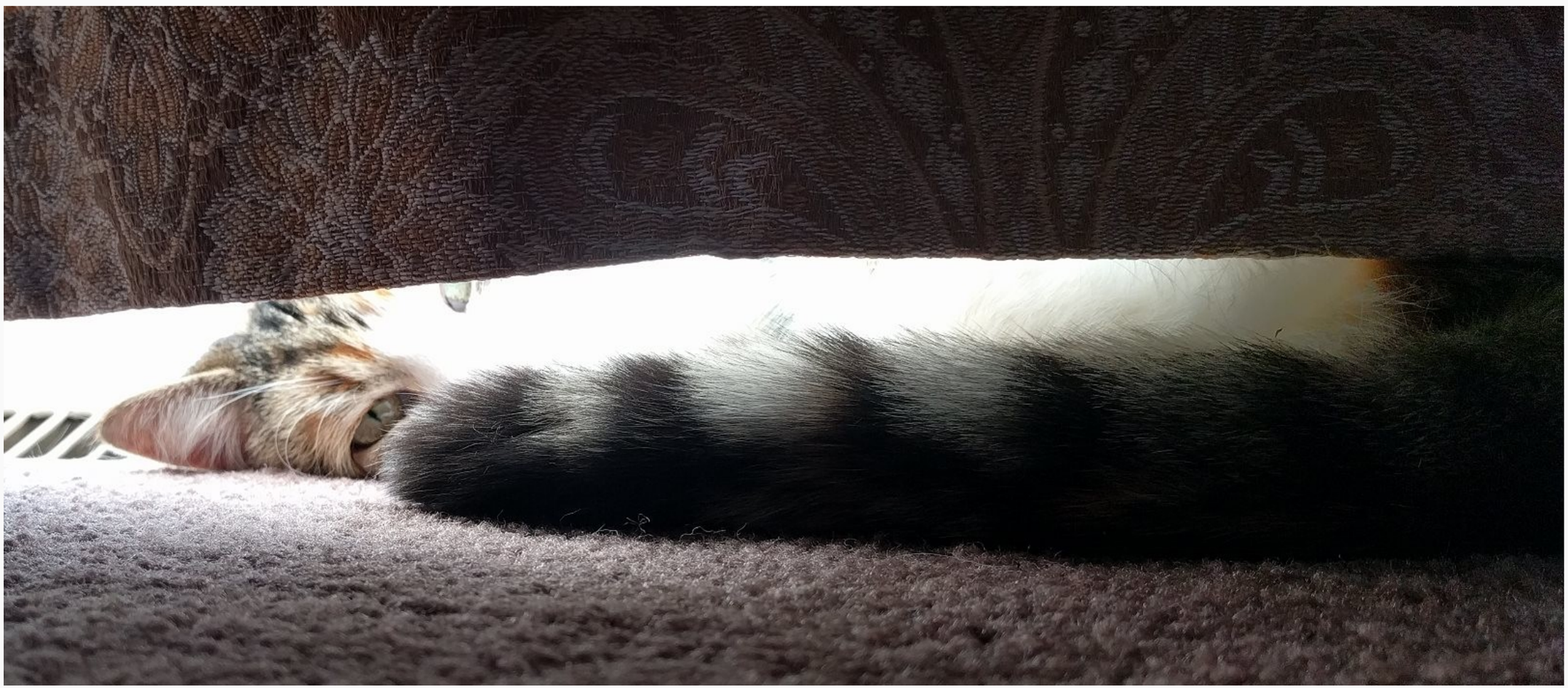
bscrypt
A Cache Hard Password Hash

Steve “Sc00bz” Thomas

Key Stretching



Types of Key Stretching

- Computationally hard
- Memory hard

Types of Key Stretching

- Computationally hard
  - PBKDF2, phpass, md5crypt, shacrypt, ...
- Memory hard
  - Argon2, scrypt, Balloon Hashing, yescrypt, ...

Types of Key Stretching

- Computationally hard
  - Sequential
  - **Parallel**
- Memory hard

Types of Key Stretching

- Sequential Computationally hard
  - PBKDF2, phpass, md5crypt, shacrypt, …
- Parallel Computationally hard
  - ??

Types of Key Stretching

- Sequential Computationally hard
  - PBKDF2, phpass, md5crypt, shacrypt, …
- Parallel Computationally hard
  - “Parallel” and “Parallel PBKDF2”

Parallel PBKDF2

```
work = xorBlocks(
  pbkdf2(password, salt,
    iterations:1024,
    length:128*cost*hashLength))
output =
  pbkdf2(password, work,
    iterations:1,
    length:outputLength)
```

Types of Key Stretching

- Computationally hard
  - Sequential
  - Parallel
- Memory hard
- **Cache hard**

Types of Key Stretching

- Cache hard
  - bcrypt (minimally cache hard at 4 KiB)
  - bscrypt
  - Pufferfish2
  - yescrypt*
  - Argon2ds*

Types of Key Stretching

- Cache hard
  - bscrypt
  - Pufferfish2

How to Key Stretch?

