Fantastic Hacks and Where to Find Them

Adapted from Miriam Pena's "BEAM Extreme" talk @ ElixirConf 2019
Slides available at tylerayoung.com

👋 Hi, I'm Tyler

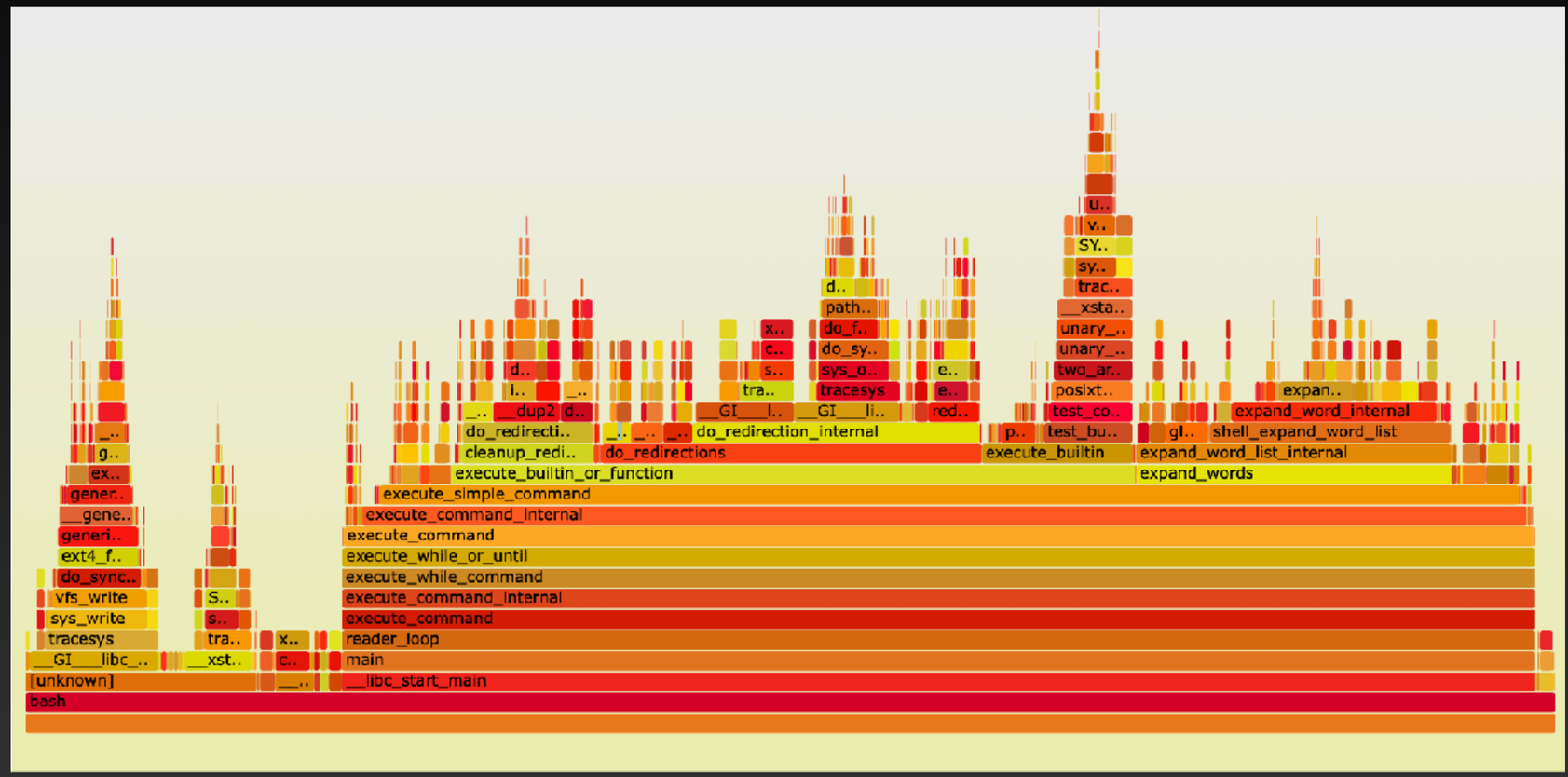
- Writing Elixir since 2019
- Built X-Plane massive multiplayer game server
- Worked at Felt on collaborative mapmaking
- Now starting SleepEasy Website Monitor

Make it work, then make it beautiful, then if you really, really have to, make it fast.

