Optimizing large applications

Honza Hubička
SuSE ČR s.r.o
Martin Liška
Charles University

SUSElabs conf, 2013

Outline

Massive performance regression from switching to gcc 4.5

Hi,
Just wanted to give a heads up on what might be the biggest compiler-upgrade-related performance difference we've seen at Mozilla.

We switched gcc4.3 for gcc4.5 and our automated benchmarking infrastructure reported 4-19% slowdown on most of our performance metrics on 32 and 64bit Linux.

...

Most of the code is compiled with `-fPIC -fno-rtti -fno-exceptions -Os -freorder-blocks -fomit-frame-pointer`.

Taras Glek (2010)

First look at firefox situation

- . . . Why they use so old GCC for release builds? It builds just fine with trunk GCC.

First look at firefox situation

- . . . Why they use so old GCC for release builds? It builds just fine with trunk GCC.
- . . . why they build with `-Os` and complain about runtime?

First look at firefox situation

- . . . Why they use so old GCC for release builds? It builds just fine with trunk GCC.
- . . . why they build with `-Os` and complain about runtime?
- . . . this beast is much bigger than I expected

First look at firefox situation

- . . . Why they use so old GCC for release builds? It builds just fine with trunk GCC.
- . . . why they build with `-Os` and complain about runtime?
- . . . this beast is much bigger than I expected
- . . . why all the performance critical functionality is in a library `libxul` rather than main firefox binary?

First look at firefox situation

- . . . Why they use so old GCC for release builds? It builds just fine with trunk GCC.
- . . . why they build with `-Os` and complain about runtime?
- . . . this beast is much bigger than I expected
- . . . why all the performance critical functionality is in a library `libxul` rather than main firefox binary?
- . . . why `libxul` contains private copies of `libffi`, `gtk`, `cairo`, you name it, ...

Outline

Optimizations levels at a glance

- `-O2` the default optimization level supposed to do wise code size vs performance tradeoffs.

Optimizations levels at a glance

- `-O2` the default optimization level supposed to do wise code size vs performance tradeoffs.
- `-O3/-Ofast` optimize performance as much as possible!

Optimizations levels at a glance

- `-O2` the default optimization level supposed to do wise code size vs performance tradeoffs.
- `-O3`/`-Ofast` optimize performance as much as possible! . . . shoot yourself into leg if program is too big
  - automatic inlining, function specialization, autovectorization, loop unswitching, memset/memcpy discovery, . . .

Optimizations levels at a glance

- `-O2` the default optimization level supposed to do wise code size vs performance tradeoffs.
- `-O3`/`-Ofast` optimize performance as much as possible! . . . shoot yourself into leg if program is too big
  - automatic inlining, function specialization, autovectorization, loop unswitching, memset/memcpy discovery, . . .
- `-Os` reduce size as much as possible
  - Inline and specialize only when code shrinks
  - Instruction selection (`push`, `pop`, `rep movsb`, `rep stosb`, `mult`/`idiv` by constant rather than sequence of arithmetics,. . .
  - No code expanding optimizations (unrolling, vectorizing)

Optimizations levels at a glance

- `-O2` the default optimization level supposed to do wise code size vs performance tradeoffs.
- `-O3`/`-Ofast` optimize performance as much as possible! . . . shoot yourself into leg if program is too big
  - automatic inlining, function specialization, autovectorization, loop unswitching, memset/memcpy discovery, . . .
- `-Os` reduce size as much as possible
  - Inline and specialize only when code shrinks
  - Instruction selection (`push`, `pop`, `rep movsb`, `rep stosb`, `mult`/`idiv` by constant rather than sequence of arithmetics,. . .
  - No code expanding optimizations (unrolling, vectorizing)

Over time `-O3` code gets bigger and slightly faster, `-Os` code gets much slower and (sometimes) slightly smaller.

Fixing `-Os` for firefox.

`-Os` was designed with low level programming in mind. It happens when:

1. When static function is inlined few enough times so whole compile units shrinks after it is fully inlined.
2. When caller is known to shrink after inlining.

Fixing `-Os` for firefox.

`-Os` was designed with low level programming in mind. It happens when:

1. When static function is inlined few enough times so whole compile units shrinks after it is fully inlined.
2. When caller is known to shrink after inlining.

C++ inline functions are usually not static/in anonymous namespace. COMDAT sections merged at linktime.

Fixing `-Os` for firefox.

`-Os` was designed with low level programming in mind. It happens when:

1. When static function is inlined few enough times so whole compile units shrinks after it is fully inlined.
2. When caller is known to shrink after inlining.

C++ inline functions are usually not static/in anonymous namespace. COMDAT sections merged at linktime.

- Rule 1 never applies here.
  `-param comdat-sharing-probability`=20% specify chance that unification at linktime happens.

Fixing `-Os` for firefox.

`-Os` was designed with low level programming in mind. It happens when:

1. When static function is inlined few enough times so whole compile units shrinks after it is fully inlined.
2. When caller is known to shrink after inlining.

C++ inline functions are usually not static/in anonymous namespace. COMDAT sections merged at linktime.

- Rule 1 never applies here. `-param comdat-sharing-probability`=20% specify chance that unification at linktime happens.
- Rule 2 is too weak. GCC inliner can anticipate just fraction of optimizations. I made inliner to gamble.

Firefox and optimizations levels

Half a year later...

Firefox and optimizations levels

Half a year later...

"Hey I fixed the `-Os` problems. It was a piece of a cake."

Firefox and optimizations levels

Half a year later...

"Hey I fixed the `-Os` problems. It was a piece of a cake."

"Oh cool. But we use `-O3` now...

Firefox and optimizations levels

Half a year later...

"Hey I fixed the `-Os` problems. It was a piece of a cake."

"Oh cool. But we use `-O3` now...

"Uh, why you don't use `-O2`?"

Firefox and optimizations levels

Half a year later...

"Hey I fixed the `-Os` problems. It was a piece of a cake."

"Oh cool. But we use `-O3` now...

"Uh, why you don't use `-O2`?"

...we solved the startup time issues."

Firefox and optimizations levels

Half a year later...

"Hey I fixed the `-Os` problems. It was a piece of a cake."

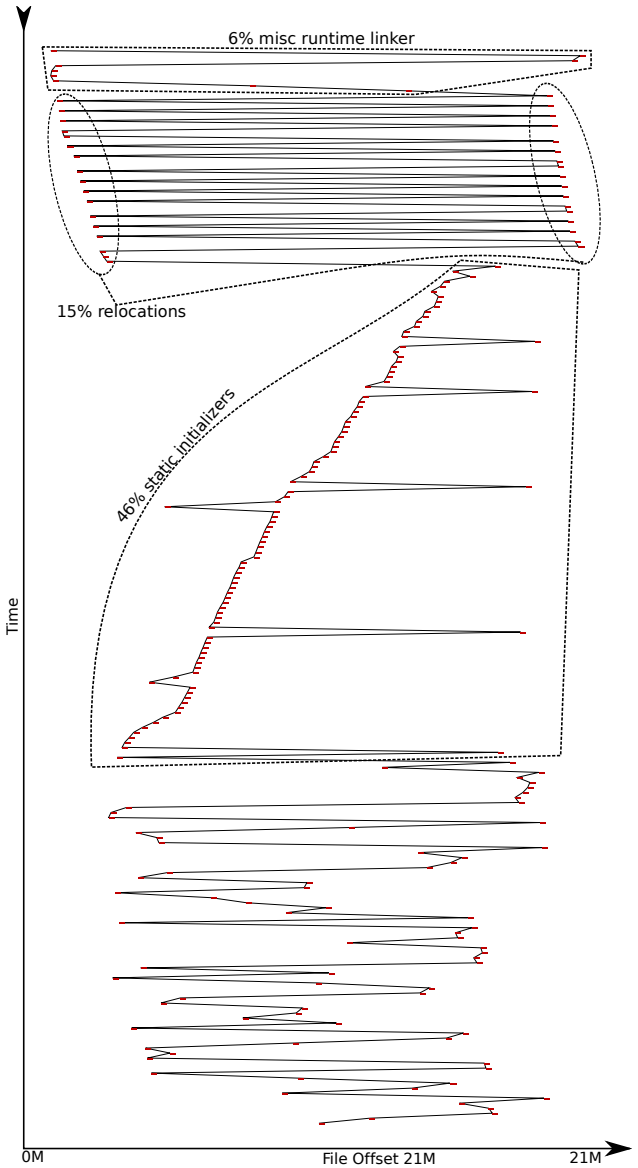
"Oh cool. But we use `-O3` now...

"Uh, why you don't use `-O2`?"

...we solved the startup time issues."

"hmm, startup time issues?"

Firefox startup



Startup overview

- Kernel memory maps the binary and starts dynamic linker
  - Page demand loading loads only pages touched by app
  - Prefetch heuristic attempts to reduce random seeking
  - Based on ELF header kernel dispatch to the dynamic linker

Startup overview

- Kernel memory maps the binary and starts dynamic linker
  - Page demand loading loads only pages touched by app
  - Prefetch heuristic attempts to reduce random seeking
  - Based on ELF header kernel dispatch to the dynamic linker
- Dynamic linker
  - Mmaps shared libraries
  - Process relocations except PLT and fixes memory image
  - Executes actual program

Startup overview

- Kernel memory maps the binary and starts dynamic linker
  - Page demand loading loads only pages touched by app
  - Prefetch heuristic attempts to reduce random seeking
  - Based on ELF header kernel dispatch to the dynamic linker
- Dynamic linker
  - Mmaps shared libraries
  - Process relocations except PLT and fixes memory image
  - Executes actual program
- Program's runtime
  - Execute all static constructors in priority order.
    If priorities match, constructors are executed backwards so libraries are constructed first
  - Execute `main()`

Startup overview

- Kernel memory maps the binary and starts dynamic linker
  - Page demand loading loads only pages touched by app
  - Prefetch heuristic attempts to reduce random seeking
  - Based on ELF header kernel dispatch to the dynamic linker
- Dynamic linker
  - Mmaps shared libraries
  - Process relocations except PLT and fixes memory image
  - Executes actual program
- Program's runtime
  - Execute all static constructors in priority order.
    If priorities match, constructors are executed backwards so libraries are constructed first
  - Execute `main()`
- program does something hopefully useful until it crashes.

ELF answer to shared libraries (1995—1999)

`a.out` shared libraries was hell to maintain

- Required central authority for address space distribution
- Required hand crafted entry points with indexes

ELF answer to shared libraries (1995—1999)

`a.out` shared libraries was hell to maintain

- Required central authority for address space distribution
- Required hand crafted entry points with indexes

ELF introduced shared libraries that are very flexible

- Linking is done based on symbol name at runtime by dynamic linker
- shared libraries are flexible first
  - Symbol interposition allows rewriting of given symbol `LD_PRELOAD`
  - Versioning allows better backward compatibility
- Some performance features (visibilities) are provided

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.
- **Assume that most symbols can be overwritten at runtime.**

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.
- **Assume that most symbols can be overwritten at runtime.** (tricky to change for exported symbols)

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.
- **Assume that most symbols can be overwritten at runtime.** (tricky to change for exported symbols)

Decades old assumptions:

- Static variable initializers rarely take address of symbol.

ELF answer to shared libraries II

PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.
- **Assume that most symbols can be overwritten at runtime.** (tricky to change for exported symbols)

Decades old assumptions:

- Static variable initializers rarely take address of symbol.
- It is not too common to take address of function or static variable in code (only for `qsort`).

ELF answer to shared libraries II

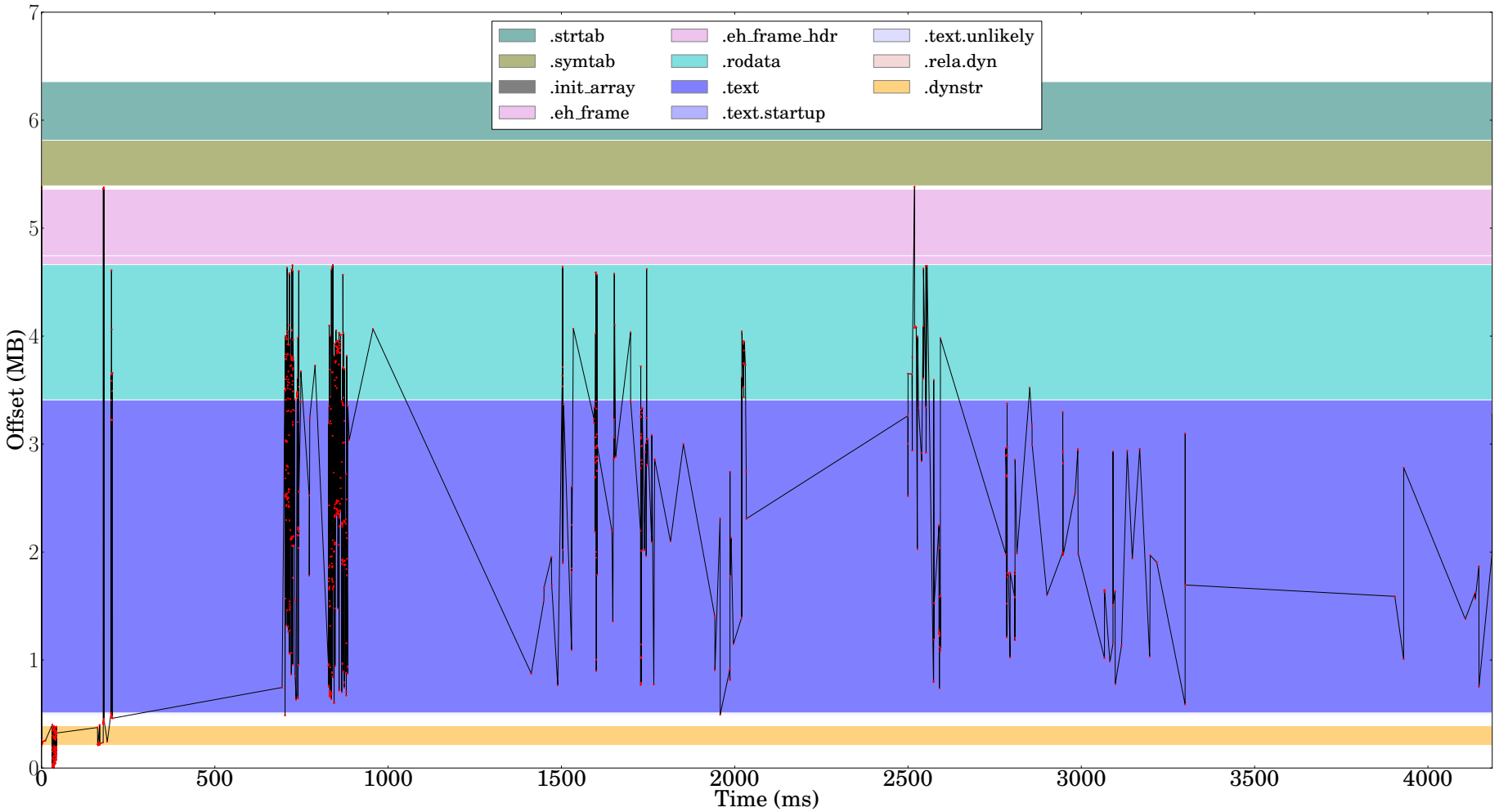
PIC programming model tells compiler to

- Use IP relative addressing whenever possible.
- Instead of calling external function directly, call to IP relative PLT entry. This triggers linking only first time function is called.
- Instead of referring to variable directly use GOT (Global Offset Table) to concentrate relocations to single place.
- **Assume that most symbols can be overwritten at runtime.** (tricky to change for exported symbols)

