Database disasters, and how to find them.

Christophe Pettus
PostgreSQL Experts, Inc
FOSDEM 2021

The day started like any other.

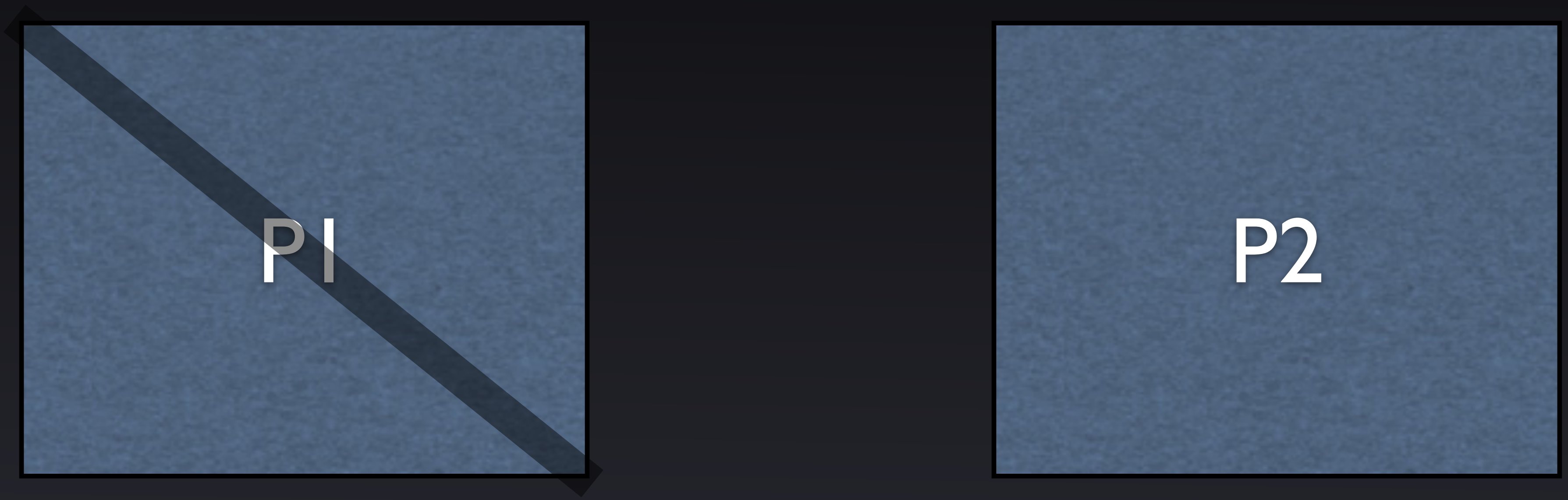
~~We~~ I had **one job**.

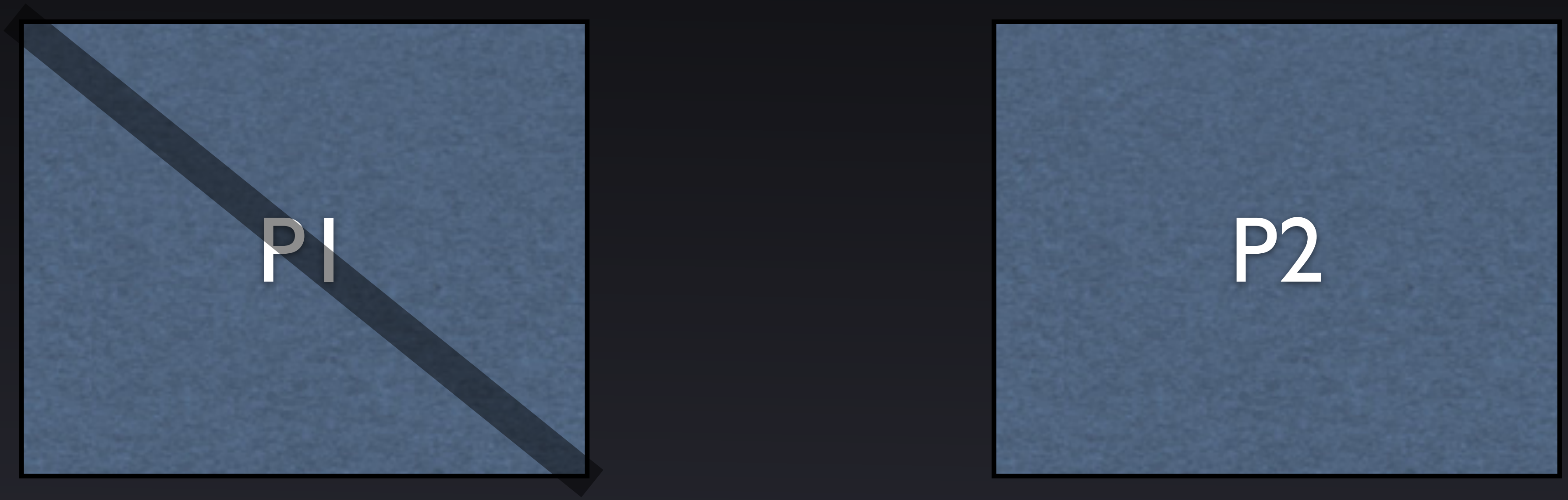
- Migrate a production database server…
- … from one Amazon instance to another…
- … with minimum downtime …
- … using streaming replication.

