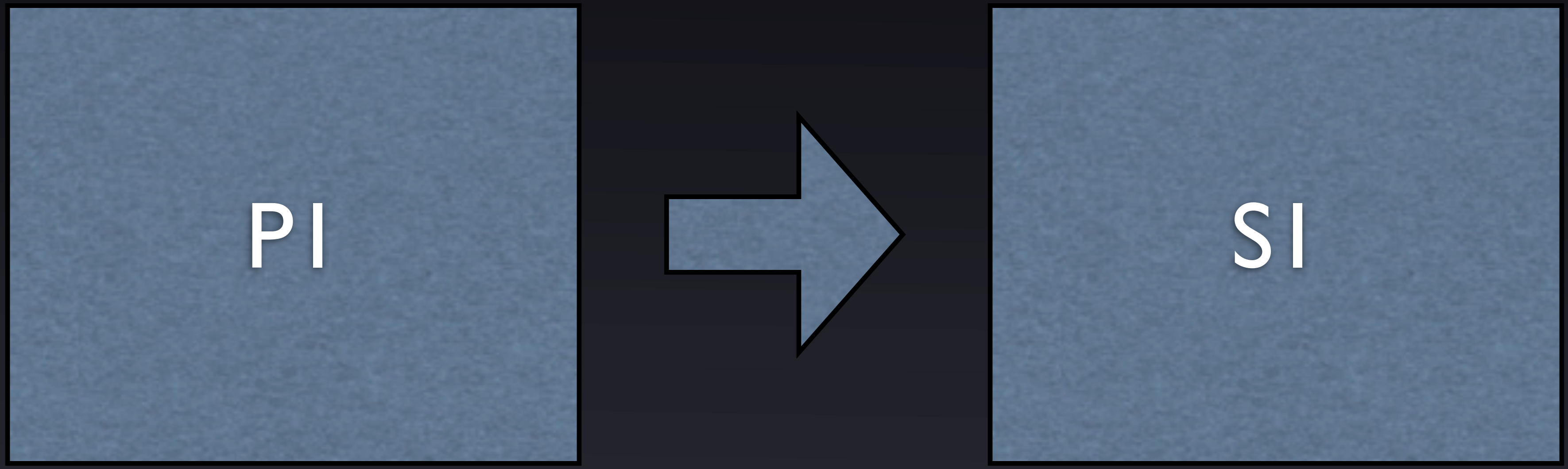
The Worst Day of Your Life

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.
- PostgreSQL 9.3.1
  - at that time, the latest version.



