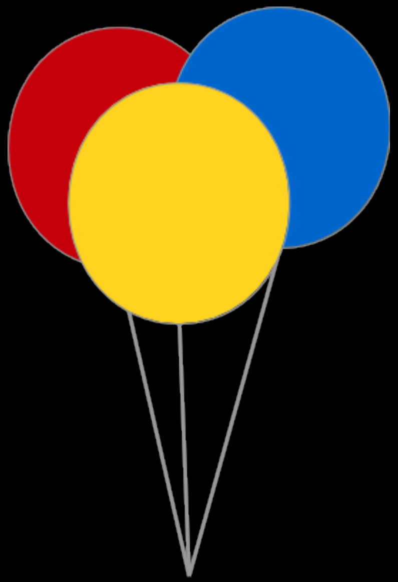


Balloon Hashing

A Memory-Hard Function with Provable Protection Against Sequential Attacks

Dan Boneh, Stanford
*Henry Corrigan-Gibbs, Stanford
Stuart Schechter, Microsoft Research

Balloon Hashing

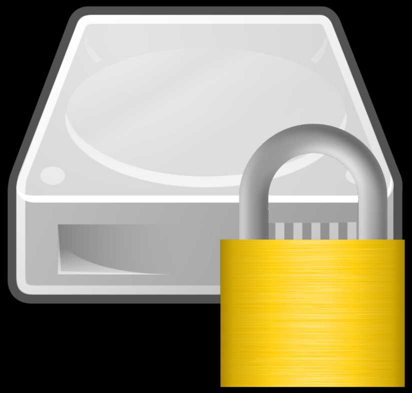
A new password hashing function that:

1. Is <u>proven</u> memory-hard (in the sequential setting)
2. Uses a password-independent data access pattern
3. Matches the performance of the best heuristically secure memory-hard functions

11:35
Google
11:35
WED, SEPTEMBER 17
THU 7:00 AM
YouTube
Maps
Gmail
Play Store

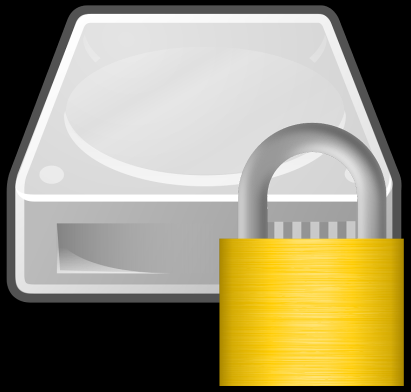
11:35
Google
11:35
WED, SEPTEMBER 17
THU 7:00 AM
YouTube
Maps
Gmail
Play Store

11:35
Google
11:35
WED, SEPTEMBER 17
THU 7:00 AM
YouTube
Maps
Gmail
Play Store



PDF
Adobe

11:35
Google
11:35
WED, SEPTEMBER 17
THU 7:00 AM
YouTube
Maps
Gmail
Play Store



PDF
Adobe

$

The Attacker's Job

User	Salt	H(passwd, salt)
alice	0x65ff0162	0x526642d8
bob	0x37ceb328	0x5a325ad2
carol	0xec967ec1	0xf4441a71
dave	0xfb791a9a	0x1dbd71f3

The Attacker’s Job

User	Salt	H(passwd, salt)
alice	0x65ff0162	0x526642d8
bob	0x37ceb328	0x5a325ad2
carol	0xec967ec1	0xf4441a71
dave	0xfb791a9a	0x1dbd71f3

For each row,
attacker wants to make
2^{30} guesses

Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.

Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.

Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.

If the authentication server can compute…

X hashes per **$ of energy**

then an attacker *with custom hardware* should only be able to compute…

(1+ε)X hashes per **$ of energy**

Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.

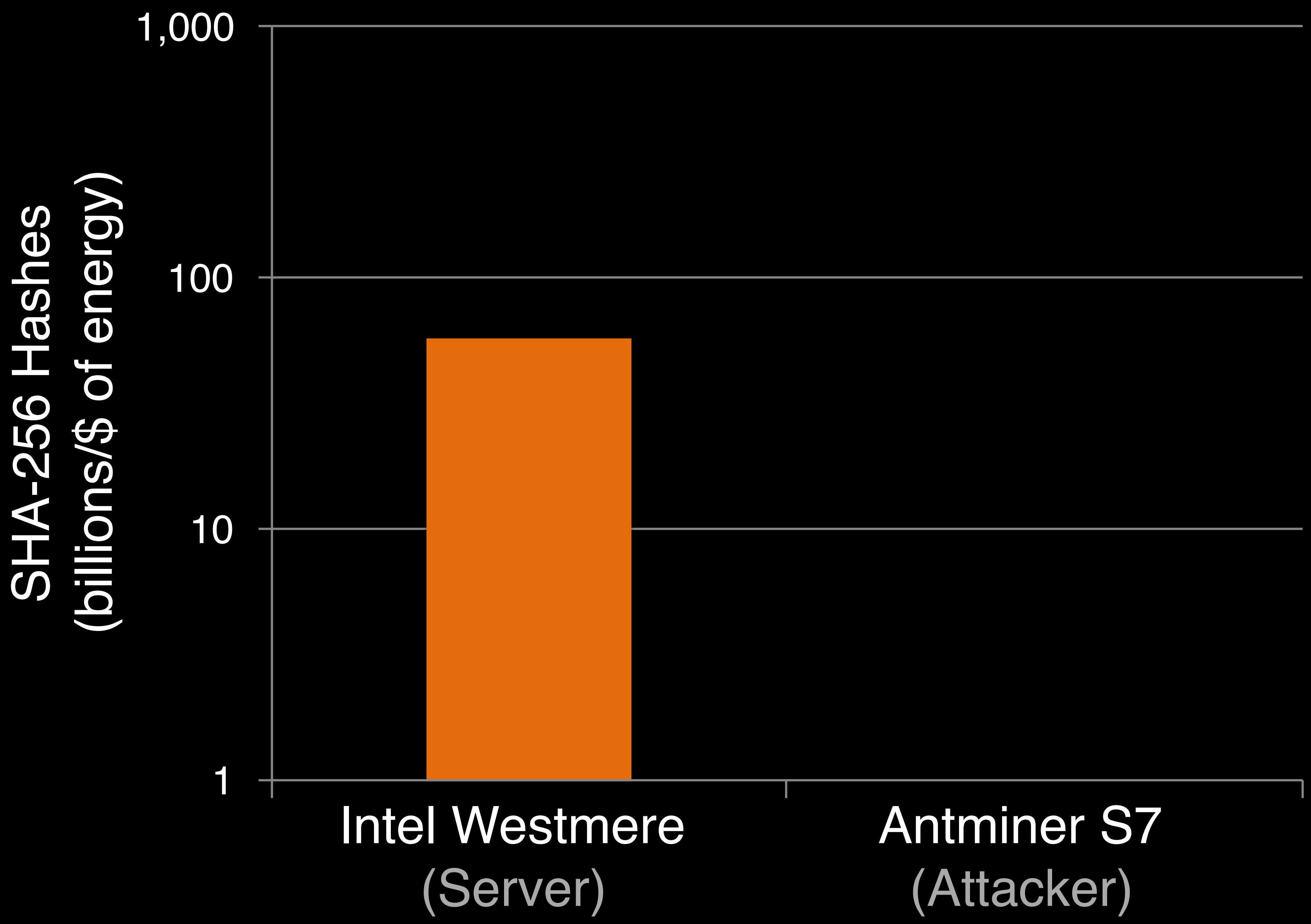
If the authentication server can compute…

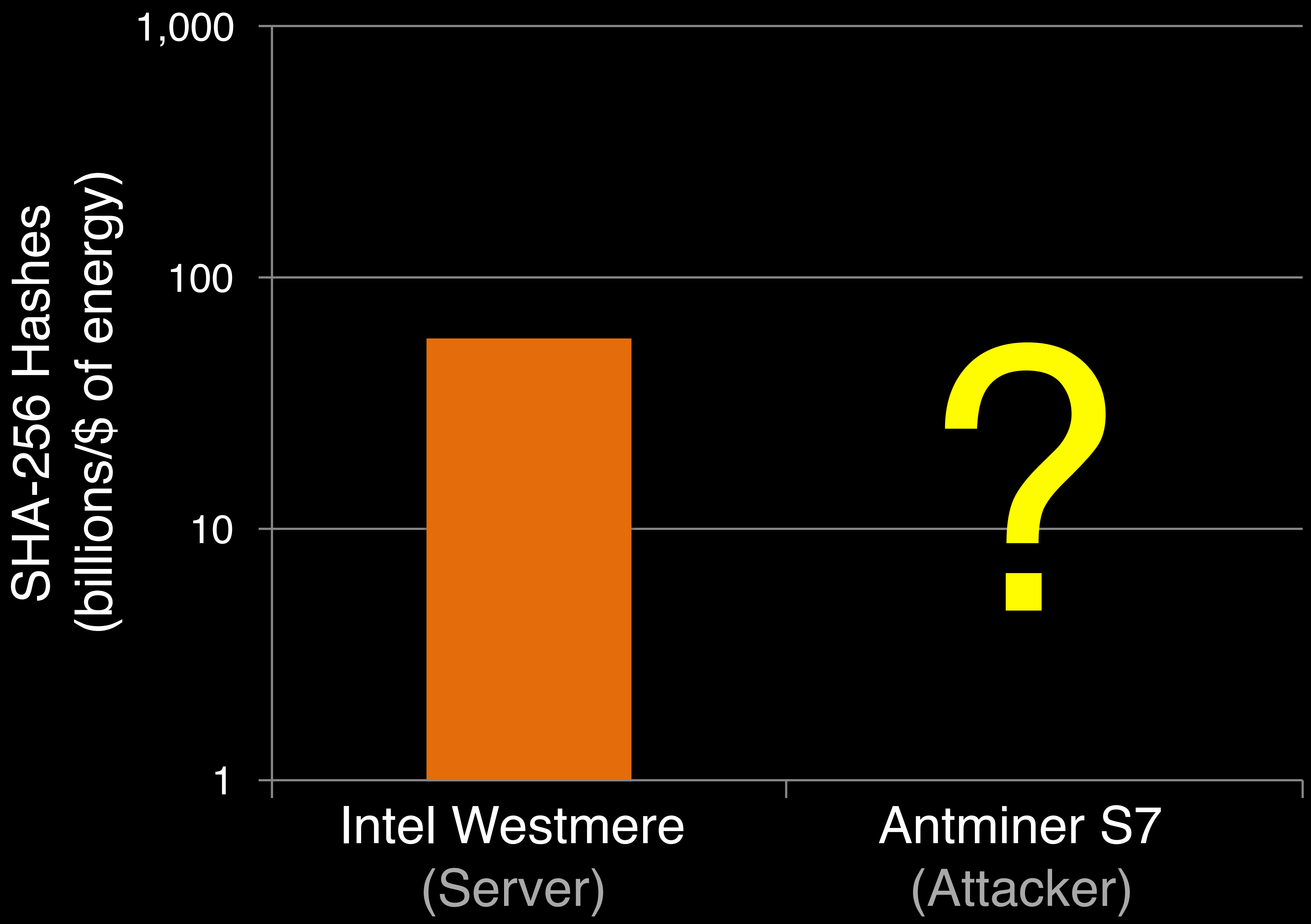
X hashes per **$ of energy**

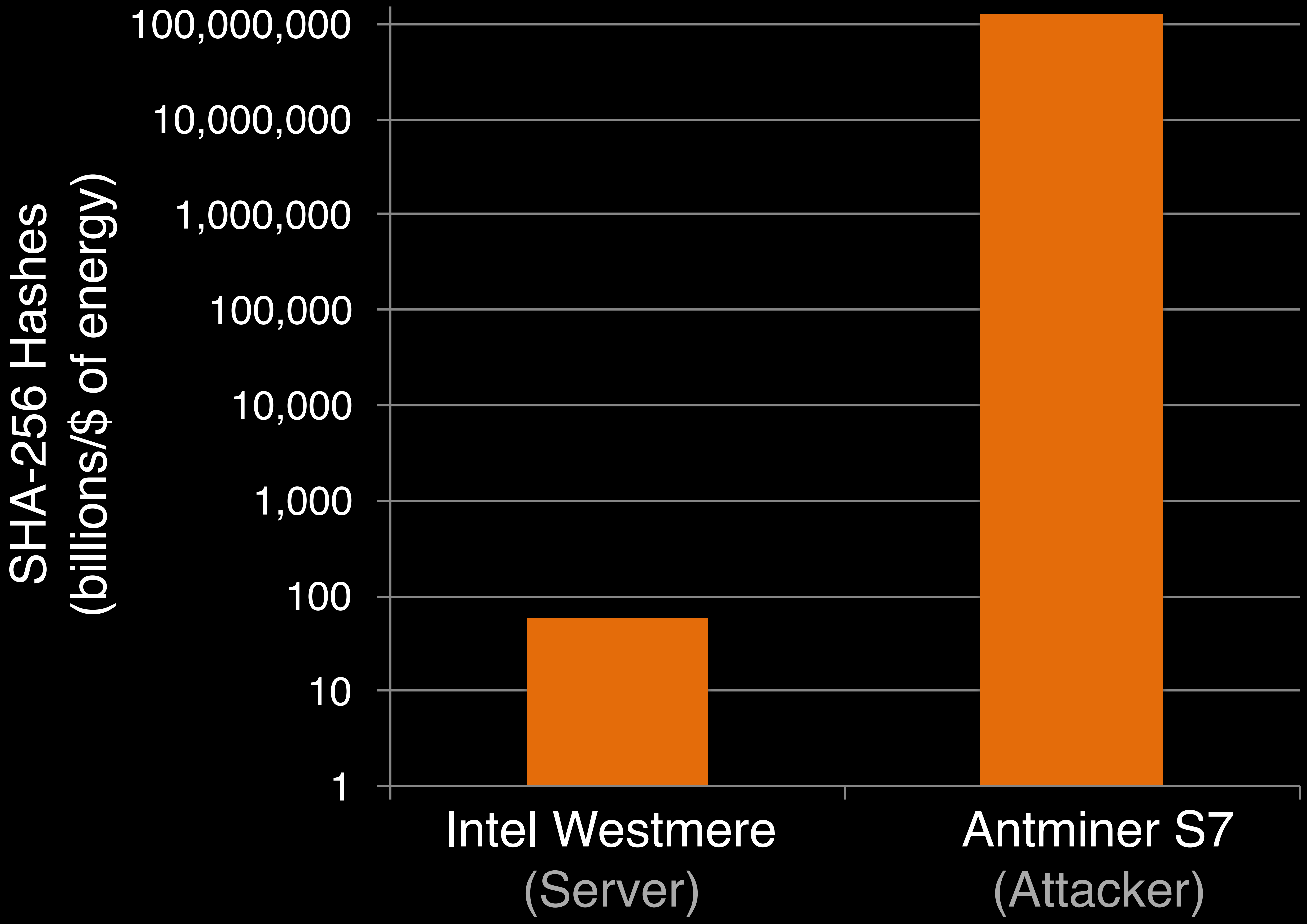
then an attacker *with custom hardware* should only be able to compute…

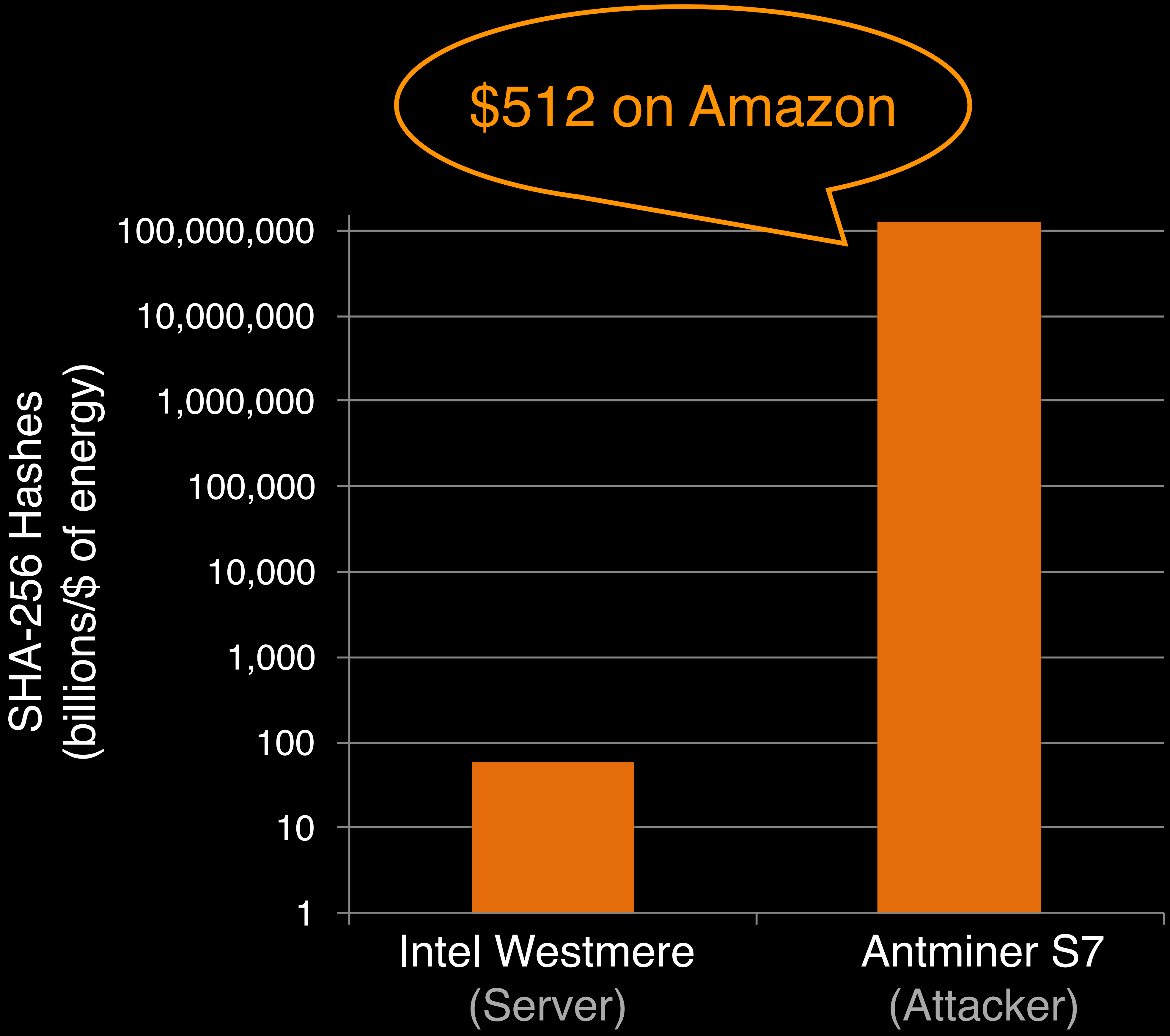
(1+ε)X hashes per **$ of energy**

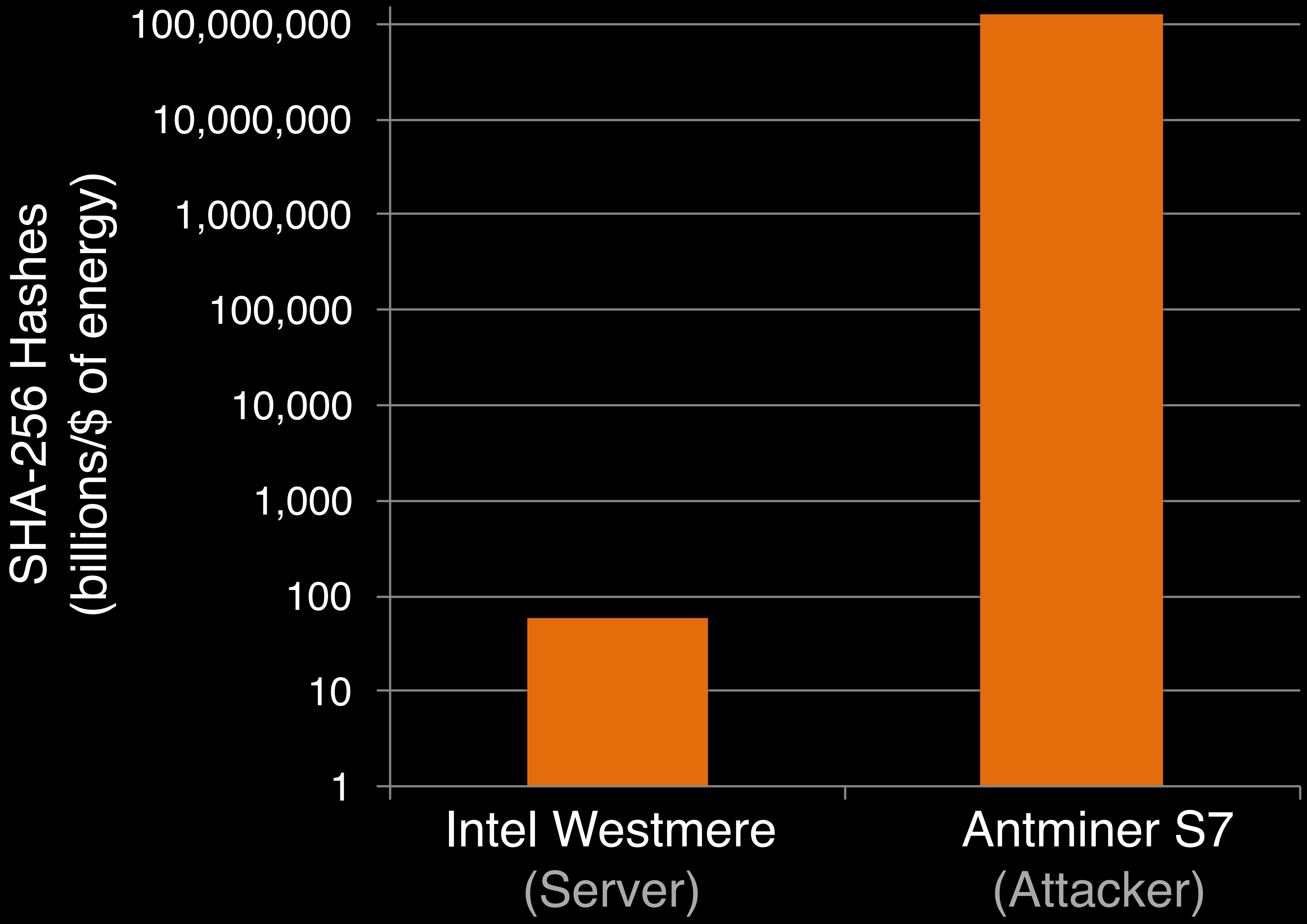
By this metric, conventional hash functions (e.g., SHA-256) are far from optimal!

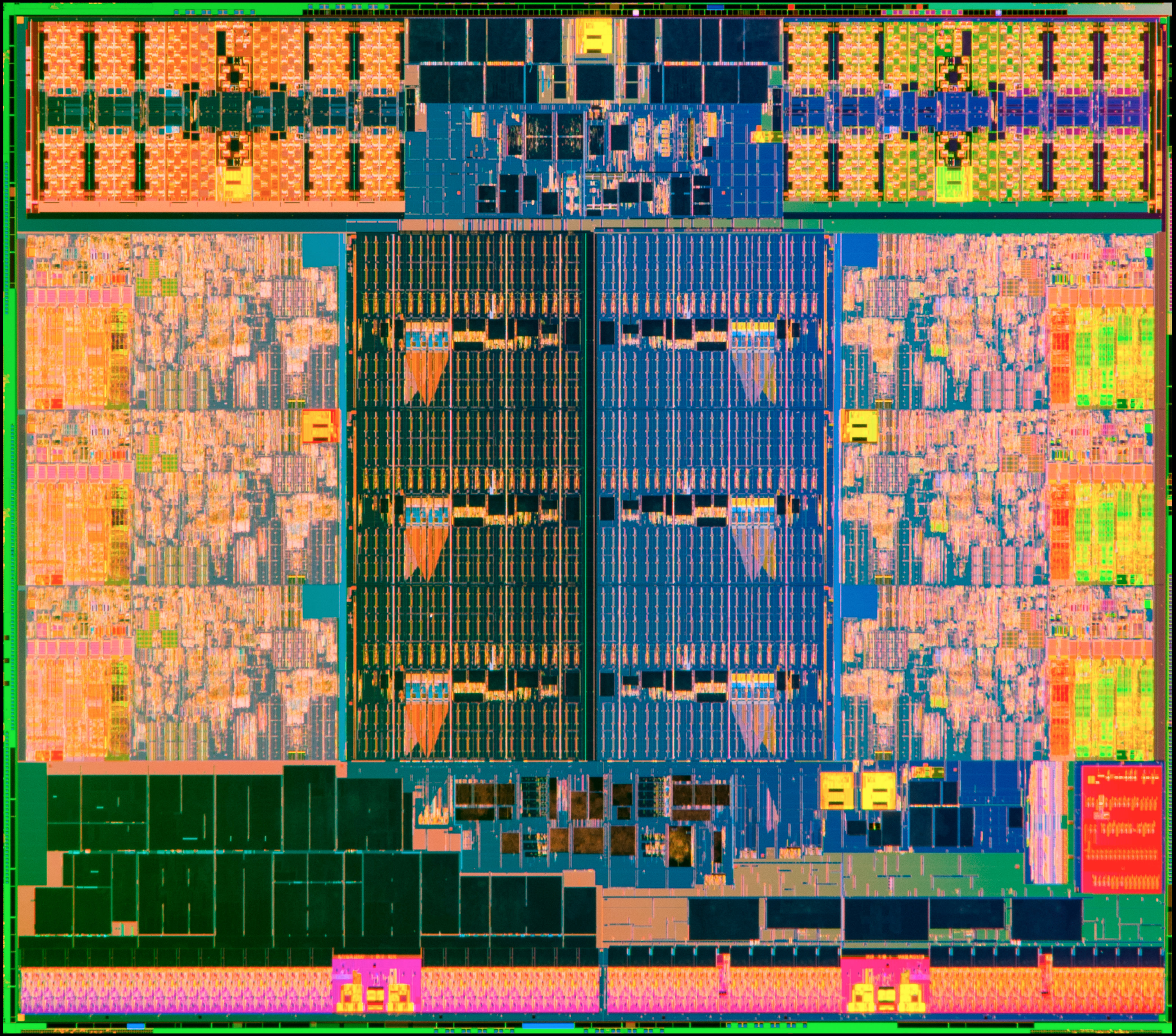

SHA-256 Hashes
(billions/$ of energy)
1,000
100
10
1
Intel Westmere
(Server)
Antminer S7
(Attacker)


SHA-256 Hashes
(billions/$ of energy)
1,000
100
10
1
?
Intel Westmere
(Server)
Antminer S7
(Attacker)

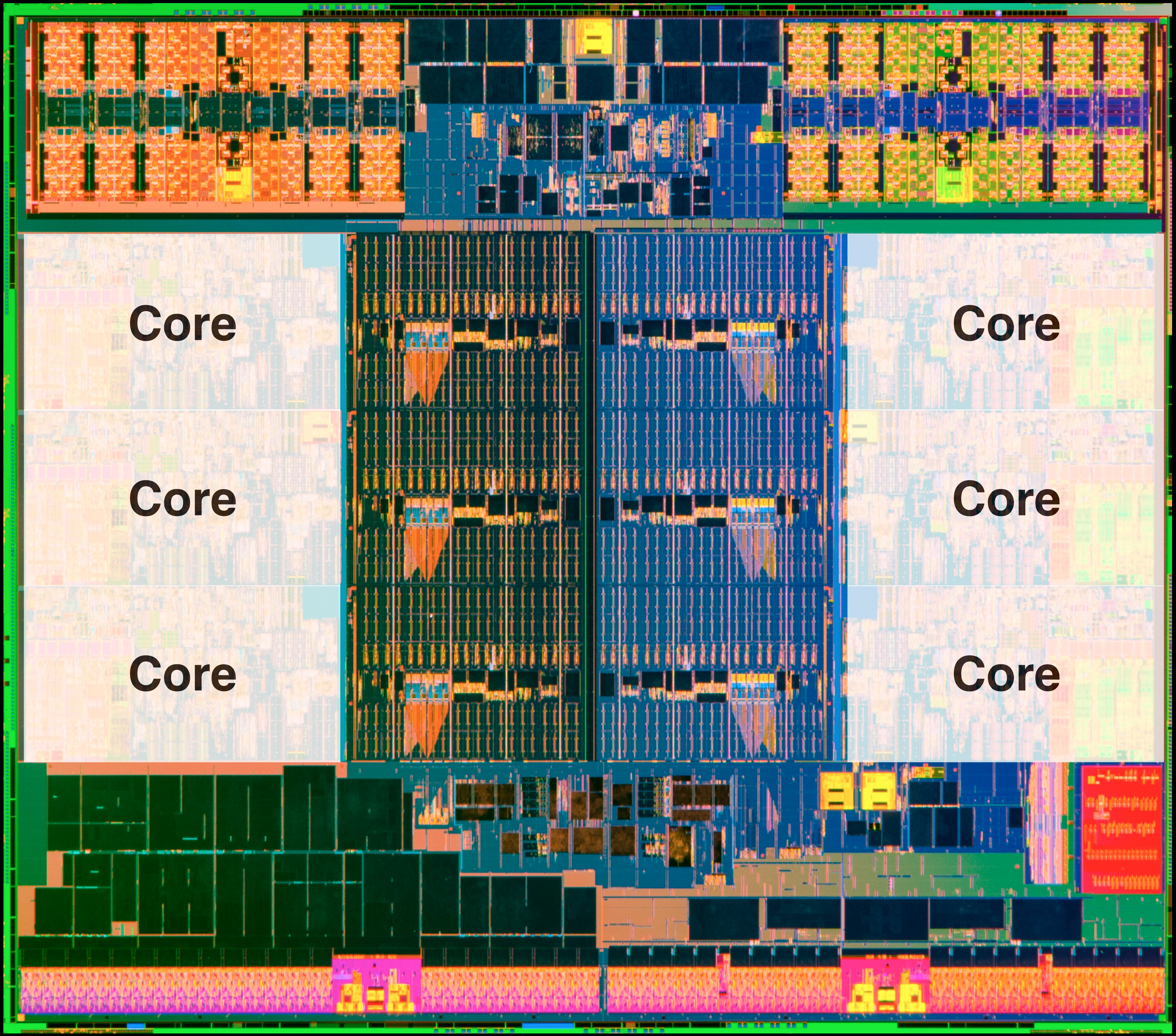

SHA-256 Hashes
(billions/$ of energy)
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
Intel Westmere
(Server)
Antminer S7
(Attacker)


$512 on Amazon
SHA-256 Hashes
(billions/$ of energy)
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
Intel Westmere
(Server)
Antminer S7
(Attacker)

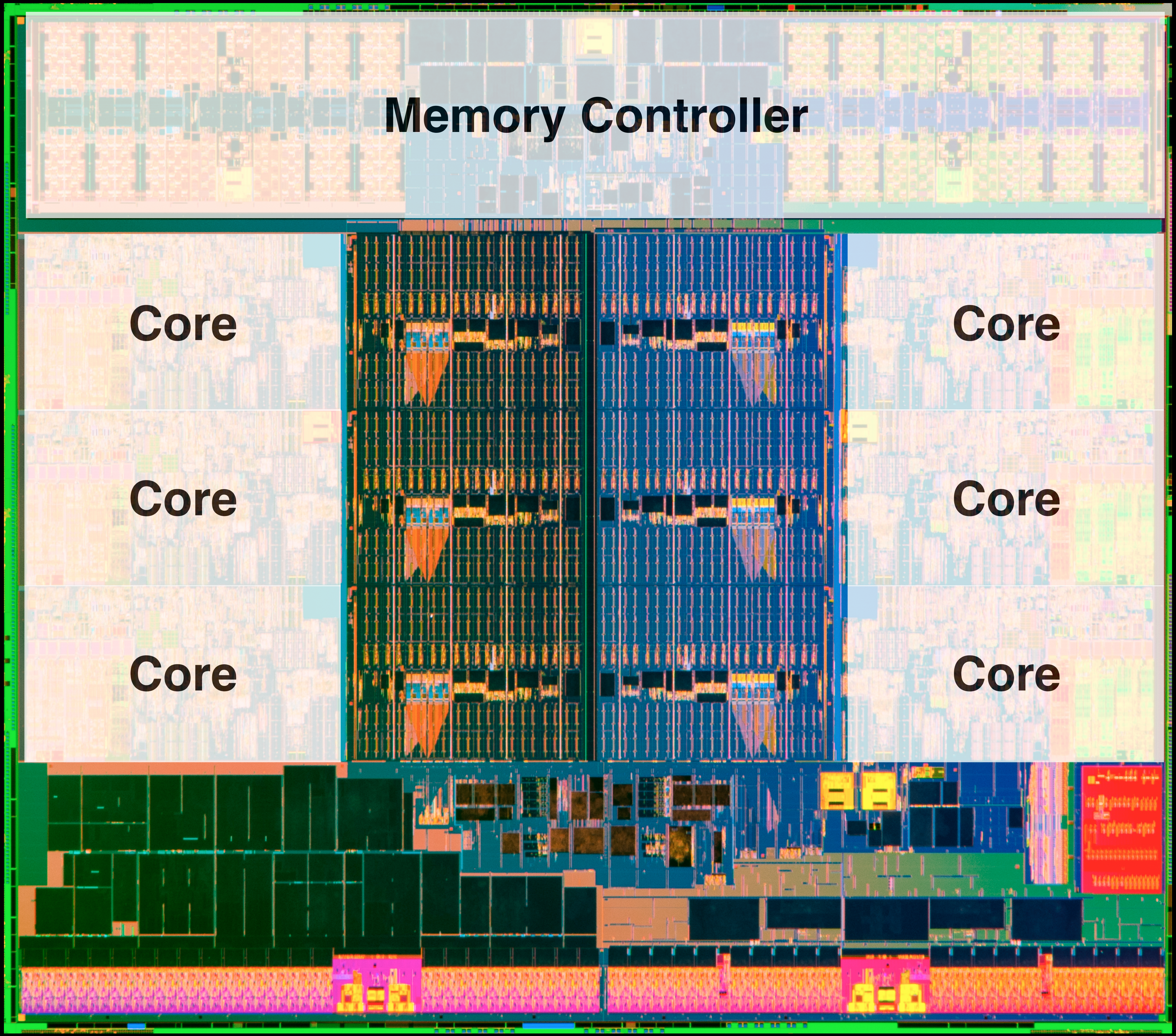

SHA-256 Hashes
(billions/$ of energy)
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
Intel Westmere
(Server)
Antminer S7
(Attacker)



