Graphene: A New Protocol for Block Propagation Using Set Reconciliation

A. Pinar Ozisik

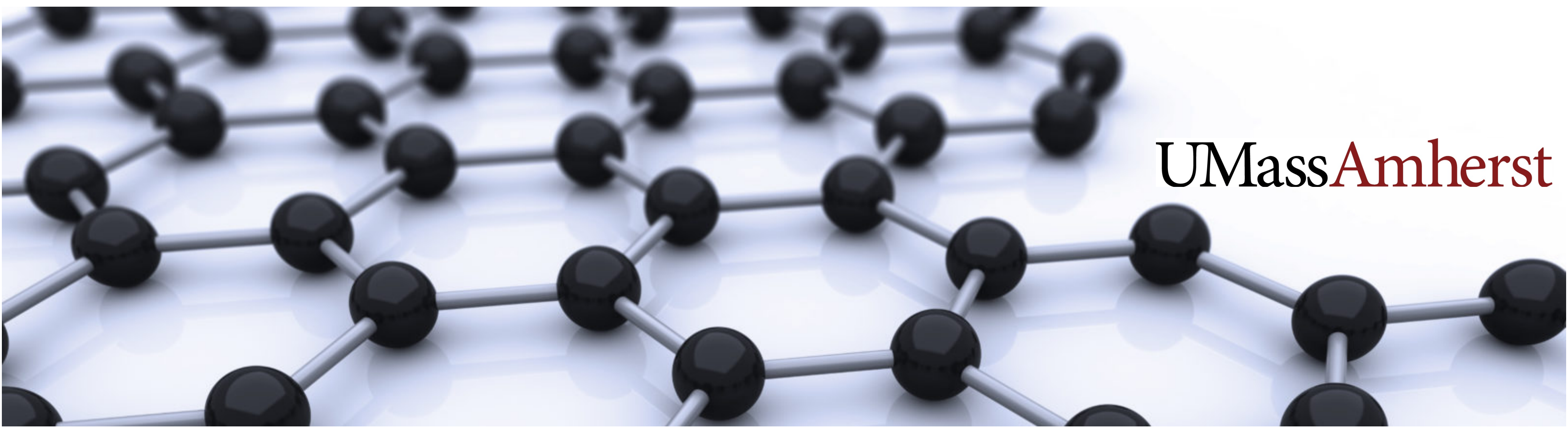
George Bissias

Gavin Andresen

Amir Houmansadr

Brian Neil Levine

UMass Amherst



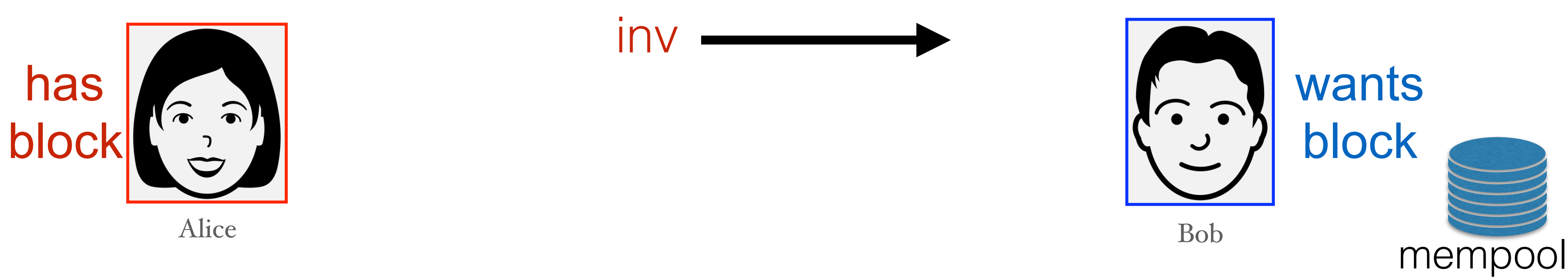
Problem Definition

- This presentation is focused on relaying information quickly to a neighbor.
  - on the fast Relay Network or the p2p network.

- It's about avoiding sending a lot of data between peers, like so:

Problem Definition

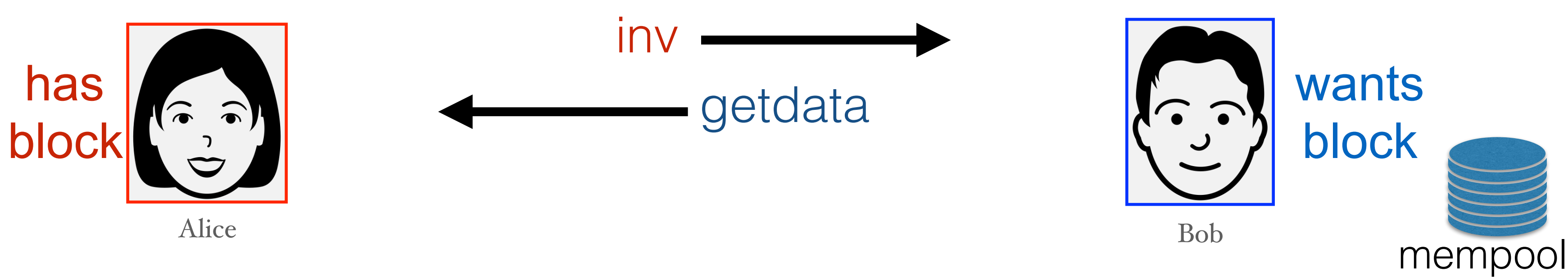
- This presentation is focused on relaying information quickly to a neighbor.
  - on the fast Relay Network or the p2p network.
- It's about avoiding sending a lot of data between peers, like so:



Problem Definition

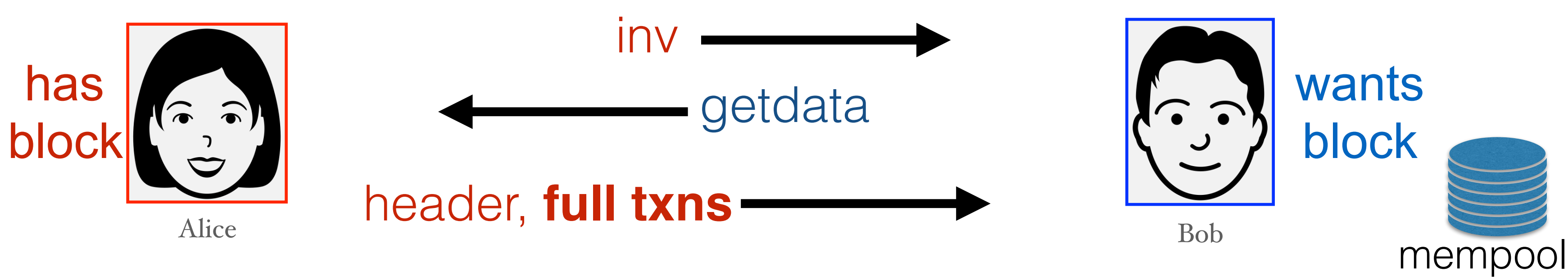
- This presentation is focused on relaying information quickly to a neighbor.
  - on the fast Relay Network or the p2p network.

- It's about avoiding sending a lot of data between peers, like so:



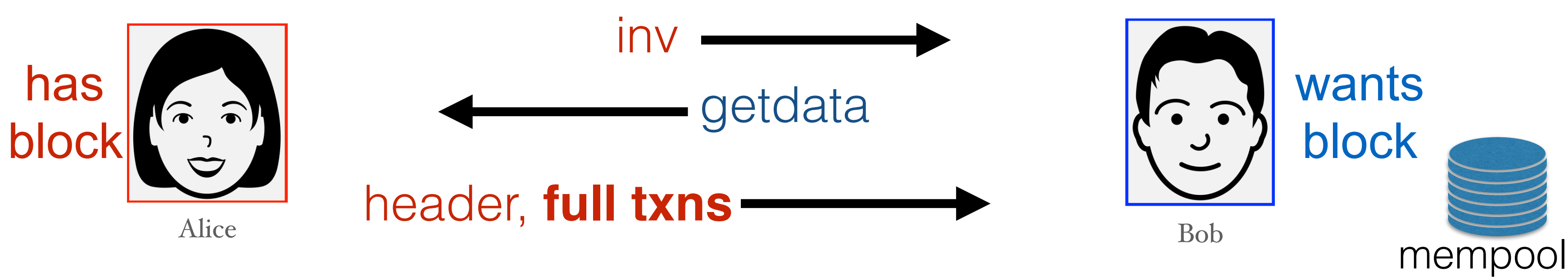
Problem Definition

- This presentation is focused on relaying information quickly to a neighbor.
  - on the fast Relay Network or the p2p network.
- It's about avoiding sending a lot of data between peers, like so:

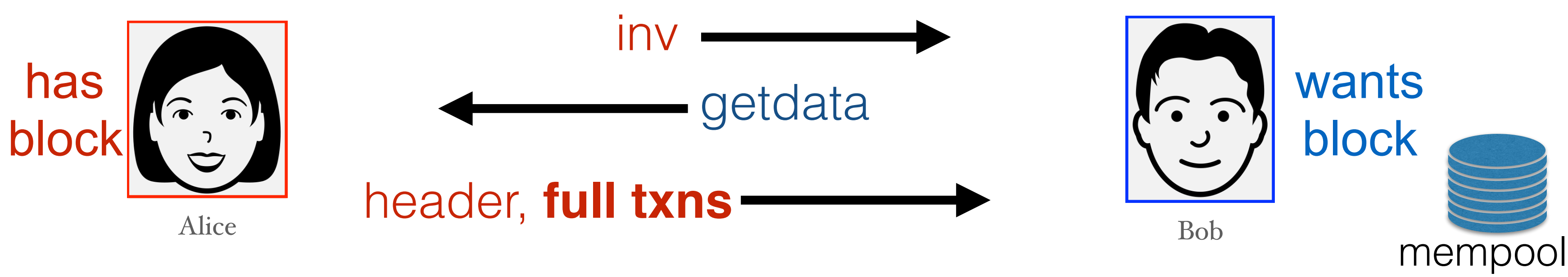


Problem Definition

- This presentation is focused on relaying information quickly to a neighbor.
  - on the fast Relay Network or the p2p network.
- It’s about avoiding sending a lot of data between peers, like so:



Problem Definition



- Block announcements propagate faster when they are smaller.
- Faster propagation means less orphaning, which means mining is efficient.
- This isn't a presentation about reducing the size of the stored blockchain.

