LevelDB Study
Bloom Filter Analysis

Made by Kim Han Su

E-Mail: khs20010327@naver.com

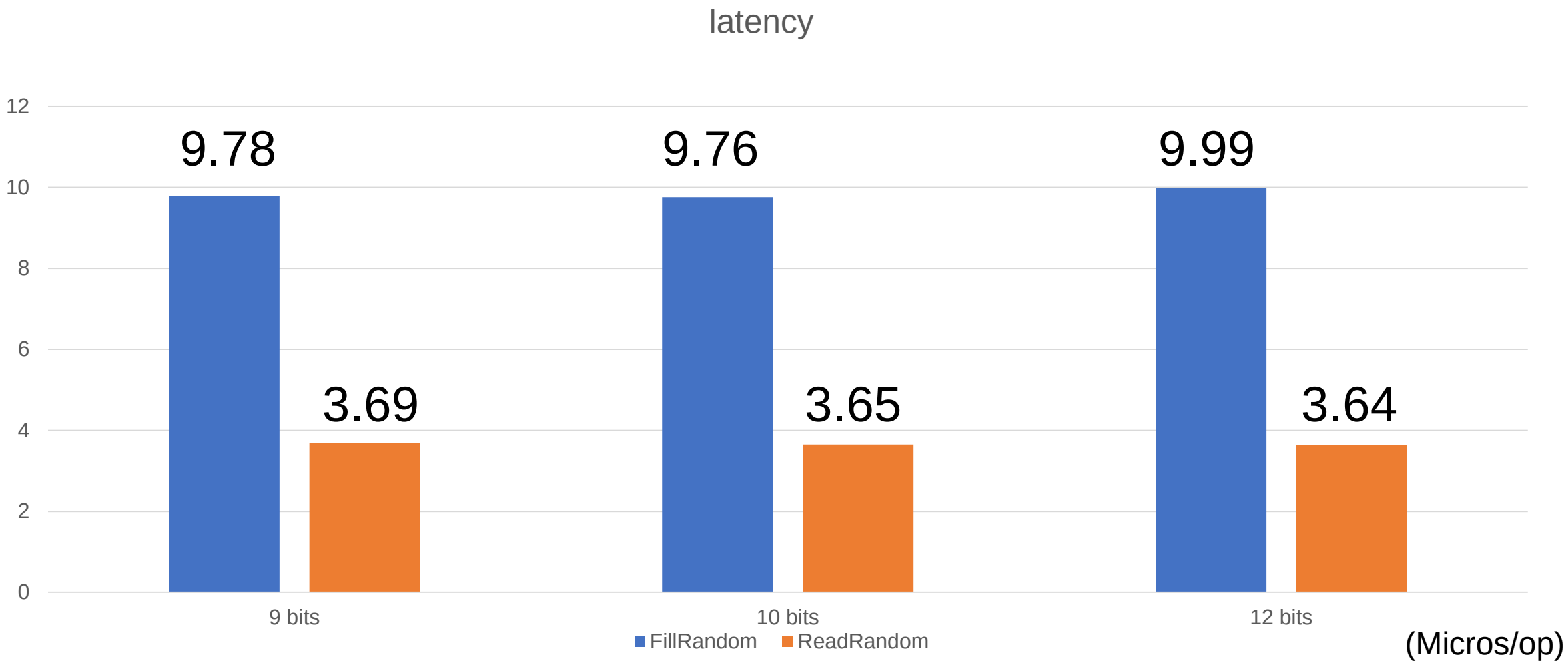
Contents

- More details about last presentation
- About “ReadMissing”

Hypothesis

- LevelDB는 ln2 * bits_per_key개의 해시 함수를 사용
- ->Bits per key 값이 10이라면 이론상 6.9개의 해시 함수를 사용해야함
- ->실제론 6개를 사용
- 9 -> 9 X 0.69 = 6.21 -> 6
- 12 -> 12 X 0.69 = 8.28 -> 8

