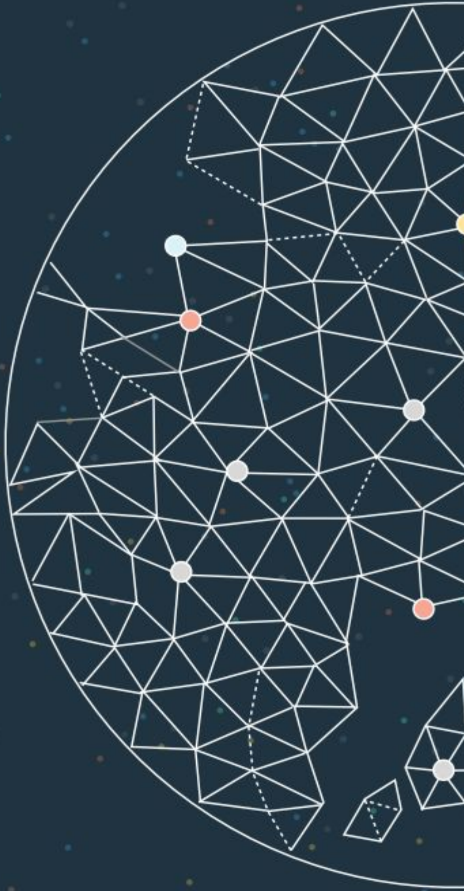
stellar

Stellar Core Data Flow

@graydon_pub, Developer @stellarorg



Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won’t usually need your flowcharts; they’ll be obvious.

Fred Brooks

Talk Overview

This is a talk about the *data* that stellar-core deals with.
It does not discuss SCP, Horizon, or applications built on Stellar.
It does not discuss cryptography, finance or trust.

It is to help you figure out
***what* is stored and transmitted *where*.**

Talk overview

1. Review: replicated state machines
2. Data types
3. Data formats
4. Places data lives
5. Movement of data
6. Bonus: external access to data

1. Review: replicated state machines

stellar-core is a

replicated state machine

State machine

Pure function of current state + input

$$F(State_n, Input_{n+1}) \dashrightarrow (State_{n+1}, Output_{n+1})$$

Deterministic

- Same state + input always makes same next-state + output
- Can replay any step, given state + input

