The rsync algorithm

Filippo Geraci, IIT CNR Pisa

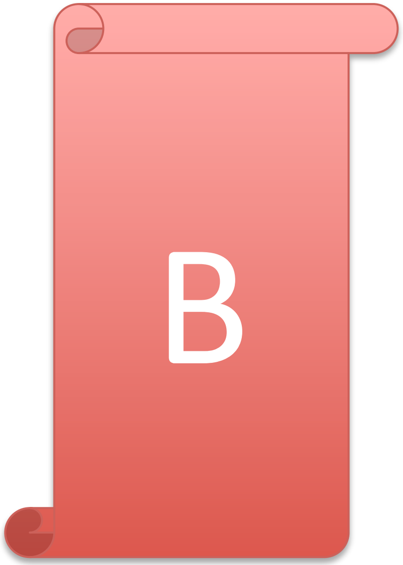
March 2020

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

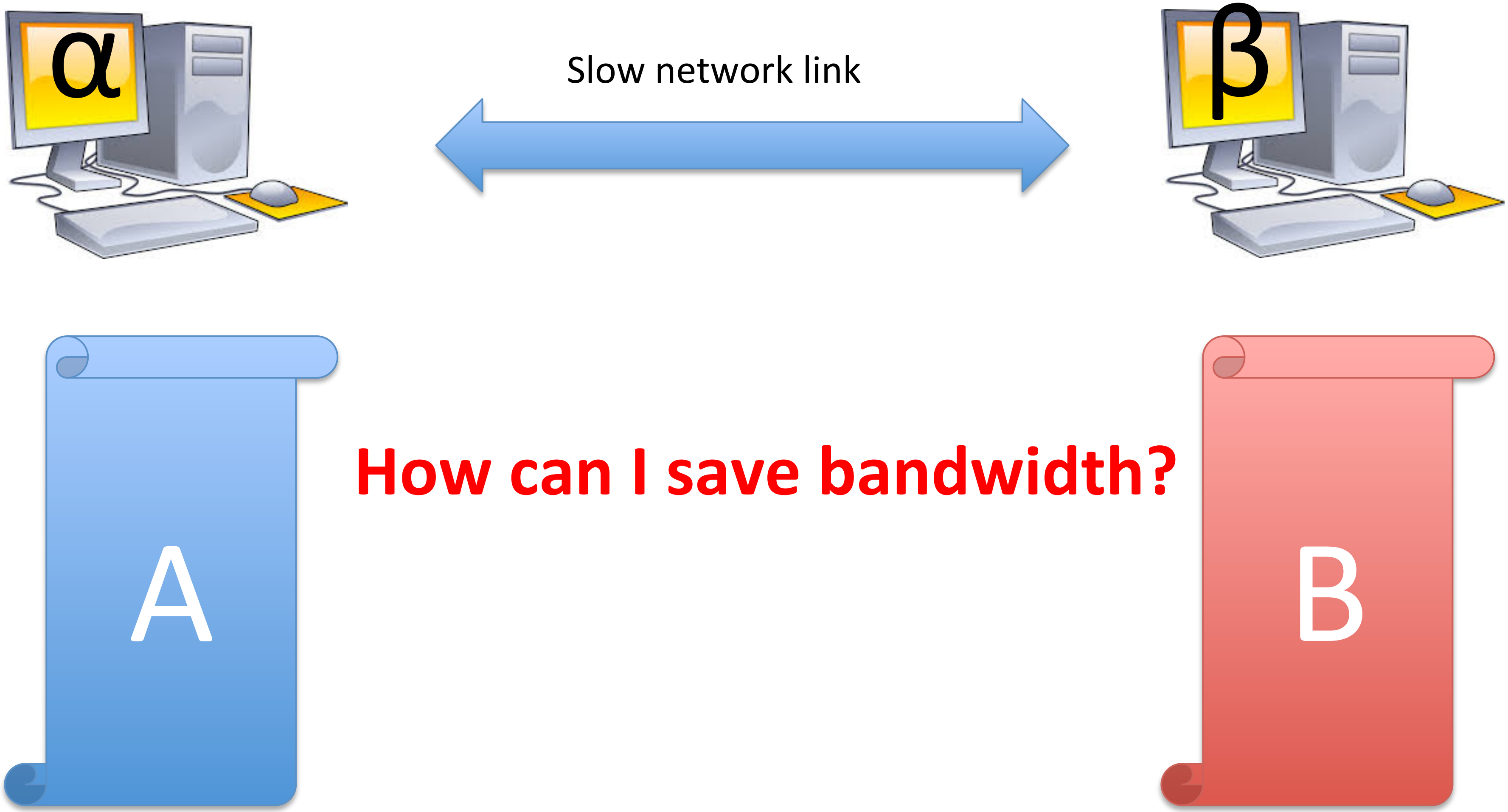
An easy problem

- I have two files A and B. I want to update B to be the same as A

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



The problem of rsync



α

Slow network link

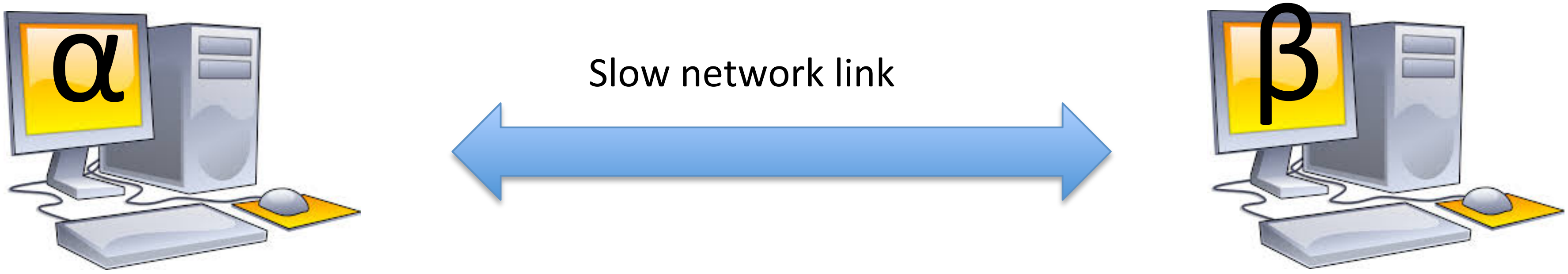
β

A

How can I save bandwidth?

