

# Roofline model

Serial machines

# What is the maximum performance possible for a given algorithm?

- What is your bottleneck?
  - Disk?
  - Math ops?
  - Reading from cache?
  - Reading from main memory?

# First a little on counting reads and ops

```
for(i2=0; i2 < n2; i2++){  
    out[i2][1] += in[i2][0] * val[i2]  
}
```

# First a little on counting reads and ops

Reads: 5

Writes: 1

```
for(i2=0; i2 < n2; i2++){  
    out[i2][1] += in[i2][0] * val[i2]  
}
```

# First a little on counting reads and ops

Reads: 5

Writes: 1

```
for(i2=0; i2 < n2; i2++){  
    out[i2][1] += in[i2][0] * val[i2]  
}
```

Multiply: 1

Add: 1

# First a little on counting reads and ops

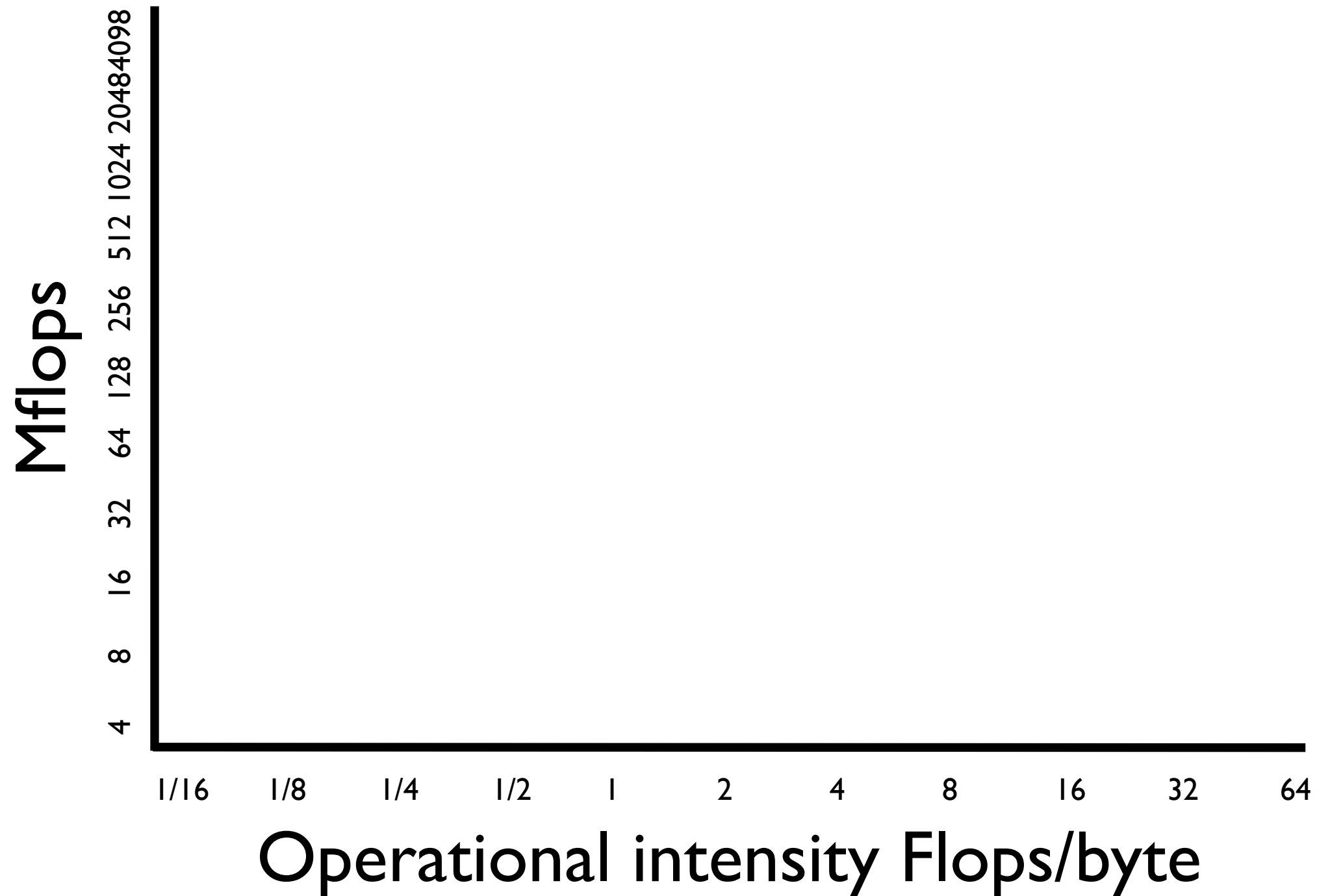
Reads: 5  
Writes: 1

```
for(i2=0; i2 < n2; i2++){  
    out[i2][1] += in[i2][0] * val[i2]  
}
```

