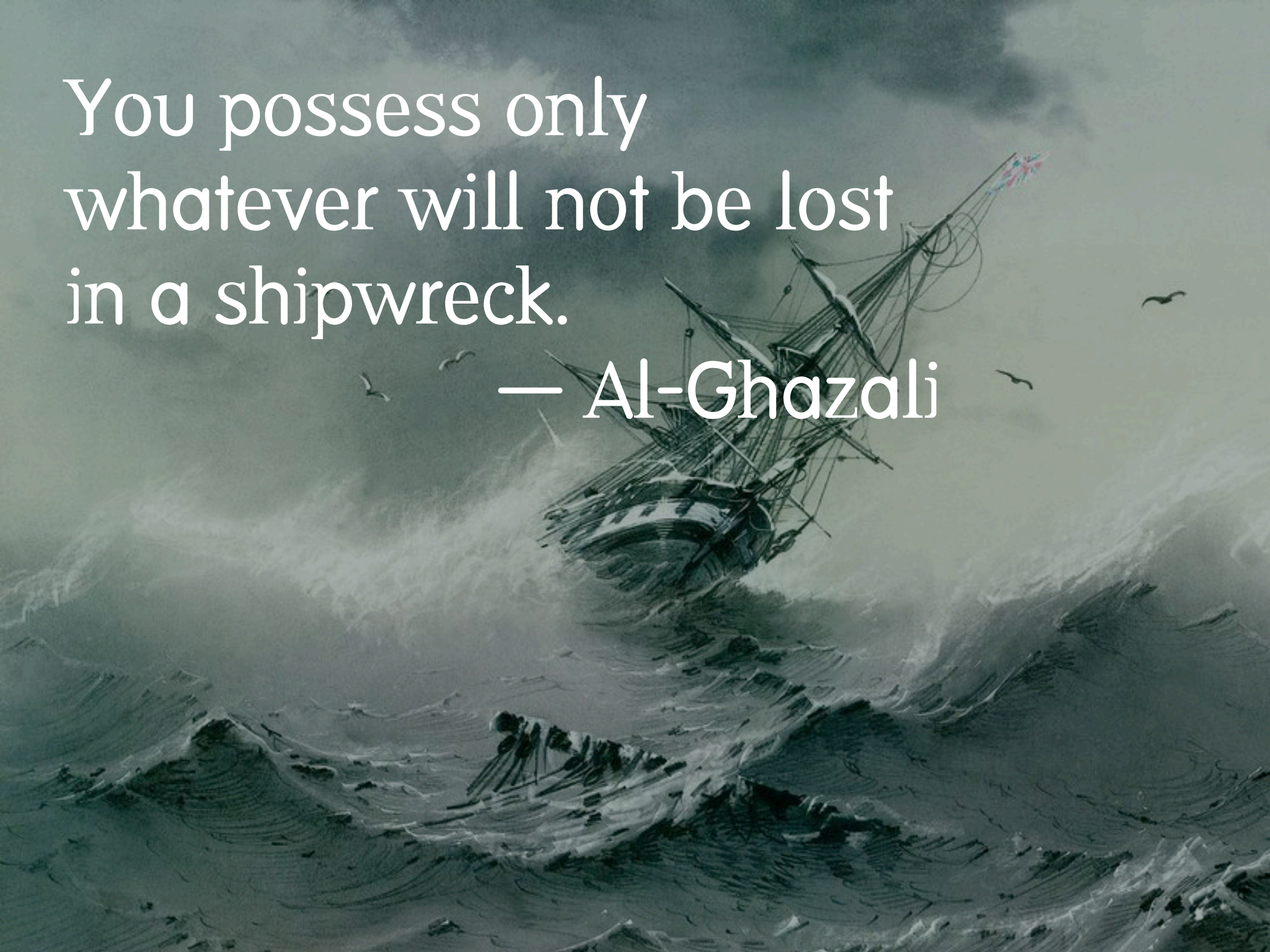
Be Very Afraid

Christophe Pettus
PostgreSQL Experts
Logical Decoding & Backup Conference Europe 2014


You possess only
whatever will not be lost
in a shipwreck.
— Al-Ghazali

Hi.

- Christophe Pettus
- Consultant with PostgreSQL Experts, Inc.
- Working with PostgreSQL since 1997.

Backups are a waste of time and money.

- … right up until you need them.
- … at which point, they become beyond price.
- Consider the “would I have paid this” rule.
  - Right now, with all my data lost, would I pay the backup cost to get it back?

You have to have a recovery solution…

- … unless the data can be reasonably recreated from other sources.
- Most PostgreSQL databases aren't in that situation.
- So, let's talk about how to implement this.

What can fail?

