Flavour of Languages

Ashish Mahabal
aam@astro.caltech.edu

Caltech, 7 Apr 2011
Ay199/Bi 199b

Quick survey

- C?
- Shell?
- Perl?
- Python?
- HTML?
- SQL?

The language you use influences how you think (about problems)

- Types of languages
- Features of languages
- Internal issues
- Extendibility, domain specific languages
- Available help, practical issues
- Architecture/compilation etc.
- Wider issues(?)
- Exercise

How to shoot yourself in the foot

(http://www-users.cs.york.ac.uk/~susan/joke/foot.htm)

- C: You shoot yourself in the foot
- C++: You accidently create a dozen instances of yourself and shoot them all in the foot. Providing emergency medical care is impossible since you can't tell which are bitwise copies and which are just pointing at others and saying, "That's me over there."
- FORTRAN: You shoot yourself in each toe, iteratively, until you run out of toes, then you read in the next foot and repeat. If you run out of bullets, you continue anyway because you have no exception handling ability.

If languages were religions

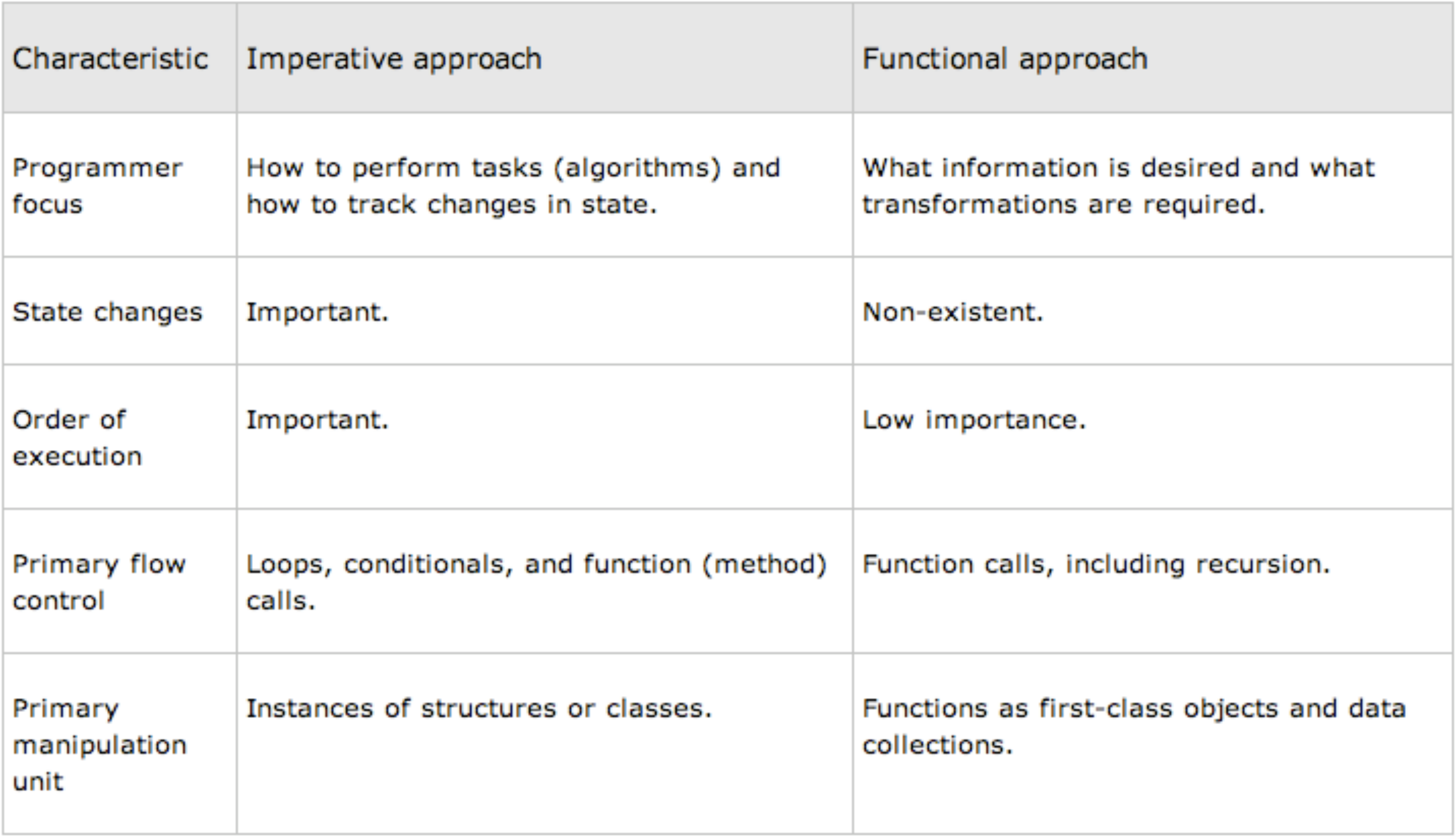
(http://www.aegisub.net/2008/12/if-programming-languages-were-religions.html)

- C would be Judaism - it's old and restrictive, but most of the world is familiar with its laws and respects them. …
- C++ would be Islam - It takes C and not only keeps all its laws, but adds a very complex new set of laws on top of it. …
- Lisp would be Zen Buddhism
- Perl would be Voodoo
- Python would be Humanism
- ….

Types of languages

(its difficult to put a single label actually)

- Imperative (e.g. C, Java, …)
- Functional (e.g. LISP, Haskell, perl, python, …)
- Logical (e.g. prolog)
- ...
- Formatting/markup (e.g. HTML, XML, KML, …)
- ...
- Database (e.g. SQL and its flavours)
- ...
- Shells (e.g. tcsh, bash, ksh, …)



Characteristic	Imperative approach	Functional approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.	What information is desired and what transformations are required.
State changes	Important.	Non-existent.
Order of execution	Important.	Low importance.
Primary flow control	Loops, conditionals, and function (method) calls.	Function calls, including recursion.
Primary manipulation unit	Instances of structures or classes.	Functions as first-class objects and data collections.

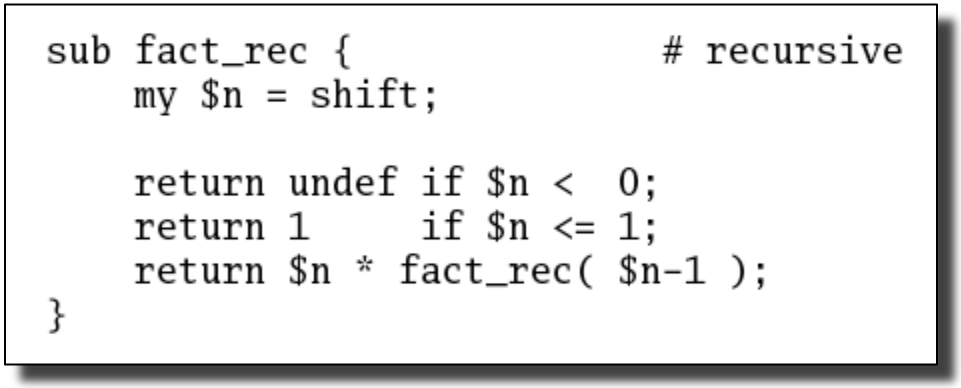
logic programming contributes **non-determinism, inversion and partial data structures**, whereas functional programming contributes **efficient evaluation and infinite data structures**.
(http://web.cecs.pdx.edu/~antoy/research/flp/index.html)

Imperative(C, Java)

- Computation as statements that change program state
    - i = 0;
    - i++;
        - n=10;
        - j=1;
        - for(i=2;i<=n;++i){j*=i;}

Procedural (perl, python)

- Method of executing imperative language programs (imperative + subprograms)

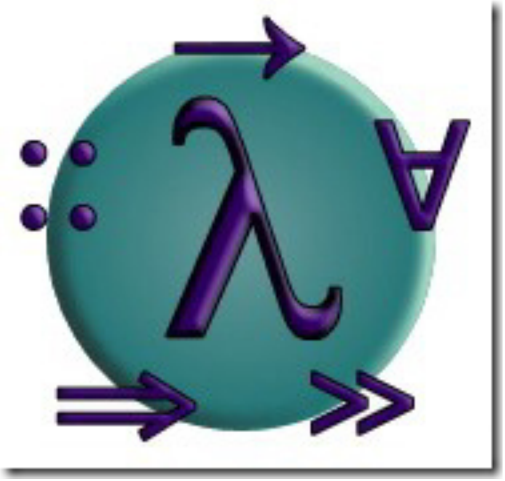


```
sub fact_rec {                    # recursive
    my $n = shift;

    return undef if $n <  0;
    return 1     if $n <= 1;
    return $n * fact_rec( $n-1 );
}
```

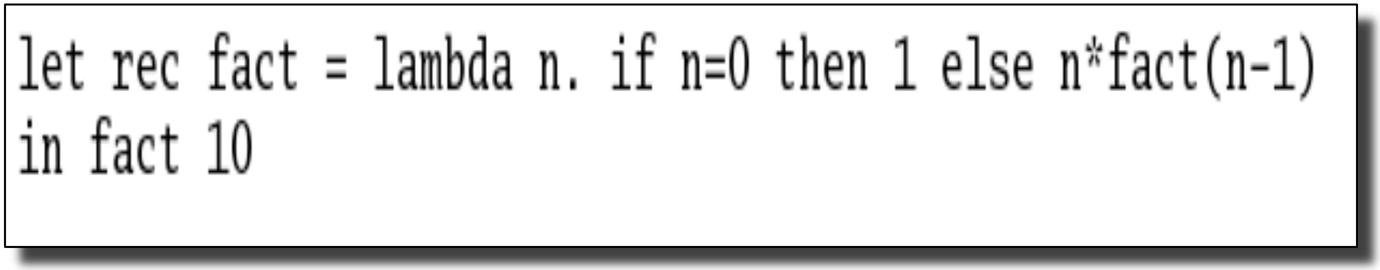
(Could have issues in list mode).

