igprof

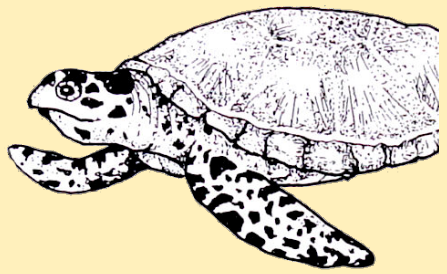
the ignominious profiler

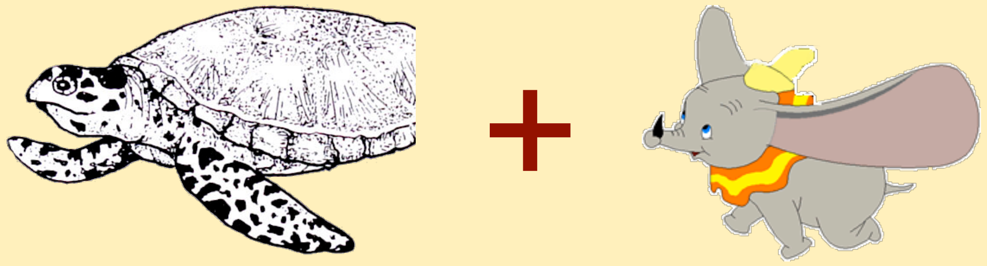
Giulio Eulisse - Lassi Tuura
Northeastern University of Boston

Some history

- igprof stands for "the IGnominious PROFiler" and it is found in the IGNOMINY project.
- A.K.A. the "IGUANA profiler"
- A.K.A. "MemProfLib"

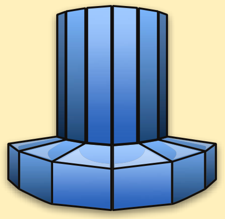
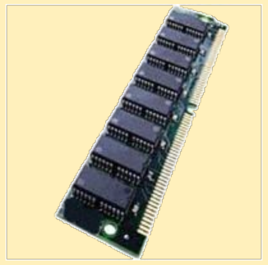
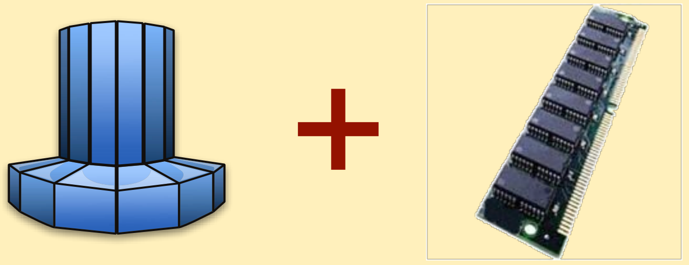
Use cases...

- Typical programming problems:
  - “My code runs slow”
  
  - “My code requires 1Gb of RAM”

  - “My code runs slow AND requires 1Gb of RAM”
  

Use cases...

- Typical solutions:
  - buy a faster computer
  - buy more memory
  - buy a faster computer **AND** more memory

...or...

- Start doing some profiling of your code, to see which parts of the system require more memory, which require more CPU power and then focus in optimizing those parts.
- igprof is a tool which allows to produce **sensible** profiling information in an easy and **unintrusive** way so that even the casual developer can collect information **useful** for code optimization.

igprof features

- Generic profiling framework: plugins already present for performance and memory usage profiling
- Memory leak detection tool
- Fast enough to actually profile CMS simulation and reconstruction software
- Non-Intrusive

