



# Fortress 0.62

**David Chase**  
**Sun Microsystems**  
**david.chase@sun.com**

**2005-06-16**

Guy Steele  
Jan-Willem Maessen  
Eric Allen  
David Chase  
Victor Luchangco  
Sam Tobin-Hochstadt  
Sukyoung Ryu  
Yossi Lev  
Carl Eastlund  
Joe Hallett  
John Dias

# A Work in Progress

- Part of Sun's HPCS effort for DARPA
  - > P = Productivity
- In phase 2 (one more year): "reduce risk"
- Fortress 0.618 will change
  - > we want feedback; we don't ignore it

# Design for General-Purpose Apps

- HPC aimed in that direction.
  - > more databases, components, complex data structures.
  - > incorporating multiple languages
- Fast numerical computing + tax-free features.
- Rich features to support library writers.
- Strong support for good programming practices.
- Chip multithreading is coming: existing general languages not a good fit.
- Attract other programmers: enlarge market.
- DARPA wants off-the-shelf technology.

# General Productivity Problems

- unsafe languages are harder to use:  
GC, checking.
- multithreading is tricky:  
transactional memory, implicit threading.
- static (early, comprehensive, expressive) checking is good.
- code readability is good:  
human-friendly syntax.
- contracts and unit tests are good:  
explicit support.

# HPC Productivity Problems

- portable performance is hard
- locality is hard
- programming communication (MPI) is hard:  
cache oblivious algorithms; multiple library implementations; runtime optimization.
- tuning is hard:  
performance reflection.

## What's it like?

- Described by *APIs*; implemented by *components*.
- *Traits*: interfaces with code, classes without fields.
- *Objects*: implement traits, like final classes.
  - > can be parameterized
  - > fields are private
- Method/functions: multiple dispatch, return multiple values, can throw exceptions.
- Runtime can be extended through traits.

## What's it like? (continued)

- Type annotations required in APIs may be inferred in components.
- Implicitly parallel generators; sequential iteration is a special case.
- Everything is an expression.
- Operator overloading.
- Mathematical syntax.
- Extensible syntax.
- (explicit support for dimensions, first-class functions, method overloading, and operators as arbitrary functions/methods)

# Hello world, v0.62

```
component Hello
  import print from io
  export executable
  run(args) = print "Hello world"
end
```

Types inferred



```
api io
  print:String -> ()
end
```

Hello imports



```
api executable
  run:(args:String...) -> ()
end
```

Hello exports



# Generics, constraints, formatting

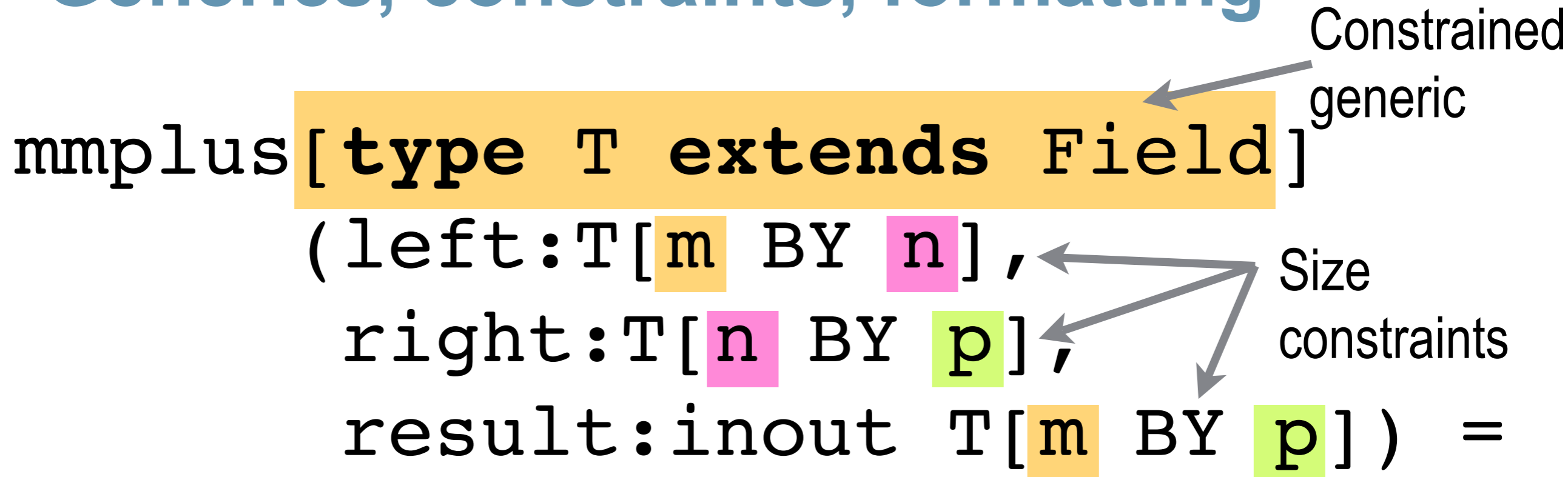
```

mmpus [type T extends Field]
  (left: T[m BY n],
   right: T[n BY p],
   result: inout T[m BY p]) =

