Understanding PostgreSQL timelines

Heikki Linnakangas / VMware

Point-in-Time Recovery

- DELETE FROM accounts <enter>
  - Oops!
- Kill server
- Restore from backup. Try to recover up to just before the DELETE
- Oops, went too far
- Restore again from backup, try to recover to correct location

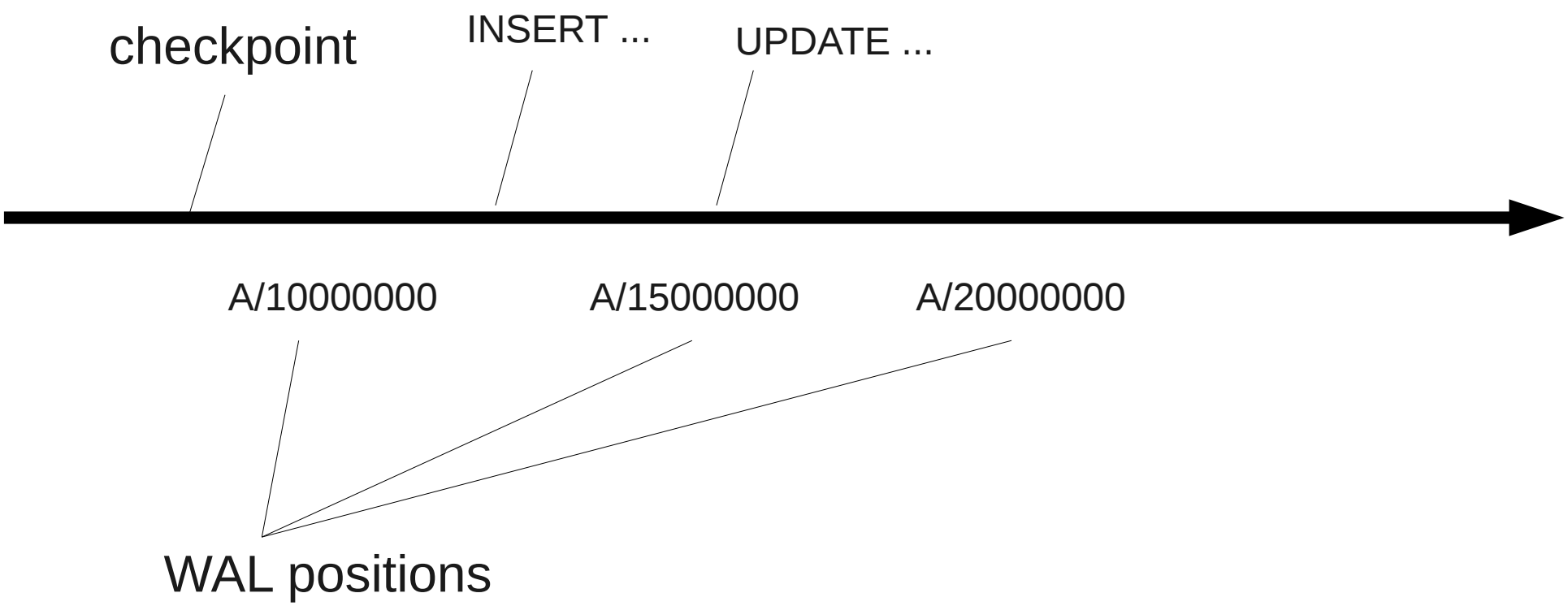
Timelines

- Introduced with Point-in-Time-Recovery in version 8.0
- Every time you do PITR, a new timeline is formed
- This helps you to differentiate WAL generated from different PITR attempts

