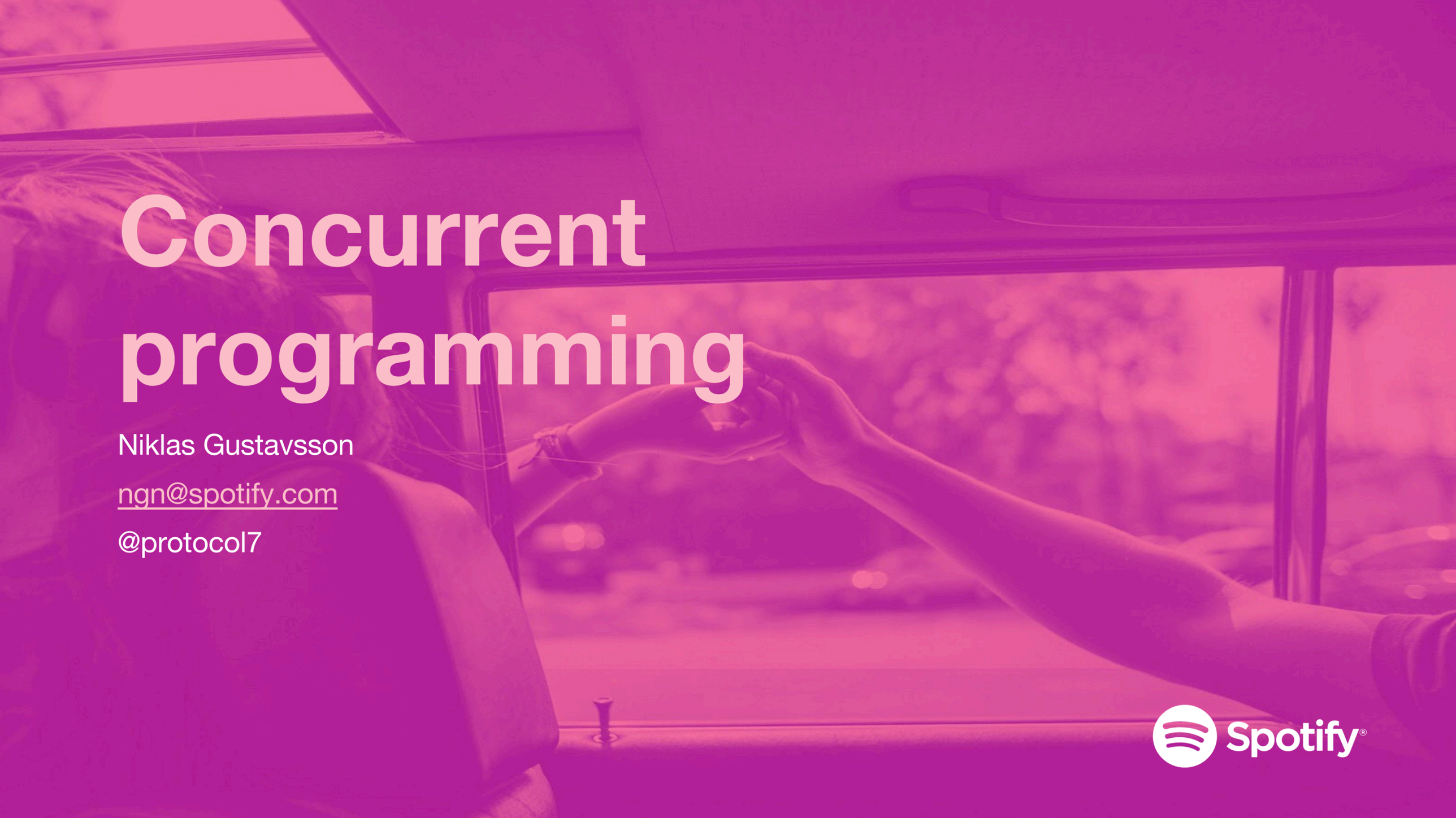


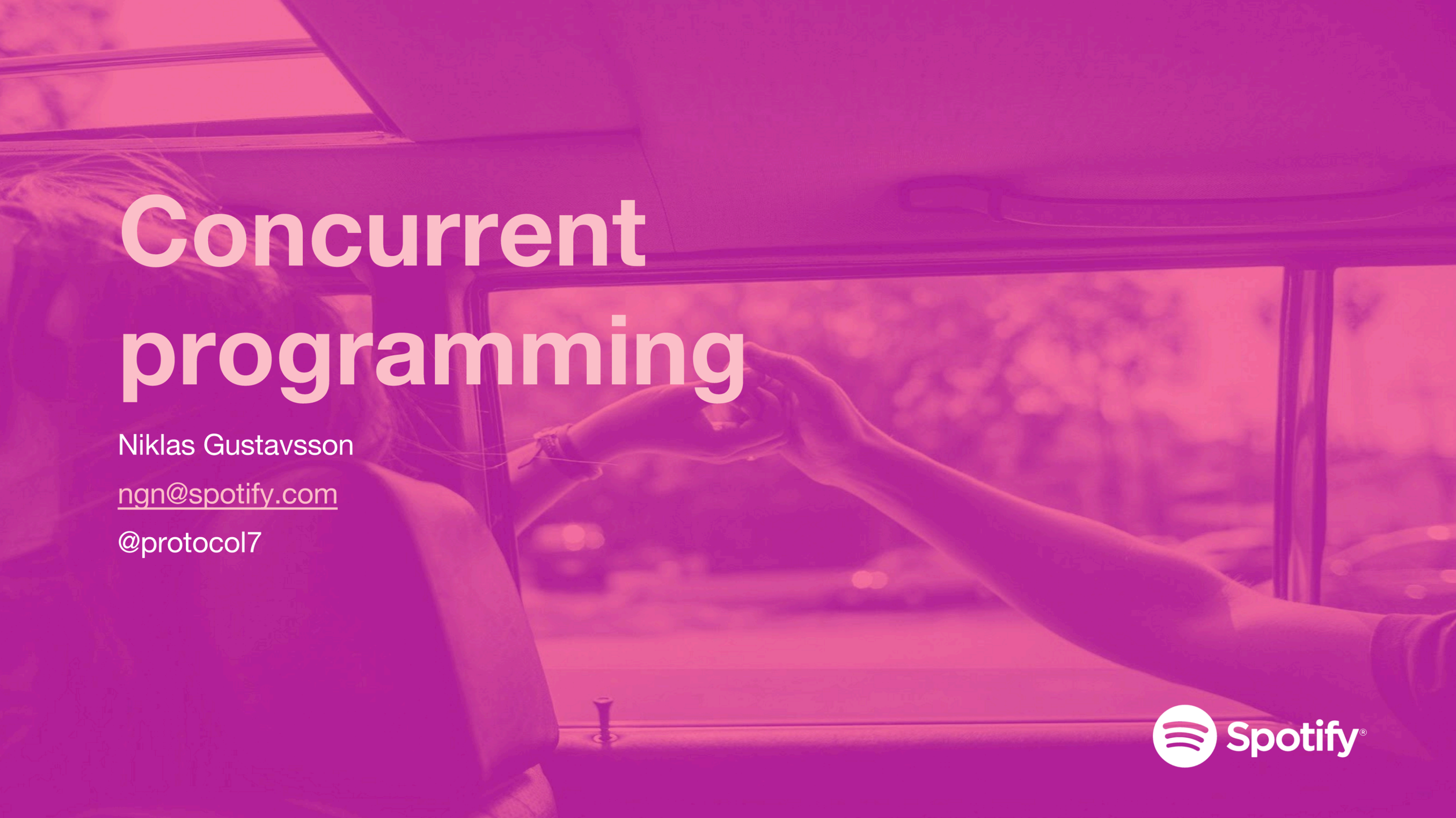
Concurrent programming

Niklas Gustavsson