```

Constrained generic

Size constraints



```

mmpus[type T extends Field]
  (left :  $T_{m \times p}$ ,
   right :  $T_{p \times n}$ ,
   result : out  $T_{m \times n}$ ) =

```

Readable form

# Formatting, Unicode

Unicode nickname

```
CircularConvolution[type T extends Field]
    ( A:T[0:m BY 0:n],
      P:T[down:up BY left:right],
      B:out T[0:m BY 0:n] ) =
B[x,y] := (SUM[i<-down:up, j<-left:right]
    A[(x+i) MOD m, (y+j) MOD n]*P[i,j],
    x <- 0:m, y <- 0:n)
```

binding low,  
observing limit

observing  
low and limit

*CircularConvolution*[type *T* extends *Field*]  
 (*A* : *T*<sub>[0,m) × [0,n)</sub>,  
*P* : *T*<sub>[down,up) × [left,right)</sub>,  
*B* : out *T*<sub>[0,m) × [0,n)</sub>) =

$$B_{x,y} = \left( \sum_{\substack{i \leftarrow [down, up) \\ j \leftarrow [left, right)}} A_{(x+i) \bmod m, (y+j) \bmod n} \cdot P_{i,j} \right), \quad \begin{matrix} x \leftarrow [0, m) \\ y \leftarrow [0, n) \end{matrix}$$

Inclusive-  
exclusive  
ranges

# An Implementation of Integers

```
value object I64(hi:Bits[32,32], lo:Bits[32,32])
  extends IntegerLike[I64]
  plus(x) = do
    (losum, carry) = lo.plusC(x.lo)
    I64(hi.plusX(x.hi, carry), losum)
  end
  minus(x) = do
    (losum, carry) = lo.minusC(x.lo)
    I64(hi.minusX(x.hi, carry), losum)
  end
  times(x) = do
    (ll_hi, ll_lo) = lo.multiplyUnsigned(x.lo)
    I64(ll_hi + lo.multiply(x.hi) + x.lo.multiply(hi),
        ll_lo)
  end
  ...
end
```

Object (constructor) parameters

Multiple return values

New object

# Self type and operator definition

```
value trait Integer extends IntegerLike[Integer] end
```

```
value trait
```

```
  IntegerLike[type T extends IntegerLike[T]]
```

```
  plus(x:T):T;  minus(x:T):T;  times(x:T):T
```

```
  ...
```

```
end
```

```
opr +[type T extends IntegerLike[T]](x:T, y:T):T =  
  x.plus(y)
```

```
opr -[type T extends IntegerLike[T]](x:T, y:T):T =  
  x.minus(y)
```

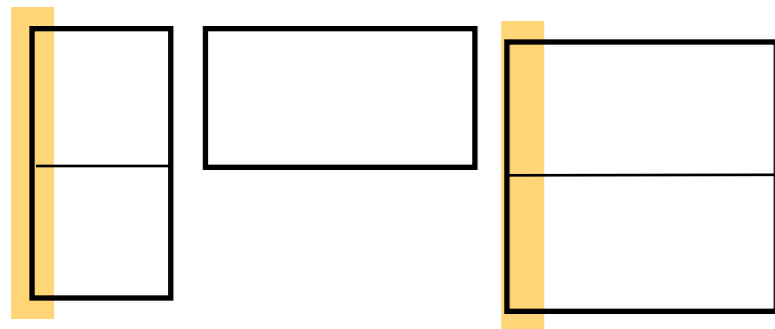
```
opr TIMES[type T extends IntegerLike[T]]  
  (x:T, y:T):T = x.times(y)
```

