Back from the 70s

- The Concurnas concurrency model!

Jason Tatton
concurnas.com

© 2020, Concurnas Ltd. All rights reserved.

Jason Tatton

- **Created Concurnas between 2017-2020**
- **Coding since age 9**
- **10 years experience in building automated trading systems**
- **Working at Amazon web services – Corretto OpenJDK team: aws.amazon.com/corretto**

jason.tatton@concurnas.com

concurnas.com

discord.gg/jFHfsqR

@concurnas

linkedin.com/company/concurnas

© 2020, Concurnas Ltd. All rights reserved.

In this talk

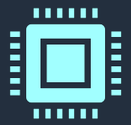
- Motivation
- Concurnas overview
- + Q&A
- Concurrency
- + Q&A
- GPU computing
- + Q&A
- Other interesting topics
- Future developments

© 2020, Concurnas Ltd. All rights reserved.

Motivation

Performance
Most code needs to be fast

Productivity
Developer performance

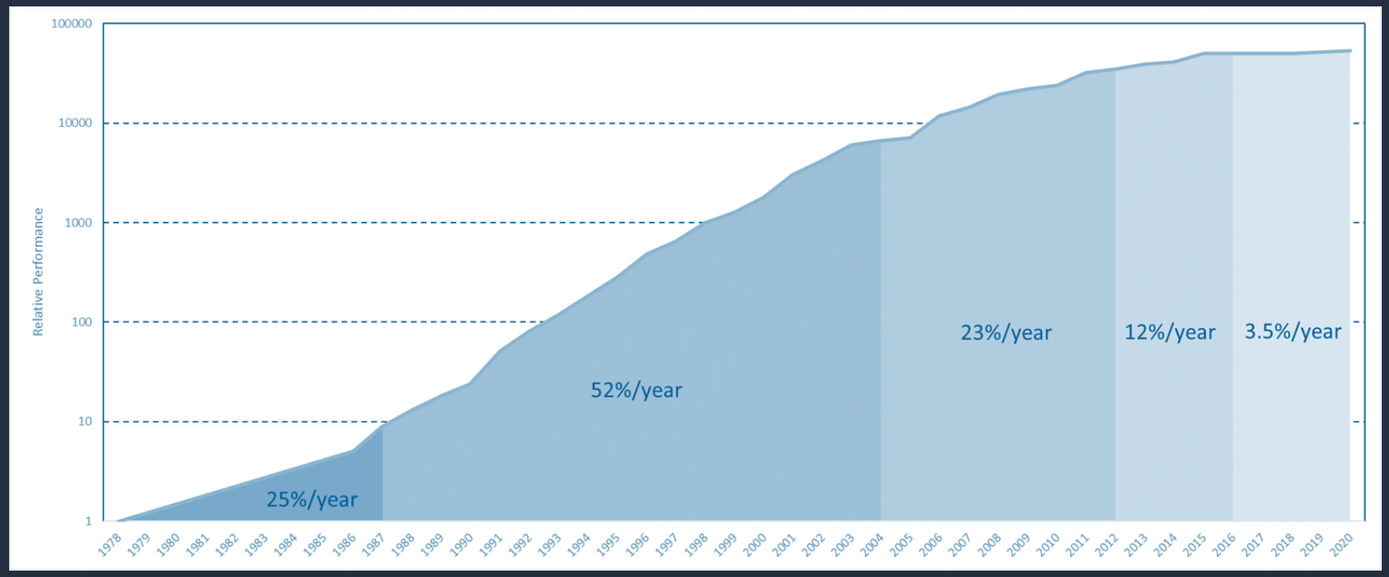


Hardware architecture
Multi-core CPU and GPU's

Modern problems
Multi Domain, mutli paradigm

Heterogenous teams
Not everyone is a software engineer

Open Source
MIT license



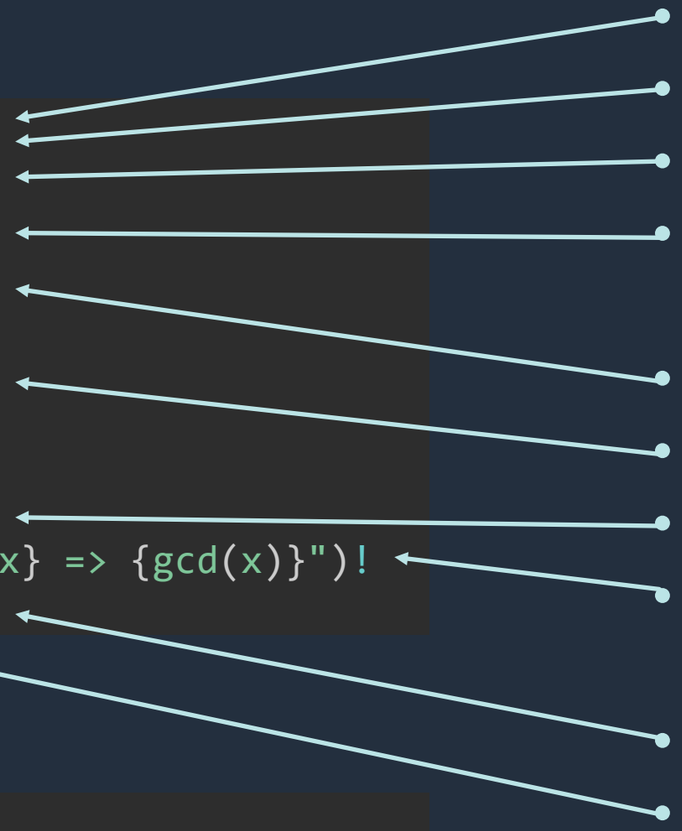
© 2020, Concurnas Ltd. All rights reserved.

