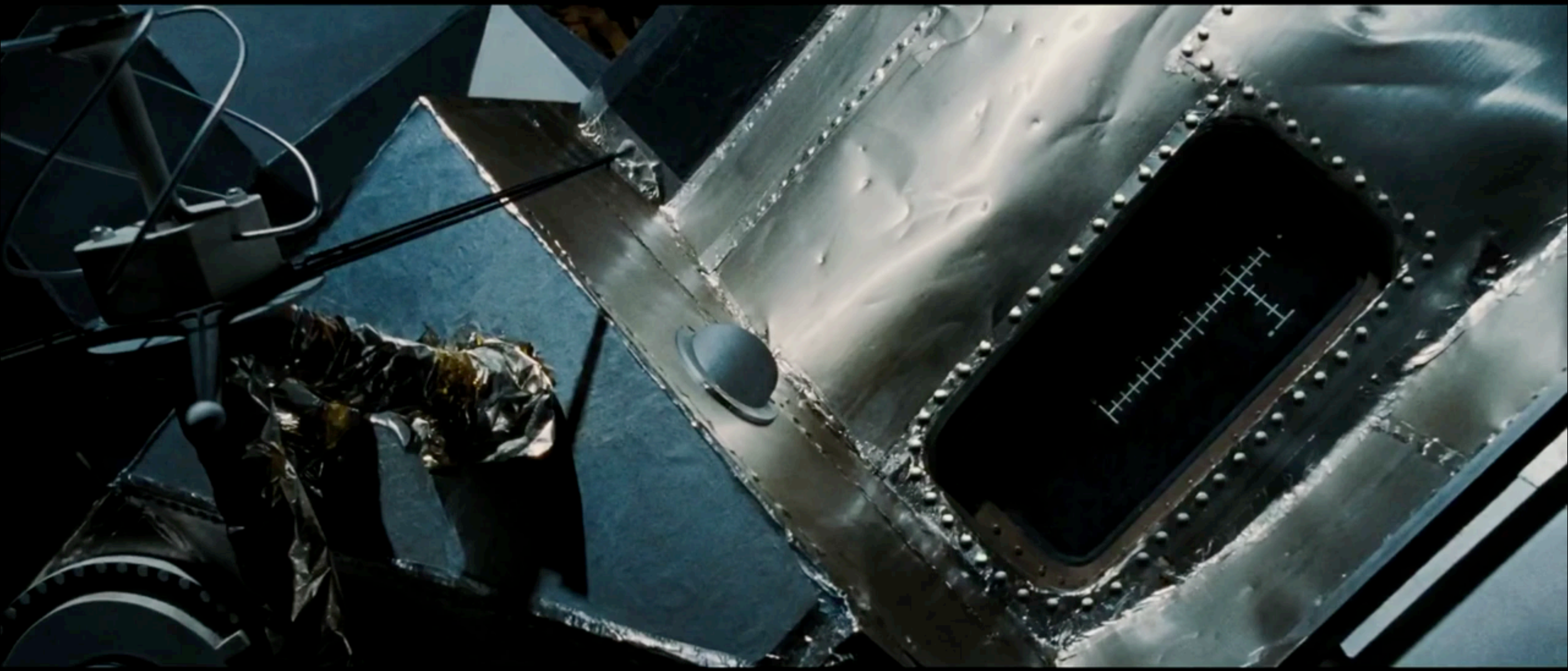
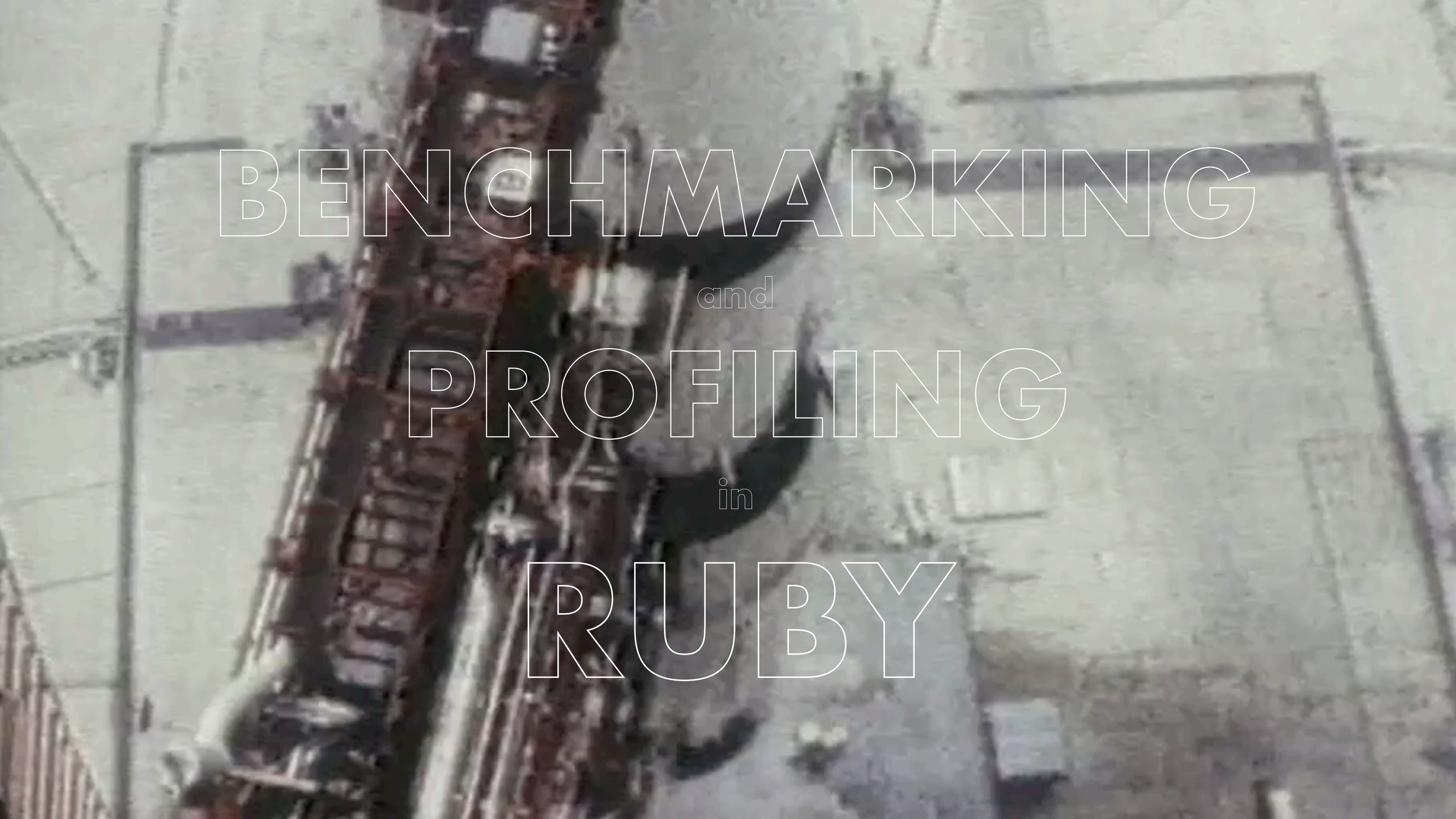



BENCHMARKING
and
PROFILING
in
RUBY


BENCHMARKING and
PROFILING in
RUBY

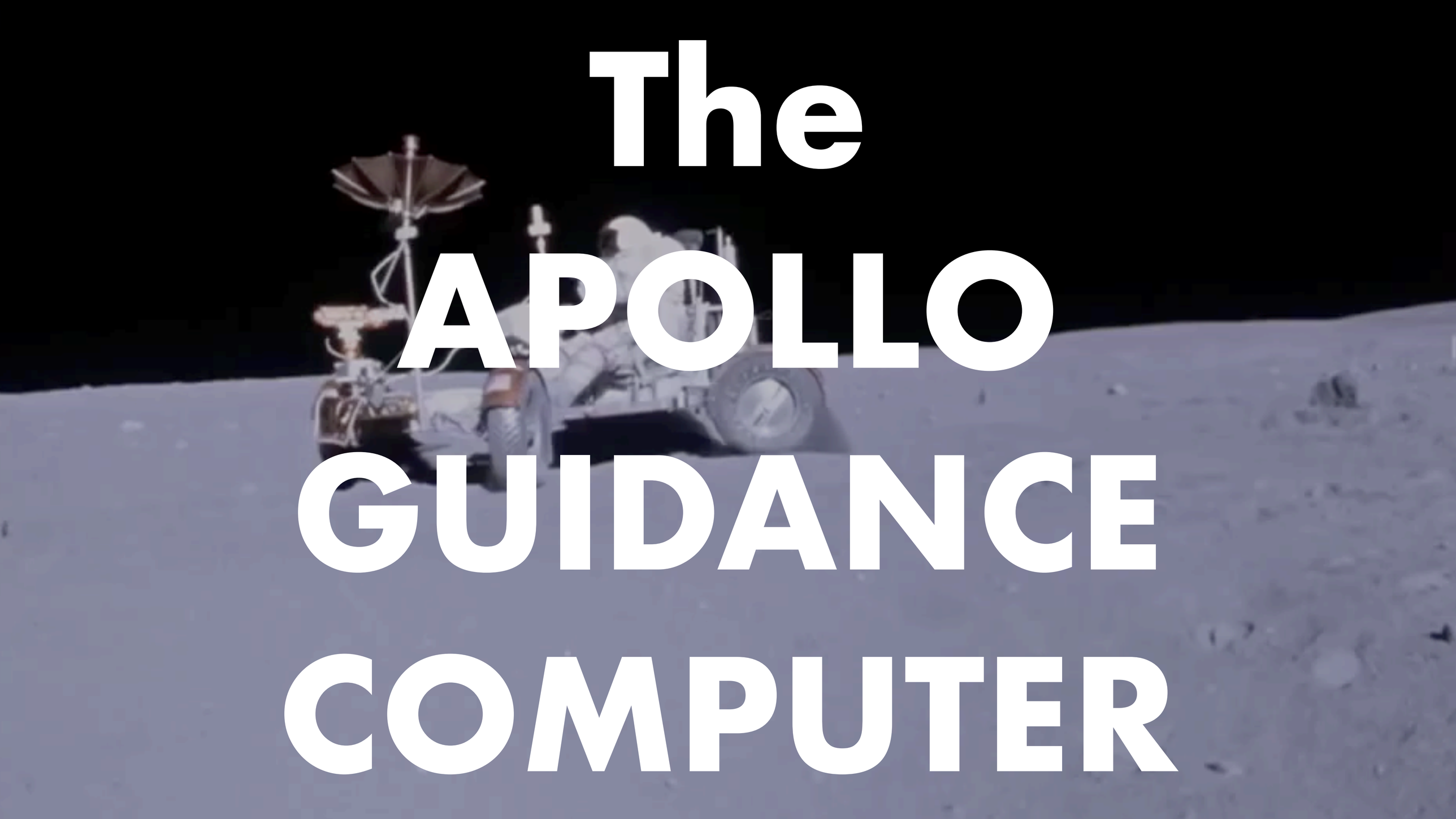
SUMMER WORKSHOP TOUR

buy @ speedshop.co

- **Minneapolis/RailsConf** Friday May 3rd. Hyatt Regency (same as the conference hotel).
- **New York City** Monday June 3rd. Chelsea.
- **New York City** Wednesday June 5th. Chelsea.
- **New York City** Friday June 7th. **Special Topic: ActiveRecord Deep Dive**.
- **Atlanta, GA** Monday June 17th.
- **Austin, TX** Wednesday June 19th.
- **Dallas, TX** Friday June 21st.
- **Denver, CO** Monday June 24th.
- **Chicago, IL** Wednesday June 26th.
- **Boston, MA** Monday July 15th.
- **Washington DC** Wednesday July 17th.
- **Philadelphia, PA** Friday July 19th.
- **Portland, OR** Monday July 29th.
- **Seattle, WA** Wednesday July 31st.
- **San Francisco, CA** Friday August 2nd.
- **San Francisco, CA** Monday August 5th.
- **San Francisco, CA** Wednesday August 7th. **Special Topic: ActiveRecord**