PI

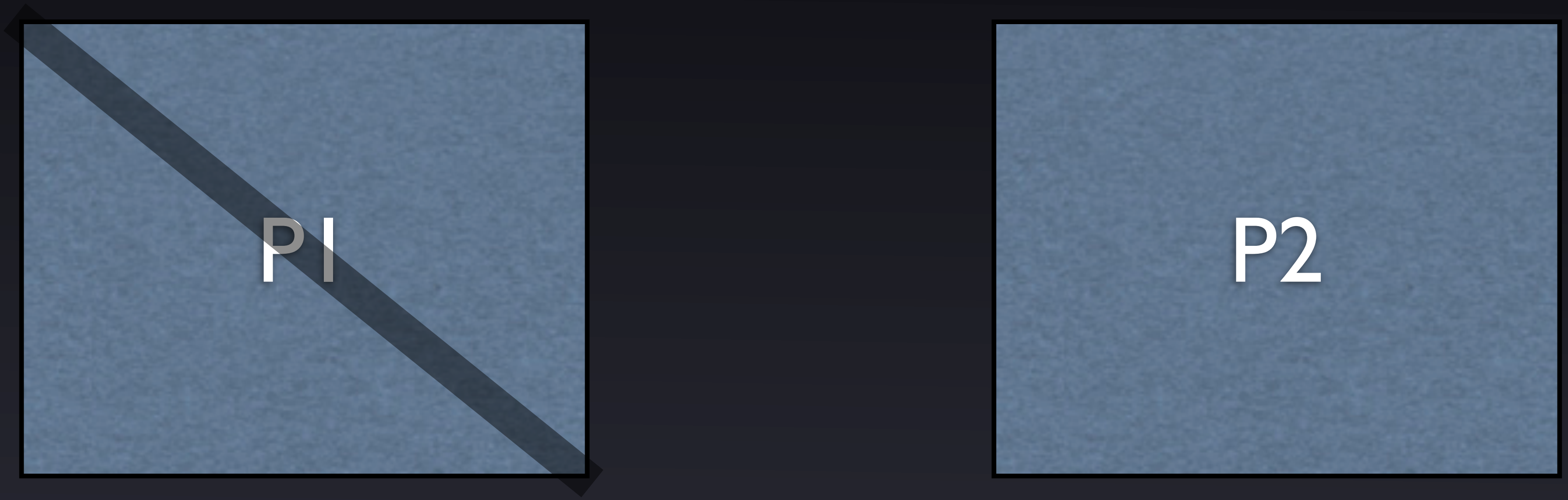
SI

~~PI~~

SI

~~P1~~

P2

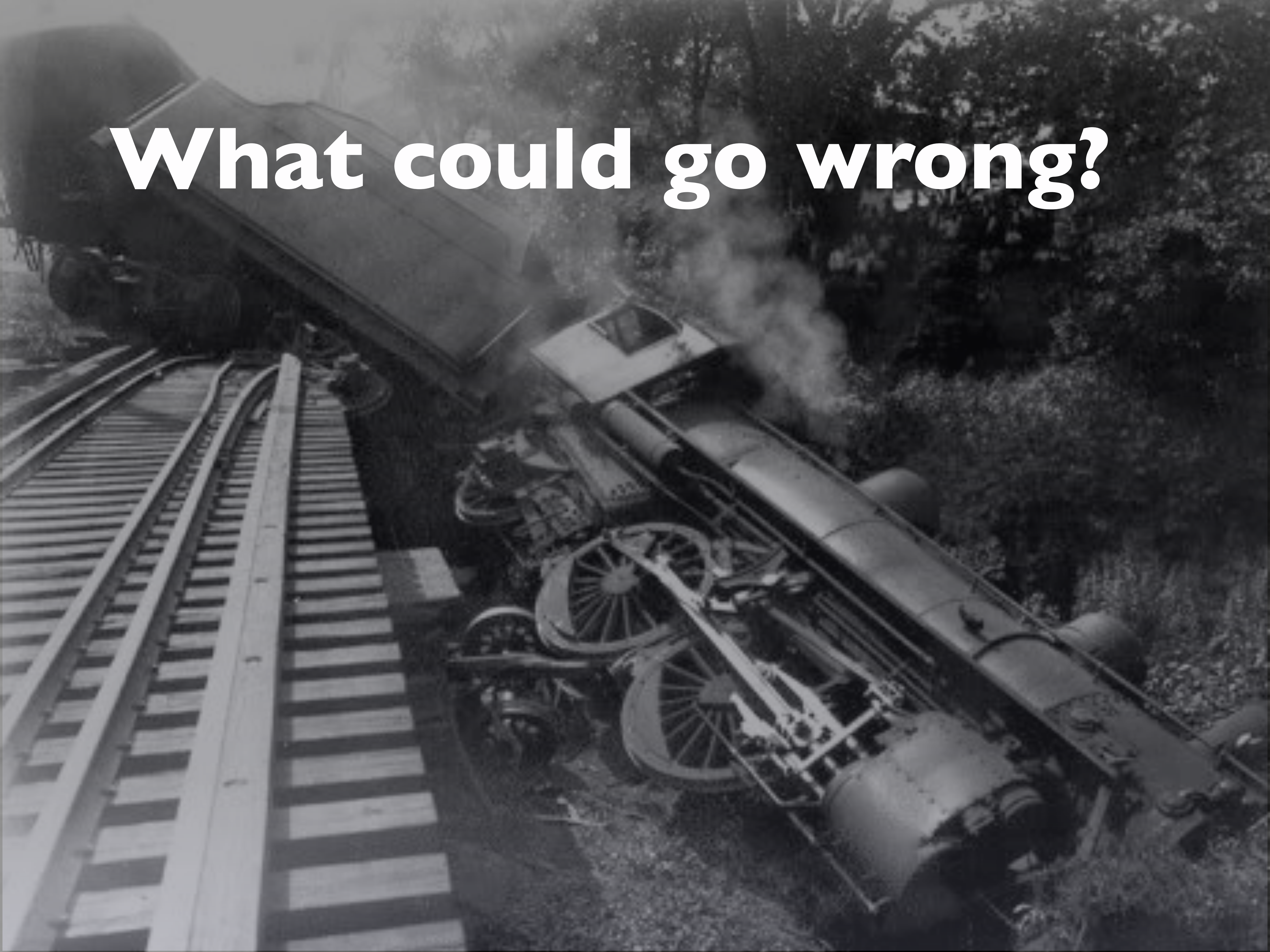


~~P1~~

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
POPVLVSQVE ROMANVS
DIVO TITO DIVI VESPASIANI F
VESPASIANO AVGVSTO

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.