Functional (Haskell, LISP)

- computation as the evaluation of mathematical functions. No state.
- Effected through lambda calculus, composition of functions



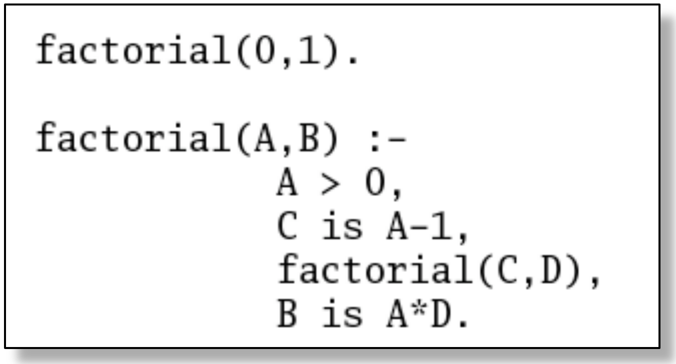
Devtopics.com



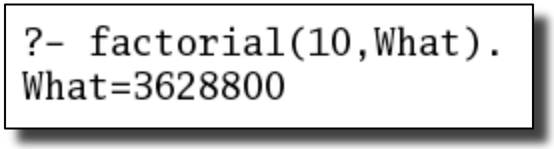
```
let rec fact = lambda n. if n=0 then 1 else n*fact(n-1)
in fact 10
```

Logical (Prolog)

- Define “what” is to be computed rather than “how” (declarative: properties of correct answers)



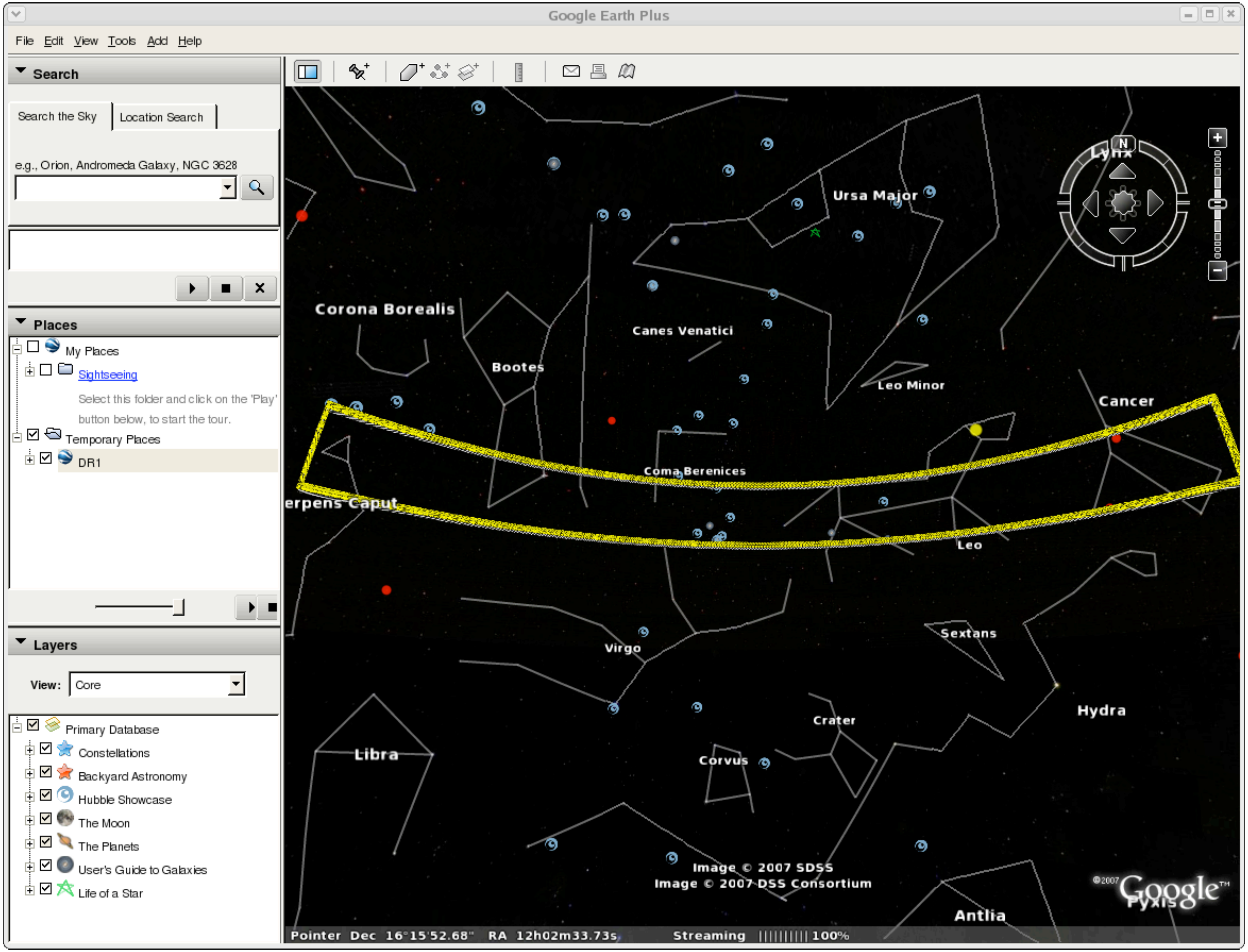
```
factorial(0,1).

factorial(A,B) :-
           A > 0,
           C is A-1,
           factorial(C,D),
           B is A*D.
```



```
?- factorial(10,What).
What=3628800
```

markup/database

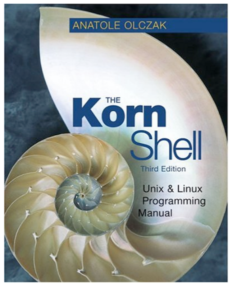
- SGML/HTML/XML – stylized rendering (XML to be covered in other talks)
  - Tags used for formatting
  - *<A HREF=SomeLink>SomeText</A>*
  - *<mytag>lalala</mytag>*
- KML – Keyhole Markup Language
  - Convert points for Google Earth/sky locations
- SQLs e.g. my, ms, pg, ... (SQL and databases will also be covered in detail in other talks)
  - For talking to databases
  - *Select * from TableX where Y=Z*



Select this folder and click on the 'Play' button below, to start the tour.

Pointer Dec 16°15'52.68" RA 12h02m33.73s Streaming |||||||||| 100%

shells



- bsh/bash/csh/ksh/tcsh ... are languages in their own right
  - awk/sed/grep
  - History *“!mv; !scp:p; ^my^ny”*
  - Loops
    - *foreach f (*.jpg)*
    - *convert $f $f:r.png*
    - *end*
  - Redirections *“(myprog < myin > myout) >& myerr &”*
  - Scripts *“at now + 24 hours < foo.csh”*

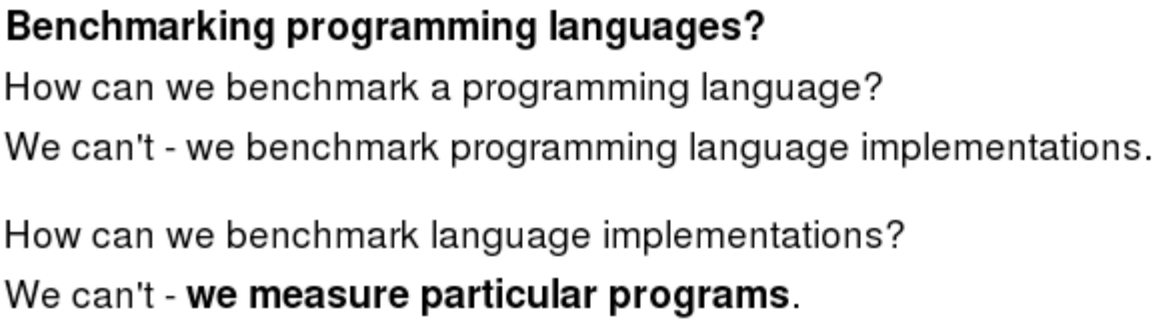
For Matlab buffs

http://www.datatool.com/downloads/matlab_style_guidelines.pdf

Optimization

The Computer Language Benchmarks Game

http://shootout.alioth.debian.org



Benchmarking programming languages?

