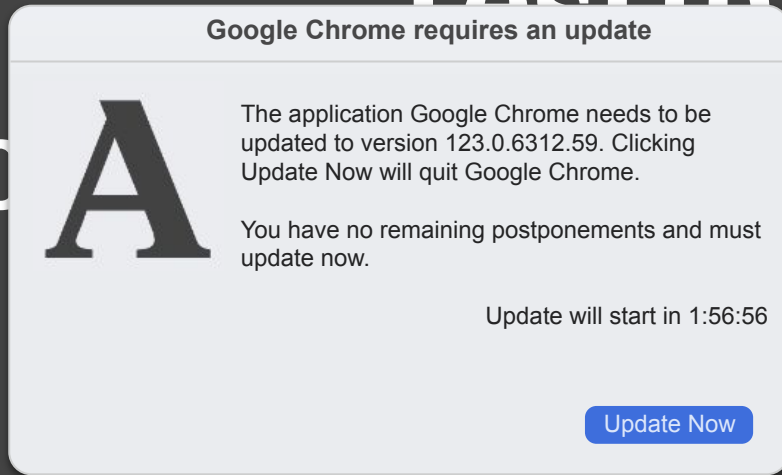


# Testing your PostgreSQL backups (a practical guide)



Nick Meyer  
Staff Software Engineer, Academia.edu  
PGConf NYC 2024

# Testing your backups guide)



Nick Meyer  
Staff Software Engineer, Academia.edu  
PGConf NYC 2024

1972: The “Blue Marble” (Apollo 17)  
Image Credit: NASA

### Google Chrome requires an update

**A**

The application Google Chrome needs to be updated to version 123.0.6312.59. Clicking Update Now will quit Google Chrome.

You have no remaining postponements and must update now.

Update will start in 1:56:56

[Update Now](#)

---

Sunday, September 1st ▾



**Slackbot** 12:00 PM

Reminder: Time to test postgres backups! We test backups on the 1st of every month to make sure we can recover from disasters. See [this post](#) for instructions.





## Postgres backups

1. How to test them
2. How to *make sure* we test them



A

## A bit about me (Nick Meyer)

- <https://github.com/aristocrates>
- **Team lead of Platform Engineering**
- **Areas of focus**
  - Developer experience
  - Interface: application and infra
  - **Data layer**
  - **Postgres**





## Academia.edu



- <https://www.academia.edu/about>
- **We're hiring!**
- **Our goals**
  1. Ensure that every paper ever written is:
    - ✓ on the internet
    - ✓ available for free
  2. Accelerate the world's research
- **Some stats**
  1. 50 million papers uploaded
  2. 20 million paper recommendations per day



A

## Quick poll

- **DB as a Service**
  - RDS/Aurora, GCP Cloud SQL, Azure/Cosmos DB
- **Infra as a Service**
  - EC2 instance, or VMs on GCP/Azure/etc
  - ansible/chef/puppet/shell scripts
- **On-prem**





## Our old postgres backup solution

- -100TB across -15 “clusters” (AWS EC2 + some RDS)
- (EC2) All backed up by: a Ruby script
  - (that wrapped pg\_basebackup)
- A great way to learn about backups...
- ... but a bad idea otherwise



**A**

“A practical guide”





## Roadmap

1. Motivation
2. How to test
3. Measurable goals
4. How to *hold ourselves accountable for testing*
5. Monitoring

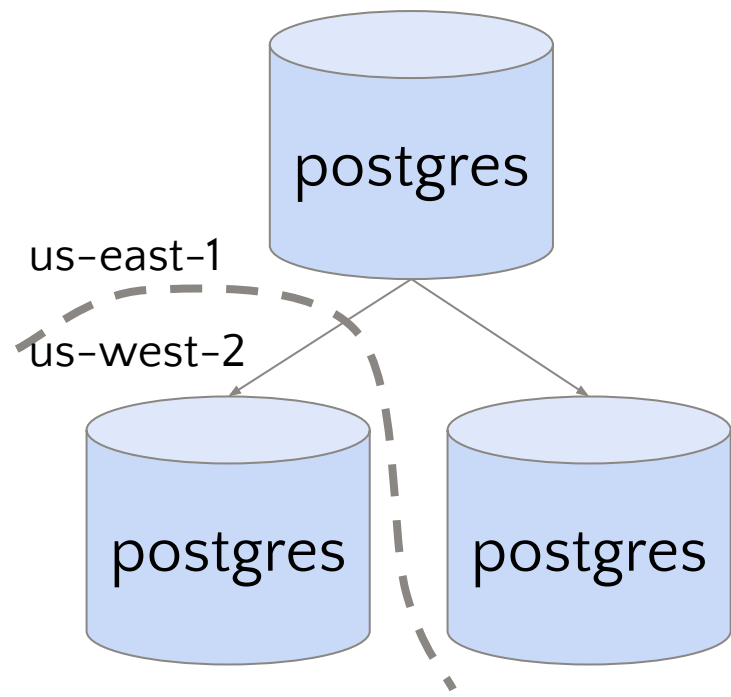
# What could go wrong?

**A**

**A**

## What could go wrong: Several nodes

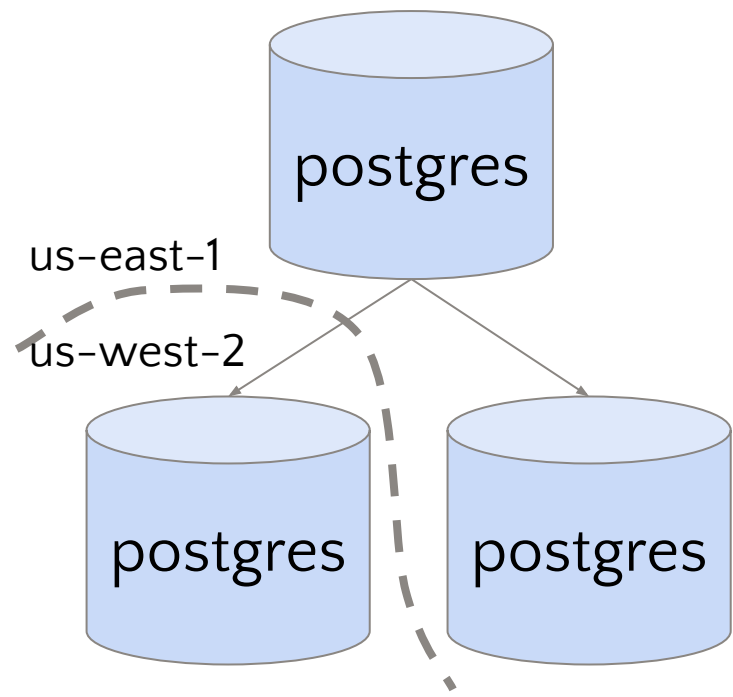
- What if all nodes go down?
- Some nodes go down: all good?



**A**

## What could go wrong: Several nodes

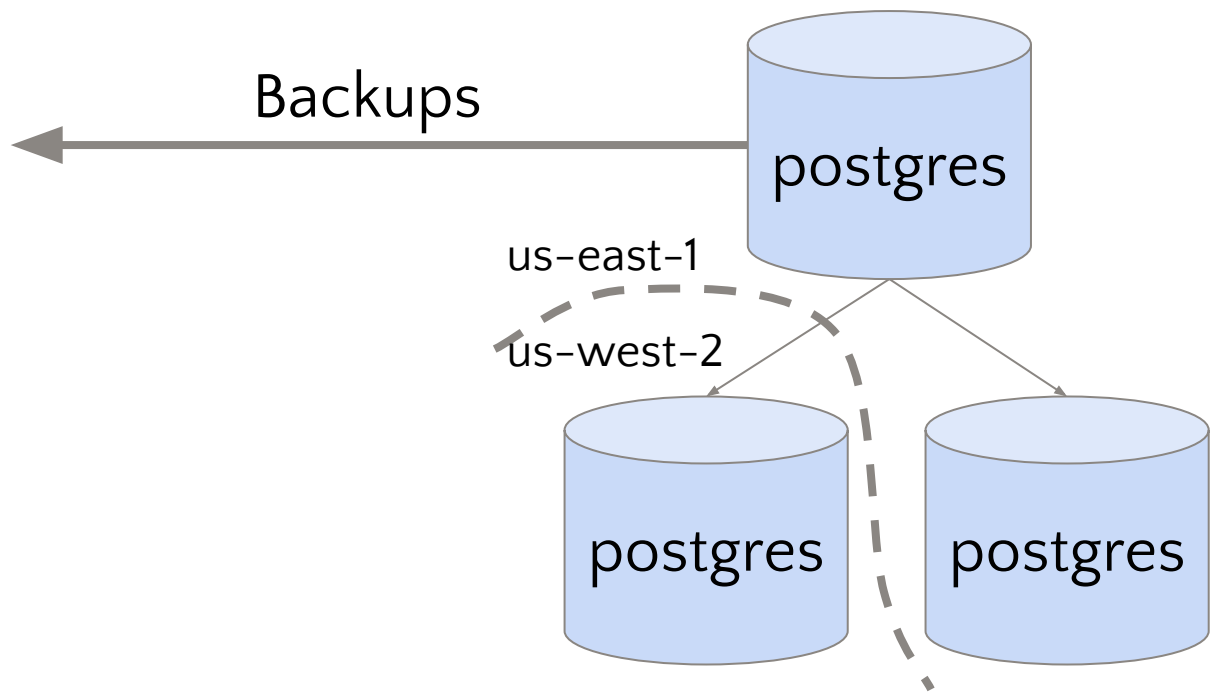
- What if all nodes go down?
- Some nodes go down: all good?
- `DELETE FROM users;`
- `DROP TABLE users;`