Results

- **Graphene's block announcements are $\frac{1}{10}$ the size of current methods.**
  - No increase in roundtrip time.
  - Not a significant use of storage or CPU.
- Combines two known tools from set reconciliation literature in a nifty way.
  - Bloom Filters and IBLTs
- Why does it work? We are optimizing Bitcoin's special case:
  - Everyone needs to know everything.
  - Blocks are comprised of transactions that everyone should have heard already.

Overview

- A series of protocols:
  - Compact Blocks
  - Xtreme Thin Blocks
  - Soot [fake]
  - IBLTs
  - Graphene

Protocol 1: Compact Blocks

BIP 152
Matt Corallo

- We don't need to send the full transactions.
- We can send just the 2xSHA256 (32-byte) transaction IDs.
- And we only need the first 5 or 6 bytes. Odds of mistake are 1 in a trillion.

Protocol 1: Compact Blocks

BIP 152
Matt Corallo

- We don't need to send the full transactions.
- We can send just the 2xSHA256 (32-byte) transaction IDs.
- And we only need the first 5 or 6 bytes. Odds of mistake are 1 in a trillion.

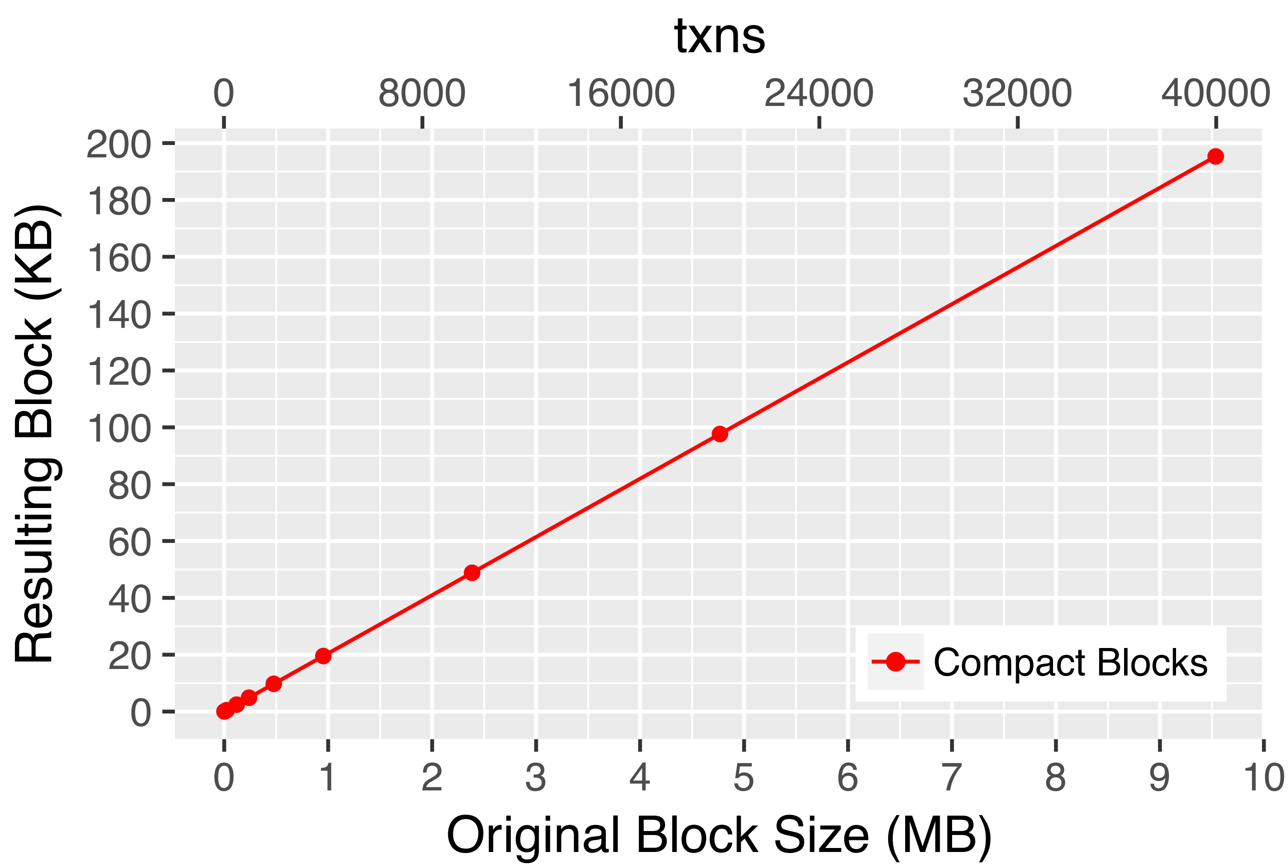
Protocol 1: Compact Blocks

BIP 152
Matt Corallo

- We don't need to send the full transactions.
- We can send just the 2xSHA256 (32-byte) transaction IDs.
- And we only need the first 5 or 6 bytes. Odds of mistake are 1 in a trillion
- Now a 1MB block with can be expressed in 80+4200*5 = 21KB
  - An 8MB block reduces to 80+4200*8*5 = 164KB

Evaluation

- Linear growth with the number of transactions included in the block.
- Size is independent of mempool.



https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/

Protocol 2: Bloom Filters

- Can we do better? Yes!
- Our neighbors already have these transactions IDs.
- They are likely only missing a few.
- Alice can each express the set of transactions in the block or her mempool as a **Bloom Filter.**
  - Bob could do the same thing!
  - Bloom filters allow us to quickly check if an item is a member of a set.

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	0	0	0	0	0	0

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors.
Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	0	0	0	0	0	0

insert: txn_1

$H_1(txn_1) = 1$

$H_2(txn_1) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	1	0	0	0	0	0

insert: txn_1
$H_1(txn_1) = 1$
$H_2(txn_1) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	1	0	0	1	0	0

insert: txn_1
$H_1(txn_1) = 1$
$H_2(txn_1) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
0	1	0	0	1	0	0

insert: txn_1

$H_1(txn_1) = 1$

$H_2(txn_1) = 4$

insert: txn_2

$H_1(txn_2) = 0$

$H_2(txn_2) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors.
Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

insert: txn_1
$H_1(txn_1) = 1$
$H_2(txn_1) = 4$

insert: txn_2
$H_1(txn_2) = 0$
$H_2(txn_2) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors.
Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filter: Insertion

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

insert: txn_1
$H_1(txn_1) = 1$
$H_2(txn_1) = 4$

insert: txn_2
$H_1(txn_2) = 0$
$H_2(txn_2) = 4$

B. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors.
Communications of the ACM 13(7), 422–426 (Jul 1970)

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

Is txn1 in the set?

$H_1(txn_1) = 1$, $H_2(txn_1) = 4$

cell 1 = 1

cell 4 = 1

Yes!

True Positive

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

Is txn1 in the set?
$H_1(txn_1) = 1$, $H_2(txn_1) = 4$
cell 1 = 1
cell 4 = 1
Yes!

True Positive

Is txn3 in the set?
$H_1(txn_3) = 1$, $H_2(txn_3) = 5$
cell 1 = 1
cell 5 = 0
No!

True Negative

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

Is txn1 in the set?
$H_1(txn_1) = 1$, $H_2(txn_1) = 4$
cell 1 = 1
cell 4 = 1
Yes!

True Positive

Is txn3 in the set?
$H_1(txn_3) = 1$, $H_2(txn_3) = 5$
cell 1 = 1
cell 5 = 0
No!

True Negative

Is txn4 in the set?
$H_1(txn_4) = 0$, $H_2(txn_4) = 1$
cell 0 = 1
cell 1 = 1
Yes!

False Positive

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

False Negatives are not possible.

Is txn1 in the set?
$H_1(txn_1) = 1$, $H_2(txn_1) = 4$
cell 1 = 1
cell 4 = 1
Yes!

True Positive

Is txn3 in the set?
$H_1(txn_3) = 1$, $H_2(txn_3) = 5$
cell 1 = 1
cell 5 = 0
No!

True Negative

Is txn4 in the set?
$H_1(txn_4) = 0$, $H_2(txn_4) = 1$
cell 0 = 1
cell 1 = 1
Yes!

False Positive

Bloom Filters: Check

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	1	0	0	1	0	0

False Negatives are not possible.

Is txn1 in the set?
$H_1(txn_1) = 1$, $H_2(txn_1) = 4$
cell 1 = 1
cell 4 = 1
Yes!

True Positive

Is txn3 in the set?
$H_1(txn_3) = 1$, $H_2(txn_3) = 5$
cell 1 = 1
cell 5 = 0
No!

True Negative

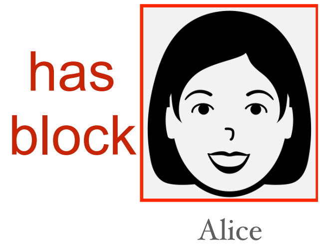
Is txn4 in the set?
$H_1(txn_4) = 0$, $H_2(txn_4) = 1$
cell 0 = 1
cell 1 = 1
Yes!

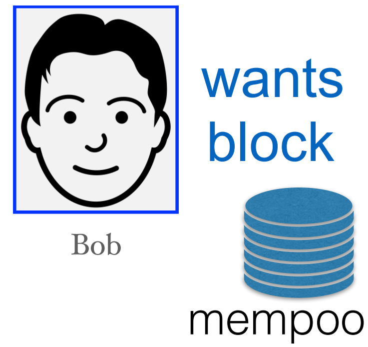
False Positive

The False Positive Rate is tunable: More bits will lower the FPR.

Protocol 2: Xtreme Thinblocks

Peter Tschipper

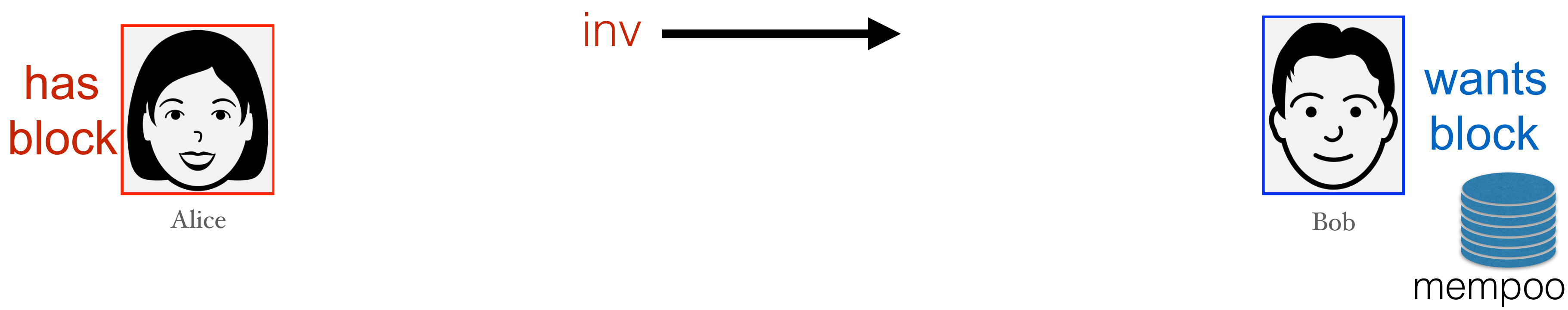




- We are sending all txnIDs **and** we are sending a Bloom Filter.
- This is more data across the network than Compact Blocks.