ngn@spotify.com

@protocol7

Spotify®



Concurrent programming

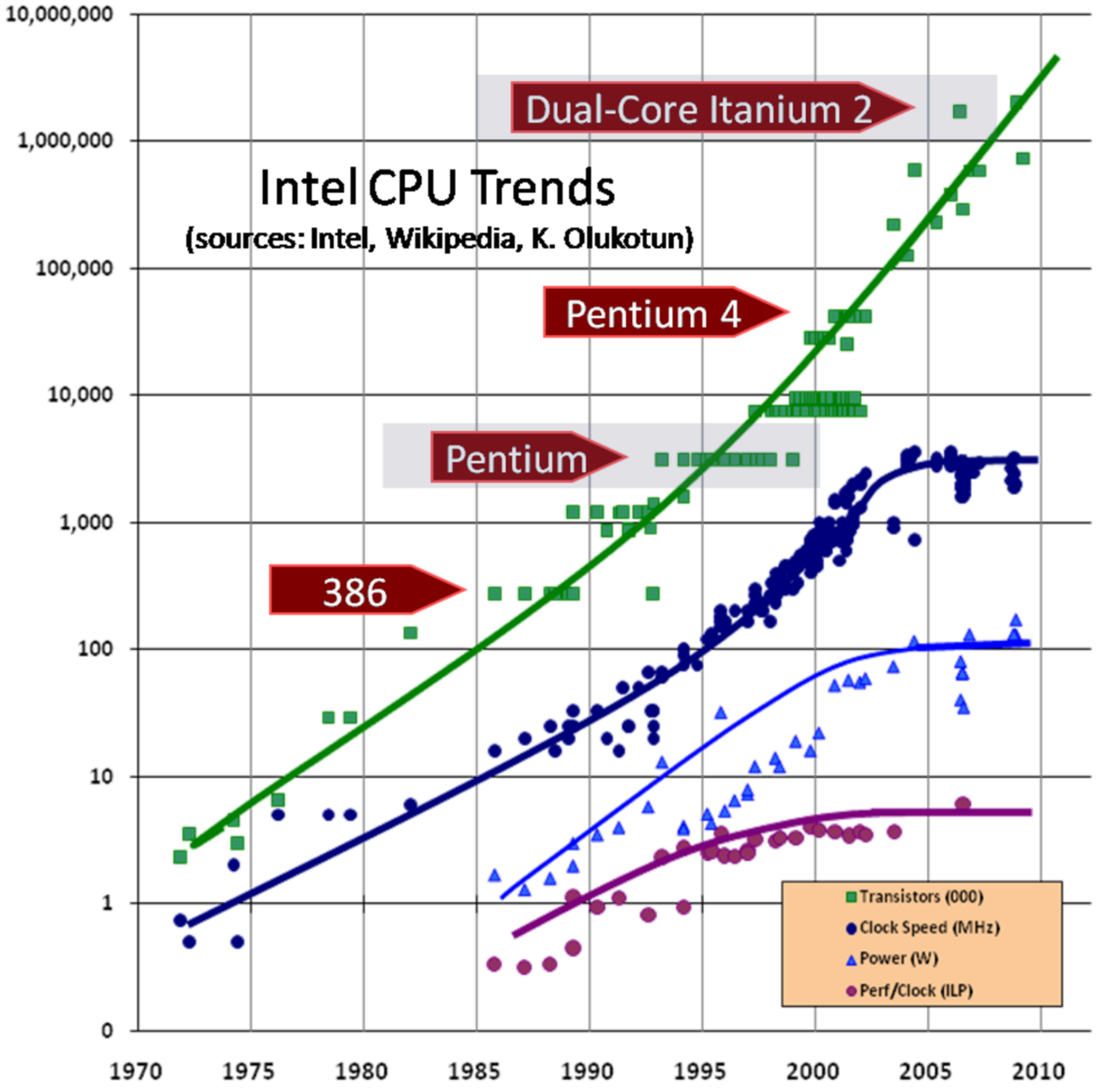
