Deconstructing an Abstraction to Reconstruct an Outage

sinjo.dev

A familiar story


single-product
do_action('woocommerce_before_single_product');
?>
<div itemscope itemtype="http://schema.org/Product" id="product-<?php the_ID(); ?>" <?php post_class(); ?>>

<?php
/**
* woocommerce_show_product_images hook
*
* @hooked woocommerce_show_product_sale_flash - 10
* @hooked woocommerce_show_product_images - 20
*/
do_action('woocommerce_before_single_product_summary');
?>
<!-- end large-6 - product-gallery -->

<?php
/**
* woocommerce_single_product_summary hook
*
* @hooked woocommerce_template_single_title - 5
* @hooked woocommerce_template_single_price - 10
* @hooked ProductShowReviews() (inc/template-tags.php) - 15
* @hooked woocommerce_template_single_excerpt - 20
* @hooked woocommerce_template_single_add_to_cart - 30
* @hooked woocommerce_template_single_meta - 40
* @hooked woocommerce_template_single_sharing - 50
*/
do_action('woocommerce_single_product_summary');
?>

<!-- end product-info large-4 -->
<!-- row -->

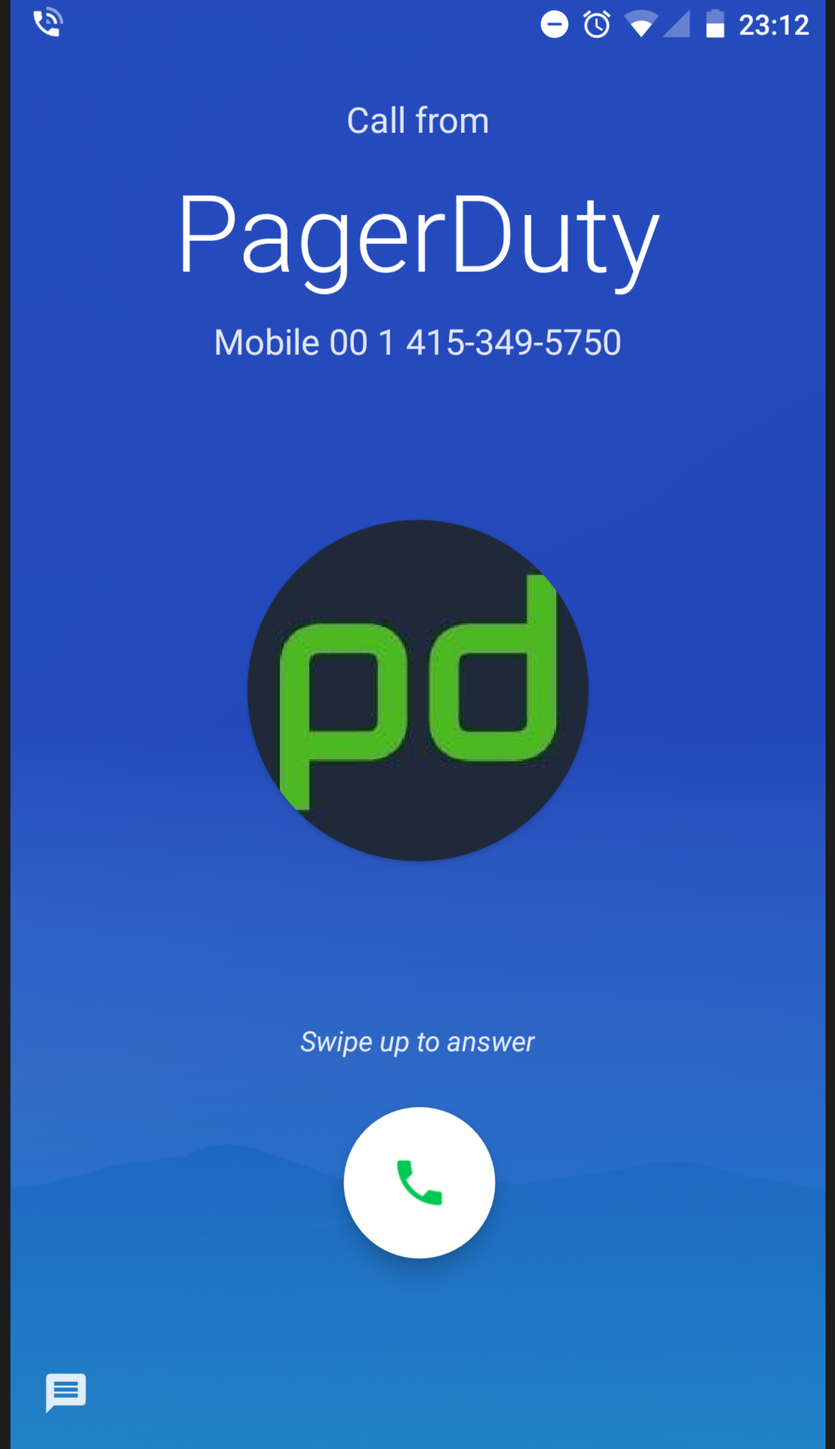
Ligne 24, colonne 34 — 117 lignes
INS

foocorp / api Public

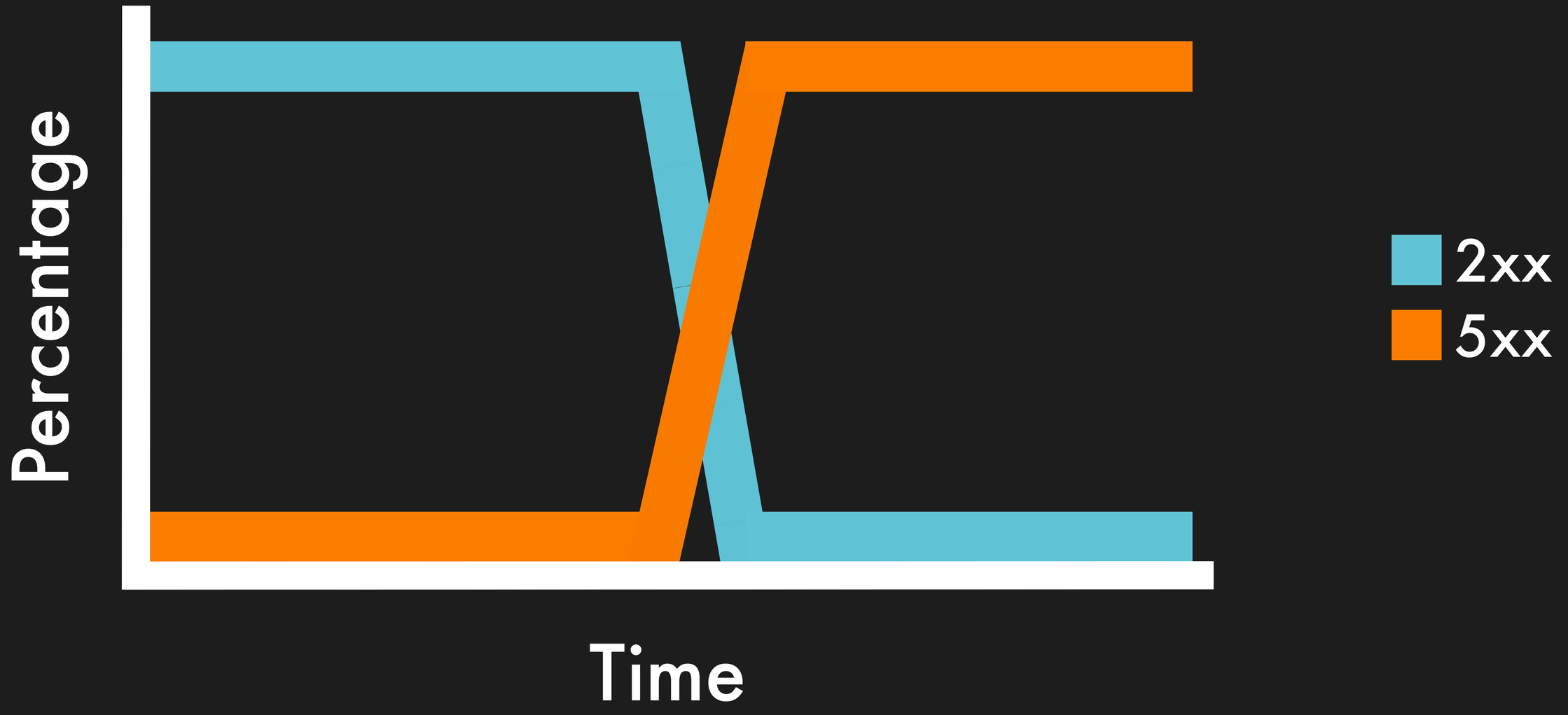
Code | Issues 14 | Pull requests 2 | Discussions | Actions | Projects | Security | Insights

Add checkout endpoint #279

Open **Sinjo** wants to merge 1 commit into `main` from `add-checkout-endpoint`


23:12
Call from
PagerDuty
Mobile 00 1 415-349-5750
pd
Swipe up to answer

API response status



```
DB::ConnectionFailure – could
not connect to server:
Connection refused
```


40/0.65
160/0.17

Hi

sinjo.dev

sinjo.dev

Infra Engineer

Databases & Distributed Systems

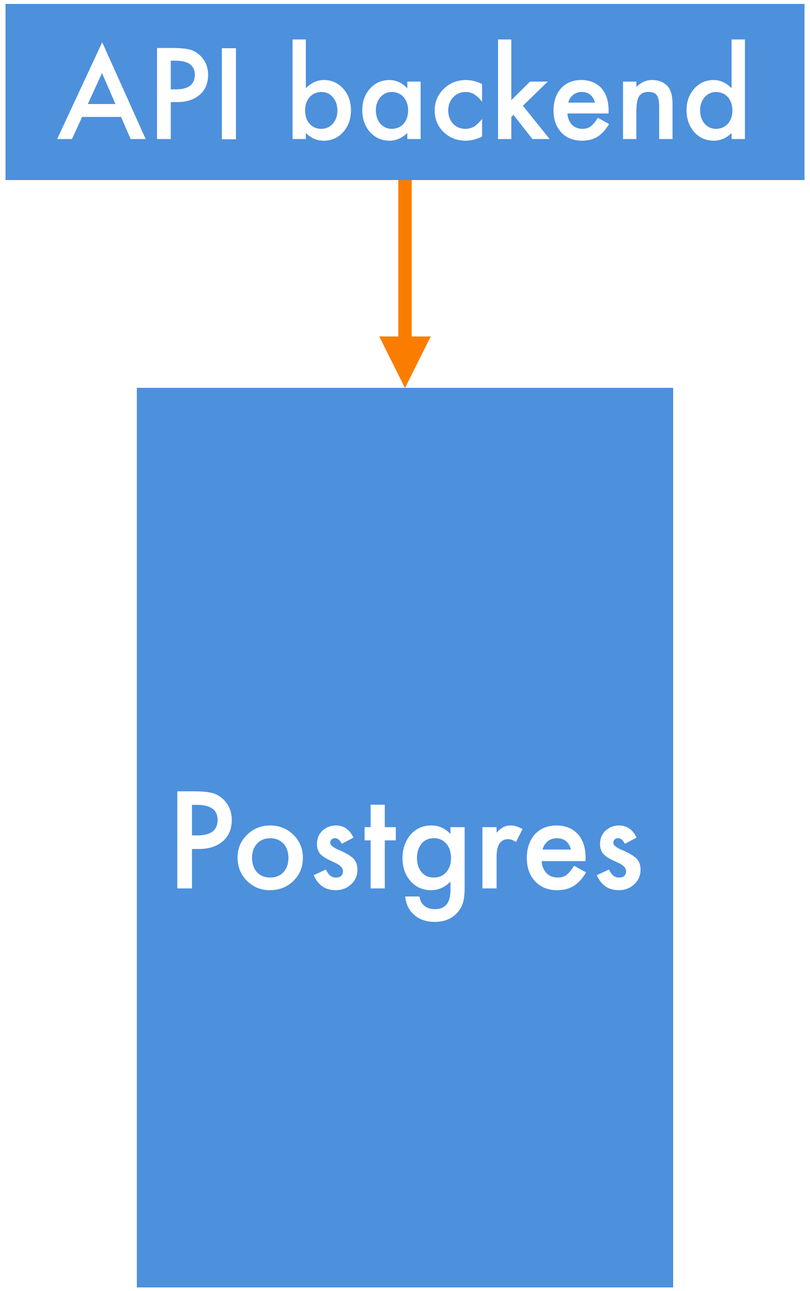
PlanetScale

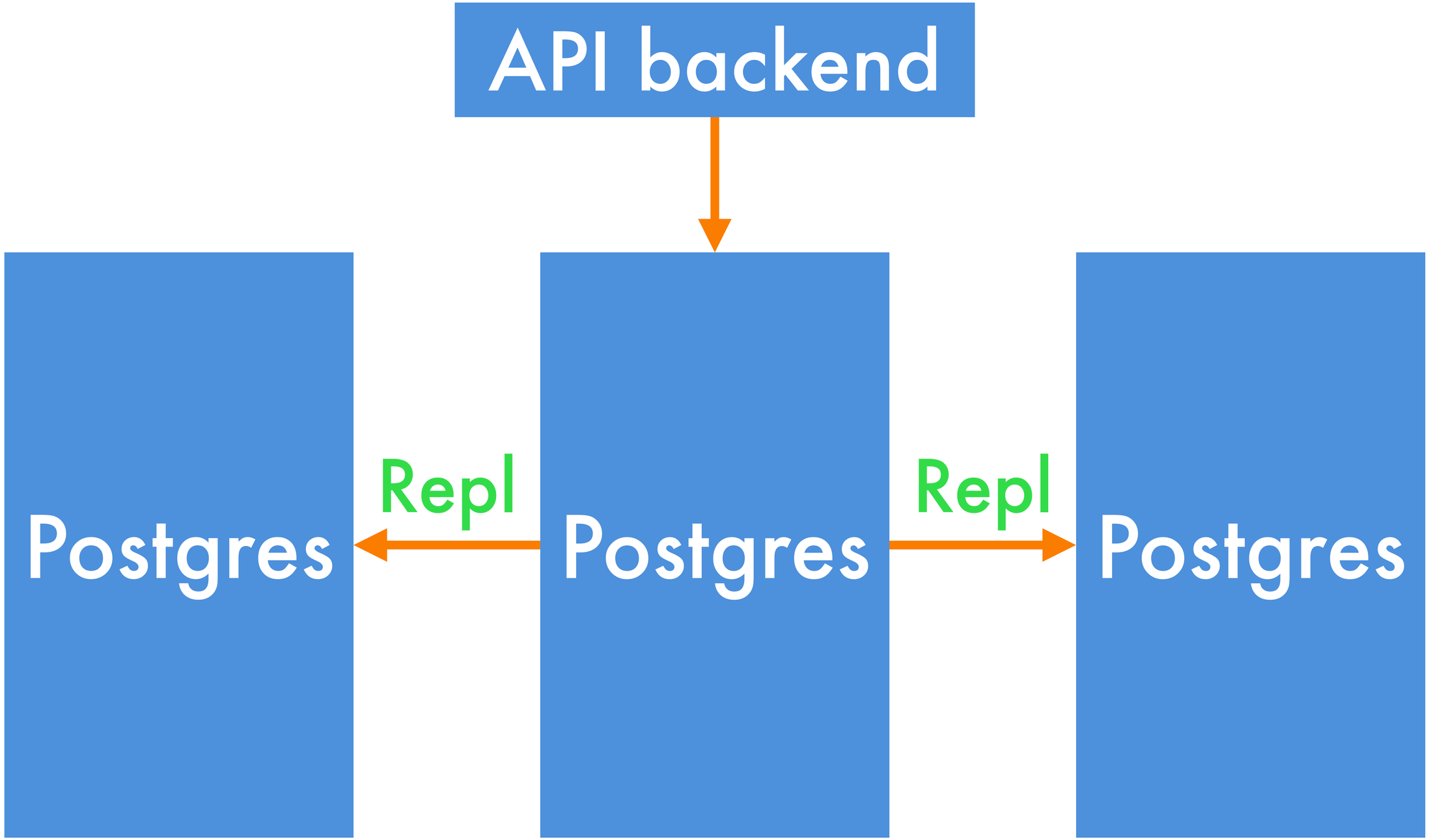
GoCardless

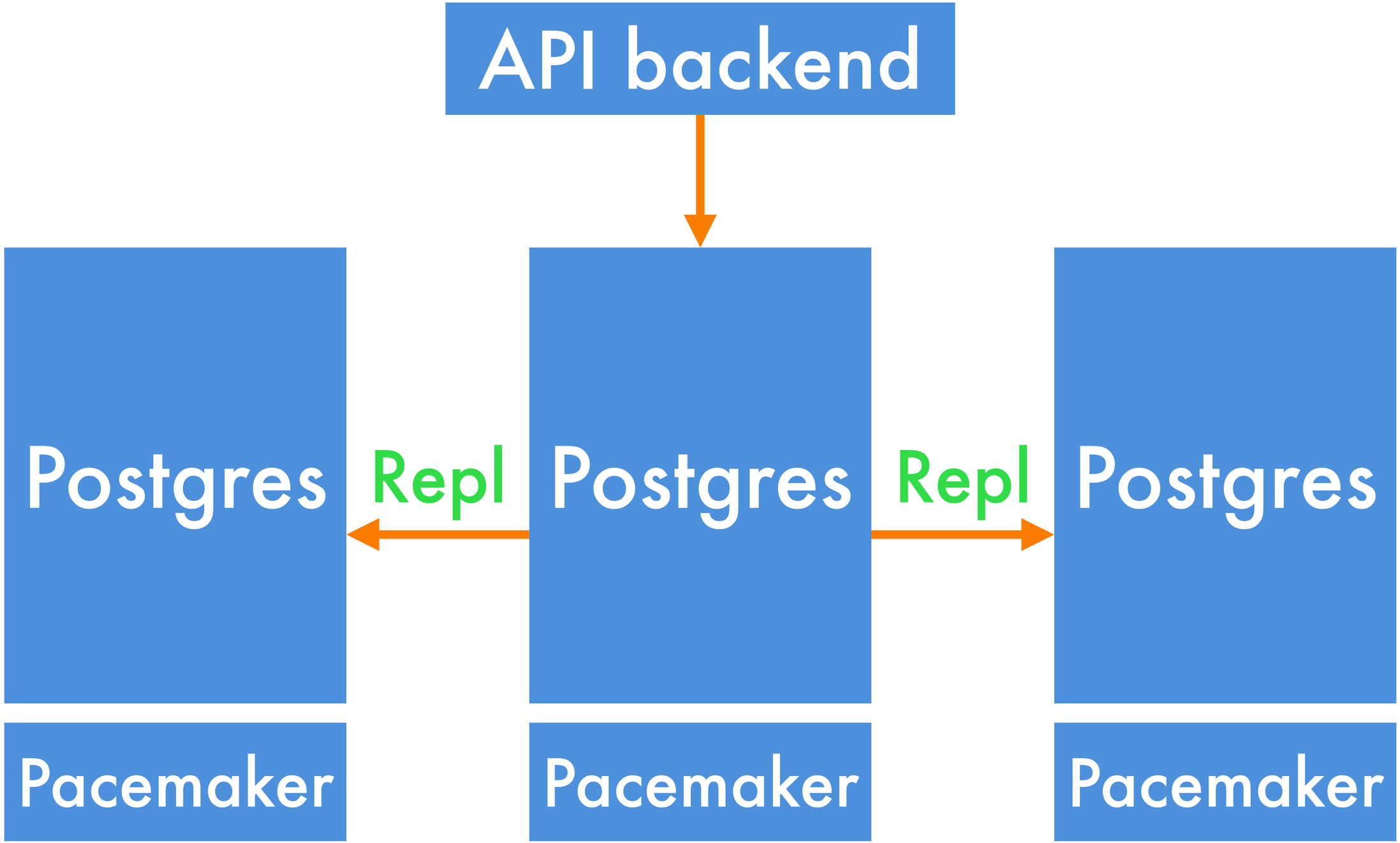
Deconstructing an Abstraction to Reconstruct an Outage