B

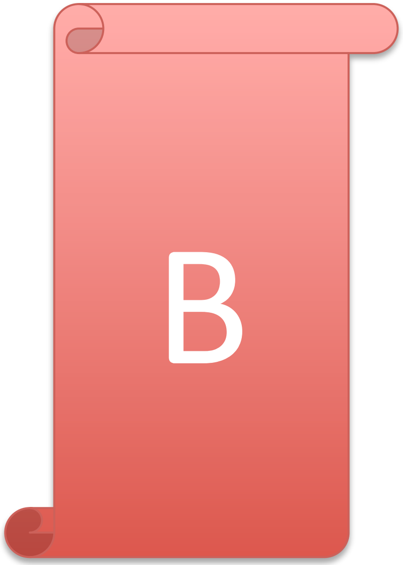
The problem of rsync



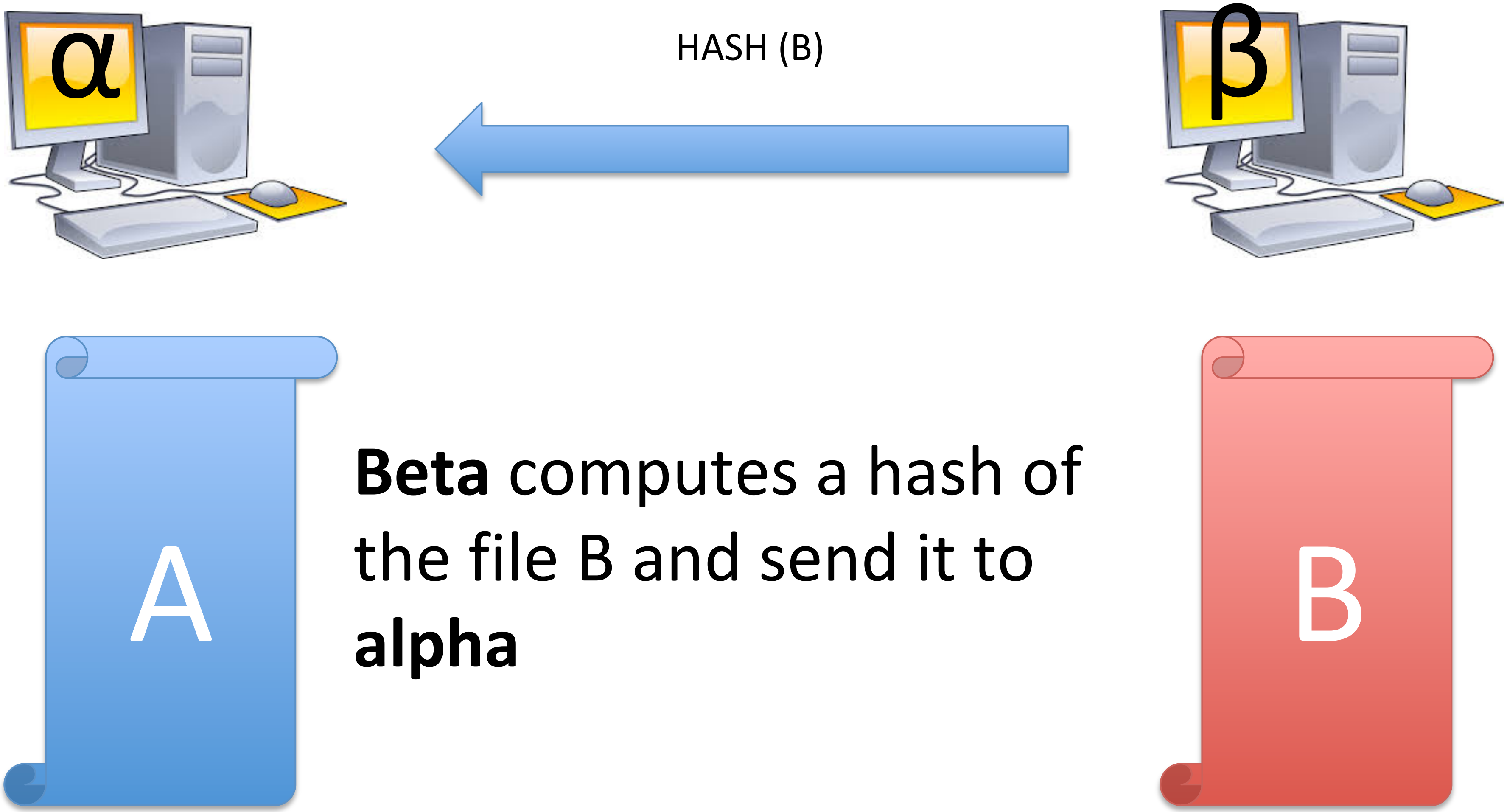
How can I save bandwidth?