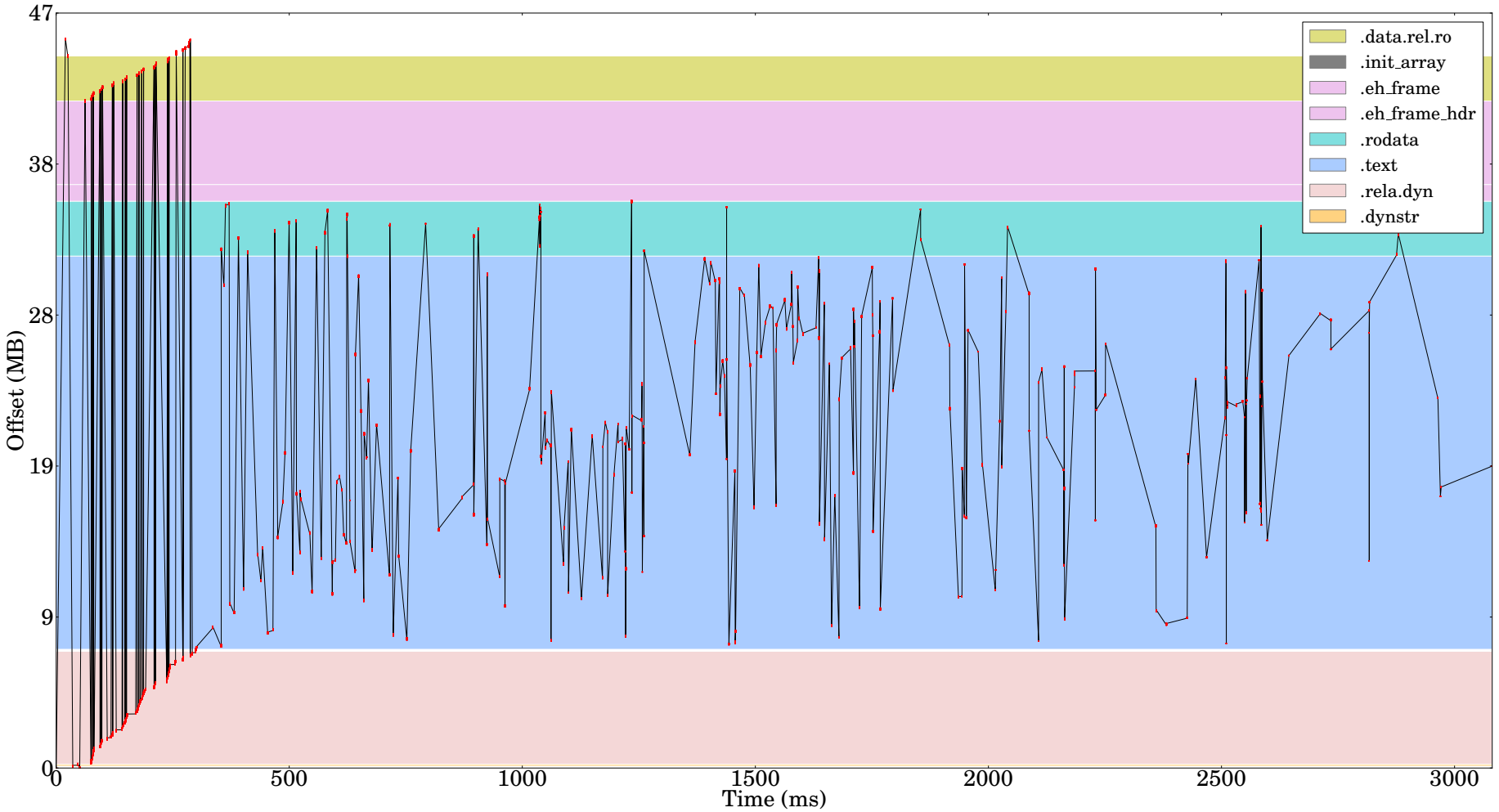
Decades old assumptions:

- Static variable initializers rarely take address of symbol.
- It is not too common to take address of function or static variable in code (only for `qsort`).
- Hot parts of programs are not in shared library.

