On-The-Fly Verification of Rateless Erasure Codes

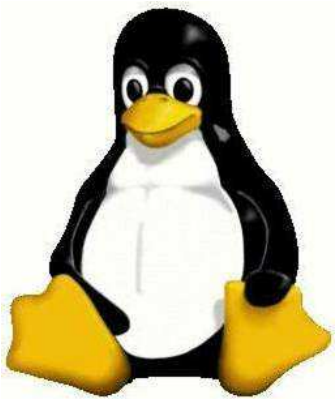
Max Krohn (MIT CSAIL)
Michael Freedman and David Mazières (NYU)

Multicast Authentication: Dead/Exhausted

On-The-Fly Verification of Rateless Erasure Codes

Max Krohn (MIT CSAIL)
Michael Freedman and David Mazières (NYU)

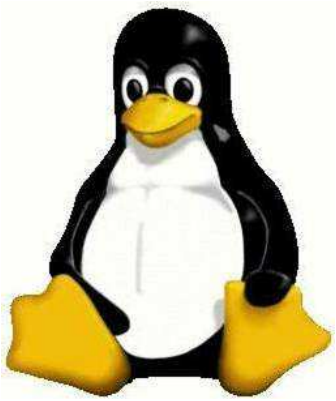
The Setting



- A large file F
  - Linux ISO (650MB)
- $H(F)$ is available
  - signed by Publisher (RedHat)
- A handful of untrusted sources/mirrors $S_1,\ldots S_8$

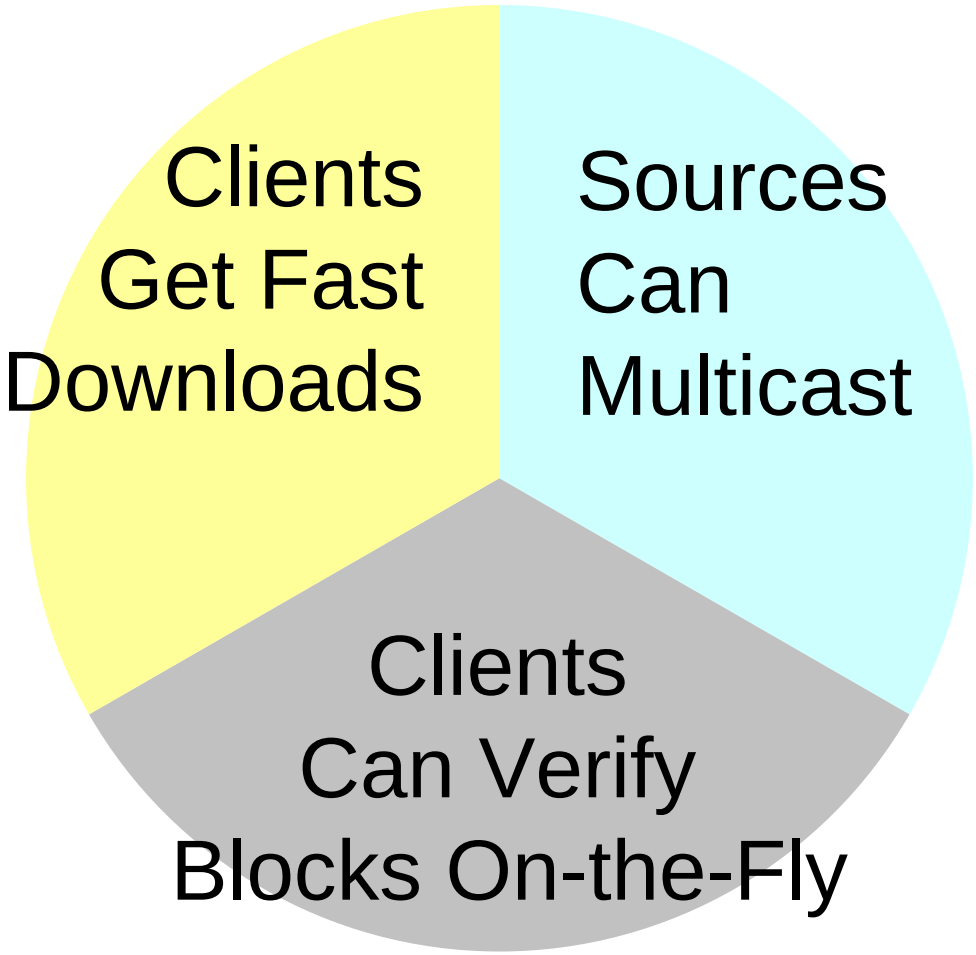
A Handful of Senders

The Setting



- A large file F
  - Linux ISO (650MB)
- $H(F)$ is available
  - signed by Publisher (RedHat)
- A handful of untrusted sources $S_1,\ldots S_8$
  - Their aggregate BW is limited
- A slew of receivers $R_1,\ldots,R_{1,000,000}$
  - Version 81.3 just released! Want it Now!

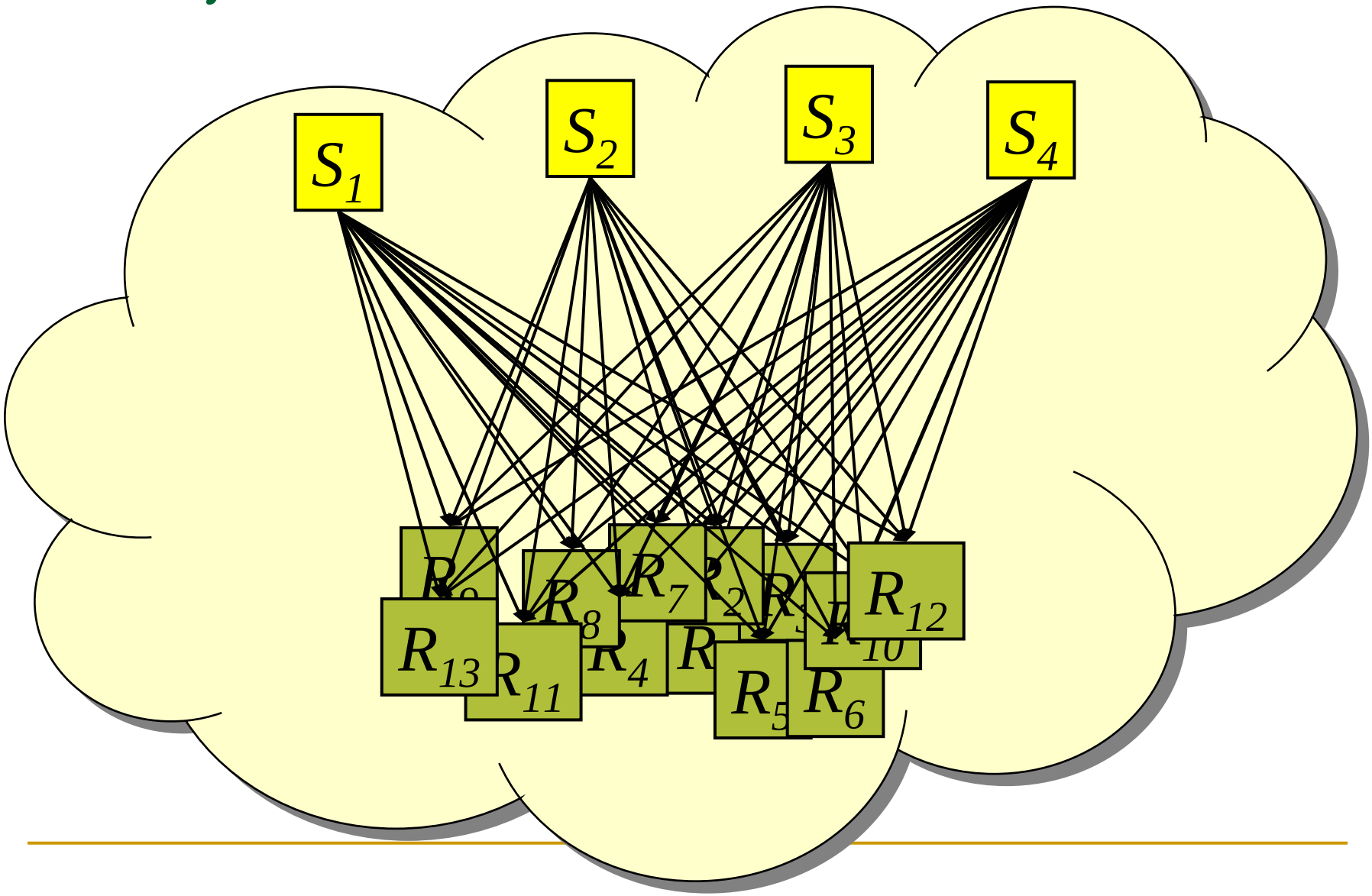
Three Desirable Properties



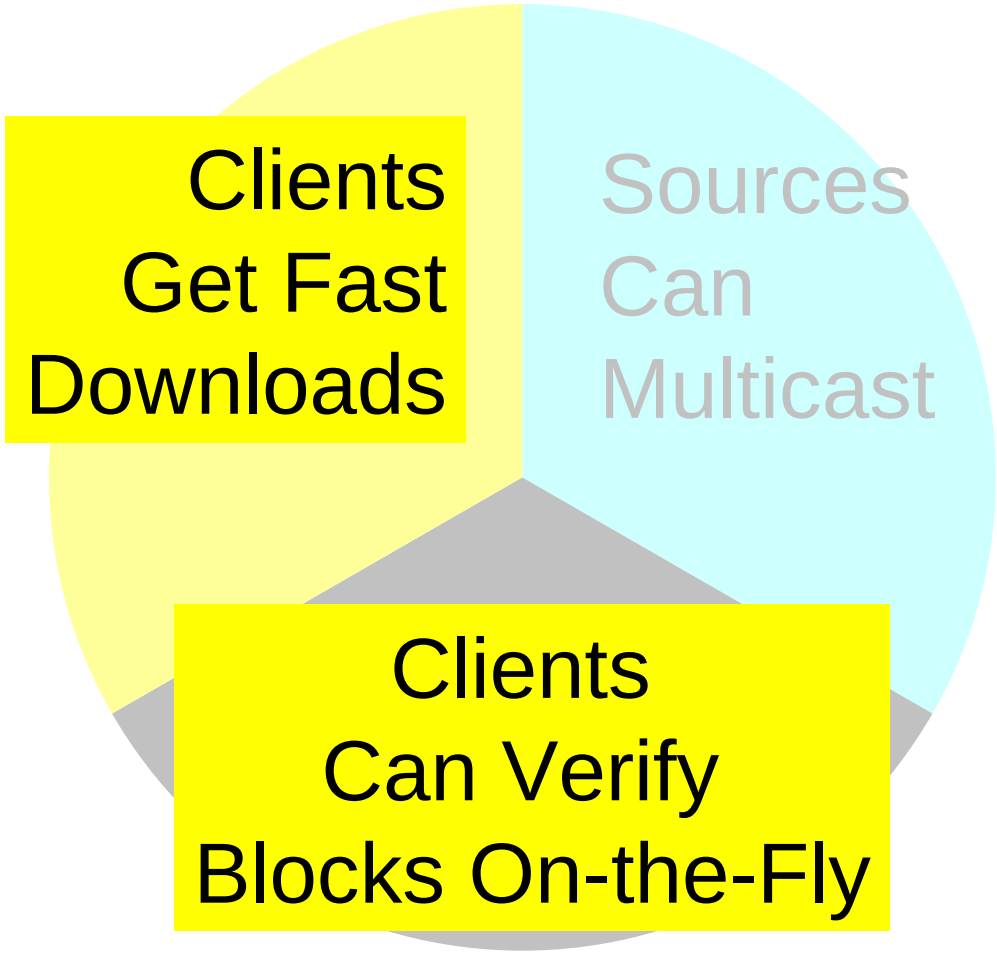
Receivers Get Fast, Verifiable Downloads

- The trusted publisher (RedHat)
  - Splits up F into n blocks
  - Hashes all blocks
  - Signs all hashes (or hash tree)
- Receivers:
  - Download and verify hashes
  - Download needed file blocks in parallel

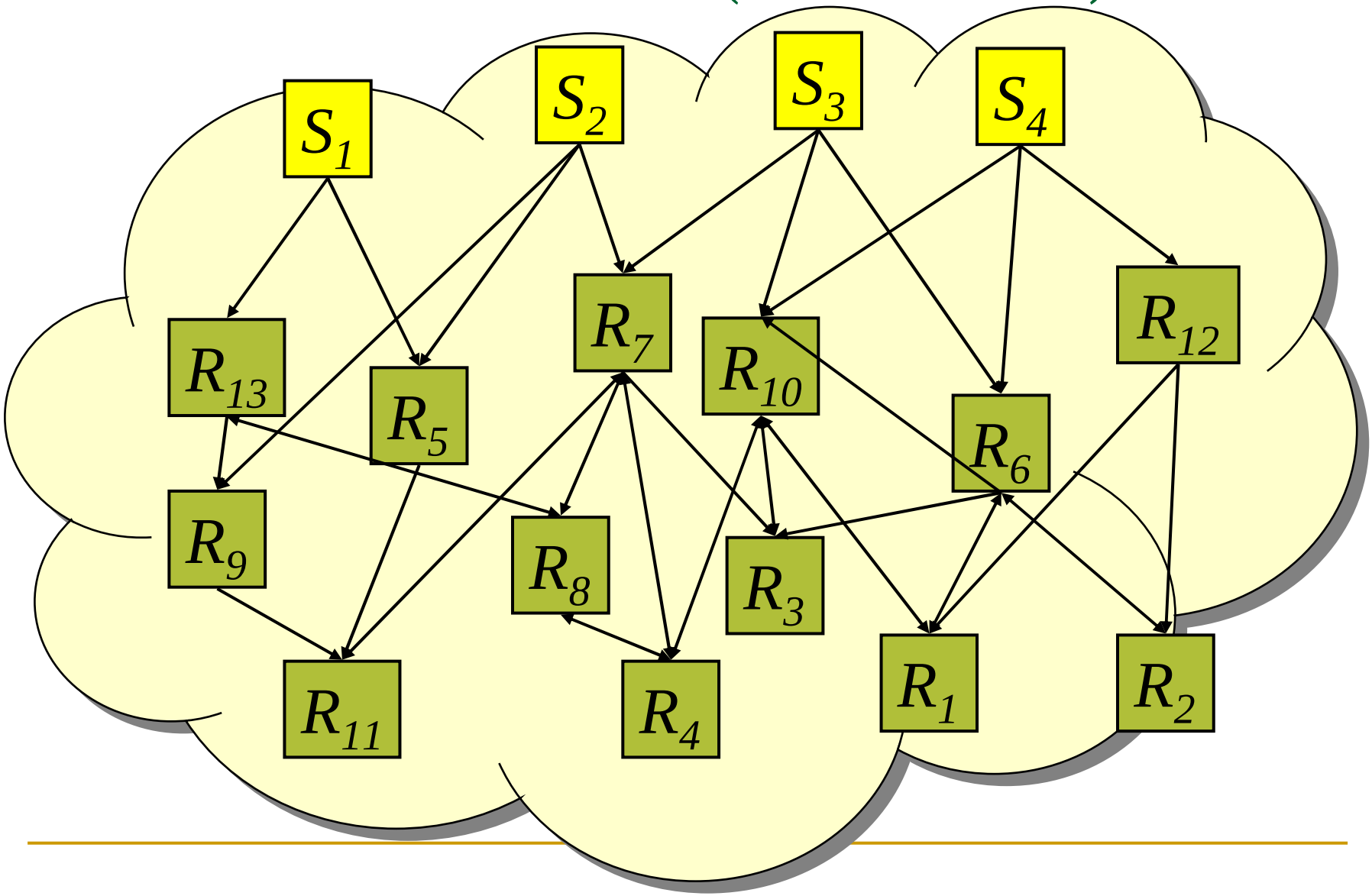
Everyone for Themselves



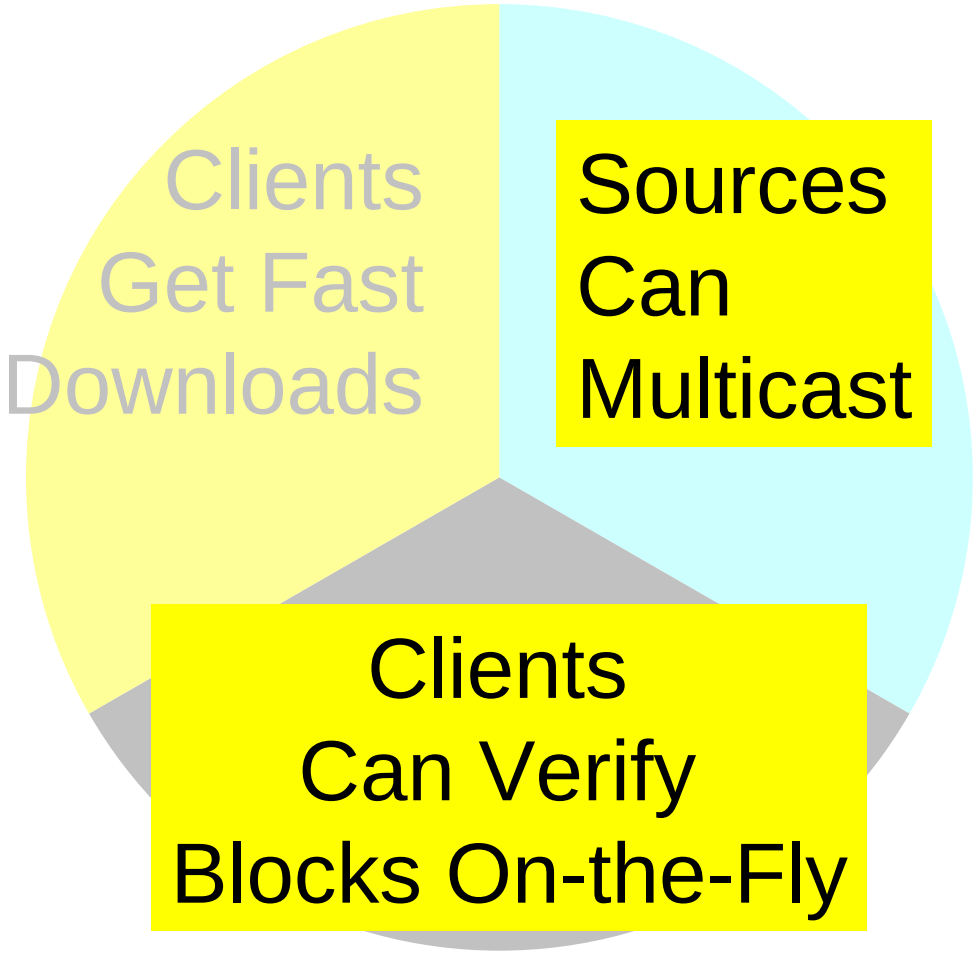
Everyone For Themselves



Verifiable Multicast (BitTorrent)