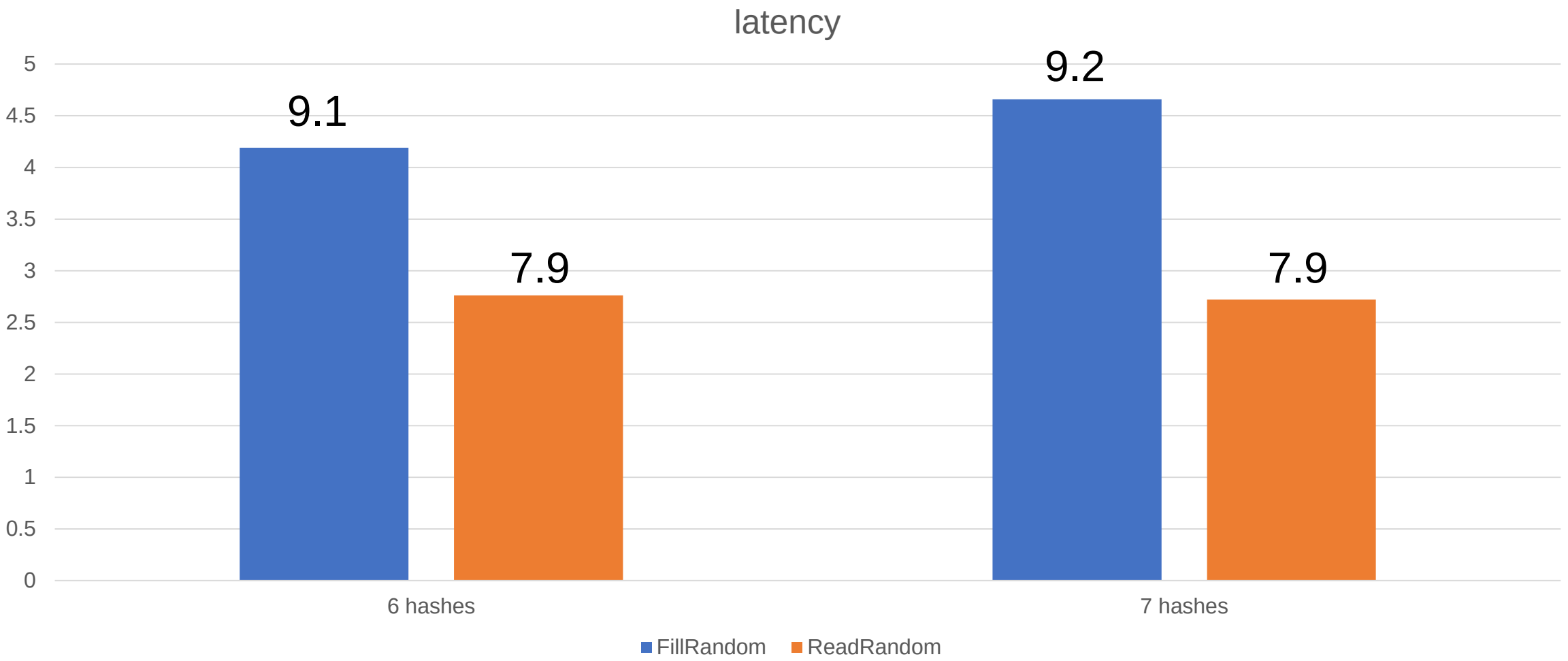
Result


latency
12
10
8
6
4
2
0
9.78
3.69
9.76
3.65
9.99
3.64
9 bits
10 bits
12 bits
FillRandom
ReadRandom
(Micros/op)

Code edit

```
class BloomFilterPolicy : public FilterPolicy {
 public:
  explicit BloomFilterPolicy(int bits_per_key) : bits_per_key_(bits_per_key) {
    // We intentionally round down to reduce probing cost a little bit
    k_ = static_cast<size_t>(bits_per_key * 0.71);  // 0.69 =~ ln(2)
    if (k_ < 1) k_ = 1;
    if (k_ > 30) k_ = 30;
  }
```

Result


latency
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
9.1
7.9
9.2
7.9
6 hashes
7 hashes
FillRandom
ReadRandom

Mathematical calculation

$$\left(1 - e^{-\ln 2}\right)^{\ln 2 \cdot 10}$$

소수 형태:

$0.00819254\ldots$

$$\left(1 - e^{\frac{-7}{10}}\right)^{7}$$

소수 형태:

$0.00819372\ldots$

$$\left(1 - e^{\frac{-6}{10}}\right)^{6}$$

소수 형태:

$0.00843620\ldots$

Mathematical calculation

$$\left(1 - e^{\frac{-7}{10}}\right)^7$$

소수 형태:

0.00819372...

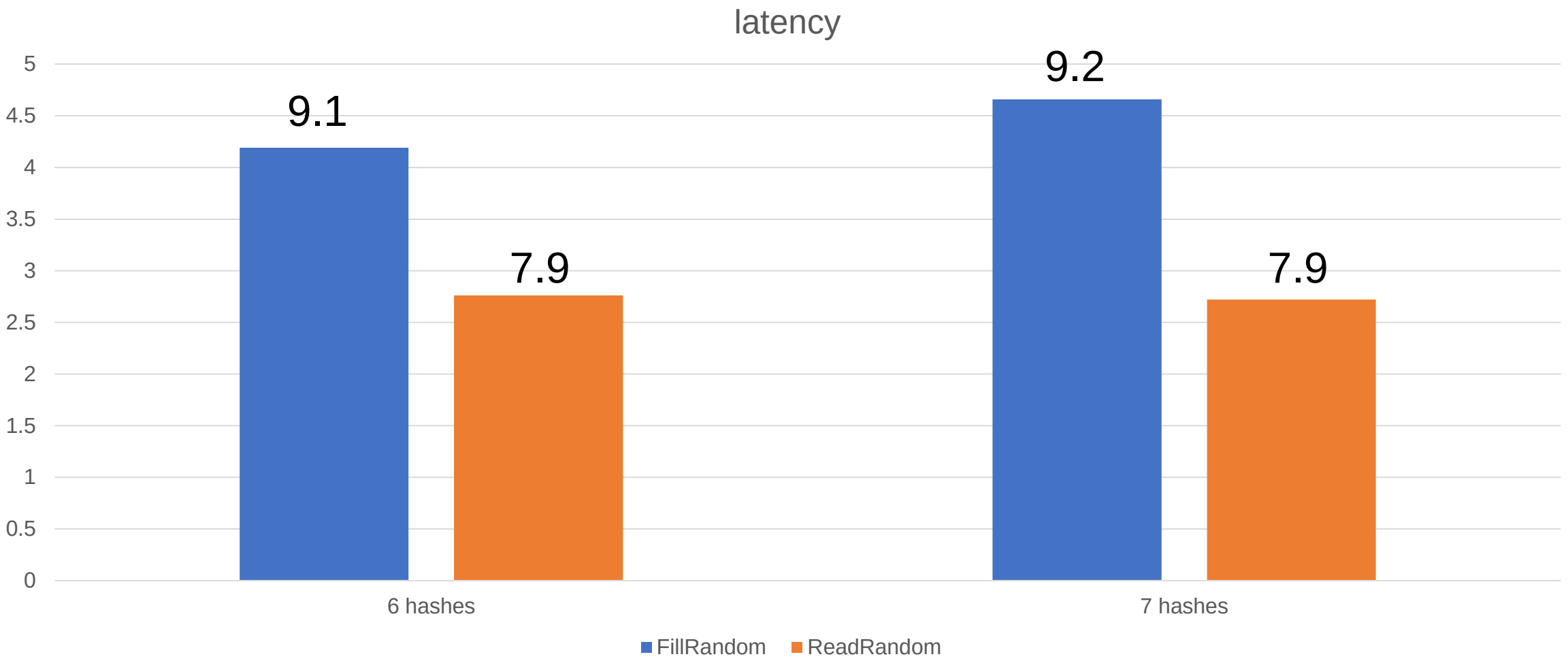
$$\left(1 - e^{\frac{-6}{10}}\right)^6$$

