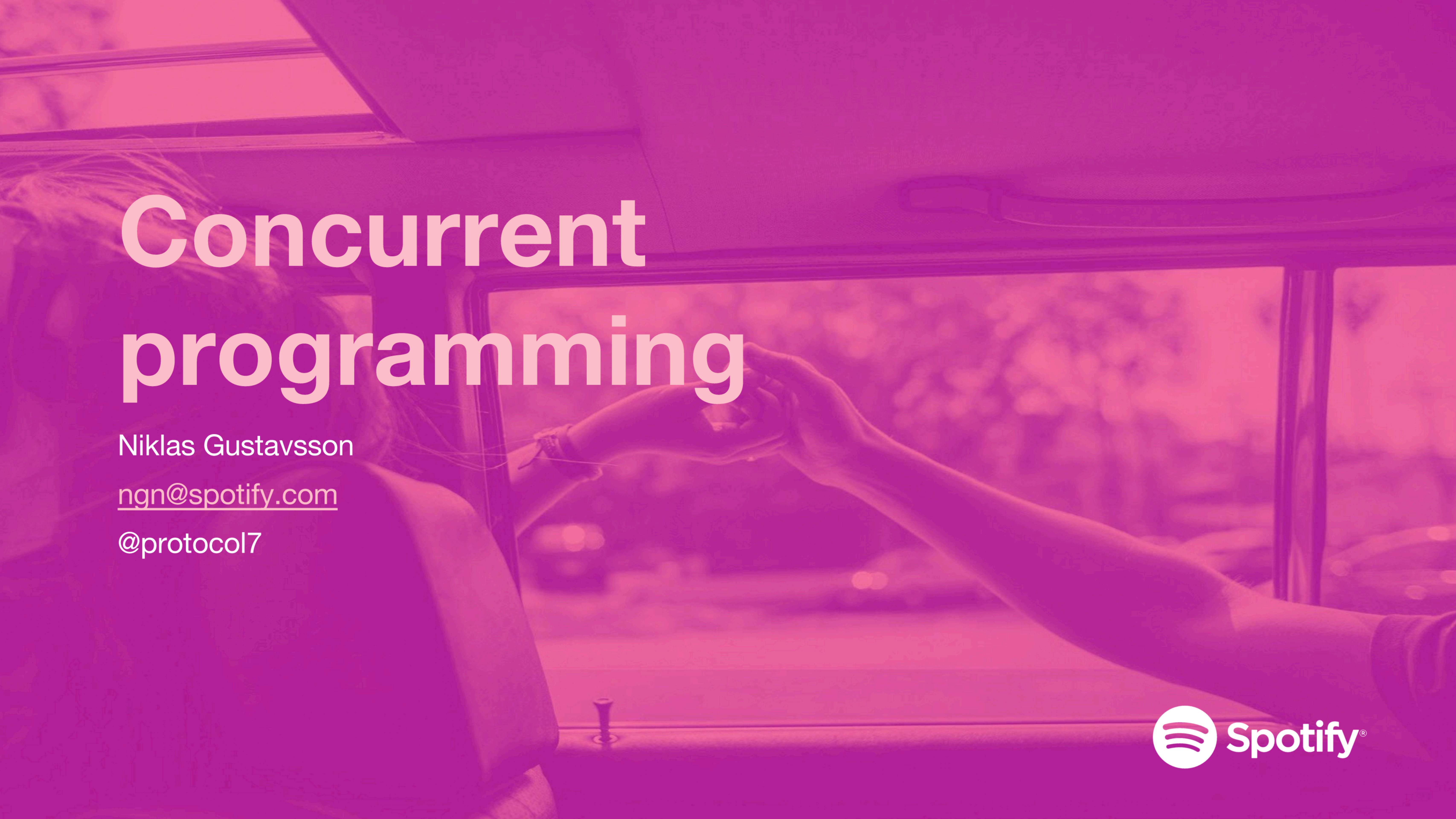


Concurrent programming

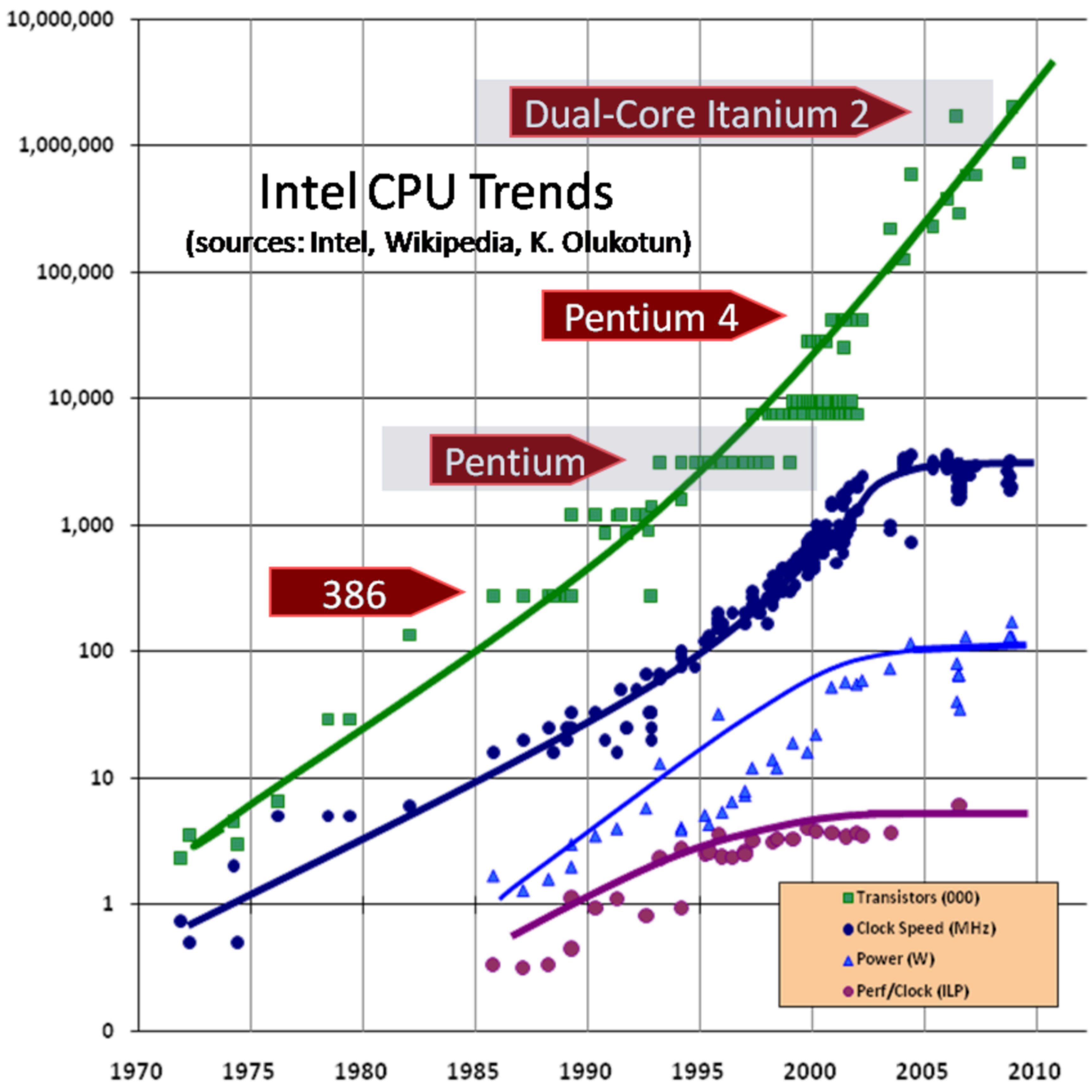
Niklas Gustavsson

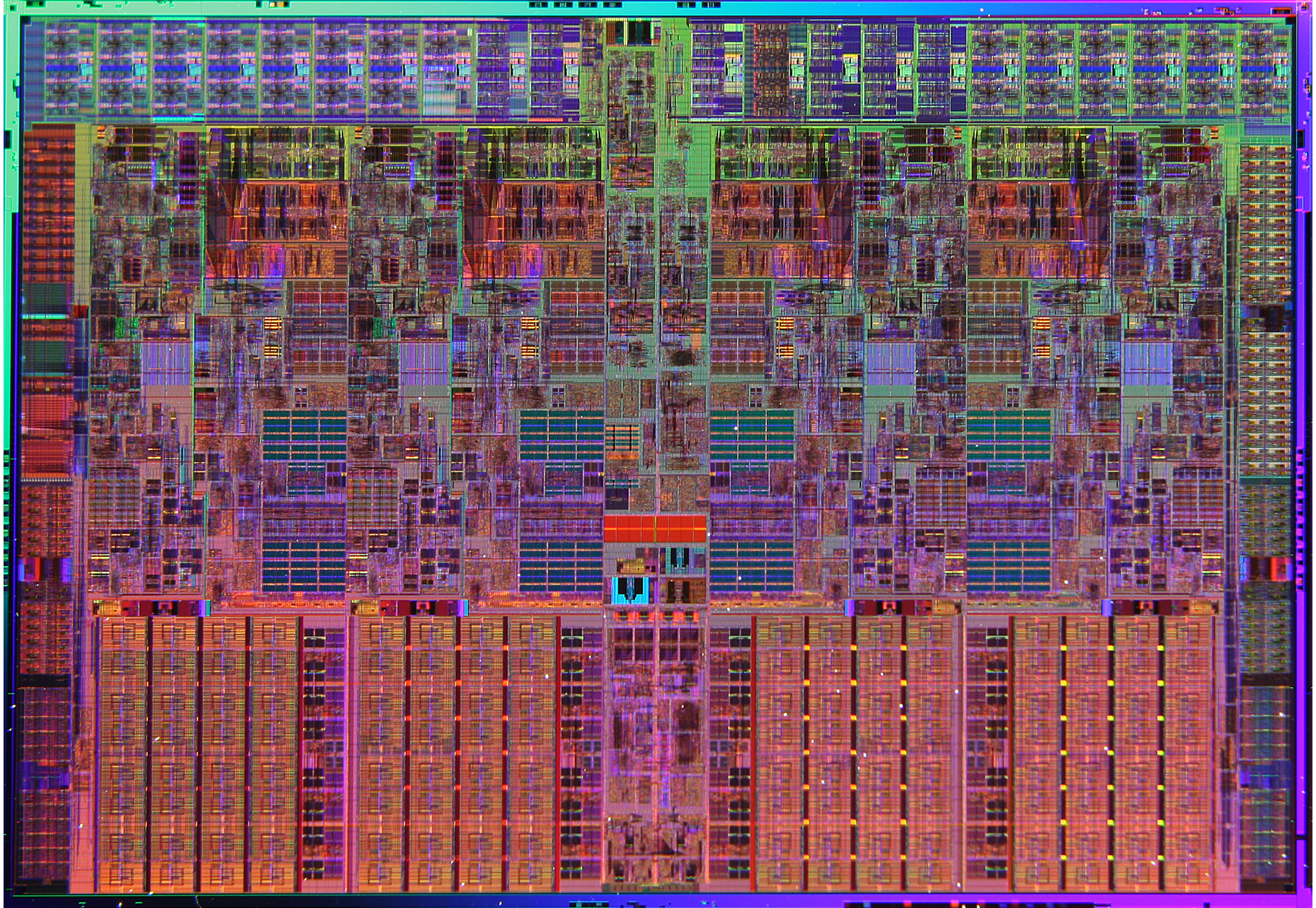
ngn@spotify.com

@protocol7

Spotify Gbg

- Development office
- 30-ish developers
- Owns playback and Your music and Social
- Street team


Intel CPU Trends
(sources: Intel, Wikipedia, K. Olukotun)
Dual-Core Itanium 2
Pentium 4
Pentium
386
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
🟩 Transistors (000)
● Clock Speed (MHz)
▲ Power (W)
● Perf/Clock (ILP)



How can you go faster?

