RubyConf Denver

11/8/2021

☆ 8:30 am registration
☆ 9:30 am breakfast
11 am workshops
☆ 12:30-1:00 lunch
1:30-3:30 workshops
☆ 2-3pm Ruby Podcast #608
2-5pm In person live streams
3-5pm Happy hour #registration area
☆ 3:30-5:30 pm Exhibit hall #4 seasons 2-4

Virtual

8am async ruby
9am parallel testing w/ ractors
10am programming w/ something
☆ 11am variable width allocation
12pm Dungeons and collaboration
☆ 1pm ~~12 pm~~ not a talk about airplane crashes
2pm black swans in open source
3pm perceptual learning == more ruby experts?
4pm beyond blameless
5pm hello computer: voice recognition
6pm picoruby
7pm performance scaling
☆ 8pm debug.gem
9pm Ruby in IDE and TypeProf

Shopify

11/8/21 **11am: Variable Width Allocation**

★ Memory model:

Heap page → freelist ⇒ freelist pointers

Singly-linked list of empty slots
- ↳ allocations are fast and constant time

★ GC: when it's full, we must gc
- ↳ stop the world!
  - ↳ marking - determine what are alive. Breadth-first search of live objects, starting with root objects, and find everything that is reachable
  - ↳ sweep. reclaim any unmarked objects
  - ↳ compaction
    - step 1: move obj to start of heap, leave fwd. address
    - step 2: update object references to new address (if they point to a fwd. address)

★ Large objects in the heap

"embedded" strings vs heap allocated strings

< RString-embed-len-max
- ↳ store directly, or malloc and store a pointer
- ↳ issues: malloc is not cheap, results in cache misses
  - → ruby 2.6: transient heap, reduces malloc calls
- ↳ Variable Width Allocation project:

variable pool sizes ← (allow ruby to manage its own memory. Avoid Malloc calls → move array data from system heap to ruby heap

★ We can now embed strings into larger pool sizes!
- ↳ resizing must fall back on malloc
- ↳ no great solution for this yet.

github.com/shopify/ruby

2pm: Ruby Podcast Live 11/8/21

"Listening to Ruby podcasts is like having friends who live in your head."

Episodes:

- Jason - Sandi Metz
- 100th episode - ~~Jason~~ Remote Ruby
- Adhd episode - Remote Ruby

What is interesting

Jason
- worst: something to promote or preach about
- best: conversations about random stuff

→ The bar for what other people find interesting is way lower than you might imagine.

→ Intuition: how do we know? well it's stuff we're interested in. Also stuff we know audiences are on. Also a degree of "guest swiping."

Ruby Heroes:

- Aaron Patterson
- WNB.RB
- Brittany
- Sandi Metz
- Hernesto Tay..
- Chris Oliver
- _why
- Obie Fernande #Rails way

Up and Coming:

- Colin J..bear?
- Emily Giurleo
- Japanese Ruby Twitter
- Jemma
- Andrew A(rcone?) → gems
- Jason Swett
- Brandon

High!

my name is

Nathan Griffith

Betterment

Weedmaps event

11/9 ←

11/9 9:00am Keynote

"I think we can all agree that Computers are the worst." - [redacted] (audience claps)
???

Beyond 3.0

1. "Everyone goes static typing"
   Matz does not like type declarations
   ↳ against DRY principle
   Yet, we want stronger checks, earlier error detection, and code completion
   → Accomplish these with tools, not type declarations.
   ~ RBS ~ TypeProf ~ Steep/Sorbet

2. Concurrency: Async fibers for I/O-bound and Ractors for CPU-bound

3. New Syntax: Pattern Matching (2.7) and Numbered block parameters
   "Succinctness is power"

4. Performance: Ruby 3x3

→ Ruby Ux was from language, but these days (now) it is from tools.
"Tools are more important today"
→ solargraph, sorbet, rubocop, but we need more
"Ruby is a Human-Oriented language"
↳ Performance is not the first priority
↳ Ruby devs are willing to pay the price of migrating not because of better Ux, but because of faster performance.