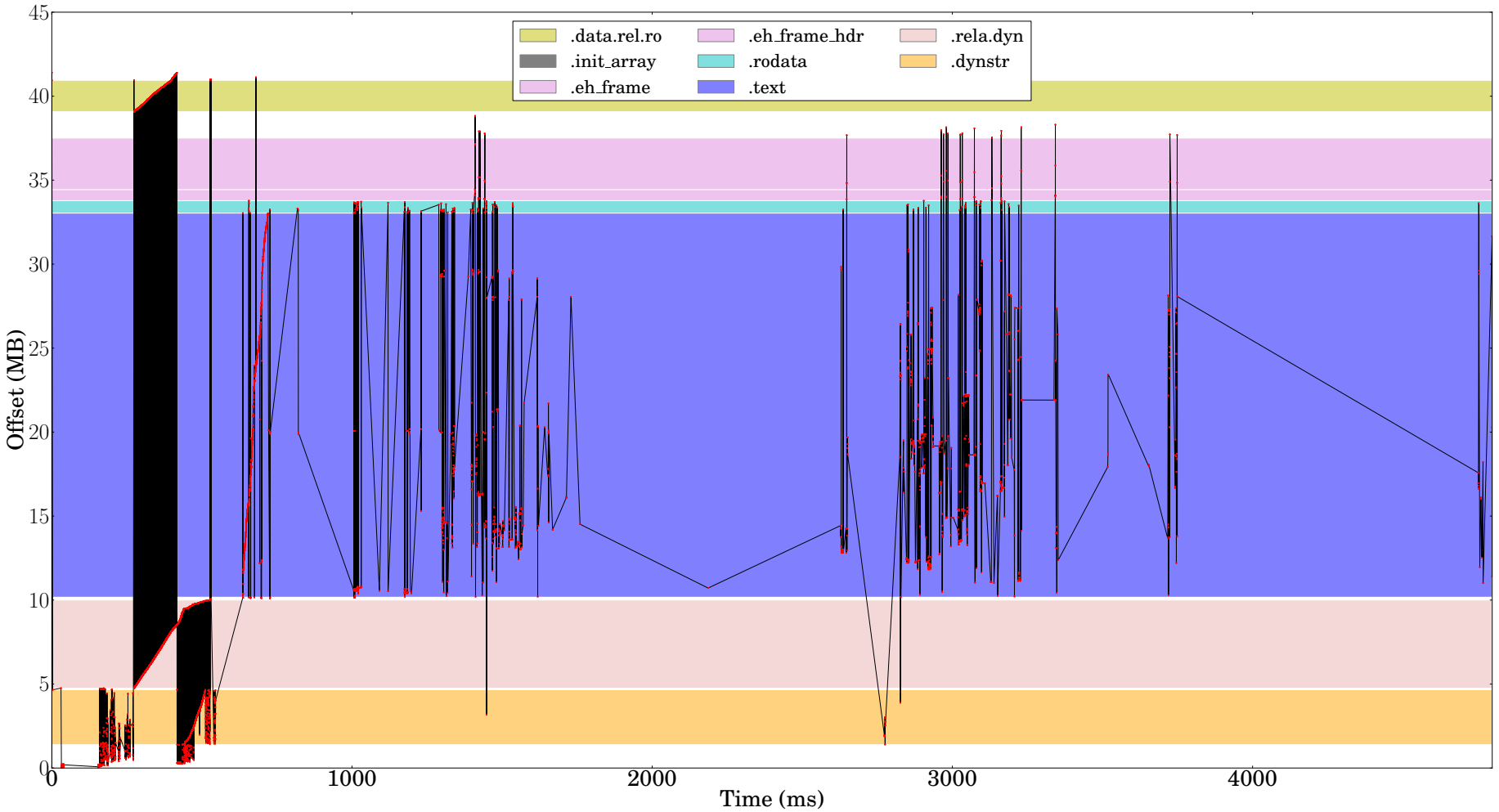
GIMP startup



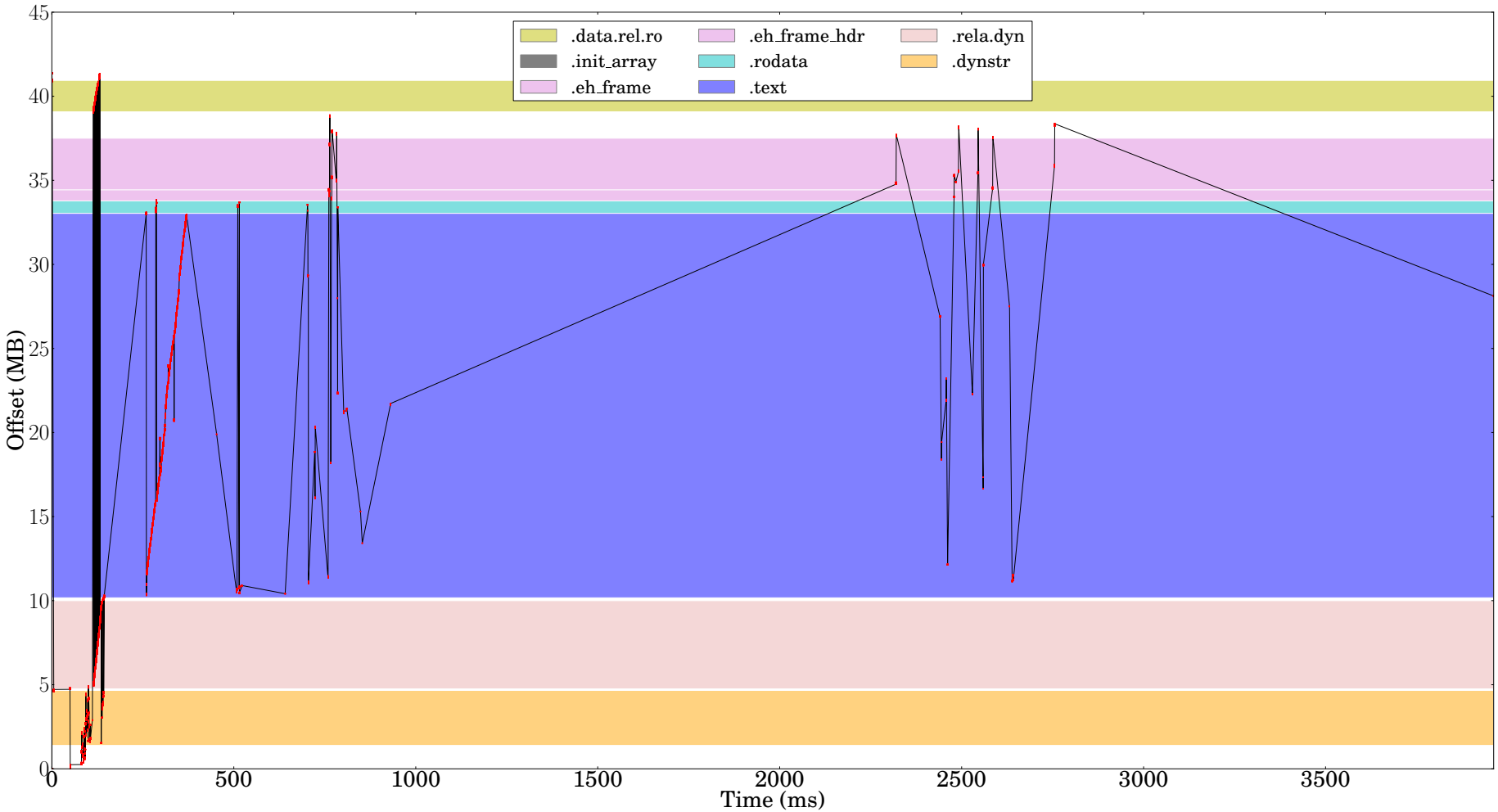
libxul startup



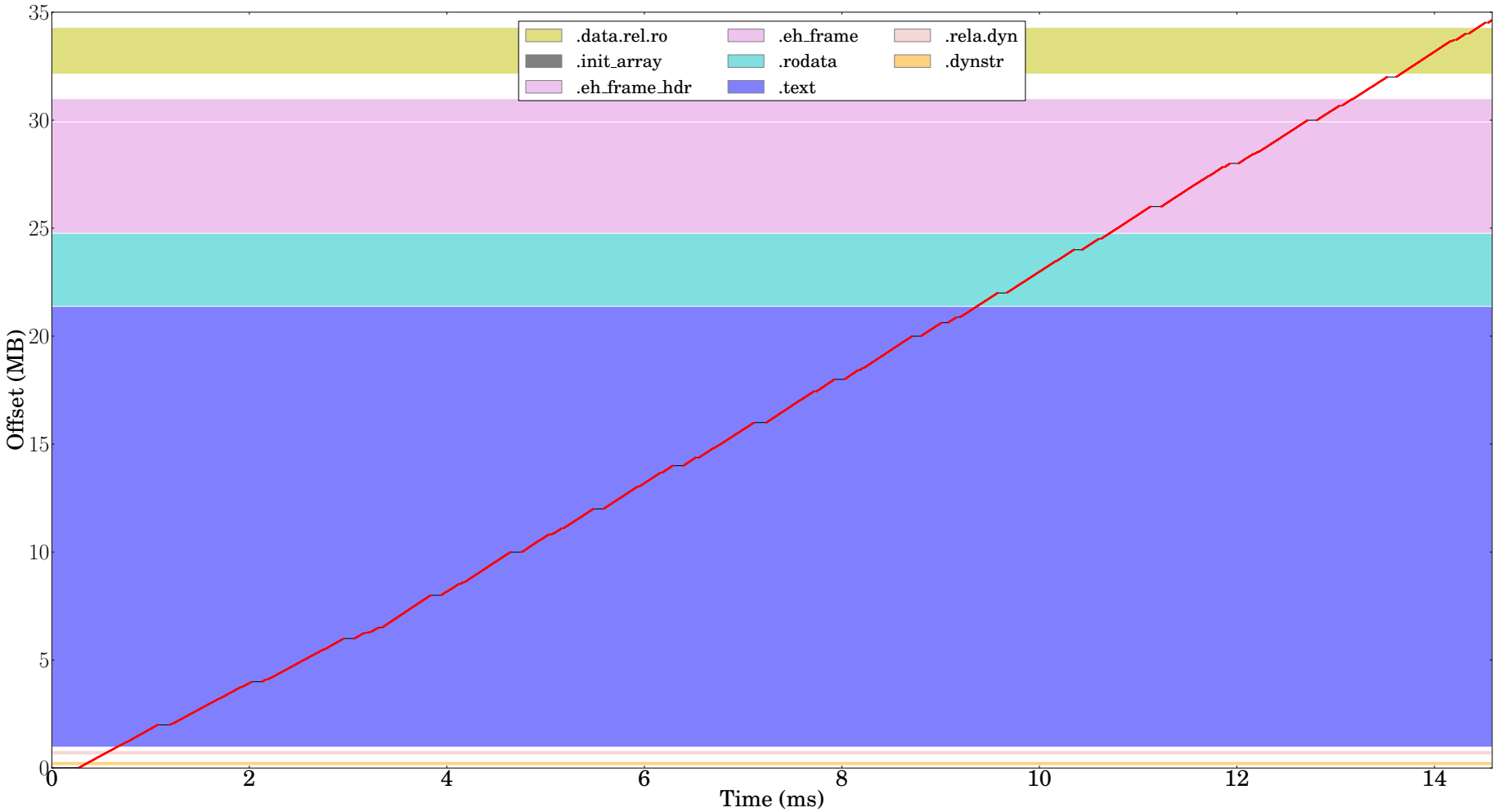
Libreoffice startup



Libreoffice startup, read-ahead enabled



libxul startup, startup problem solved



Kdeinit (2000)

- During login procedure start process `kdeinit` containing all the shared libraries
- Instead of executing an KDE application, fork `kdeinit` and dlopen.

Prelink (2004)

Tool developed by Jakub Jelínek

- After installation whole distro is walked, binaries are analyzed and conflict graph of shared libraries is built
- Shared libraries gets assigned fixed addresses in the address space
- Binaries are prelinked - i.e. linked with assumption that libraries are at given positions
- Dynamic linking is performed only when something changed from prelinking time. (fallback mode)

Prelink (2004)

Tool developed by Jakub Jelínek

- After installation whole distro is walked, binaries are analyzed and conflict graph of shared libraries is built
- Shared libraries gets assigned fixed addresses in the address space
- Binaries are prelinked - i.e. linked with assumption that libraries are at given positions
- Dynamic linking is performed only when something changed from prelinking time. (fallback mode)

Prelink offers great speedups, but has number of issues

- The prelinking modifies all binaries on disk making it difficult to detect changes, increasing fragmentation
- The fallback mode is triggered often (by `dlopen`, ...)

Speeding up dynamic linker runtime (2006)

C++ programs spent a lot of CPU time in dynamic linking comparing symbol names. It grows with num-libraries$\times$avg-symbol-length

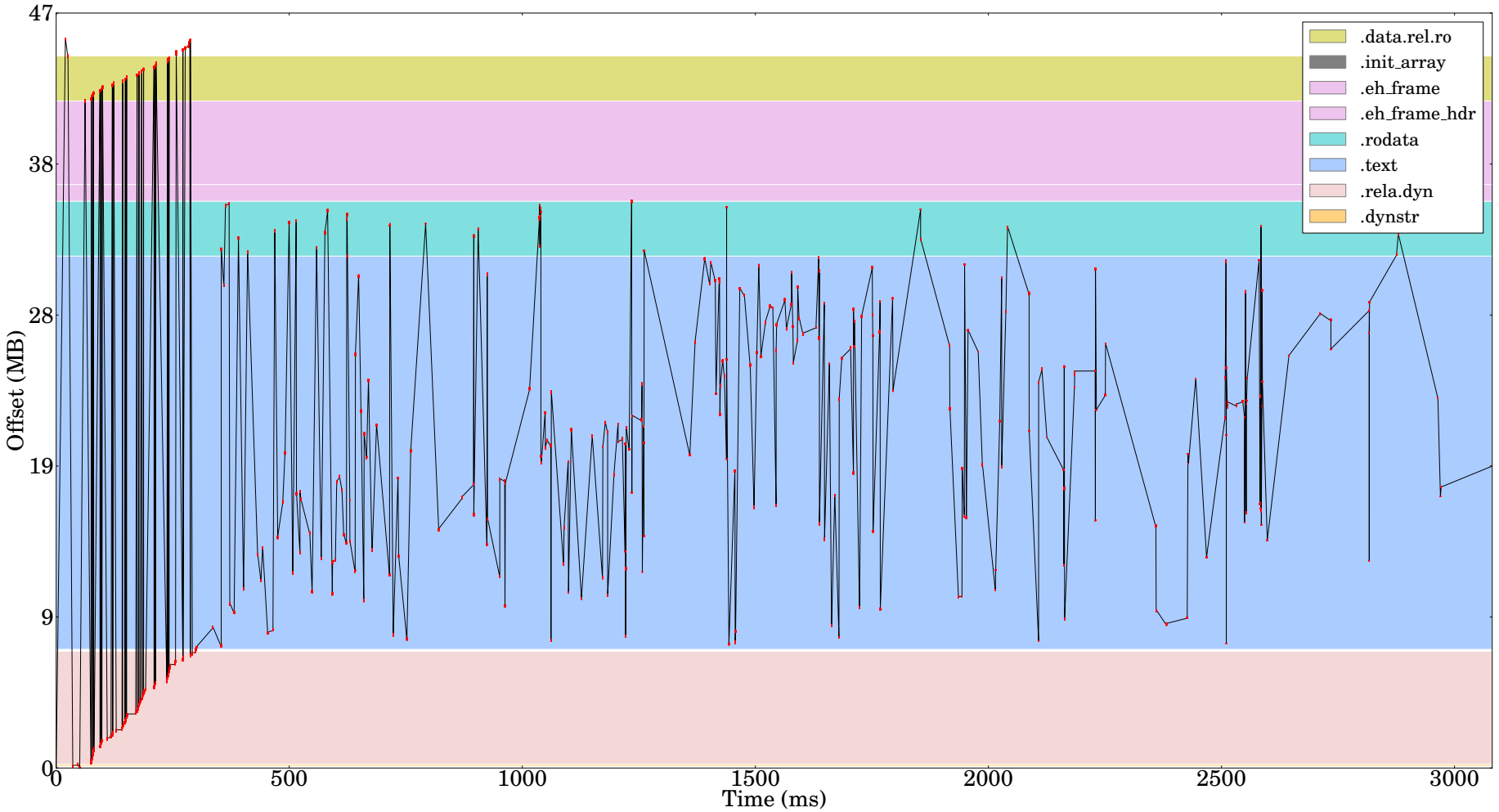
Speeding up dynamic linker runtime (2006)

C++ programs spent a lot of CPU time in dynamic linking comparing symbol names. It grows with num-libraries$\times$avg-symbol-length

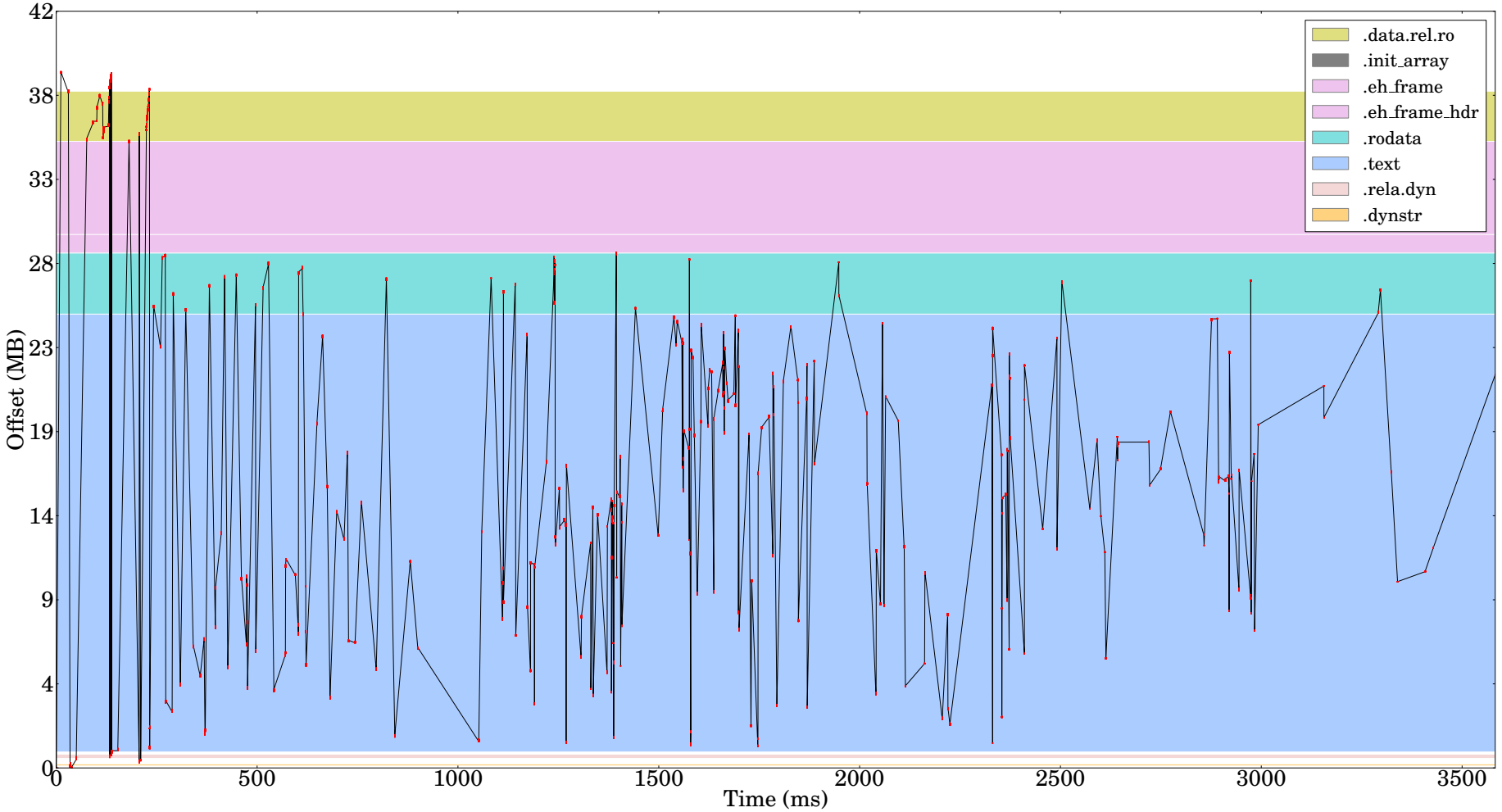
- `.gnu.hash` section
- 2-bit Bloom filter used for fast lookup if symbol is defined at all in a in given DSO.
- Stronger hash function for actual lookup
- Optimized `strcmp`

Overall GNU hash reduce about 15% of firefox dynamic linking time. (by `LD_DEBUG=statistic`)

libxul startup



libxul startup, elfhack applied



ELFhack = relocations on diet (2010)

Hack introduced by Mike Hommey

- 20% of Firefox `libxul` image are relocations
- 208k relocations out of 239k relocations are IP relative.
- ELF relocations are not terribly size optimized
  - REL relocations on x86 take 8 bytes
  - RELA relocation on x86-64 take 24 bytes

ELFhack = relocations on diet (2010)