$$\begin{tikzcd}\boxed{\text{PI}} \arrow[r] & \boxed{\text{SI}}\end{tikzcd}$$

~~PI~~

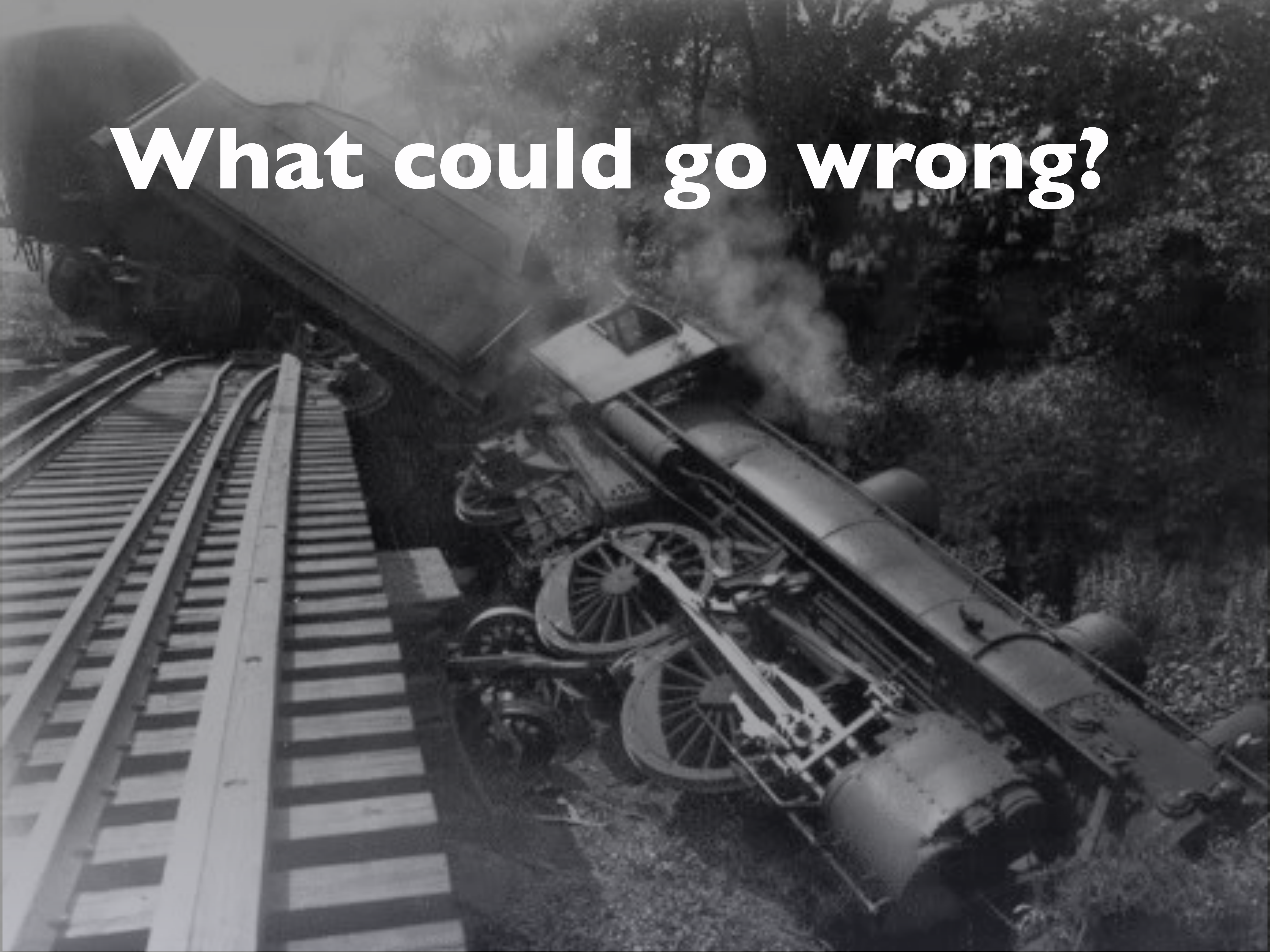
SI


~~P1~~
P2



Profit!

What could go wrong?



36 hours later...

“Huh. That’s weird.”

Oh, no.

- Rows in P1 were missing in P2.
- Deleted rows in P1 were still on P2.
- Rows in P1 were duplicated in P2.
  - … in violation of primary key constraints.
  - But no one told the indexes.

It was surreal.

- Multiple versions of the same row, before and after modification by a committed transaction.
- Newly-created rows were not pushed over onto the secondary.

Oh, we found it!

- The tables had a last_modified timestamp…
- … and the bad rows clustered right around the cutover time.
- … and queries were running!
- That must be it! Active queries at the cutover time!

Spoiler Alert!

This makes no sense.

No problem!

- Couldn’t roll back to P1, but we could fix the database.
- Did a pg_dump / pg_restore.
- Patched up everything very, very tediously.
- Brought it back up.

We’re so smart it hurts.

- Problem solved!
- Brought up a new secondary…
  - … after making sure there were no queries running.
- Everything looks great.

SENATVS
POPVLVSQVEROMANVS
DIVOTITODIVIVESPASIANIF
VESPASIANOAVGVSTO

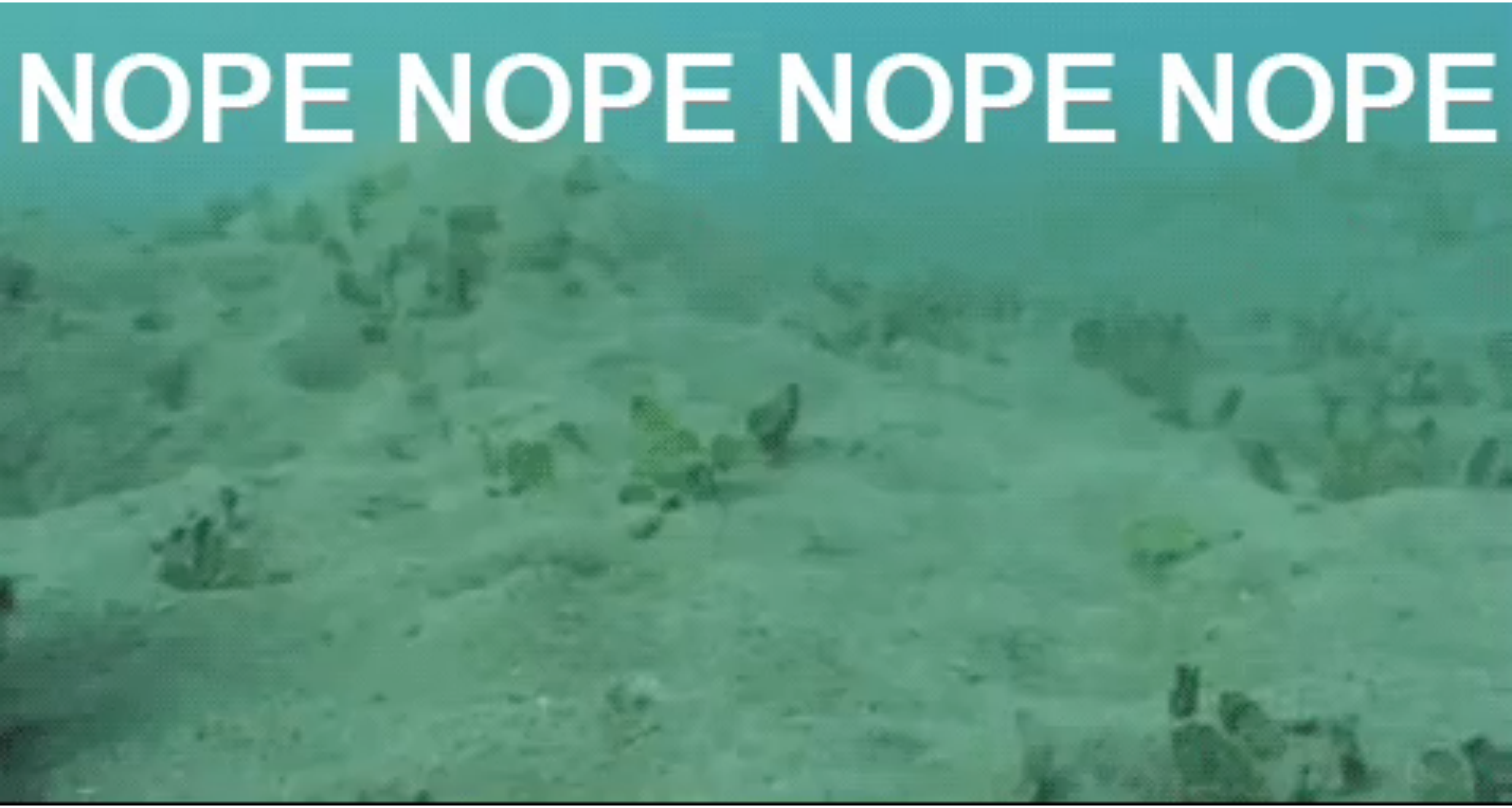
Declare Victory!

