Package Relay Intro

Now (Presentation)

1. Motivation, Concepts, Definitions
2. Sub-Projects and what problems they solve
    a. Mempool validation & policy
    b. p2p protocol
    c. package RBF, v3
3. Progress so far

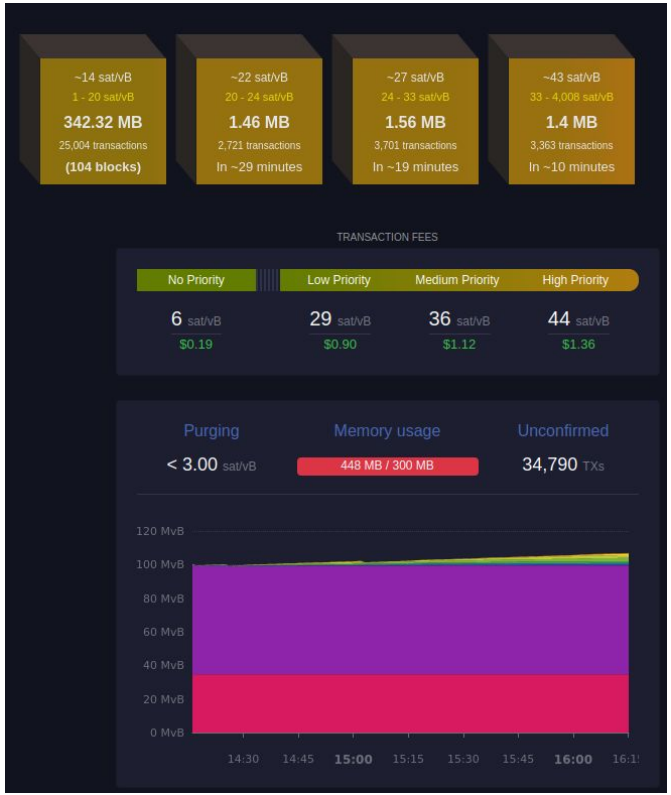
Later (Breakout Group)

4. BIP331 Walkthrough
5. Implementation Walkthrough & Open Questions

1. Motivation, Concepts, Definitions

What problems are we trying to solve?

Problem 1: CPFP doesn’t work when our mempool’s full.



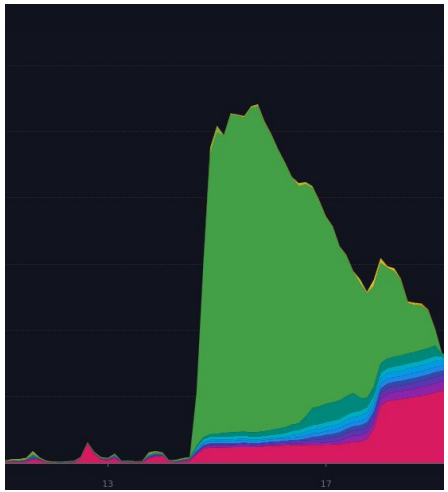
(and it is sometimes full nowadays)

Problem 2: Pinning

“The counterparty can’t cheat you as long as you get your tx confirmed on time”

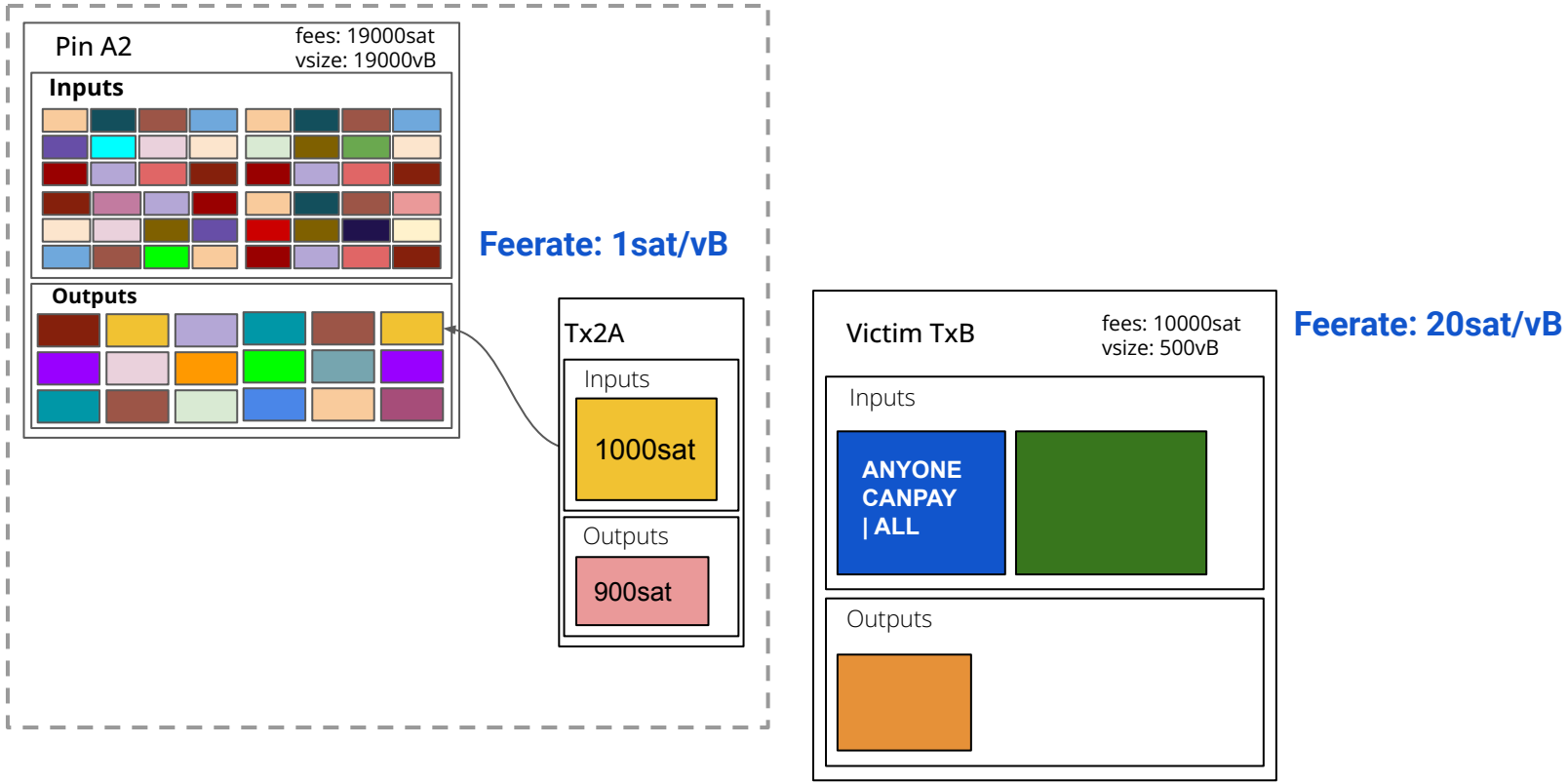
== The counterparty wins if your tx never gets mined

