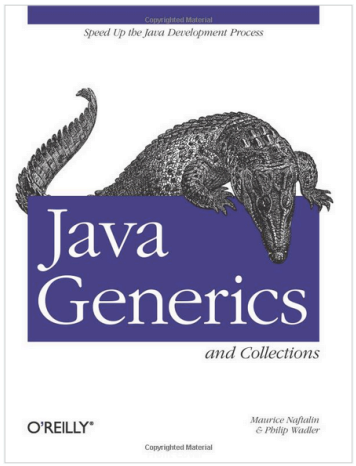
What's New in Java 8

Maurice Naftalin

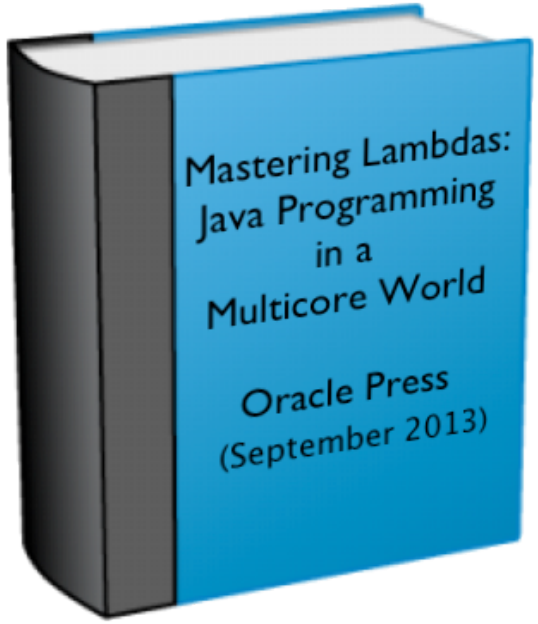
Incept5

Maurice Naftalin

Developer, designer, architect, teacher, learner, writer



Co-author



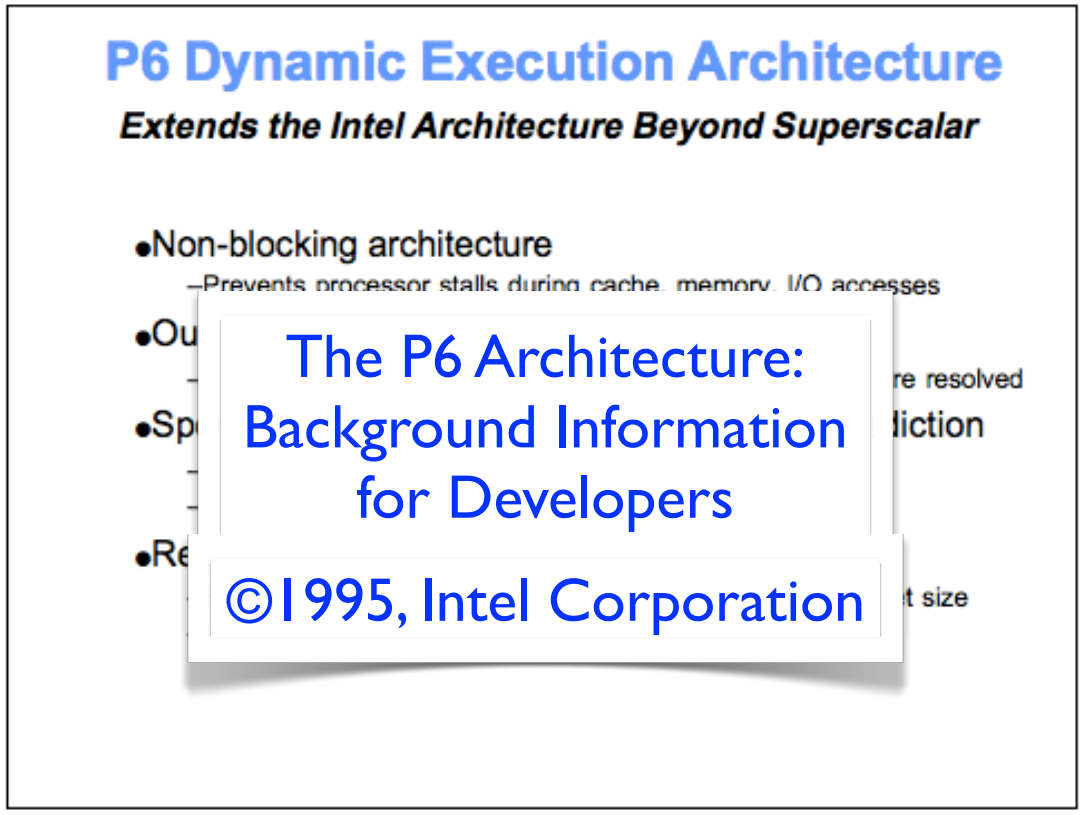
Current Projects

Jpoint

What's New in Java 8

- The Big Picture
- Date and Time API
- Type Annotations
- ... , ... ,

Always Later Than You Think



Was Java Really Asleep?

The library and VM developers certainly weren't asleep! Java 5 introduced

- JSR 133 — fixing the Java Memory Model
- java.util.concurrent (new locks, blocking queues, atomic variables, non-blocking algorithms)

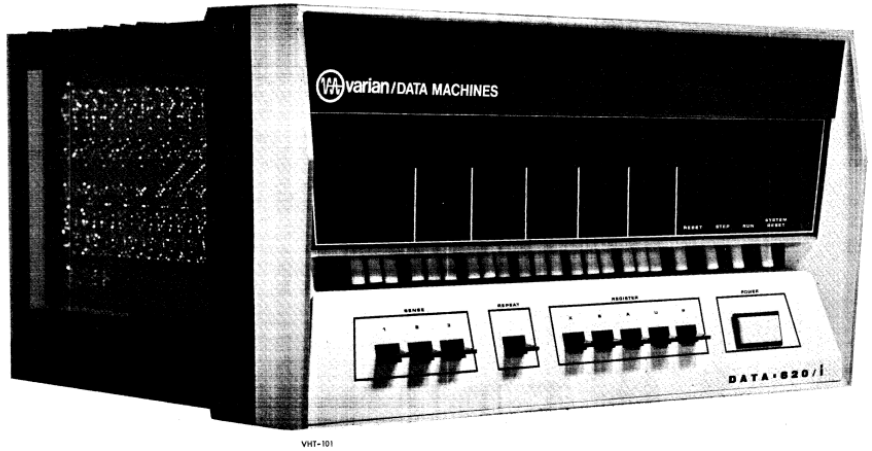
So what's the problem?

- Without adequate synchronization, the Java Memory Model allows
  - race conditions
  - data visibility problems
  - early writes, word tearing, ...

But why would anyone get synchronization wrong? :)

Programming Used to be *Really* Hard

Varian 620/i



Fast operation:	1.8-microsecond memory cycle.
Large instruction repertoire:	107 standard, 18 optional; with approximately 200 additional instruction configurations which can be microcoded.
Word length:	16- or 18-bit configurations.
Modular memory:	4096 word minimum, 32,768 words maximum.

Writing machine code on the bare metal, there's a lot to remember

The Progress of Programming

There's been a lot of progress:

- Assemblers let us forget opcodes
- Linkers let us forget absolute data location
- Compilers let us forget register allocation and stack management
- Virtual memory let us forget about paging
- Garbage collectors let us forget memory management

Progress is being allowed to forget things!

