Google

V8 Garbage Collection

Reducing Memory Consumption and Latency

Hannes Payer
Chrome/V8 Team

Overview

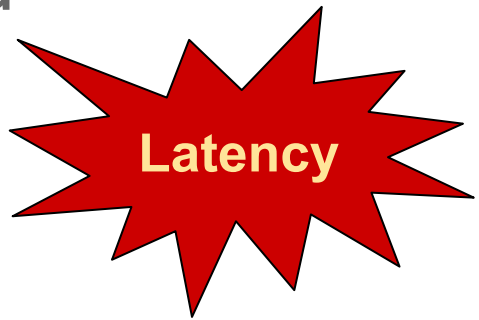
- Metrics we care about
- How does garbage collection work in V8?
- The three deadly sins of garbage collection
- Idle time garbage collection

V8 Garbage Collection
Ulan Degenbaev
Jochen Eisinger
Michael Lippautz
and me.

Blink Scheduler
Ross McIlroy
Sami Kyostila
& others...

Compositor
Manfred Ernst

Metrics we care about



60 frames per second

=

16.66ms per frame

⇒

ideally max GC pause < 6ms

while keeping [redacted] ootprint small

V8 Garbage Collection

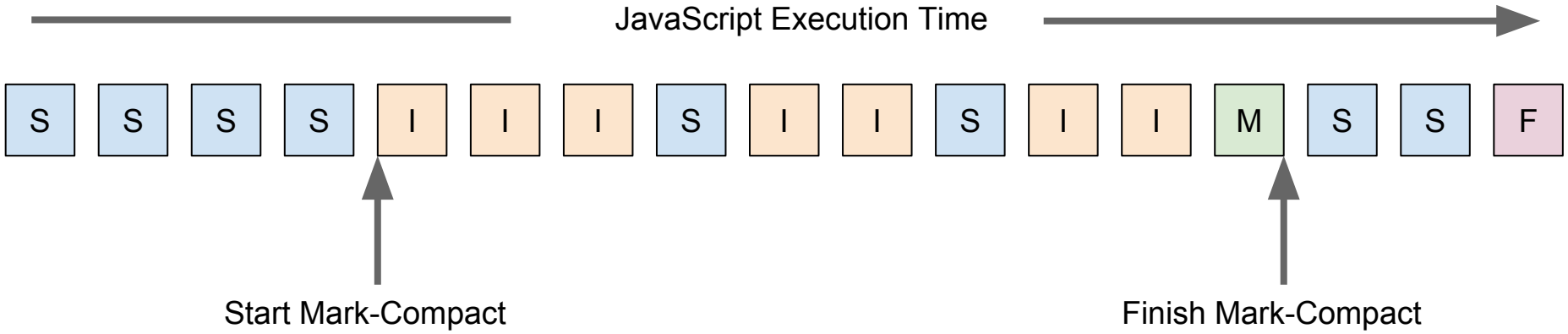
- V8’s garbage collector interrupts JavaScript execution regularly to collect unused memory
- Generational Garbage Collector
  - Generational hypothesis: most objects die shortly after their allocation
  - Small young generation (up to 16M)
    Semi-space Scavenger: touches live objects only
  - Large old generation (up to 1.4G)
    Mark-compact collector with incremental marking, concurrent sweeping, and compaction: touches the whole committed memory

V8 Garbage Collection

- Generational Garbage Collector
  - Young Generation: Semi-space Scavenger
  - Old Generation: Mark&Sweep

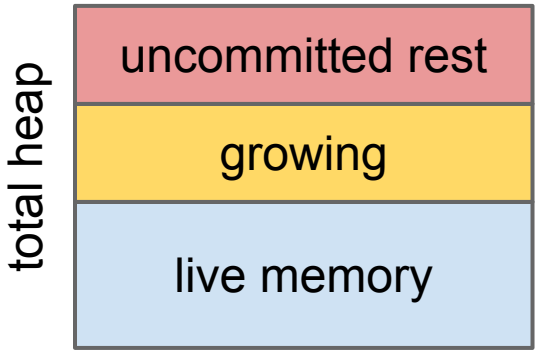
V8 Garbage Collection

S Scavenger (~0-10 ms)
I Incremental Marking (~0.01-CONFIGURABLE ms)
M Final Mark-Compact Collection (~4-40 ms)
F Full Mark-Compact Collection (>40ms)



V8 Garbage Collection

- Young generation garbage collection trigger
  - when semi-space is full
- Old generation garbage collection trigger
  - based on previous live memory size (a growing factor of 1.1 - 4)
  - start incremental marking close to full state



The three deadly sins of garbage collection

… or frequent
feature requests...