- Bad Luck
- Intentional Censorship



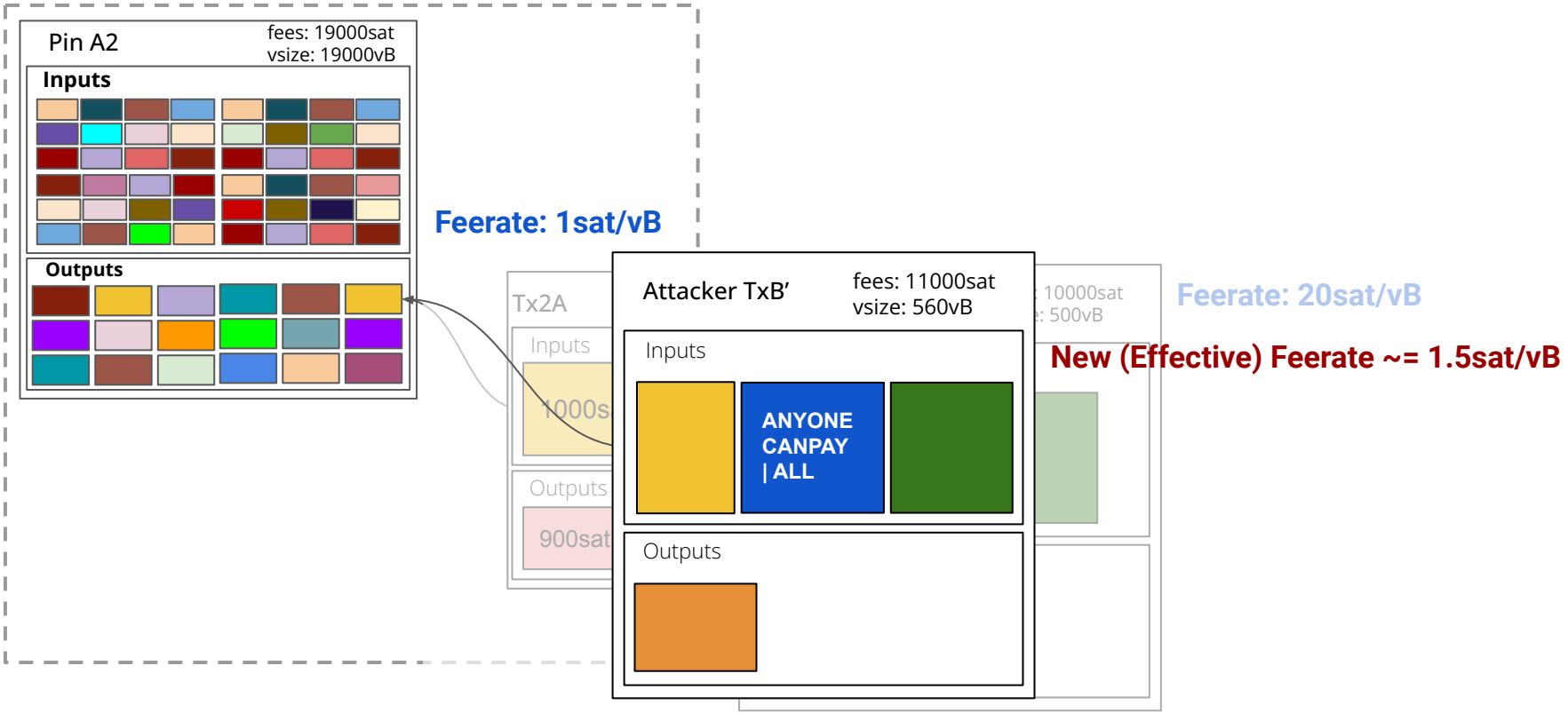
Problem 2: Pinning

Example 1: ANYONECANPAY = anyone can add a huge ancestor



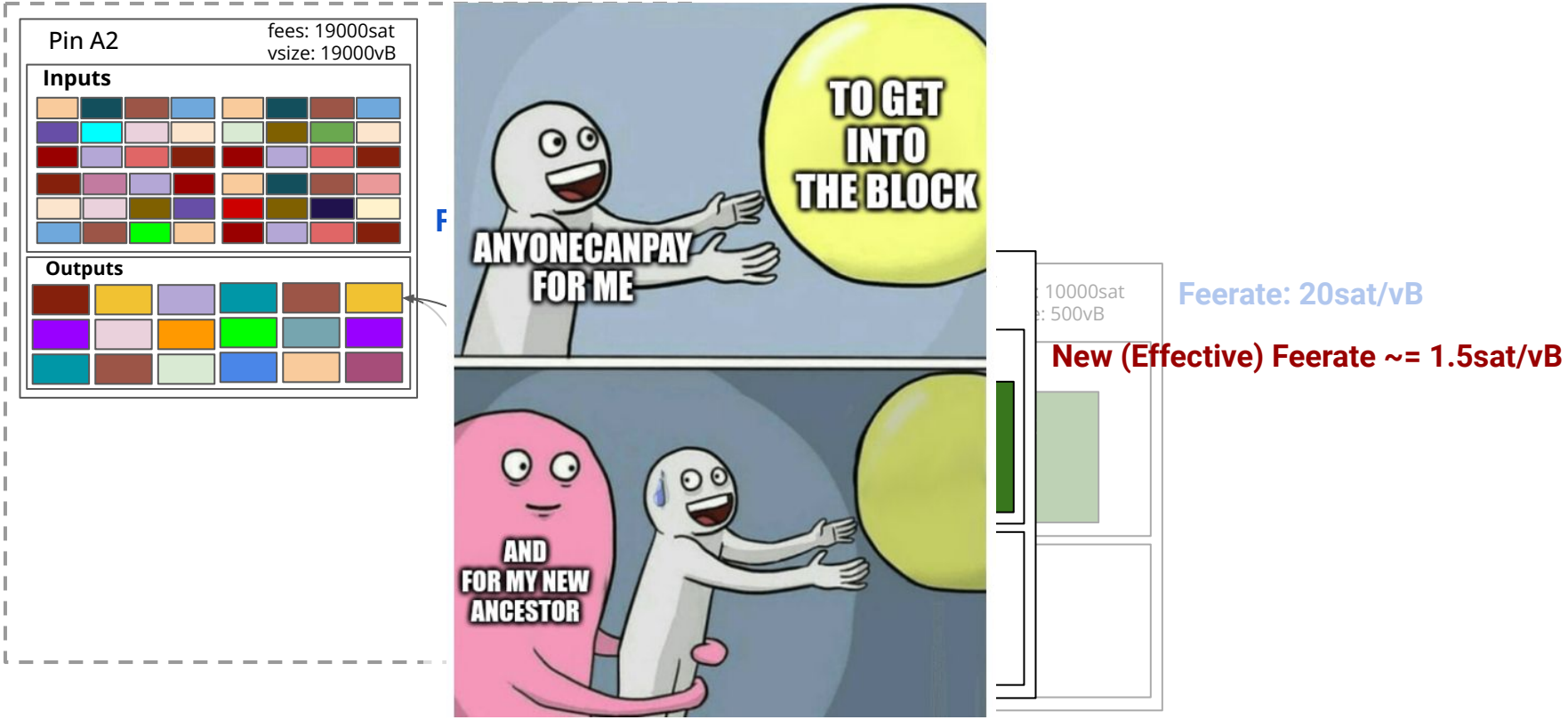
Problem 2: Pinning

Example 1: ANYONECANPAY = anyone can add a huge ancestor



Problem 2: Pinning

Example 1: ANYONECANPAY = anyone can add a huge ancestor



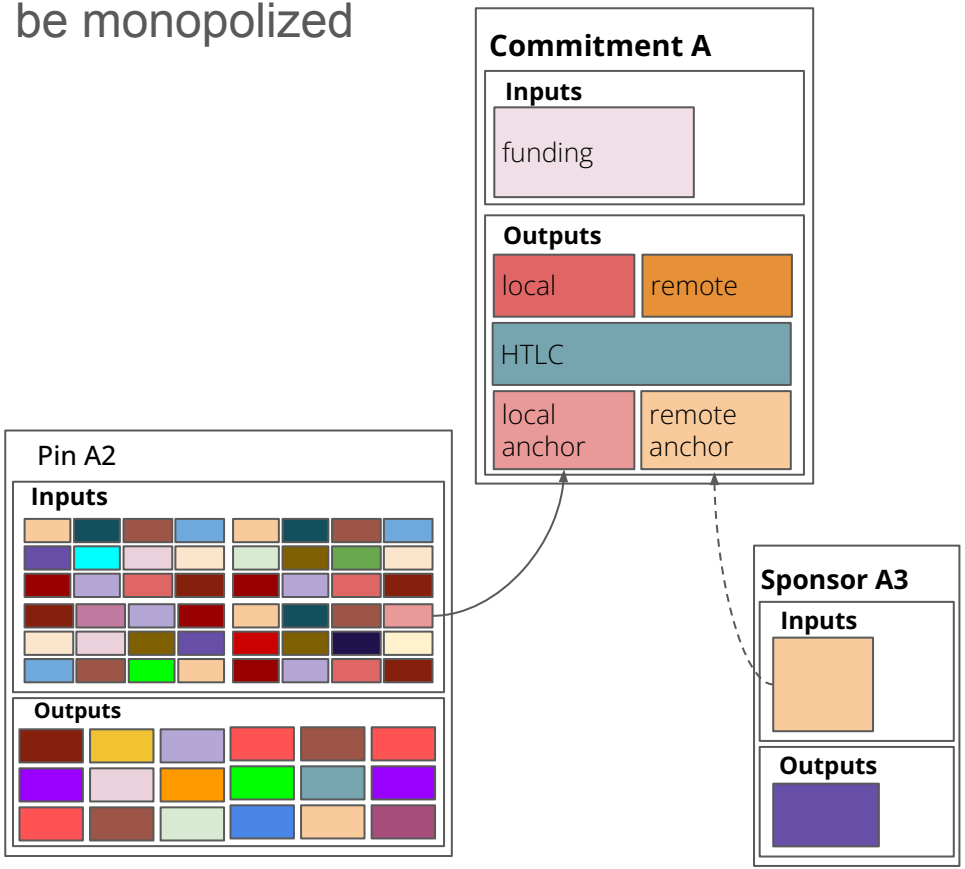
Problem 2: Pinning

Example 2: shared descendant limit can be monopolized