Timeline in a single server

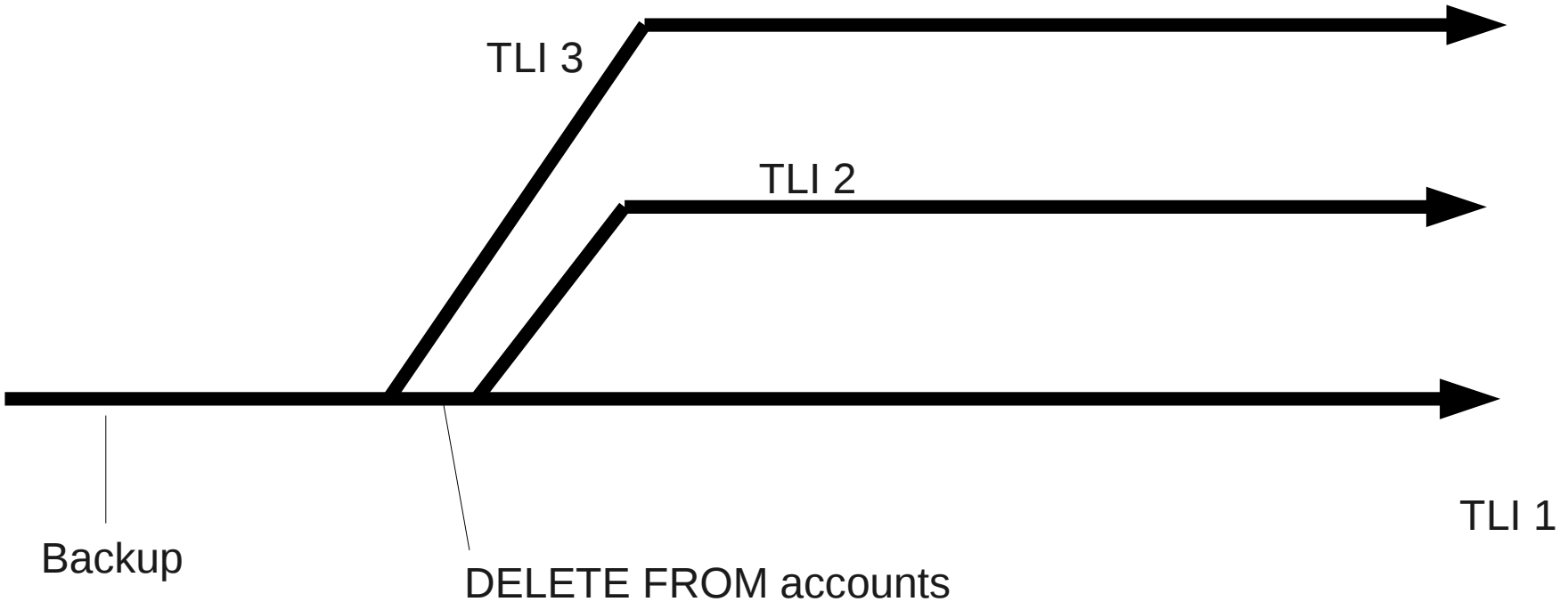
- Boring

Timeline in a single server

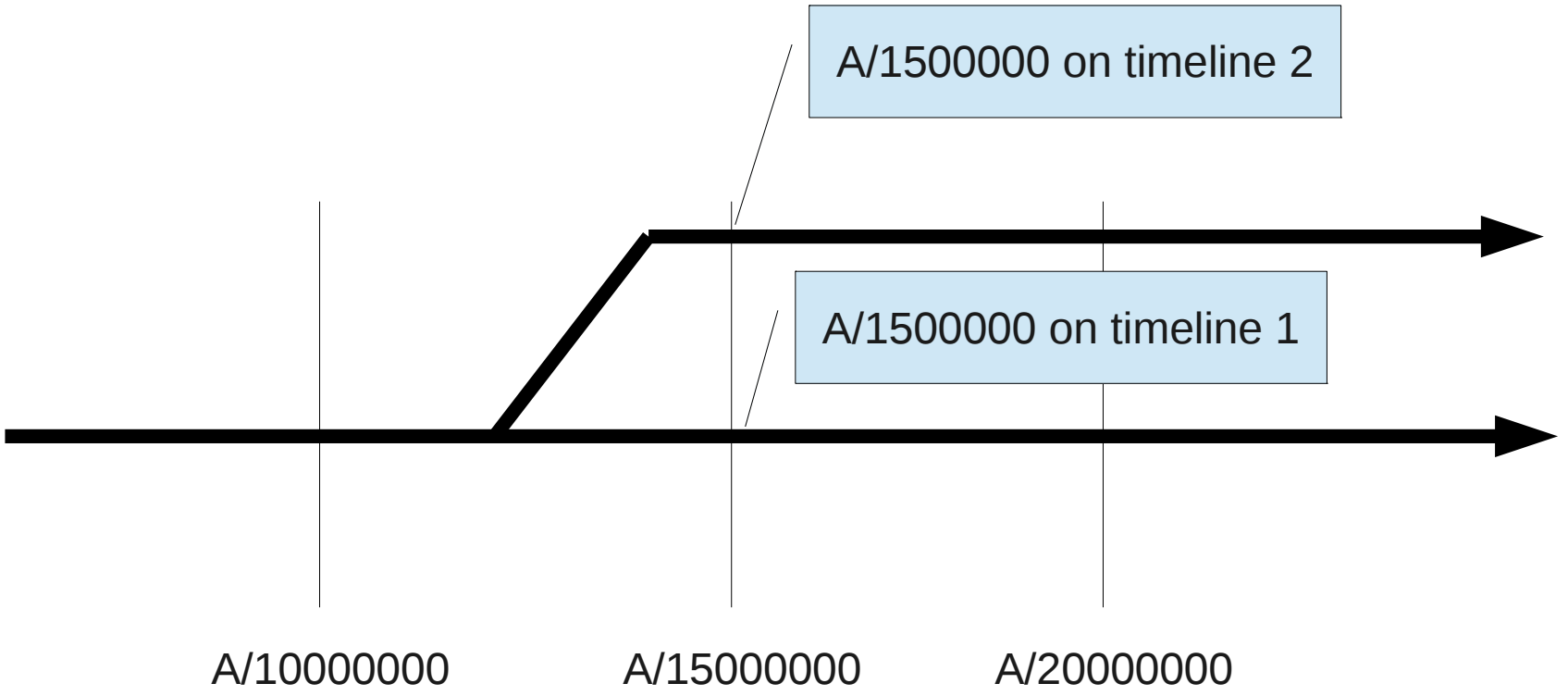


Point-in-Time Recovery

- When new WAL is genereated after PITR, you don't want to overwrite the old WAL.