Sin 1: Should I use object pooling to avoid garbage collection?

Sin 1: Should I use object pooling to avoid garbage collection?

- Why do users want that?
  - “Because it is a known fact that garbage collection is slow, so I try to avoid allocations.”
- Why is it a bad idea?
  - It defeats compiler optimizations (escape analysis, allocation folding, write barrier elimination, generational hypothesis) => it may make your code slower
  - Increases application code complexity (malloc/free style)

Sin 2: Can I turn off the garbage collector?

Sin 2: Can I turn off the garbage collector?

- Why do users want that?
  - “Because now there is a time critical phase and garbage collection would result in jank”
- Why is it a bad idea?
  - Difficult to maintain in application code base
  - Complicates the garbage collector (always allocate mode)
  - Out of memory bugs when you forget to turn on or turn on too late

Sin 3: Can I explicitly invoke the garbage collector?

Sin 3: Can I explicitly invoke the garbage collector?

- Why do users want that?
  - “Because I know that now would be a good time and later it is super critical that there is no garbage collection going on to avoid jank.”
  - “Because I would like to reduce memory consumption.”
- Why is it a bad idea?
  - Garbage collection heuristics get confused
  - An explicit invocation may take really long and may result in jank

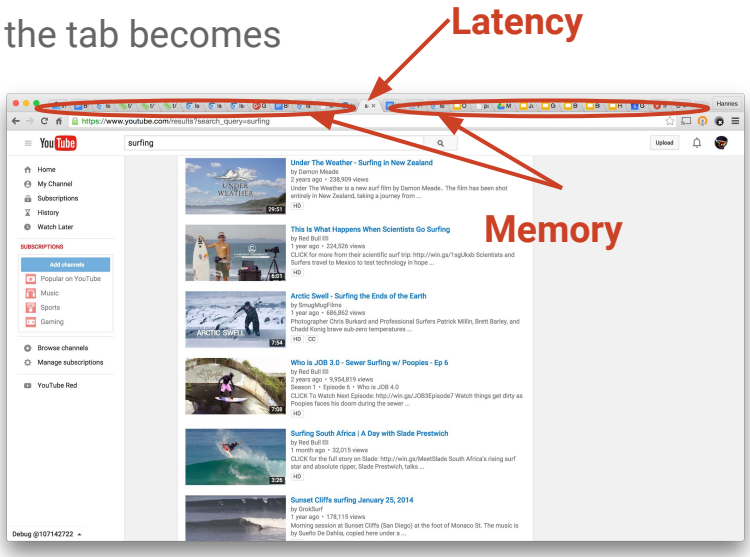
You are just saying

NO NO NO

but how can you help us???

Chrome is funky place for garbage collection...

- Foreground tab
  - Latency matters
  - New frames are drawn every 16.66 ms when animation or scrolling happens
  - Reducing memory becomes important as soon as the tab becomes inactive
- Background tab
  - Latency is secondary
  - But we can reduce memory consumption



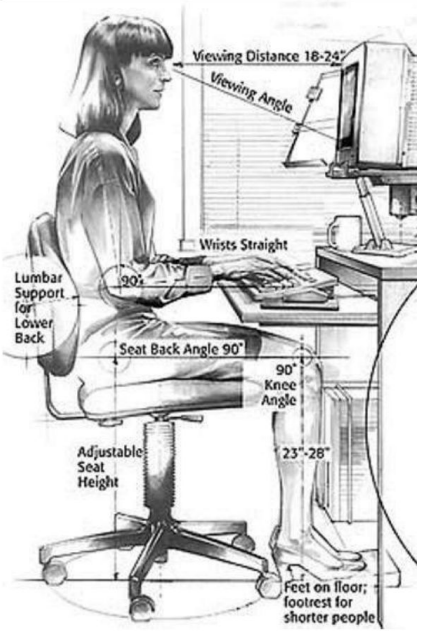
Idea: Make garbage collection invisible

Google

When is the best time to do a GC?