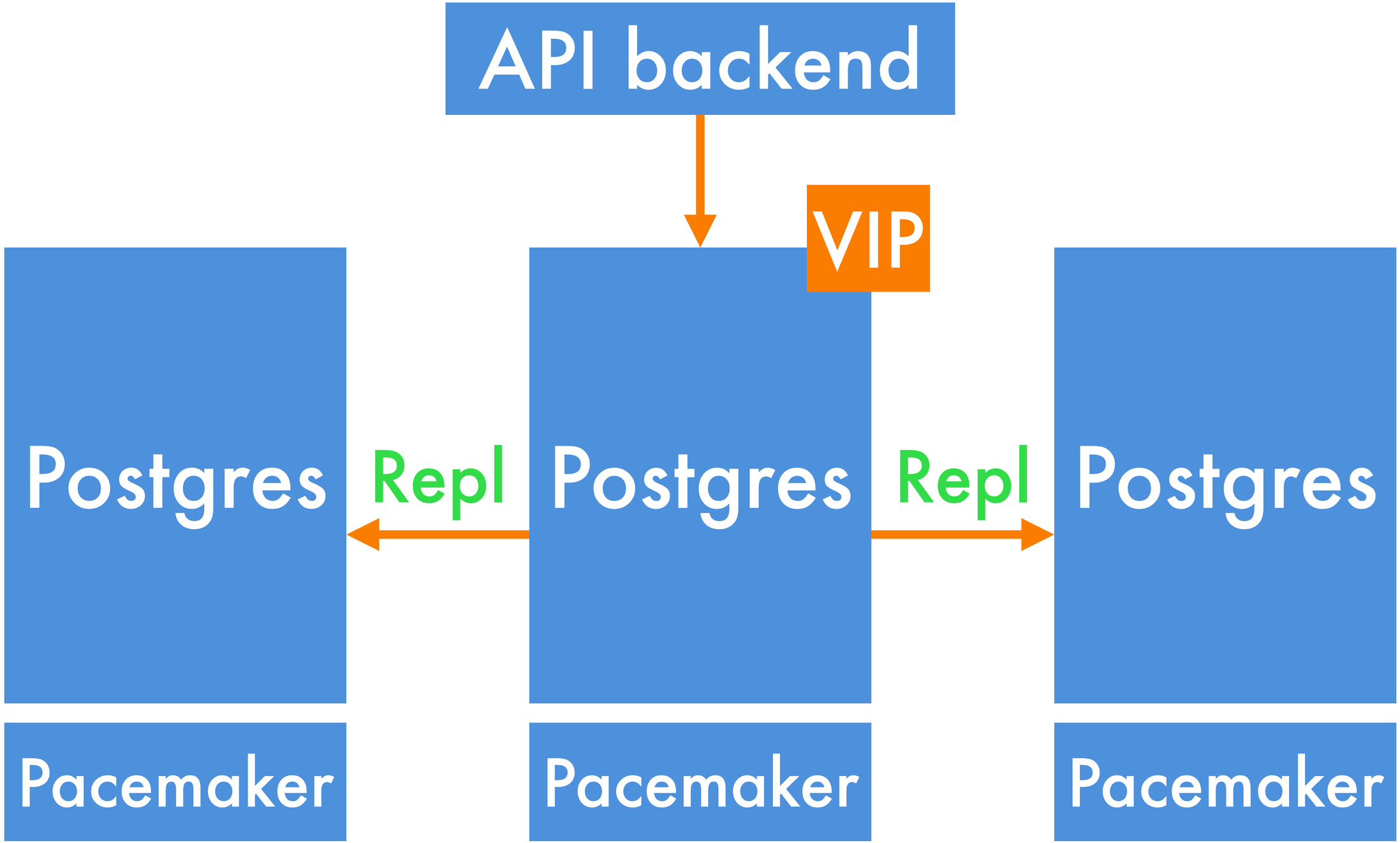
sinjo.dev

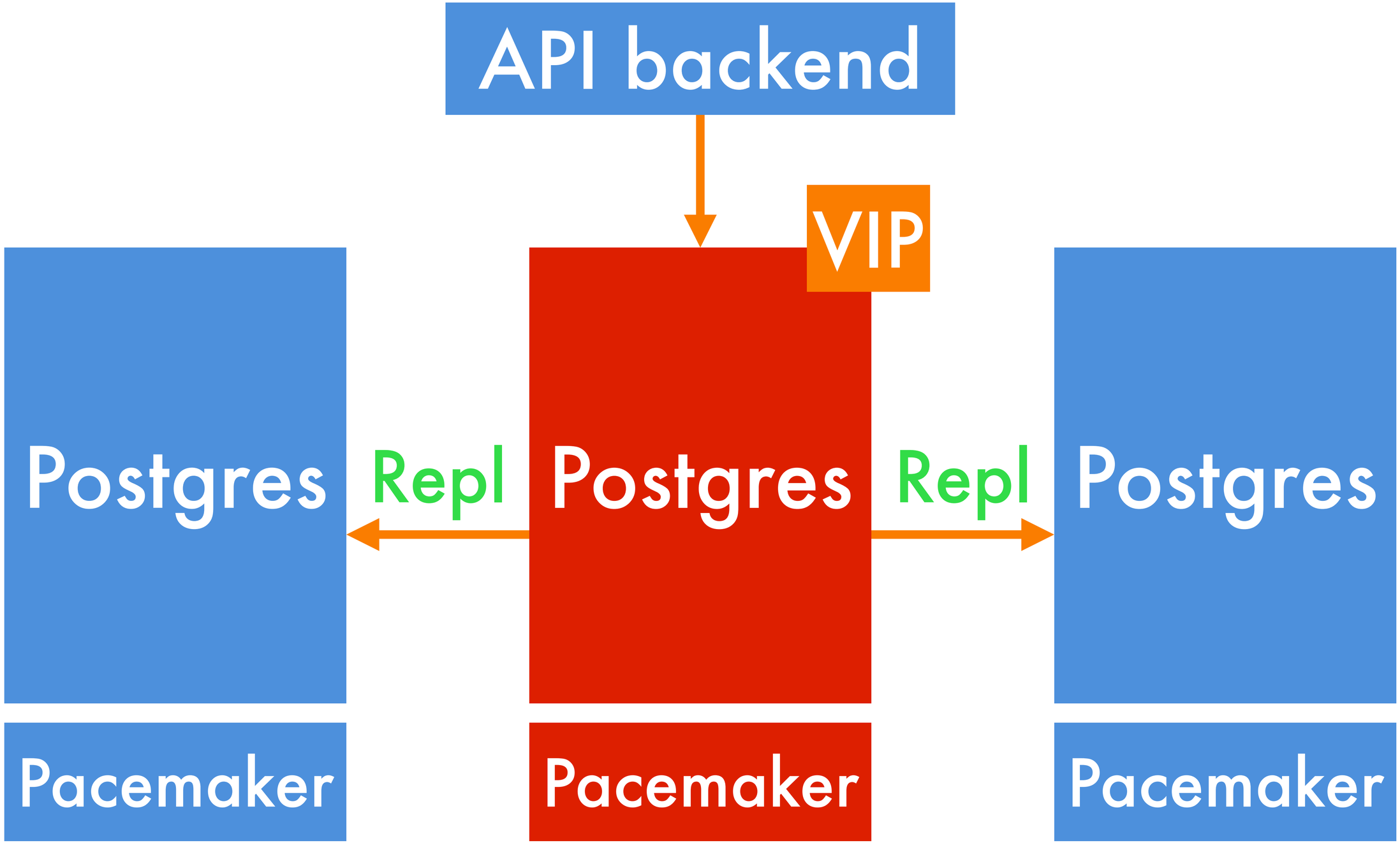
First:
Our cluster setup

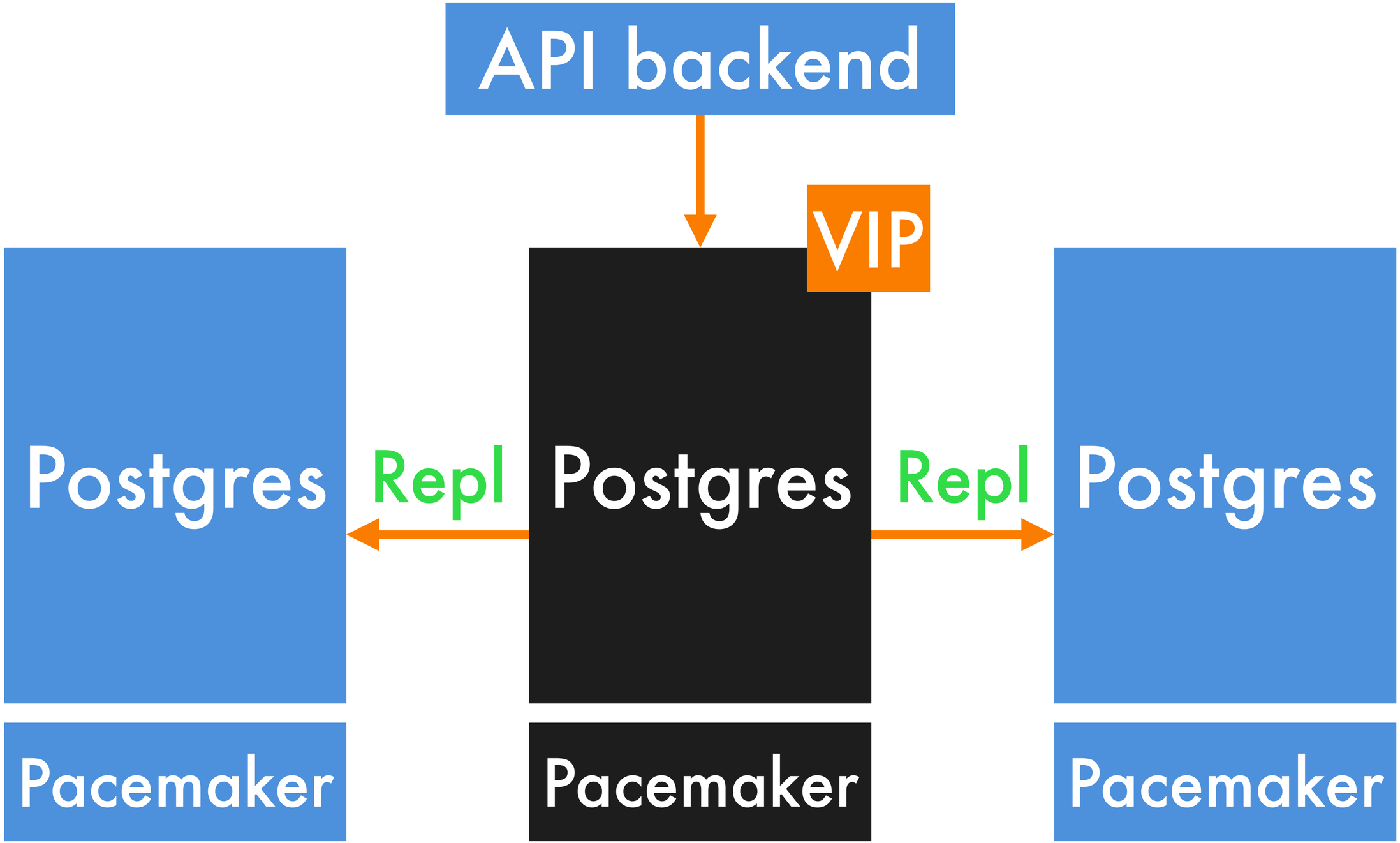

API backend
Postgres

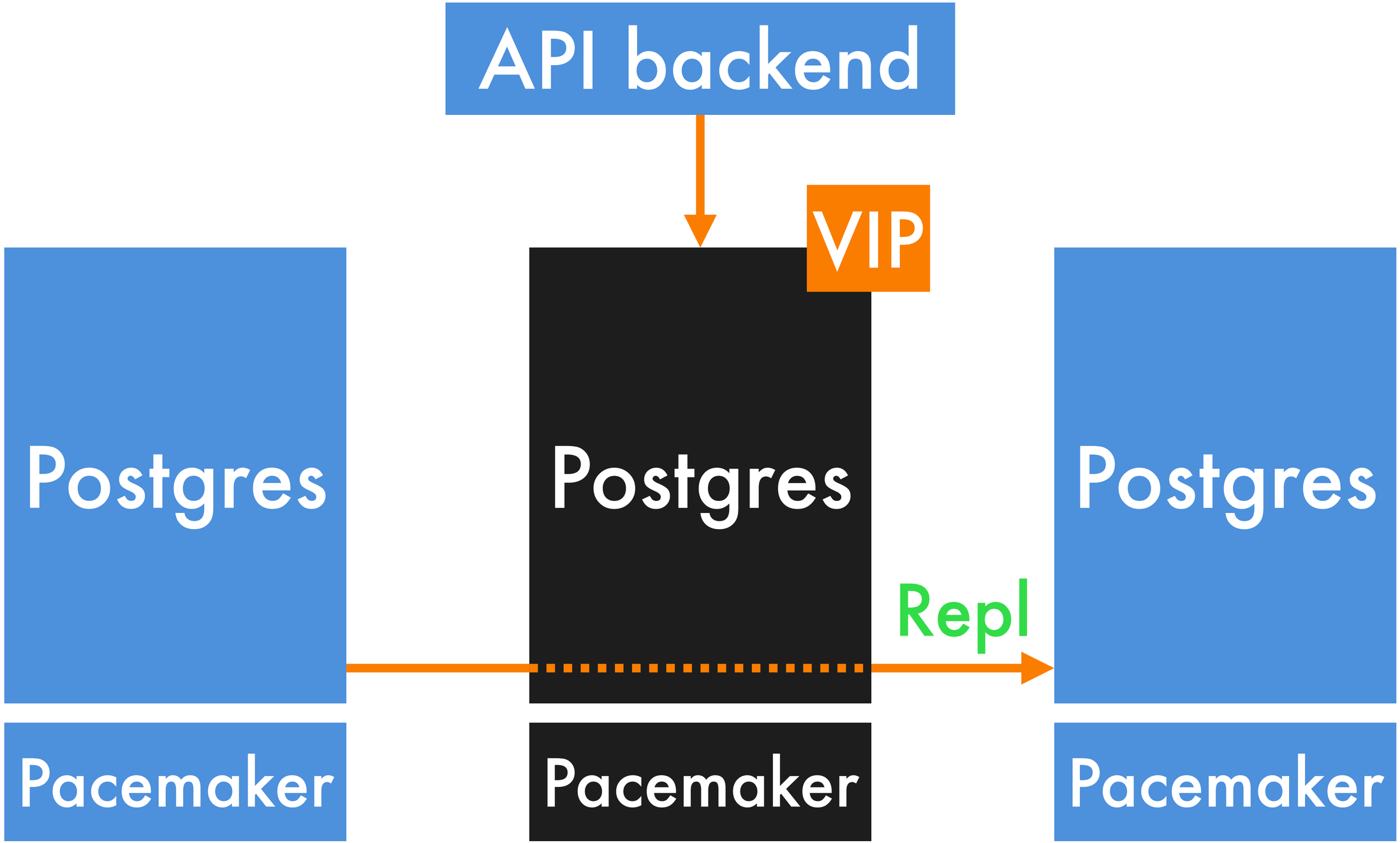

API backend
Postgres
Repl
Postgres
Repl
Postgres

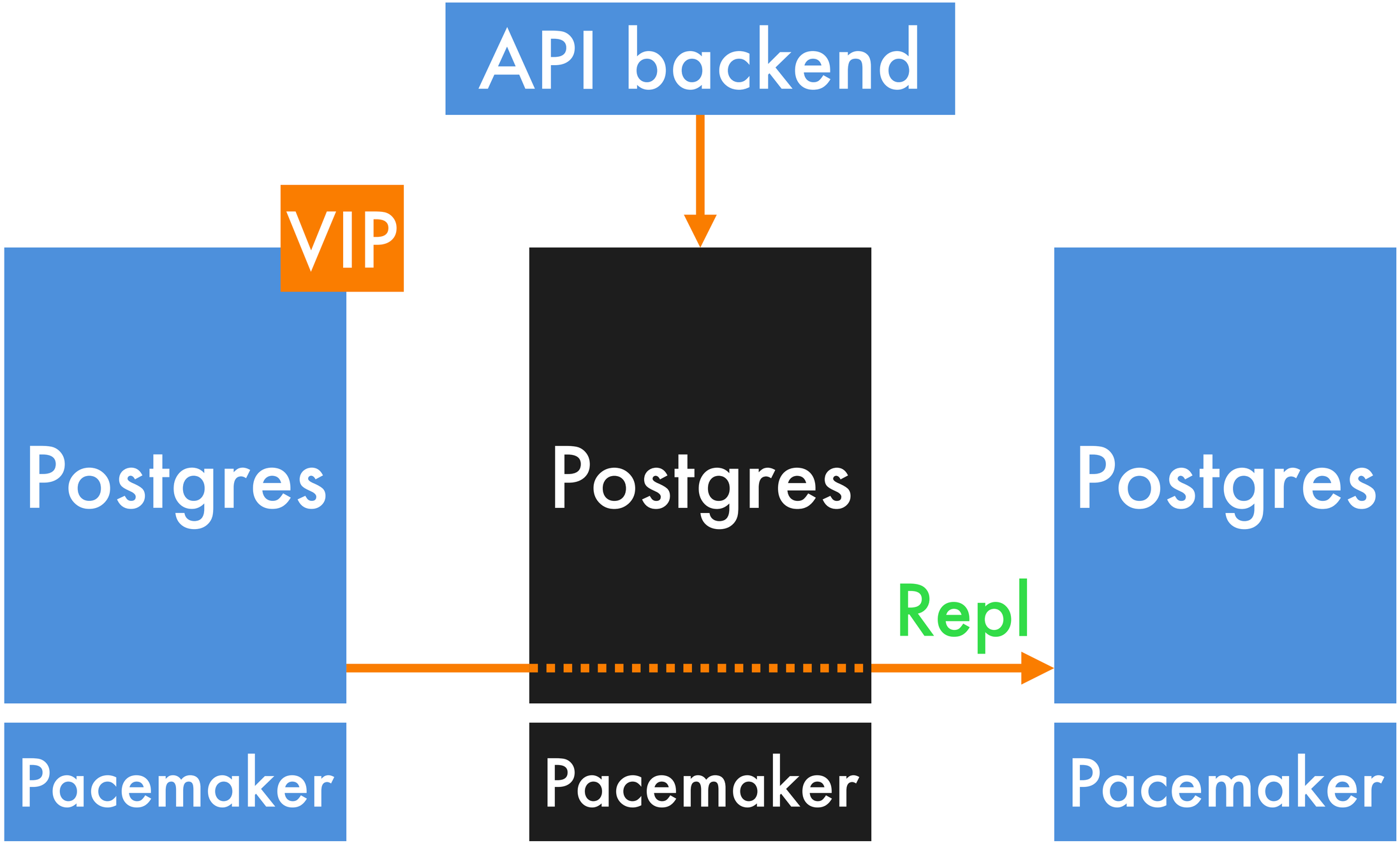

API backend
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker

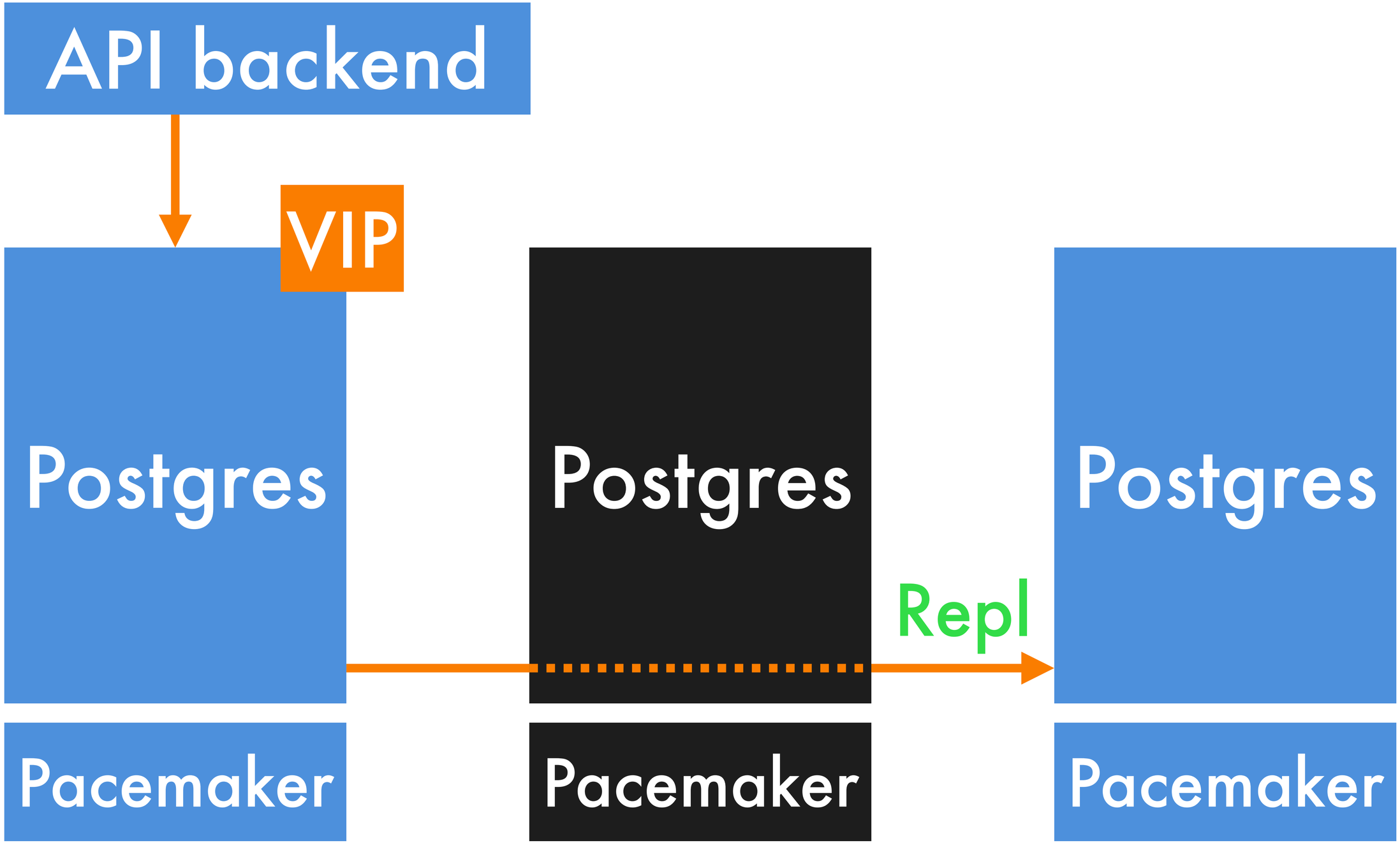

API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Postgres
Postgres
Repl
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Postgres
Postgres
Repl
Pacemaker
Pacemaker
Pacemaker