How can we benchmark a programming language?

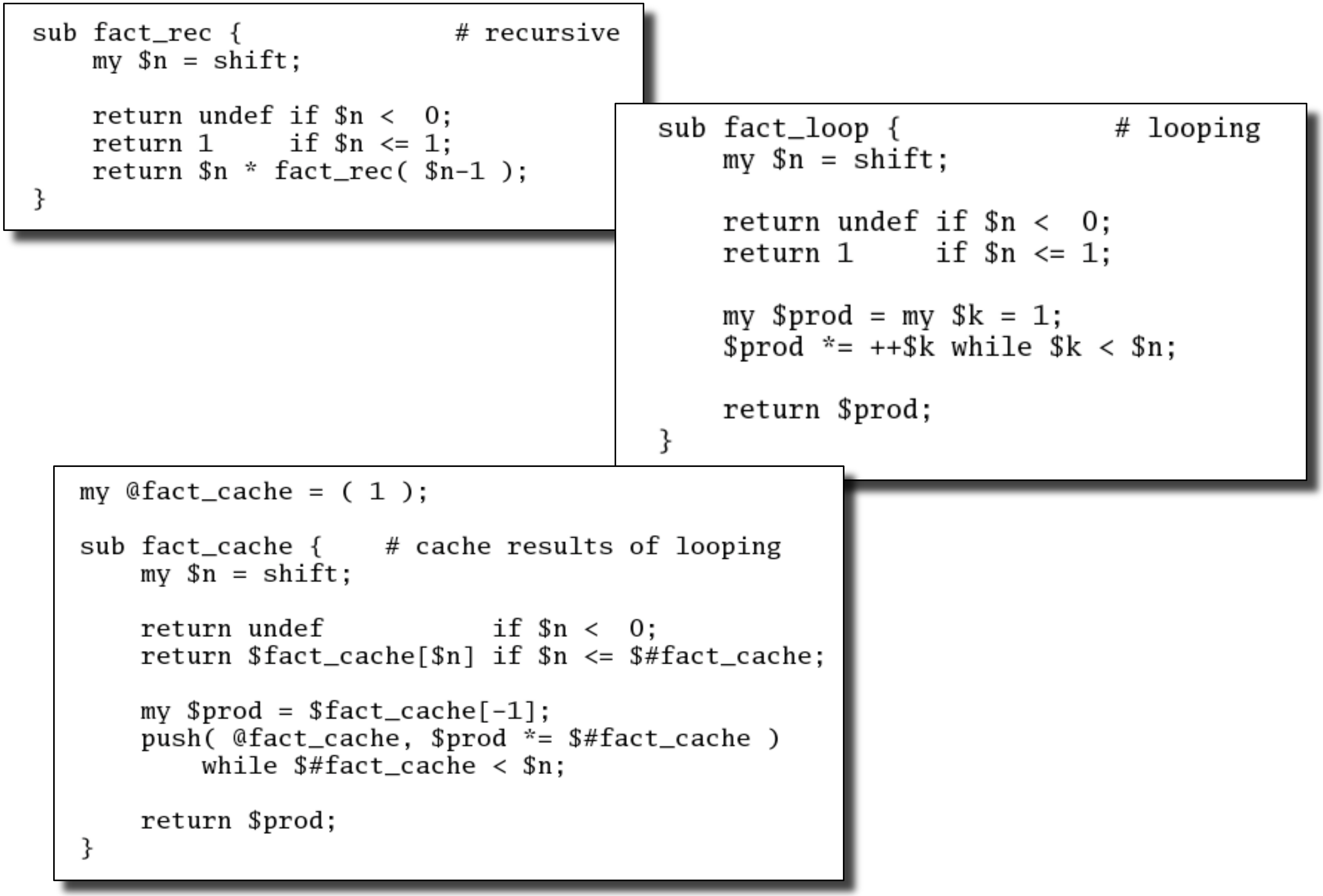
We can't - we benchmark programming language implementations.

How can we benchmark language implementations?

We can't - **we measure particular programs**.

Full CPU Time	1
Memory Use	5
GZip Bytes	0

benchmark	weight
binary-trees	5
chameneos	0
cheap-concurrency	1
fannkuch	1
fasta	5
k-nucleotide	1
mandelbrot	1
meteor-contest	5
n-body	1
nsieve	1
nsieve-bits	1
partial-sums	1



```
sub fact_rec {                # recursive
    my $n = shift;

    return undef if $n <  0;
    return 1     if $n <= 1;
    return $n * fact_rec( $n-1 );
}
```

```
sub fact_loop {               # looping
    my $n = shift;

    return undef if $n <  0;
    return 1     if $n <= 1;

    my $prod = my $k = 1;
    $prod *= ++$k while $k < $n;

    return $prod;
}
```

```
my @fact_cache = ( 1 );

sub fact_cache {    # cache results of looping
    my $n = shift;

    return undef           if $n <  0;
    return $fact_cache[$n] if $n <= $#fact_cache;

    my $prod = $fact_cache[-1];
    push( @fact_cache, $prod *= $#fact_cache )
        while $#fact_cache < $n;

    return $prod;
}
```

And then there is built-in memoizing

Features of Languages

- strong/weak/no typing; datatypes
- safe/unsafe typing
- dynamic/static datatype conversions
- side effects/monads
- concurrency
- distributedness

strong/weak typing

- *#include <stdio.h> main() {int fill; fill=42; printf(“%s\n”,fiil);}*

strong/weak typing

- *#include <stdio.h>   main() {int fill; fill=42; printf(“%s\n”,fiil);}*
  - This will not compile for at least two reasons:
    - fiil (mistyped) is not declared
    - Even if that is corrected, it is not a string

strong/weak typing

- `#include <stdio.h>                              main() {int fill; fill=42; printf(“%s\n”,fiil);}`
  - This will not compile for at least two reasons:
    - fiil (mistyped) is not declared
    - Even if that is corrected, it is not a string
- `#!/usr/bin/perl $fill=42;printf(“%s\n”,$fiil);`

strong/weak typing

- *#include <stdio.h> main() {int fill; fill=42; printf("%s\n",fiil);}*
  - This will not compile for at least two reasons:
    - fiil (mistyped) is not declared
    - Even if that is corrected, it is not a string
- *#!/usr/bin/perl $fill=42;printf("%s\n",$fiil);*
    - This also fails, but silently. No error is announced
    - Change fiil to fill (leaving it as %s) and you get the correct result (by coincidence)

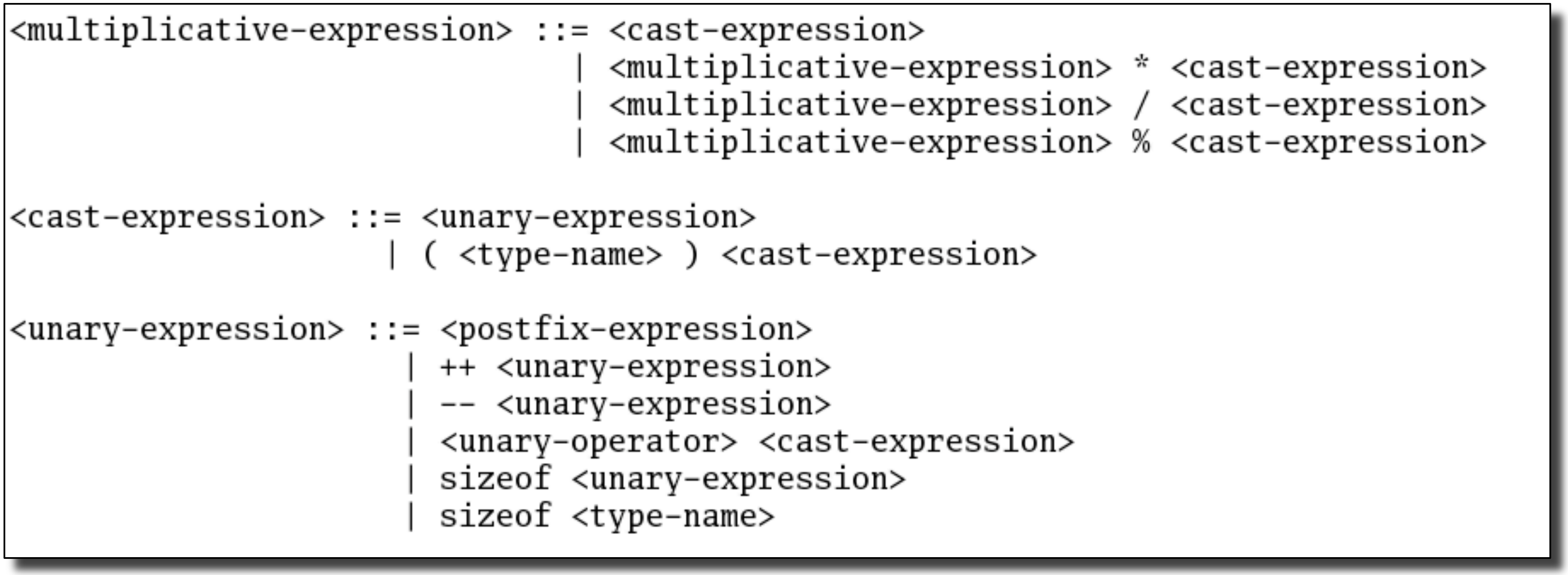
- *#!/usr/bin/perl -w*
- *use strict;*

A language is only as rigid or flexible as your understanding of it.

