What Should Be In A Parallel Hashing Standard?

John Kelsey, NIST, 2014 SHA3 Workshop

Two Standards

Tree hashing (NOT this discussion)

- Binary Merkle trees for crypo applications
- Arbitrary depth of tree
- Hash based signatures, timestamping, redactable signatures, etc.

Fast parallel hashing (this discussion)

- Focused on performance
- SIMD, multicore, multiple processors, etc.
- One- or two-level trees

Parallel Hashing Goals

(It's all about performance)

We want to...

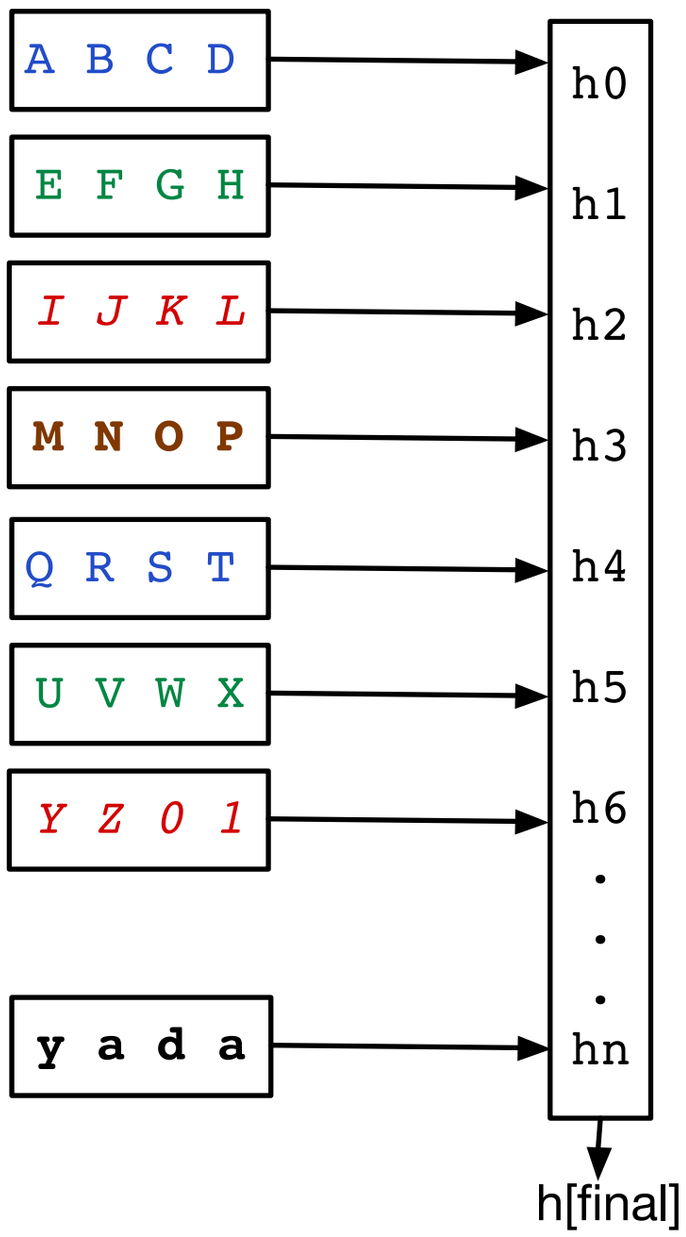
- Benefit from parallelism (SIMD and multicore)....
- ... but don't impose too many costs on weaker machines checking hash!
- Allow enough options to get performance benefit...
- ...but not too many to test!

Our Ideas So Far

- Limited tree depth (1-2 max)
    - More levels of tree = more hash states for sequential implementations
- Support *segmentation* for long messages
- Support *interleaving*
- Support *combination* of segmentation and interleaving(?)

Note: There are many other options I'm not even covering.