Niklas Gustavsson

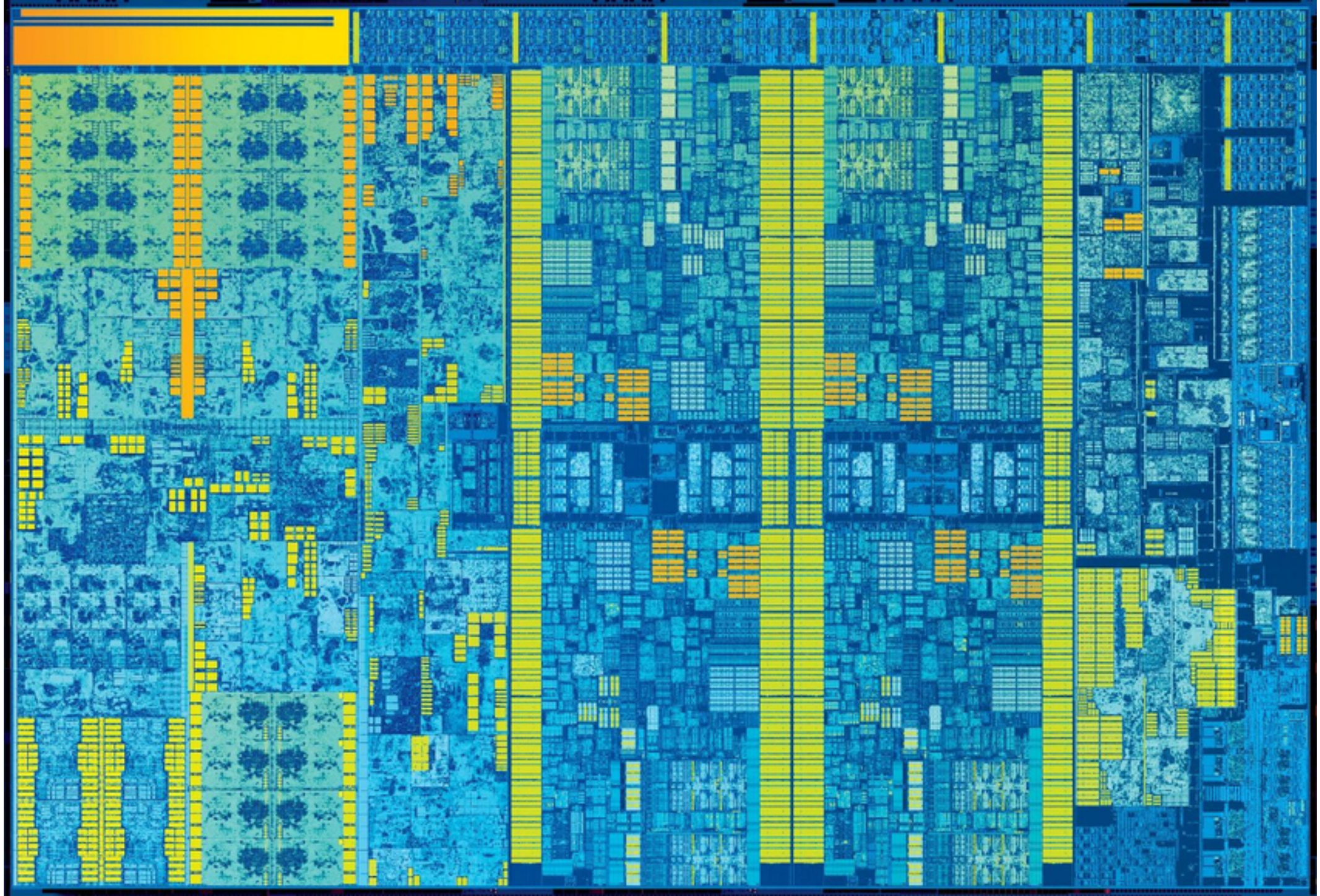
ngn@spotify.com

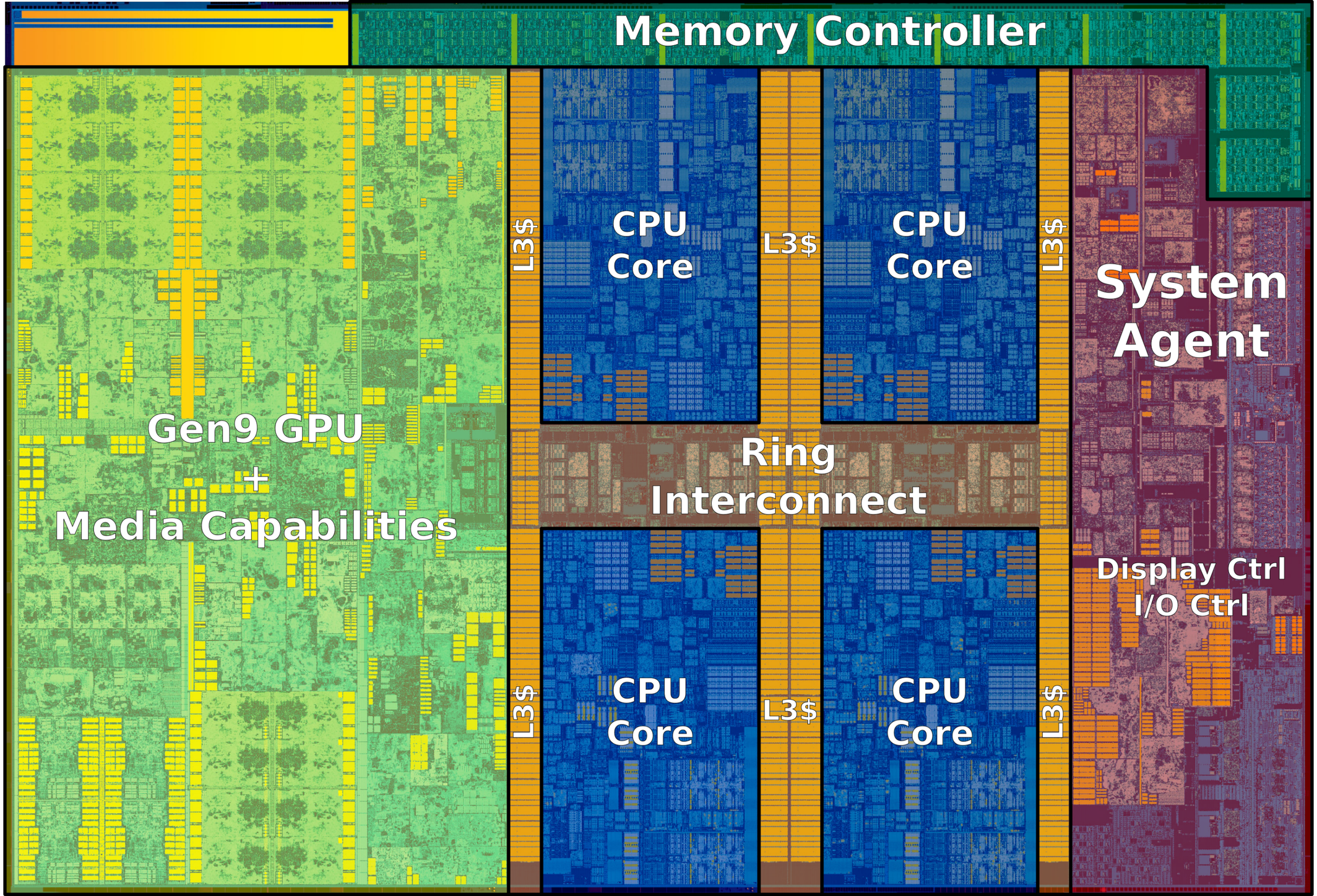
@protocol7

Spotify®


Spotify®


Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
Transistors (000)
Clock Speed (MHz)
Power (W)
Perf/Clock (ILP)
10,000,000
1,000,000
100,000
10,000
1,000
100
10
1
0
1970
1975
1980
1985
1990
1995
2000
2005
2010