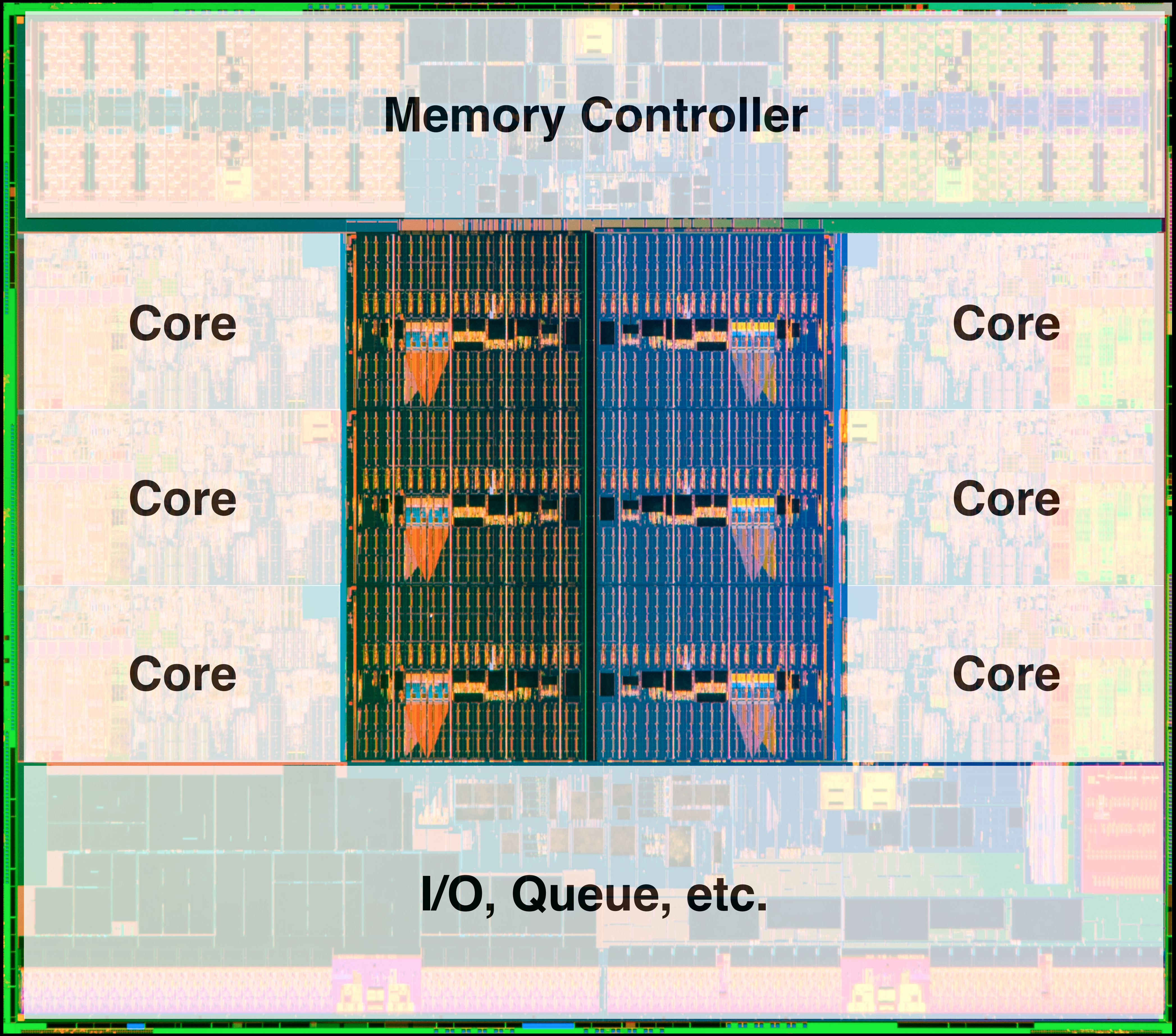
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



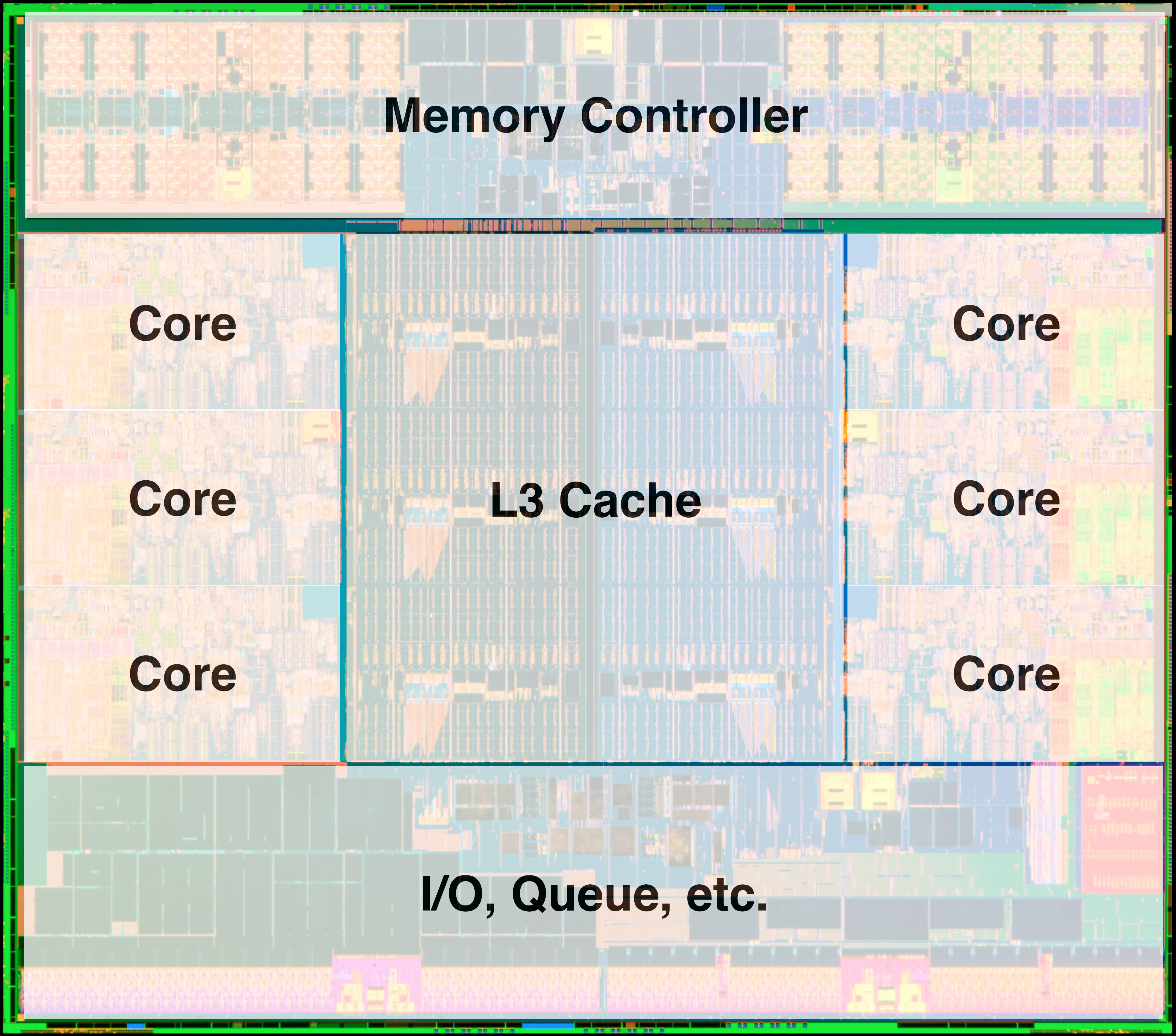
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



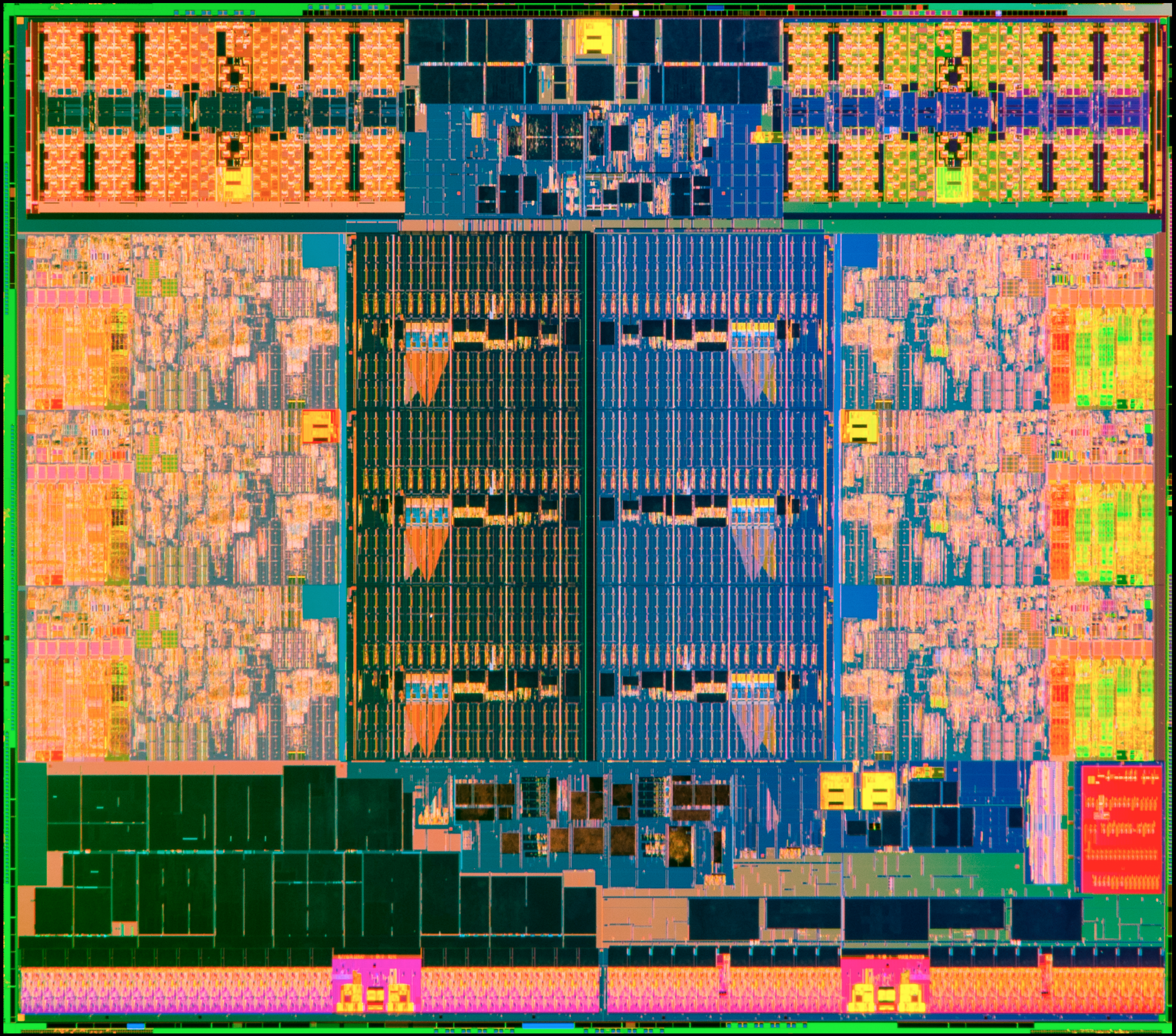
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



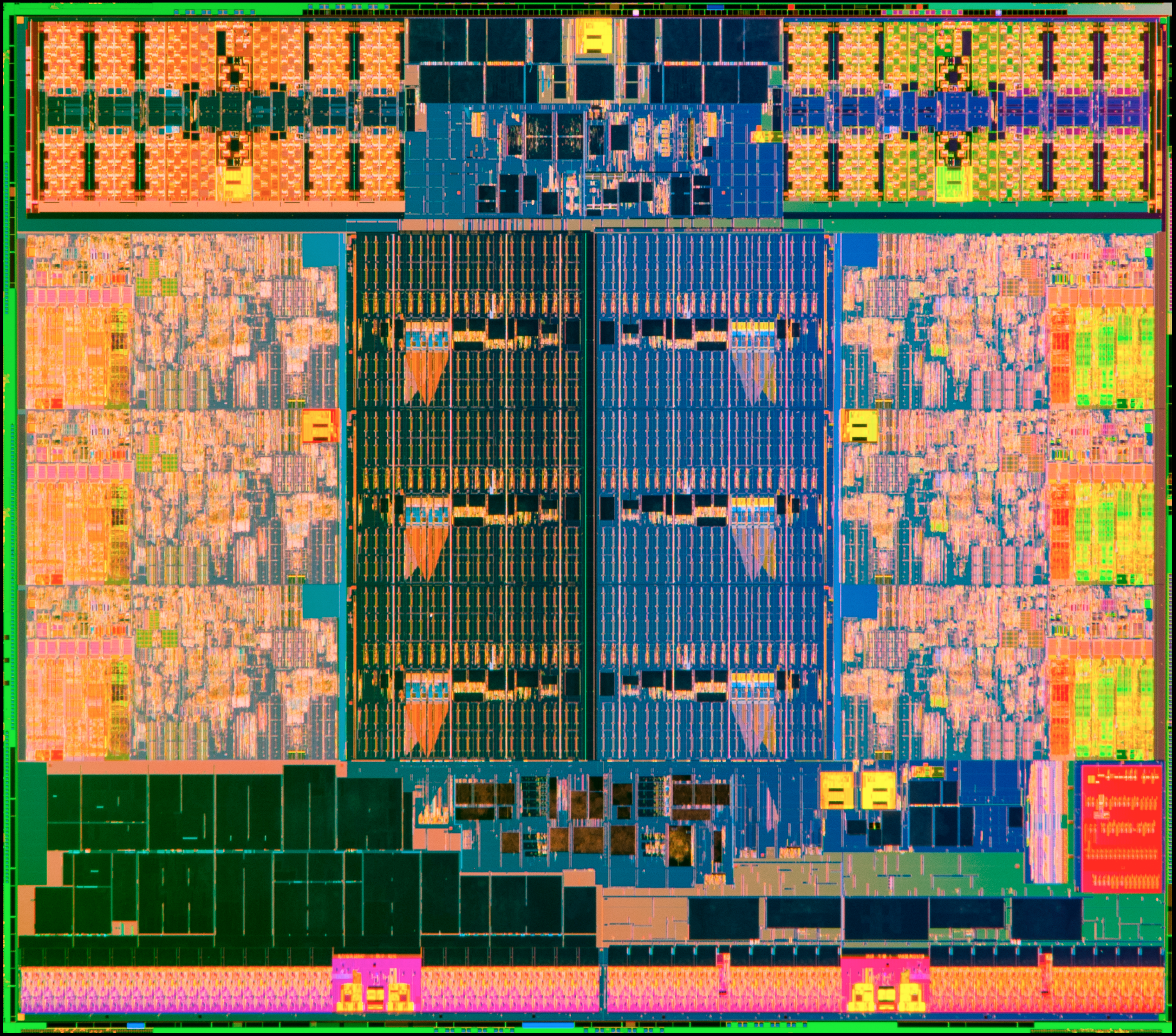
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



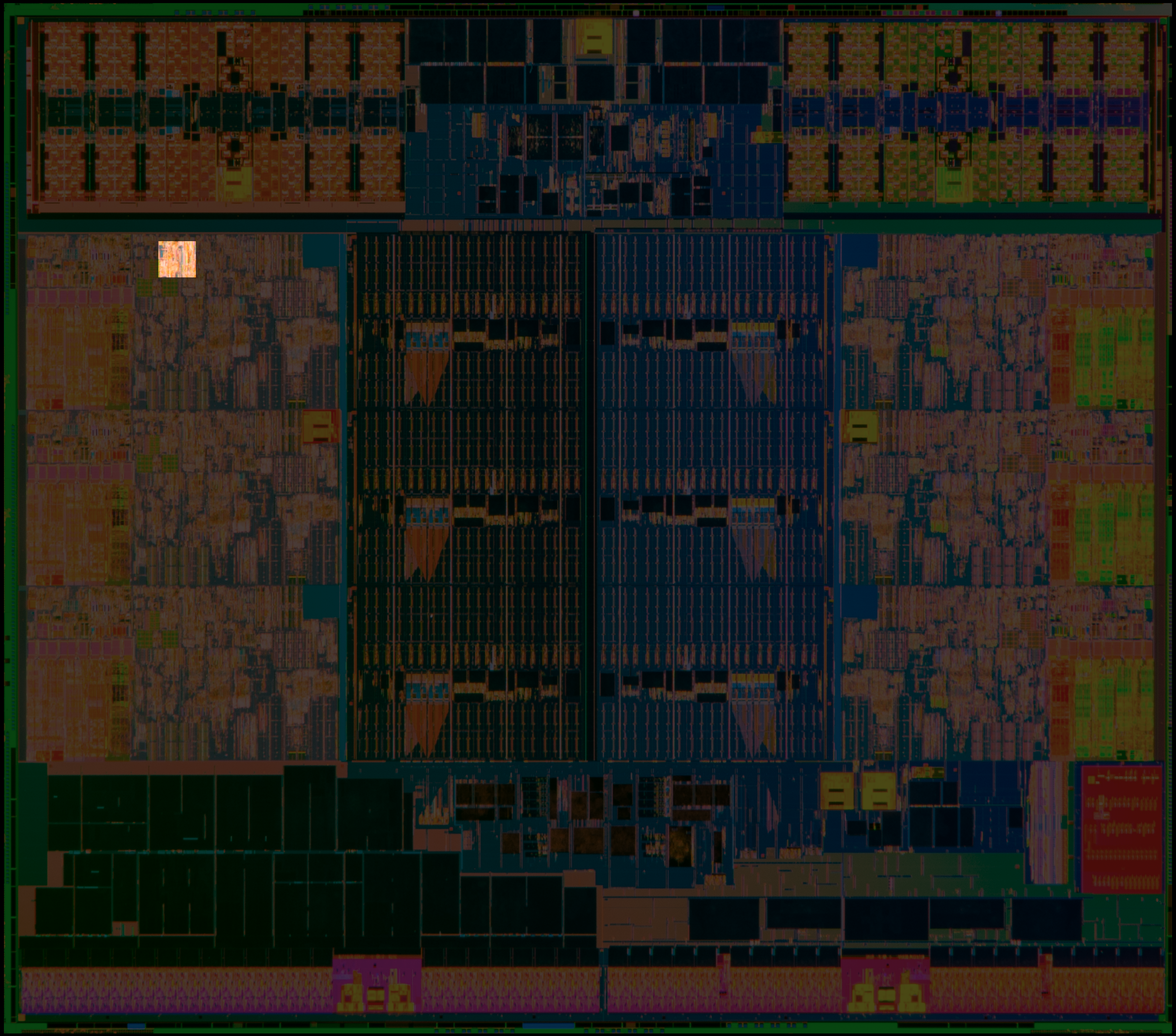
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



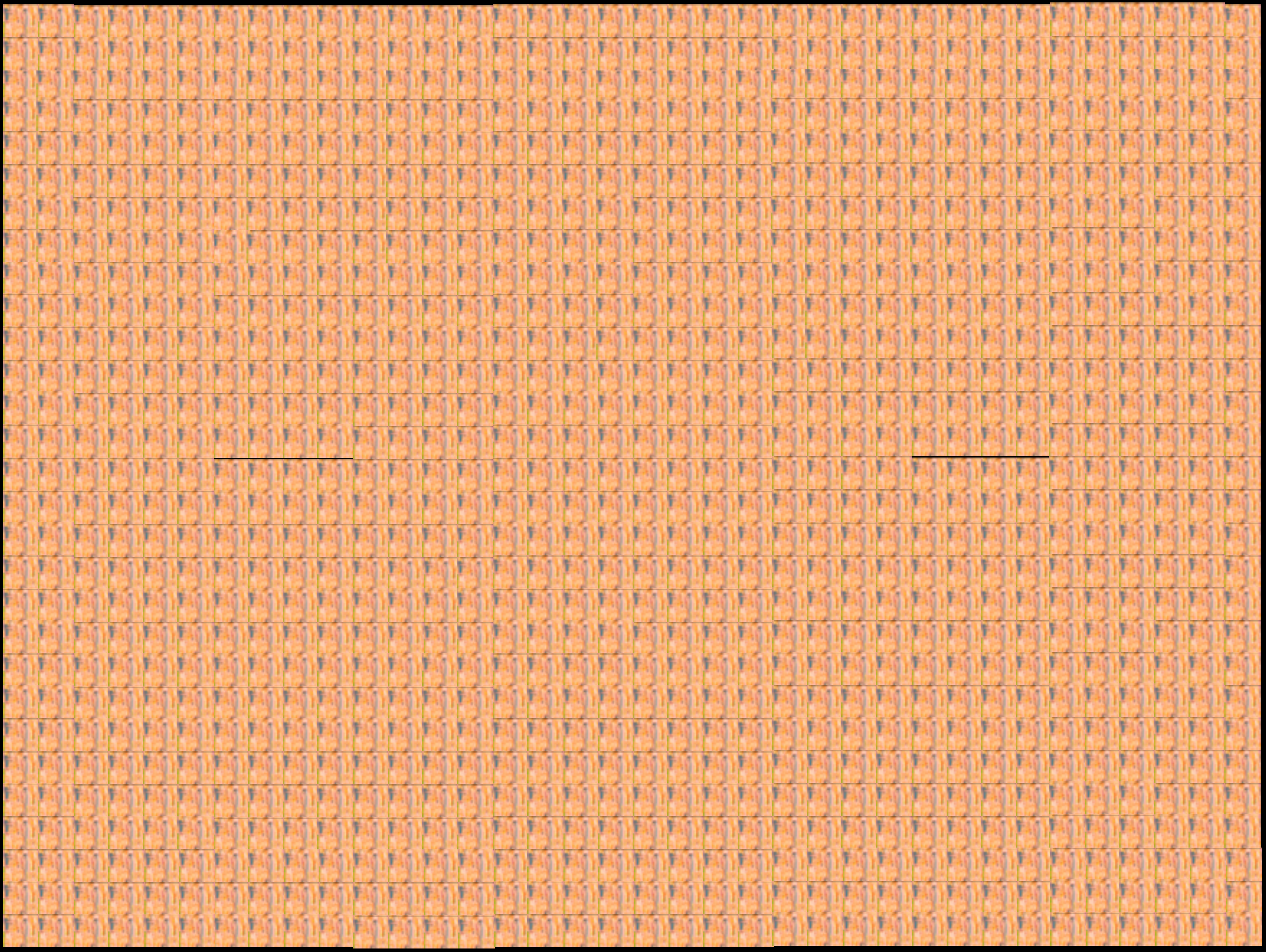
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg

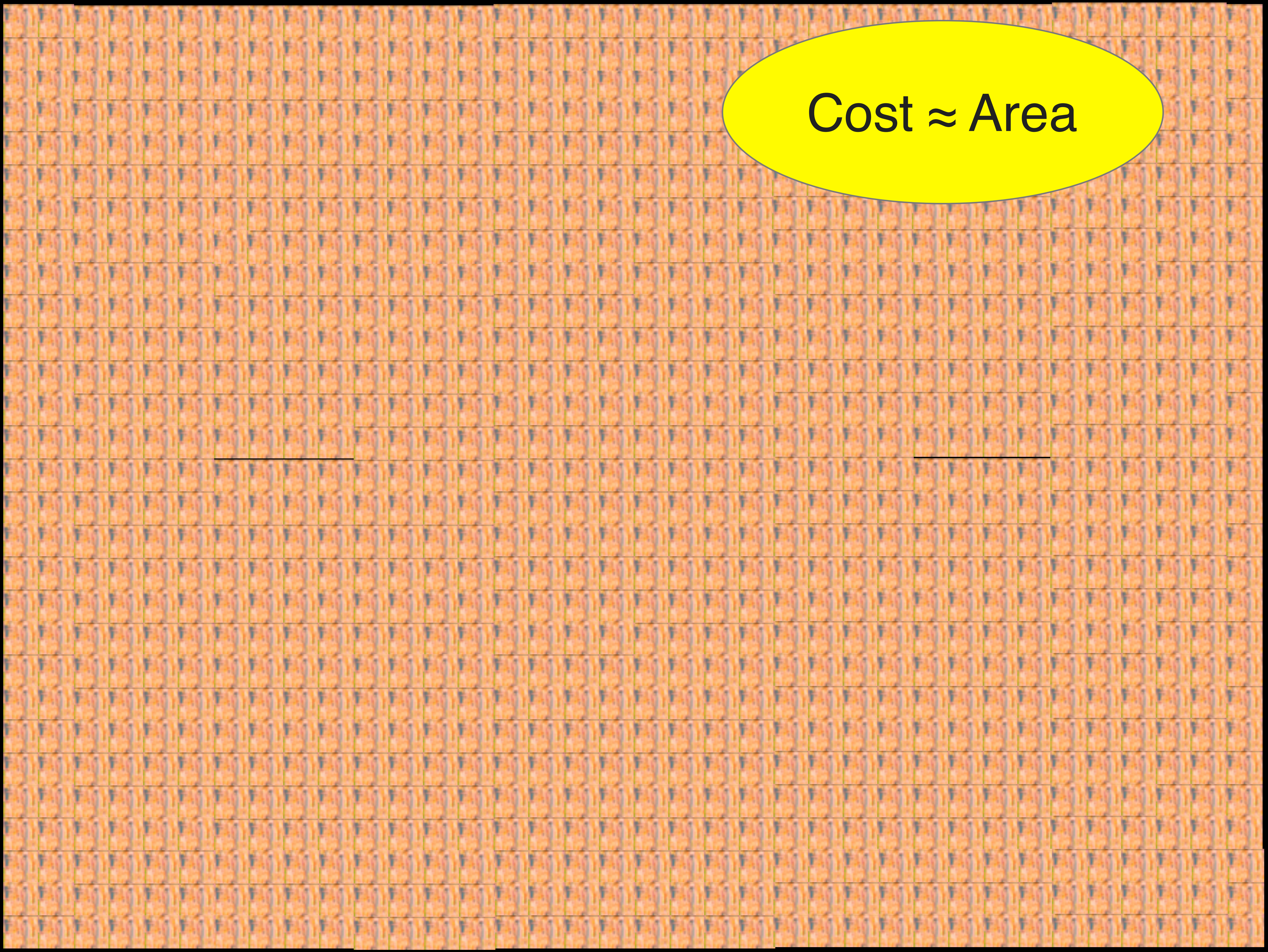


Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg

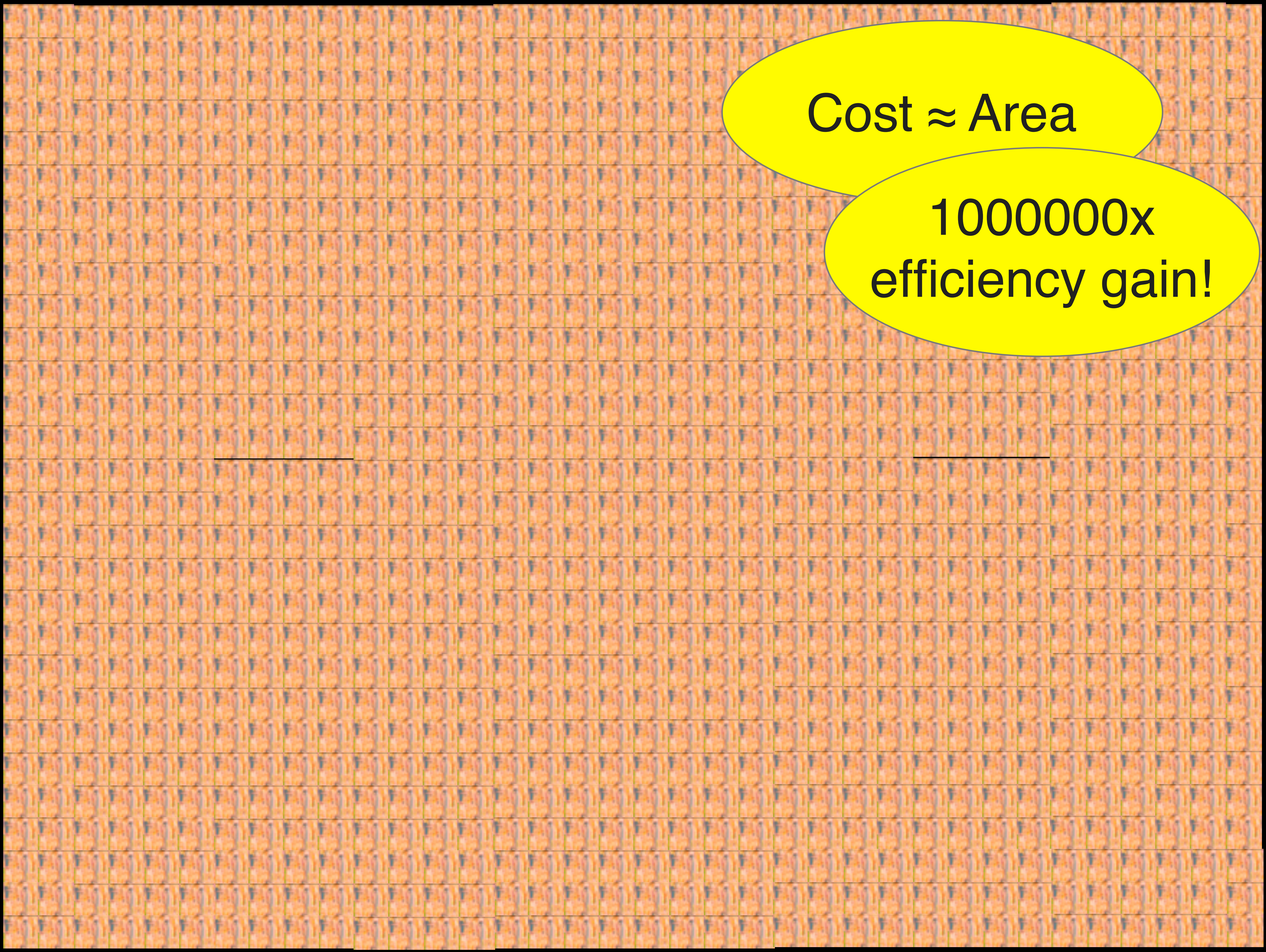


Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg



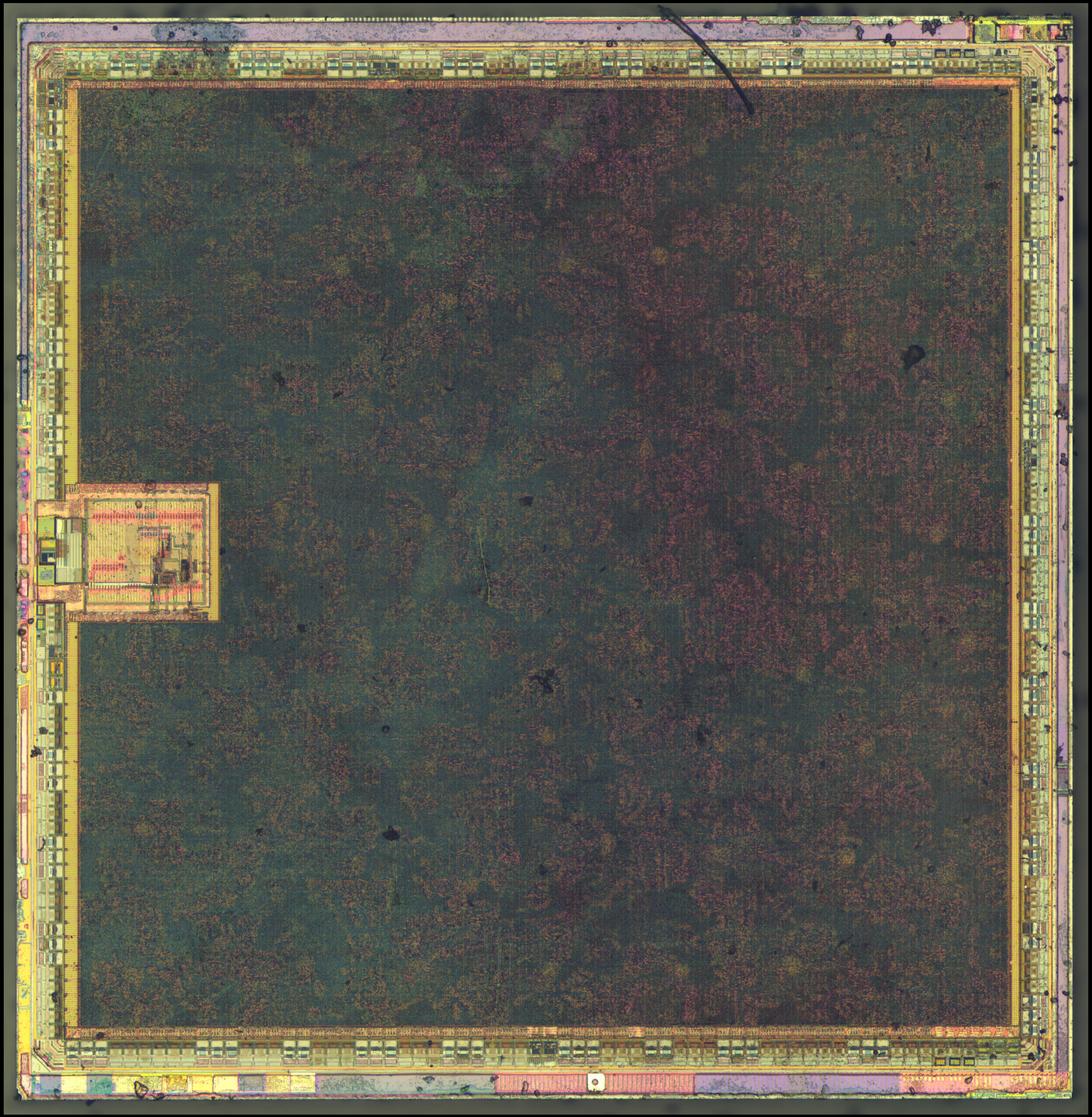


Cost ≈ Area



Cost ≈ Area

1000000x
efficiency gain!



