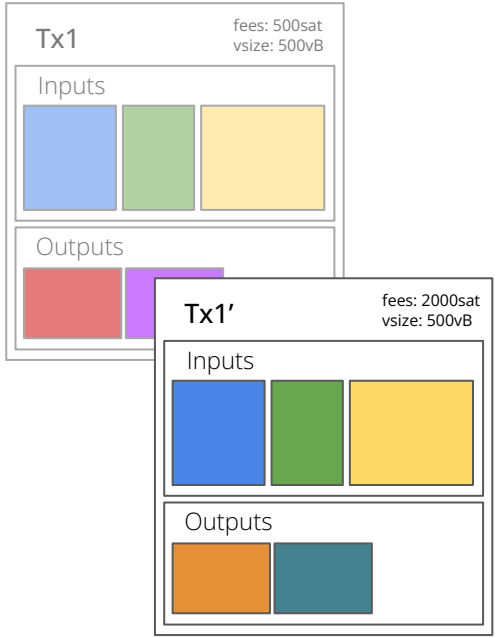
Package Mempool Accept
& Package Relay

@glozow

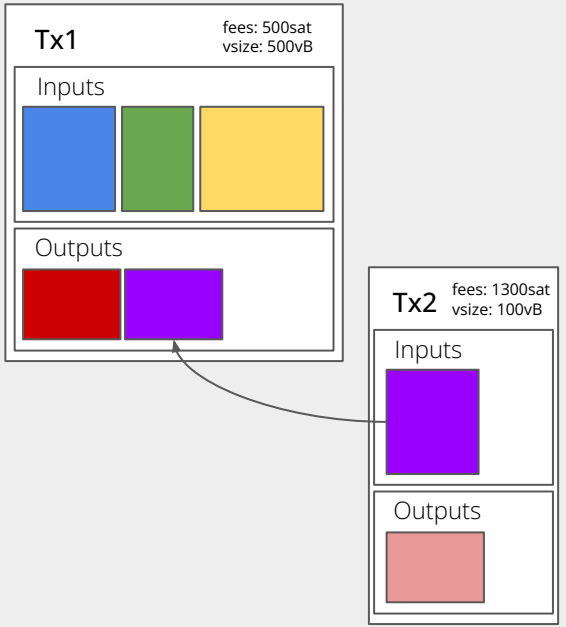
Replace By Fee (RBF)

Create a new tx with the same inputs



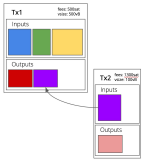
Child Pays for Parent (CPFP)

Create a high-fee child tx to pay for both



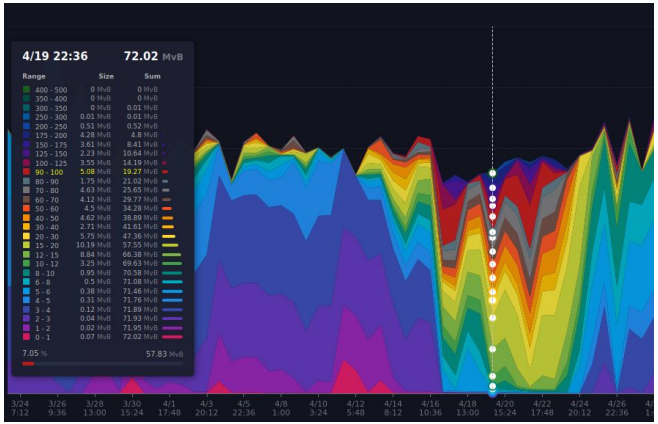
Replace By Fee Limitations

- Requires signaling
- Any additional inputs must be confirmed
- May be **expensive**, especially if attacker is trying to *pin* the transaction.
- **Requires new signatures** for the inputs, might not be possible
- **Only considers 1 replacement tx at a time**; descendant fees not counted



Child Pays for Parent Limitations

- Only works if both already in mempool
- **If parent's feerate is below mempool minimum, both rejected**

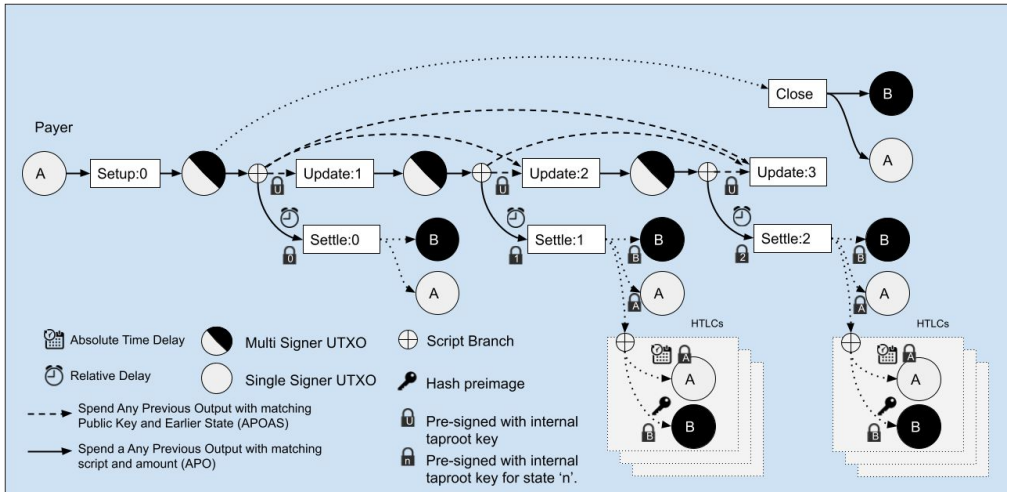


https://node210.bitcoin.wiz.biz/graphs#1y

1. **Feerate is negotiated ahead of time**

2. **RBF is not an option**
   “Hello cheating counterparty, can you help me sign a new tx?”