FLIGHT PLAN

Why Measure?
Benchmarking vs. Profiling
Profiling Jumpstart
Benchmarking Jumpstart



The APOLLO GUIDANCE COMPUTER

FIRST MAN (2018)

04 06 38 30 CDR (EAGLE)
It's a 1202.

04 06 38 32 LMP (EAGLE)
1202.

04 06 38 48 CDR (EAGLE)
Give us a reading on the 1202 PROGRAM ALARM.

04 06 38 53 CC
Roger. We got - We're GO on that alarm

APPLICABLE TO: IN DESCENT, AVERAGE-G ON

ALARM CODE		TYPE	PRE-MANUAL CAPABILITY	MANUAL CAPABILITY
00105	MK ROUT. BUSY	POODOO	PGNCS GUID. LOST,	PGNCS GUIDANCE NO/GO
00430	CAN'T INTG. S.V.	"		
01103	CCSHOLE-PROG. BUG	"	PGNCS/AGS ABRT/ABRT STG	(PGNCS GO for TAPE METERS, CROSS-POINTERS, CONTROL, ABORTING)
01204	NEG. WAITLIST	"	(decision how on current rules)	
01206	DSKY, TWO USERS	"	(NO LR DATA)	(NO LR DATA)
01302	NEG. SQ. ROOT	"		
01501	DSKY, PROG. BAD	"		
01502	DSKY, PROG. BUG	"		
00607	LAMB. NO SOLN	"		
"O.F." = Overflow, to many. CONTINUING ⟸ OCCURRENCE OF:			DUTY CYCLE MAY DEGRADE PGNCS (AGS CONTROL MAY HELP - SEE BELOW) (WATCH FOR OTHER CUES)	SAME AS LEFT
01104	DELAY ROUT. O.V.	BAILOUT	PGNCS CONDITION UNKNOWN,	(except "other cues" which would otherwise be cause for ABORT PROBABLY AREN'T; INSTEAD IT WOULD BE PGNCS GUIDANCE NO/GO — COMPLETE MANUAL LANDING IN AGS.)
01201	EXECT. O.F (VAC)	"	DSKY MAY BE LOCKED UP,	
01202	EXECT. O.F. (JOBS)	"	DUTY CYCLE MAY BE UP TO POINT OF MISSING SOME FUNCTIONS (NAV, LAST TO DIE), SWITCH TO AGS (FOLLOW ERR NEEDLES) MAY HELP (REDUCES PGNCS DUTY CYCLE SIGNIF.)	
01203	EXECT. O.F. (TASKS)	"		
01207	EXECT. O.F. (MARKS)	"		
01210	TWO USERS	"		
01211	MRK ROUT. INTRPT	"		
02000	DAP O.F.	"		
ISS WARNINGS WITH:				
00777	PIPA FAIL	LIGHT ONLY	PIPA/CDU/IMU FAIL DISCRETES PRESENT	same as left
03777	CDU FAIL	"		
04777	PIPA, CDU FAIL	"	(Other mission rules suffice; alarm may help point to what rule will be broken)	
07777	IMU FAIL	"		
10777	PIPA, IMU FAIL	"		
13777	CDU, IMU FAIL	"		
14777	PIPA, CDU, IMU FL	"		
00214	IMU TURNED OFF	LIGHT ONLY	AGS ABRT/ABRT STAGE	SWITCH TO AGS PGNS NO/GO on G and C (poss. NO/GO on NAV.)
01107	E-Mem. Destroyed	FRESH STRT	AGS ABRT/ABRT STAGE	SWITCH TO AGS, PGNCS NO/GO! (IMU as ref. okay)
CONTINUING ⟸ 00402	BAD GUID. CMDS	LIGHT ONLY	IF ALARM DOESN'T STOP: same as "POODO's" (ABRT.!)	IF ALARM DOESN'T STOP: Same as "POODO's
CONTINUING ⟸ 01406	GUID. NO SOLN	LIGHT ONLY	PGNCS GUID. NO/GO AS LONG AS ALARM OCCURRING (ATT. HOLD, CONST. GTC, CONT. OK) (ABRT WILL PROB. COME FROM CURRENT RULES e.g. GTC vs. V) WATCH GTC ⟸	same as left (except prob. no abort.)
01410	GUID O.V.			

"O.F." = Overflow, to many.
CONTINUING ⟸
OCCURRENCE OF:

01104	DELAY ROUT. O.V.	BAILOUT
01201	EXECT. O.F (VAC)	"
01202	EXECT. O.F. (JOBS)	"

DUTY CYCLE MAY DEGRADE PGNCS (AGS CONTROL MAY HELP - SEE BELOW)
(WATCH FOR OTHER CUES)
PGNCS CONDITION UNKNOWN,
DSKY MAY BE LOCKED UP,

BAILOUT
No available core sets

```
ALARM_AND_ABORT.AGC

BAILOUT  INHINT

    CA Q

    TS ALMCADR

    INDEX  Q

    CAF 0

    TC BORTENT
```

PERFORMANCE EVALUATION OF A MULTIPROCESSOR IN A REAL TIME ENVIRONMENT

by

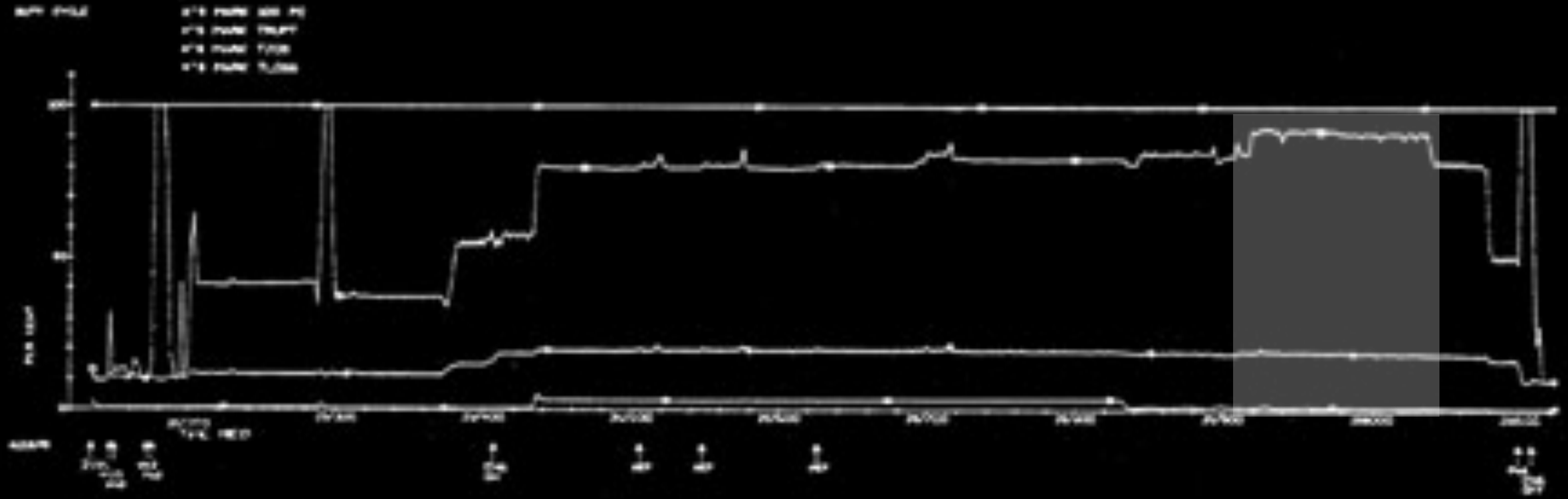
Jaynarayan H. Lala

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 1976

The largest computational loads maintained on the Apollo Guidance Computer during the mission were imposed during the descent phase of the lunar landing. The programs were analyzed to determine their execution times in terms of the number of instructions executed, data requirements, and cylic execution rates for periodic jobs.