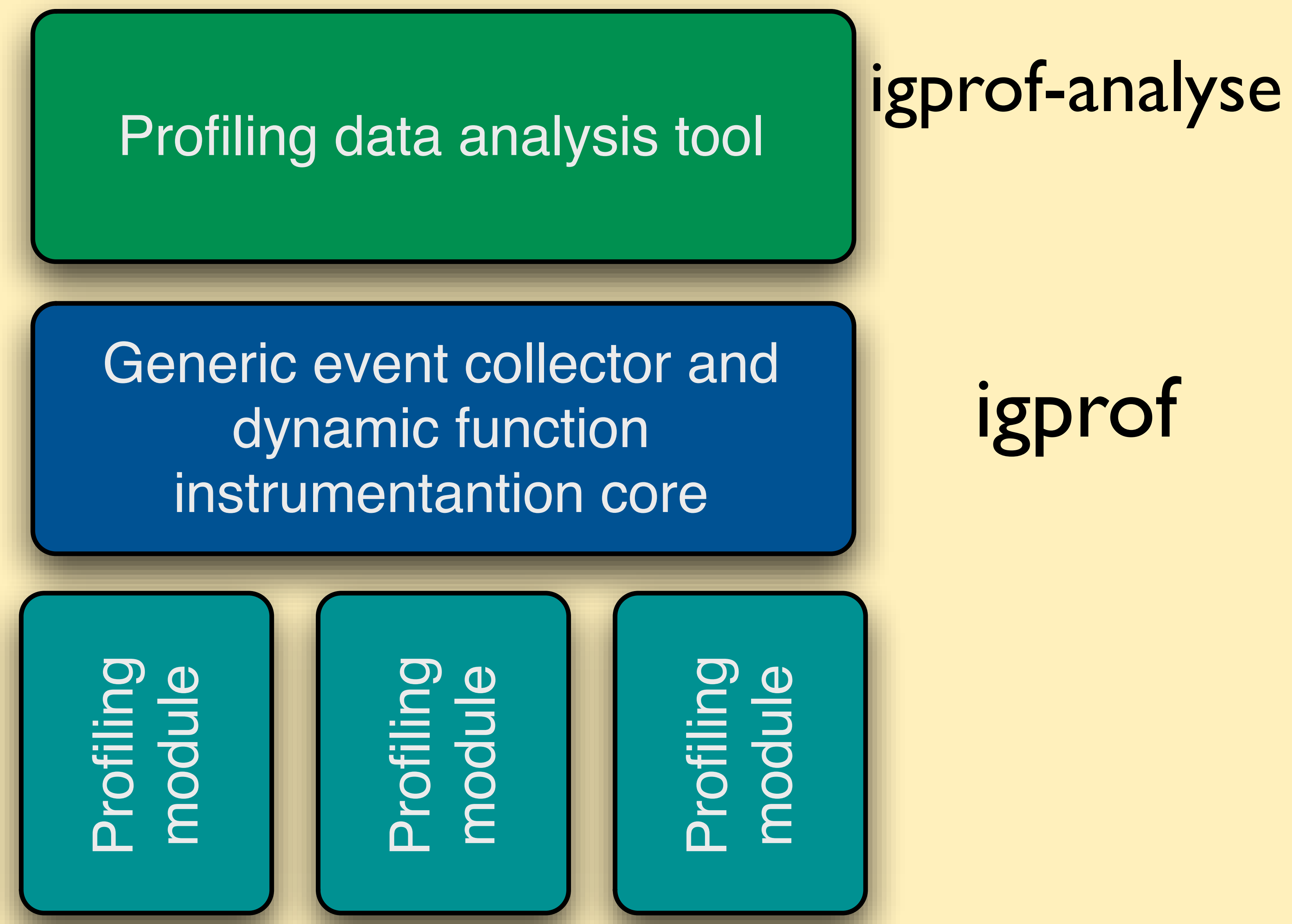
more features...

