The rsync algorithm

https://rsync.samba.org/tech_report/tech_report.html

An easy problem

- I have two files A and B. I want to make B equals to A

- What is the cost?
  - CPU
  - Data moved (reads, writes)

The problem of rsync

- A is stored in computer **alpha** and B in computer **beta**
- The network link can be slow (at least it is much slower than CPU)

- **How can I save bandwidth?**

A naïve approach

- **Beta** compute a hash of the file B and send it to **alpha**
- **Alpha** compute the hash of A and send back to **beta** either the hash (if the two hash are the same) or the content of A if they differ
- **Beta** check if the message is the hash or has to update B

- What is the cost?
- What is the hash function?

Cryptographic hash

1. Deterministic
2. Quick to compute
3. Infeasible to generate a message from the hash
4. A small change in the message should drastically change the hash
5. It is infeasible to find collisions

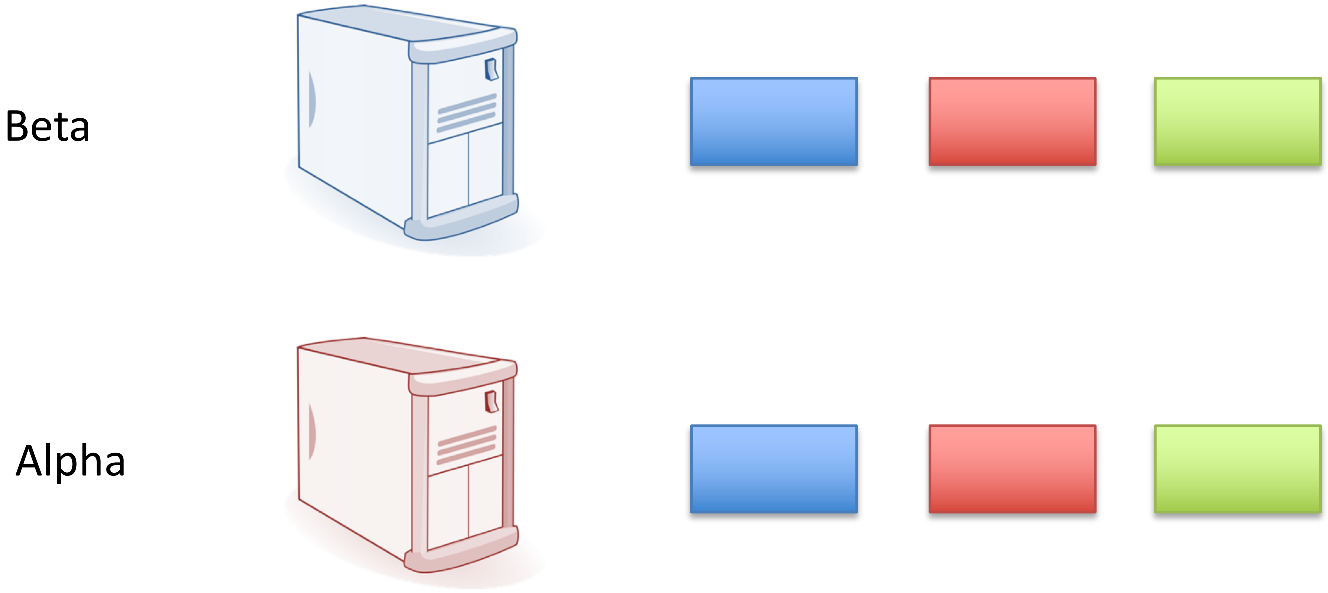
Cryptographic hash

1. Deterministic
2. Quick to compute
3. Infeasible to generate a message from the hash
4. A small change in the message should drastically change the hash
5. It is infeasible to find collisions

Can I do better?

- Can I save bandwidth when A and B are similar?

