Latency

Andrew Kondratovich
andrew.kondratovich@gmail.com

*«**Latency** is a time interval between the stimulation and response, or, from a more general point of view, is a time delay between the cause and the effect of some change in the system being observed»*

— https://en.wikipedia.org/

Latency versus bandwidth

— Latency is not the same as bandwidth.

— Reducing latency does not always improve bandwidth, in many cases it has the reverse effect.

— It still takes the same amount of time to get the first bit but if you need 10kb before you can start processing then bandwidth matters.

Infinite bandwidth, zero latency

$$C = 299{,}792{,}458 \text{ m/s}$$

— Every 300m of cable/fibre/microwave adds at least 1μS of latency.

— 300km will add at least 1ms.

— 1ms is worth about $100m in the trading world.

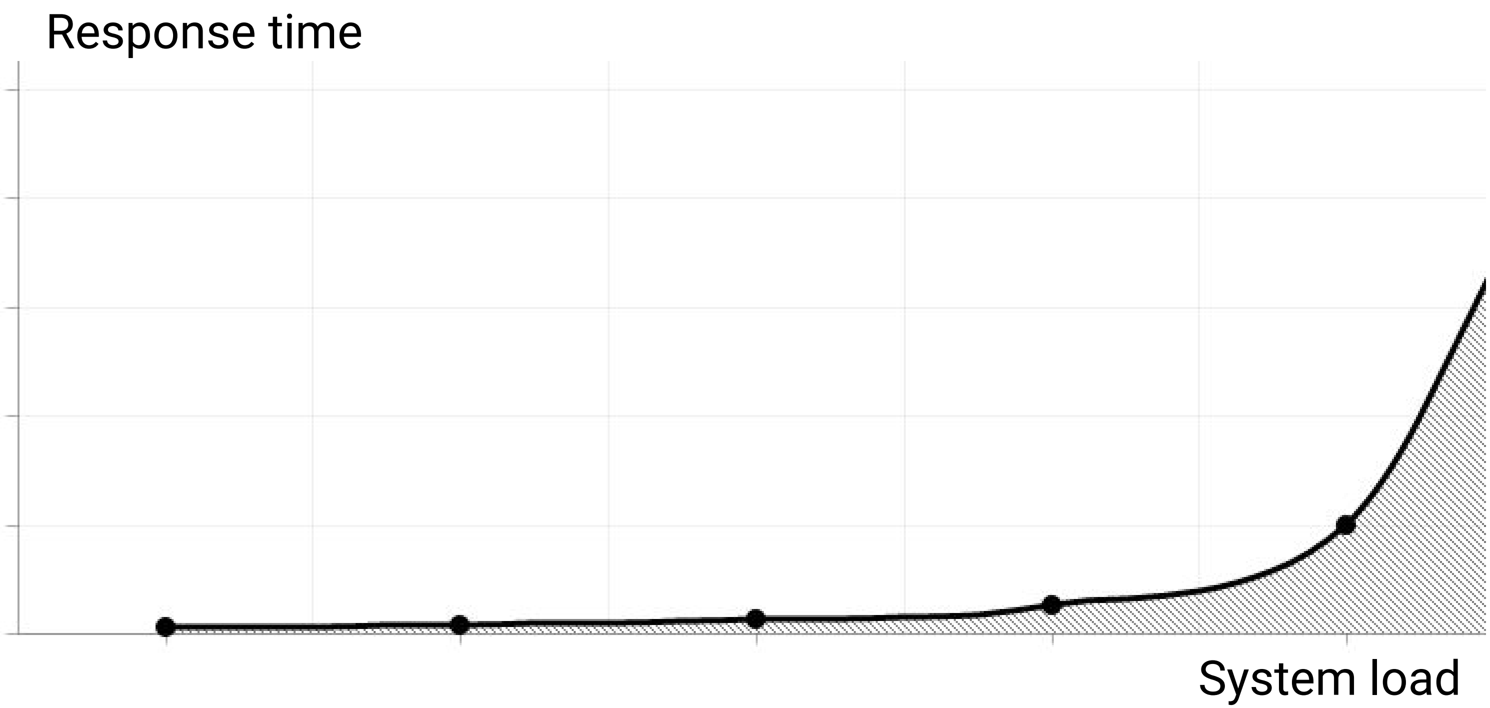
Latency behaviour

— Each operation occurrence has its own latency.