Grammers: (Extended) Bachus-Nour form

::= < > " " [ ] | { }

Partial grammer for C



```
<multiplicative-expression> ::= <cast-expression>
                              | <multiplicative-expression> * <cast-expression>
                              | <multiplicative-expression> / <cast-expression>
                              | <multiplicative-expression> % <cast-expression>

<cast-expression> ::= <unary-expression>
                    | ( <type-name> ) <cast-expression>

<unary-expression> ::= <postfix-expression>
                     | ++ <unary-expression>
                     | -- <unary-expression>
                     | <unary-operator> <cast-expression>
                     | sizeof <unary-expression>
                     | sizeof <type-name>
```

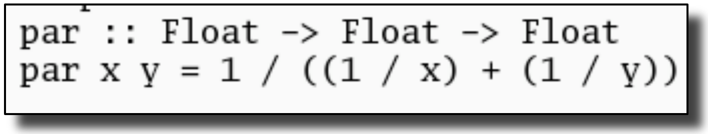
Extendibility

- With other languages
  - Perl through C
  - C through perl
- Packages for particular domains and their extensibility (e.g. matlab/iraf/idl)
  - Domain specific core functionality
  - Can be extended further using packages

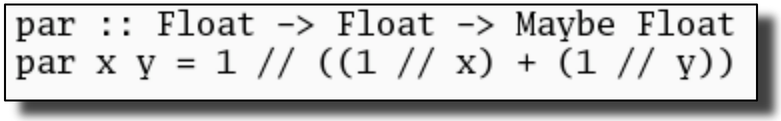
- Domain specific languages
  - Define terms/keywords close to the domain
  - Overload terms in domain appropriate way
    - *select RA,Dec from PQ where mag > 15 join radio > 1Jy*

Other esoteric sounding but important stuff

- syntactic sugar
  - $a[i]$ rather than $*(a+i)$
  - $a[i][j]$ rather than $*(*(a+i)+j)$
- side effects/monads



```
par :: Float -> Float -> Float
par x y = 1 / ((1 / x) + (1 / y))
```

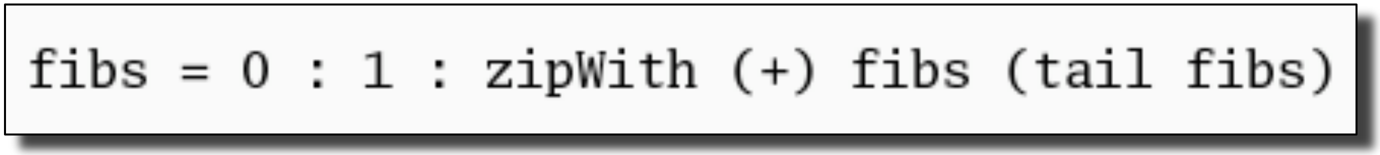


```
par :: Float -> Float -> Maybe Float
par x y = 1 // ((1 // x) + (1 // y))
```

Avoid the pitfall of division by 0 by returning a “maybe” monad of value “nothing”

- Lazy evaluation (delayed until needed)
  - x=f(y) will remain as is until x is needed
  - Possible to define infinite lists
  - Control structure: *a==b?c:d*

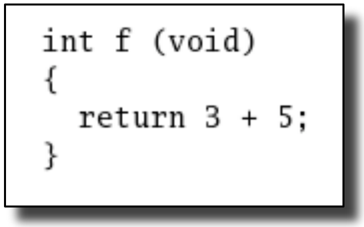
•Haskell’s implementation of Fibonacci numbers



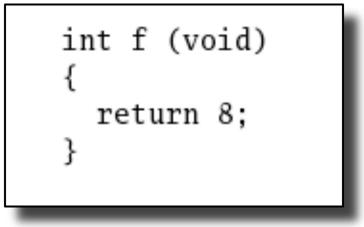
```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

•constant folding/argumentless functions

(evaluating constants at compile time)



```
int f (void)
{
  return 3 + 5;
}
```

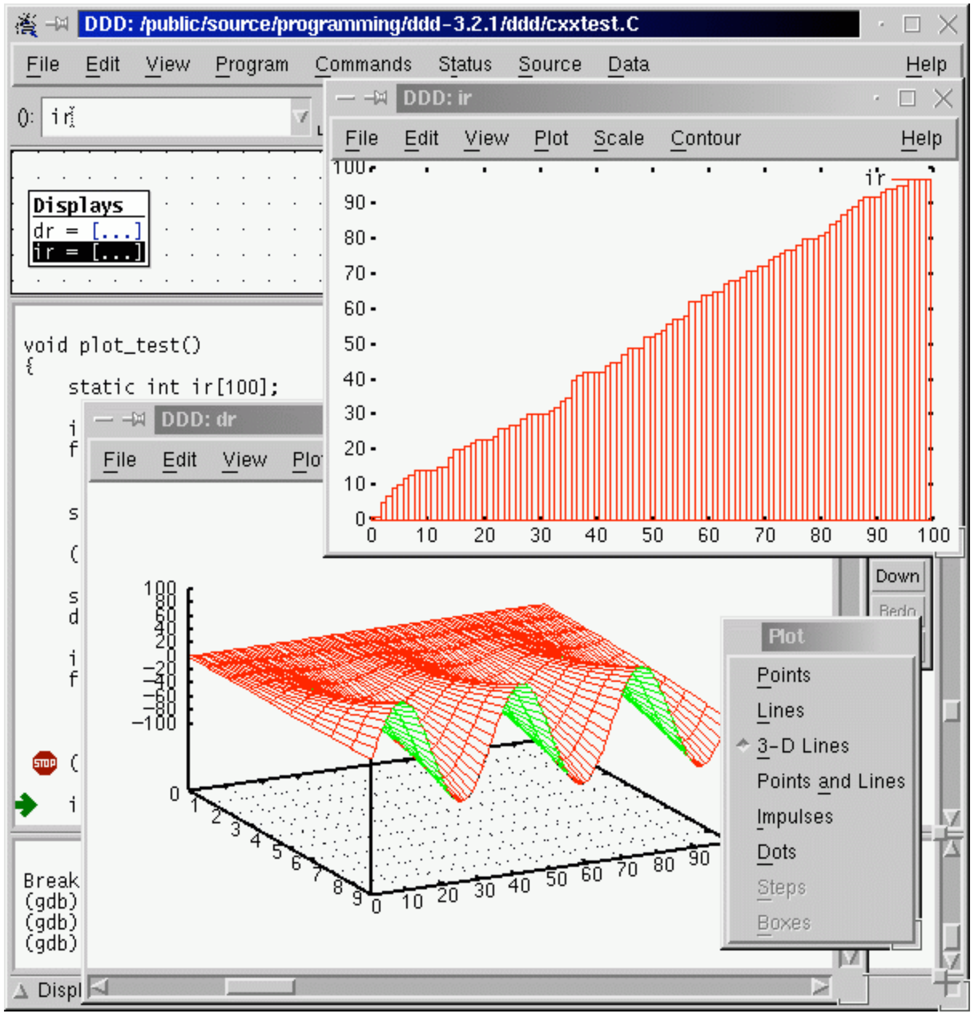