Solution 1 - bucketing

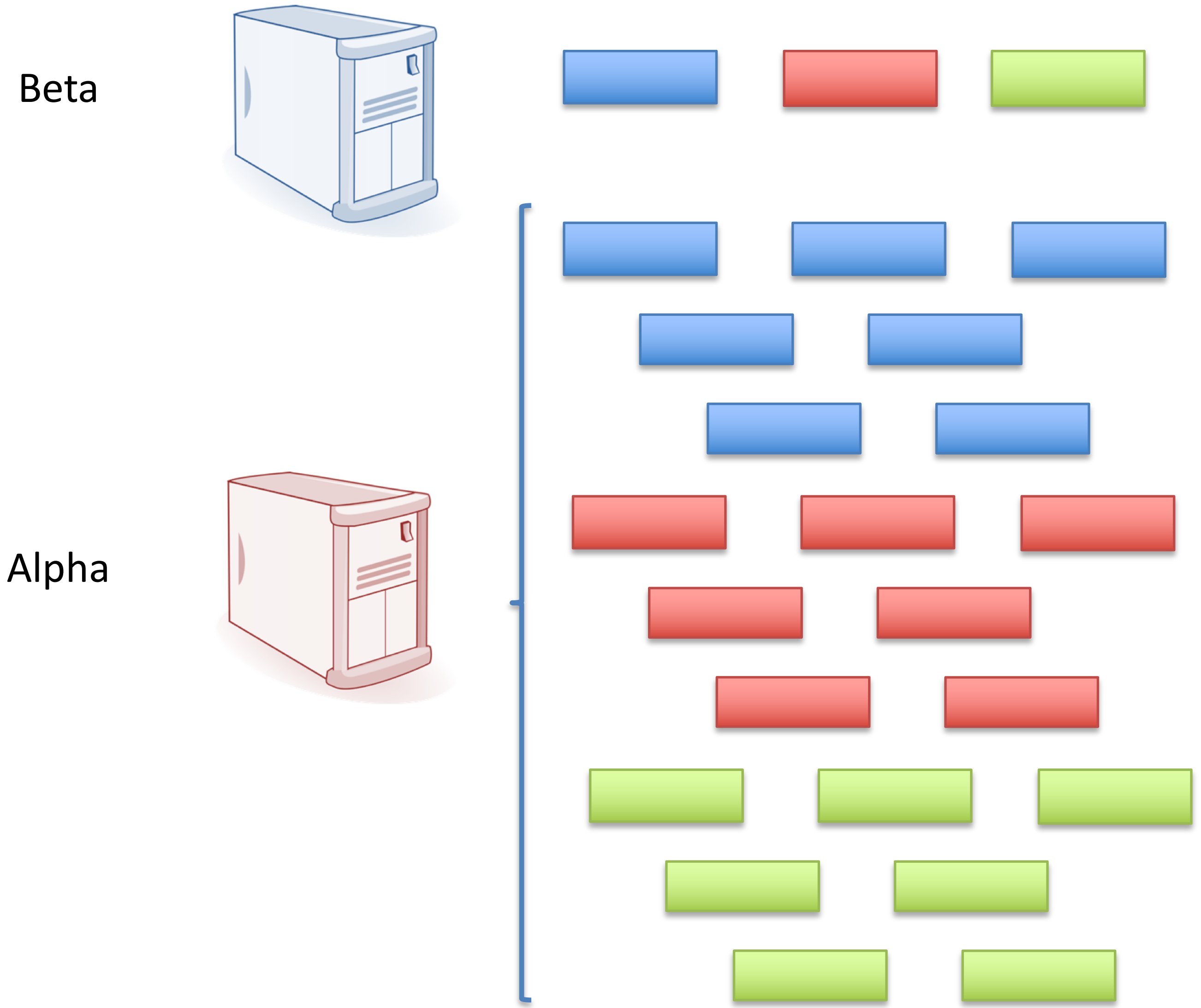


Beta

Alpha

- Weakness?
- Can I do better?

Solution 2 - rolling



Can I do better?

- Intense use of cpu in **alpha**

Solution 3 – rolling hashing

- A two hashing strategy

$$Document\ = X_{1,}X_2 \dots X_n$$

$$a(k,l) = \left(\sum_{i=k}^{l} X_i\right) mod\ M$$

$$b(k,l) = \left(\sum_{i=k}^{l} (l - i + 1) X_i\right) mod\ M$$

$$s(k,l) = a(k,l) + 2^{16}\, b(k,l)$$

Solution 3 – rolling hashing

- A convenient way to derive next hash

$$a(k+1, l+1) = (\, a(k, l) + \, X_{l+1} - \, X_k)\ mod\ M$$

$$\begin{aligned} b(k&+1, l+1) \\ &= \, (b(k, l) - (l - k + 1)X_k \\ &+ \, a(k+1, l+1))\ mod\ M \end{aligned}$$

- Is it M=2^{16} a good idea?
- Collisions?

Update an example (1)

- Sequence: ABCDE
- Window size: 4
- Get rid of the modulo for simplicity

- $a(1,4) = A + B + C + D$
- $a(2, 5) = a(1,4) - A + E =$

  $= A + B + C + D – A + E =$

  $= B + C + D + E$

Update an example (2)

- Sequence: ABCDE, window size = 4

- b(1,4) = 4A + 3B + 2C + 1D
- b(2,5) = b(1,4) - 4A + a(2,5) =

  = 4A + 3B + 2C + 1D - 4A + a(2,5) =

  = 3B + 2C + 1D + a(2,5) =

  = 3B + 2C + 1D + B + C + D + E =

  = 4B + 3C + 2D + E

Can I do better?