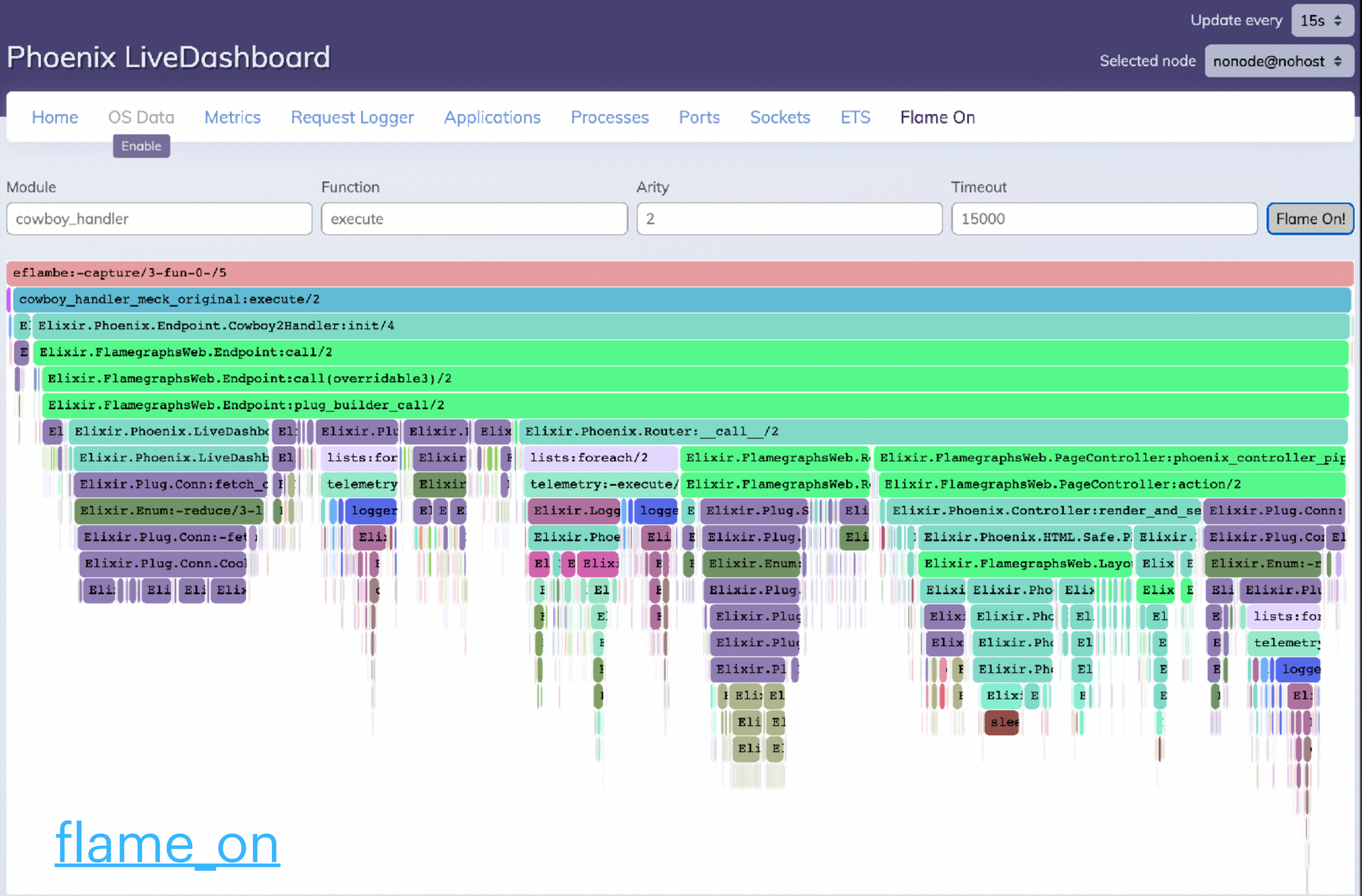
Joe Armstrong, *Erlang & OTP in Action*

But first, profile.