6 hours later…

“Hey, Christophe…”

Oh, no, not again.

- The problem reoccurred on the new secondary.
- Same problem.
- Same symptoms.
- Even though the obvious clear no-question must-be-it cause was gone.


NOPE NOPE NOPE NOPE

So, what happened?

- It was, in fact, a PostgreSQL bug.
- Downgraded to an earlier minor release.
- Waited until the next minor release, upgraded.

~~We~~ I did everything wrong.

- Didn’t keep the parts.
- Didn’t work up the stack.
- Didn’t methodically track down the error.
- Ruled out a PostgreSQL bug prematurely.

When disaster strikes.

Bad things are happening.

- CPU is pegged.
- Out of disk space.
- Data corruption.
- Lock pileups.

The First Step.

STOP
ARRÊT

Crisis

=

Problem + Panic

First, do no harm.

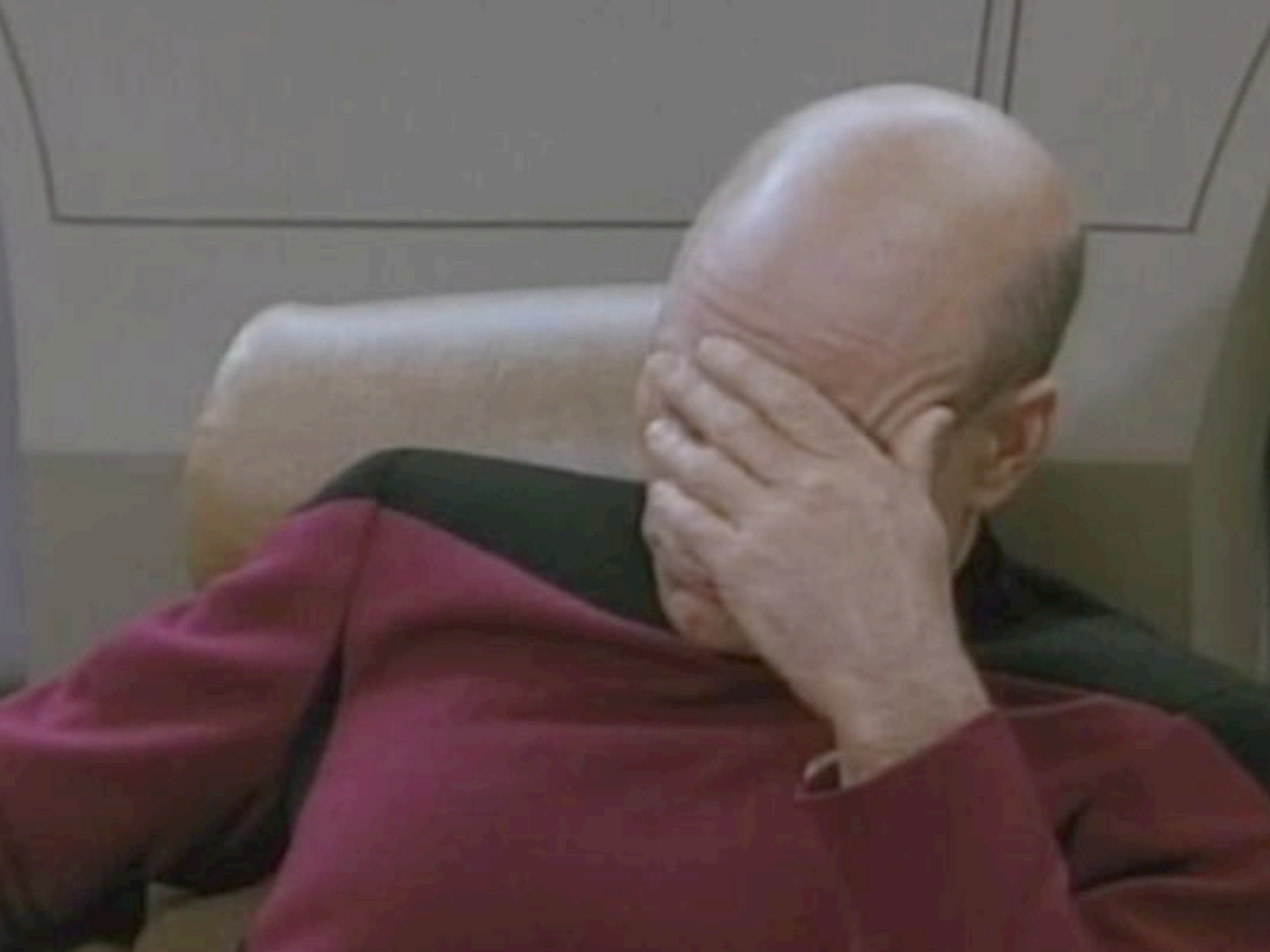
- If you're down, you're down. Take a deep breath, and move cautiously.
  - Minimize communication channels.
- Don't delete anything unless you **know** that is a solution to the problem.
  - Like, you're out of disk and it's full of **text** logs.