Hack introduced by Mike Hommey

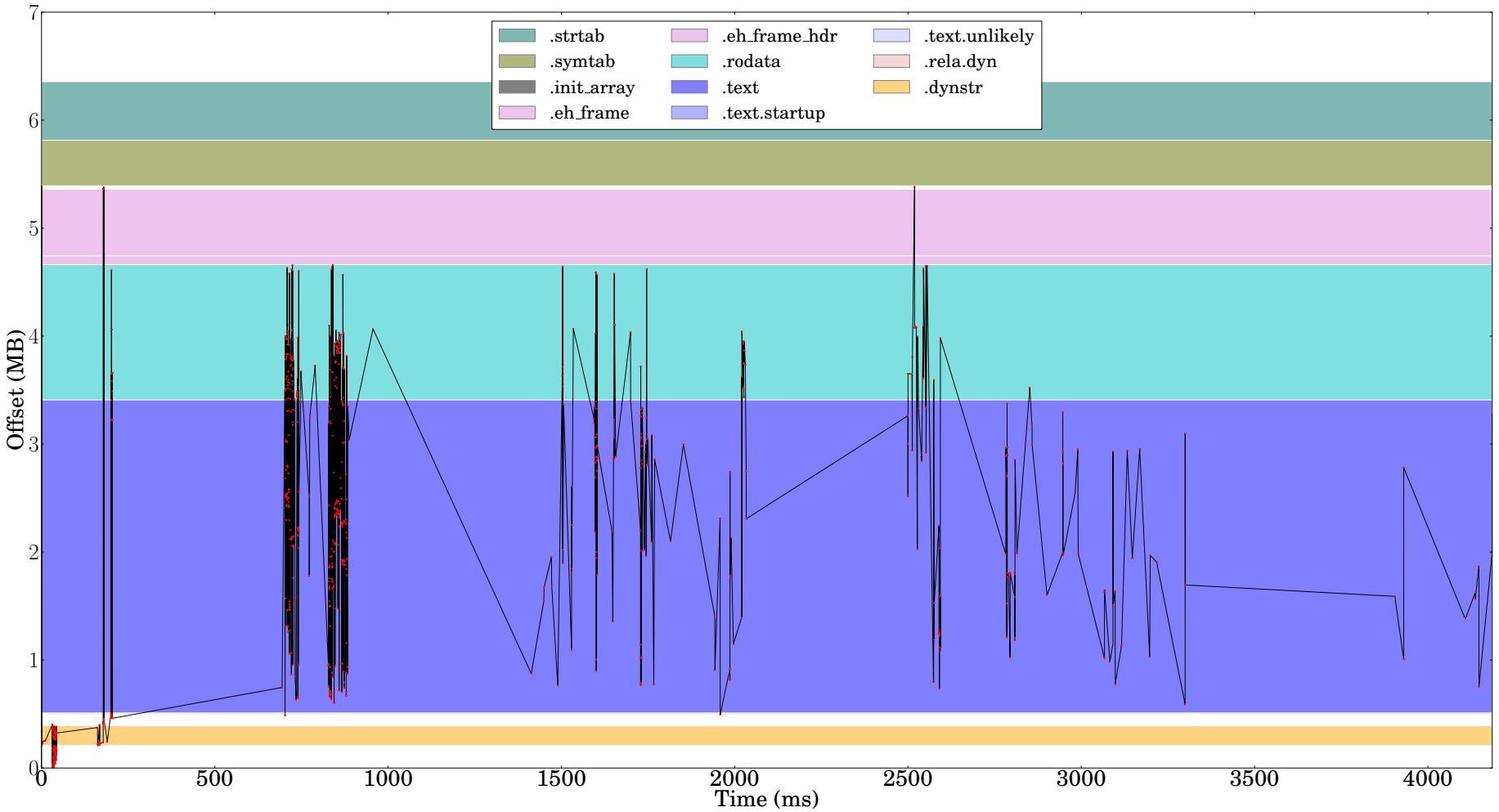
- 20% of Firefox `libxul` image are relocations
- 208k relocations out of 239k relocations are IP relative.
- ELF relocations are not terribly size optimized
  - REL relocations on x86 take 8 bytes
  - RELA relocation on x86-64 take 24 bytes
- Elfhack compress the relocations
  - ELFhack removes IP relative ELF relocations and store them in compact custom format. It handles well sequences of IP relative relocations in vtables.
  - After ELF linking, ELFhack linking completes the process.
  - ELFhack is general tool but not compatible with -z relro security feature.
- 7.5MB of relocations $\rightarrow$ 0.3MB.

Disabling page demand loading (2010)

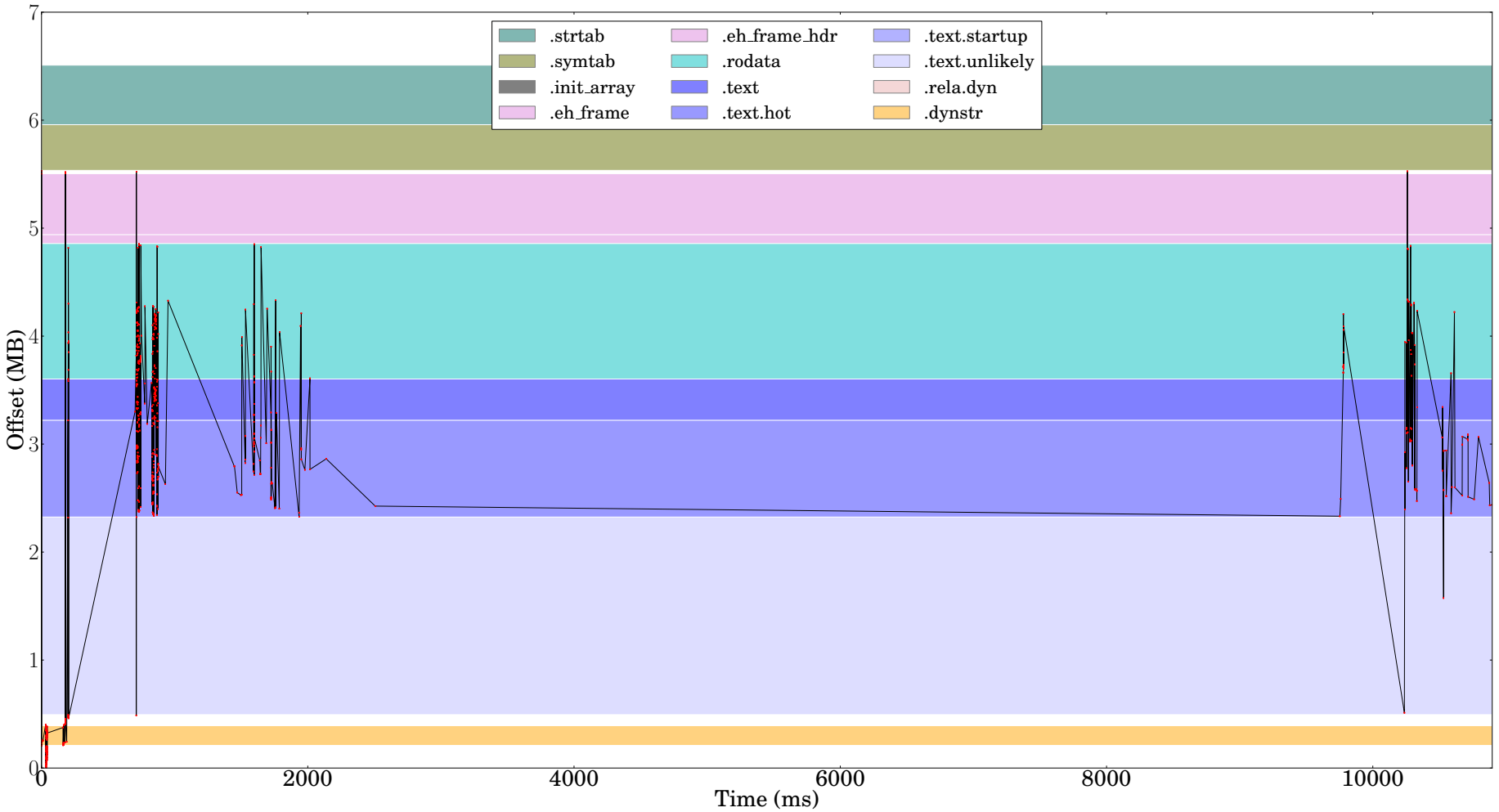
Current Firefox' solution to startup time problems

- Firefox startup touches almost every page in the binary
- Hacking dynamic linker to do `mmap` makes kernel to load it sequentially

GIMP startup



GIMP startup with subsections



Optimizing code placement (2004, 2010)

Split text section into:

- Hot subsection
- Normal subsection
- Unlikely executed subsection
- Startup only subsection (new in 2010), ordering solved by `.initarray`
- Exit only subsection (new in 2010)

Split data into

- readonly data
- data w/o relocations in it
- data with IP relative relocations
- readonly data with IP relative relocations
- data with all kinds of relocations
- readonly data with all kinds of relocations

The catch

Warning: no `gold` support until the next release of binutils

The catch

Warning: no `gold` support until the next release of binutils
Hot/cold code decisions are difficult for the compiler.
`-O2` $\Longleftrightarrow$ “everything may be hot unless you know it is not”

The catch

Warning: no `gold` support until the next release of binutils
Hot/cold code decisions are difficult for the compiler.
`-O2` $\Longleftrightarrow$ “everything may be hot unless you know it is not”

- Compiler heuristics to disprove that given code is hot are limited.

The catch

Warning: no `gold` support until the next release of binutils
Hot/cold code decisions are difficult for the compiler.
`-O2` $\Longleftrightarrow$ “everything may be hot unless you know it is not”

- Compiler heuristics to disprove that given code is hot are limited.
- Feedback directed optimization is feasible even for GUI apps.
  - It is used by firefox
  - GUI code is usually not the bottleneck, train the rest
  - Even not too representative data often works in practice

The catch

Warning: no `gold` support until the next release of binutils
Hot/cold code decisions are difficult for the compiler.
`-O2` $\Longleftrightarrow$ "everything may be hot unless you know it is not"

- Compiler heuristics to disprove that given code is hot are limited.
- Feedback directed optimization is feasible even for GUI apps.
  - It is used by firefox
  - GUI code is usually not the bottleneck, train the rest
  - Even not too representative data often works in practice
- `cold` and `hot` function attributes (2007)
  - Paths leading to calls to `cold` function are cold
  - Functions called only by cold functions are cold.
  - No use of `cold` attribute in `/usr/include` found :(

Feedback directed reordering (2013)

- Measure first time of function execution
- Order functions increasingly in time in the resulting binary

Feedback directed reordering (2013)

- Measure first time of function execution
- Order functions increasingly in time in the resulting binary
- Initial experiments by Taras Glek with hacked valgrind

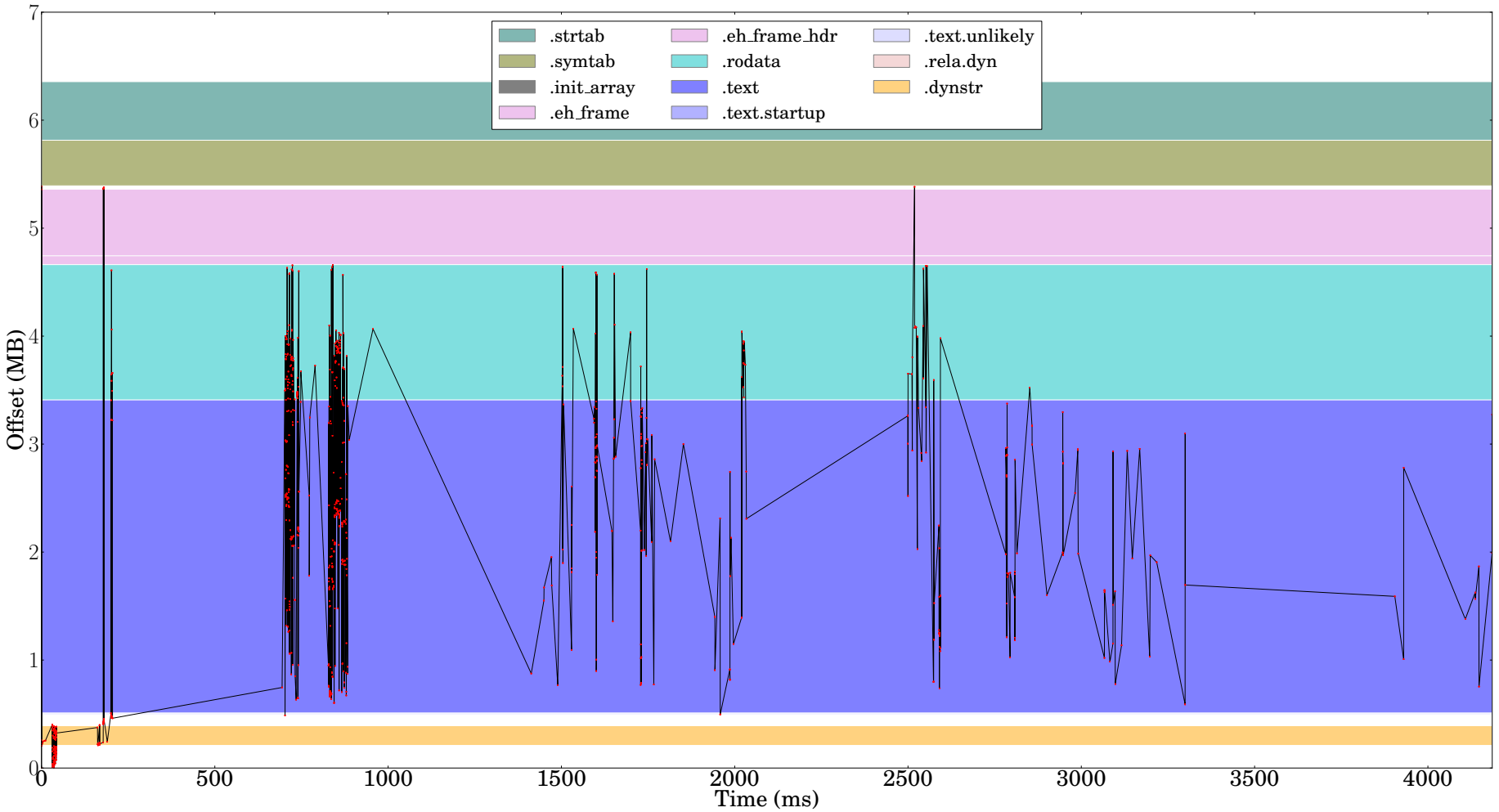
Feedback directed reordering (2013)

- Measure first time of function execution
- Order functions increasingly in time in the resulting binary
- Initial experiments by Taras Glek with hacked valgrind
- Implemented to GCC FDO by Martin Liška as his thesis and SoC project
  (will be merged into GCC 4.9 soon)