Performance heals problems, ~~[illegible]~~ fixes a reputation

Microbenchmarks are not related to real-world perf.
↳ but they contribute to reputation

Ruby 3x3 Ready "in some benchmarks" (audience laugh)

"I am a Dreamer" ⇒ "I see my Dream come true"
⇒ "I am still a dreamer"

10:10am How To Make a Gem of a Gem #603

1. - `bundle gem` command
   - write the MVP
   - `rake release`
2. How to make a CLI
   ↳ set up plumbing. gemspec → exe folder
   ↳ Load-path unshift, etc, make a CLI class that accepts ARGV

☆ How do we make a gem good? 4 C's
Clarity, color, caret, cut

1. "Dedicate space for mental clarity" → new folder, new gem
2. "Surround yourself with mild irritants" → be "core mod"
   "Don't let your priors color your thinking"
3. ~~[illegible]~~ "Beware optional modes" – combinatorial problem
   "If it works for you, it'll work for most"
4. "I was proud to make gimme. I was proud to share methy"
   Maintainable, resilient, inclusive, exemplary
   "Never regret measuring twice and cutting once"

10:50am ACIDic Jobs: A layman's guide to job bliss #605
@fractaledmind → Stephen Margheim
→ What is a job? A job is a verb. (like a model is a noun)
  ↳ Jobs represent mutations → they are side effects
→ "Jobs must be idempotent & transactional"
→ ACIDic job principles
  ↳ using a database record "JobRun" ↳ lock on that.
    JobRun → uniqueness on job-class and job_id
      → or uniqueness on job-class and args.
    ↳ configurable: uniquely-identified-by-X

do you "trust" enqueues ~~to~~ ~~do~~

Staging jobs co-transactionally → `deliver_acidic`
  ↳ after-commit instead of a de-staging process

Stepwise idempotency → add a "recovery point"
generic → "perform_step" abstraction
sidekiq batches → multi-step workflow → why? decomposability overall

Q's

- singleton jobs: transactional dequeue + enqueue of the next job.
- bulk de-staging
  - ↳ outbox pattern / job drain pattern
  - ↳ have our cak

1:00 pm - Building Native Extensions
Isolated, System, etc...

1. C extensions - compile & link
   extconf.rb → "mkmf" (make makefile)
2. What can go wrong?
   ↳ you need make/gcc/libs/etc
3. Accessing an external library
   Strategy 1: "system" libraries
   ↳ use library already in system
   issues: dirs, versions, compile-time features
   Strategy 2: "packaged" libraries

UX: nokogiri one of the most frustrating gems

Strategy 3: pre-compiled libraries
→ rake-compiler-dock
x-compile for other platforms

"Test all the f'in time"

bad: System-specific compiler optimizations
gud: UX, maintenance costs, Customer support

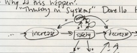
"Trust" → supply chain. checksums!
opt your gems into MFA for all future versions
↳ gem signing!

1:40 pm Nickolas Means – taking 737 to the max

→ Boeing being run as a business, instead of as an engineering firm.
↳ A culture of "win at all costs"

→ MCAS – a software system that emulated the previous handling of the plane.
↳ omitted from flight manual, with FAA approval, based on stale information.

☆ Why did this happen?
"Thinking in Systems" Donella H. Meadows



→ feedback loops
→ all actors were informed by existing safety. Every actor borrowed a little safety in favor of everything else.

→ Cumulative effect of the system == tragedy
→ systems tend to be adaptive and interested in their own survival.

2:20 pm Your Team, as Saga

Narrative shapes our World
↳ narratives shape teams, teams shape code, code shapes narrative/teams.
The stories we tell ourselves feedback into these systems.