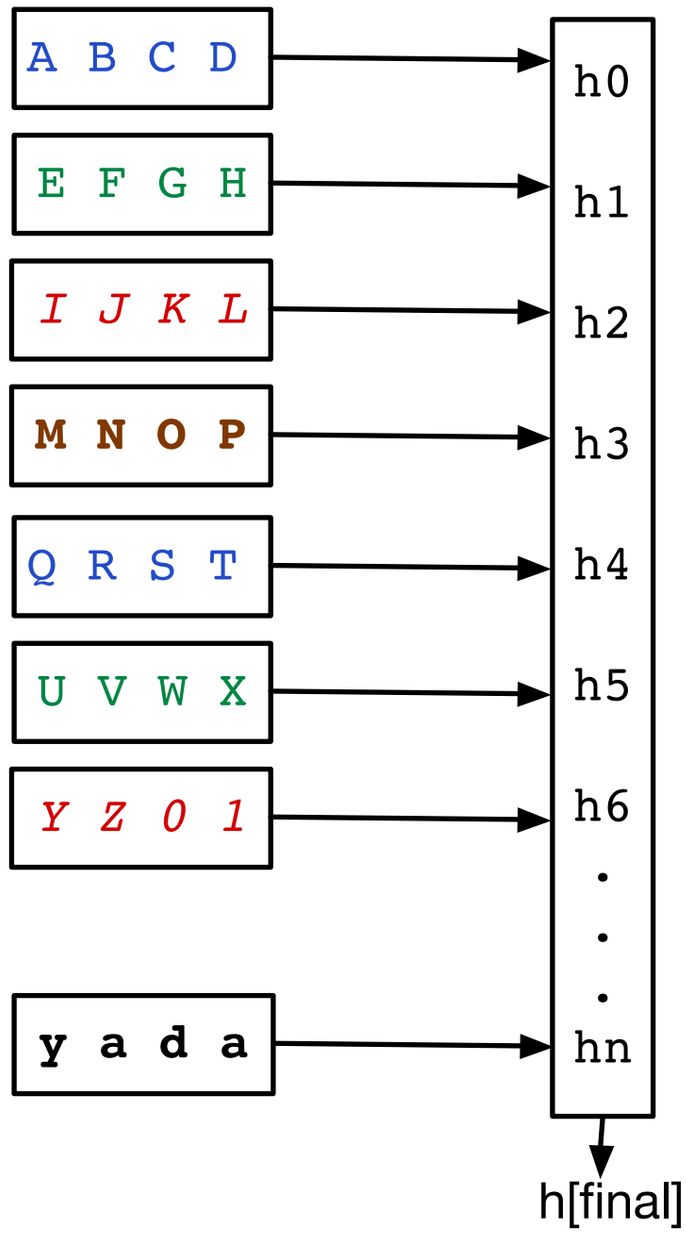
Segmented Hashing



1. **Break** message into large segments (16 KB +)
2. **Hash** each segment and store result.
3. **Repeat** until whole message hashed.
4. **Finally,** hash resulting hashes to get the final hash value.

Segmented Hashing (2)

- Each segment hashed independently
- Hash computation not bound to architecture of any one machine
- Tree with only one level
- Easy to compute sequentially



Questions about Segmenting

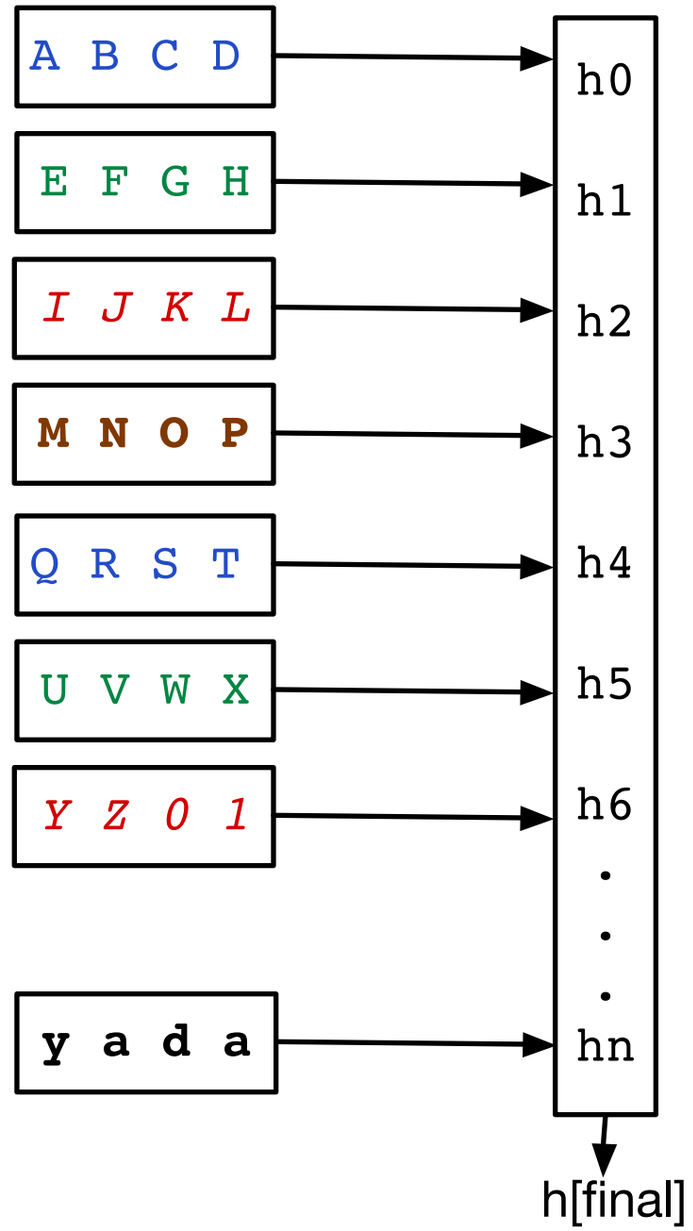
- **Which** segment sizes should be supported?