“Avalon”
http://zeptobars.ru/en/read/avalon-bitcoin-mining-unit-rig

Memory-Hardness

Memory-Hardness

Memory-hard functions use a large amount of working space during their computation
→ Attacker must keep caches on chip
→ Decreases the advantage of special-purpose HW
[Reinhold 1999], [Dwork, Goldberg, Naor 2003], [Abadi et al. 2005], [Percival 2009]

Memory-Hardness

Memory-hard functions use a large amount of working space during their computation
→ Attacker must keep caches on chip
→ Decreases the advantage of special-purpose HW
[Reinhold 1999], [Dwork, Goldberg, Naor 2003], [Abadi et al. 2005], [Percival 2009]

Typical technique:

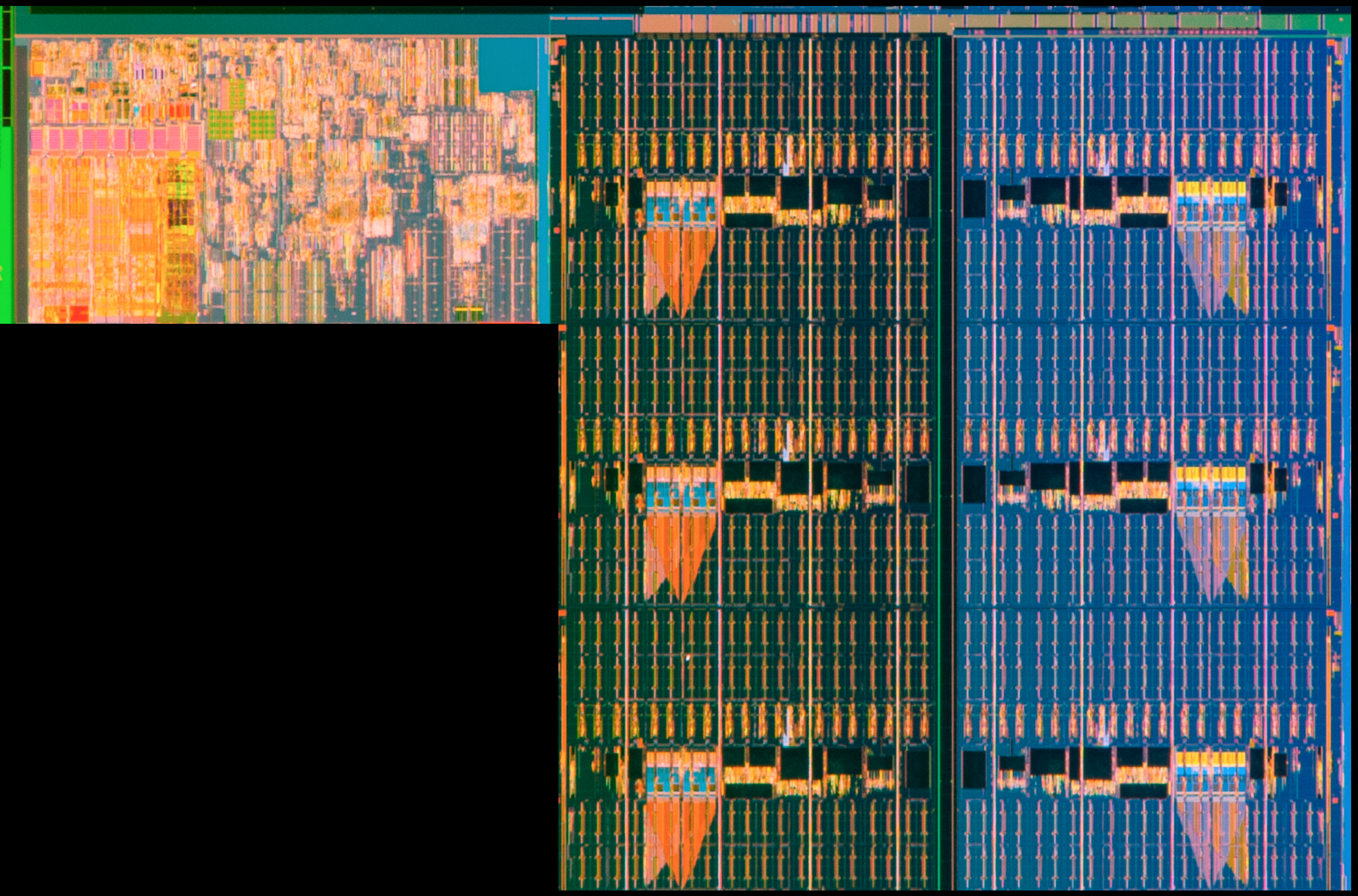
1. **Fill** – fill buffer with pseudo-random bytes
2. **Mix** – read and write pseudo-random blocks in buffer
3. **Extract** – extract function output from buffer contents

Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg

Without memory-hardness

Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg

Without memory-hardness



With memory-hardness

Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Plan

I. **Background on password hashing**

II. Goals

III. The Balloon algorithm

IV. Discussion

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Goal 1: Memory-Hardness

Random oracles: [Bellare & Rogaway 1993]

Memory-hard functions: [Abadi et al. 2005] [Percival 2009]

Goal 1: Memory-Hardness

Informally, a memory-hard function, with hardness parameter N, requires space *S* and time *T* to compute, where

$$S \cdot T \in \Omega(N^2)$$

in the random-oracle model.

Random oracles: [Bellare & Rogaway 1993]
Memory-hard functions: [Abadi et al. 2005] [Percival 2009]

Goal 1: Memory-Hardness

Informally, a memory-hard function, with hardness parameter N, requires space S and time T to compute, where

$$S \cdot T \in \Omega(N^2)$$

in the random-oracle model.

Intuition: any adversary who tries to save space will pay a large penalty in computation time.

Random oracles: [Bellare & Rogaway 1993]
Memory-hard functions: [Abadi et al. 2005] [Percival 2009]

Goal 2: Side-Channel Resistance

Goal 2: Side-Channel Resistance

- The memory access pattern should not leak information about the password being hashed
  [Tsunoo *et al.* 2003] [Bernstein 2005] [Bonneau & Mironov 2006] […]

Goal 2: Side-Channel Resistance

- The memory access pattern should not leak information about the password being hashed
  [Tsunoo *et al.* 2003] [Bernstein 2005] [Bonneau & Mironov 2006] […]

Goal 3: Real-World Practical

Goal 2: Side-Channel Resistance

- The memory access pattern should not leak information about the password being hashed
  [Tsunoo *et al.* 2003] [Bernstein 2005] [Bonneau & Mironov 2006] […]

Goal 3: Real-World Practical

- The hash should be able to support hundreds of logins per second while filling L2 cache (or more)

Existing Schemes

Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]

Not memory hard

Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]

Not memory hard

scrypt [Percival 2009]

Password-dependent memory access pattern

Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]

Not memory hard

scrypt [Percival 2009]

Password-dependent memory access pattern

Parallel-secure schemes [Alwen & Serbinenko 2015] [Alwen, Blocki, Pietrzak 2016]

May be impractical for realistic parameter sizes

Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]

Not memory hard

scrypt [Percival 2009]

Password-dependent memory access pattern

Parallel-secure schemes [Alwen & Serbinenko 2015] [Alwen, Blocki, Pietrzak 2016]

May be impractical for realistic parameter sizes

Argon2i and Catena [Biryukov et al. 2015] [Forler et al. 2015]

Lack formal security analysis

Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]

Not memory hard

scrypt [Percival 2009]

Password-dependent memory access pattern

Parallel-secure schem

May be impractical fo es

We demonstrate a practical attack against Argon2i

Argon2i and Catena [Biryukov et al. 2015] [Forler et al. 2015]

Lack formal security analysis

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Balloon Hashing

A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions

Balloon Hashing

A password hashing function that:

1. Is **proven** memory-hard (in the sequential setting)
2. Uses a password-independent data access pattern
3. Matches the performance of the best heuristically secure memory-hard functions

```
Balloon(password, salt, N = space_cost, R = num_rounds):
    $\delta \leftarrow 3$                  // A security parameter.
    var $B_1, \dots, B_N$            // A buffer of N blocks.

    // Step 1: Fill Buffer
    $B_1 \leftarrow$ Hash(password, salt)
    for i = 2, …, N:
        $B_i \leftarrow$ Hash($B_{i-1}$)

    // Step 2: Mix Buffer
    for r = 1, …, R:
        for i = 1,…, N:
            // Chosen pseudorandomly from salt
            $(v_1, \dots, v_\delta) \leftarrow$ Hash( salt, r, i ) $\in Z_N^\delta$
            $B_i \leftarrow$ Hash($B_{(i-1 \bmod N)}, B_i, B_{v_1}, \dots, B_{v_\delta}$)

    // Step 3: Extract
    return $B_N$
```

```
Balloon(password, salt, N = space_cost, R = num_rounds):
    $\delta \leftarrow 3$                  // A security parameter.
    var $B_1, \ldots, B_N$          // A buffer of N blocks.

    // Step 1: Fill Buffer
    $B_1 \leftarrow$ Hash(password, salt)
    for i = 2, …, N:
        $B_i \leftarrow$ Hash($B_{i-1}$)

    // Step 2: Mix Buffer
    for r = 1, …, R:
        for i = 1,…, N:
            // Chosen pseudorandomly from salt
            $(v_1, \ldots, v_\delta) \leftarrow$ Hash( salt, r, i ) $\in Z_N^\delta$
            $B_i \leftarrow$ Hash($B_{(i-1 \bmod N)}, B_i, B_{v_1}, \ldots, B_{v_\delta}$)

    // Step 3: Extract
    return $B_N$
```

A conventional hash function (e.g., SHA-256)

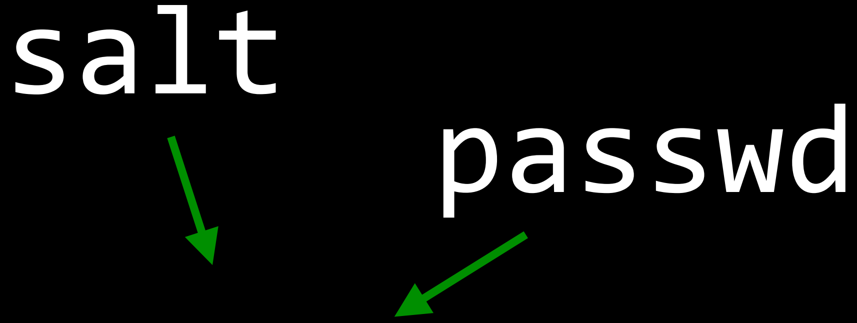
```
Balloon(password, salt, N = space_cost, R = num_rounds):
    $\delta \leftarrow 3$                   // A security parameter.
    var $B_1, \ldots, B_N$          // A buffer of N blocks.

    // Step 1: Fill Buffer
    $B_1 \leftarrow$ Hash(password, salt)
    for i = 2, …, N:
        $B_i \leftarrow$ Hash($B_{i-1}$)

    // Step 2: Mix Buffer
    for r = 1, …, R:
        for i = 1,…, N:
            // Chosen pseudorandomly from salt
            $(v_1, \ldots, v_\delta) \leftarrow$ Hash( salt, r, i ) $\in Z_N^\delta$
            $B_i \leftarrow$ Hash($B_{(i-1 \bmod N)}, B_i, B_{v_1}, \ldots, B_{v_\delta}$)

    // Step 3: Extract
    return $B_N$
```