When nobody is looking.

Using camera to track eye movement
When subject looks away do a GC.

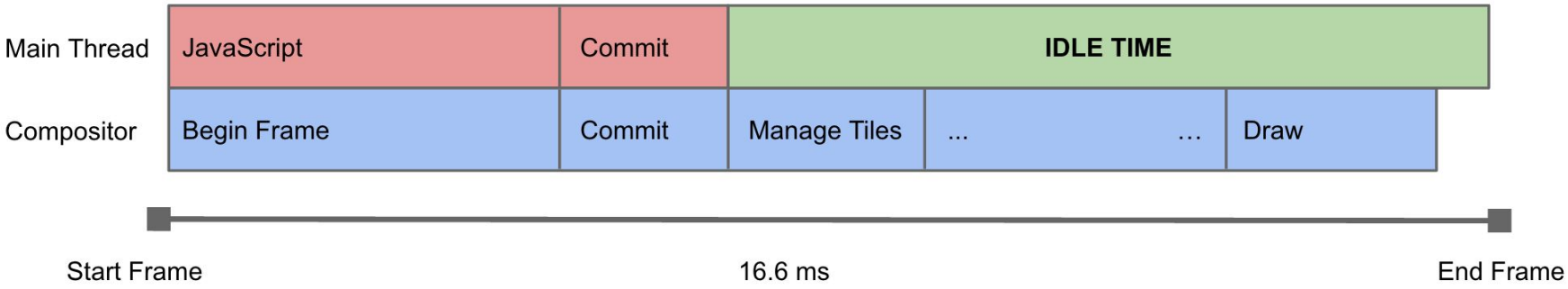


https://upload.wikimedia.org/wikipedia/commons/3/35/Computer_Workstation_Variables.jpg