```
int f (void)
{
  return 8;
}
```

Help Available

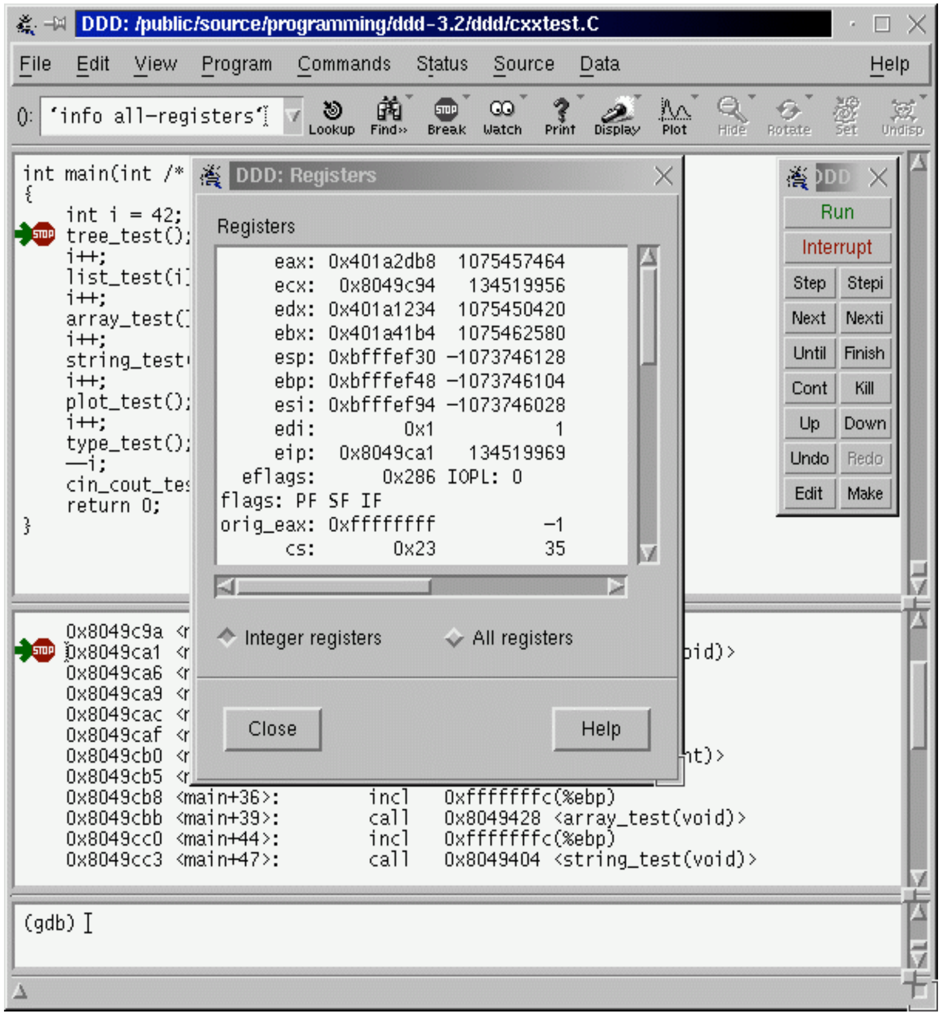
- debugging tools
  - Internal debuggers
  - External/graphical debuggers
    - *perl –c* checks syntax
    - *perl –d* default die handler
    - *ddd* debugger (works with most language debuggers)
- Macro editing modes
  - Emacs, vim (autotab, headers, brace matching)

ddd



```
void plot_test()
{
    static int ir[100];

    i
    f

    s

    (

    s
    d

    i
    f

    (

    i
```

```
Break
(gdb)
(gdb)
(gdb)
```



```
int main(int /*
{
    int i = 42;
    tree_test();
    i++;
    list_test(i]
    i++;
    array_test(]
    i++;
    string_test(
    i++;
    plot_test();
    i++;
    type_test();
    --i;
    cin_cout_tes
    return 0;
}
```

```
      eax: 0x401a2db8  1075457464
      ecx:  0x8049c94   134519956
      edx: 0x401a1234  1075450420
      ebx: 0x401a41b4  1075462580
      esp: 0xbfffef30 -1073746128
      ebp: 0xbfffef48 -1073746104
      esi: 0xbfffef94 -1073746028
      edi:        0x1           1
      eip:  0x8049ca1   134519969
   eflags:      0x286 IOPL: 0
flags: PF SF IF
orig_eax: 0xffffffff          -1
       cs:       0x23          35
```

```
0x8049c9a <r
0x8049ca1 <r                                              oid)>
0x8049ca6 <r
0x8049ca9 <r
0x8049cac <r
0x8049caf <r
0x8049cb0 <r                                              nt)>
0x8049cb5 <r
0x8049cb8 <main+36>:      incl   0xfffffffc(%ebp)
0x8049cbb <main+39>:      call   0x8049428 <array_test(void)>
0x8049cc0 <main+44>:      incl   0xfffffffc(%ebp)
0x8049cc3 <main+47>:      call   0x8049404 <string_test(void)>
```

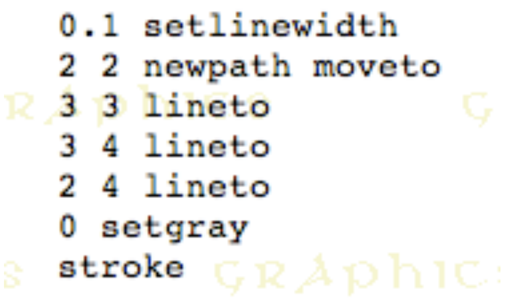
```
(gdb)
```

Practical Issues

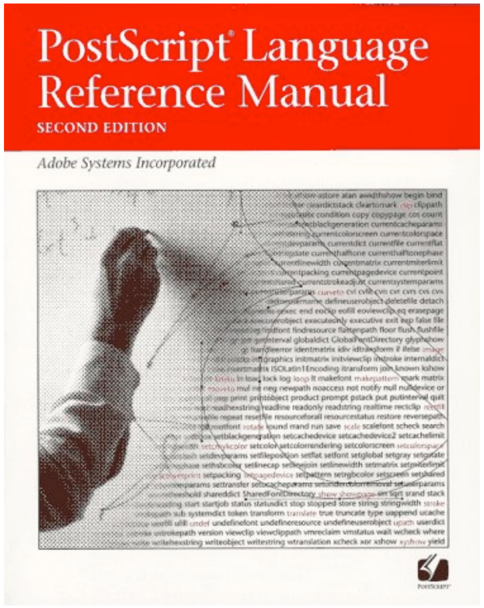
- OS support
  - *perl/c* supported on practically all platforms
- ease of learning (how to shoot your foot ...)
  - Functional/logical may seem non-intutive initially
  - So do java and C++
- readability across teams
  - Structure of syntax e.g. tabs in python
- Speed, scalability, reusability

Wider issues

- We have scratched only the surface
  - Did not even mention entities like
    - Postscript
    - Tcl
  - Text processing
- Non-Von Neumann computers