descendant limits are 25 txns or 101KvB

somebody else can fill up that limit

CPFP carve out

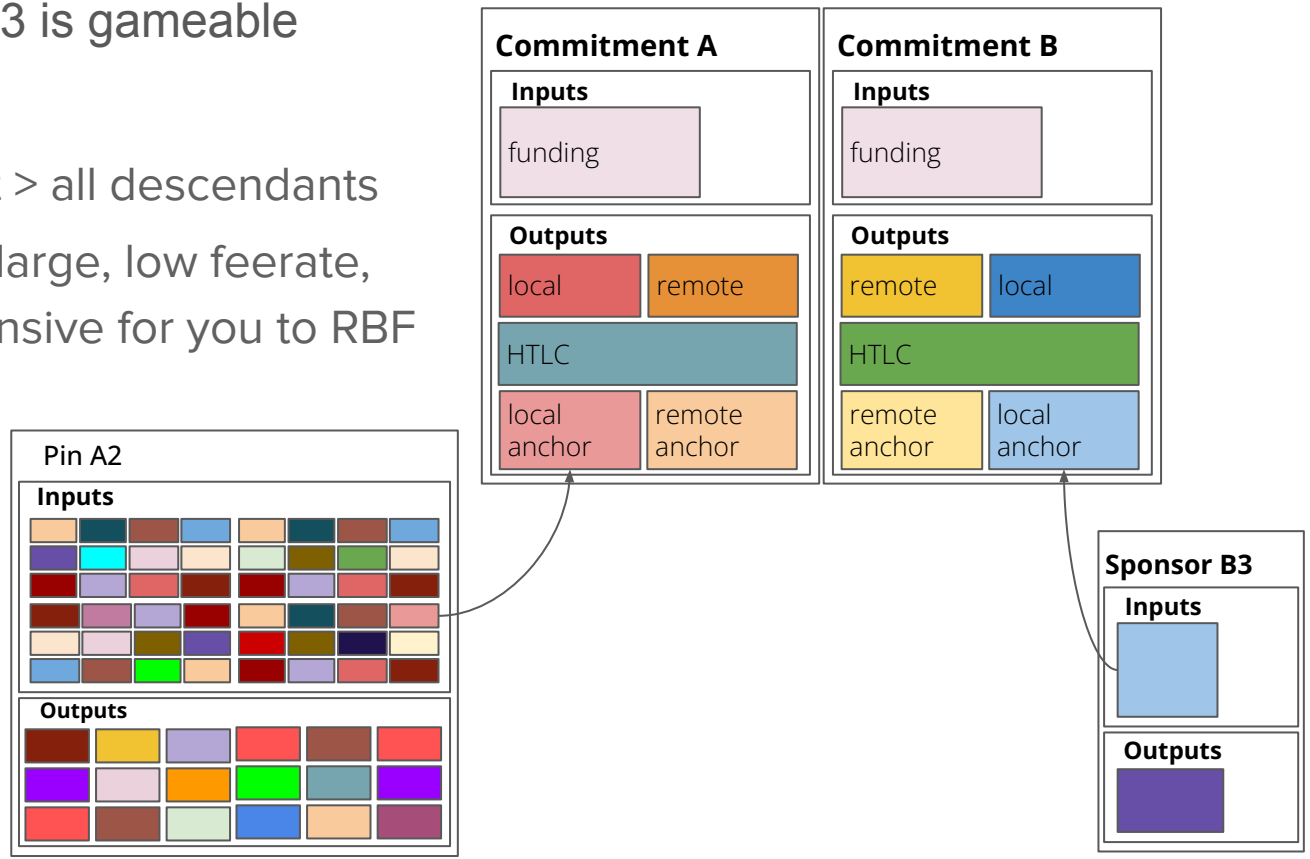


Problem 2: Pinning

Example 3: RBF Rule 3 is gameable

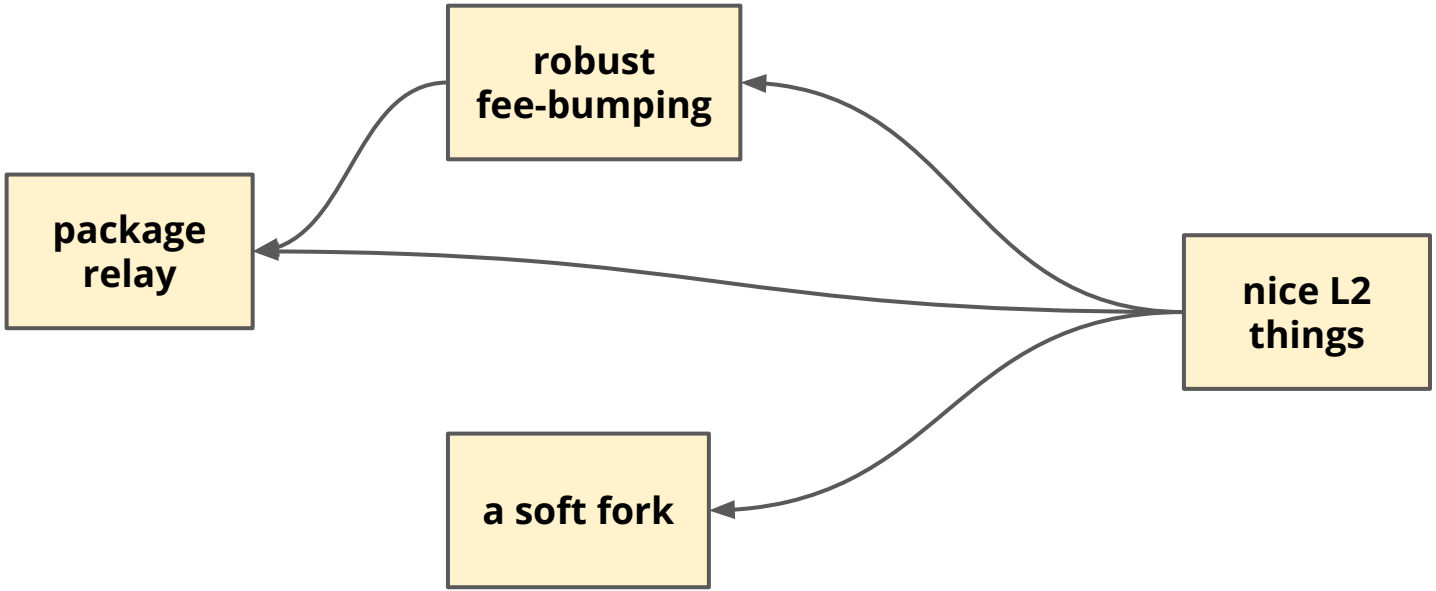
replacement fees must > all descendants

descendant(s) may be large, low feerate, making it unfairly expensive for you to RBF



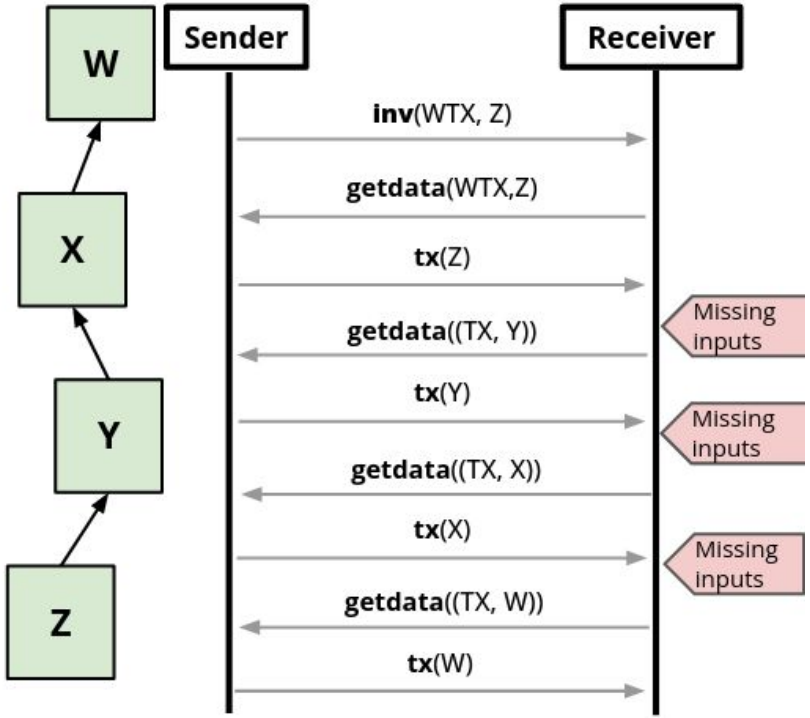
Most L2s have pinning problems

Most proposed L2s have this dependency graph:



Problem 3: Get rid of txid-based relay

- We want to avoid txid-based relay
  - Can’t deduplicate txid and wtxid that correspond to the same tx
  - Can’t deduplicate transactions that only differ in witness
- Last use case: orphan-handling. We request missing inputs by txid

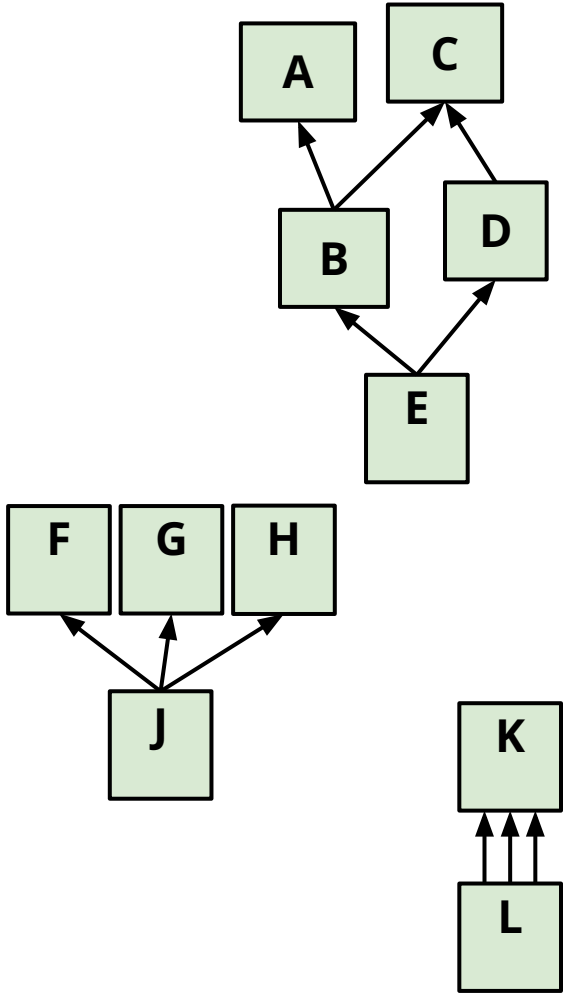


Definitions

Pinning Attack: censorship in which attacker takes advantage of mempool policy limitations to prevent a tx from getting mined or entering a mempool