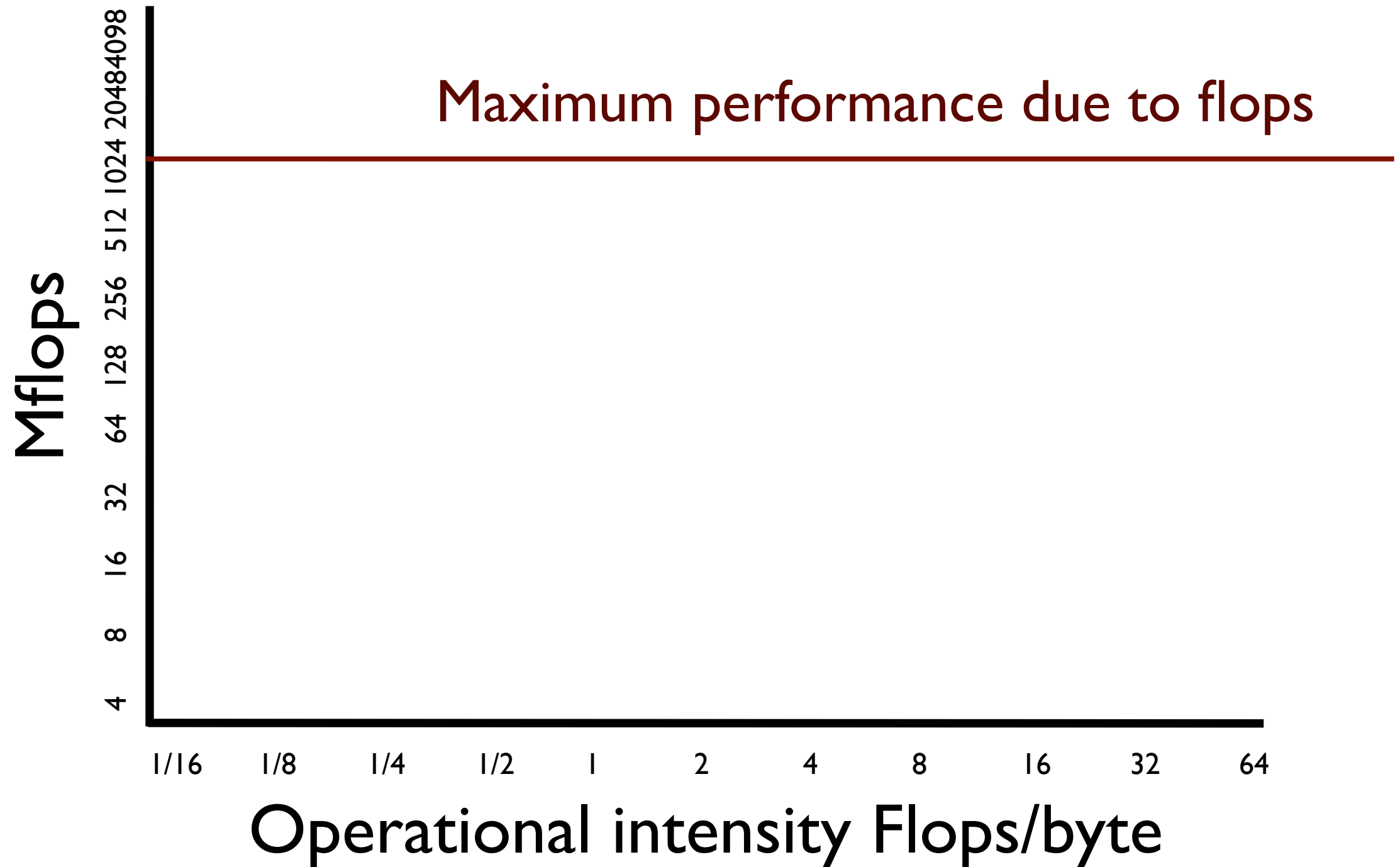
Multiply: 1  
Add: 1

IO: 24  
Flops: 2  
Ratio: 1/12

# Roofline model

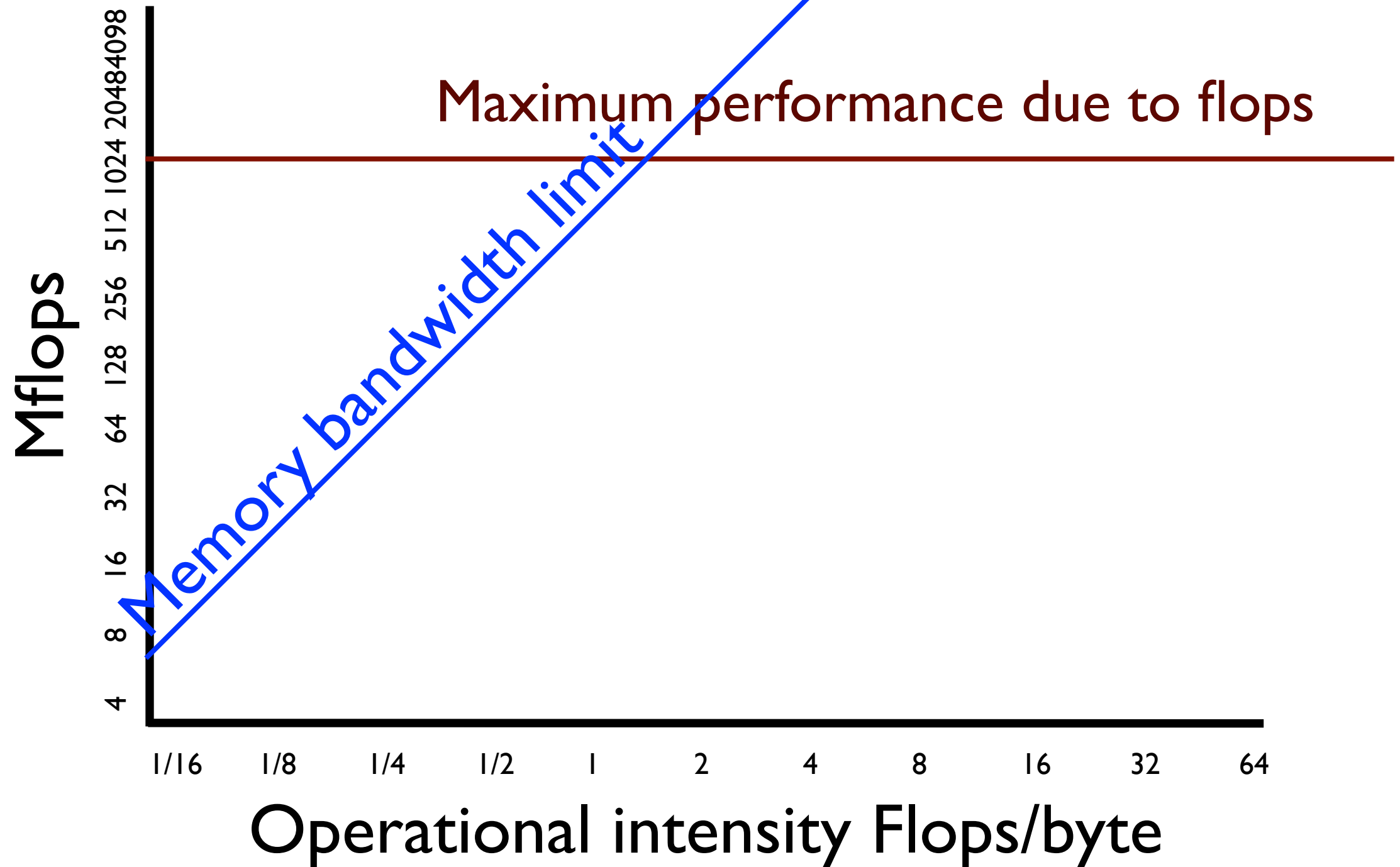


# Roofline model

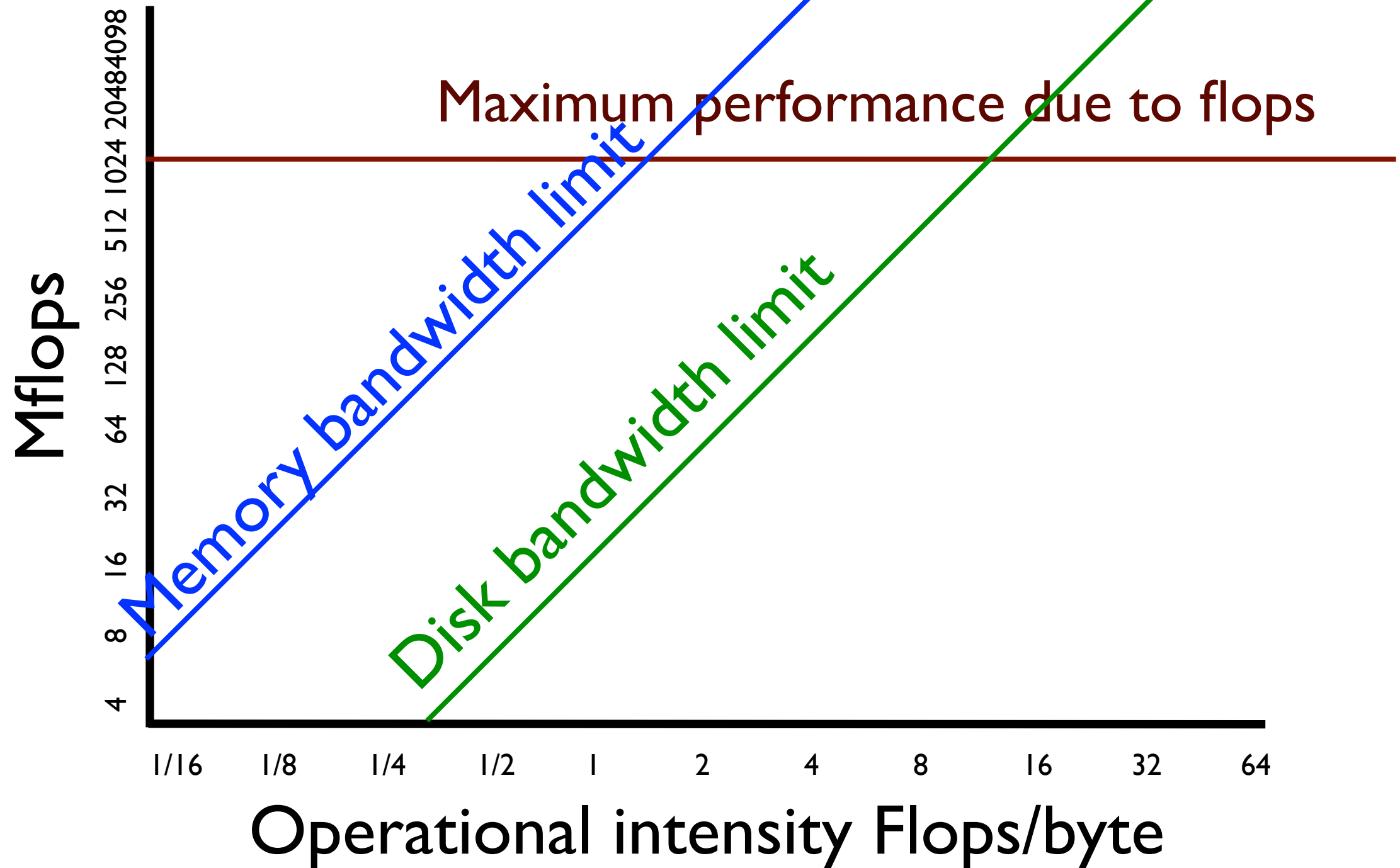




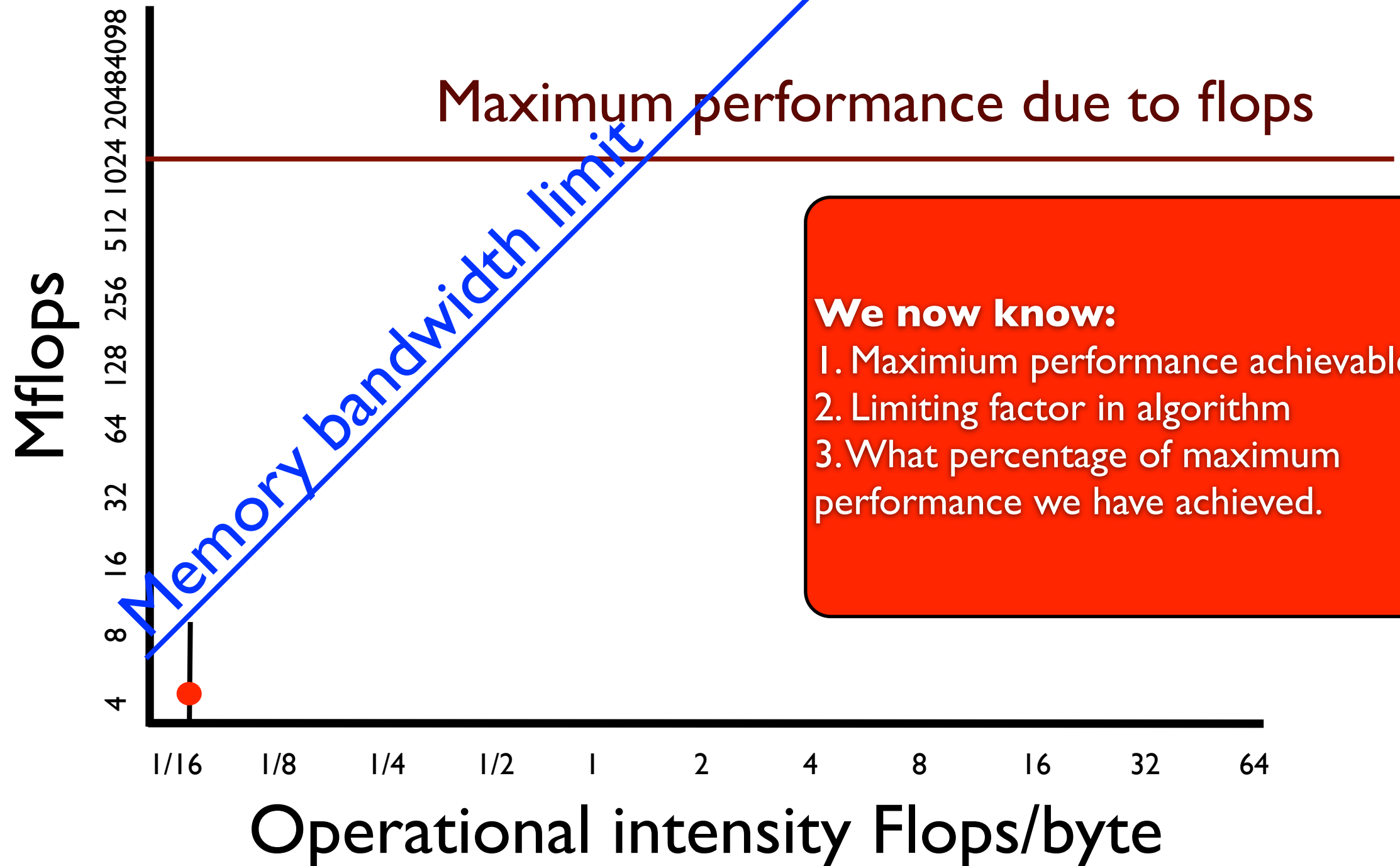
# Roofline model



# Roofline model



# Roofline model



# Causes for not achieving maximum performance

1. Indirection (iloc needs to be looked up before knowing what to grab from in, out)
2. Poor use of cache line for in,out
3. Inability to use prefetch on in,out

```
for(i2=0; i2 < n2; i2++){  
    out[iloc[i2][1]]+=in[iloc[i2][0]] *val[i2]  
}
```

What if the relative filter locations were always the same?

```
i=0;
for(i2=3; i2 < n2-3; i2++){
    ib=i2-3;
    for(i1=0; i1 < 7; i1++){
        out[i2]+=in[ib++]*val[i++];
    }
}
```

What if the relative filter locations were always the same?

```
i=0;
```

```
for(i2=3; i2 < n2-3; i2++){
```

```
    ib=i2-3;
```

```
    for(i1=0; i1 < 7; i1++){
```

```
        out[i2]+=in[ib++]*val[i++];
```

```
    }
```