```
0.1 setlinewidth
2 2 newpath moveto
3 3 lineto
3 4 lineto
2 4 lineto
0 setgray
stroke
```



PostScript® Language Reference Manual
SECOND EDITION
Adobe Systems Incorporated

Larry Wall in ‘State of the Onion’ (2006)

(http://www.perl.com/pub/a/2007/12/06/soto-11.html)

- Early/late binding
- Single/multiple dispatch
- Eager/lazy typology
- Limited/rich structures
- Symbolic/wordy
- Immutable/mutable classes
- Scopes (various kinds)

Perligata (Damian Conway)

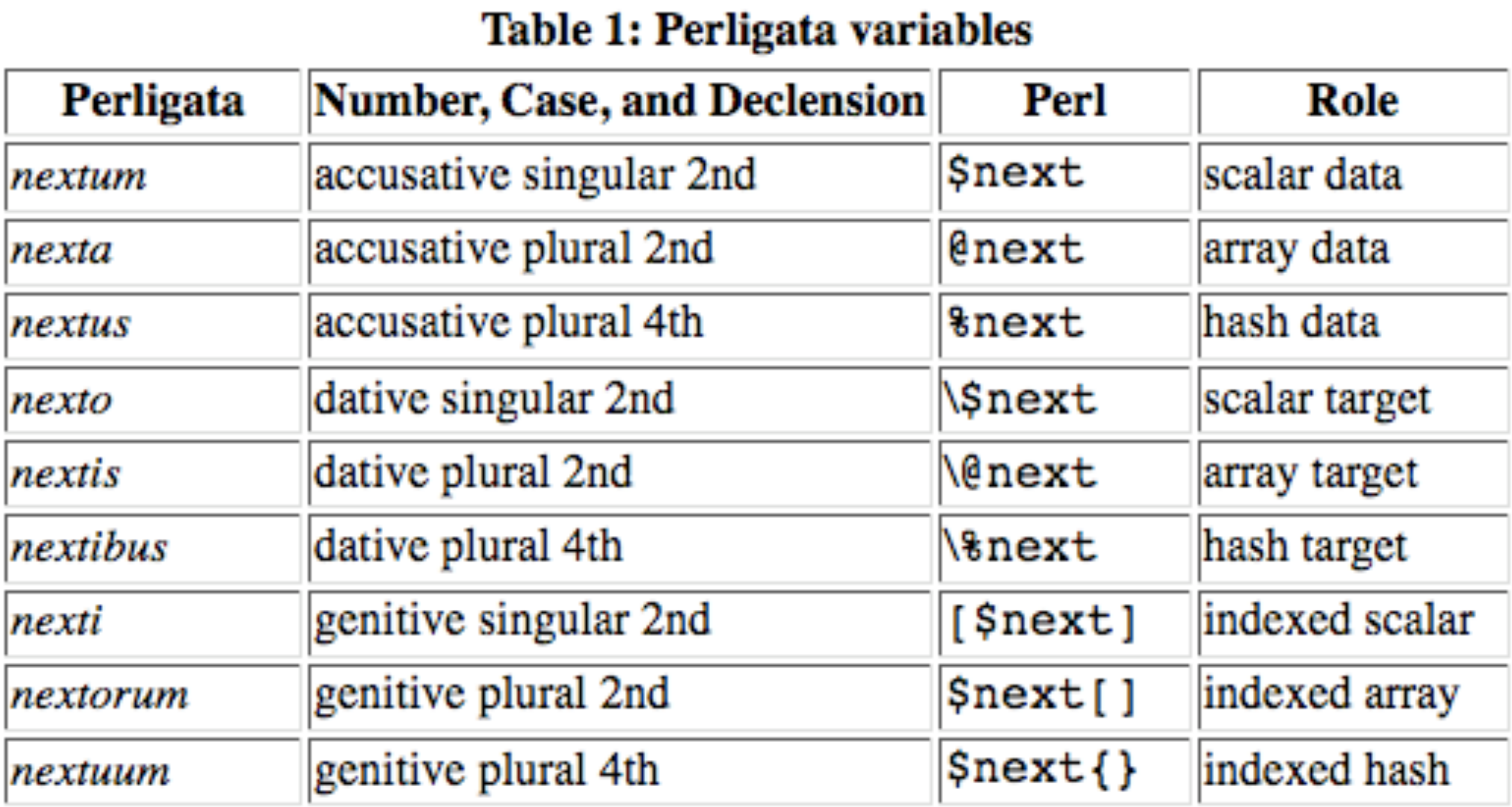
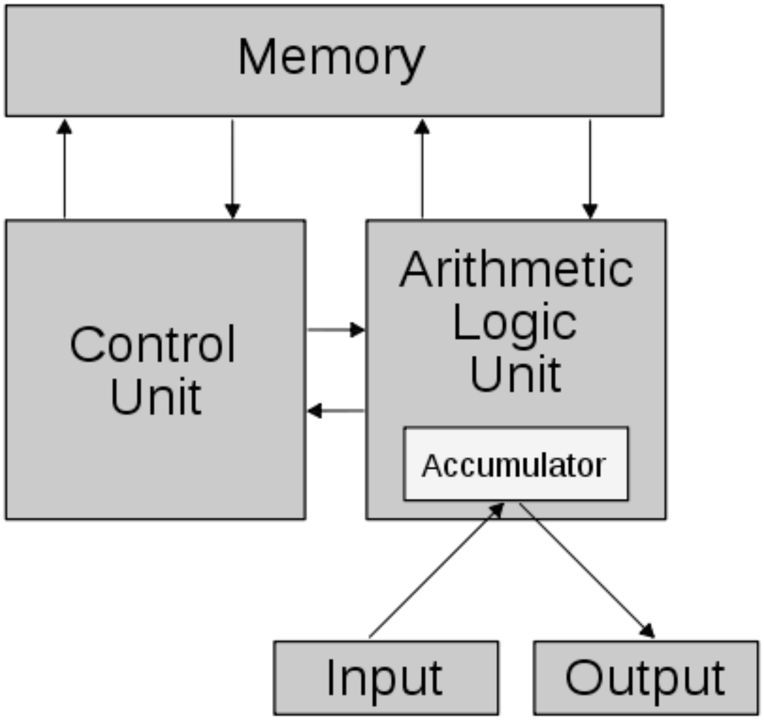


Table 1: Perligata variables

Perligata	Number, Case, and Declension	Perl	Role
nextum	accusative singular 2nd	`$next`	scalar data
nexta	accusative plural 2nd	`@next`	array data
nextus	accusative plural 4th	`%next`	hash data
nexto	dative singular 2nd	`\$next`	scalar target
nextis	dative plural 2nd	`\@next`	array target
nextibus	dative plural 4th	`\%next`	hash target
nexti	genitive singular 2nd	`[$next]`	indexed scalar
nextorum	genitive plural 2nd	`$next[]`	indexed array
nextuum	genitive plural 4th	`$next{}`	indexed hash

Von Neumann architecture

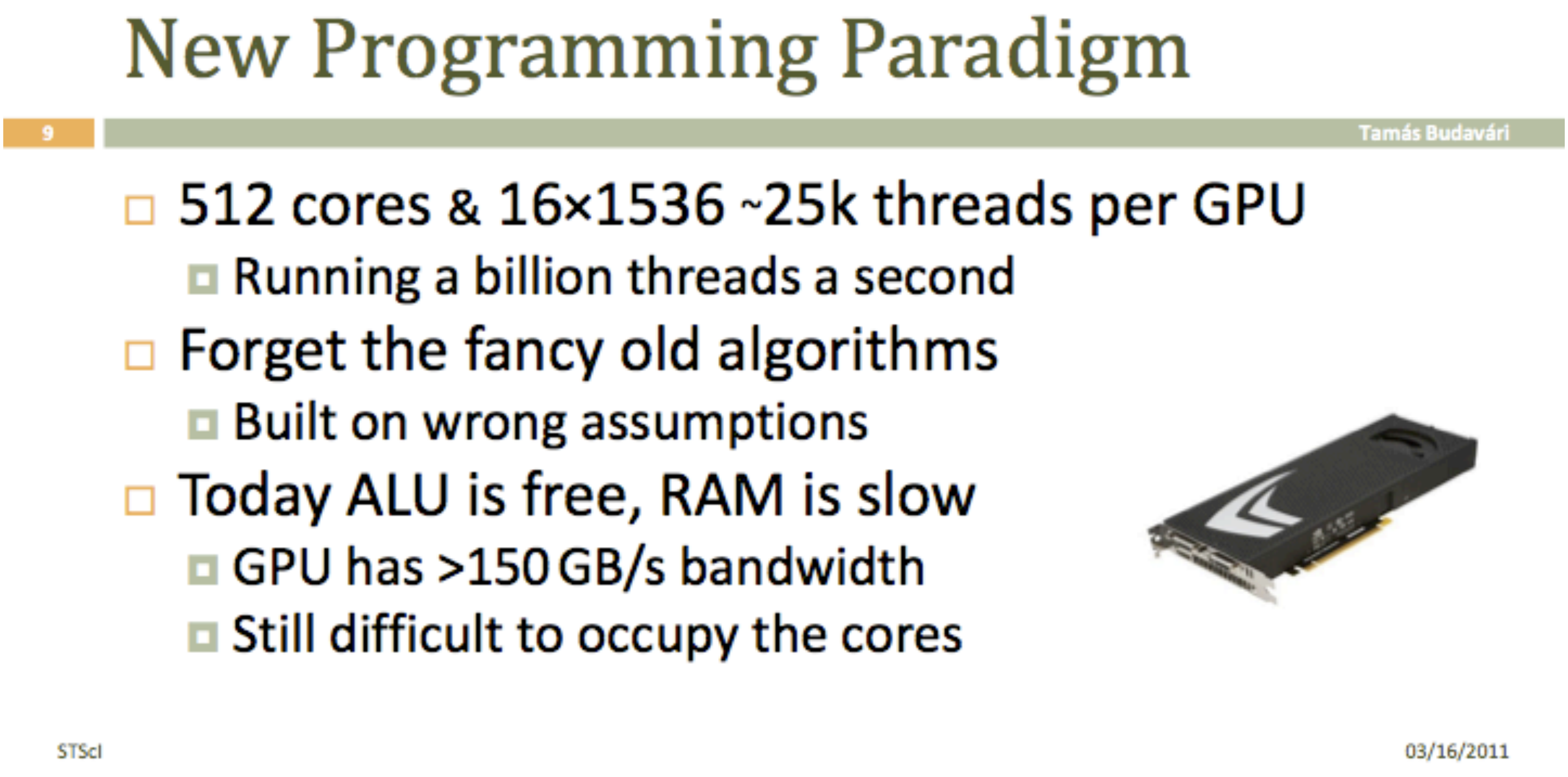
- instructions and data are distinguished only implicitly through usage
- memory is a single memory, sequentially addressed
- memory is one-dimensional
- meaning of the data is not stored with it
- Things looking better with Virtual machines and multi-core processors



Things we have left out

- Interpreters/compilers and the vagueness in between
- memory management
- garbage collection
- bytecode
- virtual machines
- Many core (parallelism)

Slides from Budavari



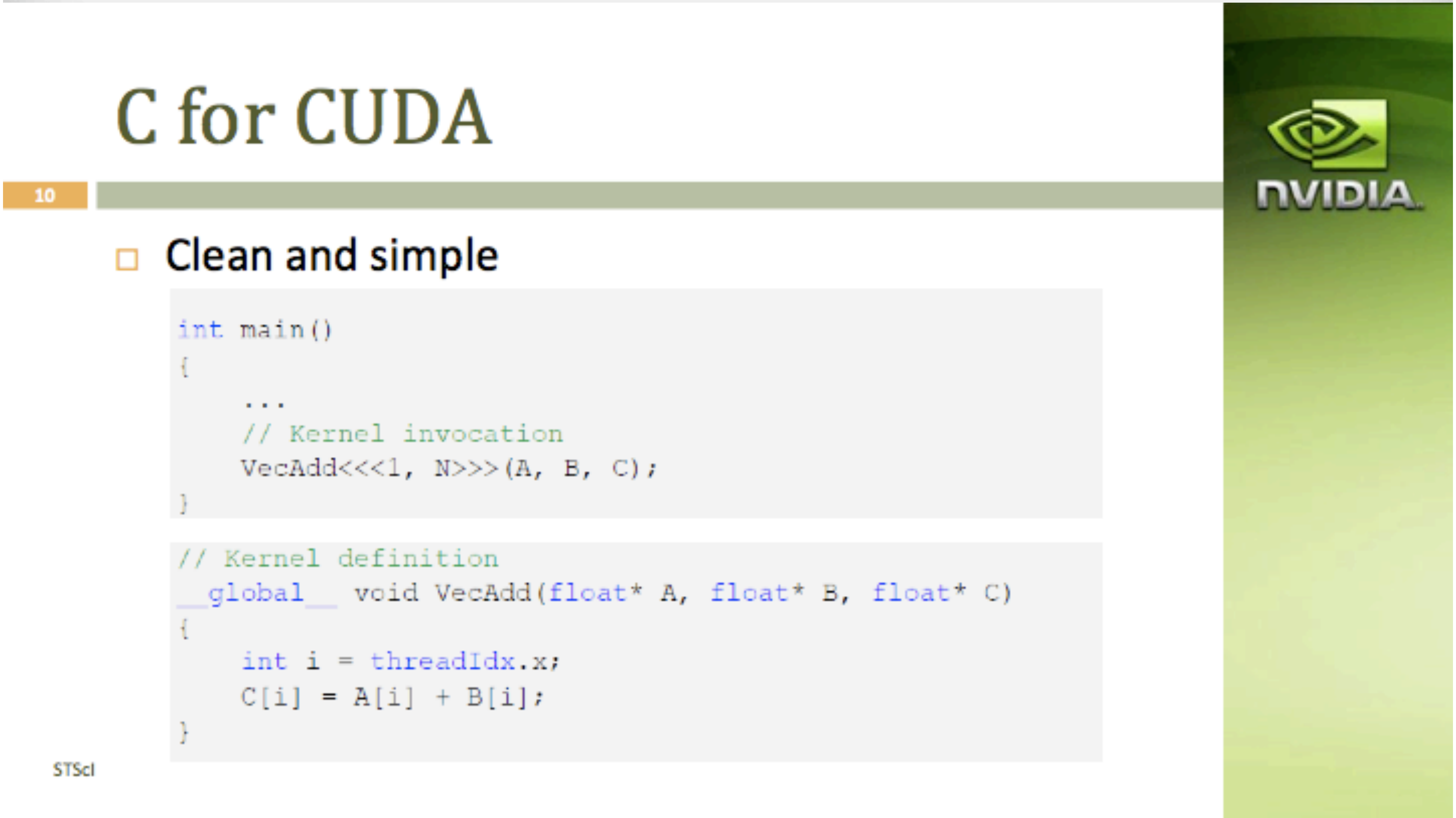
New Programming Paradigm

9 Tamás Budavári

- 512 cores & 16×1536 ~25k threads per GPU
  - Running a billion threads a second
- Forget the fancy old algorithms
  - Built on wrong assumptions
- Today ALU is free, RAM is slow
  - GPU has >150 GB/s bandwidth
  - Still difficult to occupy the cores

STScI 03/16/2011

Slides from Budavari



C for CUDA

10

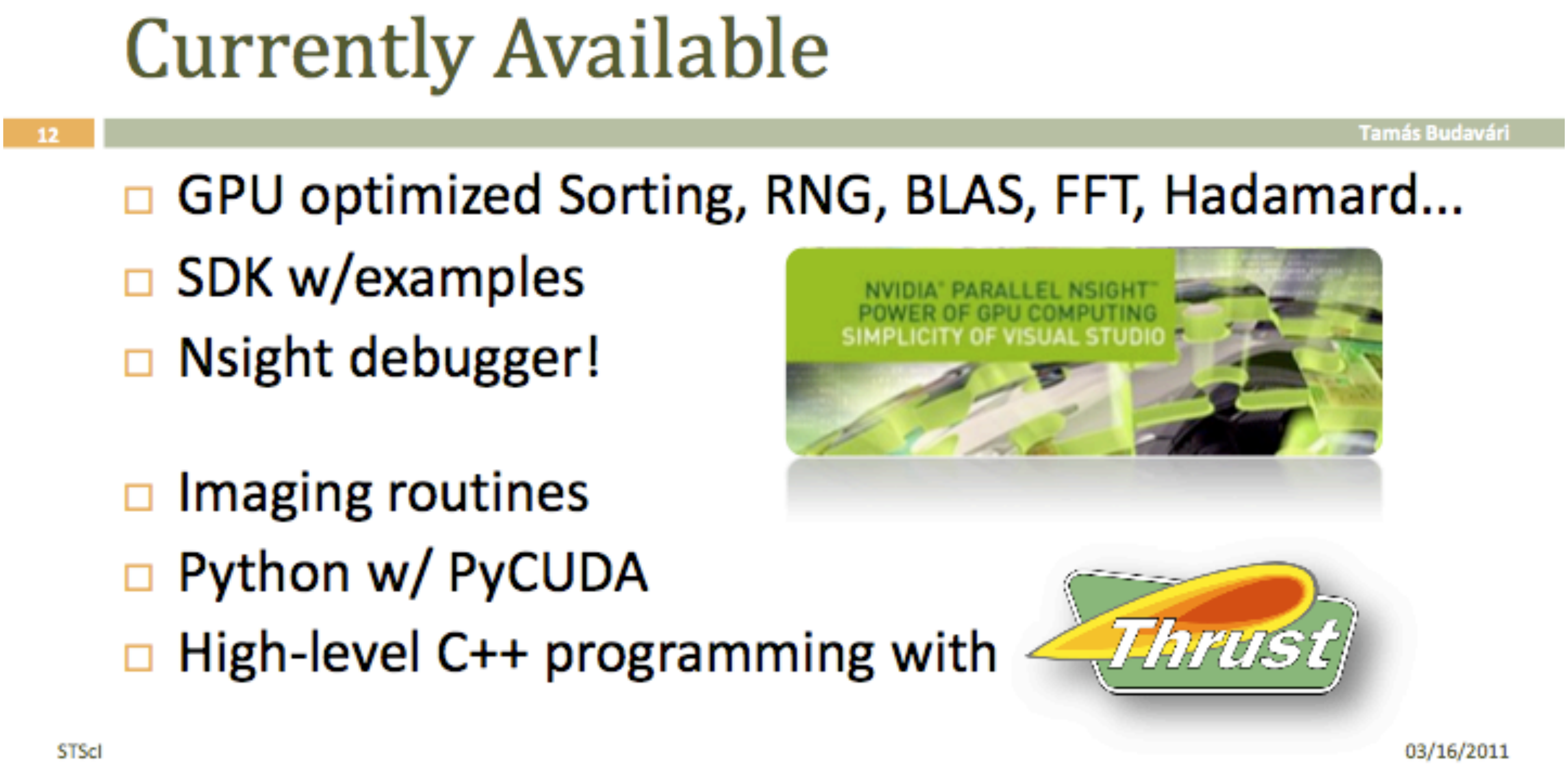
- Clean and simple