1) seed = H(inputs)
2) work = doWork(settings, seed)
3) key = KDF(outSize, work, seed)

bcrypt (Blowfish)

```
L ^= p[ 0];
R ^= p[ 1] ^ (((   s0[ L >> 24          ]
               + s1[(L >> 16) & 0xff])
               ^ s2[(L >>  8) & 0xff])
               + s3[ L        & 0xff]);
L ^= p[ 2] ^ (((s0[R >> 24] + ...
...
L ^= p[16] ^ (((s0[R >> 24] + ...
R ^= p[17];
```

Pufferfish2

```
L ^= p[ 0];
R ^= p[ 1] ^ (((   s0[ L >> shift     ]
                ^ s1[(L >> 35) & mask])
                + s2[(L >> 19) & mask])
                ^ s3[(L >>  3) & mask]);
L ^= p[ 2] ^ (((s0[R >> shift] ^ ...
...
L ^= p[16] ^ (((s0[R >> shift] ^ ...
R ^= p[17];
```

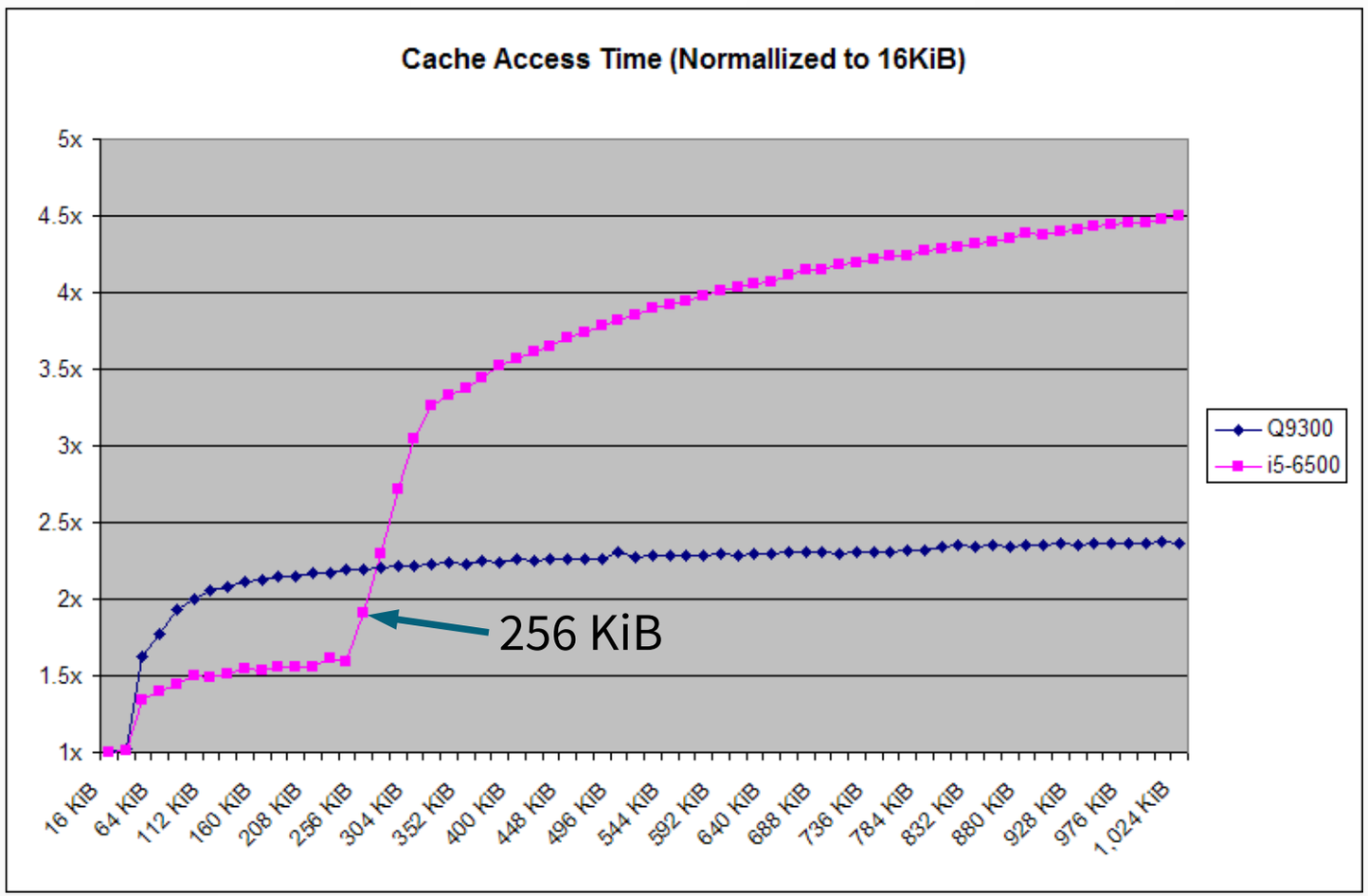
bscrypt Beta (April 2019)

```
R ^= p[ 1] ^ ((((((((
                      s0[ L >> 56          ]
                    + s1[(L >> 48) & 0xff])
                    ^ s2[(L >> 40) & 0xff])
                    + s3[(L >> 32) & 0xff])
                    ^ s4[(L >> 24) & 0xff])
                    + s5[(L >> 16) & 0xff])
                    ^ s6[(L >>  8) & 0xff])
                    + s7[ L        & 0xff]);
```

Weird cache sizes

- Ice Lake (Sep 2019)
  - L1: 48 KiB data
- Tiger Lake (Sep 2020) / Alder Lake (Nov 2021)
  - L1: 48 KiB data
  - L2: 1.25 MiB
- Rocket Lake (May 2021)
  - L1: 48 KiB data

bscrypt Beta (April 2019)