Depends partly on

  - Message size
    - Time spent in leaves vs root
  - Hash details (padding, message block length)

- **How many** segment sizes should be supported?

- Should we have more levels of tree?

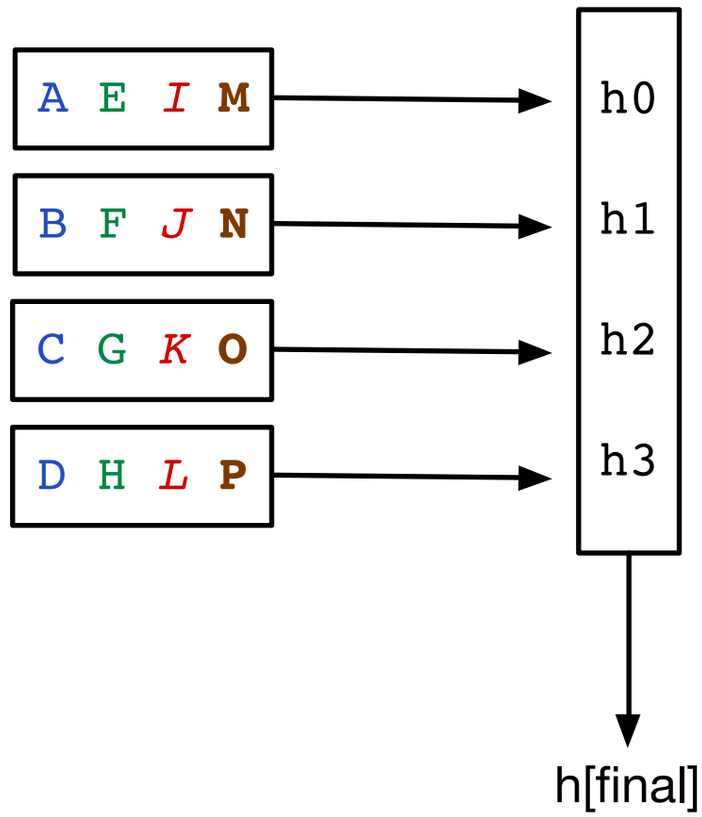


Interleaved Hashing

(It's all about SIMD)