Hello Word!

```
def gcd(x int){
  y = 3
  while(y){
    (x, y) = (y, x mod y)
  }
  x
}

for(x in 101 to 105){
  System.out.println("hello world {x} => {gcd(x)}")!
}
```

Output:

```
hello world 102 => 3
hello world 104 => 1
hello world 101 => 1
hello world 105 => 3
hello world 103 => 1
```



- Functions may exist in isolation
- Function return type is implicit
- Type of y is inferred
- All types can be used in Boolean expressions
- Tuples are supported
- return statement is implicit
- Numerical range expression
- ! creates a concurrent ‘isolate’ (light-weight thread)
- String formatting
- Utilizing the Java class library

Your output may vary!

© 2020, Concurnas Ltd. All rights reserved.

Concurnas Overview

© 2020, Concurnas Ltd. All rights reserved.

Introduction *paradigms*

Imperative

- *Assignment, expressions, control stmts*
- *Type inference*
- *Nested functions*
- *Optionally concise functions*
- *Implicit return values*
- *All blocks can return values*

Functional

- *Lambdas*
- *Function references*
- *Partial functions*
- *List comprehensions*
- *Pattern matching*
- *Lazy evaluation*

- *Multi-core CPU*
- *Distributed*
- *GPU*
- *Expression lists*
- *Lang extensions*

Object Oriented

- *Optionally compact class syntax*
- *Traits,enum,generics,annotations*
- *Null safe*
- *Unchecked exceptions*
- *Operator Overloading*
- *Extension functions*
- *Object providers*

Reactive

- *Isolates, refs*
- *Actors*
- *onchange/every*
- *Optionally concise syntax*
- *Software Transactional Memory*
- *Temporal computing*

© 2020, Concurnas Ltd. All rights reserved.

Imperative – Expressions and variables

```
π = Math.PI     // unicode
πString = "π=" + π // π=3.141592653589793
/* ~~~ multiline comment ~~~ */
10, 9., 10e5, 10.0f, 345345L, true, 'c' // normal constants

[1 2 3 4]    // an integer array
[1, 2, 3, 4] // an integer list
[1 2 ; 3 4]  // an integer matrix definition

2+2**4 mod 5            // expression
a == 3 or b < 9         // boolean expression
2 if something() else 15 // if expression
```

```
anint = 8
avar int = 99        // explicit type
var reassignOk = 12  // new variable
val nonReassign = 99 // new variable cannot reassign
```

© 2020, Concurnas Ltd. All rights reserved.

Imperative – Control flow

```
for(a in [1 2 3]){
 processIt(a)
} // iterator for

for(a = 1; a <== 3; a++) {
 processIt(a)
} // c style for

while(xyz()){
 doSomething()
}

loop{ //while(true)
 if(doSomething()){
  break
 }
}
```

```
if(a){
 doThis()
}elif(b){
 doAnother()
}else{
 another()
}
```

```
while(xyz()){
 doSomething()
}else{ // if while loop never entered
 onfail()
}
```

© 2020, Concurnas Ltd. All rights reserved.

Imperative – All blocks return

```
speed = {
 distance = 100
 time = 24.
 distance/time
} // all bracketed blocks return
```

```
astring = if(condition()){
 "value1"
} else{
 "value2"
}
```

```
pows = for(x in 2 to 6){
 2**x
}
// pows == [4, 8, 16, 32, 64]
```

© 2020, Concurnas Ltd. All rights reserved.

Imperative – Functions

```
// an ordinary function:
def adder(a int, b int) int {
 return a + b
}

// we can use => to compact the function to:
def adder(a int, b int) int => return a + b

// infer the return type:
def adder(a int, b int) => return a + b

// implicit return expression - most compact form!
def adder(a int, b int) => a + b
```

```
//calling a function
adder(8, 7) // == 15
```

© 2020, Concurnas Ltd. All rights reserved.

Imperative – Functions

```
def adder(a int, b int)   => a + b
def adder(a int, b float) => a + b
def adder(a int) => adder(a, 10) //overloading
```

```
def manyAdder(a int, nums int...) => for(n in nums){n + a}

//varargs called as:
manyAdder(10, 1, 2, 3) // returns: [11, 12, 13]
```

```
def wdefaults(a=10, b=2, c=3) => a + b * c //default args

wdefaults()     //== 16

wdefaults(c=10) //named parameter == 30
```

© 2020, Concurnas Ltd. All rights reserved.

Imperative – Exceptions

```
class ArgumentException(msg String) < Exception(msg)

def process(a int) int {
 if(a < 2){
  throw new ArgumentException("a is smaller than 2")
 }
 a ** 2
}

result = try{
 process(1)
}catch(e ArgumentException){
 0 //react as appropriate
}catch(e){
 throw e //re-throw
} finally{
 afterProcCall() //always called
}
```

© 2020, Concurnas Ltd. All rights reserved.