**A**

## What could go wrong: Several nodes and backups

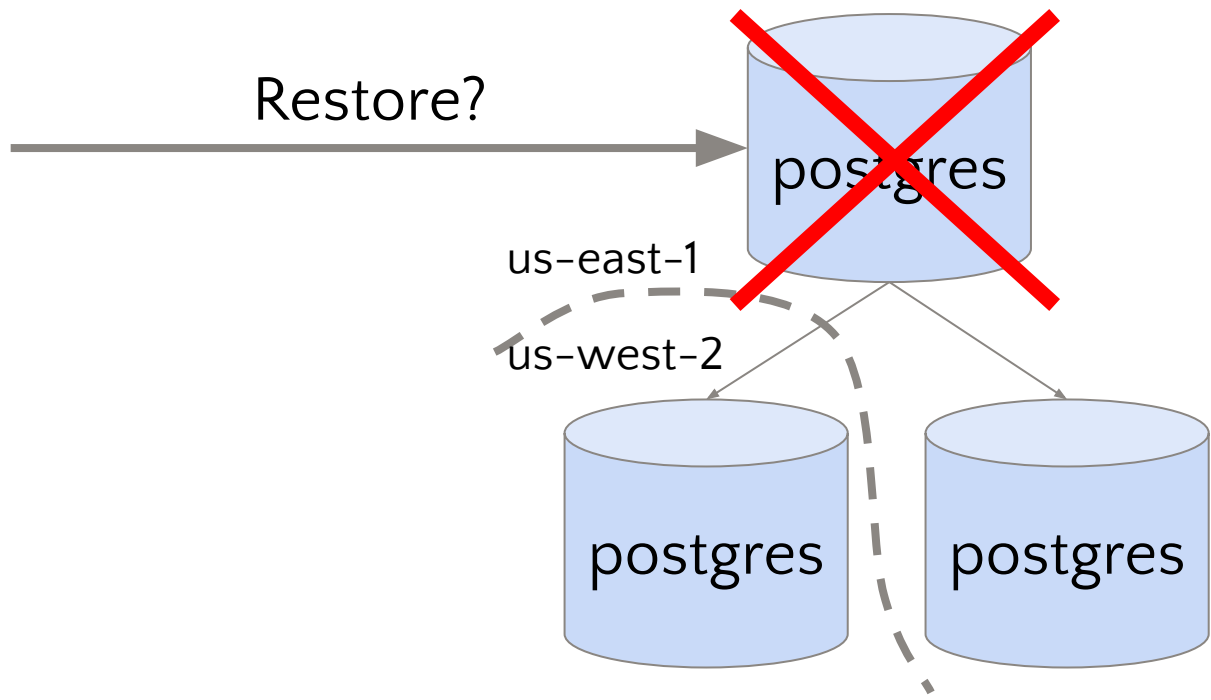
S3 bucket



**A**

## What could go wrong: Several nodes and backups

S3 bucket





Schrödinger's Backup:  
“The condition of any backup is  
unknown until a restore is  
attempted.”

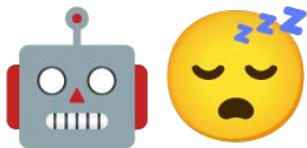
—Spotlight on IT series #212,  
Spiceworks 2013



**A**

## Backup failures that I have witnessed in prod

1. Backups just weren't happening



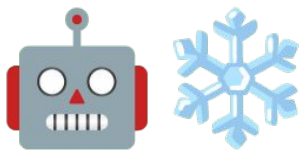
## **A** Backup failures that I have witnessed in prod

1. Backups just weren't happening
2. "Successful" backups in s3 that are just an empty file



## **A** Backup failures that I have witnessed in prod

1. Backups just weren't happening
2. “Successful” backups in s3 that are just an empty file
3. 🤪 Looked good, but postgres never finished starting...

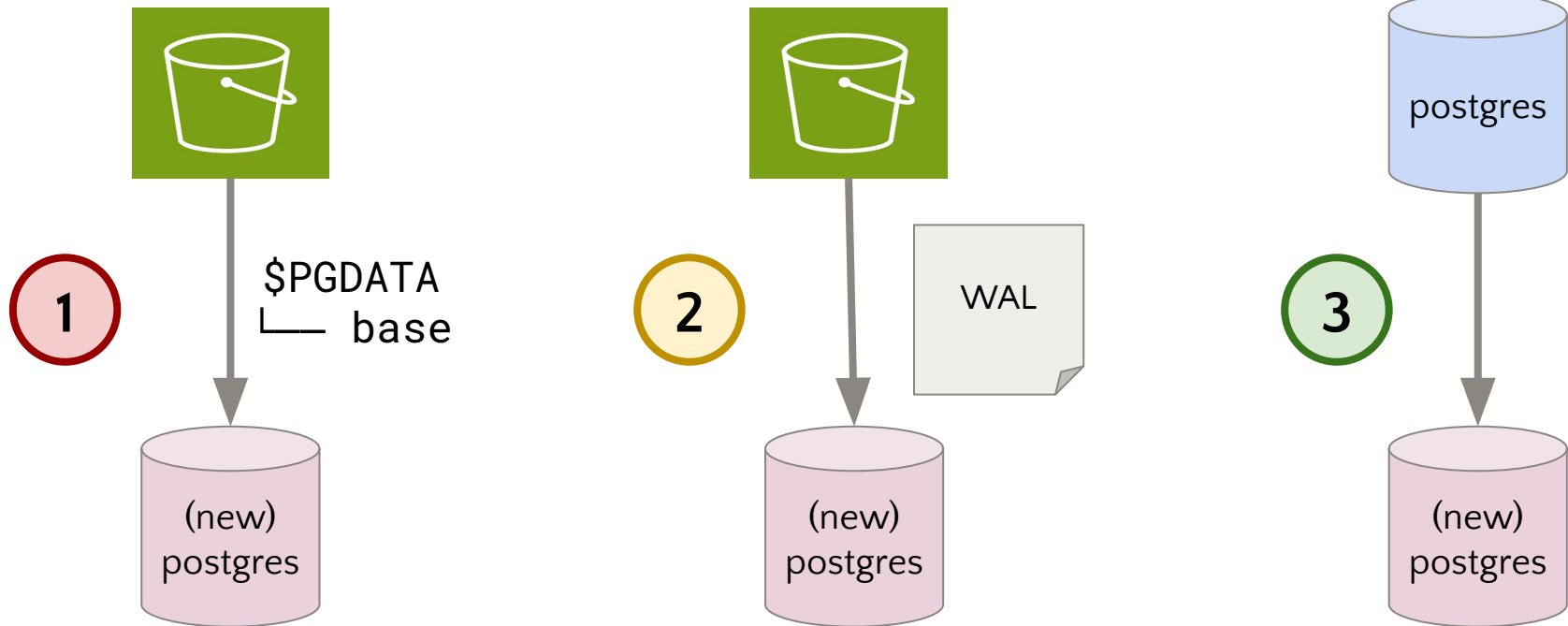


# How do we test restores?

**A**

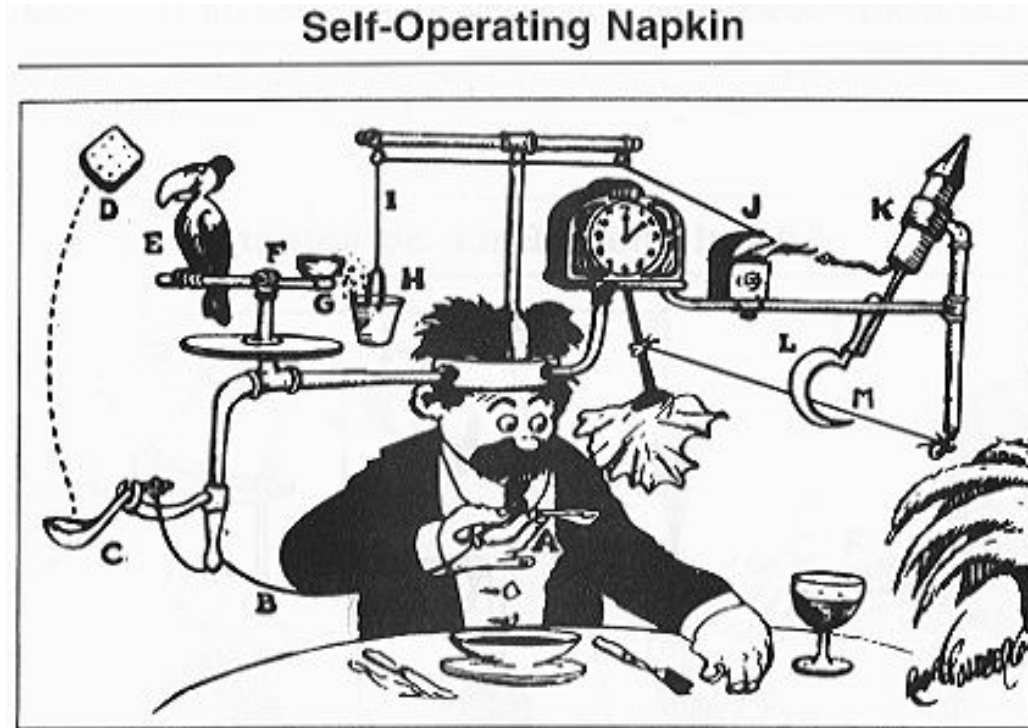
2/5

# A Bring up a streaming replica



**A**

## Step 0: Bring up infrastructure



## **A** Step 0: Bring up infrastructure

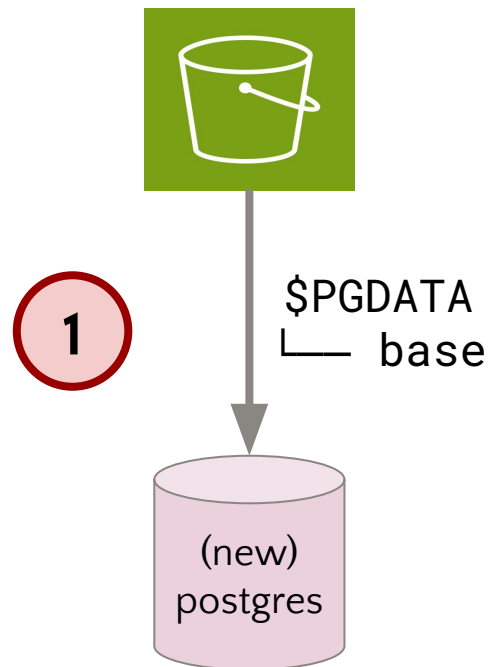
- A lot of failures happen here
- Great news!
- Your backups *might* still be valid



