Programming language design in multi-core/many-core era

Gheorghe Stefanescu

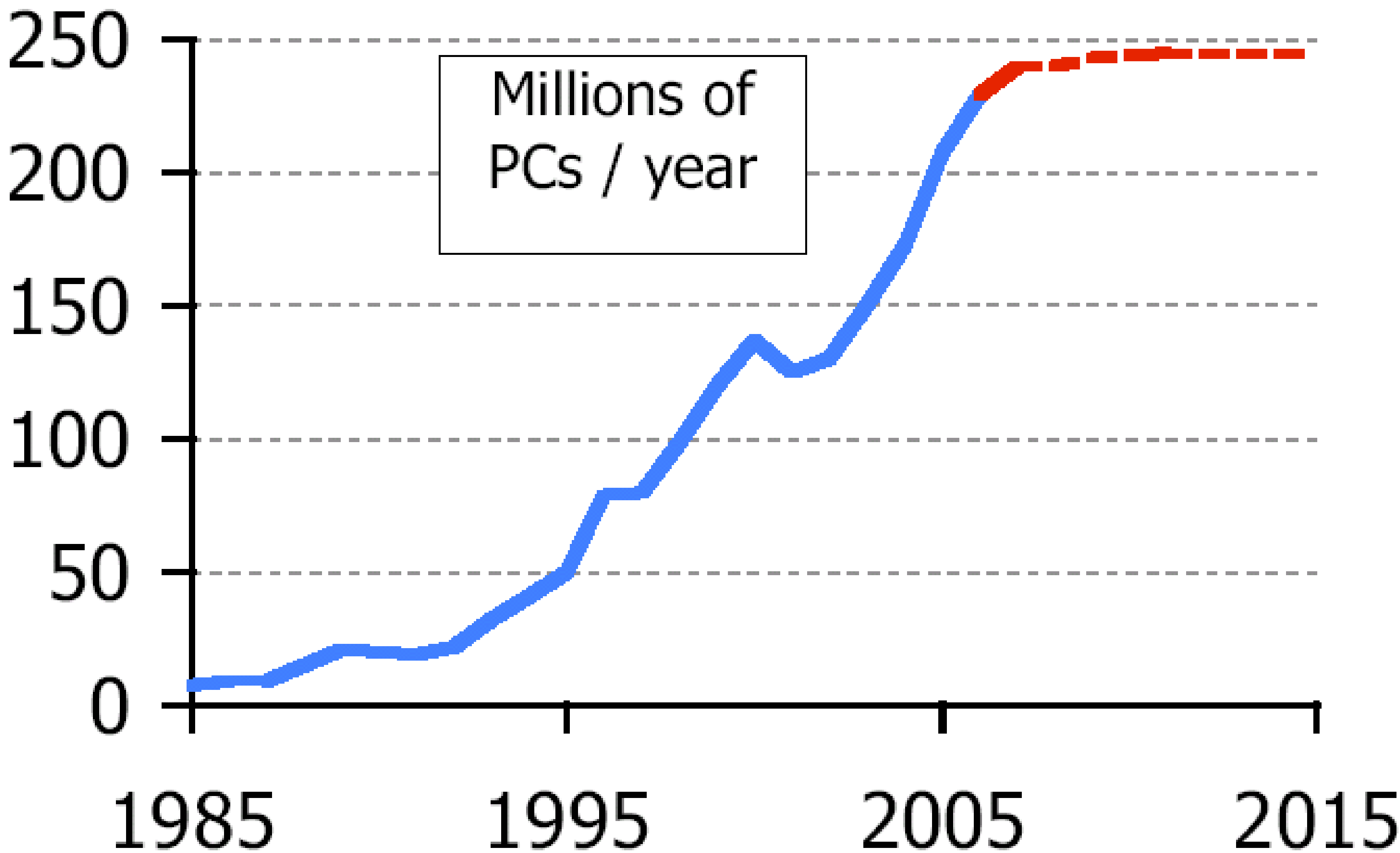
University of Bucharest

Fall, 2014

A software crisis?