Balloon Hashing Algorithm

Balloon Hashing Algorithm

```
salt
         passwd
```

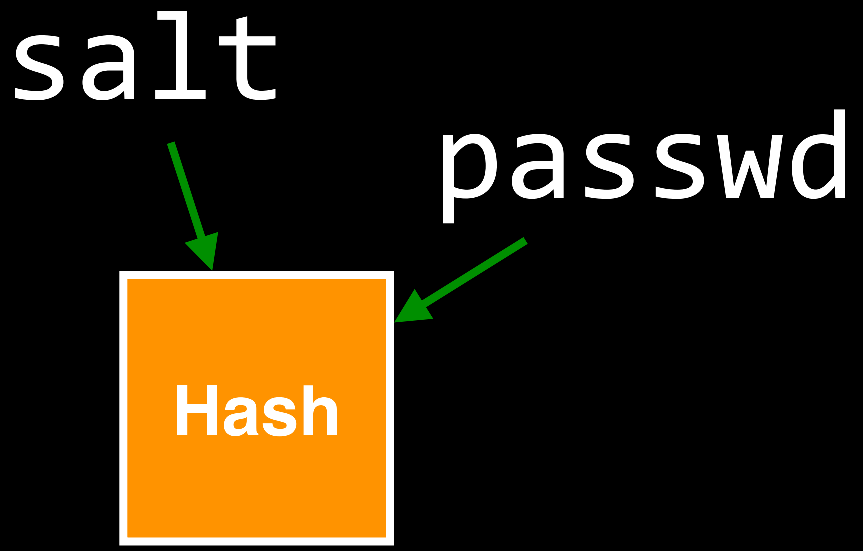
Balloon Hashing Algorithm



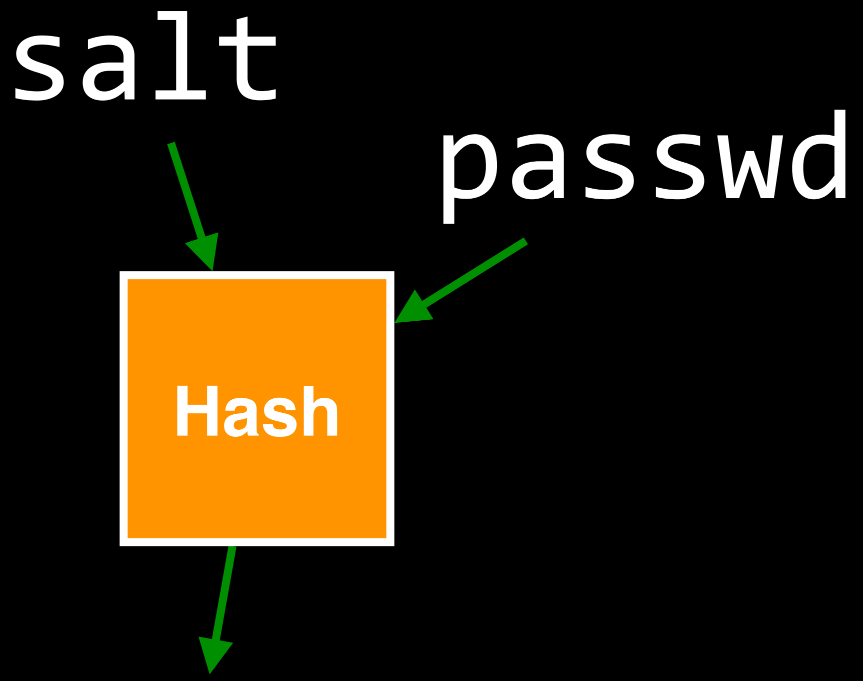
salt

passwd

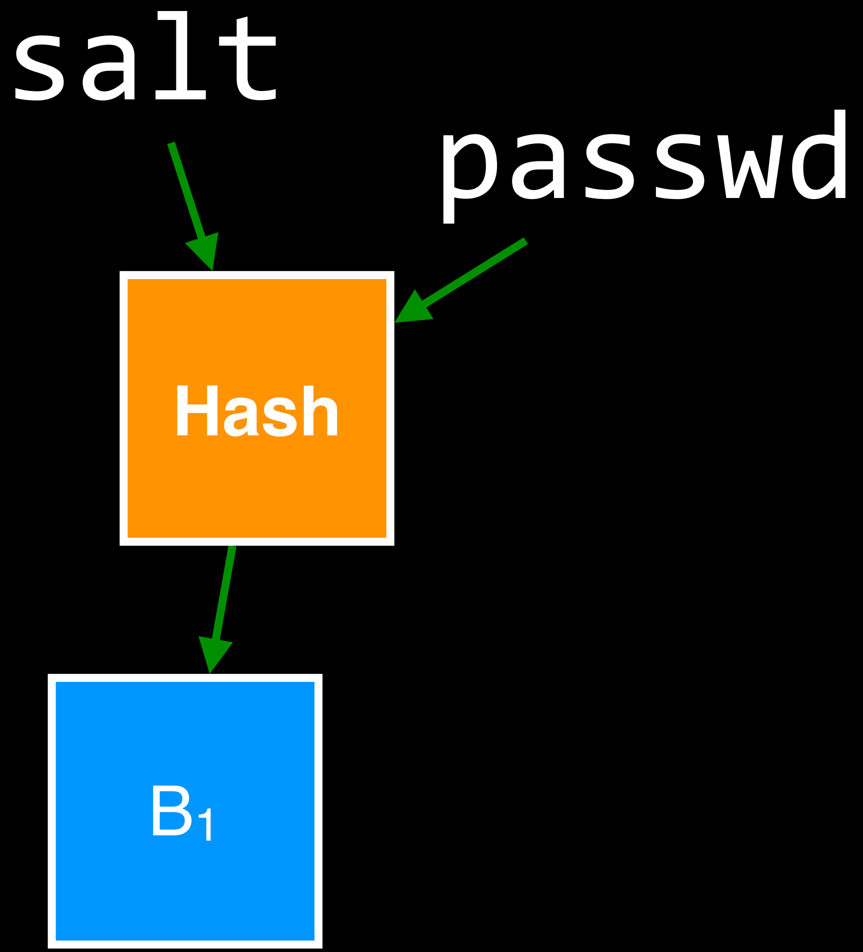
Balloon Hashing Algorithm



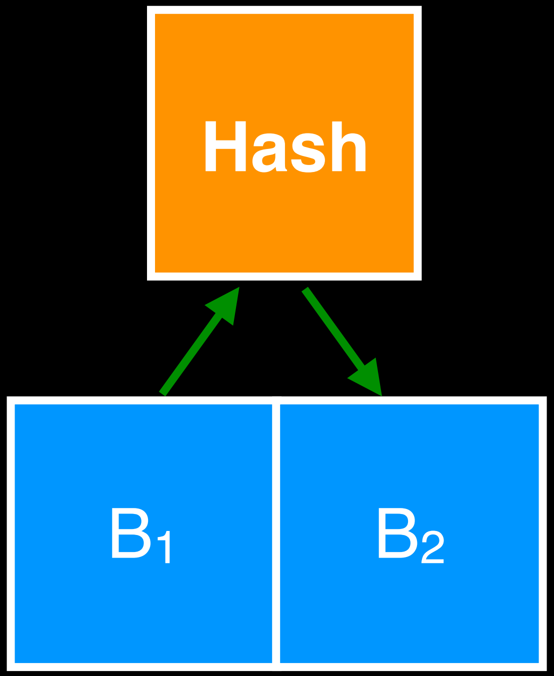
Balloon Hashing Algorithm



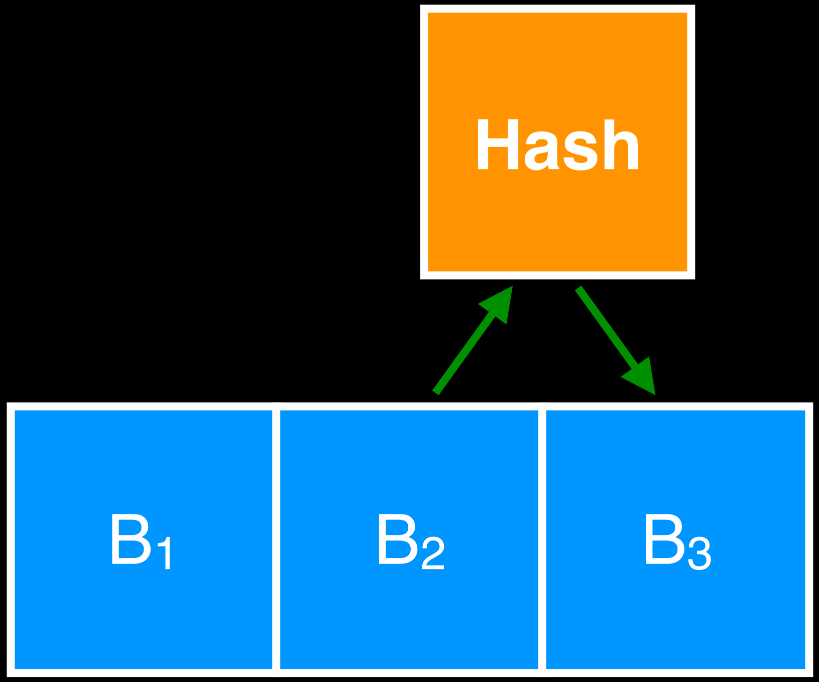
Balloon Hashing Algorithm



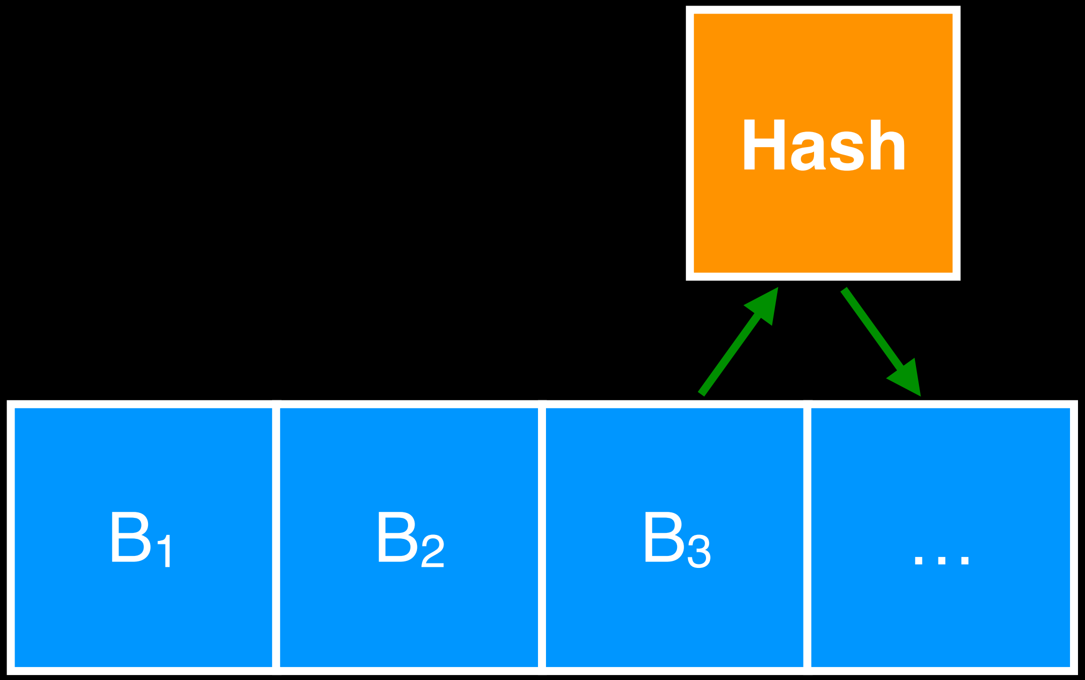
Balloon Hashing Algorithm



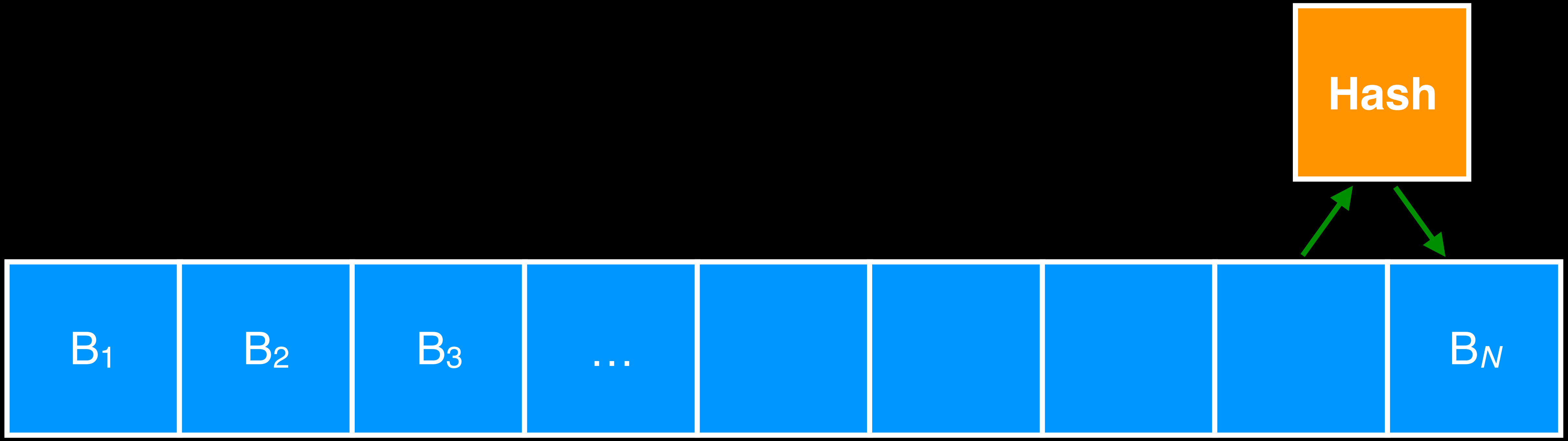
Balloon Hashing Algorithm



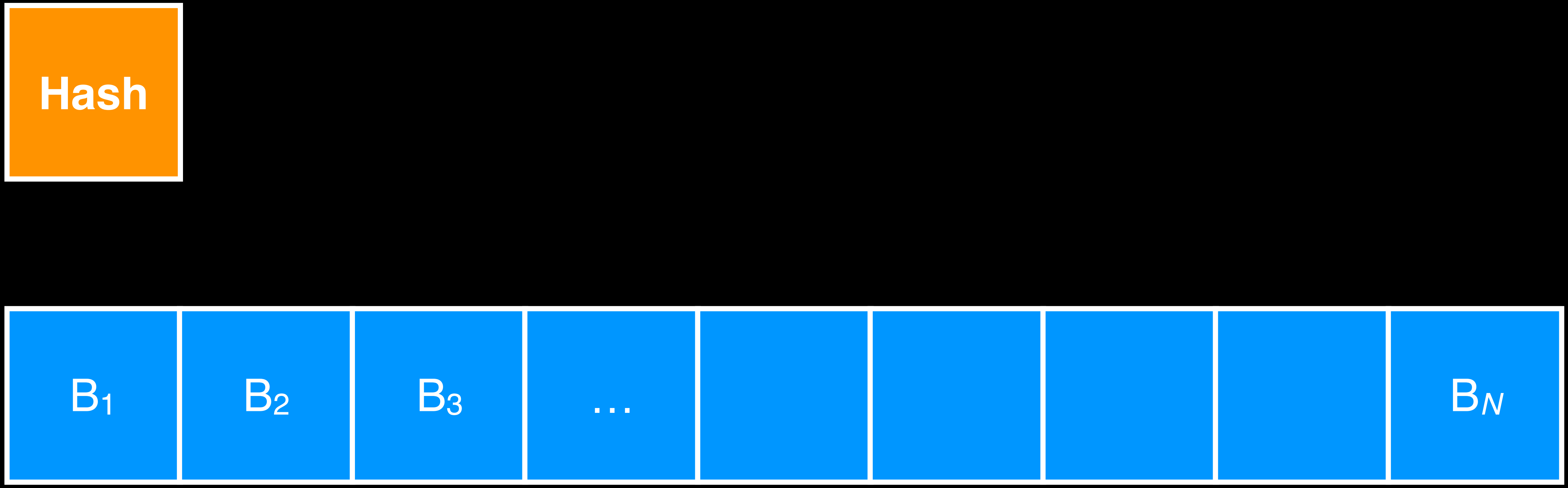
Balloon Hashing Algorithm



Balloon Hashing Algorithm



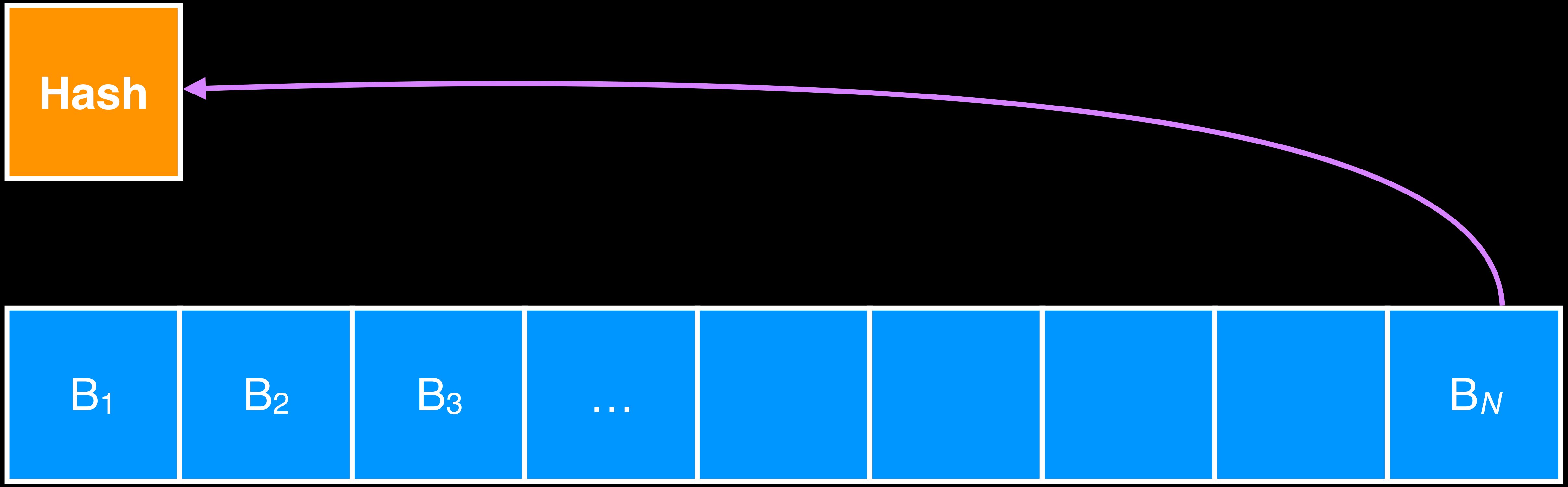
Balloon Hashing Algorithm



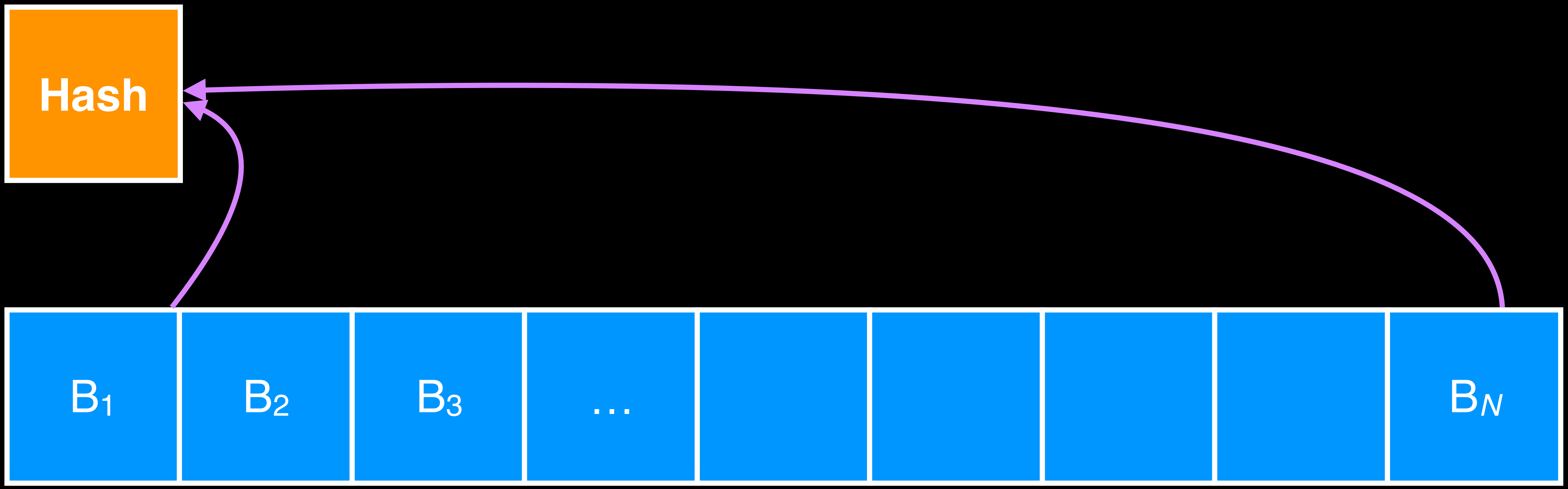
Hash

B_1 B_2 B_3 … B_N

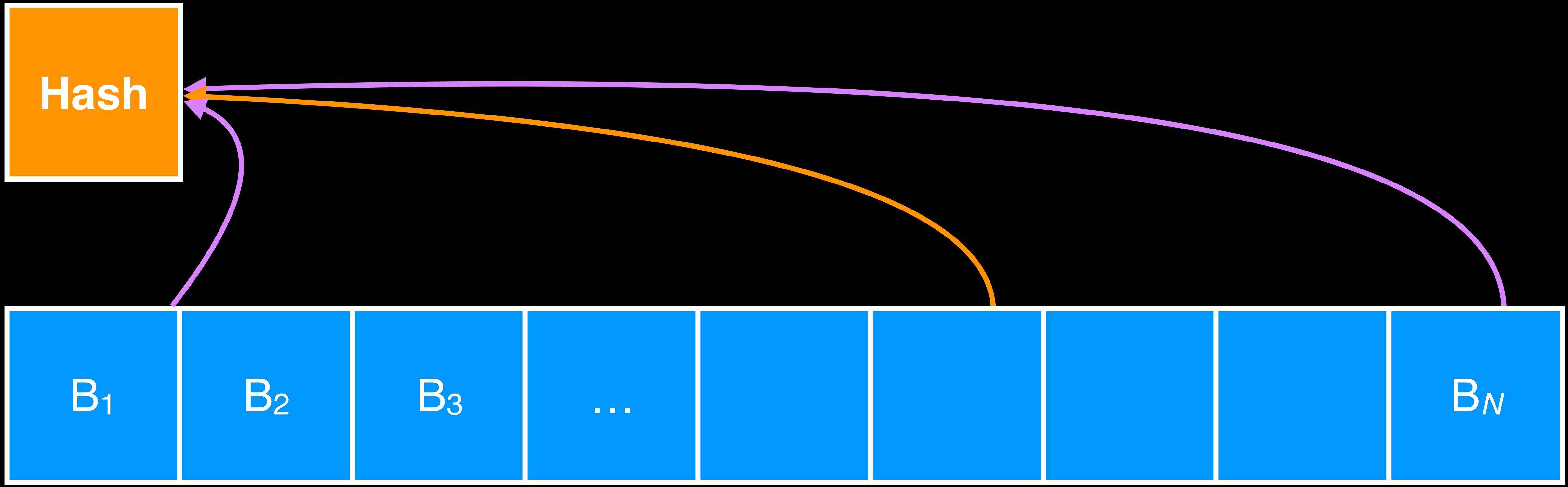
Balloon Hashing Algorithm



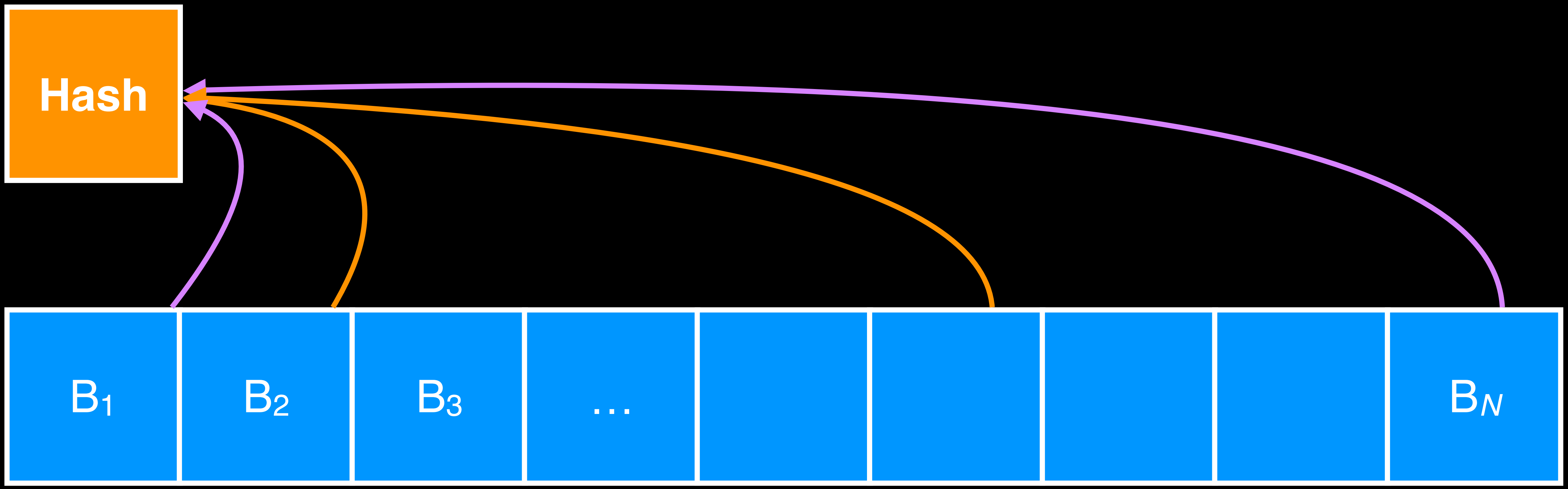
Balloon Hashing Algorithm



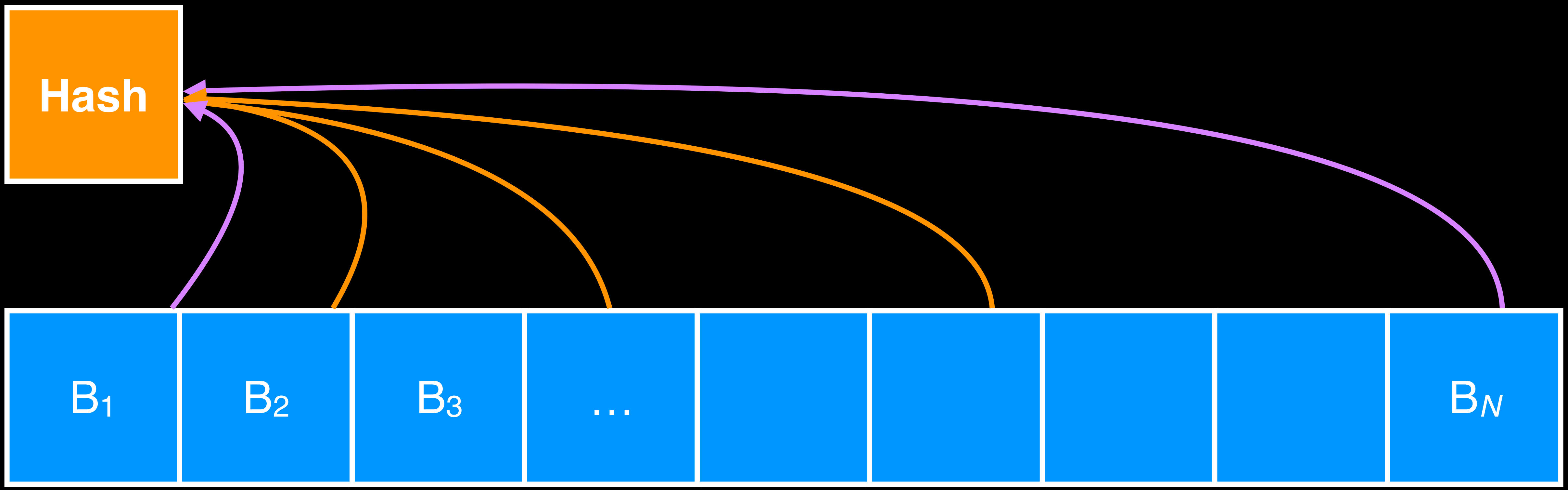
Balloon Hashing Algorithm



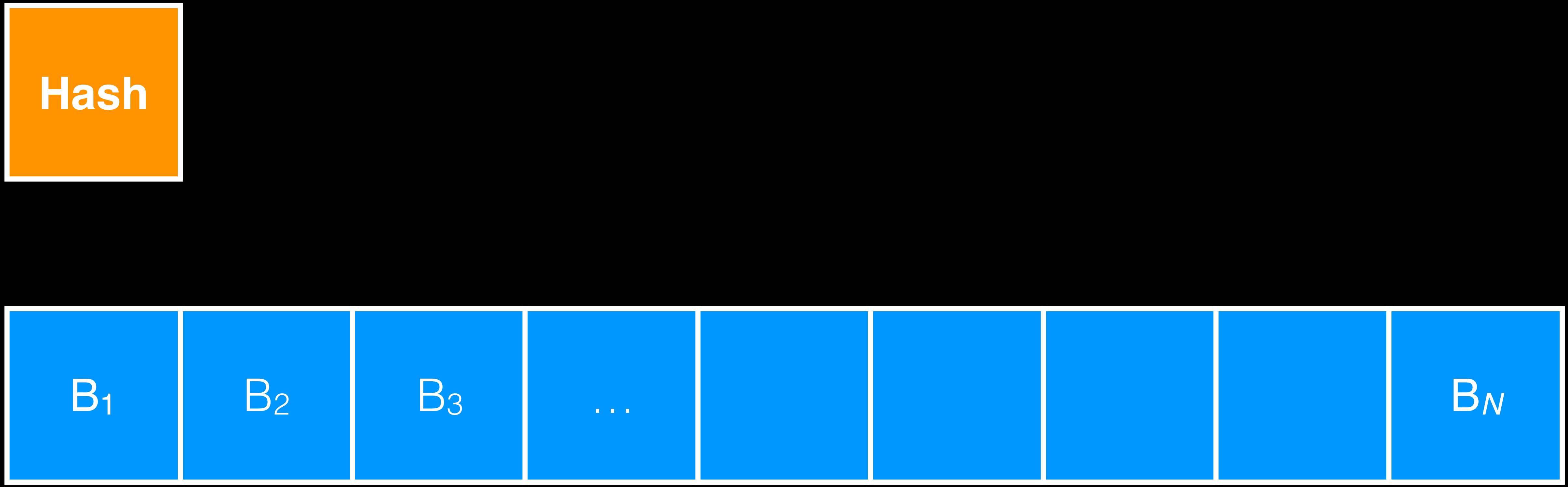
Balloon Hashing Algorithm



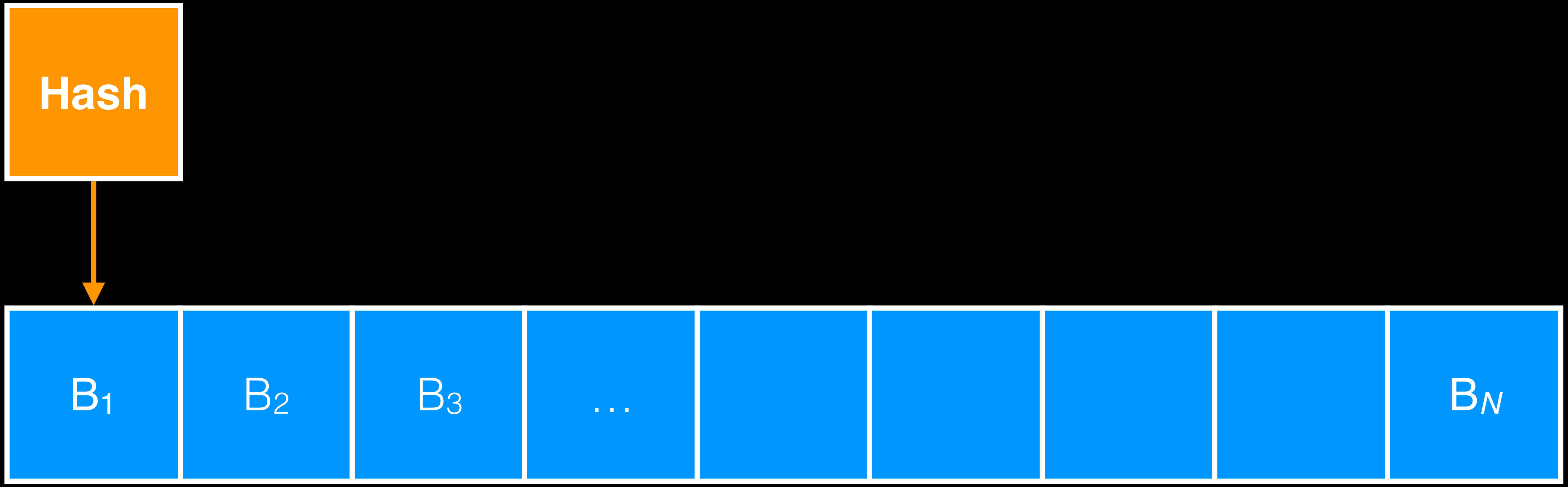
Balloon Hashing Algorithm



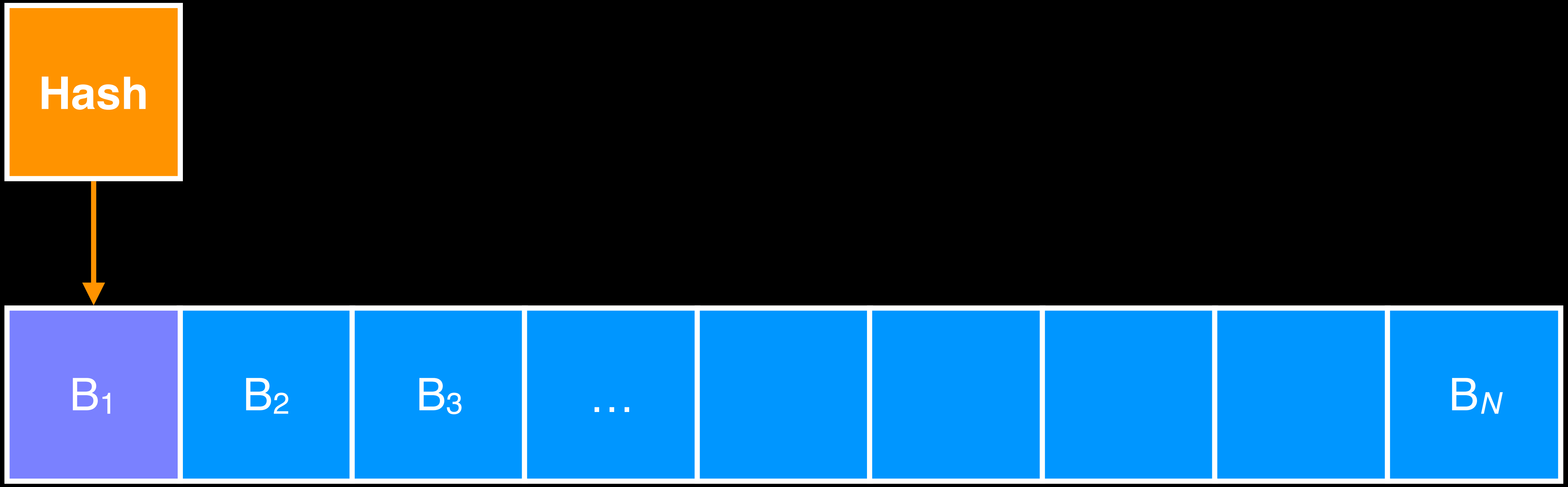
Balloon Hashing Algorithm



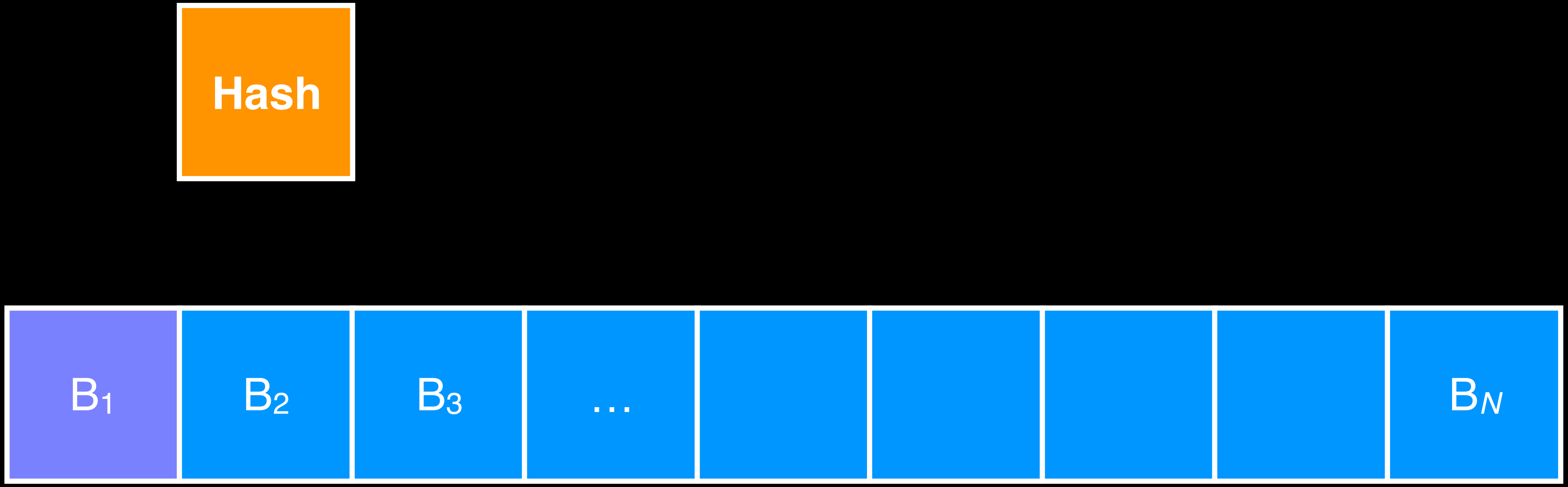
Balloon Hashing Algorithm



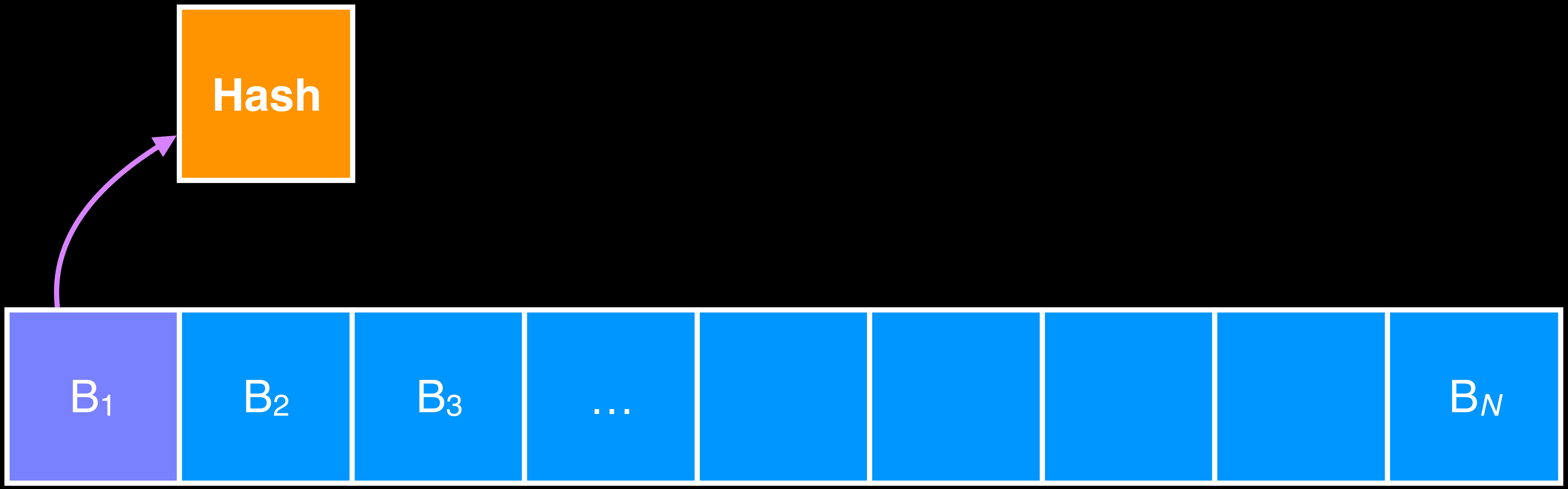
Balloon Hashing Algorithm



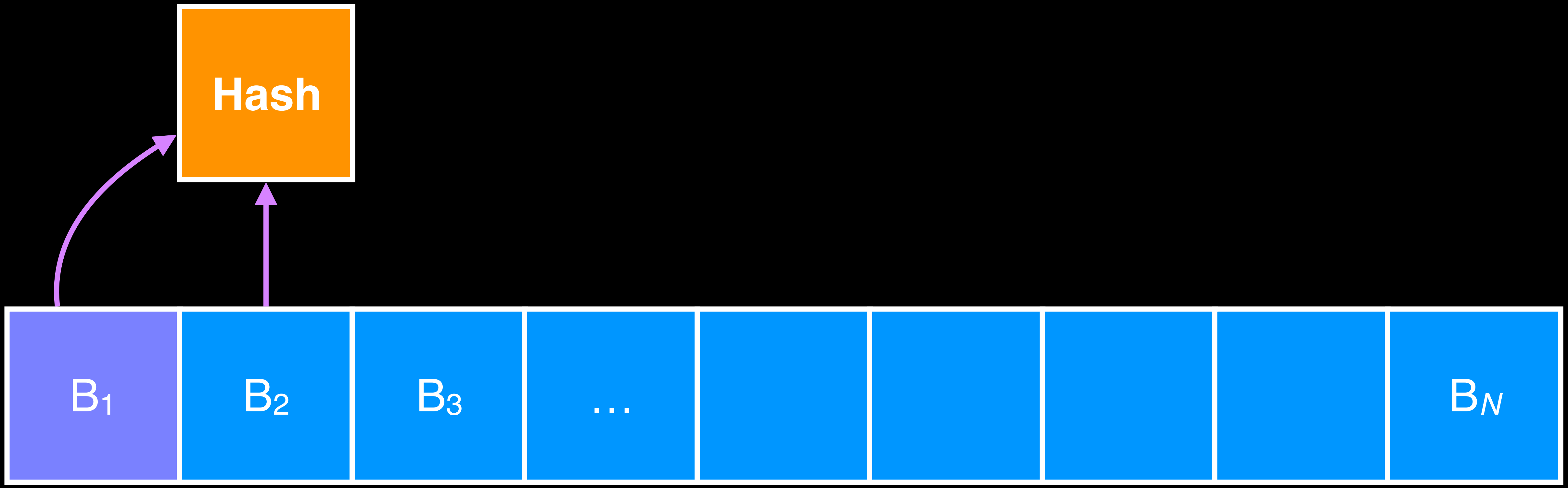
Balloon Hashing Algorithm



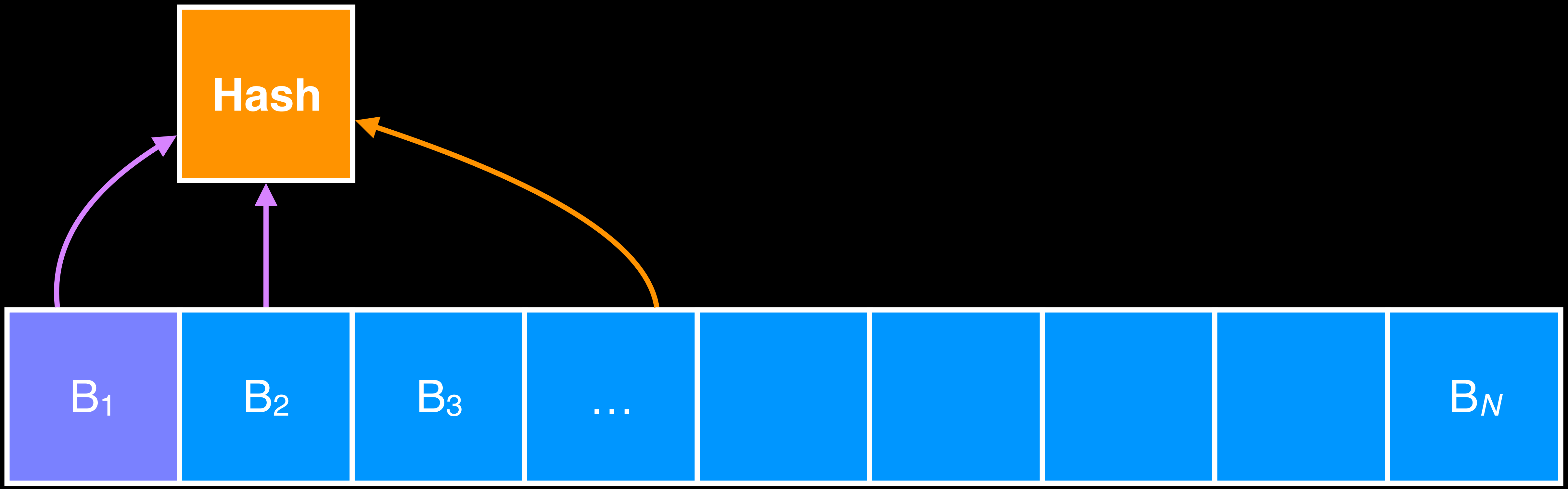
Balloon Hashing Algorithm



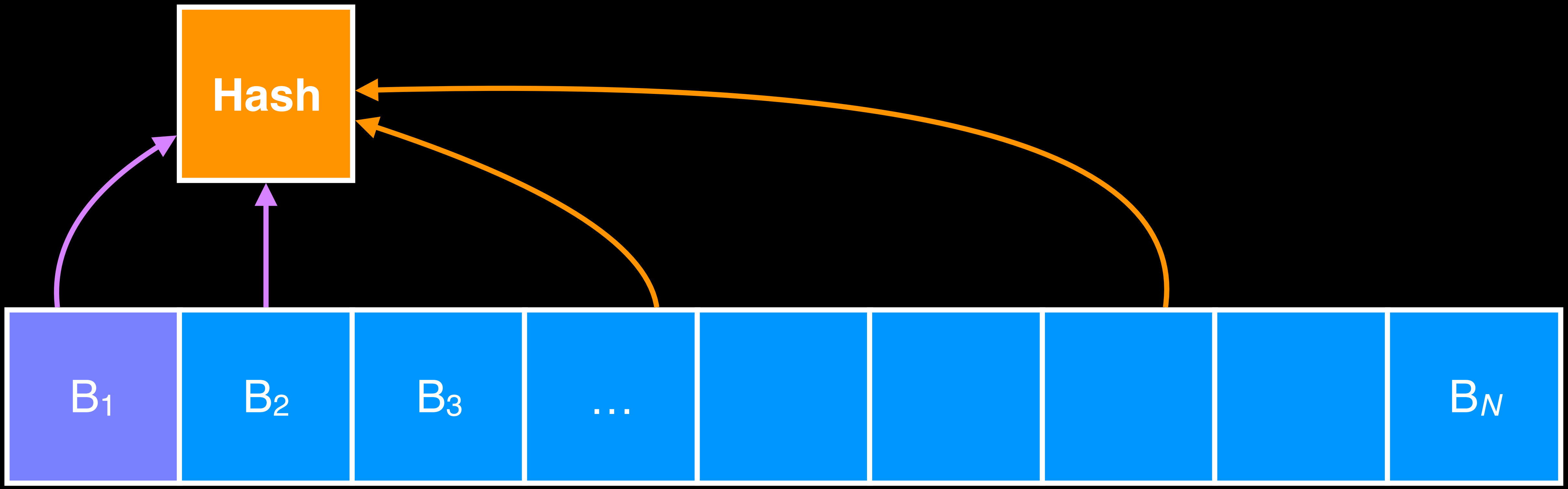
Balloon Hashing Algorithm



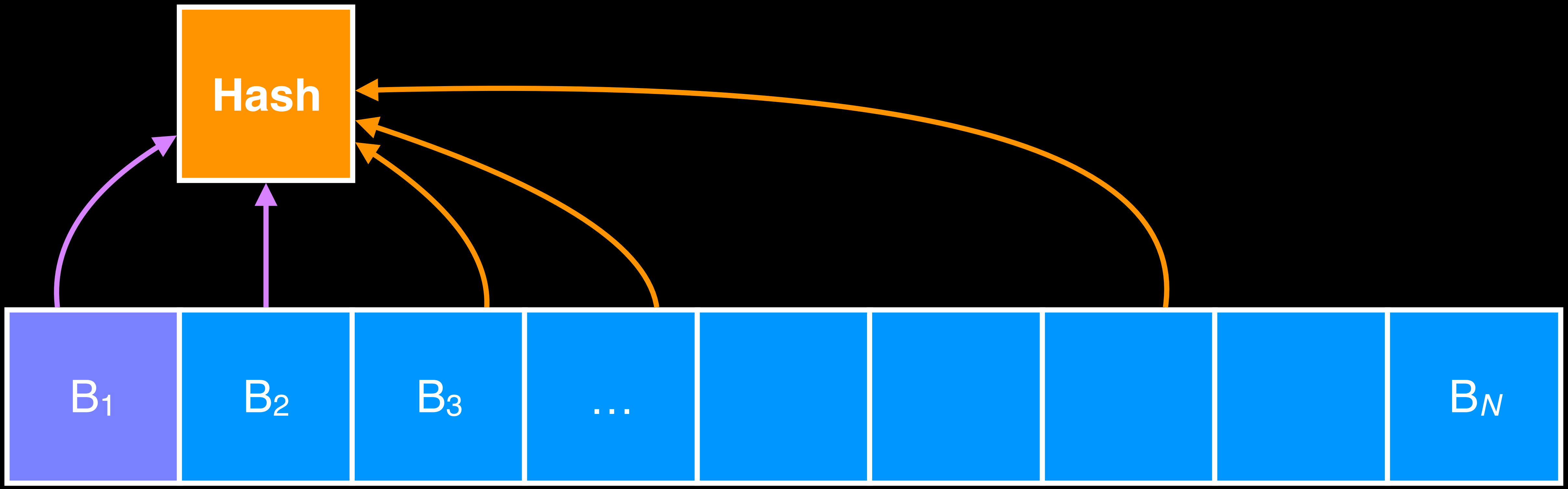
Balloon Hashing Algorithm



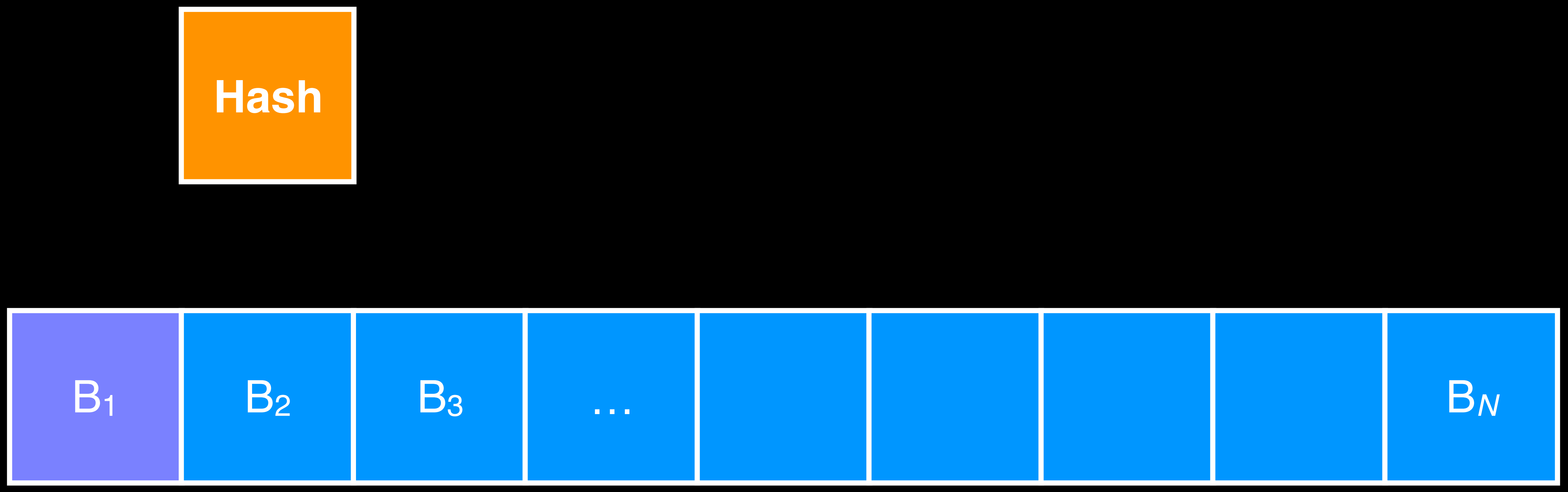
Balloon Hashing Algorithm



Balloon Hashing Algorithm



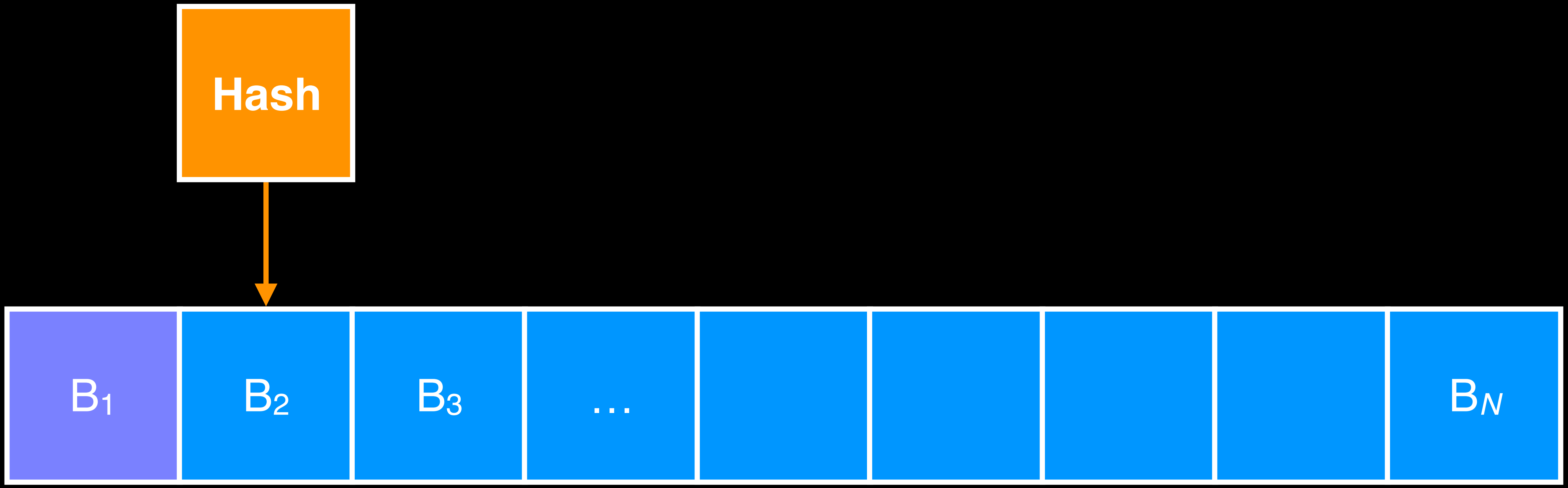
Balloon Hashing Algorithm



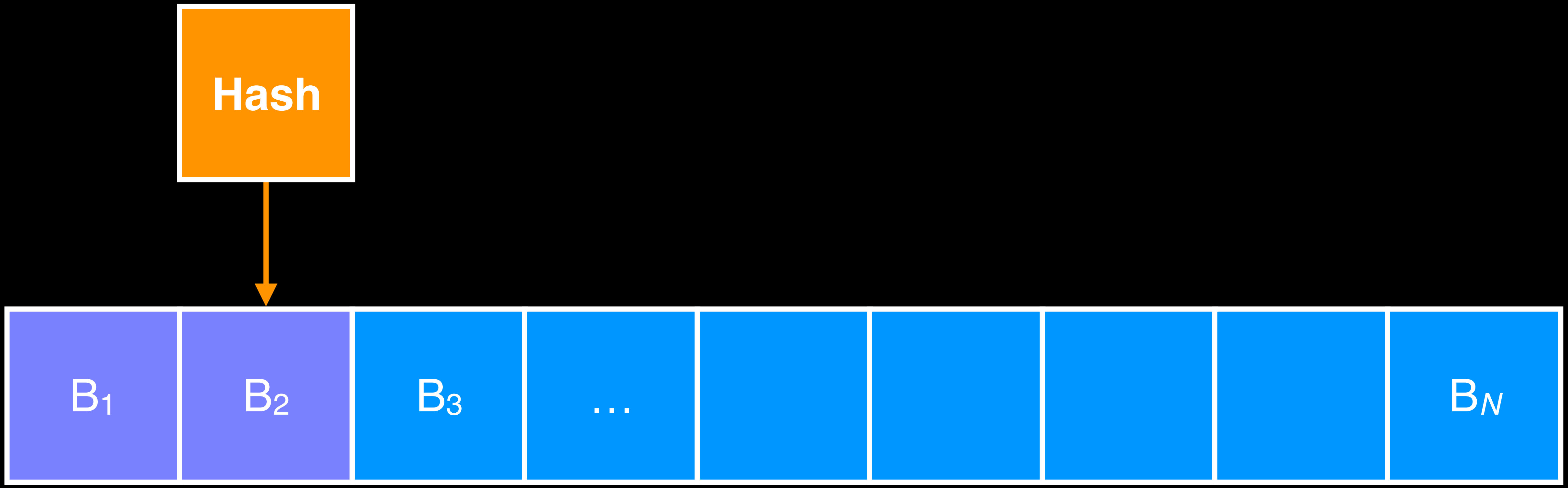
Hash

B_1 B_2 B_3 … B_N

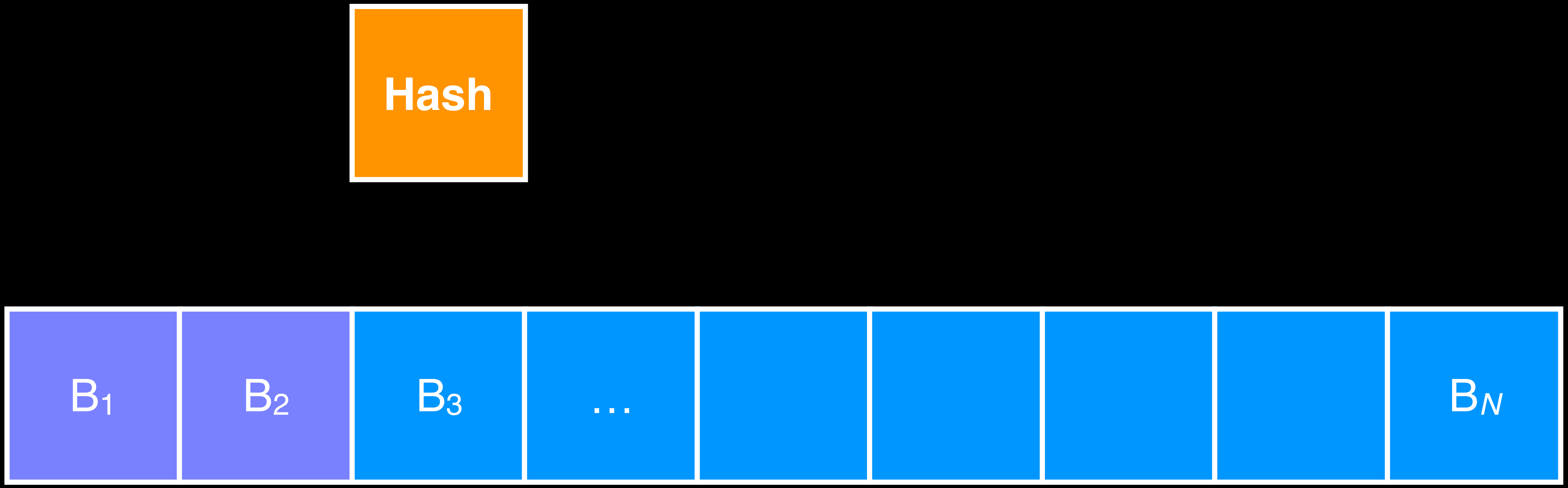
Balloon Hashing Algorithm



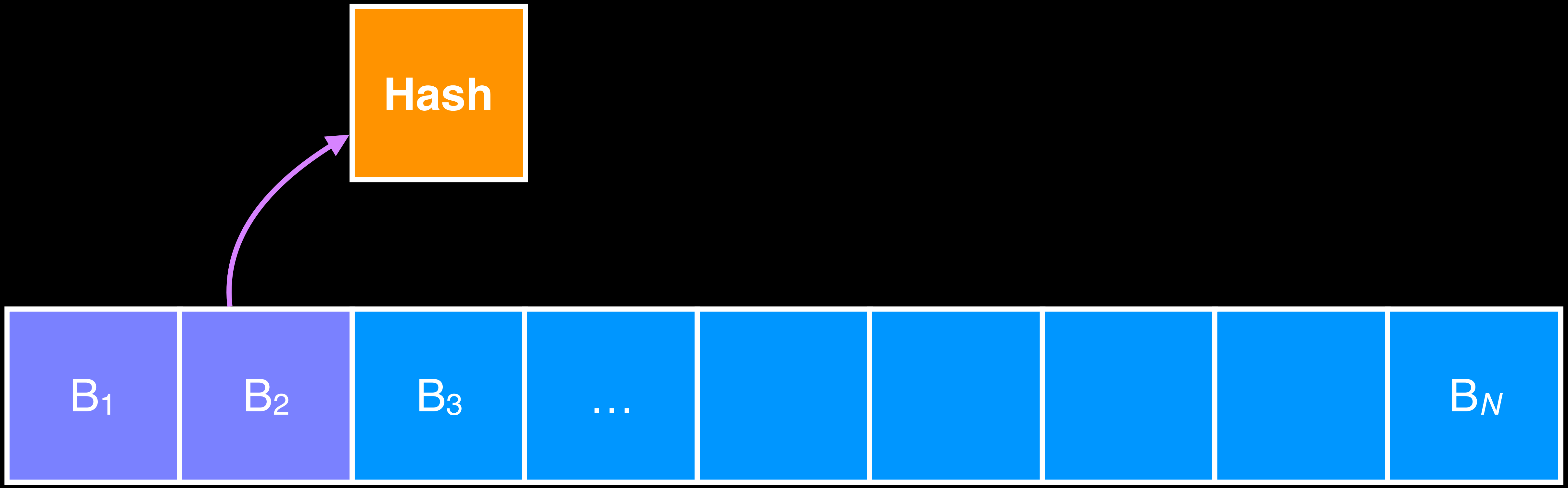
Balloon Hashing Algorithm



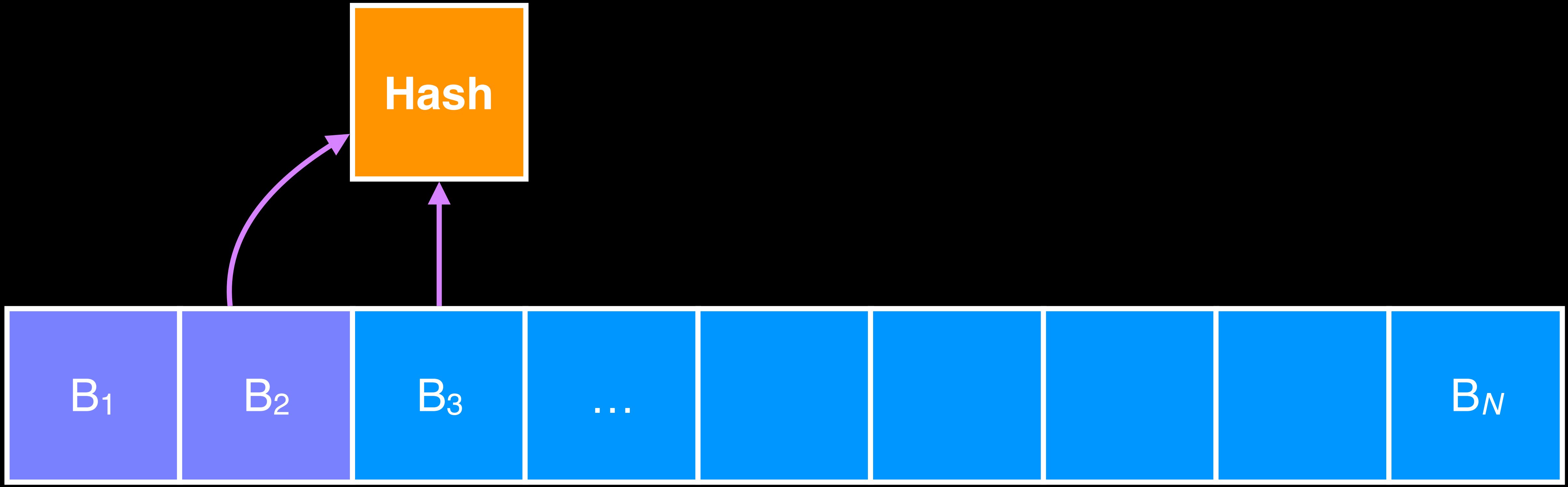
Balloon Hashing Algorithm



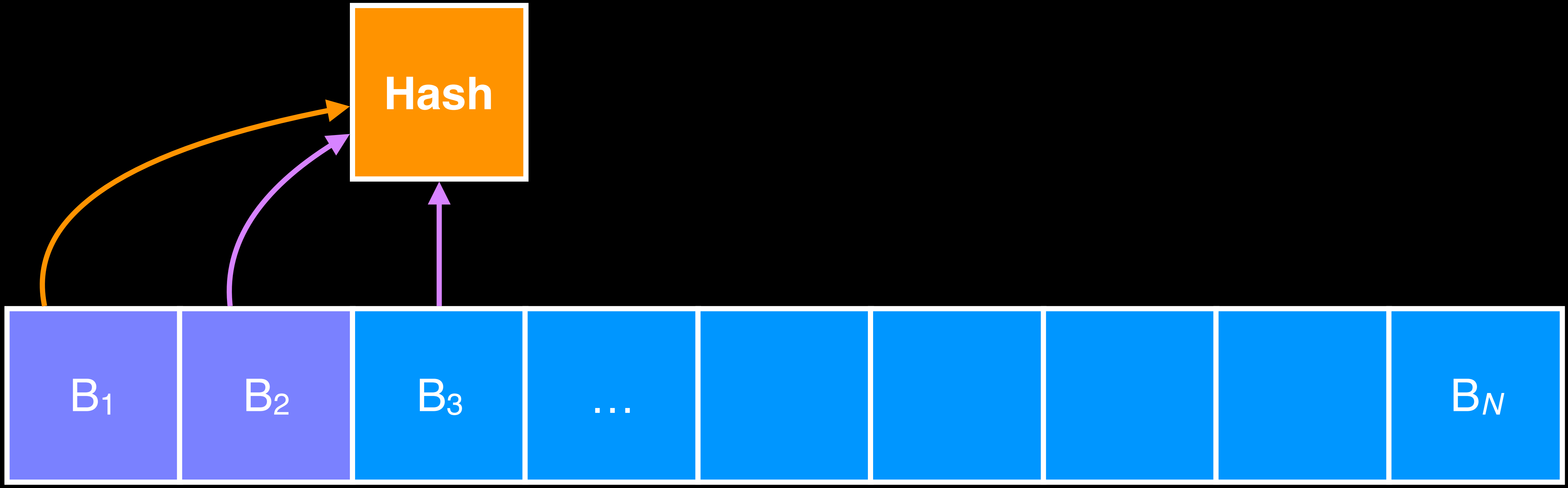
Balloon Hashing Algorithm



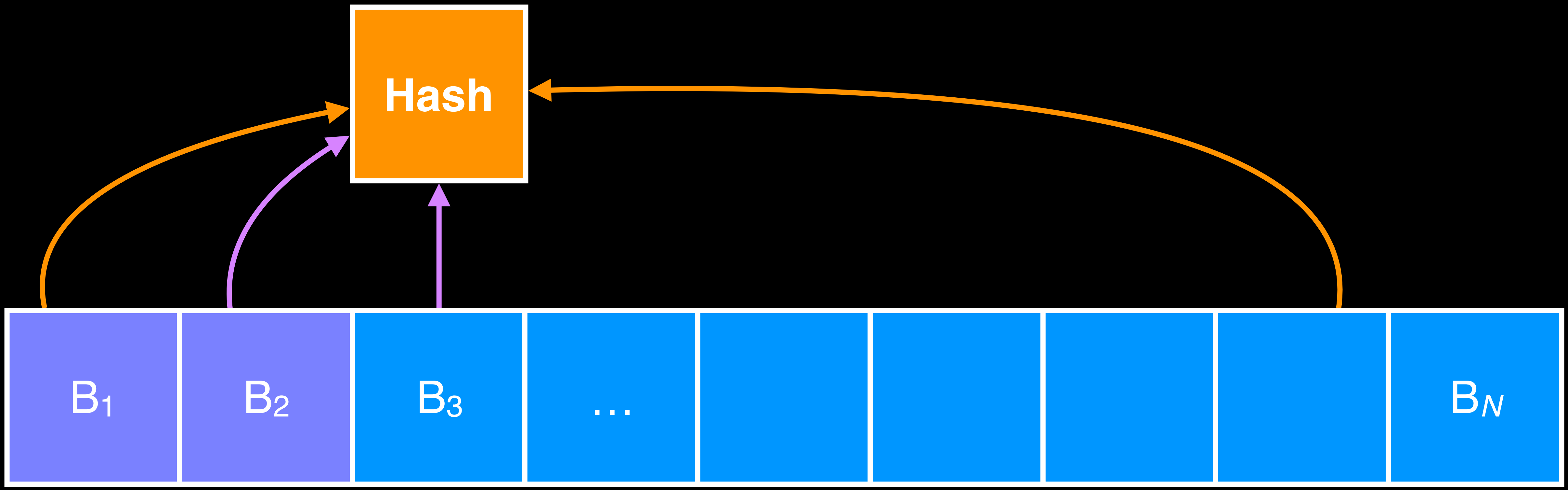
Balloon Hashing Algorithm



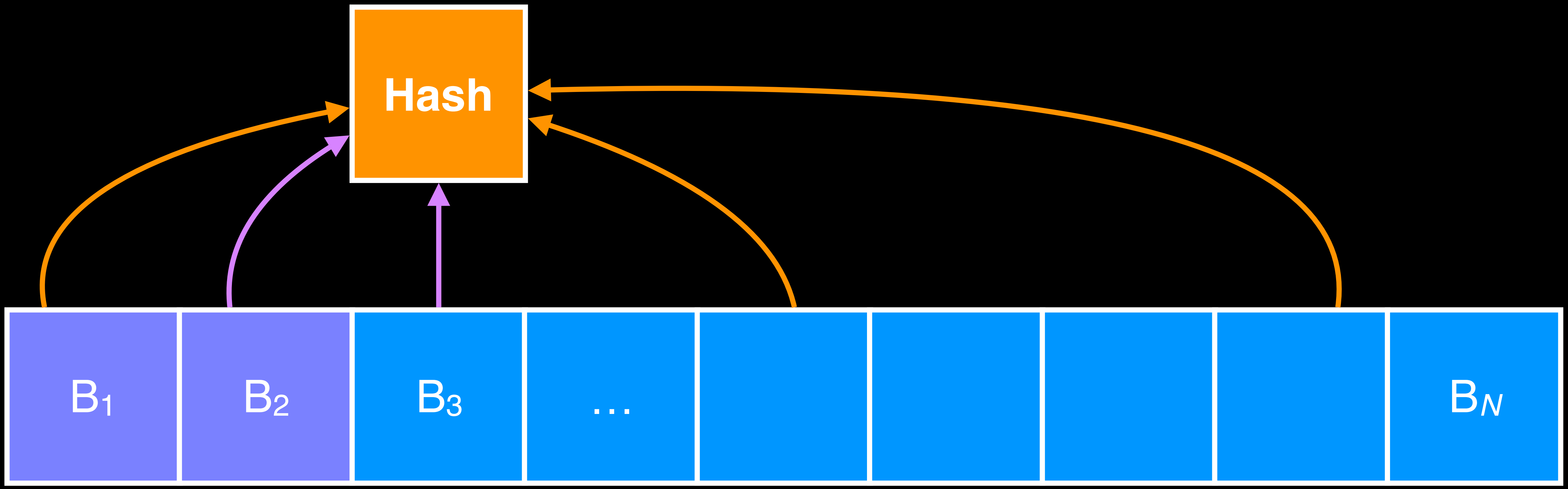
Balloon Hashing Algorithm



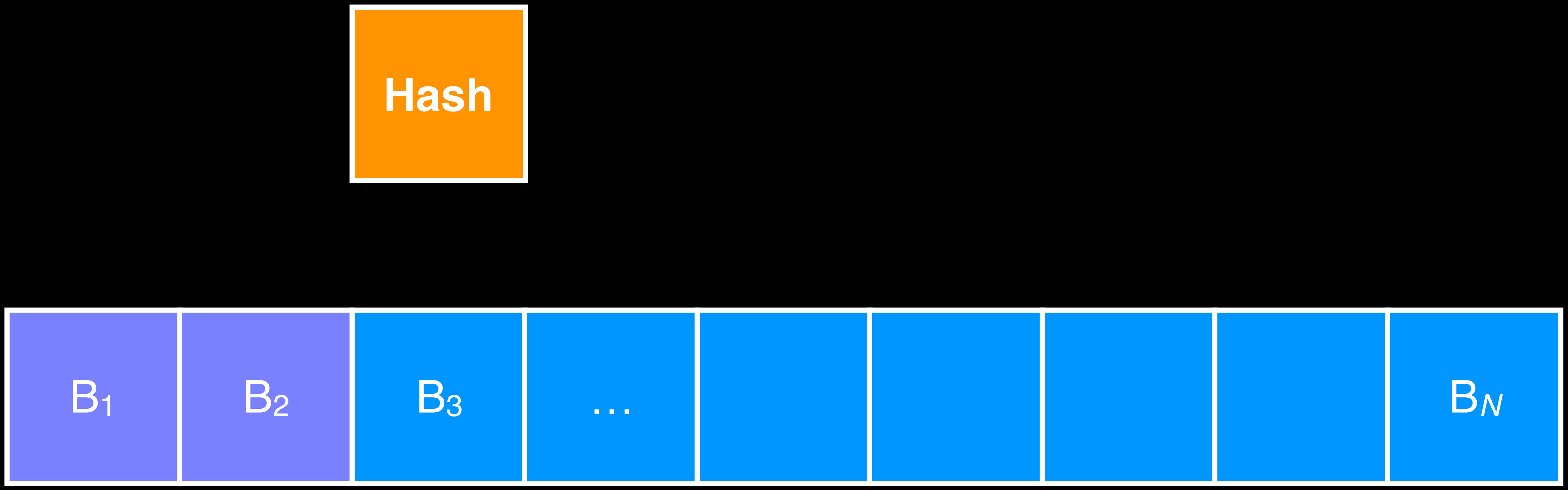
Balloon Hashing Algorithm