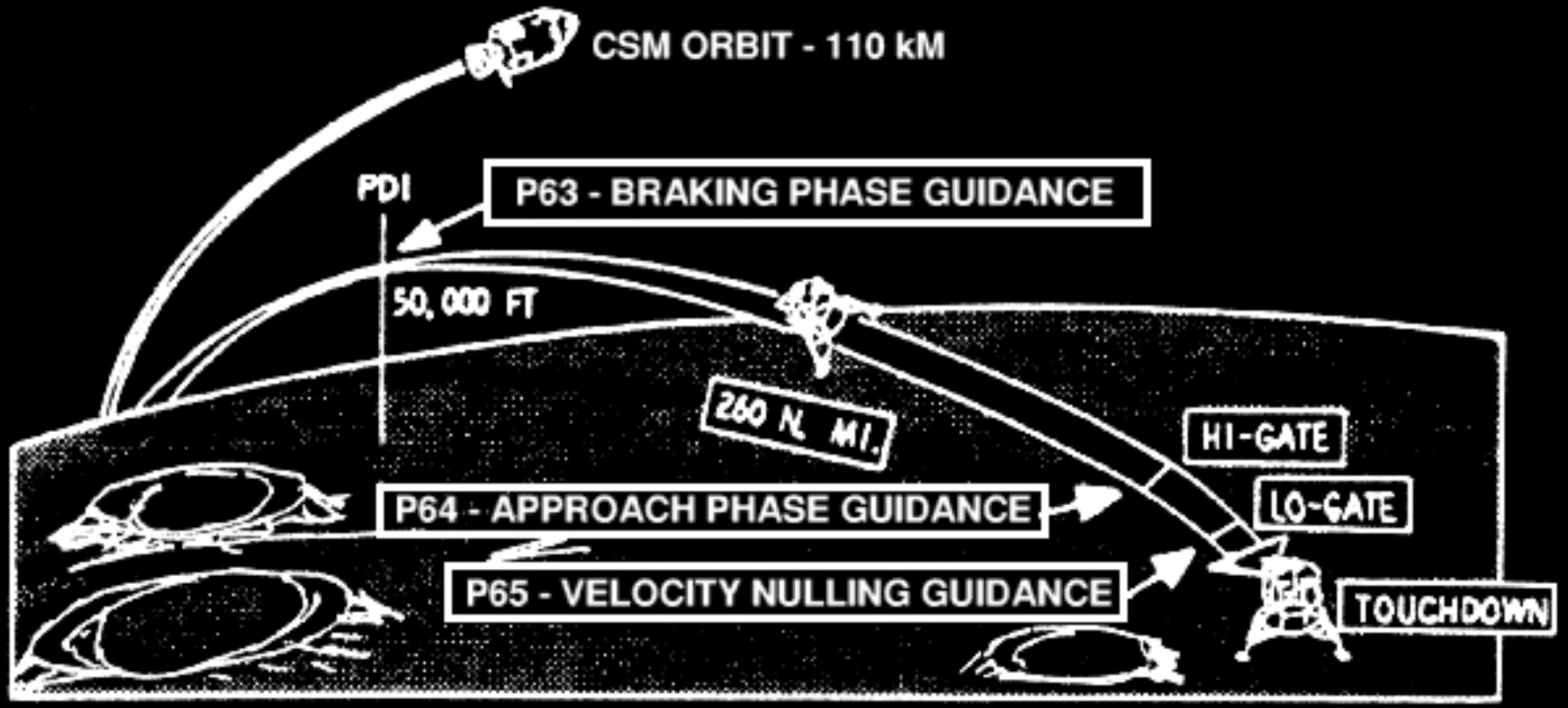


Fig. 3. Operational phases of powered descent. Design criteria: P 63 - Braking phase (PD1 to Hi-Gate), efficient reduction of orbital velocity. P 64 - Final approach phase (Hi-Gate to Lo-Gate), crew visibility (safety and site assessment). P 65 - Landing phase (Lo-Gate to touch-down), manual control takeover.

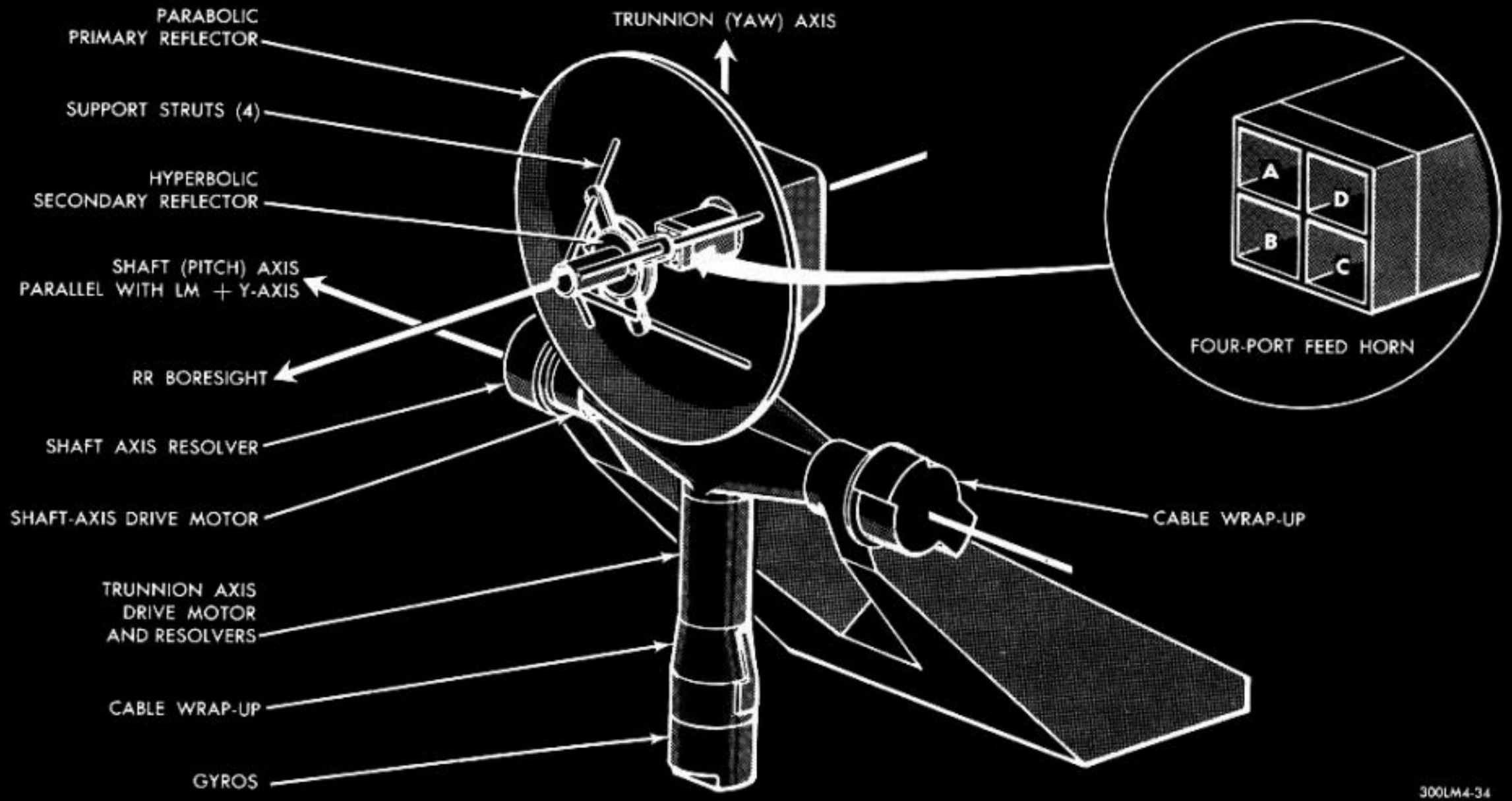


Figure 2.2-24. Rendezvous Radar Antenna Assembly

APOLLO:
the ultimate
"TESTING IN PROD"

321 (239 unique) GCC-8 backend passes in execution order

all_lowering_passes
1. warn_unused_result
2. diagnose_omp_blocks
3. diagnose_tm_blocks
4. lower_omp
5. lower_cf
6. lower_tm
7. refactor_eh
8. lower_eh
9. build_cfg
10. warn_function_return
11. expand_omp
12. sprintf_length[1]
13. walloca[1]
14. build_cgraph_edges

all_small_ipa_passes
15. ipa_free_lang_data
16. ipa_fun_and_var_visibility
17. ipa_chkp_versioning
18. ipa_chkp_early_produce_thunks
build_ssa_passes
19. fixup_cfg[1]
20. build_ssa
21. warn_nonnull_compare
22. ubsan
23. early_warn_uninitialized
24. nothrow
25. rebuild_cgraph_edges[1]

chkp_instrumentation_passes
26. fixup_cfg[2]
27. chkp
28. rebuild_cgraph_edges[2]

local_optimization_passes
29. fixup_cfg[3]
30. rebuild_cgraph_edges[3]
31. local_fn_summary[1]
32. early_inline
all_early_optimizations
33. remove_cgraph_callee_edges[1]
34. object_sizes[1]
35. ccp[1]
36. forwprop[1]
37. early_thread_jumps
38. sra_early
39. build_ealias
40. fre[1]
41. early_vrp
42. merge_phi[1]
43. dse[1]
44. cd_dce[1]
45. early_ipa_sra
46. tail_recursion[1]
47. convert_switch
48. cleanup_eh[1]
49. profile
50. local_pure_const[1]
51. split_functions
52. strip_predict_hints[1]
53. release_ssa_names
54. rebuild_cgraph_edges[4]
55. local_fn_summary[2]

ipa_oacc
56. ipa_pta[1]
ipa_oacc_kernels
oacc_kernels
57. ch[1]
58. fre[2]
59. lim[1]
60. dominator[1]
61. dce[1]
62. parallelize_loops[1]
63. expand_omp_ssa[1]
64. rebuild_cgraph_edges[5]

65. target_clone
66. ipa_chkp_produce_thunks
67. ipa_auto_profile
68. ipa_tree_profile
69. feedback_split_functions

70. ipa_free_fn_summary[1]
71. ipa_increase_alignment
72. ipa_tm
73. ipa_lower_emutls

all_regular_ipa_passes
74. ipa_whole_program_visibility
75. ipa_profile
76. ipa_icf
77. ipa_devirt
78. ipa_cp
79. ipa_ditor_merge
80. ipa_hsa
81. ipa_fn_summary
82. ipa_inline
83. ipa_pure_const
84. ipa_free_fn_summary[2]
85. ipa_reference
86. ipa_single_use
87. pass_ipa_comdats

all_late_ipa_passes
88. materialize_all_clones
89. ipa_pta[2]
90. omp_simd_clone

91. fixup_cfg[4]
92. lower_eh_dispatch
93. oacc_device_lower
94. omp_device_lower
95. omp_target_link
all_optimizations
96. remove_cgraph_callee_edges[2]
97. strip_predict_hints[2]
98. ccp[2]
99. post_ipa_warn[1]
100. complete_unrolli
101. backprop
102. phiprop
103. forwprop[2]
104. object_sizes[2]
105. build_alias
106. return_slot
107. fre[3]
108. merge_phi[2]
109. thread_jumps[1]
110. vrp[1]
111. chkp_opt
112. dce[2]
113. stdarg
114. call_cdce
115. cselim
116. copy_prop[1]
117. tree_ifcombine
118. merge_phi[3]
119. phiopt[1]
120. tail_recursion[2]
121. ch[2]
122. lower_complex[1]
123. sra
124. thread_jumps[2]
125. dominator[2]
126. isolate_erroneous_paths
127. phi_only_cprop[1]
128. dse[2]
129. reassoc[1]
130. dce[3]
131. forwprop[3]
132. phiopt[2]
133. ccp[3]
134. cse_sincos
135. optimize_bswap
136. laddress
137. lim[2]
138. walloca[2]
139. pre
140. sink_code
141. sancov[1]
142. asan[1]
143. tsan[1]
144. dce[4]
145. fix_loops
tree_loop
146. tree_loop_init
147. tree_unswitch
148. scev_cprop
149. loop_split
150. loop_jam
151. cd_dce[2]
152. iv_canon
153. loop_distribution
154. linterchange
155. copy_prop[2]
graphite
156. graphite_transforms
157. lim[3]
158. copy_prop[3]
159. dce[5]
160. parallelize_loops[2]
161. expand_omp_ssa[2]
162. ch_vect
163. if_conversion
164. vectorize
165. dce[6]