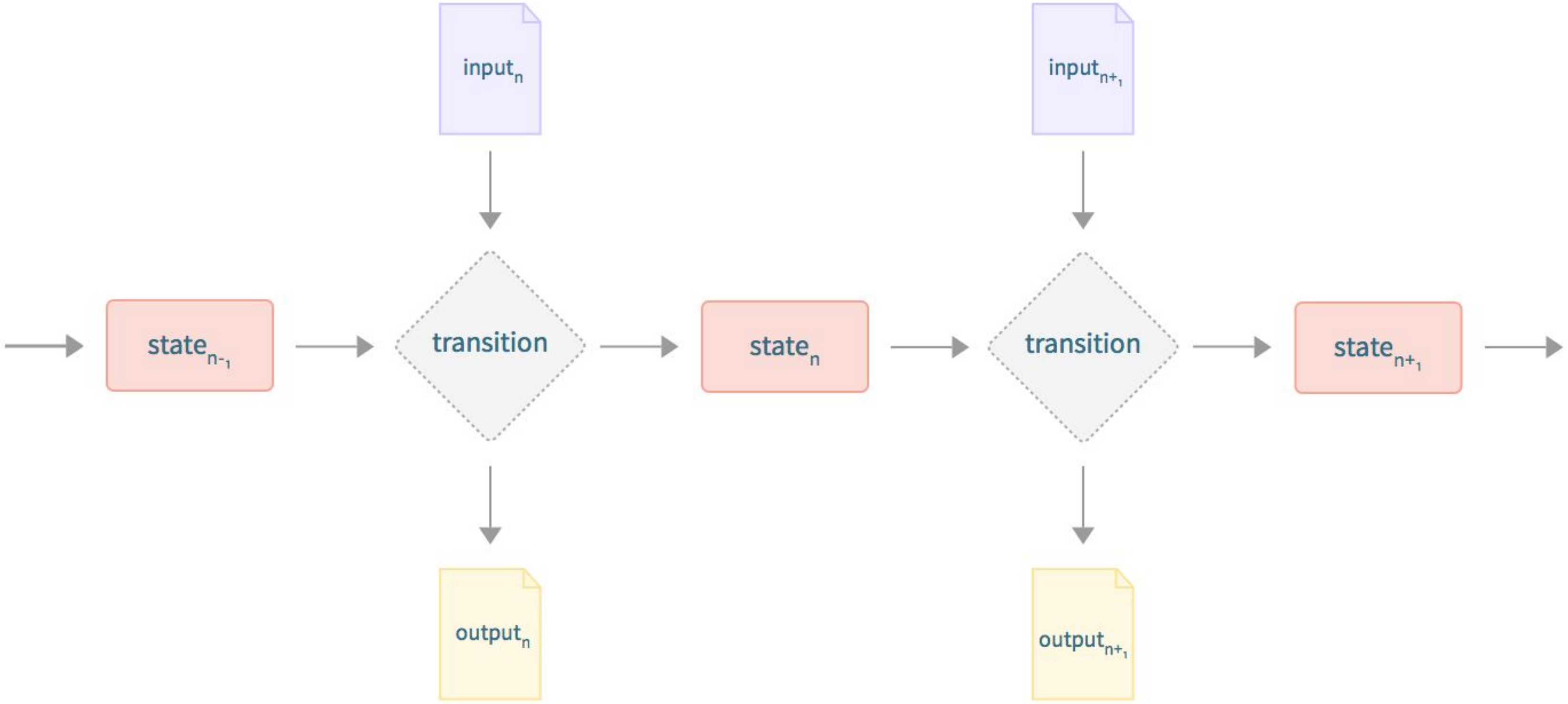
State machine

We will not discuss the function `F` much.

> Suffice to say it's "applying transactions."

The other 3 parts are *data*, which we'll talk about:

1. State
2. Input
3. Output



input$_n$
input$_{n+1}$
state$_{n-1}$
transition
state$_n$
transition
state$_{n+1}$
output$_n$
output$_{n+1}$