Idle Time Garbage Collection

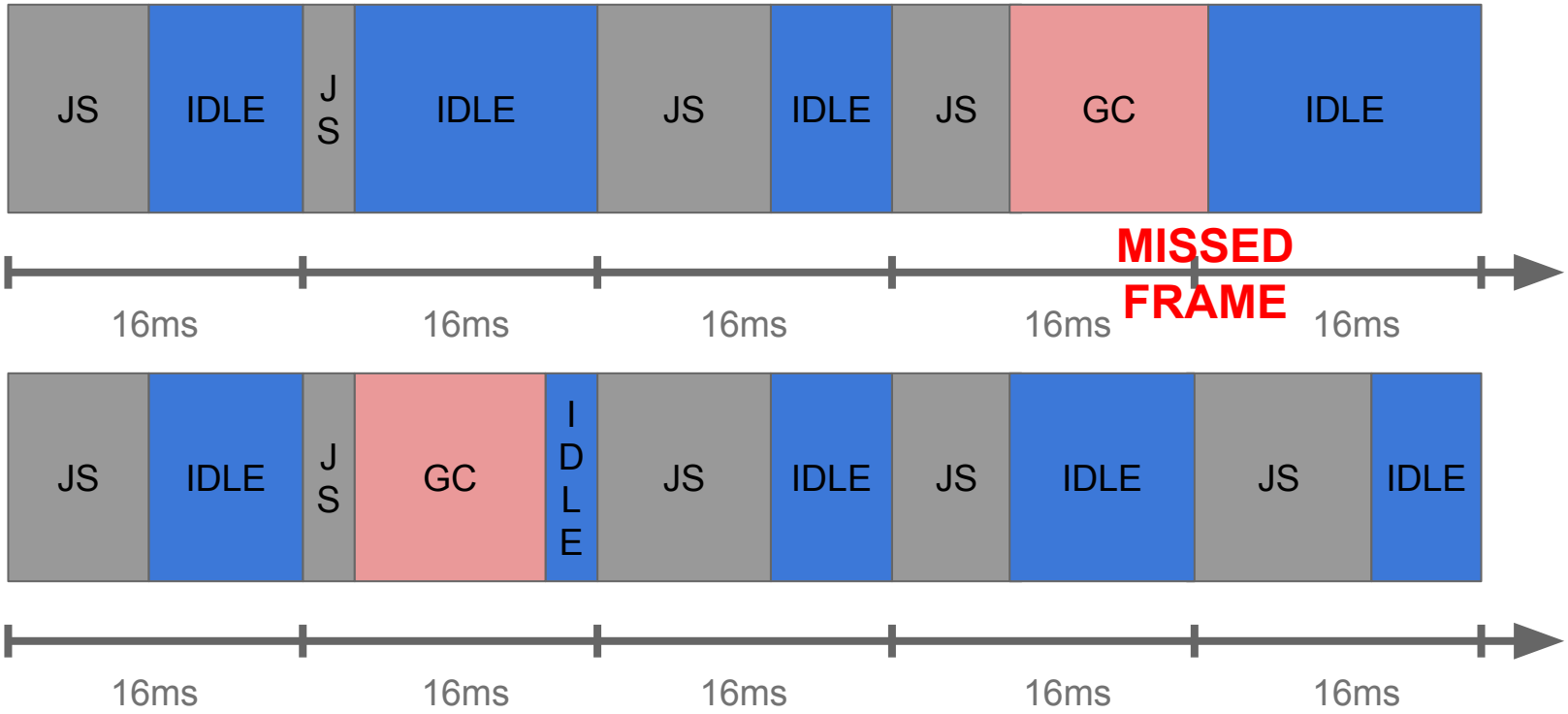
Latency

Life of a Frame



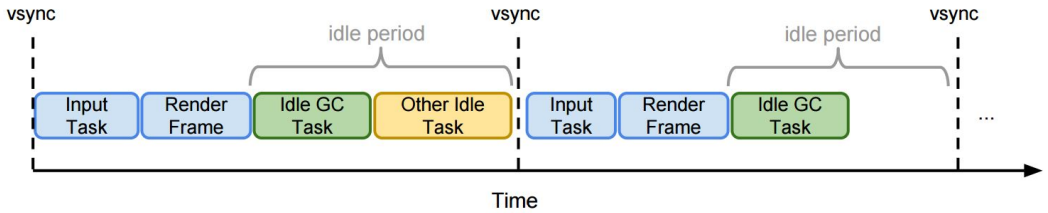
Life of a Frame

Life of a frame



Latency-driven Idle Time GC Scheduling

- We have a bunch of heuristics and monitoring code that will try to estimate:
  - average young generation collection speed/MB
  - average incremental marking speed/MB
  - average finalization of mark-compact speed/MB
- We register an idle garbage collection task in the Blink scheduler when a given garbage collection operation should happen soon.
- The task scheduler will schedule us as soon as there is idle time (and will give us up to 50ms to perform garbage collection)



History of Idle Garbage Collection Systems

- M38
  - V8 was directly called from the Compositor
    - No global notion of idleness
    - Maximum idle time 16.66ms
    - Constant rendering overhead after each rendered frame
- M41
  - Integrated into Blink task scheduler
    - Whenever Blink task scheduler detected idleness a V8 idle tasks was scheduled -> many unnecessary V8 invocations, because most of the time there was no garbage collection work pending
- M45-M46
  - V8 registers idle tasks in the Blink task scheduler when there is work pending

oortonline.gl

M41 vs M46

M46 --disable-v8-idle-tasks

Scavenge 289.3 (388.6) -> 283.3 (388.6) MB, 18.8 / 0 ms [allocation failure].
Scavenge 294.2 (388.6) -> 288.1 (388.6) MB, 20.7 / 0 ms [allocation failure].
Scavenge 299.0 (388.6) -> 291.6 (388.6) MB, 15.4 / 0 ms [allocation failure].
Scavenge 303.9 (388.6) -> 295.8 (388.6) MB, 10.2 / 0 ms [allocation failure].
Scavenge 307.4 (388.6) -> 299.6 (388.6) MB, 9.8 / 0 ms [allocation failure].
Scavenge 311.5 (388.6) -> 304.6 (388.6) MB, 16.5 / 0 ms [allocation failure].
Scavenge 315.4 (388.6) -> 309.9 (388.6) MB, 19.3 / 0 ms [allocation failure].
…