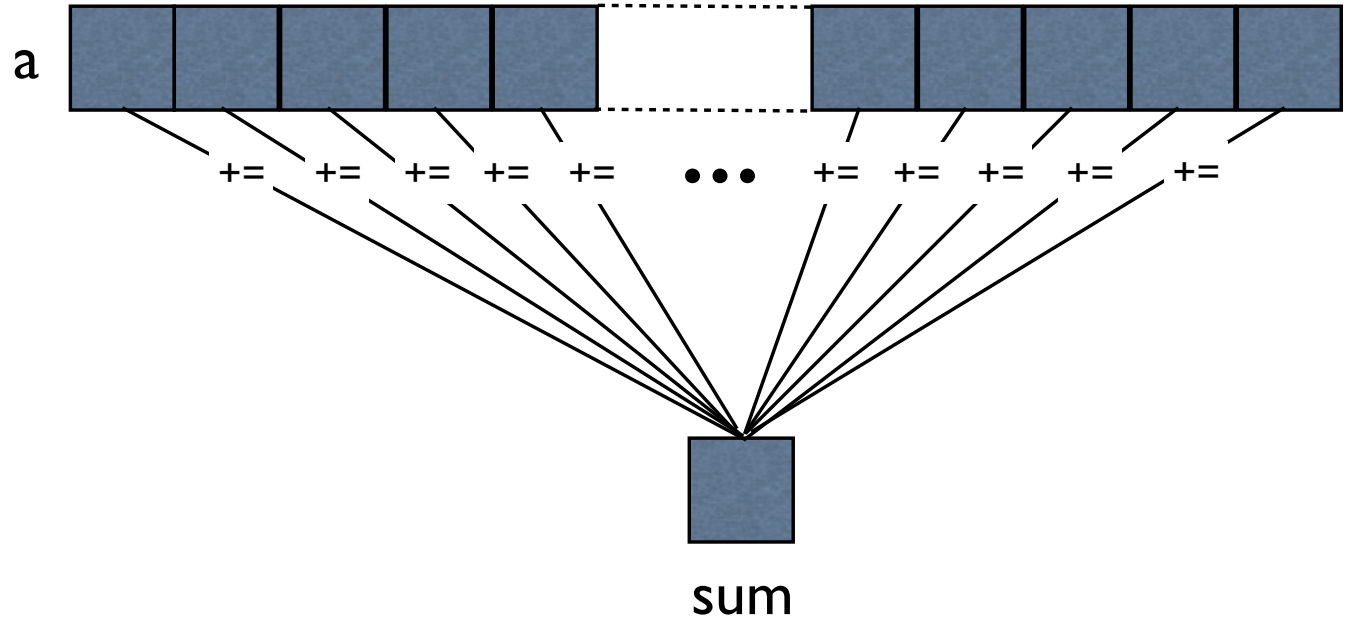
How can we forget about parallelism?

Why Can't We Forget About Parallelism?

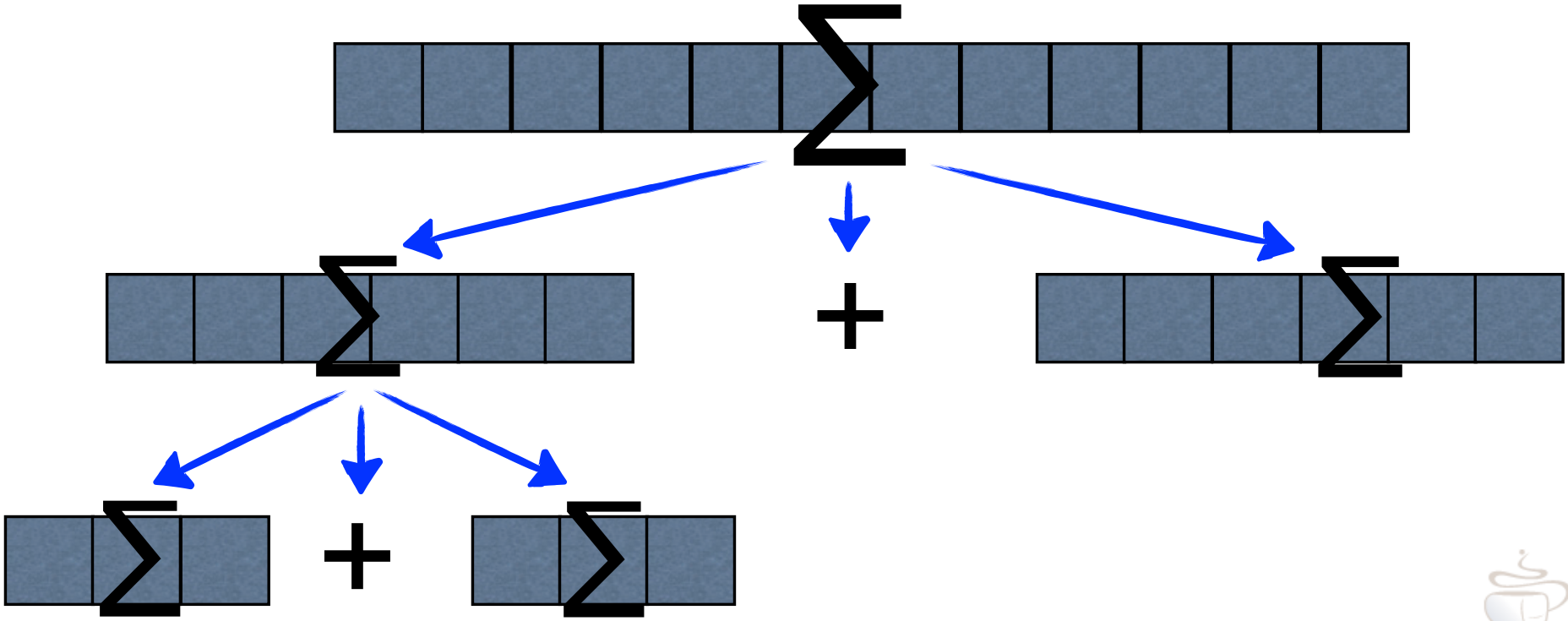
Because we keep writing code like this:

```
int sum = 0;
for (int i = 0 ;  i < a.length ; i++) {
    sum += a[i];
}
```