Cache lines

Java data structures

Lock-less programming

Lock-less programming

- Locks requires coordination among cores
    - `synchronized {}`

RMW

- Atomic operations supported by the CPU

CAS

```java
public final int getAndSet(int newValue) {
  for (;;) {
    int current = get();
    if (compareAndSet(current, newValue))
      return current;
    }
  }
}
```

Java memory model

Threads

Concurrent programming models

- Threads, shared memory
- Async, futures, promises
- Actors, message passing
- Reactive
- …

Why async?

Scaling it out

- Requests block for extended times
- Large number of incoming requests

Executor and ExecutorService

- Abstraction for running tasks
- Usually on a thread pool
- Submit tasks (Runnable or Callable) for execution

Futures and Promises

- Futures holds a future result of a computation
- Promises are a promise to, in the future, provide result of a computation

ListenableFuture

- A Future require blocking when getting the value
- CompletableFuture in Java 8
- ListenableFuture in Google's Guava
- Allows for simple, async composition

Futures.allAsList()/successfulAsList()

- allAsList() returns the results from all futures, or fails it at least one future fails
- successfulAsList() returns the results from those futures which do not fail
- CompletableFuture.allOf()

Futures.transform()

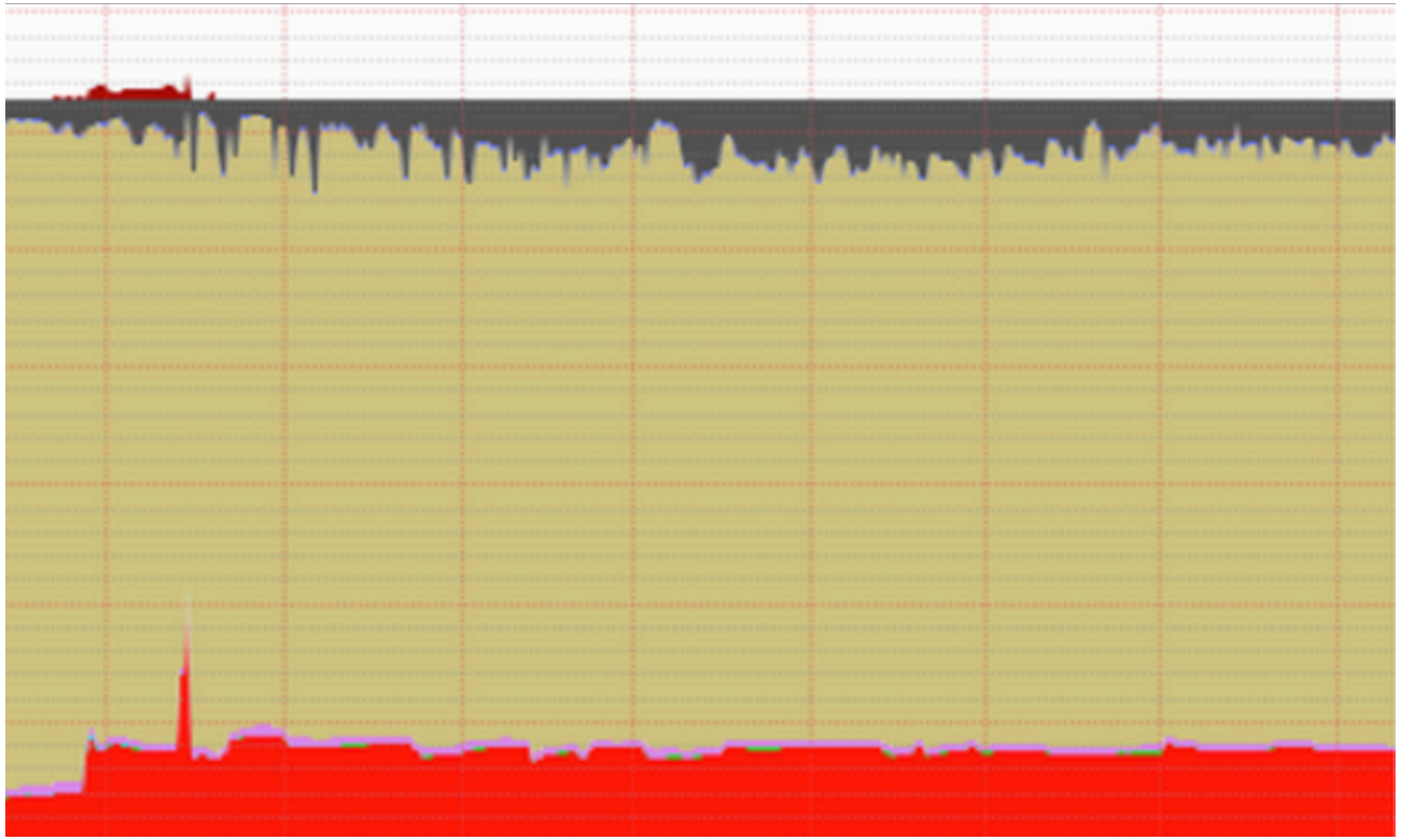
- Applies a function, sync or async, to the result of a ListenableFuture
- CompletableFuture.thenApply()/thenApplyAsync()