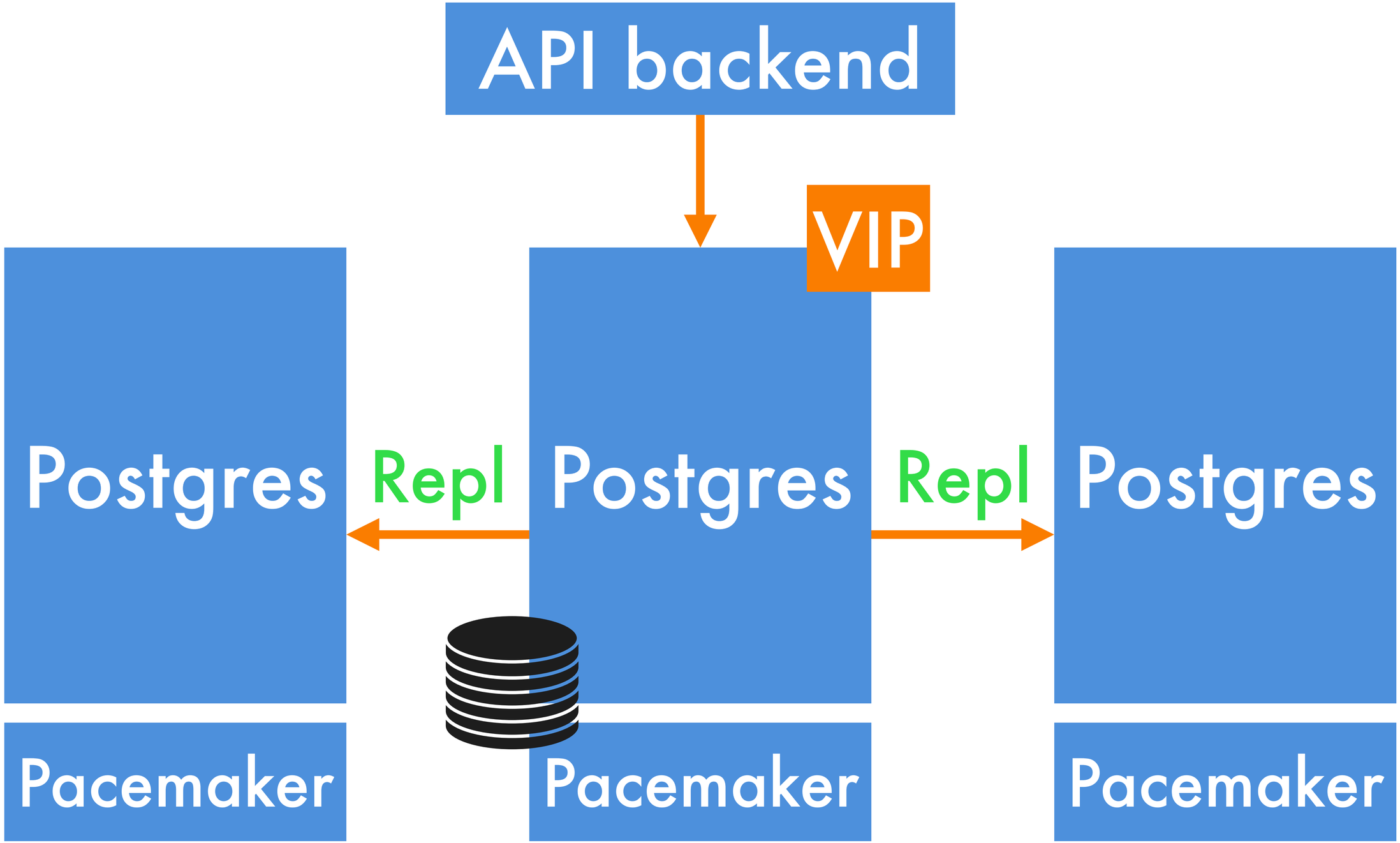

API backend
VIP
Postgres
Postgres
Postgres
Repl
Pacemaker
Pacemaker
Pacemaker

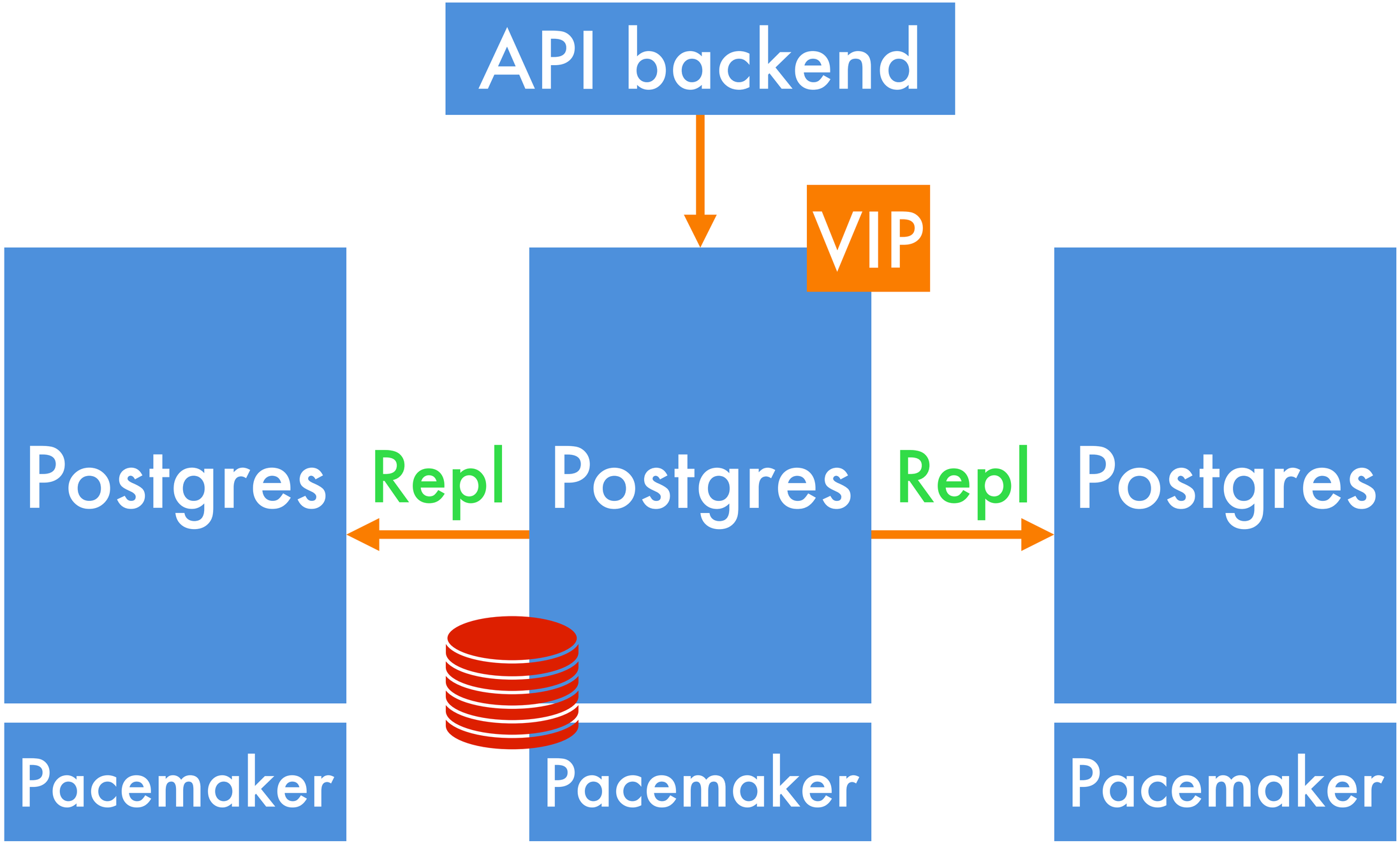
Note:
One replica always synchronous

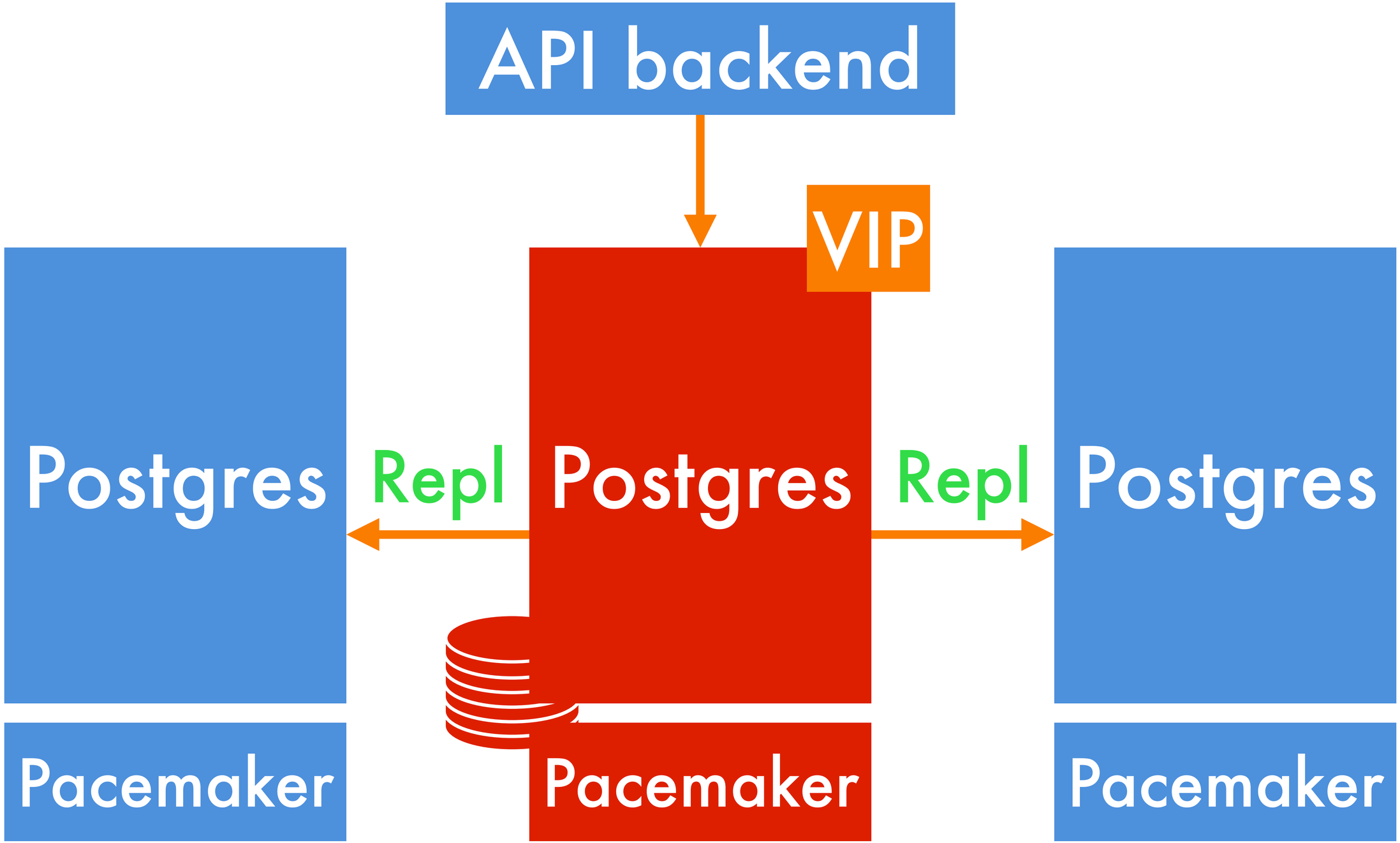
So...

So...

Unfortunately...