Replicas

Recall: stellar-core is intended as a *replicated* state machine

Meaning: keep multiple copies of state machine *and its data*

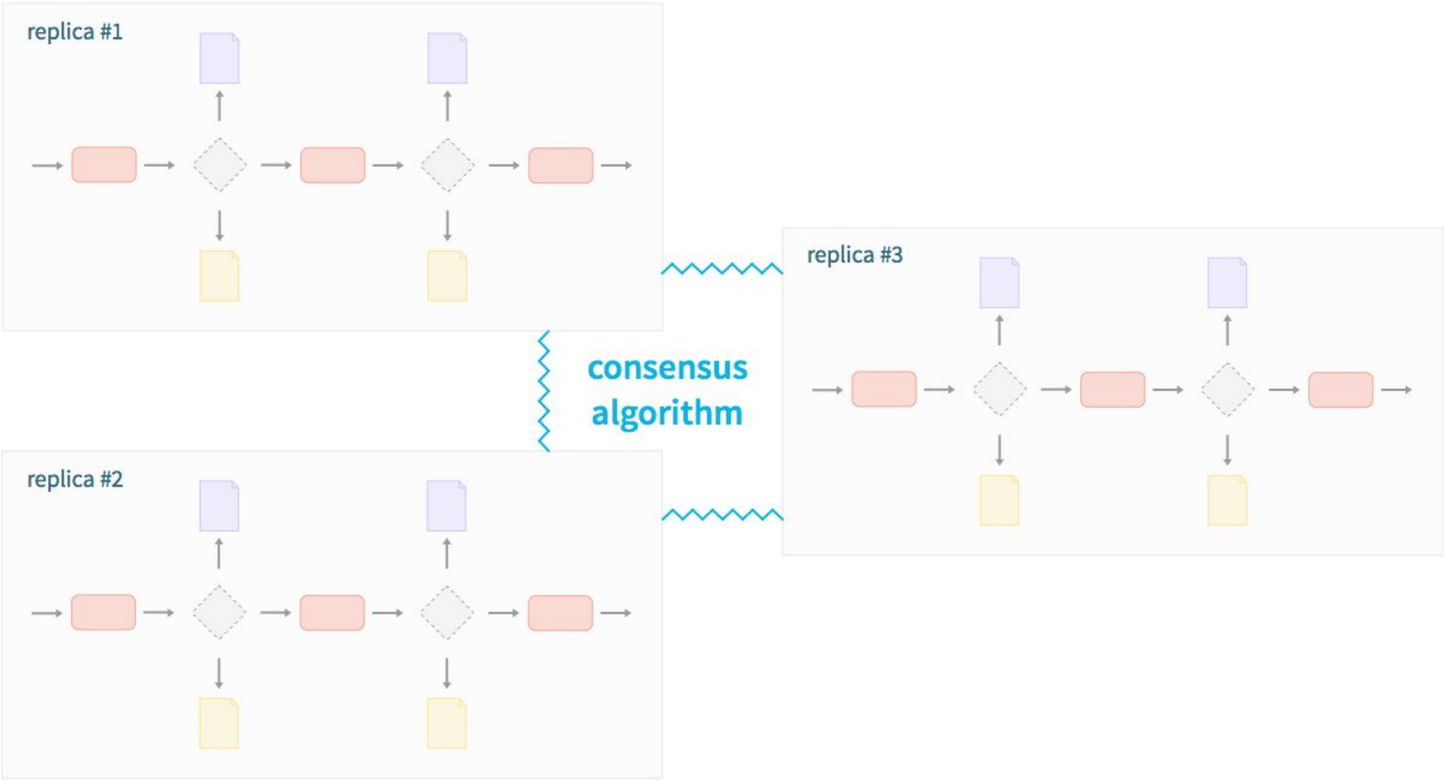
- On different physical computers
- Run at *same* time, in lock-step
- Run *same* function on *same* "input + state" data
- Produce *same* "output + next state" data

Replication is for reliability, decentralization

"lots of copies keeps stuff safe"