How to get a team on track?
Use Narrative as a lever.
world-construction and storytelling

"the"
Tickets are presented as grey blobs. -
Can we fill in narrative details?

The core of any story is conflict.
- Person vs. Environment
- M.I.C.E. - milieu, inquiry, character, event
feature, bugfix, refactor, hardening

"Sprints" create a problematic narrative structure.
forces us to resolve the main "feature" plot without resolving any subplots

How to resolve this? get in the habit of fixing little things. Encourage your teammates to. Change the narrative.

"I don't want people fighting over who gets to be the main character."

→ Romance novels → relationship itself is the main character
→ Focus "stretch projects" around growing a team as a whole.
→ Redemption arcs: gradual. Sinked in time and hard work.

"Advance your team's capacity with every project."
If you build a world where your capacity is growing where everyone is learning, they'll all be amazing.

4pm Squashing Security bugs with Rubocop

Static analysys, ASTs, rubocop → save the nitpicking for the robots

Security antipatterns

↳ more likely to lead to vulns

Example Cop

Developer Experience

↳ How equipped are devs to deal with offenses?

↳ Are the cops easy to configure?

↳ What kind of burden are we adding by introducing these cops?

Minimizing toil

4:40pm Finding Purpose and Cultivating Spirituality

Left a career, founded a tech company.

Fear - willpower, not enough, depleted

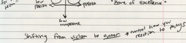
↳ finding spiritual tools, connect deeper w/self, having energy, not take things personally, more joy and happiness

Connecting within: mindfullness & focus

sun on head, rays coming through

Spirituality - finding joy, wellness, calm, self confidence, within you - not because somebody complimented you etc

Self confidence - finding your super powers, your flow



Shifting from victim to owner → control how you reaction to things

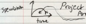
Using Affirmations and Living Visions

Audience: Skiing - "find your line" - don't focus on the trees

9:00 am

11/10 On the Care and Feeding of Feedback Cycles

Deming cycle - Plan → Do → Check → act
OODA loop - observe - orient - check - act
Lean startup - build → measure → learn
scientific method - hypothesize → experiment → observe measure/



Project lifecycle →
Area under the curve is Risk

Schrödinger's Release → we don't know if the product is alive or dead until we release

- Theory of Agile vs Reality of Agile
- The "speculation build up" → even if you "release" in small increments.
  ↳ yet, if you wait for that "in the wild" feedback, you've waited too long
- Different types of feedback, with different cycle times
  unit test, ci, exploratory testing
  acceptance test, stakeholder feedback, user feedback
- PR code review cycle - enforcing untrustness in the org (latency & cycles)
  ↳ rather than a review step, pair up front
- Hilarious long-running team branch diagram
- Test Pollution
  ↳ flaky tests / false failures
  ↳ stop shipping until you can trust your tests
  unpopular, but customers and support needed this

fixes: short cycle time!

Healthy Feedback Loops

↳ why unit test if you get system coverage?
different cycle time

Kolb learning cycle: experiment → ~~ex~~experience → observe/ reflect → abstract gem: `memo_wise`

10:10 am Achieving fast method metaprogramming #605

Traditional memoization
- falsey values
- cumbersome, especially w/ multiple args
- pollutes namespace (methods and vars)

`memoize` directive
- made data pipeline faster
- wasn't fast enough in cases of millions of calls

Observe, Hypothesize, ~~Experiment~~, Analyse
benchmarks ↵
- ~~Disable GC~~ when benchmarking
- split benchmarks by a dimension, normalize to 1.0x per dimension, to test iterations of improvements