Cache Hard

```
R ^= (((s0[ L >> 24          ]
      + s1[(L >> 16) & 0xff])
      ^ s2[(L >>  8) & 0xff])
      + s3[ L        & 0xff]);
L ^= (((s0[ R >> 24          ]
      + s1[(R >> 16) & 0xff])
      ^ s2[(R >>  8) & 0xff])
      + s3[ R        & 0xff]);
...
```

Accumulators

```
R ^= s0[ L >> 24        ];
R += s1[(L >> 16) & 0xff];
R ^= s2[(L >>  8) & 0xff];
R += s3[ L        & 0xff];
L ^= s0[ R >> 24        ];
L += s1[(R >> 16) & 0xff];
L ^= s2[(R >>  8) & 0xff];
L += s3[ R        & 0xff];
...
```

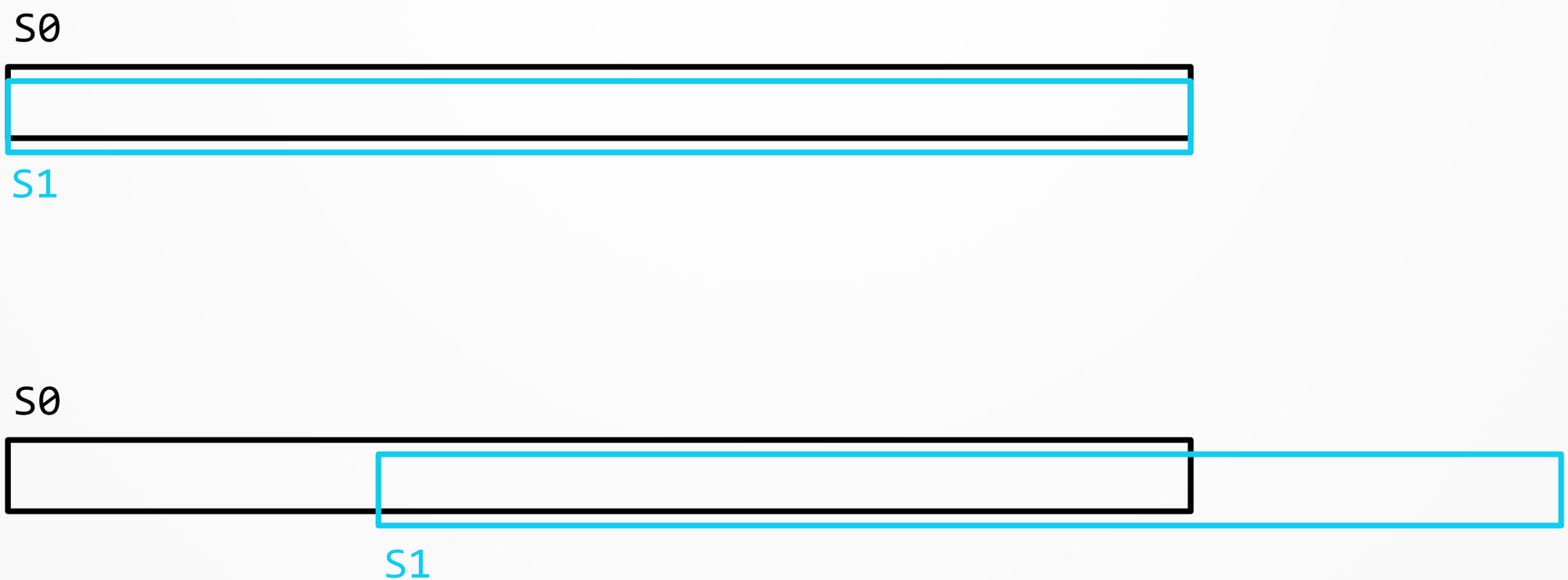
Accumulators

```
(0 + x) ^ x          = 0
((a + x) ^ x) % 2 = a % 2
```

Max S-box Size

- 8 look ups, 8 bit mask: max 2 KiB/S-box
- 4 look ups, 16 bit mask: max 512 KiB/S-box
- 3 look ups, 21 bit mask: max 16 MiB/S-box
- 2 look ups, 32 bit mask: max 32 GiB/S-box
- 1 look up, 64 bit mask: max 128 EiB/S-box

Overlapping S-boxes



Max size with 2 S-boxes

- 8 look ups, 8 bit mask: max 4 KiB
- 4 look ups, 16 bit mask: max 1 MiB
- 3 look ups, 21 bit mask: max 32 MiB
- 2 look ups, 32 bit mask: max 64 GiB
- 1 look up, 64 bit mask: max 256 EiB

Max size with 2 S-boxes

- ~~8 look ups, 8 bit mask: max 4 KiB~~
- 4 look ups, 16 bit mask: max 1 MiB
- 3 look ups, 21 bit mask: max 32 MiB
- 2 look ups, 32 bit mask: max 64 GiB
- ~~1 look up, 64 bit mask: max 256 EiB~~

Max size with 2 S-boxes

- ~~8 look ups, 8 bit mask: max 4 KiB~~
- ~~4 look ups, 16 bit mask: max 1 MiB~~
- 3 look ups, 21 bit mask: max 32 MiB
- 2 look ups, 32 bit mask: max 64 GiB
- ~~1 look up, 64 bit mask: max 256 EiB~~