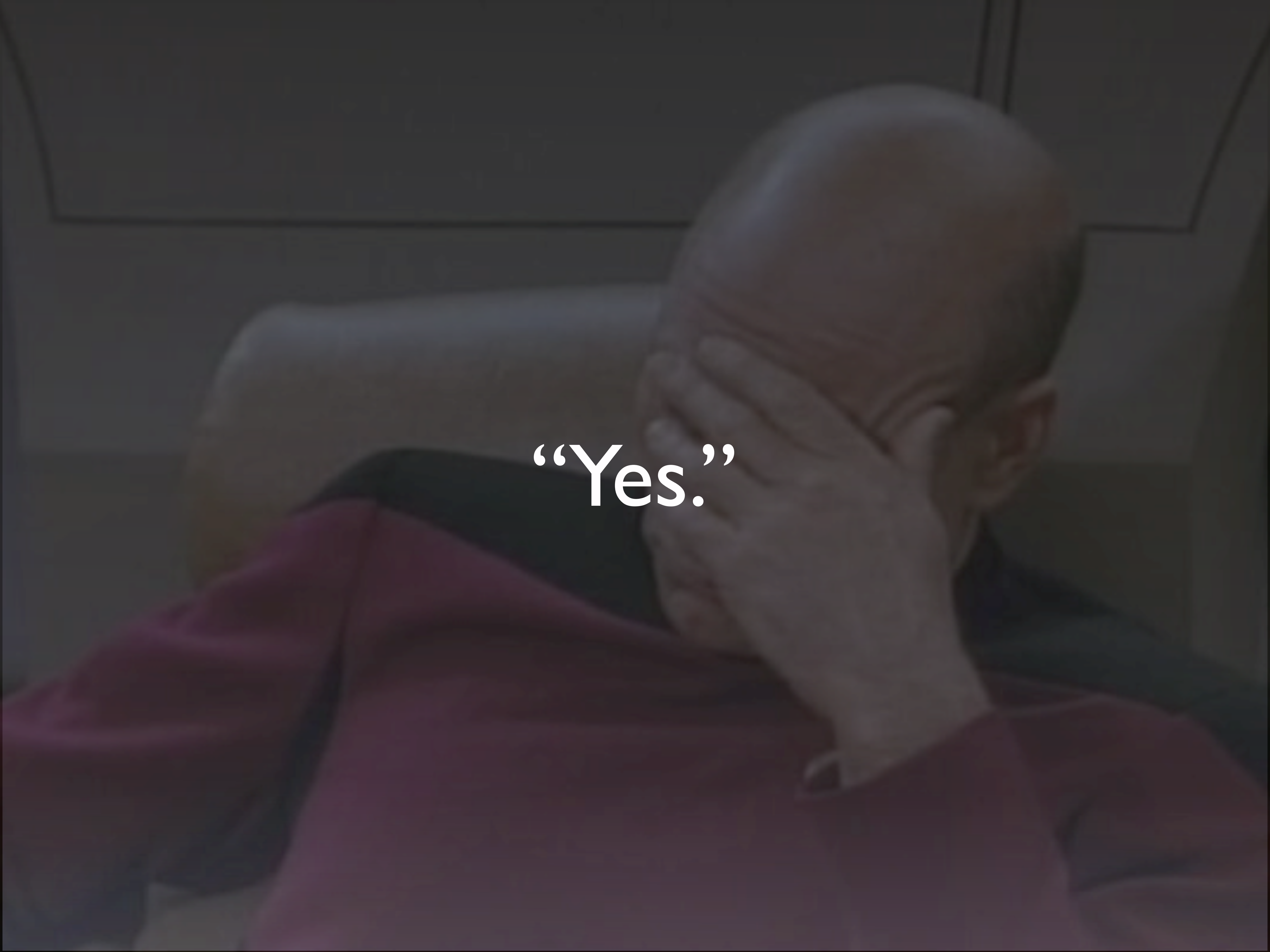
text logs.

- “The disk filled up, so we deleted the log files. Now, PostgreSQL won’t start.”
  - “What did you delete?”
- “Everything in the log directory.”
  - “Um, which log directory?”

“pg_xlog”

“Is that bad?”