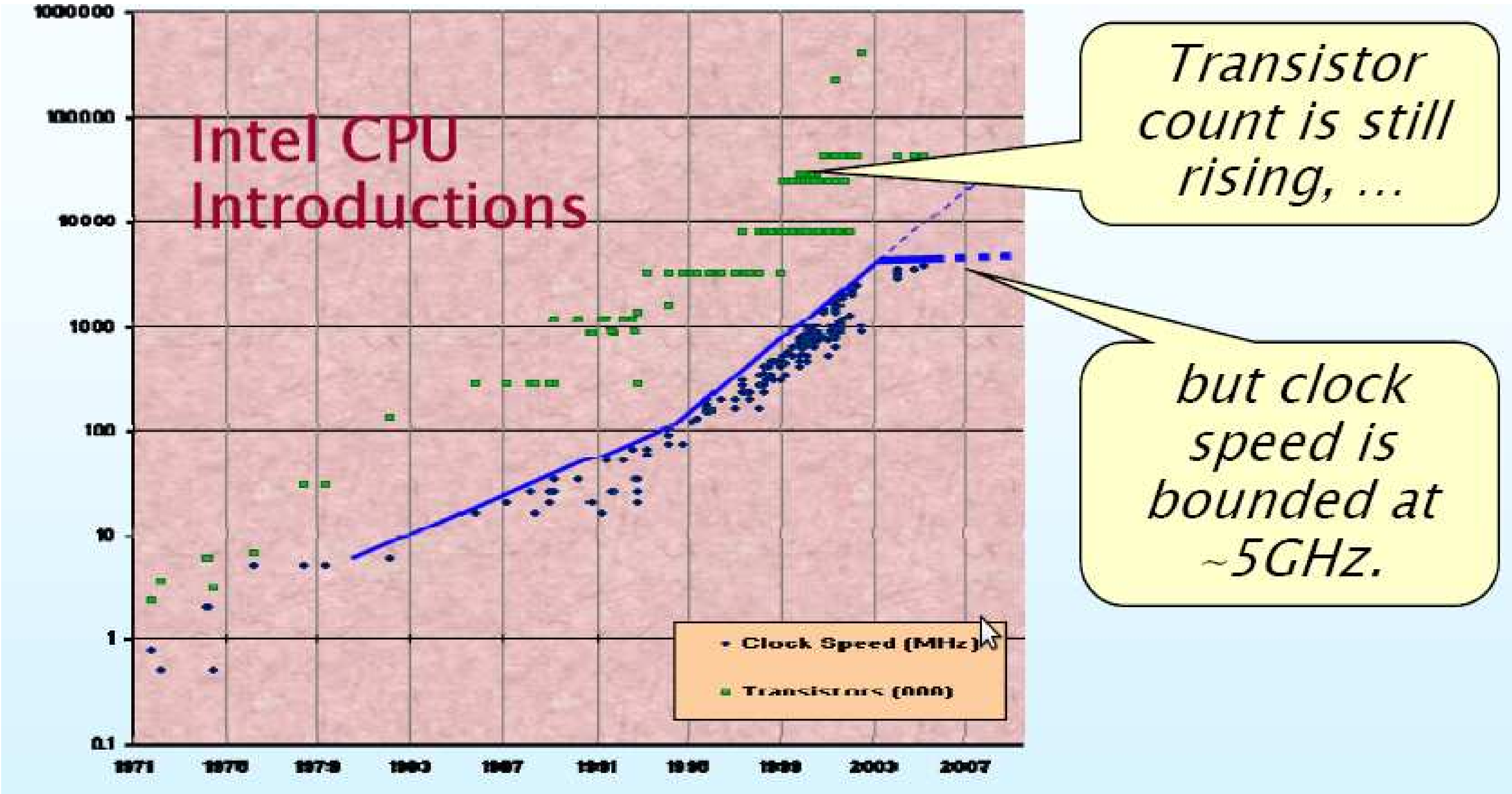
Sales PC's

Sales PC's - Industry of *replacement*



Moore’s law

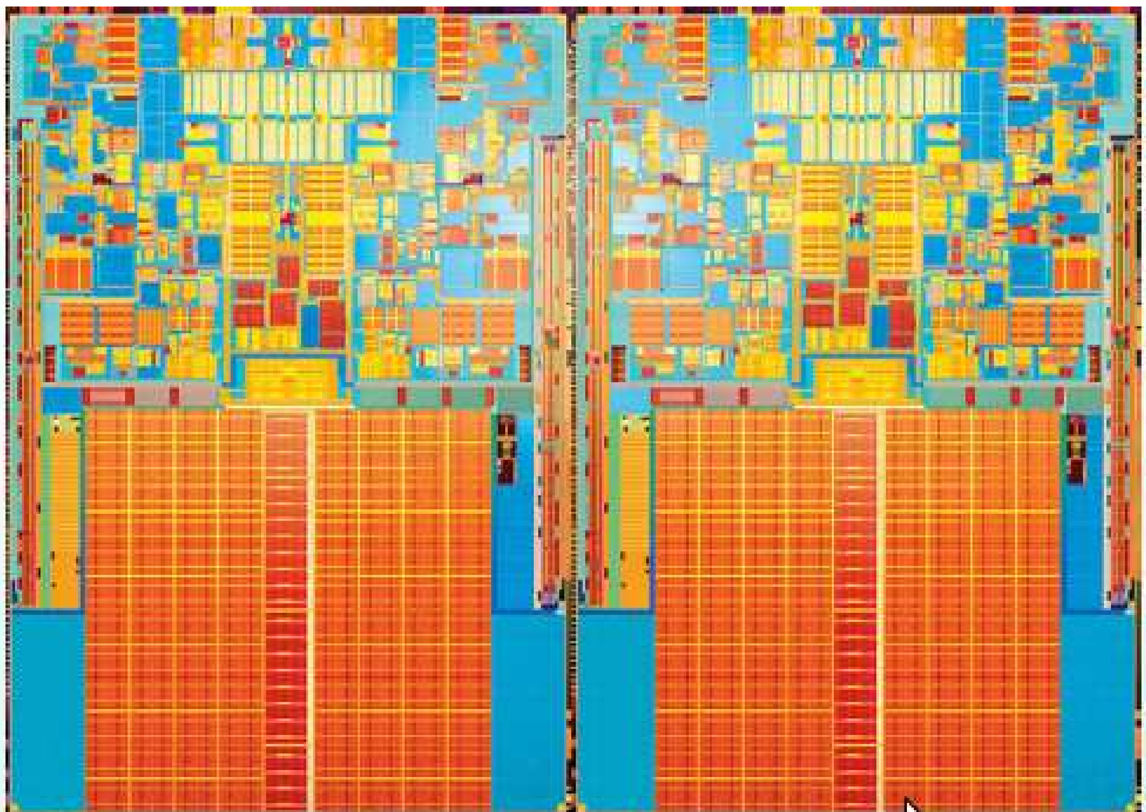
Moore’s law - bound clock cycle



Source: Herb Sutter, “The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software,” *Dr. Dobb's Journal*, 30(3), March 2005.

Intel many-core

Intel many-core



Intel 45nm quad-core processor

- To scale performance, put many processor cores on a chip.
- Intel will release 48 core chips by 2010

Promote parallel programming

Solutions for the “software crisis”

- promote *applications* demanding for *more computer power*
- design *new parallel programming languages*
- make parallel programming *easy*, accessible for the *average programmer*
- *interoperability*: various platforms - *Microsoft, Linux*

Multicore Software Dilemma

- Every software maker out there ... has got to learn how to program parallel code ... to remain competitive.

