  - PU, PE, NS and the protocols ASAP and ENRP
- Our Implementation: The rsplib Prototype
  - Design Decisions
  - Classes
- The rsplib API
  - PU and PE functionality
- Current project status and future plans
- Distributing Computing using the RSerPool architecture

Any Questions?



Project Homepage:

http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool/

Thomas Dreibholz, dreibh@exp-math.uni-essen.de