Outline

1 History of application binary interfaces
   - Why is a new binary interface necessary?
   - x32-ABI - A new 32-bit ABI for x86-64
   - Preconditions

2 How ROOT can profit
   - Results within other CERN applications
   - Results within ROOT-Benchmarks
   - Reasons for performance differences

3 Support of x32-ABI

4 Conclusion
History of application binary interfaces

x86:
- 32-bit word size → 4GB addressable

x86_64 as an extension of x86:
- runs 32-bit as well as 64-bit applications
- 64-bit mode supports new hardware features
  - Number of CPU registers
  - Floating-point performance
  - Function parameters passed via registers
  - Syscall instruction ...
Why is a new binary interface necessary?

x86-64 (64-bit mode):
- new hardware features
- no memory limit
- slowdown due to memory issues
  - contention
  - page faults
  - paging ...

x86:
- very low memory footprint
- performance issues
Application Binary Interface based on 64-bit x86 architecture

- Reduces size of pointers and C-datatype long to 32-bit
- Takes advantage of all x64-features
- Avoids memory overhead

advantages of 64-bit instruction set
+ memory footprint of a 32-bit application
x32-ABI - A new 32-bit ABI for x86-64

- Developed by H.J. Lu (Intel)
- Introduced in Linux-Kernel 3.4 (released in summer 2012)
- Opinions in Linux-community differ quite a lot
  - 32-bit time values will overflow
  - Addressable memory
  - ...

Rauschmayr (CERN)
Impressive results from x32-developers:

181.mcf from SPEC CPU 2000 (memory bound):

**Intel Core i7**
- ~ 40% faster than x86-64
- ~ 2% slower than ia32

**Intel Atom**
- ~ 40% faster than x86-64
- ~ 1% faster than ia32
186.crafty from SPEC CPU 2000 (64bit integer):

**Intel Core i7**

~ 3 % faster than x86-64  
~ 40% faster than ia32

**Intel Atom**

~ 4 % faster than x86-64  
~ 26% faster than ia32

⇒ CERN applications will definitely reduce memory footprint and very likely CPU-time
Preconditions

x32-ABI requires:

- Linux Kernel 3.4 compiled with CONFIG_X86_X32=y

- Gcc 4.7

- Binutils 2.22

- Glibc 2.16

- Recompiling all system libraries, required by an application, with gcc -mx32

ELF 32-bit LSB shared object, x86-64, version 1 (SYSV)
CERN applications use millions of pointers
- GAUDI framework stores all events as pointers
- Many virtual functions (virtual tables: function pointers and hidden pointers)
- ROOT (histogram as a tree of pointers ...)
- ...
Results within other CERN applications

Figure: Comparison of memory consumption and CPU-time within the HEPSPEC-benchmarks
Results within other CERN applications

LHCb Reconstruction with Gaudiv23r3 and ROOT 5.34.00

Reconstruction of 1000 Events:
- Physical Memory: -20%
- Total elapsed: -2% reduction

LHCb Analysis with Gaudiv23r3 and ROOT 5.34.00

Analysis of 10000 Events:
- Physical Memory: -21%
- Total elapsed: 1.5% increase
Results within ROOT-Benchmarks

Following ROOT-benchmarks:

- stressHistogram
- stress 1000
- stressHepix
  - IO
  - linear algebra
- vector
- sparse matrix
Figure: Comparison of memory consumption and CPU-time within the ROOT-benchmarks
stress 1000:

![Graph showing memory usage and reduction percentages for different operations and architectures (64 vs x32)]
Reasons for performance differences

- Memory seen by system and by process are different.

Default thresholds of `malloc`:
- \( t = 4 \cdot 1024 \cdot 1024 \cdot \text{sizeof}(\text{long}) \)
- if \( x > t \): `malloc` uses `mmap`
- if \( x < t \): `malloc` uses `sbrk`

`mmap`:
- memory blocks can be independently given back
- anonymous memory blocks as an extension of the heap

`sbrk`:
- memory blocks cannot be returned to the operating system
Looking into *dealII*, *omnetpp*, **ROOT stress**:  
- *dealII*: page faults reduced by 10%
- *omnetpp*: page faults reduced by 38%
- *stress*: difference in CPU-time likely caused by memory issues

Further Issues:
- code and data alignment
- x32 loop unrolling needs some tuning
- zero-extensions for pointer conversion

⇒ x32-ABI still under development and there are possibilities for optimization
1 History of application binary interfaces
   - Why is a new binary interface necessary?
   - x32-ABI - A new 32-bit ABI for x86-64
   - Preconditions

2 How ROOT can profit
   - Results within other CERN applications
   - Results within ROOT-Benchmarks
   - Reasons for performance differences

3 Support of x32-ABI

4 Conclusion
Support of x32-ABI

- Ubuntu: Ongoing work, support planned for 13.04: https://blueprints.launchpad.net/ubuntu/+spec/foundations-r-x32-planning
- Red Hat: No plans according to: http://comments.gmane.org/gmane.linux.redhat.fedora.devel/164019
- LLVM-compiler: supports x32-ABI since summer 2012 http://www.phoronix.com/scan.php?page=news_item&px=MTI3NTk
1. History of application binary interfaces
   - Why is a new binary interface necessary?
   - x32-ABI - A new 32-bit ABI for x86-64
   - Preconditions

2. How ROOT can profit
   - Results within other CERN applications
   - Results within ROOT-Benchmarks
   - Reasons for performance differences

3. Support of x32-ABI

4. Conclusion
Conclusion

- New room for improvement, (CERN) applications can profit substantially
- Computing Grid limits memory anyway to 4 GB per process
- Gain in performance is for **FREE**
Conclusion

Recompilation:
- Cast from time values to long (...) will produce wrong results (xrootd)
- New pointer size required modifications in CINT (function and data pattern)

Getting a working environment:
- does not work out of the box
- building gcc fails due to missing glibc x32
- building glibc x32 with a partly built gcc
Any questions?