Compression

- Typically gain a factor of 2 to 4

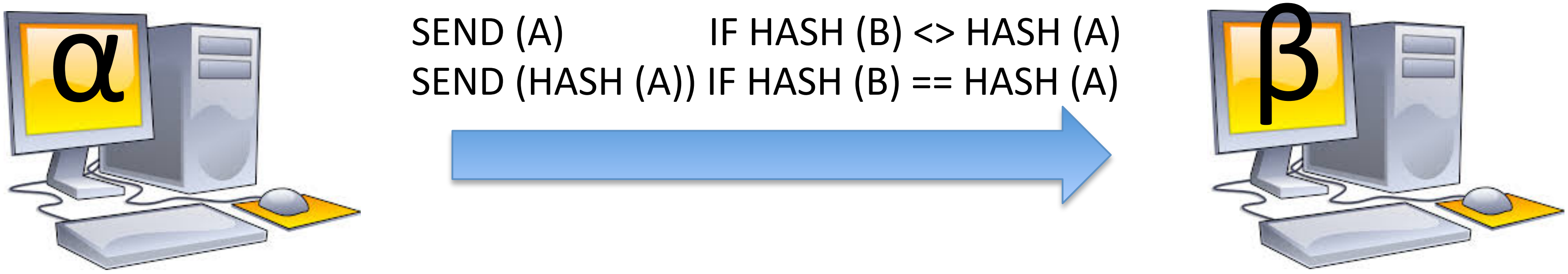


A naïve approach



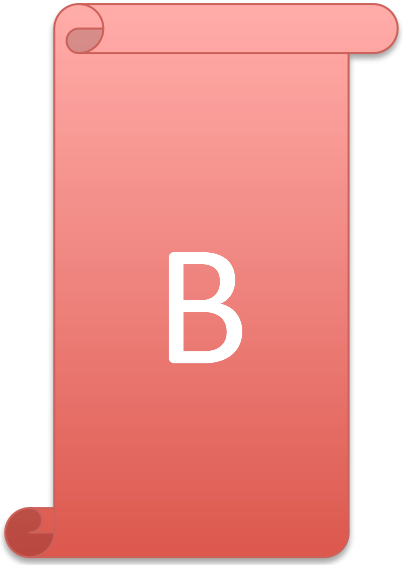
Beta computes a hash of the file B and send it to **alpha**

A naïve approach



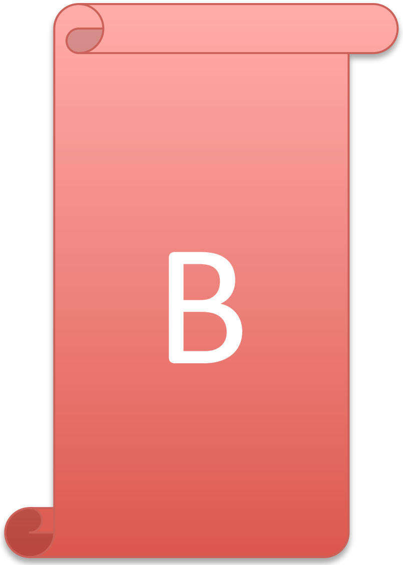
SEND (A) IF HASH (B) <> HASH (A)
SEND (HASH (A)) IF HASH (B) == HASH (A)

Alpha computes 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



A naïve approach

Beta checks if the message is the hash or has to update B



A naïve approach

- **Beta** computes a hash of the file B and send it to **alpha**
- **Alpha** computes 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** checks 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

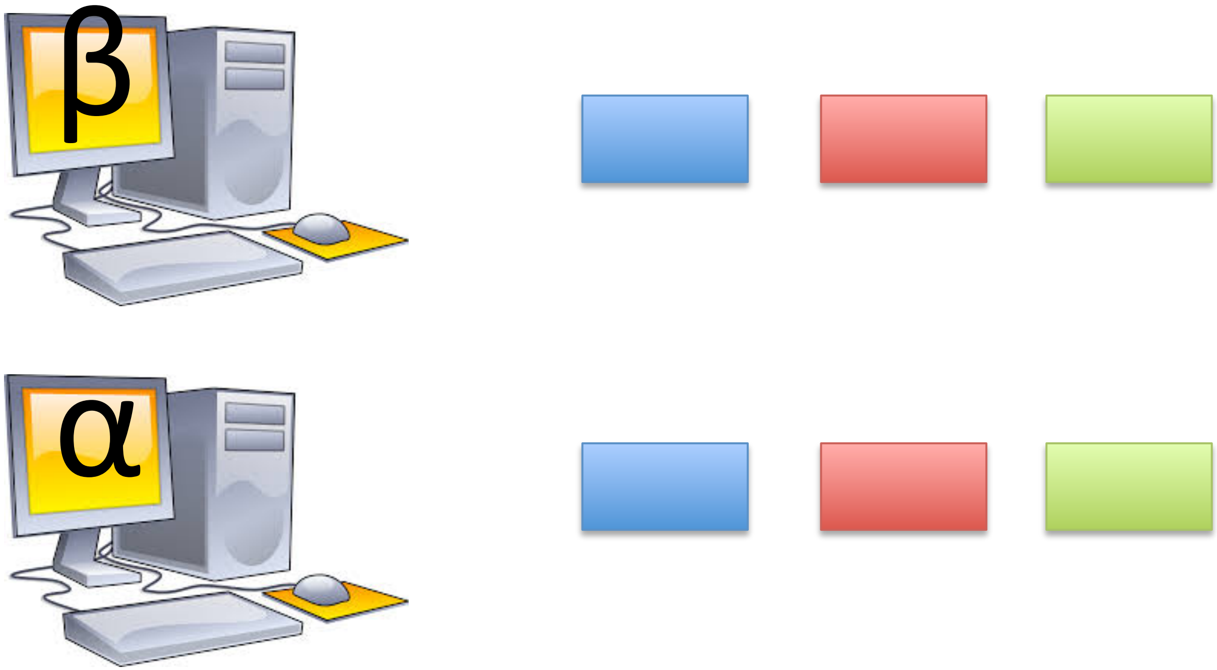
Cryptographic hash

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?