Max size with 2 S-boxes

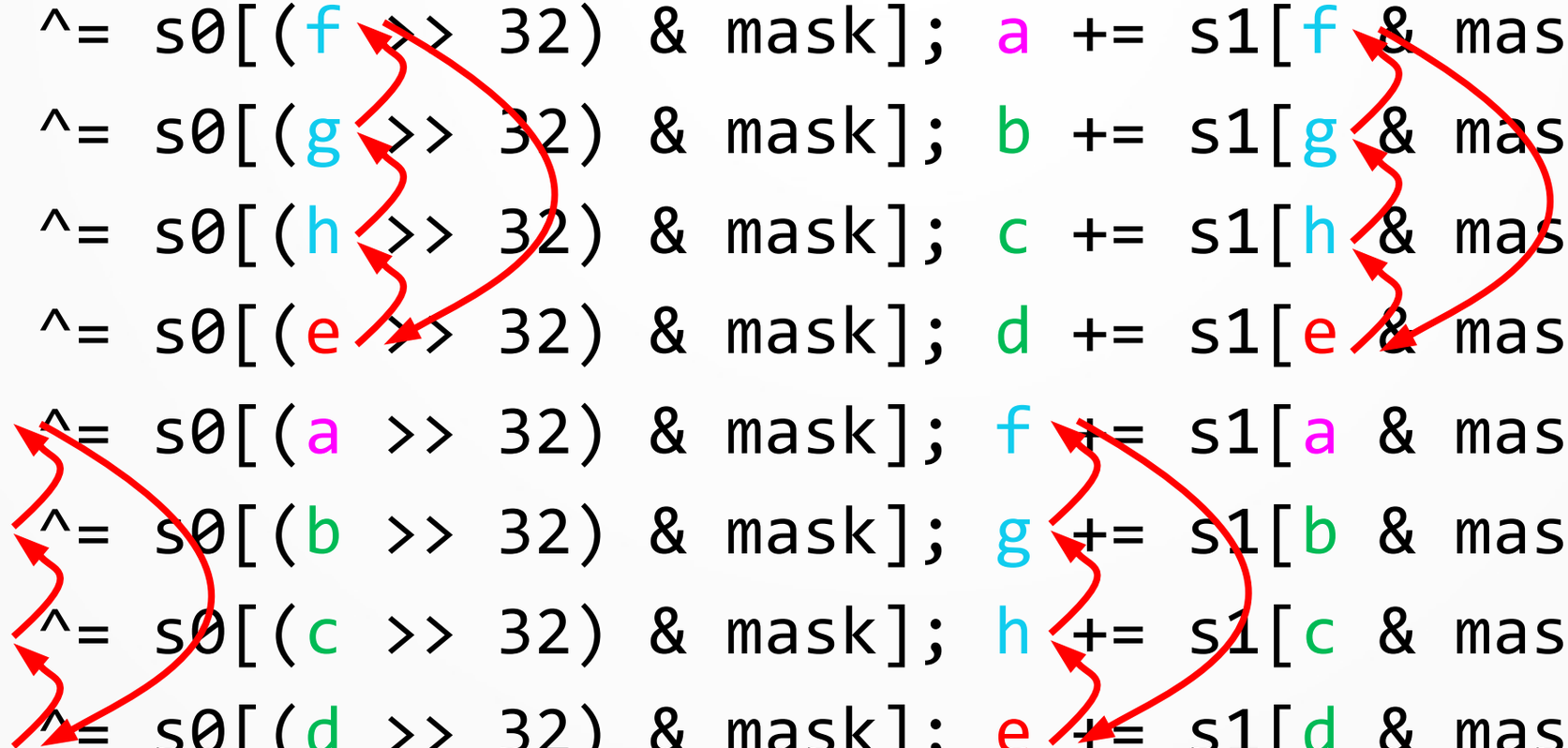
- ~~8 look ups, 8 bit mask: max 4 KiB~~
- ~~4 look ups, 16 bit mask: max 1 MiB~~
- ~~3 look ups, 21 bit mask: max 32 MiB~~
- **2 look ups, 32 bit mask: max 64 GiB**
- ~~1 look up, 64 bit mask: max 256 EiB~~

bscrypt

```
a ^= s0[(e >> 32) & mask]; a += s1[e & mask];
b ^= s0[(f >> 32) & mask]; b += s1[f & mask];
c ^= s0[(g >> 32) & mask]; c += s1[g & mask];
d ^= s0[(h >> 32) & mask]; d += s1[h & mask];
e ^= s0[(a >> 32) & mask]; e += s1[a & mask];
f ^= s0[(b >> 32) & mask]; f += s1[b & mask];
g ^= s0[(c >> 32) & mask]; g += s1[c & mask];
h ^= s0[(d >> 32) & mask]; h += s1[d & mask];
...
```

bscrypt

```
a ^= s0[(f >> 32) & mask]; a += s1[f & mask];
b ^= s0[(g >> 32) & mask]; b += s1[g & mask];
c ^= s0[(h >> 32) & mask]; c += s1[h & mask];
d ^= s0[(e >> 32) & mask]; d += s1[e & mask];
f ^= s0[(a >> 32) & mask]; f += s1[a & mask];
g ^= s0[(b >> 32) & mask]; g += s1[b & mask];
h ^= s0[(c >> 32) & mask]; h += s1[c & mask];
e ^= s0[(d >> 32) & mask]; e += s1[d & mask];
...
```