Object Oriented – classes

```
class Person{
 private name String
 private surname String
 private likes = java.util.HashSet<String>()

 this(name String, surname String){
  this.name = name
  this.surname = surname
 }
 this(surname String) => this('dave', surname)

 public getName() => this.name
 public setName(name String) => this.name = name

 public getSurname() => this.surname
 public setSurnae(surname String) => this.surname = surname

 def addLike(like String) => likes.add(like)
}
```

```
class Person(~name String, ~surname String){
 this(surname String) => this('dave', surname )
 likes = java.util.HashSet<String>()
 def addLike(like String) => likes.add(like)
}
```

© 2020, Concurnas Ltd. All rights reserved.

Object Oriented – classes

```
class Person(~name String, ~surname String){
 this(surname String) => this('dave', surname )
 likes = java.util.HashSet<String>()
 def addLike(like String) => likes.add(like)
}
```

```
p1 = new Person(‘talyor’)
p2 = Person('amber', 'smith')
```

```
p1.addLike('sprouts') //method call
oldname = p1.name //same as: oldname =p1.getName()
p1.name = "jon"      //same as: p1.setName("jon")
```

```
p3 = p1@ //copy operator

assert p1 == p3 //== p1.equals(p3)
assert p1 &<> p2//p1 not equal to p3 via ref

people = set()
people.add(p1)
people.add(p3)
assert people.size() == 1//one item stored by value
```

© 2020, Concurnas Ltd. All rights reserved.

Object Oriented – Traits and generics

```
abstract class AbstractFooClass{
 def foo() => "version AbstractFooClass"
}

trait A{ def foo() => "version A" }
trait B{ def foo() => "version B" }

class FooClass extends AbstractFooClass with B, A{
 override def foo() => "" + [super[AbstractFooClass].foo(), super[A].foo(), super[B].foo()]
}

FooClass().foo() //returns [version AbstractFooClass, version A, version B]
```

```
class Pair<X, Y>(-x X, -y Y) //generic class

p1 = Pair<String, int>("one", 1)
p2 = Pair("name", "another")
```

© 2020, Concurnas Ltd. All rights reserved.

Object Oriented – Null safety

```
aString String = "something"
aString = null //compilation error, aString is not of a nullable type.
```

```
aString String? = "something"
aString = null //this is ok
```

```
len = aString.length()       // compilation error
```

```
len = aString?.length()      // ok - null handled
```

```
len = (aString?:"").length() // ok - null handled
```

```
len = aString??.length()     // ok - null handled (sort of)
```

```
len = if(null == aString){
 -1
}else{
 aString.length() // ok - cannot be null
}
```

© 2020, Concurnas Ltd. All rights reserved.

Functional – Method references and Lambdas

```
def plus(a int, b int) => a + b

op2 (int, int) int = plus&

result = op2(10, 1)
```

```
op (int) int = plus&(10, int)

result = op(1)
```

```
def toEach(opon int[], func (int) int) {
 for(o in opon) {
  func(o)
 }
}

toEach([1 2 3], op)
```

```
toEach([1 2 3], a => a+10) // lambda definition
```

© 2020, Concurnas Ltd. All rights reserved.

Functional – Pattern matching

```
class Person(-yearOfBirth int)

def matcher(an Object){
 match(an){
  Person(yearOfBirth < 1970) => "Person. Born: {an.yearOfBirth}"
  Person    => "A Person"
  int; < 10 => "small number"
  int       => "another number"
  x         => "unknown input"
 }
}

res = matcher(x) for x in [Person(1829), Person(2010), "oops", 43, 5]
//res == [Person. Born: 1829, A Person, unknown input, another number, small number]
```

© 2020, Concurnas Ltd. All rights reserved.

Implementation

- Core – Java (90%) and Concurnas (10%)
- ANTLR – Lexing and parsing - www.antlr.org
- ASM – Bytecode generation - asm.ow2.io
- Iterative multi-pass frontend oriented compiler/interpreter
- Heavy use of Visitor pattern
- REPL support is provided

© 2020, Concurnas Ltd. All rights reserved.

Q&A

Mini break 1

© 2020, Concurnas Ltd. All rights reserved.

Concurrency

© 2020, Concurnas Ltd. All rights reserved.

Concurrency - Isolates

```
def gcd(x int){
  y = 25
  while(y){
    (x, y) = (y, x mod y)
  }
  x
}

anarg = 92312
res1 int: = {anarg += 3; // int: is a ref type
             gcd(anarg)
            }!
res2      = gcd(anarg)!

larger    = res1 if res1 > res2 else res2
```

© 2020, Concurnas Ltd. All rights reserved.