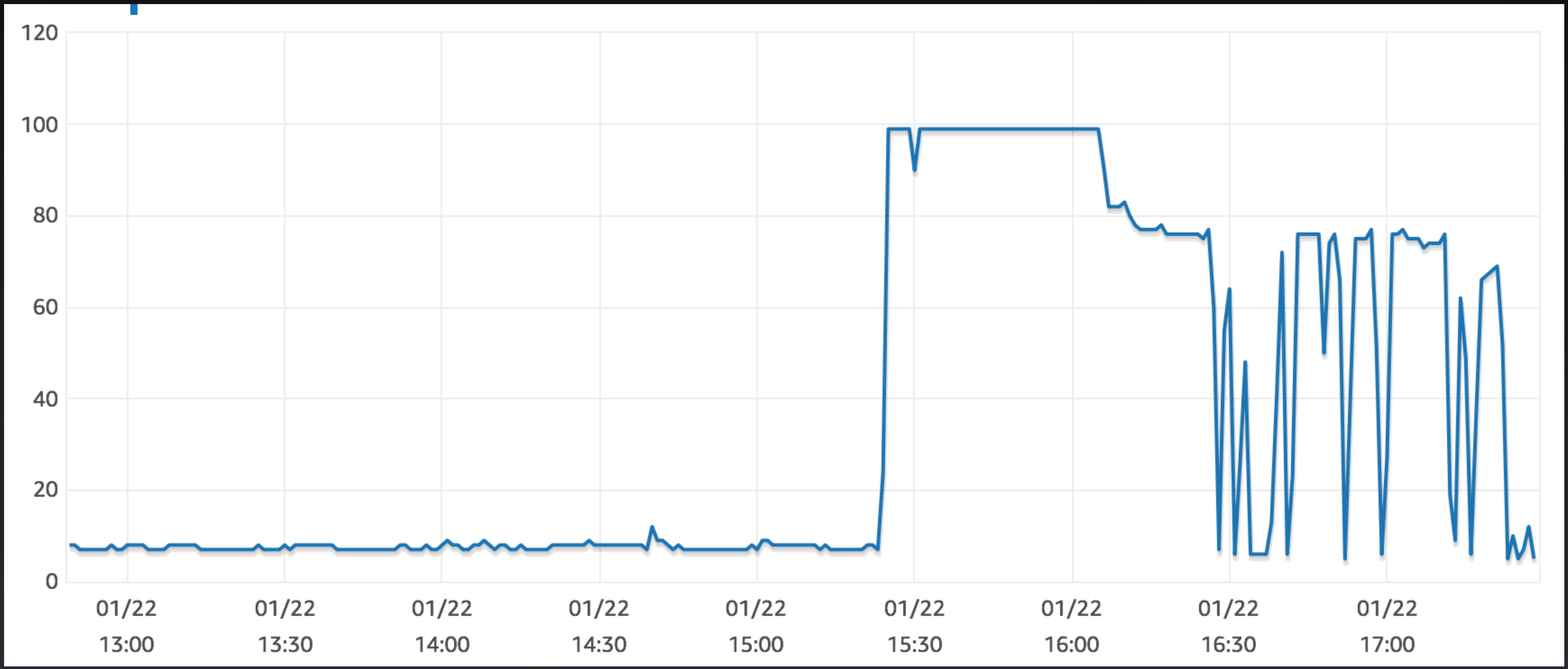


"Yes."

Some Bad Situations.

"All good servers are alike, but each bad server is bad in its own way."
— Anna Katerina,
Database Administrator

Situation: CPU Pegged.



Possible reason: Connection Storm

- Starting a new connection in PostgreSQL requires forking a new process.
- A large number of these at the same time can be very high CPU.
- Especially bad if connections are opening and closing fast.

Resolution:
Don't Do That.

- Add a pooler into the stack.
- Fix thundering herd problems on mass cache invalidation, app server restart, etc.
- Fix error conditions that can force a connection close.

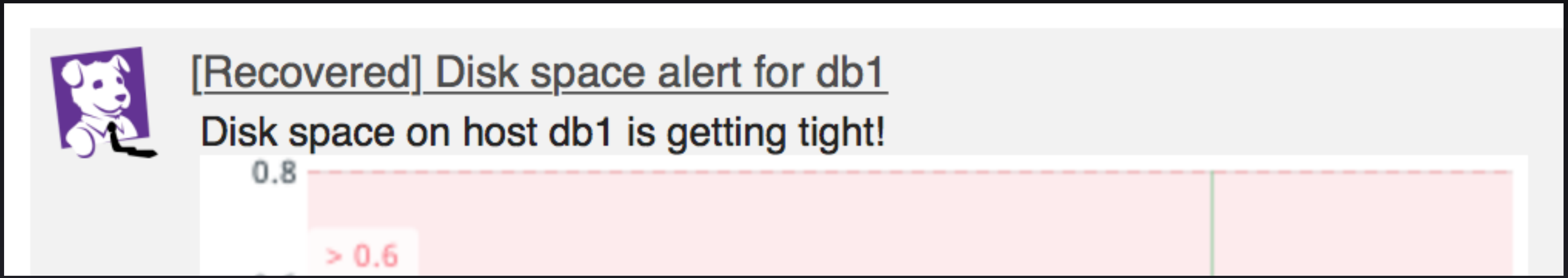
Possible reason: Bad Query Plans

- Previously-benign queries suddenly having bad plans.
- BitmapIndex/Heap scan in place of Index scan, etc.
- Often caused by tables and indexes getting badly bloated.

Resolution:
Fix Bloat Issues.

- Make sure autovacuum is keeping up.
- Do manual VACUUM operations if required.
- Rebuild badly bloated indexes.
- Use pg_repack to repack badly bloated tables.

Situation:
Out of Disk Space.



[Recovered] Disk space alert for db1

Disk space on host db1 is getting tight!

Possible reason: WAL Pileup.

- Write-ahead log segments not being recycled by PostgreSQL.
- archive_command failing.
- Logical replication slot not keeping up.

Resolution:
Fix Underlying Issue.

- Fix archive_command.
- Drop the bad replication slot.
- Can require a CHECKPOINT or two to recycle the log segments… be patient!

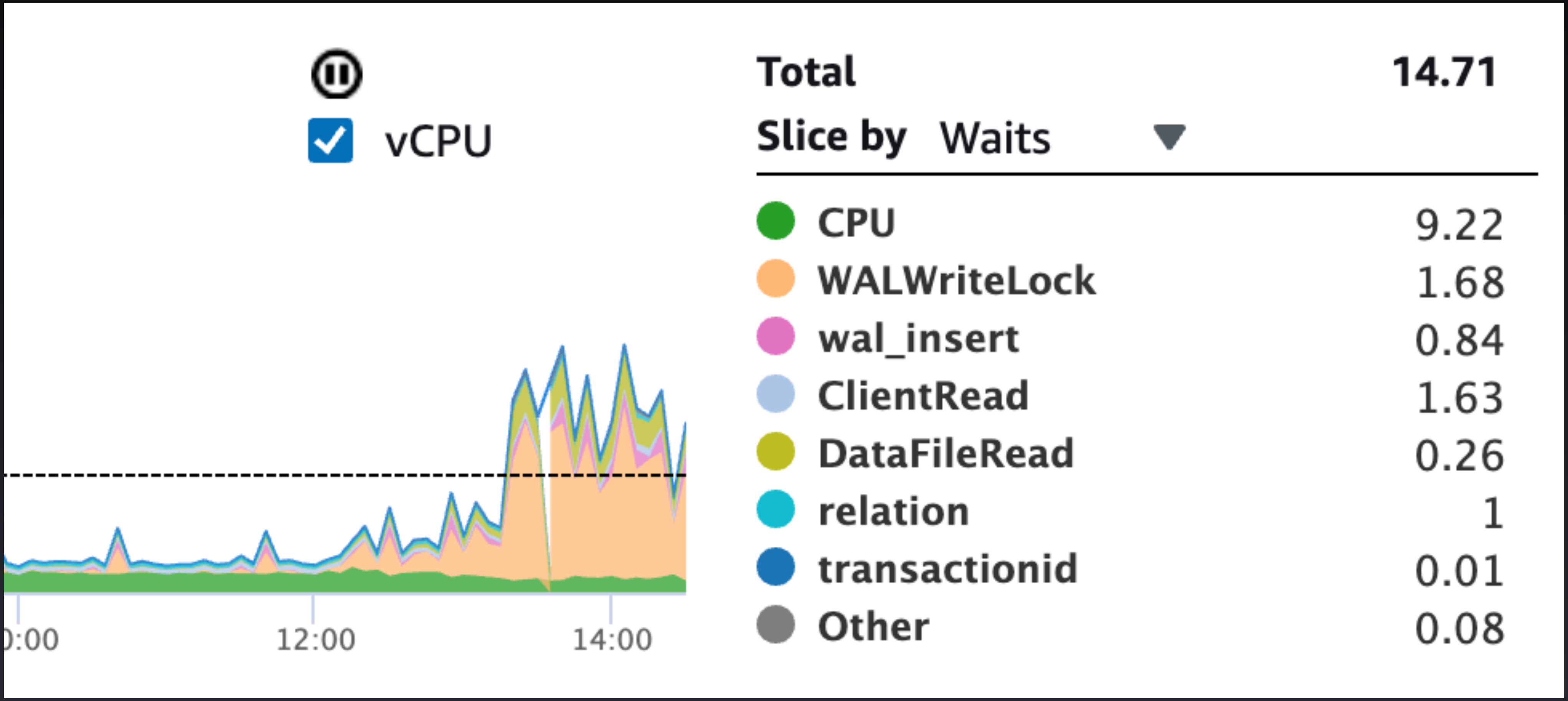
Possible reason: Text Log Bloat.