**A**

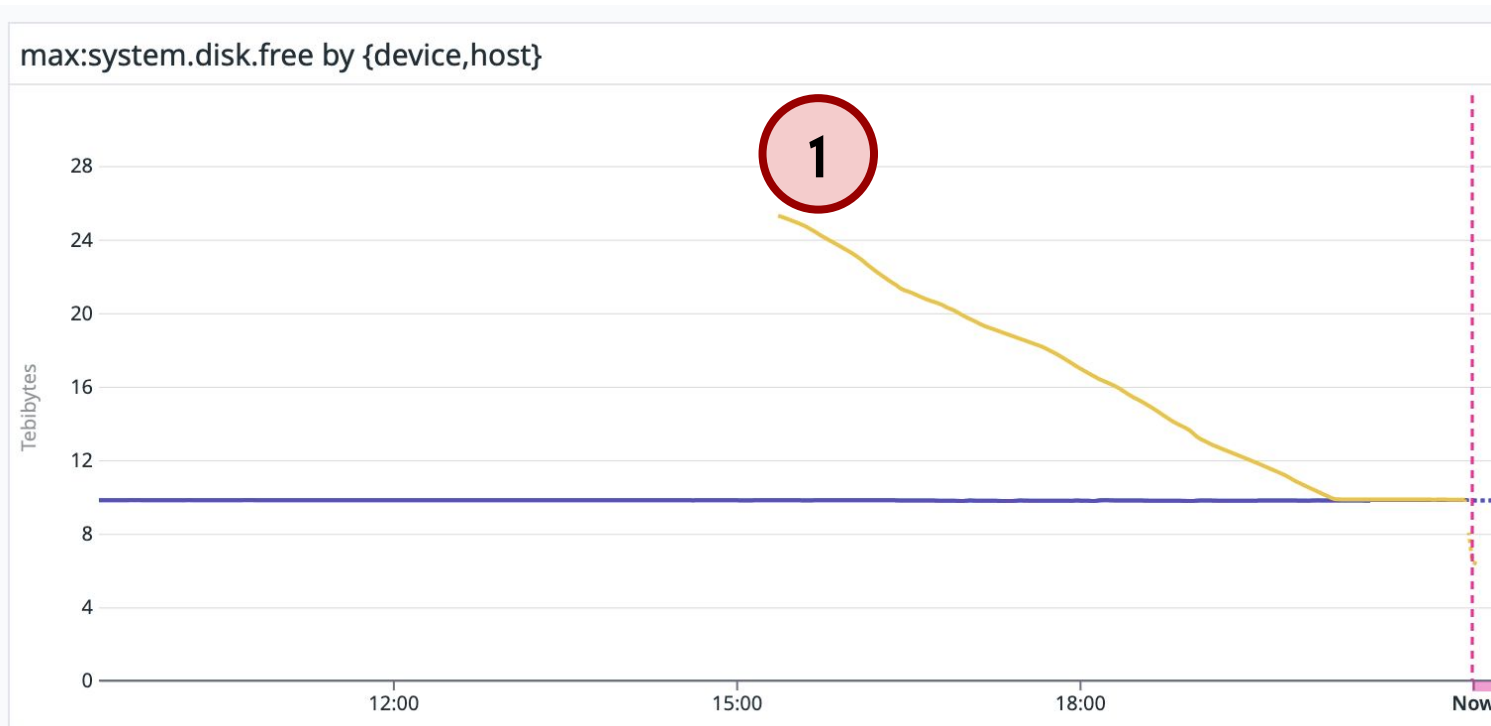
Step 1: restore \$PGDATA snapshot

```
pgbackrest restore [args]
```



**A**

## Step 1: restore \$PGDATA snapshot



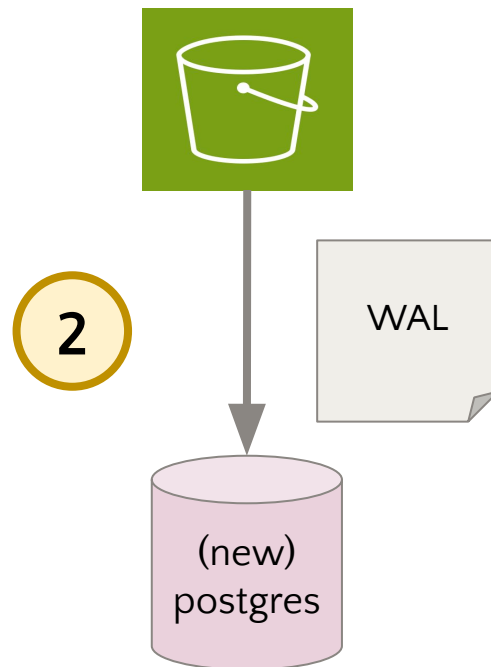
## **A** Step 1.5: Set up configs

- `postgresql.conf`
- `pg_hba.conf`
- TLS (`ssl_cert_file`, `ssl_key_file`, `ssl_ca_file`)

**A**

## Step 2: Replay WAL to catch up

Start postgres



# Aside: the role of WAL



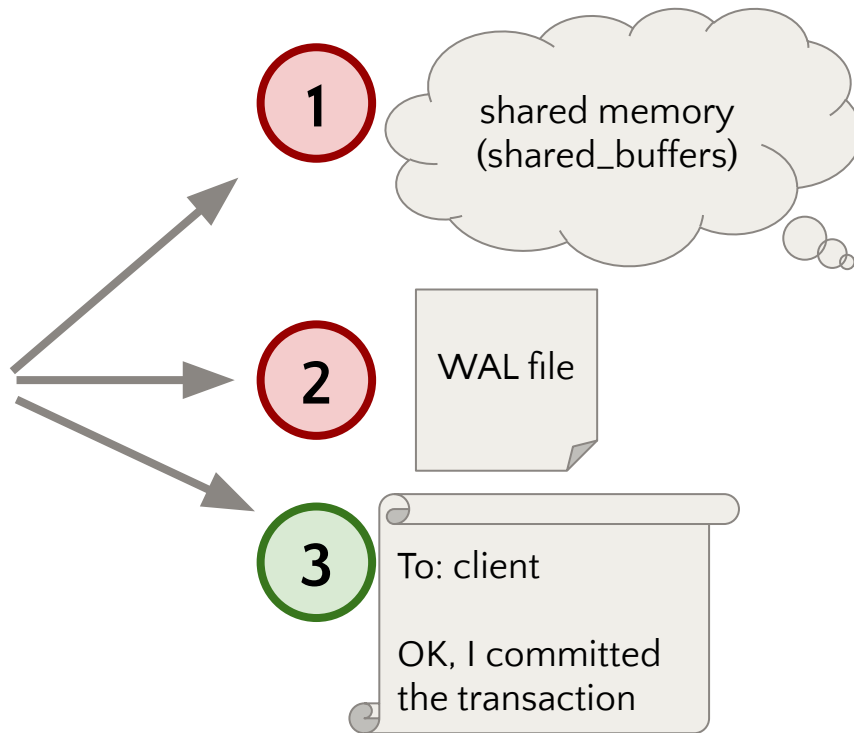
2.5/5

# A Write-Ahead Log (WAL)



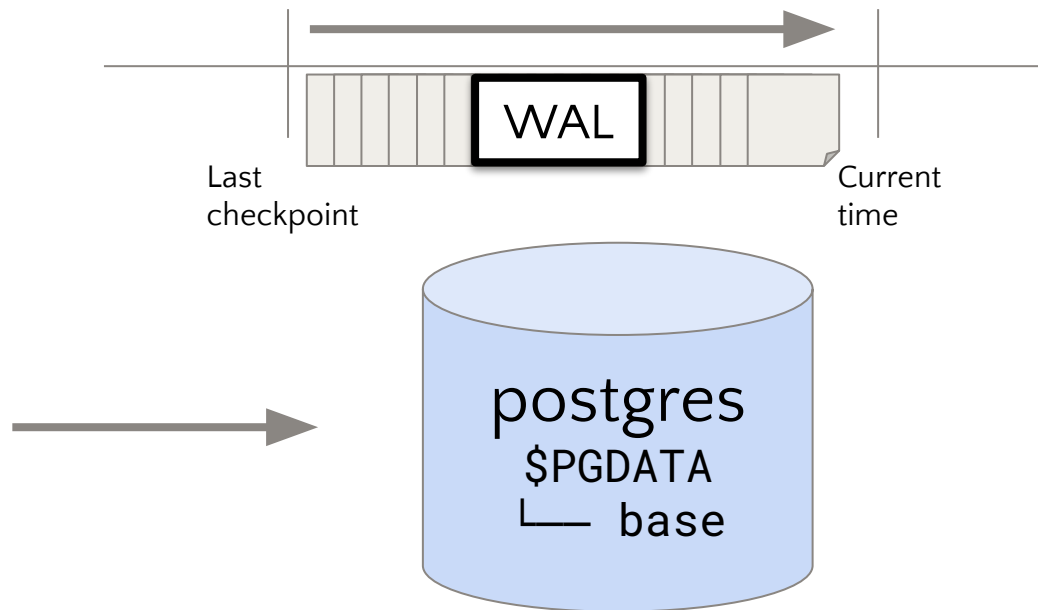
INSERT INTO users(id, name)  
VALUES (1, "Nick");