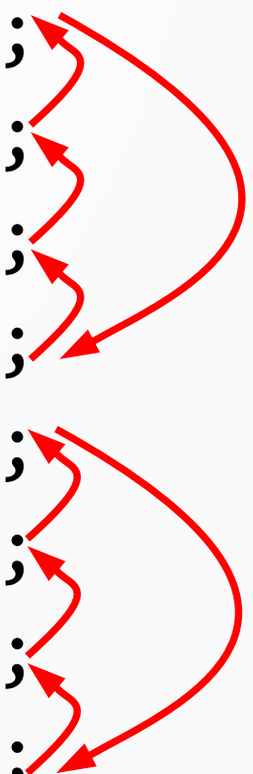
bscrypt

- Read

```
a += s[j    ];
b += s[j + 1];
c += s[j + 2];
d += s[j + 3];
e += s[j + 4];
f += s[j + 5];
g += s[j + 6];
h += s[j + 7];
```

- Write

```
s[j    ] ^= f;
s[j + 1] ^= g;
s[j + 2] ^= h;
s[j + 3] ^= e;
s[j + 4] ^= b;
s[j + 5] ^= c;
s[j + 6] ^= d;
s[j + 7] ^= a;
```



Following “a”

```
a += s[j];
```
Read

```
a ^= s[(e >> 32) & mask];
...
a += s[(f & mask) + s1_offset];
```
S-boxes

```
s[j + 7] ^= a;
```
Write

```
next_j = j + 8;
a += s[next_j];
```
Read next

Following “a”

```
a += s[j];
```
Read

```
a ^= s[(e >> 32) & mask];
...
a += s[(f & mask) + s1_offset];
```
S-boxes

```
s[j + 7] ^= a;
```
Write

```
next_j = j + 8;
a += s[next_j];
```
Read next

Following “a”

```
a += s[j];
```
Read

```
a ^= s[(e >> 32) & mask];
...
a += s[(f & mask) + s1_offset];
```
S-boxes

```
s[j + 7] ^= a;
```
Write

Block starts with ADD

```
next_j = j + 8;
a ^= s[next_j];
```
Read next

Block starts with XOR

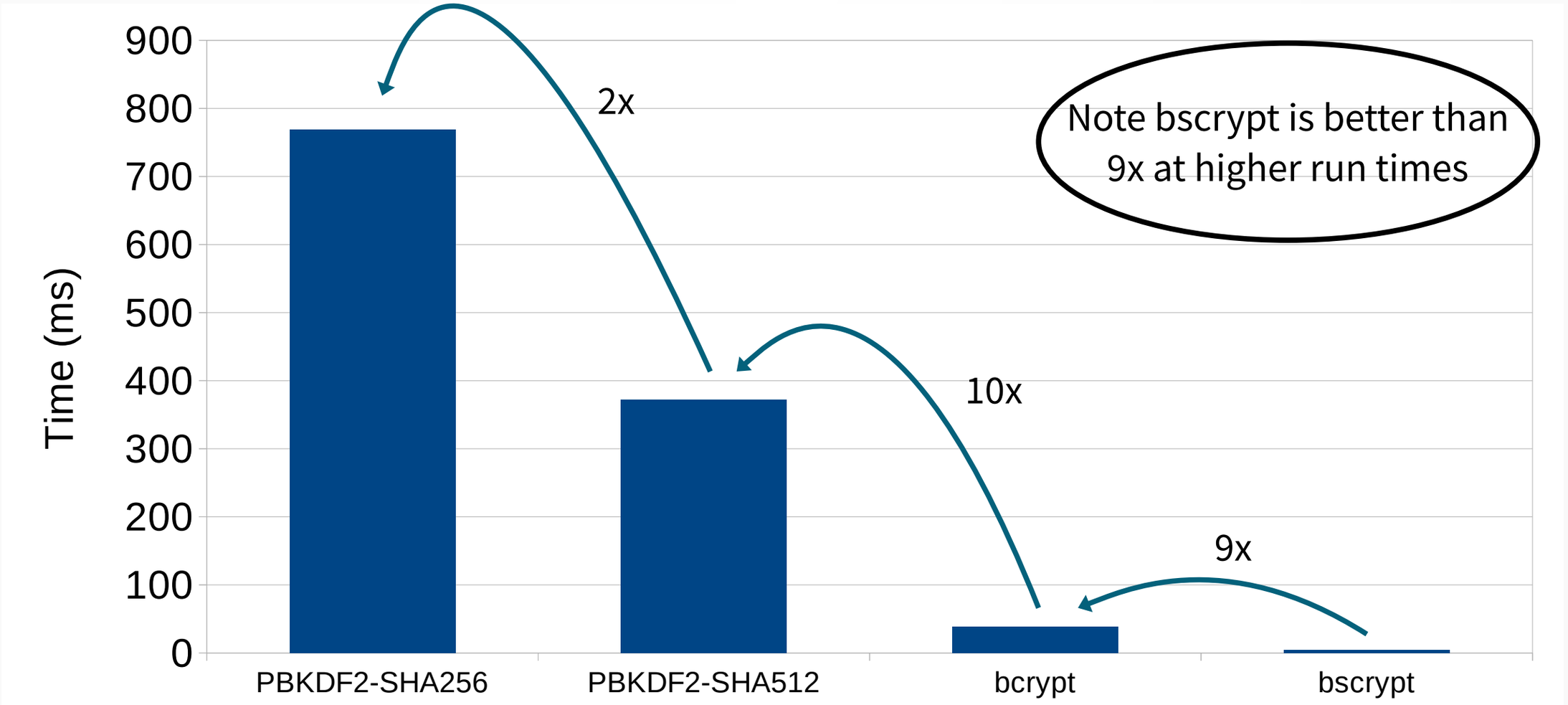
bscrypt

```
a = ROTR64(a, 15);
b = ROTR64(b, 35);
c = ROTR64(c, 17);
d = ROTR64(d, 41);
e = ROTR64(e, 21);
f = ROTR64(f, 45);
g = ROTR64(g, 27);
h = ROTR64(h, 47);
```