소수 형태:

0.00843620...

- 0.00024 => 240 per 1 Million

Result


latency
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
9.1
7.9
9.2
7.9
6 hashes
7 hashes
FillRandom
ReadRandom

Mathematical calculation

$$\left(1 - e^{\frac{-5}{10}}\right)^5$$

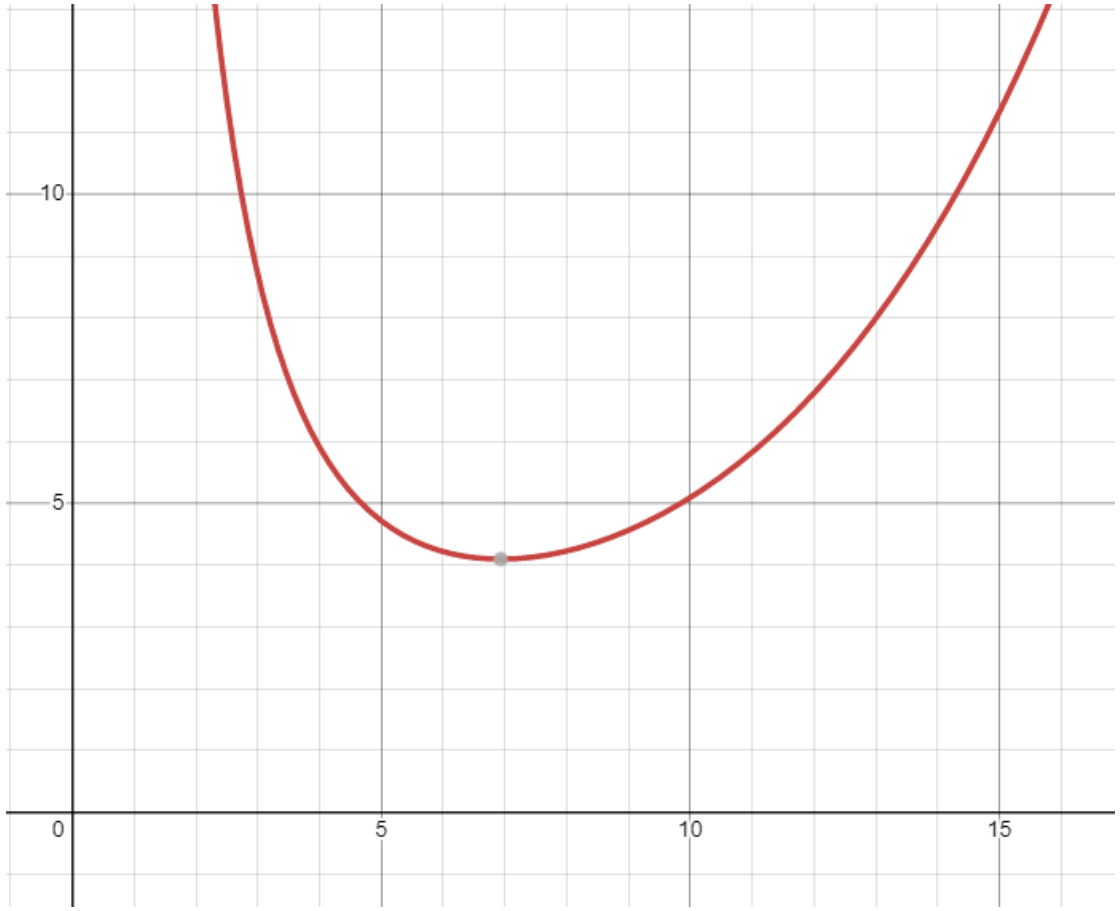
소수 형태:

$0.00943092\ldots$

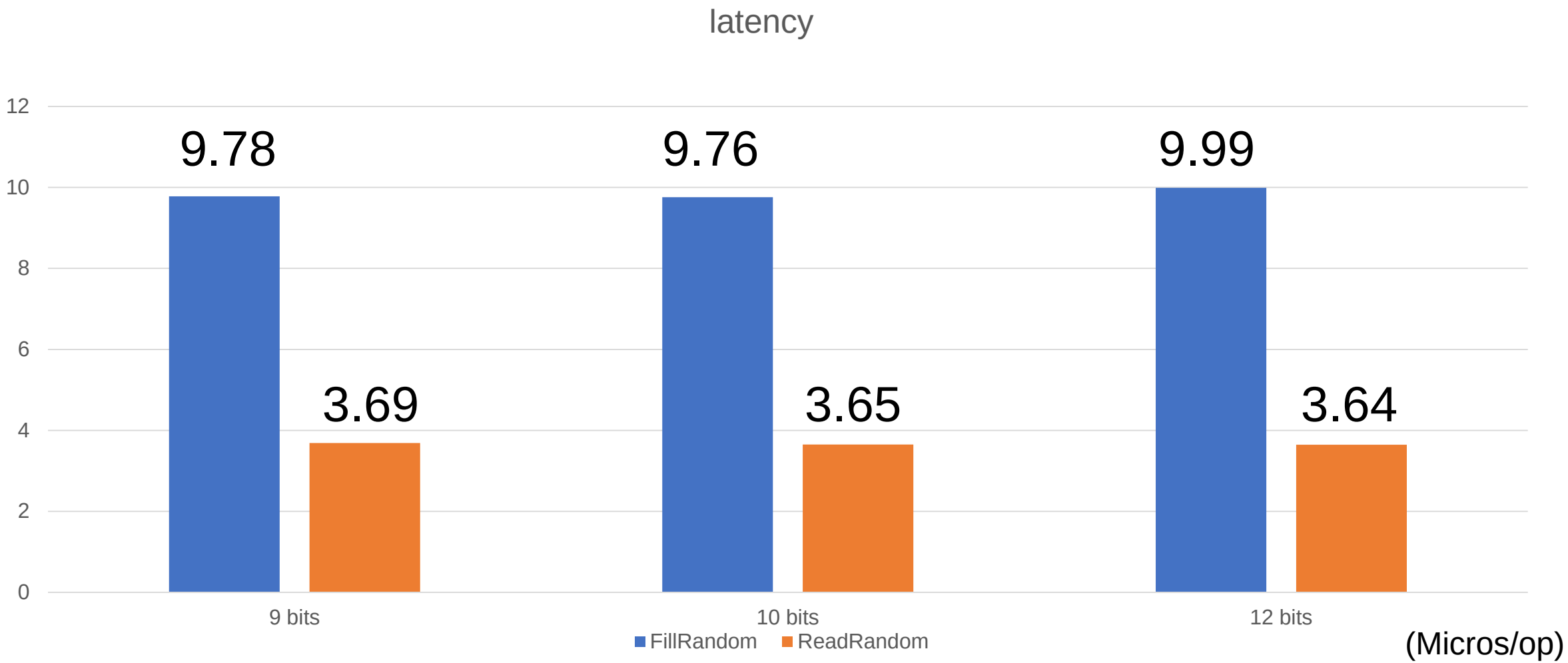
$$\left(1 - e^{\frac{-8}{10}}\right)^8$$

소수 형태:

$0.00845547\ldots$



Result


latency
12
10
8
6
4
2
0
9.78
3.69
9.76
3.65
9.99
3.64
9 bits
10 bits
12 bits
FillRandom
ReadRandom
(Micros/op)

Mathematical calculation

$$\left(1 - e^{\frac{-8}{12}}\right)^8$$

소수 형태:

$0.00314235\ldots$

$$\left(1 - e^{\frac{-6}{10}}\right)^6$$

소수 형태:

$0.00843620\ldots$

$$\left(1 - e^{\frac{-6}{9}}\right)^6$$

소수 형태:

$0.01327213\ldots$

Result

$$\left(1 - e^{\frac{-8}{12}}\right)^8$$

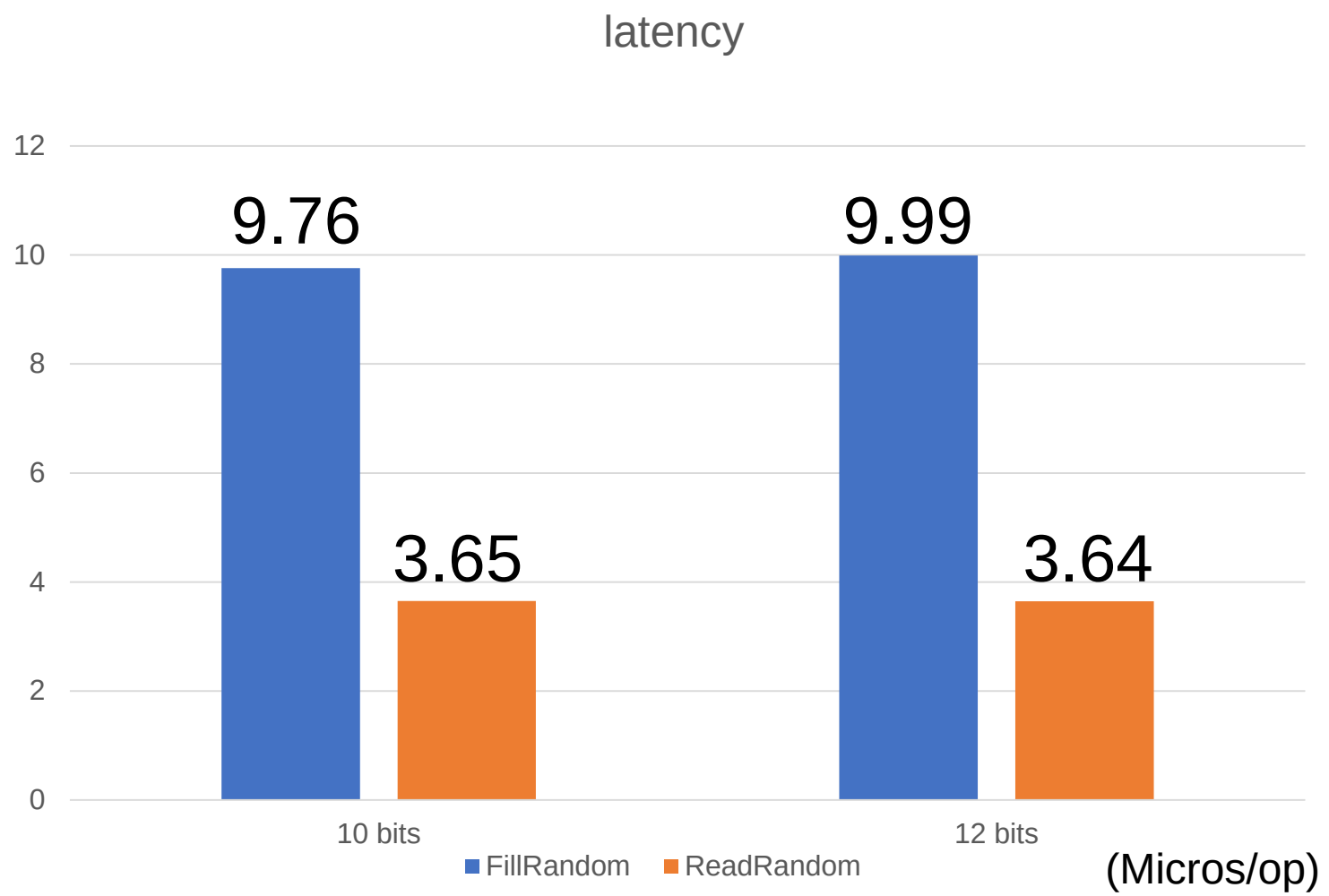
소수 형태:

0.00314235 . . .

$$\left(1 - e^{\frac{-6}{10}}\right)^6$$

소수 형태:

0.00843620 . . .



Mathematical calculation

$$\left(1 - e^{\frac{-6}{10}}\right)^6$$

소수 형태:

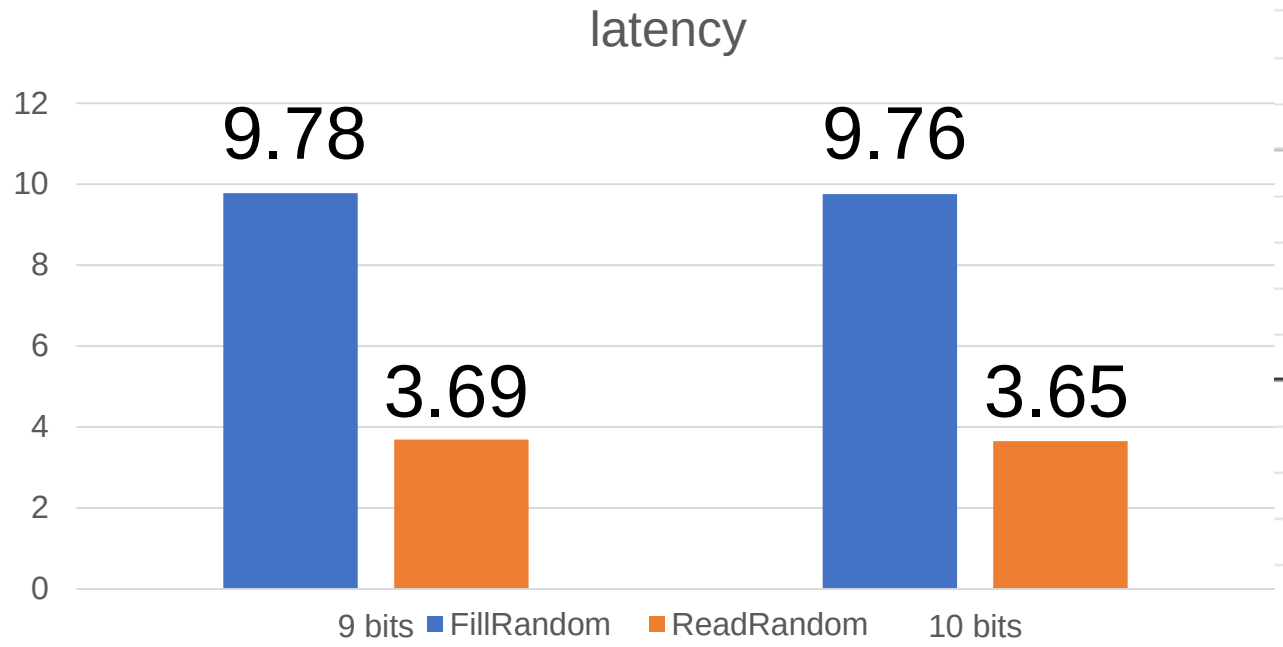
0.00843620 . . .

$$\left(1 - e^{\frac{-6}{9}}\right)^6$$

소수 형태:

0.01327213 . . .

