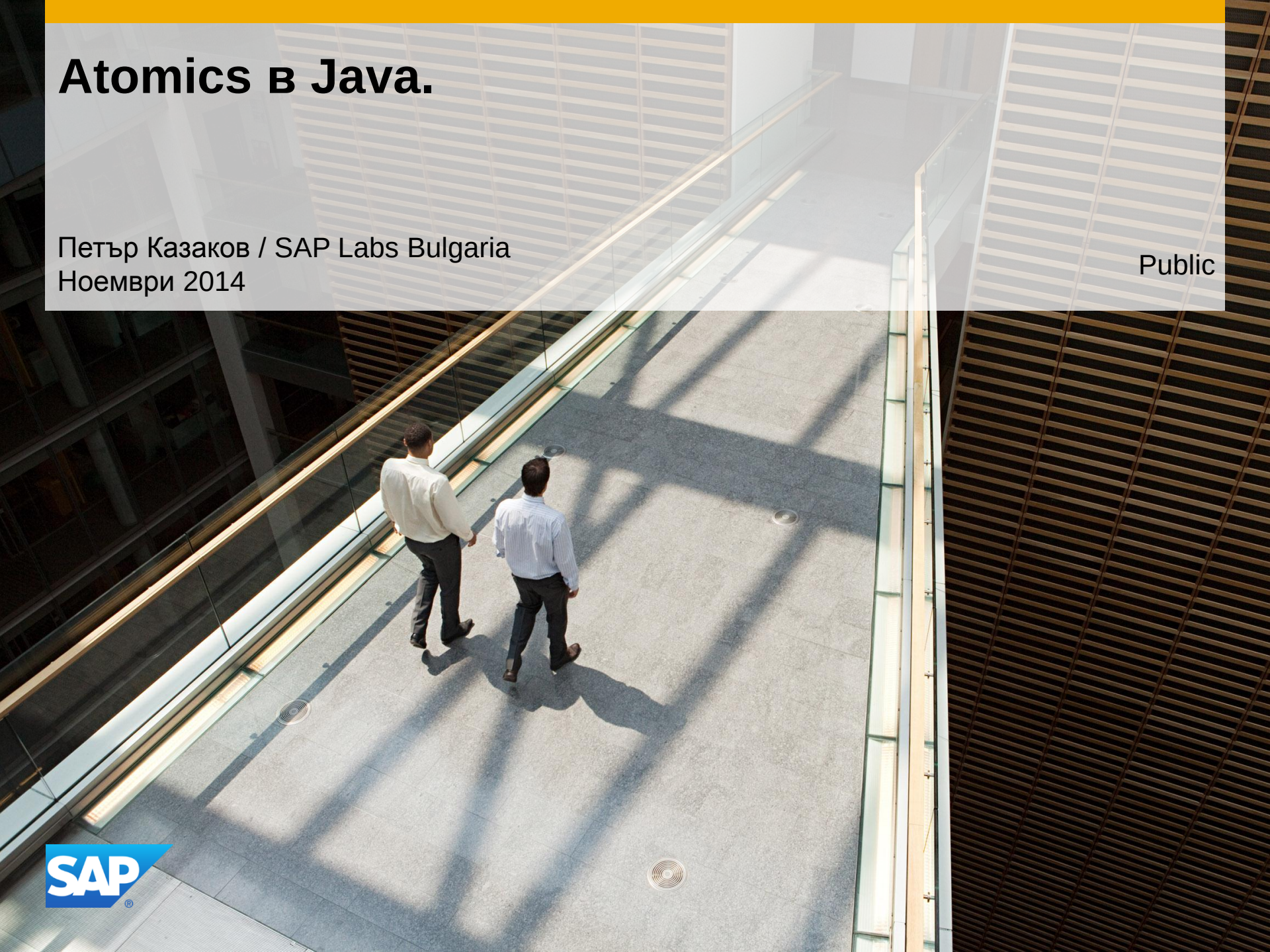
Atomics в Java.

Петър Казаков / SAP Labs Bulgaria
Ноември 2014

Public



Проблем на конкурентния достъп

Конкурентен достъп до определен ресурс:

- Нишка 1 и Нишка 2 извличат стойността на определена променлива
- Нишка 1 и Нишка 2 извършват операция с тази променлива
- Нишка 1 и Нишка 2 записват резултата в променливата
- Резултат: операцията е извършена веднъж
- Очакван резултат: операцията е извършена два пъти

Compare-and-swap

CAS е атомарна инструкция използвана за постигане на синхронизация в многонишкови програми. Сравнява съдържанието на паметта с дадена стойност и я променя само в този случай с нова стойност. Извършва се като атомарна операция.

Част от IBM 370 (и последващи) архитектури от 1970.

Към 2014, почти всички мултипроцесорни архитектури имат поддръжка на CAS. Тази операция е най-популярната примитивна операция за имплементиране на бариери и не блокиращи конкурентни структури от данни.