- NOT they paid more to get theirs confirmed

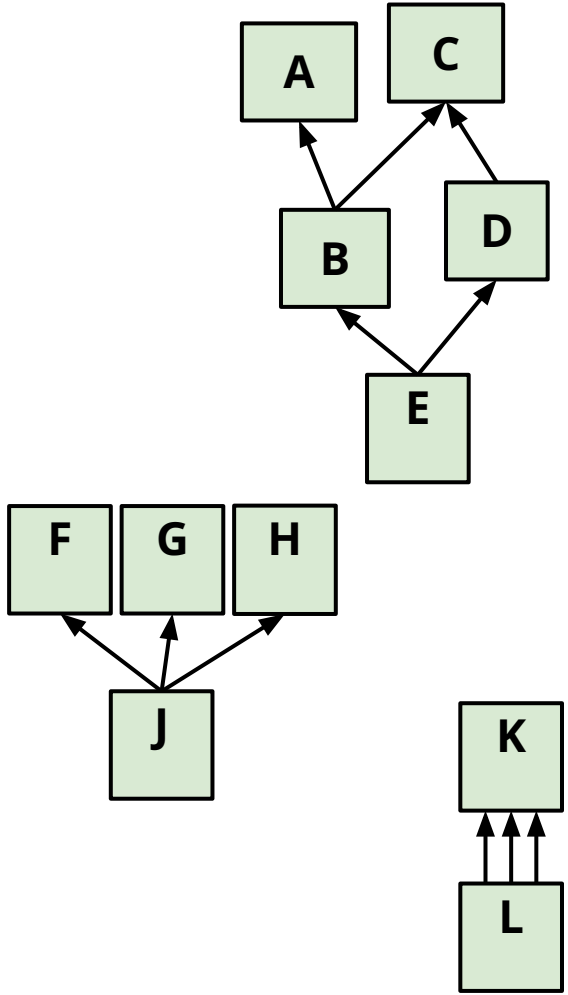
Package = a list of transactions that can be represented as a connected DAG



Package = a list of transactions that can be represented as a connected DAG

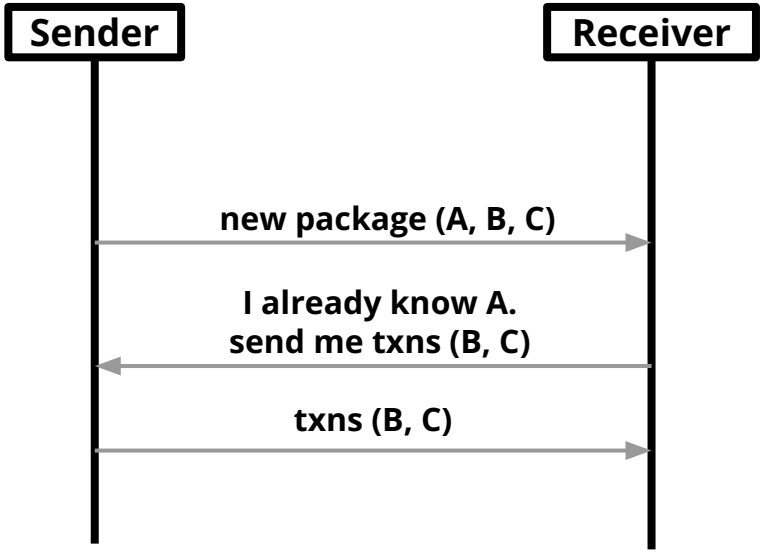
Ancestor Package = a package of 1 tx and its (unconfirmed) ancestors

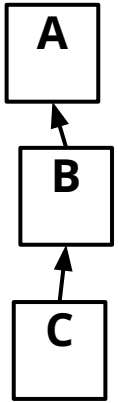
Descendant Package = a package of 1 tx and its (unconfirmed) descendants



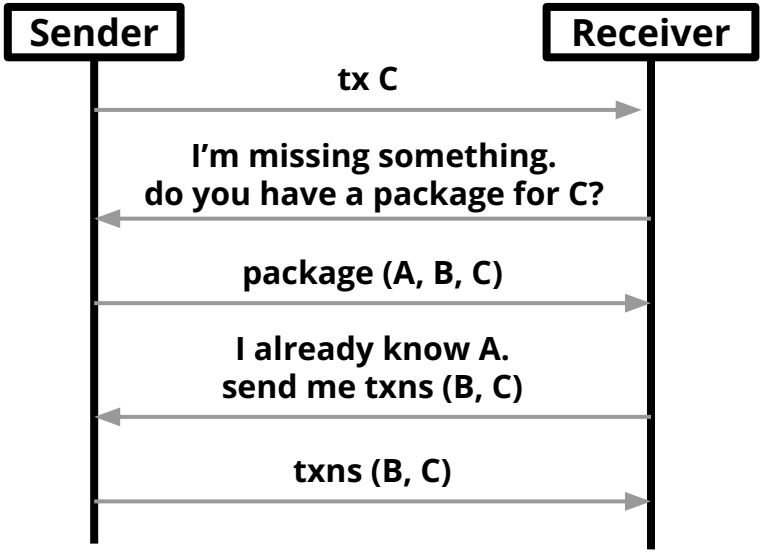
Package Relay = Relaying and validating packages together

Sender-initiated: nodes proactively announce packages that they think their peers should download and validate together





Receiver-initiated: nodes can request packages when they recognize they're missing something



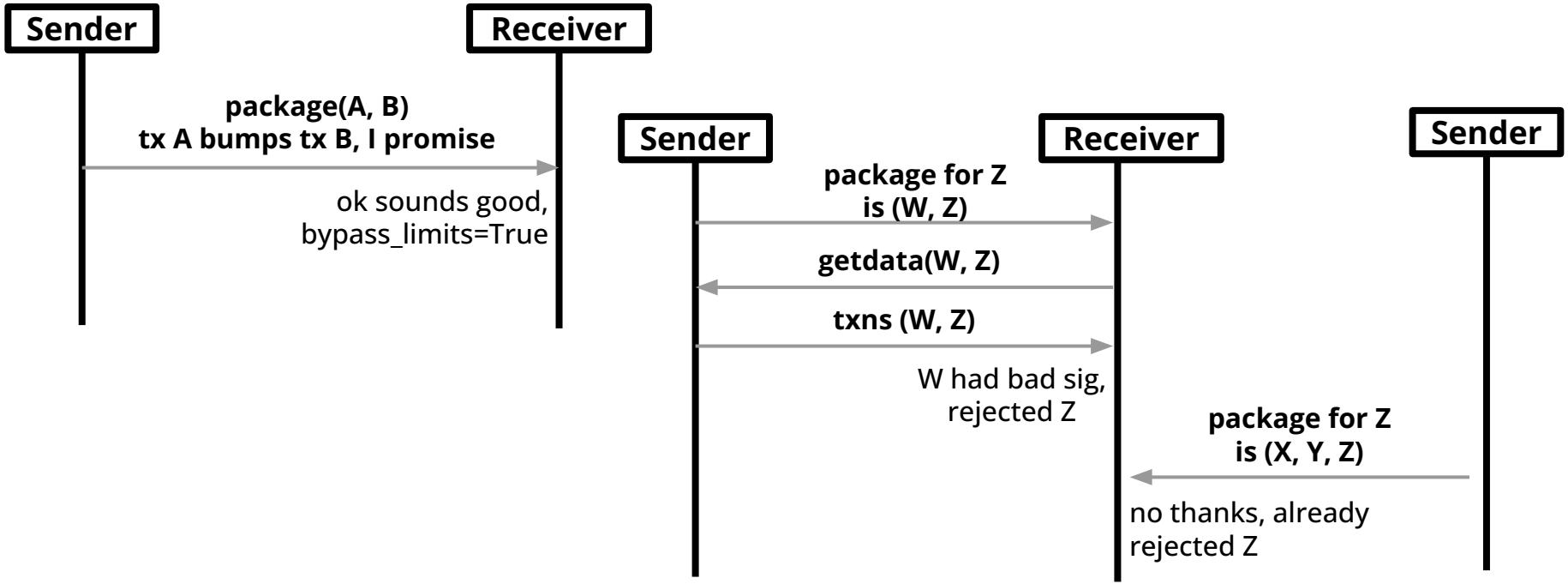
Why is there so much code?