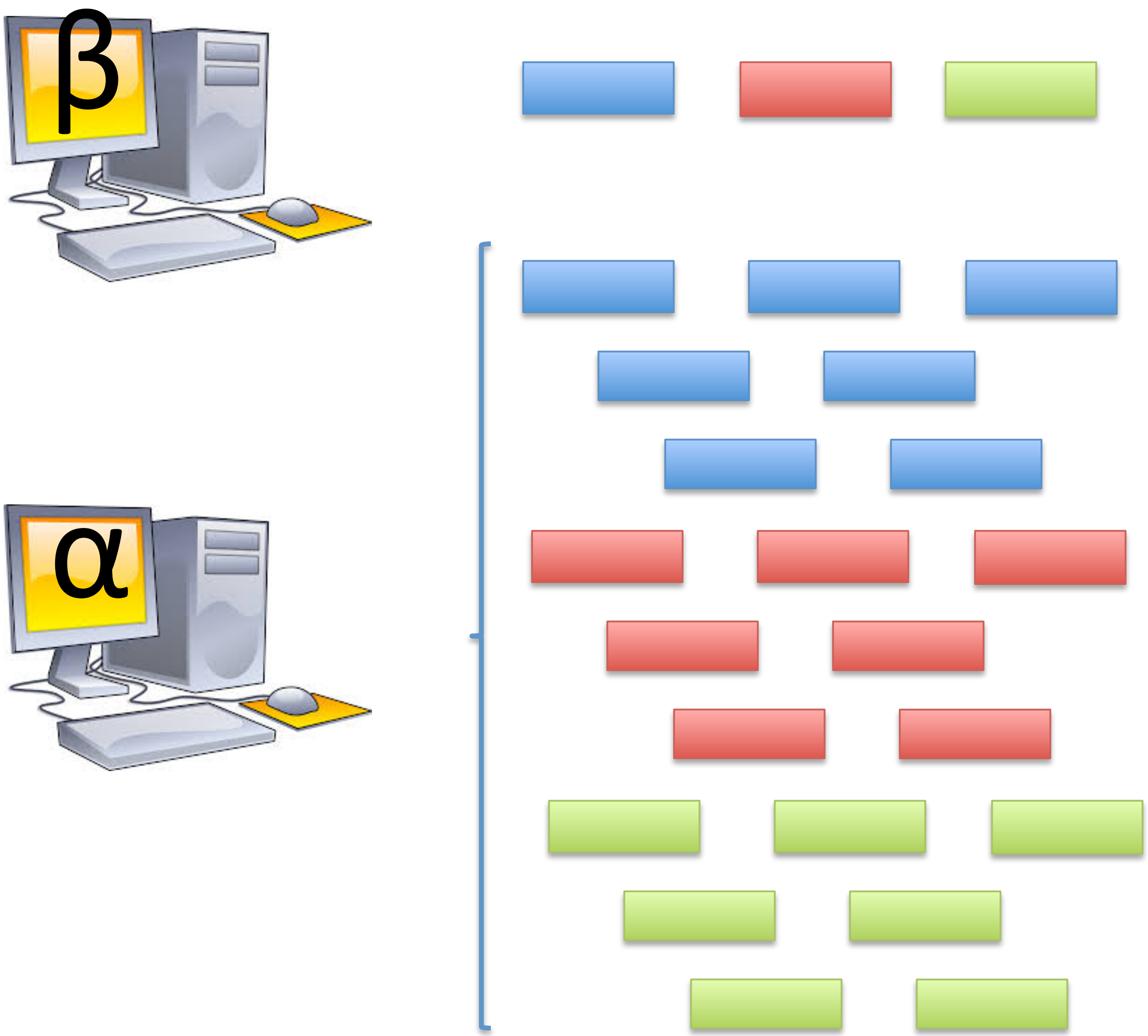
- Can I save bandwidth when A and B are similar?

Solution 1 - bucketing



- Weakness?
- Can I do better?