На практика …

```
public void increment(AtomicInteger integer){
        while(true){
                int current = integer.get();
                int next = current + 1;
                if(integer.compareAndSet(current, next)){
                        return;
                }
        }
}
```

Пакет java.util.concurrent.atomic

AtomicBoolean

AtomicInteger

AtomicIntegerArray

AtomicLong

...

AtomicReference<V>

AtomicReferenceArray<E>

DoubleAccumulator

DoubleAdder

Клас AtomicInteger

int addAndGet(int delta)

int getAndAdd(int delta)

int decrementAndGet()

int getAndDecrement()

int incrementAndGet()

boolean compareAndSet(int expect, int update)

//non volatile writes

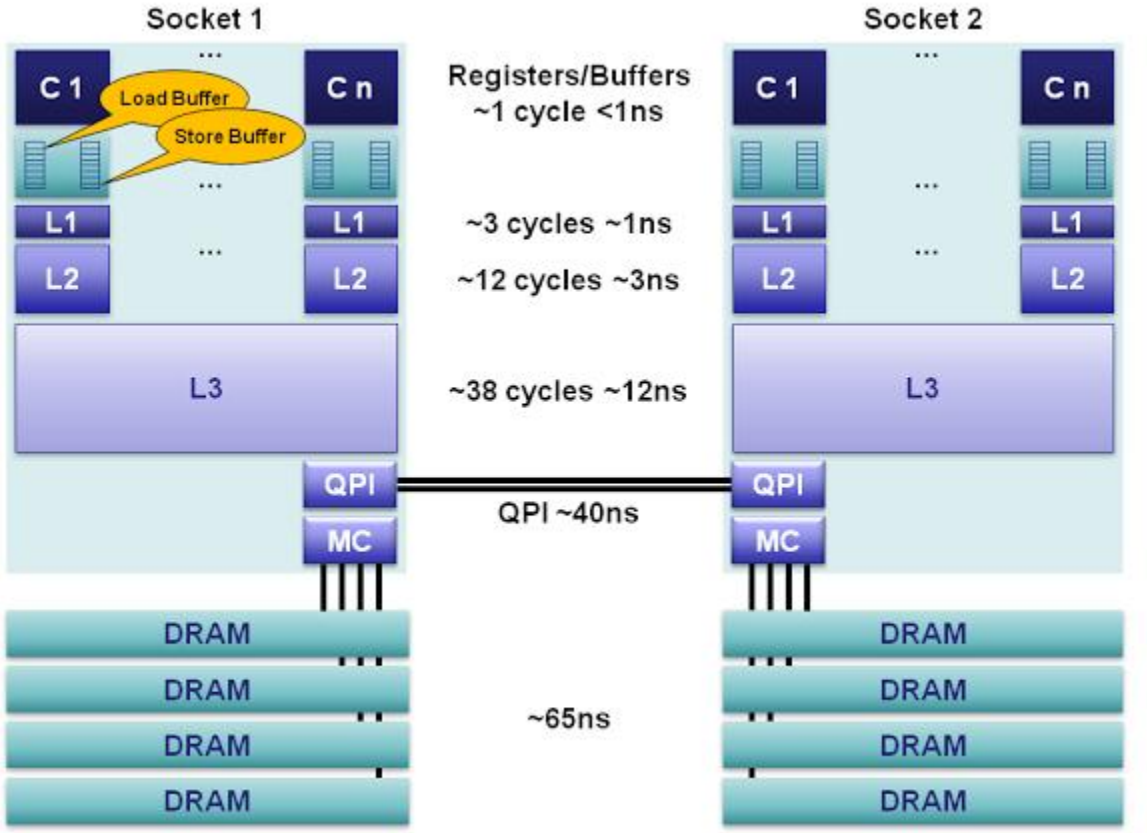
boolean weakCompareAndSet(int expect, int update)

void lazySet(int newValue)

Atomics vs volatile

Атомарните класове могат да се считат като обобщение на volatile променливите, разширявайки концепцията с поддръжка на атомарни compare-and-set промени

Четенето и записването атомарни променливи има същата семантика за четене и запис като на volatile променливи.



Клас AtomicReference

```
public static AtomicReference<String> shared = new AtomicReference<>();

String init="Inital";

shared.set(init);

//modification

prevValue=shared.get();
String newValue=shared.get()+"lets add something";

//single instruction
boolean worked=shared.compareAndSet(prevValue,newValue);
```

Благодаря за вниманието!

Петър Казаков
Peter.kazakov@sap.com

ИЗПОЛЗВАНА ЛИТЕРАТУРА

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html

https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/atomic/AtomicInteger.html

http://en.wikipedia.org/wiki/Compare-and-swap

http://mechanical-sympathy.blogspot.ru/2013/02/cpu-cache-flushing-fallacy.html