Protocol 2: Xtreme Thinblocks

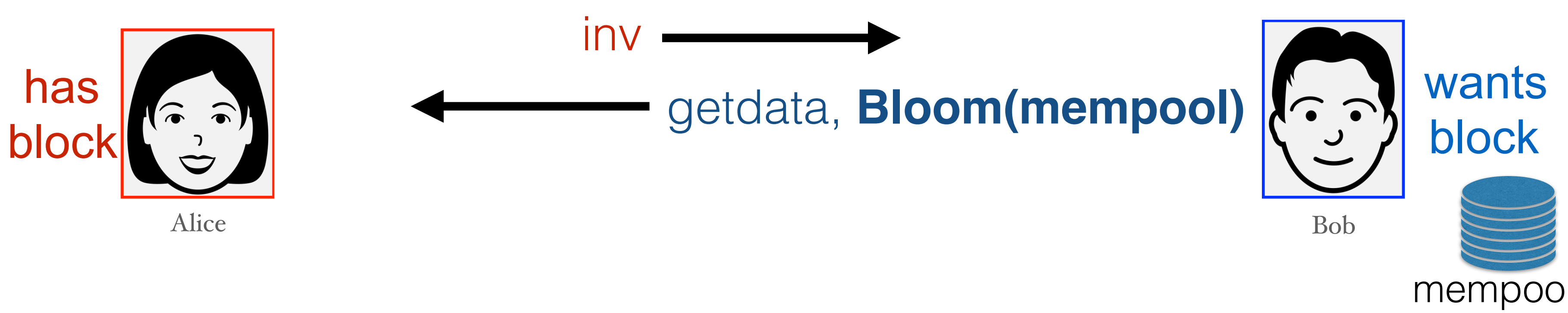
Peter Tschipper



- We are sending all txnIDs **and** we are sending a Bloom Filter.
- This is more data across the network than Compact Blocks.

Protocol 2: Xtreme Thinblocks

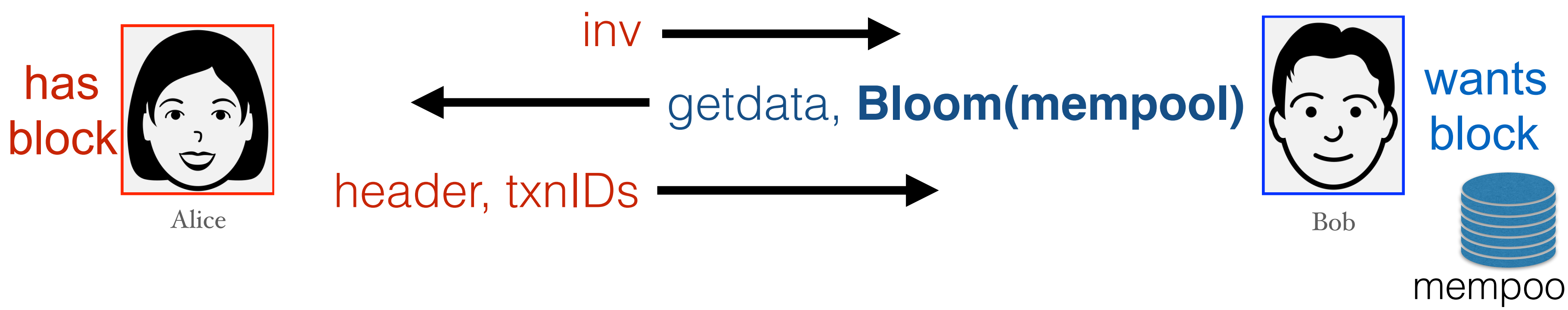
Peter Tschipper



- We are sending all txnIDs **and** we are sending a Bloom Filter.
- This is more data across the network than Compact Blocks.

Protocol 2: Xtreme Thinblocks

Peter Tschipper



- We are sending all txnIDs **and** we are sending a Bloom Filter.
- This is more data across the network than Compact Blocks.

Protocol 3: Soot

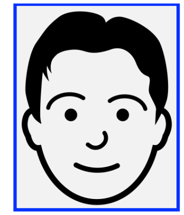
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
  - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot

(prioritize TXN inv's)

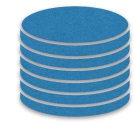
has block

Alice



Bob

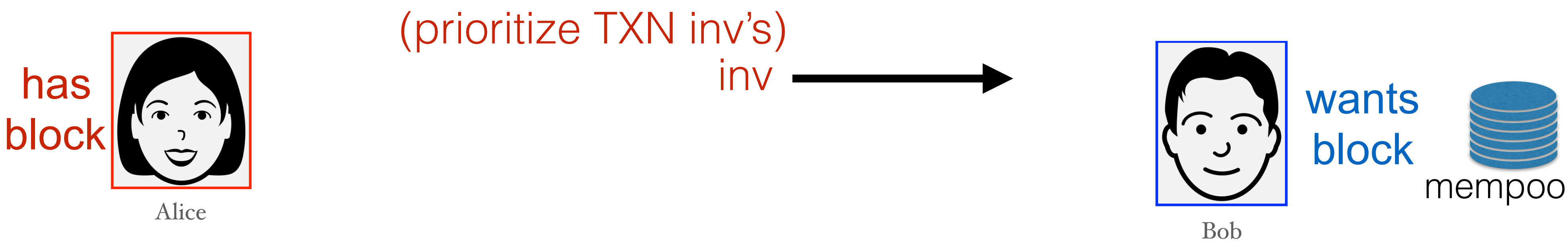
wants block



mempool

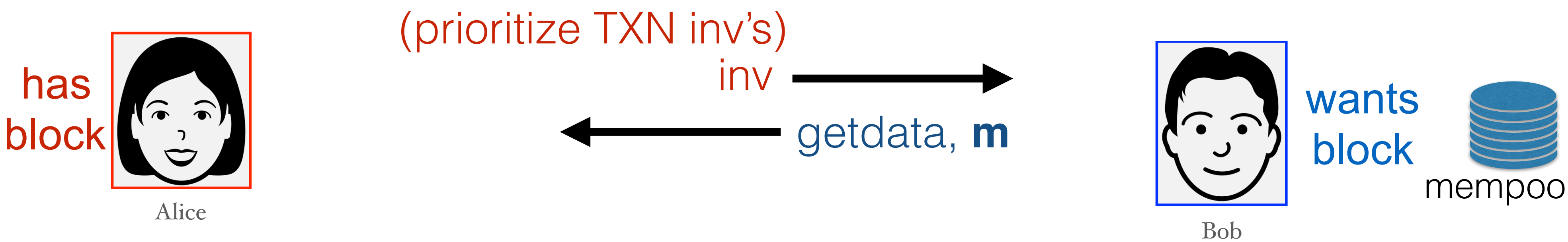
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
  - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot



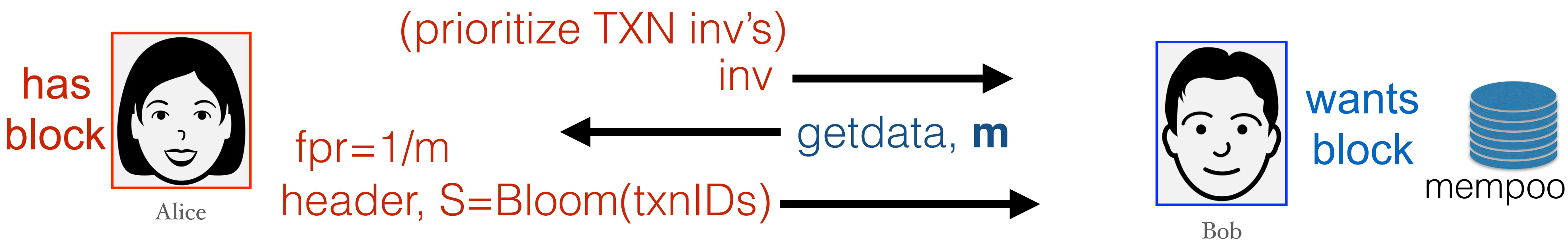
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
    - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot



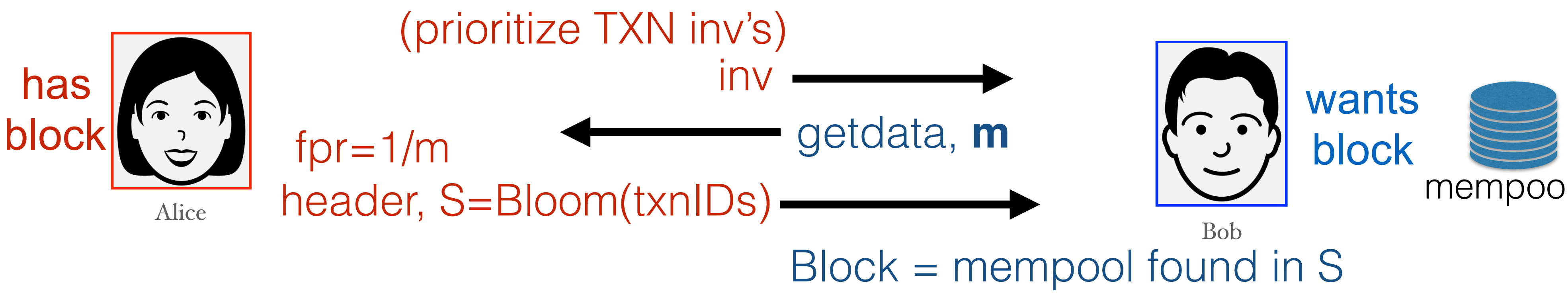
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
  - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot



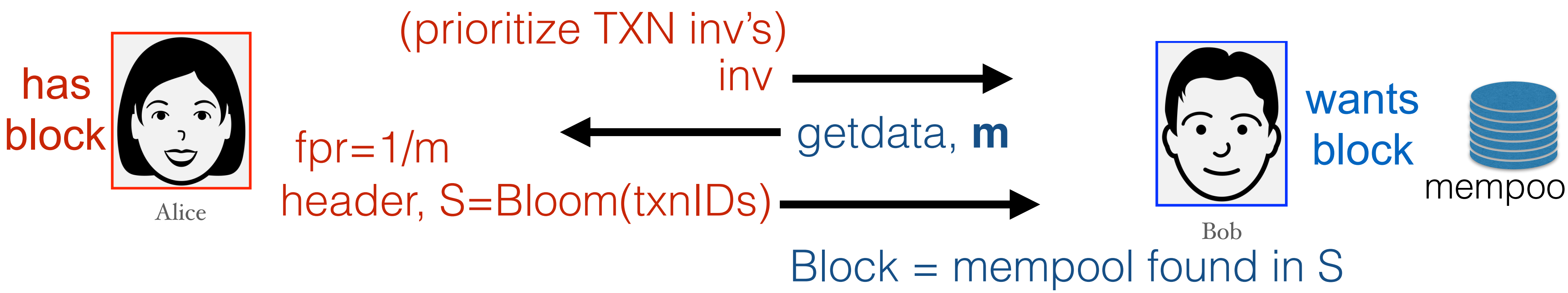
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
  - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot



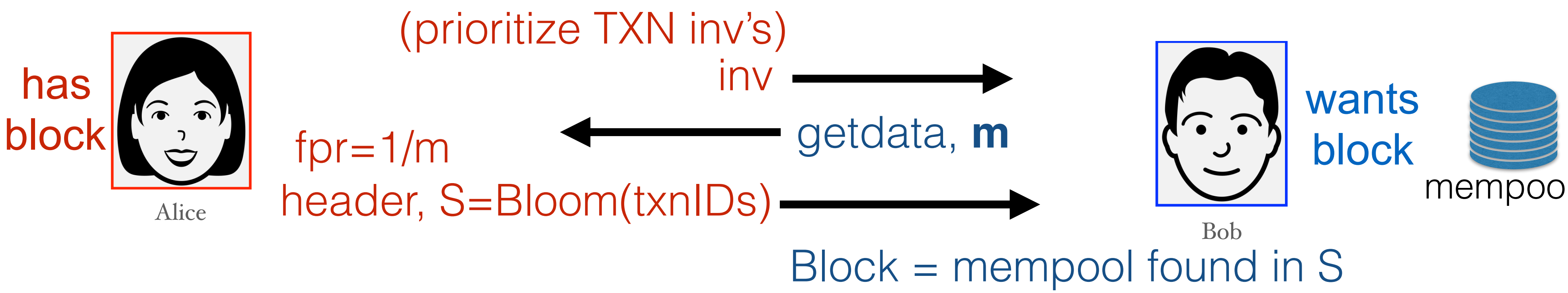
- Soot is not a real protocol…
- Send INV for each TXNs in the block ahead of the block INV.
  - if they haven't already been sent or received.
- We need a low FPR for the Sender's Bloom filter.
- Can't base it on size of the block!
- Let **m** be the number of transactions in the mempool.

Protocol 3: Soot



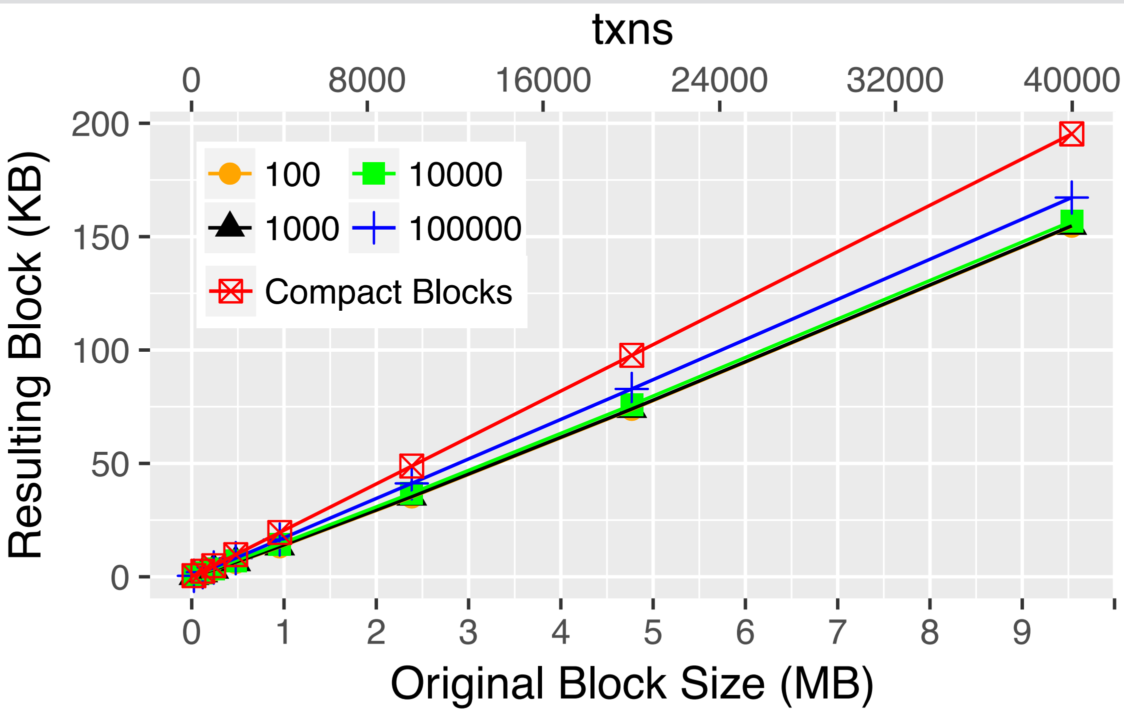
- If **FPR=1/m**, then we expect 1 transaction from mempool to falsely appear to be in the block.
  - Block reconstruction will fail every block!

Protocol 3: Soot



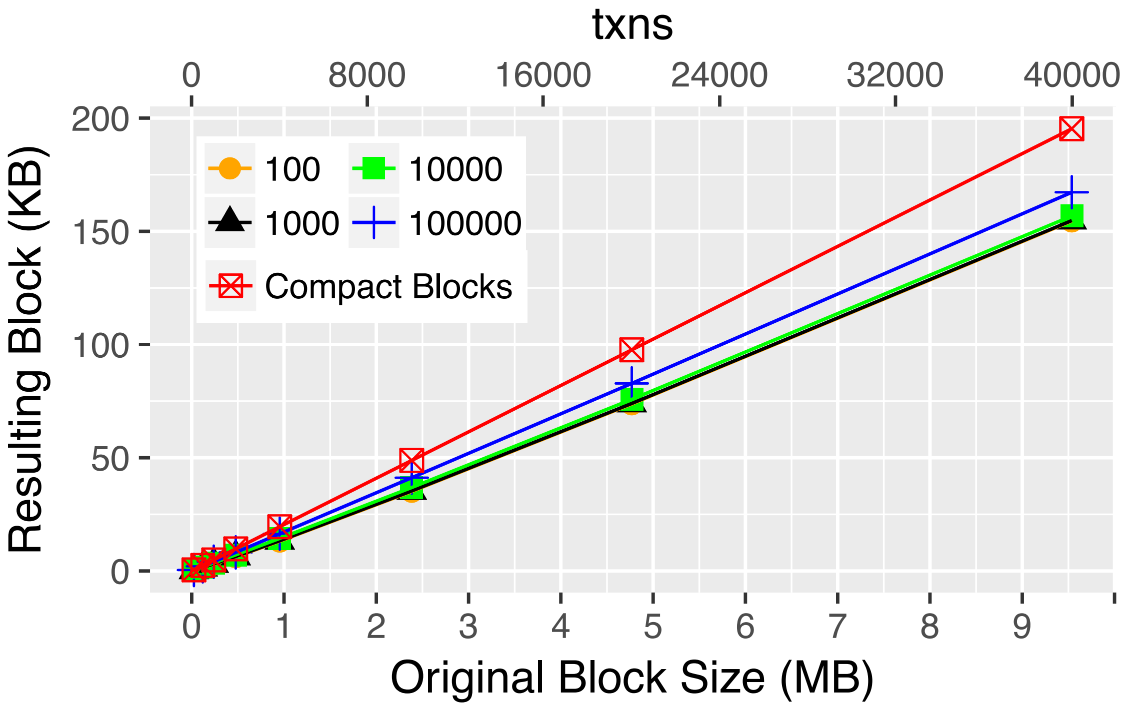
- If **FPR=1/m**, then we expect 1 transaction from mempool to falsely appear to be in the block.
  - Block reconstruction will fail every block!
- If **FPR=1/(100m)**, once every 100 blocks, the receiver will fail to reconstruct the block.
  - In that case, fall back to Compact Blocks.

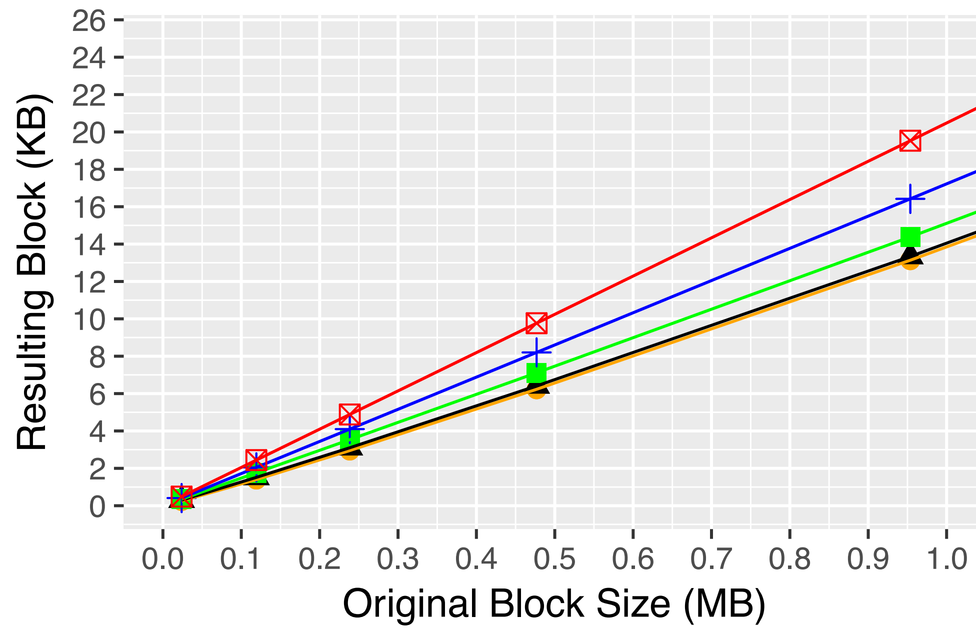
Performance of 1/(100m) Soot



Performance now depends on size of the mempool.