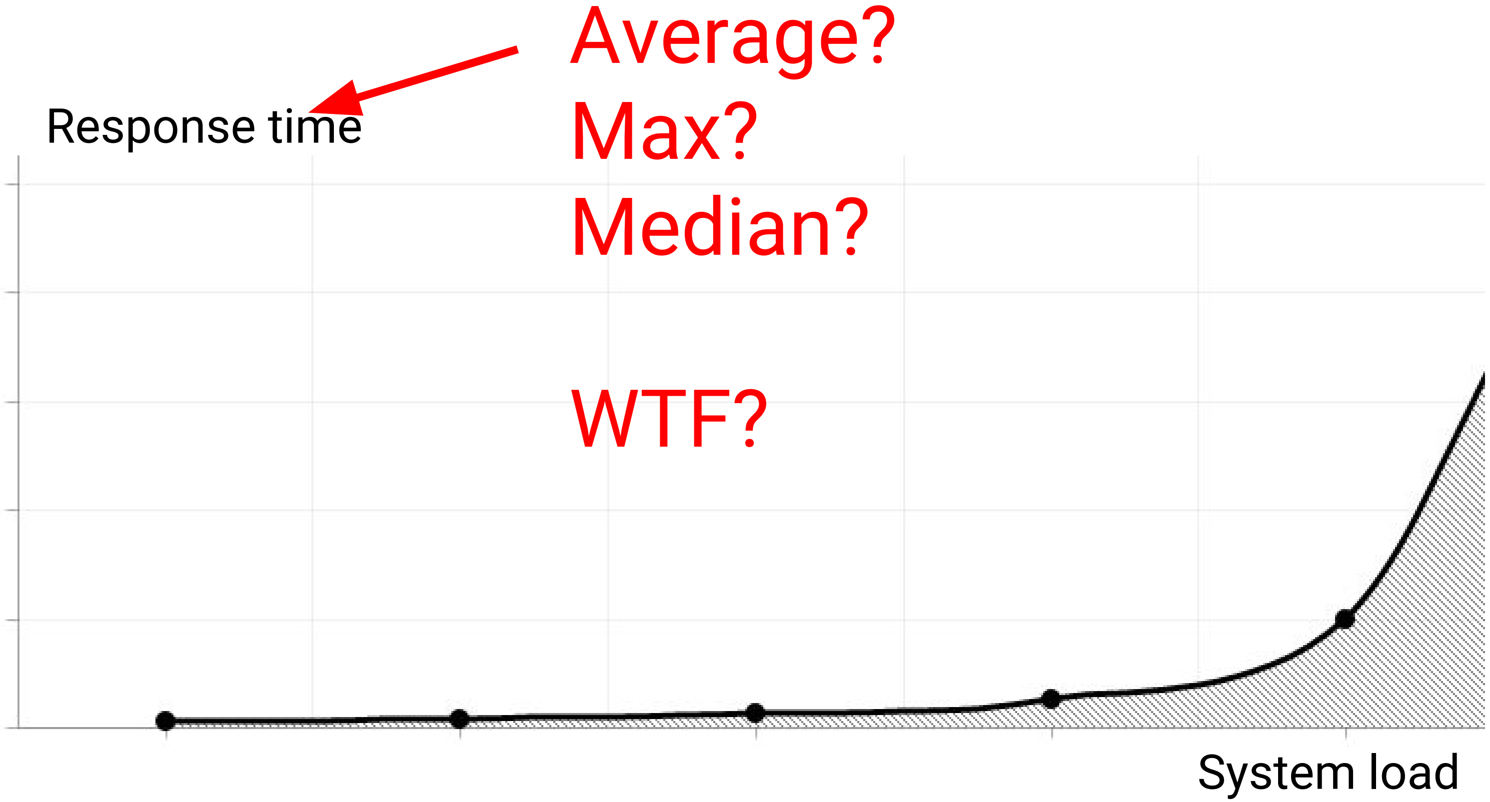
— What we care about is how latency *behaves*.

— Behavior is more than «what was the common case?»