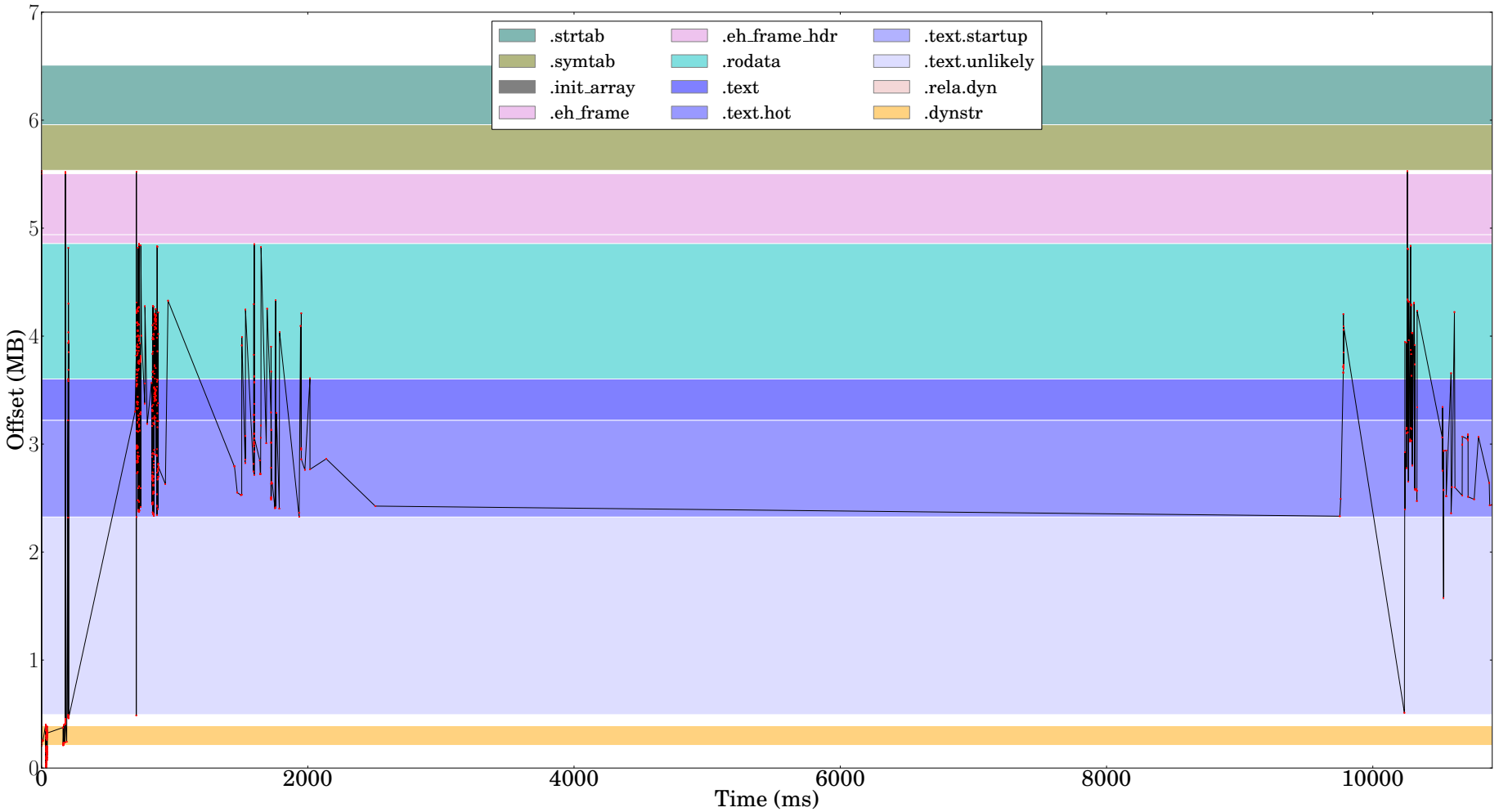
Feedback directed reordering (2013)

- Measure first time of function execution
- Order functions increasingly in time in the resulting binary
- Initial experiments by Taras Glek with hacked valgrind
- Implemented to GCC FDO by Martin Liška as his thesis and SoC project
  (will be merged into GCC 4.9 soon)
- Currently needs linktime optimization. For non-LTO use needs linker support that is being discussed.

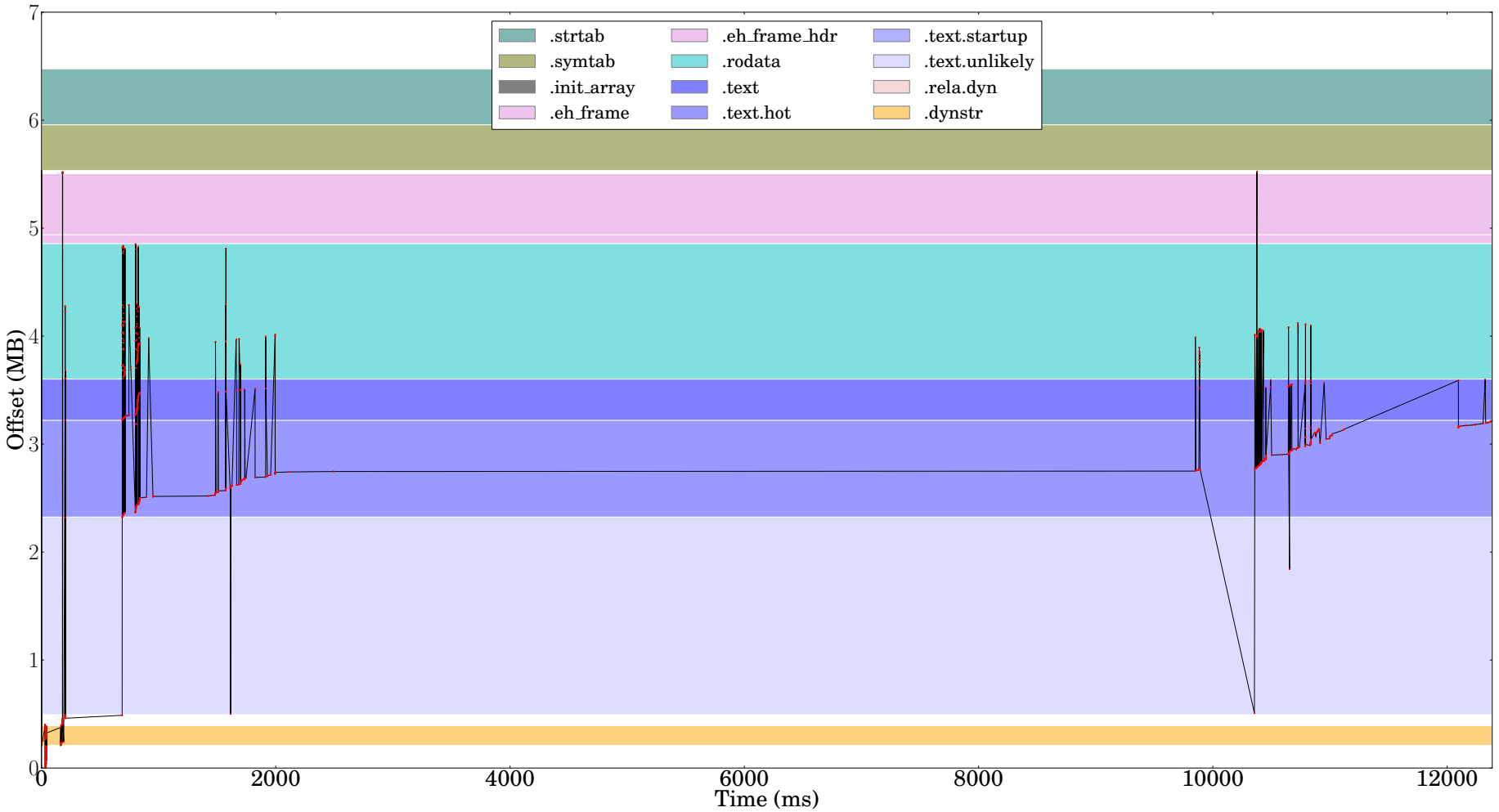
GIMP startup



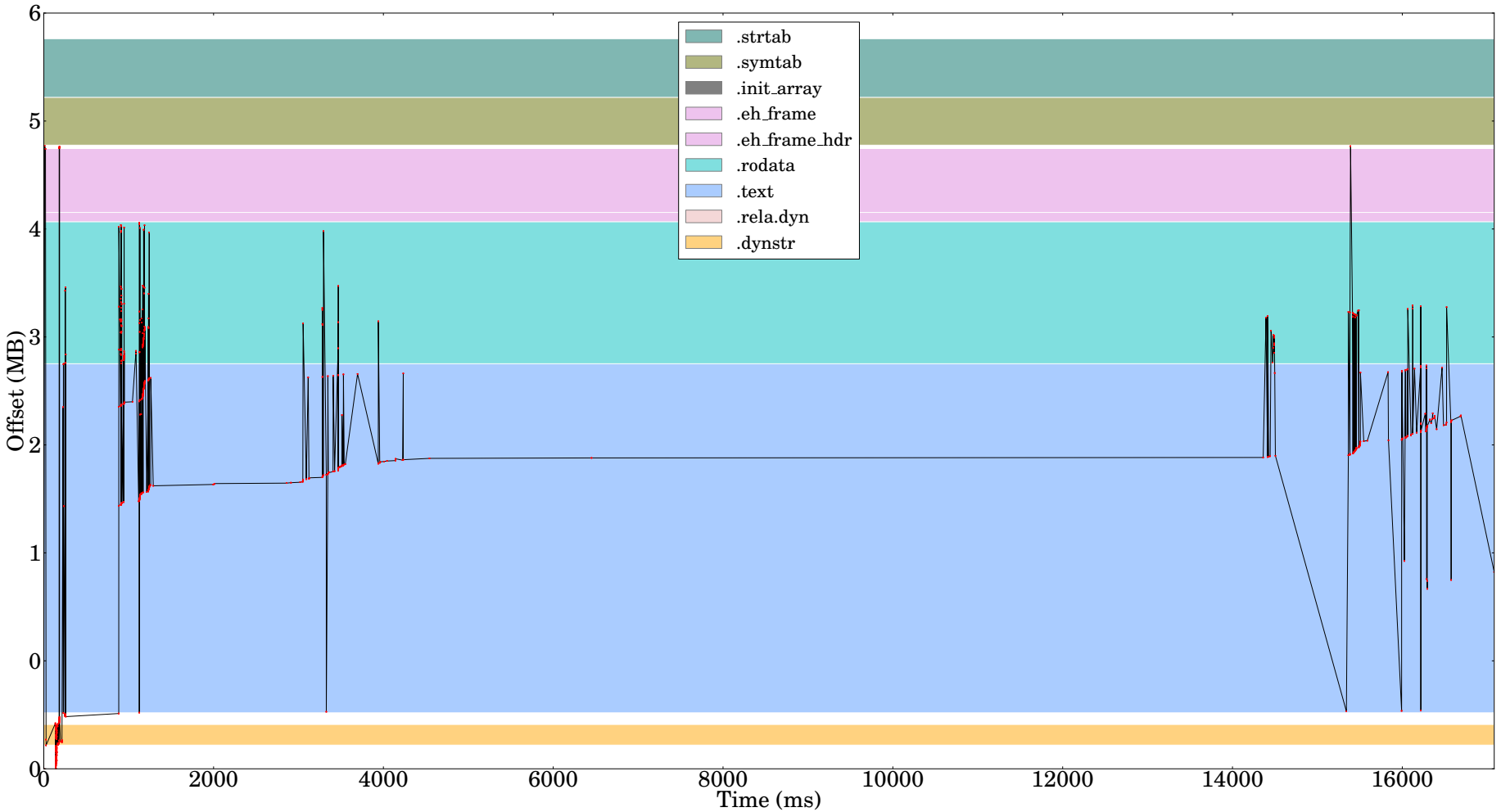
GIMP startup with subsections



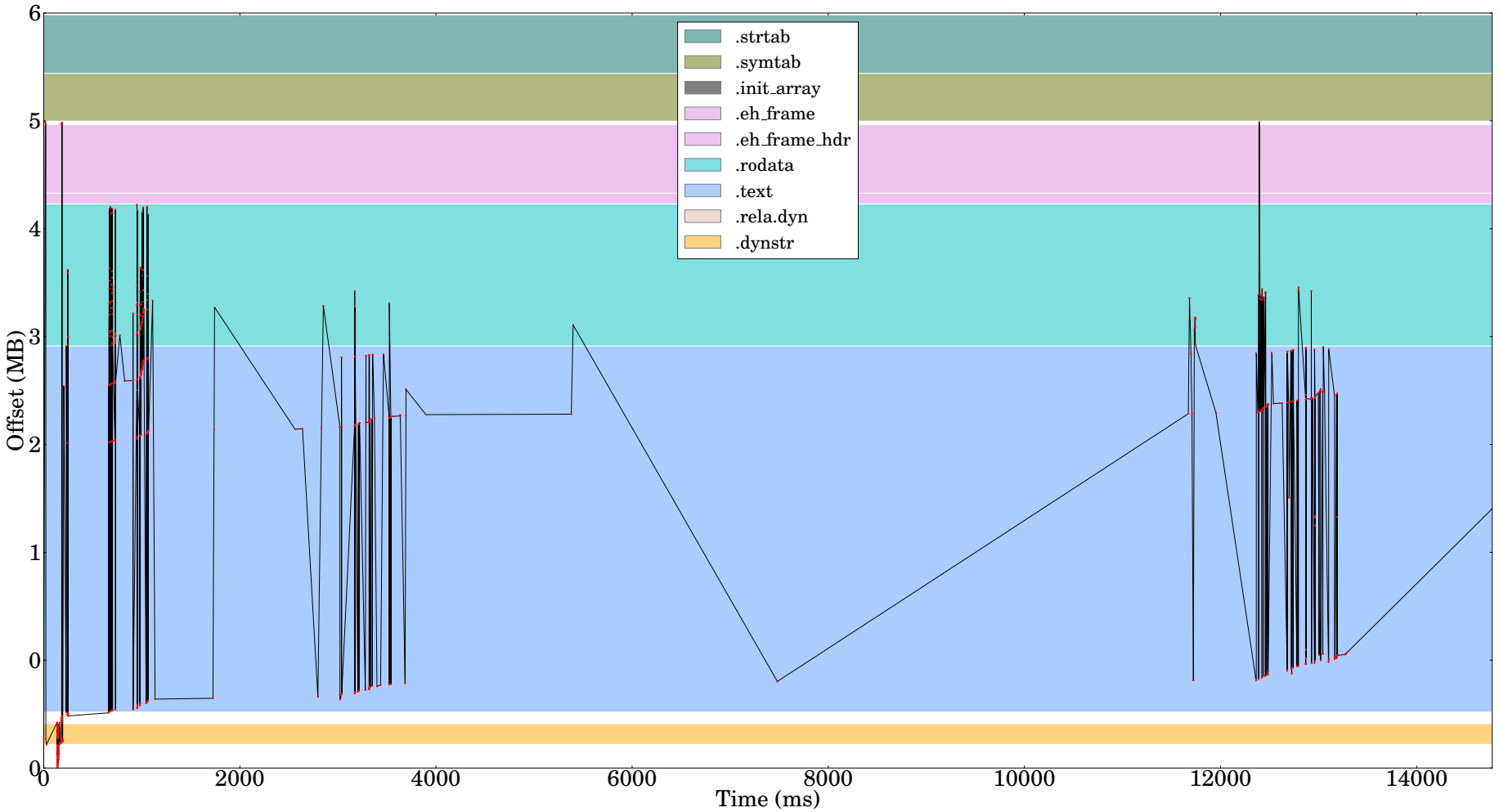
GIMP startup with reordering



Inkscape with reordering



Inkscape with function splitting



Refined code placement (2013+)

- Ordering by invocation time is cool for startup
- For main execution it is good to minimize call distance
  - Reverse postorder is a good first try, but too simplistic
  - Clustering program by edges ordered by frequency ignore indirect calls. Experiments from 2010 did not show any benefits over RPO.
- Profile data needs to be complete (work currently in progress)
  - Better support for COMDAT functions
  - Crossmodule indirect call profiling
  - Profiling of thunks
- We plan to experiment with algorithm starting from invocation time order performing local optimizations to minimize hot calls

Static code placement (2013)

We need non-profile based code placement algorithm

- Main problem seems to be lack on information on indirect call
  - Polymorphic call target analysis implemented last week
  - Normal indirect call are in minority, can be pruned by types and points-to
- Maybe we need global profile propagation for educated guesses on what is hot call edge.
- We plan to honor original program order as a starting point.
  - Is it better than reverse postorder or random order?