Performance of 1/(100m) Soot





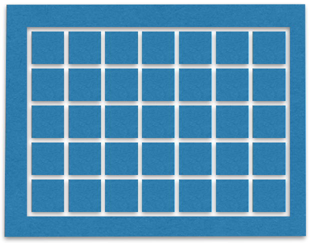
Performance now depends on size of the mempool.

Invertible Bloom Lookup Tables (IBLTs)

- Can we do better? Yes!
- M. Goodrich and M. Mitzenmacher
  "Invertible Bloom Lookup Tables"
  Proc. Conf. on Comm., Control, and Computing. pp. 792–799, Sept 2011
- D. Eppstein, M. Goodrich, F. Uyeda, G. Varghese
  "What's the difference?: efficient set reconciliation without prior context."
  Prof. ACM SIGCOMM 2011

Invertible Bloom Lookup Tables (IBLTs)

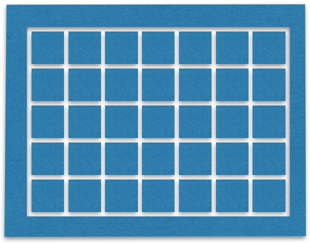
- Invertible Bloom Lookup Tables are a generalization of Bloom Filters.
  - Instead of a bit, cells include a count and actual content.



A,B, C, **D**,
E, F, G

Invertible Bloom Lookup Tables (IBLTs)

- Invertible Bloom Lookup Tables are a generalization of Bloom Filters.
  - Instead of a bit, cells include a count and actual content.

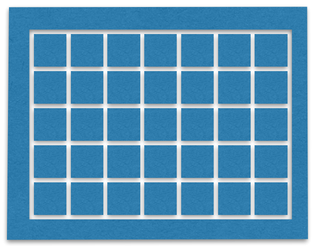


A,B, C, **D**,
E, F, G

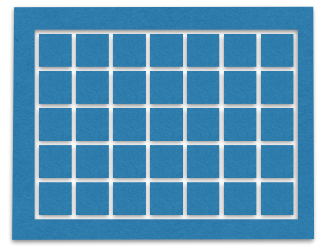
- Special IBLT feature:
  - If you have two lists **that differ by no more than ~15%**, you can compare an IBLT of each list and recover the items that are different.

Invertible Bloom Lookup Tables (IBLTs)

- Invertible Bloom Lookup Tables are a generalization of Bloom Filters.
  - Instead of a bit, cells include a count and actual content.



A,B, C, **D**,
E, F, G

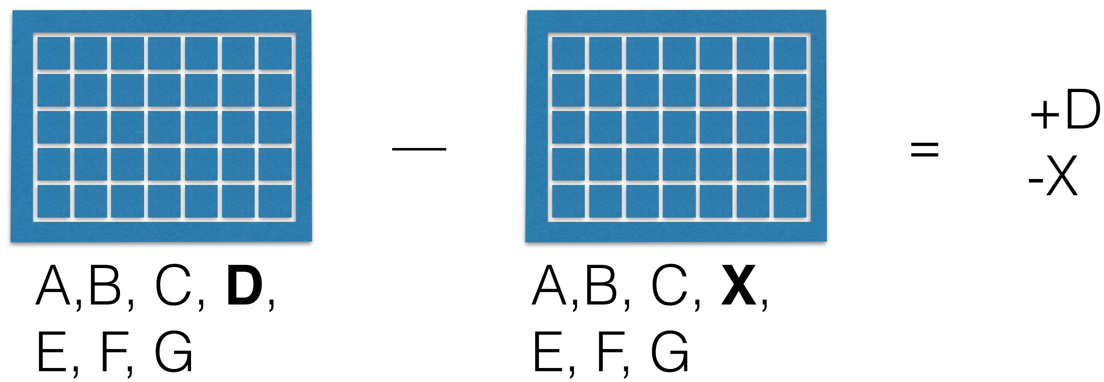


A,B, C, **X**,
E, F, G

- Special IBLT feature:
  - If you have two lists **that differ by no more than ~15%**, you can compare an IBLT of each list and recover the items that are different.

Invertible Bloom Lookup Tables (IBLTs)

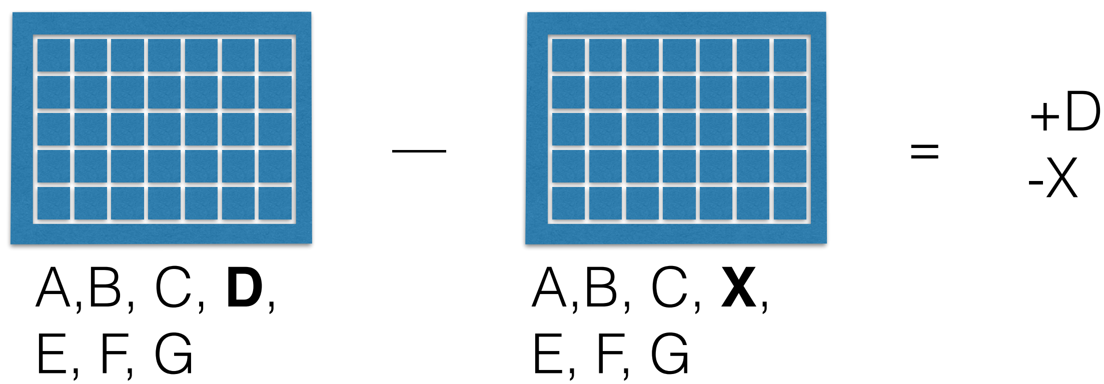
- Invertible Bloom Lookup Tables are a generalization of Bloom Filters.
  - Instead of a bit, cells include a count and actual content.



- Special IBLT feature:
  - If you have two lists **that differ by no more than ~15%**, you can compare an IBLT of each list and recover the items that are different.

Invertible Bloom Lookup Tables (IBLTs)

- Invertible Bloom Lookup Tables are a generalization of Bloom Filters.
  - Instead of a bit, cells include a count and actual content.

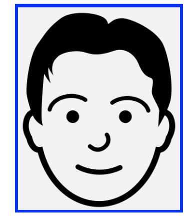


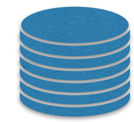
- Special IBLT feature:
  - If you have two lists **that differ by no more than ~15%**, you can compare an IBLT of each list and recover the items that are different.
- The size of IBLTs does not depend on the original list.
- The size depends on only the expected difference between the two lists.

Protocol 4: IBLTs

Gavin Andresen;
Rosenbaum and Russell

has block 
Alice

 wants block
Bob


mempool

- Works very well until the receiver's mempool size is much larger than the block.

- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
    - But we don't know this value and don't want to waste roundtrip times failing.

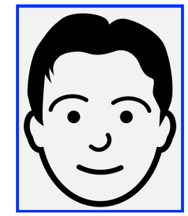
Protocol 4: IBLTs

Gavin Andresen;
Rosenbaum and Russell

(prioritize TXN inv's)

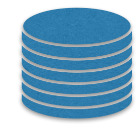
has block

Alice



Bob

wants block



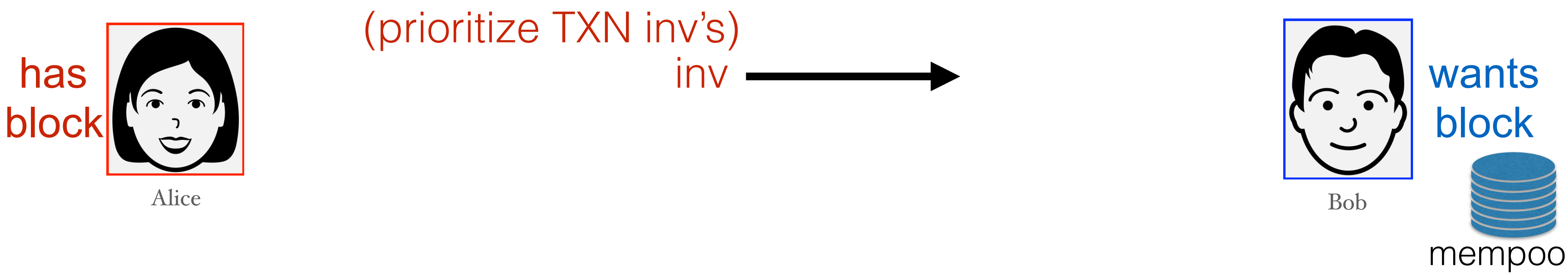
mempool

- Works very well until the receiver's mempool size is much larger than the block.

- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
  - But we don't know this value and don't want to waste roundtrip times failing.

Protocol 4: IBLTs

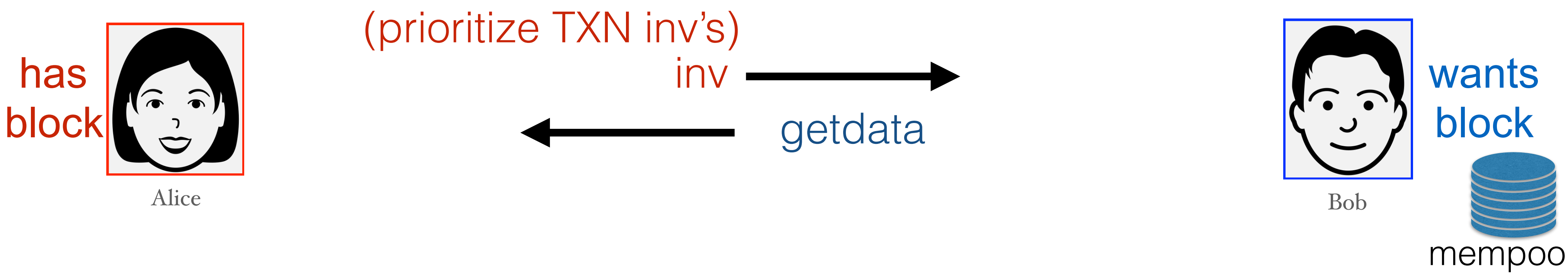
Gavin Andresen;
Rosenbaum and Russell



- Works very well until the receiver's mempool size is much larger than the block.
- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
  - But we don't know this value and don't want to waste roundtrip times failing.