Replicas are coordinated by a *consensus algorithm*

ensures same current state and input on all replicas


replica #1
consensus
algorithm
replica #3
replica #2

stellar-core uses SCP for replica consensus

this talk is not about SCP

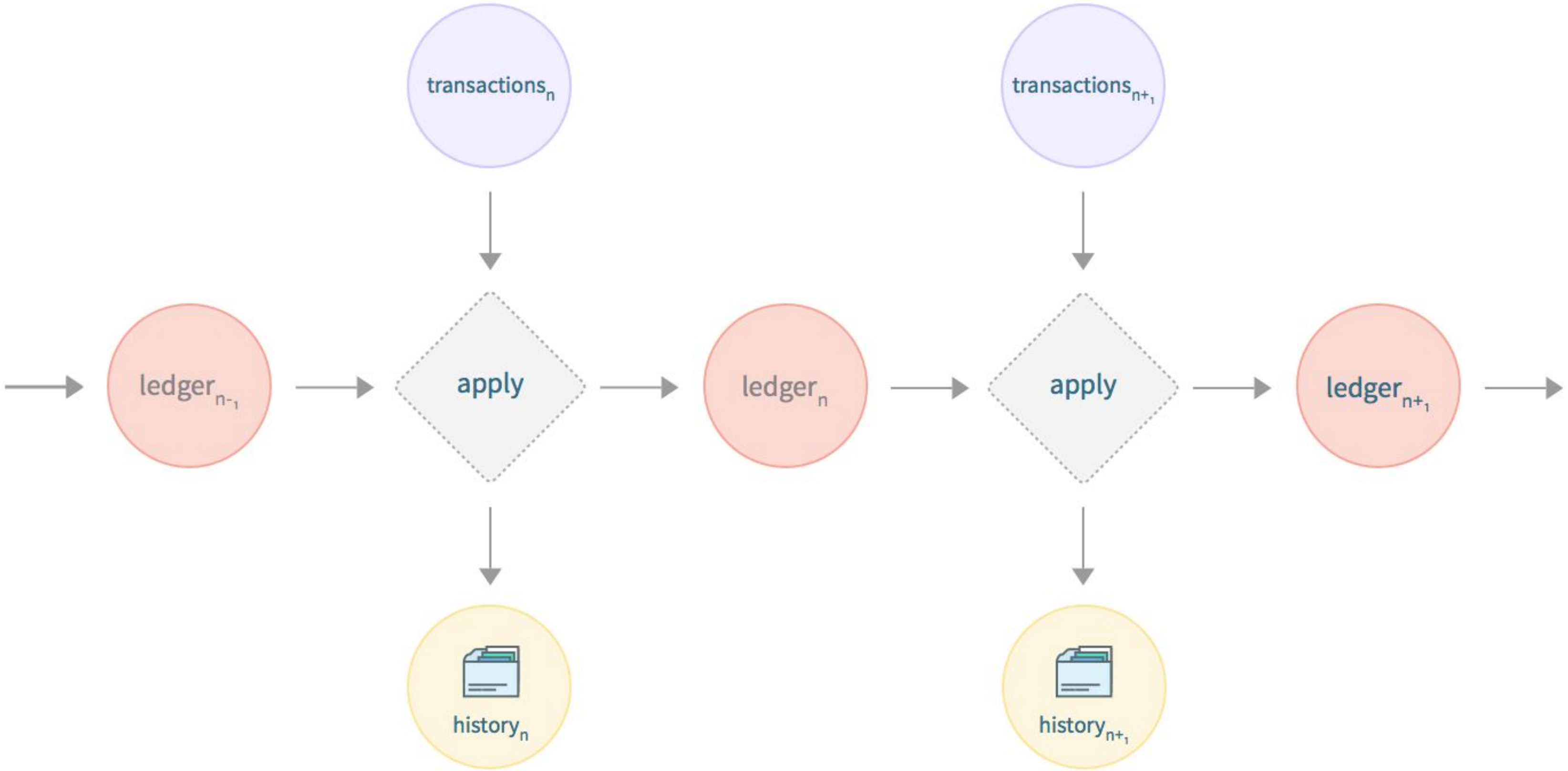
In the stellar-core state machine:

State = Ledger

Input = Transactions

Output = History

$F(Ledger_{n}, Transactions_{n+1}) \dashrightarrow (Ledger_{n+1}, History_{n+1})$



transactions$_n$
transactions$_{n+1}$
ledger$_{n-1}$
apply
ledger$_n$
apply
ledger$_{n+1}$
history$_n$
history$_{n+1}$

Every stellar-core peer follows this cycle

(endless loop, every 5 seconds)

1. Acquire consensus on state and input
2. Apply input to state
3. Emit output, advance to new state

Every stellar-core peer follows this cycle

(endless loop, every 5 seconds)

1. Acquire consensus on **ledger** and **transactions**
2. Apply **transactions** to **ledger**
3. Emit **history**, advance to new **ledger**

2. Data Types

Recall: data of stellar-core state machine

1. Ledgers (state)
2. Transactions (input)
3. History (output)

Ledger

Recall: this is the *state* data

Description of how-things-are at the present moment

Set of 3 kinds of *entries*:

- Accounts
- Trustlines
- Offers

Transactions

Recall: this is the *input* data

Is truly *data*: encoded descriptions of actions-to-perform

Handful of possible actions on ledger entries:

- Create/modify/delete entry
- Transfer amount between entries
- Miscellaneous others (inflation, set options, etc.)