Concerns that make tx relay complex:

- Don’t get DoSed
- Be somewhat bandwidth-efficient:
  - Only download tx data 1x per node
  - Only announce transactions 1x per connection
- Don’t accidentally enable censorship
  - e.g. when caching rejections or expiring announcements
- Try not to leak information
  - Timing of when you received a transaction
  - Exactly what transactions you have in your mempool

Why is there so much mempool code?

Peers are not trusted to provide correct information.



2. Sub-Projects and what problems they solve

Sub-Projects

Package CPFP: Accept packages, allow a child to bump a parent past mempool min feerate

P2P Package Relay: Additional protocol messages to {request, provide, download} package information on p2p network

Package RBF: Also allow a child to pay for parents' conflicts (treat as 1 aggregated tx). However, painful pinning issues still exist.

Sub-Projects

v3 Policy: Additional topology restrictions

- make it feasible to allow 0-fee transactions in a DoS-resistant way
- naturally more pinning-proof

Ephemeral Anchors (instagibbs): allowing v3 anchors to be 0-value, eliminating many issues with current anchor outputs

- eltoo

Progress so far

We’ve done a lot, and there’s a lot more to do

https://github.com/bitcoin/bitcoin/issues/27463

Big Branch if you prefer to look at code:
https://github.com/glozow/bitcoin/tree/package-relay-orphan-eviction

Package Relay Review Session

3. BIP331 Walkthrough

https://github.com/bitcoin/bips/pull/1382

Goals for this session

- Make review easier
  - I have a lot of PRs open, it confuses people
  - Explain architecture, interfaces, and non-obvious design decisions
  - Discuss, answer any questions
- Narrow in on the approach
  - Transaction Relay is really scary, I don’t want to screw this up
  - Sometimes projects suffer from approach feedback being too late
    - Including this one (I thought child-with-parents would be less complex, turns out it’s more)
- Figure out best path forward
  - The problems we want to solve require lots of changes
  - Some things can be worked on in parallel

Functional Milestones

1. Mempool, Validation, Policy
    a. Cool package policies are accessible through submitpackage RPC
2. Make orphan-handling more reliable
    a. Eviction improved to make better use of orphanage memory
3. Add TxPackageTracker to manage orphan handling
    a. TxOrphanage is a private data store for unvalidated txns
    b. Use all announcers as potential request candidates
4. Negotiate package relay, use ancpkginfo to resolve orphans
    a. Use ancpkginfo to download transactions (1 at a time)
    b. Never use txid-based relay except when package relay isn’t available
5. Download and validate packages
    a. Cool package policies are accessible through p2p

For each step,

- design decisions
- open questions

1. Mempool, Validation, Policy
2. Make orphanage more reliable
3. Add TxPackageTracker to manage orphan handling
4. Negotiate package relay, use ancpkginfo to resolve orphans
5. Download and validate packages

1. Mempool, Validation, Policy

Don’t allow anything below min relay feerate (#26933)

- Package policy allows you to bypass mempool min feerate
- Why not minrelay feerate?
  - RBF can leave it unbumped
  - RBF can leave it somewhat-bumped s.t. it’s not evicted, but will also never get selected by BlockAssembler
  - Problem exists due to just how our eviction/selection algorithms differ.
  - No easy solution. A full solution is complex (see Suhas/Pieter presentation)
- As usual, the edge cases don’t apply to v3 due to its topological simplicity, so they will be allowed to be below min relay feerate.
  - Anybody who *needs* 0-fee transactions most likely wants to use v3 anyway.

Persist CPFP'd transactions across restarts (#27476)

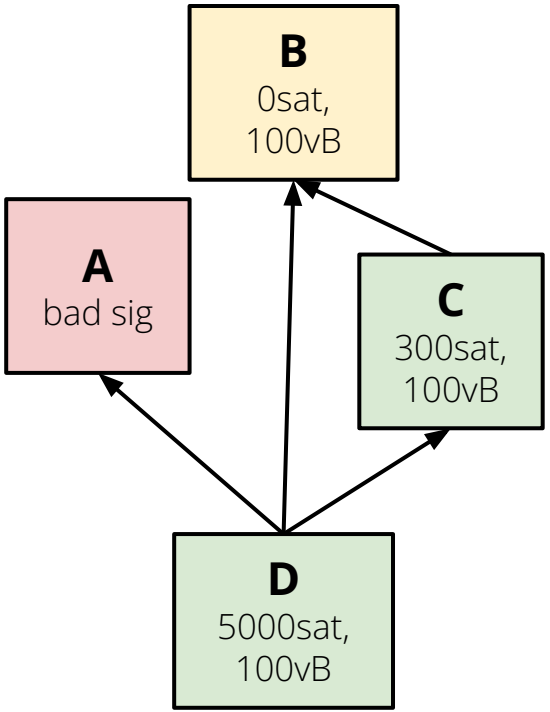
When loading from mempool.dat, we call ATMP for transactions 1 at a time.

- Anything low feerate is rejected, even if there’s a fee-bump later in the mempool.dat
- #27476 just does bypass_limits=True and calls TrimToSize() at the end
- This is simple and works well, but not for ephemeral anchors

Q: Should we add a new mempool.dat format for packages?

Validate txns with their in-package ancestor sets (#26711)

We want to handle this case:



Allow any ancestor package (#26711)

- Interface: you can throw any ancestor package at the mempool and it will do something sensible
- Why not required to be sorted?
  - You can't verify whether it's sorted without first establishing it's an ancestor package, and you can't verify it's an ancestor package unless you have the same chainstate.
  - We shouldn't require chainstate syncing for tx relay, and if we've already downloaded the package we might as well try to validate it.