```
int main()
{
    ...
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
}
```

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

STScI

Slides from Budavari



Currently Available

12

Tamás Budavári

- ☐ GPU optimized Sorting, RNG, BLAS, FFT, Hadamard...
- ☐ SDK w/examples
- ☐ Nsight debugger!
- ☐ Imaging routines
- ☐ Python w/ PyCUDA
- ☐ High-level C++ programming with

STScI

03/16/2011

Slides from Budavari

Projects on CUDA Zone

NVIDIA Announces CUDA 4.0

Save the Date! GTC 2011

New Book: GPU Computing Gems

NVIDIA Announces Project Denver

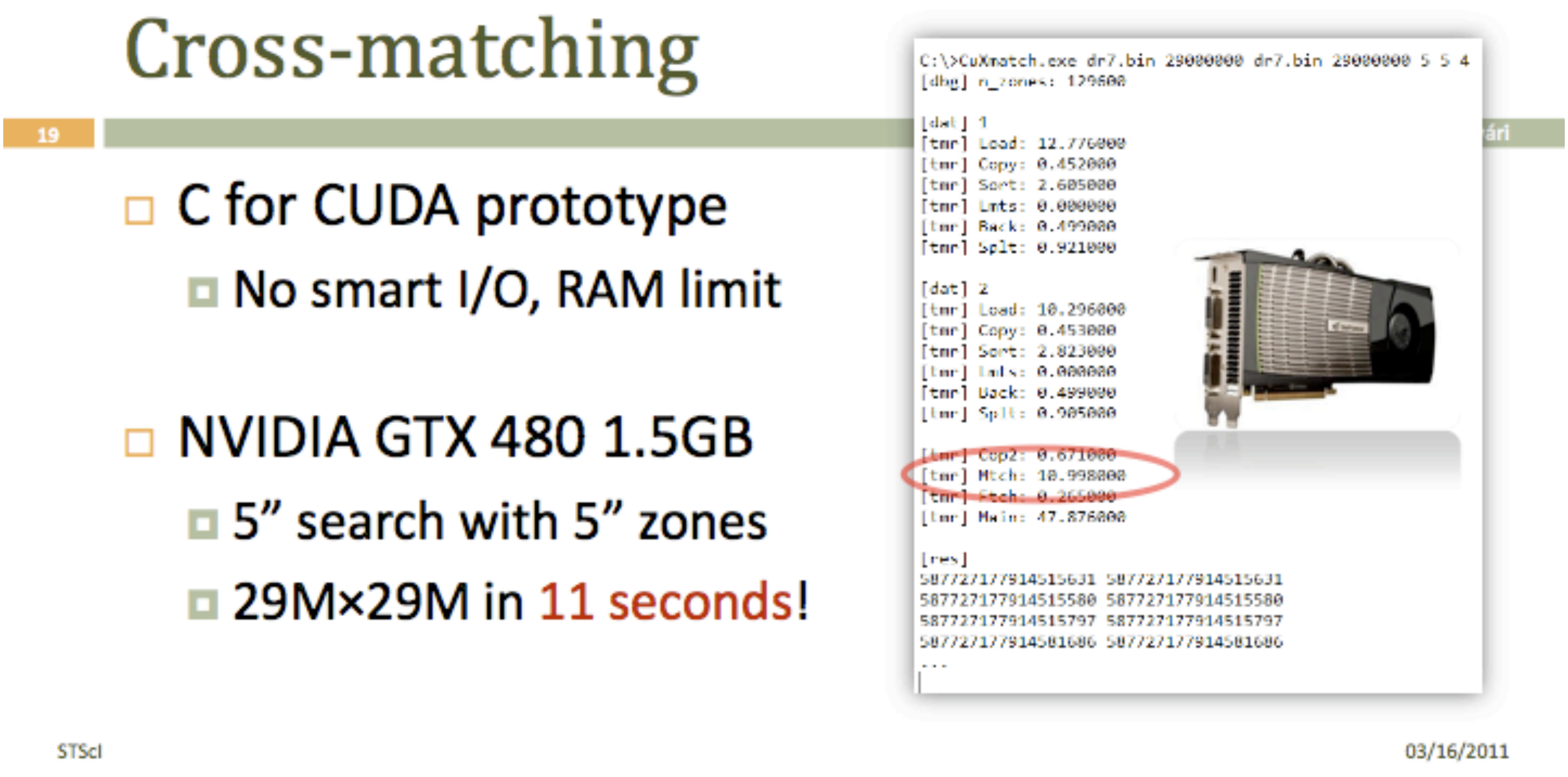
CUDA COMMUNITY SHOWCASE

View over 1000 papers and apps developed on the CUDA architecture by programmers, scientists, and researchers around the world.

STScI

03/16/2011

Slides from Budavari



Cross-matching

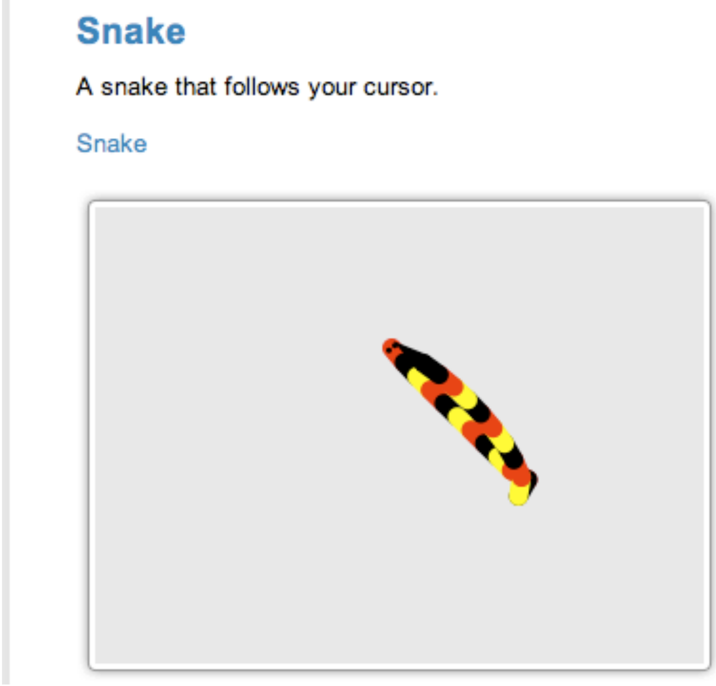
- C for CUDA prototype
  - No smart I/O, RAM limit
- NVIDIA GTX 480 1.5GB
  - 5” search with 5” zones
  - 29M×29M in 11 seconds!

