

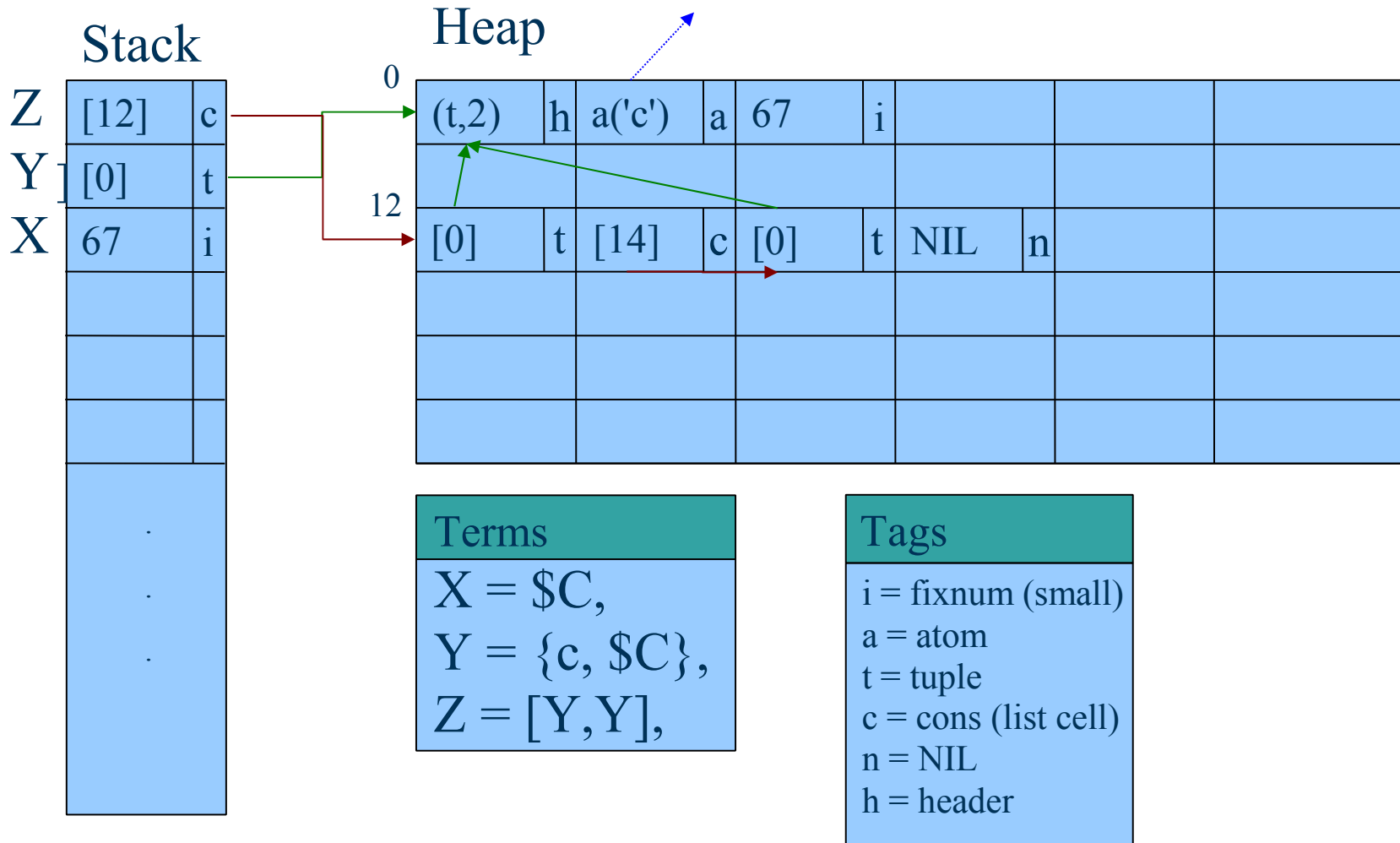
Reducing memory footprint

On Linux x86_64

The problem

- Processors get faster and get more cores
- Memory bandwidth can't really keep up
- 64bit virtual Erlang machines consume a lot more memory than 32bit
 - Erlang terms contain a lot of pointer data
 - An Erlang term's size is a multiple of the memory address size
 - Atoms, characters and small numbers does seldom benefit from the larger term size of a 64bit virtual machine
- Even if we buy twice the amount of physical memory, the memory bandwidth will reduce performance

Erlang terms



Simply put

- Each Erlang term is either a immediate value or a pointer to the heap
- The heap is a continuous array of terms. Each element being “`union {Eterm term, Eterm *termp};`”
- On a 64bit machine each heap unit is 8 bytes and i.e. a cons cell is 16 bytes (two terms), which means a string is almost 16 times as memory consuming as the corresponding string in C (at least in latin1)
- 64bit terms are twice as big as the corresponding 32bit terms and only large numbers really benefit from the larger size
- Half the number of terms fit in the cache...

But...

- Then stick with a 32bit machine
 - The 32bit mode limits the x86 in terms of registers
 - We want to be able to break the 4GB limit on the virtual machine size of a 32bit environment
 - However, not all data in the Erlang machine is terms on a heap...
- The heap is hardly so big that we need 64bits to address it, why not use indexes into the heap instead of memory addresses?
 - Unfortunately there is not only one heap per process, its both two generations of heaps, temporary heaps and constants
 - Maybe it could fit in a 32bit address space though?

So...

- A “halfword” virtual machine, where all the heaps are in one 32bit address range (preferably the address space of the lowest 4GB)
- Only “half pointers” are used in Erlang terms, when removing the tag to actually dereference the pointer, it is also expanded to a “real” 64bit pointer
- Only heaps, constants and code need to be in the “low address range”
- ETS data, content of shared binaries, internal structures, atom strings, driver data etc can be anywhere in the address range
- ETS data can be as compact as the heap data if we handle it smart

Benefits

- Term data (including ETS data) take less space
- The full instruction set of the x86_64 processors can be used, including all the registers
- Caches are not bashed much more than in the 32bit VM
- Compared to a 32bit VM, we have more memory for heaps (almost all of 4GB)
- Compared to a 32bit VM we have almost unlimited space for ETS data and large binaries

Prototype

- A simple mmap wrapper that keeps data in the lower range
- A pure halfword VM where everything is in that low range (issues of ETS and different allocators using different memory ranges not addressed)
- Unaligned access is left to the processor to sort out (not many issues with that, only some types of lambdas, but will be an issue with binaries as well)
- Expanding pointers showed to be almost without cost
- Benchmarking showed astonishing results
 - As good as the 32bit virtual machine at handling large terms
 - As good as the 64bit machine at “crunching”
 - Better than any VM at sombined tests

Future

- No founding at the moment, but hopefully before R15
- Much more work on memory allocation needed
- ETS data should be stored with offset-based terms and moved out of the “low range”, requires more job than it sounds like
- Shared binaries etc should also be moved out of the low range (requires more than just allocating them in another way)
- Ideas for a more general approach where processes can reside in different 32bit ranges could replace the pure 64bit VM altogether

ERICSSON 

TAKING YOU FORWARD