# A Write-Ahead Log (WAL)



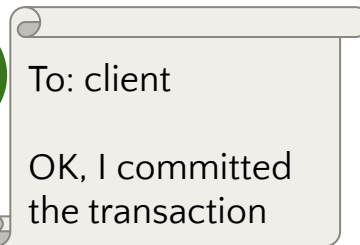
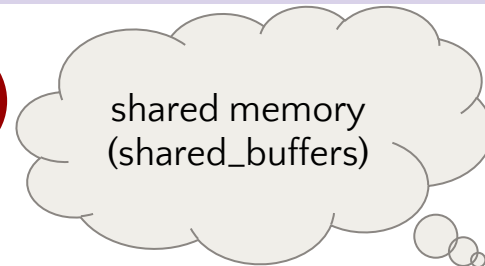
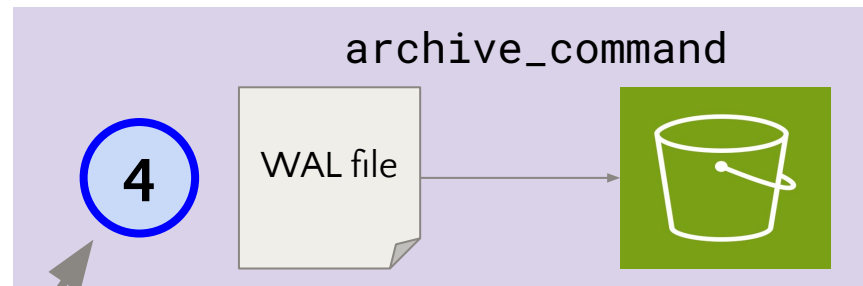
**A**

# Write-Ahead Log (WAL) -> "Checkpointing"





# A Write-Ahead Log (WAL)



`</aside>`

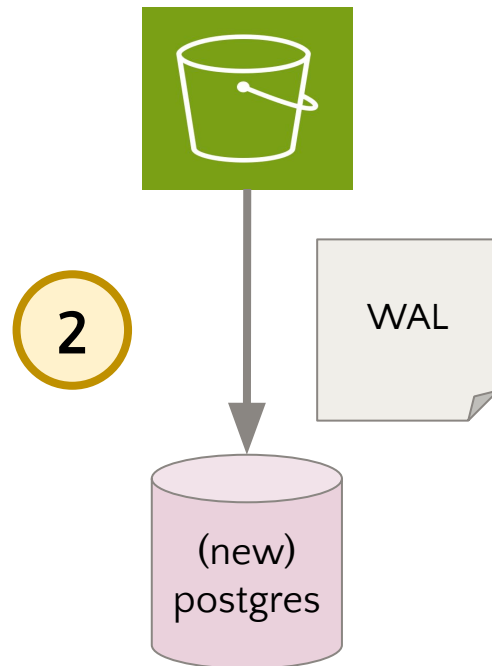


2.5/5

## **A** Step 2: Replay WAL to catch up

### Start postgres

- 1. Needs to reach “initial consistency”**
  - a. The snapshot of \$PGDATA is not useful without WAL from start to finish
- 2. Catches up with all WAL written after the backup finished**



# A

## Step 2: Replay WAL to catch up

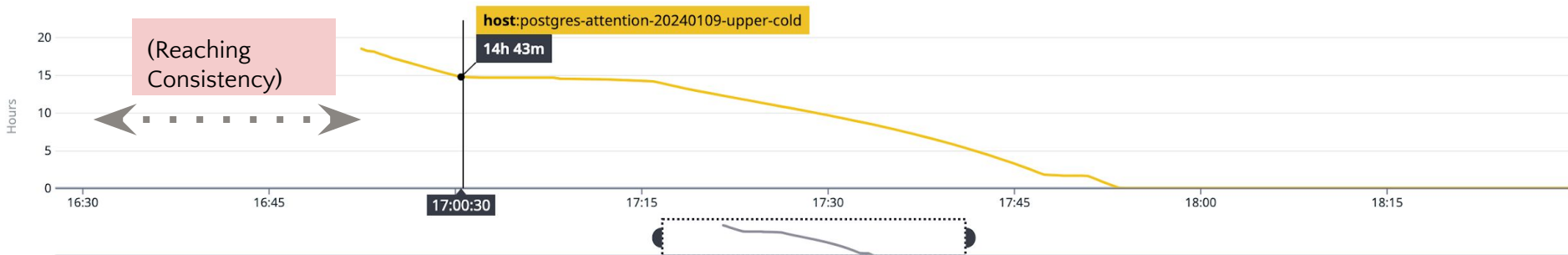
- Look for the “slopes of catching up”
  - This comes **after** initial consistency is reached

Edit Overview Split Graph NEW Correlations

2h Jan 9, 4:27 pm - Jan 9, 6:29 pm

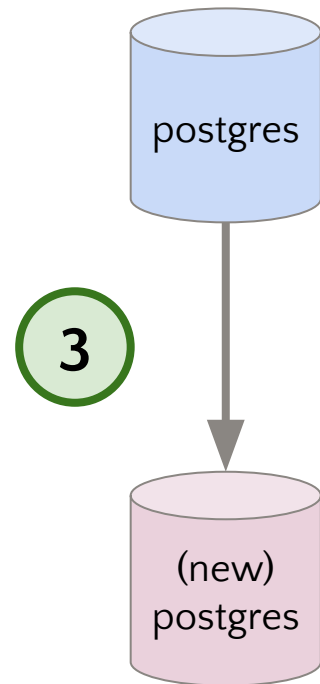
postgresql.replication\_delay, production.replication\_heartbeats\_delay\_minutes

Save to Dashboard More...



## A Step 3: Start replicating

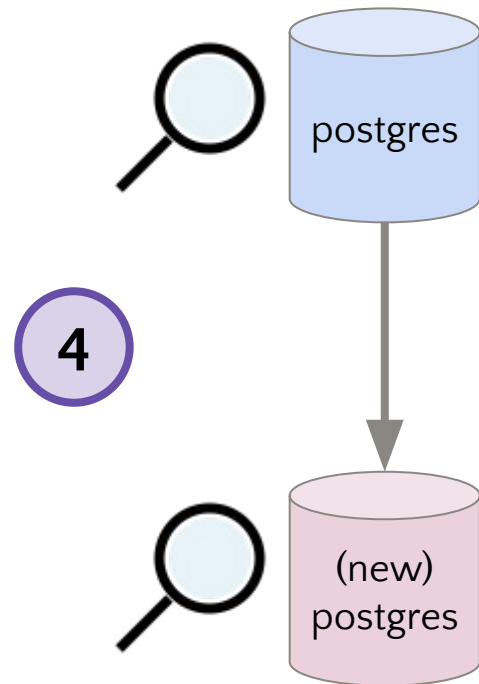
- `primary_conninfo`
- Replicate without error: good sign
- This is (probably) enough



**A**

## Step 4: Spot-check some data

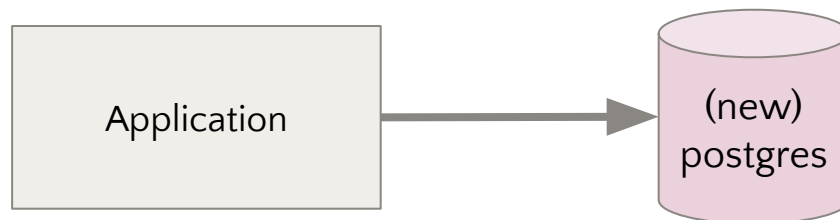
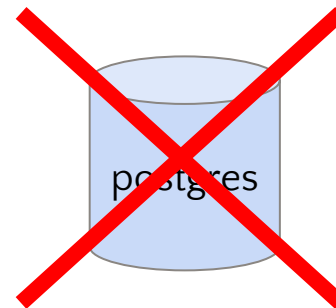
- **“heartbeats” table good for this**
  - You can watch it update