166. predcom
167. complete_unroll
168. slp_vectorize[1]
169. loop_prefetch
170. iv_optimize
171. lim
172. tree_loop_done

tree_no_loop
173. slp_vectorize[2]

174. sinduid_cleanup[1]
175. lower_vector_ssa[1]
176. cse_reciprocals
177. sprintf_length[2]
178. reassoc[2]
179. strength_reduction
180. split_paths
181. tracer
182. thread_jumps[3]
183. dominator[3]
184. strlen
185. thread_jumps[4]
186. vrp[2]
187. warn_restrict
188. phi_only_cprop[2]
189. dse[3]
190. cd_dce[3]
191. forwprop[4]
192. phiopt[3]
193. fold_builtins[1]
194. optimize_widening_mul
195. store_merging
196. tail_calls
197. dce[7]
198. split_crit_edges[1]
199. late_warn_uninitialized[1]
200. uncprop[1]
201. local_pure_const[2]

all_optimizations_g
202. remove_cgraph_callee_edges[3]
203. strip_predict_hints[3]
204. lower_complex[2]
205. lower_vector_ssa[2]
206. ccp[4]
207. post_ipa_warn[2]
208. object_sizes[3]
209. fold_builtins[2]
210. sprintf_length[3]
211. copy_prop[4]
212. dce[8]
213. sancov[2]
214. asan[2]
215. tsan[2]
216. split_crit_edges[2]
217. late_warn_uninitialized[2]
218. uncprop[2]
219. local_pure_const[3]

tm_init
220. tm_mark
221. tm_memopt
222. tm_edges

223. sinduid_cleanup[2]
224. vtable_verify
225. lower_vaarg
226. lower_vector
227. lower_complex_00
228. sancov_00
229. lower_switch
230. asan_00
231. tsan_00
232. sanopt
233. cleanup_eh[2]
234. lower_resx
235. nrv
236. cleanup_cfg_post_optimizing
237. warn_function_noreturn
238. gen_hsail
239. expand
rest_of_compilation
240. instantiate_virtual_regs
241. into_cfg_layout_mode
242. jump[1]
243. lower_subreg[1]
244. df_initialize_opt
245. cse[1]
246. rtl_fwprop
247. rtl_cprop[1]
248. rtl_pre
249. rtl_hoist
250. rtl_cprop[2]
251. rtl_store_motion
252. cse_after_global_opts
253. rtl_ifcvt
254. reginfo_init
loop2
255. rtl_loop_init
256. rtl_move_loop_invariants
257. rtl_unroll_loops
258. rtl_doloop
259. rtl_loop_done

260. web
261. rtl_cprop[3]
262. cse[2]
263. rtl_dse[1]
264. rtl_fwprop_addr
265. inc_dec
266. initialize_regs
267. ud_rtl_dce
268. combine
269. if_after_combine
270. partition_blocks
271. outof_cfg_layout_mode
272. split_all_insns
273. lower_subreg[2]
274. df_initialize_no_opt
275. stack_ptr_mod
276. mode_switching
277. match_asm_constraints
278. sms
279. live_range_shrinkage
280. sched[1]
281. early_remat
282. ira
283. reload
postreload
284. postreload_cse
285. gcse
286. split_after_reload
287. ree
288. compare_elim_after_reload
289. branch_target_load_optimize[1]
290. thread_prologue_and_epilogue
291. rtl_dse[2]
292. stack_adjustments
293. jump[2]
294. duplicate_computed_gotos
295. sched_fusion
296. peephole
297. if_after_reload
298. regrename
299. cprop_hardreg
300. fast_rtl_dce
301. reorder_blocks
302. branch_target_load_optimize[2]
303. leaf_regs
304. split_before_sched2
305. sched[2]
stack_regs
306. split_before_regstack
307. stack_regs_run

late_compilation
308. compute_alignments
309. variable_tracking
310. free_cfg
311. machine_reorg
312. cleanup_barriers
313. delay_slots
314. split_for_shorten_branches
315. convert_to_eh_region_ranges
316. shorten_branches
317. set_nothrow_function_flags
318. dwarf2_frame
319. final

320. df_finish

321. clean_state

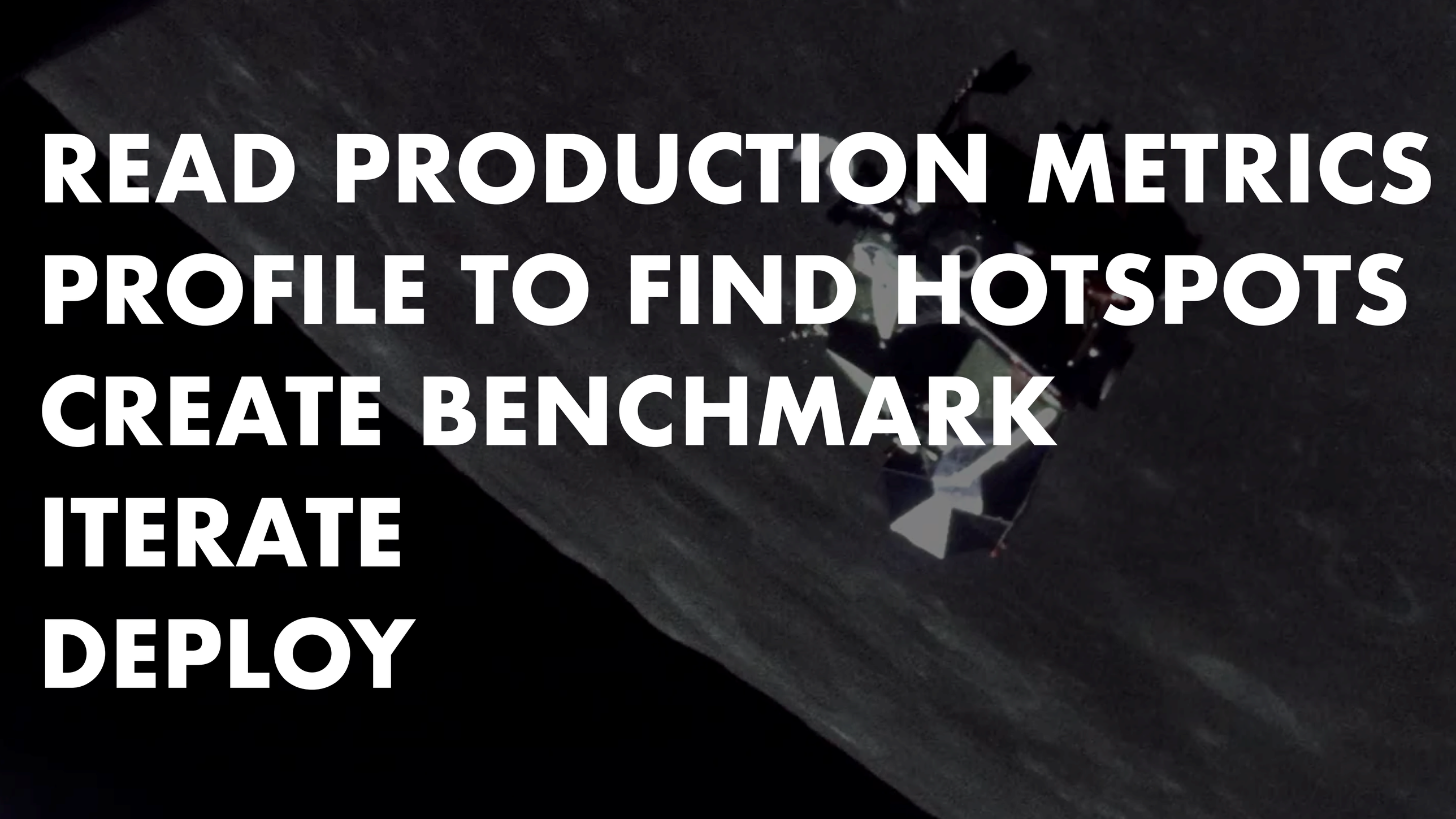
WE
are not
COMPILERS

ALL PERFORMANCE WORK WITHOUT MEASUREMENT is PREMATURE

MEASURE TWICE PR ONCE

A BENCHMARK
is a SYNTHETIC MEASUREMENT
of the RESOURCES or TIME consumed
by a defined PROCEDURE

A PROFILE
is a STEP BY STEP ACCOUNTING
of the RELATIVE CONSUMPTION
of RESOURCES by a procedure's
many SUBROUTINES