- Multiple profiling counters
- Post processing application provides a number of filtering possibilities
- Multiple counters are allowed
- Optional code instrumentation
- Works with shared libraries and multithreaded programs
- User space (no kernel/root fiddling)

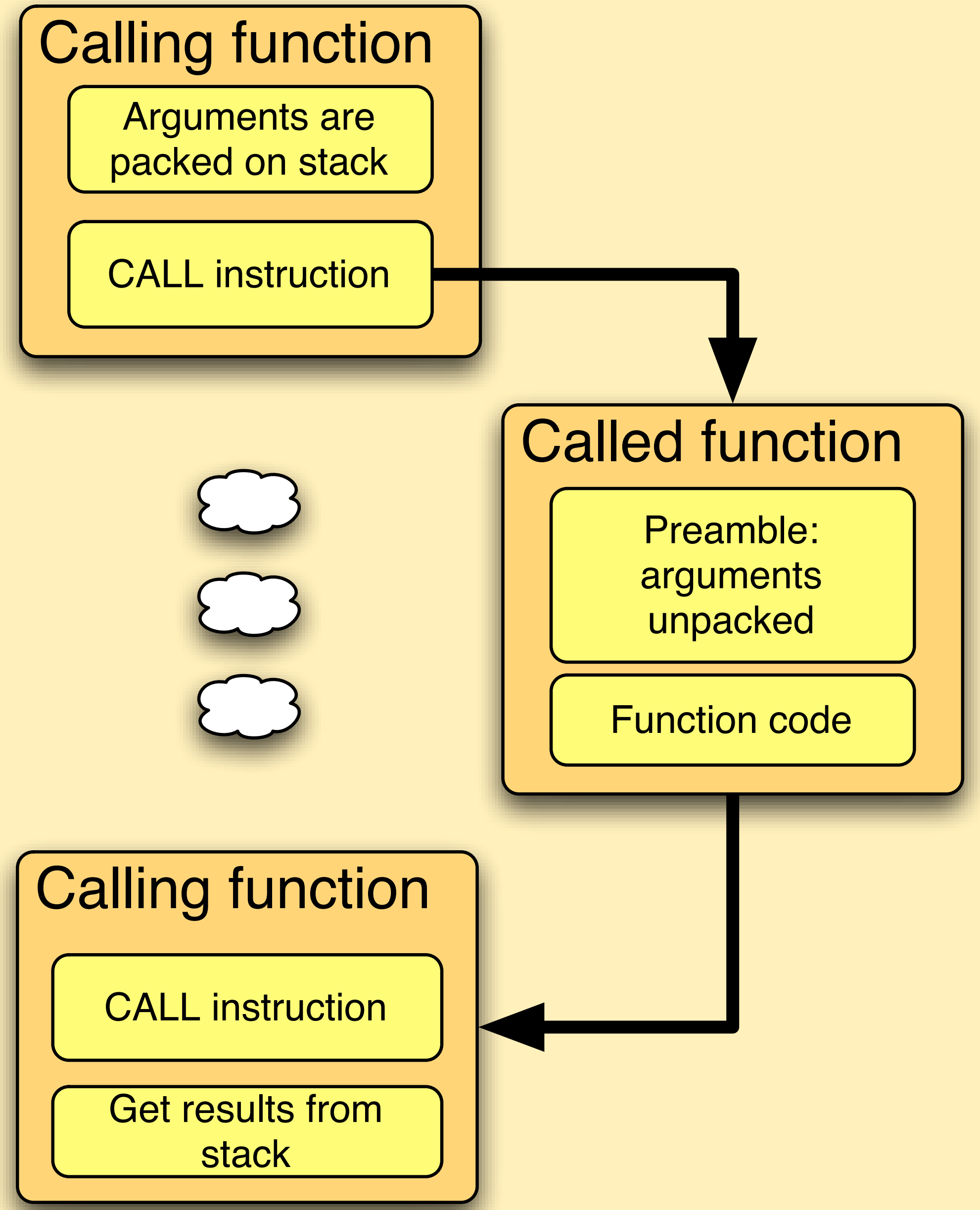
igprof & other tools

- Complementary to oprofile & valgrind
- Oprofile: it is a more low level profiler, requires kernel instrumentation and root user access, but has much richer choice of possible performance measurements.
- Valgrind is much more accurate memory checking tool, but does not profile allocations, it is much slower, and works only on X86 architecture (at least as of 2.1)