Perf improvements
- initialize cache in initializer, instead of with ||=
- use module_eval w/ string, instead of define_method with block
  ↳ obviates need for "send" ⇒ hardcode the call
- switch key from [method, args, kwargs] to [method, args] if kwargs is empty. Also, flatten array. use [method] if no args.
- switch from [method, arg1] ⇒ value to method ⇒ {arg1 ⇒ value}, first with single arg (no array) then with [arg1, arg2] ⇒ value
- use array + index referencing method, hardcode index in method string
- use a sentinel array to store if a value has been set (since using array means no key?)
- optimize inner hash for "truthy" case, unless > 3% falsey

fetch(a) to
...
⇓
out = h[a]
if out || h.key?(a)
out

a

Q: How do you learn what incurs a cost? A: tinkering

10:50am Fake your test away: how to abuse your test doubles

- with Ruby the sky is ~~the limit~~ an object.
- Douglas Adams
  - How → survival
  - Why → inquiry
  - Where → sophistication
- Test doubles: stub, mock, spy
- Testing pyramid - in reality (lol)
  - ↳ Model.create == integration test!

→ Integration test wannabes

- → Tests with randomly inserted test doubles because real models are ~~hard to deal with~~ awkward.
- → These:
  - cause random test failures
  - decrease productivity - rabbit holes of implement. details

[Test doubles are contracts]

→ Helicopter Unit Tests

- → Tests that try to control every implementation detail
- → These:
  - create a fake empire, divorced from reality
  - create Brittle tests → implementation :: test.

→ hard-to-use code == hard-to-write test
easy-to-use code == easy-to-write test

→ Tips

- use the Principle of Least Knowledge (→ Law of Demeter)
  (don't talk to strangers → low coupling)
- Separate Decisions and Delegations
  - ↳ color code all lines ⎣ dependency, relationship
  - ↳ split into responsibilities, each focused on decisions
  - ↳ unit test decision methods
    integration test delegation methods
  - ↳ results in net fewer test doubles
- Make dependencies really obvious
  - ↳ specifically, messy dependencies with side effects
  - ↳ encapsulate services in a "repo," use dependency injection

Design your system so there's no room to abuse your test doubles

1pm Compile Sorbet

- API speed is a *feature* → latency is a combination of I/O + Ruby
- API perf - making one thing at a time faster is rarely useful.
  - making everything faster at once is best.
- AOT vs JIT - AOT is simpler, supported by Sorbet/types
- # compiled: true → opt-in, for incremental rollout
- compiler turns Ruby into a DSL for ruby-native extensions

→ Rollout Strategy:

1 - Plan for when things go wrong
2 - Compare performance on real traffic
3 - Must be incremental

separate host

1. compiler tests, entire Stripe test, stage, blue/green, 9
2. deploy on a subset - canary builds - ~~Stackprof~~
3. → use Stackprof to find things to compile, get compiler working w/ those files first

Q: ? A: it gets faster even without type info → due to control flow
Q: perf improvements? A: Not available yet
Q: How does it know to replace Array each w/ C func
→ Sorbet supports marking methods as "final"

1:40 pm Contractualism & Software Engineering

Agent-centered ethics → is this person a good person?
↳ allows person to do bad things
Consequentialism → ends justify the means
↳ tech does this all the time
Non-consequentialism
→ Contractualism - combines these nearly
↳ an action is morally right if it is justifiable within your community - anyone can veto

Professions w/ code of ethics
Dr, lawyers, architects

Hard for tech to enforce a code of ethics
→ violaters, have nothing to lose
(not licensed)

* "User Stack" (vs ~~[illegible]~~ "Tech Stack")

Company → Me → Engineers → fellow employees
→ customers → end users → end users' users → World

customers	end users	end users' users	World
opt-in	accessible?	affected, has no choice	is this good for the world?

- find a solution that nobody in the group will reject on principle!
- When making a decision, ask who it benefits, and who it harms.
- Harm tends to cascade downwards.