```
C:\>CuXmatch.exe dr7.bin 29000000 dr7.bin 29000000 5 5 4
[dbg] n_zones: 129600

[dat] 1
[tmr] Load: 12.776000
[tmr] Copy: 0.452000
[tmr] Sort: 2.605000
[tmr] Lmts: 0.000000
[tmr] Back: 0.499000
[tmr] Splt: 0.921000

[dat] 2
[tmr] Load: 10.296000
[tmr] Copy: 0.453000
[tmr] Sort: 2.823000
[tmr] Lmts: 0.000000
[tmr] Back: 0.499000
[tmr] Splt: 0.905000

[tmr] Cop2: 0.671000
[tmr] Mtch: 10.998000
[tmr] Ftch: 0.265000
[tmr] Main: 47.876000

[res]
587727177914515631 587727177914515631
587727177914515580 587727177914515580
587727177914515797 587727177914515797
587727177914501686 587727177914501686
...
```

Horses for courses

- Don't marry a particular language
- Know one well, but do sample many other
- Use a language close to your domain
- Use tools which aid during programming

Processing.js



Snake

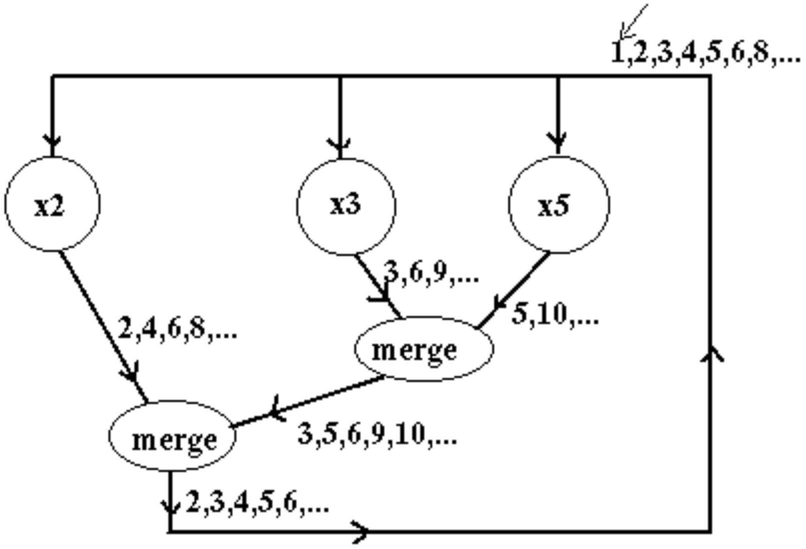
A snake that follows your cursor.

Snake

Hamming (regular) numbers

- $2^i * 3^j * 5^k$ (int i,j,k>=0)
- 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, ...
- Merge these lists:
  - 1;
  - 2, 4, 8, 16, ...;
  - 3, 9, 27, 81,...;
  - 5, 25, 125, ...
  - Is 7 in the list? 10? 333?

```
let rec
 merge = lambda a. lambda b.
   if hd a < hd b then (hd a)::(merge tl a b)
   else if hd b < hd a then (hd b)::(merge a tl b)
   else (hd a)::(merge tl a tl b),

 mul = lambda n. lambda l. (n* hd l)::(mul n tl l)

in let rec
 hamm = 1 :: (merge (mul 2 hamm)
             (merge (mul 3 hamm)
                    (mul 5 hamm)))

in hamm
```



L.Allison, Dept. Computer Science, Monash University

Exercise

- Write a program to generate Hamming numbers in at least 3 different (types?) of languages
- Compare them against each other in a few different ways (speed, memory, typing requirements)
- Use a debugger during the exercise and when testing it

In J:

```
hamming=: {. (/:~@~.@], 2 3 5 * {)/ @ (1x,~i.@-)
hamming 20
```