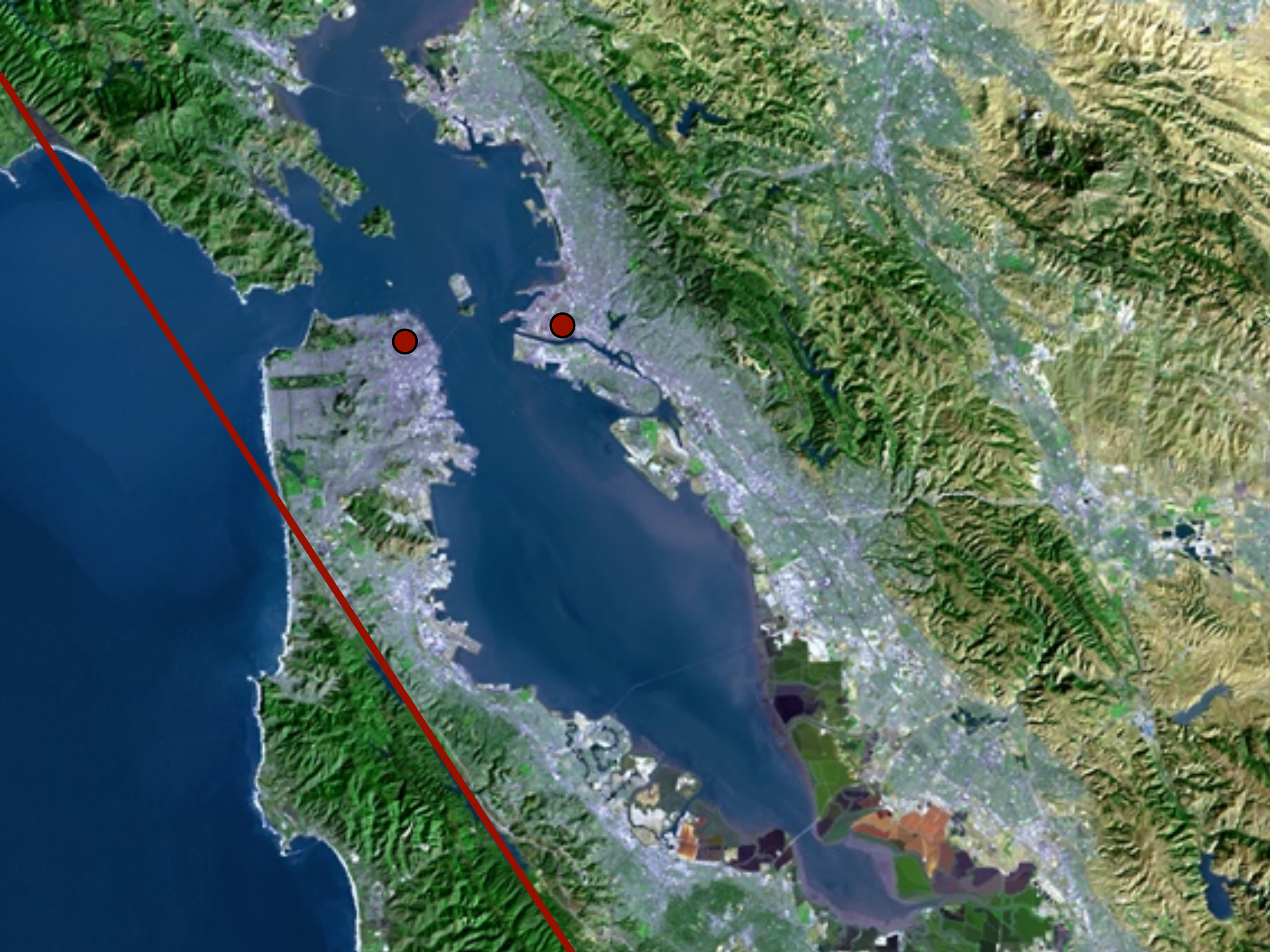
- Anything.
- The hardware.
- The software.
  - Application, PostgreSQL bugs.
- The network.
- The place you've stored your backups.