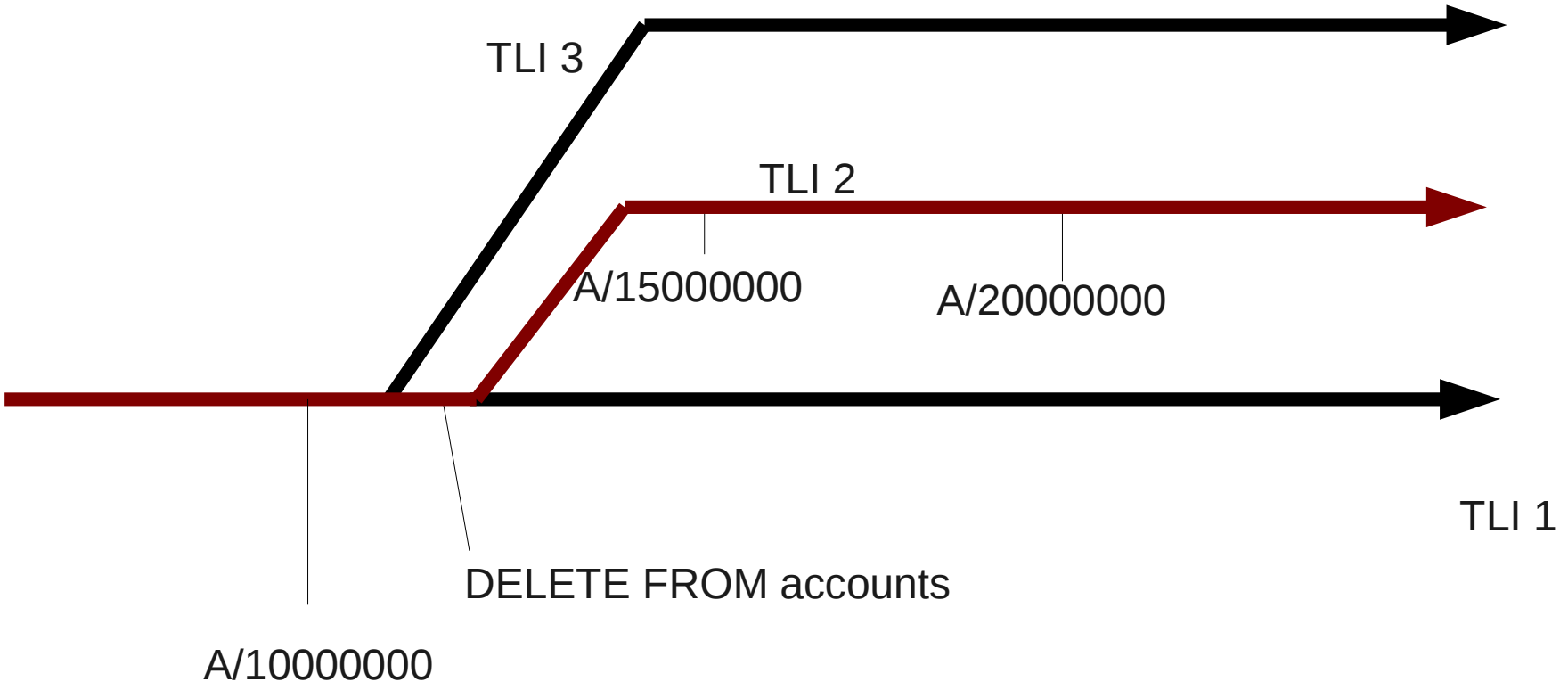


Timeline ID


A/1500000 on timeline 2
A/1500000 on timeline 1
A/10000000
A/15000000
A/20000000

Timelines

- Looking back from any point in time, the history is linear



WAL archive

000000010000013000000E1

000000010000013000000E2

000000010000013000000E3

000000010000013000000E4

000000010000013000000E5

000000020000013000000E3

000000020000013000000E4

000000020000013000000E5

WAL archive

00000001000000130000000E1

00000001000000130000000E2

00000001000000130000000E3 00000002000000130000000E3

00000001000000130000000E4 00000002000000130000000E4

00000001000000130000000E5 00000002000000130000000E5

What happens at a end of recovery?

- End of recovery means the point where the the database opens up for **writing**
- New timeline is chosen
- A timeline history file is written
- The partial last WAL file on the previous timeline is copied with the new timeline's ID
- A checkpoint record is written on the new timeline

Example: End of recovery

LOG: database system was interrupted; last known up at 2013-01-30 21:45:14 EET

LOG: starting archive recovery

LOG: redo starts at 13/E00000C8

LOG: could not open file "pg_xlog/000000010000000E000000E4": No such file or directory