**A**

**If this is not a drill, skip steps 3 and 4**

- **Hopefully you tested restores**



# What goals should we set?

**A**

3/5



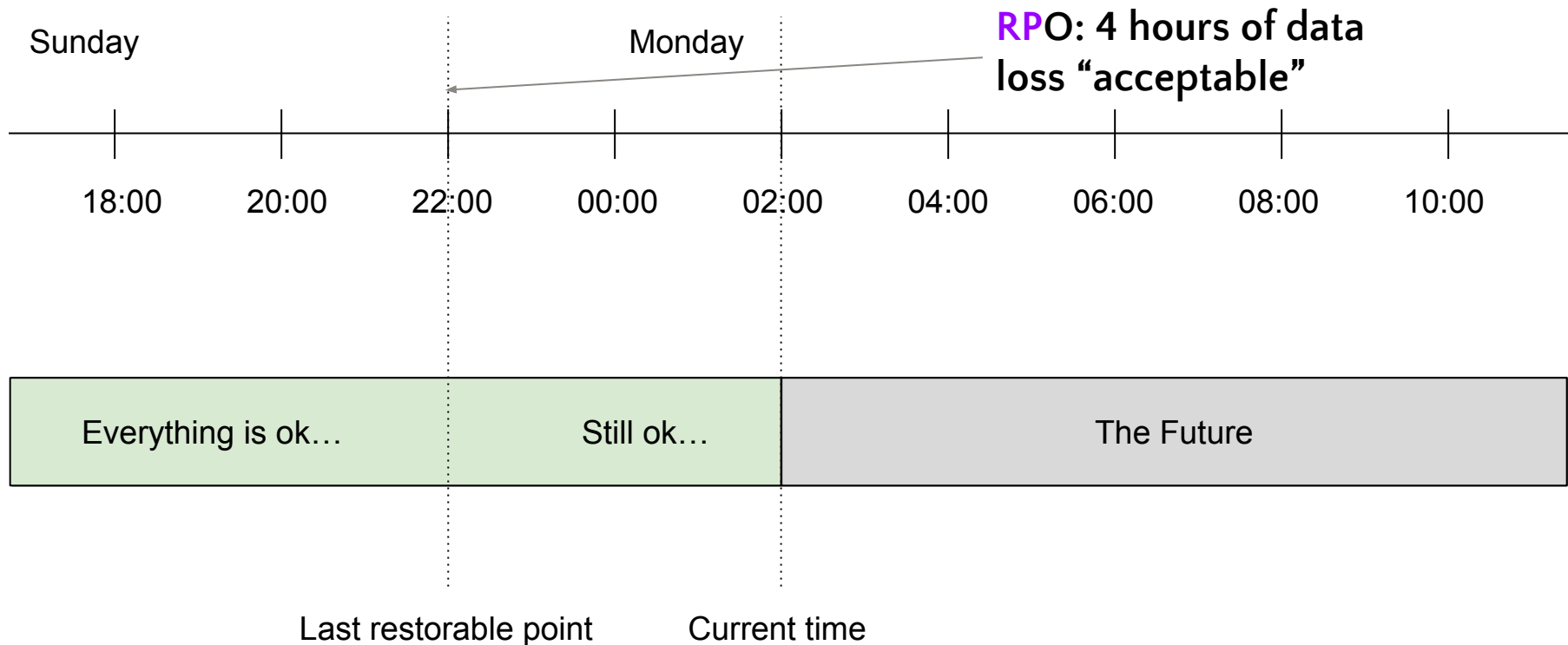
# A

## RPO and RTO

- How much data loss?
  - Recovery Point Objective (RPO)
- How long until we're back?
  - Recovery Time Objective (RTO)