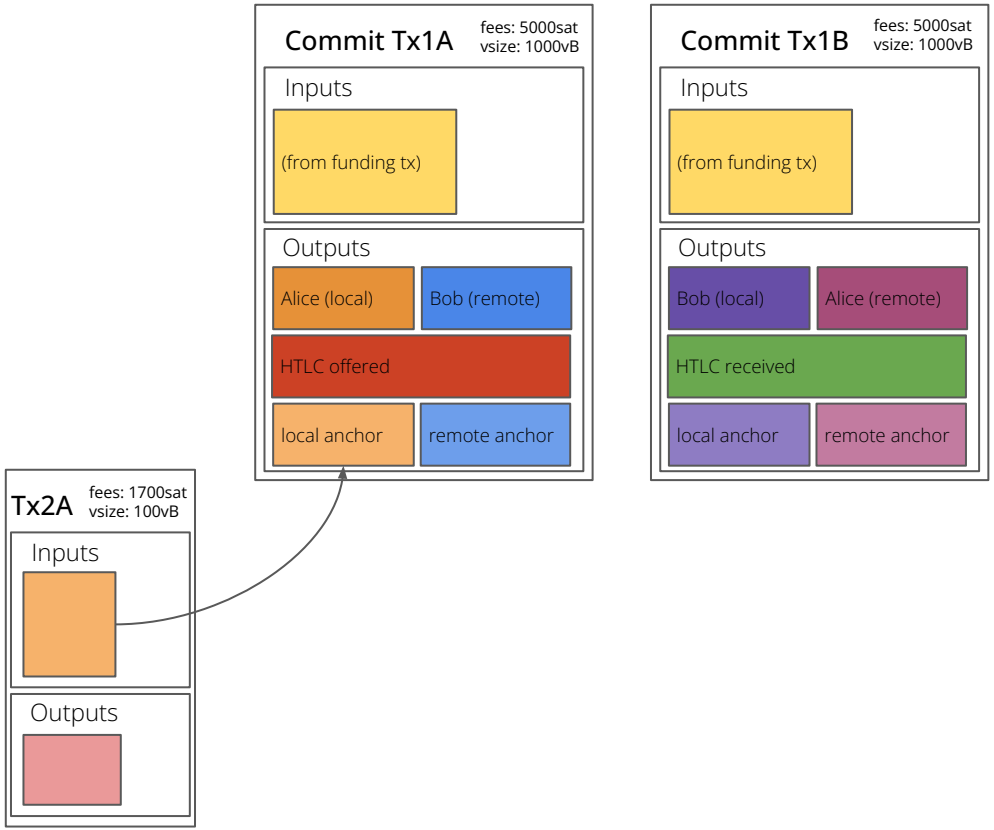
3. **Often constrained by a timelock**
   Usually need to confirm “justice transaction” before their relative timelock expires



https://yakshaver.org/2021/07/26/first.html

CPFP and RBF are mutually exclusive

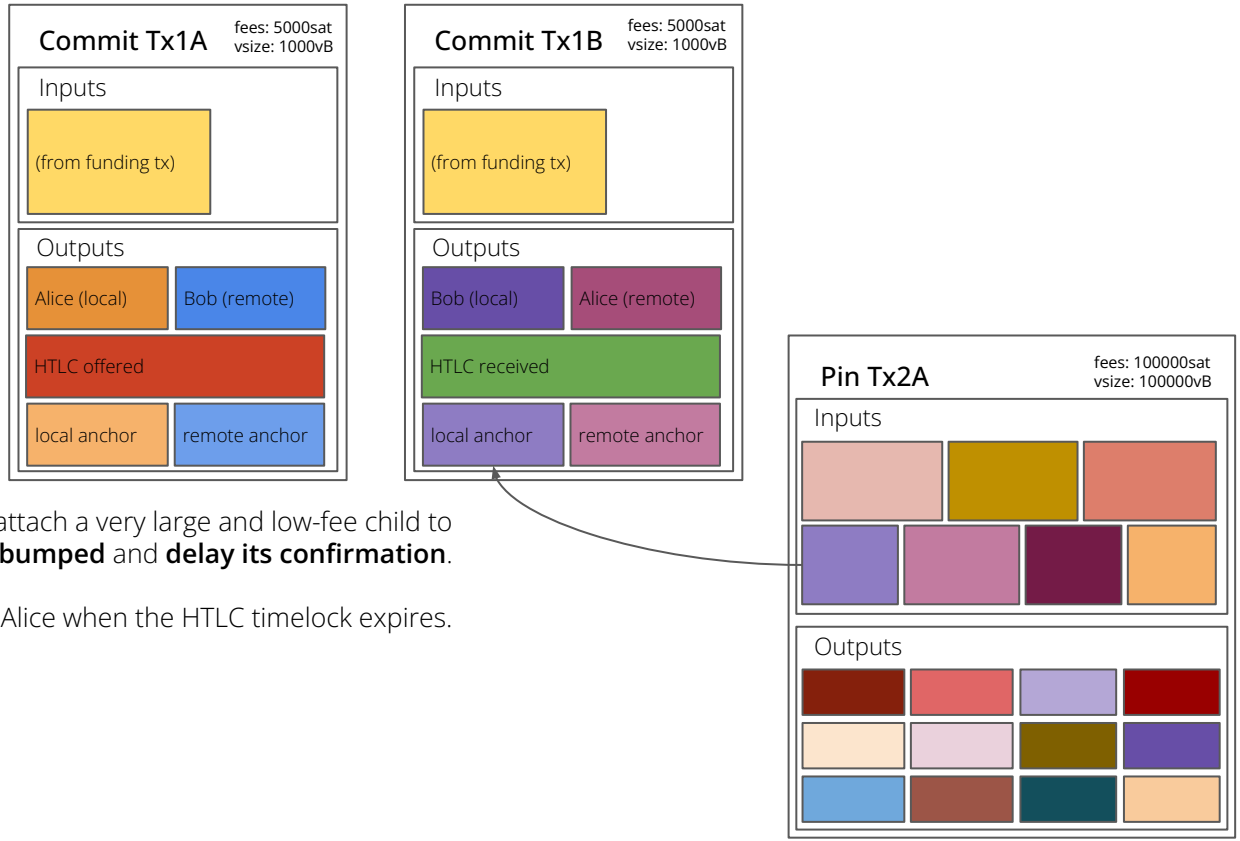
Commitment Transactions cannot replace one another