Examples
- Google Youtube desktop/mobile pricing example
- ~~Algorithms~~ Weapons of Math Destruction
  ↳ Cathy O'Neil
- → toggle to downschedule shifts to prevent benefits
- → face recognition software in Detroit, arresting the wrong people

How to live a contractualist life?
- Try to do it. Try to imagine objections people might have. Put our focus in the right places.

practical tips → blameless ethics retro
use your privilege
make use of tools at your disposal ← Trust, Safety, ES ethical use requirements
push back as a group
remember: the buck stops w/ you
Don't give up!
↳ work is an economic reality, you will fail but you can move the line

How we sped up

2:20: Cvars in Ruby 3.1

- why are cvars complex? Inheritance.
  Cvars are shared among descendents → global!
- fun fact: ivars and cvars are stored in the same hash table
- Every time a class variable is assigned or called, Ruby must check up the entire inheritance tree!
  - ↳ this will impact perf on any deep heirarchy
- Is a perf fix worth it? How often are these used?
  - ↳ Rails makes extensive use of cvars (mattr-accessor, etc)
- Benchmarking: shows that 100 modules now behave like 1 module
  → 6-7% improvement in Rails ~~response~~ request time
- Every ~~OSS~~ change has trade-offs
  - ↳ Ruby dev team had concerns, because this cache increased cvar complexity
  - ↳ Another concern: this might encourage more people to use cvars. (core team, et. al, hates cvars)
  - ↳ Increased maintenance for core team. Responsibility is theirs once it is merged.
- ↳ Every ~~OSS~~ change is a negotiation
  - ↳ Cvars aren't going anywhere, in some cases they are the only available tool
  - ↳ demonstrated real world improvements
  - ↳ improvement to all libraries, consumers
- ↳ How to start working on Ruby
  - ↳ Learn C (be a "C tourist")
  - ↳ Read "Ruby under a Microscope"
  - ↳ Make small changes (doc updates, fix bugs, benchmarks)

4:00pm Dishonest Software [fighting back against industry norms]

- User-focused security
  - ↳ slack employees instead of locking down
- honest.security ← URL
- Most people do not intend to build or benefit from dishonest software yet, many of us will.

→ GE and APT - infiltration of IP
- ↳ wireshark, monitor employee traffic
- ↳ legal because of 1986 ECPA law
- ↳ limit: they can't remotely activate webcam

→ Legal, but how do you justify it?
- ↳ "our mission is pure," we are vetted, good people, etc ends justify the means

→ employee got fired because of false positive match

Honest: Trust us, because you can independently verify that we are telling the truth.

Not Honest: Trust us, we're the good guys.

Honest: You have the right to know what we can see.

Bad test: Does this software break the law?

☆ Good test: Would requiring informed consent break the software's value proposition.

Dishonest: You should make sure your tool isn't illegal/being used illegally.

How to be honest:
- ask for consent (explicit)
- let them see the data that is collected ↑ without talking to person
- Allow them to revoke consent at any time

Do not advocate just for yourself

Building honest software is a competitive advantage
- ↳ more regulations, GDPR, device vendors
- ↳ People who make dishonest software can justify other dishonesty → create a culture of honesty

4:40 closing keynote

Whimsy

- ↳ pry. prefix w/ "." to run bash. (.vim ...)
- ↳ cd into objects, or out w/ cd ..
- ↳ edit | ClassName → opens vim
- ↳ ls | object
- ↳ _ex_ | → get 1st exception
- ↳ wtf | shows last exception → more ! = more stacktrace
- ↳ hist | previous commands
- ↳ amend-line N !

Matz Q&A

- YJIT will not fully replace MJIT ↗ multi-tier if both model available
  - ↳ intel only for now ↳ uses gcc (very optimized)
- mruby 3.1 → trying to be cruby 3.0 compat
- async fibers feels nostalgic - tricks from green threads in 1.8
- revamping 'async' gem for using higher order API
- RBS → catching up to TypeScript experience
- Matz still uses emacs → VS code "it's okay"
- Matz not satisfied with C ⟷ Rust integration
  - ↳ but if it became more interoperable, it might hurt Rust
- Matz likes Zig ↳ Rust for ruby extensions? in gems?
- Matz doesn't use a ruby version manager - just renames bin
- focus for 3.1 - 4.0
  - ↳ rich set of tools (stargraph, rubocop)
  - ↳ new language features less likely to happen
  - ↳ focusing on perf / microbenchmarks ruby 3x3 redux

