



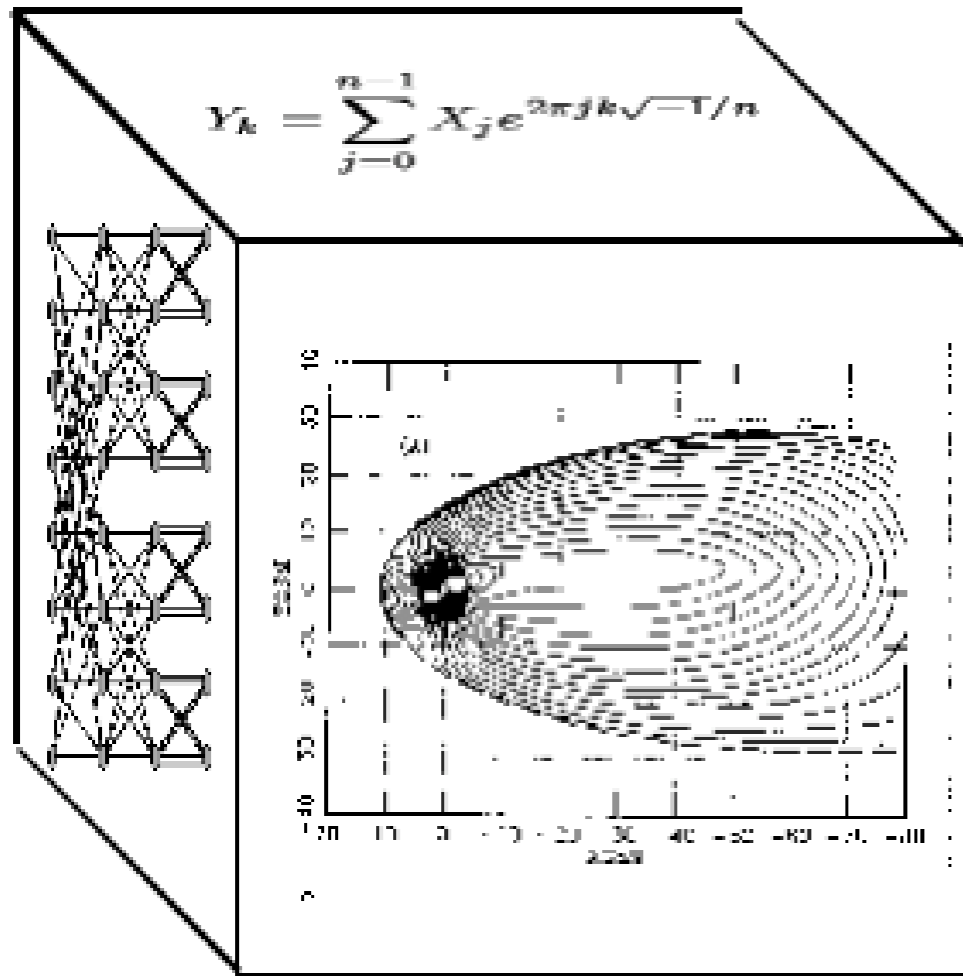
Fortress for Productive Computing

Jan-Willem Maessen
Sun Microsystems
JanWillem.Maessen@sun.com

PMUA, 2005-06-21

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

Goal: Science-centered computation



- Program structure should reflect the science
- Not FLOPS
- Not communication structure
- Fortress is not there yet
- Still a work in progress
- Listening to all input