So, what happened?

- It was, in fact, a PostgreSQL bug in 9.3.1 (and 9.2.5, and 9.1.10, and 9.0.14).
- Downgraded to 9.3.0 until 9.3.2 came out.
- Applied the 9.3.2 upgrade without incident.

~~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.

By the way…

- If you are running 9.3.1 or 9.2.5 or 9.1.10 or 9.0.14…
- … and using streaming replication …
- … upgrade *right now.* This instant. I'll wait.
- OK, everyone back?

When disaster strikes.

Bad things are happening.

- PostgreSQL is crashing repeatedly.
- Queries returning bad results.
- Scary-looking error messages in the log.
- Backends are running for extended periods without an obvious reason.

The Stages of a Crisis.

- Denial
- Anger
- Bargaining
- Depression
- Actually fixing the problem already.

Denial.

- It must be something unrelated to PostgreSQL.
- PostgreSQL doesn’t have bugs.
- Oh, you were running queries while the cutover happened?

Anger.

- YOU SHOULDN'T HAVE DONE THAT!
- WHY DIDN'T YOU TELL ME!
- YOU SAID THE APPLICATION WAS QUIESCENT!

Bargaining.

- “OK, we’ll just repair the database from the missing rows.”
- “A pg_dump/pg_restore will fix everything.”
- “It was probably a transient EBS failure. You know EBS. EBS totally sucks. It’s all EBS’ fault.”

Depression.

- We.
- are.
- all.
- going.
- to.
- die.

Fix the problem.

- Best to skip straight to this stage.
- Move slowly.
- Keep good notes.
- Don’t panic.

The First Step.

STOP
ARRÊT

Crisis

=

Problem + Panic

First, do no harm.

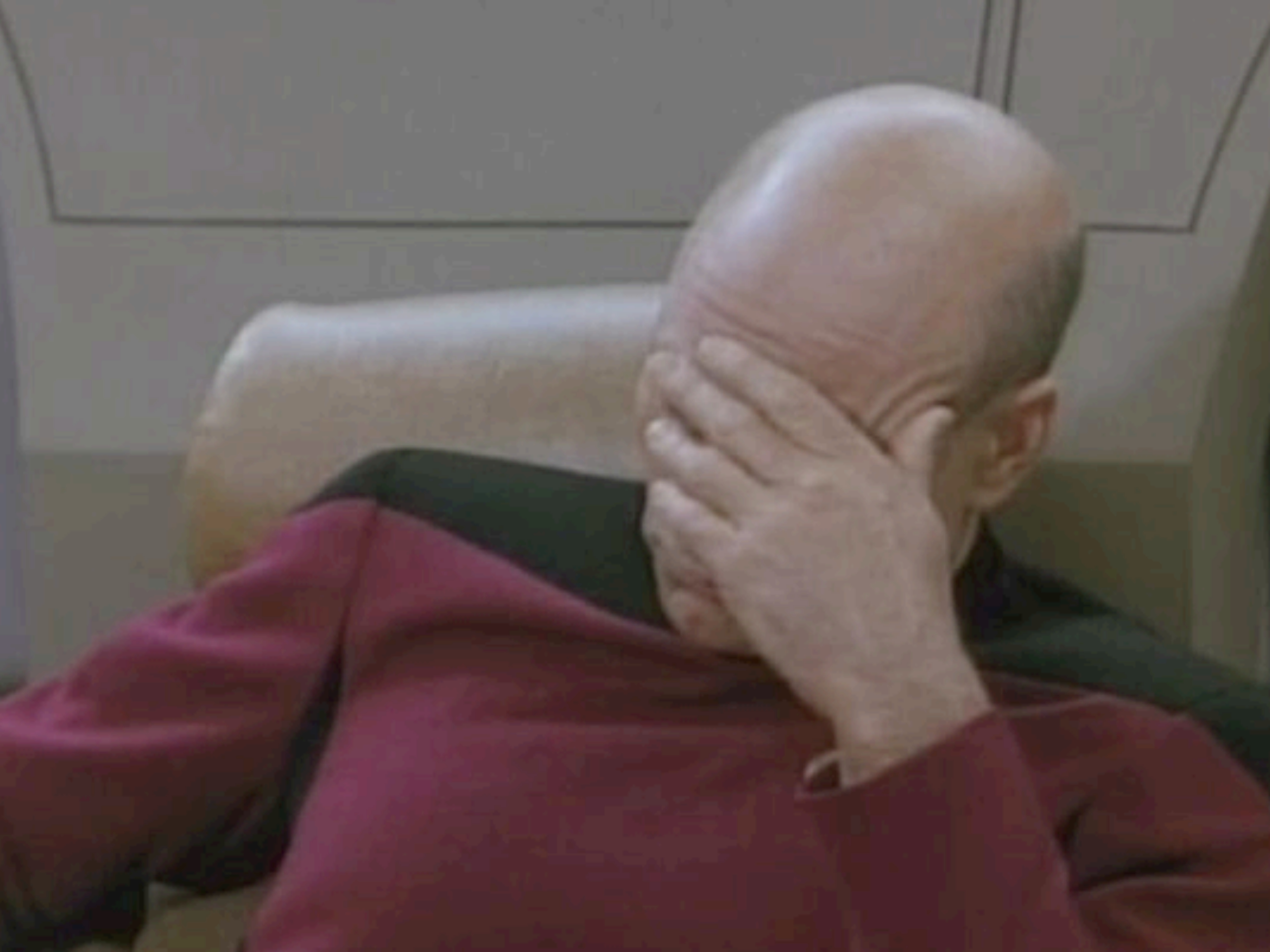
- If you’re down, you’re down. Take a deep breath, and move cautiously.
  - Minimize communication channels.
- Don’t delete anything unless you know that is a solution to the problem.
  - Like, you’re out of disk and it’s full of text logs.