- Collision probability high enough to ensure equality of blocks
- One scan of the file A in **alpha** for each block of B in **beta**

Solution 4 - rsync

- Use two hash functions
- The rolling hashing for each possible offset
- A stronger 128bit hash in case a collision is detected
  - Rsync uses MD4

Solution 4 - rsync

- Use two hash functions
- The rolling hashing for each possible offset
- A stronger 128bit hash in case a collision is detected
  - Rsync uses MD4

- How to generate collisions in MD4
  - https://eprint.iacr.org/2005/151.pdf

Checksum searching

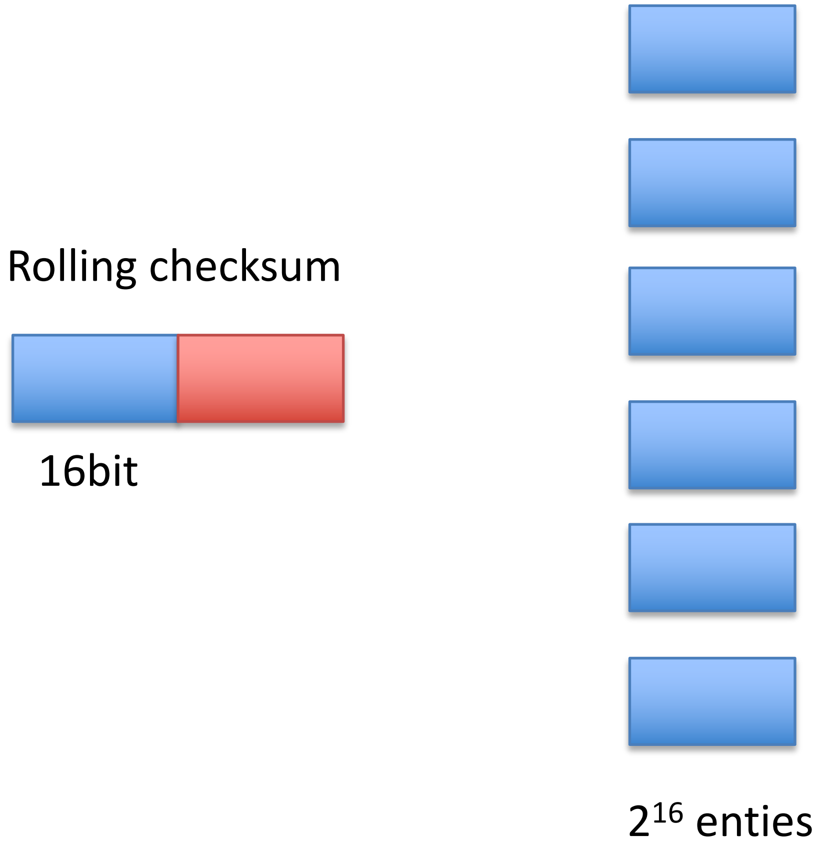
- **Beta** send several checksums
- For each test **alpha** performs a search on these checksums

- Is linear scanning an option?

Checksum searching: possible solutions

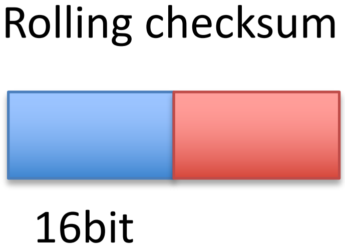
- Binary search
  - Preprocessing requires sorting O(n lg n)
  - Searching requires O (lg n0
- Bloom filters
  - Constant time insert and query, but can have false positives
- Perfect hashing
  - Preprocessing space/time tradeoff
  - Constant time searching