RBF only applies for a single replacement transaction.
Mempools accept the one they see first.

CPFP and RBF are mutually exclusive

Commitment Transactions cannot replace one another

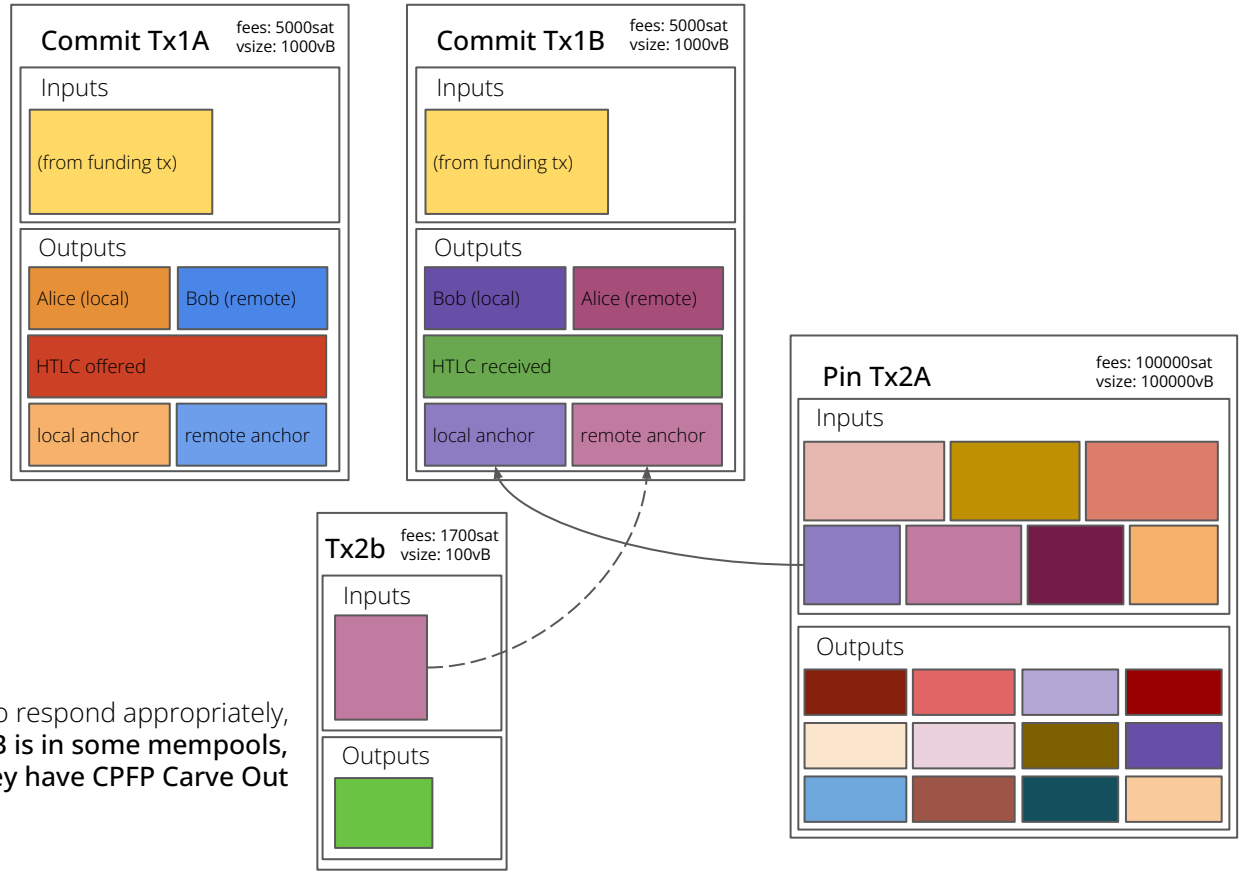


Bob can also attach a very large and low-fee child to **prevent it from being fee-bumped** and **delay its confirmation**.