- Perf is usually good enough by default
- Don’t thrash without a reason
- Don’t write horrifying hacks without a *really* good reason



fprof |> eflame


Update every 15s
Phoenix LiveDashboard
Selected node nonode@nohost
Home
OS Data
Enable
Metrics
Request Logger
Applications
Processes
Ports
Sockets
ETS
Flame On
Module
cowboy_handler
Function
execute
Arity
2
Timeout
15000
Flame On!
eflambe:-capture/3-fun-0-/5
cowboy_handler_meck_original:execute/2
Elixir.Phoenix.Endpoint.Cowboy2Handler:init/4
Elixir.FlamegraphsWeb.Endpoint:call/2
Elixir.FlamegraphsWeb.Endpoint:call(overridable3)/2
Elixir.FlamegraphsWeb.Endpoint:plug_builder_call/2
Elixir.Phoenix.Router:__call__/2
lists:foreach/2
Elixir.FlamegraphsWeb.PageController:phoenix_controller_pip
Elixir.FlamegraphsWeb.PageController:action/2
Elixir.Phoenix.Controller:render_and_se
Elixir.Plug.Conn:
Elixir.Phoenix.HTML.Safe.P
Elixir.Plug.Co
Elixir.FlamegraphsWeb.Layo
Elixir.Enum:-r
Elixir.Pho
Elixir.Plu
lists:for
telemetry
logge
sle

flame_on

It's probably the database.

- Database queries are the #1 driver of performance in modern web apps
- Beware N + 1 queries

```
Enum.map(event_ids, fn id →
  event = load_event(id)
  # do stuff
end)
```

It's probably the database.

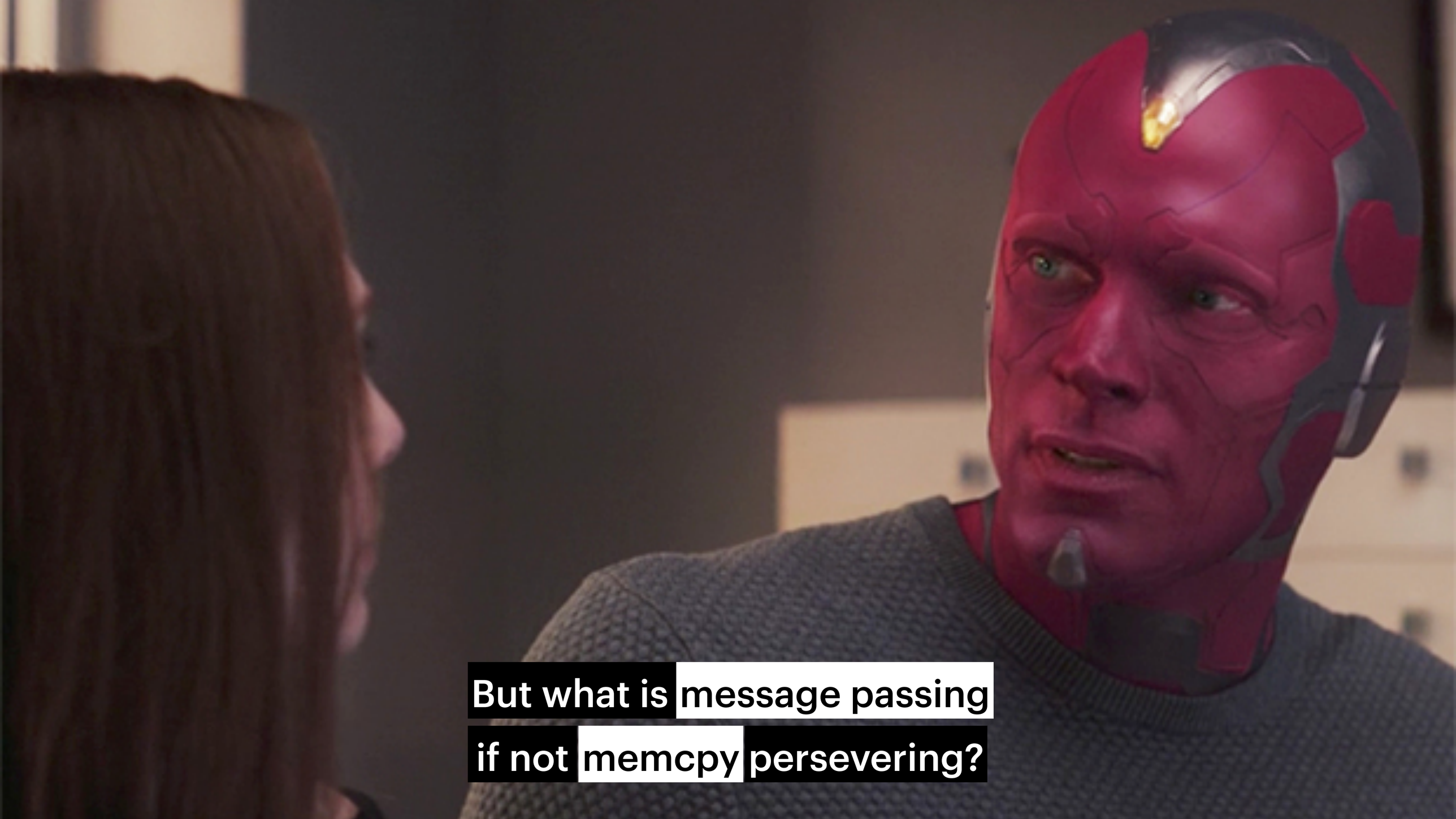
- Database queries are the #1 driver of performance in modern web apps
- Beware N + 1 queries
- Indexing (and multi-column indexing) can help
  - 😎 Read speed (`WHERE`, `ORDER_BY`, `JOIN ON`)
  - 😩 Disk space
  - 😱 Insert speed?