History

Recall: this is the *output* data

Log of changes during each state-transition:

- Transaction set that was used as *input*
- Success or failure of each transaction, and its effects
- Compact description of *next state*

3. Data Formats

Data in stellar-core takes 2 forms:

1. **XDR**
2. **SQL**

plus a few auxiliary TOML and JSON files

XDR

External Data Representation

Generic binary serialization format[1]

Internet standard[2]

Driven by plain-text schemas

[1] Like ASN.1, Protocol Buffers, Thrift, Avro
[2] RFC 4506 / STD 67

SQL

Structured Query Language

Generic relational database access format

International standard[3]

Implies: **stellar-core always paired with a database**[4]

[3] ISO/IEC 9075

[4] Currently support PostgreSQL and SQLite

Uses of XDR

All 3 kinds of data in stellar-core are expressed in XDR:

- Transactions (input) received in XDR
- Ledger (state) stored on disk in XDR
- History (output) emitted in XDR

Plus all SCP and P2P network messages

Uses of SQL

Mostly[5] just the ledger (state)

Mostly[6] just read / written while applying transactions

[5] Some history (output) is also buffered there, on the way out

[6] Consensus does some reading in order to validate potential input

Wait, isn't the ledger in XDR?
Yes: the ledger is stored *twice*
In XDR *and* SQL, simultaneously

for two good reasons—we'll get to them

4. Places data lives

Stellar-core deals with data in 4 places

1. XDR in flight (between replicas)
2. SQL tables in a relational database
3. XDR files on local disk
4. XDR files in a "history archive"

XDR in flight

Peer-to-peer network *between replicas*

Messages flood to all peers

Mainly transactions & SCP messages

Held in memory until consensus

SQL tables in a relational database

Consulted during consensus

Modified during state-machine transition

Modified *atomically*: $Ledger_n \dashrightarrow Ledger_{n+1}$

Random-access, fine-grained

Fast: hundreds to thousands of updates per second

XDR files on local disk

So-called "buckets"

Store the ledger in *canonical form*

Duplicate of data stored in SQL tables

Needed for 2 operations[7]:

- Efficient, incremental *cryptographic hashing*
- Efficient, incremental *storage* and *transmission of differences*

[7] See *https://github.com/stellar/stellar-core/blob/master/src/bucket/BucketList.h*

XDR files in a "history archive"

Long-term, flat-file, mostly cold storage

User-defined backends[8]

Stores *checkpoints*: XDR buckets and XDR history logs

Mostly[9] write-once, read-many

Used by peers to catch up to one another

[8] Typically AWS S3, Google Cloud Storage, Azure Blob Storage, SCP/SFTP, etc.

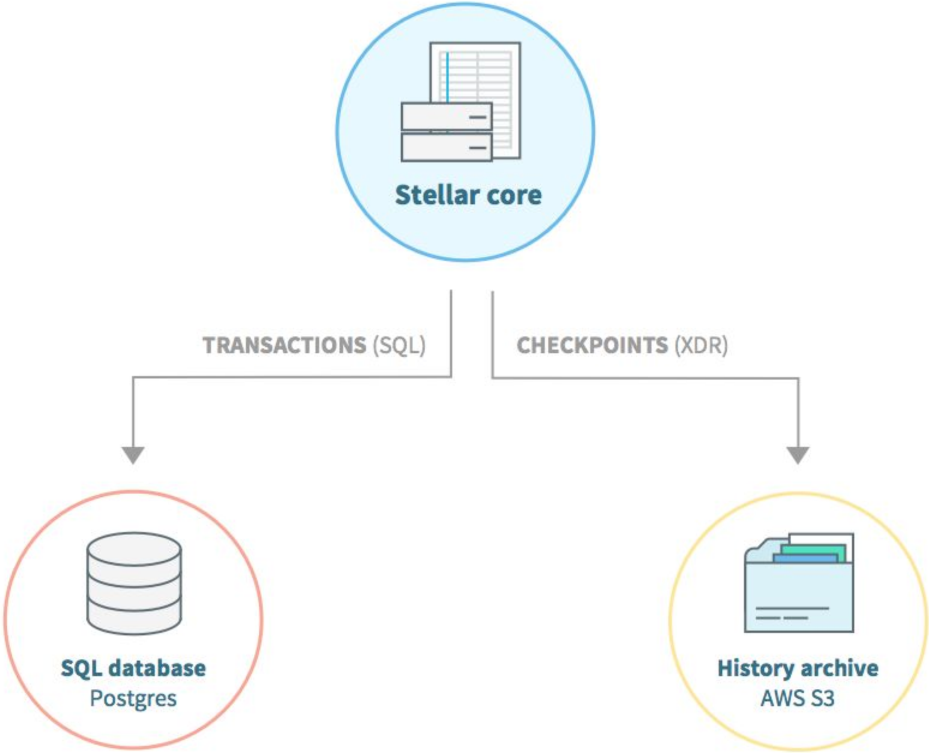
[9] A single JSON file is rewritten to point to the "most recent" checkpoint