Now, Bob has a head start on Alice when the HTLC timelock expires.

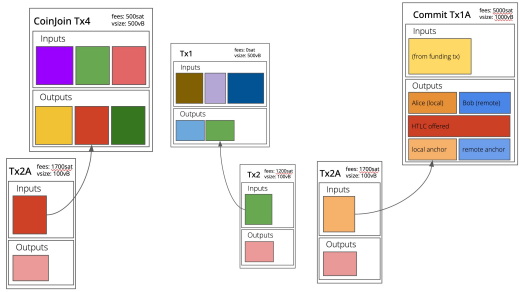
CPFP and RBF are mutually exclusive

Commitment Transactions cannot replace one another



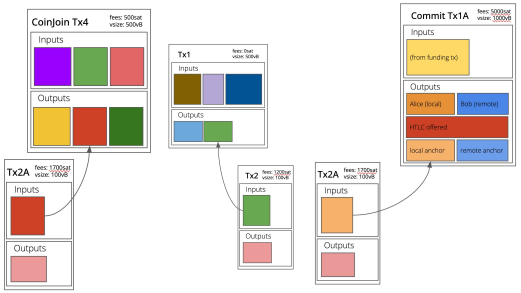
To respond appropriately,
Alice must be aware that Tx1B is in some mempools, and hope that they have CPFP Carve Out

Package Mempool Accept: validate and submit packages in mempool, policies to assess package feerates

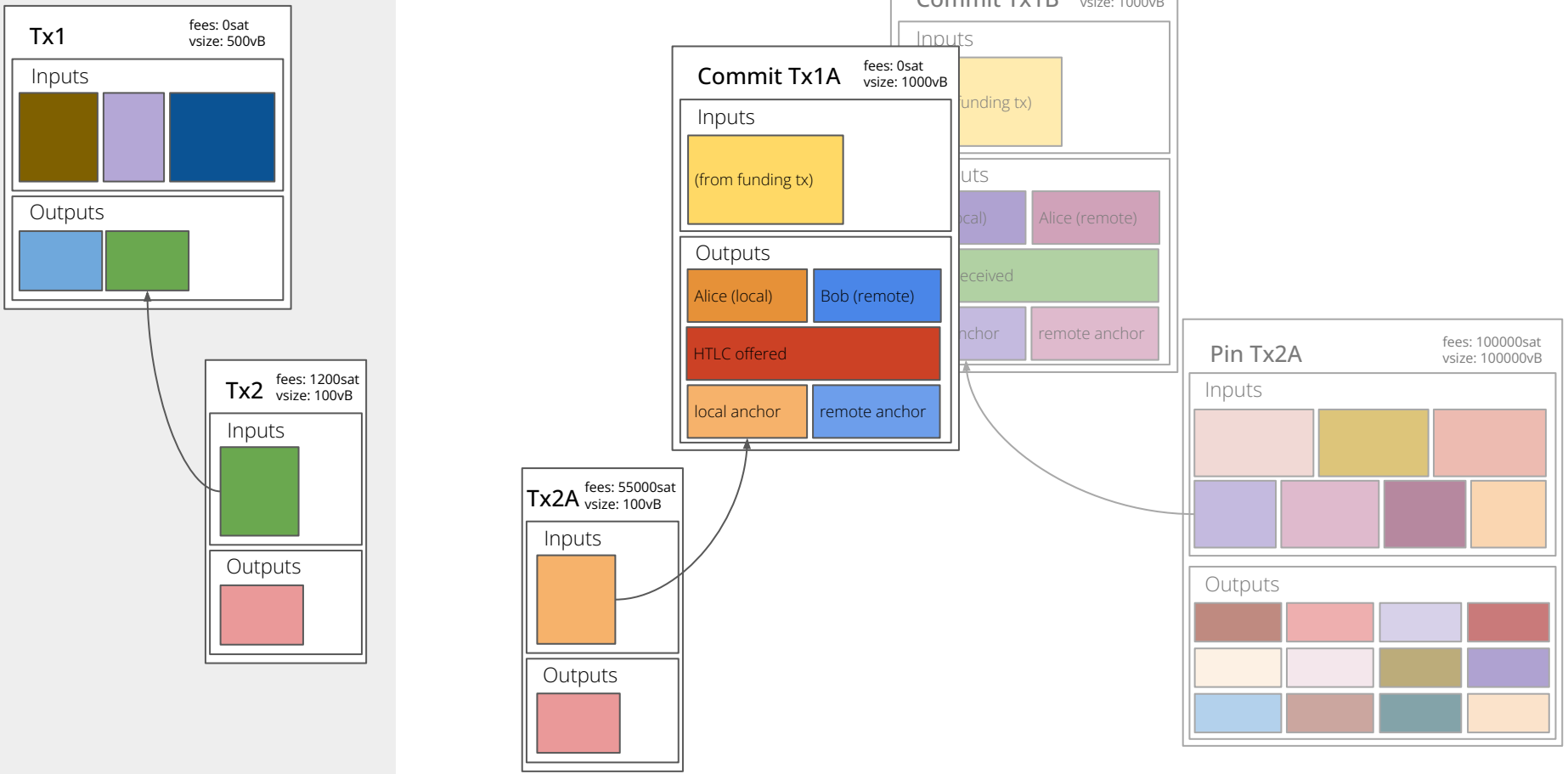


Package Mempool Accept: validate and submit packages in mempool, policies to assess package feerates