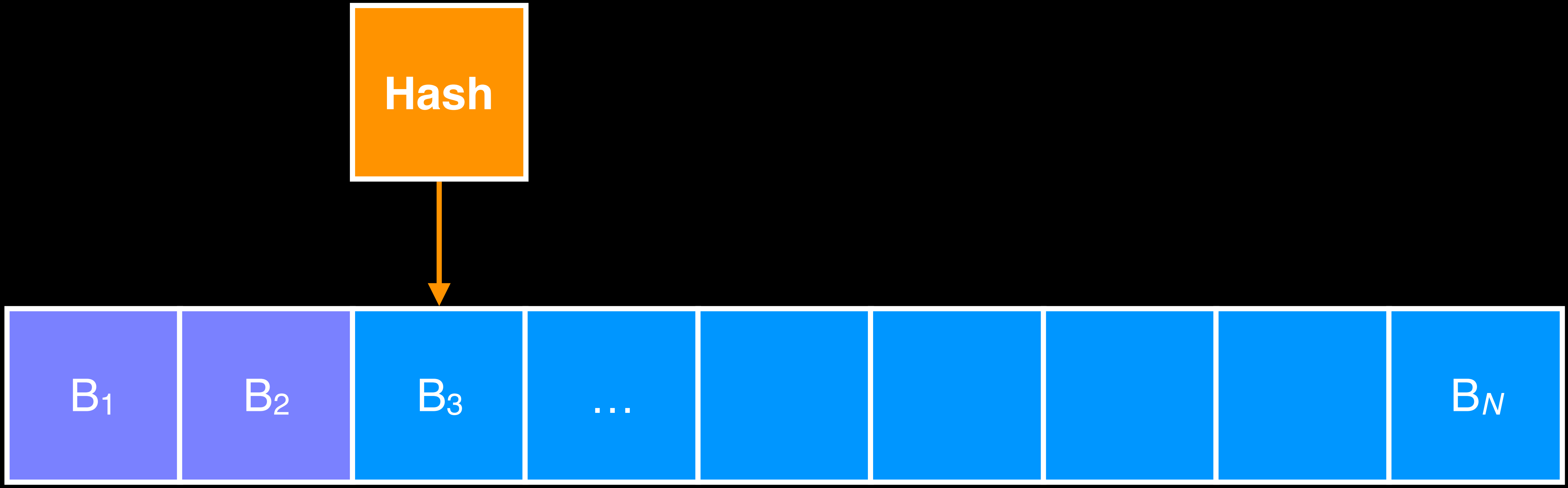
Balloon Hashing Algorithm



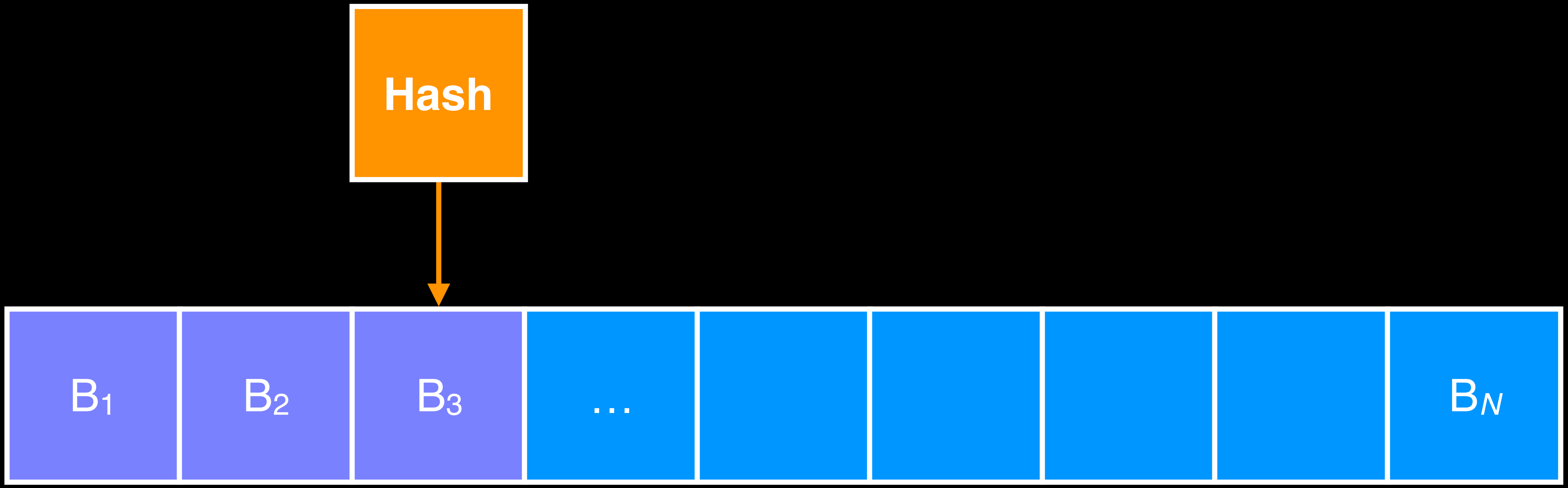
Balloon Hashing Algorithm



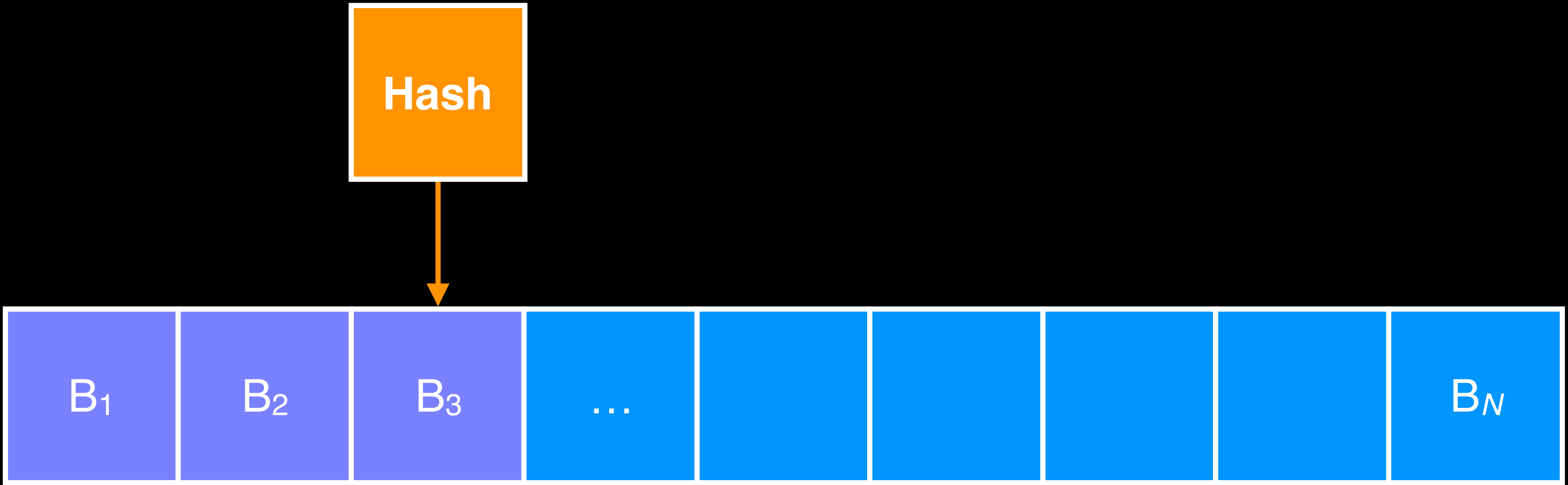
Balloon Hashing Algorithm



Balloon Hashing Algorithm



Balloon Hashing Algorithm



A “mode of operation” for a cryptographic hash function

Balloon Hashing

A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions

Balloon Hashing

A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

✓ 2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions

Balloon Hashing

A password hashing function that:

1. Is proven memory-hard (in the sequential setting)
2. ✓ Uses a password-independent data access pattern
3. ✓ Matches the performance of the best heuristically secure memory-hard functions

Balloon Hashing

A password hashing function th

The challenge

1. Is proven memory-hard (in the sequential setting)
2. ✓ Uses a password-independent data access pattern
3. ✓ Matches the performance of the best heuristically secure memory-hard functions

Proving Memory-Hardness

Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when δ=7, with space **S ≤ N/8** requires time **T** such that

$$\mathbf{S} \cdot \mathbf{T} \geq (2^R - 1) / 8 \cdot \mathbf{N^2} \,.$$

Proving Memory-Hardness

Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when $\delta=7$, with space $\mathbf{S \le N/8}$ requires time $\mathbf{T}$ such that

$$\mathbf{S} \cdot \mathbf{T} \ge (2^R - 1) / 8 \cdot \mathbf{N^2} \,.$$

Saving a factor of 8 in space causes a slowdown **exponential** in # rounds

Proving Memory-Hardness

Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when δ=7, with space **$S \leq N/8$** requires time **T** such that

$$S \cdot T \geq (2^R - 1) / 8 \cdot N^2 .$$

When R=20, using 8× less space requires using **60,000×** more time

Proving Memory-Hardness

Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when δ=7, with space **$S \leq N/8$** requires time **T** such that

$$S \cdot T \geq (2^R - 1) / 8 \cdot N^2 .$$

Proving Memory-Hardness

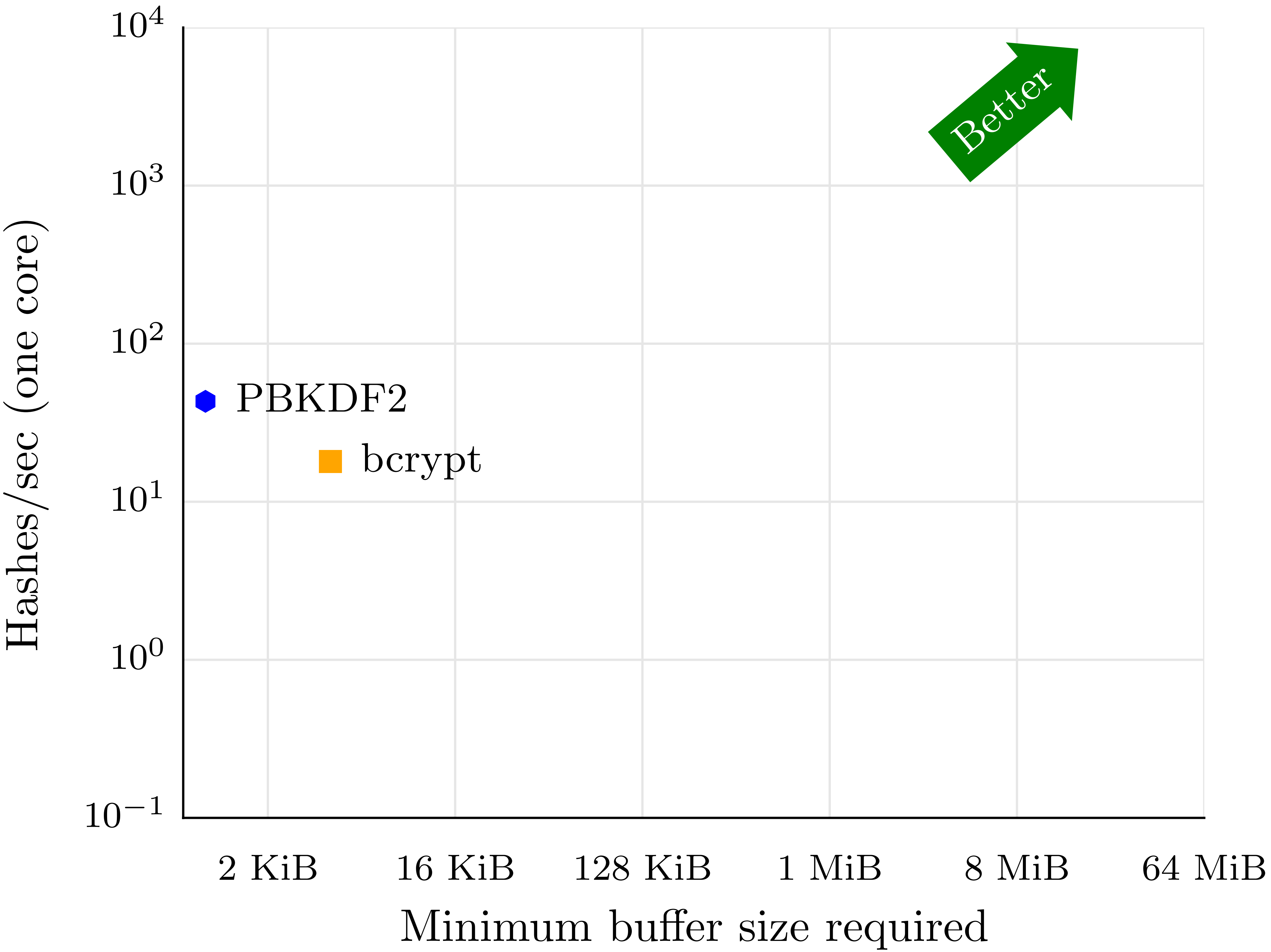
Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when δ=7, with space **$S \leq N/8$** requires time **T** such that

$$S \cdot T \geq (2^R - 1) / 8 \cdot N^2 .$$

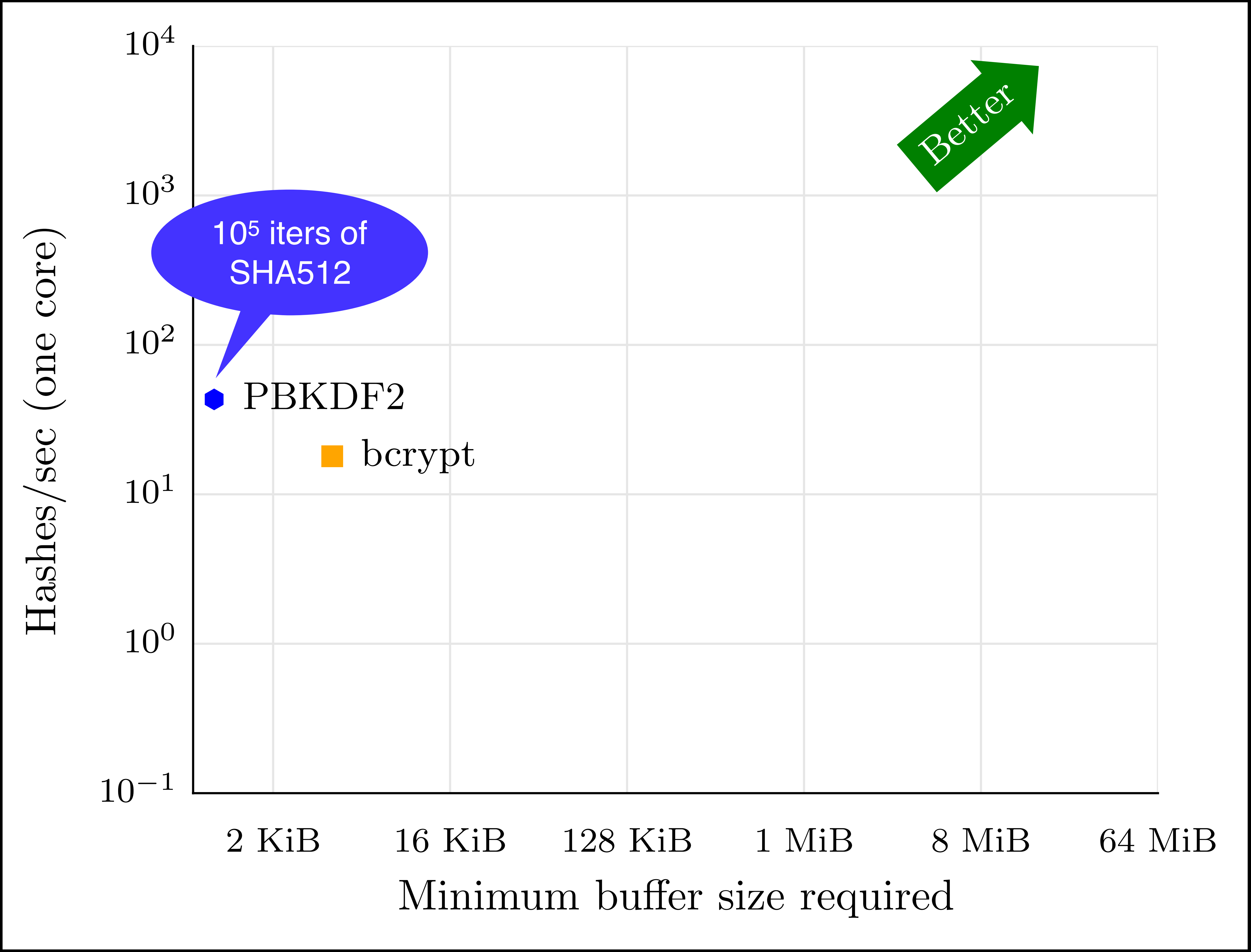
The proof works by inspecting the Balloon computation's data-dependency graph.