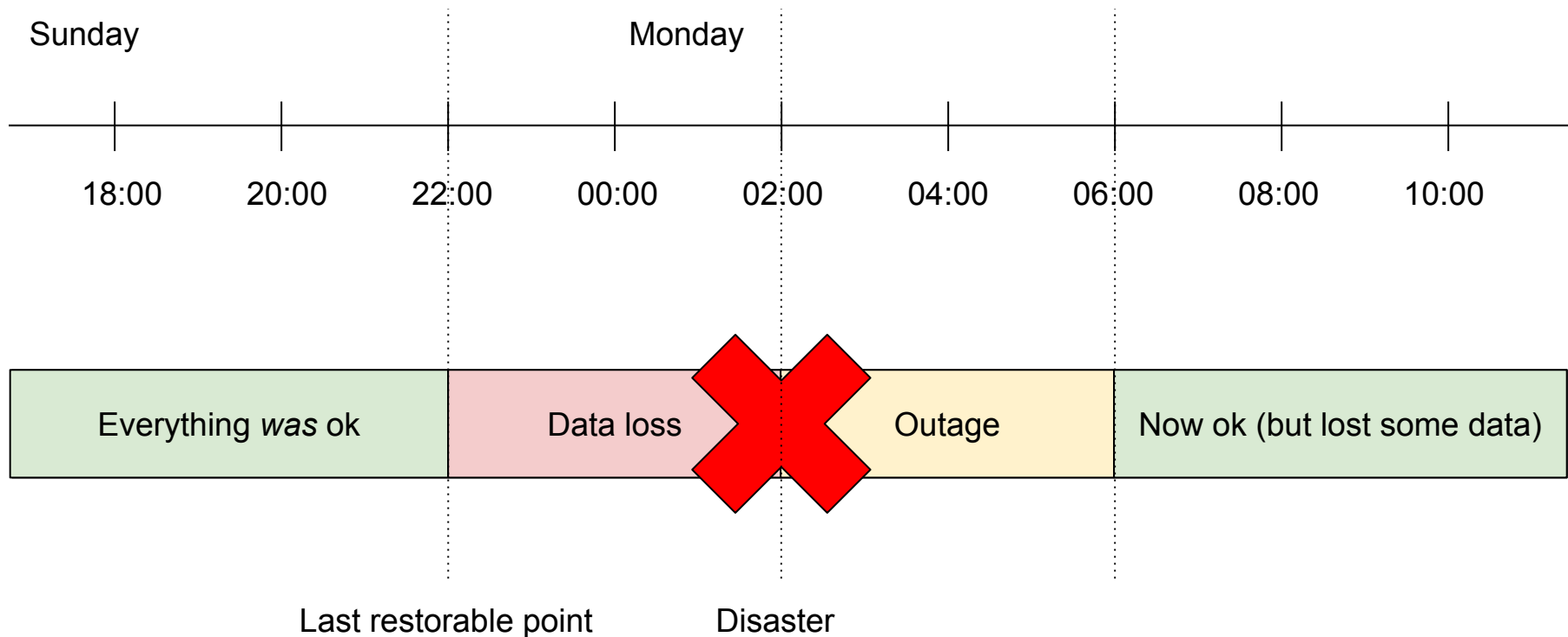
**A**

## Recovery Point Objective (RPO)



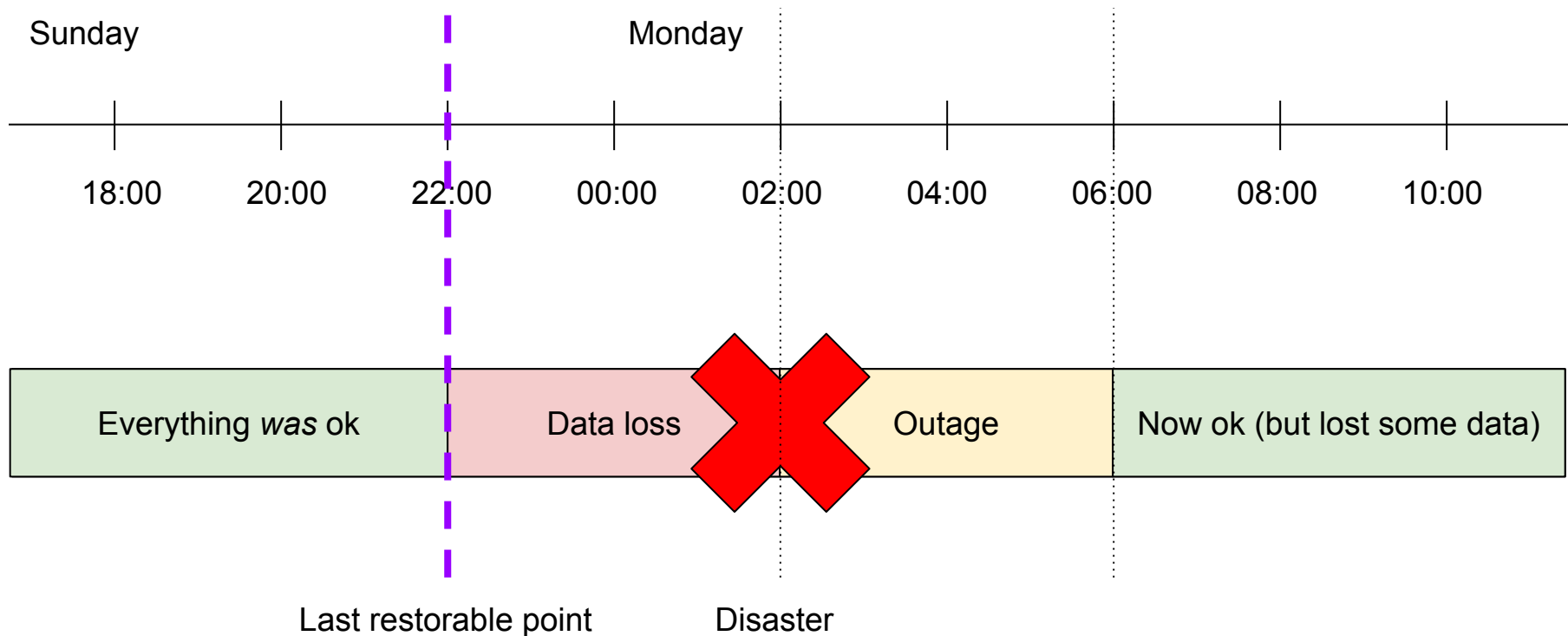
**A**

## Recovery Point Objective (RPO)



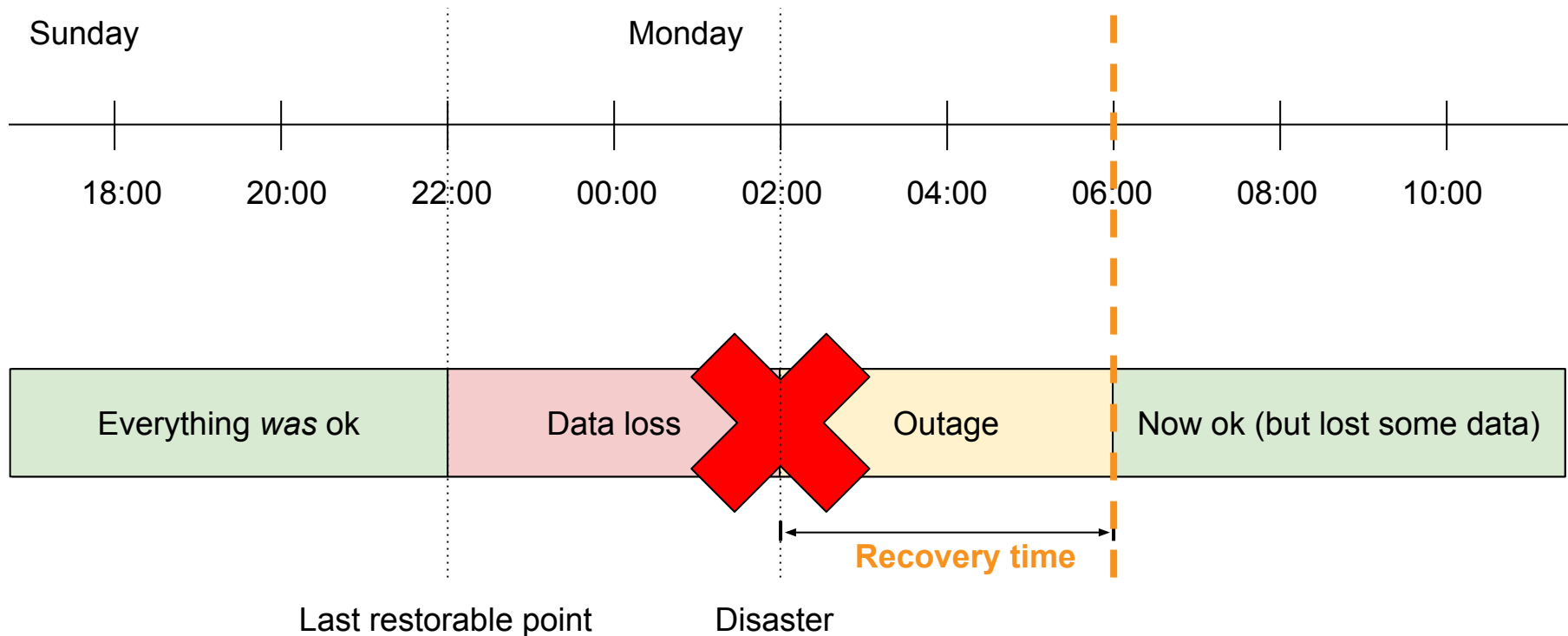
**A**

## Recovery Point Objective (RPO)



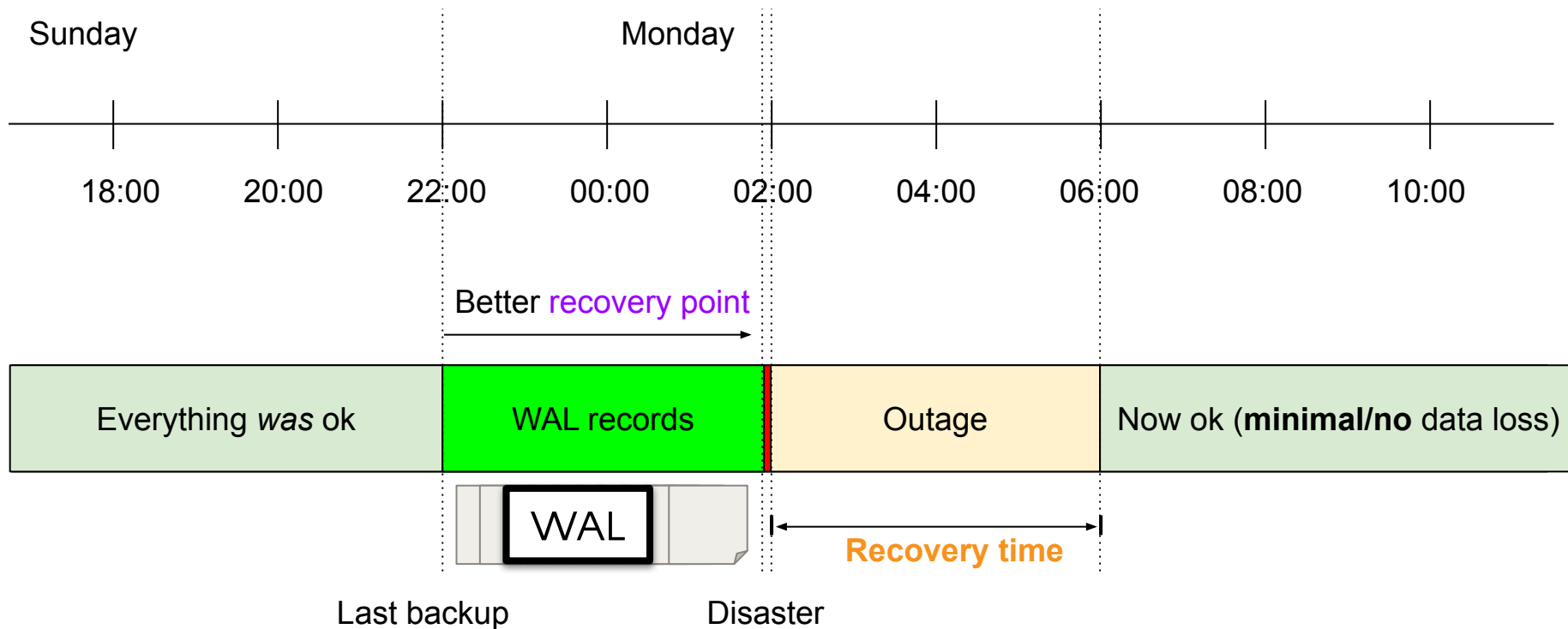
**A**

## Recovery Time Objective (RTO)

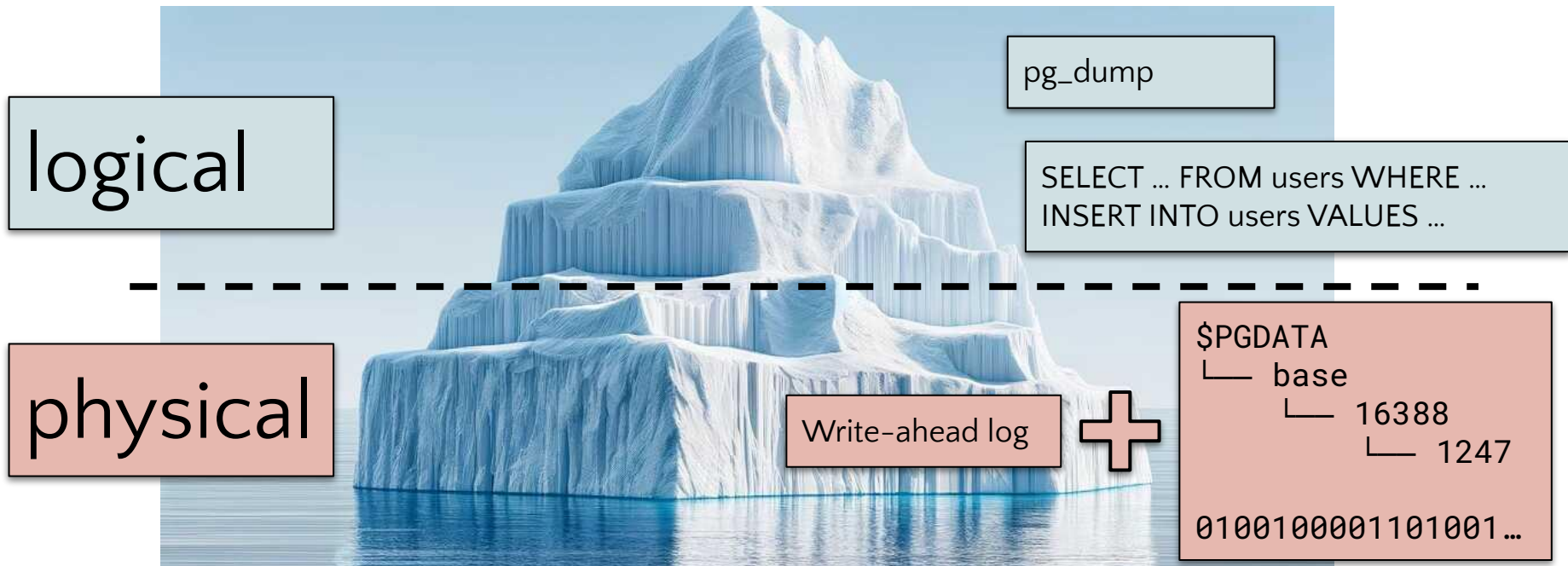


**A**

# Point-in-time Recovery (PITR) with WAL



# A Physical vs Logical



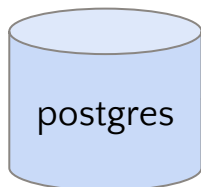
## A Physical vs logical

- Backups are faster, more frequent
- WAL => better RPO, continuous PITR
- Restores are faster => better RTO



**A**

**“Super physical”**



Filesystem /  
block device  
snapshots

ZFS

LVM

EBS  
snapshots

logical

physical

“super  
physical”

## **A** Physical vs “super physical”

- “Super physical”: can use with e.g. MySQL too
- Physical: Less fragile
  - CREATE TABLESPACE ...
- Better postgres tooling for physical
- Physical has PITR, does “super physical”?