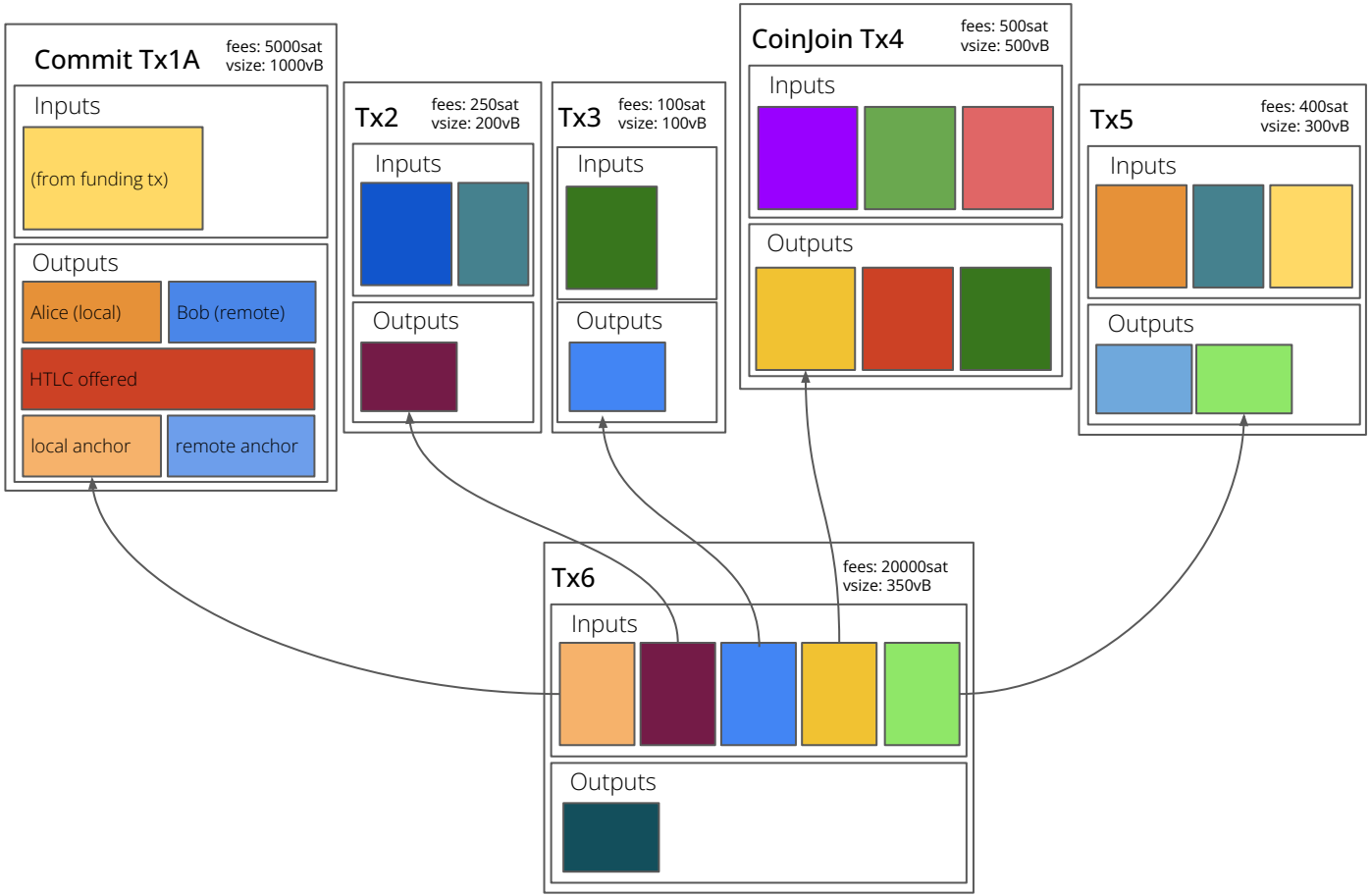
Package Relay: relay packages on the P2P network