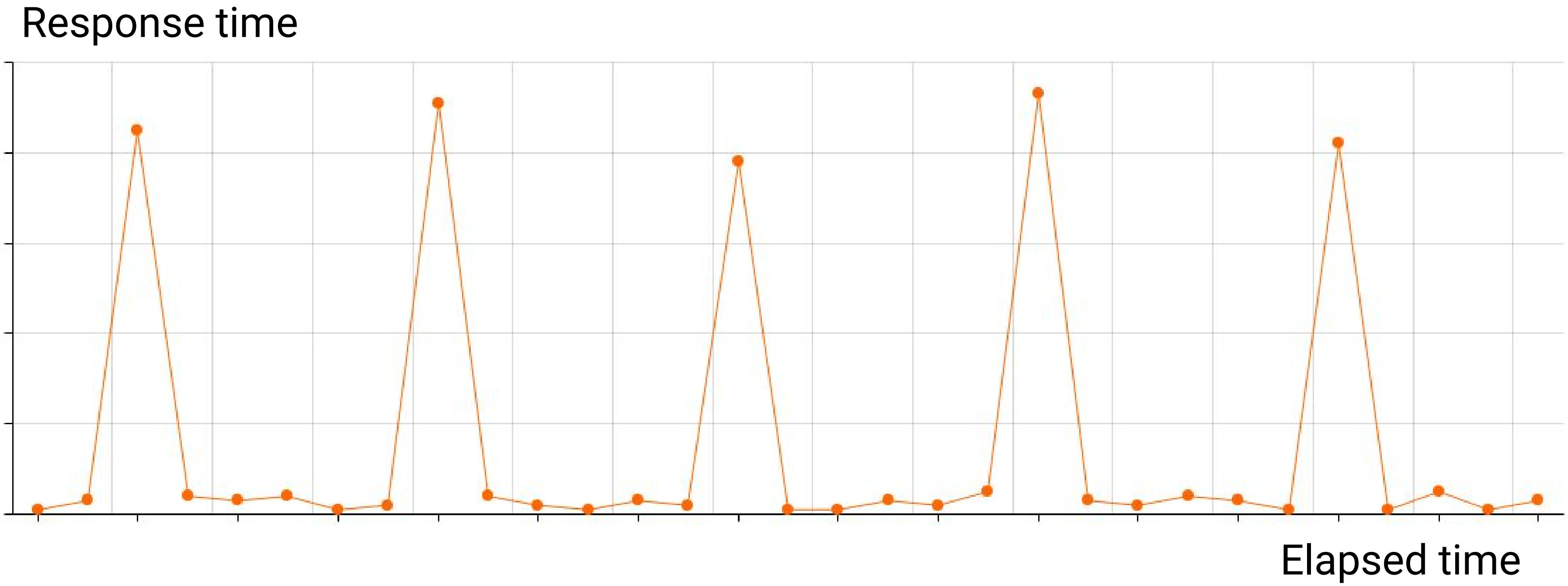
Response time over load



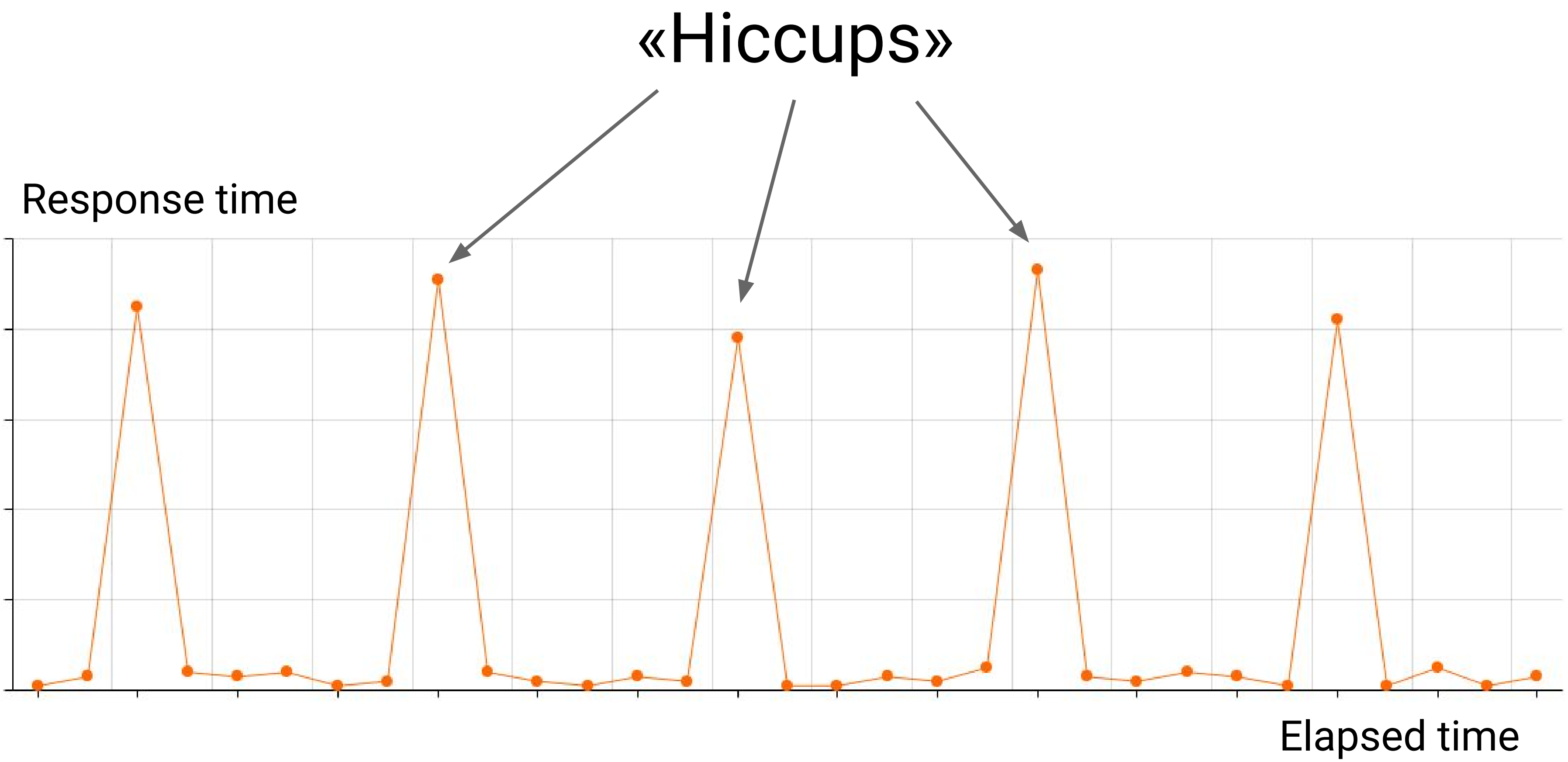
Response time over load



Response time over time



Response time over time



Hiccups

- They don’t look like a normal distribution.

- They usually look like periodic freezes.

- A complete shift from one state to another:
    1. everything is okay
    2. something gone bad
    3. disaster

Deal with it

— What do we want the latency to be?

— Different applications have different needs.
— Requirements should reflect application needs.

— Better way to deal with hiccups is measuring ***percentiles***.

What do we care about?

— Care about the worst case?

— Care about the 99.99%?

— Care if 1% of operations fail?

— Care about few fastest events?

...

Latency Numbers

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

Java memory 1/4

— Java is very inefficient at storing data in memory.

— Objects get created everywhere, they’re good for OOP but crap if you’re trying to get performance out of your machine.

— Each object created will need to be garbage collected.

Java memory 2/4

— Objects have 12/16 bytes headers.

— Java bloating is endemic: *String*, *Double*, *BigDecimal*, *Date*, *LinkedList*, *Iterator*.

```
public class Quote {
  private Date tradeDate;
  private BigDecimal bid;
  private String currency;