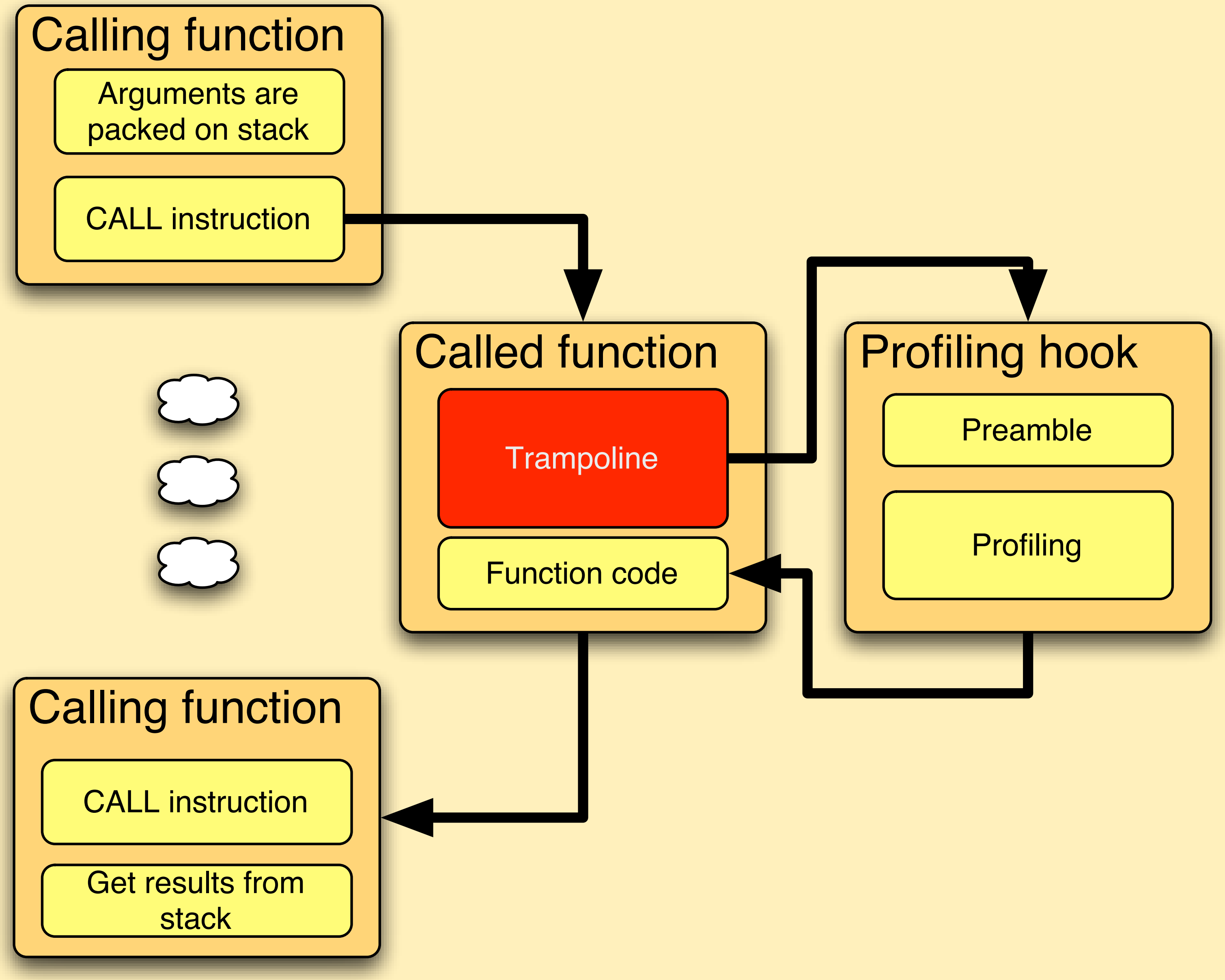
IgProf architecture


Profiling data analysis tool
igprof-analyse
Generic event collector and dynamic function instrumentantion core
igprof
Profiling module
Profiling module
Profiling module