Turtles all the way down

- Java NIO
  - select/epoll based I/O
  - Low level API
  - Netty/MINA
- Async protocols
  - Message queuing
  - https://github.com/spotify/netty-zmtp

Limiting concurrency

Unbounded queues are evil

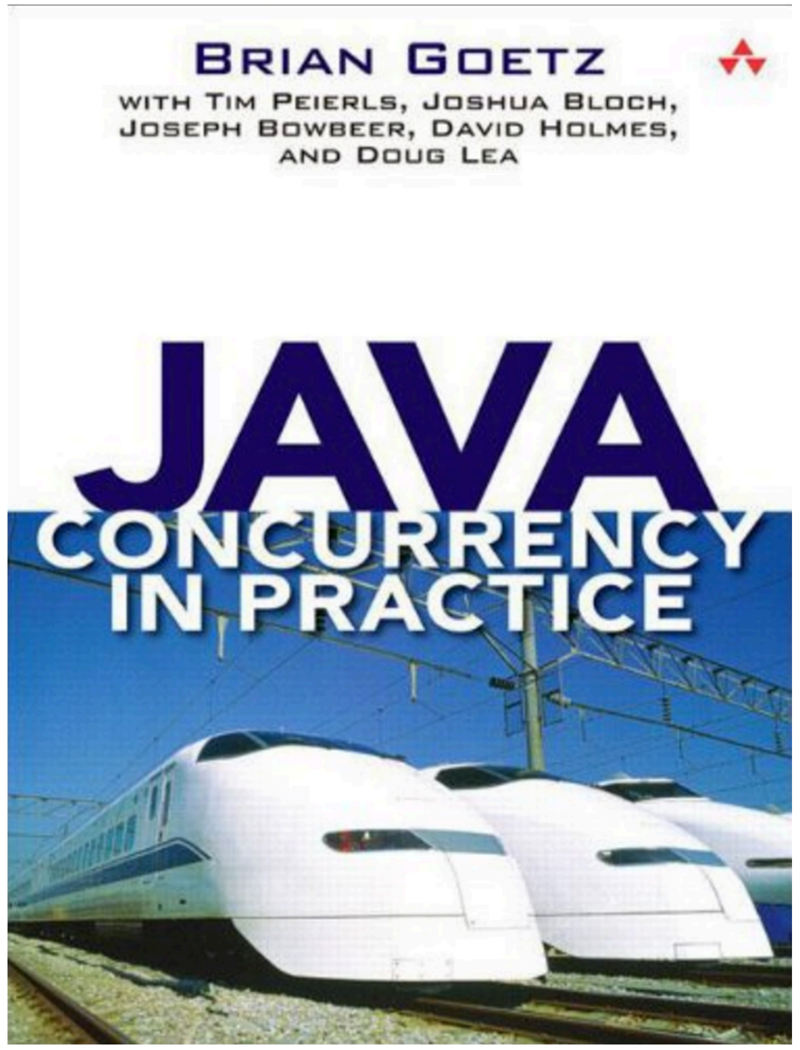


Limiting concurrency

- Thread pools are much like a queue
- Always limit thread pools
  - Approximately number of cores
- Prefer dropping tasks if possible
- Provide proper back pressure

Further reading

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


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

Questions?

ngn@spotify.com

@protocol7

Spotify®