Elixir: Message passing by default

👍 No shared memory, no problem!

😎 Concurrency is harder to screw up

😎 Garbage collection is easy!


But what is message passing
if not memcpy persevering?

Elixir: Message passing by default

👍 No shared memory, no problem!

😎 Concurrency is harder to screw up

😎 Garbage collection is easy!

😤 All those copies eat CPU time

Elixir: Message passing by default*

👍 No shared memory, no problem!

😎 Concurrency is harder to screw up

😎 Garbage collection is easy!

😤 All those copies eat CPU time

* Except atoms, binaries over 64 bytes, literals, & code

Elixir: Message passing by default*

👍 No shared memory, no problem!

😎 Concurrency is harder to screw up

😎 Garbage collection is easy!

😤 All those copies eat CPU time

* Except atoms, binaries over 64 bytes, literals, & code

Don’t copy more than you have to

- Be smart about what you send between processes
  - Consider not separating out a process at all?
  - ”Big Data” is calling from the year 2010:
    Bring your computation to the data, rather than the data to the function
- Look for instances where a data structure is getting copied due to immutability guarantees

```
20  -     {far_index, _, far_squared_dist} =
21  -       Enum.zip(0..(length(segment) - 1), segment)
22  -       |> Enum.drop(1)
23  -       |> Enum.drop(-1)
24  -       |> Enum.map(fn {i, p} -> {i, p, seg_dist(p, first, last)} end)
25  -       |> Enum.max_by(&elem(&1, 2))
22  +     {_, far_value, far_index, far_squared_dist} =
23  +       Enum.reduce(middle, {1, nil, 1, 0}, fn element, {idx, max_val, max_idx, max_dist} ->
24  +         dist = seg_dist(element, first, last)
25  +
26  +         if dist >= max_dist do
27  +           {idx + 1, element, idx, dist}
28  +         else
29  +           {idx + 1, max_val, max_idx, max_dist}
30  +         end
31  +       end)
```

https://github.com/pkinney/simplify_ex/pull/4/files


Call Kenny Loggins, because you're in the *danger zone*!

Choose a better data type

- Lower overhead
  - Prefer `iolist` to concatenation
  - Prefer BIFs & pattern matching to function calls

Choose a better data type

- Lower overhead
  - Prefer `iolist` to concatenation
  - Prefer BIFs & pattern matching to function calls
- Better memory locality
  - Prefer binaries to (char)lists
  - Prefer tuples/records to lists/maps/structs? (😱)