Protocol 4: IBLTs

Gavin Andresen;
Rosenbaum and Russell

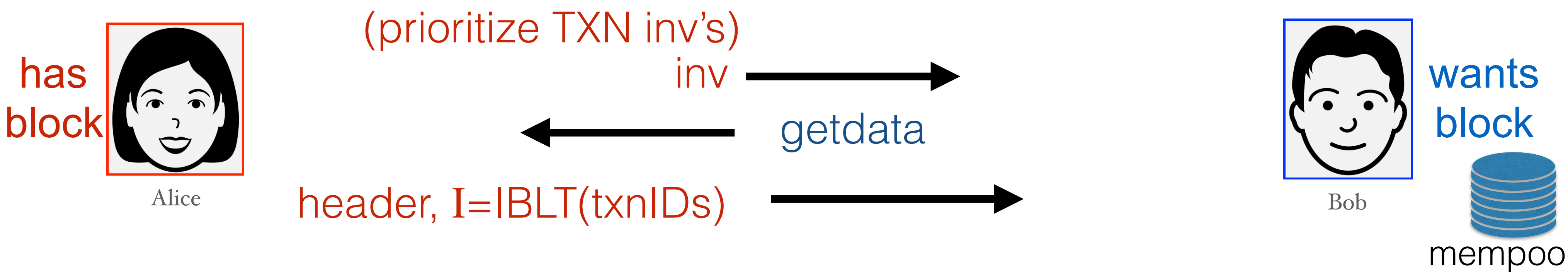


- Works very well until the receiver's mempool size is much larger than the block.

- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
  - But we don't know this value and don't want to waste roundtrip times failing.

Protocol 4: IBLTs

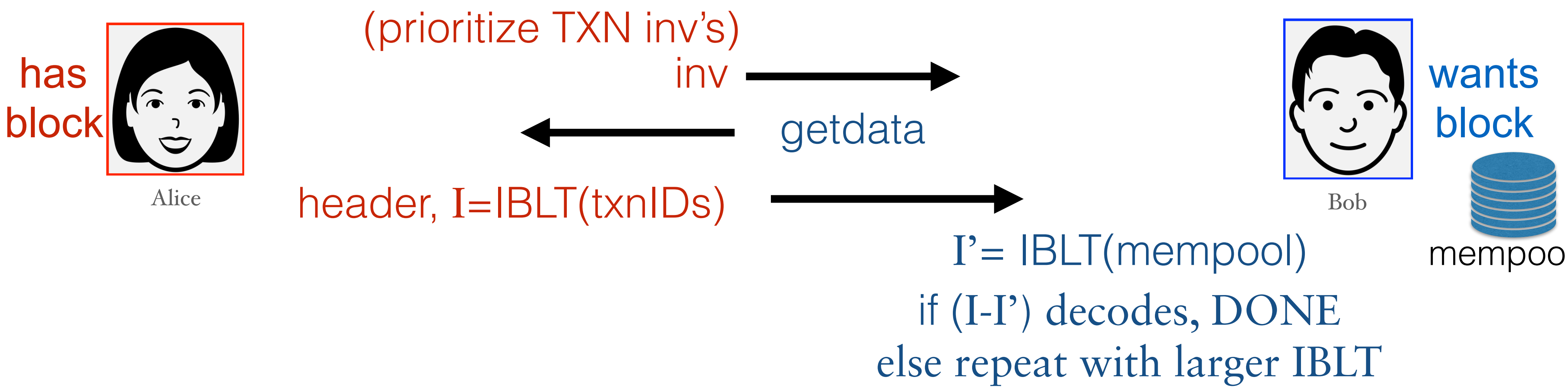
Gavin Andresen;
Rosenbaum and Russell



- Works very well until the receiver's mempool size is much larger than the block.
- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
  - But we don't know this value and don't want to waste roundtrip times failing.

Protocol 4: IBLTs

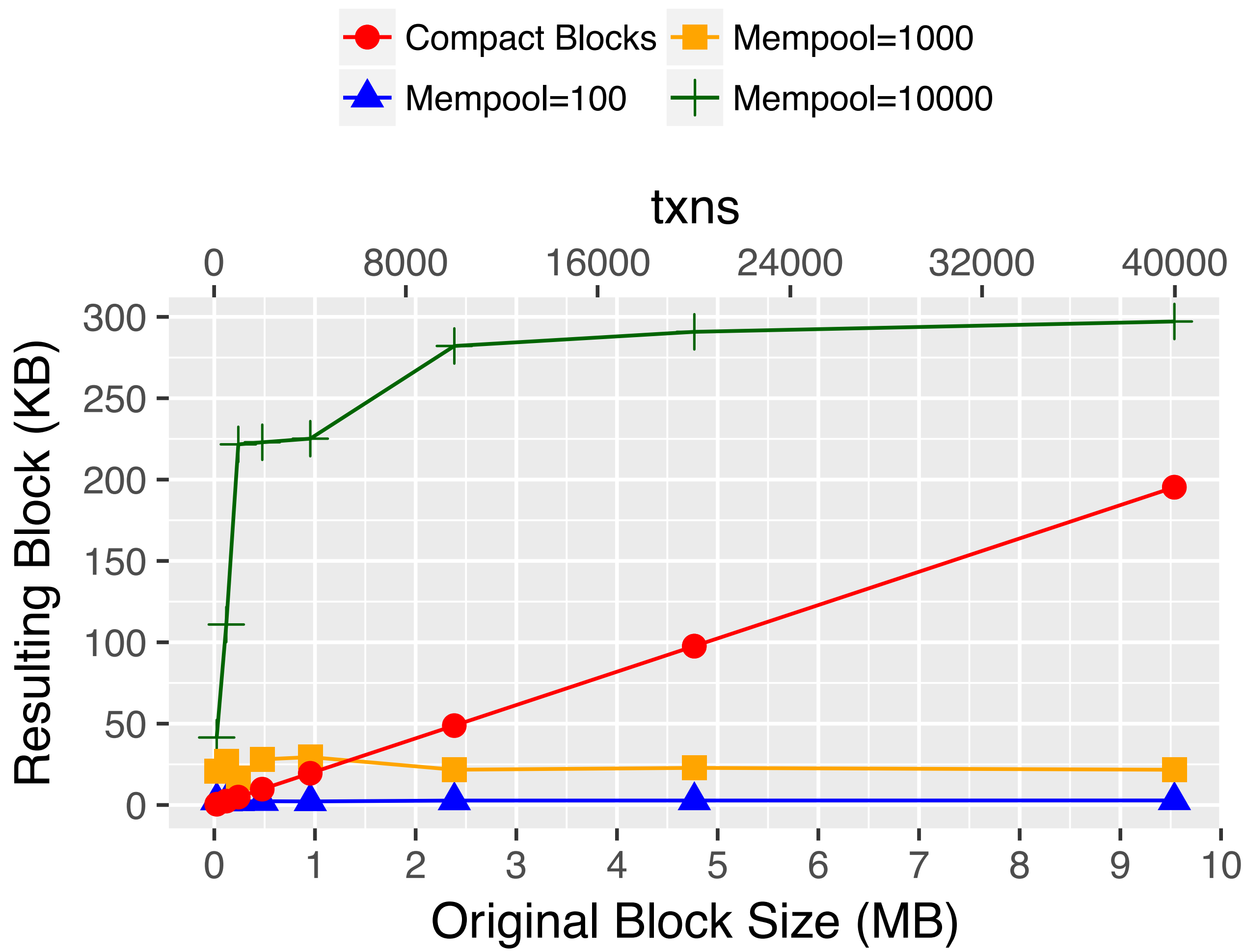
Gavin Andresen;
Rosenbaum and Russell



- Works very well until the receiver's mempool size is much larger than the block.
- The size of the IBLT will depend on the symmetric difference between the block and the receiver's mempool.
    - But we don't know this value and don't want to waste roundtrip times failing.

Performance

- Bytes are proportional to symmetric difference between block and mempool.
- Can we do better? Yes!



U N I V E R S

Protocol 5: Graphene

- It's expensive to make Bloom Filters when symmetric difference is high.
  It's expensive to make IBLTs when symmetric difference is high.

- Solution:
  - **use a Bloom Filter to reduce the symmetric difference between block and mempool.**
  - **use the IBLT to recover from small errors in the Bloom Filter**

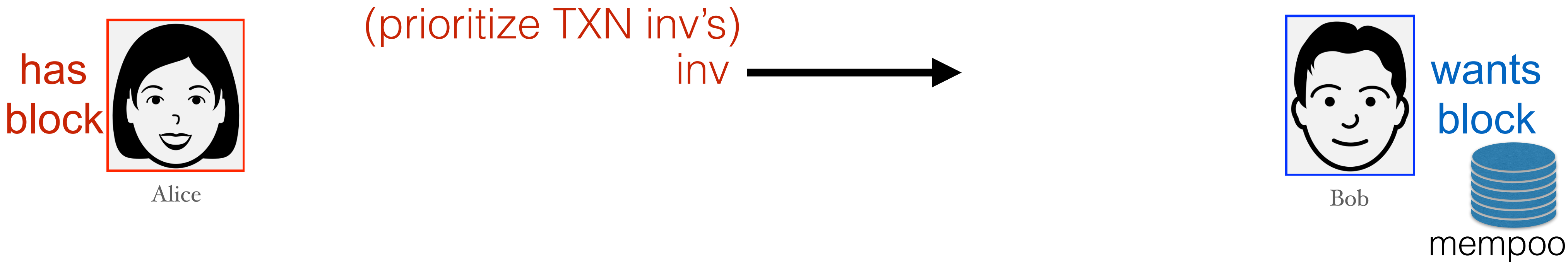
- We don't need a very low FPR for the Bloom Filter because the IBLT will help us recover.
  - Recall that the size of the IBLT is based on only the difference between two lists.