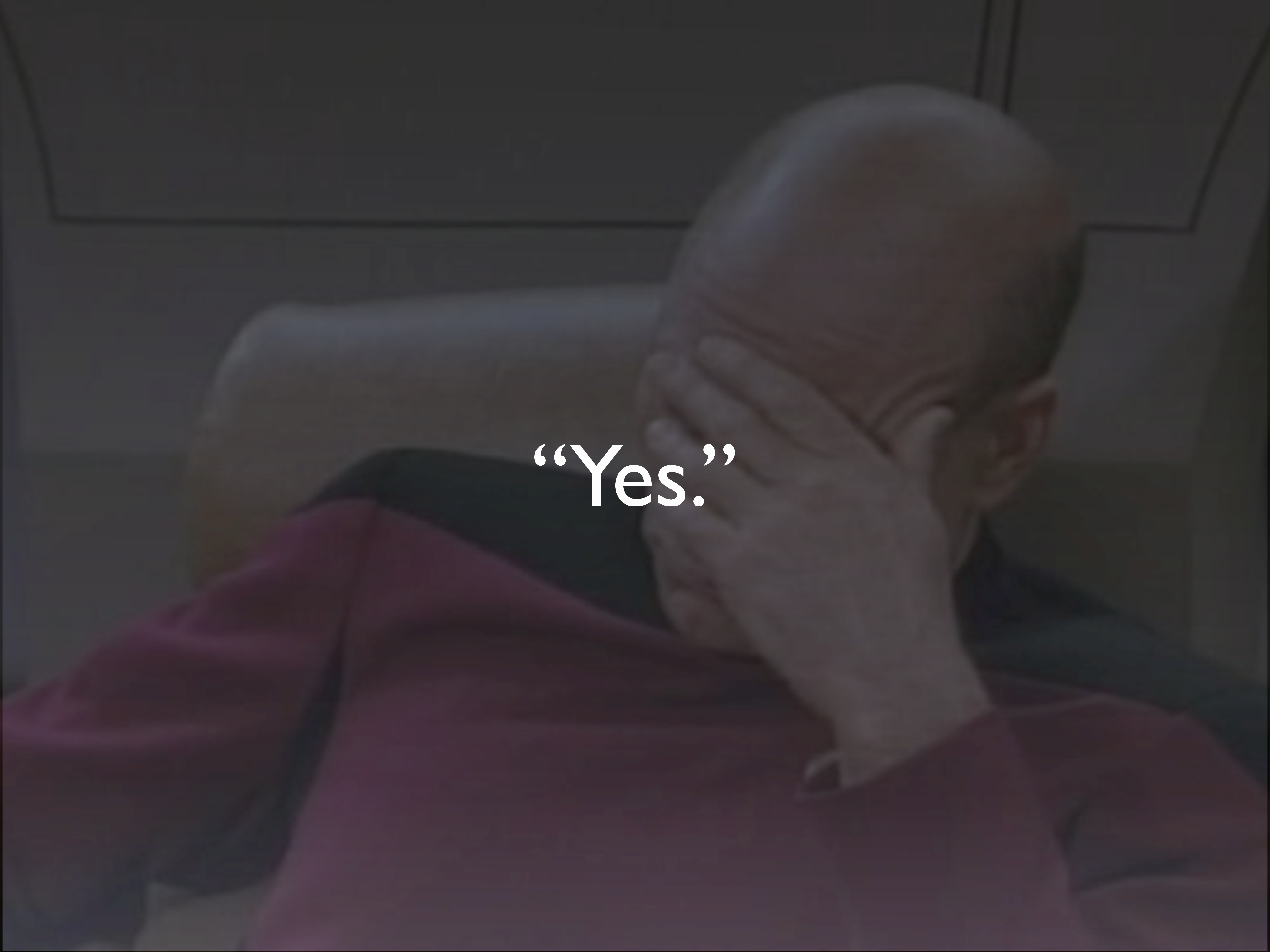
For example…

- “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.”

Keep the parts.

- If you possibly can, make a copy of the database before touching anything.
- If you can’t, document meticulously what you change.

Some crucial points.

- There are tens if not hundreds of thousands of PostgreSQL installations.
- PostgreSQL works very, very well.
- PostgreSQL does have bugs, but…
  - … rule out everything else first.

Work up the stack.

- Are there errors in demsg indicating a hardware or OS problem?