Concurrency - Reactive

```
assetprice int:
// Other code interacting with assetprice...
every(assetprice){
 // perform action here...
}
```

```
asset1price int:; asset2price int:

every(asset1price, asset2price){
 if(asset1price > asset2price){
  // ... initiate trading action here!
  return // terminate future invocation of the every block
 }
}
```

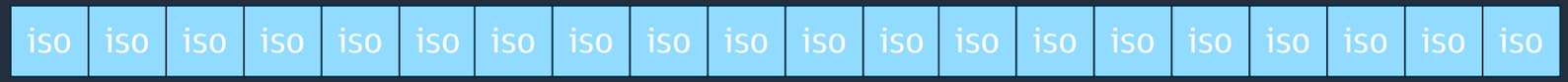
```
a int:; b int:

c = every(a, b){ a + b }

every(c){
 System.out.println("latest sum: {c}")
}
```

```
c <= a + b
```

© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

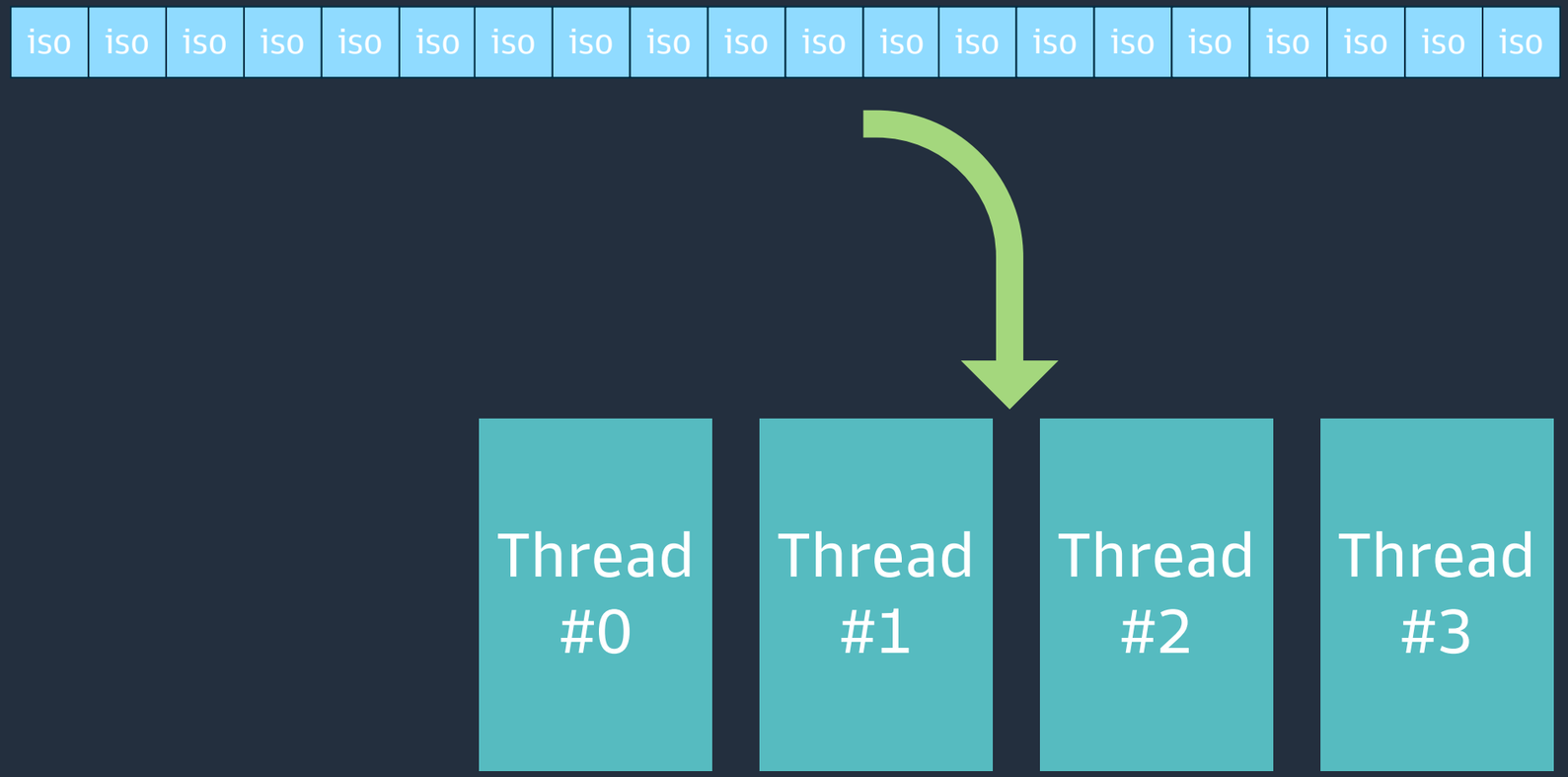
- Isolates multiplexed on to threads
- Isolates are cooperative