# Cache oblivious matrix multiplication

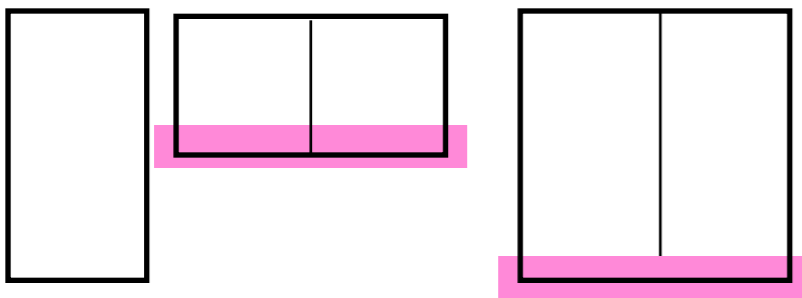
```
mmplus[type T extends Field] (A:T[m BY n],
                                B:T[n BY p],
                                result:inout T[m BY p]):() =
```

```
case largest of
```

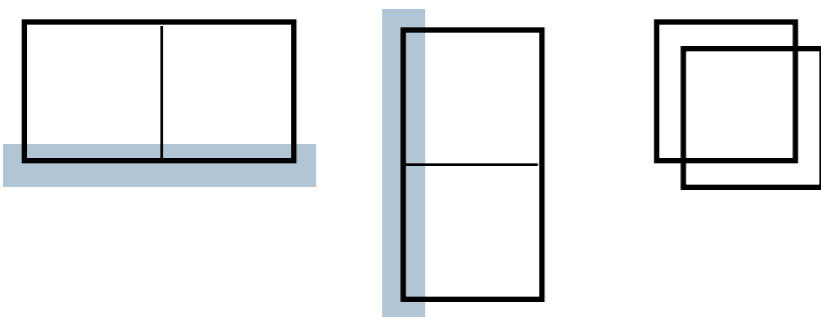
```
1 => result[0,0] = result[0,0]+A[0,0]*B[0,0]
end
```



```
m => [ Atop ; Abottom ] = A
      [ restop ; resbottom ] = result
      t1 = spawn do mmplus(Atop, B, restop) end
      t2 = spawn do mmplus(Abottom, B, resbottom) end
      t1.wait(); t2.wait()
end
```



```
p => [ Bleft Bright ] = B
      [ resleft resright ] = result
      t1 = spawn do mmplus(A, Bleft, resleft) end
      t2 = spawn do mmplus(A, Bright, resright) end
      t1.wait(); t2.wait()
end
```



```
n => [ Aright Aleft ] = A
      [ Btop ; Bbottom ] = B
      (* IN ORDER *)
      mmplus(Aright, Btop, result)
      mmplus(Aleft, Bbottom, result)
end
```

```
end
```

# GTC RNG fragment

```
object rng() =  
index:Integer; array:Int64[0:100]  
  
uniform[type T]() : T;  
uniform[type T](T[:]) : ();  
uniform[type T](T[:,:]) : ();  
uniform[type T](T[:, :, :]) : ();  
  
uniform[Float]() : Float = do  
  if index >= |array| then rand_batch() end  
  bits = array[index] RIGHTSHIFT (47-23); index += 1  
  (Float.coerce(bits) + 0.5) * 2^-23  
end  
  
uniform[Double]() : Double = do  
  if index >= 100 then rand_batch() end  
  bits = array[index]; index += 1  
  (Double.coerce(bits) + 0.5) * 2^-47  
end  
  
uniformFill[type T](x:T[:]) : () =  
  for i <- rowMajor(x#) do x[i] = uniform[T]() end
```

# GTC RNG fragment

```
uniformFill[type T](x:T[:,:]):() =  
  for (i,j) <- rowMajor(x#) do x[i,j] := uniform[T]() end  
  
uniformFill[type T](x:T[:, :, :]):() =  
  for (i,j,k) <- rowMajor(x#) do x[i,j,k] := uniform[T]() end  
  
gaussianFill[type T](x:T[0:]):() = do  
  len = |x|  
  uniformFill(x)  
  for i <- x#, even i do  
    theta = pi (2 x[i] - 1)  
    z = SQRT(-2 log x[i+1])  
    x[i] := z cos theta  
    x[i+1] := z sin theta  
  end  
  if odd len then  
    z = uniform()  
    theta = pi (2 x[len-1] - 1)  
    x[len-1] := SQRT(-2 log z) cos theta  
  end  
end
```

# Implementation challenges

- Parsing!
- Dependent type inference
- Compilation at runtime
- Deriving communication from cache-oblivious algorithms
- Leaf compilation of recursive decomposition.
- Efficient transactional memory
- Very-large-and-parallel GC
- Library design
- Extensible (domain-specific) language.



# Fortress 0.62

Guy Steele

Jan-Willem Maessen

Eric Allen

David Chase

Victor Luchangco

Sam Tobin-Hochstadt

Sukyoung Ryu

Yossi Lev

Carl Eastlund

Joe Hallett

John Dias