  - Is the OOM killer terminating backends?
- Disk I/O errors?
- Can you cp -R the data directory to /dev/null?

Frequent problems.

- Disk I/O subsystem not honoring fsync.
  - SAN boxes are notorious for this.
- Memory corruption problems.
  - RAM errors are remarkably common.

Strange little issues.

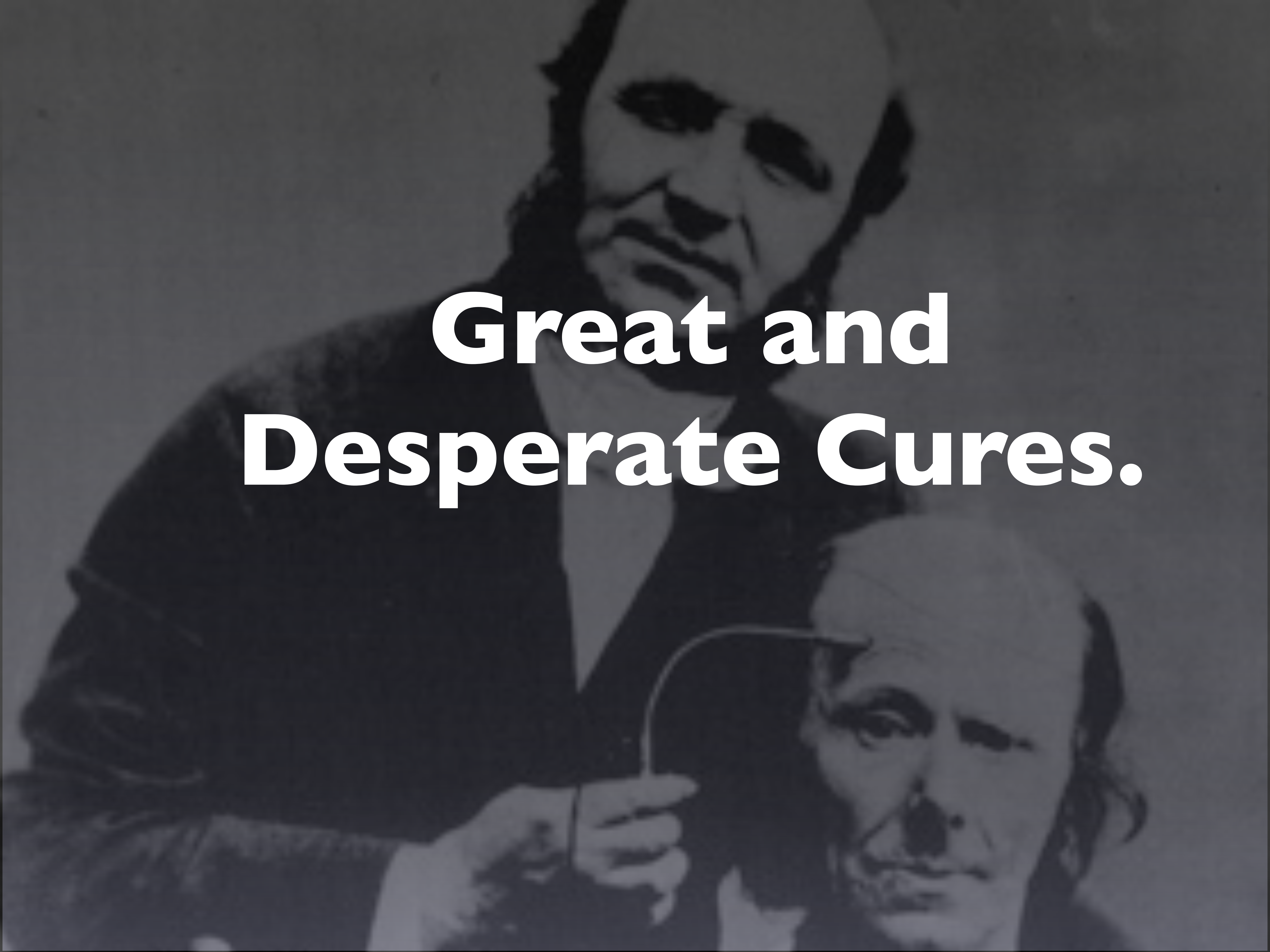
- Remember to let pg_start_backup() complete before taking a snapshot.
- Use rsync and not scp to move WAL files around.
- Keep the WAL files, not just the snapshot backup.