We draw heavily on prior work on pebbling arguments

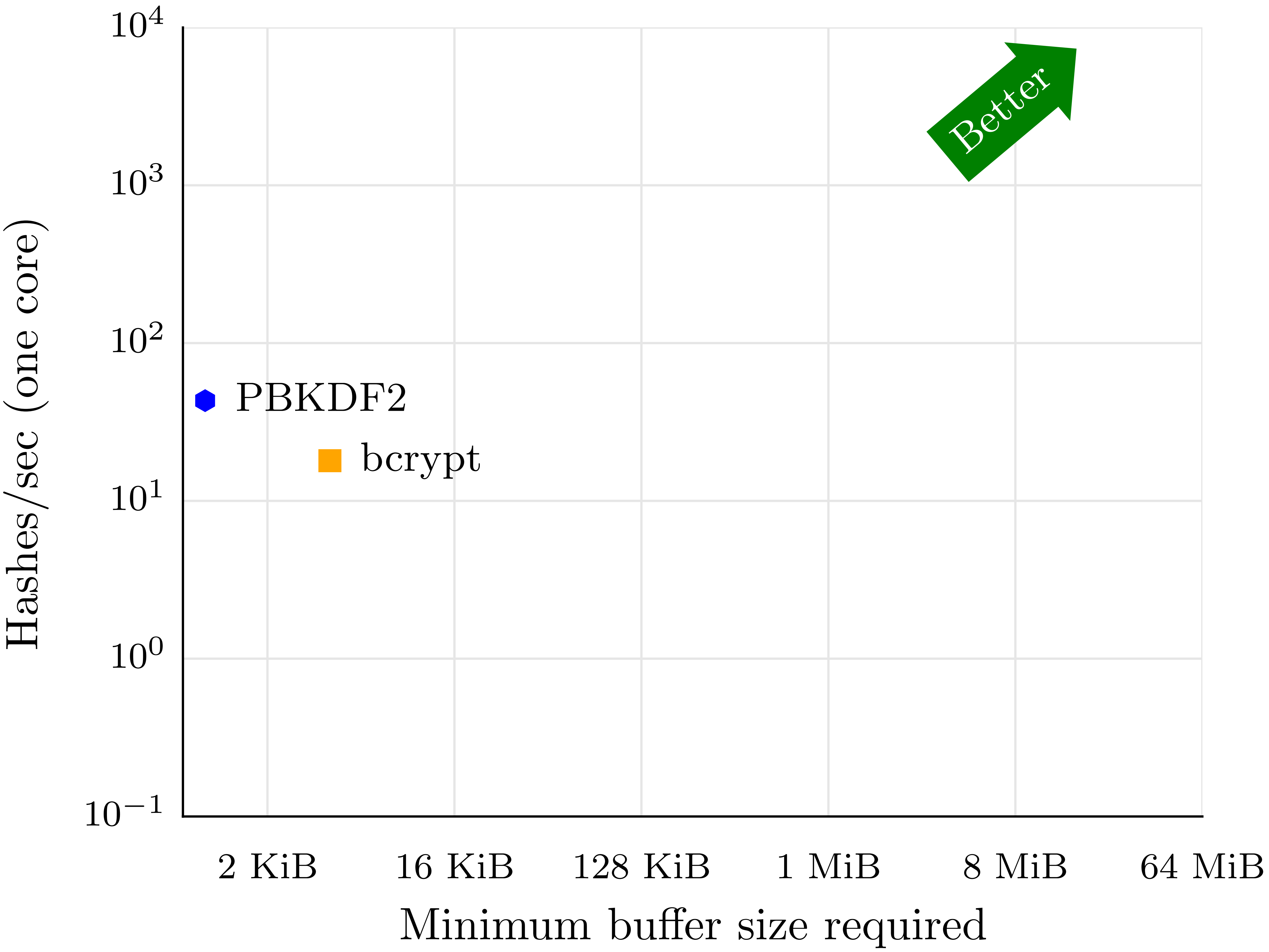
[Paterson & Hewitt 1970] [Paul & Tarjan 1978] [Dwork, Naor, Wee 2005] [Dziembowski, Kazana, Wichs 2011] [Alwen & Serbinenko 2015]



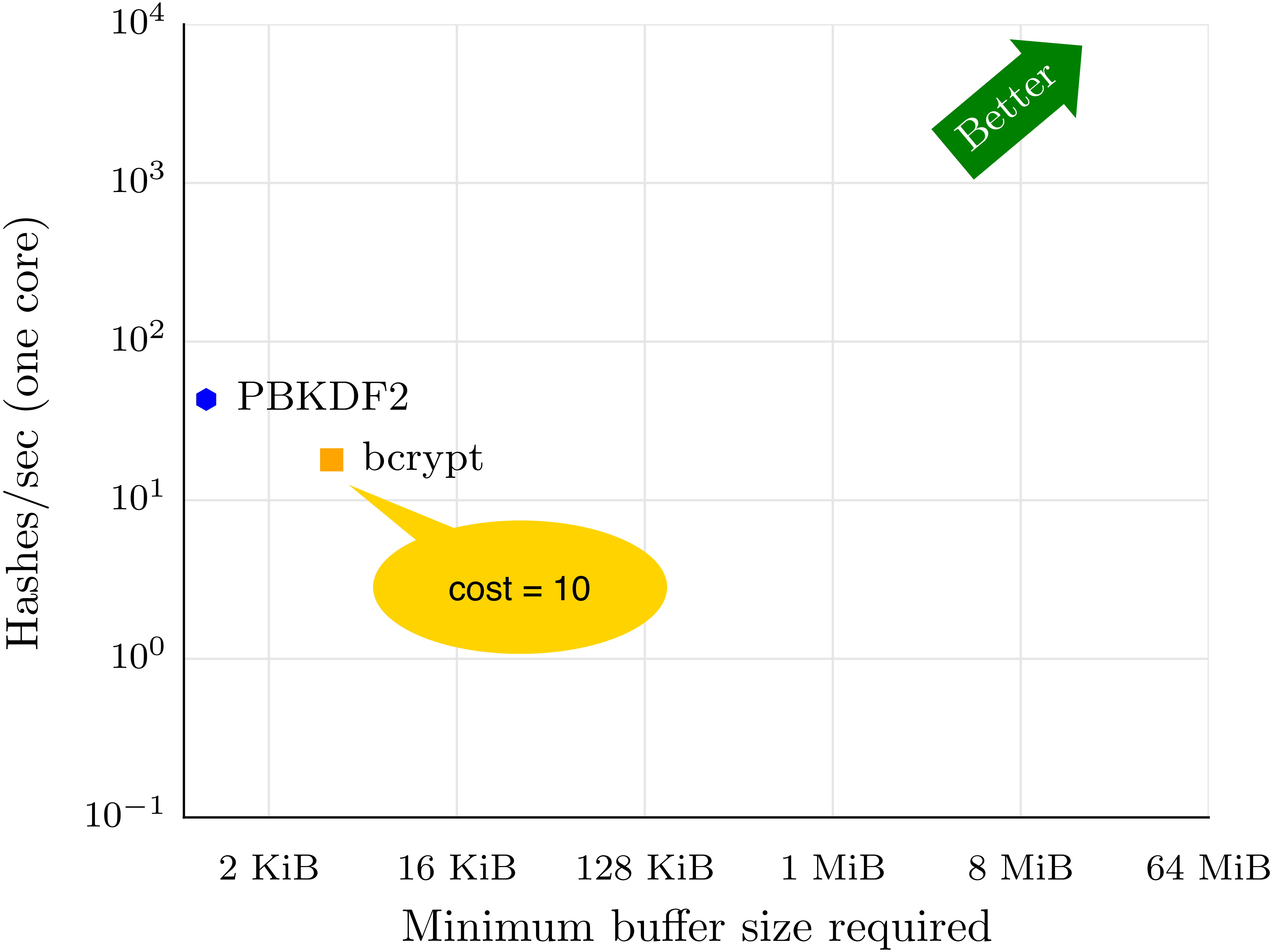
Using Balloon (δ=3). Both algorithms take four passes over memory.



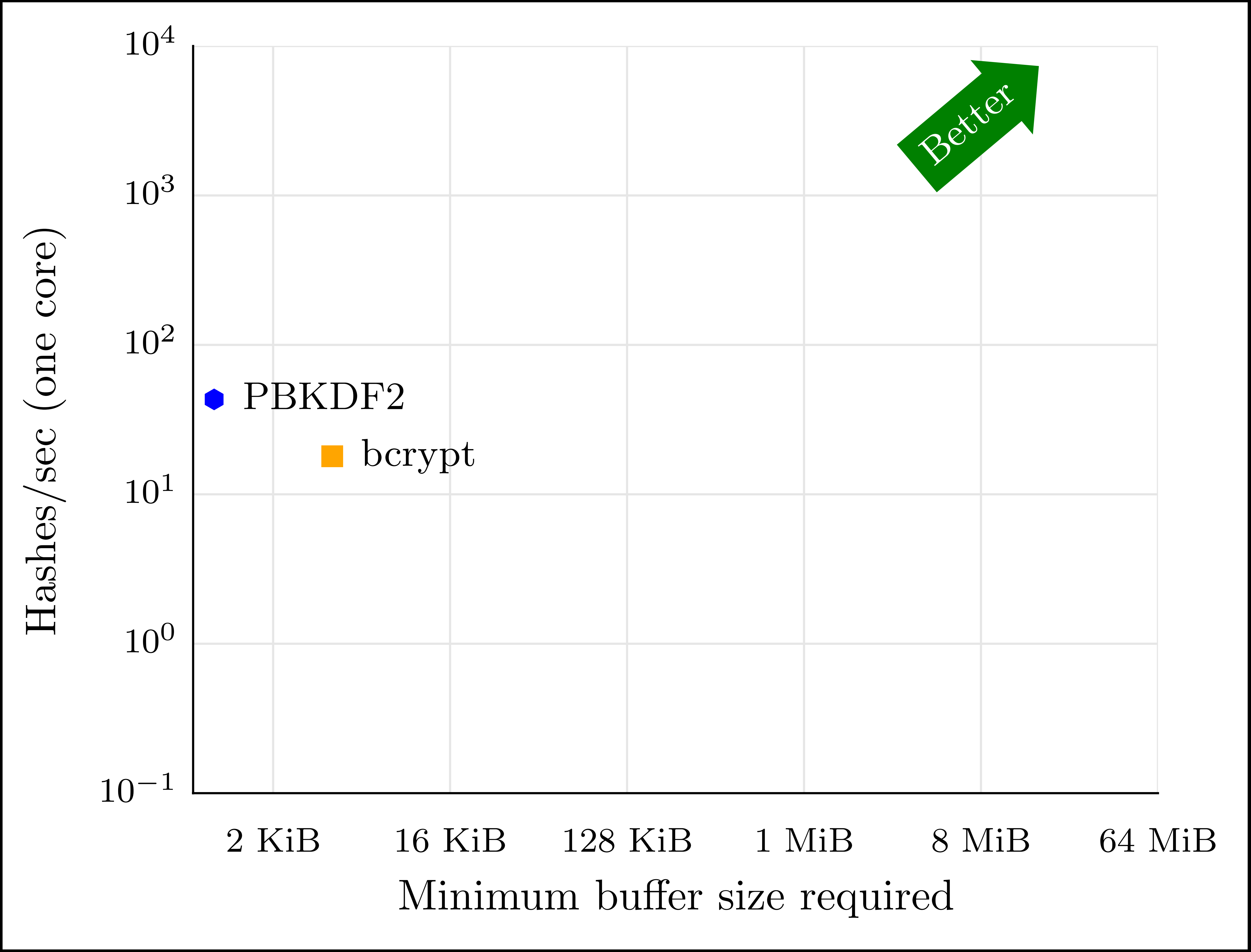
Using Balloon (δ=3). Both algorithms take four passes over memory.



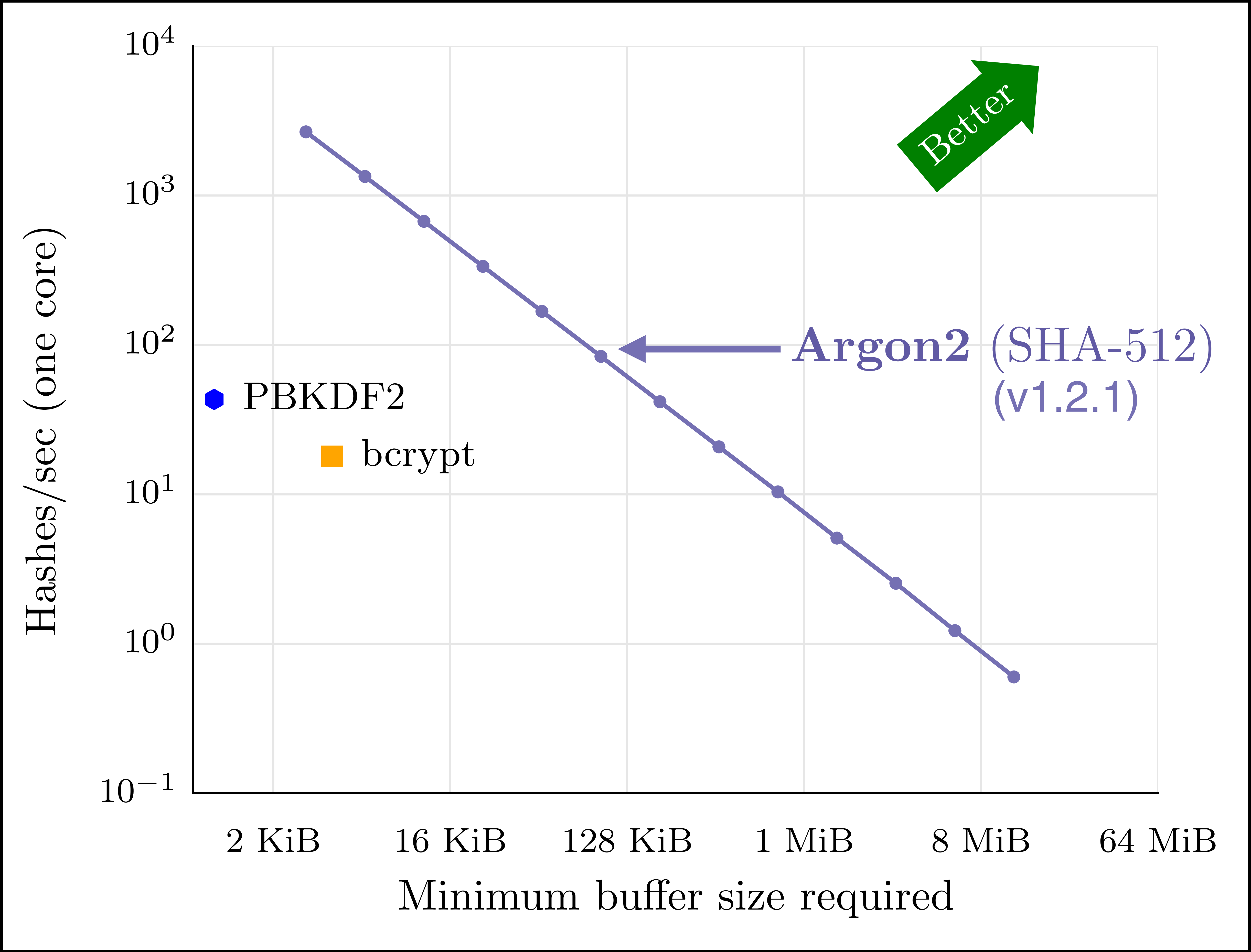
Using Balloon (δ=3). Both algorithms take four passes over memory.



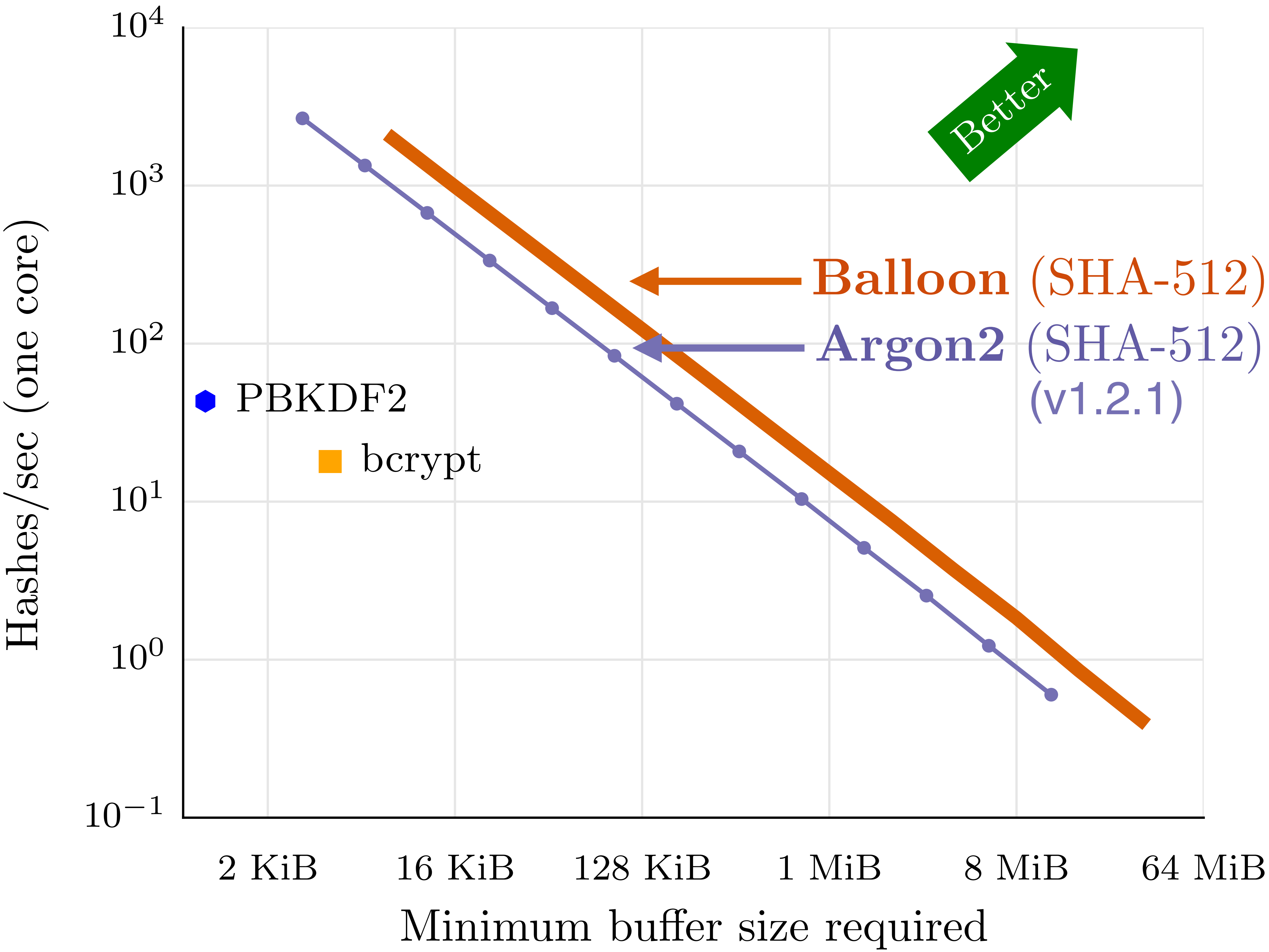
Using Balloon (δ=3). Both algorithms take four passes over memory.



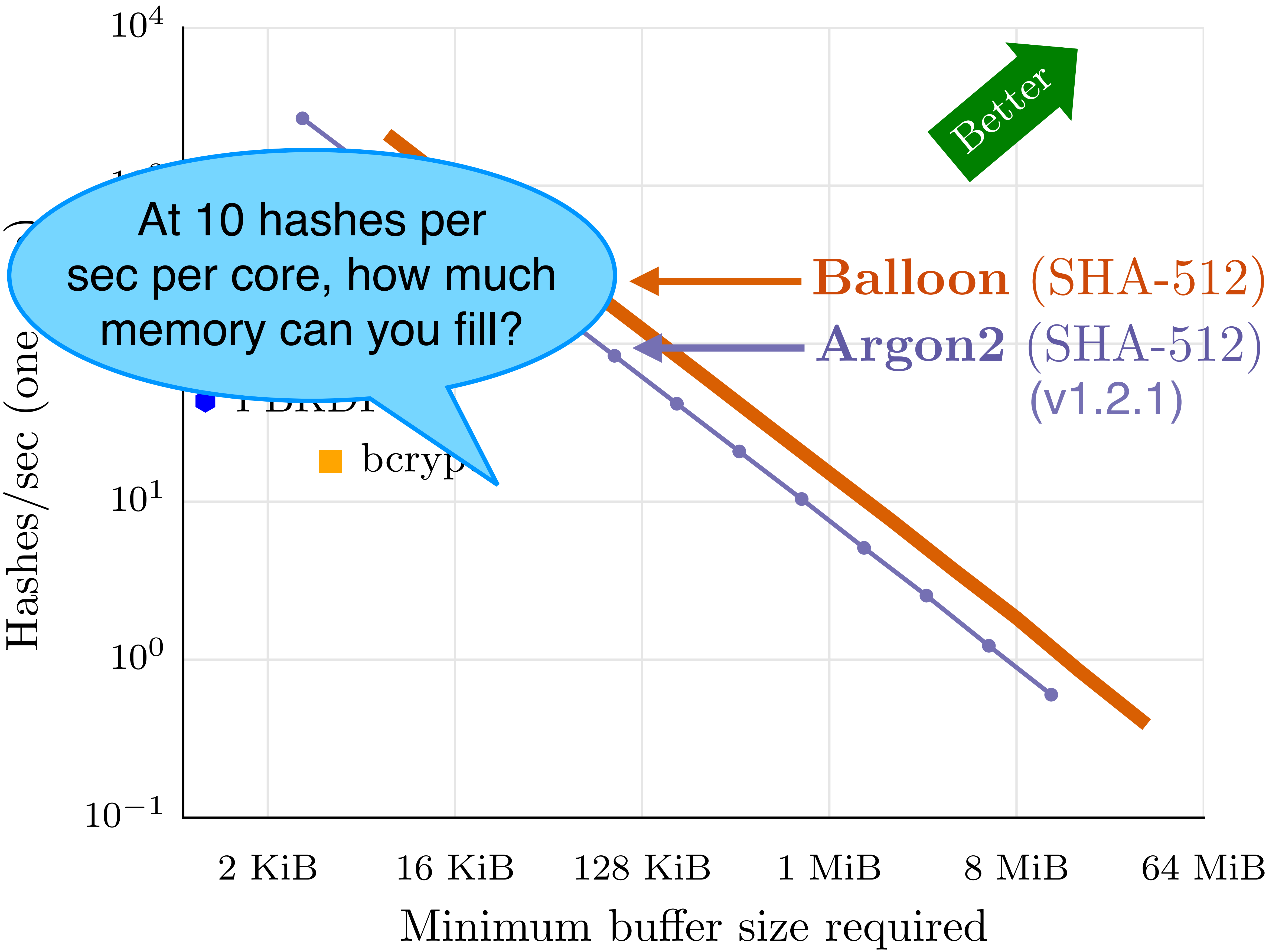
Using Balloon (δ=3). Both algorithms take four passes over memory.



Using Balloon (δ=3). Both algorithms take four passes over memory.

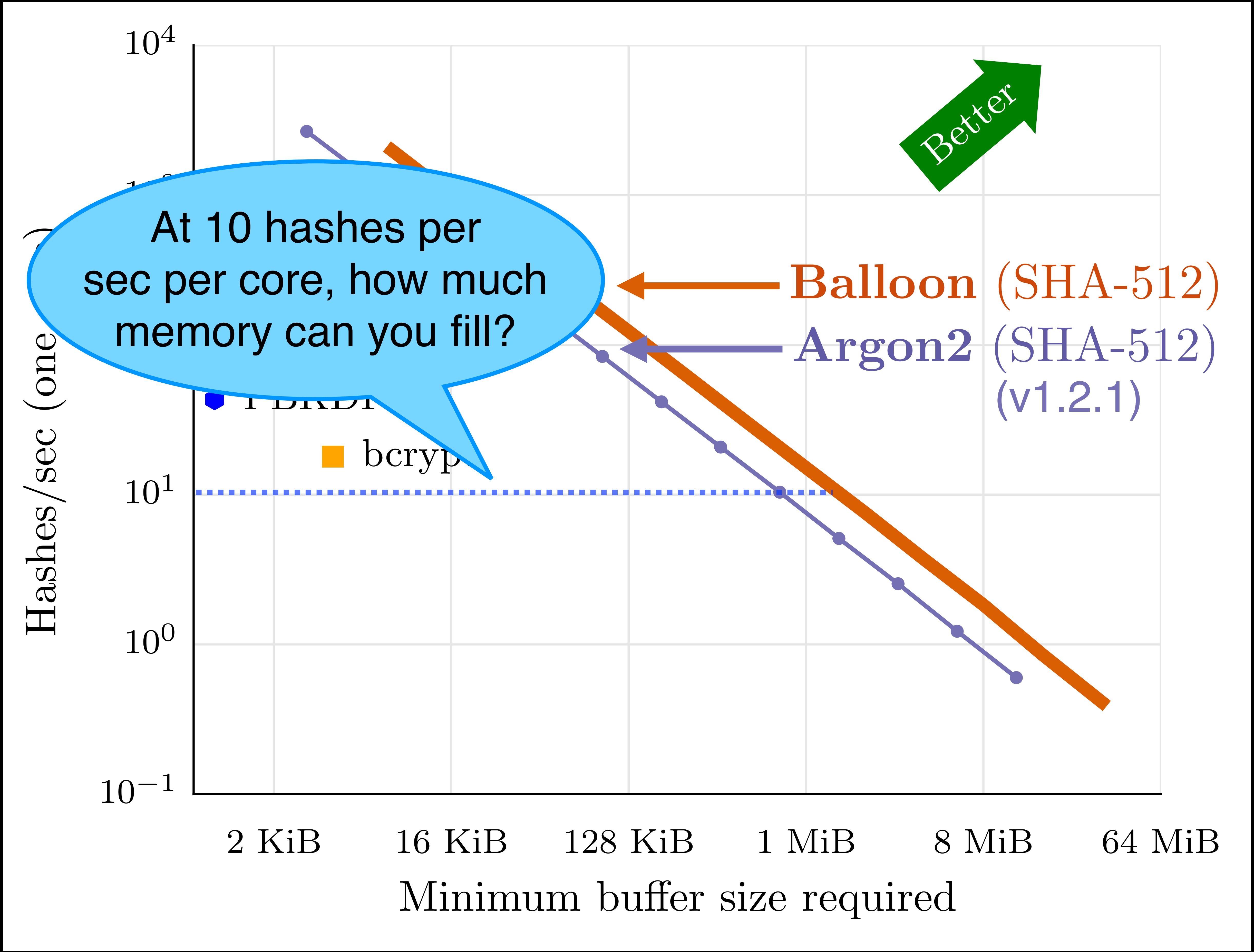


Using Balloon (δ=3). Both algorithms take four passes over memory.



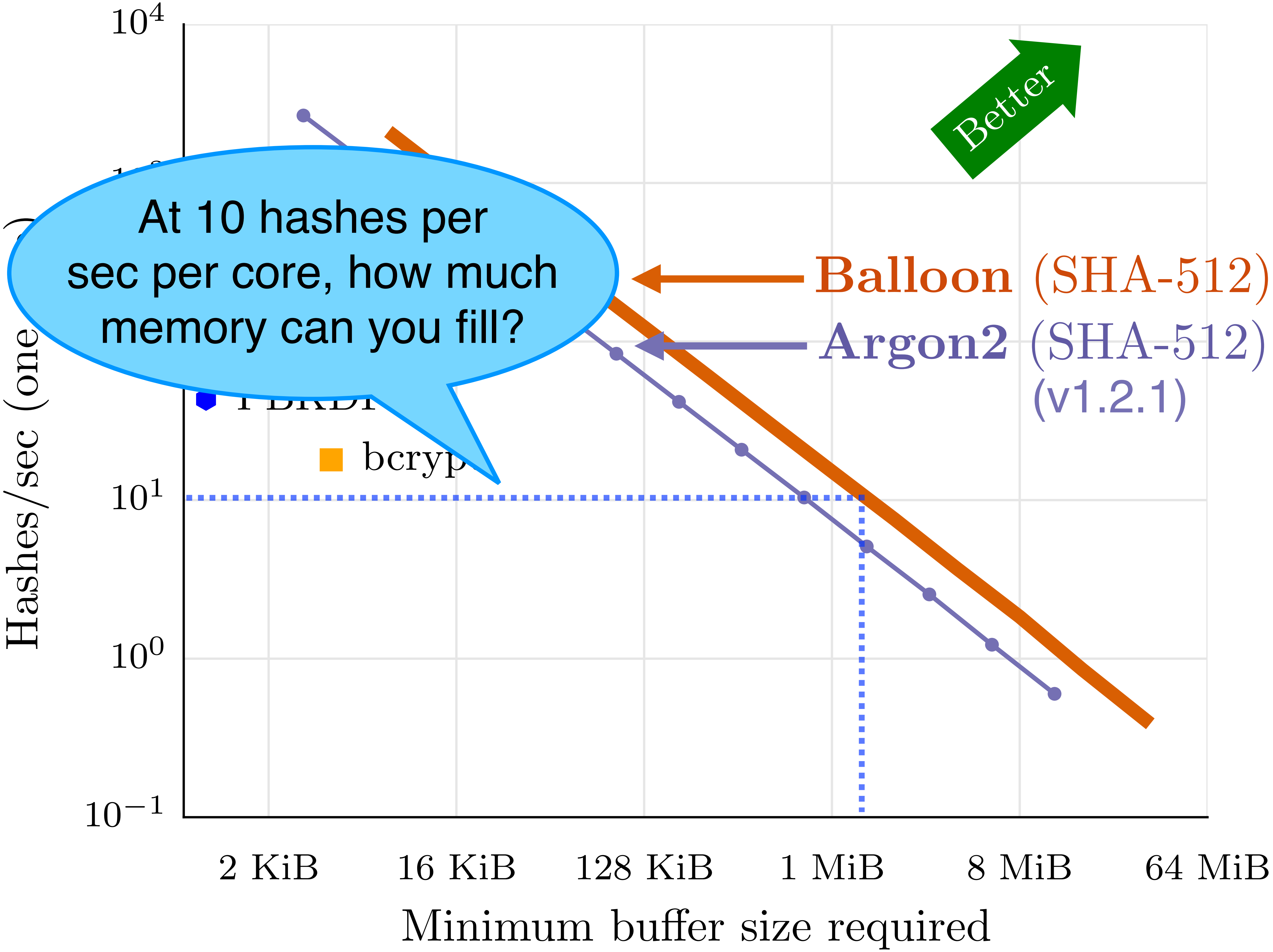
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon (δ=3). Both algorithms take four passes over memory.



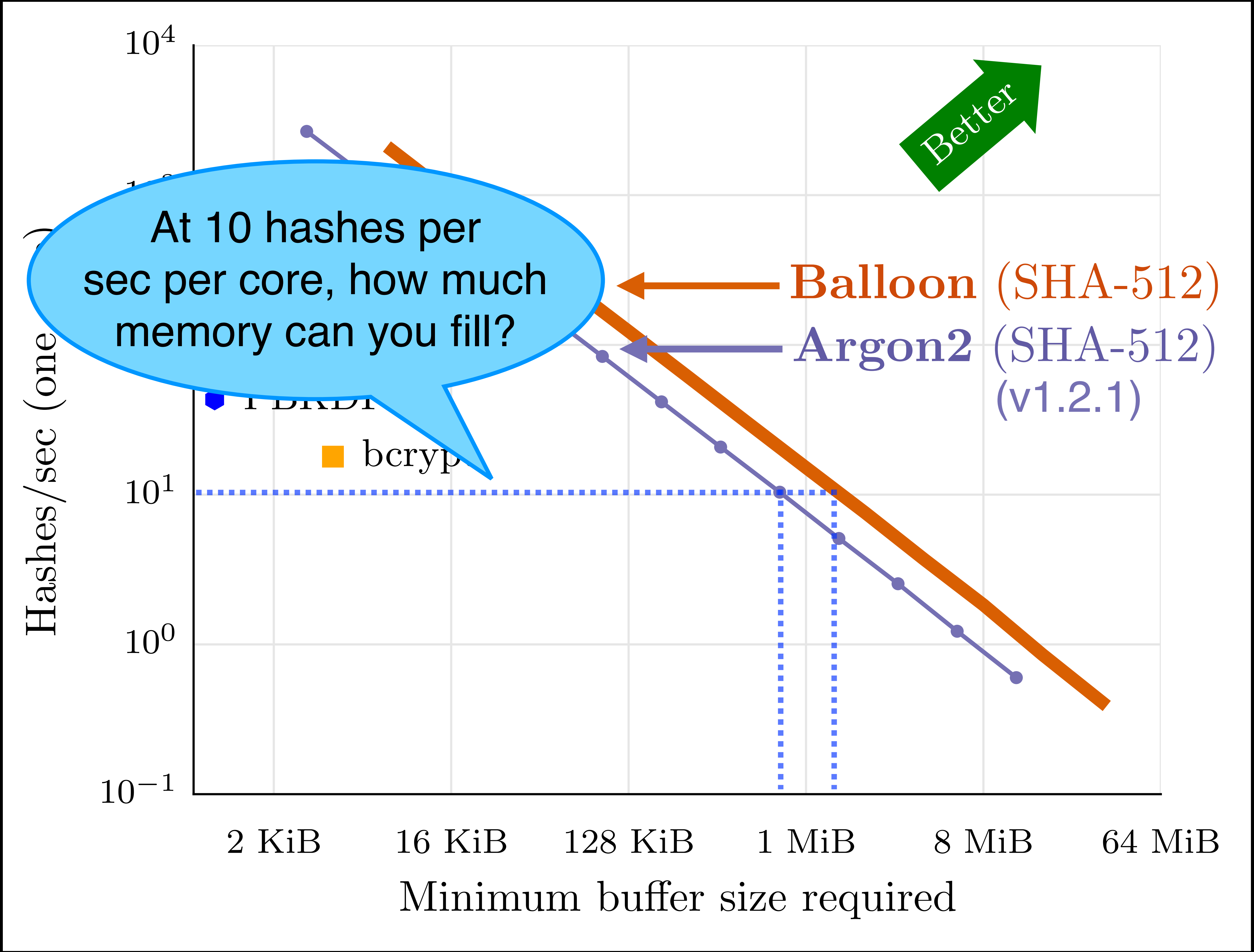
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon (δ=3). Both algorithms take four passes over memory.