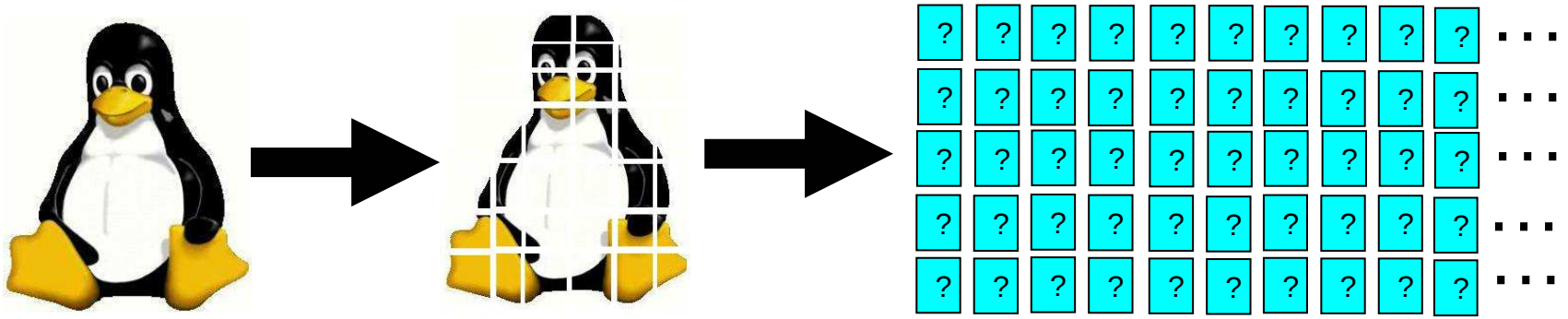


Verifiable Multicast (BitTorrent)



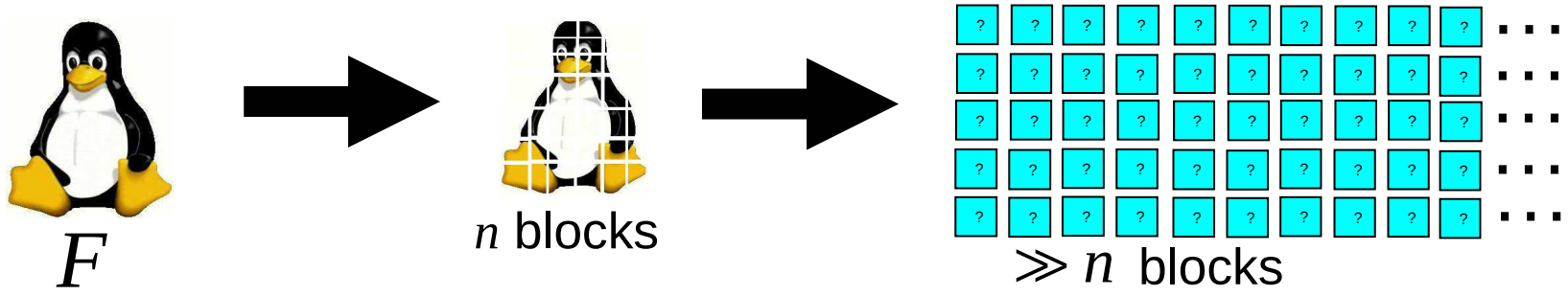
Multicast With Erasure Codes

- Sources erasure encode the file F

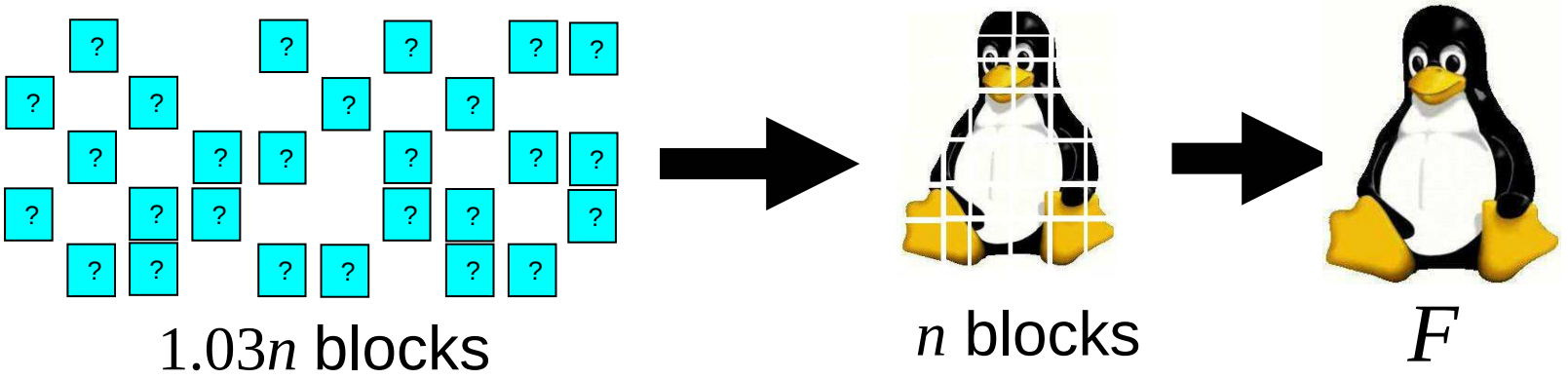


Multicast With Erasure Codes

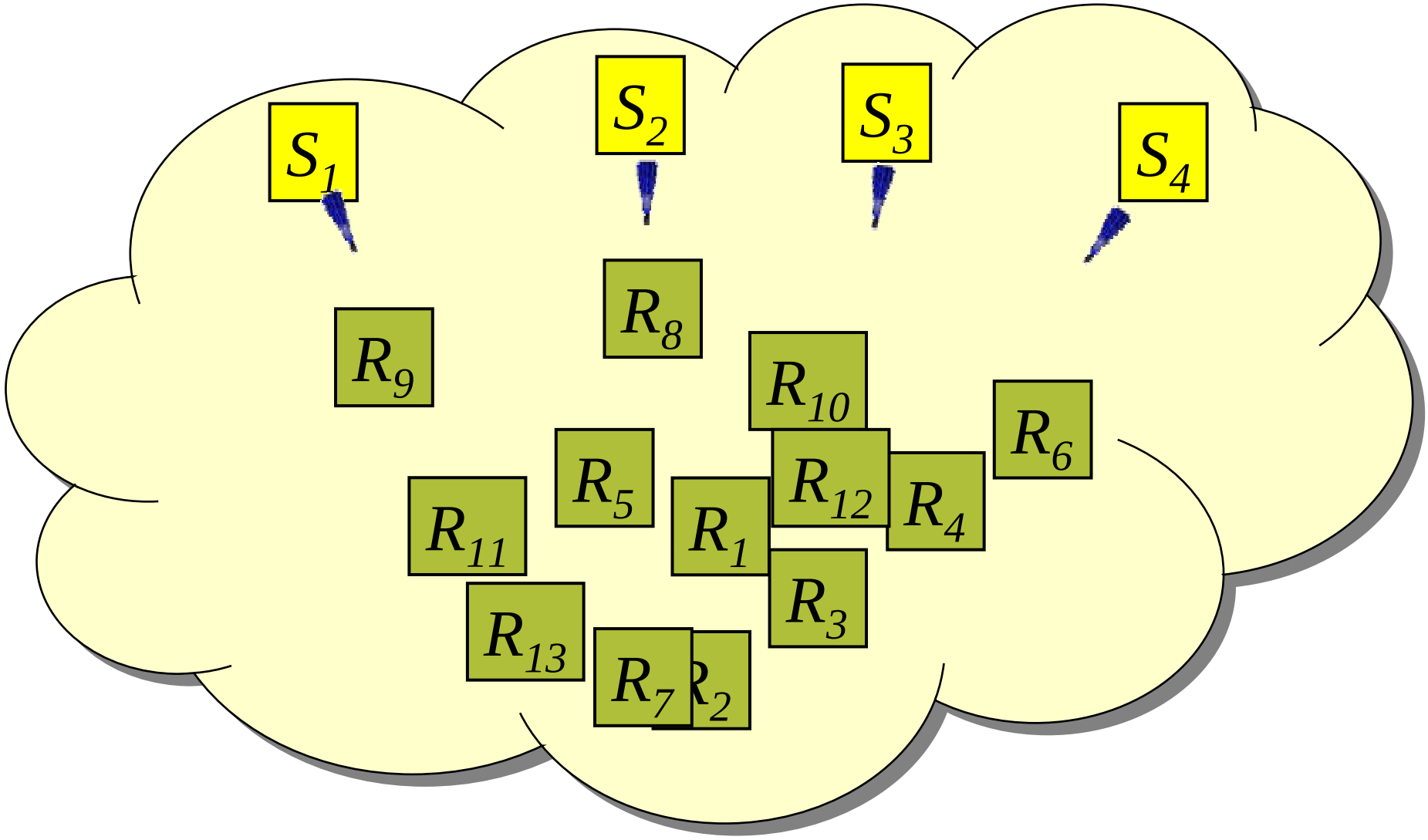
- Sources erasure encode the file F



- Receivers collect blocks and decode



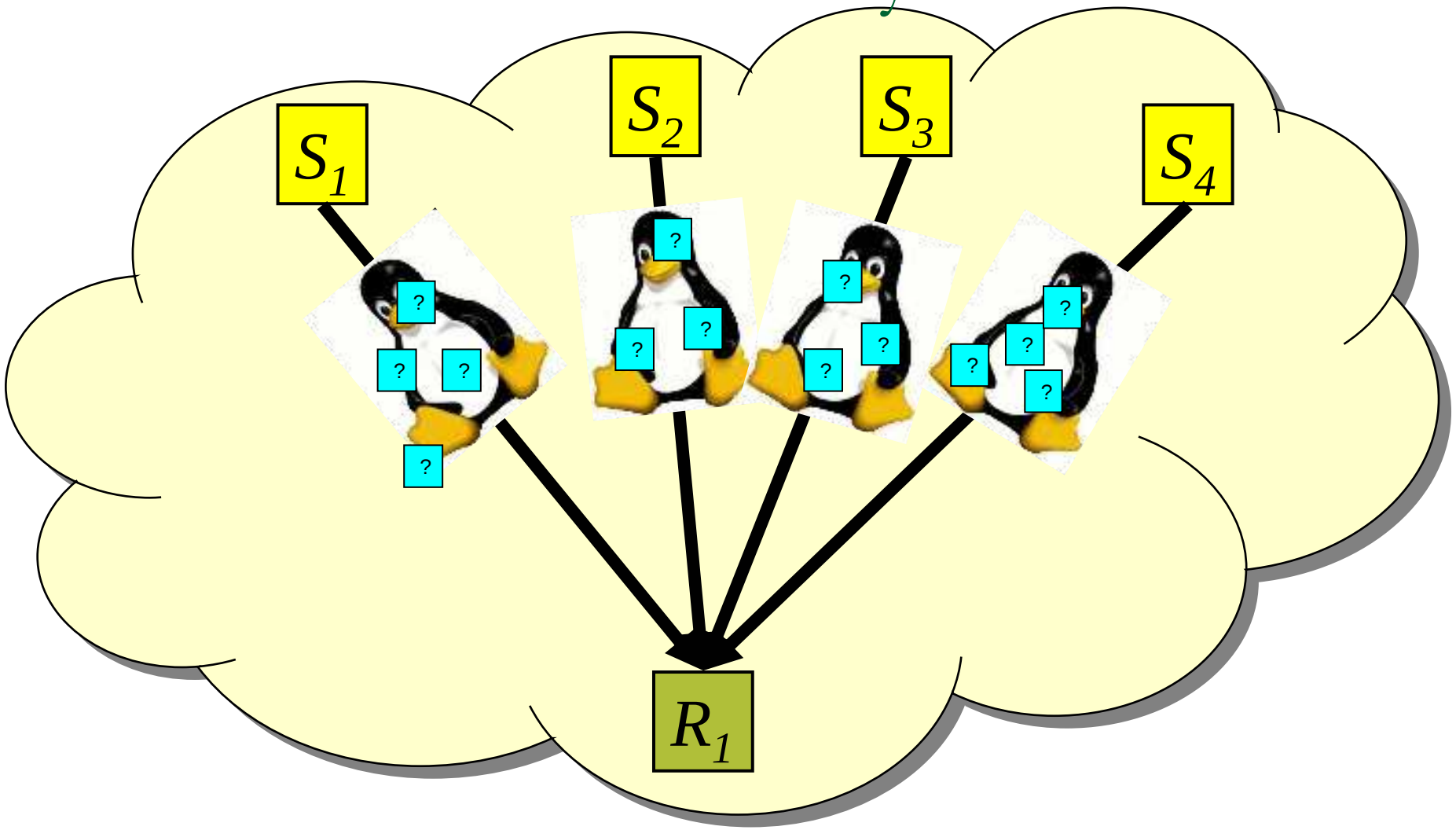
Multicast With Erasure Codes



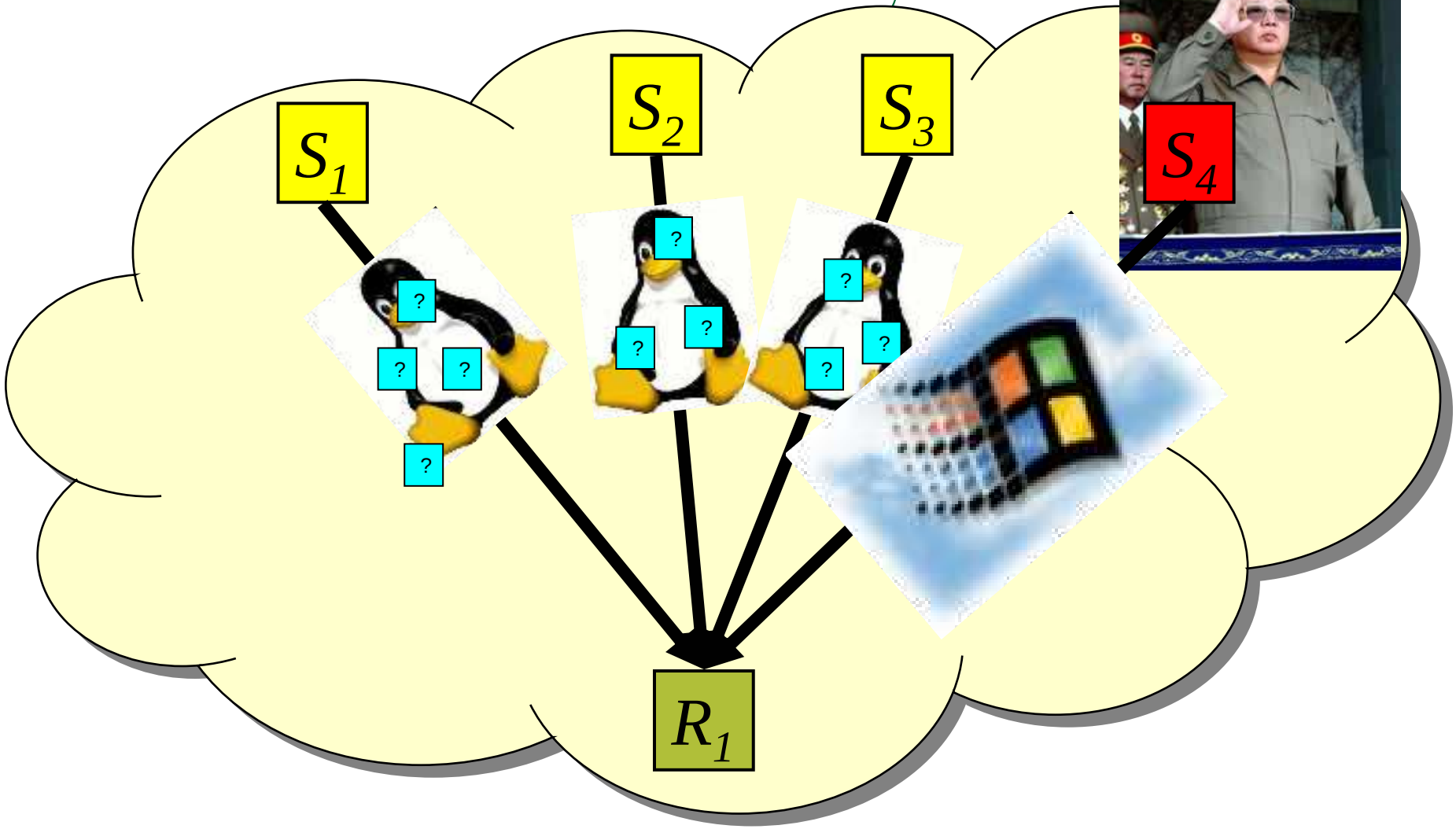
Multicast With Erasure Codes

- Bullet [SOSP 2003]
- SplitStream [SOSP 2003]
- Big Downloads [IPTPS 2003]
- Informed Content Delivery [SIGCOMM 2002]