Dan Olds, Principal Analyst, Gabriel Consulting, Jan. 18, 2007

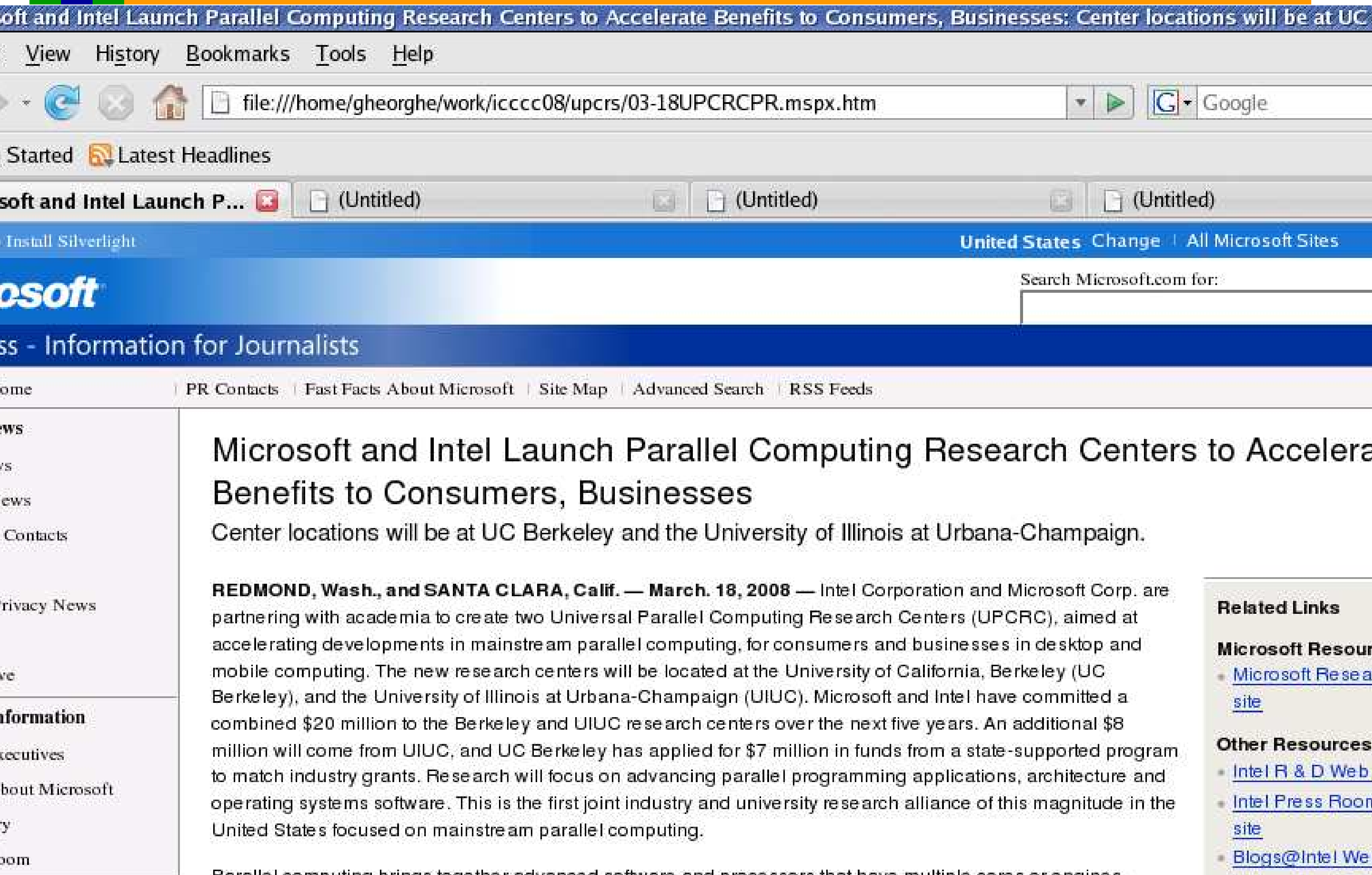
- This could become the biggest software remediation task of this decade.

Multicore Will Induce Operational and Software Headaches,Gartner Group, Jan. 31, 2007

- There's going to be a huge learning curve for developers to take on multi-threading in such a big way.

Sharon Gaudin, Information Week, Jan. 18, 2007

Microsoft and Intel initiative



oft and Intel Launch Parallel Computing Research Centers to Accelerate Benefits to Consumers, Businesses: Center locations will be at UC

View History Bookmarks Tools Help

file:///home/gheorghe/work/icccc08/upcrs/03-18UPCRCPR.mspx.htm

Google

Started Latest Headlines

soft and Intel Launch P... (Untitled) (Untitled) (Untitled)

Install Silverlight United States Change | All Microsoft Sites

osoft

Search Microsoft.com for:

ss - Information for Journalists

ome | PR Contacts | Fast Facts About Microsoft | Site Map | Advanced Search | RSS Feeds

ews
s
ews
Contacts
rivacy News
e
nformation
ecutives
bout Microsoft
y
oom

Microsoft and Intel Launch Parallel Computing Research Centers to Accelera Benefits to Consumers, Businesses

Center locations will be at UC Berkeley and the University of Illinois at Urbana-Champaign.