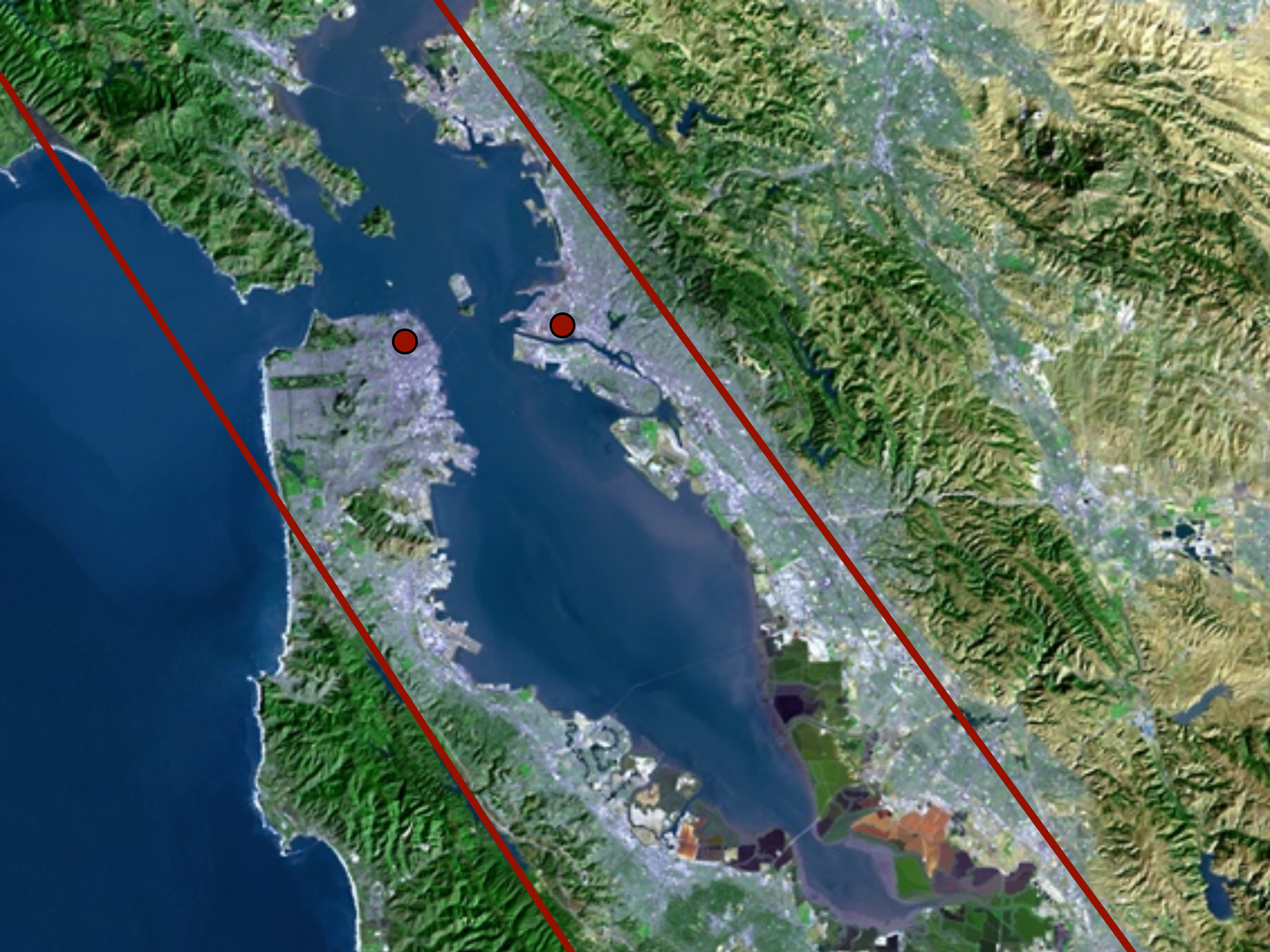
A thought experiment.

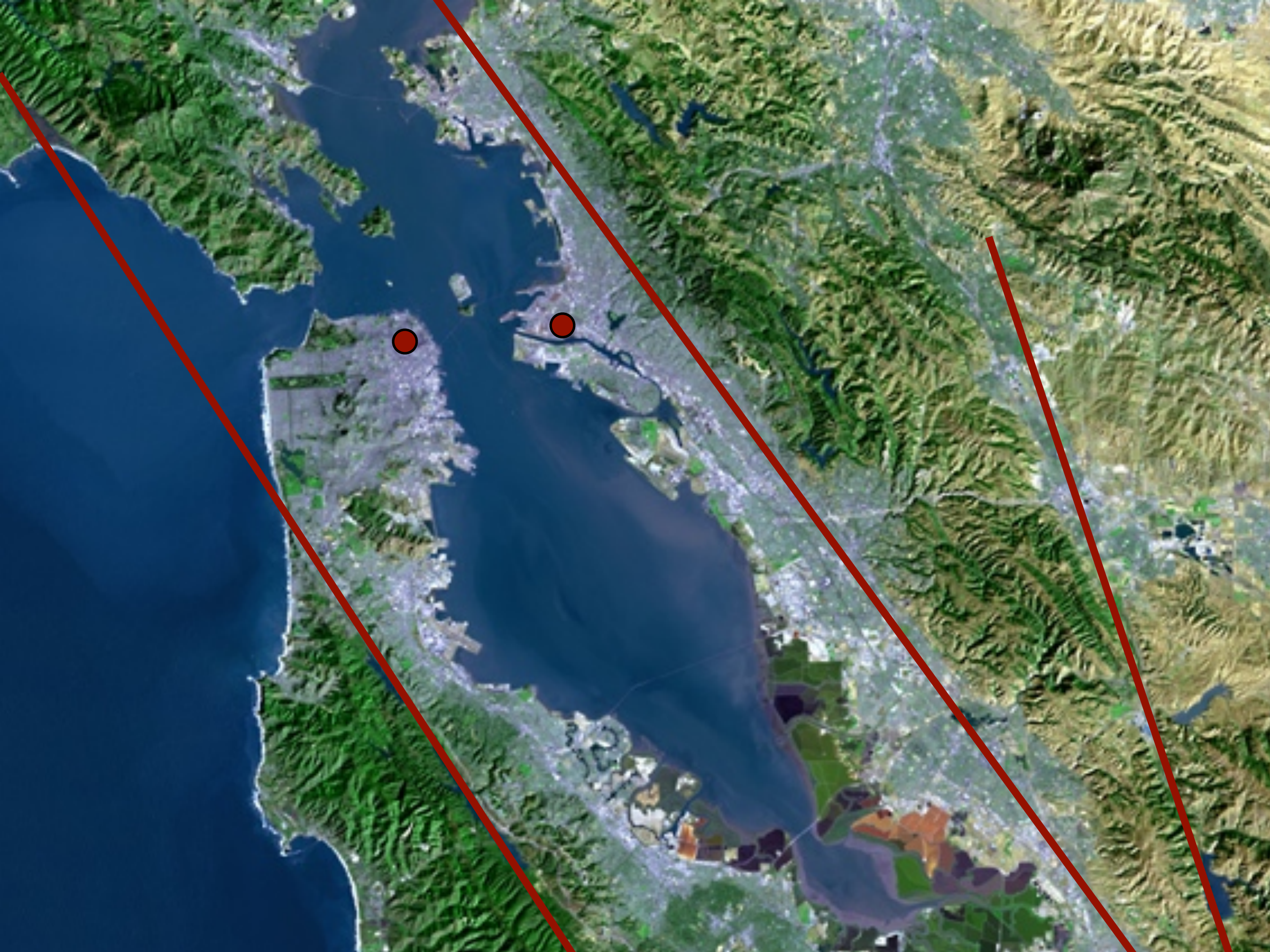
- Right now, what happened if the data center holding your database burned down?
- While you were at the conference.
- In a no-mobile-reception zone?
- The goal is to say, "Oh, that? We know exactly what to do. We know it works."