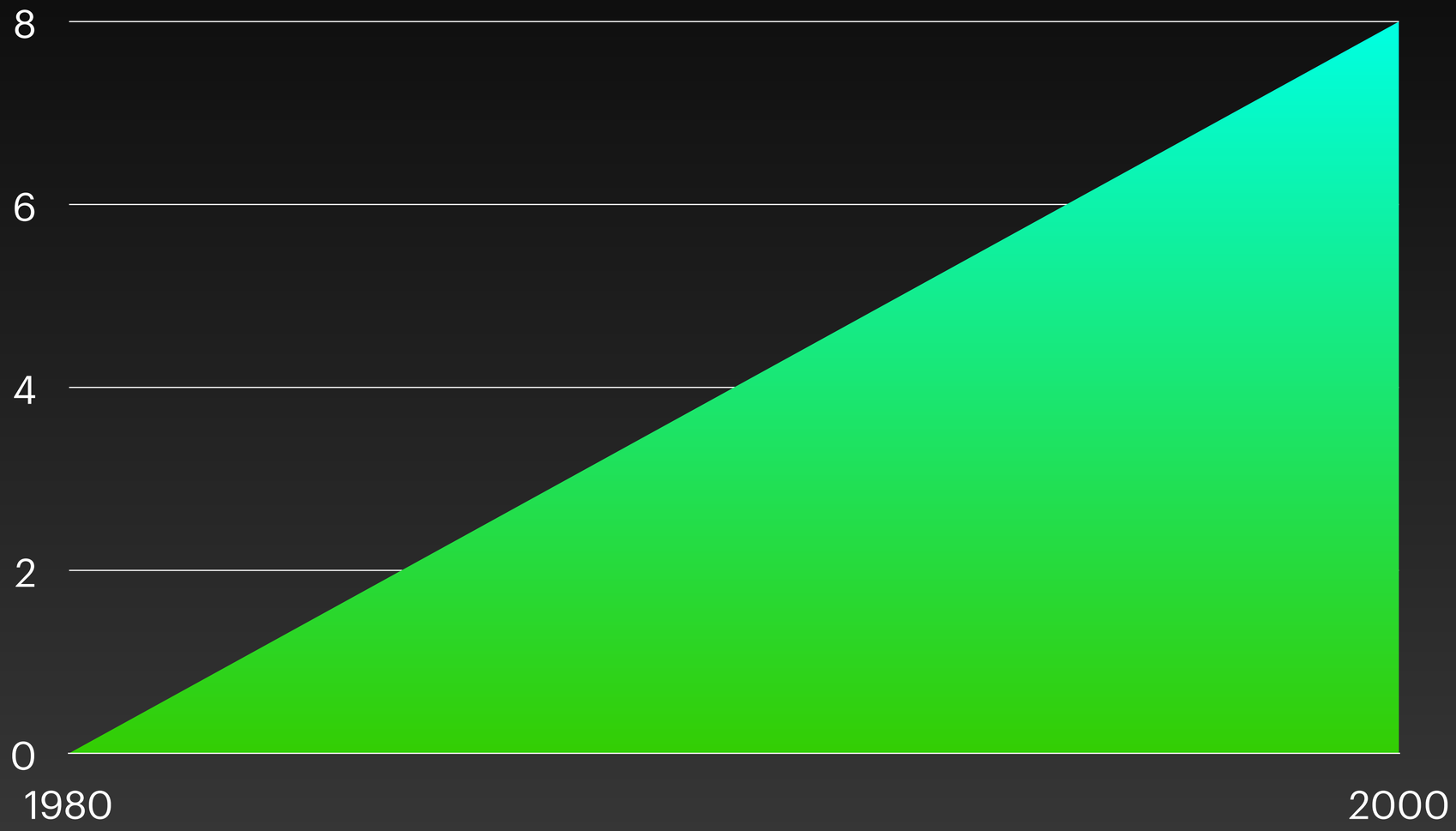
Aside: Memory locality

Aside: Memory locality

Aside: Memory locality

Aside: Why does memory locality matter?