READ PRODUCTION METRICS
PROFILE TO FIND HOTSPOTS
CREATE BENCHMARK
ITERATE
DEPLOY

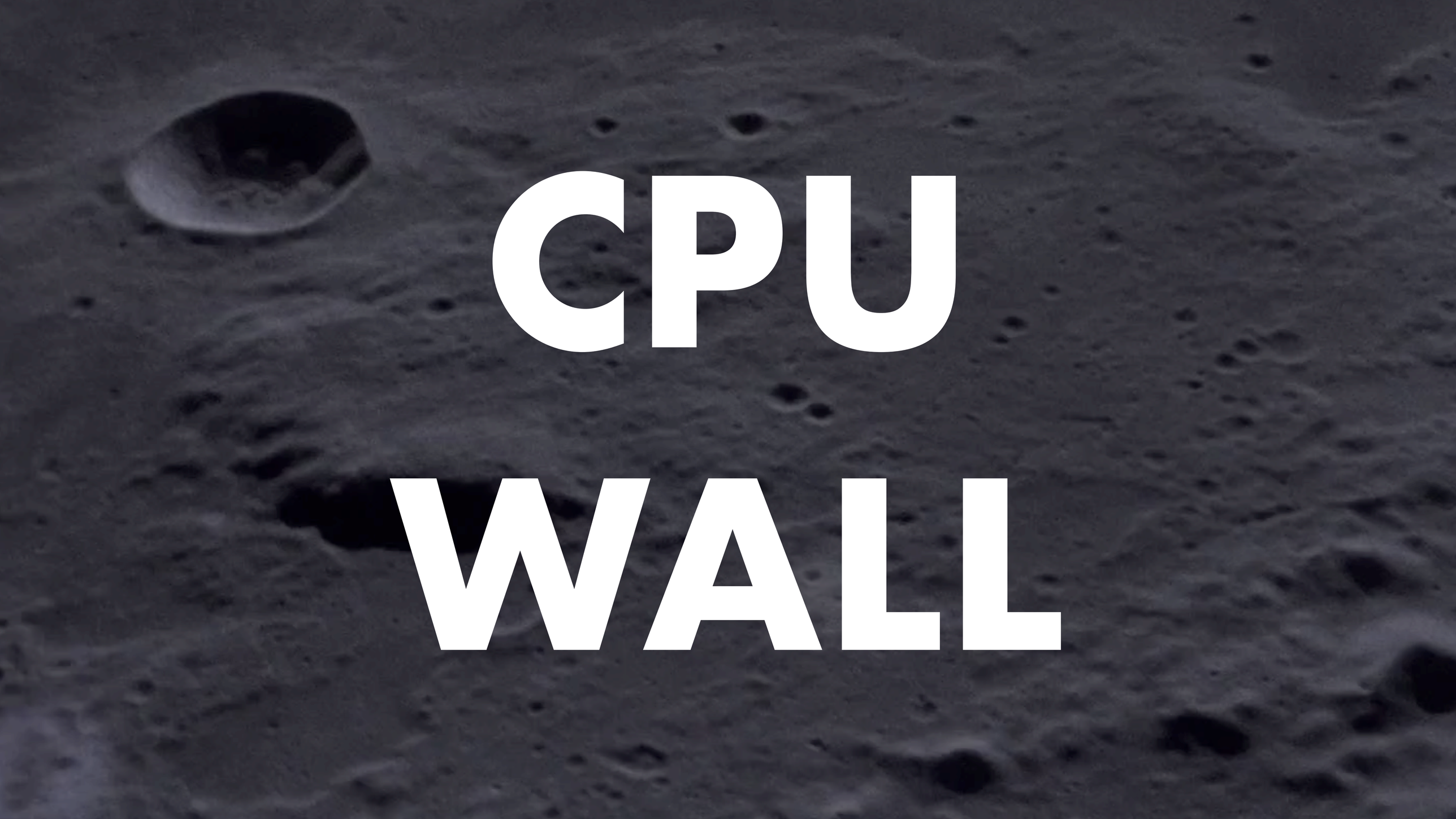
PROFILING

answers

"WHAT'S SLOW?"


PROFILING
ENVIRONMENT

```
config.cache_classes = true
config.eager_load = true
config.public_file_server.enabled = true
config.assets.compile = false
config.assets.digest = true
config.active_record.migration_error = false
```

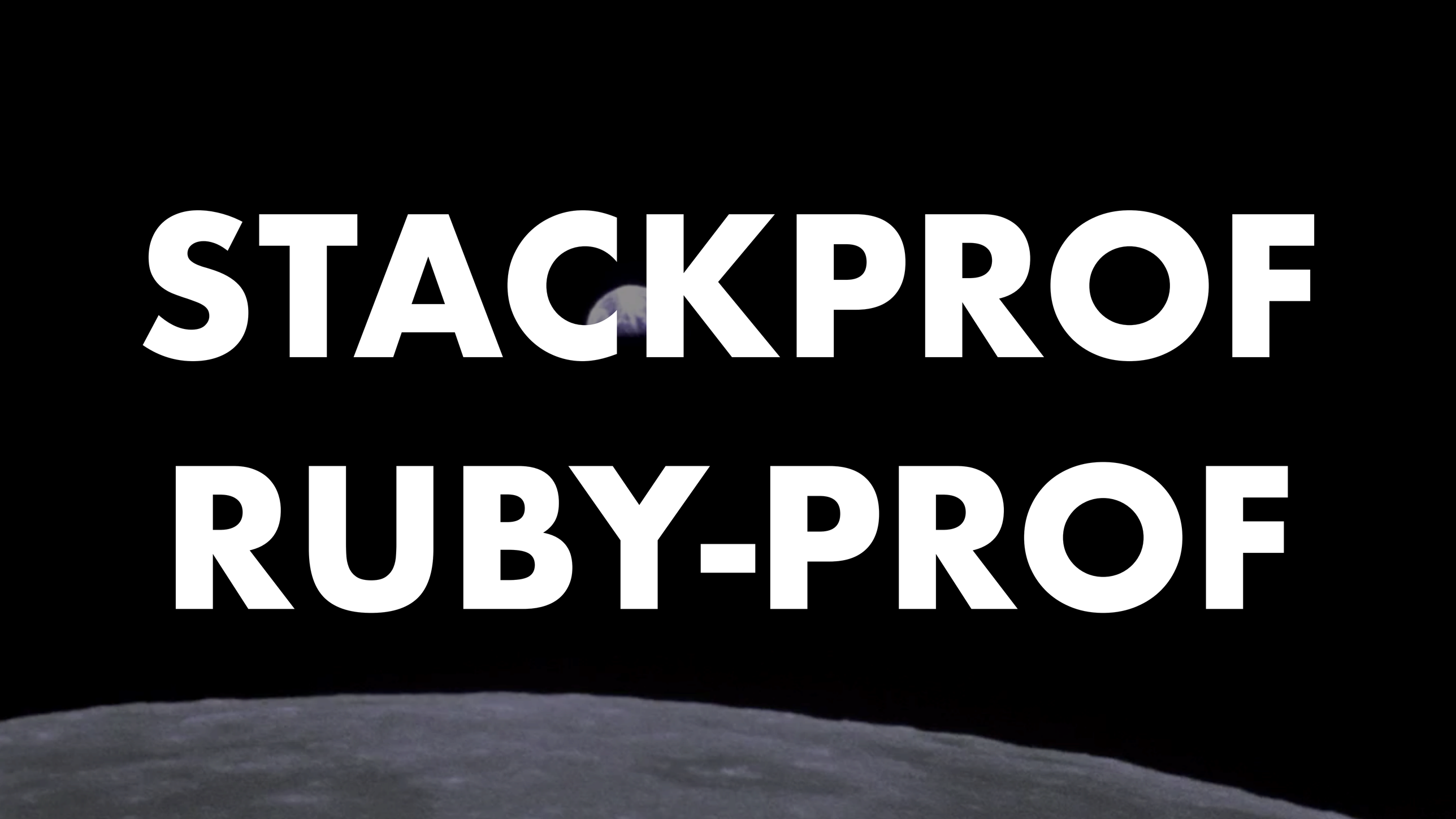

CPU
WALL

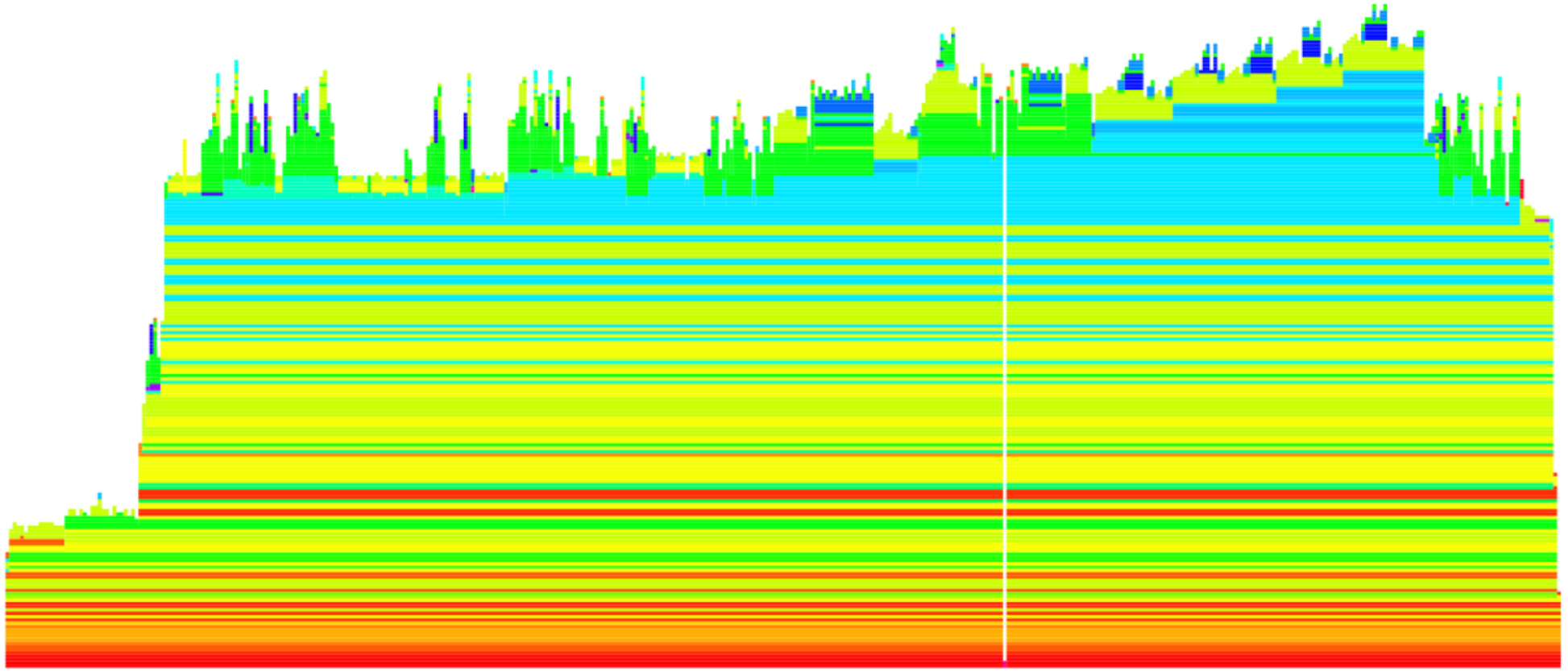
```
def sleeper
  sleep(4)
end
```

```
def sleeper
  sleep(4)
end
```