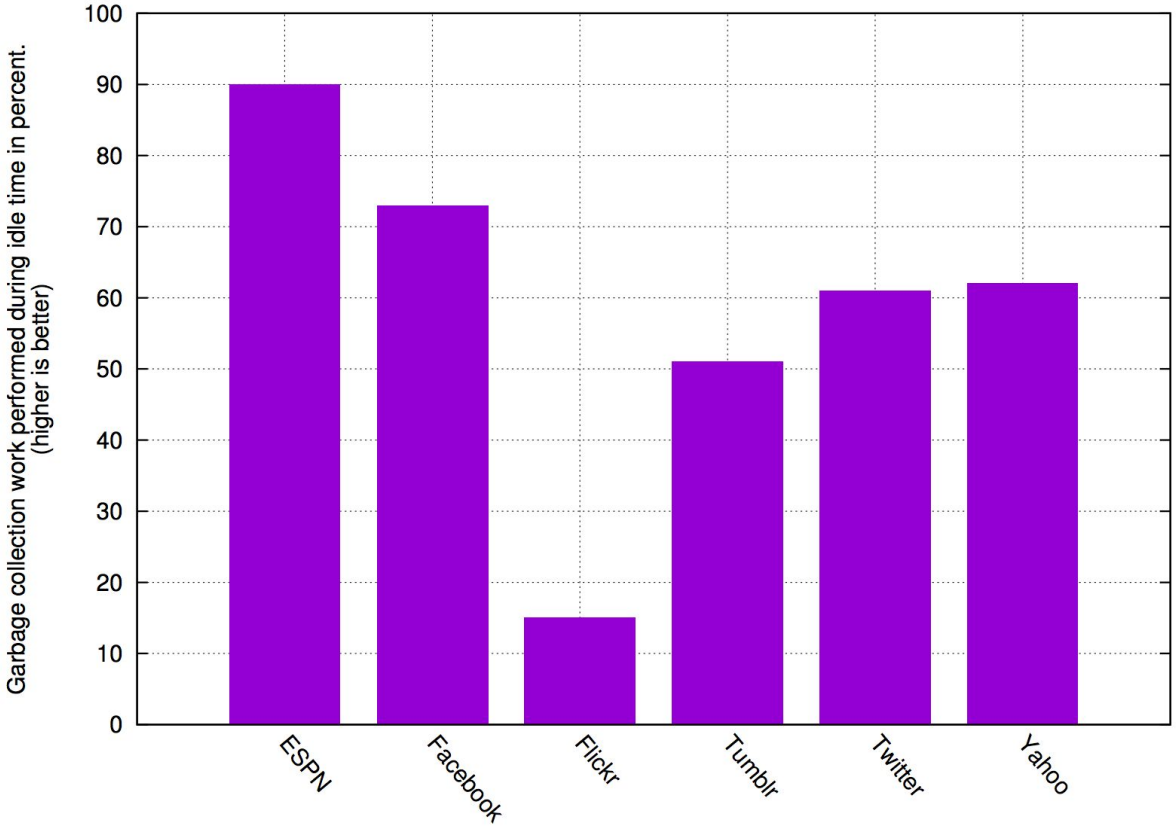
M46

Scavenge 363.1 (401.9) -> 358.6 (402.9) MB, 4.5 / 0 ms [idle task: scavenge].
Scavenge 362.1 (402.9) -> 358.6 (402.9) MB, 3.3 / 0 ms [idle task: scavenge].
Scavenge 364.8 (402.9) -> 363.3 (402.9) MB, 5.7 / 0 ms [idle task: scavenge].
Scavenge 363.5 (402.9) -> 363.3 (406.9) MB, 10.7 / 0 ms [idle task: scavenge].
Scavenge 366.9 (406.9) -> 363.4 (406.9) MB, 5.3 / 0 ms [idle task: scavenge].
Scavenge 367.0 (406.9) -> 363.5 (406.9) MB, 5.3 / 0 ms [idle task: scavenge].
Scavenge 370.4 (406.9) -> 368.2 (406.9) MB, 6.4 / 0 ms [idle task: scavenge].
…

Improvements M46 over --disable-v8-idle-tasks

Frames per second:	17.73ms (2.70% better)
Frame time discrepancy:	138.65ms (44.13% better)