Reads: 3

Writes: 1

Multiply: 1

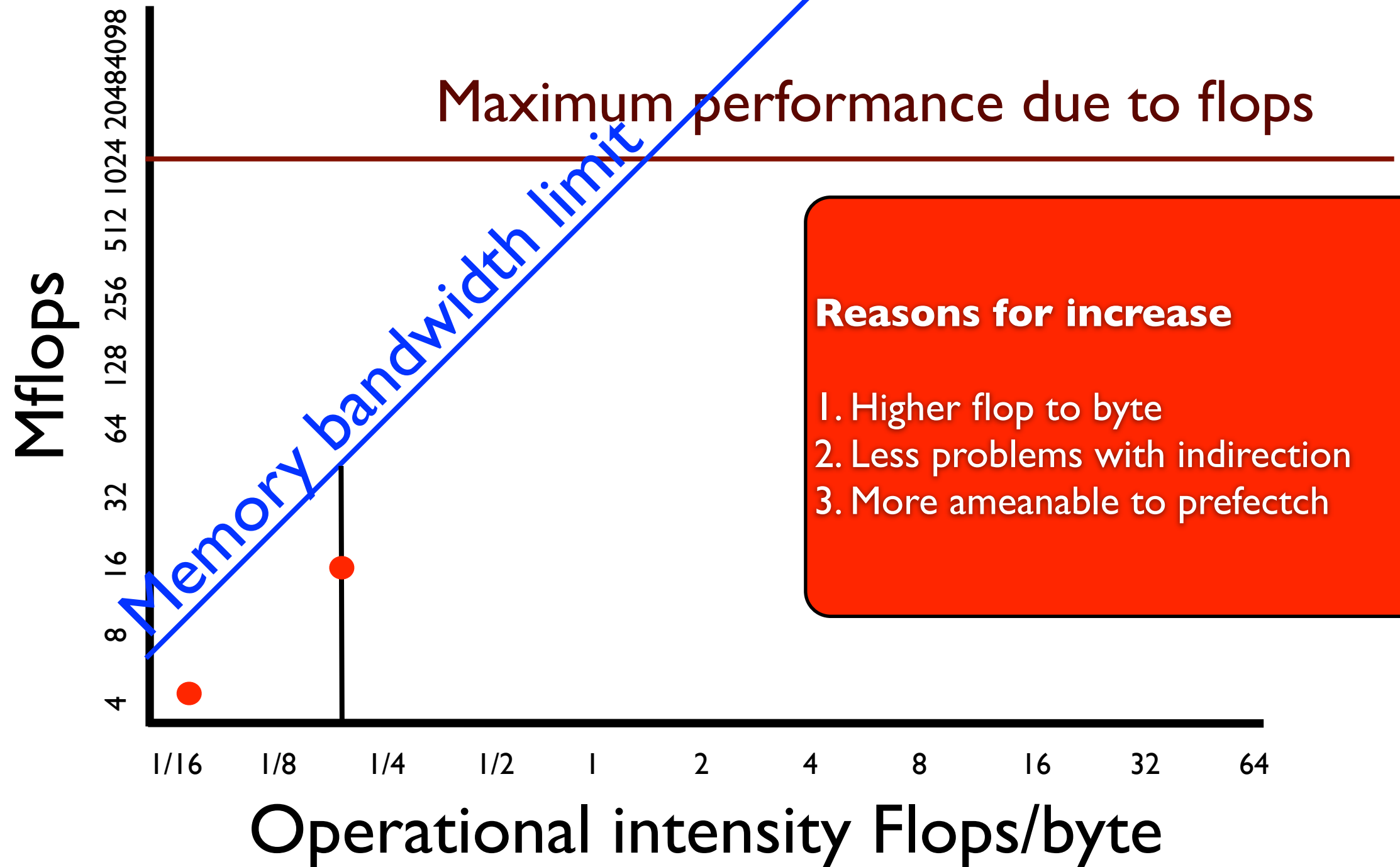
Add: 1

IO: 12

Flops: 2

Ratio: 1/6

# Roofline model



But we aren't dealing with a Von  
Neumann machine, we have cache

```
i=0;  
for(i2=3; i2 < n2-3; i2++){  
    ib=i2-3;  
    for(i1=0; i1 < 7; i1++){  
        out[i2]+=in[ib++]*val[i++];  
    }  
}
```

**In**

Has a reuse rate of 7. Only when  $i1=6$  does a new value need to be read in from memory

**out**

Has a reuse rate of  $1/7$ . Only when  $i1=0$  does a new value need to be read. Only when  $i1=6$  does a new value need to be written.