Mutable State Makes Parallelism Hard



Recursive Decomposition is "Easier"



Let the Library Writers do it!

Collections developers know the recursive structure of their data

But right now they can't use that knowledge:

```
int sum = 0;
for (Iterator<Integer> itr = myList.iterator() ;  itr.hasNext() ; ) {
    sum += itr.next();
}
```

The problem is ***external iteration***.

Internal Iteration

Basic idea:

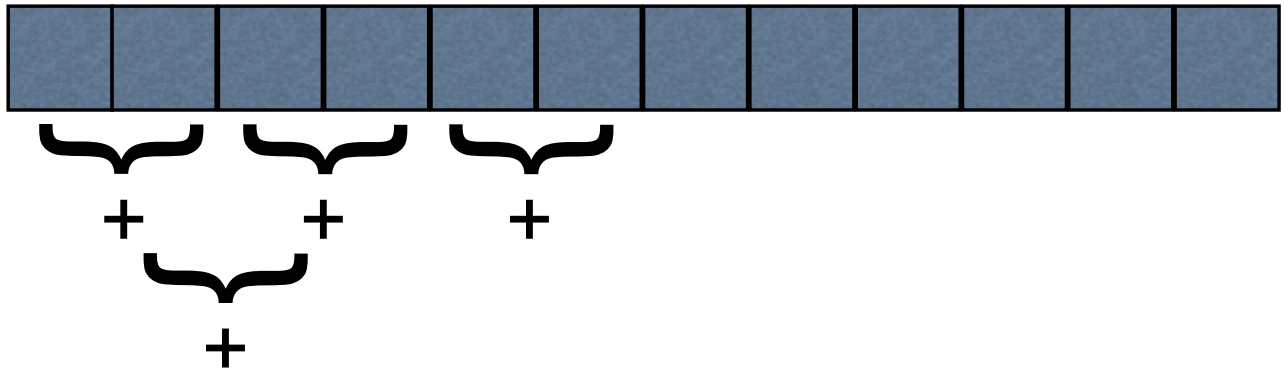
Let the collection choose its iteration strategy (parallel, serial, out-of-order, etc)

Instead of for we write forEach (examples)

Key is *abstraction over behaviours*

Summing an Integer List

Provide the basic operation, collection uses it to implement recursive decomposition



We just need to say “add two elements together” (<show, without receiver>)

Associative!

History

java.util.Date, java.util.Calendar

- strong candidates for the all-time worst platform library design

Joda Time

- quality date and time library

JSR-310

- builds on experience of Joda Time

Goals

- Comprehensive model for date and time
- Supporting commonly used global calendars
- Immutable, so as to work well with lambdas/functional
- Type-safe

Design Principles

Immutable

- thread-safe, allows caching

Fluent

- easy to read, like a DSL

```
LocalDate.of(2010, Month.DECEMBER, 3). withYear(2011).with(Month.May);
```

Extensible

Two ways of 'counting' time

- Continuous, designed for machines
  - Single incrementing number
  - java.time.Instant — nanos from 1970-01-01T00:00:00Z

```
Instant start = Instant.ofEpochMilli(123450L);
Instant end = Instant.now();
assert start.isBefore(end);
assert end.isAfter(start);
```

Two ways of 'counting' time

- Field-based, designed for humans
  - Year, month, day, hour, minute, second
  - LocalDate, LocalDateTime, ZonedDateTime, Period, Duration, ...

```
Duration duration = Duration.ofSeconds(12);
Duration bigger = duration.multipliedBy(4);
Duration biggest = bigger.plus(duration);

Instant later = start.plus(duration);
Instant earlier = start.minus(duration);
```