Solution 2 – sliding windows

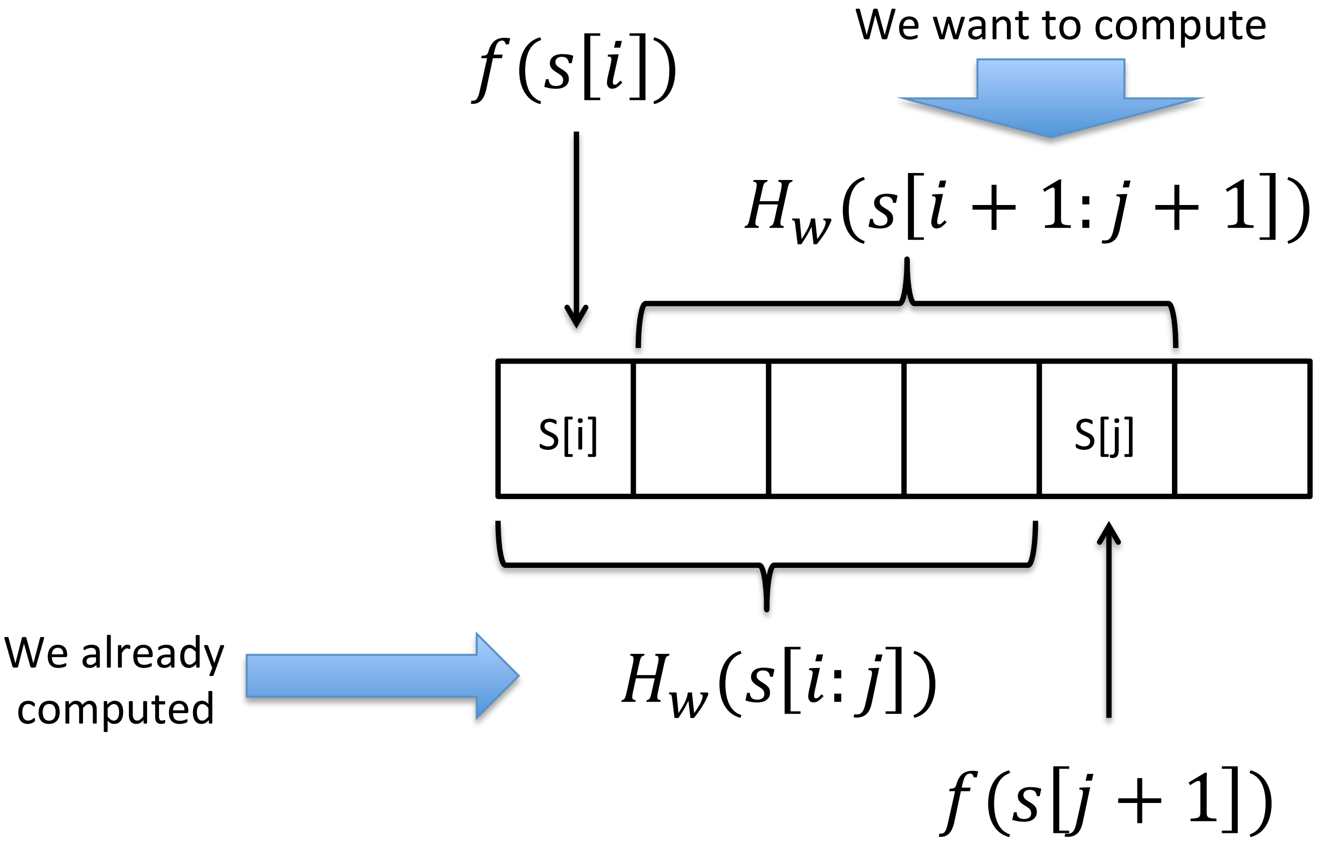


Can I do better?

- Intense use of cpu in **alpha**

Solution 3 – rolling hashing

$$H_w(s[i+1{:}\, j+1]) = H_w(s[i{:}\, j]) + f(s[j+1]) - f(s[i])$$



$f(s[i])$

We want to compute

$H_w(s[i+1{:}\, j+1])$

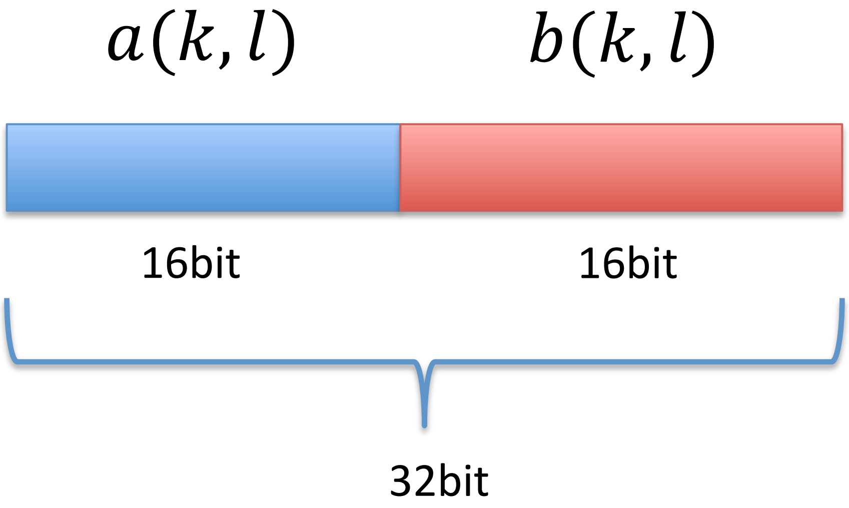
We already computed

$H_w(s[i{:}\, j])$

$f(s[j+1])$

Solution 3 – rsync rolling hashing

- A 32 bit long hash consisting in merging 2 16 bit hash functions



$a(k,l)$ $b(k,l)$

16bit 16bit

32bit

Solution 3 – rsync rolling hashing

- A two hashing strategy

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

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

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

$$s(k,l) = a(k,l) + M\, b(k,l)$$

Solution 3 – rolling hashing

- A convenient way to derive next hash

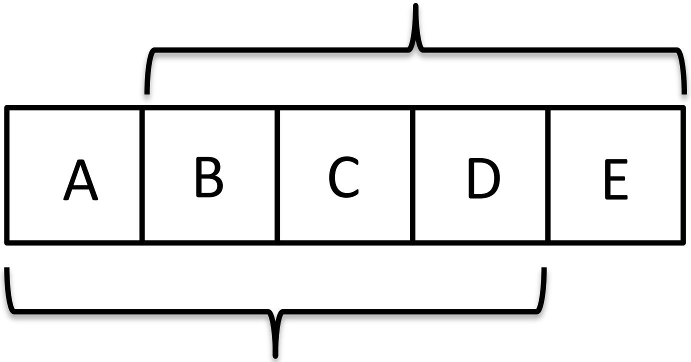
$$a(k+1, l+1) = (\ a(k, l) + \ X_{l+1} - \ X_k)\ mod\ M$$

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

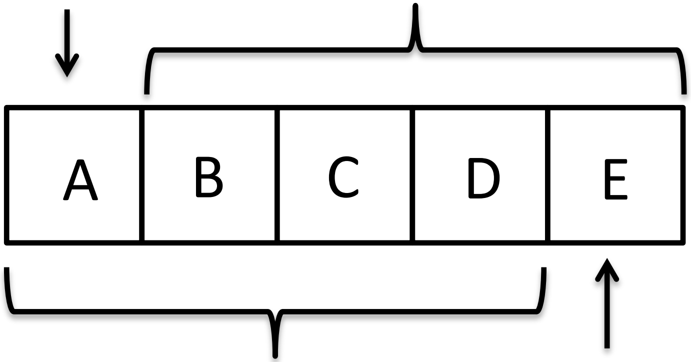
- Is it M=2^{16} a good idea?
- Collisions?