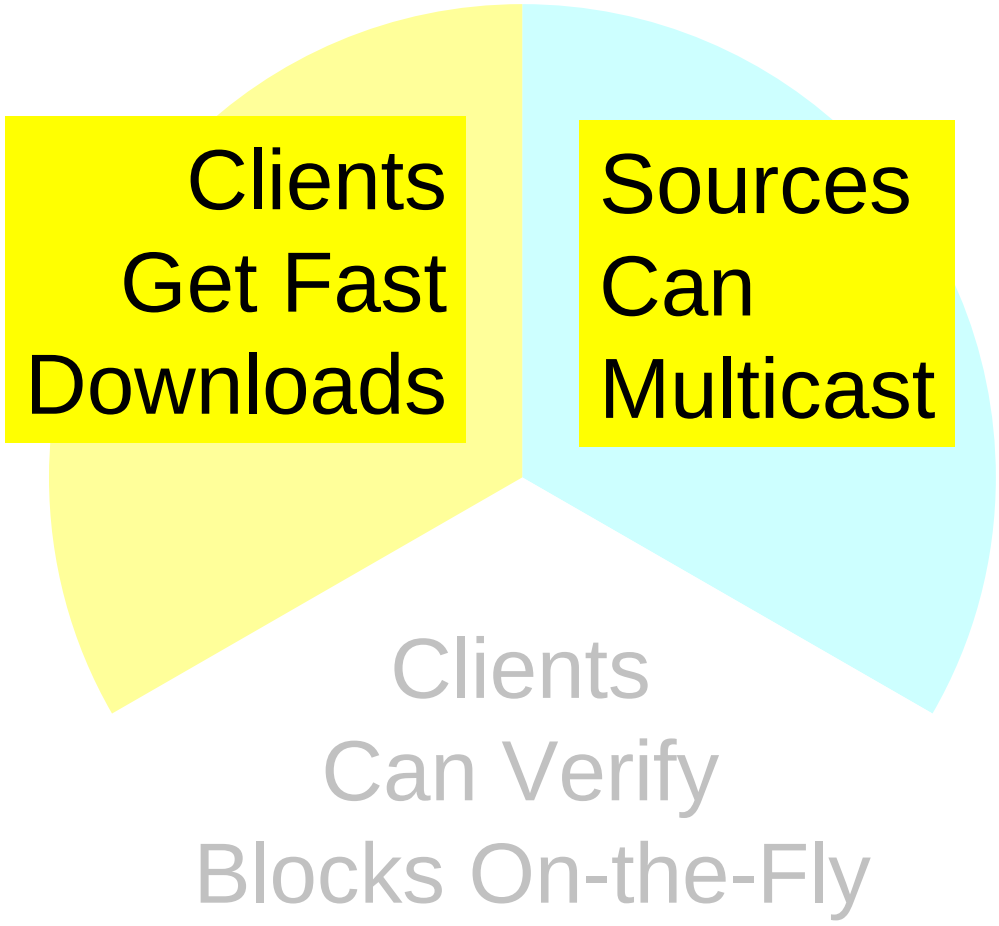
Receivers Cannot Verify Content



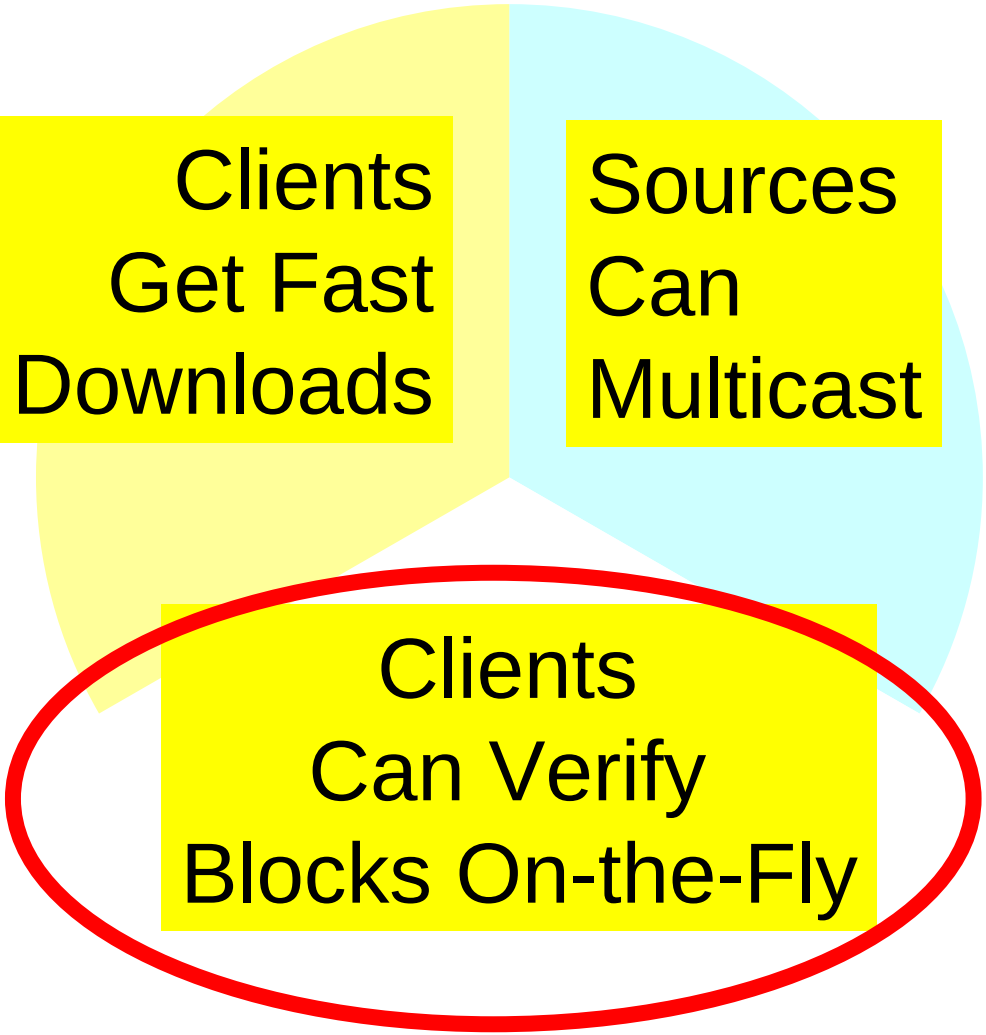
Receivers Cannot Verify Conte



Multicast With Erasure Codes

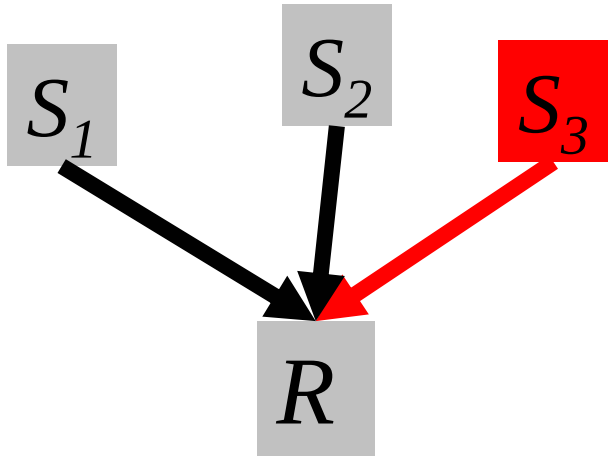


Multicast With Erasure Codes



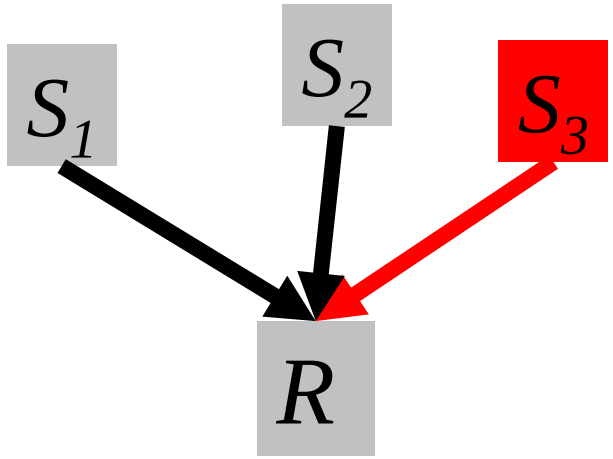
What is the Attack Goal?

- To corrupt the file.
- To waste bandwidth.



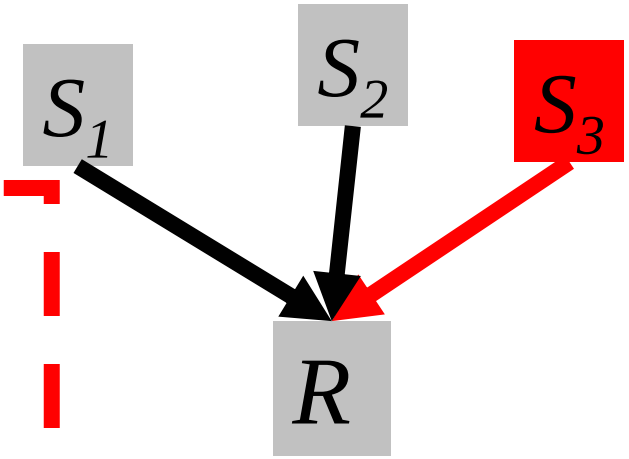
How To Attack?

- Send correct blocks but with skewed distributions.
  - “Distribution Attack”
- Send incorrect blocks
  - “Pollution Attack”
    - Karlof *et al.* [NDSS ’04]



Properties of a Solution to Pollution

- OK: Receivers can tell good from bad.
- Much better: Receivers can finger bad blocks *as they arrive*.

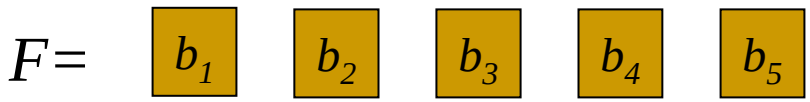


CONTRIBUTION

Outline

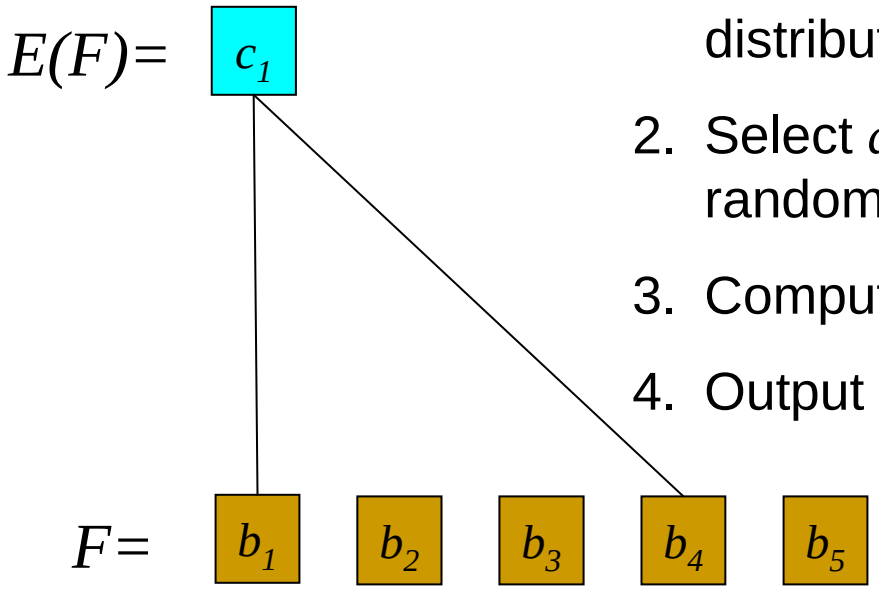
- Introduction
- **Review of LT Codes**
- Strawman #1
- Strawman #2
- Efficiently Catching Bad Blocks as They Arrive

LT-Codes [Luby, FOCS 2002]



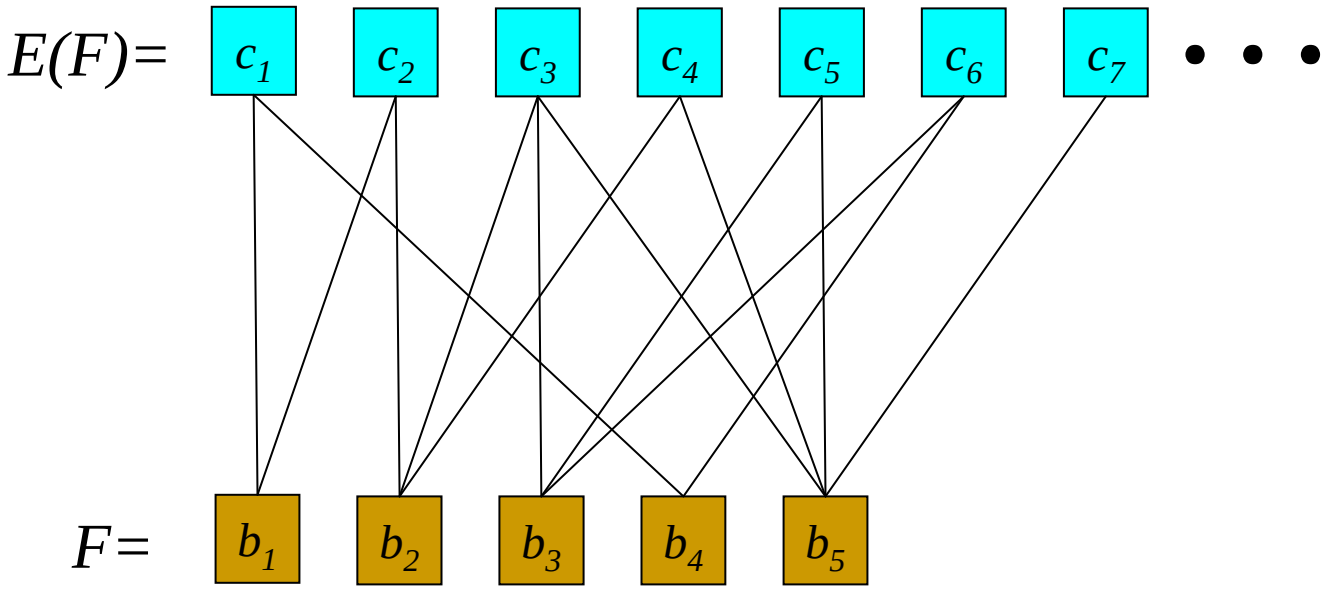
n=5 input blocks

LT-Codes – How To Encode

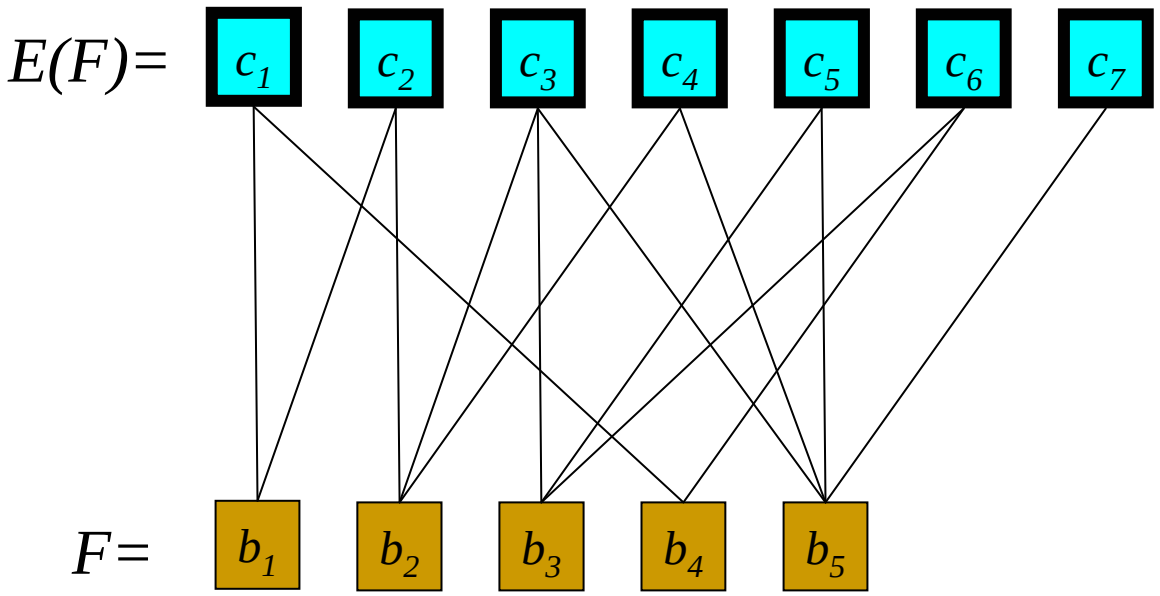


1. Pick *degree* d_1 from a pre-specified distribution. ($d_1=2$)
2. Select d_1 input blocks uniformly at random. (Pick b_1 and b_4)
3. Compute their sum. ($c_1 \leftarrow b_1 + b_4$)
4. Output $\langle c_1, \{1,4\} \rangle$