- Text logs can be very big if configured improperly.
- Some systems require that they be written to the same volume as the database.

Resolution: Reduce Chattiness.

- Decrease per-query log volume.
- Move to a separate volume.
- Move to a remote collector.
- Fortunately, safe to delete to free space.

Situation: Lock Pileups.



Total	14.71
Slice by Waits ▼	
● CPU	9.22
● WALWriteLock	1.68
● wal_insert	0.84
● ClientRead	1.63
● DataFileRead	0.26
● relation	1
● transactionid	0.01
● Other	0.08

Possible reason: Migrations.

- Schema changes generally require exclusive locks.
- PostgreSQL is first-in, first-out in lock grants.
- Schema change waits, other sessions pile up behind it.

Resolution: Good Migration Technique.

- Minimize migrations that have to do full-table scans or writes.
- Do changes during low-load periods.
- In extreme cases, take a maintenance window.

Possible reason: Long-Running Transactions.

- All locks are held to the end of the transaction that took them.
- Easy for locks to build up.
- Long-running transactions can block other transactions.
- Idle-in-transaction sessions are particularly problematic.

Resolution: Reduce Transaction Length.

- Fix idle-in-transaction sessions.
- Only use prepared transactions if you must.
- Break up very large operations to reduce the time locks are held.

Possible reason: LWLock pileup.

- Internal lightweight locks that protect various PostgreSQL data structures.
- New tools have provided more visibility into waiting on them.
- No one technique for all of them.

WALWriteLock

- Held while WAL segments are being written to disk.
- Sometimes, just too much I/O.
- Increase wal_buffers.
- Turn off synchronous_commit.

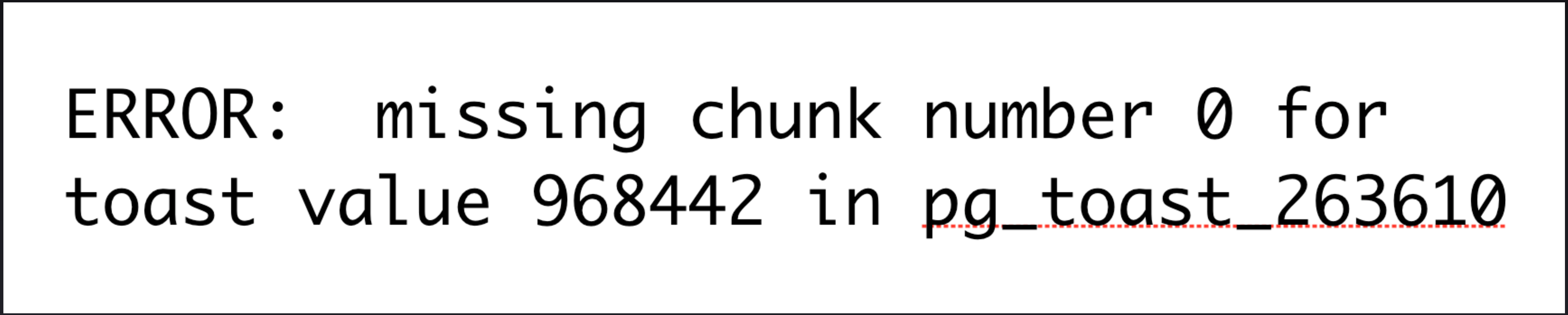
ProcArrayLock

- Often a sign of too many concurrent sessions COMMTing.
- Reduce concurrency with a pooler.
- Concurrency failures are often non-linear.

buffer_content

- Waiting to map a shared buffer to the underlying disk page.
- Often a sign that the working set is much larger than shared_buffers.
- Increase shared_buffers (although be judicious; 30%+ of RAM is usually not a benefit).

Situation: Data Corruption.



```
ERROR:  missing chunk number 0 for
toast value 968442 in pg_toast_263610
```

Step 1: Restore last-good backup

Step 2:
Receive the praise of a grateful nation.

Time for coffee!

Oh.

- You don’t have a known-good backup?
- That’s a shame.
- Sadly, even good backups can…
  - have hidden long-term corruption.
  - be too old.
  - *(whisper it)* be hit by PostgreSQL bugs.

Save all the parts!

- Stop PostgreSQL.
- Do a full file-system level backup.
- Keep that backup safe.
- Make changes methodically, and document each step.

Index Corruption.

- The most common kind of corruption.
- Drop the index in a transaction, and confirm that solves the problem.
- If so, rebuild the index.
- If not, it’s probably not index corruption.

Take a pg_dump.