Memory Controller
Gen9 GPU
+
Media Capabilities
L3$
CPU
Core
L3$
CPU
Core
L3$
Ring
Interconnect
L3$
CPU
Core
L3$
CPU
Core
L3$
System
Agent
Display Ctrl
I/O Ctrl

Cache latency

- L1: 1ns
- L2: 3ns
- L3: 15-20ns
- RAM: 65ns

Counting things

```
public class NaiveCounter {
  private long count = 0;

  public void increment() {
    count++;
  }

  public long count() {
    return count;
  }
}
```

volatile

```
public class VolatileCounter {
  private volatile long count = 0;

  public void increment() {
    count++;
  }

  public long count() {
    return count;
  }
}
```

synchronized

```
public class SyncronizedCounter {
  private long count = 0;

  public synchronized void increment() {
    count++;
  }

  public synchronized long count() {
    return count;
  }
```

ReadWriteLock

```
public class ReadWriteLockCounter {
  private long count = 0;
  private ReadWriteLock lock = new ReentrantReadWriteLock();

  public void increment() {
    lock.writeLock().lock();
    try {
      count++;
    } finally {
      lock.writeLock().unlock();
    }
  }

  public long count() {
    lock.readLock().lock();
    try {
      return count;
    } finally {
      lock.readLock().unlock();
    }
  }
}
```

CAS

java.util.concurrent.atomic

AtomicLong

```
public class AtomicCounter {
  private final AtomicLong count = new AtomicLong();

  public void increment() {
    count.incrementAndGet();
  }

  public long count() {
    return count.longValue();
  }
}
```

```
public final long incrementAndGet() {
  for (;;) {
    long current = get();
    long next = current + 1;
    if (compareAndSet(current, next))
      return next;
  }
}
```

LongAdder

```
public class LongAdderCounter {
  private final LongAdder count = new LongAdder();

  public void increment() {
    count.increment();
  }

  public long count() {
    return count.sum();
  }
}
```

Cache lines

Aside: Lists

JEP 169: Value Objects[1]

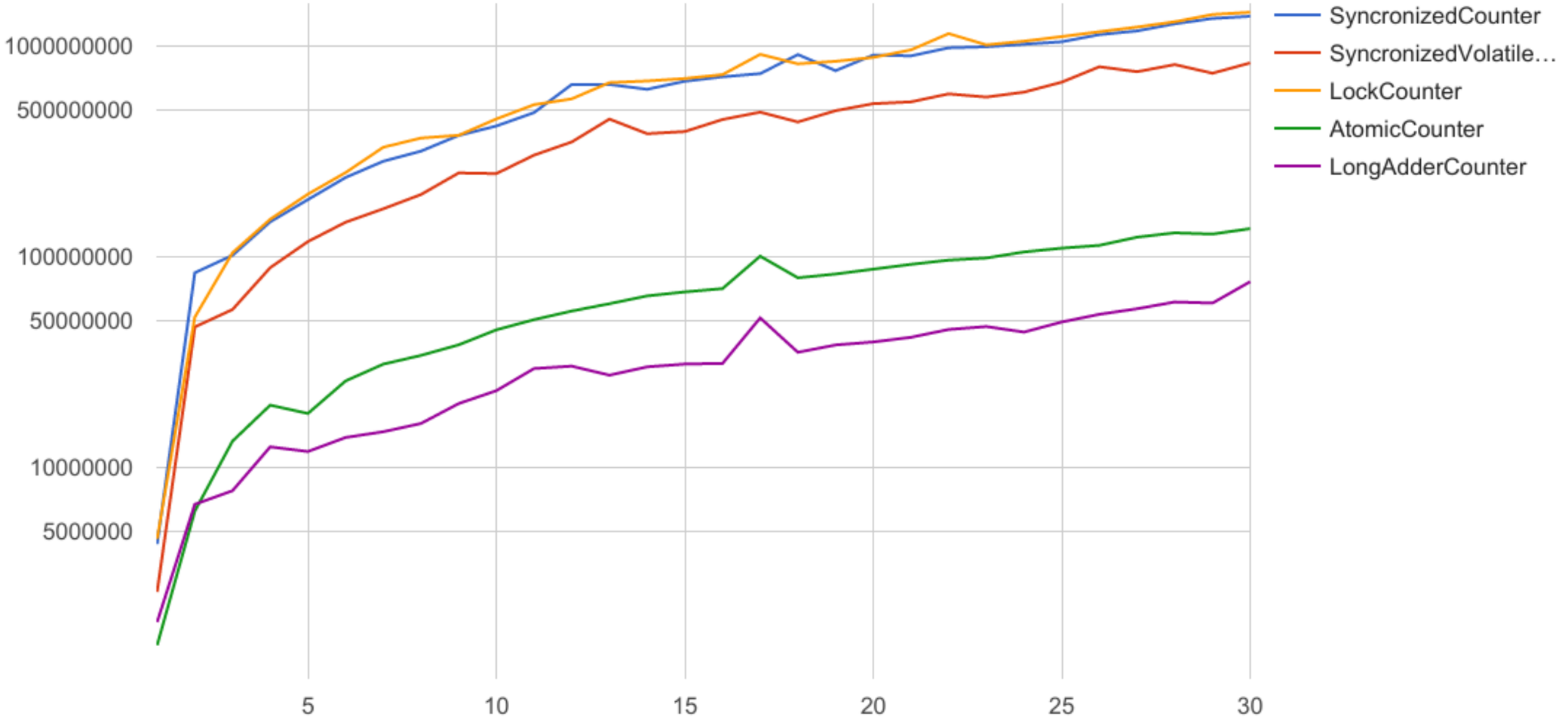
[1] http://cr.openjdk.java.net/~jrose/values/value-type-hygiene.html