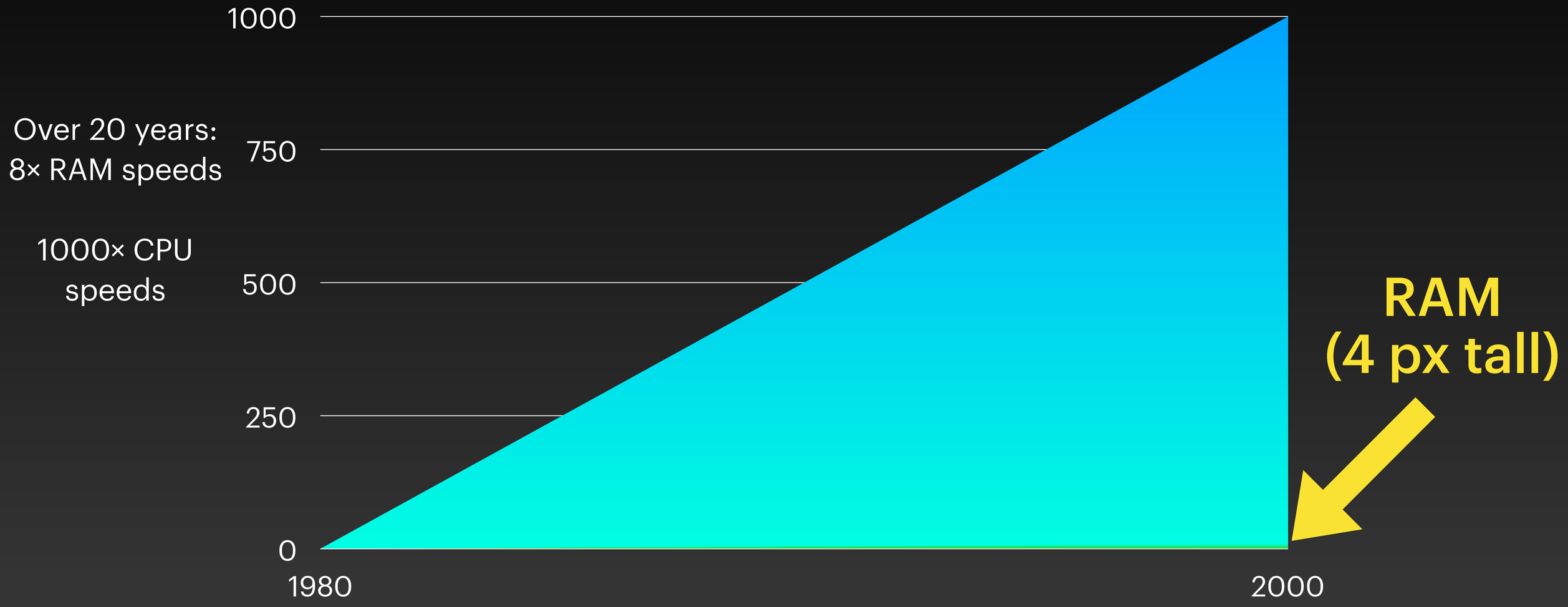
Over 20 years:
8× RAM speeds



@TylerAYoung

Source
tylerayoung.com

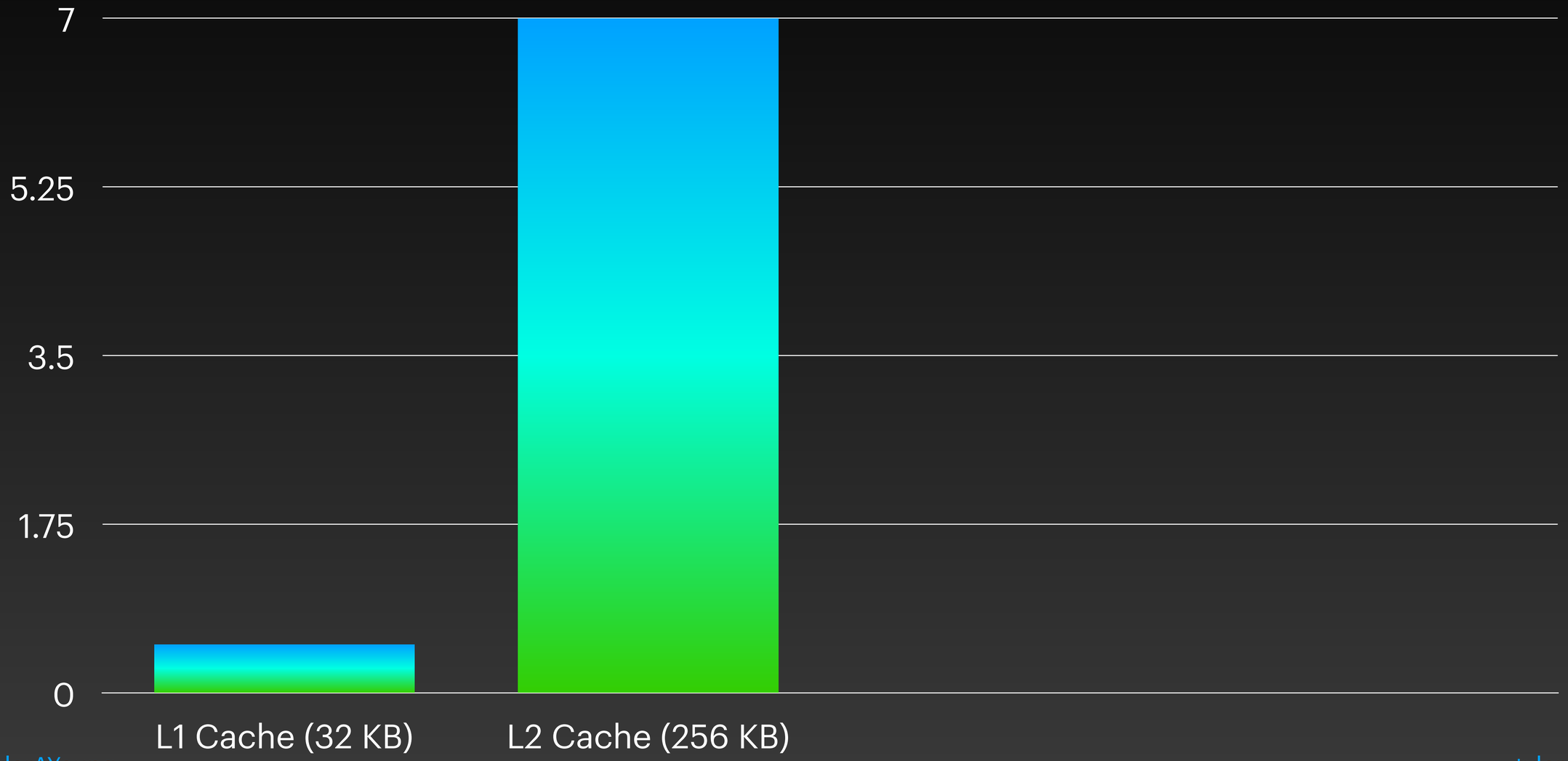
Aside: Why does memory locality matter?