  ...
}
```

— Abstraction costs dearly.

Java memory 3/4

— We can use **compression**, but it’s slow.
— We want **compaction** not compression.

```
public class Quote {
  private byte[] data;
}
```

— Just one object, fast to allocate.
— If we can encode the data in the binary then it’s fast to query too.

Java memory 4/4

– Identical API.
– We can use ByteBuffer instead of byte array.

```
public Date getTradeDate() {
  return tradeDate;
}

public Date getTradeDate() {
  return new Date((((long) data[0] & 0xff) << 0) |
                  (((long) data[1] & 0xff) << 8) |
                  ...);
}
```

Optimizations 1/3

– Preallocation

Queues, buffers, networking, etc...

– Redundant write elimination

```
elements[index] = item;
// versus
if (elements[index] != item) {
  elements[index] = item;
}
```

Optimizations 2/3

- Passive and Active waiting

  Parking versus Spinning
- Backoff

  Thread.yield();
- Thread affinity

  Thread context relocation workaround
- Sequential and random access

  ArrayList and LinkedList
- True and false sharing

Optimizations 3/3

– CAS/TAS and TTAS

– Atomic.lazySet()

Non-volatile write into volatile variable

volatile store	**lazySet**
`mbar(store|store)`	`mbar(store|store)`
`store a`	`store a`
`mbar(store|load)`	

Q/A

http://kondratovich.me/talks/october-2015-latency.pdf