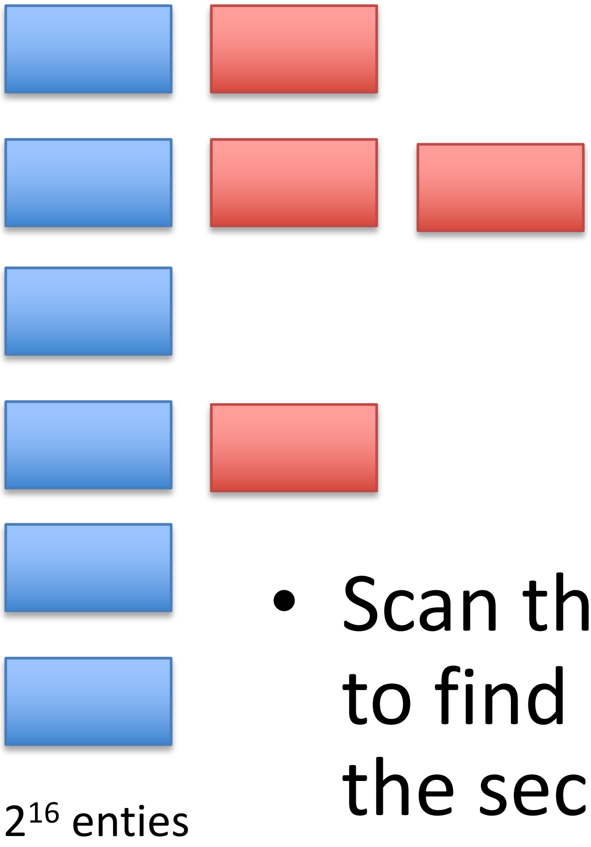
The rsync three way test



- Search for a match in the table
  - If nul the block is not found

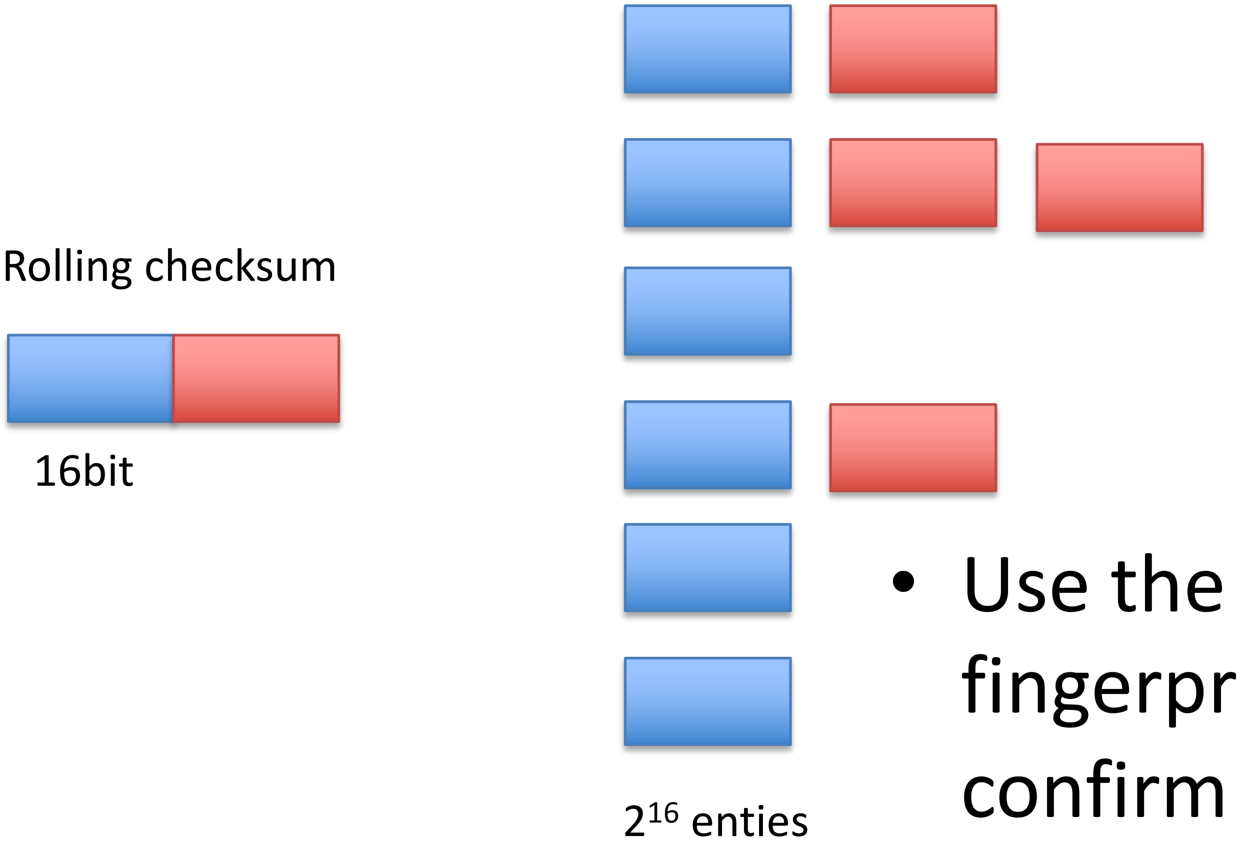
The rsync three way test





- Scan the sorted list to find a match with the second half of the checksum

The rsync three way test



- Use the strong fingerprint to confirm the match

The rsync three way test

- What happens if two blocks in B have the same fingerprint?
- How the list of blocks can be organized?
- Is it possible to copy a corrupted file?

Things you may want to try and discuss next week

- Test binary search or perfect hashing
- Test the impact of the length of the block
- Small vs huge files