Reiteration in case this was not clear

A stellar-core node **usually requires** two other storage facilities:

- A relational database[10]
- One or more history archives[11]

[10] SQLite is bundled and may be sufficient for small networks; PostgreSQL is recommended.
[11] At least configuring an archive to *read* from; *writing* to an archive is optional, but recommended.


Stellar core
TRANSACTIONS (SQL)
CHECKPOINTS (XDR)
SQL database
Postgres
History archive
AWS S3

5. Movement of data

Data moves in 5 interesting flows

1. History archives → Peers = **"Catchup"**
2. External clients → Peers = **"Submission"**
3. Peers → Peers = **"Flooding"**
4. Peers → Databases and local files = **"Applying"**
5. Peers → History archives = **"Publishing"**

Catchup

Happens when a peer is new or out of sync

Downloads[12] XDR history files from history archive

One of two operator-chosen modes, either:

- replays state-transitions in order, or
- snaps to most recent state[13]

[12] Archive-specific, configured by user. Usually HTTP GET or similar.
[13] This mode only downloads differences, one of the two reasons for duplicating the ledger in buckets.

Submission

Happens when an external client has new transaction

Contacts peer through HTTP (likely via Horizon)

Sends XDR representation of transaction

Receives status code indicating "rejected" or "pending"

Flooding

Happens continuously

Peers hold long-lived TCP connections to one another

All messages are XDR, repeated to all peers

- Transactions: flood as they're submitted
- SCP messages: a burst of activity every 5 seconds

Applying

Happens when SCP decides on consensus state + input

- About every 5 seconds

Transactions in memory applied to ledger in SQL database

Duplicate copies of changed ledger entries put in XDR buckets[14]

Transactions and results written to accumulating XDR checkpoint

[14] Cryptographic hash of ledger is efficiently calculated here: the other reason for duplicating the ledger in buckets.