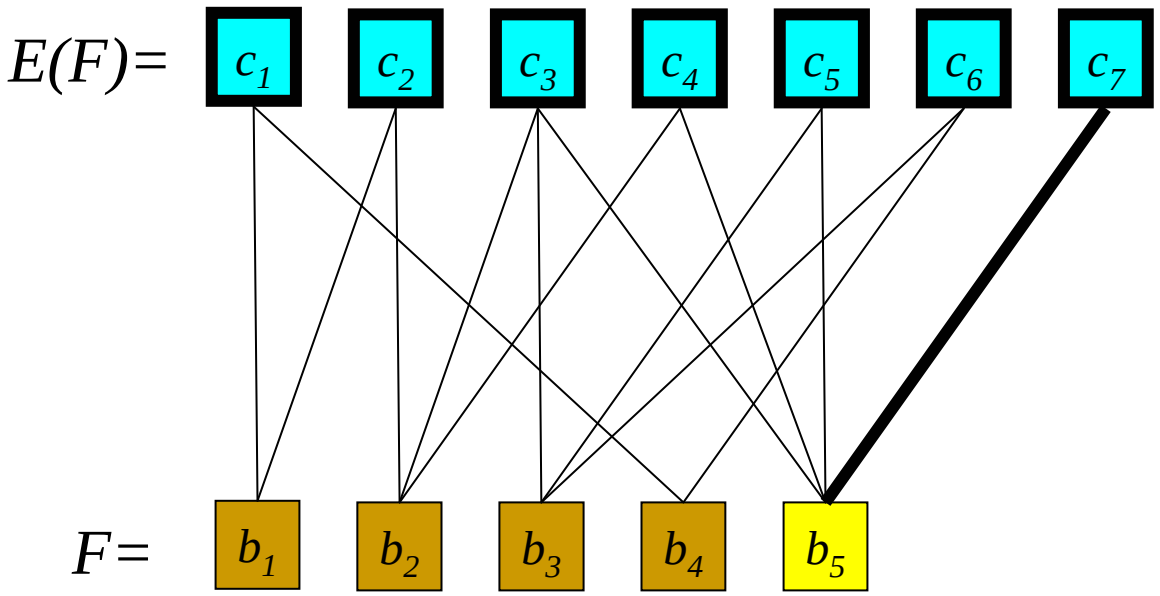
LT-Codes – How To Encode (cont'd)



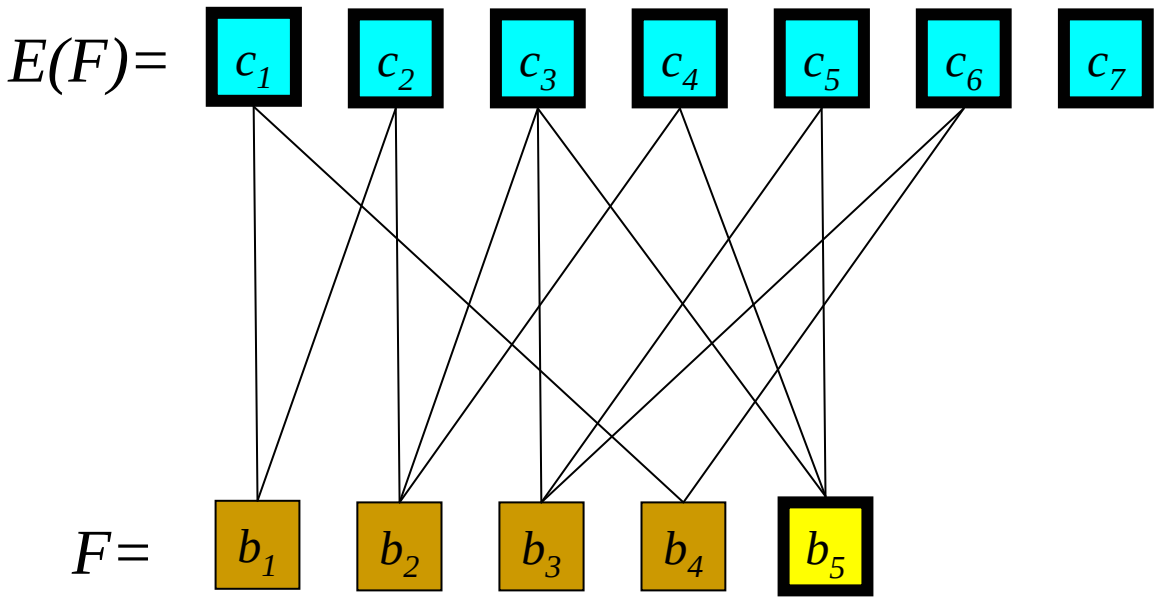
How To Decode



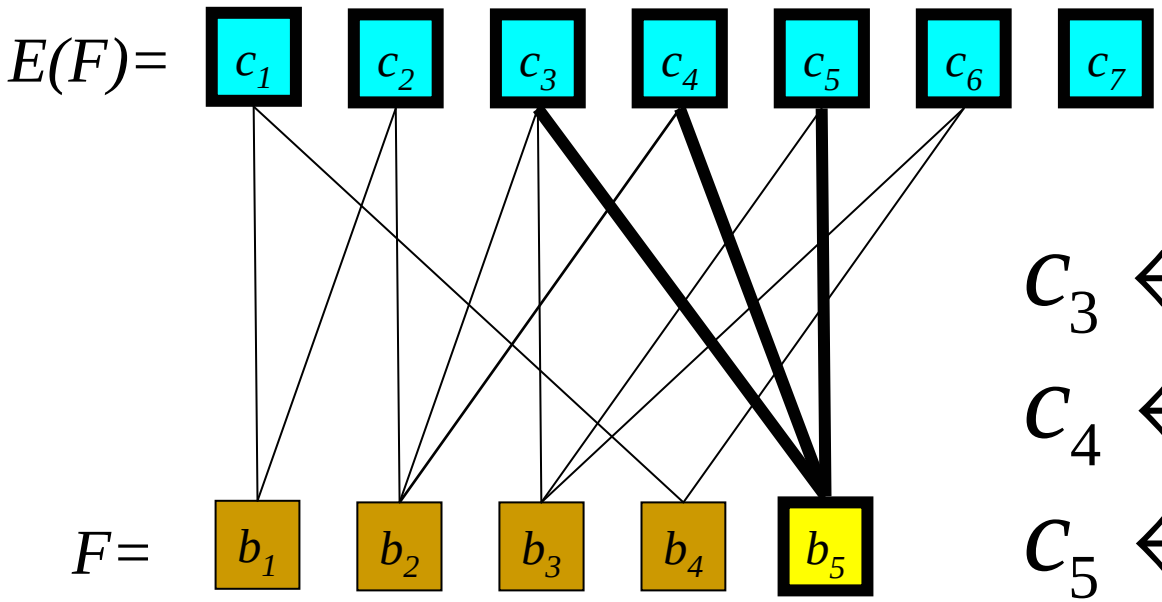
How To Decode



How To Decode

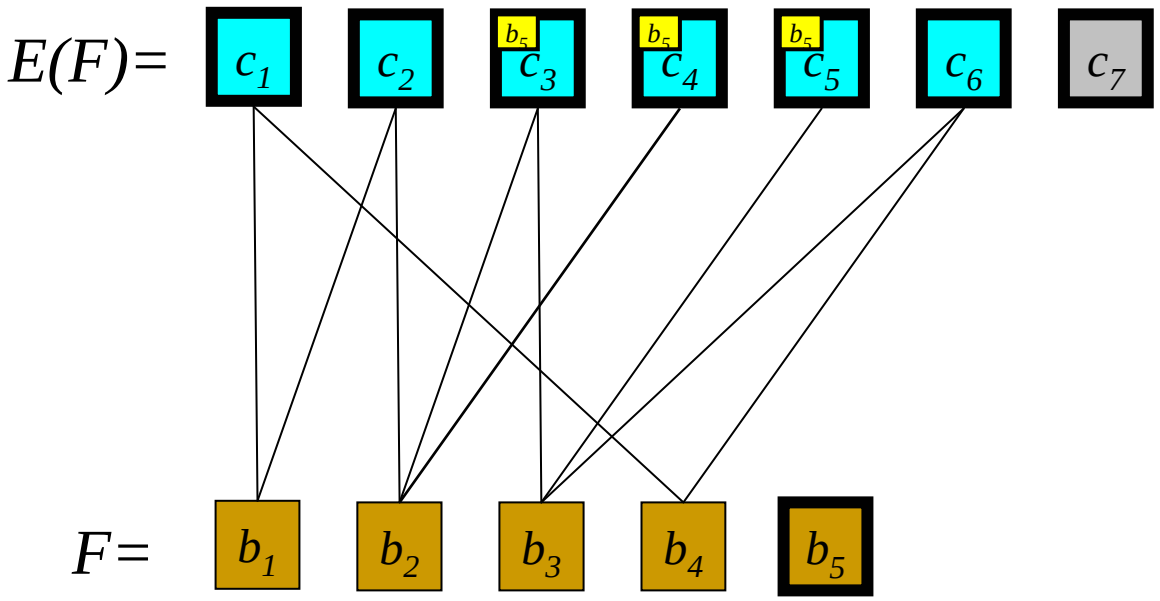


How To Decode

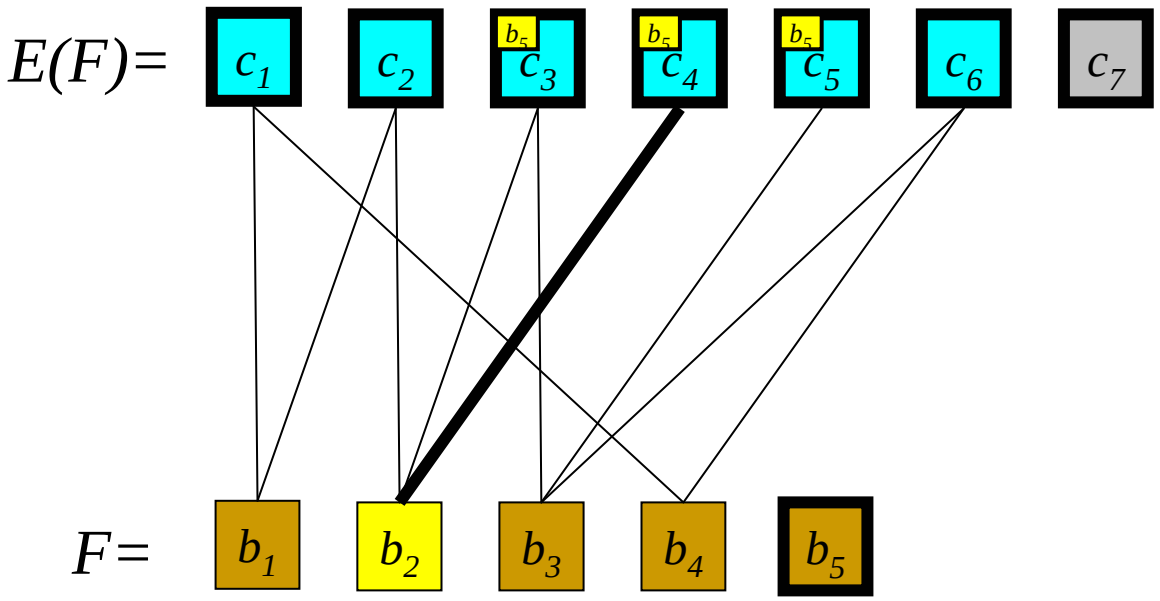


$$c_3 \leftarrow c_3 - b_5$$
$$c_4 \leftarrow c_4 - b_5$$
$$c_5 \leftarrow c_5 - b_5$$