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

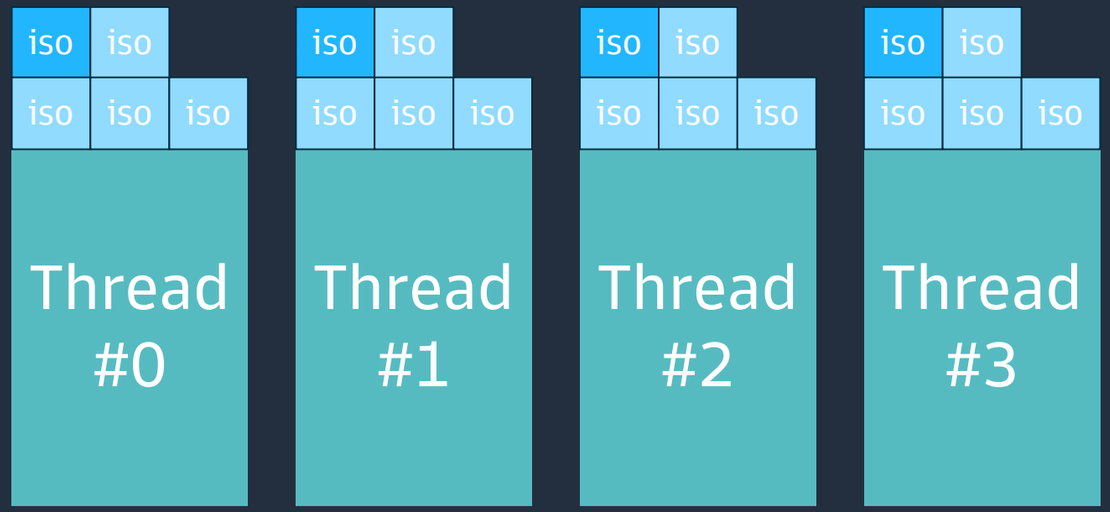
- Isolates multiplexed on to threads



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

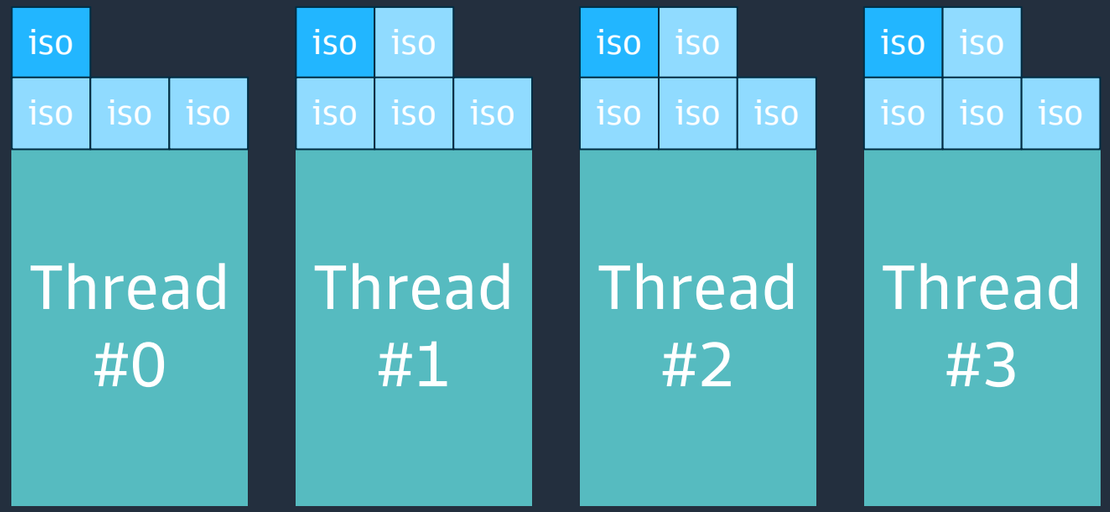
- Isolates multiplexed on to threads



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

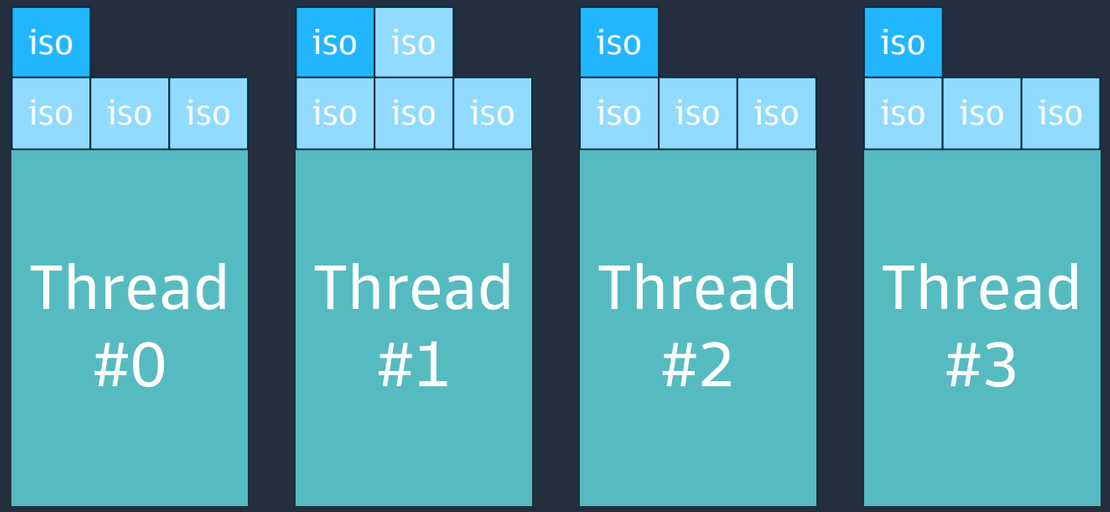
- Isolates multiplexed on to threads



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

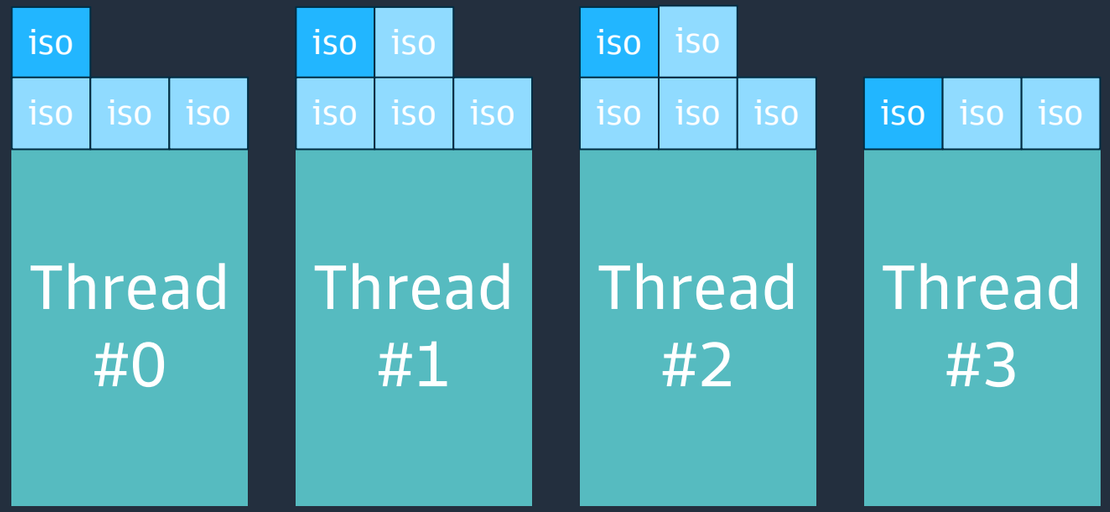
- Isolates multiplexed on to threads
- Isolates execute in a FIFO manner



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

- Isolates multiplexed on to threads
- Isolates execute in a FIFO manner
- Isolates are cooperative



© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

- “One-shot delimited continuations”
- Package up state on pause point and yield to another isolate
- Reconstruct state on resumption
- Achieved via extra argument to all method calls

```
def foo(){
 // code here...
 bar()
 // code here...
}

def bar(){
 // pauses execution and yields
}
```

```
:isolate#0
|-> foo()
 |-> bar()

:isolate #1
|-> ...

:isolate #2
|-> ...

    ...

:isolate#28534
|-> ...
```

```
def foo(f Fiber){
 // code here...
 bar(f)
 // extra code to (un)package state
 // code here...
}

def bar(f Fiber){
 // extra code to package up state using f
 f.pause(); return
 // pauses execution and yields
}
```

Thread #2

© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

- Ref's implement pause and wakeup functionality

```
class Ref<X>{
 value X
 def get(f Fiber) X
 def set(value X, f Fiber)
}
```

```
myref int:
myref = 80    // myref:set(80)
valu  = myref // myref:get()
```

```
class Ref<X>{ // pseudocode
 value X

 def get(f Fiber) X{
  if(value not set){