Linktime optimization overview

Link time optimization (LTO) extends the scope of interprocedural analysis from single source file to whole program visible at the link time

Linktime optimization overview

Link time optimization (LTO) extends the scope of interprocedural analysis from single source file to whole program visible at the link time

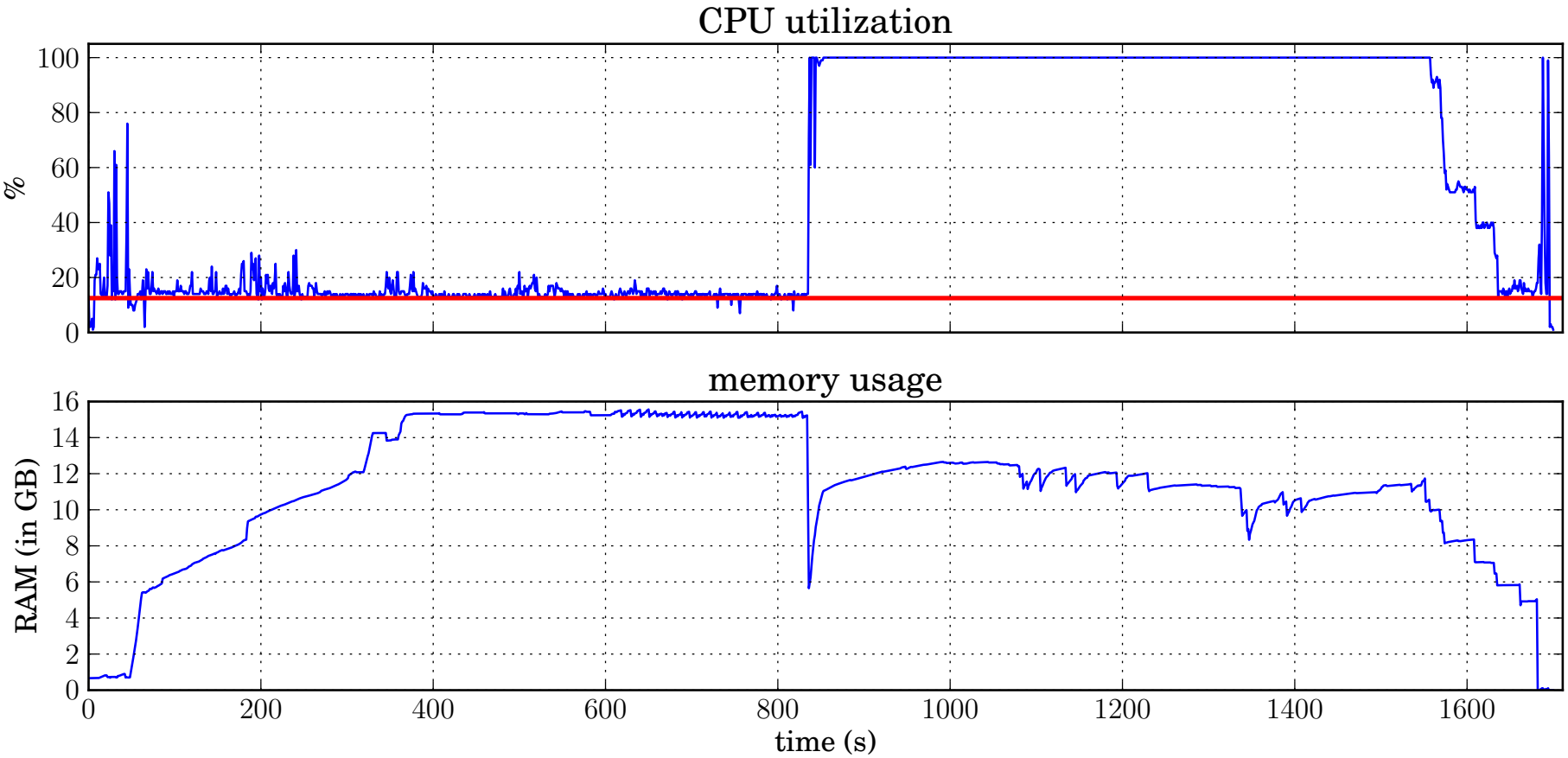
- Implemented by calling back to the optimizer backend from the linker.
- Development started in 2005, merged to mainline in 2009.
- First released in GCC 4.5.

What can be built

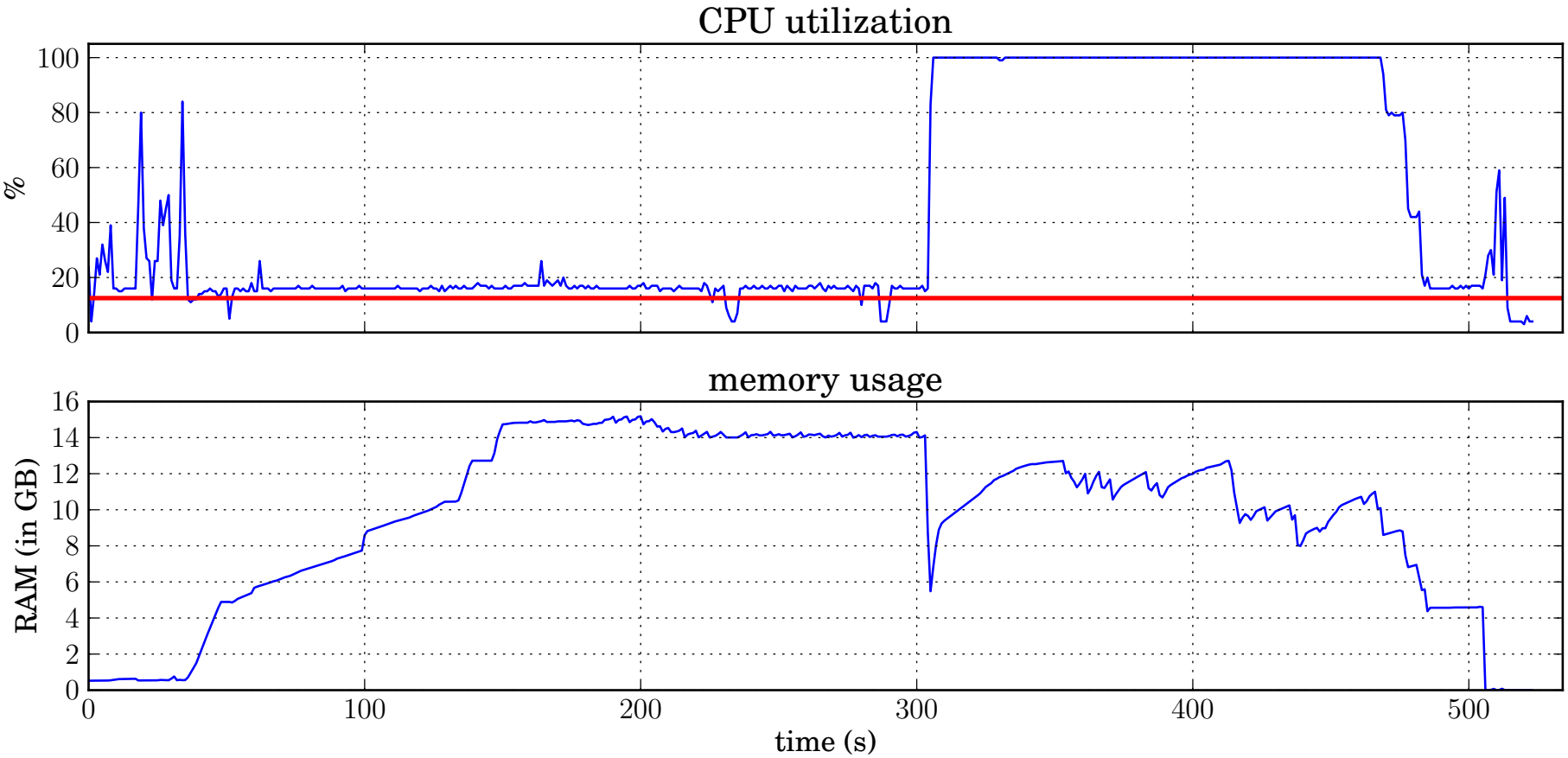
- GCC itself (GCC 4.5+)
- SPEC2k6 benchmarks (GCC 4.5+)
- Firefox (GCC 4.7+)
- Kernel (thanks to Andi Kleen, GCC 4.8+)
- Chrome and Libreoffice (thanks to Martin Liška GCC 4.9+)

Minor patches usually needed for symbols used from ASM statements. Major hacks needed for kernel.