Achieving Productivity

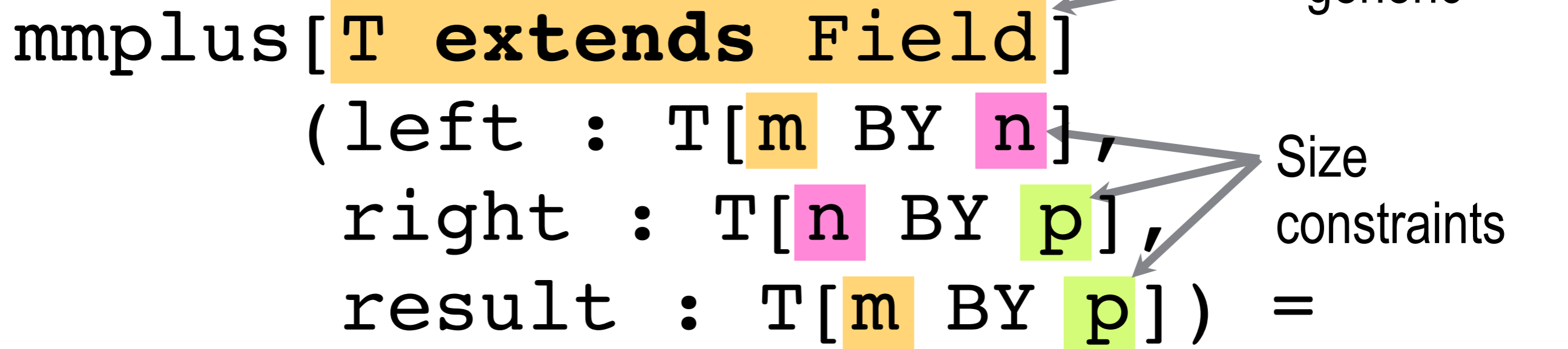
- Readability
- Conciseness
- Safety
- Programming in the large
- Parallelization with grace
- Simplicity of tool support
 - > Parser and AST included for tool builders
- Domain-specific syntactic extension

- Build a language that's useful for more than just HPC

Generics, constraints, formatting

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

Constrained generic
 Size constraints



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

Readable form

Readability through formatting

Unicode nickname

```

CircularConvolution[T extends Field]
  ( A:T[0:m, 0:n],
    P:T[down:up, left:right],
    B:out T[0:m, 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[0,m) × [0,n),
P : T[down,up) × [left,right),
B : out T[0,m) × [0,n)) =

```

Inclusive-exclusive ranges

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

Fortress: A Trait-Based Language

- Objects are described by *traits* rather than classes
- Collection of related functionality, with default implementations of everything
- Traits can be parametric

```
trait Collection[T extends Any] extends { Generator[T] }  
  insert(T) : Collection[T]  
  
  union(other: Collection[T]) = do  
    result := self  
    for i ← other do  
      result := result.insert(i)  
    end  
  result  
end  
end
```

Method defined
by implementing
objects

Default code for
derived method

Operator definition

Goal: principled overloading (only use + for addition!)

```
trait Equality[T extends Equality[T]]  
  equals(T):Boolean
```

Idiom for
self types



```
trait Ordering[T extends Ordering[T]]  
  extends Equality[T]  
  le(T):Boolean  
  gt(other) = not (self.le(other))  
  ge(other) = other.le(self)  
  lt(other) = not (self.ge(other))
```

```
opr =[T extends Equality[T]](x:T, y:T):T = x.eq(y)
```

```
opr ≤[T extends Ordering[T]](x:T, y:T):T = x.le(y)
```

```
opr <[T extends Ordering[T]](x:T, y:T):T = x.lt(y)
```

```
opr >[T extends Ordering[T]](x:T, y:T):T = x.gt(y)
```

```
opr ≥[T extends Ordering[T]](x:T, y:T):T = x.ge(y)
```

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



Machine Assumptions

- Programmers don't know the number of processors
- It will change on the fly
 - > A large parallel machine is always broken
 - > Temperature rise may reduce number of threads
 - > Machine load may drop; put extra compute to use!
- Need to expose vast amounts of parallelism
 - > 10K CPUs + 50 threads each + Slack for load balancing
= millions of simultaneous threads
 - > Fine-grained threading is a must
- We can create a shared address space
 - > But distant access may be slow