## Some of our specific numbers

- For a 15 TB DB @ Academia:

Objective	Target
Recovery Point	Everything*
Recovery Time	6 hours**
Point-in-time	1 month

\* Allowance for several seconds to several minutes

\*\* Multiply by 3 in full disaster (restore from nothing)

# Holding ourselves accountable

**A**

# A

## How to *make sure we test restores*

- 1. Every time you need a new replica, use your backups**
  - a. If you never/rarely need new replicas, bring one up for fun
- 2. When you need to test, bring up a copy in staging**
  - a. Restore some point in time **other than latest**

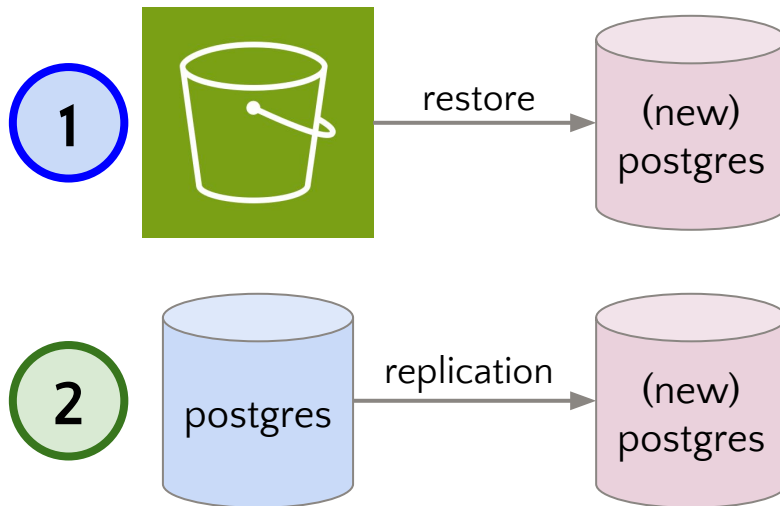
# A

## Business need #1: Need for reads

1. Every time you need a new replica, use your backups
  - a. If you never/rarely need new replicas, bring one up for fun

### ➤ When backups break:

- ✓ You will notice
- ✓ Fixing will be a priority



## **A** Business need #2: QA

- 2. Bring up a copy of prod in a staging environment**
  - a. Restore some point in time other than latest**

### **➤ Confirm:**

- ✓ You can restore from nothing
- ✓ You can restore earlier points in time

**A**

How often?

0 -> once

once -> yearly

yearly -> monthly

(etc)



# Monitoring



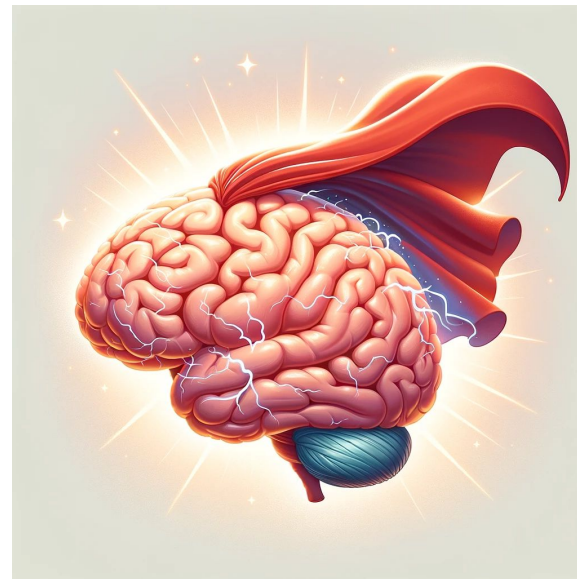
**A** Good monitoring

Loud when it needs to be



**A** Great monitoring

Quiet the rest of the time



**A**

**How to notice when restores are failing?**

- **Alerts?**
- **Dashboards?**

A

## Slack bot (failure)



**incoming-webhook** APP 10:41 AM

Monday, January 29th ▾

Setup: i-0123456789abcdef0 being set up as Postgres::News in qa in us-east-1b




**incoming-webhook** APP 10:50 AM


Setup: Couldn't setup [postgres-news-20240129-innocent-sam qa](#)  
i-0123456789abcdef0 Postgres::News

A

## Slack bot (success + recovery time)

Today ▾

 **incoming-webhook** APP 3:06 PM  
Setup: i-0123456789abcdef2 being set up as Postgres::News in production in us-east-1d

 **incoming-webhook** APP 9:17 PM  
Setup: i-0123456789abcdef2 now available as postgres-news-20240314-status-absence

~6 hours



## The search for leading indicators

- Restores are what we care about
- Broken restores = lagging indicator of broken backups
- Are there any **leading indicators** to monitor?

**A** **pgbackrest info command: pipe to head**

```
postgres@host $ pgbackrest info | head
```

```
stanza: news
```

```
status: ok
```



```
cipher: [value]
```

```
[...]
```



## **A** pgbackrest info command: pipe to tail

```
postgres@host $ pgbackrest info | tail
```

```
[...]
```

```
    full backup: 20240309-181002F
```

```
        timestamp start/stop: 2024-03-09 18:10:02 /  
2024-03-09 18:10:45
```

```
        wal start/stop: 00000002000003D1000000BB /  
00000002000003D1000000BB
```

```
        database size: 2.9GB, database backup size: 2.9GB
```

```
        repo1: backup set size: 696.8MB, backup size: 696.8MB
```



# Check S3: is anything there?

[Amazon S3](#) > [Buckets](#) > [\[bucket name\]-us-east-1](#) > [pgbackrest/](#) > [news-15/](#) > [backup/](#) > [news/](#)

news/

Objects

Properties

Objects (17) [Info](#)



Copy S3 URI

Copy URL

Download

Open

Delete

Actions

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permission.

Show versions

<input type="checkbox"/>	Name	Type	Last modified	Size
<input type="checkbox"/>	20240203-181003F_20240206-181003/	Folder	-	-
<input type="checkbox"/>	20240203-181003F_20240208-181002/	Folder	-	-
<input type="checkbox"/>	20240203-181003F/	Folder	-	-





# WAL archiving stats: throughput, failures

Edit

Overview

Split Graph NEW

Correlations

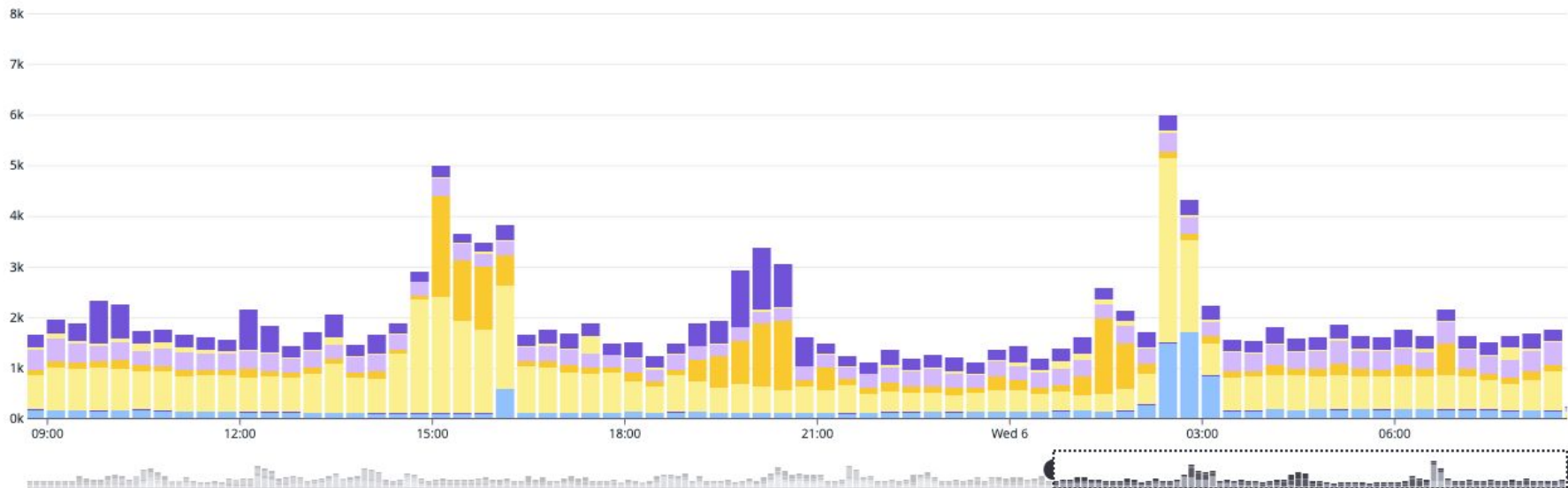
1d

Past 1 Day

Archiving throughput (WAL file count)

[Save to Dashboard](#)

[More...](#)





# WAL archiving stats: throughput, failures

Edit

Overview

Split Graph **NEW**

Correlations

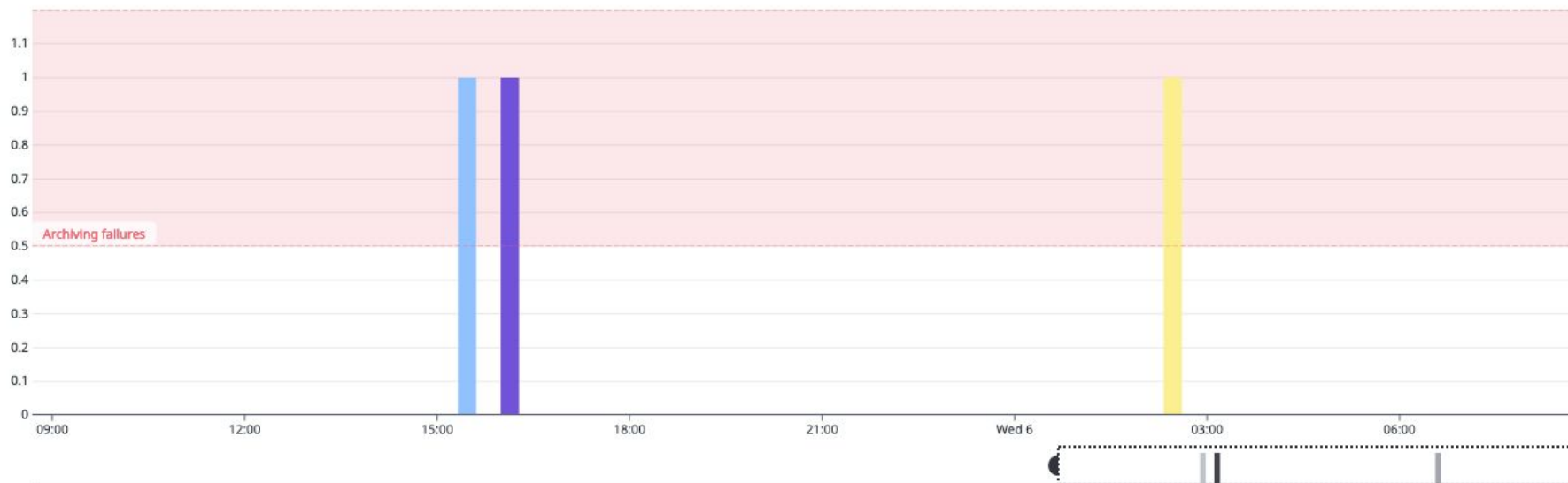
1d

Past 1 Day

Archive failed count

Save to Dashboard

More...