IO: 9.1

Flops: 2

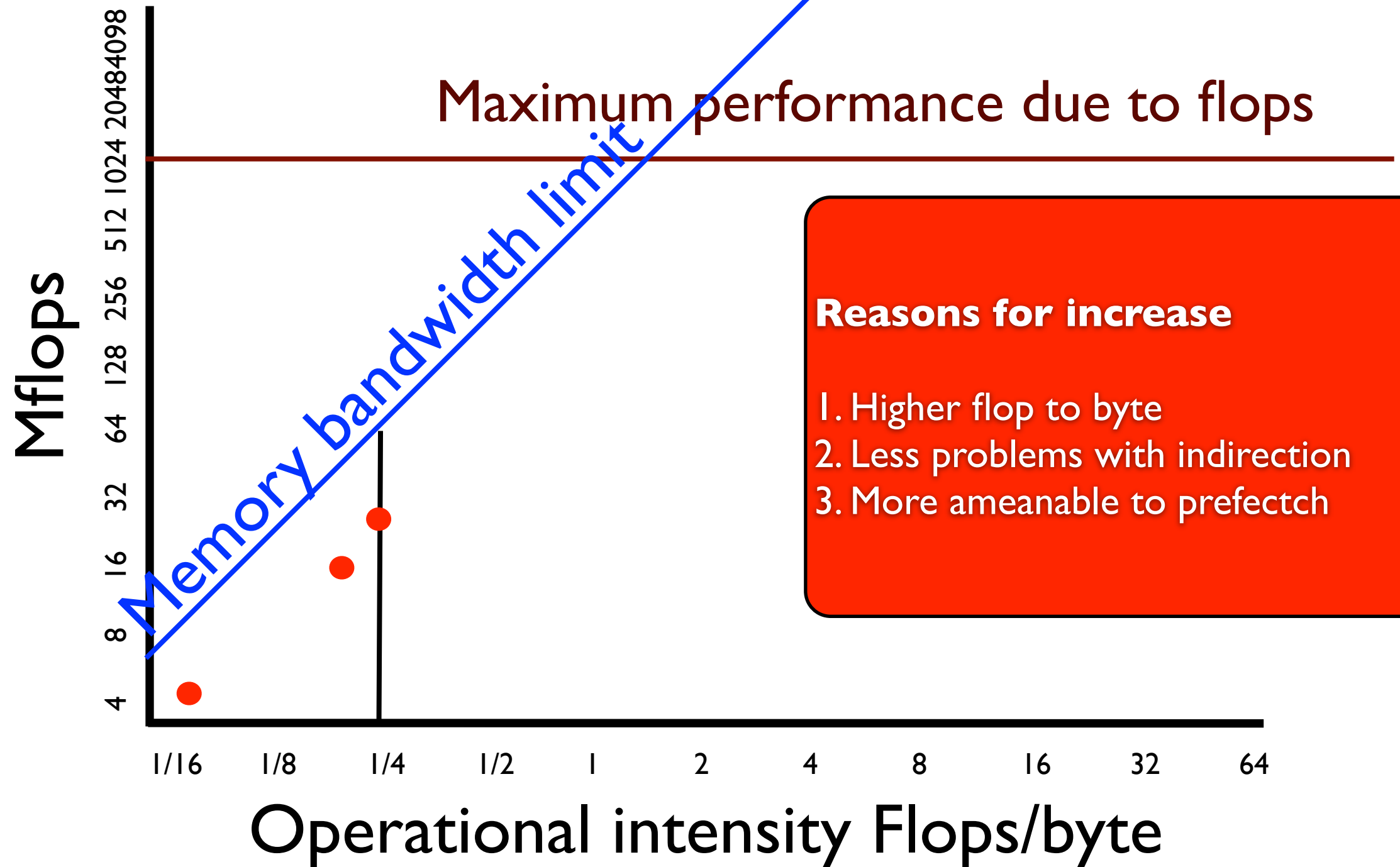
Ratio:  $1/4.5$

Reads:  $11/7$

Writes:  $1/7$



# Roofline model



What if val is only a function of i1?

```
for(i2=3; i2 < n2-3; i2++){  
    ib=i2-3;i=0;  
    for(i1=0; i1 < 7; i1++){  
        out[i2]+=in[ib++]*val[i++];  
    }  
}
```

What if val is only a function of i1?

```
for(i2=3; i2 < n2-3; i2++){  
    ib=i2-3;i=0;  
    for(i1=0; i1 < 7; i1++){  
        out[i2]+=in[ib++]*val[i++];  
    }  
}
```

IO: 1.12

Flops:2

Ratio: 1/4.5

Reads: 1/7

Writes: 1/7

# Roofline model