Optimally Small

- We shrink the Bloom filter to an FPR=1/m.
- We expect one false positive.
    - Make an IBLT expecting just one difference. It will be a small IBLT.
    - The output of comparing the two IBLTs will be exactly which txnID is the false positive.
- It turns out, we can parameterize the FPR and IBLT together so that the sum bytes are optimally small.
    - Roughly, given a block of **n** transactions and a mempool of **m** transactions, the FPR that provides the optimally small sized of IBLT and BF is

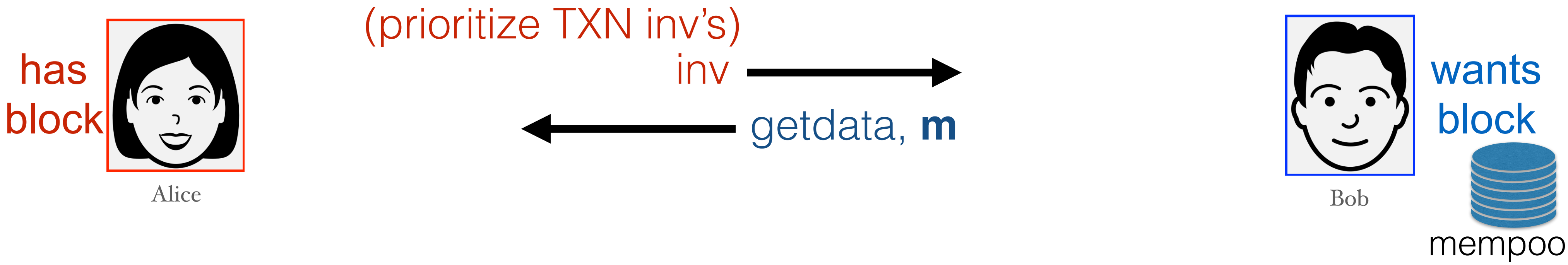
$$\text{FPR} = \frac{n}{132 \cdot (m-n)\ln^2(2)}$$

Protocol 5: Graphene



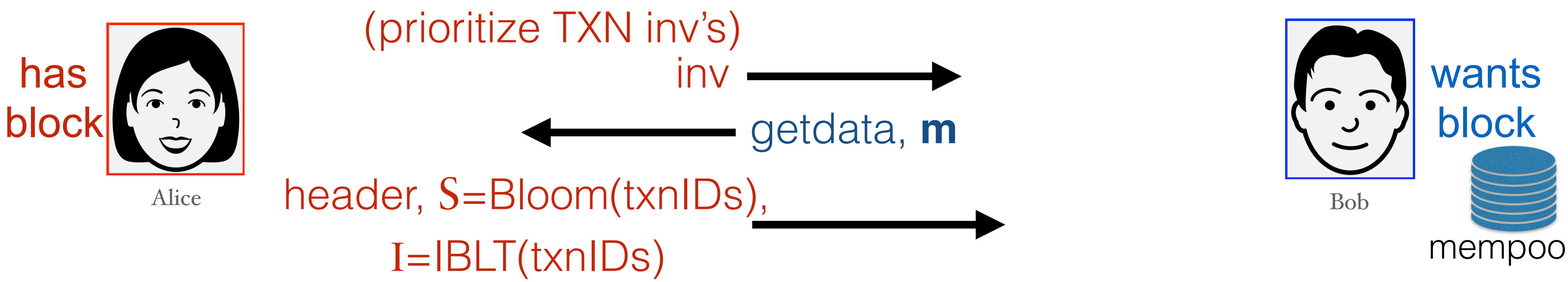
- We ensure that the IBLT decodes by setting the FPR correctly.
  - Decode failure is 1 in a 1000.

Protocol 5: Graphene



- We ensure that the IBLT decodes by setting the FPR correctly.
    - Decode failure is 1 in a 1000.

Protocol 5: Graphene



- We ensure that the IBLT decodes by setting the FPR correctly.
  - Decode failure is 1 in a 1000.

Protocol 5: Graphene

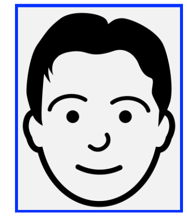
has block

Alice

(prioritize TXN inv's)
inv →

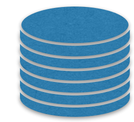
← getdata, **m**

header, S=Bloom(txnIDs),
I=IBLT(txnIDs) →



Bob

wants block

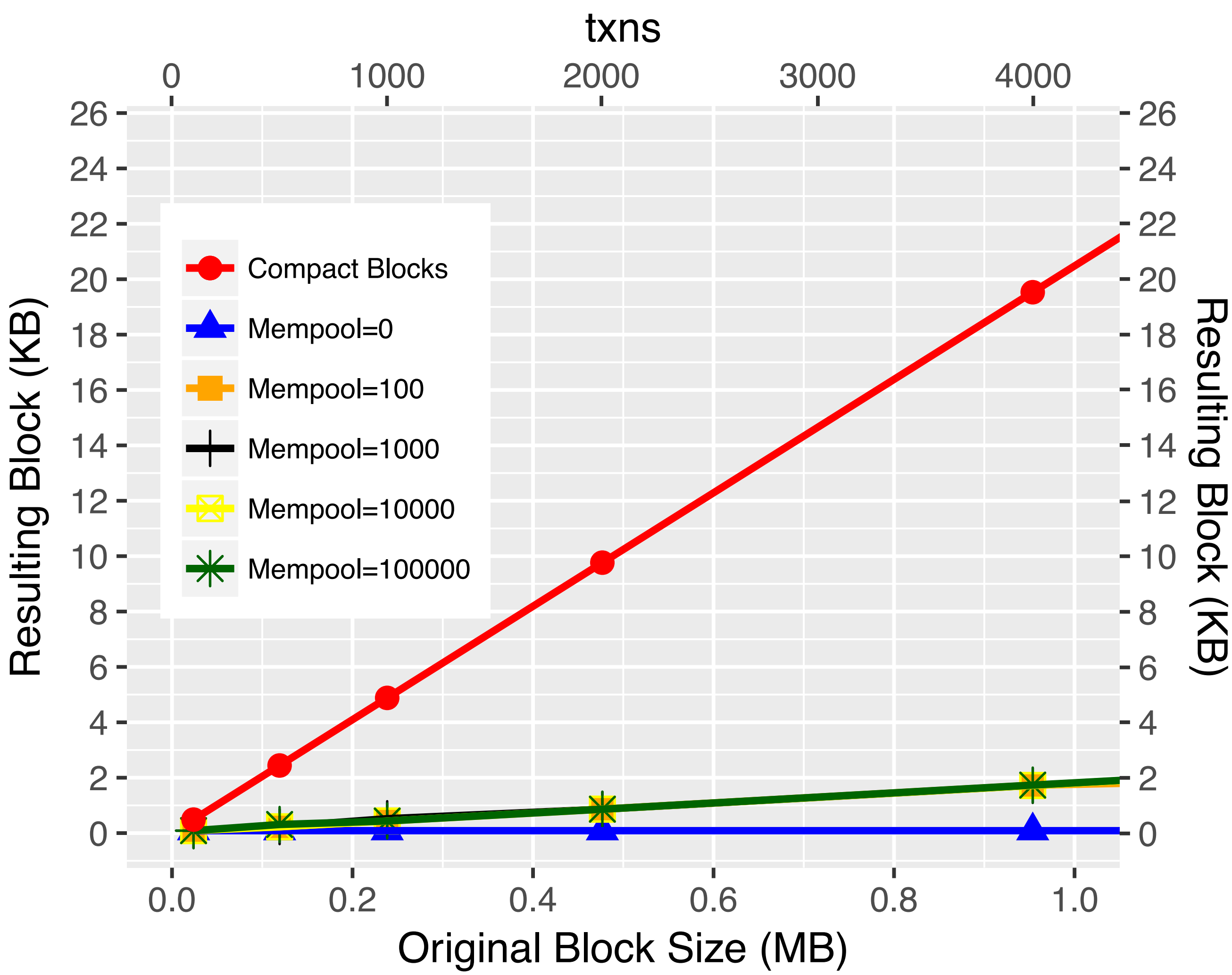


mempool

m' = mempool found in S
I'= IBLT(m')
if I-I' decodes, DONE;
else repeat with larger IBLT

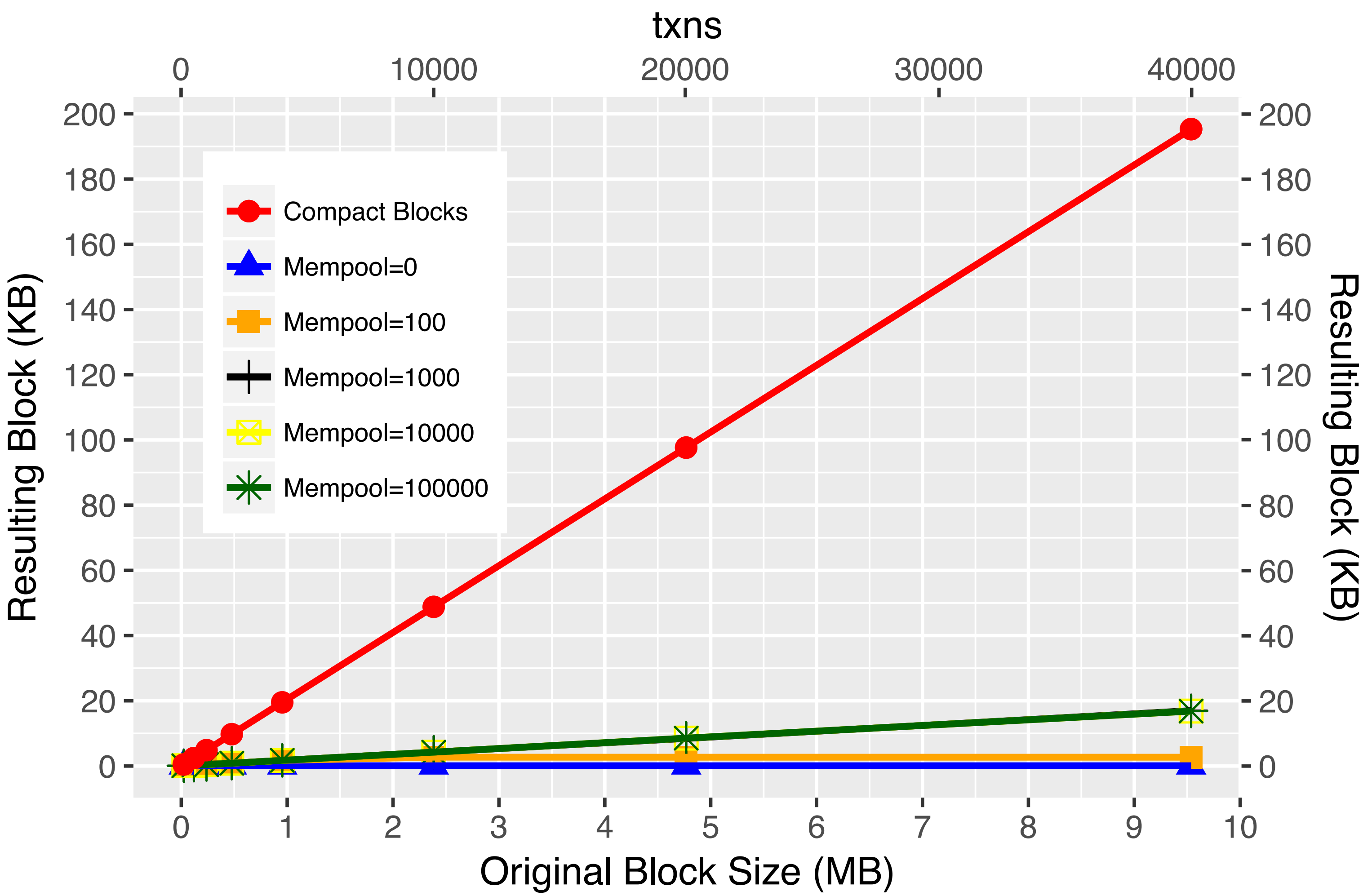
- We ensure that the IBLT decodes by setting the FPR correctly.
    - Decode failure is 1 in a 1000.