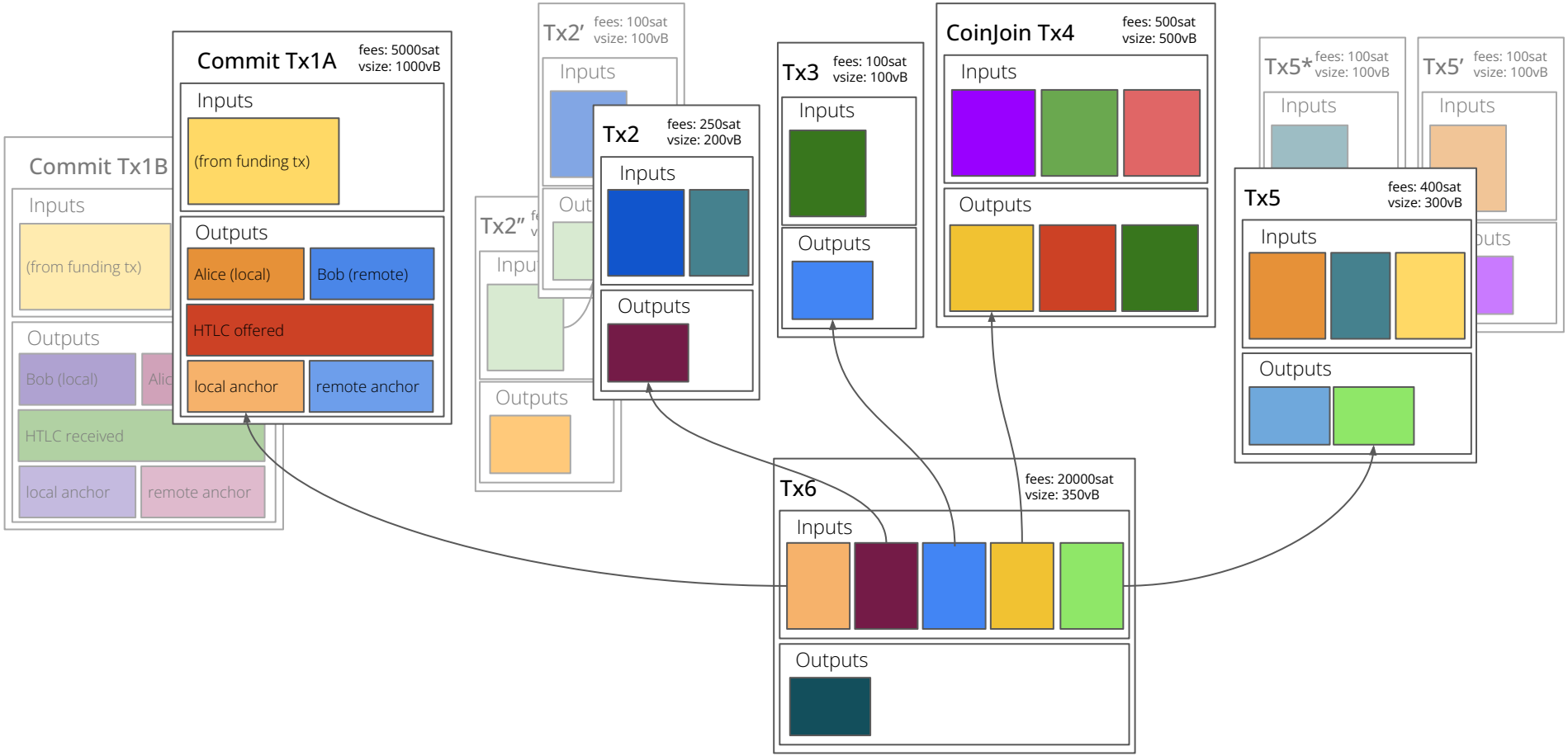
Package CPFP



Package *Batched* CPFP



Package Batched CPFP *with Replacement of Parents' Conflicts*



In the wild west of the Bitcoin P2P network,

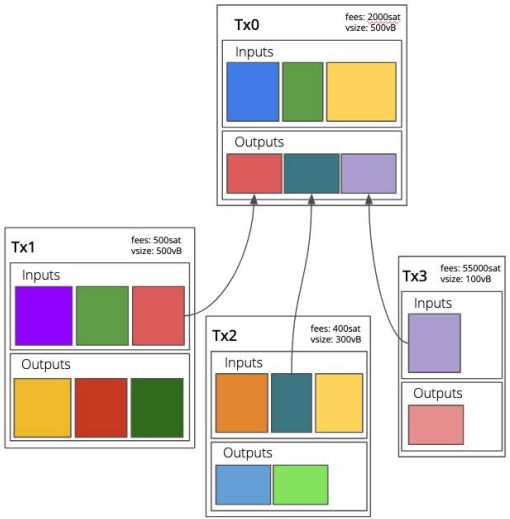
Who might be sending us a package?

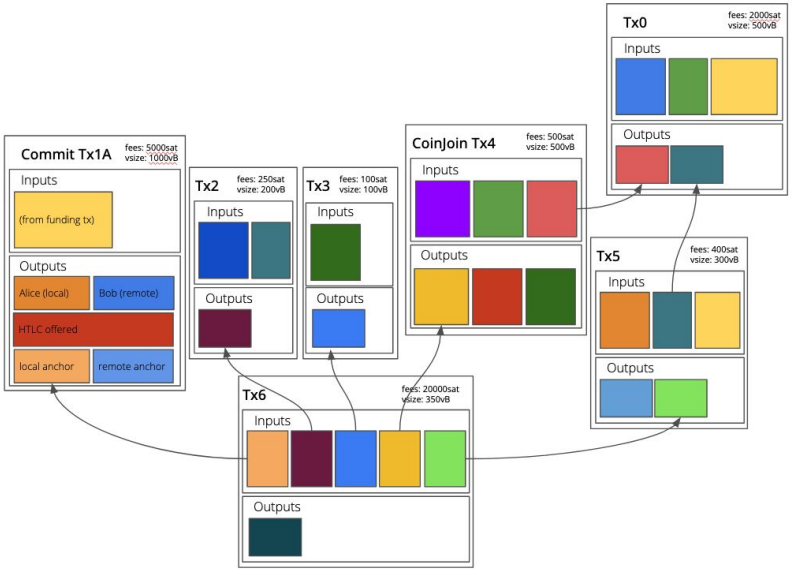
- Honest Users
- DoSer trying to exhaust our CPU
- DoSer trying to cause OOM
- Attacker trying to fill mempools with garbage
- DDoSer trying to stall the network for 0.5sec
- Attacker trying to cause network splits
- Lightning counterparty trying to pin
  or censor the honest user's package
- Spy node trying to deanonymize transactions
- Spy node trying to analyze network topology

How do we enforce mempool ancestor/descendant limits in packages?