LOG: redo done at 13/E3D389A0

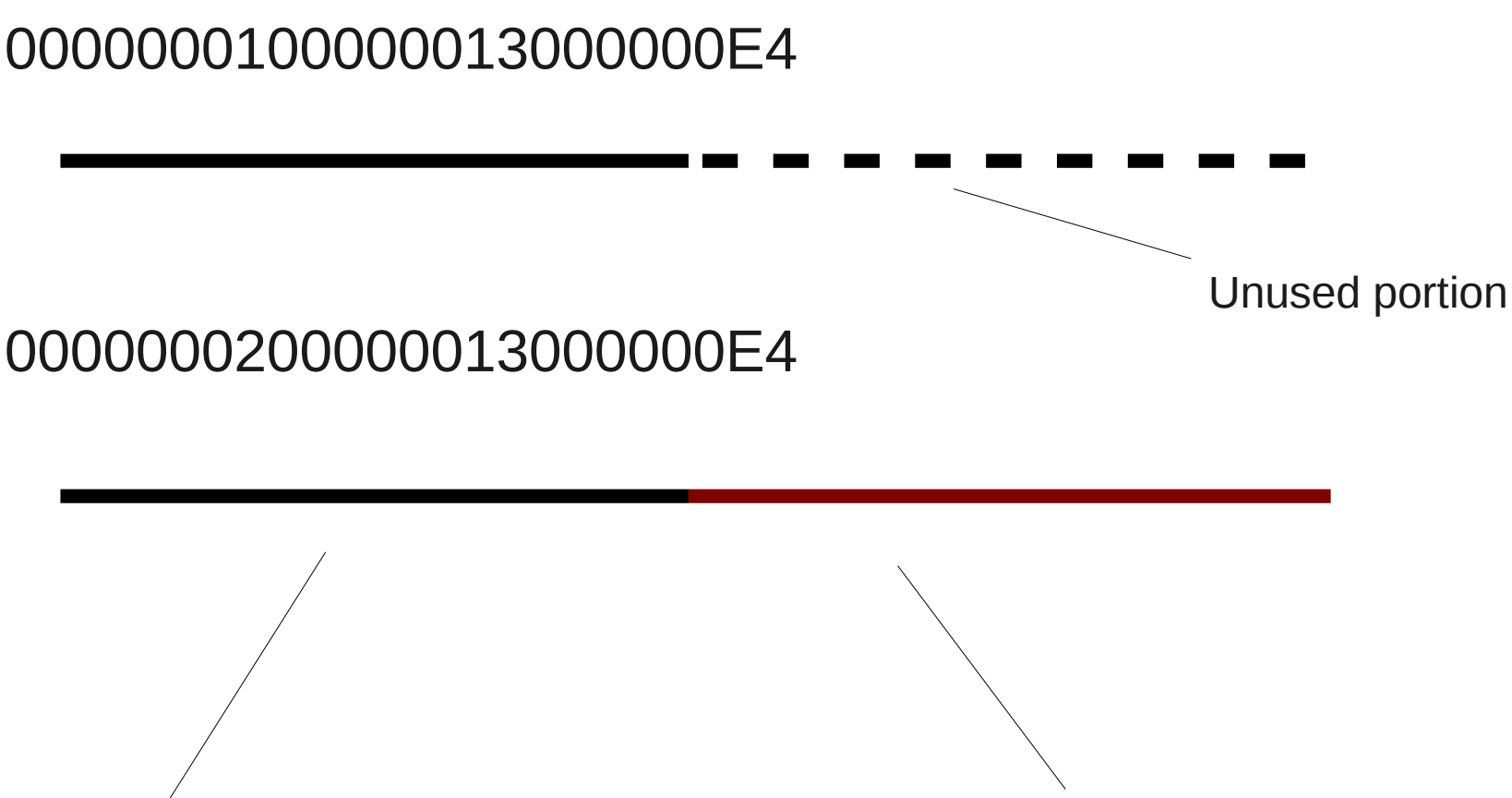
LOG: last completed transaction was at log time 2013-01-30 21:45:20+02

LOG: selected new timeline ID: 2

LOG: archive recovery complete

LOG: database system is ready to accept connections

First WAL file with new timeline



Timeline history file

00000001000000013000000E1

00000001000000013000000E2

00000001000000013000000E3

00000001000000013000000E4

00000001000000013000000E5

00000002.history

00000002000000013000000E3

00000002000000013000000E4

00000002000000013000000E5

Timeline history file

$ cat data-standby1/pg_xlog/00000003.history

```
1  13/E4000000  no recovery target specified
2  13/ED000090  at restore point "before FOSDEM"
```