$$y = \left(1 - e^{\left(-\frac{6}{x}\right)}\right)6$$



Conclusion

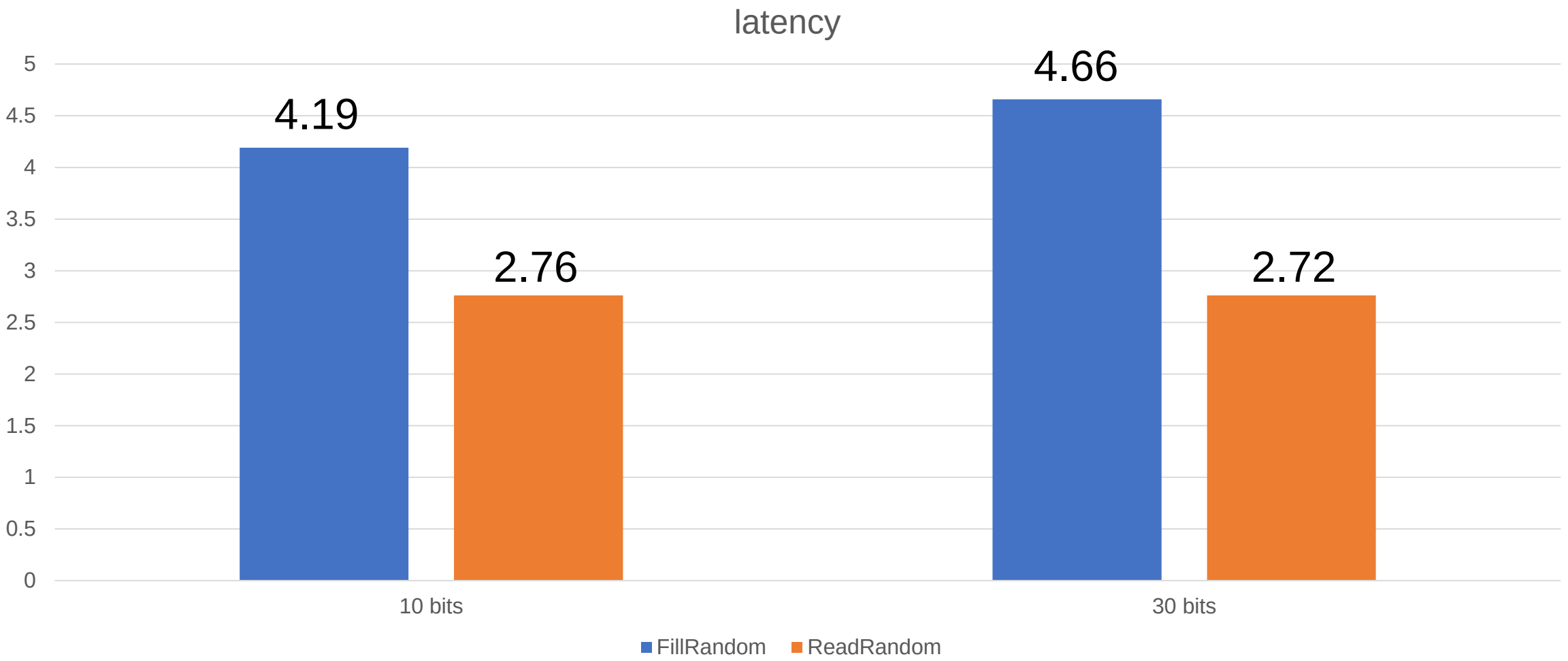
- 1.같은 hash면 bloom_bits값이 큰 쪽이 읽기 성능이 좋다
- 2.hash 개수에 비해 bloom_bits가 false positive에 영향이 크다

$$\left(1 - e^{-\frac{6}{10}}\right)^{6}$$

Code edit 2

```
class BloomFilterPolicy : public FilterPolicy {
 public:
  explicit BloomFilterPolicy(int bits_per_key) : bits_per_key_(bits_per_key) {
    // We intentionally round down to reduce probing cost a little bit
    k_ = static_cast<size_t>(bits_per_key * 0.69);  // 0.69 =~ ln(2)
    if (k_ < 1) k_ = 6;
  }
```

Result


latency
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
4.19
2.76
4.66
2.72
10 bits
30 bits
FillRandom
ReadRandom

Readmissing?

```
readseq      -- read N times sequentially
readreverse  -- read N times in reverse order
readrandom   -- read N times in random order
readmissing  -- read N missing keys in random order
readhot      -- read N times in random order from 1% section of DB
```

Readmissing

```
void ReadRandom(ThreadState* thread) {
  ReadOptions options;
  std::string value;
  int found = 0;
  KeyBuffer key;
  for (int i = 0; i < reads_; i++) {
    const int k = thread->rand.Uniform(FLAGS_num);
    key.Set(k);
    if (db_->Get(options, key.slice(), &value).ok()) {
      found++;
    }
    thread->stats.FinishedSingleOp();
  }
  char msg[100];
  std::snprintf(msg, sizeof(msg), "(%d of %d found)", found,
  thread->stats.AddMessage(msg);
}
```

```
void ReadMissing(ThreadState* thread) {
  ReadOptions options;
  std::string value;
  KeyBuffer key;
  for (int i = 0; i < reads_; i++) {
    const int k = thread->rand.Uniform(FLAGS_num);
    key.Set(k);
    Slice s = Slice(key.slice().data(), key.slice().size() - 1);
    db_->Get(options, s, &value);
    thread->stats.FinishedSingleOp();
  }
}
```

Readmissing

```
namespace {
static uint32_t BloomHash(const Slice& key) {
  return Hash(key.data(), key.size(), 0xbc9f1d34);
}
```