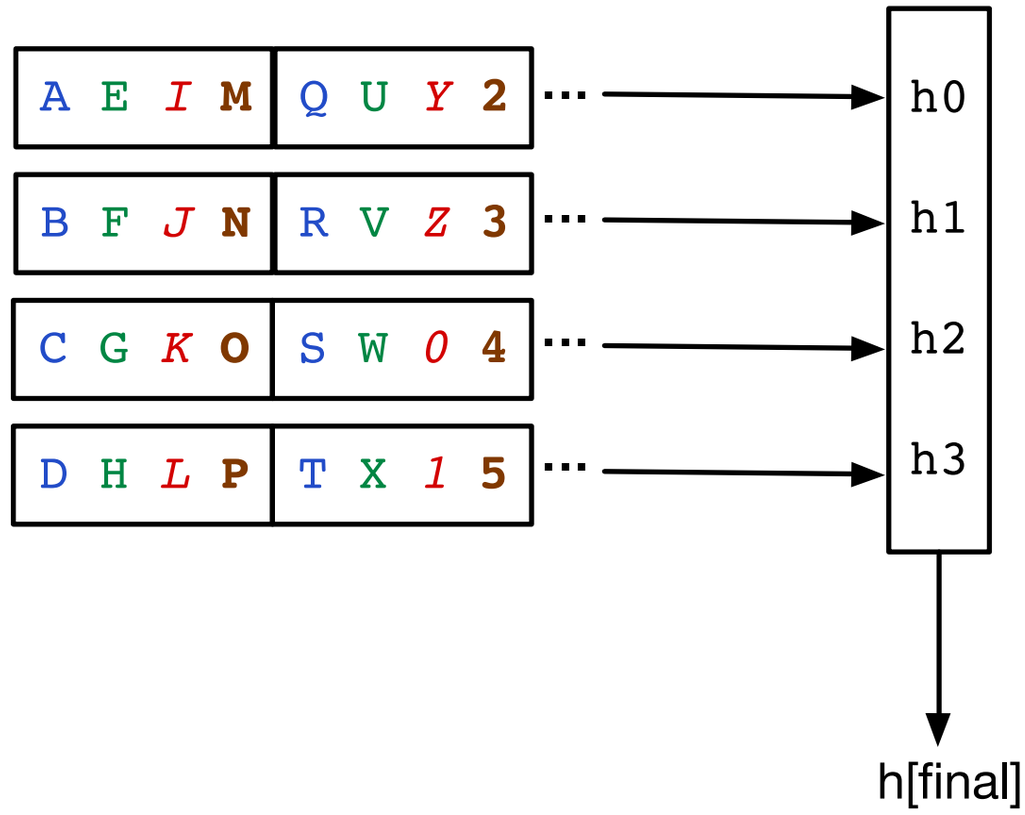
`Original Message:` A B C D E F G H *I J K L* **M N O P**

1. **Feed** every Nth word into different hash context.
2. **Use** SIMD to compute all N hashes in parallel.
3. **Repeat** until whole message hashed.
4. **Finally,** hash resulting hashes to get the final hash value.



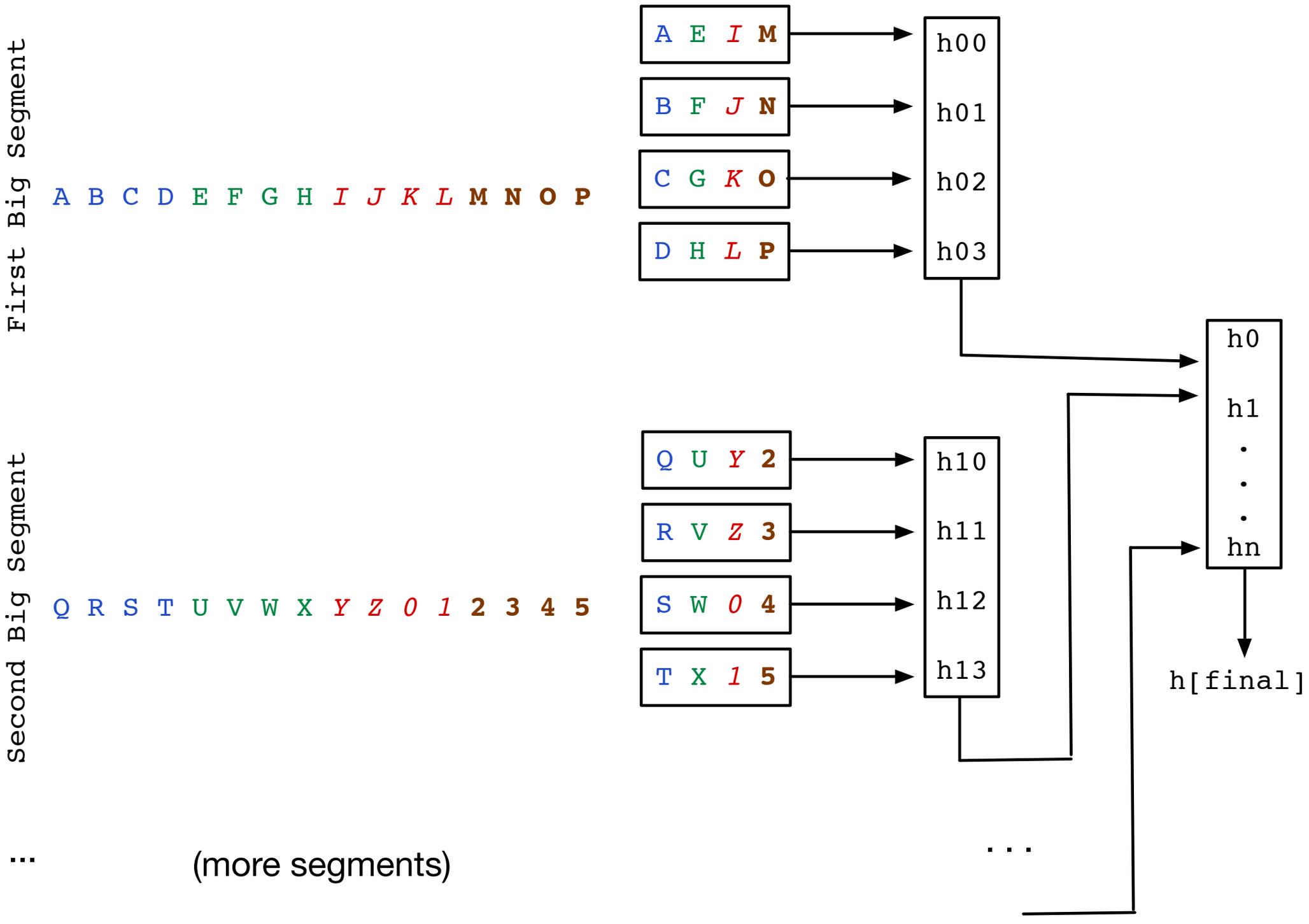
Interleaving (2)