Timeline ID

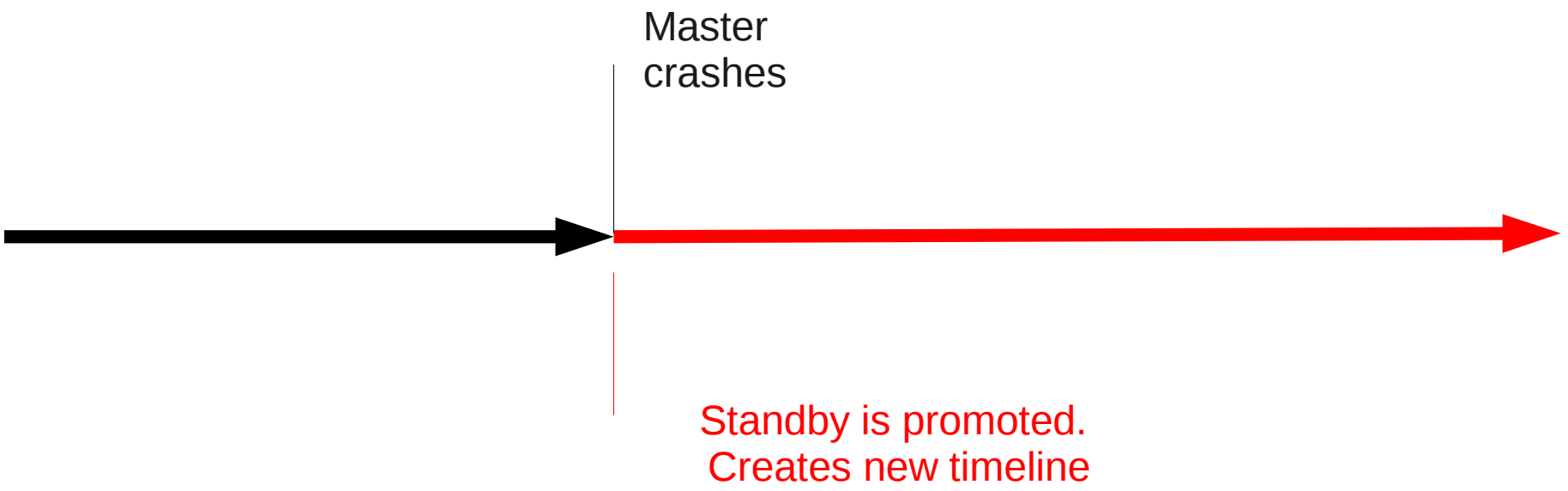
Point in WAL where new timeline begins

Reason for the timeline switch

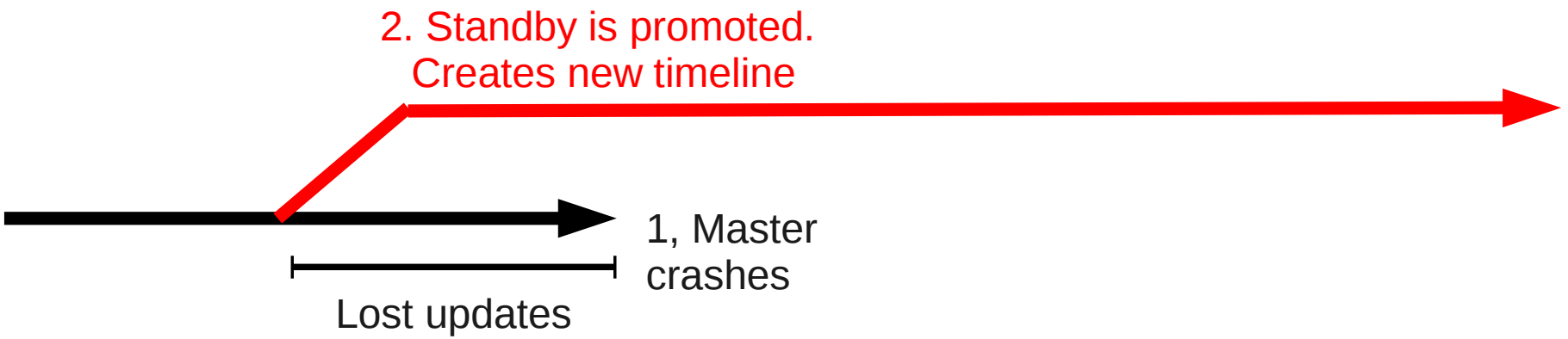
But I don't do PITR!

- Are you sure?
- Do you have a standby?

Promoting a standby



In reality, it's more like PITR



But I use synchronous replication

- Doesn't matter
- In synchronous replication, commits are not acknowledged to the client until the commit record is replicated
- It's still written to disk in the master first
- Even if you don't lose any commits, other WAL records are not synchronous (that would totally kill performance)

Ok, so I do PITR

- Embrace the timelines

Upcoming 9.3 enhancements

- Streaming replication can follow a timeline switch
  - Previously you needed a WAL archive for that
- pg_receivexlog can follow a timeline switch