Parallelism Model

- Data parallel is just dandy
- The for loop is parallel by default
- Use recursive subdivision
 - > Scheduling selects appropriate granularity at run time
 - > Can adapt to changes in machine size
- Distributions make subdivision easy
 - > Loops, arrays subdivide according to a distribution
 - > Distributions contain locality information for scheduling
- Transactional memory simplifies synchronization
 - > Still need efficient bulk operators (eg prefix)

Data Parallelism

```
conjGrad[nat n, E extends Number]
  (A: E[n × n], x: E[n]): (E[n], E) = do
  cgitmax = 25
  z: E[n] = 0
  r: E[n] = x
  p: E[n] = r
  ρ: E = r·r
  for j ← sequential(0:cgitmax) do
    q = A p
    α = ρ / p·q
    z := z + α p
    r := r - α q
    ρ0 = ρ
    ρ := r·r
    β = ρ / ρ0
    p := r + β p
  end
  (z, ||x - A z||)
end
```

Ask for a sequential
loop if it is required



Transactional Memory

- Primitive synchronization mechanism in Fortress
- Code in an atomic block appears to execute in a single atomic step.

```
histogram[nat lo, nat hi, A extends Any]
    (bin: (A)→Int, a: A[:, :]): Int[lo:hi] =
do hist : Array(lo:hi) = 0
    for i, j ← a# do
        atomic do
            hist[bin(a[i, j])] += 1
        end
    end
    hist
end
```

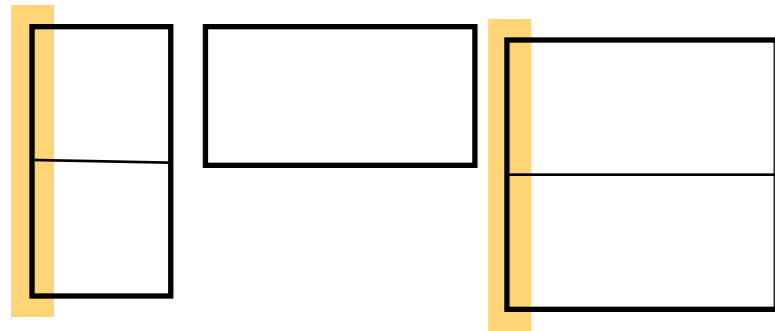
We update the histogram atomically so that parallel updates are not lost.

Cache oblivious matrix multiplication

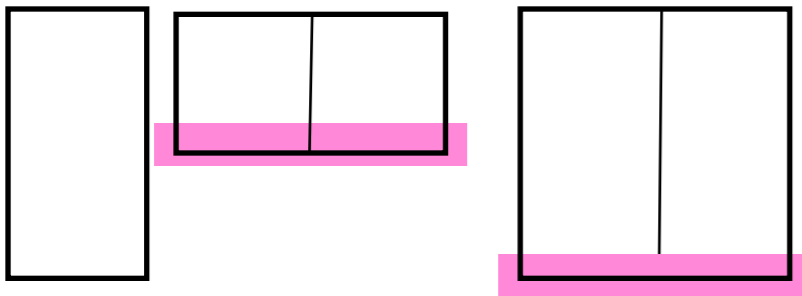
```
mmplus[T extends Field] (A: T[m × n],
                          B: T[n × p],
                          result: T[m × p]):() =
```

case largest of

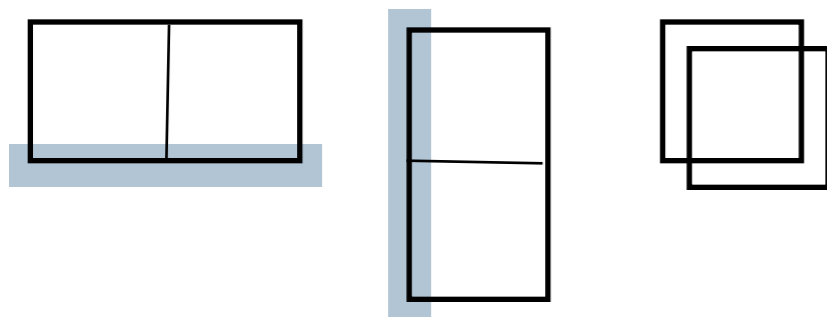
```
1 ⇒ result[0,0] += A[0,0]·B[0,0]
  end
```



```
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
```