Dealing with crashes.

- Eliminate system-level causes.
- Isolate the crashing behavior (what table? what query?).
- Other processes on the same machine showing unusual behavior?

But what if…

- … you don’t have a clean backup?
- … you need to get the system patched and back up?
- … you can avoid repeating the problem?
- … you have nerves of steel?



Great and Desperate Cures.

Before you push that button.

- It's always better to roll back to a known-good system.
- These are no substitute for a solid backup and disaster recovery strategy.
- No user-serviceable parts inside.
- Proceed at your own risk.

There are no recipes.

Known unknown knowns.

- All corruption is, by its nature, a one-off situation.
- Be sure to determine the extent of it before continuing.
- Be sure you can step backwards!

REMEMBER.

WORK ON A COPY.

Safe(-ish) stuff.

- Index corruption is probably the most common kind of database issue.
    - Indexes have much more internal structure than the heap.
- Drop indexes that are involved in bad-result queries, or scary error messages.

Finding the data

- All data is located inside base/
- Every relation has a relfilenode
  - Find it in pg_class
- base/<database oid>/<relfilenode>
  - .1, .2, .3 for relations over 1GB.

Heap structure.

- Divided into 8KB blocks.
- Each block has a variable number of tuples on it.
- Every row has a ctid
  - (block, tuple-in-block)
- select ctid from my_table;

Fixing heap damage.

- set zero_damaged_pages = true;
- Automatically zeros a page that PostgreSQL thinks is invalid.
- PostgreSQL interprets an all-zero page as empty.
- Drop indexes first.

Fixing really really bad heap damage.

- Backend crashes when it touches a particular row or set of rows.
- Use dd to zero those pages.
  - Double-check your math.
  - Drop indexes first.

clog problems

- The clog keeps track of the state of “visible” transactions.
- Missing, damaged, accidentally deleted clog files can be recreated as all-zero.
  - This can cause… interesting data situations.
- Be aware of irrational clog values.

clog follies.

- The clog keeps track of transaction state.
- “Repairing” it can cause rolled back transactions to reappear, and other exciting events.
- Be prepared to do further cleanup if you touch the clog.

pg_dump / pg_restore

- Forces database-level consistency.
    - Application-level consistency is another matter.
- Fix serious data corruption, do a dump / restore onto a clean host.

COPY

- If you can’t get a clean dump, consider manually COPYing out tables.
- Can sub-select around corruption.
- Do a schema-only dump to create the new, empty receiving database.

System catalog corruption.

- The heap cannot be correctly read without a valid system catalog.
- You can modify the system catalogs directly to patch isolated errors.
- If the system catalog is deeply corrupted, you may need to scavage data.

Tools

- pageinspect — contrib/ module to inspect low-level page information.
- pg_controldata — View control data for the cluster.
- pg_resetxlog — Reset WAL and control information.

Expecting the Unexpected.

Planning for disaster.

- If you run a PostgreSQL installation of any size, something like this 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.

The right kind of ~~leaves~~ backups.

- Do PITR backups.
- Keep a reasonable number of backups and associated WAL segments.
    - S3 is cheap.
- Corruption can lurk for an extended period before it's found.