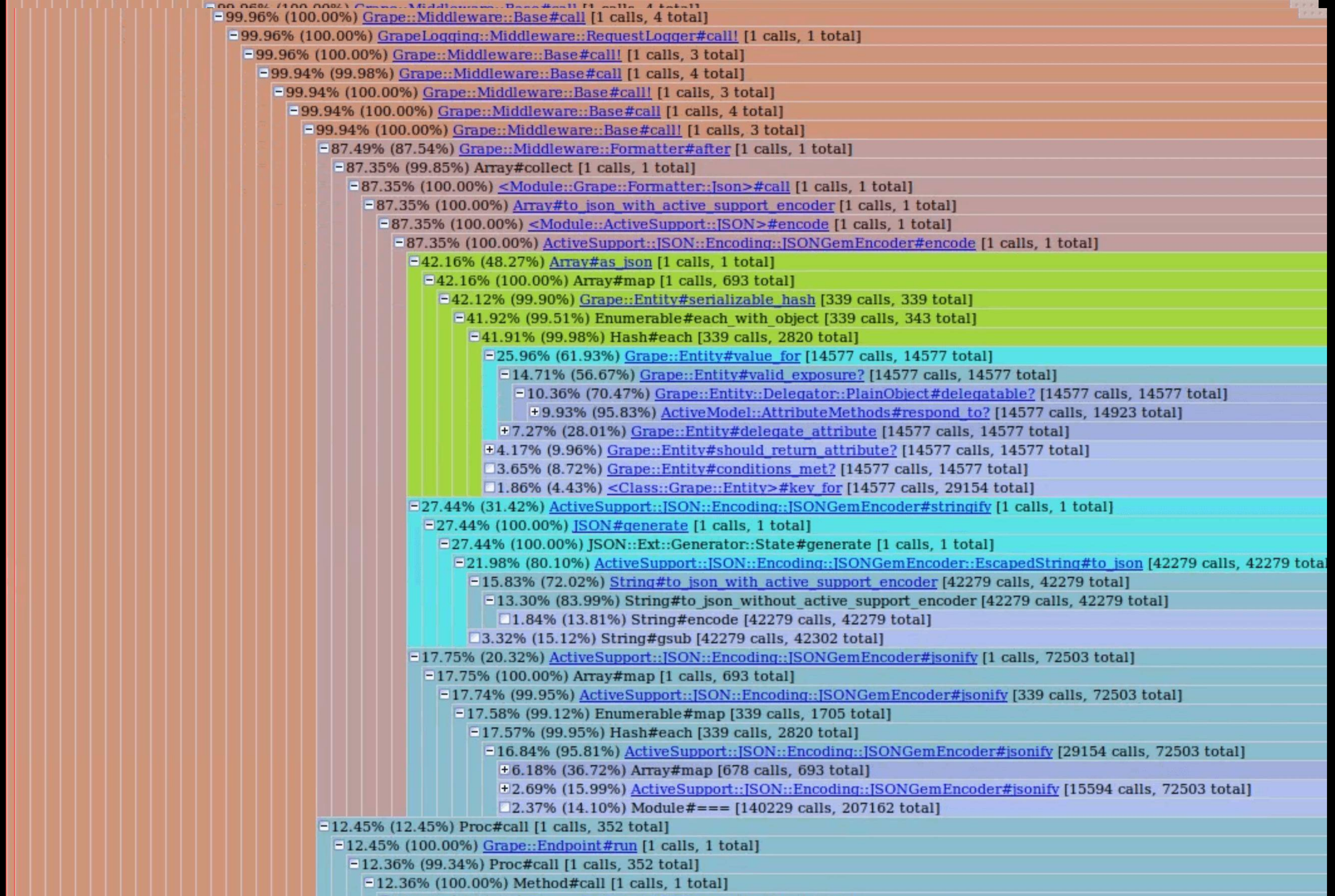
Measure	Mode: wall_time					
Thread	ID: 70311995300360					
Fiber I	D: 70311999772520					
Total:	4.00332					
Sort by	: self_time					
%self	total	self	wait	child	calls	name
100	4.003	4.003	0	0	1	Kernel#sleep
0	4.003	0.000	0	4.003	1	Global#[No method]
0	4.003	0.000	0	4.003	1	Object#sleeper

STATISTICAL TRACING


STACKPROF
RUBY-PROF



puma-2.12.2 (420 samples - 99.53%)
rack-1.6.4 (420 samples - 99.53%)
railties-4.2.6 (420 samples - 99.53%)
rack-mini-profiler-0.10.1 (420 samples - 99.53%)
flamegraph-0.9.5 (420 samples - 99.53%)
sentry-raven-0.14.0 (420 samples - 99.53%)
actionpack-4.2.6 (420 samples - 99.53%)
activesupport-4.2.6 (420 samples - 99.53%)
request_store-1.3.1 (419 samples - 99.29%)
quiet_assets-1.1.0 (419 samples - 99.29%)
web-console-2.2.1 (418 samples - 99.05%)
better_errors-2.1.1 (418 samples - 99.05%)
activerecord-4.2.6 (403 samples - 95.50%)
clearance-1.10.1 (383 samples - 90.76%)
jquery-fileupload-rails-0.4.7 (382 samples - 90.52%)
actionview-4.2.6 (381 samples - 90.28%)
eventstream (380 samples - 90.05%)
2.3.1 (377 samples - 89.34%)
active_model_serializers-c029c00758d5 (376 samples - 89.10%)
thread_safe-0.3.5 (112 samples - 26.54%)
binding_of_caller-0.7.2 (47 samples - 11.14%)
activemodel-4.2.6 (31 samples - 7.35%)
nilify_blanks-1.2.1 (27 samples - 6.40%)
did_you_mean-1.0.0 (27 samples - 6.40%)
arel-6.0.3 (13 samples - 3.08%)
acts_as_tenant-0.4.0 (8 samples - 1.90%)
paranoia-2.1.3 (6 samples - 1.42%)
friendly_id-5.1.0 (5 samples - 1.18%)
json-1.8.6 (4 samples - 0.95%)
kaminari-0.16.3 (2 samples - 0.47%)
pubnub-3.8.0 (1 sample - 0.24%)
event_machine.rb (1 sample - 0.24%)
eventmachine-1.2.0.1 (1 sample - 0.24%)
jsonapi-0.1.1.beta2 (1 sample - 0.24%)
tzinfo-1.2.2 (1 sample - 0.24%)


99.96% (100.00%) Grape::Middleware::Base#call [1 calls, 4 total]
99.96% (100.00%) GrapeLogging::Middleware::RequestLogger#call! [1 calls, 1 total]
99.96% (100.00%) Grape::Middleware::Base#call! [1 calls, 3 total]
99.94% (99.98%) Grape::Middleware::Base#call [1 calls, 4 total]
99.94% (100.00%) Grape::Middleware::Base#call! [1 calls, 3 total]
99.94% (100.00%) Grape::Middleware::Base#call [1 calls, 4 total]
99.94% (100.00%) Grape::Middleware::Base#call! [1 calls, 3 total]
87.49% (87.54%) Grape::Middleware::Formatter#after [1 calls, 1 total]
87.35% (99.85%) Array#collect [1 calls, 1 total]
87.35% (100.00%) <Module::Grape::Formatter::Json>#call [1 calls, 1 total]
87.35% (100.00%) Array#to_json_with_active_support_encoder [1 calls, 1 total]
87.35% (100.00%) <Module::ActiveSupport::JSON>#encode [1 calls, 1 total]
87.35% (100.00%) ActiveSupport::JSON::Encoding::JSONGemEncoder#encode [1 calls, 1 total]
42.16% (48.27%) Array#as_json [1 calls, 1 total]
42.16% (100.00%) Array#map [1 calls, 693 total]
42.12% (99.90%) Grape::Entity#serializable_hash [339 calls, 339 total]
41.92% (99.51%) Enumerable#each_with_object [339 calls, 343 total]
41.91% (99.98%) Hash#each [339 calls, 2820 total]
25.96% (61.93%) Grape::Entity#value_for [14577 calls, 14577 total]
14.71% (56.67%) Grape::Entity#valid_exposure? [14577 calls, 14577 total]
10.36% (70.47%) Grape::Entity::Delegator::PlainObject#delegatable? [14577 calls, 14577 total]
9.93% (95.83%) ActiveModel::AttributeMethods#respond_to? [14577 calls, 14923 total]
7.27% (28.01%) Grape::Entity#delegate_attribute [14577 calls, 14577 total]
4.17% (9.96%) Grape::Entity#should_return_attribute? [14577 calls, 14577 total]
3.65% (8.72%) Grape::Entity#conditions_met? [14577 calls, 14577 total]
1.86% (4.43%) <Class::Grape::Entity>#key_for [14577 calls, 29154 total]
27.44% (31.42%) ActiveSupport::JSON::Encoding::JSONGemEncoder#stringify [1 calls, 1 total]
27.44% (100.00%) JSON#generate [1 calls, 1 total]
27.44% (100.00%) JSON::Ext::Generator::State#generate [1 calls, 1 total]
21.98% (80.10%) ActiveSupport::JSON::Encoding::JSONGemEncoder::EscapedString#to_json [42279 calls, 42279 total]
15.83% (72.02%) String#to_json_with_active_support_encoder [42279 calls, 42279 total]
13.30% (83.99%) String#to_json_without_active_support_encoder [42279 calls, 42279 total]
1.84% (13.81%) String#encode [42279 calls, 42279 total]
3.32% (15.12%) String#gsub [42279 calls, 42302 total]
17.75% (20.32%) ActiveSupport::JSON::Encoding::JSONGemEncoder#jsonify [1 calls, 72503 total]
17.75% (100.00%) Array#map [1 calls, 693 total]
17.74% (99.95%) ActiveSupport::JSON::Encoding::JSONGemEncoder#jsonify [339 calls, 72503 total]
17.58% (99.12%) Enumerable#map [339 calls, 1705 total]
17.57% (99.95%) Hash#each [339 calls, 2820 total]
16.84% (95.81%) ActiveSupport::JSON::Encoding::JSONGemEncoder#jsonify [29154 calls, 72503 total]
6.18% (36.72%) Array#map [678 calls, 693 total]
2.69% (15.99%) ActiveSupport::JSON::Encoding::JSONGemEncoder#jsonify [15594 calls, 72503 total]
2.37% (14.10%) Module#=== [140229 calls, 207162 total]
12.45% (12.45%) Proc#call [1 calls, 352 total]
12.45% (100.00%) Grape::Endpoint#run [1 calls, 1 total]
12.36% (99.34%) Proc#call [1 calls, 352 total]
12.36% (100.00%) Method#call [1 calls, 1 total]

Total Time (Time of Parent)

50% (25%)