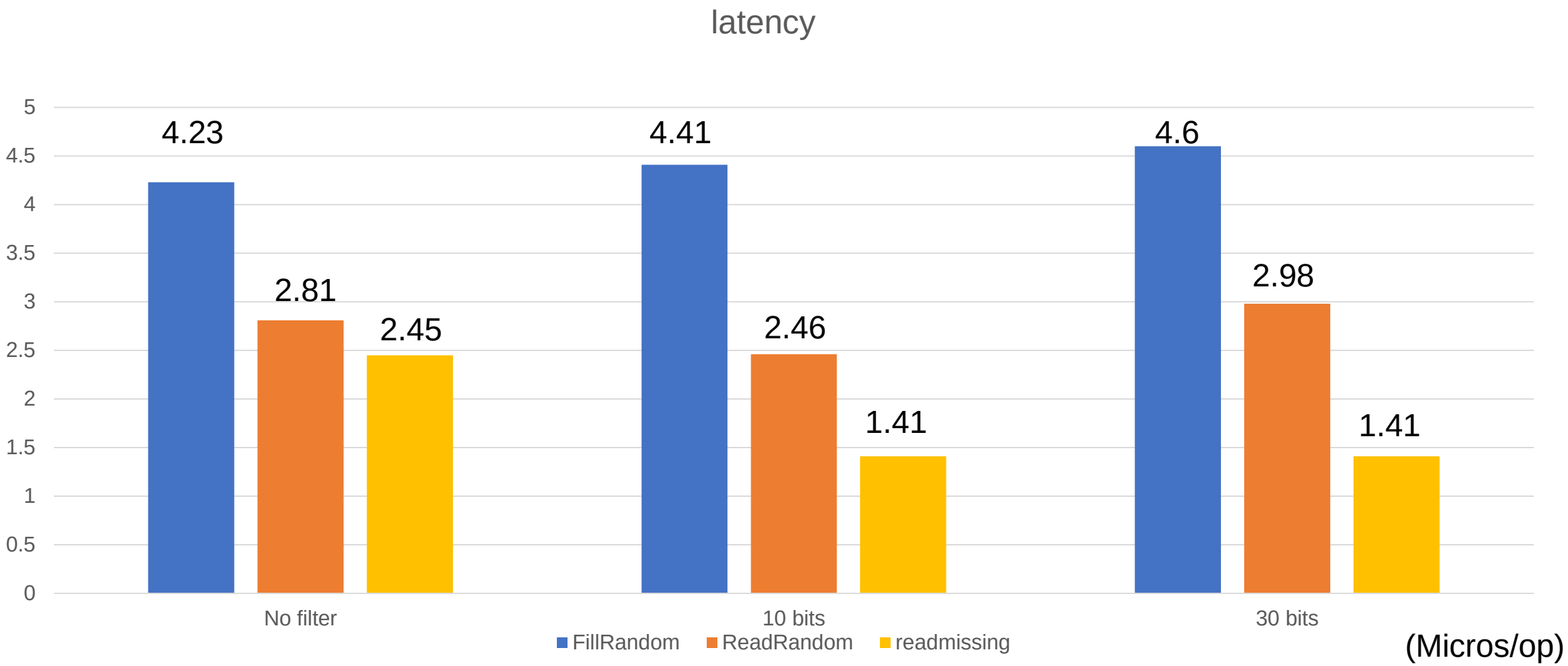
```
uint32_t Hash(const char* data, size_t n, uint32_t seed) {
  // Similar to murmur hash
  const uint32_t m = 0xc6a4a793;
  const uint32_t r = 24;
  const char* limit = data + n;
  uint32_t h = seed ^ (n * m);
```

Readmissing

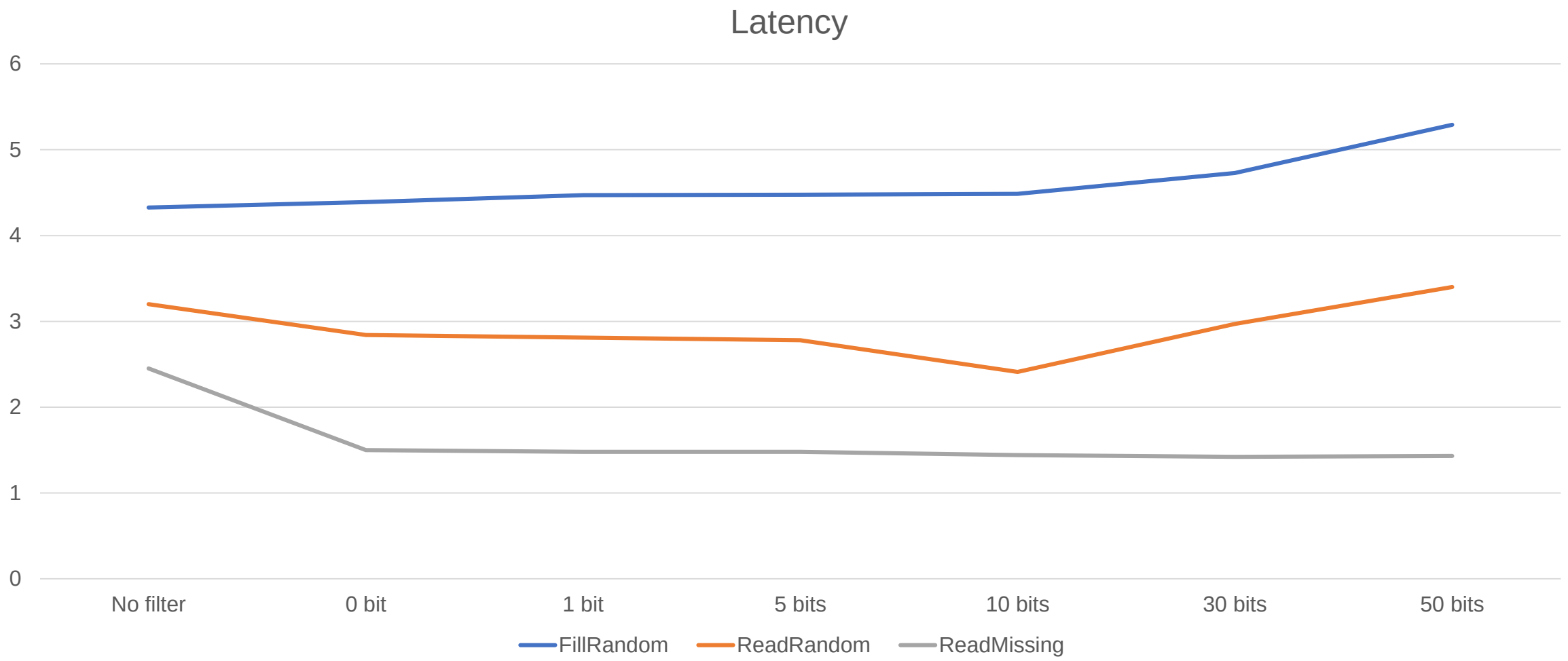
```
void ReadRandom(ThreadState* thread) {
  ReadOptions options;
  std::string value;
  int found = 0;
  KeyBuffer key;
  for (int i = 0; i < reads_; i++) {
    const int k = thread->rand.Uniform(FLAGS_num);
    key.Set(k);
    Slice s = Slice(key.slice().data(), key.slice().size() - 1);
    if (db_->Get(options, s, &value).ok()) {
      found++;
    }
    thread->stats.FinishedSingleOp();
```

```
fillrandom   :       3.344 micros/op;   33.1 MB/s
readrandom   :       3.263 micros/op; (0 of 1000000 found)
```