# WAL archiving stats: throughput, failures

Edit

Overview

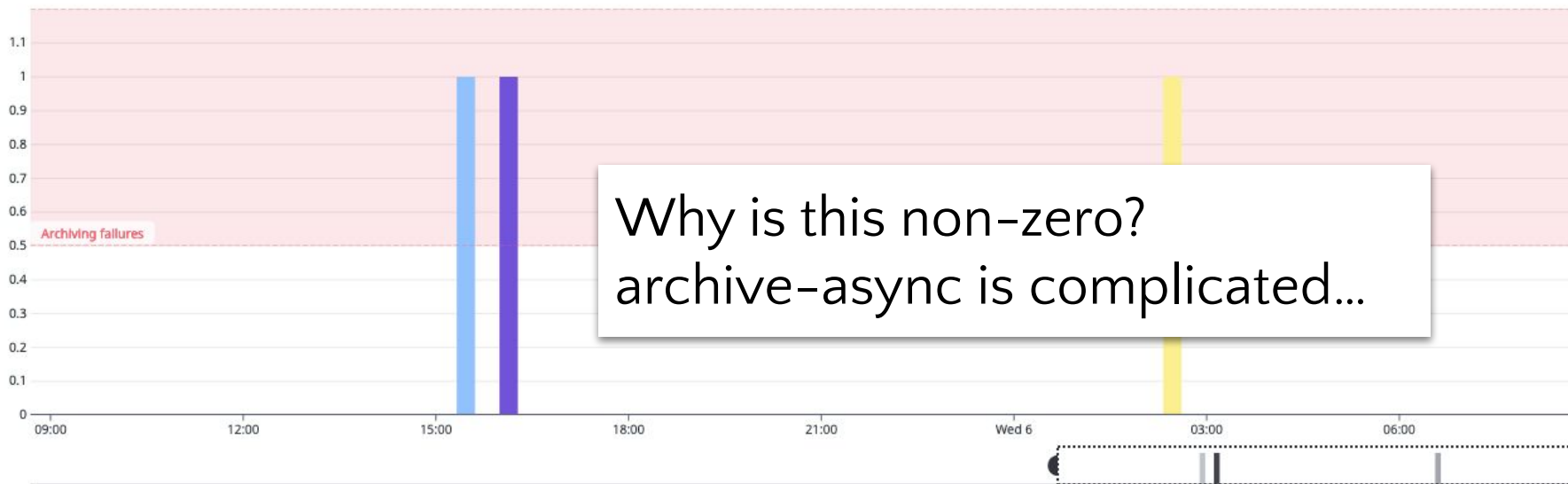
Split Graph **NEW**

Correlations

1d Past 1 Day

Archive failed count

Save to Dashboard More...



# Recap



# A

## Summary

- **Every time you need a replica, use your backups**
  - “Bring up the infrastructure” is what usually fails
- **Periodically test a cold-restore in QA/staging**
- **Visualize the restore process**
- **Make sure your monitoring pulls its weight**

# A

## Summary

- **Every time you need a replica, use your backups**
  - “Bring up the infrastructure” is what usually fails
- **Periodically test a cold-restore in QA/staging**
- **Visualize the restore process**
- **Make sure your monitoring pulls its weight**

# Questions?

<https://github.com/aristocrates>



# Appendix



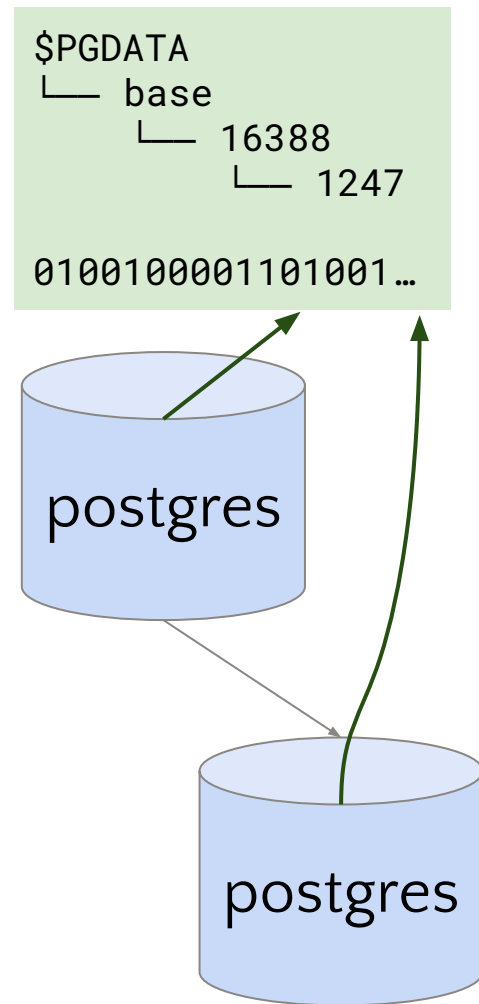
A

(There's definitely no time for this, but if you're reading this after the conference, enjoy!)

A

## Sidenote: streaming replication

- **This talk assumes some familiarity with:**
  - Streaming replication in postgres
    - “Binary compatibility”
    - Read-only replicas, HA replicas
  - The Write Ahead Log (WAL)
    - (at a high level)
- **Some resources:**
  - [pgBackRest User Guide](#)
  - [Dude, where's my byte? | ScaLE 17x](#)
    - [\(recording, youtube\)](#)



“Replication heartbeats”



## **A** replication\_heartbeats

- **Sometimes the built in Datadog metric has issues**
  - (Not always recognized until the first time a replica catches up)
- **So we have a secondary system to fill in the gaps**

## A replication\_heartbeats

```
CREATE TABLE public.replication_heartbeats (  
    created_at TIMESTAMP WITHOUT TIME ZONE PRIMARY KEY DEFAULT now()  
);
```

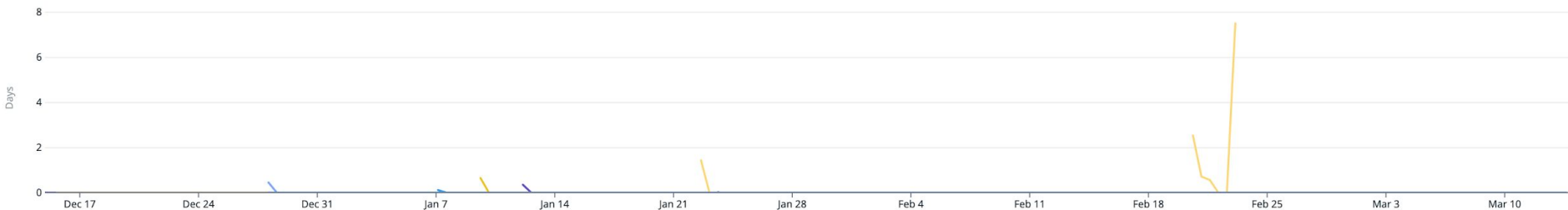
- **Cron job to insert the current time**
- **Metric: diff against replica system time**
- **Sloppiness aside...**
  - time zones
  - NTP point of failure
- **... it works pretty well in practice**



# replication\_heartbeats

Edit Overview Split Graph **NEW** Correlations

3mo Past 3 Months



## 1 Select your visualization

**Timeseries** Query Value Table Heatmap Scatter Plot Distribution Top List Host Map Change Geomap Tree Map Pie Chart

## 2 Graph your data

Edit JSON Share

[Graphing help](#)

a Metrics postgresql.replication\_delay from server\_type:postgres, \$stage, \$db, \$postgres\_cluster max by host

`</>` as...

b Metrics production.replication\_heartbeats\_delay\_minutes from server\_type:postgres, \$stage, \$db, \$postgres\_cluster max by host

`</>` as...



# replication\_heartbeats

Edit Overview Split Graph **NEW** Correlations

3mo Past 3 Months



## 1 Select your visualization

**Timeseries** Query Value Table Heatmap Scatter Plot Distribution Top List Host Map Change Geomap Tree Map Pie Chart

## 2 Graph your data

Edit JSON Share

[Graphing help](#)

a Metrics postgresql.replication\_delay from server\_type:postgres, \$stage, \$db, \$postgres\_cluster max by host

</> as...

b Metrics production.replication\_heartbeats\_delay\_minutes from server\_type:postgres, \$stage, \$db, \$postgres\_cluster max by host

</> as...