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker

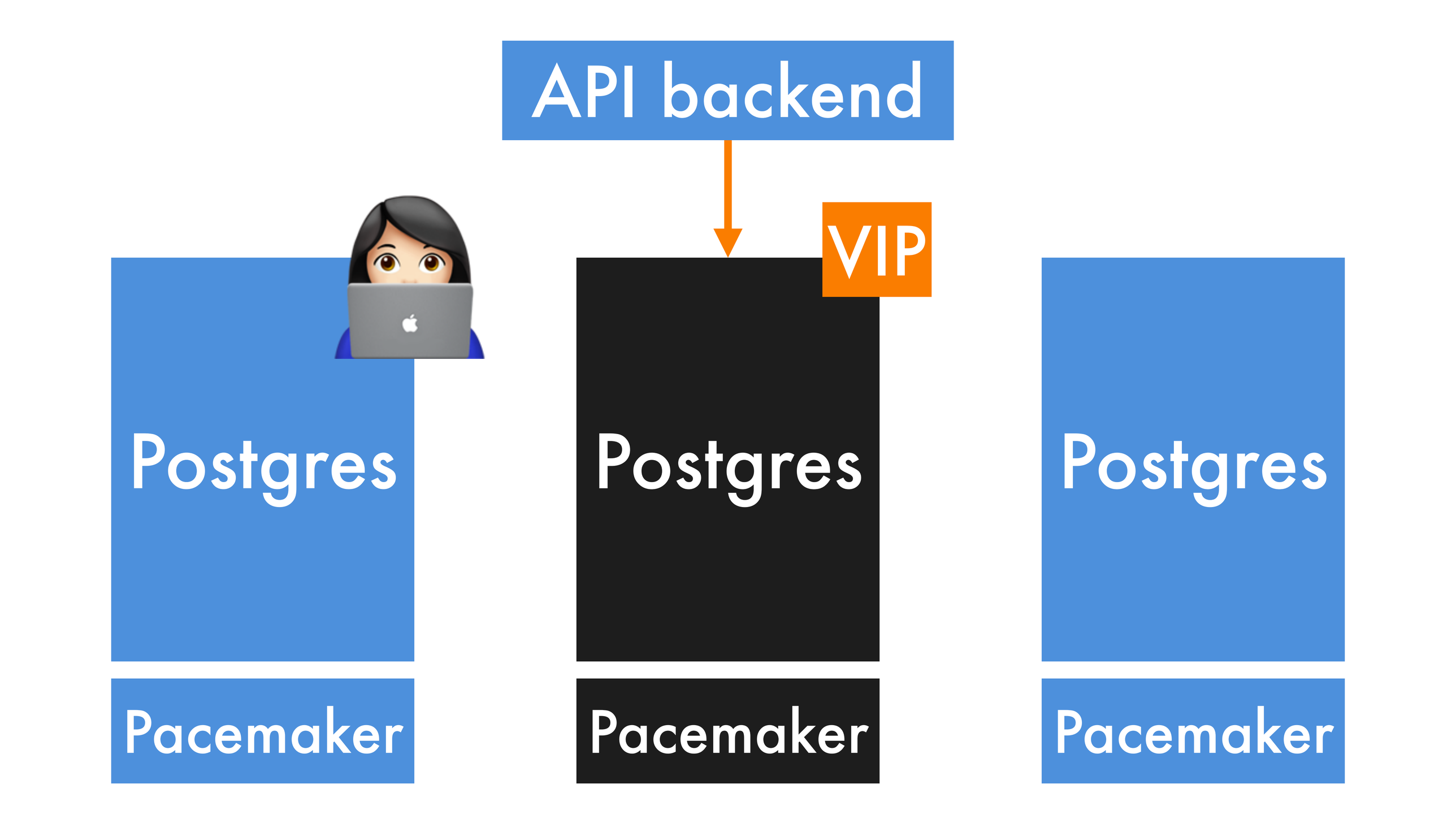
Except it

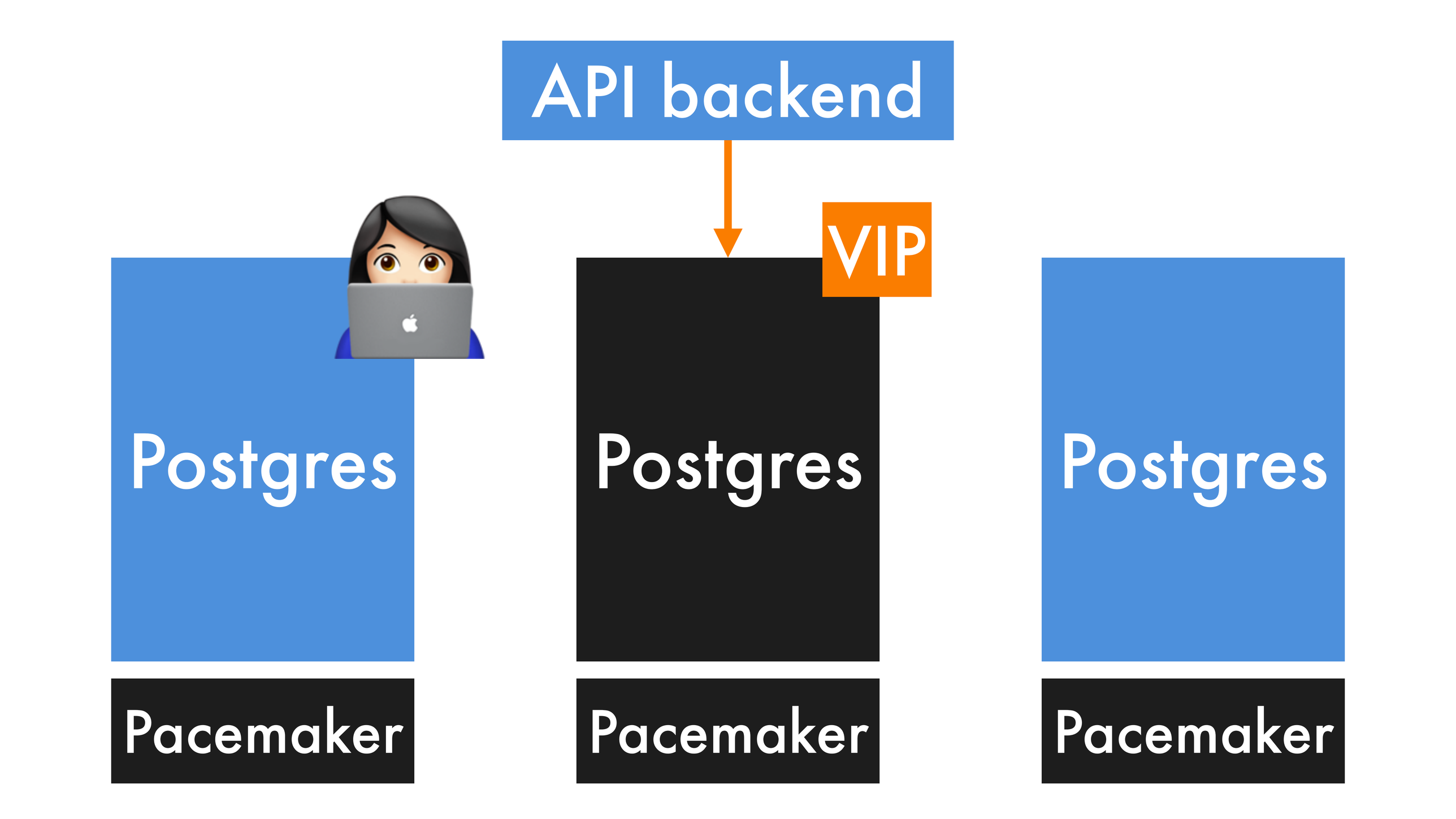
didn't

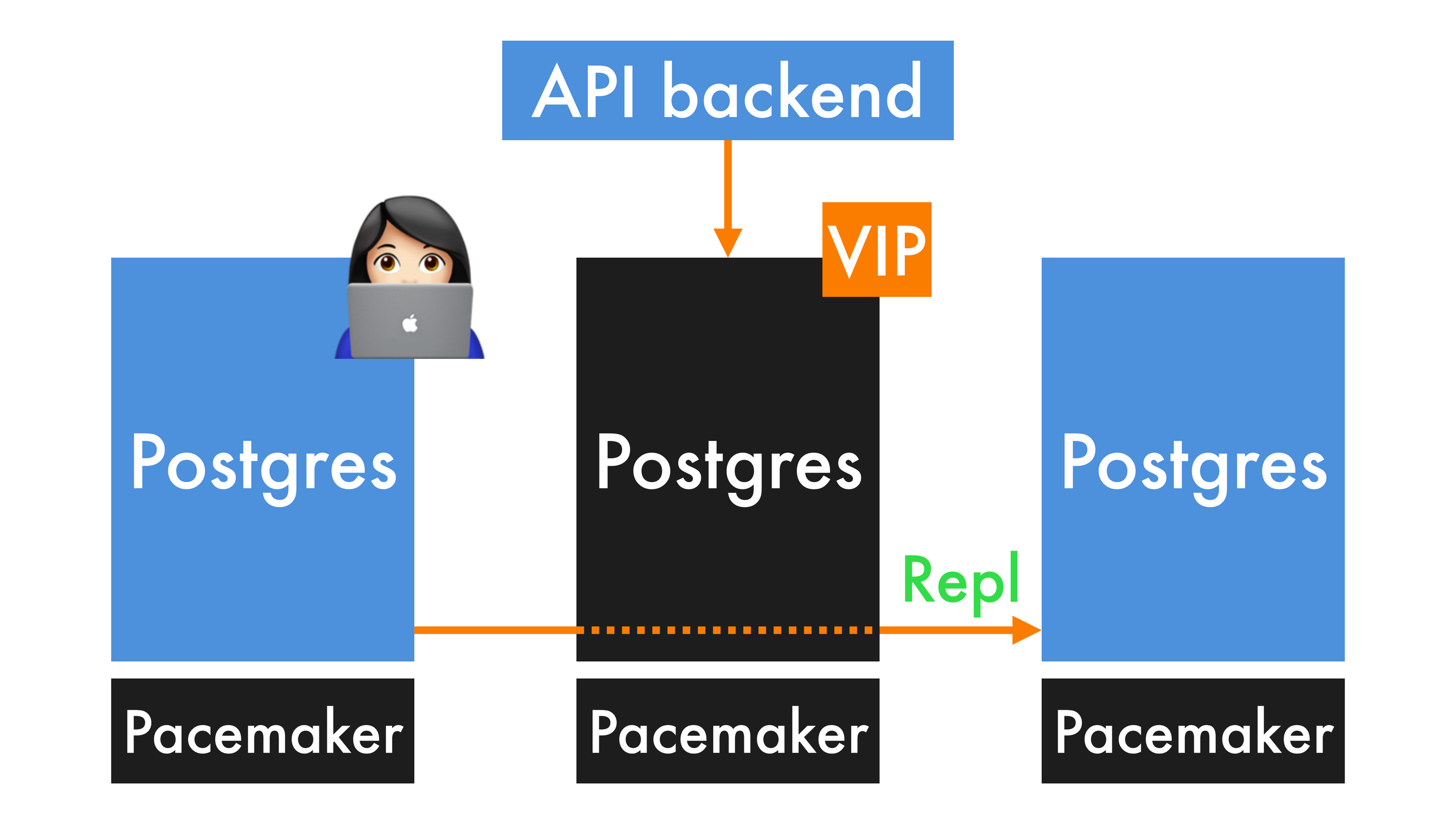
Our API

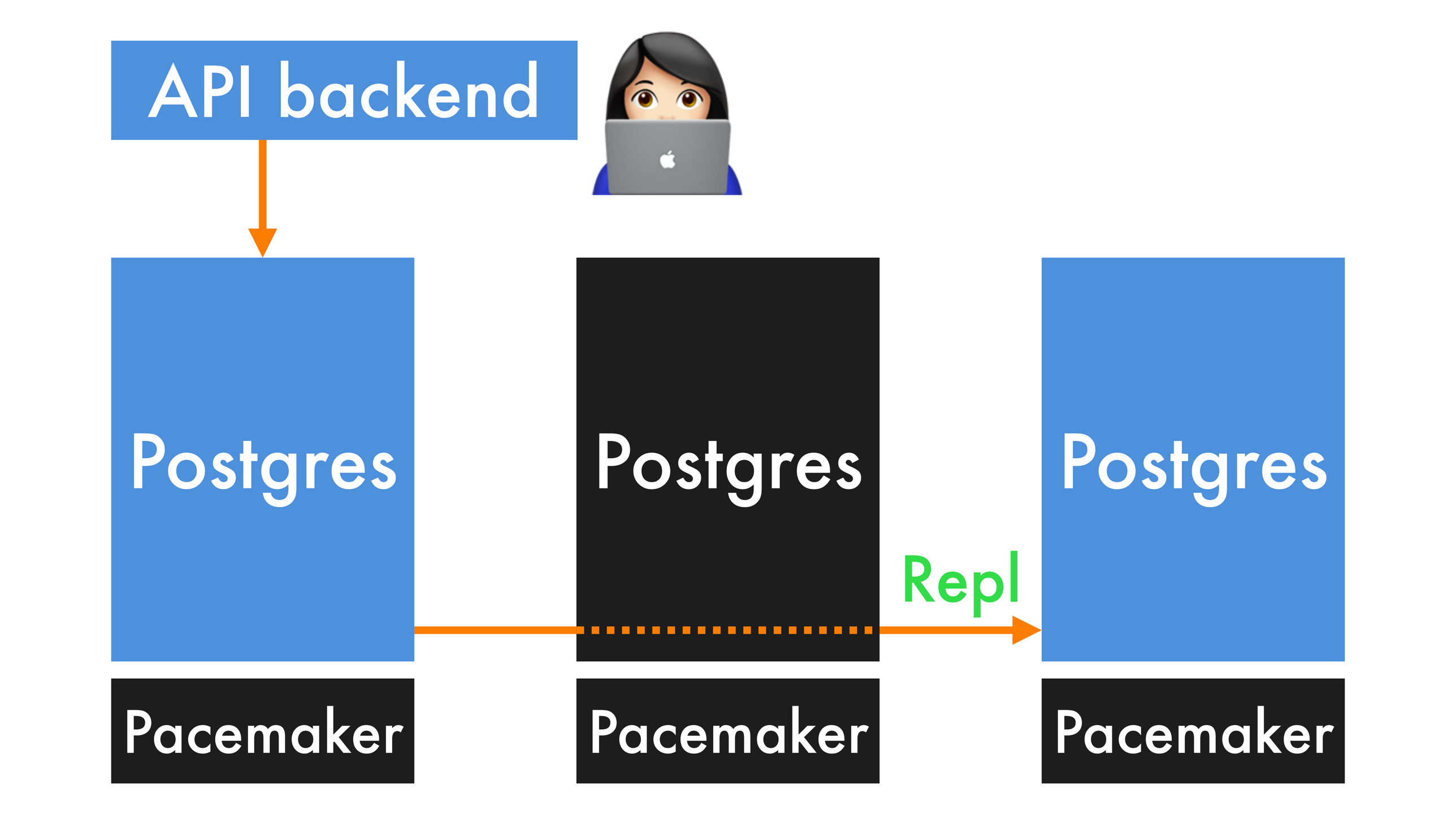
was down

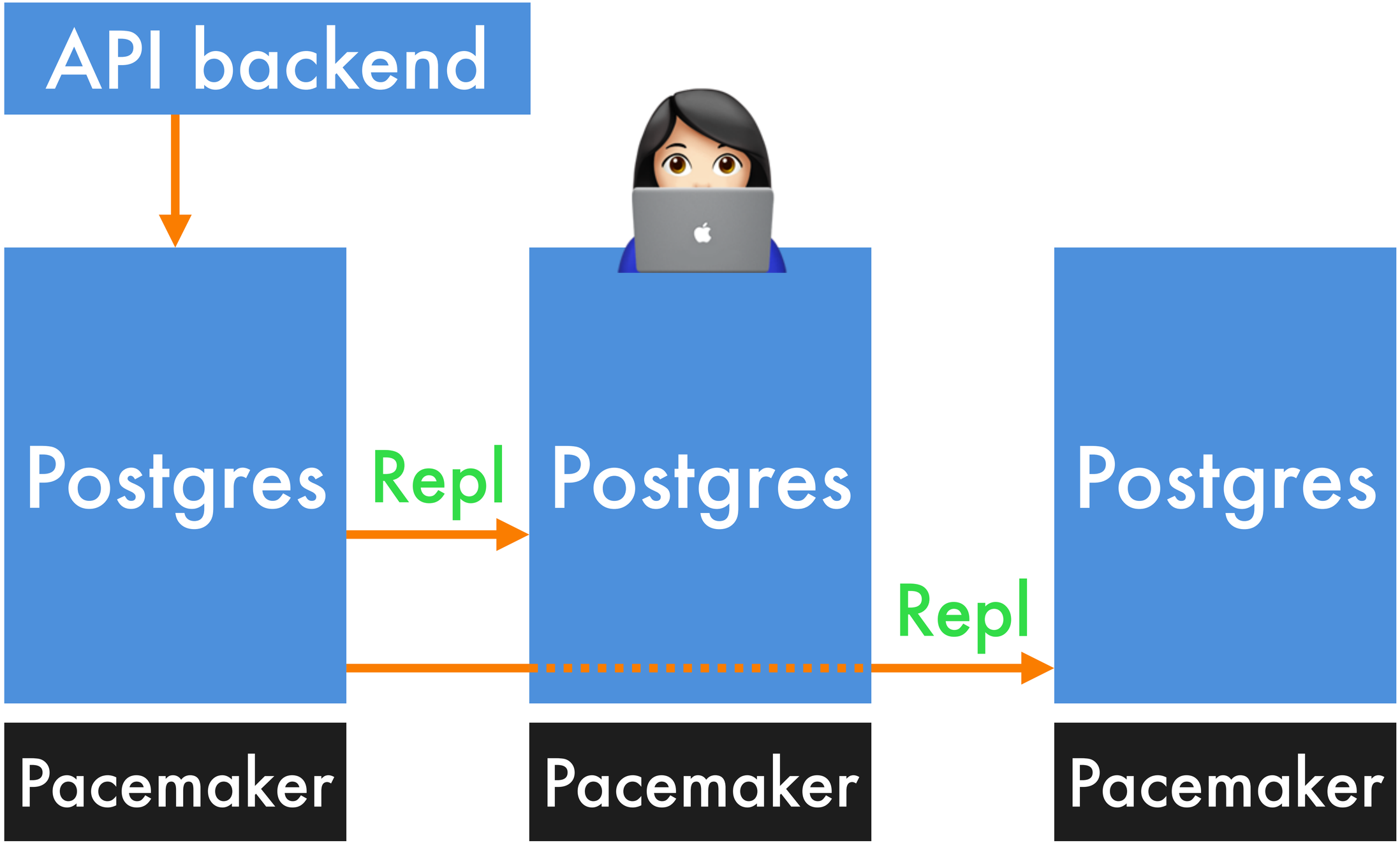
Fallback:
fully manual setup


API backend
VIP
Postgres
Postgres
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Postgres
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Postgres
Postgres
Repl
Pacemaker
Pacemaker
Pacemaker


API backend
Postgres
Postgres
Postgres
Repl
Pacemaker
Pacemaker
Pacemaker


API backend
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker

We're safe,
for now...

But only one failure away from downtime

Mission:

Recreate the outage

There's a lot

We'll go step-by-step

1. RAID array loses disks

1. RAID array loses disks
2. Kernel sets filesystem read-only

1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure

1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash

1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica

Suspicious log on synchronous replica

```
2023-02-24 17:23:01 GMT LOG: restored log file
"000000020000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length
at 0/3000180
```

1. RAID array loses disks
2. Kernel sets filesystem read-only
3. Pacemaker detects primary failure
4. Synchronous replica crash
5. Suspicious log on synchronous replica

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

~~5. Suspicious log on synchronous replica~~

Everyone's favourite fault-injection tool

You know it well...

KILL(1) General Commands Manual KILL(1)

NAME
kill – terminate or signal a process

SYNOPSIS
kill [-s signal_name] pid ...
kill -l [exit_status]
kill -signal_name pid ...
kill -signal_number pid ...

DESCRIPTION
The kill utility sends a signal to the processes specified by the pid operands.

Only the super-user may send signals to other users' processes.

The options are as follows:


docker

```
# on primary – hard kill
kill -SIGKILL <main_pid>
```

```
# on primary – hard kill
kill -SIGKILL <main_pid>

# on synchronous replica – subprocess crash
kill -SIGABRT <subprocess_pid>
```

We kept our expectations low...

...which was

the right

choice

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

~~5. Suspicious log on synchronous replica~~

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

Suspicious log on synchronous replica

```
2023-02-24 17:23:01 GMT LOG: restored log file
"000000020000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length
at 0/3000180
```

What do we mean by "log"?

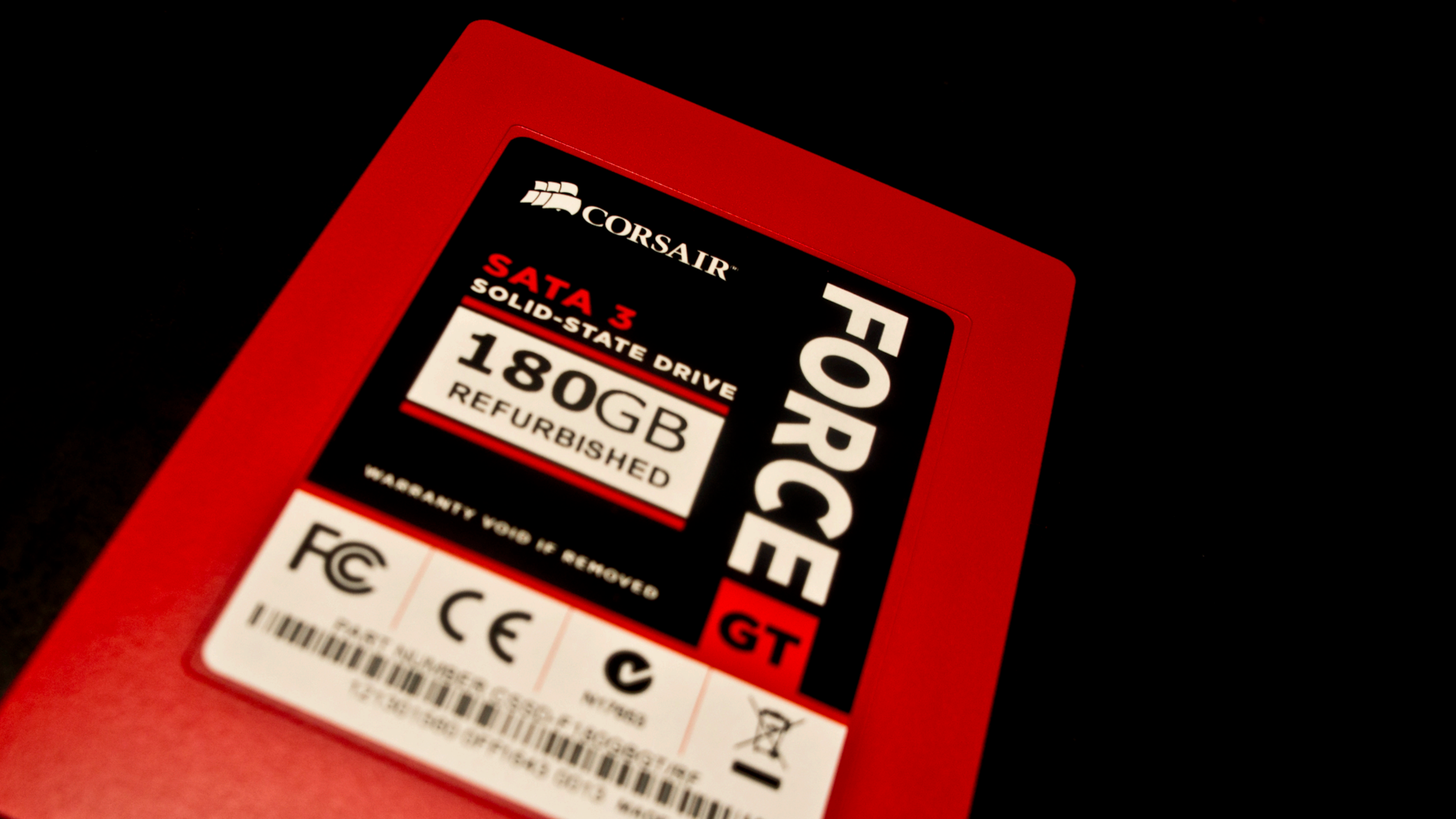
What we normally mean by logs

```
[2023-02-26 23:02:37Z] GET / - 200
[2023-02-26 23:02:49Z] GET /favicon.ico - 200
[2023-02-26 23:02:52Z] POST /login - 200
[2023-02-26 23:33:52Z] POST /posts - 201
[2023-02-26 23:33:57Z] GET /posts/binary-logs-talk - 200
```

A different kind of log: binary logs

Some extremely boring SQL

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```


CORSAIR
SATA 3
SOLID-STATE DRIVE
180GB
REFURBISHED
WARRANTY VOID IF REMOVED
FORCE
GT

Warning:

simplifying lie ahead

A different kind of logs

(if they were textual)

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```

```
Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
```

Postgres calls these "Write Ahead Logs" (WALs)

But why bother doing that?

Crash safety

Index

id
1
2

Table

id	username
1	codd
2	lovelace

Index

id
1
2

Table

id	username
1	codd
2	lovelace
3	turing

Index

id
1
2


Table

id	username
1	codd
2	lovelace
3	turing

Index

id
1
2
???

Table

id	username
1	codd
2	lovelace
3	turing

We can replay this operation

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```

```
Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
```

Index

id
1
2
???