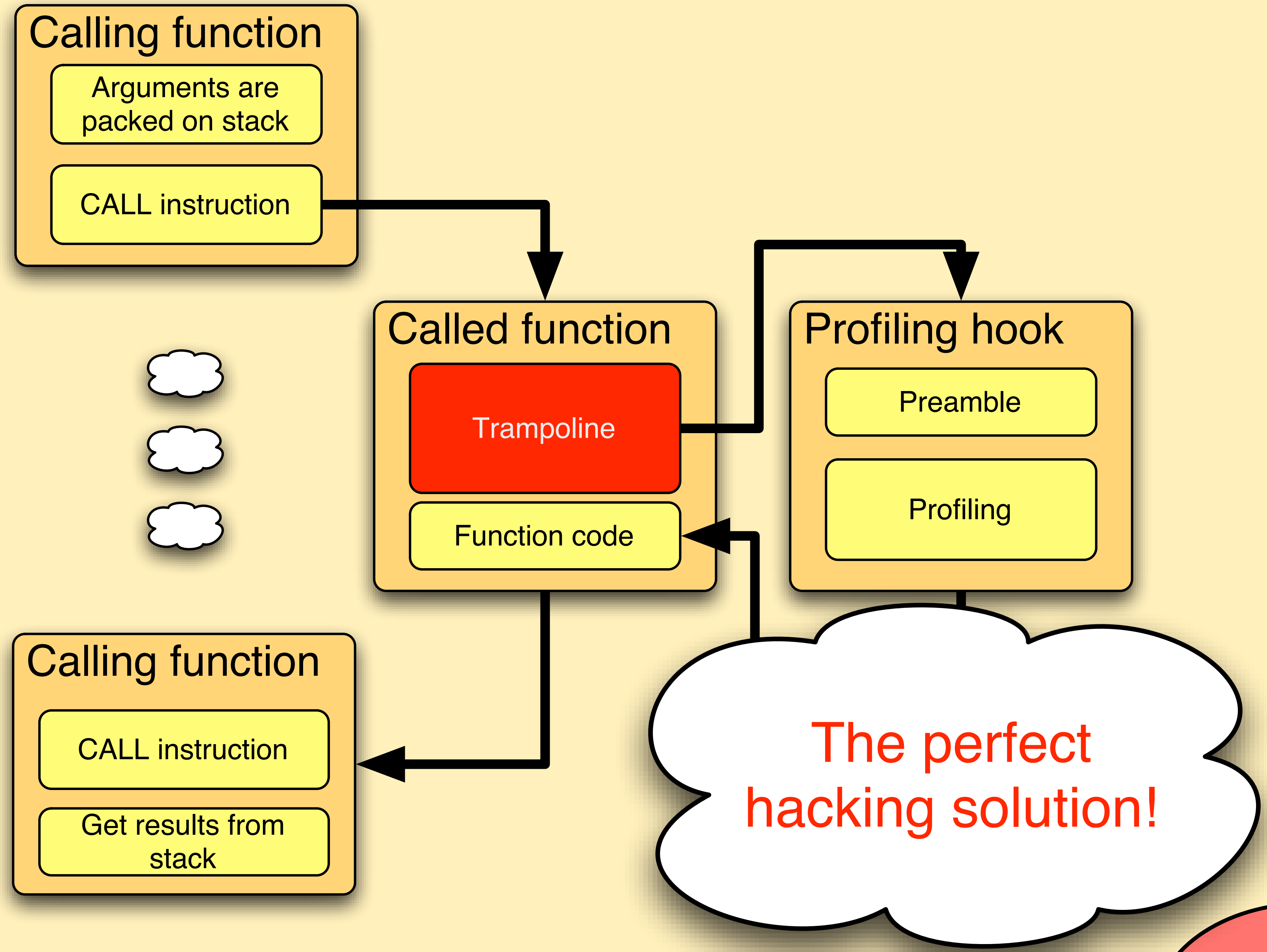
Hooking mechanism



Hooking mechanism

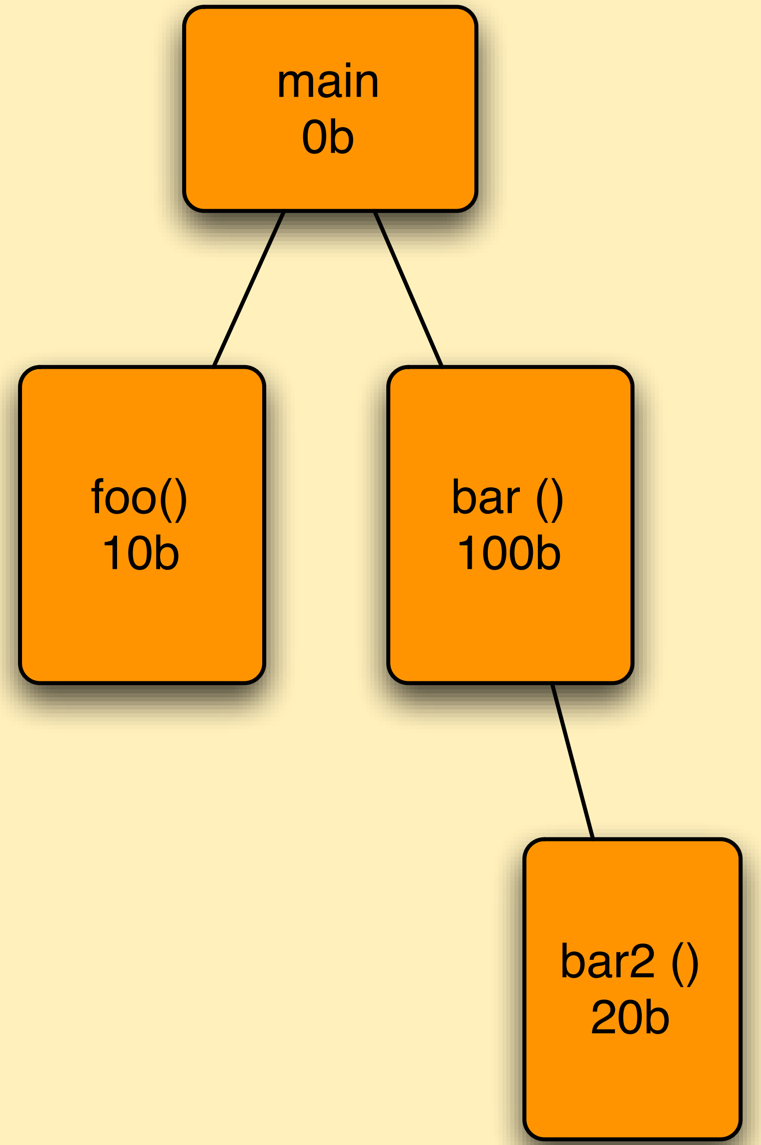