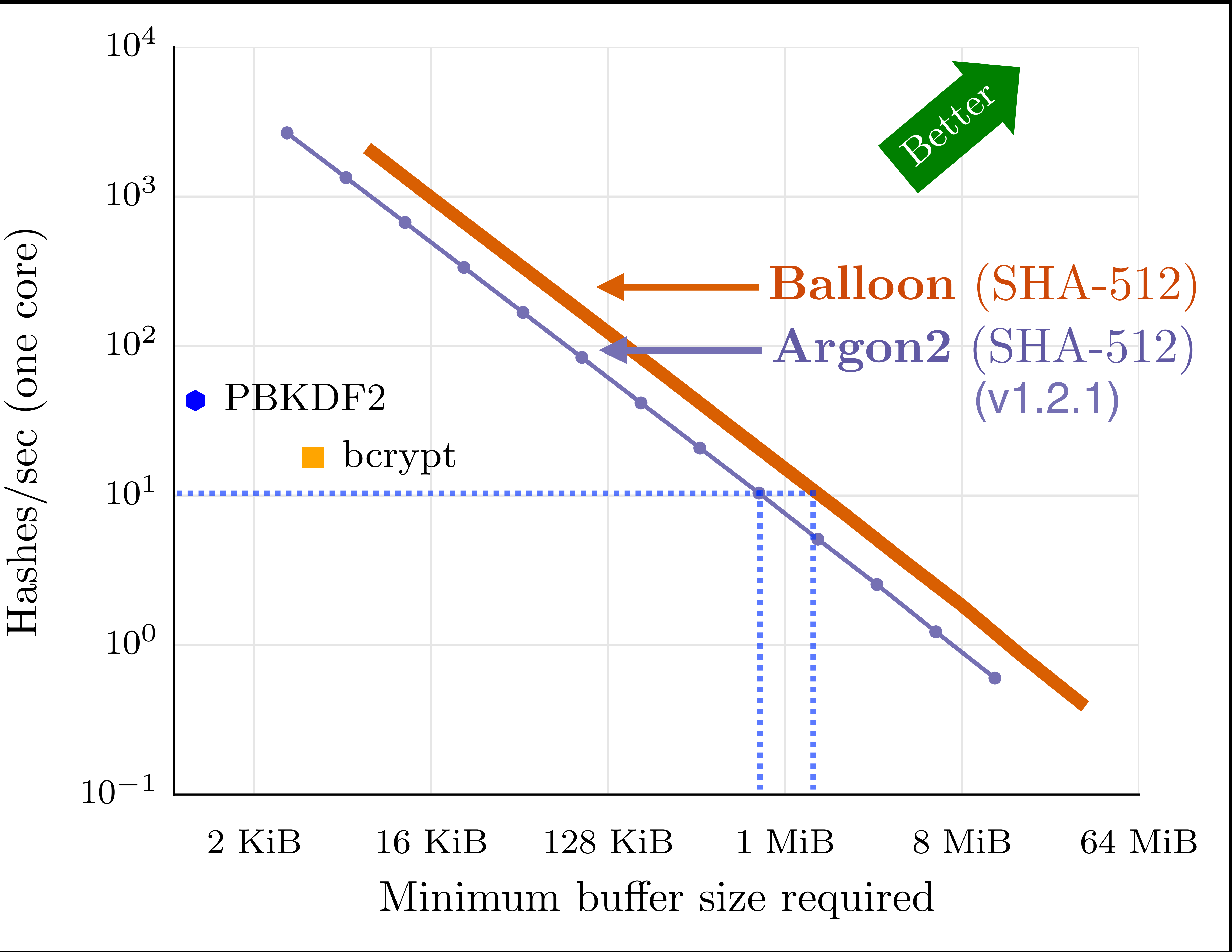
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon (δ=3). Both algorithms take four passes over memory.

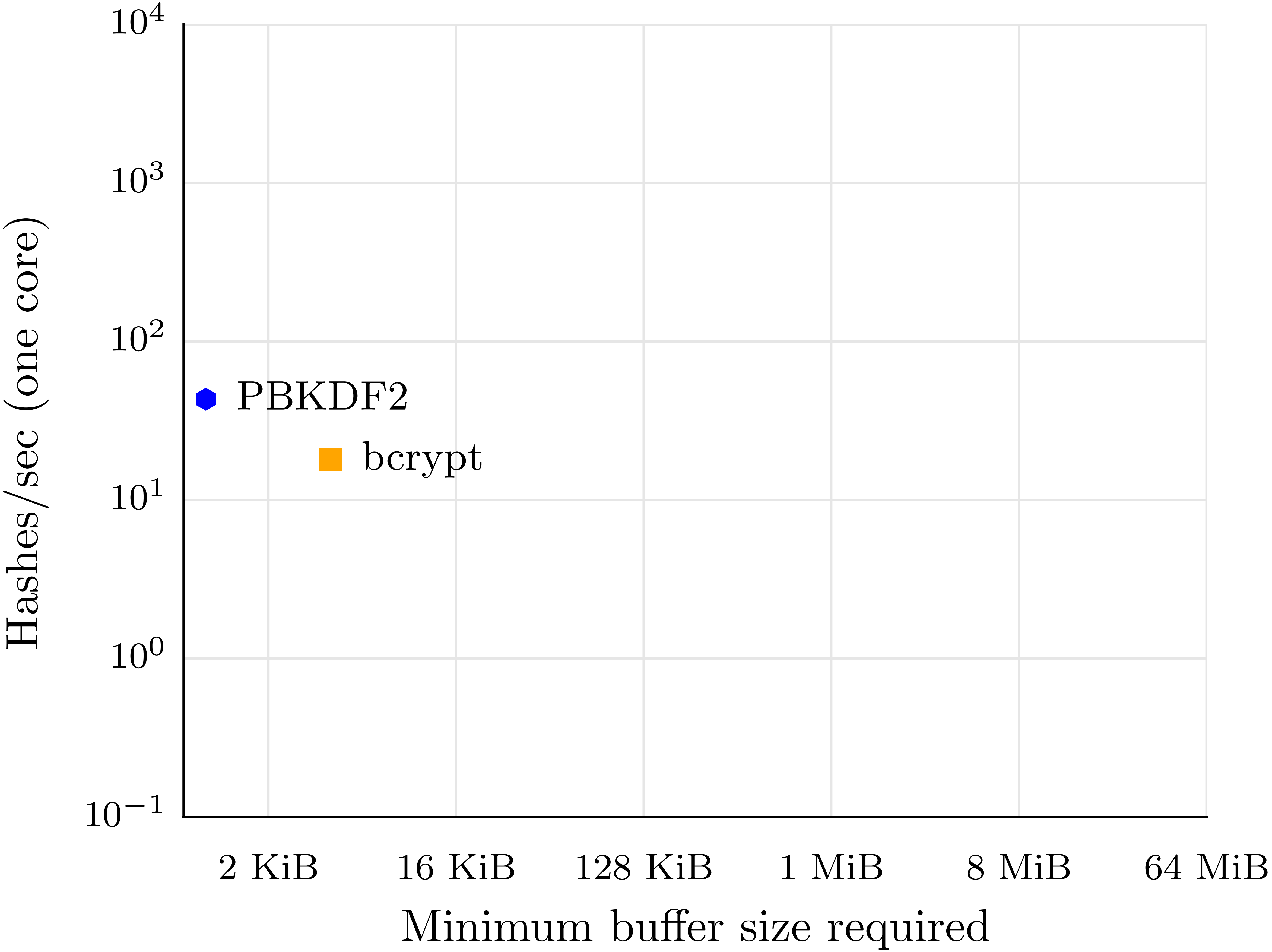


At 10 hashes per sec per core, how much memory can you fill?

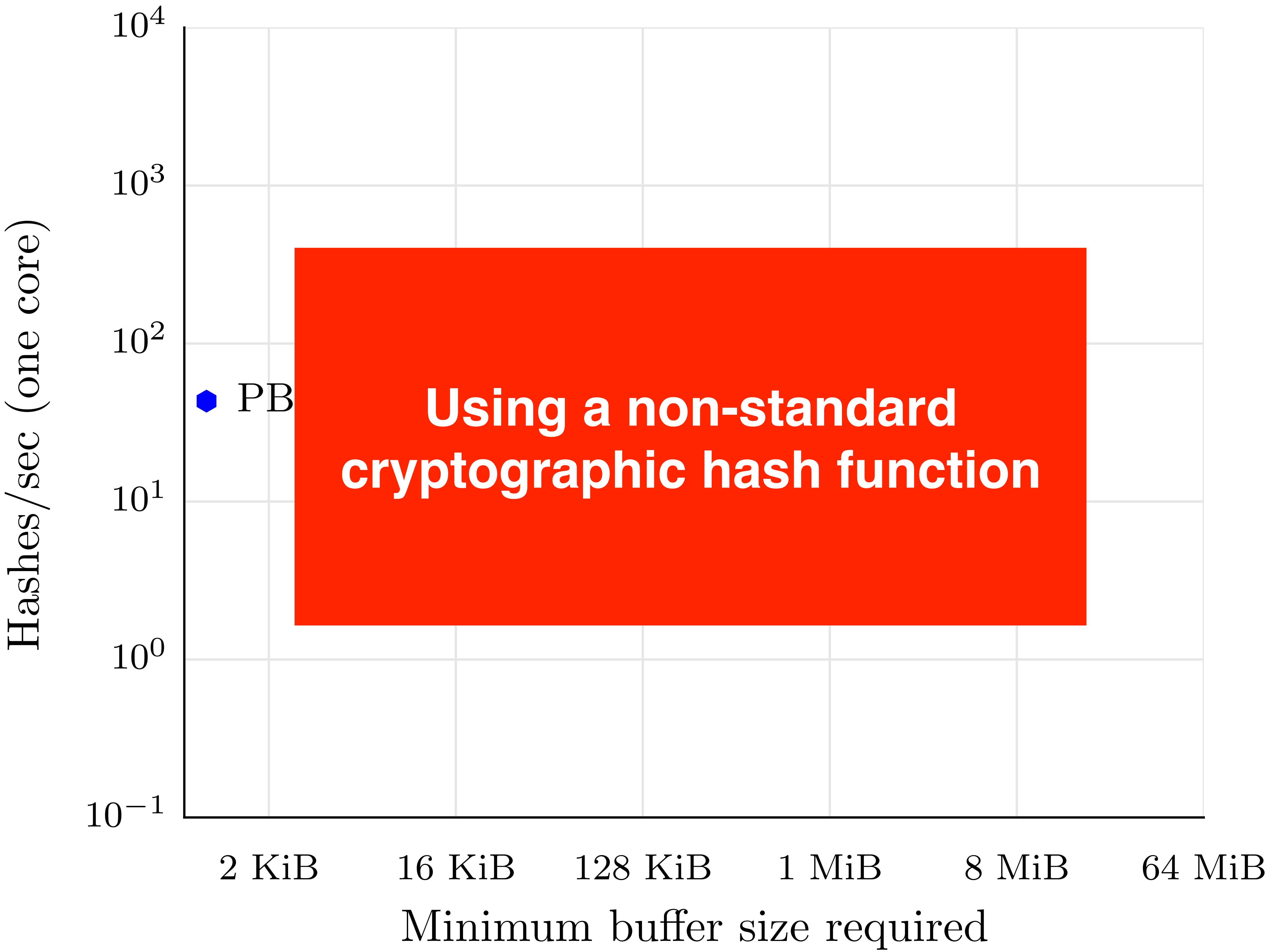
Using Balloon (δ=3). Both algorithms take four passes over memory.



Using Balloon (δ=3). Both algorithms take four passes over memory.

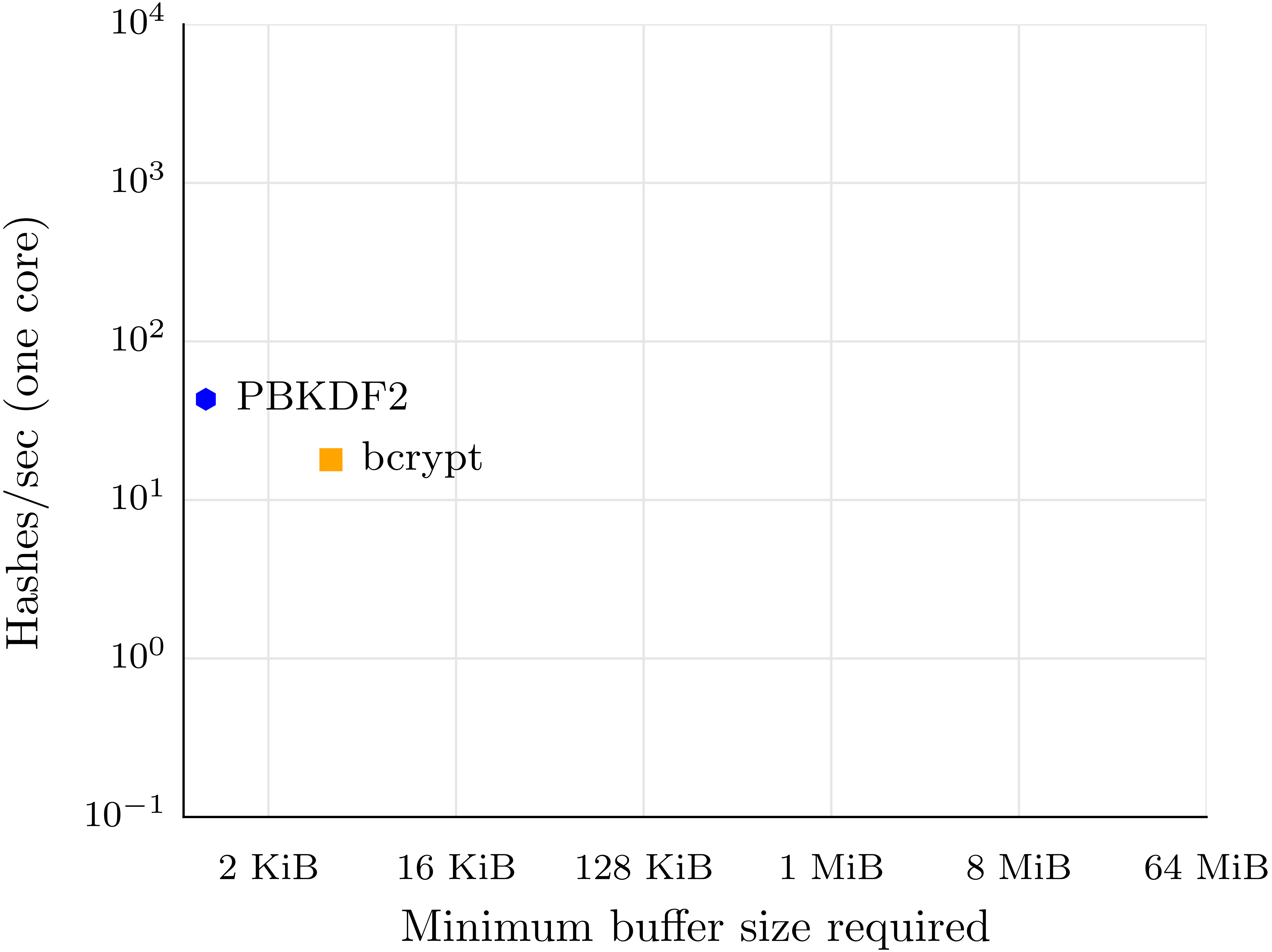


Using Balloon (δ=3). Both algorithms take four passes over memory.

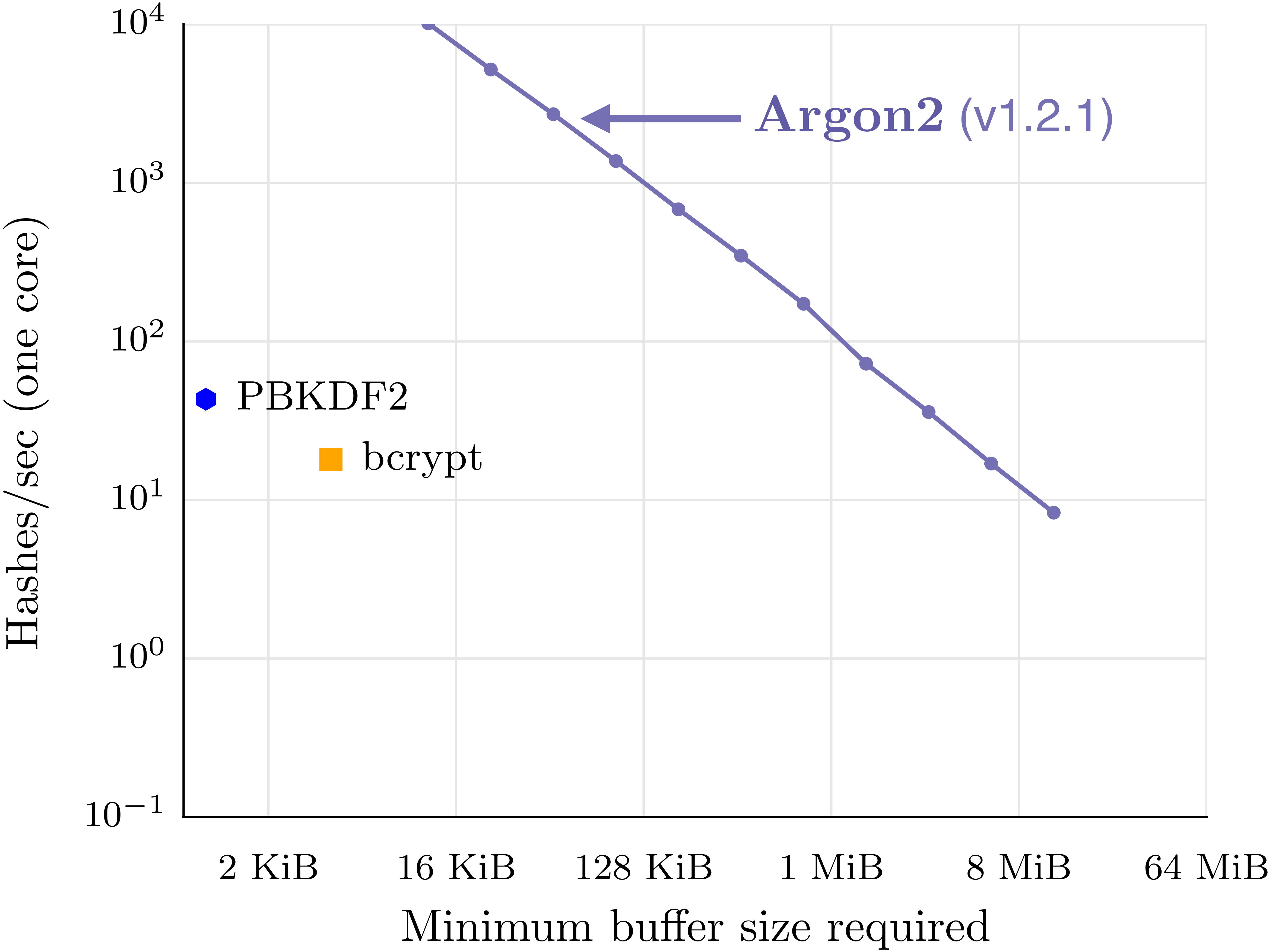


Using a non-standard cryptographic hash function

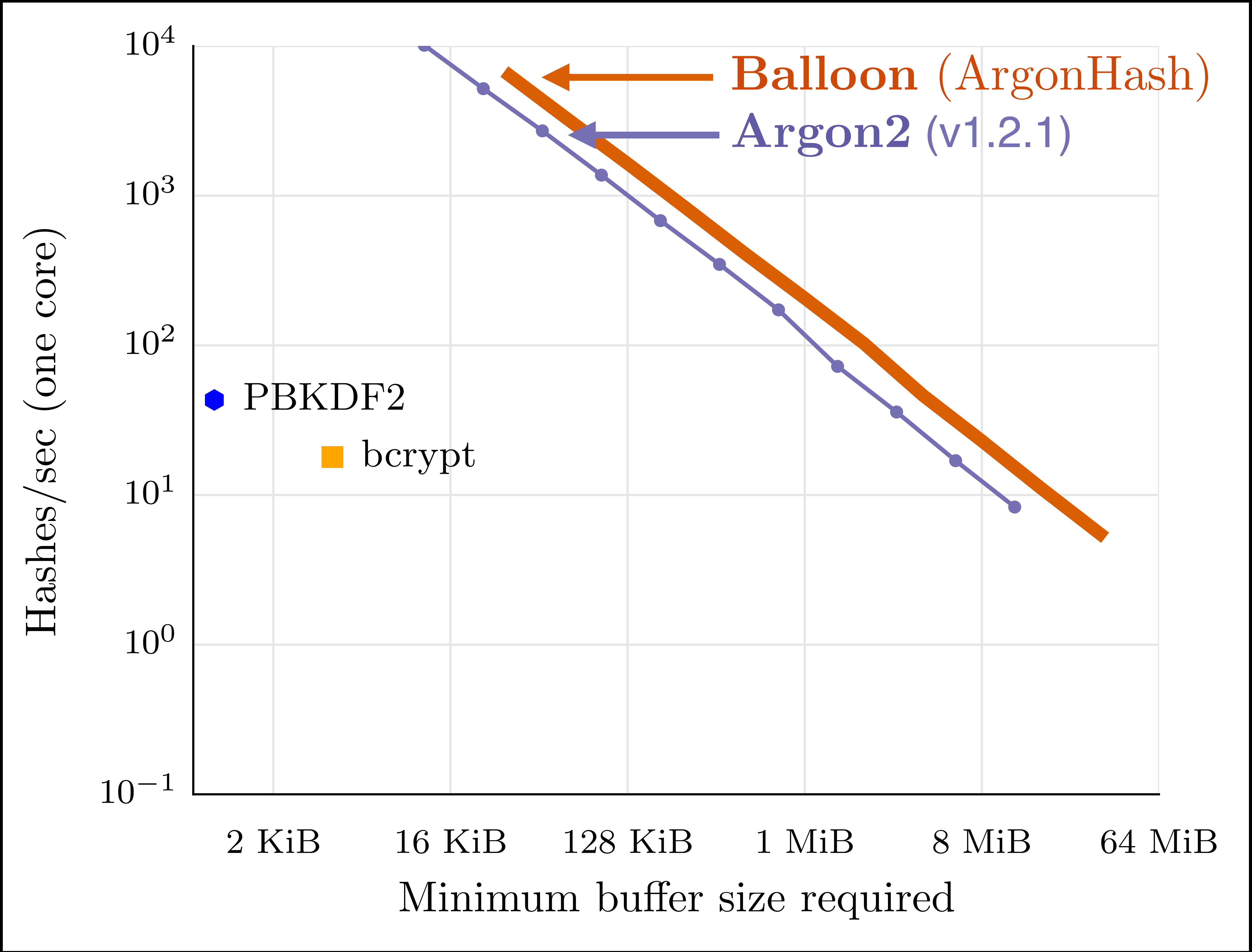
Using Balloon (δ=3). Both algorithms take four passes over memory.



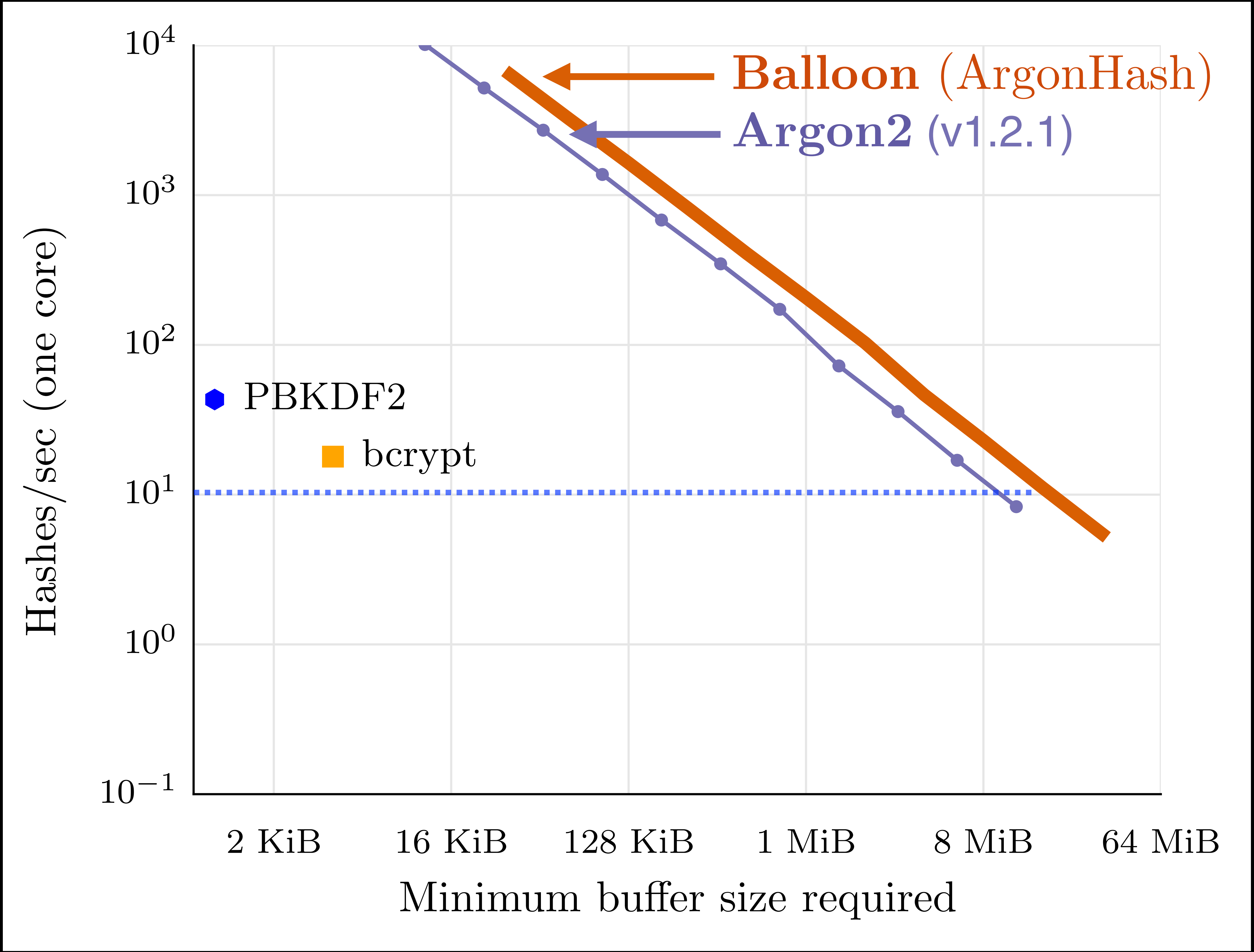
Using Balloon (δ=3). Both algorithms take four passes over memory.



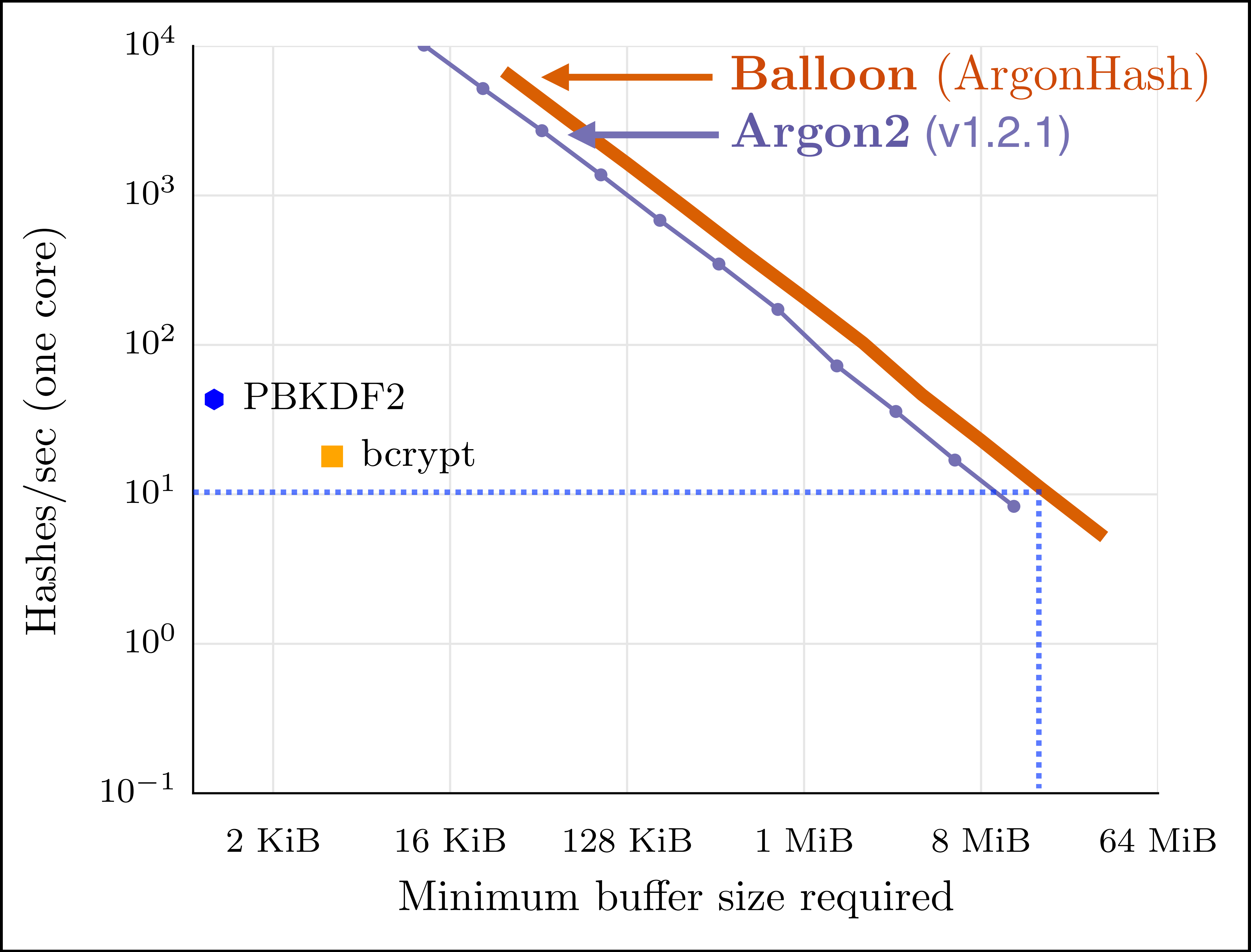
Using Balloon (δ=3). Both algorithms take four passes over memory.



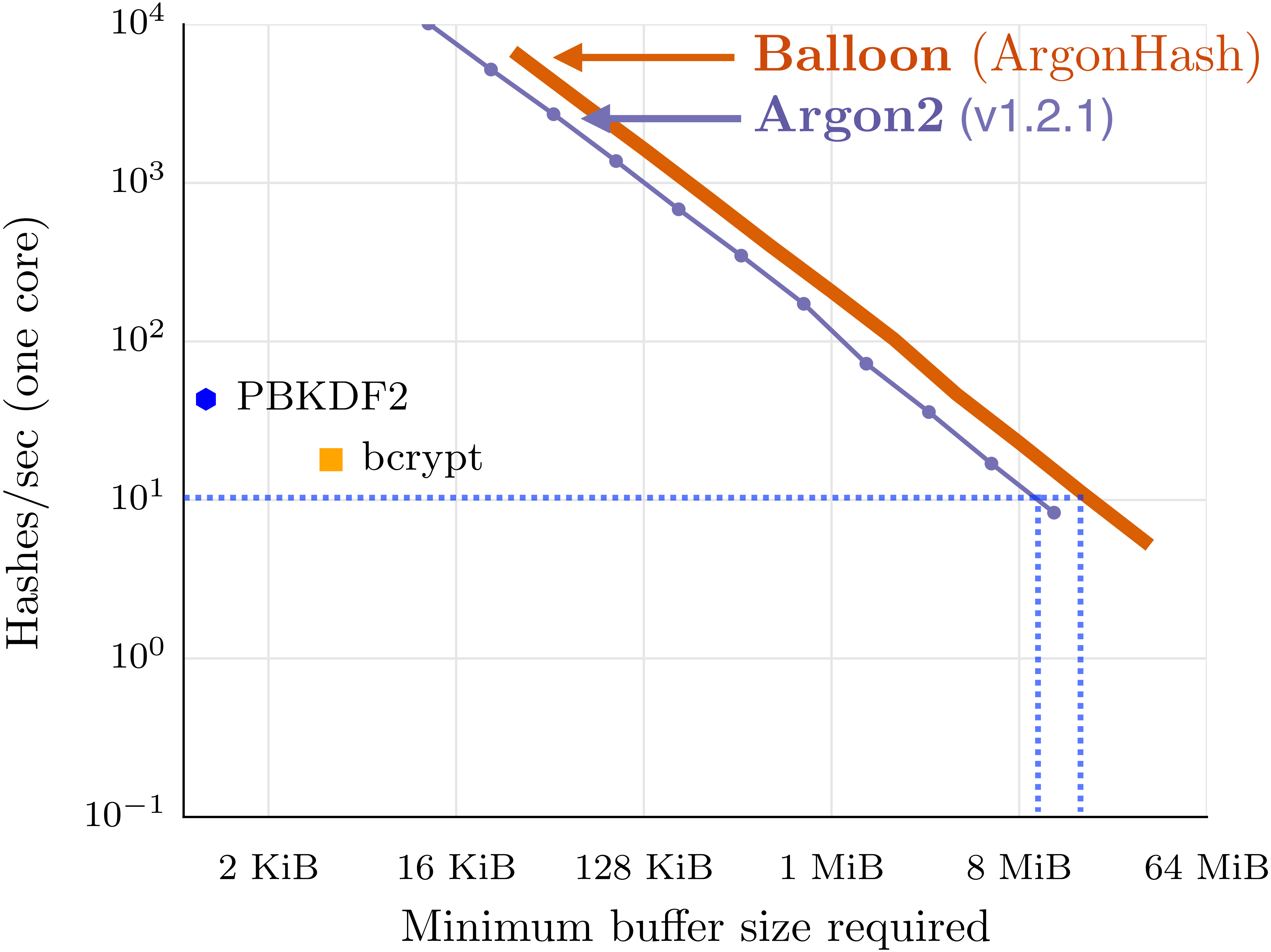
Using Balloon (δ=3). Both algorithms take four passes over memory.