Update: an example

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



Update: an example



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

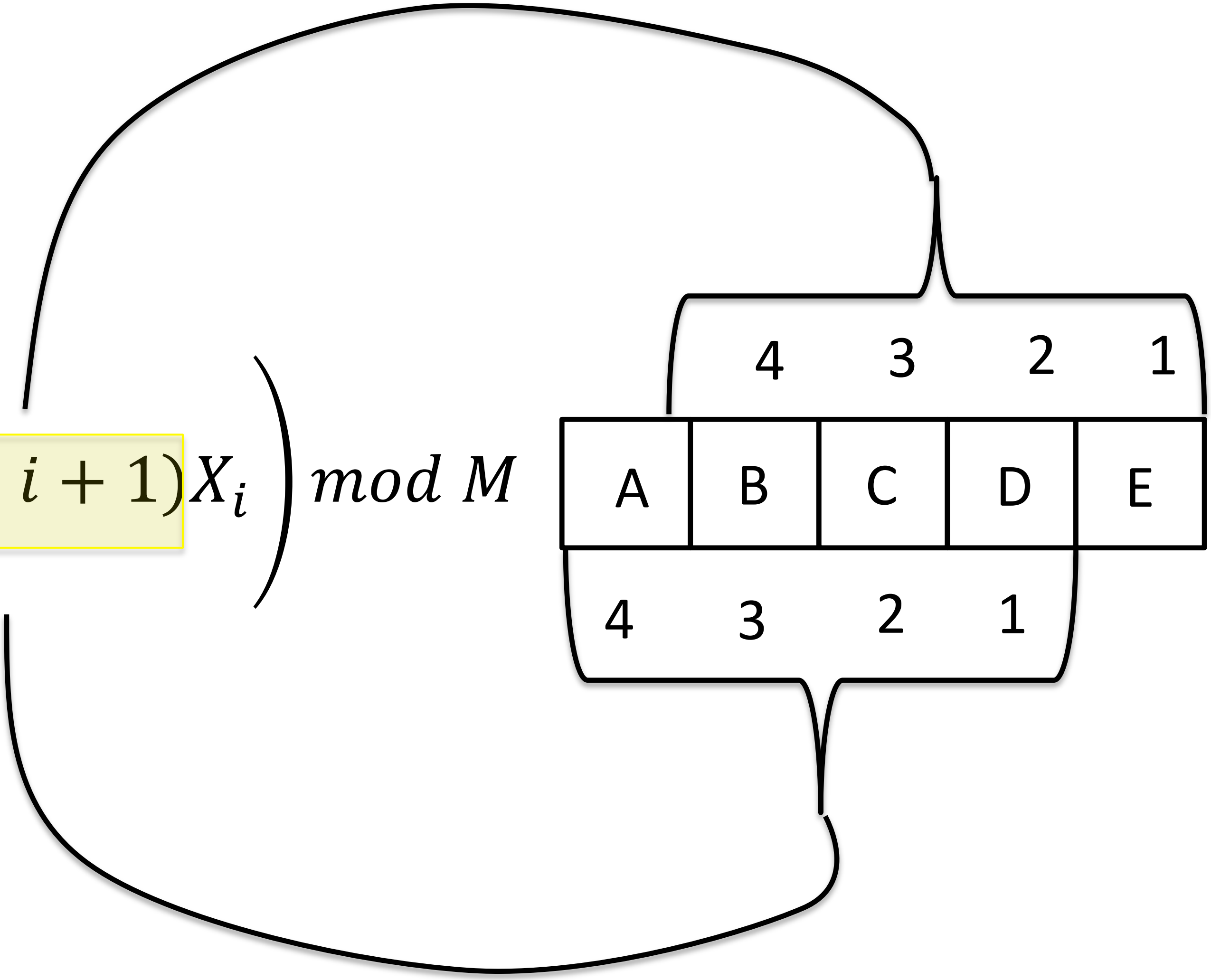
- a(1,4) = A + B + C + D
- a(2,5) = a(1,4) - A + E =

  = A + B + C + D – A + E =

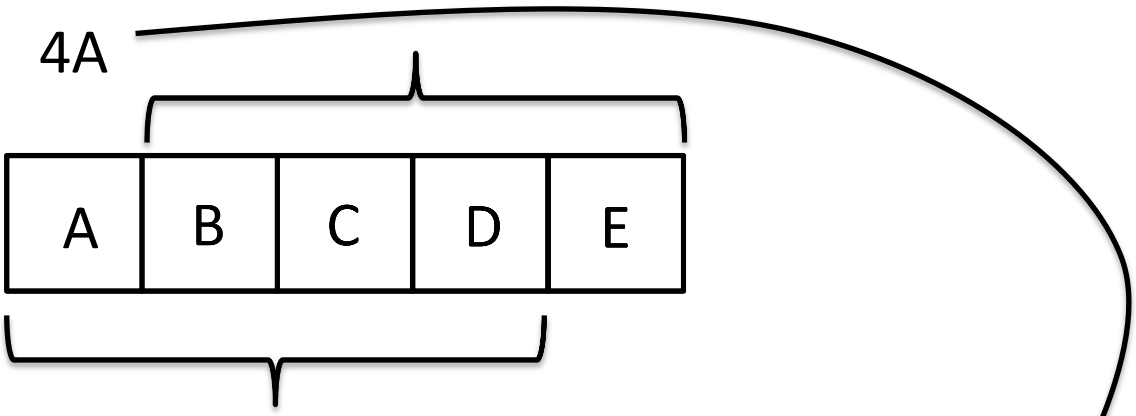
  = B + C + D + E

Update: an example

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



Update an example (2)