REDMOND, Wash., and SANTA CLARA, Calif. — March. 18, 2008 — Intel Corporation and Microsoft Corp. are partnering with academia to create two Universal Parallel Computing Research Centers (UPCRC), aimed at accelerating developments in mainstream parallel computing, for consumers and businesses in desktop and mobile computing. The new research centers will be located at the University of California, Berkeley (UC Berkeley), and the University of Illinois at Urbana-Champaign (UIUC). Microsoft and Intel have committed a combined $20 million to the Berkeley and UIUC research centers over the next five years. An additional $8 million will come from UIUC, and UC Berkeley has applied for $7 million in funds from a state-supported program to match industry grants. Research will focus on advancing parallel programming applications, architecture and operating systems software. This is the first joint industry and university research alliance of this magnitude in the United States focused on mainstream parallel computing.

Related Links

Microsoft Resour
- Microsoft Resea site

Other Resources
- Intel R & D Web
- Intel Press Roon site
- Blogs@Intel We

Microsoft and Intel initiative

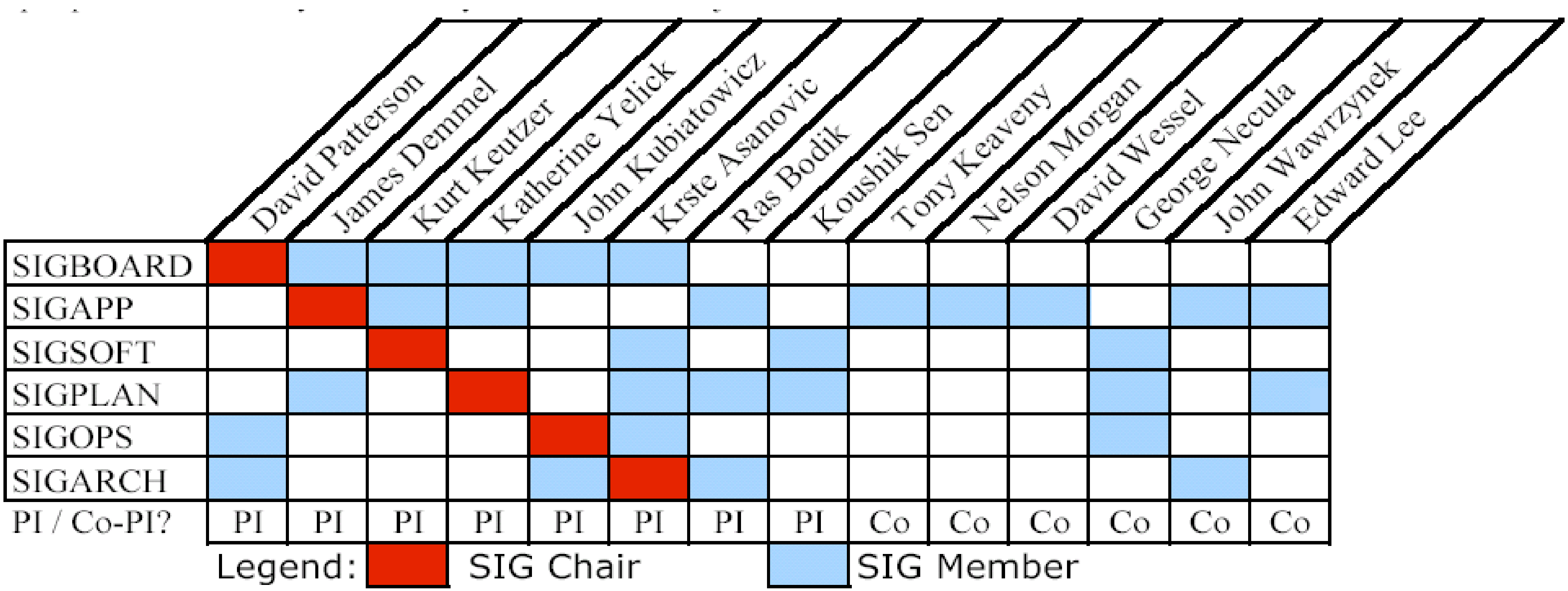
Microsoft and Intel initiative

- March. 18, 2008: *Microsoft* and *Intel* has launched *Universal Parallel Computing Research Center (UPCRC)* to accelerate benefits to consumers, businesses
- 2 UPCRCs:
  University of California, Berkeley (UC Berkeley) and
  the *University of Illinois at Urbana-Champaign* (UIUC)
- *\$20 million* to the Berkeley and UIUC research centers over the next five years (*plus \$15 million* from local sources)
- "Intel has already shown an *80-core* research *processor*, and we're quickly moving the computing industry to a *many-core world*," said Andrew Chien, vice president Intel