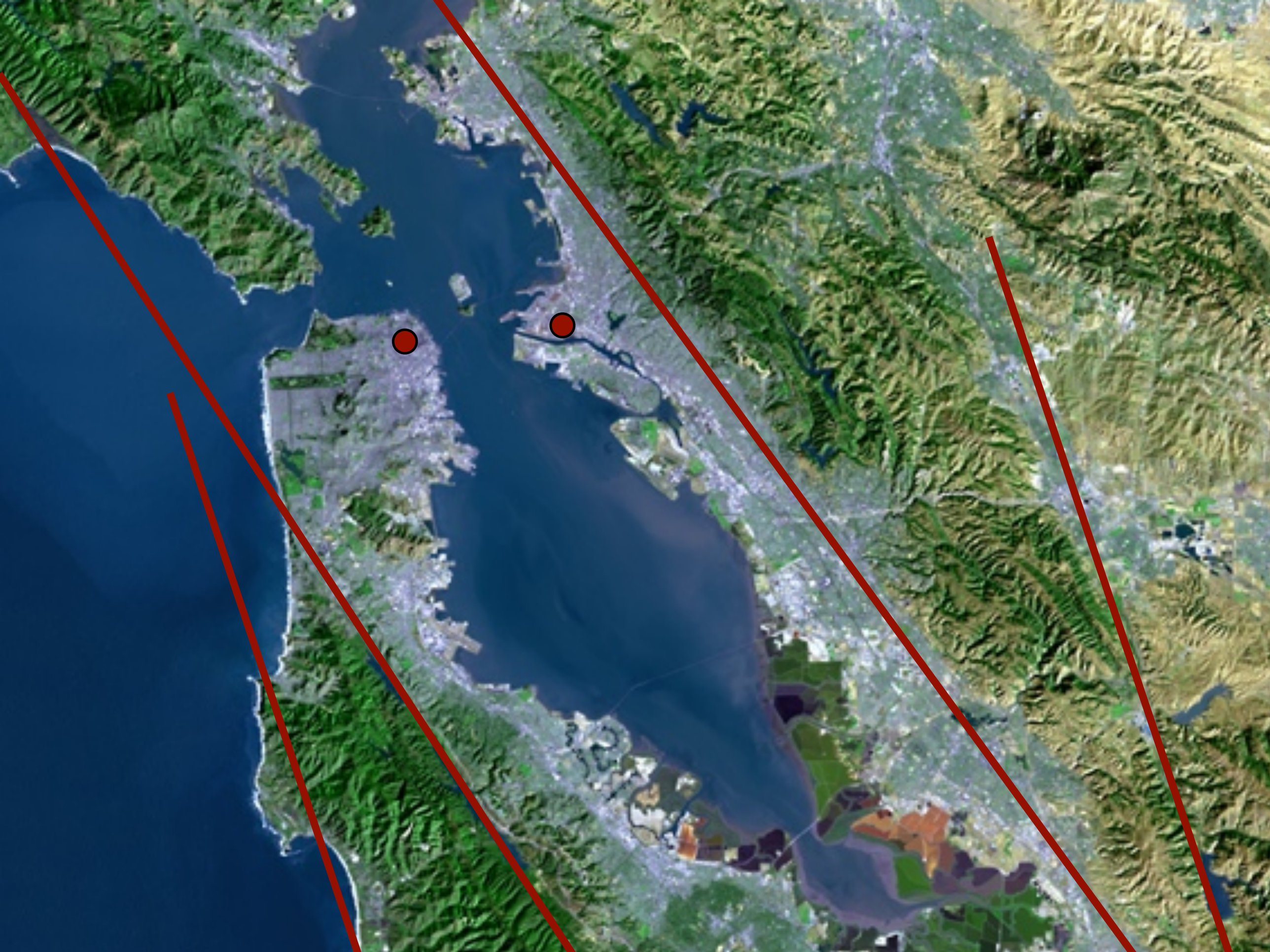
Hardware failures.