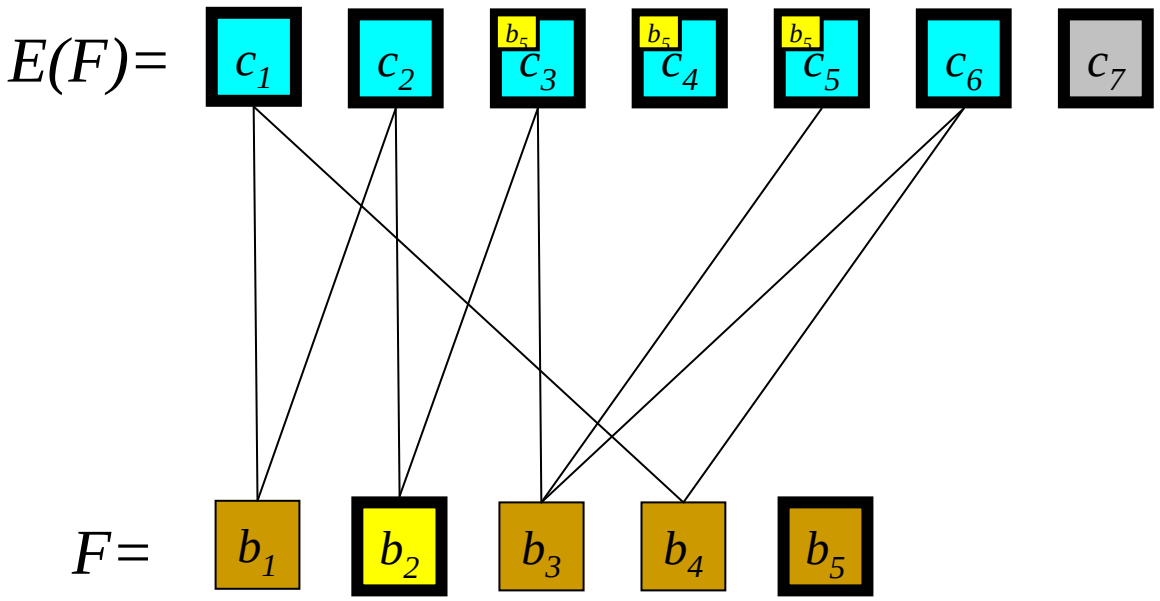
How To Decode



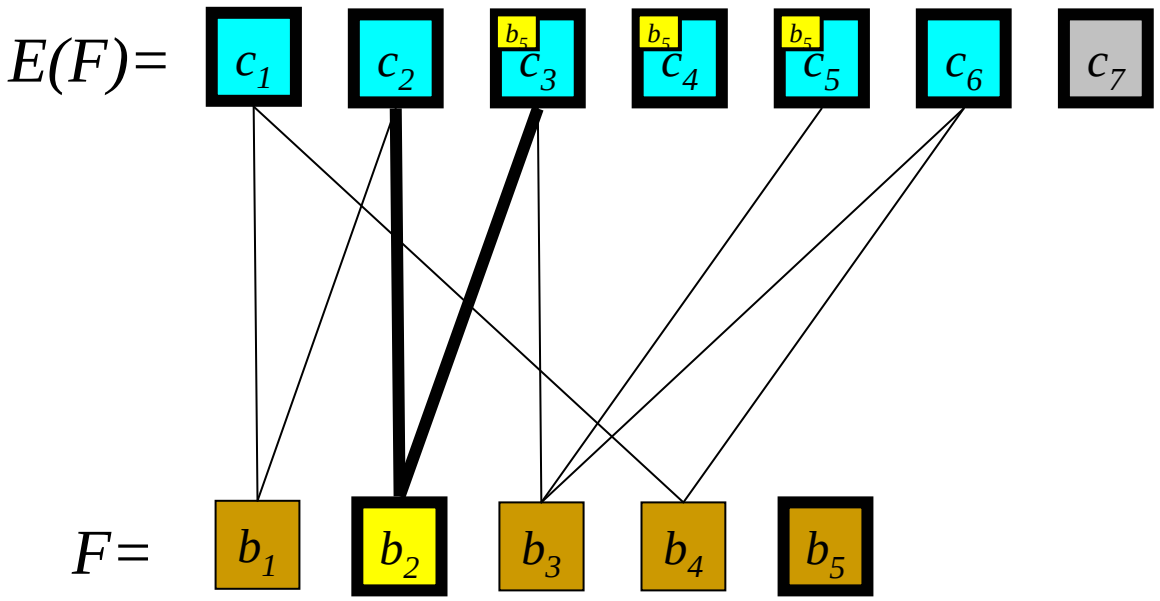
How To Decode



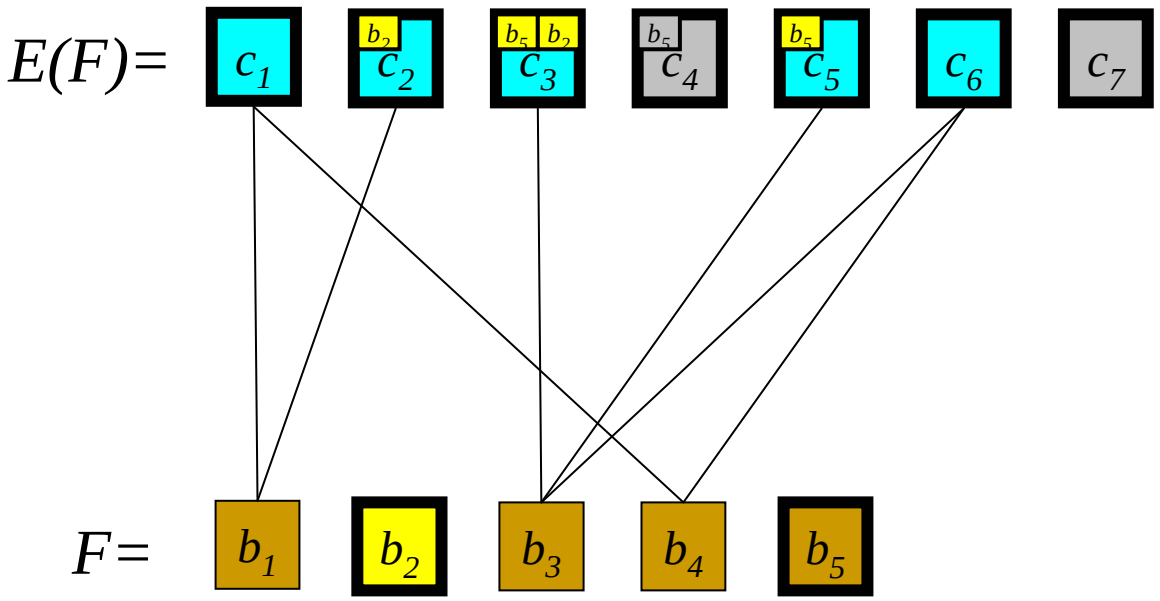
How To Decode



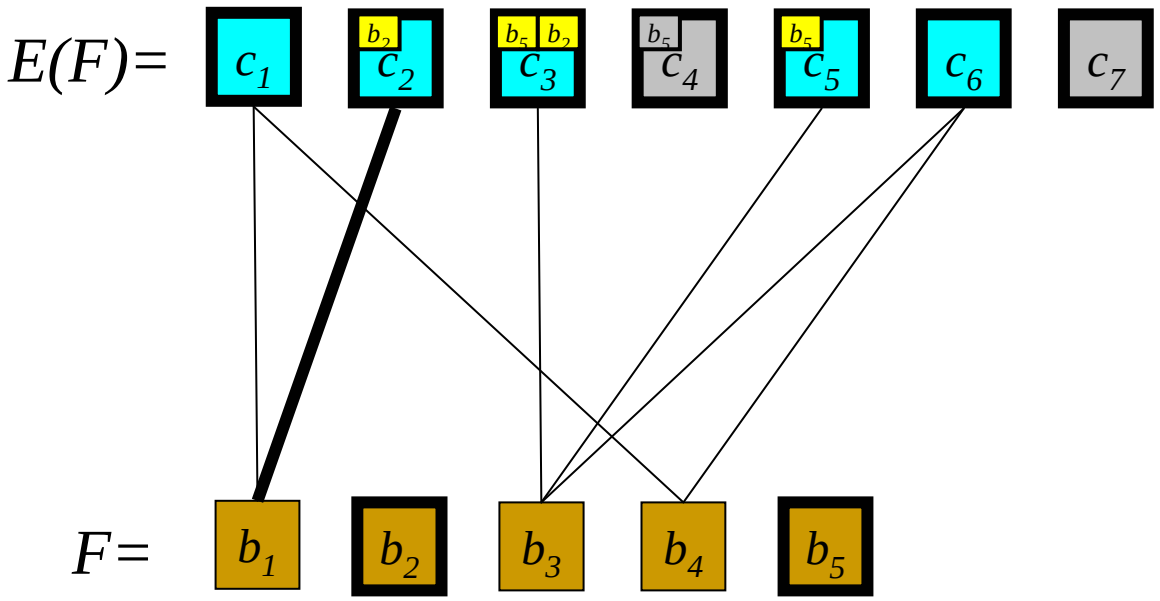
How To Decode



How To Decode



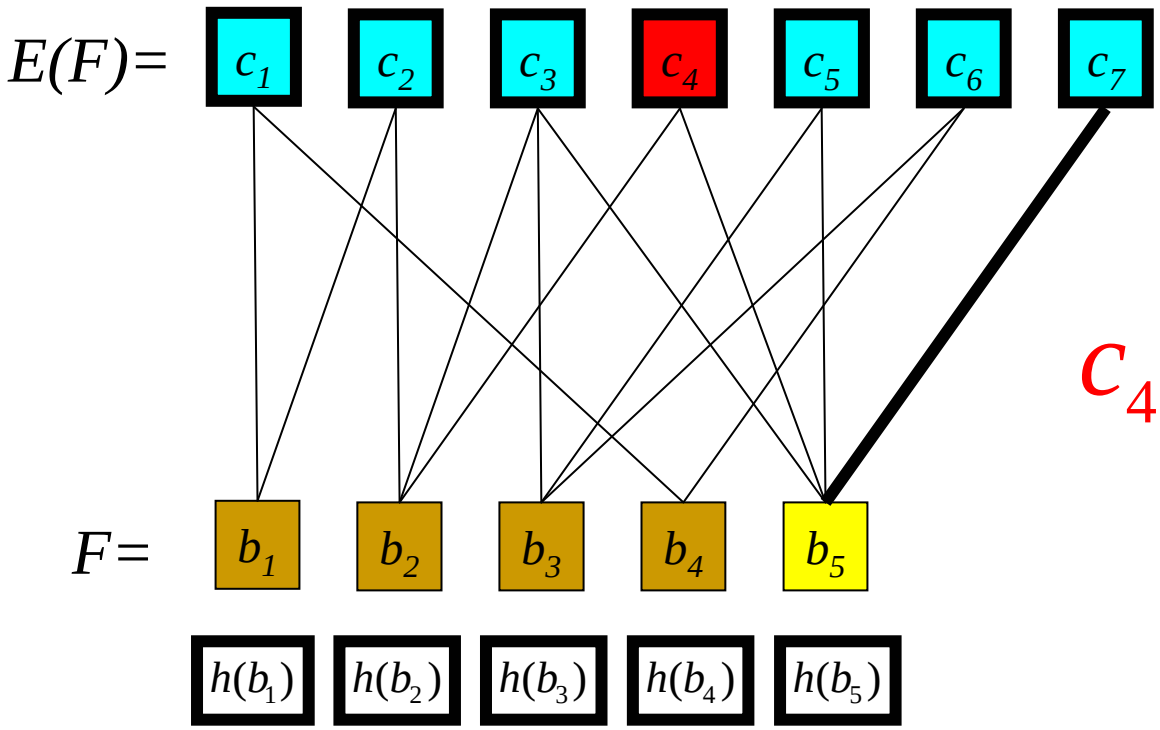
How To Decode



Outline

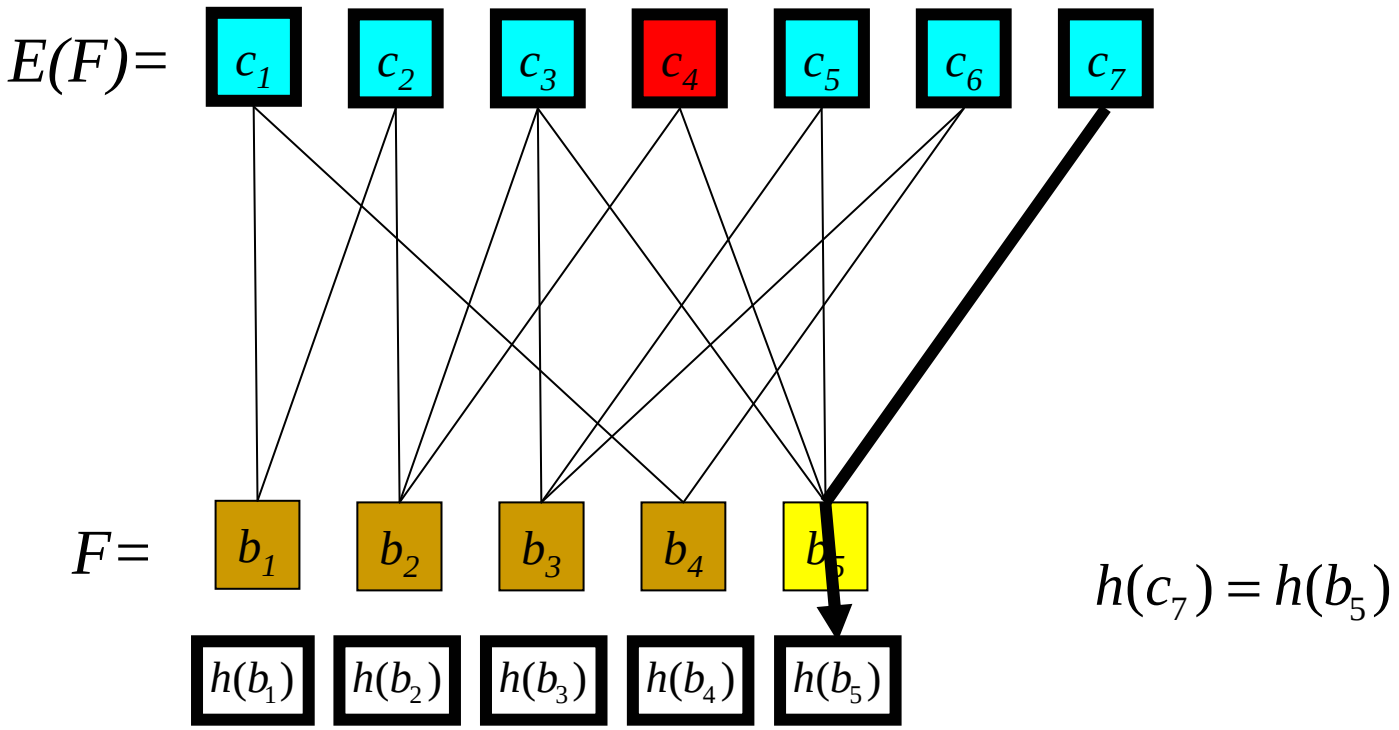
- Introduction
- Review of LT Codes
- **Strawman #1**
  - **Simple Solution To Tell Good Blocks From Bad**
- Strawman #2
- Efficiently Catching Bad Blocks as They Arrive