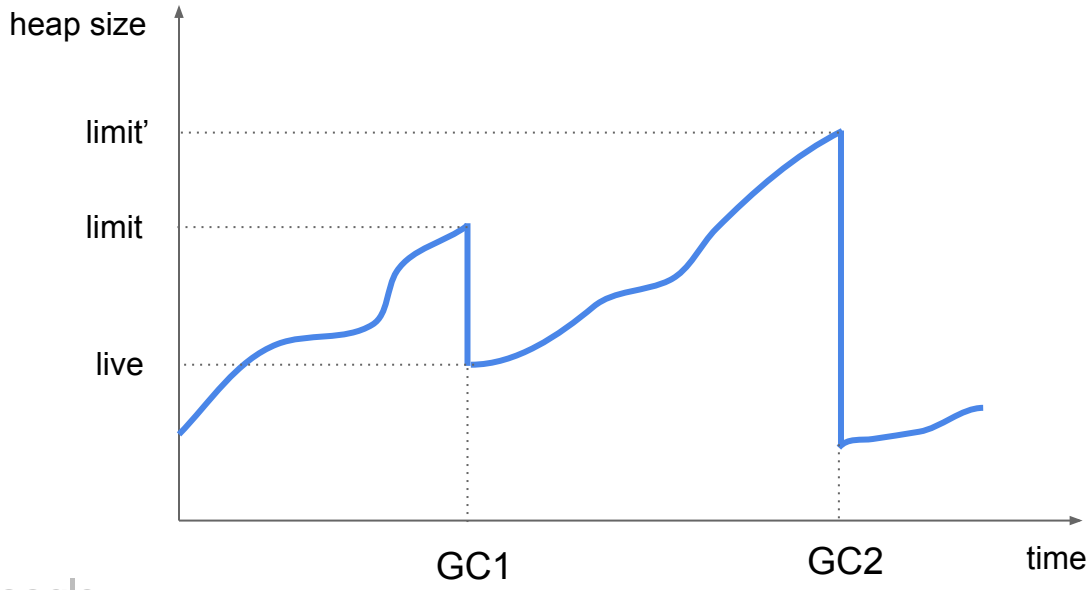
Telemetry Infinite Scrolling Benchmarks



Memory

Heap Growing Strategy

A new garbage collection is triggered (incremental marking is started) as soon as we reach a certain heap size (limit).

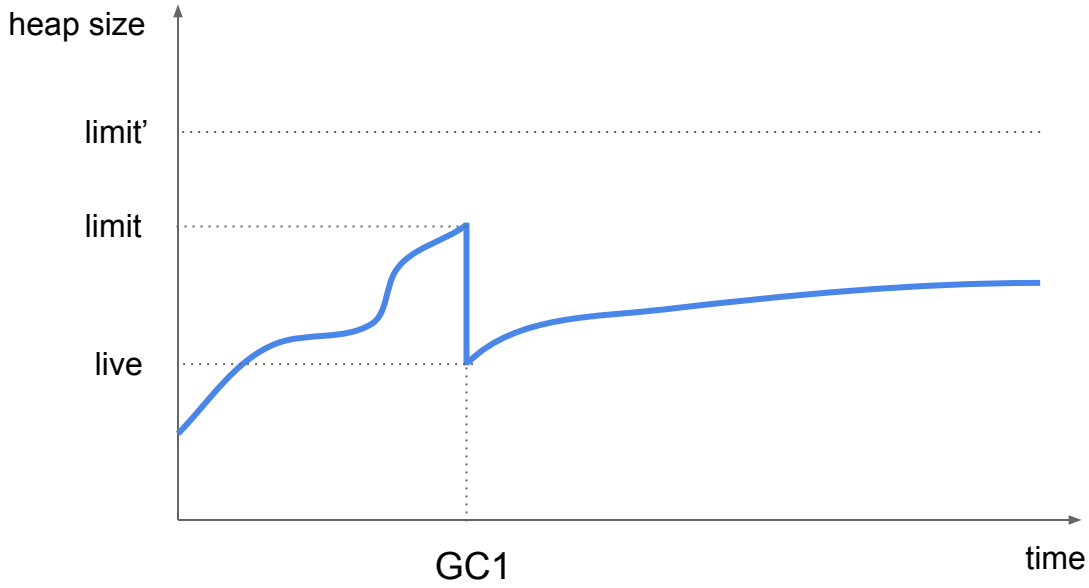


limit' = live memory x [1.1, 4]

Assumption: The allocation rate is constant. Hence limit' is reached soon which triggers garbage collection.