Using Balloon (δ=3). Both algorithms take four passes over memory.



Using Balloon (δ=3). Both algorithms take four passes over memory.



Using Balloon (δ=3). Both algorithms take four passes over memory.

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion

Discussion: Parallel attacks

Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a *parallel* setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern

Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a *parallel* setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  - Including Balloon, Argon2i, etc.

Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a *parallel* setting, it's possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  - Including Balloon, Argon2i, etc.

- The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)

Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a *parallel* setting, it's possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  - Including Balloon, Argon2i, etc.
- The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)
- The attack would require special-purpose hardware with many cores and shared memory

Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a *parallel* setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  - Including Balloon, Argon2i, etc.
- The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)
- The attack would require special-purpose hardware with many cores and shared memory

→ Not yet clear whether these attacks are of practical concern.

Discussion: Comparison with Argon2

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions

- We demonstrate a practical attack against Argon2i's memory-hardness properties

(Designers have since modified the construction)

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions

- We demonstrate a practical attack against Argon2i's memory-hardness properties

  (Designers have since modified the construction)
- We prove that much better attacks are impossible

Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition

- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

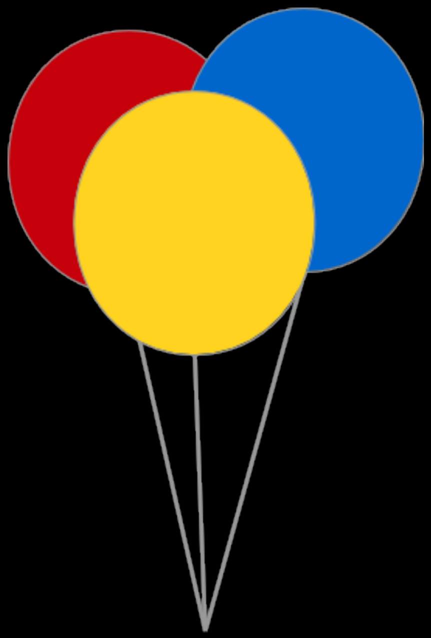
Our Contributions

- We demonstrate a practical attack against Argon2i's memory-hardness properties

  (Designers have since modified the construction)
- We prove that much better attacks are impossible

→ Balloon has stronger proven security properties than Argon2i.
(In practice…)

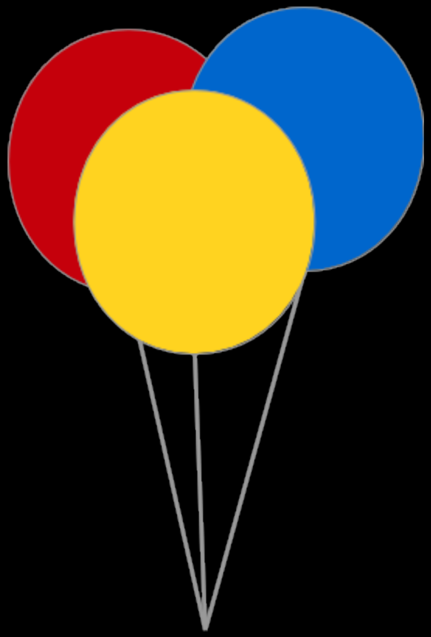
Conclusion



Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Conclusion

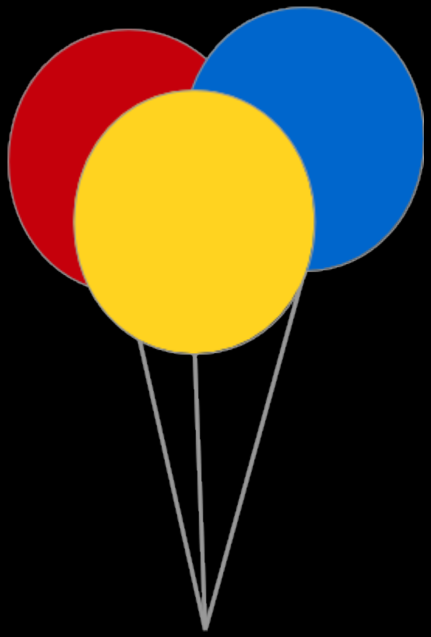


- Memory-hard password hashing functions increase the cost of offline dictionary attacks.

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Conclusion



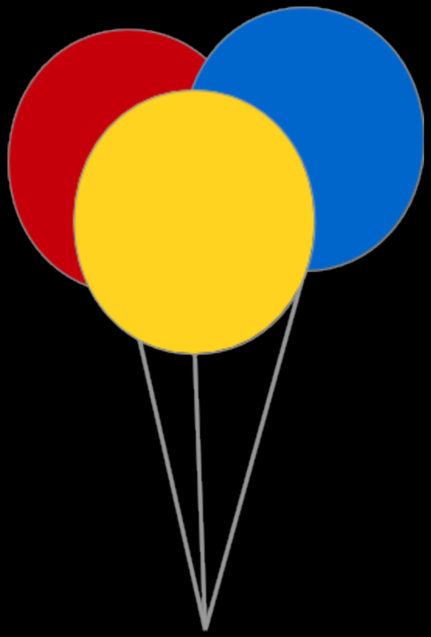
- Memory-hard password hashing functions increase the cost of offline dictionary attacks.

- Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Conclusion

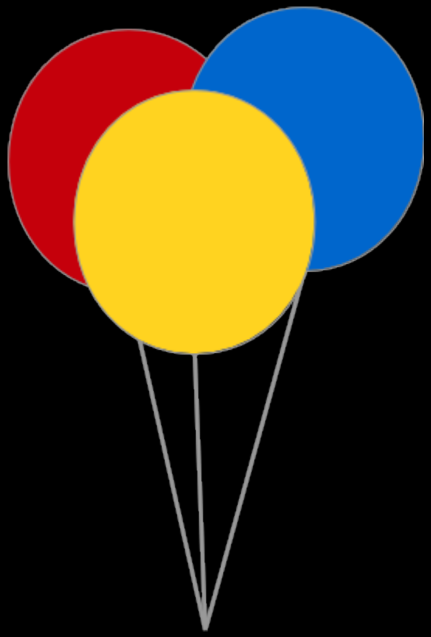


- Memory-hard password hashing functions increase the cost of offline dictionary attacks.
- Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.
- Balloon+SHA512* **is strictly better** than iterated hashing (PBKDF2-SHA512).

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Conclusion

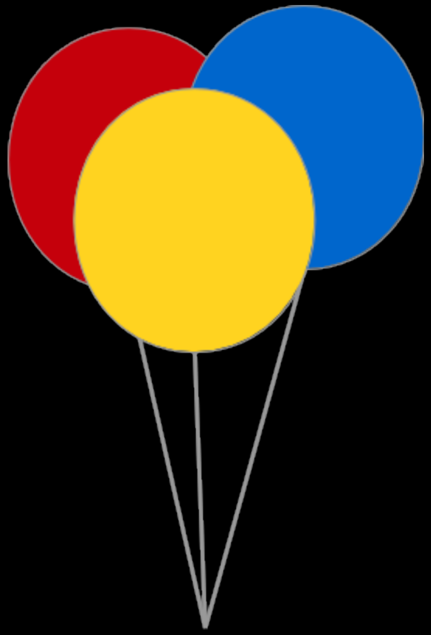


- Memory-hard password hashing functions increase the cost of offline dictionary attacks.
- Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.
- Balloon+SHA512* **is strictly better** than iterated hashing (PBKDF2-SHA512).

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Conclusion



- Memory-hard password hashing functions increase the cost of offline dictionary attacks.

- Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.

- Balloon+SHA512* **is strictly better** than iterated hashing (PBKDF2-SHA512).

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*

Attacking scrypt

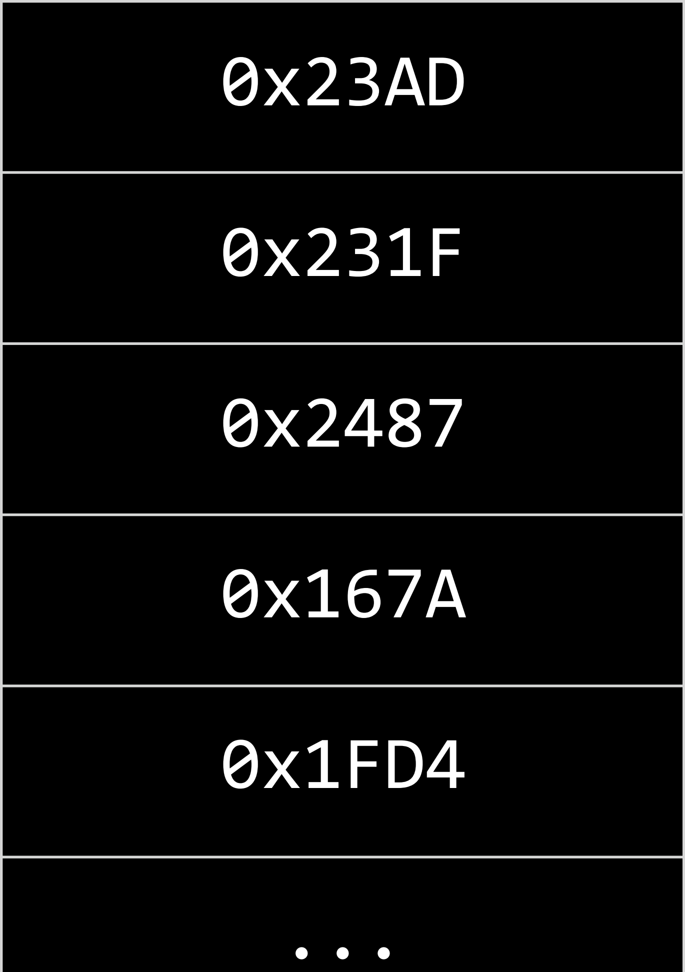
An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*

```
scrypt(passwd)
```

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*

`scrypt(passwd)`



```
0x23AD
0x231F
0x2487
0x167A
0x1FD4
...
```

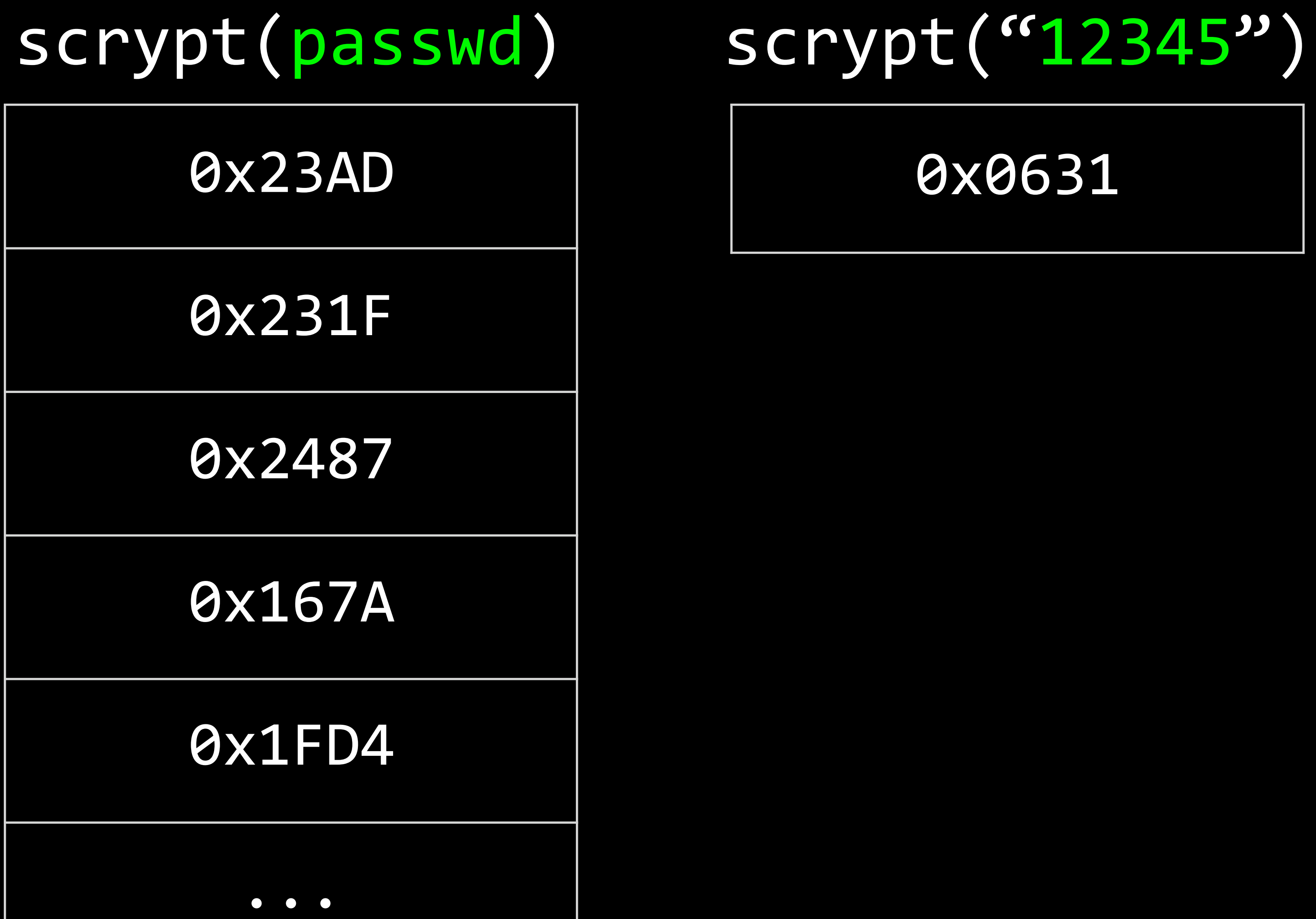
Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*

`scrypt(passwd)`	`scrypt(“12345”)`
0x23AD	
0x231F	
0x2487	
0x167A	
0x1FD4	
...	

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



`scrypt(passwd)`

`0x23AD`

`0x231F`

`0x2487`

`0x167A`

`0x1FD4`

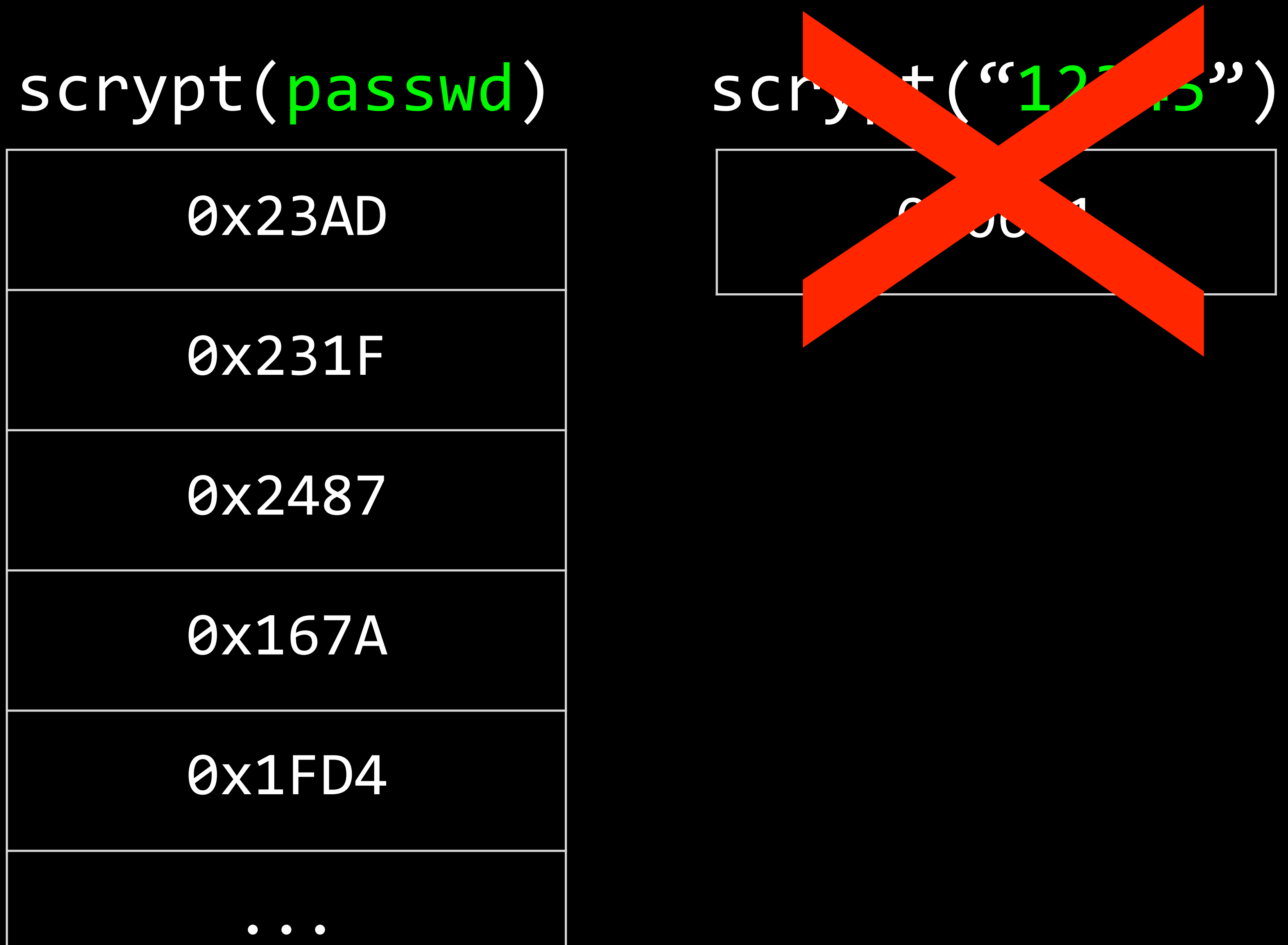
`...`

`scrypt(“12345”)`

`0x0631`

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



scrypt(passwd)

0x23AD

0x231F

0x2487

0x167A

0x1FD4

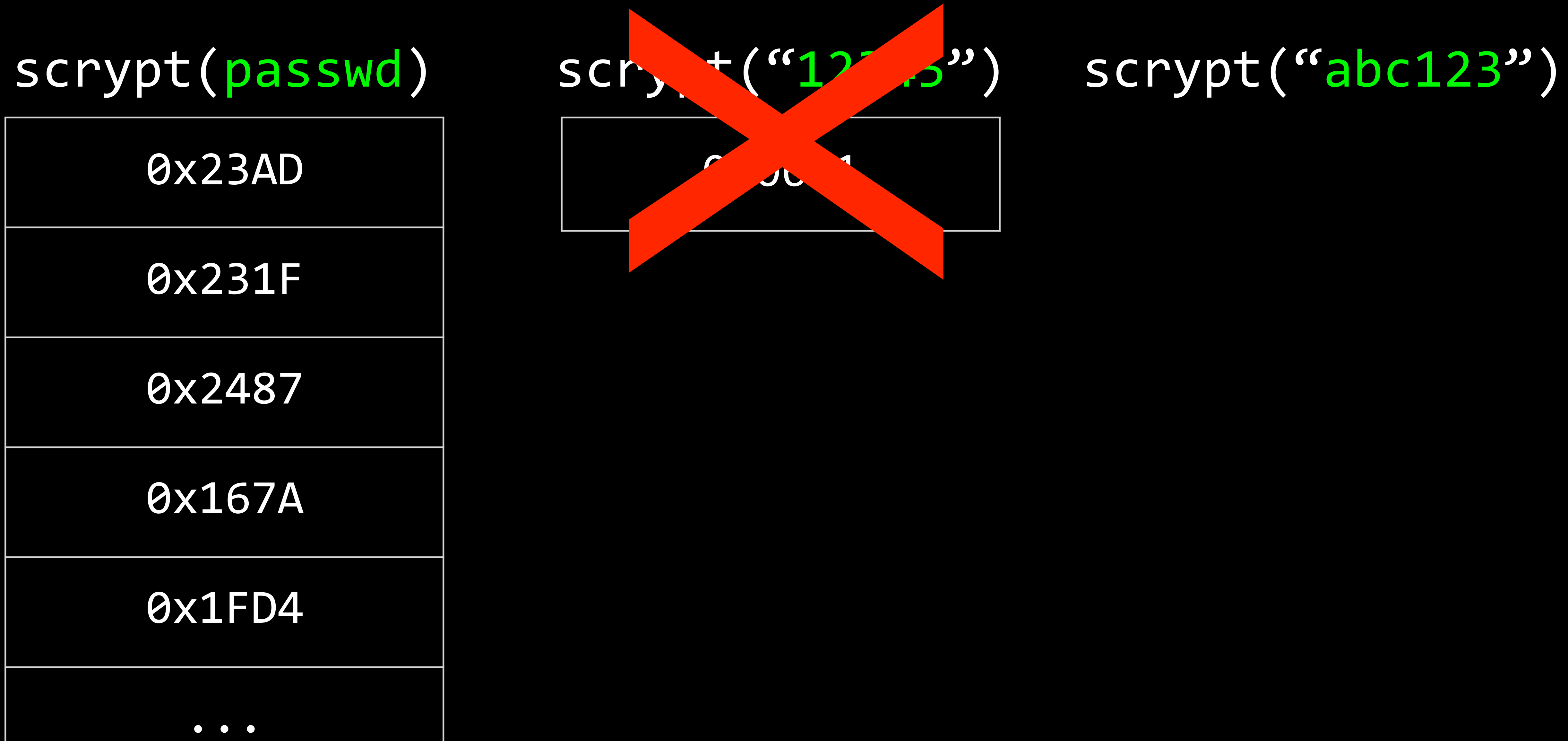
...

scrypt(“12345”)

0x0001

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



`scrypt(passwd)`

```
0x23AD
0x231F
0x2487
0x167A
0x1FD4
...
```

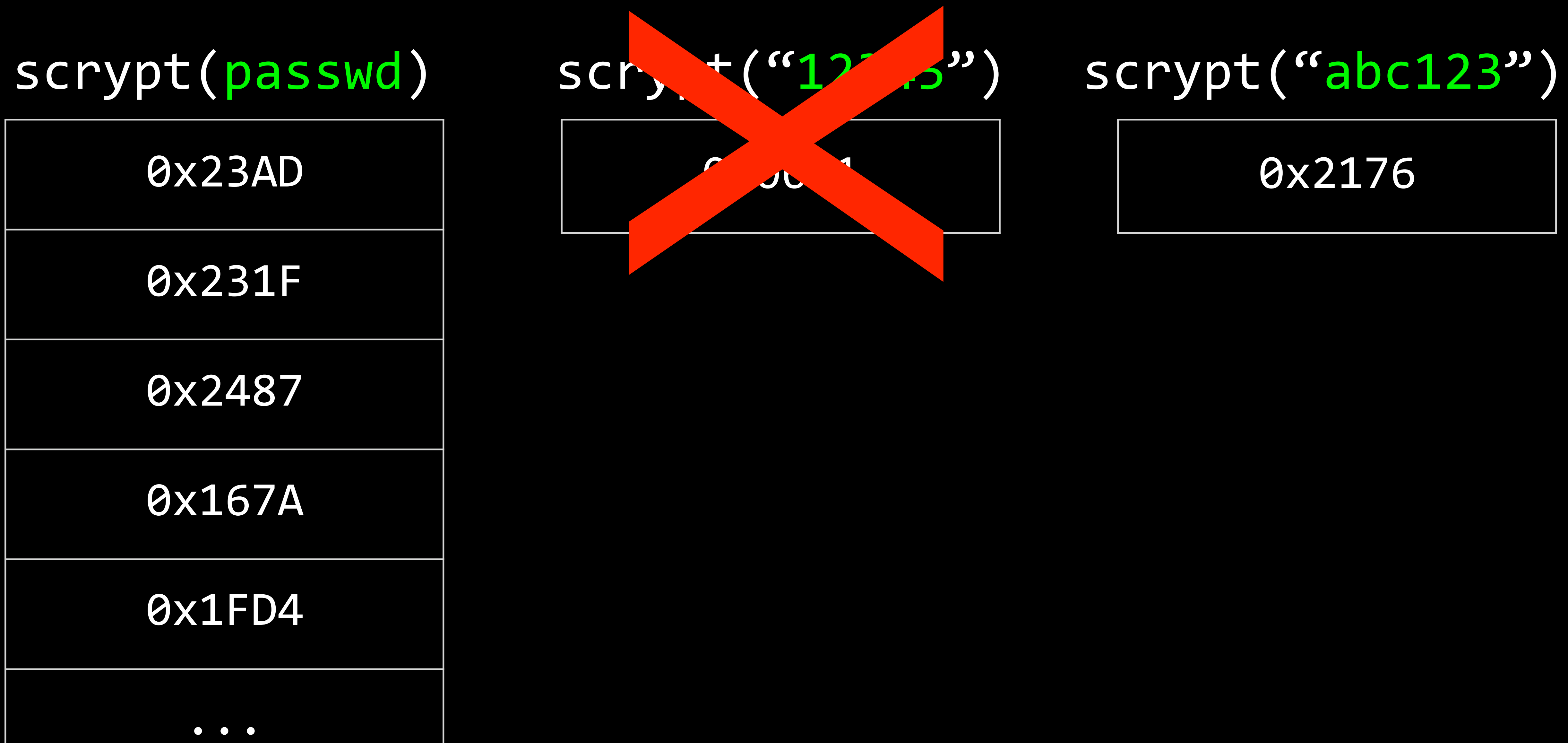
~~`scrypt("12345")`~~

~~`0x00[illegible]1`~~

`scrypt("abc123")`

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



```
scrypt(passwd)
0x23AD
0x231F
0x2487
0x167A
0x1FD4
...
```

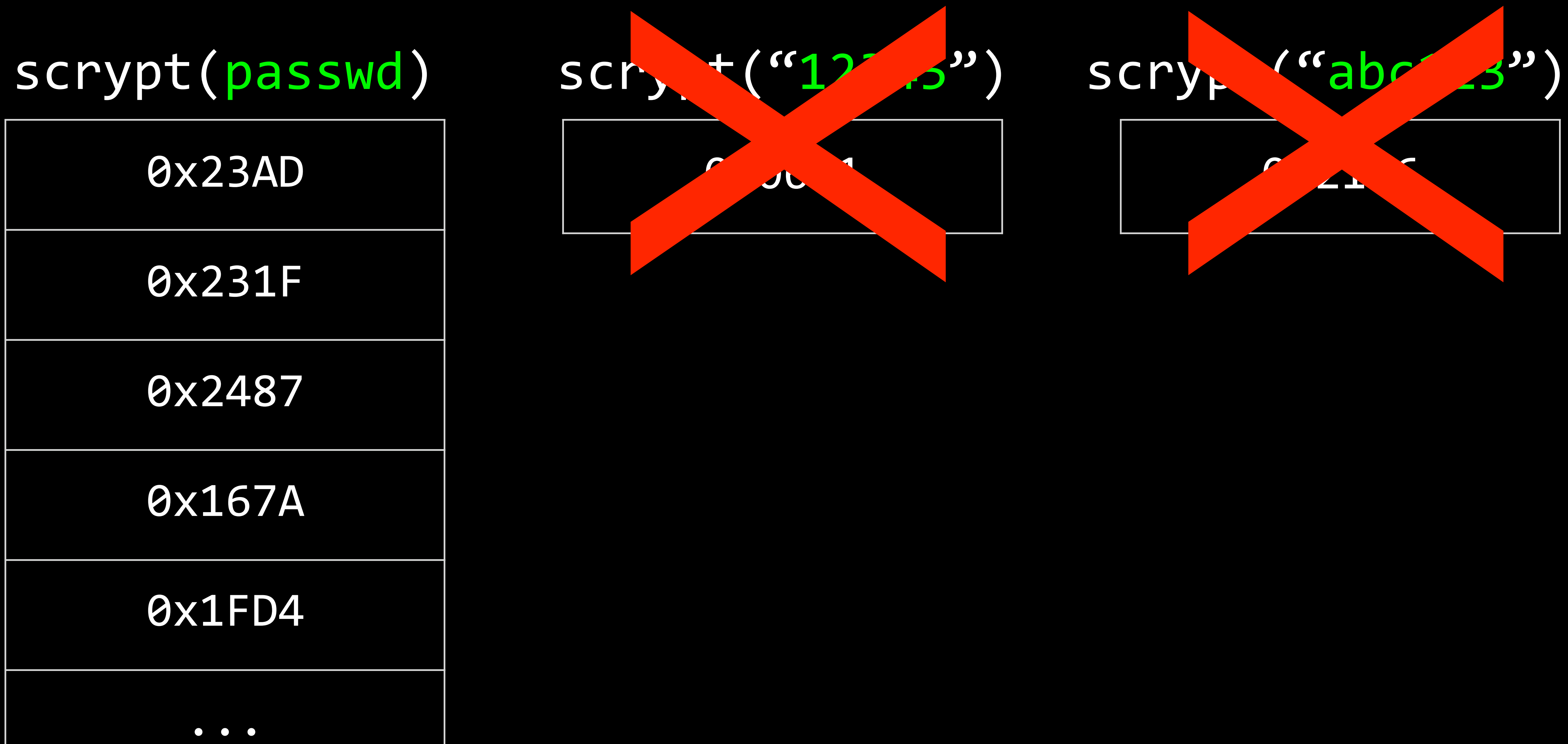
~~scrypt(“12345”)~~

[illegible]

```
scrypt(“abc123”)
0x2176
```

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



scrypt(passwd)

0x23AD

0x231F

0x2487

0x167A

0x1FD4

...

~~scrypt(“12345”)~~

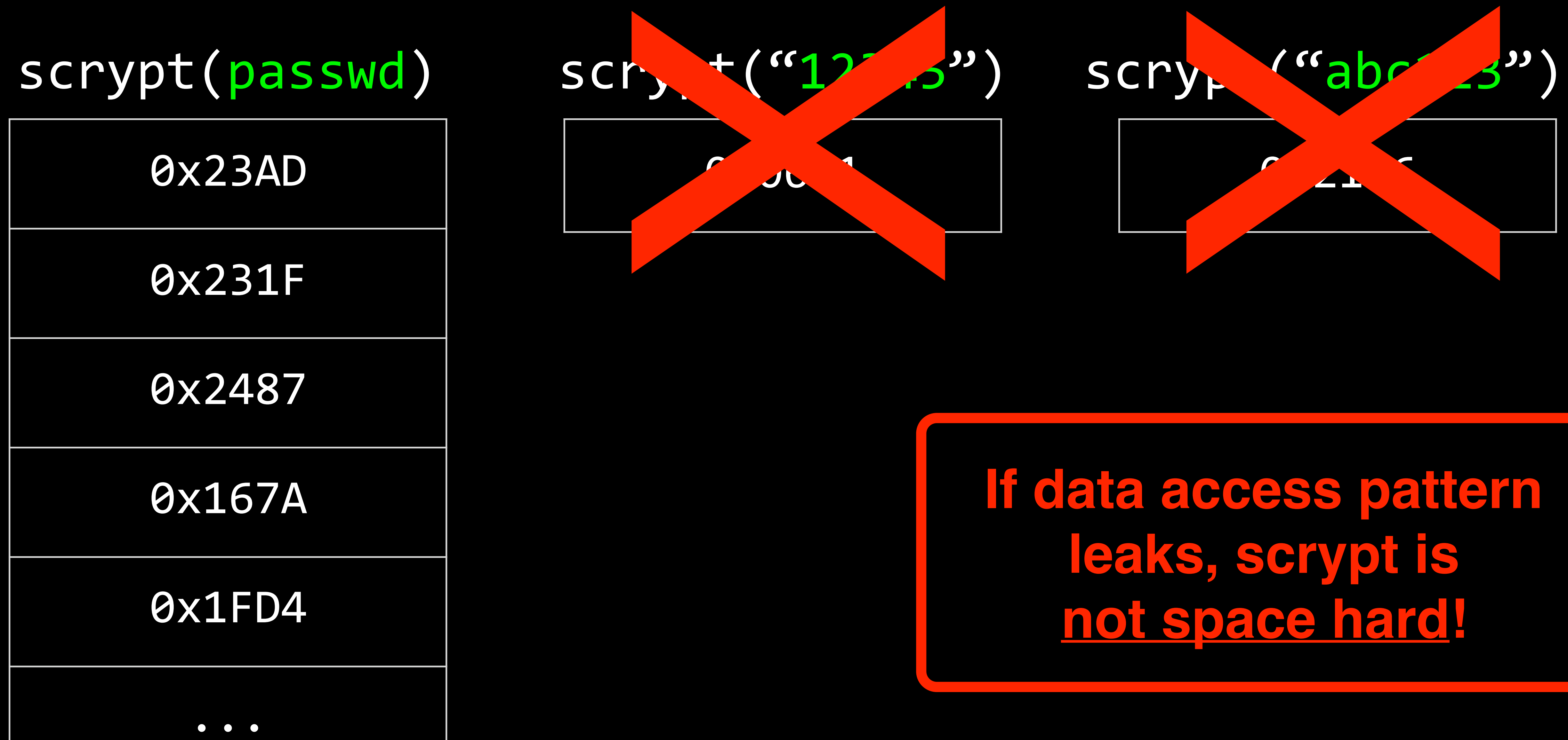
~~0x00[illegible]1~~

~~scrypt(“abc123”)~~

~~0x21[illegible]C~~

Attacking scrypt

An attacker who learns the memory access pattern of `scrypt(passwd)` can run a dictionary attack in *very little space*



`scrypt(passwd)`

```
0x23AD
0x231F
0x2487
0x167A
0x1FD4
...
```

~~scrypt(“12345”)~~

~~[illegible]~~

~~scrypt(“abc123”)~~

~~[illegible]~~

If data access pattern leaks, scrypt is <u>not space hard</u>!