- pg_dump reads every row, and…
- … creates a logically-good snapshot.
- Restore that into a clean database.

Bad Data Page.

- Checksum failures, complaints about bad headers, etc.
- Can you do a pg_dump of the table?
- zero_damaged_pages = on.

Really Bad Data Pages.

- Can you SELECT around them?
- Do a COPY out of the good data, drop table, COPY back in.
  - Or do a CREATE TABLE from the SELECT, rename appropriately.
- DELETE just the bad rows by ctid, if you can isolate them.

Finding bad data pages.

- Iterate through rows in PL/pgSQL…
- … with an exception block around the SELECT.
- Catch and log any rows that throw an exception.
- Very helpful for finding TOAST corruption.

Expecting the Unexpected.

Planning for disaster.

- If you run a PostgreSQL installation of any size, these things will happen to you.
- Sooner or later.
- The best way to avoid turning a problem into a crisis is to be prepared for it.

Test. Your. Backups.

- A backup that is not tested is not a backup.
- Give them to developers.
- Use them for analytics.
- But **make sure** that the restore steps are automated and foolproof…
    - … because you probably will have to do it on no sleep.

Monitor.

- Alert well before disk space exhaustion.
- Summarize errors in logs.
- Track lock waits.
- Track temporary file creation.

The right kind of ~~leaves~~ backups.

- Do PITR backups.
- Don’t roll your own.
  - pgBackRest
  - barman.
- Corruption can lurk for an extended period before it’s found.

PostgreSQL hygiene, 1.

- Make sure autovacuum is happening.
  - Never disable it!
- Monitor query execution time.
  - Note queries that are starting to slow down.

PostgreSQL hygiene, 2.

- fsync = on
  - Make sure this really happens.
- full_page_writes = on
  - Very few file systems guard against torn pages.
- Don’t kill -9 anything.

Stay up-to-date.

- Deploy minor versions as they roll out.
  - Yes, the bug at the start of the presentation was introduced in a minor upgrade.
  - That's **extremely** uncommon.
- Plan an upgrade strategy so you are not caught by a major version going EOL.

Turn on checksums.

- Flags corruption immediately.
  - Does not fix the damage, though.
- Use it unless your filesystem does checksums.
  - Which it probably doesn’t.

Which host?

- Provisioning a new host can be time-consuming.
  - Even in a cloud environment.
- Can you produce your exact database server's configuration, including packages?
  - Provision using a proper management system (Ansible, etc.)

Test, test, test.

- Have automated test tools that do application-level database scans.
- Tuples get lonely. Visit them once in a while.
  - Don’t wait for a VACUUM FREEZE.
- Make it part of your migration / upgrade strategy.

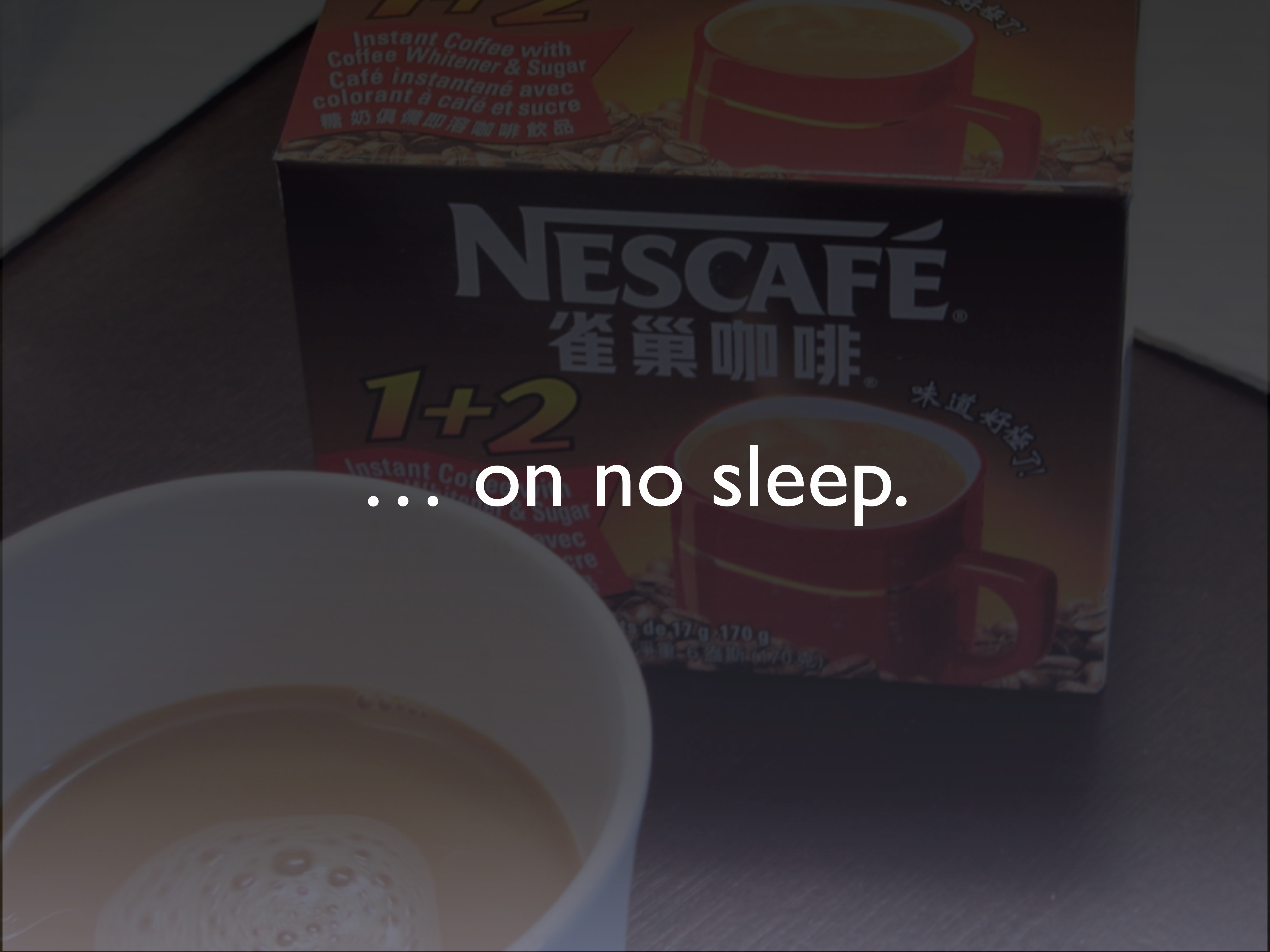
Let’s play a game.