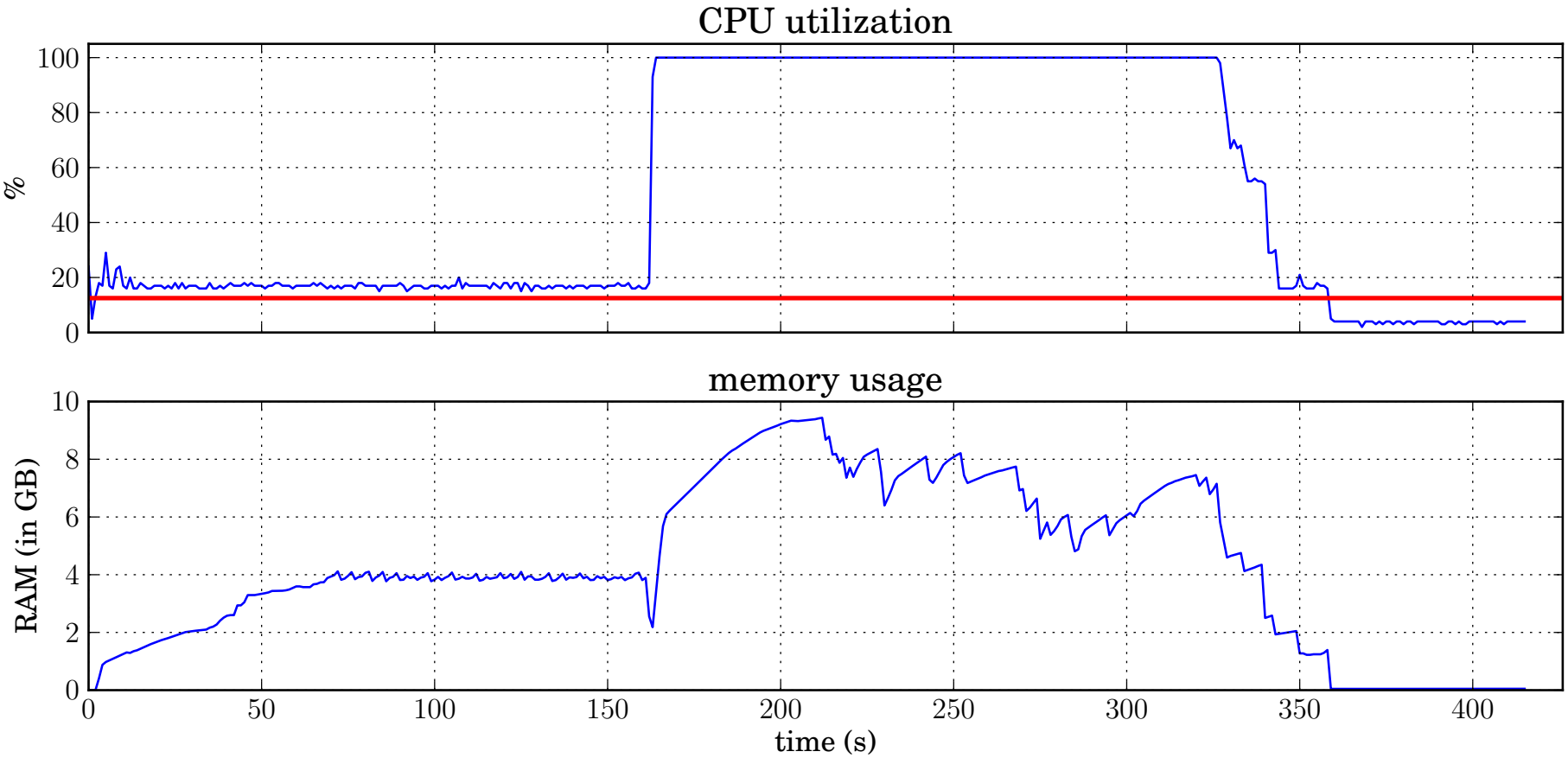
Memory/CPU usage during Firefox build



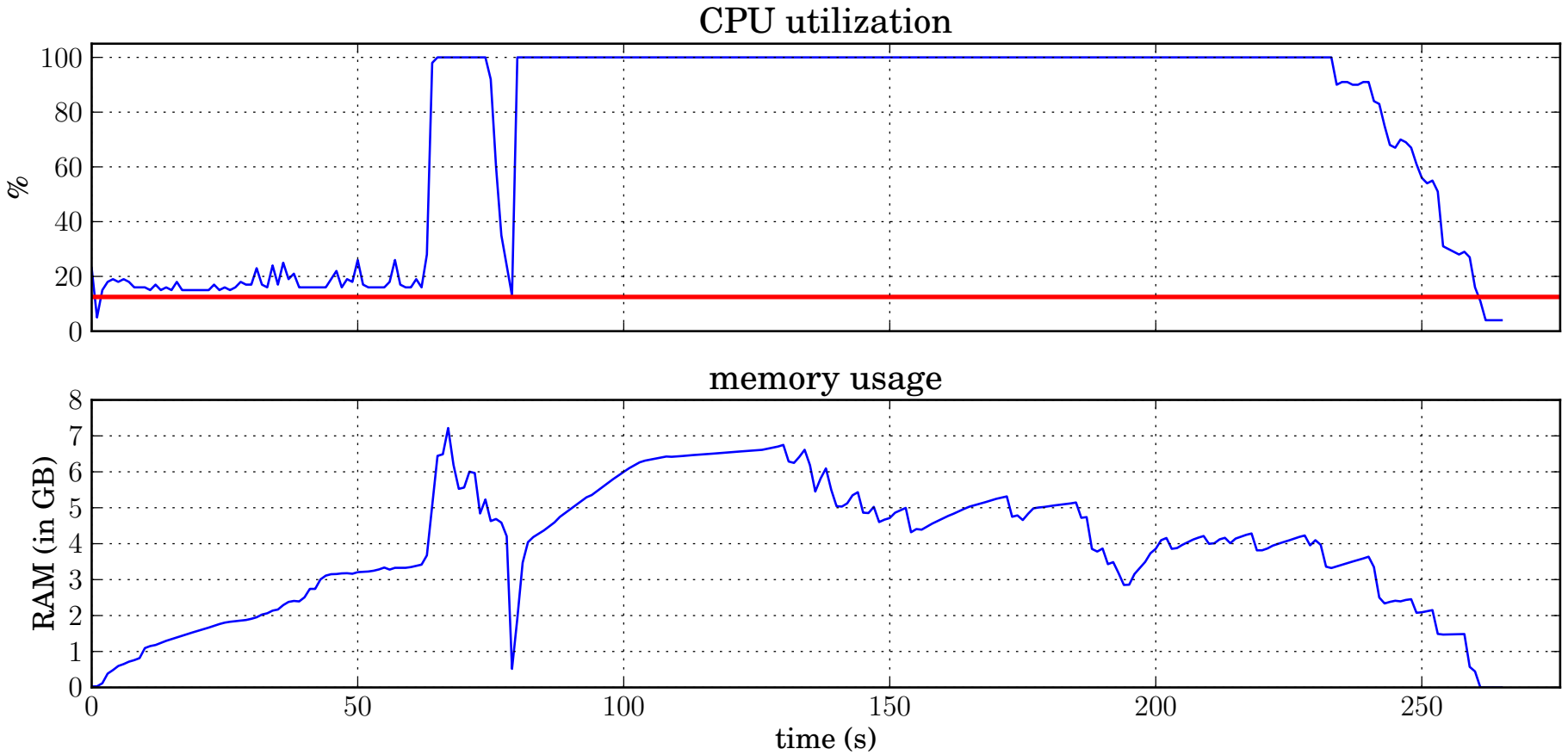
After type merging rewrite by Richard Biener



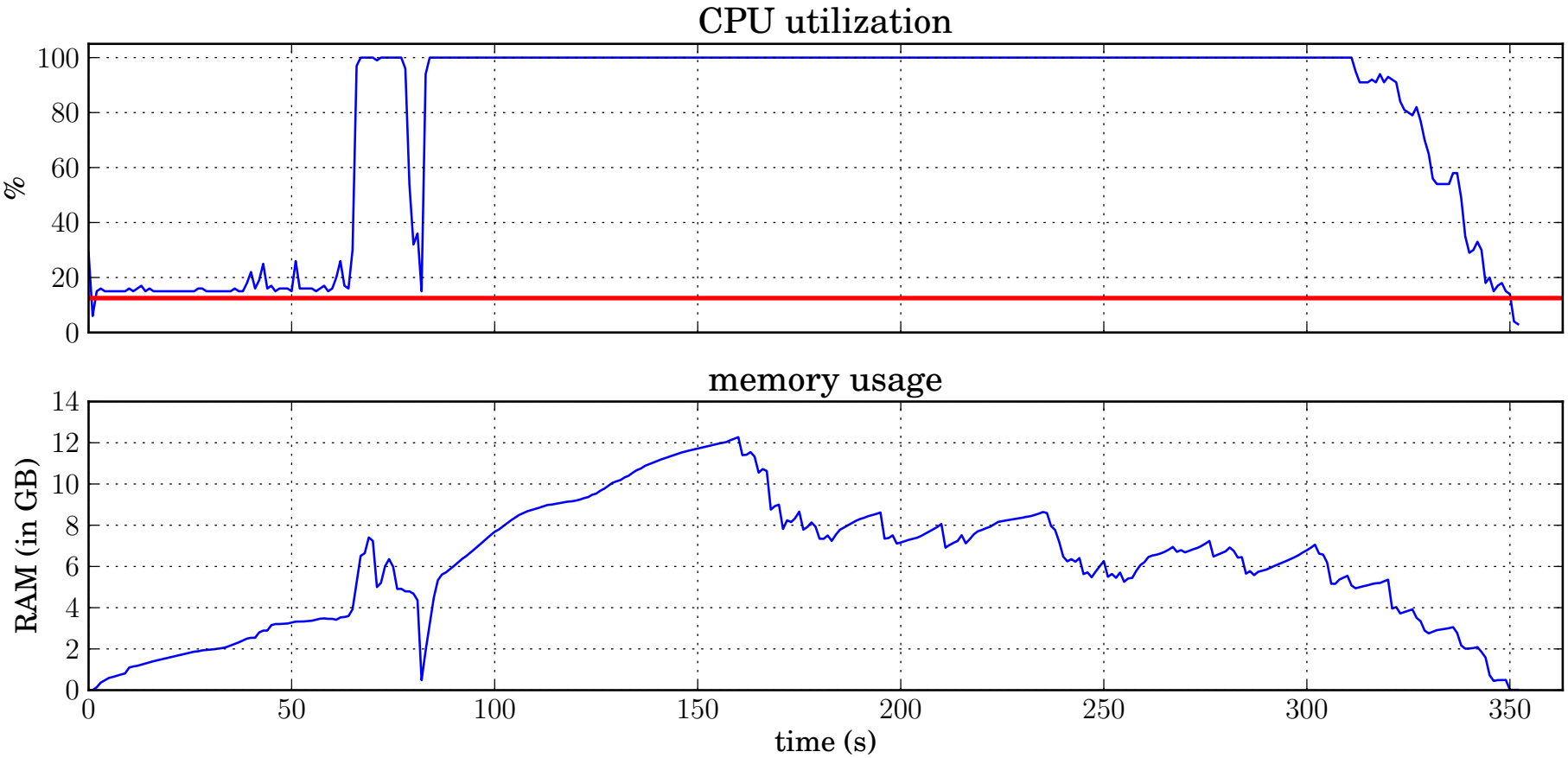
After early virtual method removal



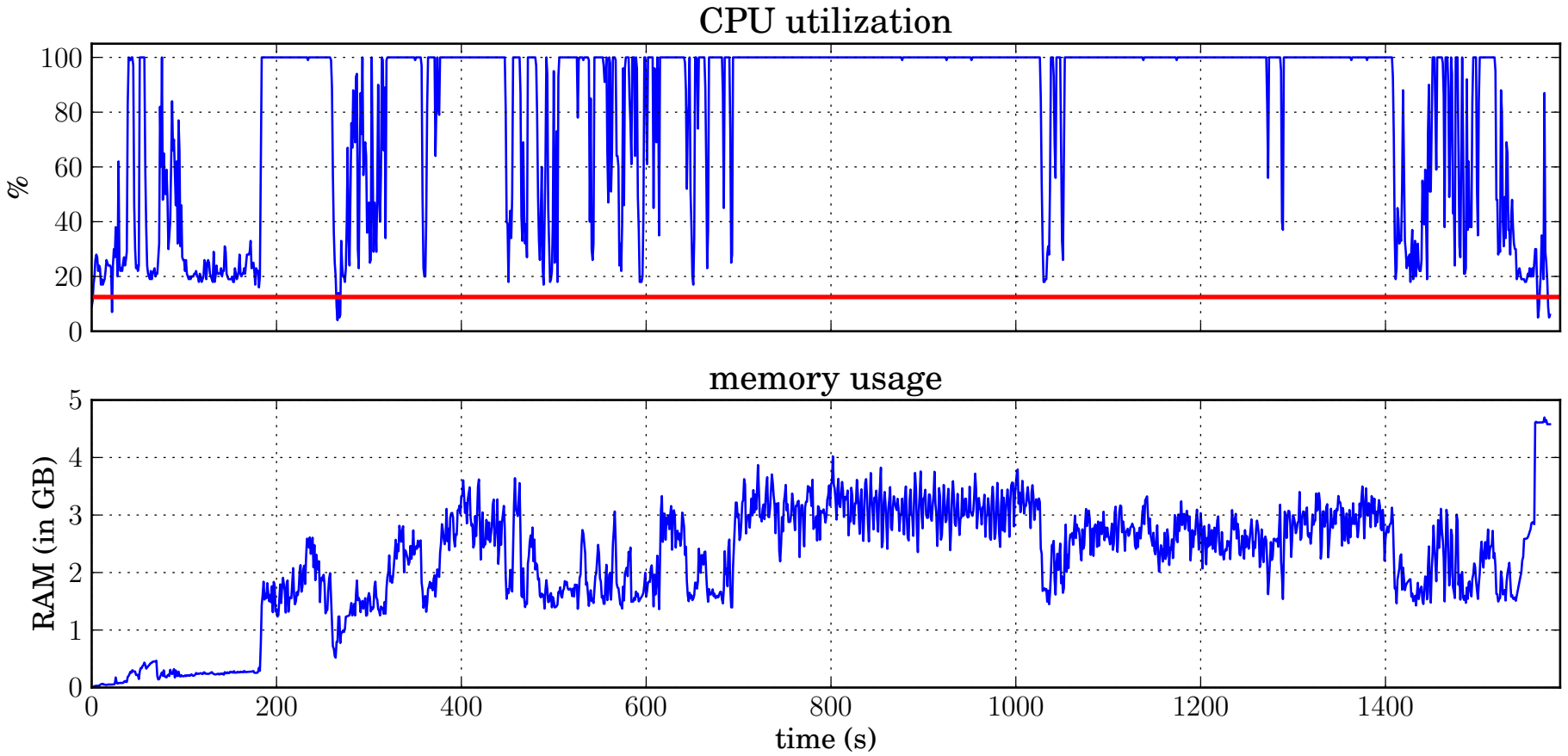
With better partitioning and parallel streaming



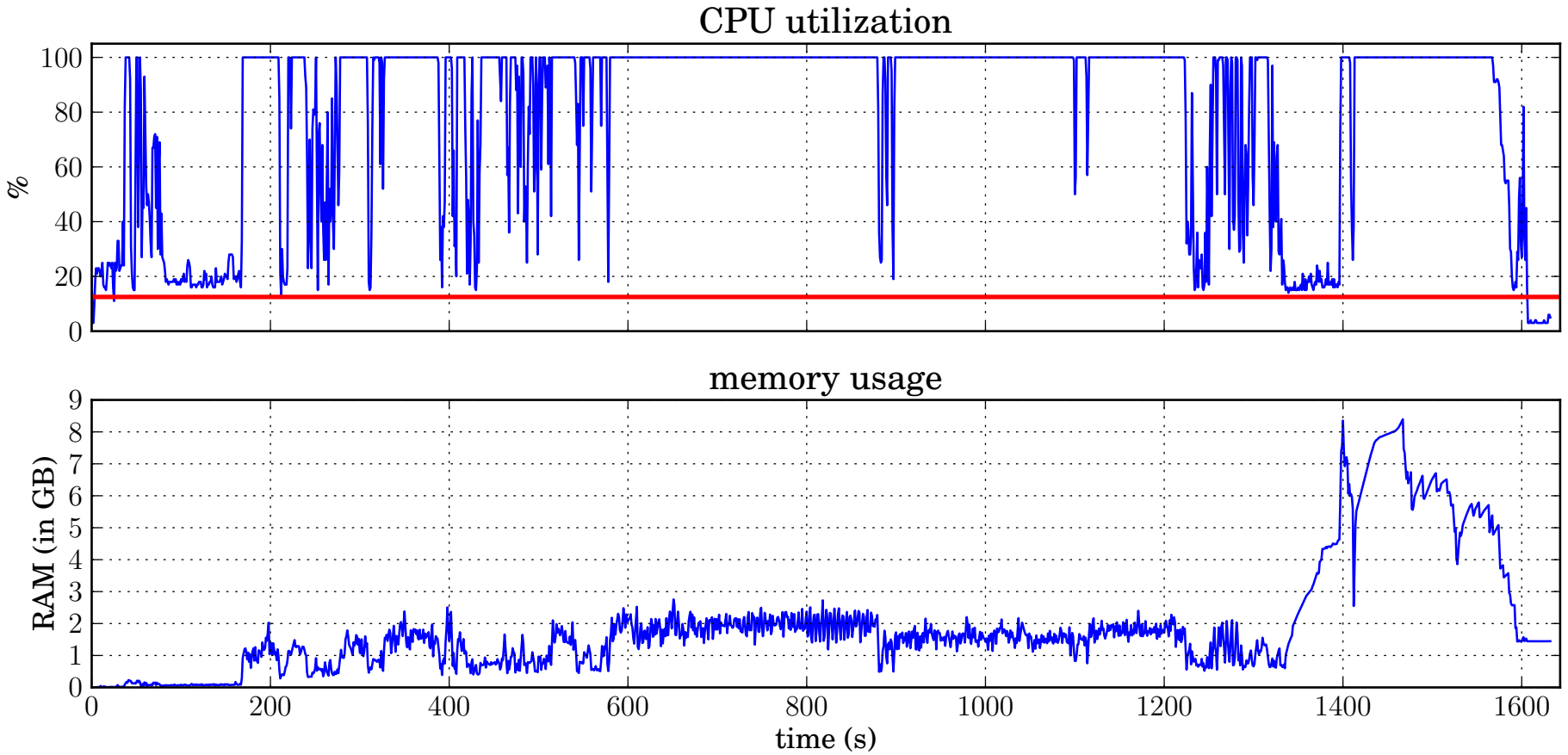
With full debug info



Full build — no-LTO, -O2



Full build — LTO, 6% slower



LTO performance and code size

SPEC2006 relative to -O2;