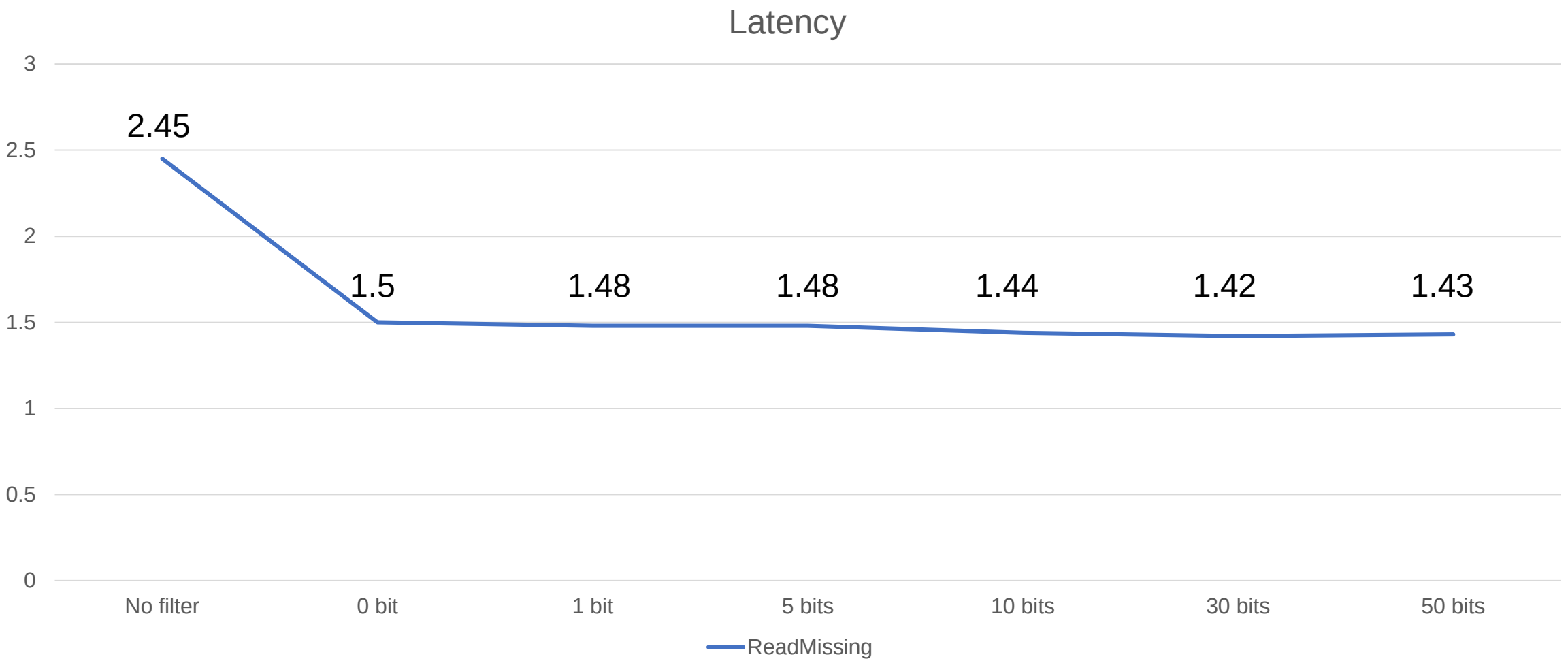
Result


latency
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
4.23
2.81
2.45
4.41
2.46
1.41
4.6
2.98
1.41
No filter
10 bits
30 bits
FillRandom
ReadRandom
readmissing
(Micros/op)

Result


Latency
6
5
4
3
2
1
0
No filter
0 bit
1 bit
5 bits
10 bits
30 bits
50 bits
FillRandom
ReadRandom
ReadMissing

Result


Latency
3
2.5
2
1.5
1
0.5
0
2.45
1.5
1.48
1.48
1.44
1.42
1.43
No filter
0 bit
1 bit
5 bits
10 bits
30 bits
50 bits
ReadMissing

Question