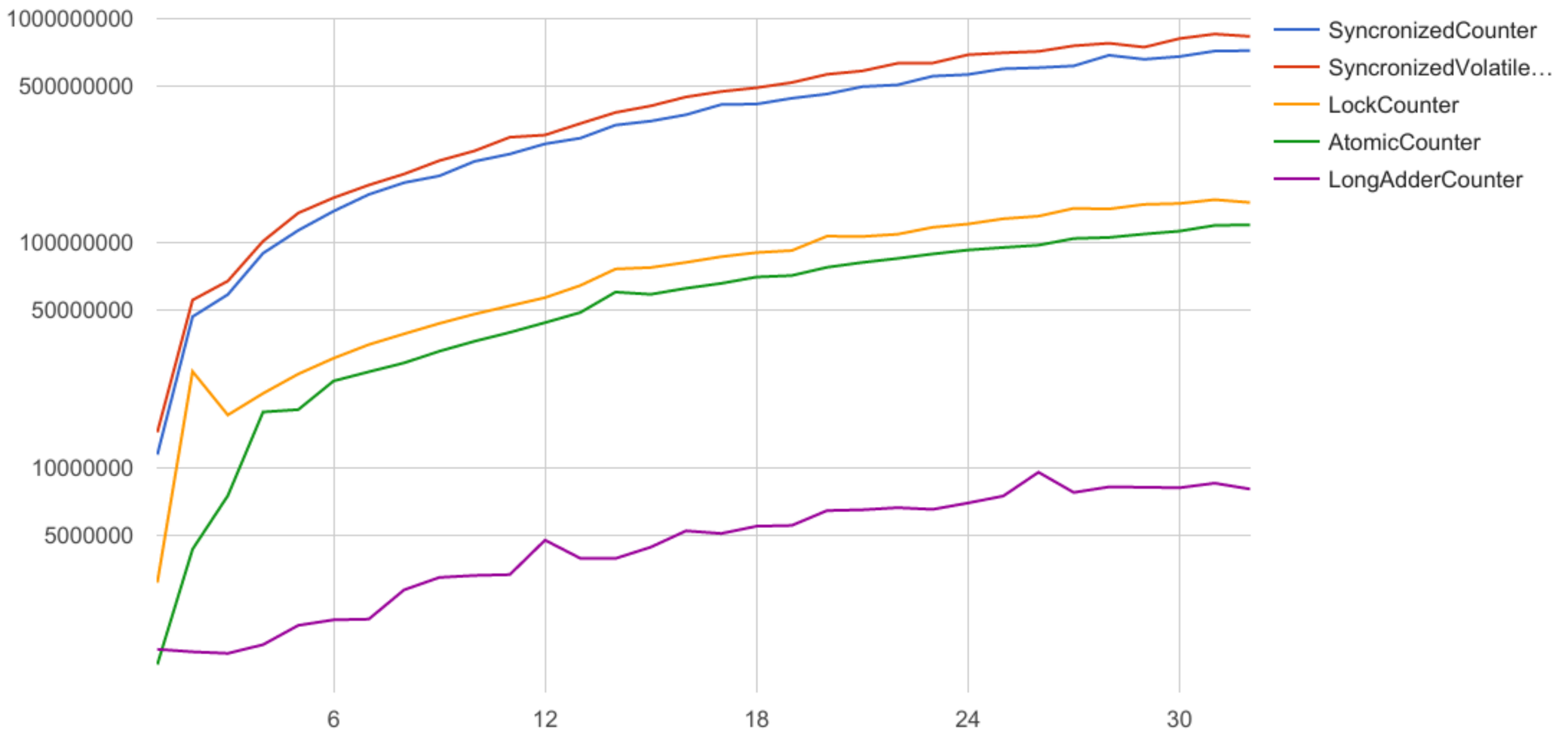
False sharing

Padding

```
volatile long p0, p1, p2, p3, p4, p5, p6;
volatile long value;
volatile long q0, q1, q2, q3, q4, q5, q6;
```

```
@sun.misc.Contended static final class Cell {
    volatile long value;
    ...
```

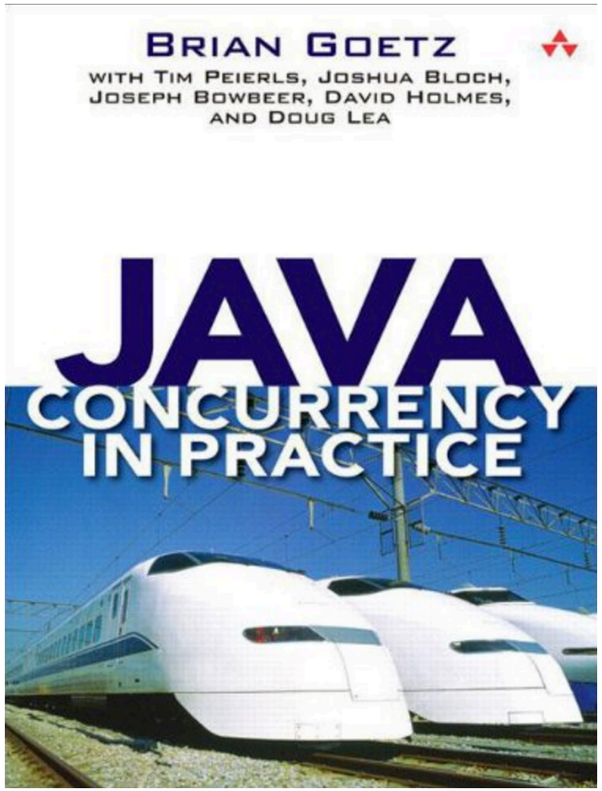

1000000000
500000000
100000000
50000000
10000000
5000000
5
10
15
20
25
30
SyncronizedCounter
SyncronizedVolatile…
LockCounter
AtomicCounter
LongAdderCounter


1000000000
500000000
100000000
50000000
10000000
5000000
6
12
18
24
30
SyncronizedCounter
SyncronizedVolatile…
LockCounter
AtomicCounter
LongAdderCounter

Java memory model

Further reading

- `java.util.concurrent` JavaDocs
- Java Concurrency in Practice
- Anything by Doug Lea
  - Including source code
- Anything by Martin Thompson,
  - @mjpt777



Questions?

ngn@spotify.com

@protocol7