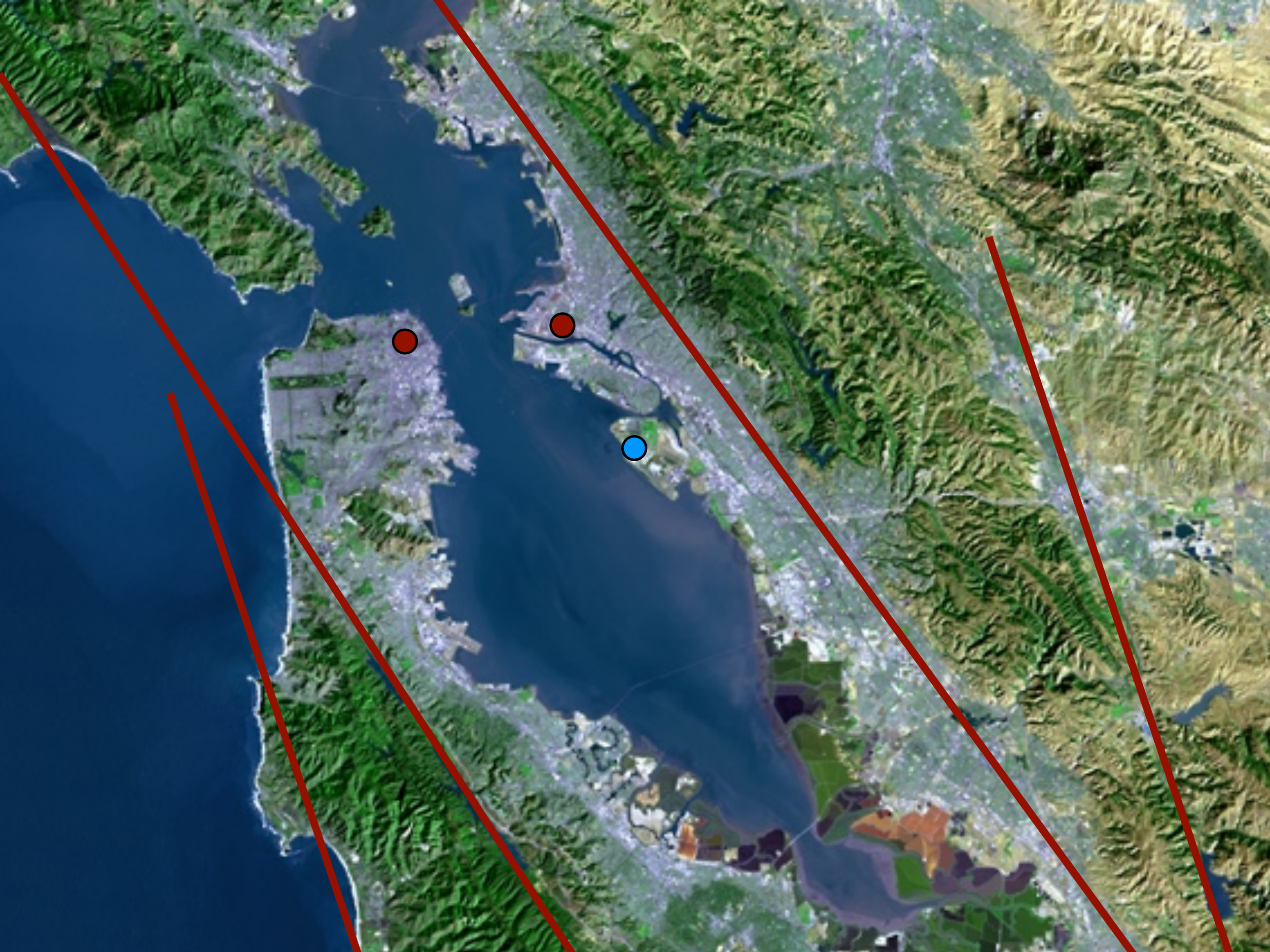
- A single disk.
- A RAID array.
  - Bad firmware upgrades.
- The whole machine.
- The whole data center.
- A whole region of data centers.