   rememberCaller(f) // maintain list of callers
   f.pause()         // yield execution
  }
  return this.value
 }

 def set(value X, f Fiber){
  this.value = value
  f.wakeup(list of callers)
 }
}
```

© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

- Achieved at runtime via bytecode manipulation.
- This applies to standard Java code as well!
- For Java: https://github.com/kilim/kilim

```
def foo() {
 state = ...
 bar()
 // other code...
}
```

```
def foo(f Fiber){
 match (f.pc) {
  case 0: goto START
  case 1: goto CALL_BAR
 }
 START: state = ...
 CALL_BAR: // pre_call
  f.down()
  bar(f)
  f.up() // post-call
 match (f.status) {
  case UnPauseNoState:
   goto RESUME
  case UnPauseState:
   restore state
   goto RESUME
  case PauseState:
   capture state
   return
  case PauseNoState:
   return
  }
  RESUME:
   // other code...
}
```

© 2020, Concurnas Ltd. All rights reserved.

Concurrency implementation

- Interesting challenges:
    - The `new` keyword.
    - `static` variables and methods.
    - Augmenting bootstrap classes.
- Compared to Kotlin:
    - `suspend` keyword not required.
    - All Java methods (including bootstrap) are pausable.
    - First class citizen support, not higher order functions
- Enables: Actors, Distributed computing, Temporal Computing, Software transactional memory, GPU computing...

© 2020, Concurnas Ltd. All rights reserved.

Concurrency – Actors

```
actor IdGenerator(prefix String){
 cnt = 0//implicit private state
 def getNextId(){
  toReturn = prefix + "-" + cnt
  cnt += 1
  toReturn
 }
}

idGen = IdGenerator("IDX")//create an actor
anId1 = idGen.getNextId()//==> IDX-0
anId2 = idGen.getNextId()//==> IDX-1
```

```
setService = actor java.util.HashSet<int>()
setService.add(65)
```

© 2020, Concurnas Ltd. All rights reserved.