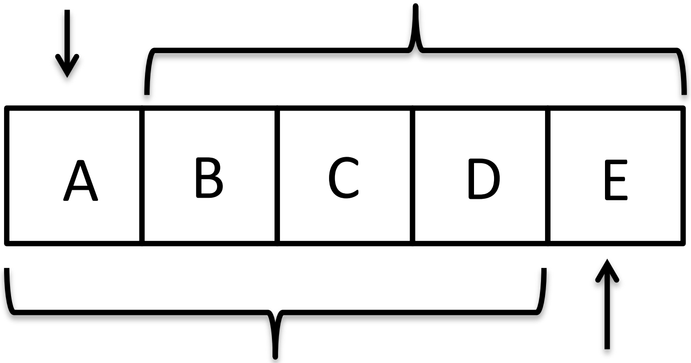
b(1,4) = 4A + 3B + 2C + 1D

$$b(k+1,l+1) = (b(k,l) - (l-k+1)X_k + a(k+1,l+1))\ mod\ M$$

↓

a(2,5) = B + C + D + E

Update: an example



A B C D E

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

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

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

$$= \quad 3B + 2C + 1D + B + C + D + E =$$

$$= \quad 4B + 3C + 2D + E$$

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
  - One 32bit rolling hashing function
  - A stronger 128bit hash (Rsync uses MD4)

- The rolling hashing for each possible offset
- The stronger hashing in case a collision is detected

Solution 4 - rsync

- Use two hash functions
  - One 32bit rolling hashing function
  - A stronger 128bit hash (Rsync uses MD4)

- The rolling hashing for each possible offset
- The stronger hashing in case a collision is detected

- 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

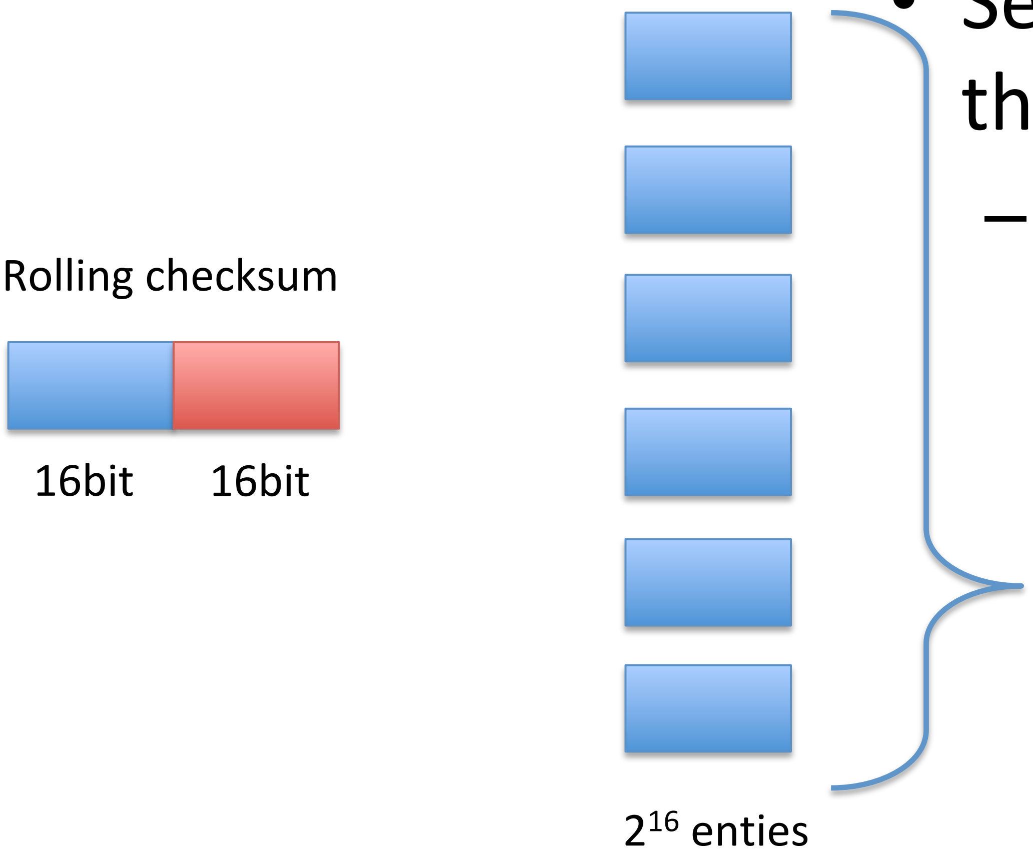
- Is linear scanning an option?