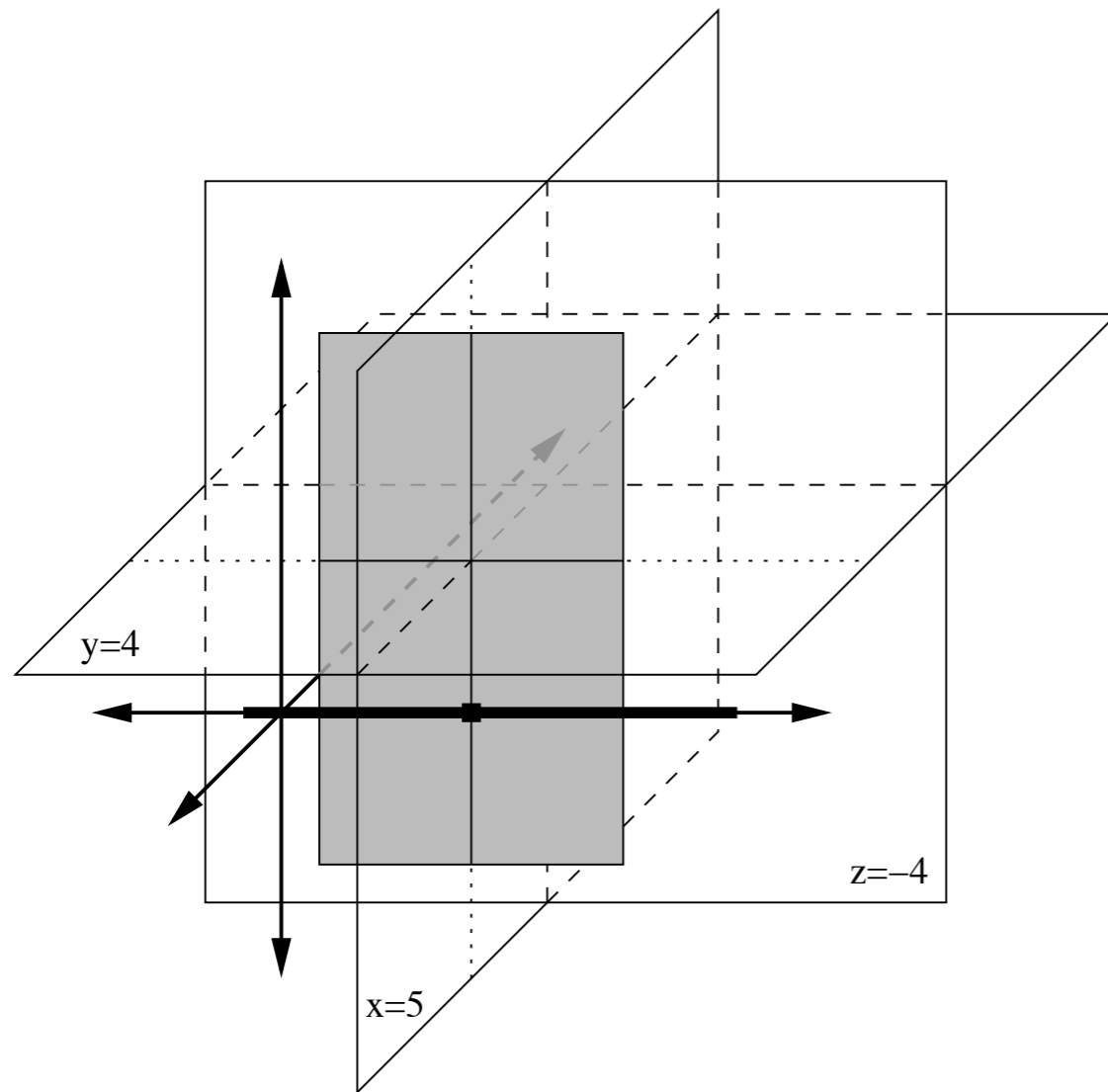
```
p ⇒ [ Bleft Bright ] = B
     [ resleft resright ] = result
     t1 = spawn do mmplus(A, Bleft, resleft) end
     t2 = spawn do mmplus(A, Bright, resright) end
     t1.wait(); t2.wait()
  end
```