Q&A

Mini break 2

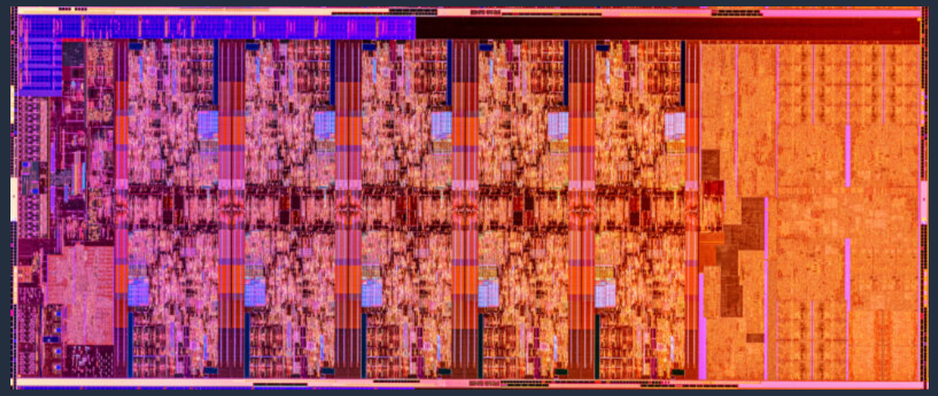
© 2020, Concurnas Ltd. All rights reserved.

GPU computing

© 2020, Concurnas Ltd. All rights reserved.

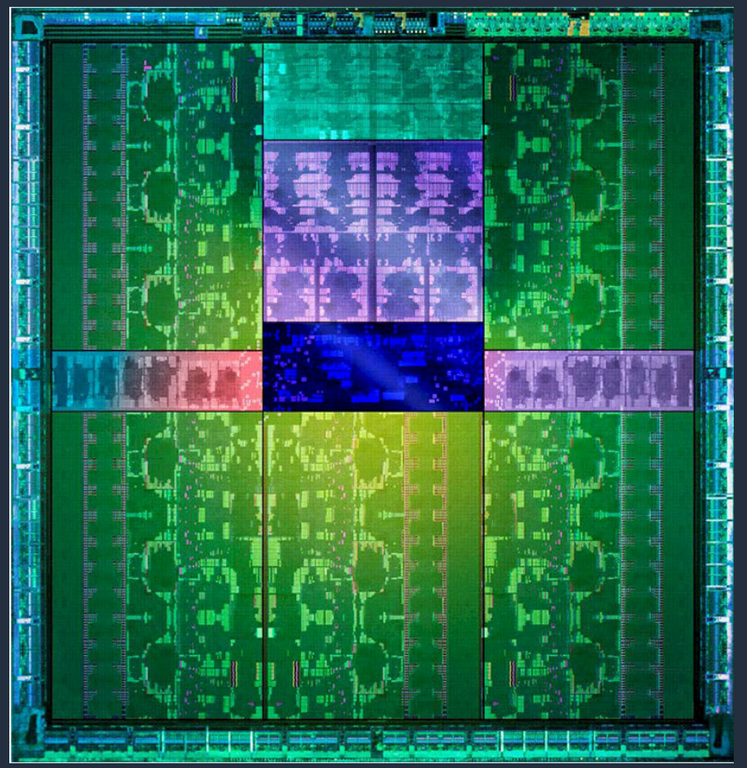
Motivation

Modern CPU Die



64 cores

Modern GPU Die



4,000+ cores

© 2020, Concurnas Ltd. All rights reserved.

GPU programming

```
A = [ 1.f 2 3 4 5 6 7 8 9 10 ]
B = [ 1.f 2 3 4 5 6 7 8 9 10 ]
result = new float[A.length]

for(idx in 0 to A.length-1){
 result[idx] = A[idx]**2 + B[idx] + 10
}
```

```
gpukernel 1 twoArrayOp(global in A float[], global in B float[], global out result float[]){
 idx = get_global_id(0)
 result[idx] = A[idx]**2 + B[idx] + 10
}
```

© 2020, Concurnas Ltd. All rights reserved.

GPU programming

```
gpukernel 1 twoArrayOp(global in A float[], global in B float[], global out result float[]){
 idx = get_global_id(0)
 result[idx] = A[idx]**2 + B[idx] + 10
}
```

```
// select a GPU device...
device = gpus.GPU().getGPUDevices()[0].devices[0]

// we create three arrays of size 10 on this GPU, 2 as input
inGPU1 = device.makeOffHeapArrayIn(float[].class, 10)
inGPU2 = device.makeOffHeapArrayIn(float[].class, 10)
result = device.makeOffHeapArrayOut(float[].class, 10) // 1 output

//now we write to the arrays on the GPU
c1 := inGPU1.writeToBuffer([ 1.f 2 3 4 5 6 7 8 9 10 ])
c2 := inGPU2.writeToBuffer([ 1.f 2 1 2 3 1 2 1 2 1 ])
```

```
inst = twoArrayOp(inGPU1, inGPU2, result)
compute := device.exe(inst, [10], c1, c2)//run 10 cores to process
ret = result.readFromBuffer(compute)
```

© 2020, Concurnas Ltd. All rights reserved.

GPU implementation

- Makes use of OpenCL for compatibility
    - With Nvidia and AMD
    - With FPGA's
- Transpilation into OpenCL C99
    - Some restrictions on language features
- Leverages Java bindings for OpenCL:
    - JNI based
    - http://www.jocl.org/
    - Linux, Mac and Windows compatible

© 2020, Concurnas Ltd. All rights reserved.