Checksum searching: possible solutions

- Binary search
  - Preprocessing requires sorting O(n lg n)
  - Searching requires O (lg n)
- 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



Rolling checksum

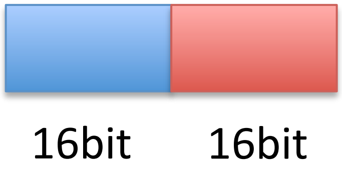
16bit 16bit

128kb

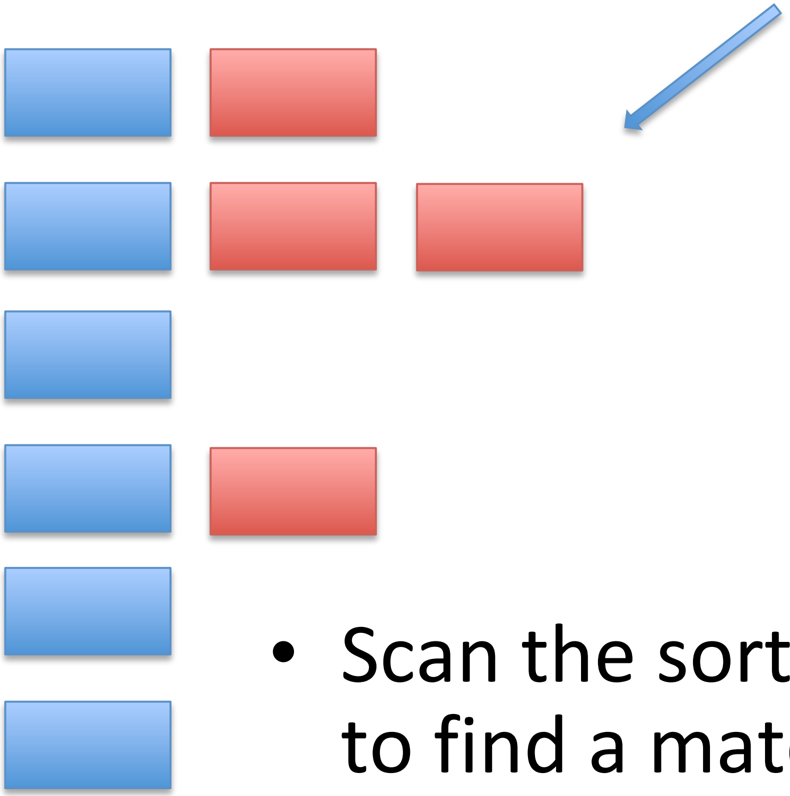
2^{16} enties

The rsync three way test

Rolling checksum



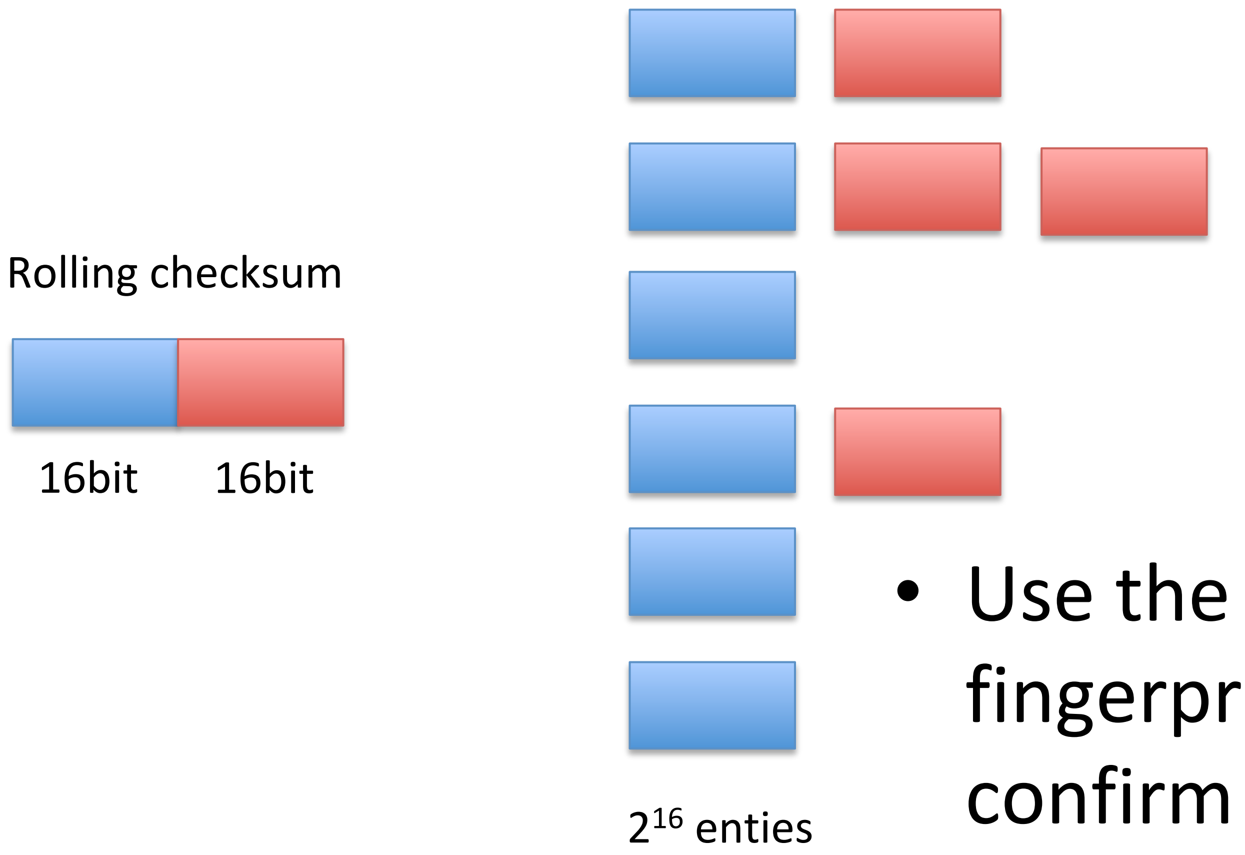
Sorted by incremental hash value



2^{16} enties

- 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?
- 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