Storage failures.

- SANs? You are one bad firmware revision from having a €25,000 end table.
- Disks purchased at the same time can have correlated failures. Very bad in a RAID10.
- SSDs are not significantly more reliable than spinning disks. Nice and fast, though!

Application Failures.

- Bad migrations.
  - Dropping a needed field.
  - “Cleanup” operations that expire needed data.
  - Bugs that destroy data.

Only Human.

- “Oh, was I logged into production?”
- rm -rf *
- Botched upgrades.
- Corrupted pushes of configuration data.

Bad People.

- Your application was hacked.
- You checked in a password or API key into Github without realizing it.
- You fall a bit behind on reading security advisories.
- An appliance is vulnerable even if you aren't.

“We’re covered.”

- Primary in AWS.
- Secondary in the same region, different AZ.
- Another set of secondaries in a different region.
- Everything backed up devotedly to S3.

Code Spaces Status

Code Spaces will not be able to operate beyond this point, the cost of
expected cost of refunding customers who have been left without the s
a irreversible position both financially and in terms of on going credibi

As such at this point in time we have no alternative but to cease tradin
affected customers in exporting any remaining data they have left with

**All that we can say at this point is how sorry we are to both our cu
living at Code Spaces for the chain of events that lead us here.**

In order to get any remaining data exported please email us at suppor
and we will endeavour to process the request as soon as possible.

On behalf of everyone at Code Spaces, please accept our sincere

OK, we're scared now! What do we do?

- Decide on a backup strategy.
  - Complexity, infrastructure support, storage costs, recovery time.
- Document the backup and restore procedures.
- Test, test, test.

Backup methods.

- pg_dump.
- File system backups + WAL archiving.
- Streaming replication.
- Logical replication.

pg_dump

- Takes a consistent, logical snapshot of the database.
- On-line backup: Low impact on the running system.
  - I/O load, no schema changes.
- Highly selective backups: Pick databases, schemas, tables, etc…

pg_dump, the good.

- Very simple to understand and script.
- Lots of control over exactly what to back up.
- Compact backups, as indexes are excluded.
- Parallel backups are fast (for a copy operation).
- Works across major versions.

pg_dump, the bad.

- It's still a full database copy.
- A snapshot as of the start of the backup.
  - Anything after that would be lost on a recovery.
- Restores can take 3-5x as long as backups, due to index recreation.

pg_dump, notes.

- Since it reads every row being backed up, it can find lurking corruption.
  - pg_dump to /dev/null
- A restore from a pg_dump is a new, clean disk image.

pg_dump, procedures.

- Use custom format for dumps.
- Use parallel restore to speed up the process.
- Always do an ANALYZE afterwards!
  - Database statistics are not saved in pg_dump's format.

When to use pg_dump?

- Small databases.
  - Small = pg_dump completes in reasonable time.
- Databases that can lose important work since the last pg_dump.
- As a compact snapshot for long-term archiving or auditing.

File system backups + WAL archiving.

- In PostgreSQL since 8.2.
- Uses file system tools and WAL archiving to produce a backup.

WAL archiving, basic procedure

- pg_start_backup()
- Copy contents of $PGDATA.
- Keep all WAL files generated while copy is going on.
- pg_stop_backup()
- File system copy + WALs is your backup.

THE WALS ARE REQUIRED.

- The file system snapshot is not consistent or valid.
- The WAL records are required to make it consistent.
- You need to save every WAL segment created during the backup.
- Without them, the backup is useless.

WAL archiving, the good.

- Backup process does not pollute shared buffers.
- Potential of using interesting ways of taking the file system backup.
- Backup can be used to prime a secondary for streaming replication.
- Point-in-time recovery.

Point-in-Time Recovery.

- When a WAL archive backup is restored, the WAL records are "replayed."
- They can be replayed all the way to the end, or to an intermediate point.
- If WAL archiving continued after the backup end, the WALs can be replayed past the end of the backup.

You need point-in-time recovery.

- Application failures.
- Schema migration disasters.
- Site hacks.
- Can recover right to the most recent WAL segment.
  - Much closer to “now.”

Taking the file system snapshot.

- rsync.
- File system-level snapshot tools.
  - SAN snapshotting.
  - EBS snapshotting.
  - ZFS magic.
- DRBD-type solutions… if fsync works.

WAL archiving, the bad.

- Complex to set up, has multiple failure modes.
- Backups are a pile of files, harder to manage.
- WAL segments can pile up.
  - Compress everything.
- Same major version only.

pg_basebackup

- Simple tool to do a base backup.
- Can use to prime a warm or hot standby.
- Can receive the required WAL segments as well.
- Great to use just to create an archive, even if you are not going to prime a secondary.

WAL-E.

- A great solution from Heroku for AWS, Rackspace, Azure.
- Handles file system copy to cloud storage.
- Handles WAL archiving to cloud storage.
- Easy to set up just from the README.