- Hash parameters bound to particular machine's architecture
  - Size of SIMD registers determines how many parallel "lanes" computed
  - Natural word size of algorithm determines size of "slices"
- Sequential machines take some performance hit, as do some other SIMD machines



Segmenting Plus Interleaving

(Many cores, each SIMD)



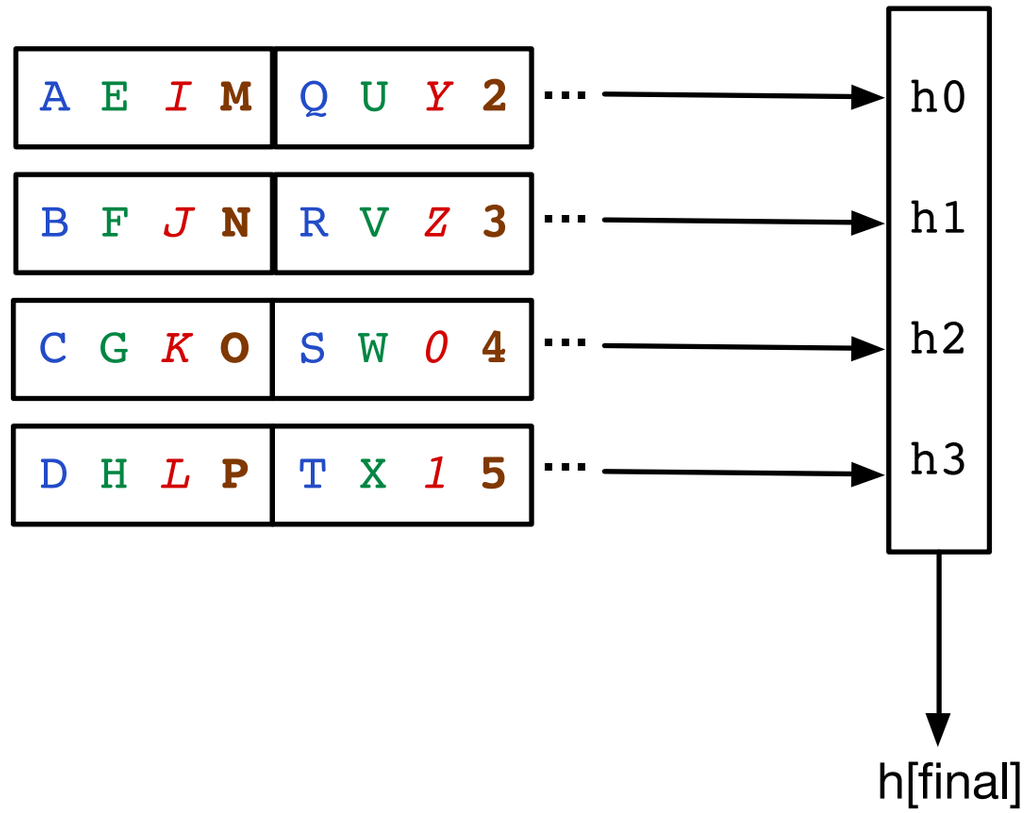
Questions about Interleaving

What choices for # of lanes should we allow?