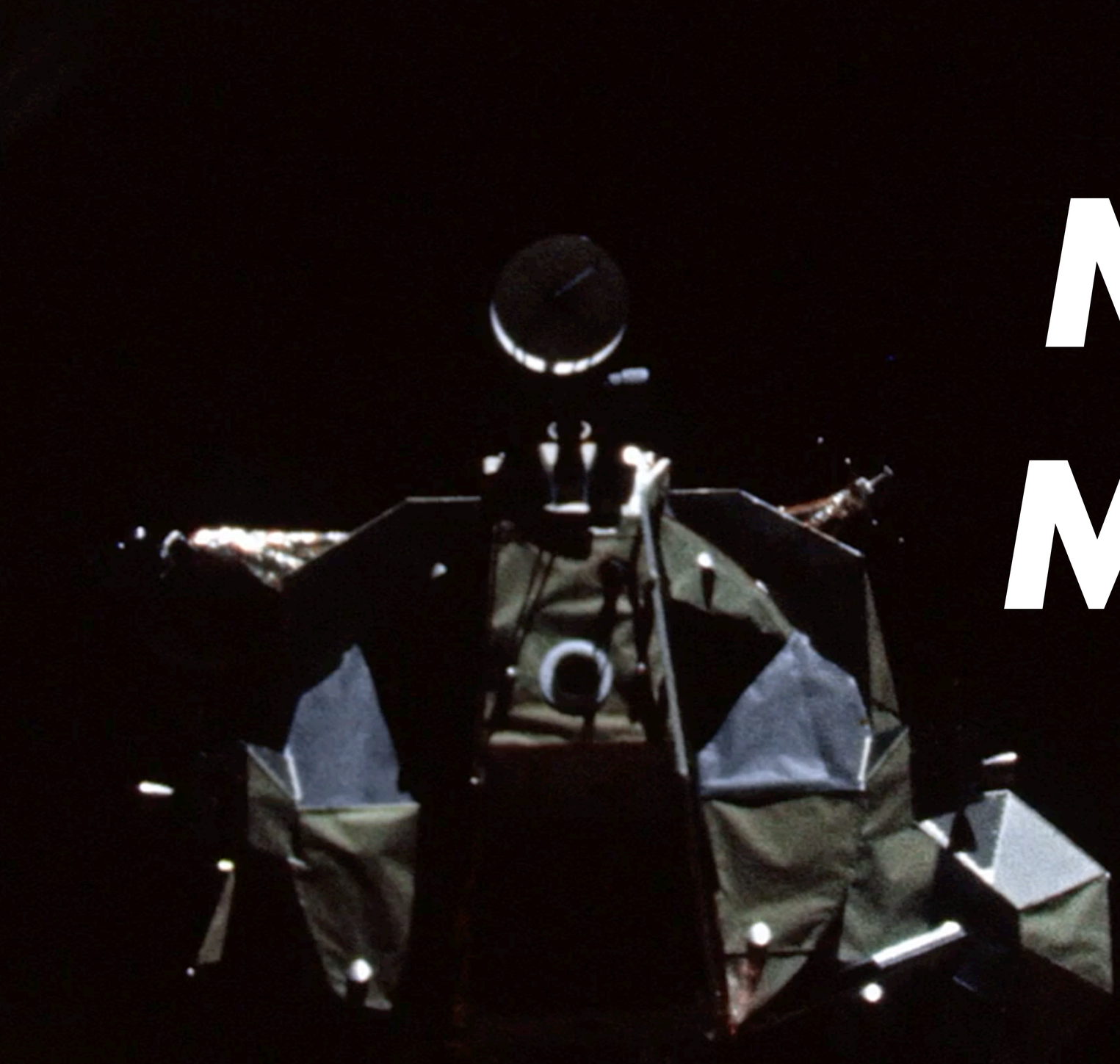
98% (99.9%)

90% (99.9%)

TRY DIFFERENT VIEWS/FORMATS


memory_profiler

rack-mini-profiler



MICRO MACRO APP

```ruby
require 'benchmark/ips'

Benchmark.ips do |x|
  x.config(:time => 5, :warmup => 2)

  x.report("simple") { Raven.capture_exception(DIVIDE_BY_ZERO) }
  x.report("rails") { Raven.capture_exception(RAILS_EXC) }

  x.compare!
end
```

```
Warming up --------------------------------------
              simple    17.000  i/100ms
               rails     6.000  i/100ms
        lots o logs    15.000  i/100ms
Calculating -------------------------------------
              simple    156.087  (±26.3%) i/s -    714.000
               rails     56.316  (±26.6%) i/s -    264.000
        lots o logs    173.704  (±11.5%) i/s -    855.000

Comparison:
        lots o logs:       173.7 i/s
              simple:       156.1 i/s - same-ish: difference falls within error
               rails:        56.3 i/s - 3.08x slower
```

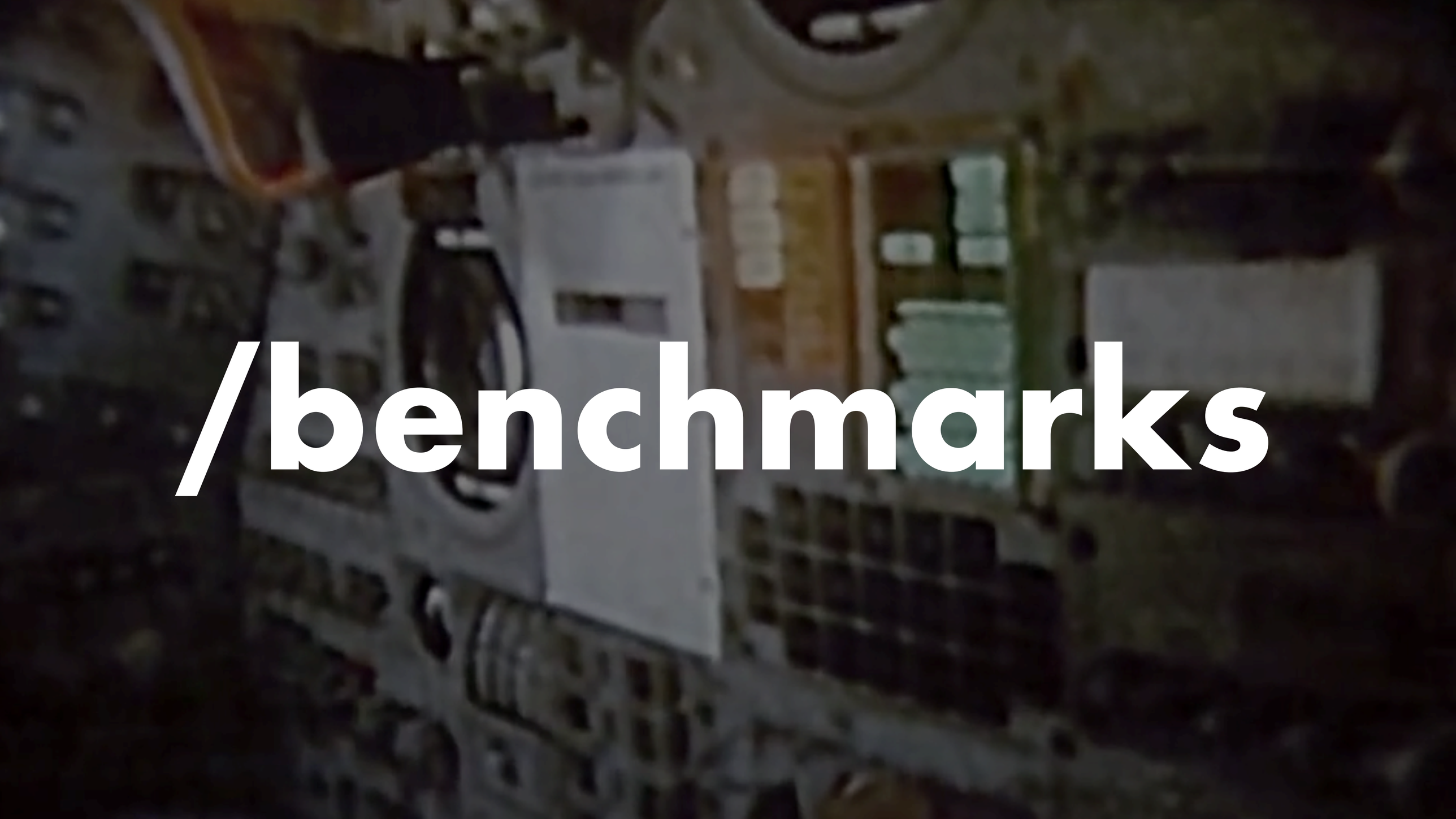
```
Allocations ------------------------------------
              simple    863/32  alloc/ret      50/14  strings/ret
               rails   5895/14  alloc/ret       50/7  strings/ret
         lots o logs  4853/410  alloc/ret      50/50  strings/ret
Warming up --------------------------------------
              simple    17.000  i/100ms
               rails     6.000  i/100ms
         lots o logs    15.000  i/100ms
Calculating --------------------------------------
              simple    156.087  (±26.3%) i/s -    714.000
               rails     56.316  (±26.6%) i/s -    264.000
         lots o logs    173.704  (±11.5%) i/s -    855.000

Comparison:
         lots o logs:      173.7 i/s
              simple:      156.1 i/s - same-ish: difference falls within error
               rails:       56.3 i/s - 3.08x slower
```