Hooking mechanism

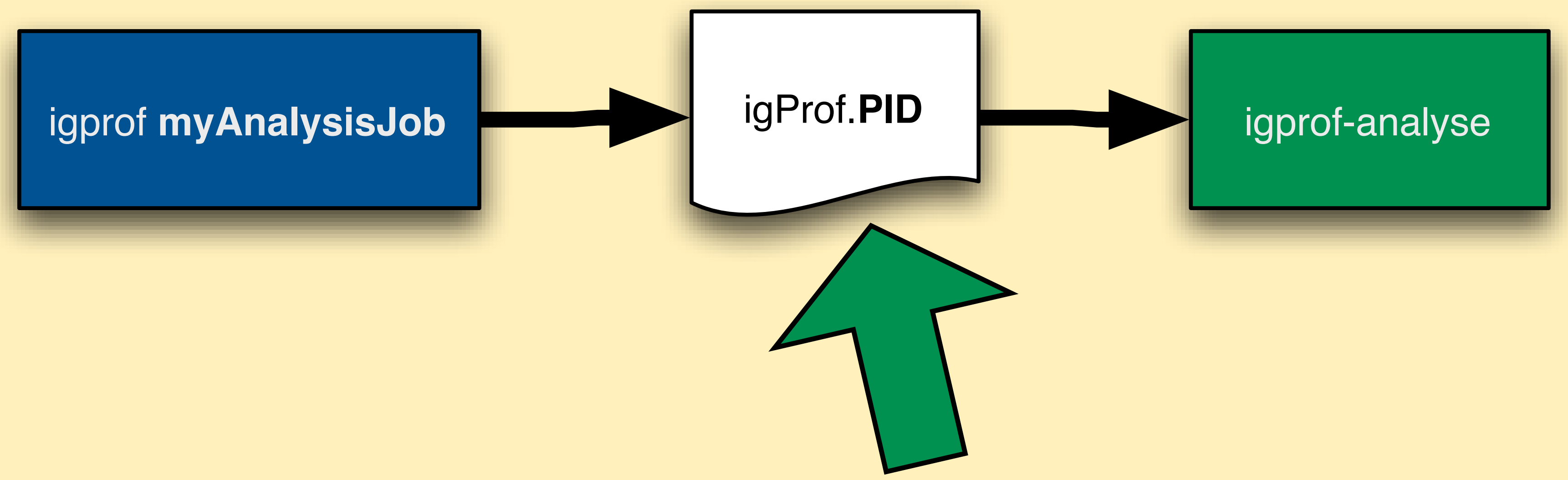


Data structure

```
foo(){new char[10];}

bar() {
    for (int i=0; i < 10; i++) {
        new char[10];
    }
}

bar2 () {
   new int[5];
}

main () {
    foo();
    bar();
}
```