PostgreSQL hygiene.

- 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.

- 9.3+ initdb option
- Flags corruption immediately.
  - Does not fix the damage, though.
- Use it unless you have a checksuming file system.
  - Which you probably don’t.

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.

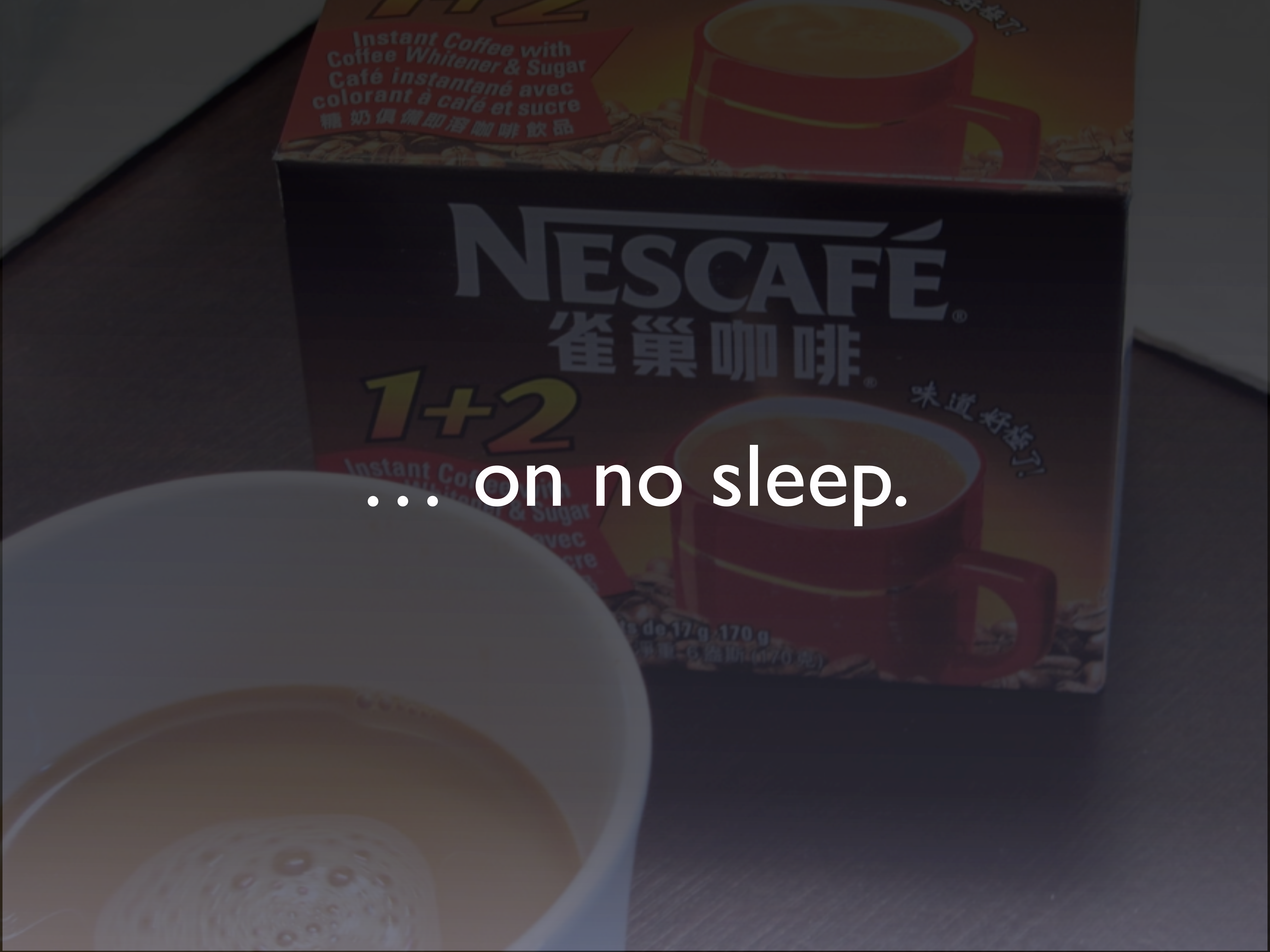
Let’s play a game.

- 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.