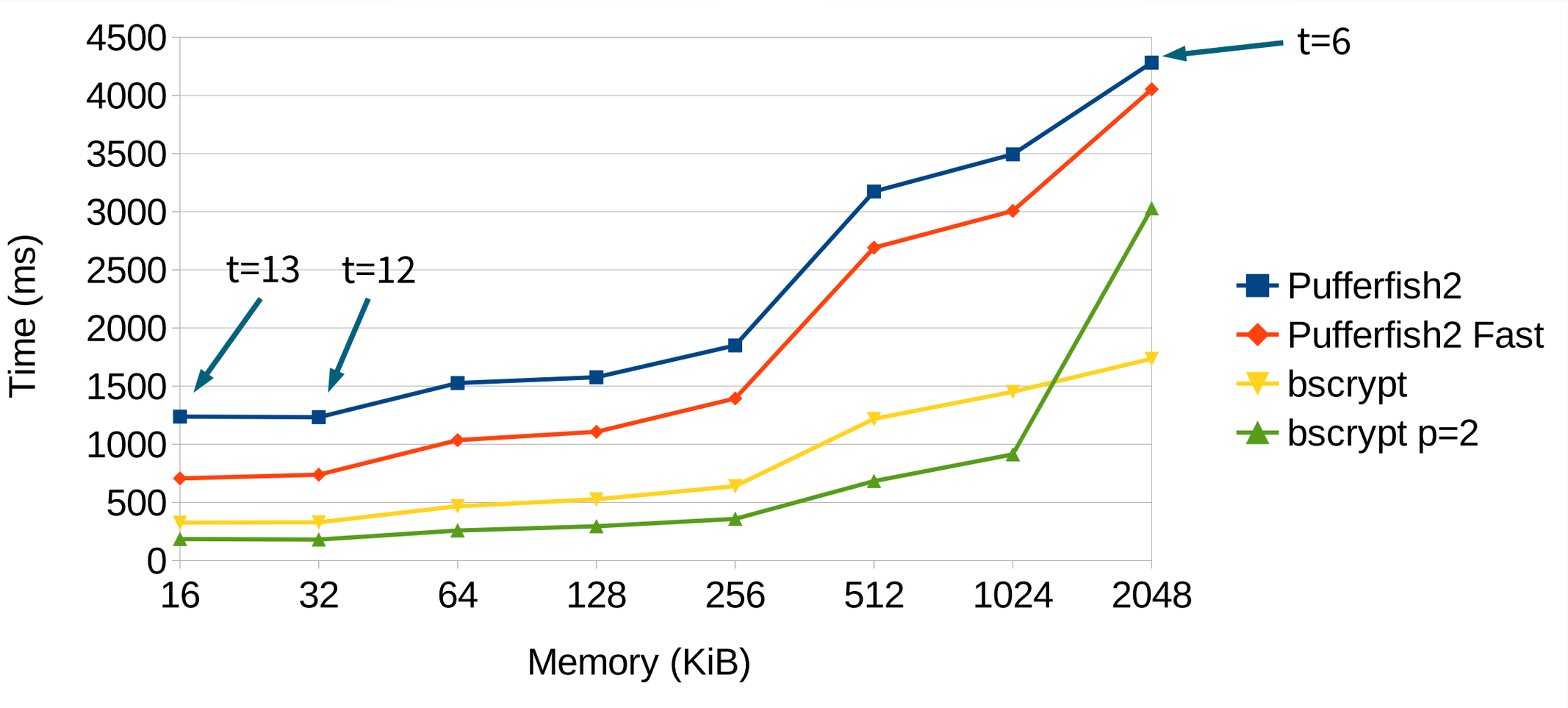
Initial Fill and Finish

This could be you

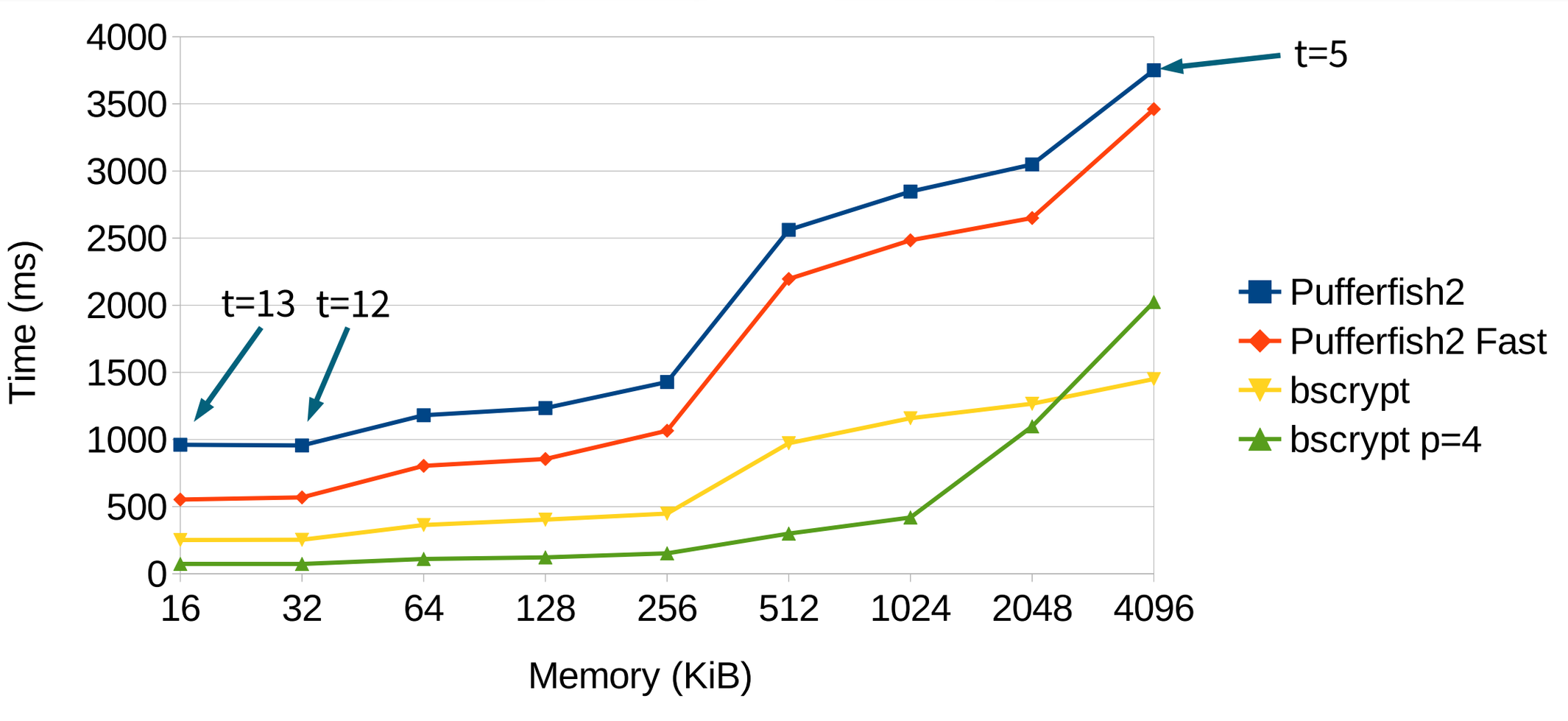