Too intensive - we exhaust CPU calculating them. Too loose - we could accidentally create a new pinning vector.

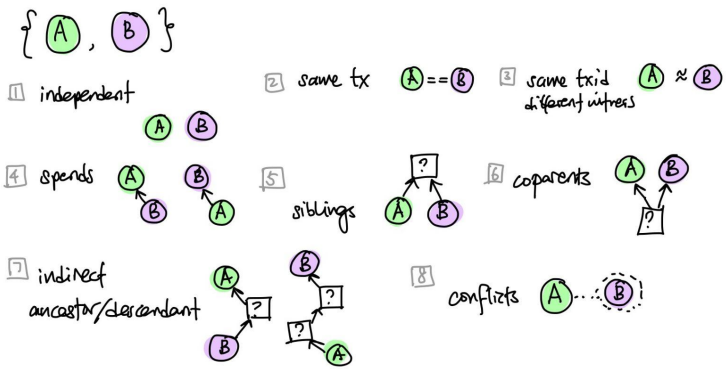
Descendants: all children, children's children, recursively (Tx0 has 3 descendants)





Ancestors: all parents, parent's parents, recursively (Tx6 has 6 ancestors)

Consider all the possibilities between 2 transactions:



Consider all the possibilities between 2 transactions: {A, B}

1 independent

2 same tx A == B

3 same txid different witness A ≈ B

4 spends

5 siblings

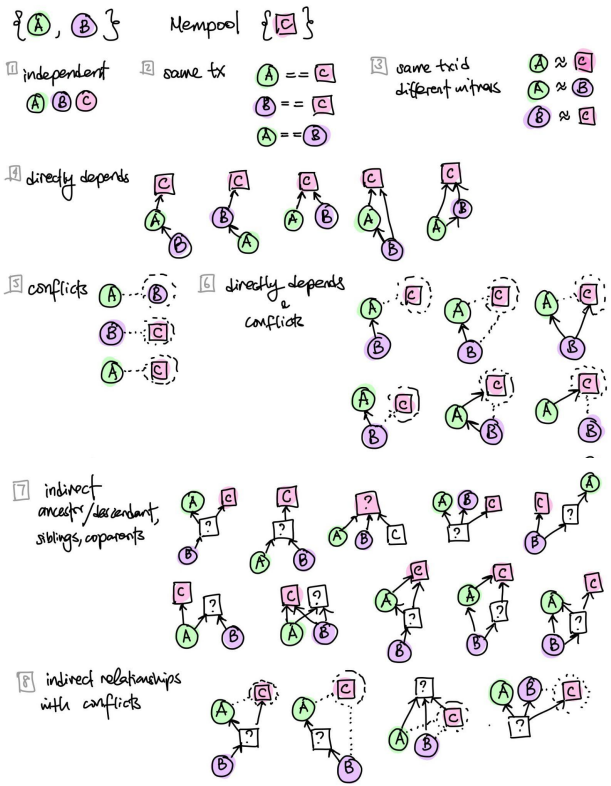
6 coparents

7 indirect ancestor/descendant

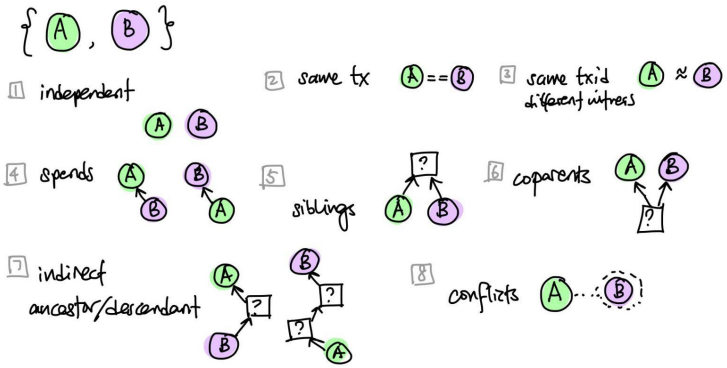
8 conflicts

Add 1 mempool transaction, and the number of possibilities increases:

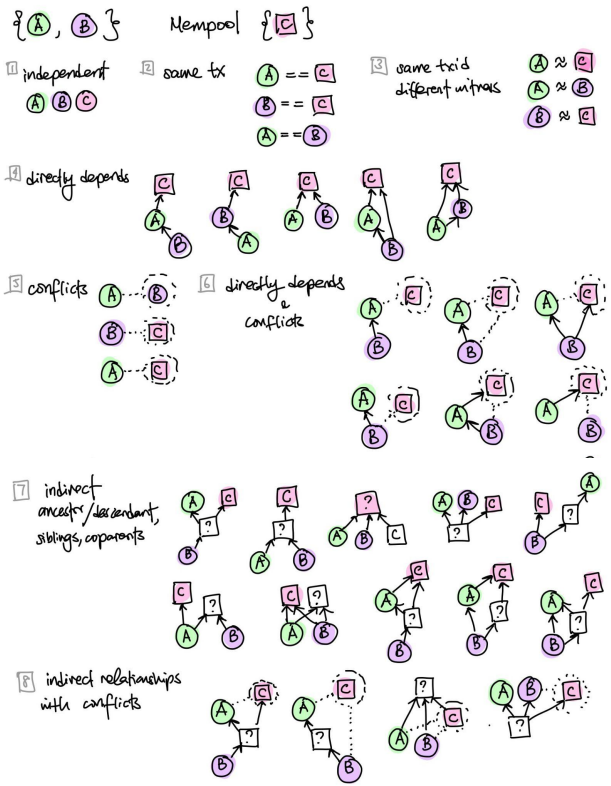
{A, B} Mempool {C}



Consider all the possibilities between 2 transactions:



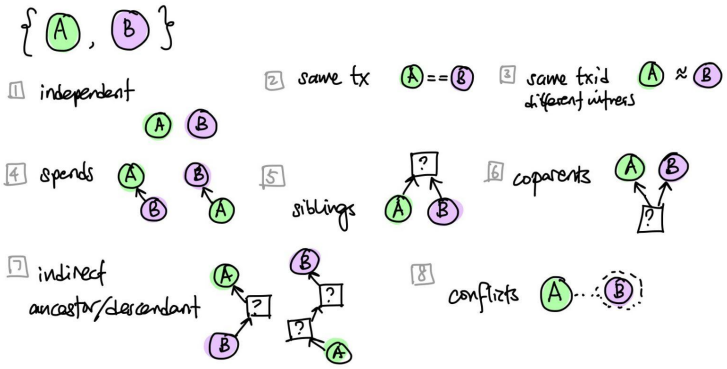
Add 1 mempool transaction, and the number of possibilities increases:



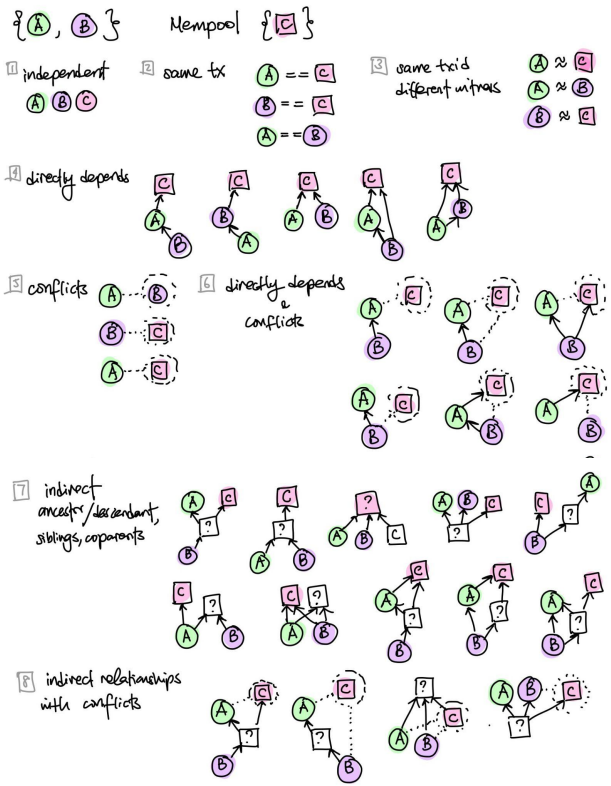
Add a whole mempool...
And RBF...
And batching...

And people start talking about multiple descendants paying for ancestors...

Consider all the possibilities between 2 transactions:



Add 1 mempool transaction, and the number of possibilities increases:



Add a whole mempool...
And RBF...
And batching....

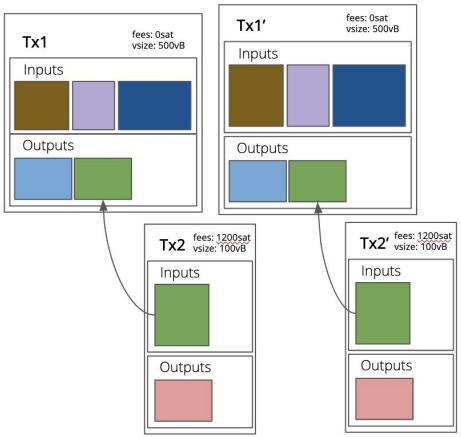
And people start talking about multiple descendants paying for ancestors...

Pretty soon, your office looks like this:

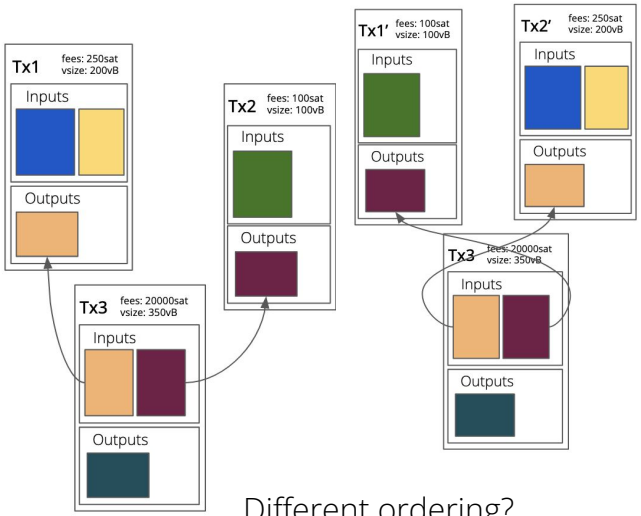
(and coworkers say you look like this)

How do we cache package failures without creating censorship vectors?

Too optimistic - can we repeatedly validate the same invalid package over and over again.
Too pessimistic - we could accidentally allow attackers to censor an honest package by sending an invalid variation of it.



Same txid, different witness?



Different ordering?