PGO = Profile Guided Optimization
`-fprofile-generate/-fprofile-use`

	size (FP)	size (INT)	speed (FP)	speed (INT)
-O3	22.92%	14.20%	6.77%	1.41%

LTO performance and code size

SPEC2006 relative to -O2;

PGO = Profile Guided Optimization
`-fprofile-generate/-fprofile-use`

	size (FP)	size (INT)	speed (FP)	speed (INT)
-O3	22.92%	14.20%	6.77%	1.41%
-O2+LTO	-19.76%	-17.46%	1.51%	1.23%
-O3+LTO	3.27%	-1.65%	9.82%	5.62%

LTO performance and code size

SPEC2006 relative to -O2;

PGO = Profile Guided Optimization
`-fprofile-generate/-fprofile-use`

	size (FP)	size (INT)	speed (FP)	speed (INT)
-O3	22.92%	14.20%	6.77%	1.41%
-O2+LTO	-19.76%	-17.46%	1.51%	1.23%
-O3+LTO	3.27%	-1.65%	9.82%	5.62%
-O3+PGO	7.43%	11.75%	8.35%	8.21%
-O3+PGO+LTO	-4.68%	-11.39%	12.41%	12.16%

LTO performance and code size

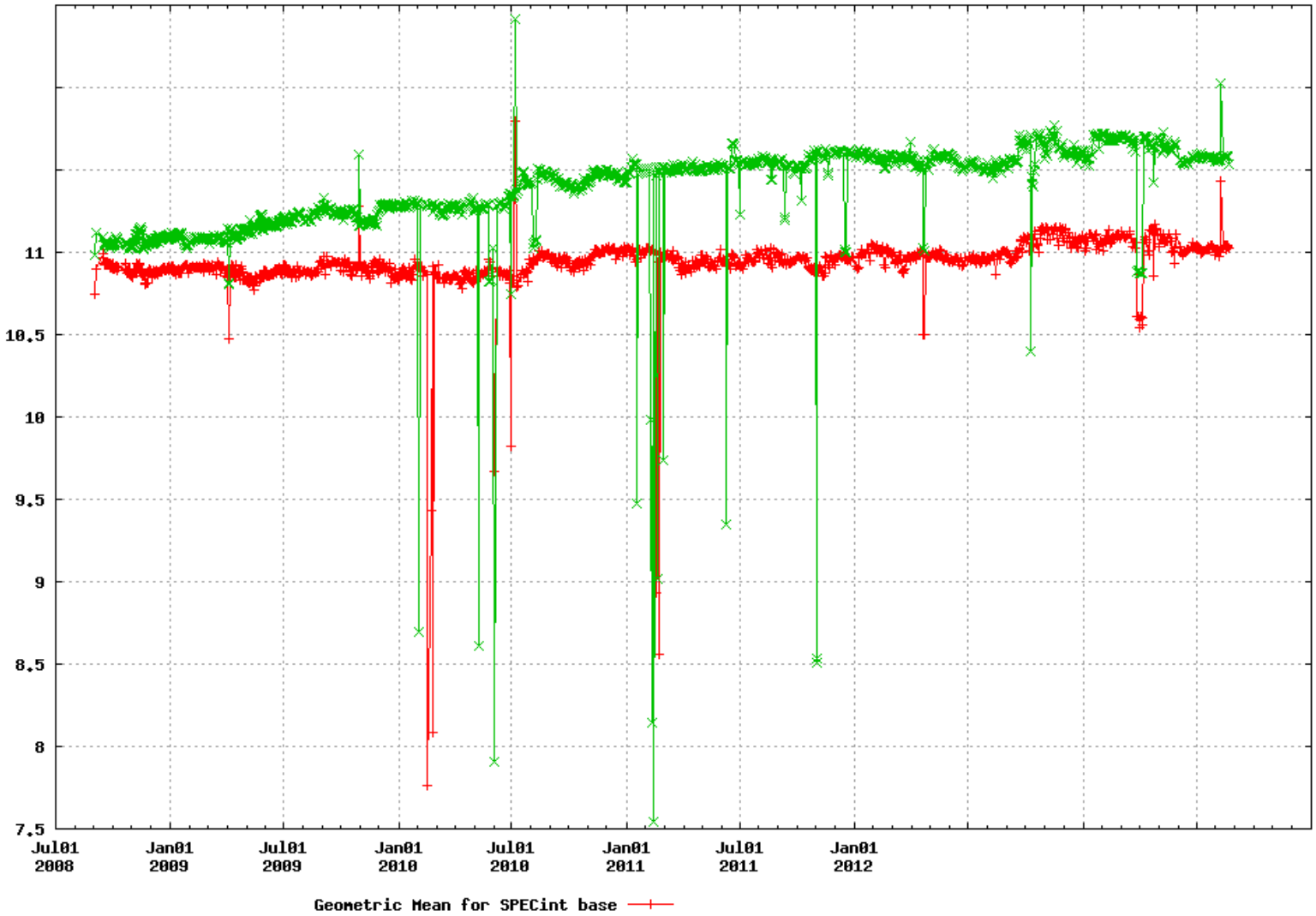
SPEC2006 relative to -O2;

PGO = Profile Guided Optimization
`-fprofile-generate/-fprofile-use`

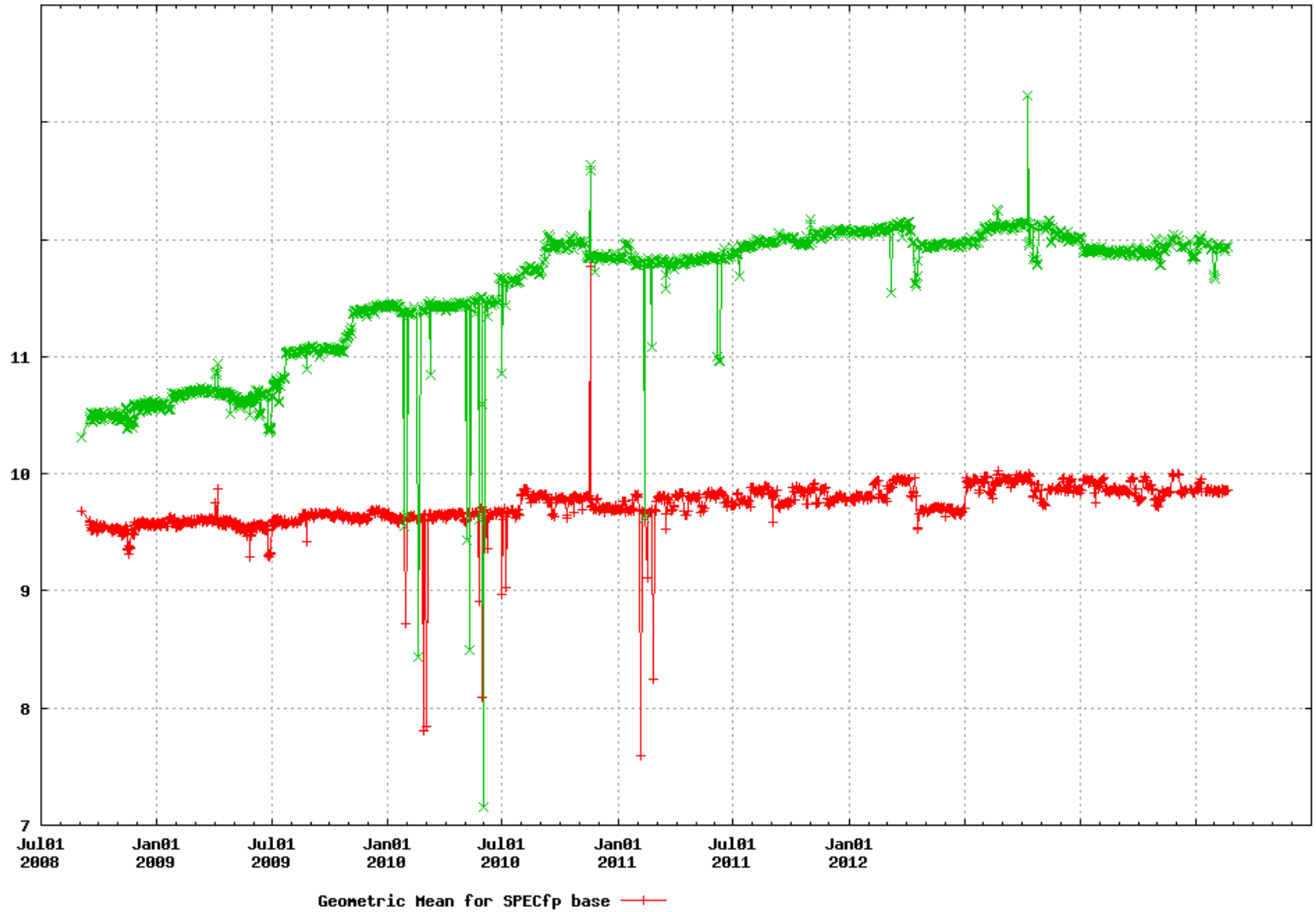
	size (FP)	size (INT)	speed (FP)	speed (INT)
-O3	22.92%	14.20%	6.77%	1.41%
-O2+LTO	-19.76%	-17.46%	1.51%	1.23%
-O3+LTO	3.27%	-1.65%	9.82%	5.62%
-O3+PGO	7.43%	11.75%	8.35%	8.21%
-O3+PGO+LTO	-4.68%	-11.39%	12.41%	12.16%
UG5	-1.23%	-9.67%	9.29%	3.77%

UG5 = `-O3, -flto -param inline-unit-growth=5%`

SpecINT2k6 non-LTO rates (-O2; -Ofast +5%)



SpecFP2k6 non-LTO rates (-O2 +3%; -Ofast, +14%)



LTO only optimization

What matters:

- Aggressive unreachable code removal
  (15%–20% code size savings)
- Cross-module inlining
  (almost all spec2k6 speedups come from it)
- Function reordering
  (over 20% fewer pages read at gimp startup, currently works well only with profile)

LTO only optimization

What matters:

- Aggressive unreachable code removal
  (15%–20% code size savings)
- Cross-module inlining
  (almost all spec2k6 speedups come from it)
- Function reordering
  (over 20% fewer pages read at gimp startup, currently works well only with profile)
- Cross-module indirect call profiling
  (important for programs with many polymorphic calls)

LTO only optimization

What matters:

- Aggressive unreachable code removal
  (15%–20% code size savings)
- Cross-module inlining
  (almost all spec2k6 speedups come from it)
- Function reordering
  (over 20% fewer pages read at gimp startup, currently works well only with profile)
- Cross-module indirect call profiling
  (important for programs with many polymorphic calls)
- Constructor/destructor merging
  (C++ only, measurable at firefox startup time)
- Identical function merging
  (Work in progress by Martin Liška, ICF in gold)

LTO only optimization

What is on the way

- Reducing program growth to 5% for larger LTO builds (5-20% code size savings)
- Getting rid of external relocations when C++ allows it Seems to help to libreoffice. Do we want `-fno-semantic-interposition` flag?
- Static function reordering
- Type inheritance analysis, devirtualization, speculative devirtualization
- `-fno-fat-lto-objects` by default

Problem still unresolved

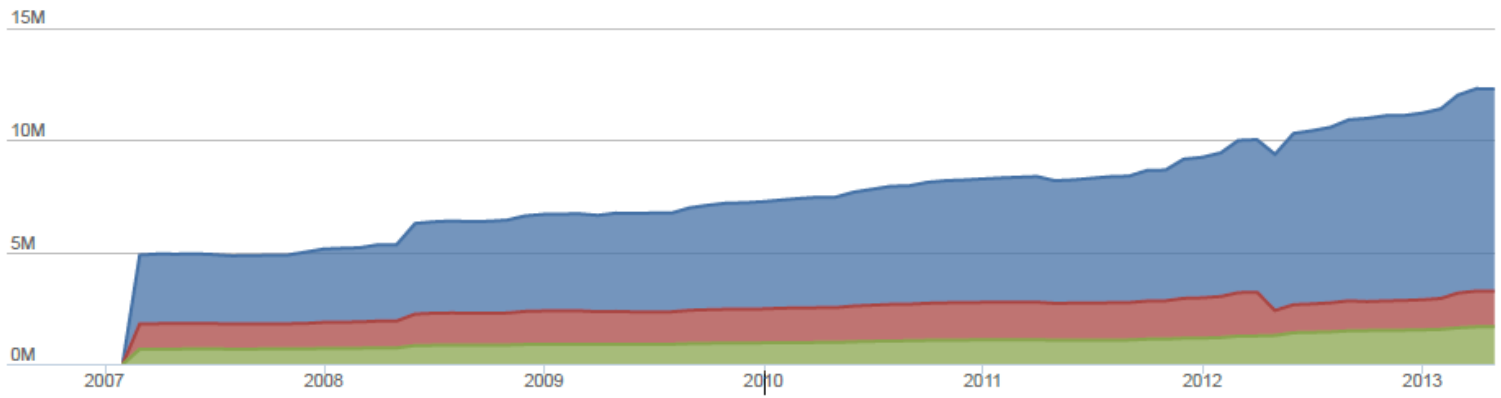
- Command line options behave in unexpected ways
  be sure to LTO only stuff that needs one global optimization setting. Do not LTO modules that needs specific flags (like `-march`, `-ffast-math` or so)
- There is no way to define symbols from asm statements in LTO units
  do not LTO these as workaround
- Debug info quality is not at match with non-LTO path
  (it gets better though)

Compared to 2010

	compile	serial link	parallel link	LTO cost	binary
2010	9m	4m 27s; 4GB	1m 03s	70%	21m
2013	35m	2m 2s; 4GB	5m 30s	6%	47m

Compared to 2010

	compile	serial link	parallel link	LTO cost	binary
2010	9m	4m 27s; 4GB	1m 03s	70%	21m
2013	35m	2m 2s; 4GB	5m 30s	6%	47m



Thank you!

Questions?