2. Make Orphanage more reliable

Orphanage Today

- When tx is missing inputs, add to orphanage and request parents
  - If any parents were rejected, don’t
- On each tx acceptance
  - Orphanage tracks who provided orphan - adds it to work set
  - On peer’s next ProcessMessages(), first process orphan before other messages
    - 1 orphan validation (accept or reject) per turn
    - see #26551
- Maximum 100 orphans
  - Each must be within standard size
  - Theoretical memory bound is 100 * 400KB
  - If exceeded, orphanage evicts one at random

Orphanage Problems

Has the most important property, "DoS Resistant" i.e. peers cannot cause the orphanage to grow unbounded

Lacks some desired properties:

- **Some Degree of Reliability:** A peer can still make your orphanage useless by sending you *tons* of orphans
  - We need reliability since orphanage is in “critical path” of package relay
  - Some txns will only propagate using package relay
- **Effective Usage of Resources:** Does not protect the actual scarce resource, which is memory
  - Theoretical maximum is 100 * 400KB, but we evict much sooner than that
  - Most “normal” transactions are just a few hundred bytes

Option 1: Protect orphans when in use for package relay

When package request is in flight, protect any orphanage transactions

- Need to track how much orphanage data is being protected per package, e.g. stop at 101KvB
- Need to limit amount per peer
  - 6 packages/peer x 125 peers x 101KvB/package x 4Wu/1vB = **3GB is our theoretical limit**
  - Pretty weird if our orphanage can be 10 times larger than our mempool…
  - 1 package/peer = 50MB
- Max 1 in-flight package per peer sucks
  - Likely each node will have a few package relay peers who they relay all their packages.
  - During high volume, we'll just drop a lot of transactions

Option 2: Use orphanage memory more effecitvely

- Limit number of bytes used by orphans instead of count, so we don't trim until we're *actually* reaching resource limits
- Track amount of orphans provided per peer, have a soft limit per peer
  - e.g. 2 * max standard size
- When we reach global limit, only evict orphans from “overloading” peers
  - If nobody is overloading, evict randomly as usual
  - If we have 1 package relay peer sending 20 packages right now, but orphanage is otherwise empty, they all succeed
  - If an orphan is announced by multiple peers, they’re less likely to be evicted (it’s less likely all of them are overloading)

Q: should larger = more likely to be evicted? Concerned about performance.

Option 3: Token bucket

- Also limit by bytes used instead of count
- Peers are given “tokens” representing storage in our orphanage
  - Tokens spent if they add something to our orphanage
  - Tokens returned if orphan is accepted, otherwise not
  - Tokens replenished at a steady rate, max out at some amount

Q: for orphans announced by multiple peers, how do we count tokens?

3. Add TxPackageTracker to manage orphan handling

TxPackageTracker (before package relay)

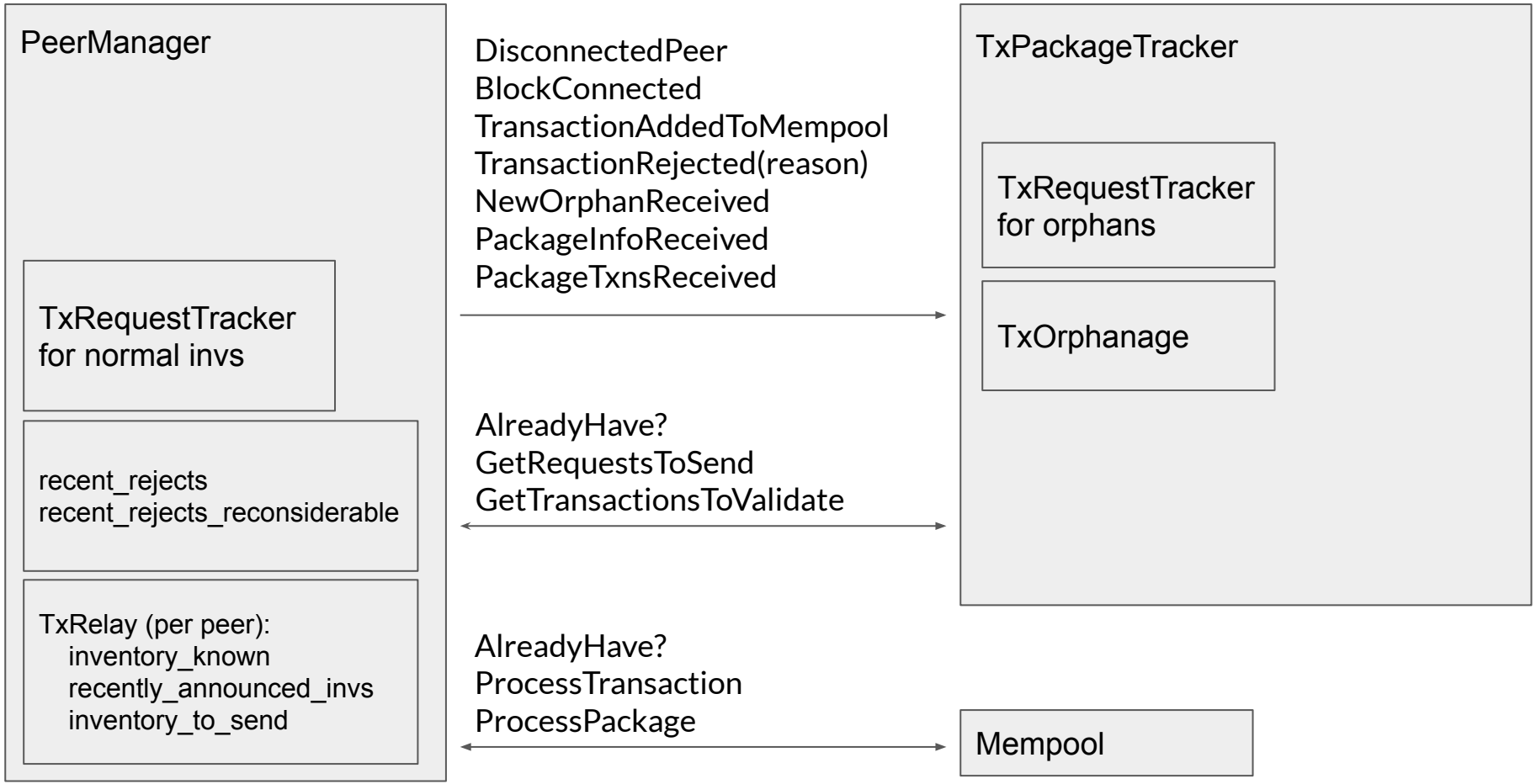
Public Interface:

- Orphanage Wrapper functions
  - OrphanageHaveTx()
  - DisconnectedPeer()
  - BlockConnected()
  - GetTxToReconsider()
- GetOrphanRequests()
  - like GetRequestable(), returns getdata requests

Internally:

- TxRequestTracker orphan_request_tracker
  - an Announcement is an orphan we need to resolve
  - is_wtxid is whether we're requesting via package relay or via parent txids
  - PriorityComputer allows us to prefer requests to outbounds, etc.
- TxOrphanage
  - Data store managed entirely by TxRequestTracker
  - Kept in sync with orphan_request_tracker

Vision for Interfaces (with package relay)



4. Negotiate package relay, use ancpkginfo to request orphans’ ancestors

recent_rejects vs recent_rejects_reconsiderable

Design Goals:

- Don't repeat work, i.e. don't download or validate transactions more than once if we can know they're going to be rejected
- Don't allow censorship due to overzealous rejection caching

recent_rejects vs recent_rejects_reconsiderable

Design Goals:

- Don't repeat work, i.e. don't download or validate transactions more than once if we can know they're going to be rejected
- Don't allow censorship due to overzealous rejection caching

Problems with recent_rejects:

- Low fee stuff goes in there too. If we receive a low-fee parent before a high-fee child, we'll reject both in orphan handling.
- We can't just not add low-fee stuff, then we'll allow repeat downloads.

recent_rejects vs recent_rejects_reconsiderable

recent_rejects: single transaction ids. rejected, and a package won't make it valid.