Table

id	username
1	codd
2	lovelace
3	turing

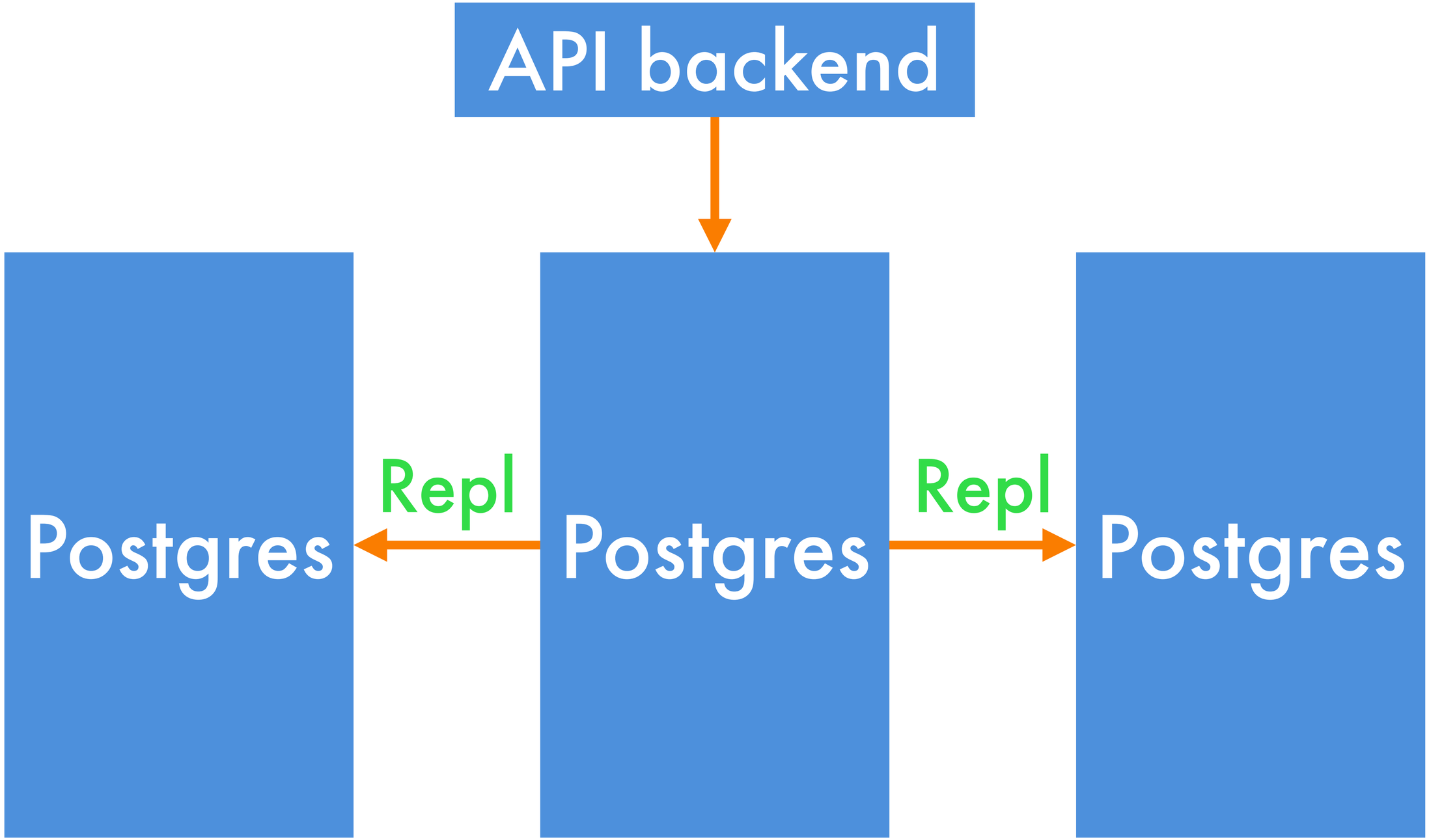
Index

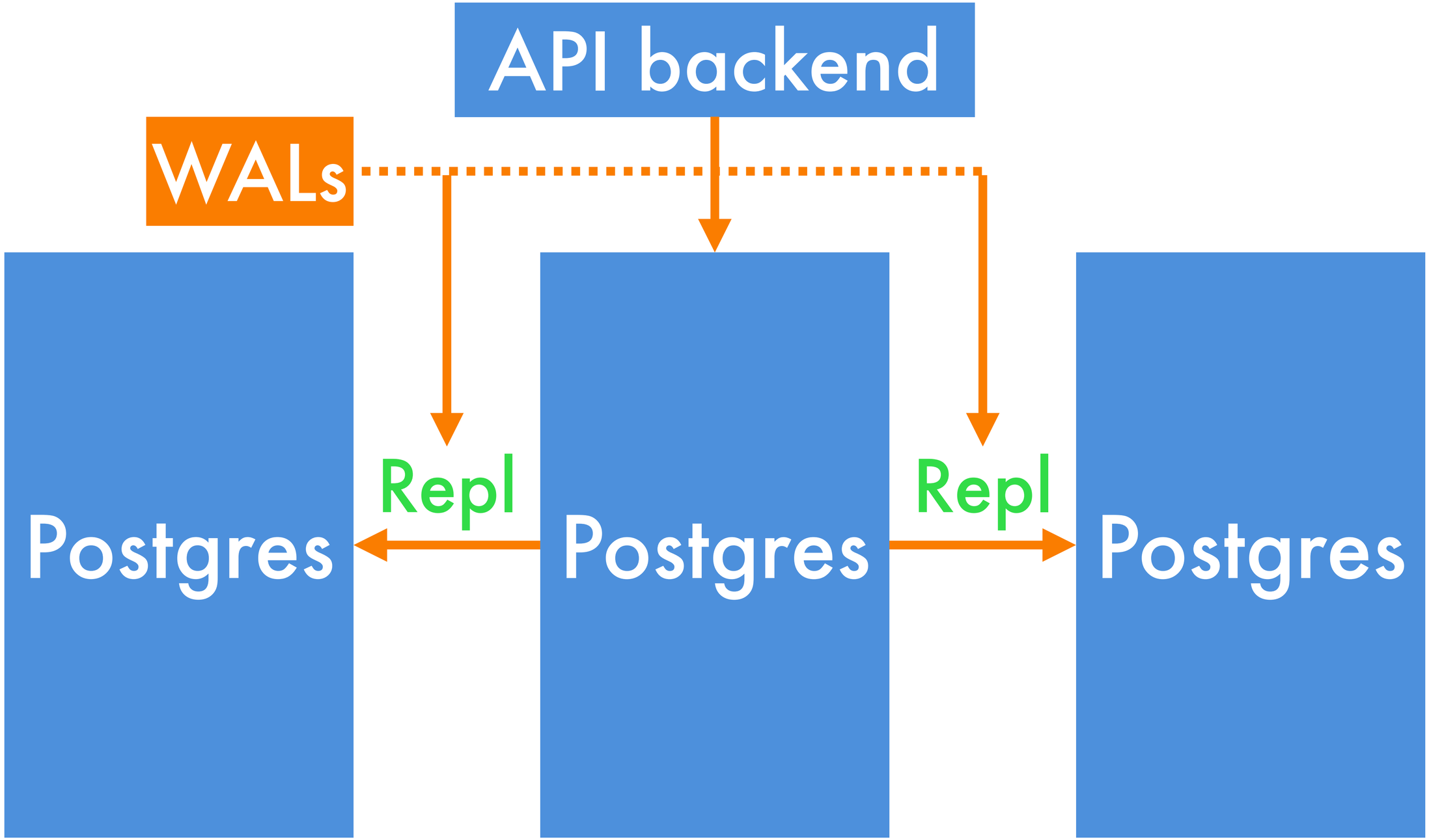
id
1
2
3

Table

id	username
1	codd
2	lovelace
3	turing

Also:
replication

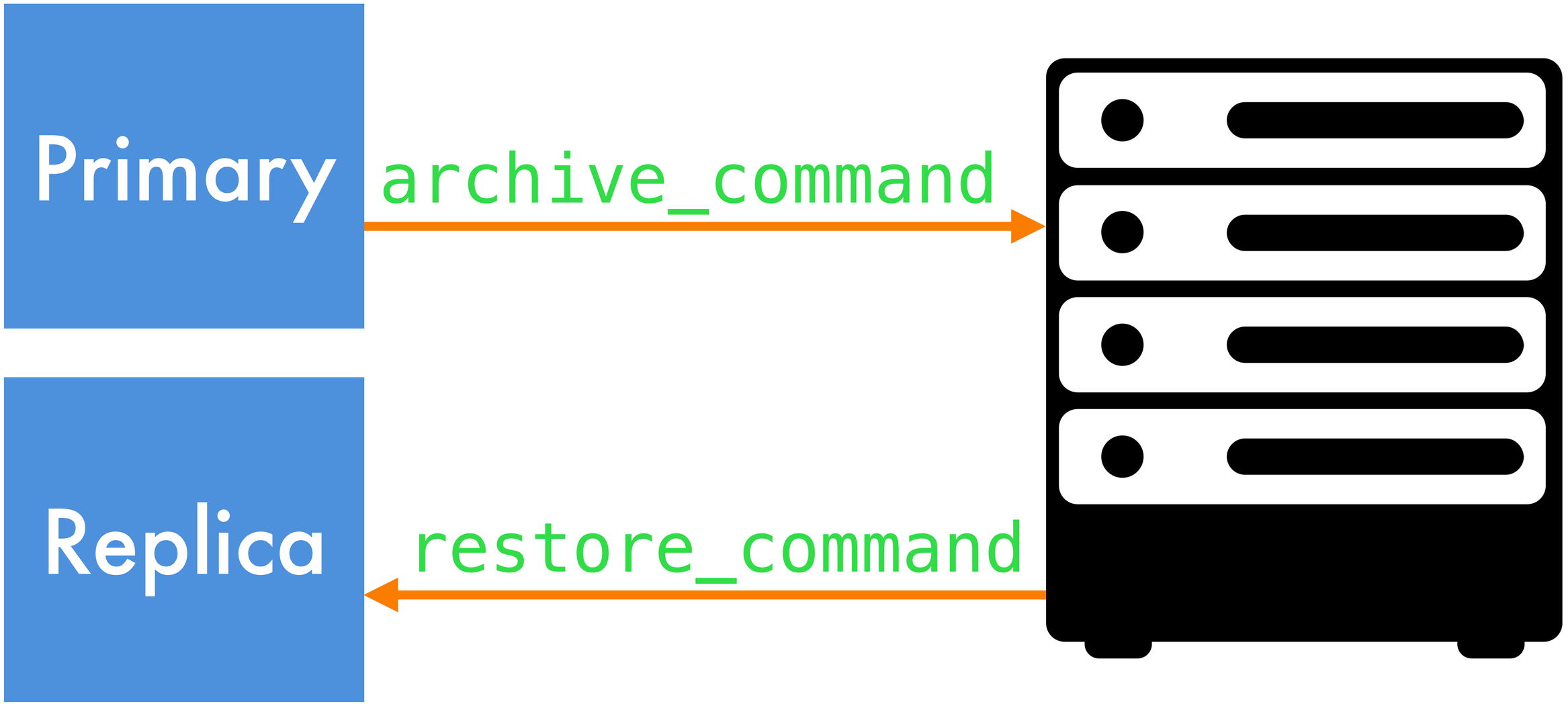

API backend
Postgres
Repl
Postgres
Repl
Postgres


API backend
WALs
Repl
Repl
Postgres
Postgres
Postgres

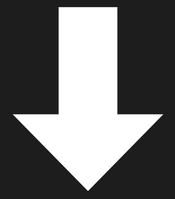
Suspicious log on synchronous replica

```
2023-02-24 17:23:01 GMT LOG: restored log file
"000000020000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length
at 0/3000180
```

WAL archival



Issue restoring WAL



Cause of failure to promote replica?

We already had those writes!

Just because something shouldn't happen doesn't mean it didn't happen

Suspicious log on synchronous replica

```
2023-02-24 17:23:01 GMT LOG: restored log file
"000000020000000000000003" from archive

2023-02-24 17:23:02 GMT LOG: invalid record length
at 0/3000180
```

I had zero experience working with binary formats

None of it is magic

We can cheat:
Postgres is
open source

But!

These techniques also work on closed source software

We just call that reverse engineering

Let's find the error

```
$ git checkout REL9_4_26 # we were running 9.4
$ git grep -n "invalid record length"

src/backend/access/transam/xlogreader.c:295: [...]
src/backend/access/transam/xlogreader.c:604: [...]
src/backend/access/transam/xlogreader.c:678: [...]
```

Let's find the error

src/backend/access/transam/xlogreader.c:291-300:

```
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```

Let's find the error

src/backend/access/transam/xlogreader.c:291-300:

```
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```

Let's find the error

src/backend/access/transam/xlogreader.c:291-300:

```
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```

Let's find the error

```
src/include/access/xlog.h:58:

#define SizeOfXLogRecord    MAXALIGN(sizeof(XLogRecord))
```

Wouldn't it be convenient if we could make `total_len == 0`?

Let's find the error

src/backend/access/transam/xlogreader.c:272-273:

```
record = (XLogRecord *) (state->readBuf + RecPtr % XLOG_BLCKSZ);
total_len = record->xl_tot_len;
```

Let's find the error

src/include/access/xlog.h:41-56:

```
typedef struct XLogRecord
{
    uint32      xl_tot_len;     /* total len of entire record */
    TransactionId xl_xid;       /* xact id */
    uint32      xl_len;         /* total len of rmgr data */
    uint8       xl_info;        /* flag bits, see below */
    RmgrId      xl_rmid;        /* resource manager for this record */
    /* 2 bytes of padding here, initialize to zero */
    XLogRecPtr  xl_prev;        /* ptr to previous record in log */
    pg_crc32    xl_crc;         /* CRC for this record */

    /* If MAXALIGN==8, there are 4 wasted bytes here */

    /* ACTUAL LOG DATA FOLLOWS AT END OF STRUCT */

} XLogRecord;
```

Let's find the error

src/include/access/xlog.h:41-56:

```
typedef struct XLogRecord
{
    uint32      xl_tot_len;     /* total len of entire record */
    TransactionId xl_xid;       /* xact id */
    uint32      xl_len;         /* total len of rmgr data */
    uint8       xl_info;        /* flag bits, see below */
    RmgrId      xl_rmid;        /* resource manager for this record */
    /* 2 bytes of padding here, initialize to zero */
    XLogRecPtr  xl_prev;        /* ptr to previous record in log */
    pg_crc32    xl_crc;         /* CRC for this record */

    /* If MAXALIGN==8, there are 4 wasted bytes here */

    /* ACTUAL LOG DATA FOLLOWS AT END OF STRUCT */

} XLogRecord;
```

What was that check doing?

src/backend/access/transam/xlogreader.c:291-300:

```
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```

What was that check doing?

Size the record says it is

src/backend/access/transam/xlogreader.c:291-300:

```
{
    /* XXX: more validation should be done here */
    if (total_len < SizeOfXLogRecord)
    {
        report_invalid_record(state, "invalid record length at %X/%X",
                              (uint32) (RecPtr >> 32), (uint32) RecPtr);
        goto err;
    }
    gotheader = false;
}
```

Smallest possible size it can be

A different kind of logs

(if they were textual)

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```

```
Wrote 'codd' into table 'users'
Wrote 'lovelace' into table 'users'
Wrote 'turing' into table 'users'
```

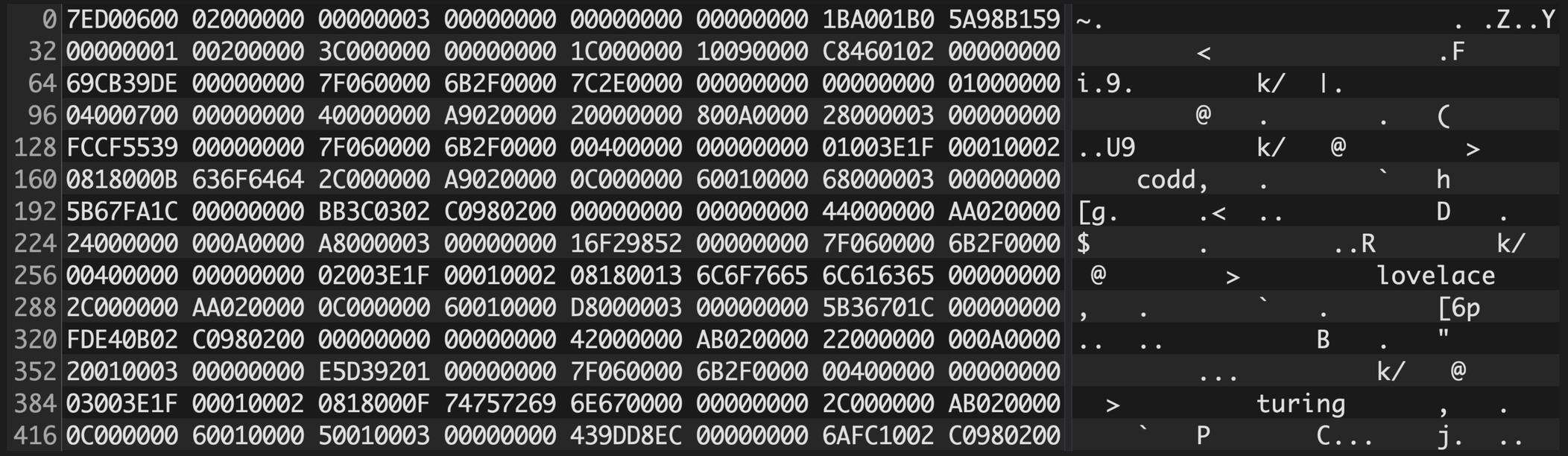
Let's see what they look like in practice

Some extremely boring SQL

```
INSERT INTO users VALUES ('codd');
INSERT INTO users VALUES ('lovelace');
INSERT INTO users VALUES ('turing');
```

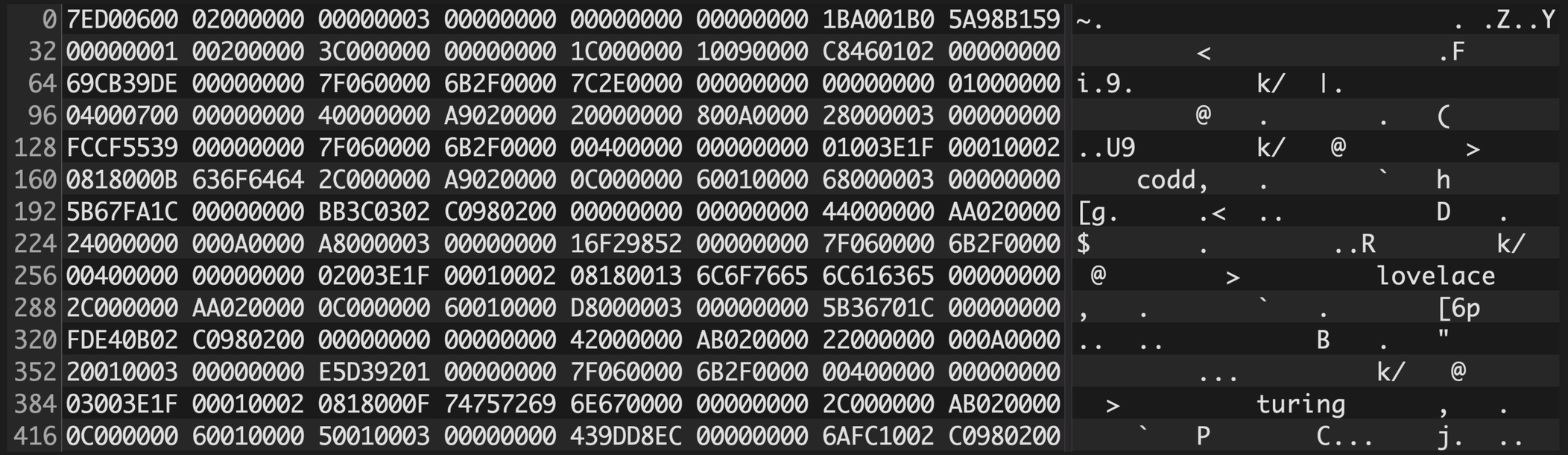
Grab the binary log file, and...