Q&A

Mini break 3

© 2020, Concurnas Ltd. All rights reserved.

Other interesting topics

© 2020, Concurnas Ltd. All rights reserved.

Working with Data

```
anArray = [1 2 3 4 5 6]
aList   = [1,2,3,4,5,6]
aMatrix = [1 2 3 ; 4 5 6]
aMap = {"one" -> 1, "two" -> 2, "three" -> 3}

cont = "one" in aMap      // checking for a value in a map
del aMap["one"]           // remove element from aMap
arrayValue = anArray[2]   // individual value from array
arrayValue = aMatrix[0,1] // individual value from matrix
subarray = anArray[4 ...] // a sub array; [5 6]
```

```
longNames = aMap[key] for key in aMap if key.length() > 3
ret = i+10 for i in aList if i mod 2 == 0
```

```
def getDetails() => ("dave", 27) // returns a tuple

(name, age) = getDetails()       // tuple decomposition
```

```
reversed(enumerate(zip([1,2,3], [4,5,6])))
// [(2, (3, 6)), (1, (2, 5)), (0, (1, 4))]
```

© 2020, Concurnas Ltd. All rights reserved.

Vectorization and ranges

```
mat = [1 2 ; 3 4]

mat2 = mat^*2 + 1 //==> [3 5 ; 7 9]

mat^^*2 + 1       //in-place vectorized operation.

mat3 = mat + 1    //implicit vectorization
```

```
numRange = 0 to 10        // a range of: [0, ..., 10]
tepRange = 0 to 10 step 2 // a range of: [0, 2, ..., 10]
revRange = tepRange reversed // a reversed range of: [10, 8, ..., 0]
decRange = 10 to 0 step 2 // a range of: [10, 8, ..., 0]
infRange = 0 to           // an infinite sequence [0,... ]
steInfRa = 0 to step 2    // a stepped infinite sequence [0, 2,... ]
decInfRa = 0 to step (-1) // a stepped infinitely decreasing sequence [0, -1,... ]

val = x for x in numRange //list comprehension over a range
check = 2 in numRange     //checking for the presence of a value
```

© 2020, Concurnas Ltd. All rights reserved.

DSL & Language Extensions

```
'concurrent'.substring(3)

'concurrent' substring 3
```

```
def int min() => java.time.Duration.ofMinutes(this)

10 min
```

```
class Complex(real double, imag double){
 def + (other Complex) => new Complex(this.real + other.real, this.imag + other.imag)
 def +=(other Complex) => this.real += other.real; this.imag += other.imag
 override toString() => "Complex({real}, {imag})"
}

c1 = Complex(2, 3)
c2 = c1@//deep copy of c1
c3 = Complex(3, 4)

result1 = c1 + c3
c2 += c3 //compound plus assignment
```

```
order = Buy 1 mil gbp after 10 seconds
```

© 2020, Concurnas Ltd. All rights reserved.

DSL & Language Extensions

```
from com.mycompany.myproduct.langs using mylisp, myFortran, myAPL
calc = mylisp||(+ 1 2 (* 2 3))|| // == 9

myFortran|| program hello print *, "Hello World!" end program hello|| //prints "Hello World!"

lotto = myAPL || x[⍋x←6?40] || //6 unique random numbers from 1 to 40
```

```
from com.mycompany.myproduct.langs using mylisp, mySQL, myAPL
class Person(name String, yearOfBirth int)
people list<Person>;

millennials = mySQL||select name from people where yearOfBirth between 1980 and 2000||

myAPL||fact{×/⍳ω}||
fact(10) //use of function defined in myAPL. returns: 3628800
```

```
from com.mycompany.myproduct.langs using mylisp
aString = "i'm a String!“

invalidCode = mylisp||(+ 1 2 (* 2 aString))|| //results in compilation time error
moreInvalidCode = mylisp||(+ 1 2 (* 2 3)||    //oops! Missing a closing ')'
```

© 2020, Concurnas Ltd. All rights reserved.

Performance

On a par with Java

- *~5% stack unrolling from isolate/ref continuations*

Productivity

- *Concurnas requires less code to do more*
- *Null safety leads to better code*

Isolates Scale better

- *Than conventional threads*
- *Easier to reason about*

The JVM is always improving

- *Concurnas gets incremental upgrades for free*

© 2020, Concurnas Ltd. All rights reserved.

Future developments

Core

- Better off heap memory mgmt.
- Improved GPU support
- Improved Iso Scheduling
- Improved resource allocation
- Better higher order functions

Tools

- IDE Support
- Gradle plugin
- Docgen
- Formatter

Compiler

- Faster Compiler
- Concurrent compiler (ironic)
- More Warnings
- Language Server Protocol support
- Findbugs style behaviour

Marketing

- Intro book
- Wikipedia
- TIOBE listing

Labs

- Auto Differentiation
- Use language extensions for Quantum computing

© 2020, Concurnas Ltd. All rights reserved.

Further information

- Concurnas website – concurnas.com
- Download – concurnas.com/download.html
    - Or use SDKMAN! - sdkman.io

```
sdk install concurnas
```

- Contribute – github.com/Concurnas/Concurnas

© 2020, Concurnas Ltd. All rights reserved.

© 2020, Concurnas Ltd. All rights reserved.