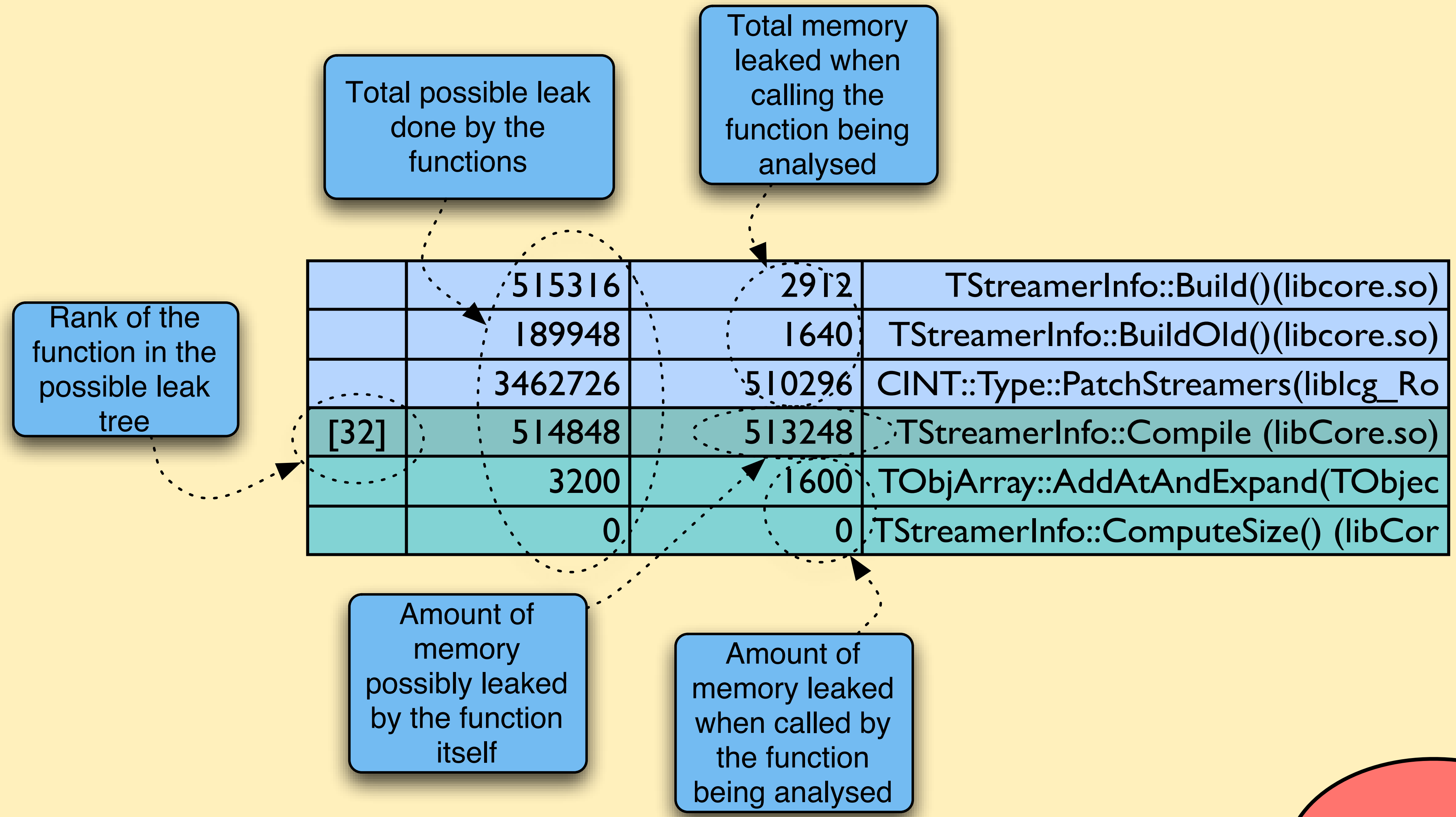
Result analysis



tree dump in XML

example output

gprof like output



Total possible leak done by the functions
Total memory leaked when calling the function being analysed
Rank of the function in the possible leak tree
Amount of memory possibly leaked by the function itself
Amount of memory leaked when called by the function being analysed

	515316	2912	TStreamerInfo::Build()(libcore.so)
	189948	1640	TStreamerInfo::BuildOld()(libcore.so)
	3462726	510296	CINT::Type::PatchStreamers(liblcg_Ro
[32]	514848	513248	TStreamerInfo::Compile (libCore.so)
	3200	1600	TObjArray::AddAtAndExpand(TObjec
	0	0	TStreamerInfo::ComputeSize() (libCor

example usage

- igprof -pp **myAnalysisJob** *myArgs*
  profiles performances

- igprof -mp **myAnalysisJob** *myArgs*
  profiles live memory allocations

- igprof -cl **myAnalysisJob** *myArgs*
  leak hunter

the output is a igprof.***PID*** xml file containing the complete information about the profiling session. Such information can be analysed and displayed with

igprof-analyse igprof.**PID**

Success stories

- It has been successfully used to improve CMS simulation and reconstruction software.
  It allowed to spot easily (for example):
  - a 180Mb/1000 events leak in ORCA muon code
  - a 20Mb/1000 events leak in ORCA tracker code

Future plans

- Improve the analysis tool
- Proper MacOS X support
- Support for Monalisa (?)