- 4,8,16,32? More? Less?

What should we standardize?

- Interleaving only?
- Segmenting+Interleaving only?
- Both?
- Neither?



Full hash function or compression function?

- SHAKEs have sakura padding (thus support for parallel and tree modes) built in.

- Other hashes don't...and we want a generic standard

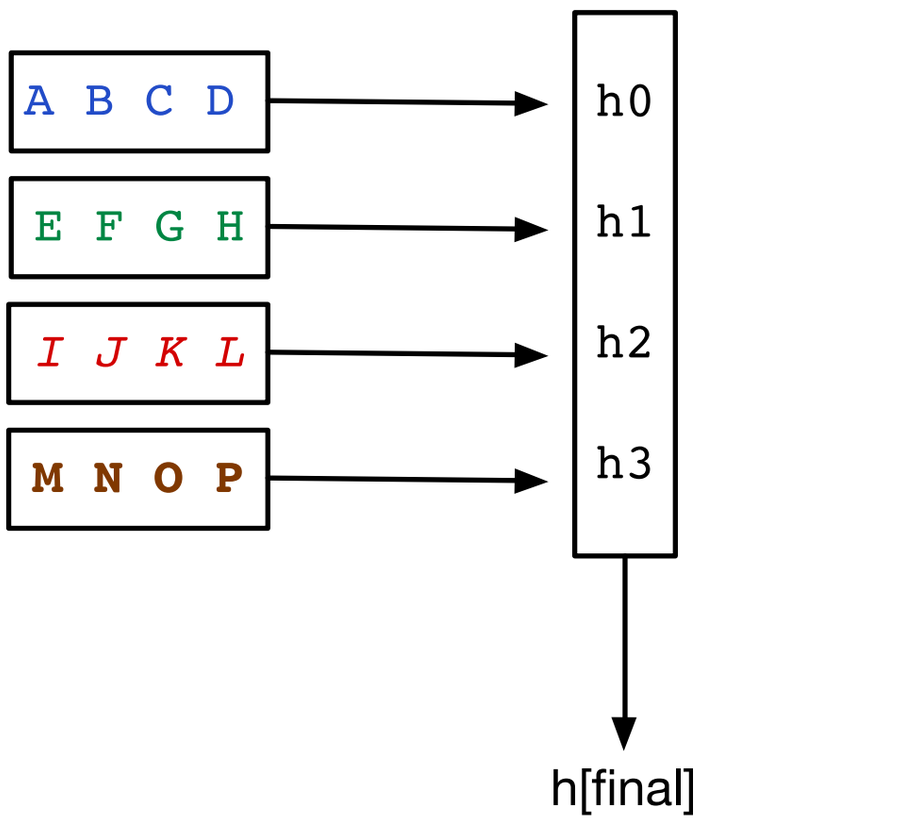
- If we use full hash function....

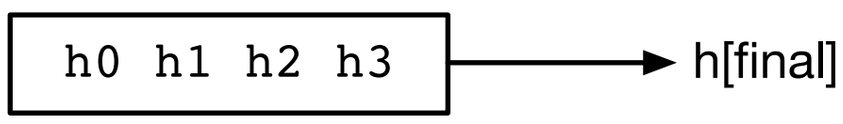
- **Good news**: existing libraries and hardware can be used to do parallel hashing mode.

- **Bad news**: collisions between sequential and parallel modes, and between parallel modes with different parameters!

Collisions between parallel and sequential hashes

Input to Segmented Parallel Hash: A B C D E F G H *I J K L* **M N O P**





If we use unaltered hash function...

- For any message you parallel hash...
- ...you can find a different message that gives the same hash value from the sequential hash.

Colliding message for sequential hash: h0 h1 h2 h3

Our Questions

Architectures

- Should we standardize all three of these or a subset?
- Should we be looking at other architectures? (Deeper trees?)

Parameters

- Interleaving: # of parallel lanes
- Segmenting: size of segment
- How many options do we need?
  - More options = more bugs, harder testing

What are we missing? Where are we about to go wrong?