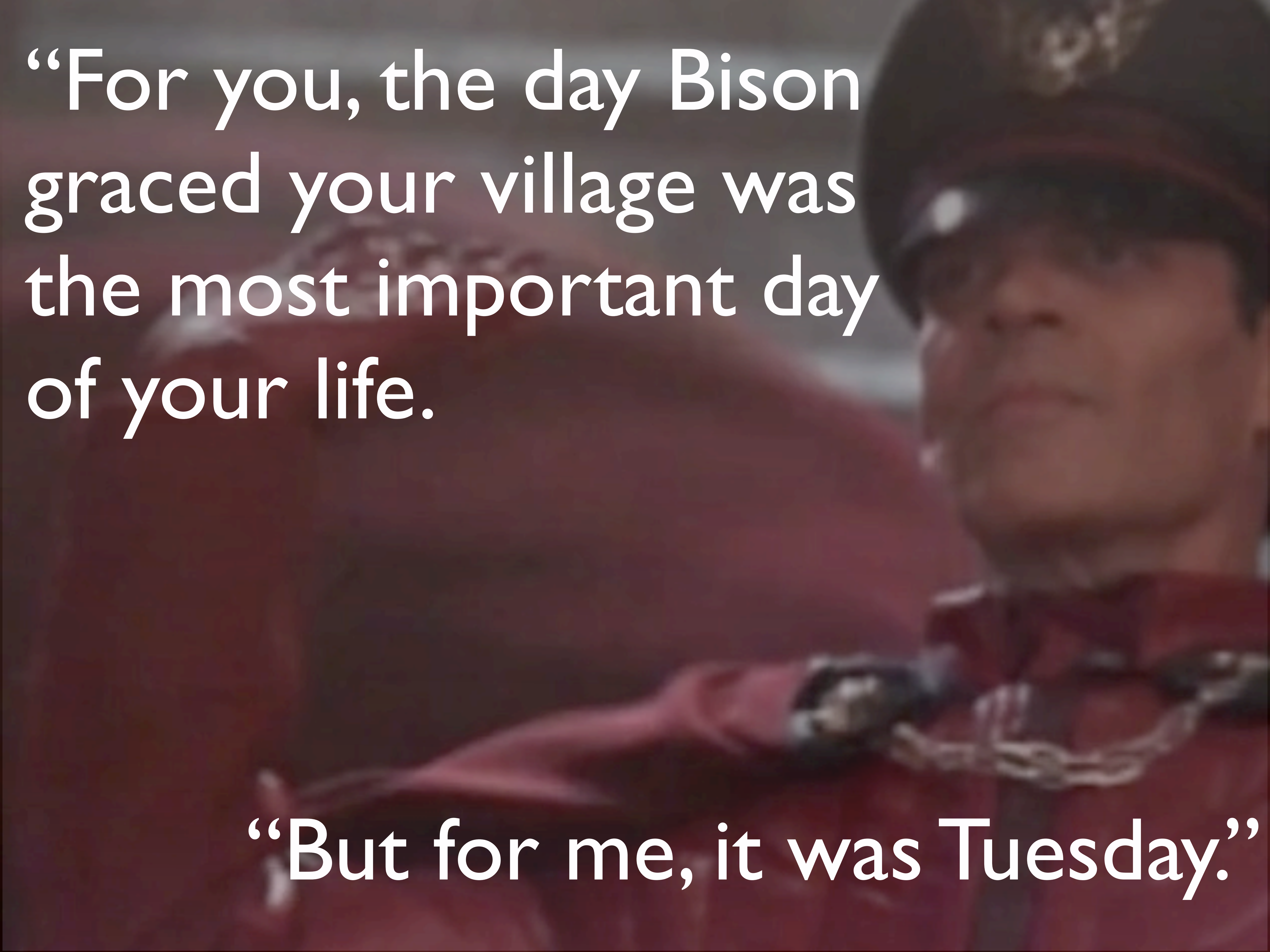
- Have a runbook for these situations.
- You’ll often have to go off-script…
  - … but it is great to have a list of things to try, and steps to take.
- Remember, you’ll be doing this…

1+2
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
Whitener & Sugar
avec
cre
s de 17 g 170 g
淨重 6 盎斯 (170克)


Instant Coffee with
Coffee Whitener & Sugar
Café instantané avec
colorant à café et sucre
精奶偵備即溶咖啡飲品
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!
- Well-documented and repeatable critical bugs get fixed pretty fast.

Consider spending money.

- Hire a company to fix the problem.
  - I might have a recommendation.
- If you think that PostgreSQL consulting is expensive…
- … it's not expensive.

This is expensive.

Thank you!

Christophe Pettus
PostgreSQL Experts, Inc.

pgexperts.com

thebuild.com

christophe.pettus@pgexperts.com

Twitter @xof