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

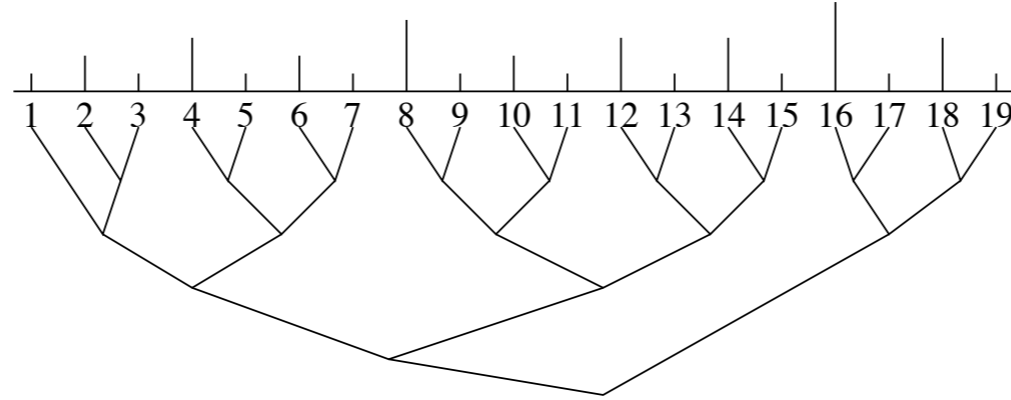
end

Distributions describe subdivision



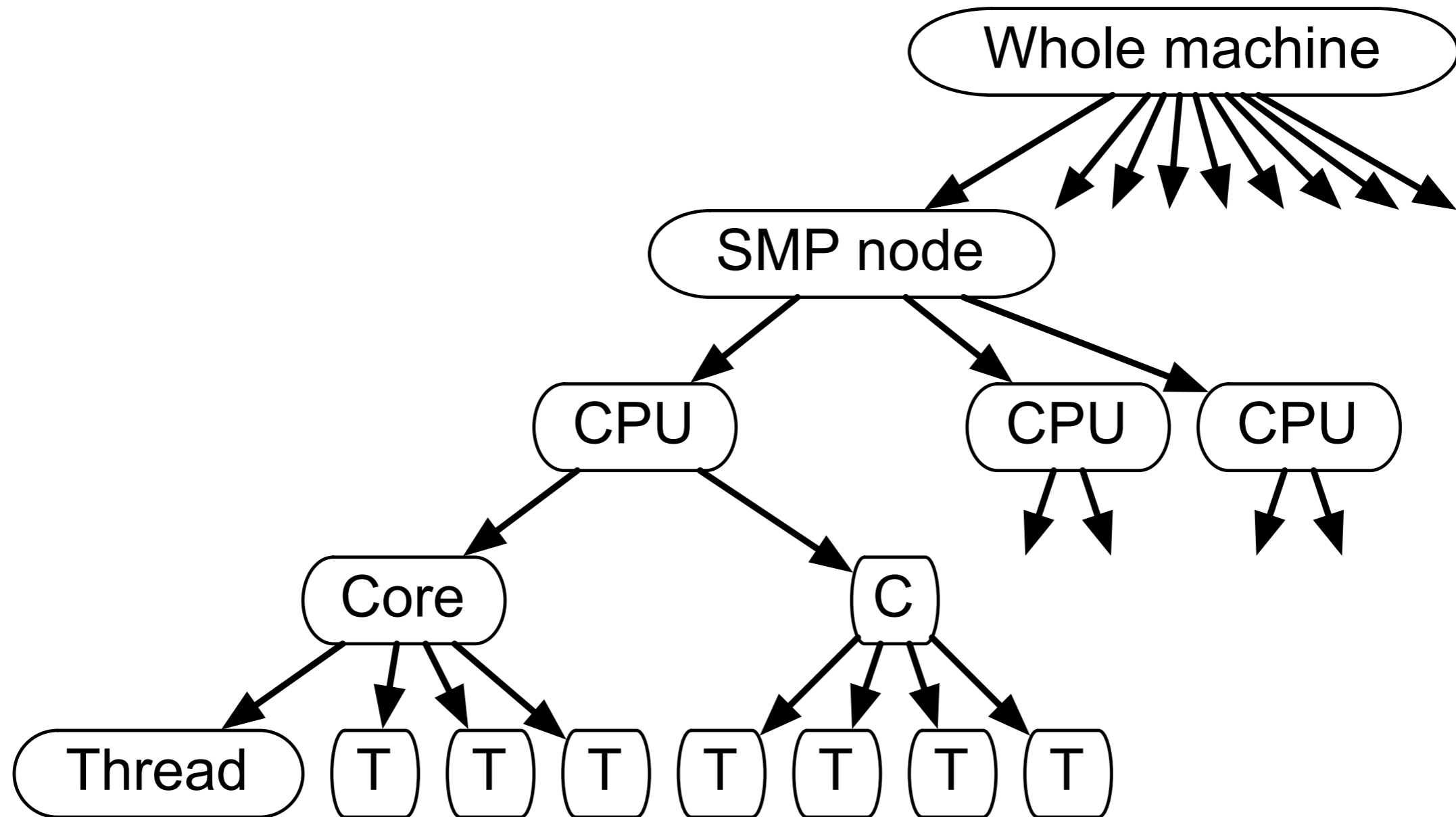
- Infinite-dimensional space
 - Cut by sequence of planes
 - Each orthogonal to an axis
 - Each oriented
-
- Arrays and iteration spaces reside in lowest dimensions of this space

The Ruler: A Simple Subdivision



- Distribution gives rise to a binary tree
- Tree describes subdivision of particular subspace
- Tree need not be balanced
 - > But a regular structure is a must
- Nodes of tree annotated with locality information
 - > Schedule via locality-guided work stealing

Machine Abstraction: A Tree



Current State of Play

- Language still in a state of flux
- Proving type soundness
- Taming the extensible parser
- Building prototype implementation
- Validating distributions



Fortress for Productive Computing

[http://research.sun.com/projects/plrg/
fortress0618.pdf](http://research.sun.com/projects/plrg/fortress0618.pdf)

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

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

GTC RNG fragment

```
object rng() = index:Integer:=0; array:Int64[0:100]
```

```
uniform[T extends Numeric[T]]():T;
```

```
uniform[Float]():Float = do  
  if index ≥ |array| then rand_batch() end  
  bits = array[index] >> (47-23); index += 1  
  (Float.coerce(bits) + 0.5) · 2-23  
end
```

```
uniform[Double]():Double = do  
  if index ≥ |array| then rand_batch() end  
  bits = array[index]; index += 1  
  (Double.coerce(bits) + 0.5) · 2-47  
end
```

```
uniformFill[T extends Numeric[T]](x:T[:]):() =  
  for i ← rowMajor(x#) do  
    x[i] := uniform[T]()  
end
```

GTC RNG fragment

```
uniformFill[T extends Numeric[T]](x:T[: × :]):() =
  for (i,j) ← rowMajor(x#) do x[i,j] := uniform[T]() end
```

```
uniformFill[T extends Numeric[T]](x:T[: × : × :]):() =
  for (i,j,k) ← rowMajor(x#) do x[i,j,k] := uniform[T]() end
```

```
gaussianFill[T extends Numeric[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
```