“Smart Decoder” for LT-Codes

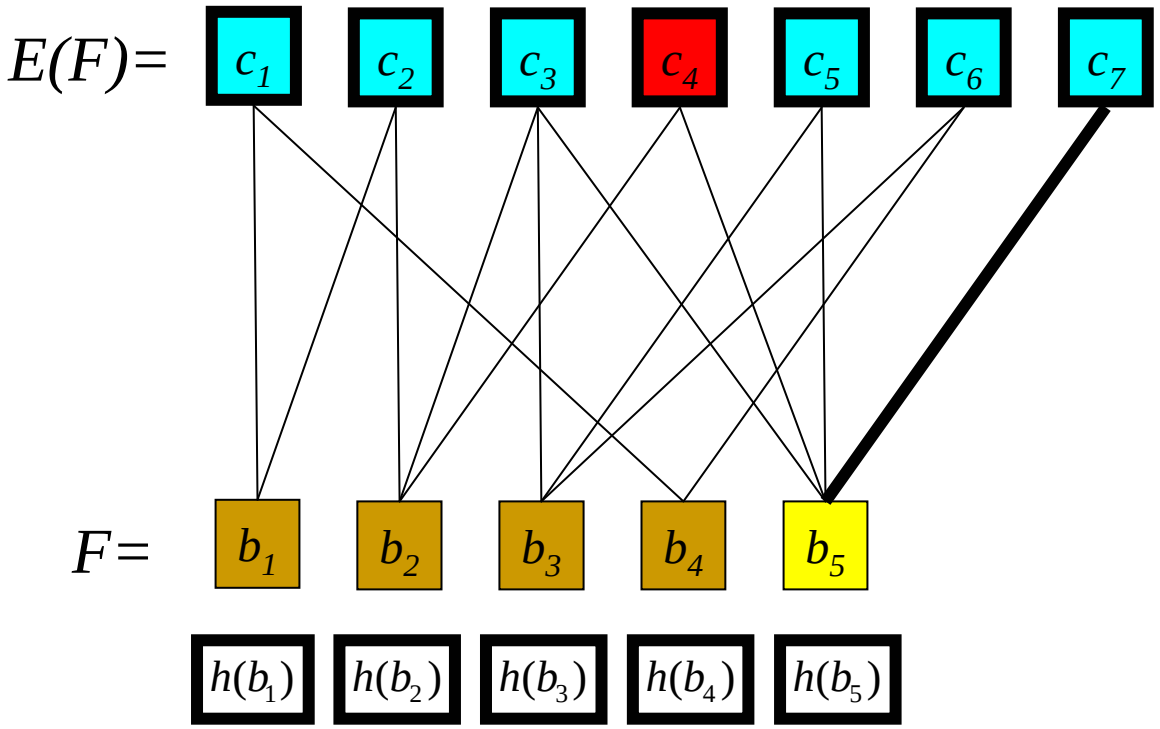


$$c_4 \neq b_2 + b_5$$

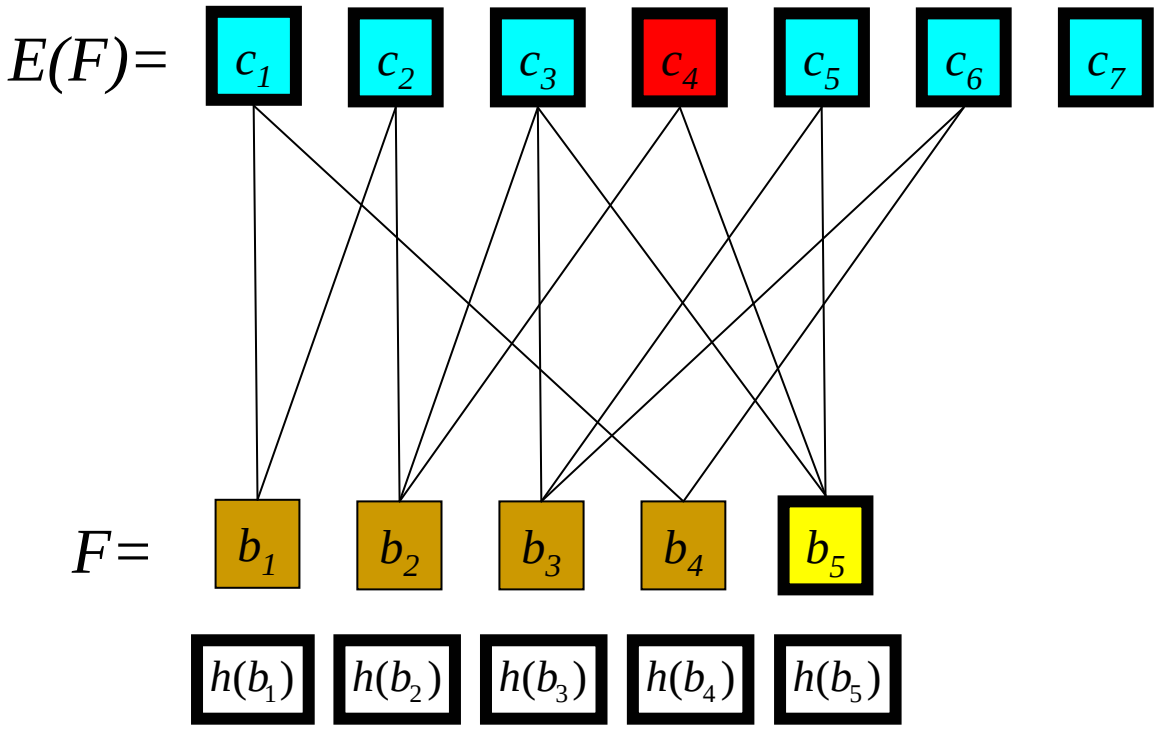
“Smart Decoder” for LT-Codes



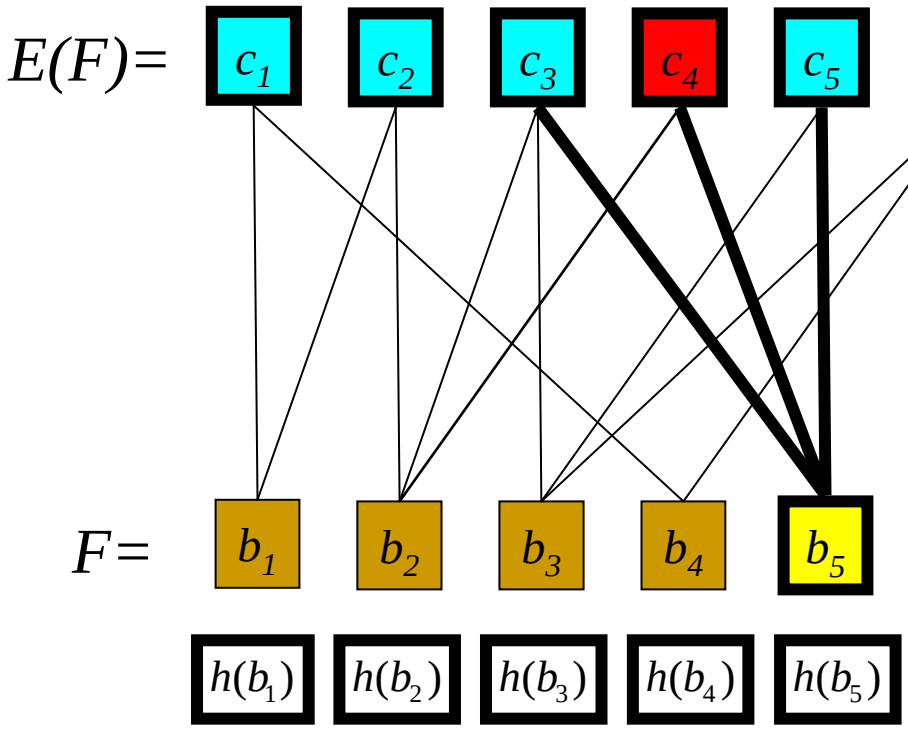
“Smart Decoder” for LT-Codes



“Smart Decoder” for LT-Codes



“Smart Decoder” for LT-Codes

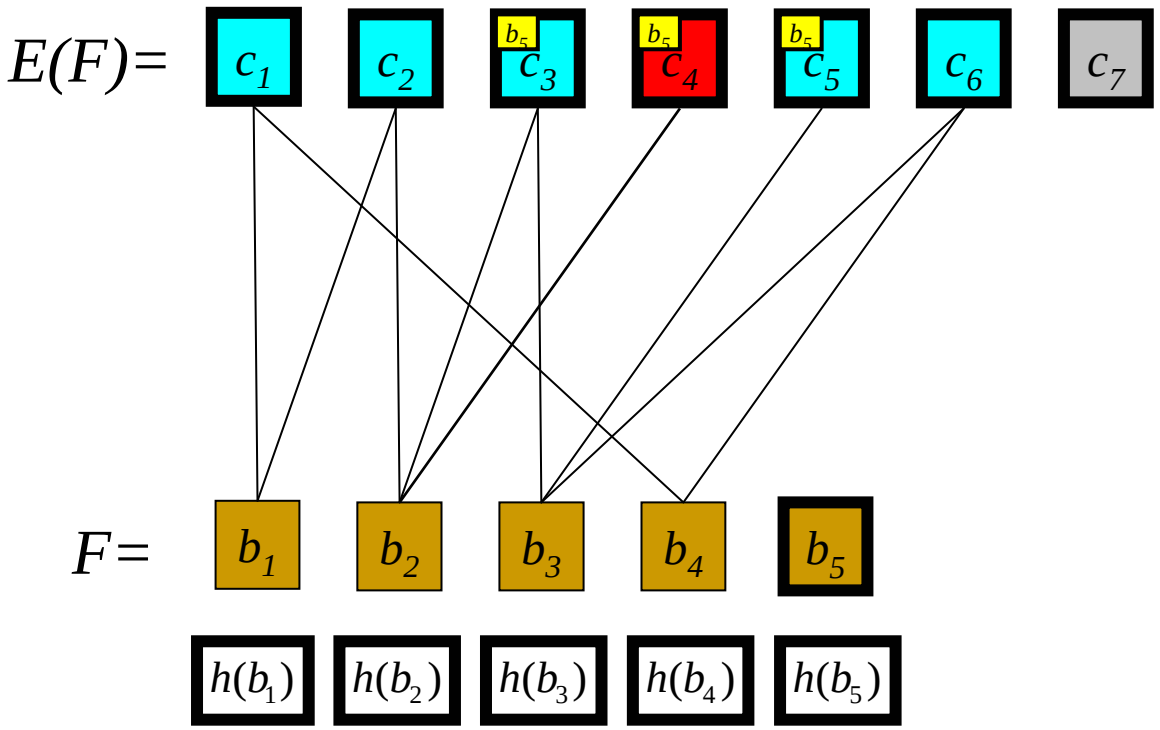


$$c_3 \leftarrow c_3 - b_5$$

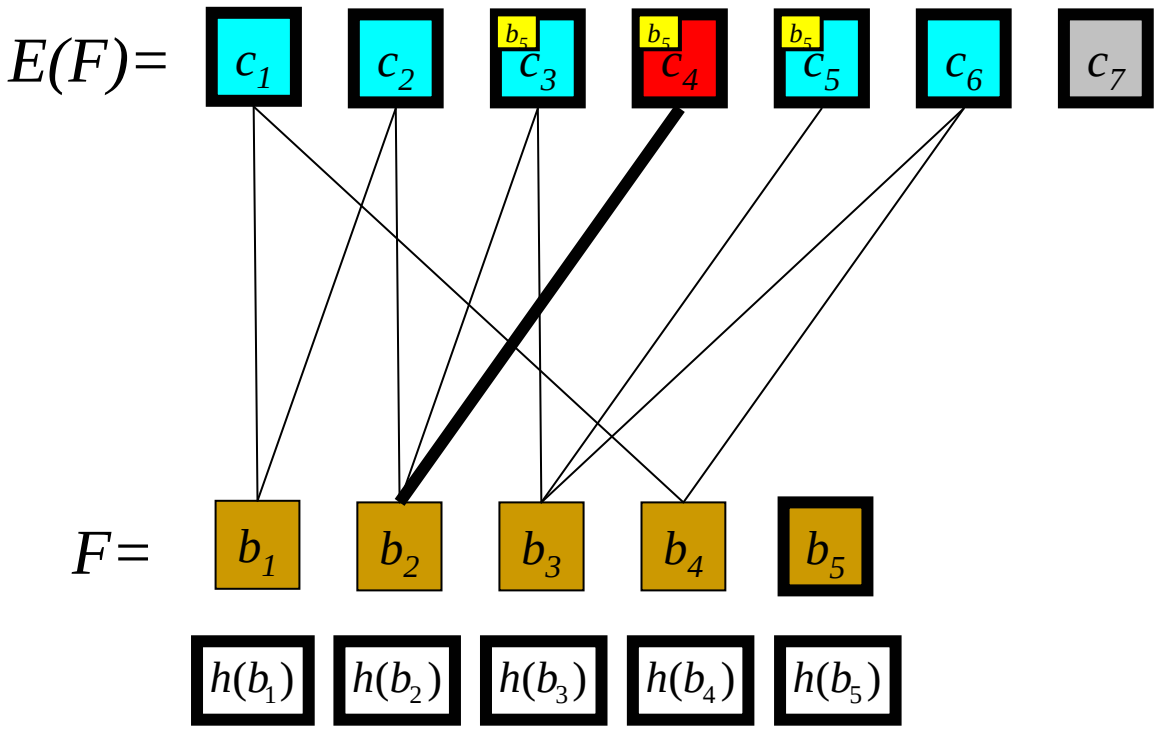
$$c_4 \leftarrow c_4 - b_5$$

$$c_5 \leftarrow c_5 - b_5$$

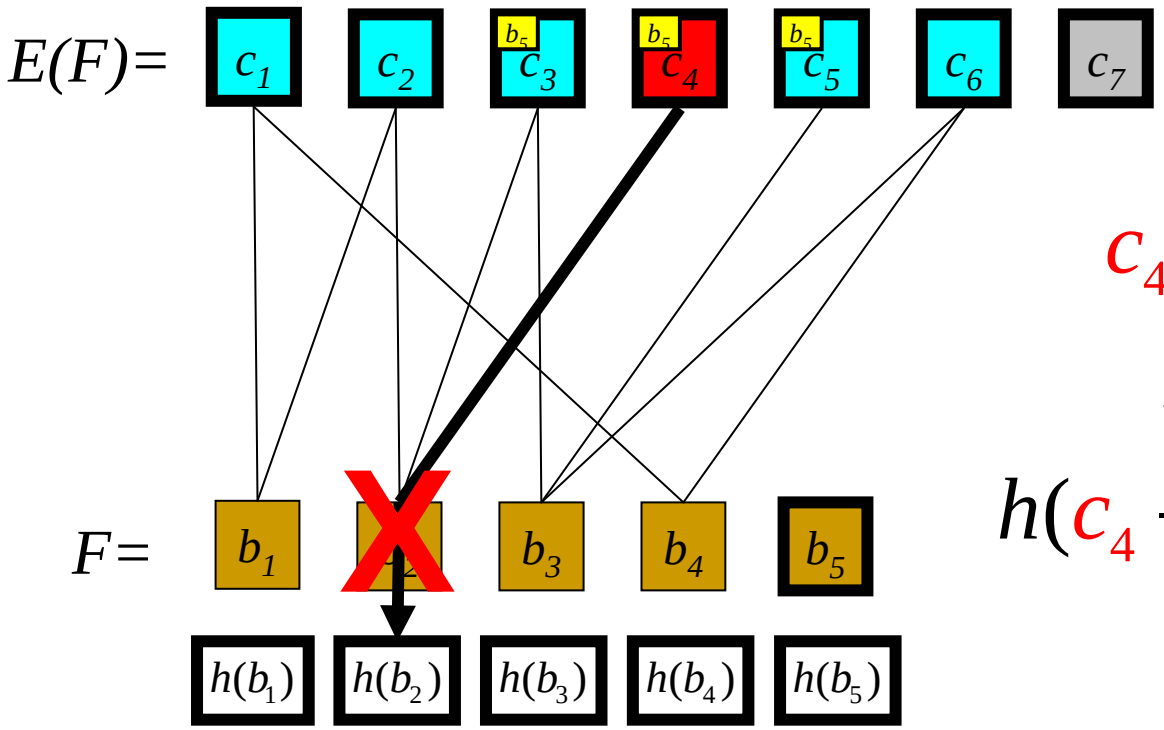
“Smart Decoder” for LT-Codes



“Smart Decoder” for LT-Codes



“Smart Decoder” for LT-Codes

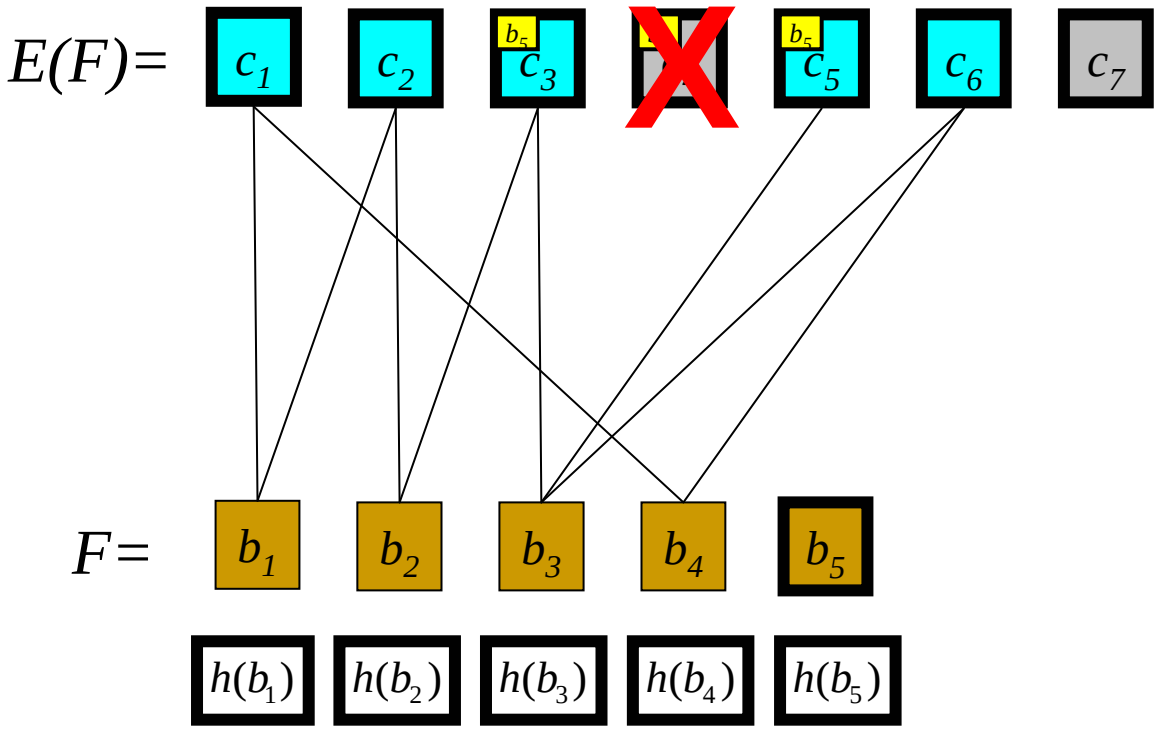


$$c_4 \neq b_2 + b_5$$

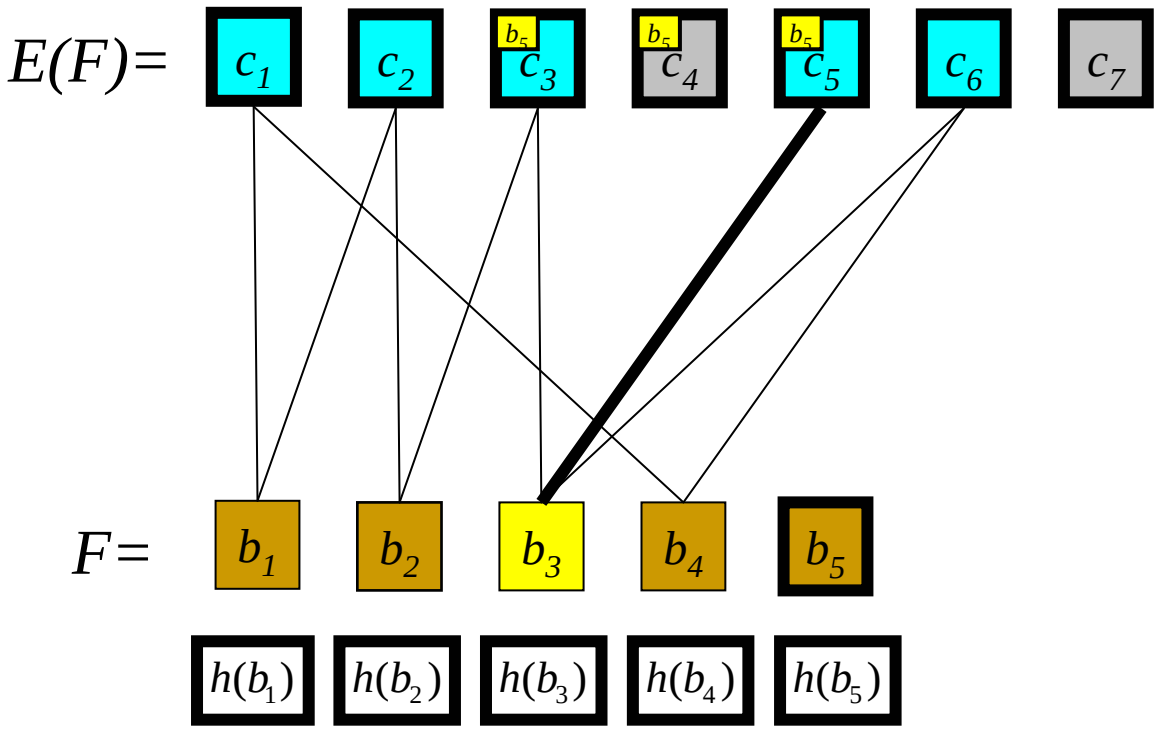
implies

$$h(c_4 - b_5) \neq h(b_2)$$

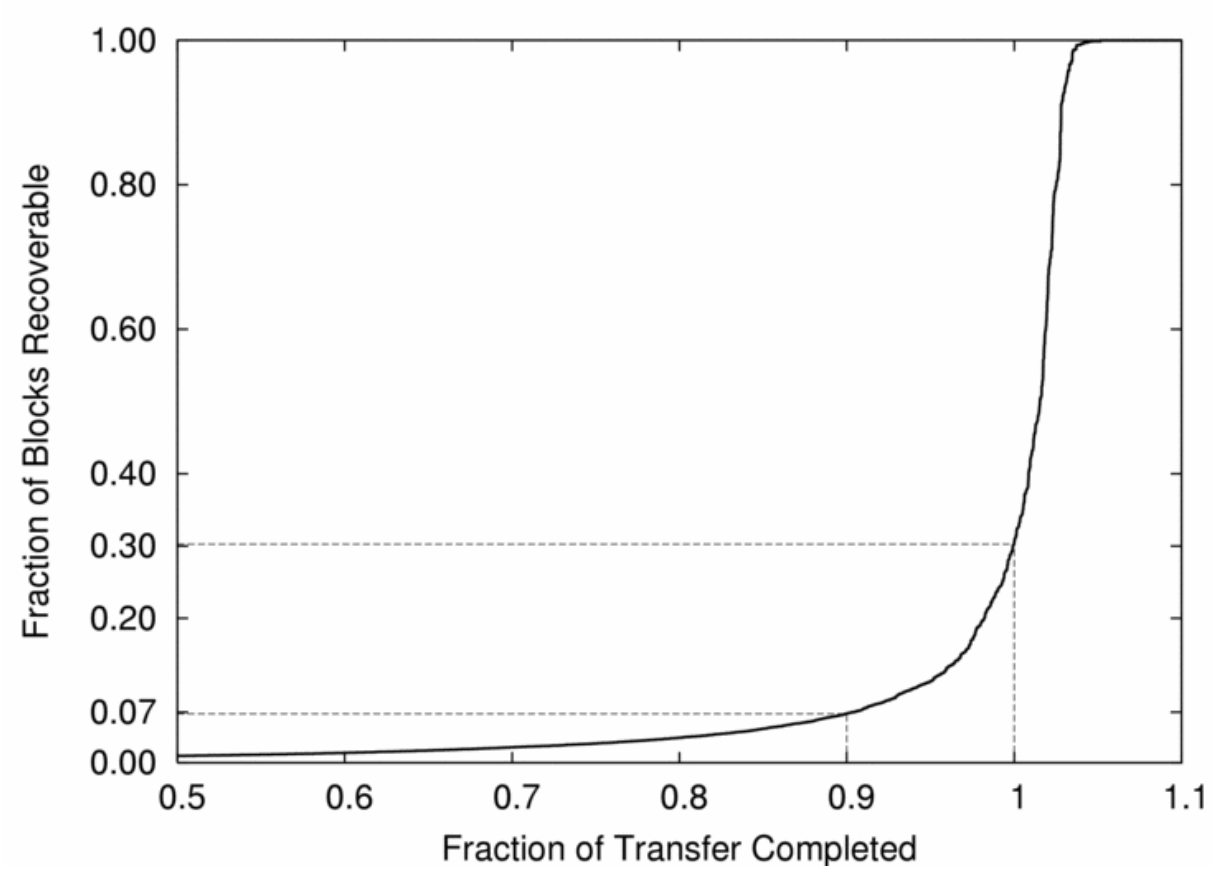
“Smart Decoder” for LT-Codes



“Smart Decoder” for LT-Codes



"Smart Decoder:" Problem

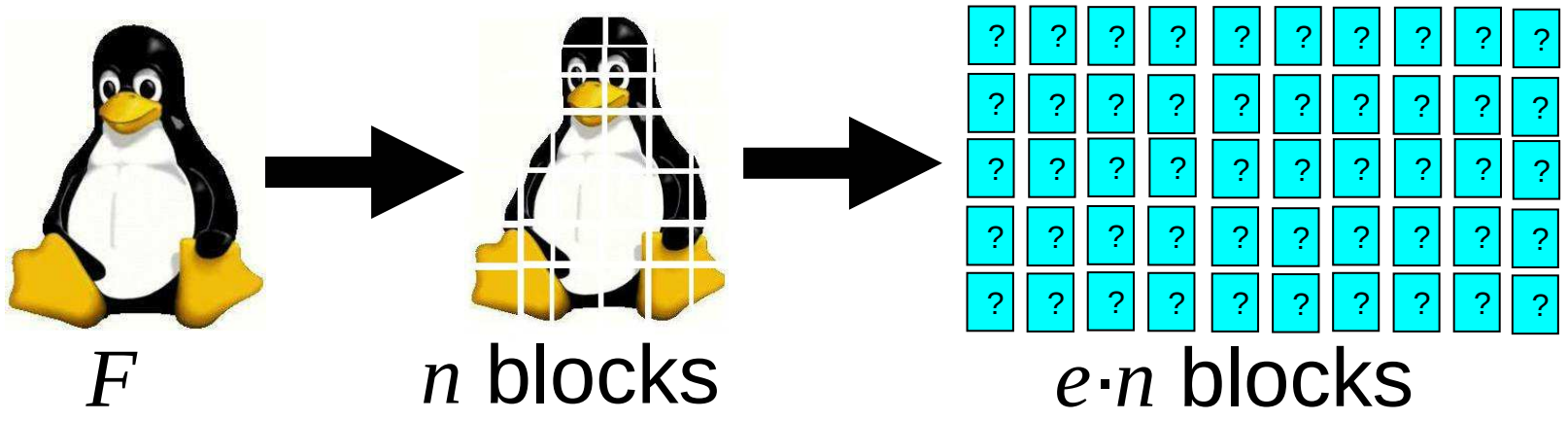


•Data collected from 50 random Online encodings of a 10,000 block file.

Outline

- Introduction
- Review of LT Codes
- Strawman #1
- **Strawman #2**
  - **Hashing/Signing Encoded Blocks**
- Efficiently Catching the Bad as They Arrive

Hashing/Signing Encoded Blocks



- Trusted Publisher (RedHat)
  - Picks e, computes $e \cdot n$ encoded blocks
  - Hashes all encoded blocks
  - Signs the hashes.