Publishing

Happens every 64 ledgers

About every 5 minutes

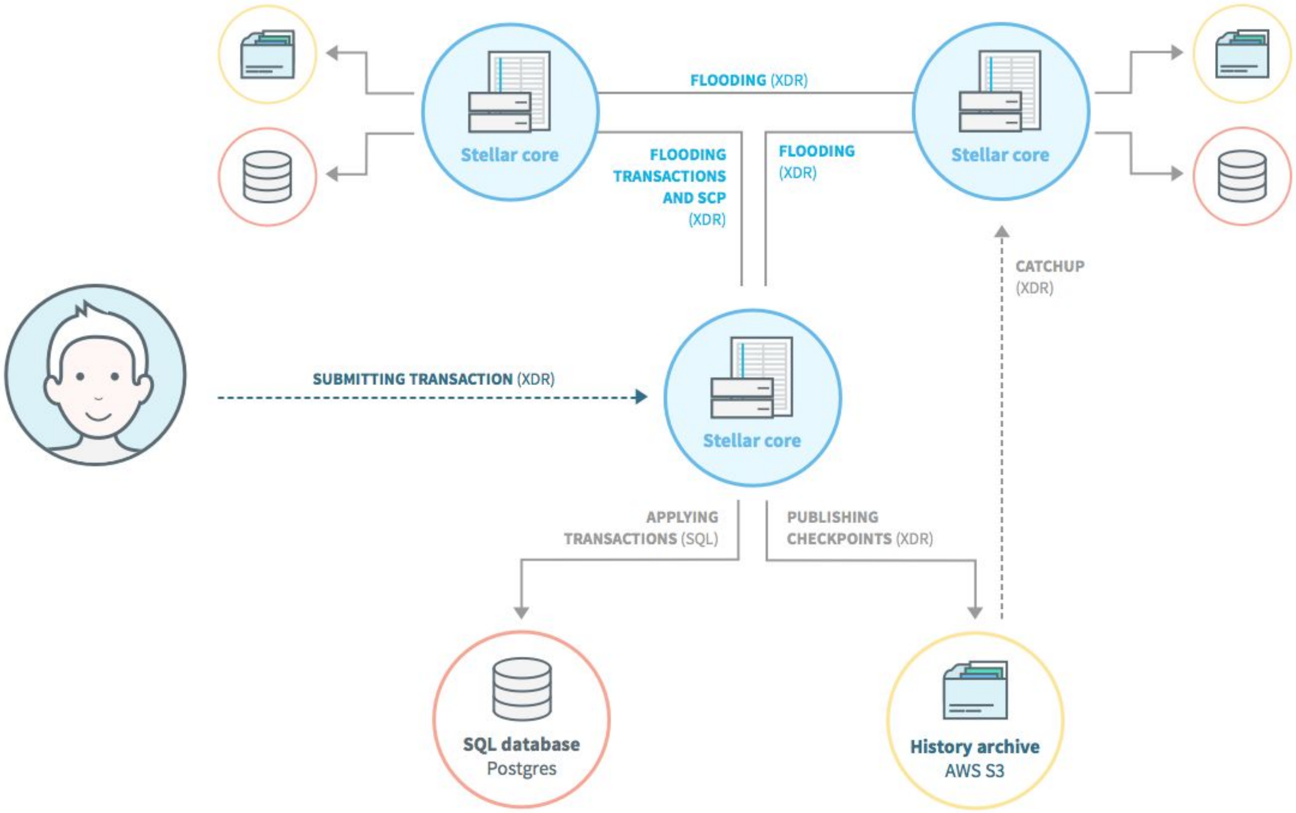
Uploads[15] accumulated checkpoint to history archive

Includes 64 state-transitions worth of history, compressed

All transactions, results, and any new buckets[16]

[14] Archive-specific, configured by user. Usually HTTP PUT or similar.
[15] Only sends buckets differing from previous checkpoints.

Most-complicated diagram time!


FLOODING (XDR)
Stellar core
FLOODING
TRANSACTIONS
AND SCP
(XDR)
FLOODING
(XDR)
Stellar core
CATCHUP
(XDR)
SUBMITTING TRANSACTION (XDR)
Stellar core
APPLYING
TRANSACTIONS (SQL)
PUBLISHING
CHECKPOINTS (XDR)
SQL database
Postgres
History archive
AWS S3

6. Bonus: external access to data

It may seem a little odd that basic functions like "catchup" go through history archives.

History archives serve several roles

Ensuring reliable backups are made

Minimizing risks of single-node failure

Controlling storage costs for largest data set (history)

Isolating catchup I/O load away from P2P flooding

Providing very simple external access to data

A moment about that last point

Stellar is intended as a broadly interoperable system

Simplicity, transparency, standardization are key

Want there to be zero barriers to "getting the data"

- Even if consensus network is offline
- Even if stuck behind a firewall
- Even if polling via shell scripts and duct tape

Benefits of flat files

You do not need to "talk to" stellar-core to get data

Command-line tools can download from history archives

- `curl/wget` usually fine

Reading/interpreting involves only gzip, JSON and XDR

- stellar-core will dump an XDR file as plain text, offline
- Decoding XDR is pretty straightforward anyways

Go forth and experiment!

XDR schemas are public[17]

Archives are just directories full of XDR files

If you want to see the transactions in ledger 0x3127:

- *transactions/00/00/31/transactions-00003127.xdr.gz*

[17] See *https://github.com/stellar/stellar-core/tree/master/src/xdr*

Fini