A barely comprehensible wall of data 😅



```
  0 7ED00600 02000000 00000003 00000000 00000000 00000000 1BA001B0 5A98B159 ~.                       . .Z..Y
 32 00000001 00200000 3C000000 00000000 1C000000 10090000 C8460102 00000000         <               .F
 64 69CB39DE 00000000 7F060000 6B2F0000 7C2E0000 00000000 00000000 01000000 i.9.        k/  |.
 96 04000700 00000000 40000000 A9020000 20000000 800A0000 28000003 00000000         @   .       .   (
128 FCCF5539 00000000 7F060000 6B2F0000 00400000 00000000 01003E1F 00010002 ..U9        k/   @         >
160 0818000B 636F6464 2C000000 A9020000 0C000000 60010000 68000003 00000000     codd,   .       `   h
192 5B67FA1C 00000000 BB3C0302 C0980200 00000000 00000000 44000000 AA020000 [g.      .<  ..          D   .
224 24000000 000A0000 A8000003 00000000 16F29852 00000000 7F060000 6B2F0000 $       .        ..R        k/
256 00400000 00000000 02003E1F 00010002 08180013 6C6F7665 6C616365 00000000  @         >         lovelace
288 2C000000 AA020000 0C000000 60010000 D8000003 00000000 5B36701C 00000000 ,   .         `   .       [6p
320 FDE40B02 C0980200 00000000 00000000 42000000 AB020000 22000000 000A0000 ..  ..           B   .   "
352 20010003 00000000 E5D39201 00000000 7F060000 6B2F0000 00400000 00000000         ...         k/   @
384 03003E1F 00010002 0818000F 74757269 6E670000 00000000 2C000000 AB020000   >         turing      ,   .
416 0C000000 60010000 50010003 00000000 439DD8EC 00000000 6AFC1002 C0980200     `   P       C...    j.  ..
```

A barely comprehensible wall of data 😅



	Hex							ASCII
0	7ED00600	02000000	00000003	00000000	00000000	00000000	1BA001B0	5A98B159
32	00000001	00200000	3C000000	00000000	1C000000	10090000	C8460102	00000000
64	69CB39DE	00000000	7F060000	6B2F0000	7C2E0000	00000000	00000000	01000000
96	04000700	00000000	40000000	A9020000	20000000	800A0000	28000003	00000000
128	FCCF5539	00000000	7F060000	6B2F0000	00400000	00000000	01003E1F	00010002
160	0818000B	636F6464	2C000000	A9020000	0C000000	60010000	68000003	00000000
192	5B67FA1C	00000000	BB3C0302	C0980200	00000000	00000000	44000000	AA020000
224	24000000	000A0000	A8000003	00000000	16F29852	00000000	7F060000	6B2F0000
256	00400000	00000000	02003E1F	00010002	08180013	6C6F7665	6C616365	00000000
288	2C000000	AA020000	0C000000	60010000	D8000003	00000000	5B36701C	00000000
320	FDE40B02	C0980200	00000000	00000000	42000000	AB020000	22000000	000A0000
352	20010003	00000000	E5D39201	00000000	7F060000	6B2F0000	00400000	00000000
384	03003E1F	00010002	0818000F	74757269	6E670000	00000000	2C000000	AB020000
416	0C000000	60010000	50010003	00000000	439DD8EC	00000000	6AFC1002	C0980200

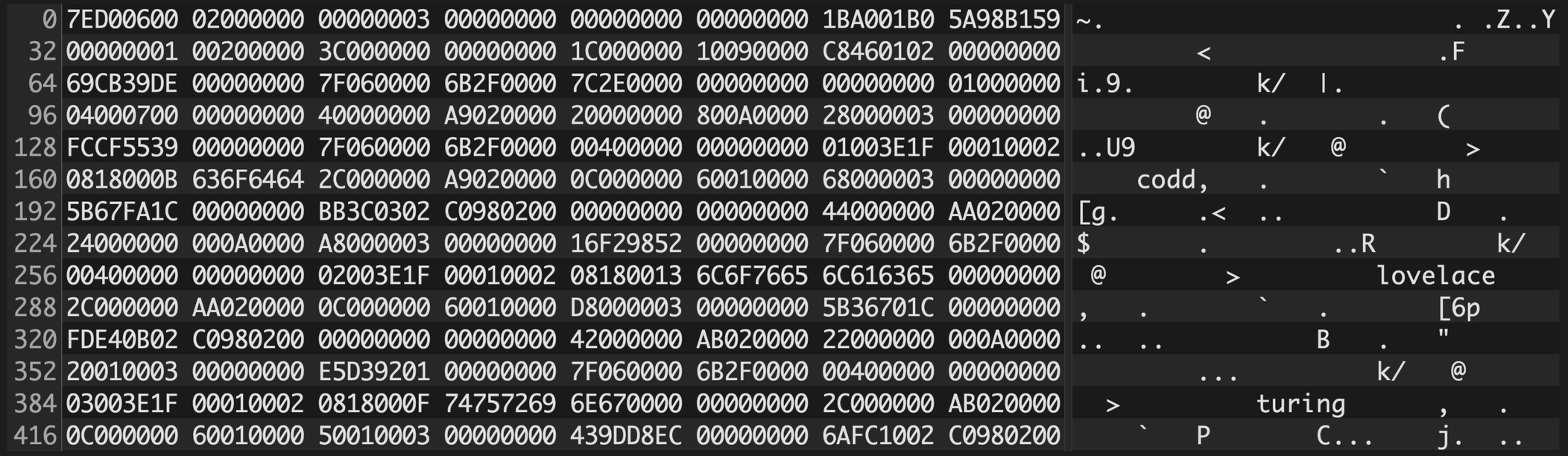
Same data, rendered differently

Decimal	Hexadecimal	Character
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B

Same data, rendered differently

Decimal	Hexadecimal	Character
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B

Some good news



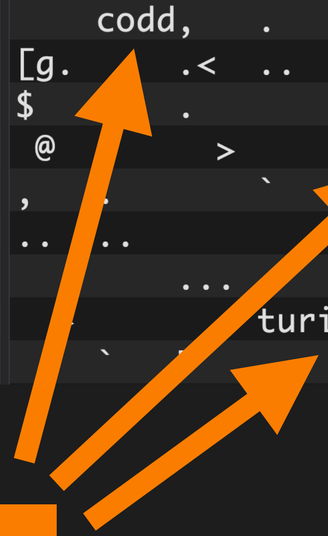
0	7ED00600	02000000	00000003	00000000	00000000	00000000	1BA001B0	5A98B159
32	00000001	00200000	3C000000	00000000	1C000000	10090000	C8460102	00000000
64	69CB39DE	00000000	7F060000	6B2F0000	7C2E0000	00000000	00000000	01000000
96	04000700	00000000	40000000	A9020000	20000000	800A0000	28000003	00000000
128	FCCF5539	00000000	7F060000	6B2F0000	00400000	00000000	01003E1F	00010002
160	0818000B	636F6464	2C000000	A9020000	0C000000	60010000	68000003	00000000
192	5B67FA1C	00000000	BB3C0302	C0980200	00000000	00000000	44000000	AA020000
224	24000000	000A0000	A8000003	00000000	16F29852	00000000	7F060000	6B2F0000
256	00400000	00000000	02003E1F	00010002	08180013	6C6F7665	6C616365	00000000
288	2C000000	AA020000	0C000000	60010000	D8000003	00000000	5B36701C	00000000
320	FDE40B02	C0980200	00000000	00000000	42000000	AB020000	22000000	000A0000
352	20010003	00000000	E5D39201	00000000	7F060000	6B2F0000	00400000	00000000
384	03003E1F	00010002	0818000F	74757269	6E670000	00000000	2C000000	AB020000
416	0C000000	60010000	50010003	00000000	439DD8EC	00000000	6AFC1002	C0980200

Hex

ASCII

Some good news

```
  0 7ED00600 02000000 00000003 00000000 00000000 00000000 1BA001B0 5A98B159 ~.                          . .Z..Y
 32 00000001 00200000 3C000000 00000000 1C000000 10090000 C8460102 00000000         <                .F
 64 69CB39DE 00000000 7F060000 6B2F0000 7C2E0000 00000000 00000000 01000000 i.9.         k/  |.
 96 04000700 00000000 40000000 A9020000 20000000 800A0000 28000003 00000000         @   .       .   (
128 FCCF5539 00000000 7F060000 6B2F0000 00400000 00000000 01003E1F 00010002 ..U9         k/   @         >
160 0818000B 636F6464 2C000000 A9020000 0C000000 60010000 68000003 00000000     codd,   .         `   h
192 5B67FA1C 00000000 BB3C0302 C0980200 00000000 00000000 44000000 AA020000 [g.      .<  ..           D   .
224 24000000 000A0000 A8000003 00000000 16F29852 00000000 7F060000 6B2F0000 $       .         ..R         k/
256 00400000 00000000 02003E1F 00010002 08180013 6C6F7665 6C616365 00000000  @         >          lovelace
288 2C000000 AA020000 0C000000 60010000 D8000003 00000000 5B36701C 00000000 ,       .         `          [6p
320 FDE40B02 C0980200 00000000 00000000 42000000 AB020000 22000000 000A0000 ..      ..                B   .   "
352 20010003 00000000 E5D39201 00000000 7F060000 6B2F0000 00400000 00000000          ...           k/   @
384 03003E1F 00010002 0818000F 74757269 6E670000 00000000 2C000000 AB020000       .          turing       ,   .
416 0C000000 60010000 50010003 00000000 439DD8EC 00000000 6AFC1002 C0980200     `   .          C...     j.  ..
```



We can see our users!!

How can we find `xl_tot_len`?

Some even more boring SQL

```
INSERT INTO repro VALUES ('A');
INSERT INTO repro VALUES ('AB');
INSERT INTO repro VALUES ('ABC');
INSERT INTO repro VALUES ('ABCD');
INSERT INTO repro VALUES ('ABCDE');
...
```

Look for a field increasing by 1

Guesswork incoming!

```
420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000     $.)]KK          ?    .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000          .       " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/   @        @?         ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC ,   .        `    .        ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000      *,]KK           @    .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000          .       FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/   @        @?         ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 ,   .        `    (        ..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000     ...]KK           A    .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 !       h        ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/   @        @?         ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E       ,    .        `    .
```

Guesswork incoming!

```
420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000     $.)]KK          ?    .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000           .       " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/   @        @?        ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC ,   .        `    .        ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000      *,]KK           @    .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000           .       FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/   @        @?        ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 ,   .        `   (        ..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000     ...]KK          A    .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 !        h        ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/   @        @?        ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E        ,    .        `    .
```

The data we inserted

A little help: ASCII codes

Decimal	Hexadecimal	Character
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B

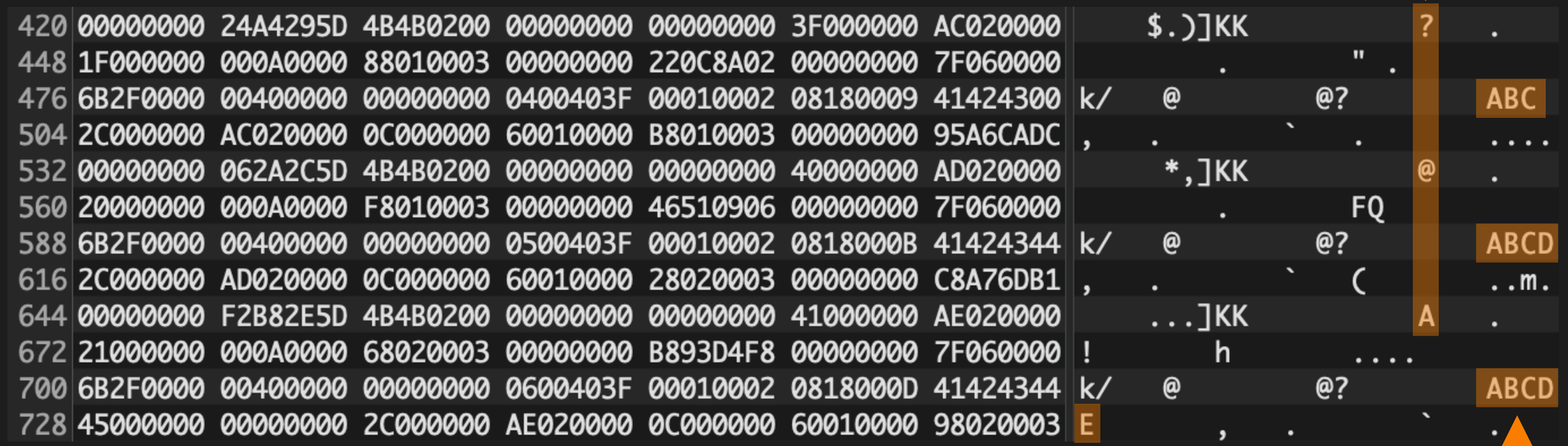
Notice anything?

```
420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000     $.)]KK          ?    .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000          .       " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/   @        @?        ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC ,   .        `    .        ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000      *,]KK           @    .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000           .       FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/   @        @?        ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 ,   .        `   (        ..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000     ...]KK          A    .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 !        h        ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/   @        @?        ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E        ,    .        `    .
```

The data we inserted

Notice anything?

Familiar characters



The data we inserted

Notice anything?

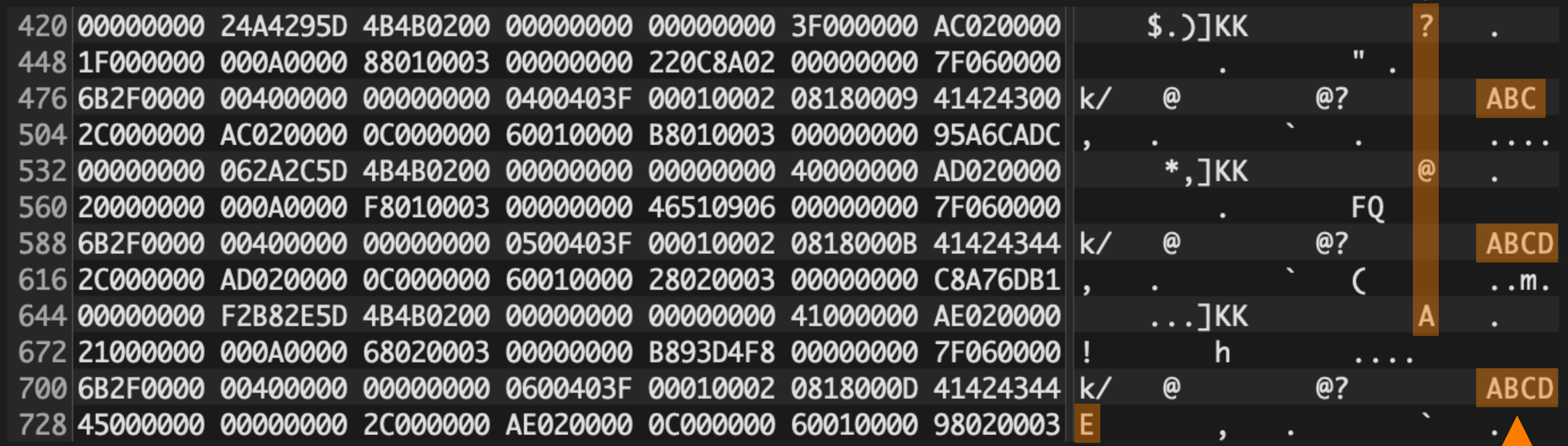
Decimal	Hexadecimal	Character
63	3F	?
64	40	@
65	41	A

Familiar characters

The data we inserted

Notice anything?

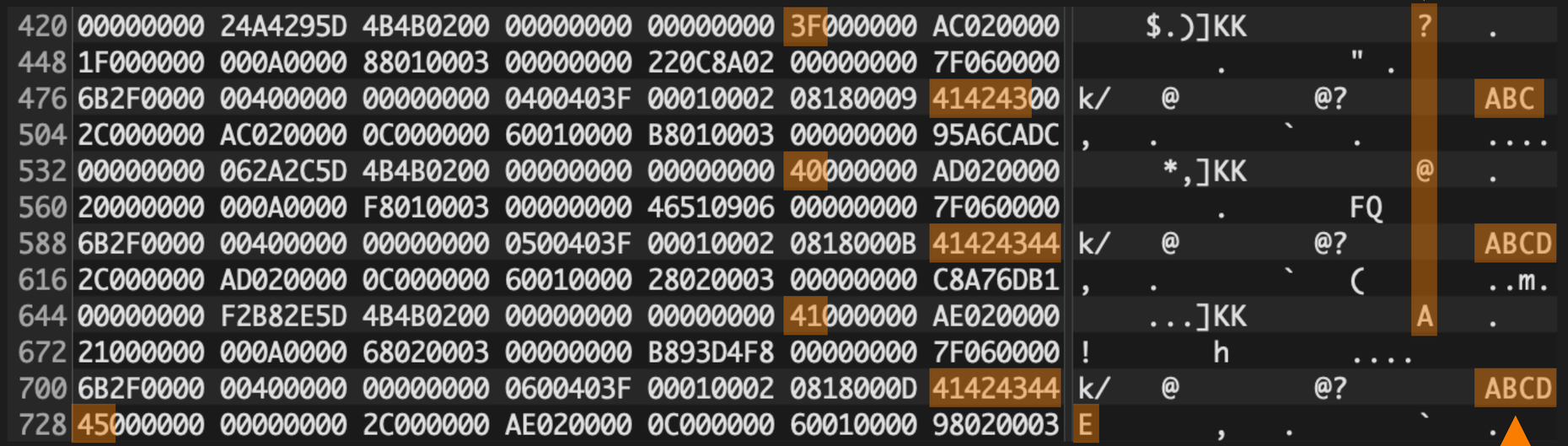
Familiar characters



The data we inserted

Notice anything?

Familiar characters

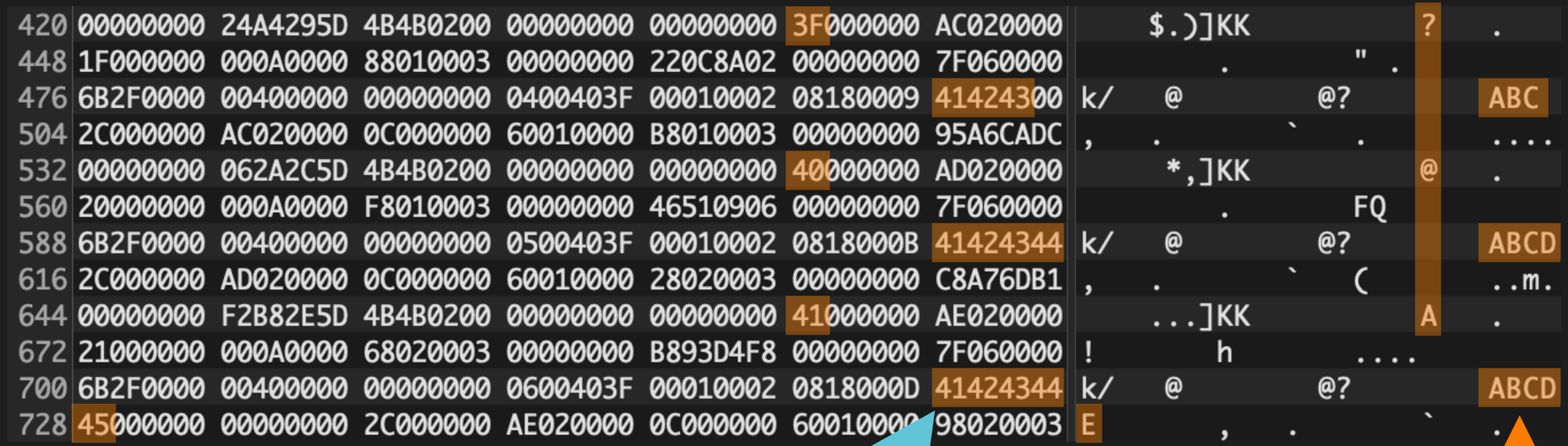


The data we inserted

Notice anything?