i5-6200U: Settings for ~5300 KH/s/GPU



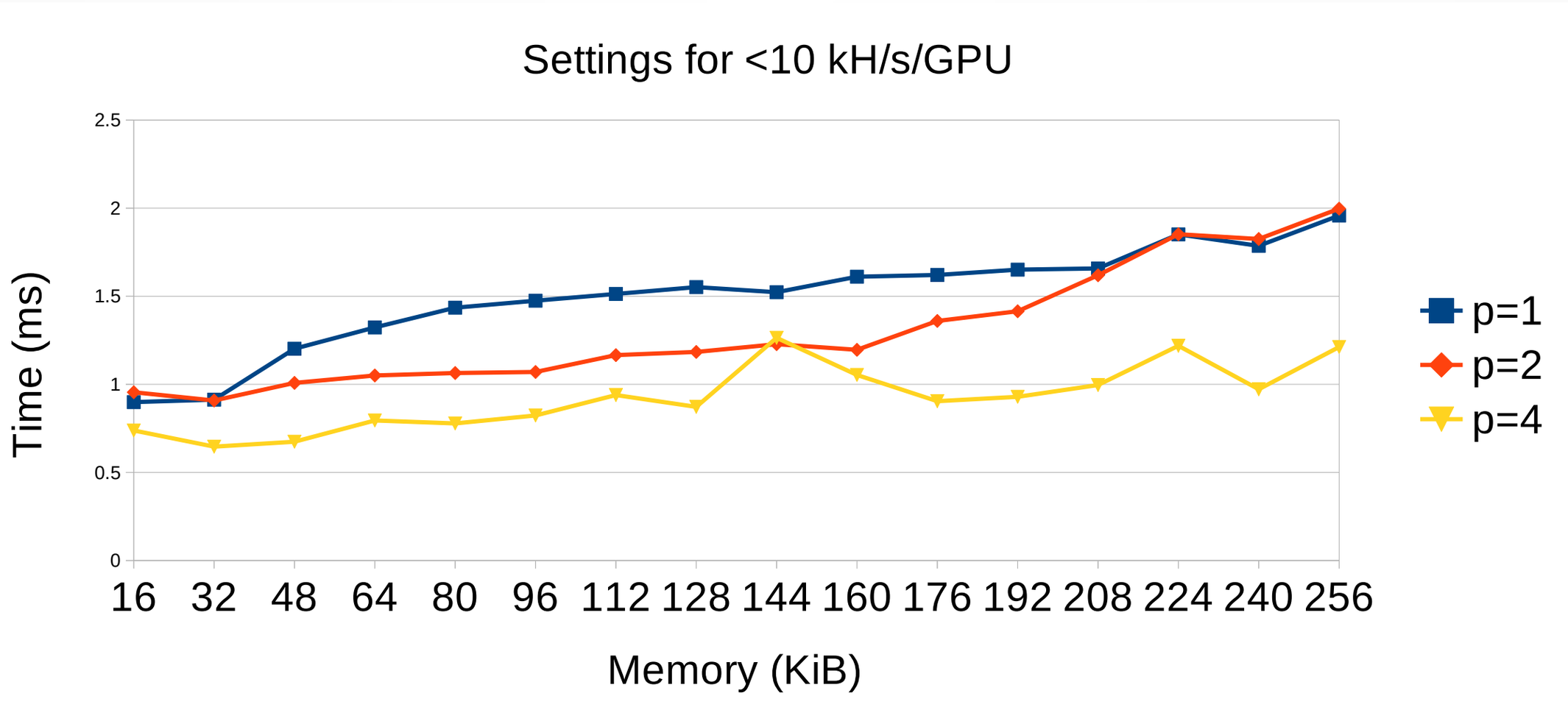
i5-6200U: 32 KiB L1, 256 KiB L2, 3 MiB L3