WAL archiving, pitfalls.

- Be sure you keep everything.
  - The backup label lets you know what you need.
- Very busy sites can create WAL files faster than you can archive them.
  - Streaming replication in that case.

WAL archiving, other pitfalls.

- Restoring from an archive is time-consuming.
- Replaying WAL files can take as long as it took to create them.
  - Although 20%-40% is more typical.
- A warm standby mitigates this.

WAL archiving, yet more pitfalls.

- Be sure you capture all of $PGDATA.
- And the data in tablespaces, if you use them.
- You have to copy everything. Really. **Everything**. pg_xlog, pg_clog…
  - Except the text logs in pg_log.

Warm Standby.

- A PostgreSQL instance continuously reading WAL records and applying them.
- You can switch over in a minute in case of primary failure.
- Still keep the original archive and WAL segments for PITR, though.
- Streaming replication is the right answer.

Streaming Replication.

- A secondary continuously receives WAL information from a primary over the network.
- WAL archiving not required (but you still want it for PITR).

Streaming Replication, the good.

- Simple to set up.
- Secondary (can) stay very close to the primary.
  - Switchover loses a minimum of data.
- Switchover is very fast… no hours-long restore process.
- DDL is pushed out for you!

Streaming Replication, the bad.

- Can't do point-in-time recovery.
  - Any problem on the primary is immediately pushed to the secondary.
    - Keep a separate archive for PITR.
- Like WAL archiving, all-or-nothing.
- Secondaries can be queried, but are read-only.

Streaming Replication, procedure.

- Create a WAL archive base backup (file system backup + WAL files).
- Set up a recovery.conf file, and fire up the secondary.
- Watch as it tracks the primary!

Streaming Replication, pitfalls.

- Very few. You want to do this.
- The primary doesn't know how far behind the secondary is (in 9.3).
- The primary might throw away WAL information if the secondary is far behind…
  - … requiring re-imaging the secondary.
  - A WAL archive avoids this.

Streaming Replication, notes.

- By default, transactions are applied asynchronously.
  - Some committed transactions may be lost in a failover.
- Synchronous replication avoids this…
  - … at a non-trivial performance penalty.

Why choose?

- WAL archiving can run alongside streaming replication.
- Allows secondaries to recover after long disconnections.
- PostgreSQL can be configured to clean up WAL segments as they are no longer required.

But be careful!

- Be sure that copying WAL files is atomic.
  - scp is not atomic.
  - A secondary going down can run both the secondary and primary out of disk space.
  - Big plus to cloud storage for WALs: Handles distribution for you.

Logical Replication.

- (9.4 has new wonderful features in this area…)
  - (… that we are not going to talk about.)
- Uses trigger-based system to push logical changes to secondary systems.
  - Slony, Bucardo, Londiste…

Logical Replication, the good.

- Lots of control over what to replicate.
- Can replicate between PostgreSQL versions (great for upgrades!).
- Full employment program for consultants like me.

Logical Replication, the bad.

- Full employment program for consultants like me.
- Fiddly to set up.
- Impact on procedures, such as migrations.
- Non-trivial load on machines.

Logical Replication, notes.

- Not really intended as a backup solution.
- Although, if you have to have it for other reasons, it can do that duty.
- 9.4 is going to completely change the landscape here.

Backups and Transactions.

- No backup will capture the state of open transactions.
- Yet another reason that long-running transactions are a bad idea.
- A proper backup will consider all open transactions rolled back when restored.

Building Your Safety Net

Pick your archive location.

- Where can you store data safely?
- Fast recovery tier — Machine in same data center.
- Emergency tier — Same hosting provider, different data center.
- Disaster recovery tier — Different hosting provider, geographically isolated.

#1: A Hot Standby…

- … or two, or three.
- Use for failover in case of primary failure.
- Do not direct significant query traffic to it.
  - That can delay the replication stream.
- Synchronous replication if no transaction loss can be tolerated.

#2: A **Cold** Standby

- A cold standby in the emergency tier.
  - Configured, but PostgreSQL not running.
  - Received file system backups and WAL archives.
  - Bring up if required for point-in-time recovery or if fast recovery tier unavailable.

#3: Long-Term Archives.

- Copies of file system backups and WALs for archiving in disaster recovery tier.
- pg_dumps, if practical.
- Different retention policies (fewer, longer) than in other tiers.

Test your backups.

- Regularly test backups.
- Use them to prime staging environment.
- Distribute them (daily, weekly) to developers.
- Script the restores! Do not rely on manual restoration.

Do scenario planning.

- Data center OK, office off-line.
- Data center destroyed.
- Entire region off-line (earthquake, major storm).
- Every bad thing you can imagine has happened at some point.

Script everything.

- Make sure you can rebuild the host from scratch.
- All real environments tend to "grow" special requirements.
- Use a configuration management system!
  - Salt, Chef, Puppet…

Document everything.