11/11/21 Tweet summaries

- Monday - Variable Width allocation
  Avoid system malloc calls, and answering questions of why strings of len x or less perform so much faster.
- Live recording of Ruby Podcasts (rogues, remote, Jason)
  A great "meta" conversation about podcasting, Ruby heroes and up-and-coming Rubyists, and so great to put some voices to faces.
- Tuesday Matz's keynote about Ruby 3.1 and beyond. Ruby 3x3 redux.
  "I am still a dreamer" after seeing his dream come true.
- Always a pleasure to watch @searls talk, this time about writing gems.
  Definitely relate to the idea of being "care mad" => "surround yourself with mild irritants" if you want to solve meaningful problems,
- ACIDic jobs - Stephen's talk gives a great overview of the value of ACID, and how to benefit from ACID when running background jobs. Excited to see an example of the "job drain" / "outbox" pattern built on 'sidekiq'.
- Sidebar: I was incredibly surprised and honored ~~that~~ to see @stephen cite my RailsConf talk. ~~I was so flustered, I couldn't thi~~ I hope I didn't seem too flustered, and it was great to meet you!
- Building Native Extensions => one of the coolest improvements to the ruby ecosystem that impacts basically all Rubyists. And a great shout out ~~to~~ about trusting: supply chains; MFA on rubygems
- Taking 737 to the max. During covid I got into aviation disasters, and n.m. does a fantastic job of exploring the problem in terms of systems and feedback loops that led to tragedy.

→ Your team as a Saga – how so much of our behavior & choices are informed by the stories we tell ourselves, and how the way to resolve conflict is to change the Narrative

→ @omar's talk on Squashing Security bugs w/ Rubocop fantastic view into the mind of a security engineer, meeting us Rubyists where we are with linter rules that are available in `betterlint`

→ Finding Purpose and Cultivating Spirituality. finding the tools (mindfulness, affirmations) to combat fear, find energy, and connect more deeply with ourselves ⇒

→ Wednesday – feedback cycles, the theory of agile vs the reality of agile, and this hilarious git branching diagram

→ Achieving fast method metaprogramming in 'memo-wise'
→ as someone who maintains my own 'memoize' mixin, some of these techniques will come in handy, and the methodology was so well explained.

→ Talk on abusing test doubles. Loved the Douglas Adams reference, and the clear examples of how to factor both code and tests to limit the need for "helicopter unit tests" → separating decisions from delegation is a principle I'm eager to try in my own code.

→ Compiling ruby w/ Sorbet → I Cannot wait to try this out. really good explanation of where AOT compiling could be useful, and how it can be incrementally rolled out.

→ Contractualism & software engineering. Loved the Good Place reference. This talk presents a great model for how to predict the impact of your tech on your users, their users, and the wider world. This talk should be given at every tech conference.

→ Speeding up Cvars. I always love @eileencodes' deep dives and how accessible she makes the internels. Diggest take-away: contributing to OSS is a negotiation, and it's important that we all do it! Definitely gonna try to be a "C tourist"

→ [illegible]'s talk on dishonest software and Problematic industry norms. ~~A must watch~~ If you take anything away you must understand how informed consent ~~matter~~ ~~matters~~ to model your ethics around
→ "Building honest software is a competitive advantage!"

→ quick shout out to @'s "whimsy" session on all of the crazy things you can do with 'pry', including editing your classes in real-time! @rowan you'd love this one.

→ Last, but not least, @matz's q&a, led by @ - ~~looking forward~~ looking forward to the new tools-focused core Ruby development, plus it was fun to hear matz's thoughts on Rust and Zig, and learning from Python