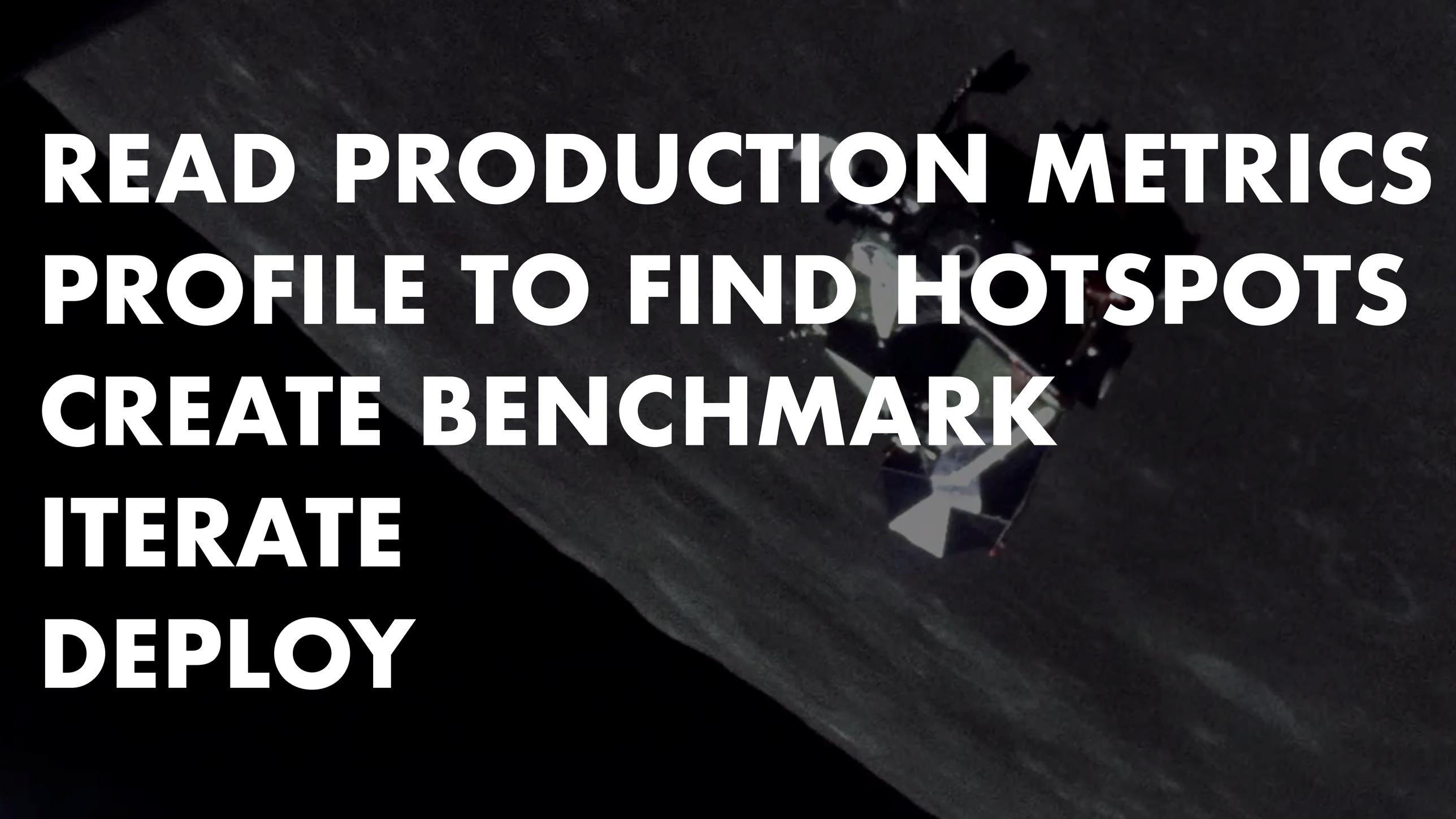
WARMUP
long enough
for STABLE RESULTS

BENCHMARK TIME
should be long enough
FOR LOW VARIANCE

BEWARE the LOOP

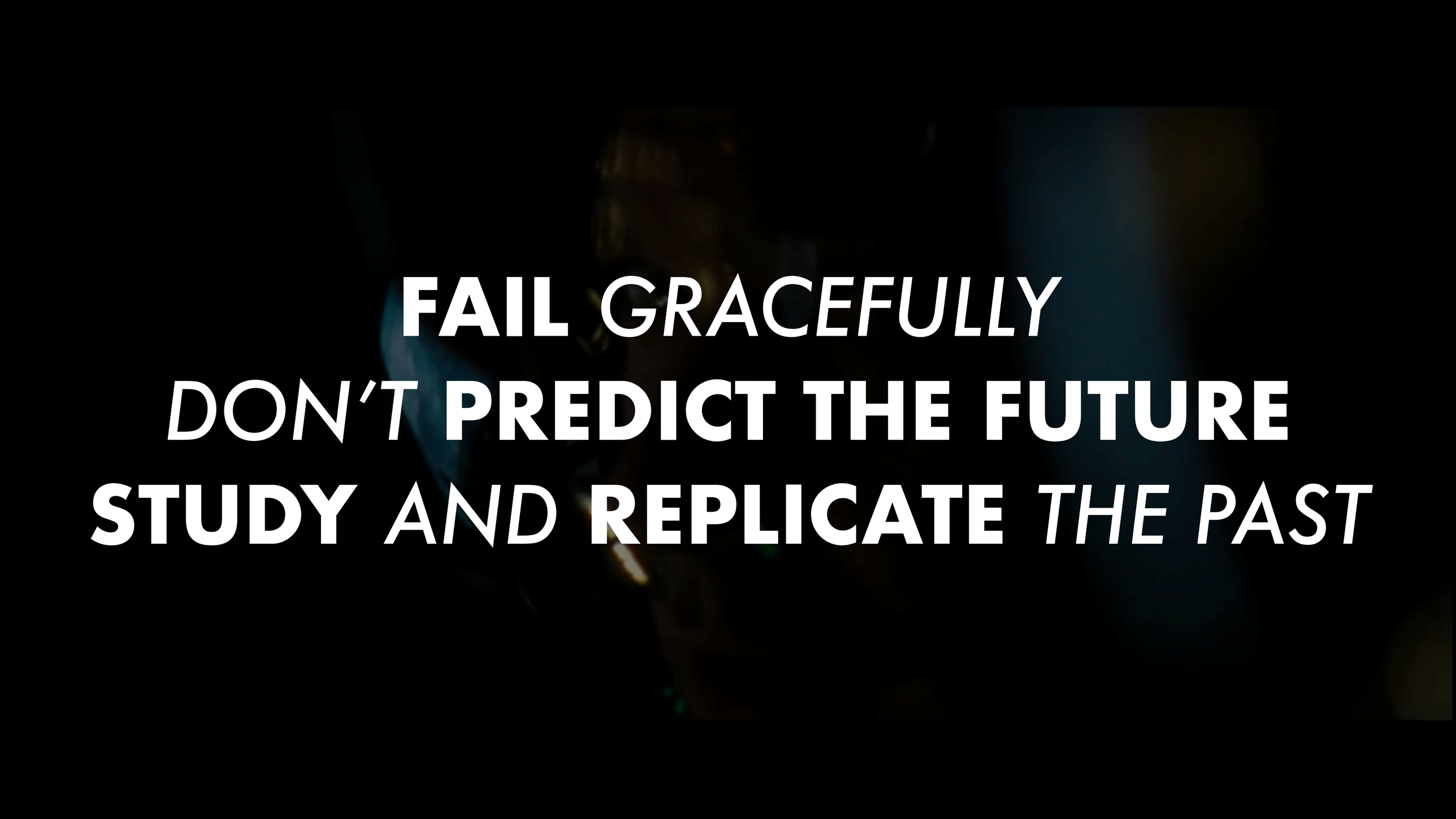
```
require 'benchmark/ips'

user = User.find(1)

Benchmark.ips do |x|
  x.report(“ar”) { User.find(1).do_thing }
  x.report(“ar2”) { user.do_thing }

  x.compare!
end
```


/benchmarks


READ PRODUCTION METRICS
PROFILE TO FIND HOTSPOTS
CREATE BENCHMARK
ITERATE
DEPLOY

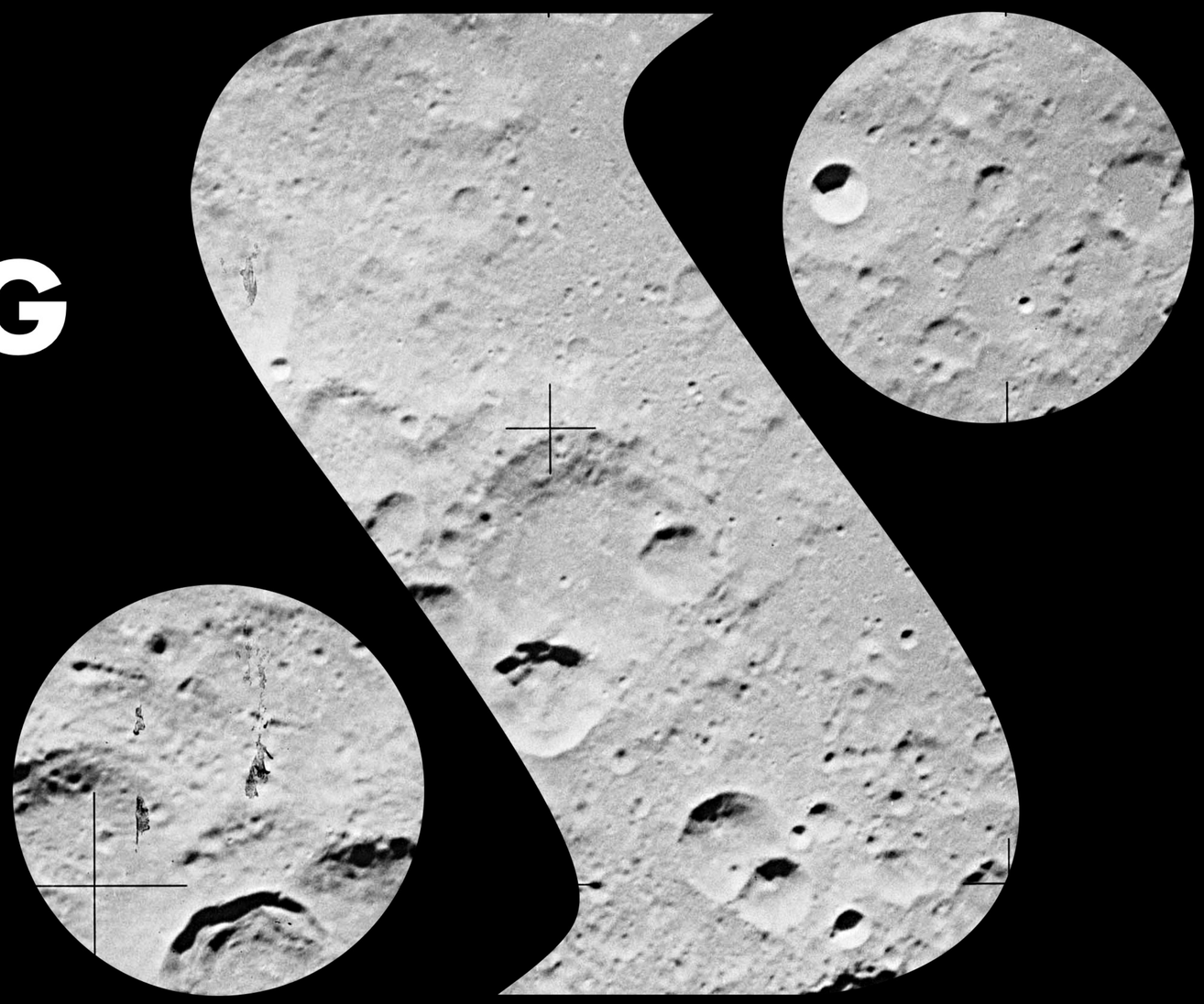
FLIGHT PLAN

Why Measure?
Benchmarking vs. Profiling
Profiling Jumpstart
Benchmarking Jumpstart


FAIL GRACEFULLY
DON'T PREDICT THE FUTURE
STUDY AND REPLICATE THE PAST

speedshop

BENCHMARKING and PROFILING in RUBY



Railsconf 2019