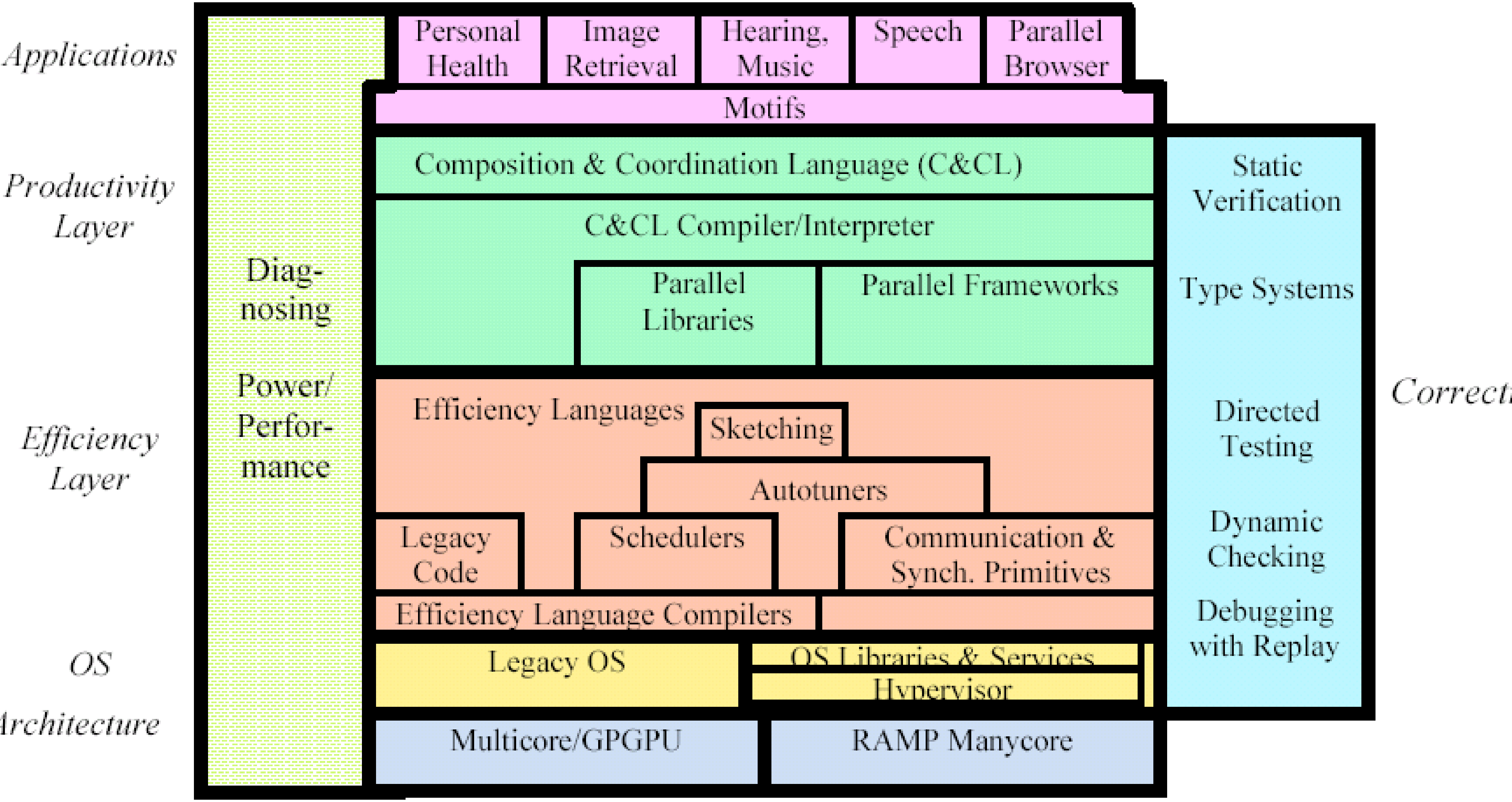
Researchers at Berkeley

Source: *The Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View*, Report EECS-2008-23, March 21, 2008;

Asanovic, Krste / Bodik, Ras / Demmel, James / Keaveny, Tony / Keutzer, Kurt / Kubiatowicz, John D. / Lee, Edward A. / Morgan, Nelson / Necula, George / *Patterson, David A.* / Sen, Koushik / Wawrzynek, John / Wessel, David / Yelick, Katherine A.