Familiar characters (hex)

Familiar characters



The data we inserted (hex)

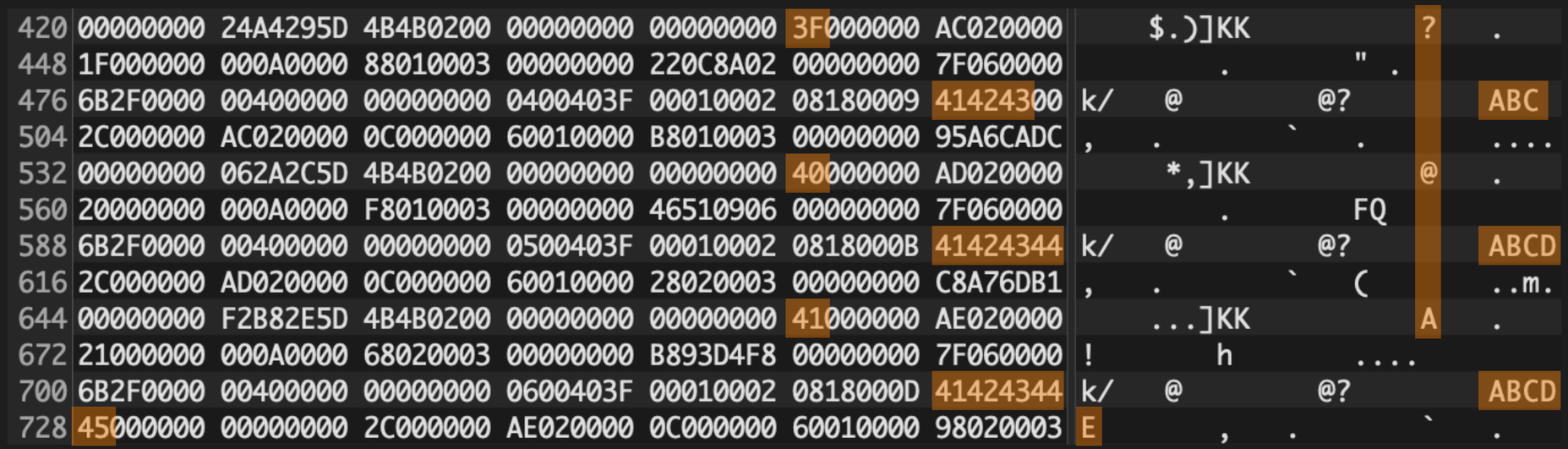
The data we inserted

Wouldn't it be convenient if we could make `total_len == 0`?

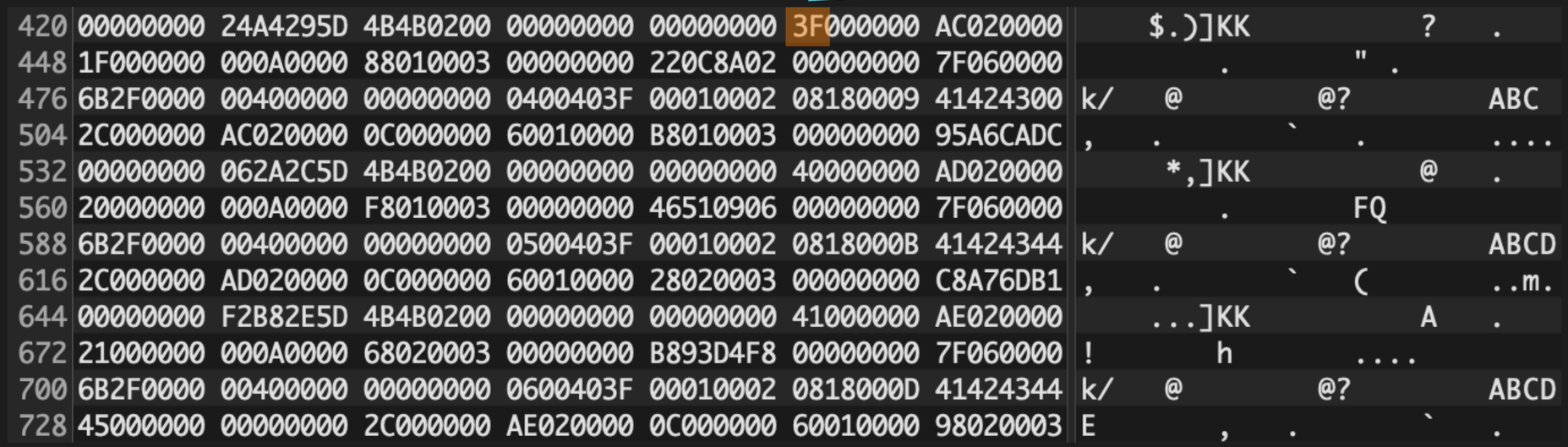
We could import the Postgres structs and do this properly...

…or we could write a regex 🤔

Let's write a regex


420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000 $.)]KK ? .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000 . " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/ @ @? ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC , . ` . ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000 *,]KK @ .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000 . FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/ @ @? ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 , . ` (..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000 ...]KK A .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 ! h ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/ @ @? ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E , . ` .

Let's break this one



```
420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000     $.)]KK          ?    .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000          .       " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/   @        @?        ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC ,   .        `    .        ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000      *,]KK           @    .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000           .       FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/   @        @?        ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 ,   .        `   (        ..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000     ...]KK          A    .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 !        h        ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/   @        @?        ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E       ,    .        `    .
```

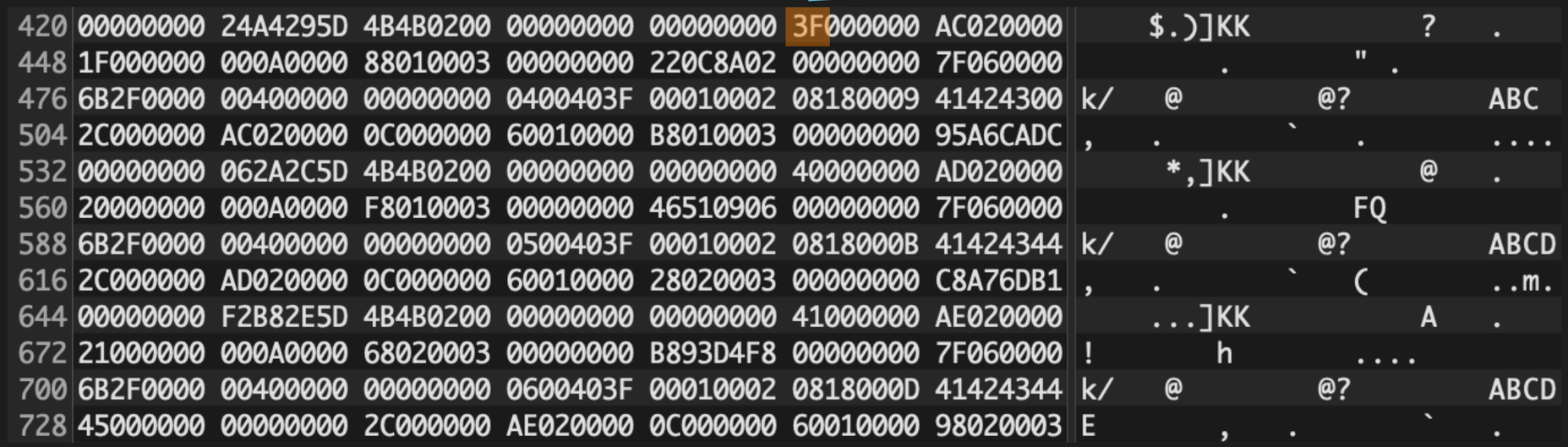
break_wal.rb

```
wal_file_name = ARGV[0]
puts wal_file_name

wal_contents = IO.read(wal_file_name, encoding: "BINARY")
hex = wal_contents.unpack("H*").first
replaced = hex.gsub(/3f(000000.+41424300)/, "00\\1")
bindata = [replaced].pack("H*")
File.write(wal_file_name + ".broken", bindata)
```

break_wal.rb

```
wal_file_name = ARGV[0]
puts wal_file_name

wal_contents = IO.read(wal_file_name, encoding: "BINARY")
hex = wal_contents.unpack("H*").first
replaced = hex.gsub(/3f(000000.+41424300)/, "00\\1") # Replaces 'ABC' size
bindata = [replaced].pack("H*")
File.write(wal_file_name + ".broken", bindata)
```

Let's break this one



```
420 00000000 24A4295D 4B4B0200 00000000 00000000 3F000000 AC020000     $.)]KK          ?    .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000          .       " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/   @        @?        ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC ,   .        `    .        ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000      *,]KK           @    .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000           .       FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/   @        @?        ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 ,   .        `   (        ..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000     ...]KK          A    .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 !        h        ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/   @        @?        ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E       ,    .        `    .
```


Broken!!
420 00000000 24A4295D 4B4B0200 00000000 00000000 00000000 AC020000 $.)]KK .
448 1F000000 000A0000 88010003 00000000 220C8A02 00000000 7F060000 . " .
476 6B2F0000 00400000 00000000 0400403F 00010002 08180009 41424300 k/ @ @? ABC
504 2C000000 AC020000 0C000000 60010000 B8010003 00000000 95A6CADC , . ` . ....
532 00000000 062A2C5D 4B4B0200 00000000 00000000 40000000 AD020000 *,]KK @ .
560 20000000 000A0000 F8010003 00000000 46510906 00000000 7F060000 . FQ
588 6B2F0000 00400000 00000000 0500403F 00010002 0818000B 41424344 k/ @ @? ABCD
616 2C000000 AD020000 0C000000 60010000 28020003 00000000 C8A76DB1 , . ` (..m.
644 00000000 F2B82E5D 4B4B0200 00000000 00000000 41000000 AE020000 ...]KK A .
672 21000000 000A0000 68020003 00000000 B893D4F8 00000000 7F060000 ! h ....
700 6B2F0000 00400000 00000000 0600403F 00010002 0818000D 41424344 k/ @ @? ABCD
728 45000000 00000000 2C000000 AE020000 0C000000 60010000 98020003 E , . ` .

And if we give it to a Postgres replica?

We reproduced the error!

```
2023-02-28 19:24:11 GMT LOG: restored log file
"000000020000000000000003" from archive

2023-02-28 19:24:11 GMT LOG: invalid record length
at 0/3000148
```

Success 

Success, with a caveat...😔

This wasn't enough to reproduce the outage

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

~~1. RAID array loses disks~~

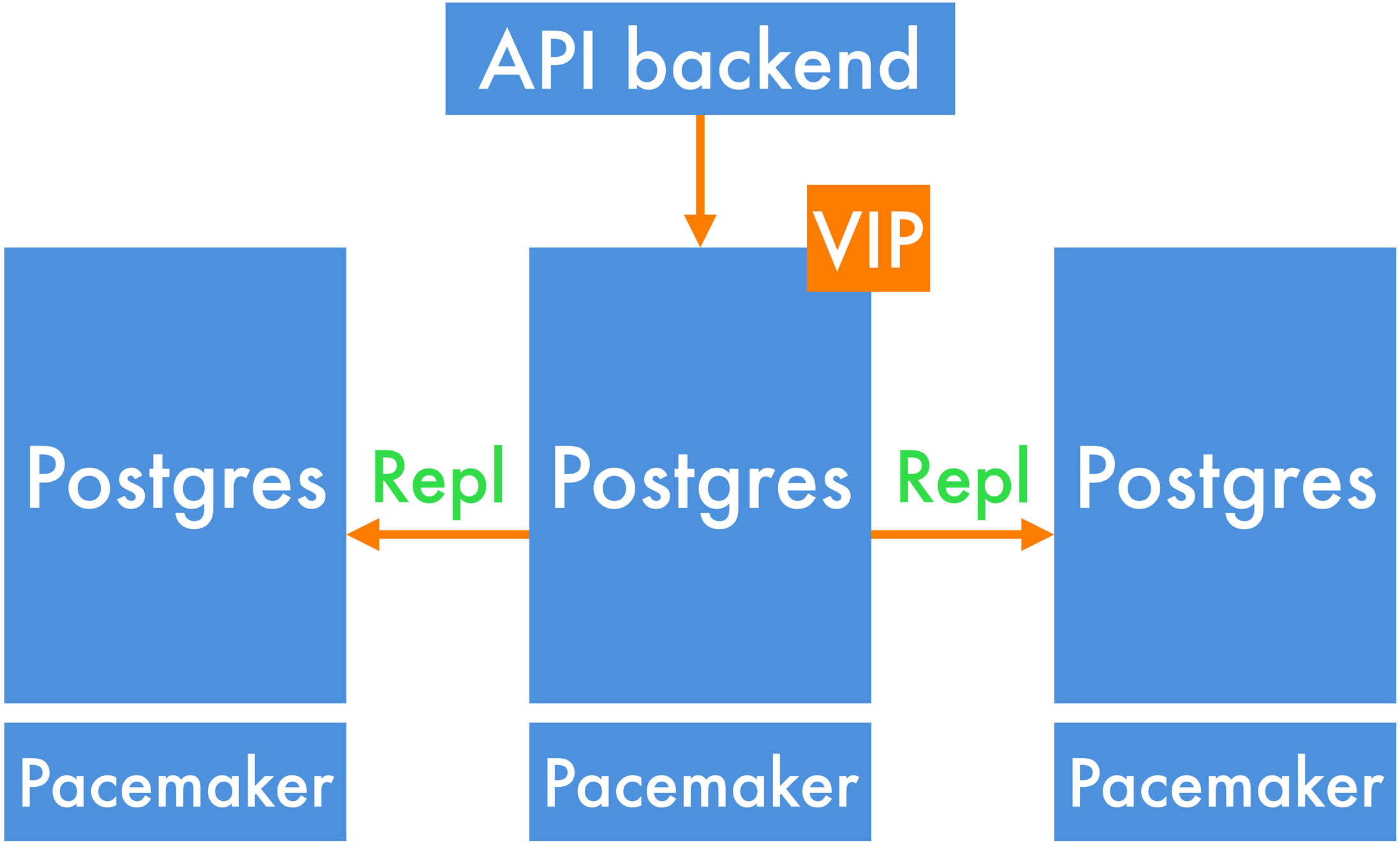
~~2. Kernel sets filesystem read-only~~

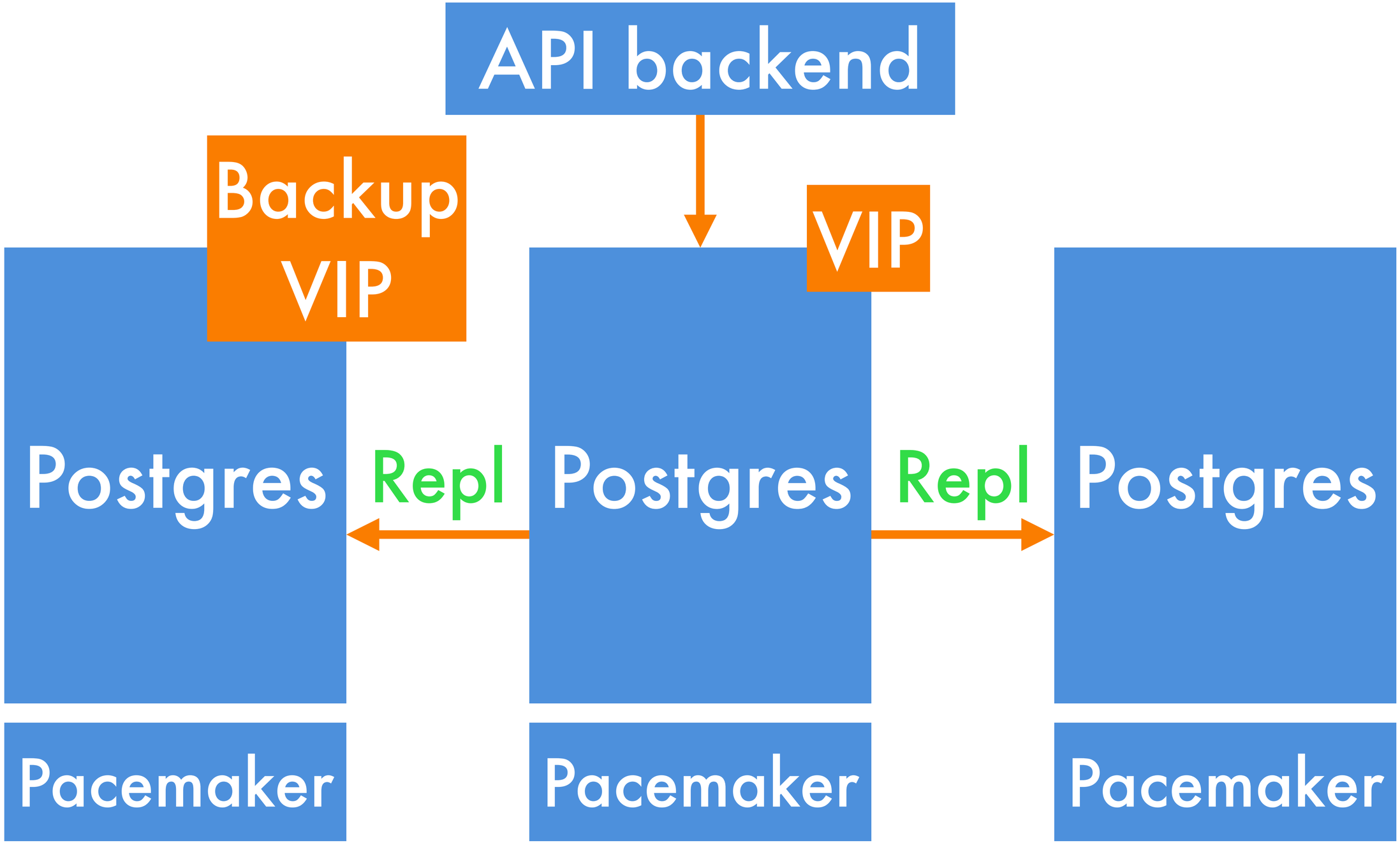
3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

6. ...


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
Backup VIP
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

4. Synchronous replica crash

5. Suspicious log on synchronous replica

6. Backup VIP on synchronous replica

We added it to the cluster

Ran the repro script

and...

Success

(no caveats)

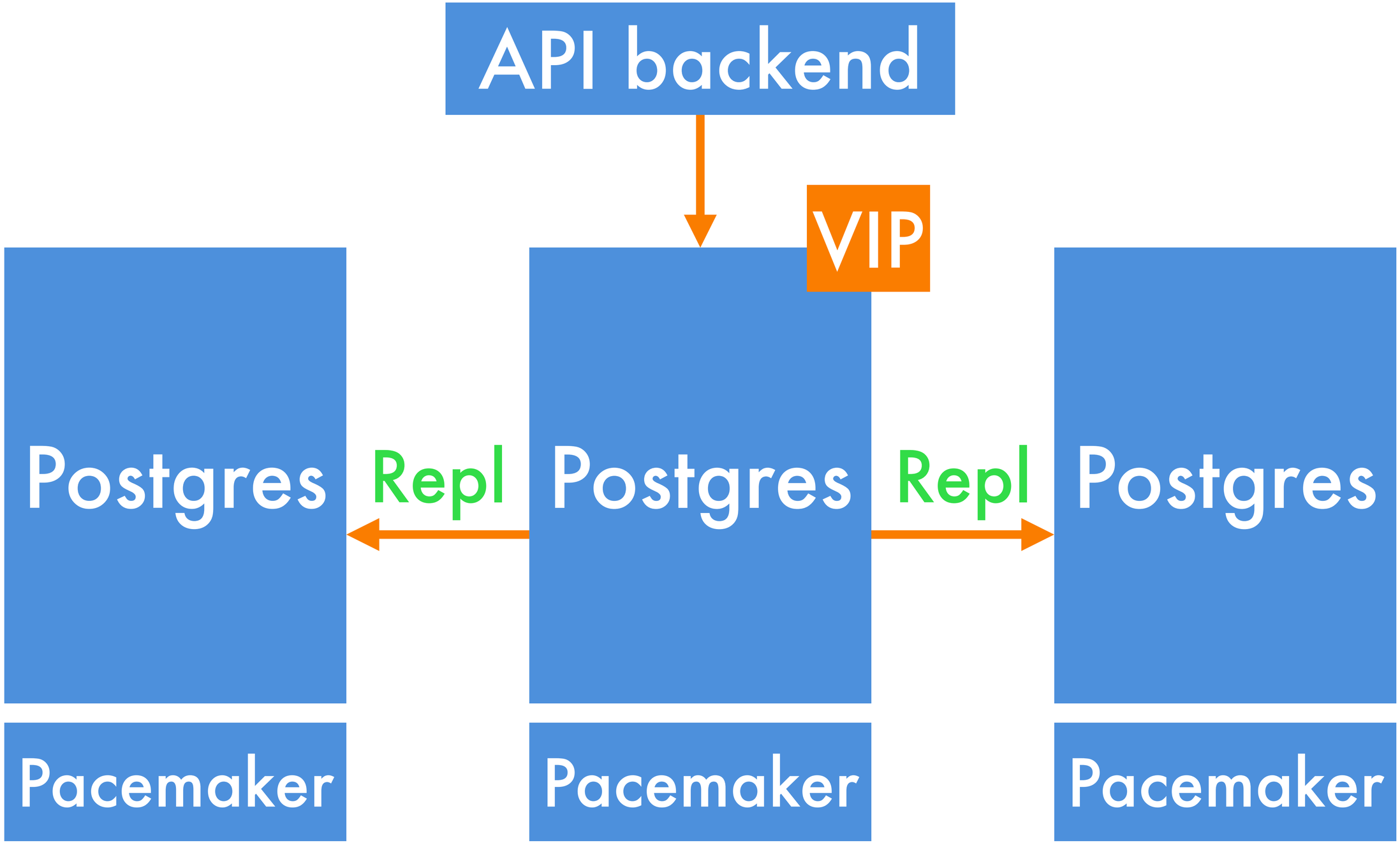
but...