Hashing/Signing Encoded Blocks

- Expansion factor e should be big to avoid duplicate blocks.
- e should be small to make crypto overhead acceptable.
- Our analysis shows there's no "sweet spot".

Hashing/Signing Encoded Blocks

- Expansion factor e should be big to avoid duplicate blocks.
- e should be small to make crypto overhead acceptable.
- Our analysis shows there's no "sweet spot".
  - *e.g.,* best case bandwidth requirements: +5%
  - *e.g.*, generating hashes is very expensive as e gets large.

Outline

- Introduction
- Review of LT Codes
- Strawman #1
- Strawman #2
- **Efficiently Catching the Bad as They Arrive**

Best of Both Worlds

- Goal:
  - Crypto overhead of one hash for every block in the input file (Strawman #1)
  - Verify blocks as they arrive (Strawman #2)
- Idea:
  - Distribute hashes of file blocks, and use them to verify *encoded* blocks.
  - Need a better hash function.

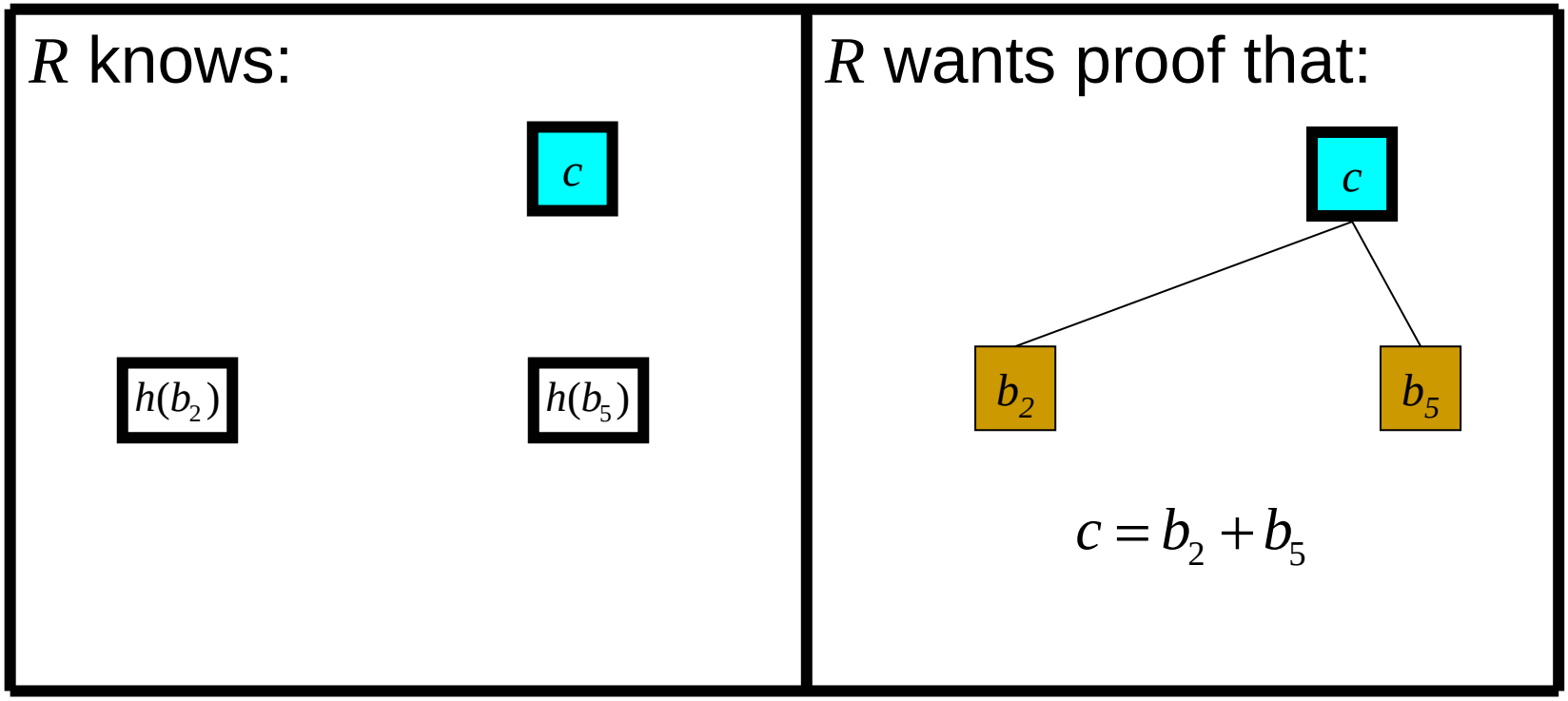
Insight: Homomorphic Hashing

- Assume function h exists such that:

1. is homomorphic: $h(x) \cdot h(z) = h(x + z)$
2. is a CRHF: $h(x) = h(y) \text{ iff } x = y$

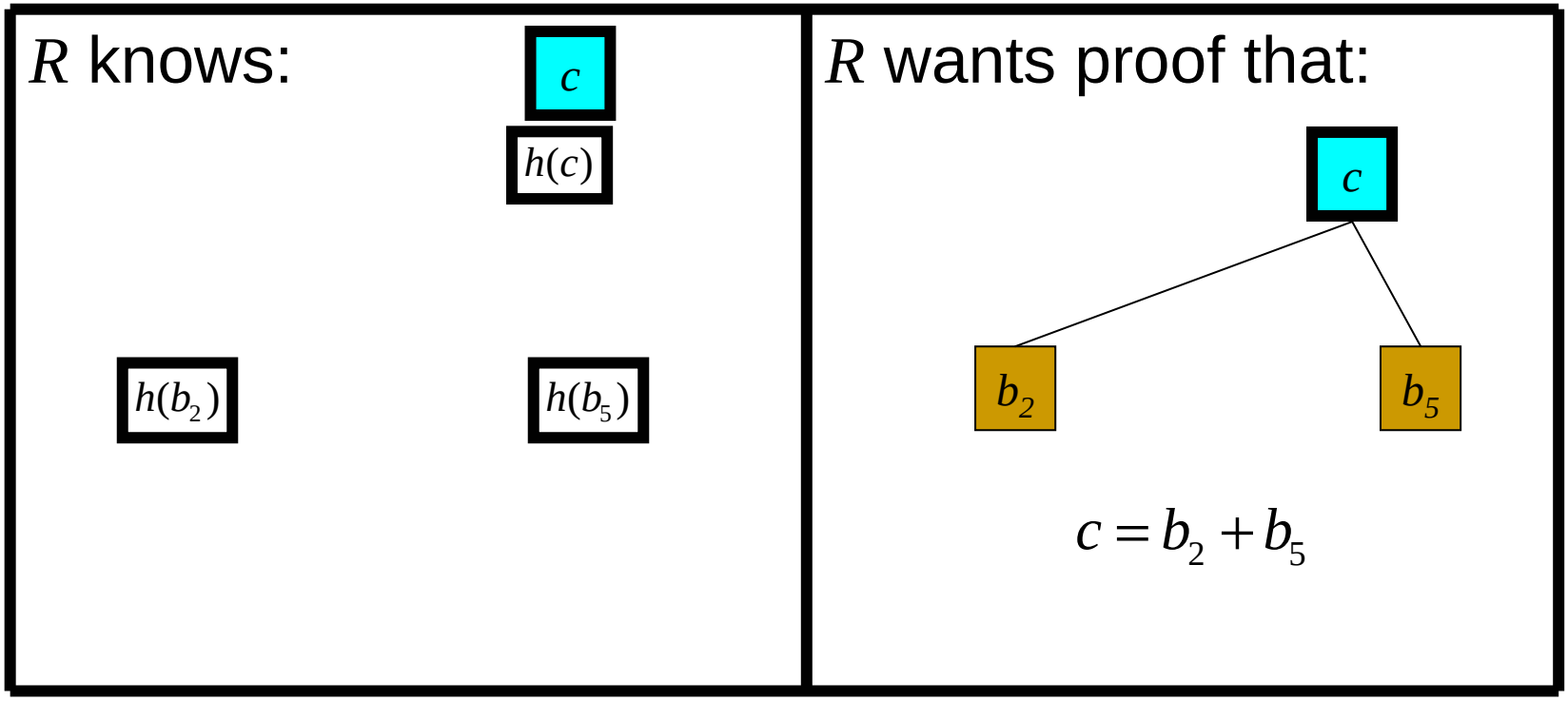
Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$



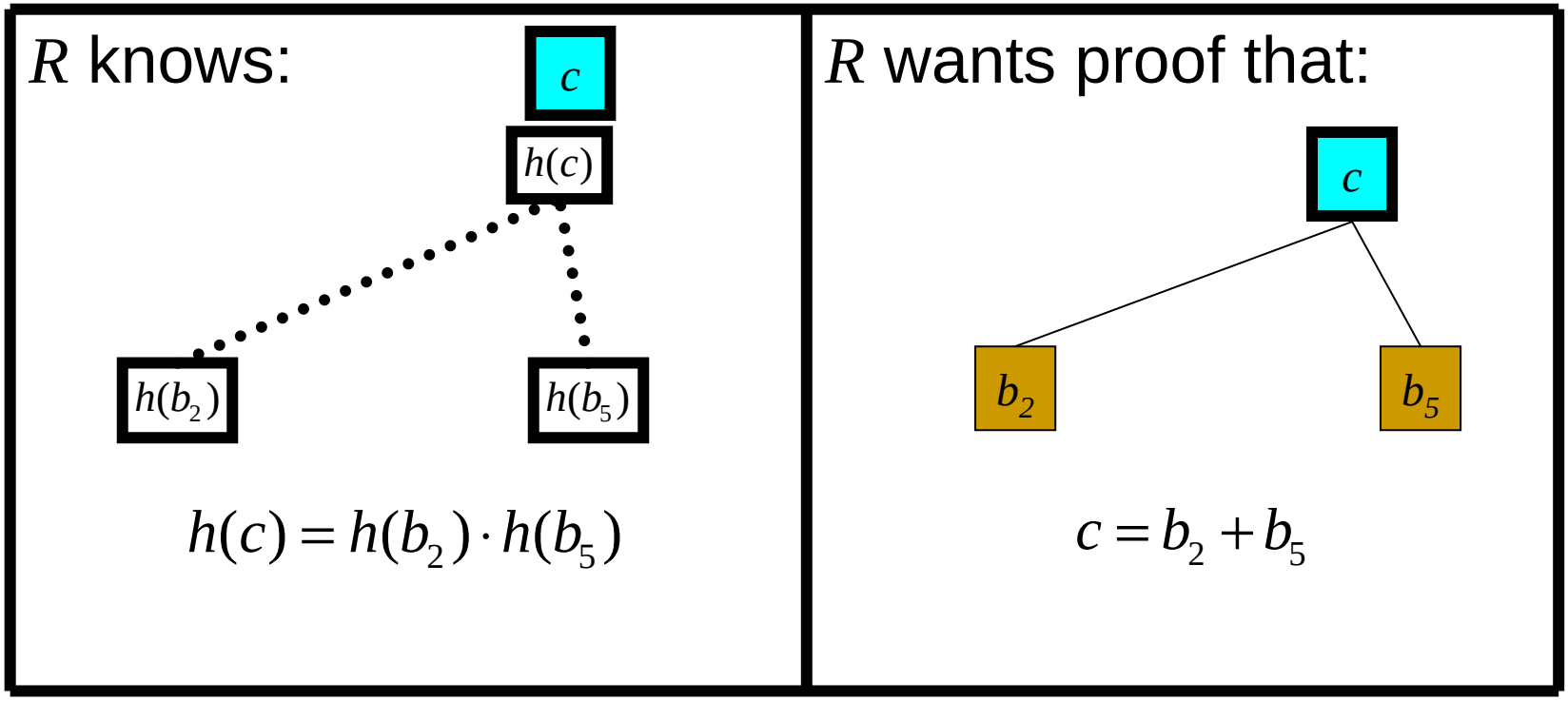
Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$



Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$



Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$

R knows:	R wants proof that:
$h(c) = h(b_2) \cdot h(b_5)$	$c = b_2 + b_5$

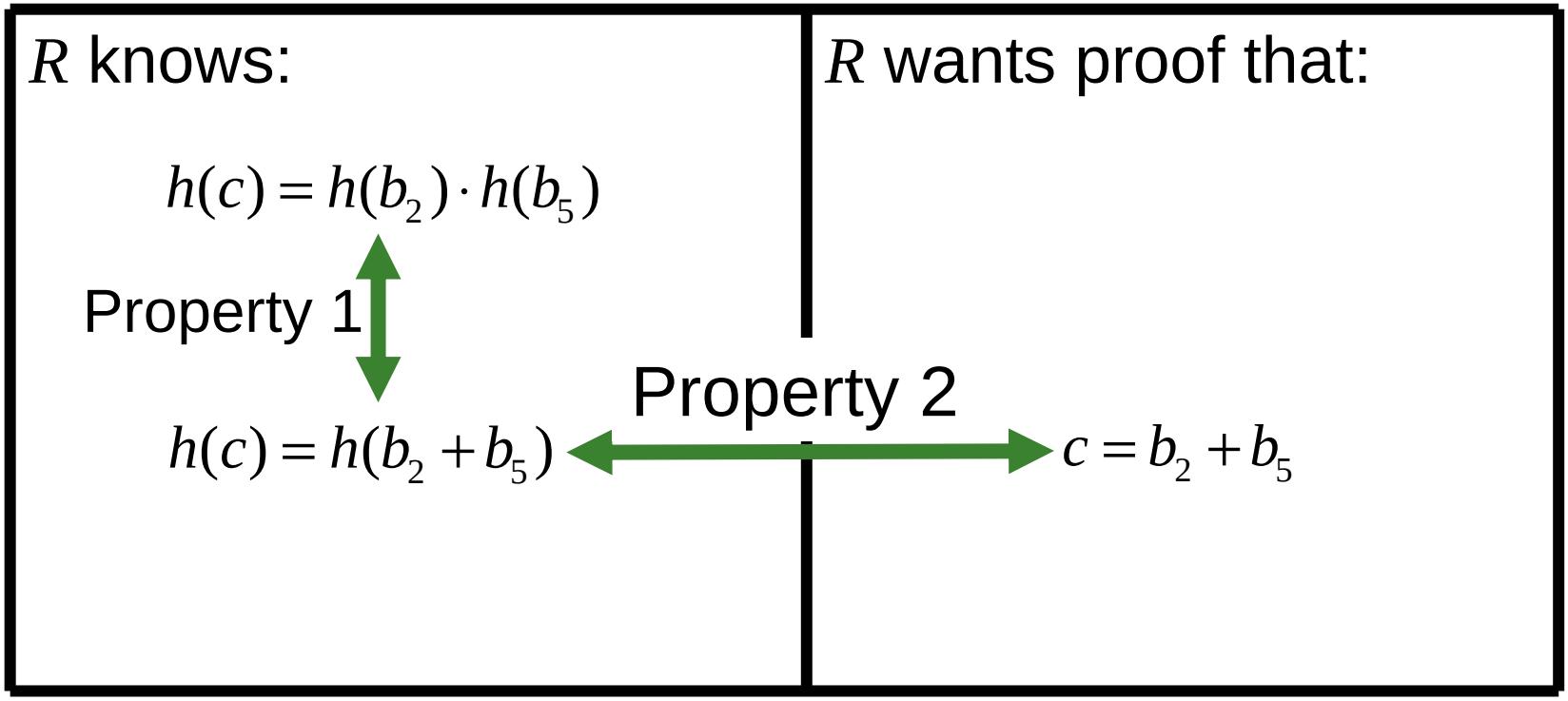
Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$

R knows:	R wants proof that:
$h(c) = h(b_2) \cdot h(b_5)$ Property 1 $h(c) = h(b_2 + b_5)$	$c = b_2 + b_5$

Homomorphic Hashing: Intuition

- R receives the block $\langle c, \{2,5\} \rangle$



Homomorphic Hashing: Protocol

- R receives the block $\langle c, \{2,5\} \rangle$
  - Compute $h(c)$
  - If $h(c) = h(b_2) \cdot h(b_5)$
    - Accept block; mark as valid
  - else
    - Suspect sender of being bad guy, and switch.

Homomorphic Hashing: Protocol

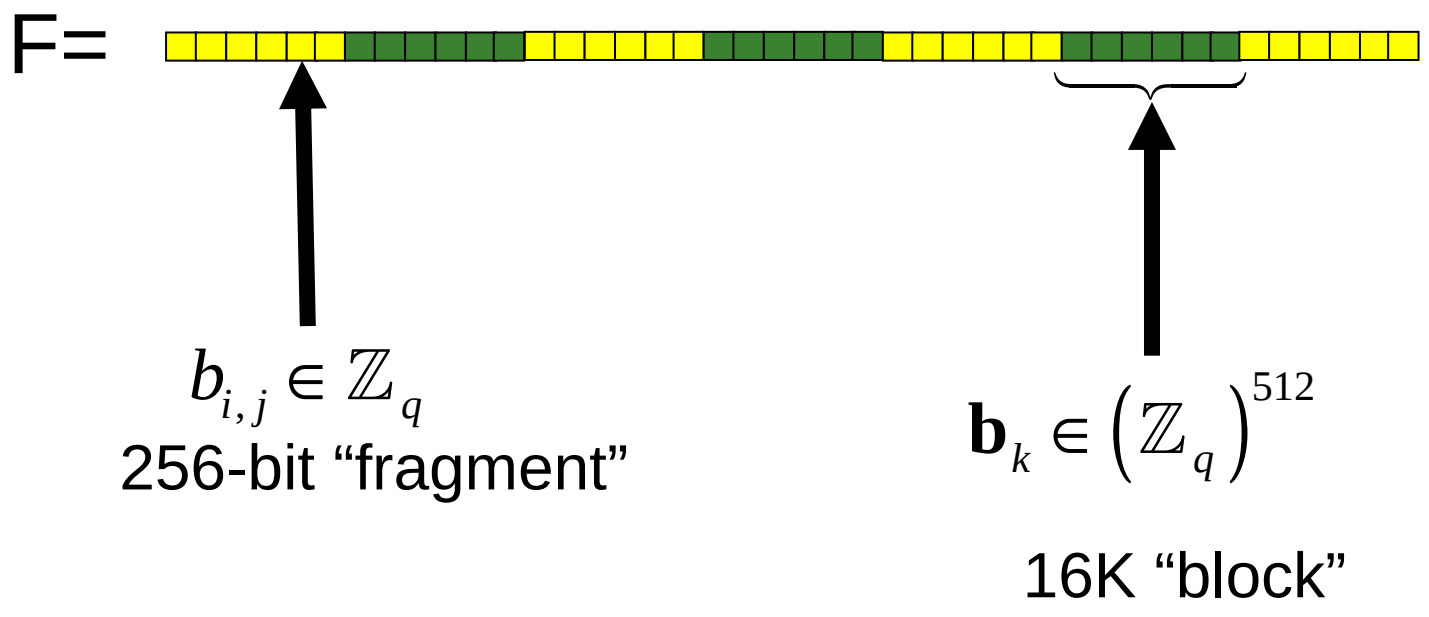
- R receives the block $\langle c, \{2,5\} \rangle$
  - Compute $h(c)$
  - If $h(c) = h(b_2) \cdot h(b_5)$
    - Accept block; mark as valid
  - else
    - Suspect sender of being bad guy, and switch.
- Can such an h possibly exist?

Homomorphic Hashing: Related Work

- DLog-Based CRHF
  - Pederson Commitment [CRYPTO ’91]
  - Chaum et *al*. [CRYPTO ’91]
- One-Way Accumulators
  - Benaloh and de Mare [EUROCRYPT ’93]
  - Barić and Pfitzmann [EUROCRYPT ’93]
- Incremental Hashing
  - Bellare et *al.* [CRYPTO ’94]
- Homomorphic Signatures
  - Micali and Rivest [RSA ’02]
  - Johnson et *al*. [RSA ’02]

Mechanics of Homomorphic Hashing

- Discrete Log Hash
- Pick 1024-bit prime p and 256-bit prime q, q divides $(p\text{-}1)$
- Pick from $\mathbb{Z}_p$ 512 generators of order q: $\mathbf{g} = (g_1, ..., g_{512})$
- Write F as elements in $\mathbb{Z}_q$



How to Encode (example)

Standard LT-Codes:	$c_3 = b_2 \oplus b_3 \oplus b_5$
Homomorphic Scheme:	$\mathbf{c}_3 = \mathbf{b}_2 + \mathbf{b}_3 + \mathbf{b}_5 \pmod{q}$

$$\mathbf{c}_3 = \begin{pmatrix} b_{1,2} \\ \vdots \\ b_{512,2} \end{pmatrix} + \begin{pmatrix} b_{1,3} \\ \vdots \\ b_{512,3} \end{pmatrix} + \begin{pmatrix} b_{1,5} \\ \vdots \\ b_{512,5} \end{pmatrix}$$

How To DLog Hash

$$h(\mathbf{b}_1) = \begin{pmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{512,1} \end{pmatrix} \xrightarrow{\mathbf{g}^x} \begin{pmatrix} g_1^{b_{1,1}} \\ g_2^{b_{2,1}} \\ \vdots \\ g_{512}^{b_{512,1}} \end{pmatrix} \xrightarrow{\boldsymbol{\Pi}} g_1^{b_{1,1}} g_2^{b_{2,1}} \cdots g_{512}^{b_{512,1}}$$

- Hashes are elements in $\mathbb{Z}_p$ (128 bytes big)
- Hash reduces 16K block by a factor of 128

How To DLog Hash

$$h(\mathbf{b}_1) = \begin{pmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{512,1} \end{pmatrix} \xrightarrow{\mathbf{g}^x} \begin{pmatrix} g_1^{b_{1,1}} \\ g_2^{b_{2,1}} \\ \vdots \\ g_{512}^{b_{512,1}} \end{pmatrix} \xrightarrow{\Pi} g_1^{b_{1,1}} g_2^{b_{2,1}} \cdots g_{512}^{b_{512,1}}$$

- Hashes are elements in $\mathbb{Z}_p$ (128 bytes big)
- Hash reduces 16K block by a factor of 128
  - +1% overhead

DLog-Hash: Key Property

- Note that: $h(\mathbf{b}_i) \cdot h(\mathbf{b}_j) = \prod_k g_k^{b_{k,i}} \prod_k g_k^{b_{k,j}}$

$$= \prod_k g_k^{b_{k,i}} g_k^{b_{k,j}}$$

$$= \prod_k g_k^{b_{k,i}+b_{k,j}}$$

$$= h(\mathbf{b}_i + \mathbf{b}_j)$$

DLog-Hash: Key Property

- Note that: $$h(\mathbf{b}_i) \cdot h(\mathbf{b}_j) = \prod_k g_k^{b_{k,i}} \prod_k g_k^{b_{k,j}}$$
$$= \prod_k g_k^{b_{k,i}} g_k^{b_{k,j}}$$
$$= \prod_k g_k^{b_{k,i}+b_{k,j}}$$
$$= h(\mathbf{b}_i + \mathbf{b}_j)$$
- Goal achieved!

“This Seems Really Expensive”

Operation on a 16K Block	Throughput (kB/sec)
DLog Hash	39
Arrival on 1.5Mbps DSL	190
SHA1 Hash	57,600

Key Optimizations

- Hash Generation
  - Each publisher picks her own parameters,
  - compute $h(\mathbf{b}_i)$ with 1 exponentiation (not 512)
- Hash Verification
  - Receiver verifies hashes probabilistically and in batches.
    - Bellare *et al.* [EUROCRYPT '98]

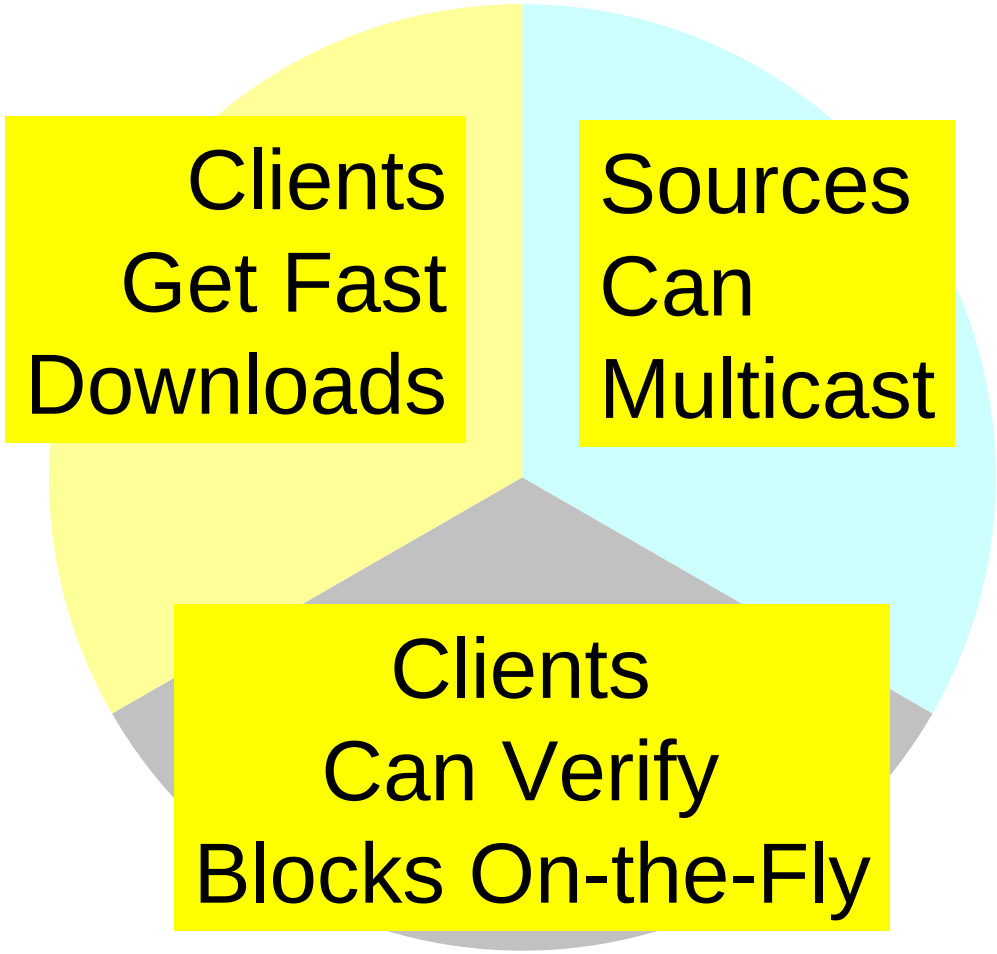
Much Better

Operation on a 16K Block	Throughput (MB/sec)
Naïve DLog Hash	0.038
Per-publisher Generation	11.210
Batch Verification	7.620
Arrival on 1.5 Mbps DSL	0.186
SHA1 Hash	56.250

Homomorphic Hashing: Key Points

- Key Algebraic Feature
  - Homomorphism: Receivers can compose hashes the way encoders sum file blocks.
  - Can check encoded blocks as they arrive.
- Fast
  - Can be optimized to achieve good generation and verification throughputs
- Provably Secure
  - As hard as discrete log (SHA1/MD5 not needed)

Conclusion


Clients
Get Fast
Downloads
Sources
Can
Multicast
Clients
Can Verify
Blocks On-the-Fly

Thank you.

Now accepting questions.