@TylerAYoung

Source
tylerayoung.com

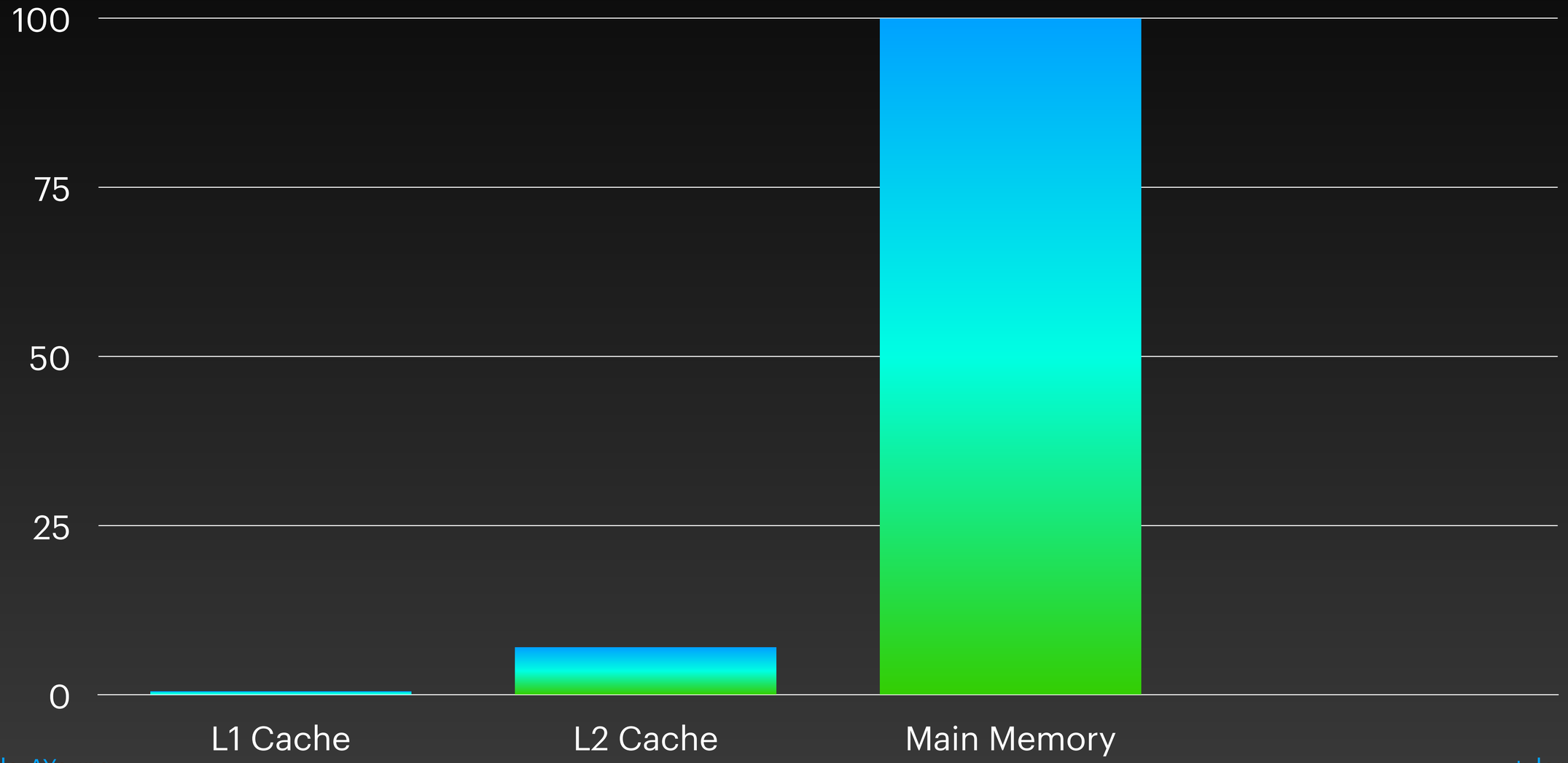
Aside: Why does memory locality matter?