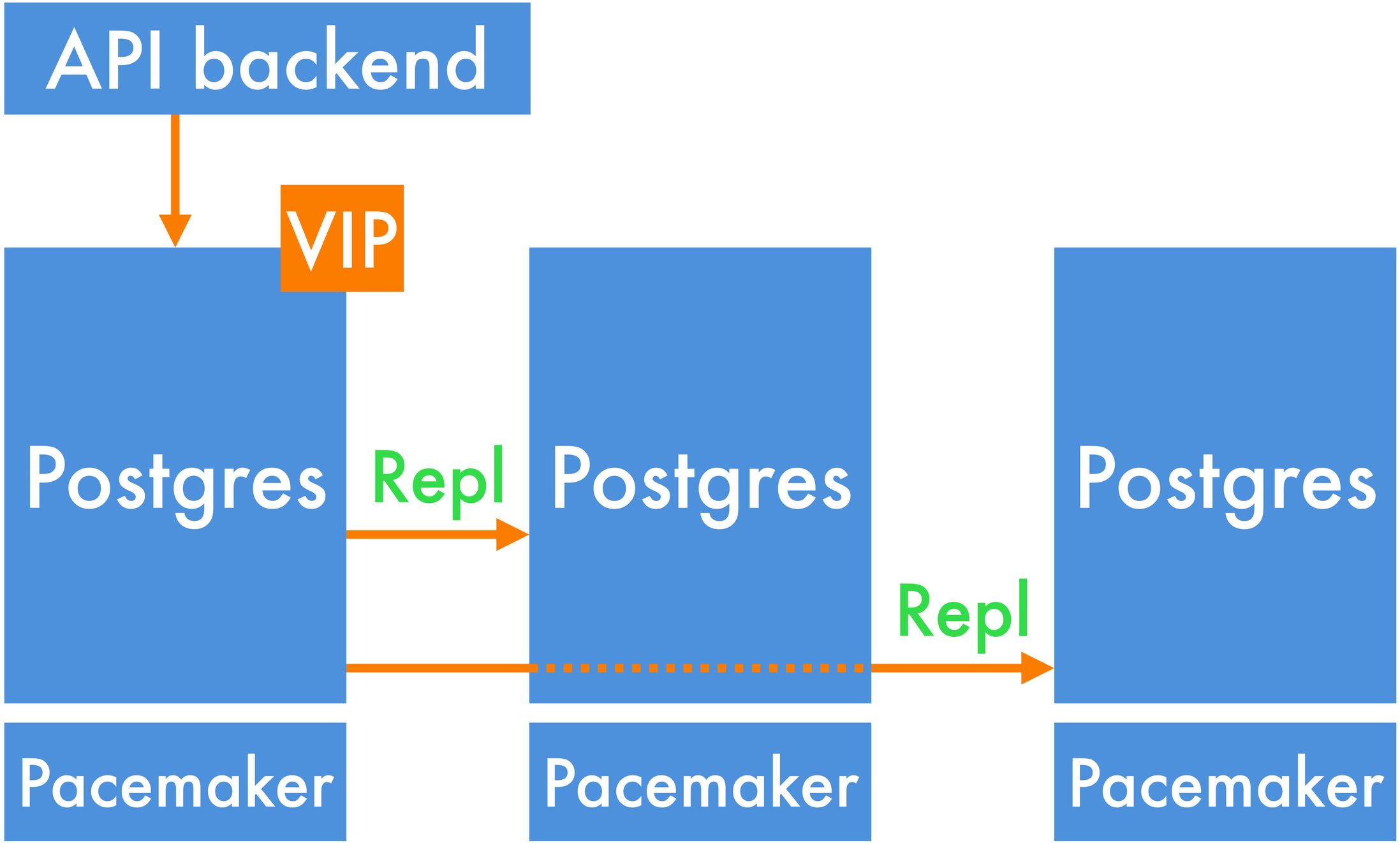
why?

Background: how Pacemaker schedules resources

2 relevant settings

By default:
reschedule
without penalty


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker


API backend
VIP
Postgres
Repl
Postgres
Repl
Postgres
Pacemaker
Pacemaker
Pacemaker

Setting:

`default-resource-stickiness`

By default:

resources can run

anywhere

Setting:

colocation

```
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
```

```
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
```

```
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
```

A very subtle semantic difference

-1000 -inf

-1000

-inf

"Avoid scheduling these together"

`-1000`

"Avoid scheduling these together"

`-inf`

"Literally never schedule these together"

```
default-resource-stickiness = 100

&

colocation -inf: BackupVIP Primary
```

default-resource-stickiness = 100

&

colocation -1000: BackupVIP Primary

Failover works

properly

P.S. The WAL error was a red herring

Sorry

I know it was the most interesting part

and it would have been kinda cool

but it was part of the debugging process

~~1. RAID array loses disks~~

~~2. Kernel sets filesystem read-only~~

3. Pacemaker detects primary failure

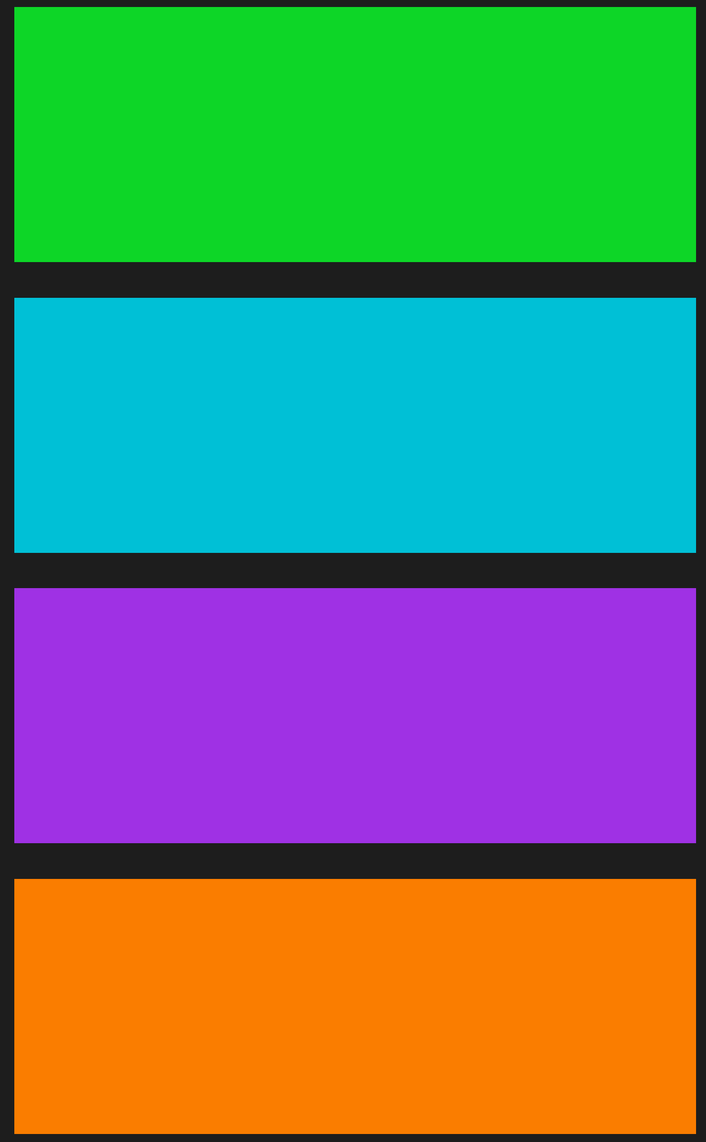
4. Synchronous replica crash

~~5. Suspicious log on synchronous replica~~

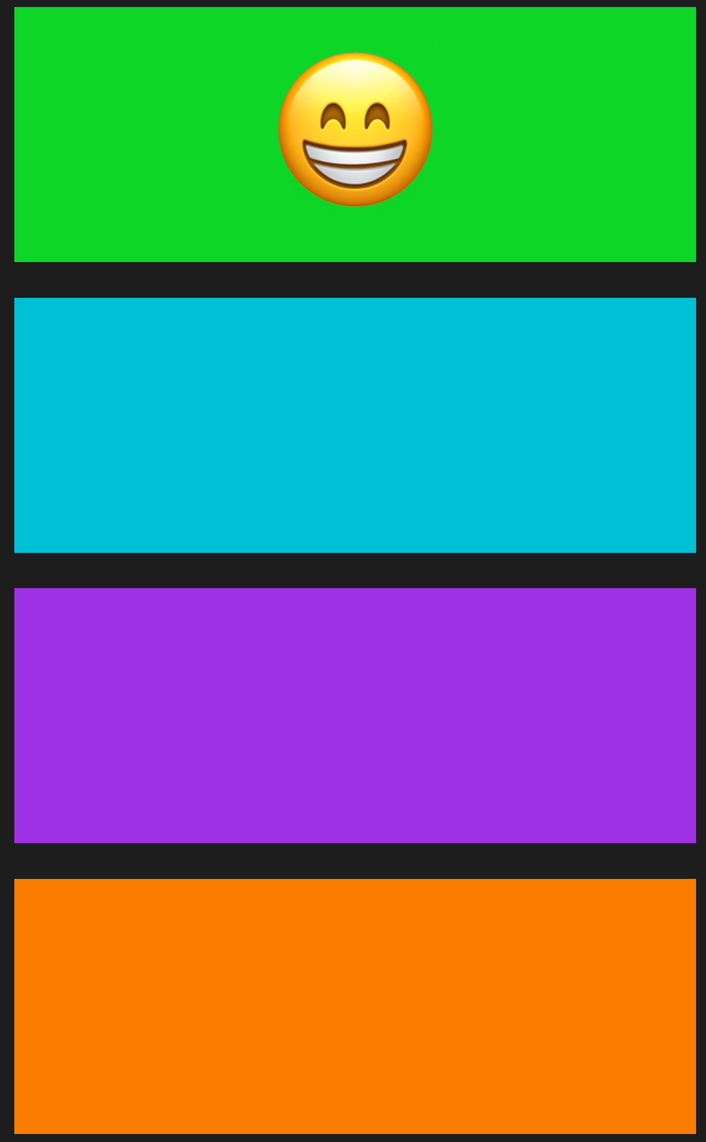
6. Backup VIP on synchronous replica

What can we learn?

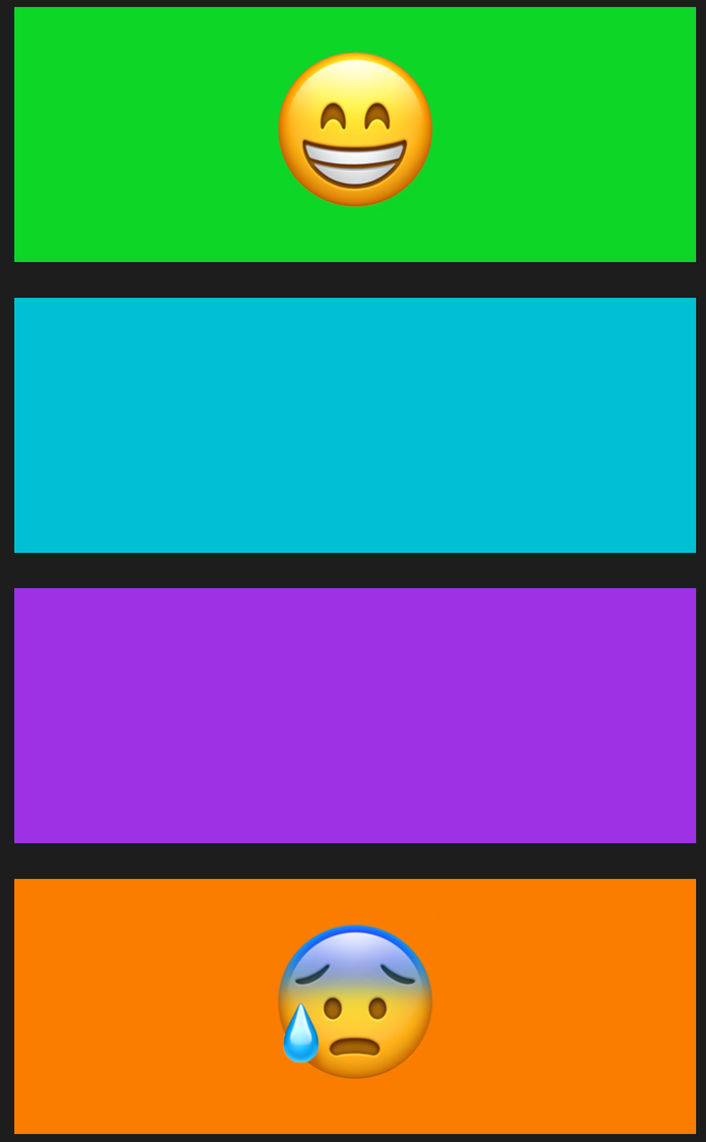
None of the stack is magic



None of the stack is magic



None of the stack is magic



"It's just someone else's computer"

"It's just someone else's abstraction"

Read

other people's

code...

...and

try to

modify it

Automation

erodes

knowledge

Game days are a partial fix

"What if we had to recover our database server manually?"

Don't stop

questioning

your repro

1. No magic in the stack
2. Automation erodes knowledge
3. Always question the repro

JSON

over

HTTP

Binary formats are coming to web development

Protobuf

over

HTTP/2

Protobuf (e.g. gRPC)

over

HTTP/2

It's
worth
getting
familiar

One last thing to ask of you

Most computing happens successfully

The

0.00001%

** not a real statistic*

Outsized

negative

impact

It's a shame

not to

learn

"We noticed a problem."

"We fixed the problem."

"We'll make sure the problem doesn't happen again."

3 good examples

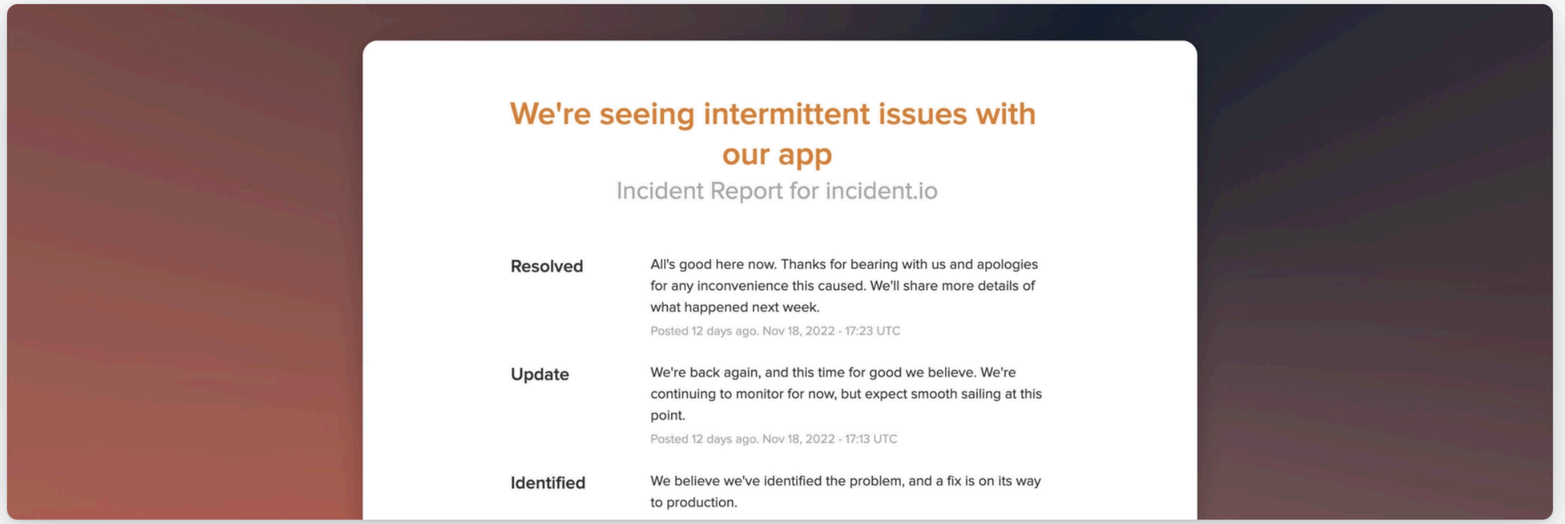
slack engineering

Slack's Outage on January 4th 2021

Slack's Outage on January 4th 2021


Laura Nolan Senior Staff Engineer

https://slack.engineering/slacks-outage-on-january-4th-2021/



Intermittent downtime from repeated crashes

Engineering | November 30, 2022 • Lawrence Jones

On Friday 18th November 2022 we experienced 13 minutes of downtime over a period of 32 minutes from 15:40 to 16:12 GMT.

At incident.io, our customers depend on us to be available even when everything else is down, so we take incidents like this seriously. We also sincerely believe in the power of transparency, and want to be as open as possible in these situations, sharing what details we can.

https://incident.io/blog/intermittent-downtime

Blog / Company

Postmortem of database outage of January 31

GitLab · Feb 10, 2017 · 21 min read · Leave a comment

On January 31st 2017, we experienced a major service outage for one of our products, the online service GitLab.com. The outage was caused by an accidental removal of data from our primary database server.

https://about.gitlab.com/blog/2017/02/10/postmortem-of-database-outage-of-january-31/

Please

Share the difficult stories too

Thank you

✌️❤️

sinjo.dev

@planetscaledata

PlanetScale

Image credits

- Programmer's Laptop - Wall Boat - Public Domain - https://www.flickr.com/photos/wallboat/36819065315/
- Pouring Latte Art - Craft Coffee Spot - CC-BY - https://www.flickr.com/photos/195403219@N08/52200966448/
- microscope - Milosz1 - CC-BY - https://www.flickr.com/photos/mikolski/3269906279
- Hard Disk Guts - CC-BY - https://www.flickr.com/photos/mattandkim/97533589/
- Corsair ForceGT 180GB - CC-BY - https://www.flickr.com/photos/ruocaled/8173124575/

Image credits

- Server - The Noun Project (via WikiMedia) - CC0 - https://commons.wikimedia.org/wiki/File:Server_-_The_Noun_Project.svg
- Rope - Robo Android - CC-BY - https://www.flickr.com/photos/49140926@N07/6798304070/
- Stargazing - Max Delaquis - CC-BY - https://www.flickr.com/photos/115000114@N07/28861043652

Questions?

✌️❤️

sinjo.dev

@planetscaledata