Graphene Performance



S AMHERST

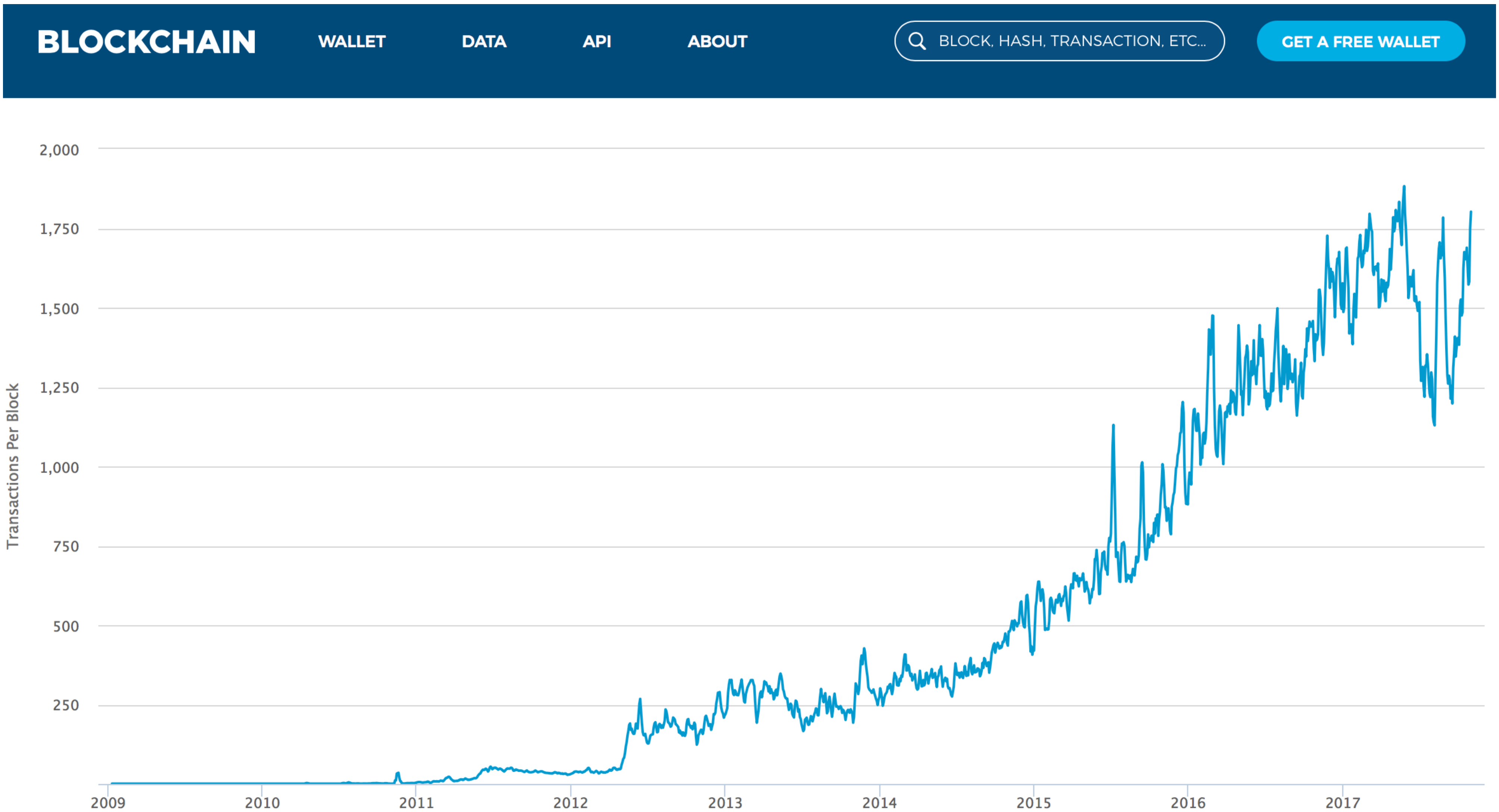
Graphene Performance

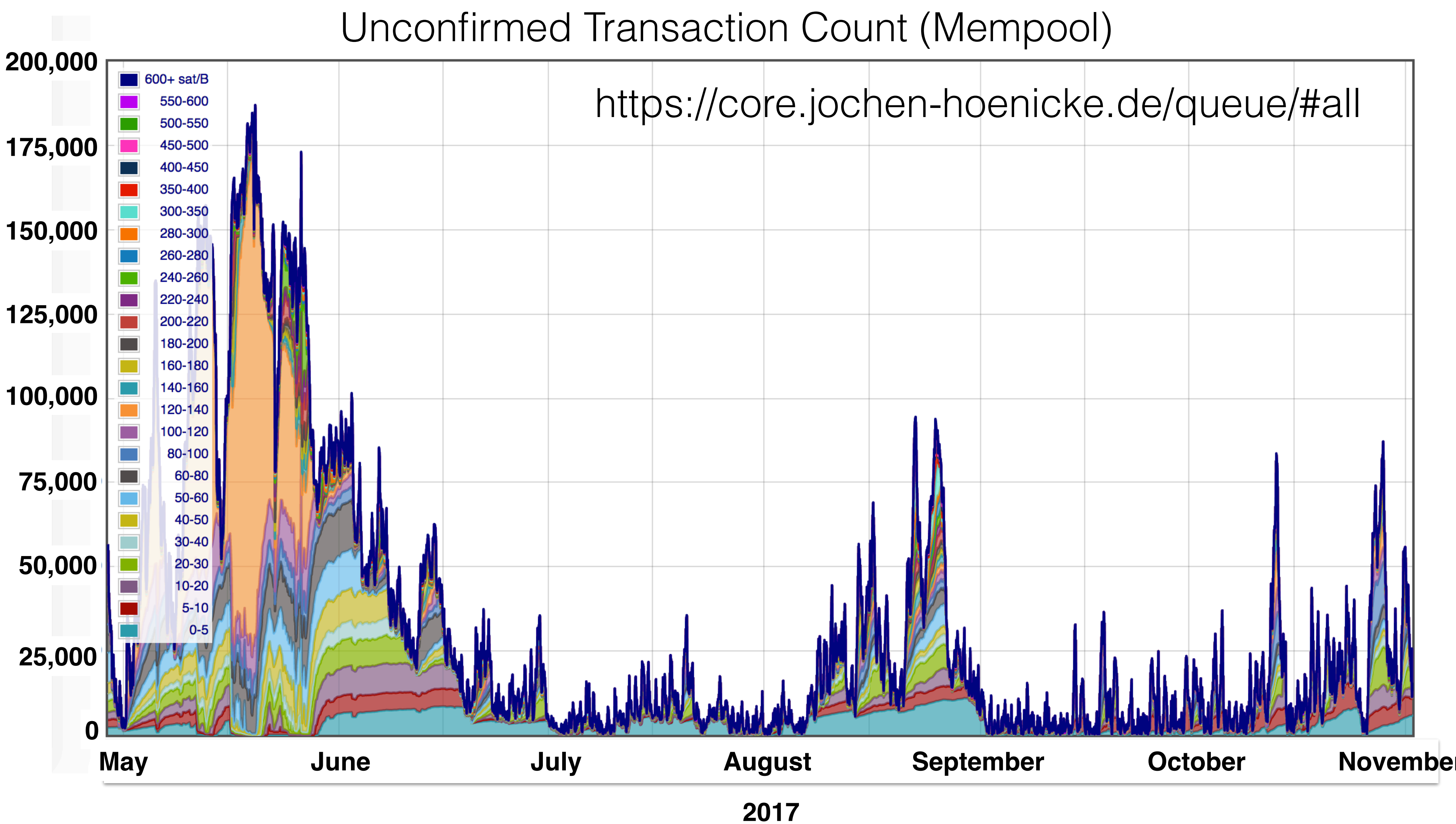


UNIVERSITY OF MASSACHUSETTS AMHERST

Conclusions

- **Graphene's block announcements are $\frac{1}{10}$ the size of current methods.**
  - Fits within one IP packet
  - No increase in roundtrip time of Compact Blocks
  - Not a significant use of storage or CPU.
- Combines two known tools from set reconciliation literature in a nifty way.
  - Bloom Filters and IBLTs
- PDF: **http:forensics.cs.umass.edu/graphene**


BLOCKCHAIN
WALLET
DATA
API
ABOUT
BLOCK, HASH, TRANSACTION, ETC...
GET A FREE WALLET
Transactions Per Block
2,000
1,750
1,500
1,250
1,000
750
500
250
2009
2010
2011
2012
2013
2014
2015
2016
2017


Unconfirmed Transaction Count (Mempool)
https://core.jochen-hoenicke.de/queue/#all
200,000
175,000
150,000
125,000
100,000
75,000
50,000
25,000
0
600+ sat/B
550-600
500-550
450-500
400-450
350-400
300-350
280-300
260-280
240-260
220-240
200-220
180-200
160-180
140-160
120-140
100-120
80-100
60-80
50-60
40-50
30-40
20-30
10-20
5-10
0-5
May
June
July
August
September
October
November
2017