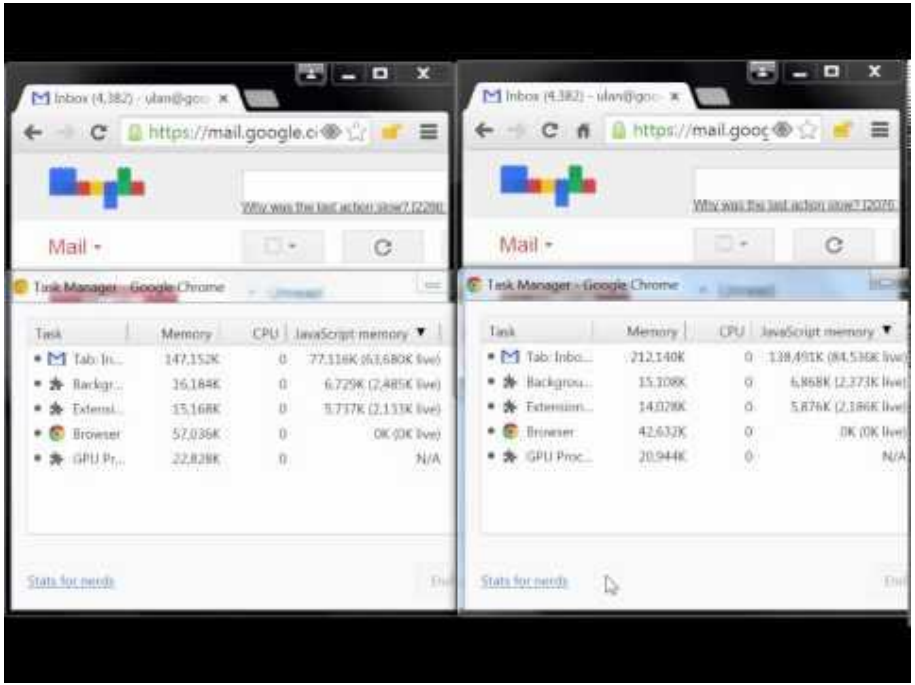
But: What if we had a high allocation rate, set a high limit', and suddenly the website became idle, i.e. allocation rate became low (or zero)?

Heap Growing Strategy



We will never reach limit' and may unnecessarily hold on to a lot of memory.

Memory-reducing Idle GC Scheduling



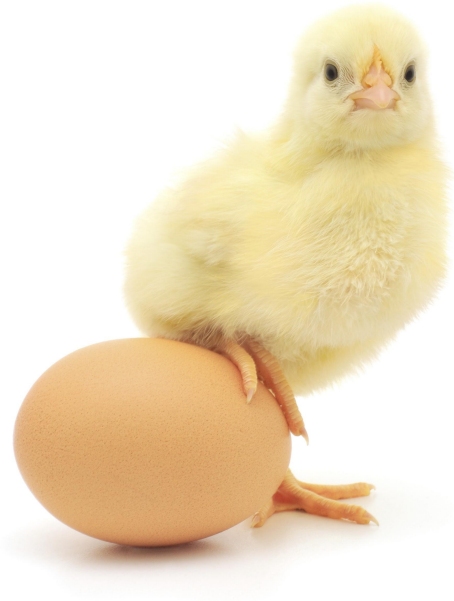
Gmail M45 vs M43

- **Memory Reducer**
  Track recent allocation rate and check back periodically if a tab became idle
- If idle:
  - start incremental marking,
  - schedule incremental marking steps,
  - perform more aggressive compaction,
  - shrink and uncommit young generation
  - set a small heap growing limit.
- Don’t drain the battery!
- ~40% less memory on average

Challenge: Benchmarks

We like long-running (>30 seconds) benchmarks that perform *interesting* actions on real websites.

How do we get benchmark that represent the web of tomorrow?



Thank You! Questions?

+Hannes Payer
hpayer@google.com
http://research.google.com/pubs/HannesPayer.html