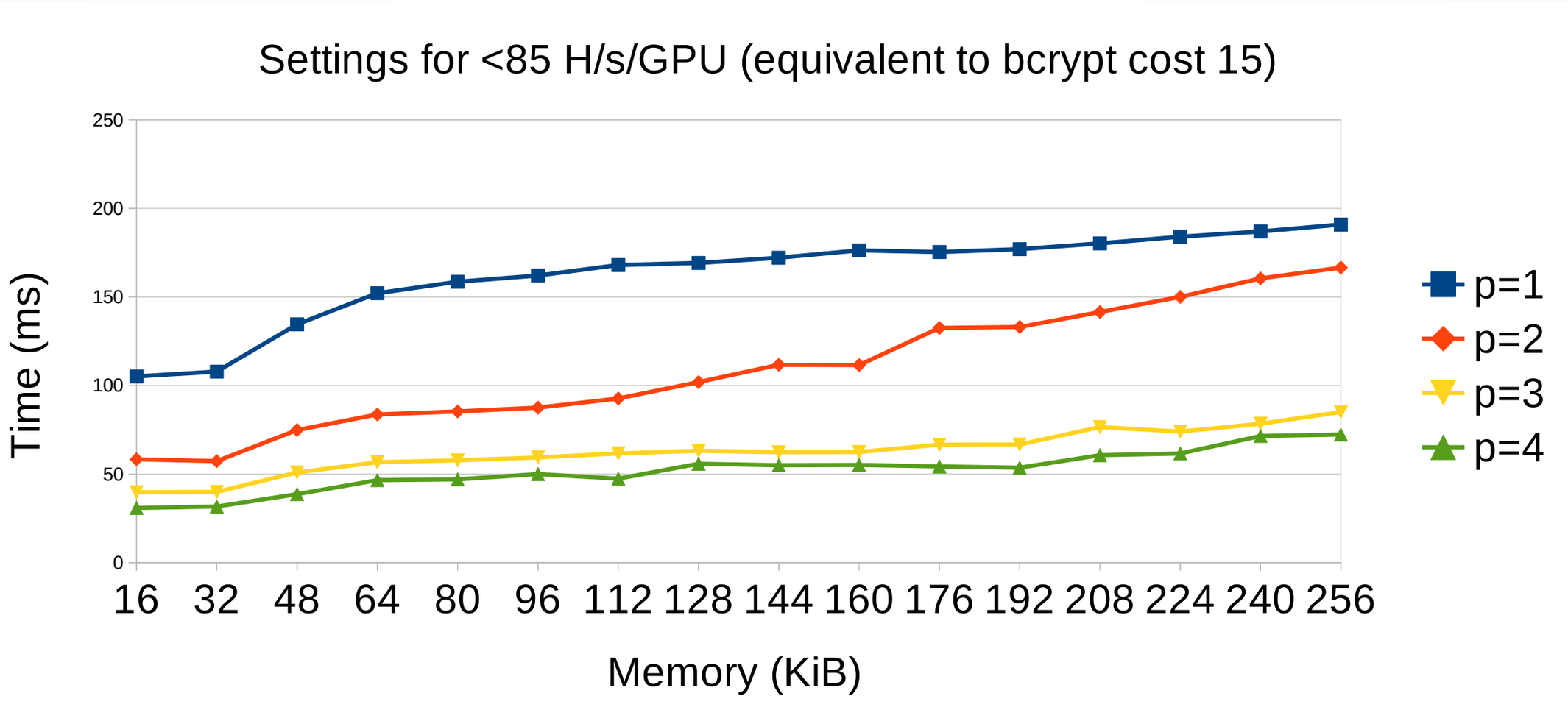
i5-6500: 32 KiB L1, 256 KiB L2, 6 MiB L3



i5-6500: 32 KiB L1, 256 KiB L2, 6 MiB L3



i5-6500: 32 KiB L1, 256 KiB L2, 6 MiB L3



Questions?

- Twitter: @Sc00bzT
- Github: Sc00bz (https://github.com/Sc00bz/bscrypt)
- steve at tobtu.com