Add any tx, by itself or in package:

- failed for any reason other than too low fee

If contains:

- On inv(tx): don't download tx
- On ancpkginfo: Reject immediately if any of the wtxids is in here

recent_rejects_reconsiderable: single transaction ids and package ids. invalid, but could be reconsidered in a package.

Add a tx or package:

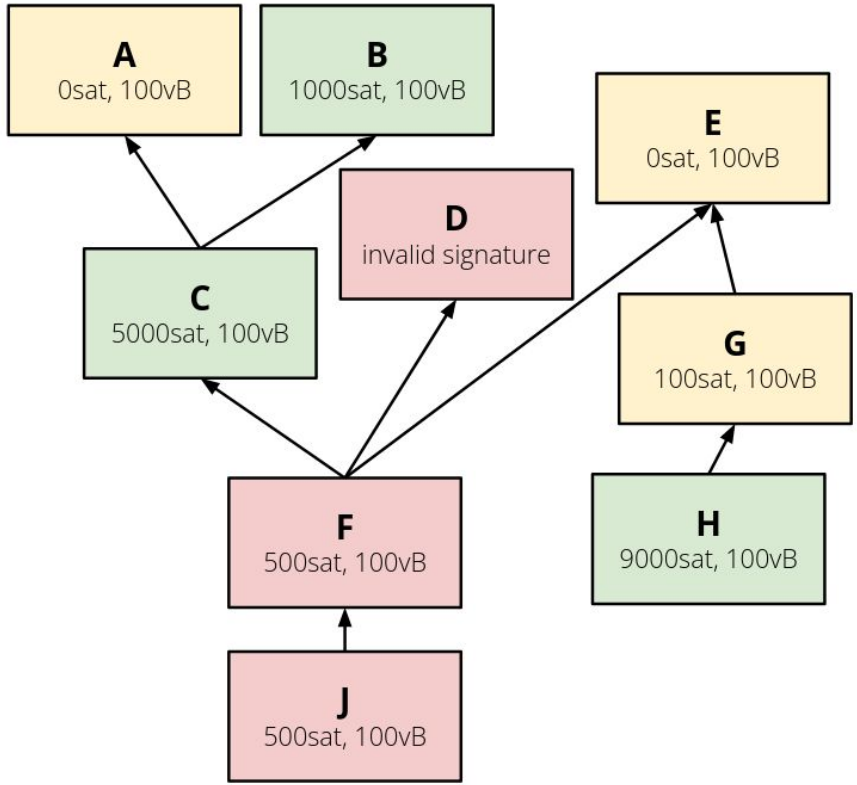
- Tx or subpackage that was too low fee
- Ancpkginfo that didn't get fully accepted
- Any subpackage that didn't get accepted

If contains:

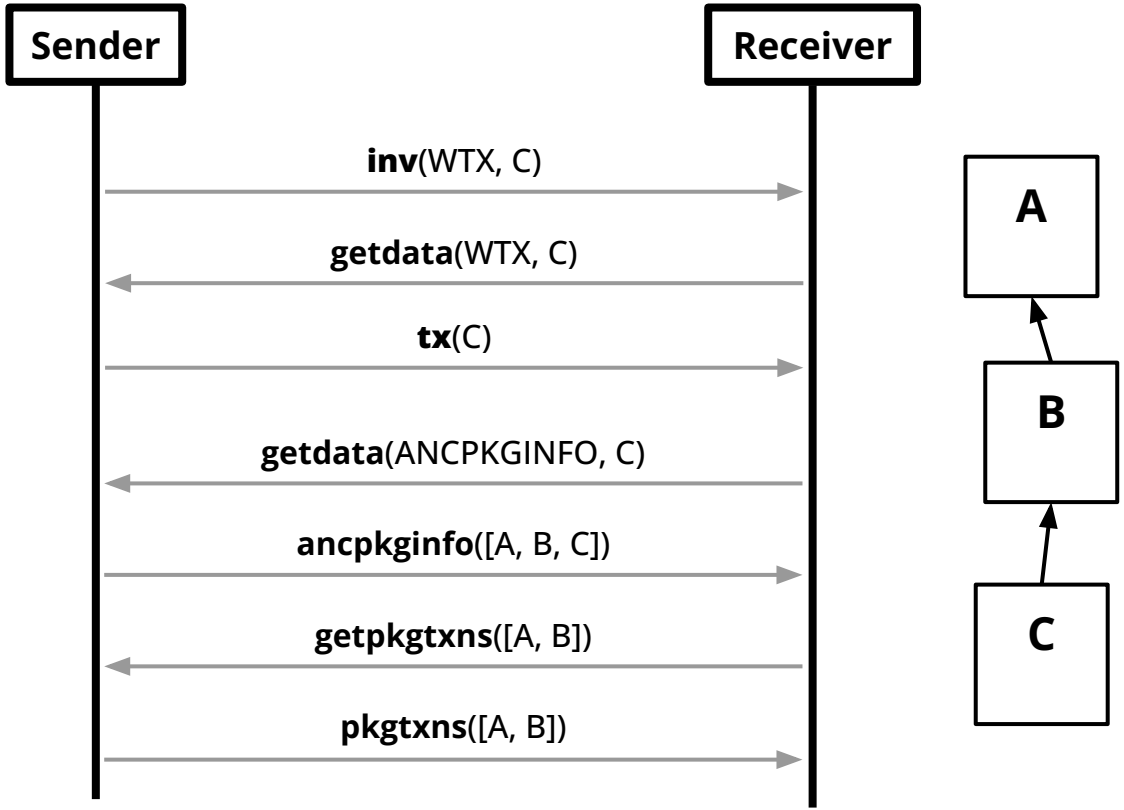
- On inv(tx): don't download tx
- On ancpkginfo: Reject immediately if package hash of all wtxids is in here

recent_rejects vs recent_rejects_reconsiderable

1. Orphan F. ancpkginfo {ABCDEF}
   a. submit, {AC}, B
   b. reject D, E, F
   c. recent_rejects: D
   d. reconsiderable: E, F, {ABCDEF}
2. inv Tx E: Don't download.
3. Orphan G. ancpkginfo {EG}.
   a. E was rejected but reconsiderable
   b. reject E, G
   c. reconsiderable: E, G, {EG}
4. inv Tx G: Don't download.
5. Orphan H. ancpkginfo {EGH}
   a. E, G, rejected but reconsiderable
   b. submit {EGH}
6. Orphan J. ancpkginfo {ABCDEFJ}.
   a. Reject immediately



5. Download and validate packages


Sender
Receiver
inv(WTX, C)
getdata(WTX, C)
tx(C)
getdata(ANCPKGINFO, C)
ancpkginfo([A, B, C])
getpkgtxns([A, B])
pkgtxns([A, B])
A
B
C