@TylerAYoung

Source
tylerayoung.com

Aside: Why does memory locality matter?



@TylerAYoung

Source
tylerayoung.com

Aside: Why does memory locality matter?

@TylerAYoung

Source
tylerayoung.com

Choose a better data type

- Lower overhead
  - Prefer `iolist` to concatenation
  - Prefer BIFs & pattern matching to function calls
- Better memory locality
  - Prefer binaries to (char)lists
  - Prefer tuples/records to lists/maps/structs? (😱)

Move it to compile time

- Macros can turn computation & data into code
- Can’t change for the life of the program

😎 Not copied between processes!

⚠️ Compile time cost

⚠️ Theoretical runtime cost due to code bloat

Use ETS to share data across processes

- Shared, in-memory table
- `:protected` by default (only writable by the process that creates it)

😎 Concurrent reads (no process bottleneck)

😎 Faster than `GenServer`

😕 Still have to copy data in & out!

⚠️ Exciting new possibilities for race conditions

Skip GenServer in favor of “raw” processes

- `GenServer` is designed for high *availability*, not high throughput
- ”Raw” processes (spawn/send/receive) avoid layers of indirection

😎 ~80% faster per call!

⚠️ Code is harder to read, less idiomatic

Process Dictionary

- Private to each process
- Easy to use: `Process.put(:my_key, my_val)`

😎 Real, real fast to access within the process

Tuple Size	Speed Advantage Over ETS
1	~2×
10	~6×
100	~60×
1,000	~600×
10,000	~6,000×

Process Dictionary

- Private to each process
- Easy to use: `Process.put(:my_key, my_val)`

😎 Real, real fast to access within the process

⚠️ Destroys referential transparency

⚠️ Makes debugging difficult

⚠️ No GC for the life of the process

😱 Survives catch/throw

Persistent Term

- Optimized for heavy reads, very infrequent writes
- Constant time access—no locks, no copies

😎 Much faster than ETS

Tuple Size	Speed Advantage Over ETS
1	~2.5×
10	~3×
100	~6×
1,000	~20×
10,000	~250×

Persistent Term

- Optimized for heavy reads, very infrequent writes
- Constant time access—no locks, no copies

😎 Much faster than ETS

😱 Overwriting can trigger *global* GC pause

Tweaking Process Priority

- Fair scheduling by default
- Can use `Process.flag/2` to make your process jump to to the front of the scheduler queue

😎 80% faster per call

😱 Can starve the scheduler, leading to deadlock

Really in the weeds

- Don’t shuffle argument order
  - E.g., `f(a, b, c)` calling `g(c, b, a)`
- Function overload order matters (sometimes)
- `decompile --to asm`

NIFs for when all else fails

👍 Bridge into “systems language” code (C++, Rust, etc.)

😎 Access to better data structures (goodbye `List`)

😩 Bridging has a measurable cost (need large batches)

😱 Crashing in a NIF brings down the whole app

Summary

👍

- Think about the DB first
- Copy less (functions > processes, bring functions to data)

⚠️

- Better data types
- ETS
- Move it to compile time
- Skip `GenServer`, use processes directly

😱

- Process dictionary
- Process priority
- Persistent term
- In the weeds
  - Don’t swap argument order
  - Function overload order matters (sometimes)