- Have a runbook for how to rebuild your database environment.
- Hand it to a junior developer and don't answer questions.
- Iterate until it can be done by anyone, no matter how unsophisticated.
  - Even the CEO.

When disaster strikes.

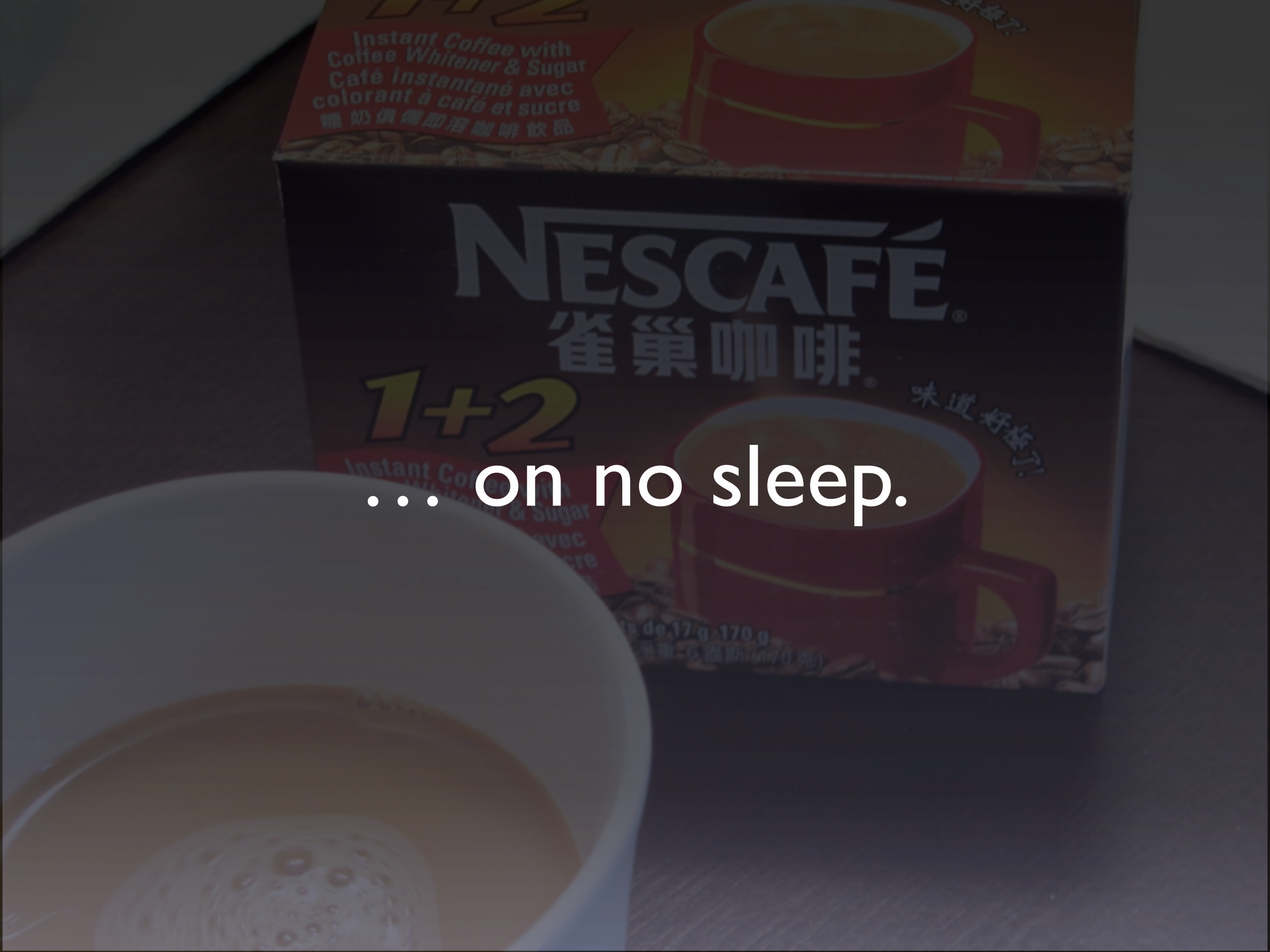
The First Step.


STOP
ARRÊT

Stop. Think. Follow your procedures.

- Minimize communication channels.
  - Decide on how to handle this in advance.
- Follow your instructions.
- If something weird happens, stop.
  - Crisis = Problem + Panic.
- Remember, you'll probably be doing this…

NESCAFÉ
雀巢咖啡
味道好極了!
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
Instant Coffee with
Coffee Whitener & Sugar
Café instantané avec
colorant à café et sucre
糖奶伴侶即溶咖啡飲品

. . . on no sleep.

In summary!

- Develop and document your backup procedures.
- Just because you're paranoid doesn't mean it's not going to fail.
- Test your procedures.
- Don't rely on knowledge locked in someone's head to do restores.

Thank you!

Christophe Pettus
PostgreSQL Experts, Inc.

pgexperts.com

thebuild.com

christophe.pettus@pgexperts.com

Twitter @xof