- Your main data center burns to the ground.
- How do you get the database back up?
- How much data have you lost?
- For “data center,” read AWS region.

Write it down.

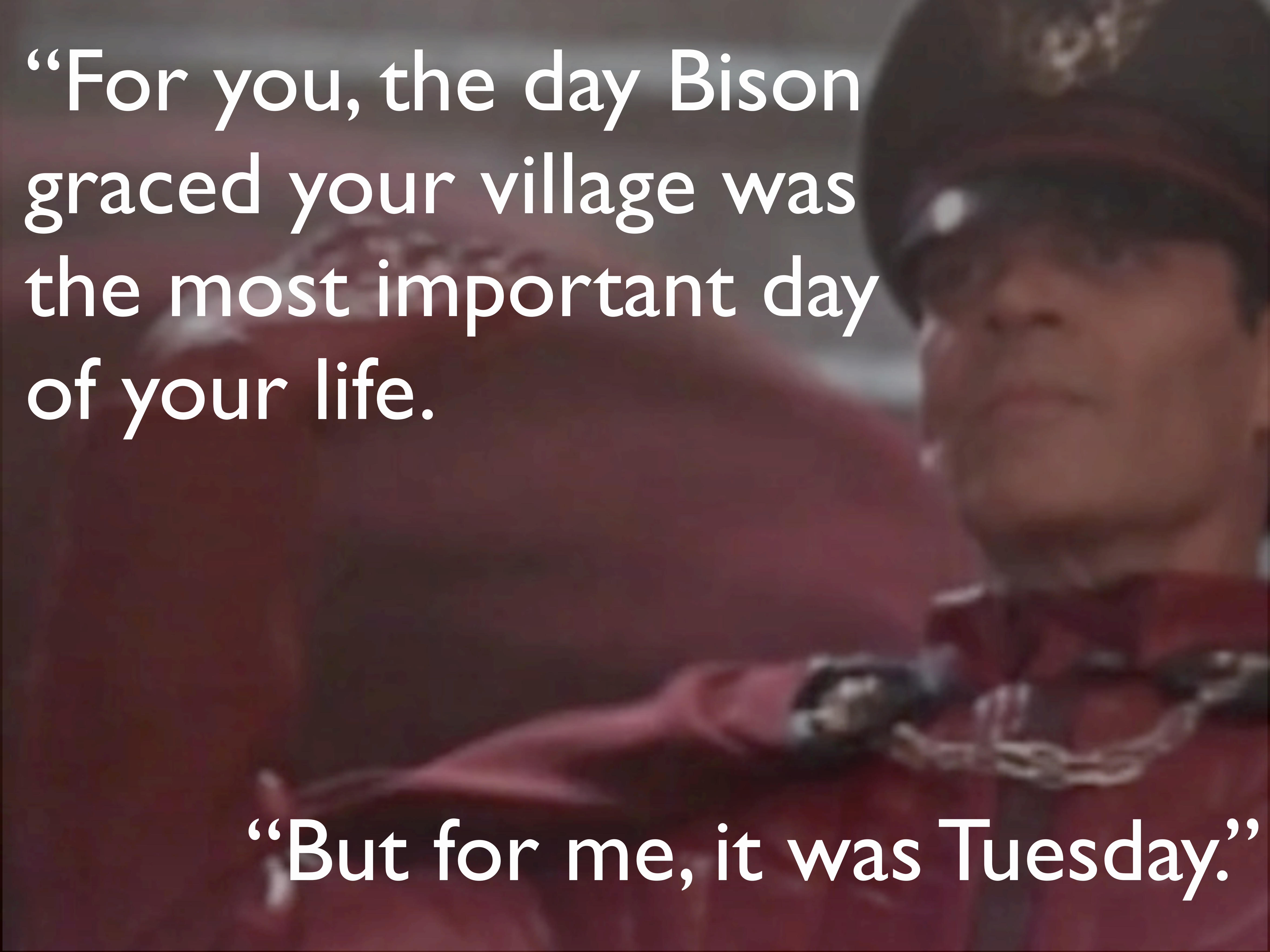
- Have a runbook for these situations.
- You'll often have to go off-script…
    - … but it is great to have a list of things to try, and steps to take.
- Remember, you'll be doing this…

Instant Coffee with Coffee Whitener & Sugar
Café instantané avec colorant à café et sucre
糖奶俱備即溶咖啡飲品
NESCAFÉ®
雀巢咖啡®
1+2
味道好極了!
Instant Coffee with
s de 17 g 170 g


Instant Coffee with
Coffee Whitener & Sugar
Café instantané avec
colorant à café et sucre
糖奶偶備即溶咖啡飲品
NESCAFÉ
雀巢咖啡
1+2
味道好極了!

… on no sleep.

Working with the Community.



“For you, the day Bison graced your village was the most important day of your life.

“But for me, it was Tuesday.”

The bug you found is the worst thing in your world.

- But if it was the worst thing in the developer's world, they'd have pushed a patch already.
- No one is paid just to fix PostgreSQL bugs.
- Everyone who can hack on PostgreSQL internals is very, very busy.

Be thorough…

- Develop a test case, if you can.
- Document everything, even if you think it is not important.
- If the data is sensitive, come up with an anonymization plan.

File a bug.

- pgsql-bugs@postgresql.org
- http://www.postgresql.org/support/submitbug/
- Read the guidelines!

If the bug is critical…

- … critical defined as data corruption or repeatable server failure…
- … consider bringing it up on -hackers.
- Remember, everyone is busy with their own crises.

Crashing / freezing bugs.

- Install the -dbg packages.
- If you are getting core dumps, get stack traces out of them.
- Use strace to find out where things are hung up.

Be persistent, but polite.

- Monitor any threads you start.
- Answer questions promptly and thoroughly.
- Don't badger the developers! They don't work for you!
  - And even if they do, be nice. :-)
- Well-documented and repeatable critical bugs get fixed pretty fast.

Thank you!

Christophe Pettus
PostgreSQL Experts, Inc.

pgexperts.com

thebuild.com

christophe.pettus@pgexperts.com

Twitter @xof