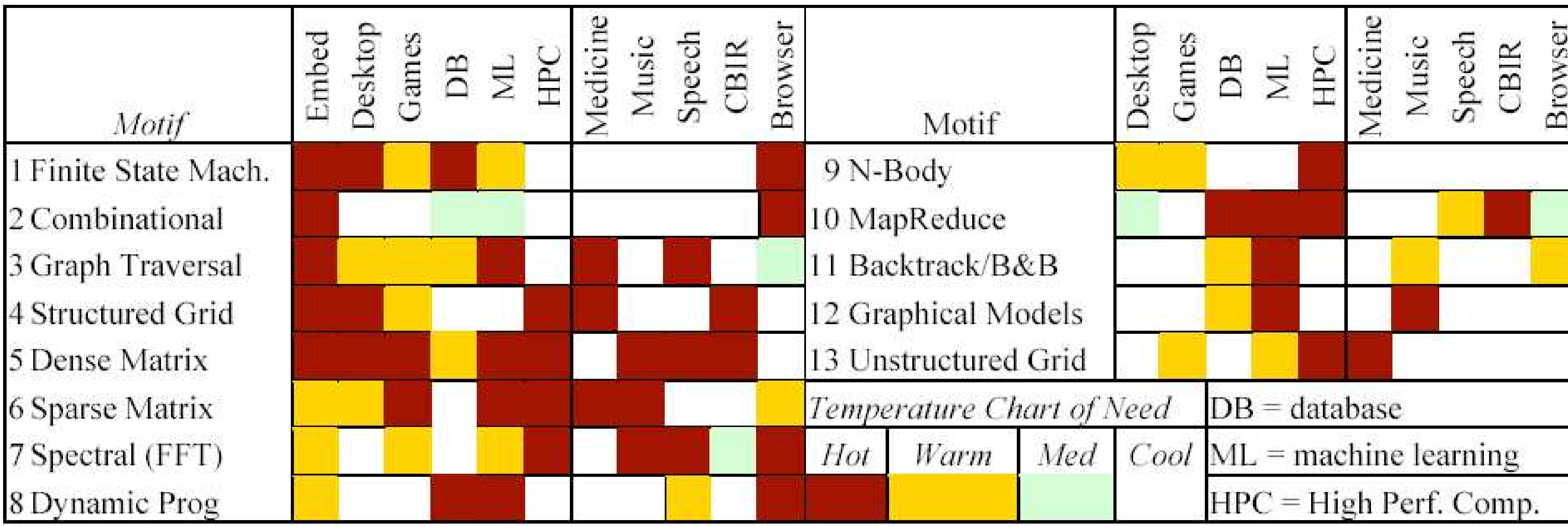


Research agenda at Berkeley



Research agenda at Berkeley - II

Parallel programming applications

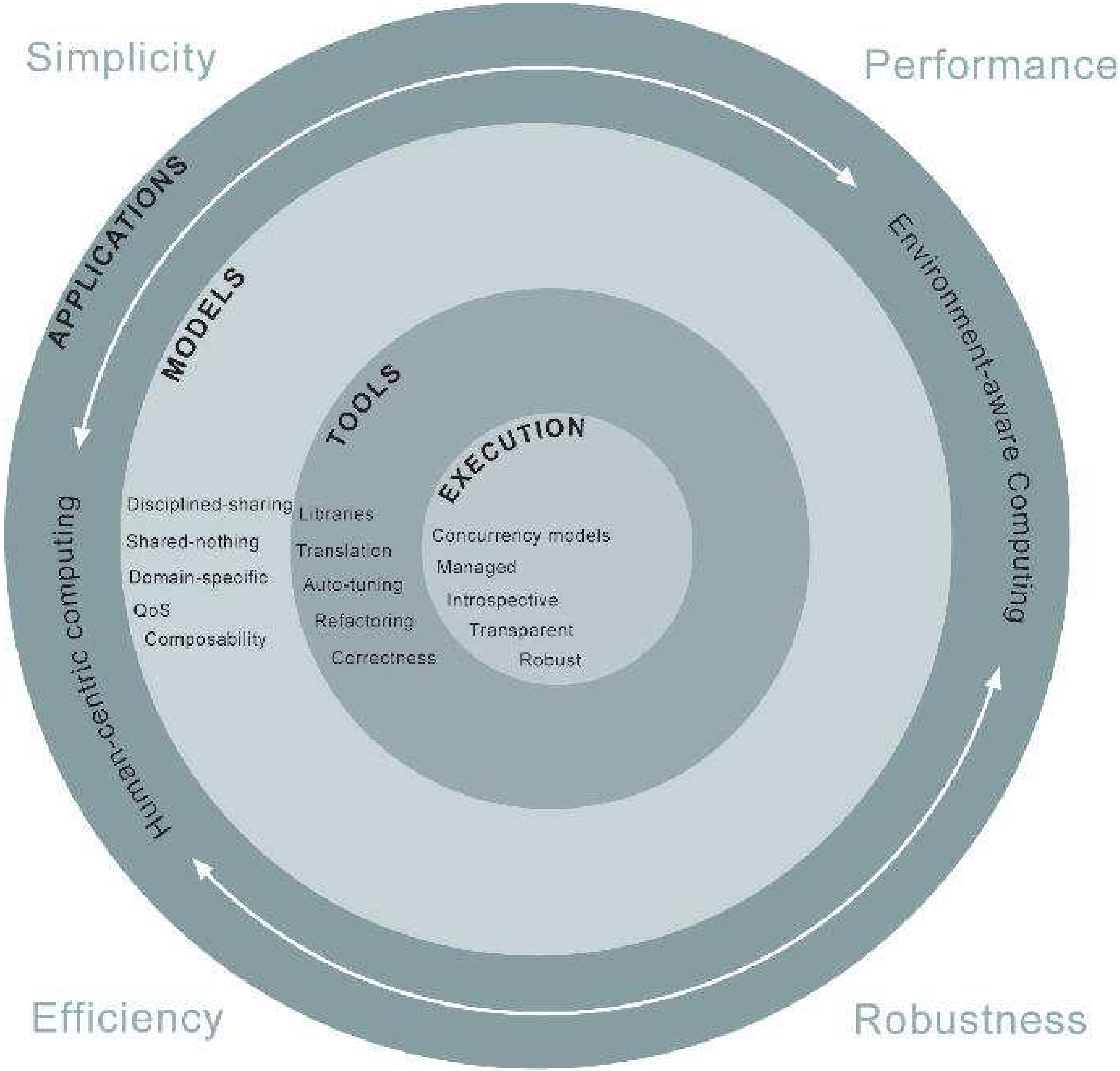


Researchers at UIUC

- *Marc Snir*, Co-Director
- *Wen-mei Hwu*, Co-Director
- *Sarita Adve*, Director of Research
- Vikram Adve, Gul Agha, Eyal Amir, David Forsyth, Matthew Frank, Maria Garzaran, John Hart, Ralph Johnson Laxmikant Kale, Rakesh Kumar, Darko Marinov, Klara Nahrstedt, David Padua, Sanjay Patel, Grigore Rosu, Dan Roth, Josep Torrellas, YuanYuan Zhou, Craig Zilles

Research agenda at UIUC

Research agenda at UIUC



Research agenda at UIUC - II

- generations of the *ILLIAC*, the *CEDAR* machine, the Illinois *Cache Coherence Protocol*, helped define the landscape of *multiprocessors*.
- Parafrase, Polaris, and IMPACT systems - *autoparallelization* for successful commercial parallelizing compilers.
- *MPI* - the most popular programming paradigm for *clusters*.
- *speculative multithreading* in the I-ACOMA project

Research agenda at UIUC - III

- *actors*, a paradigm of concurrent computation: rigorously defined formal semantics, used in novel parallel programming languages such as *Erlang, E, Ptolemy, Thal, Scala and SALSA*.
- *Java and C++ memory* models - *concurrency semantics* for the most popular threads programming languages
- the largest academic supercomputer: *NCSA* - The National Center for Supercomputing Applications
- a center for *petascale computing*

Cum s-a ajuns aici?

O scurta istorie...

Posibile motive:

- Promovarea excesiva a *sistemelor de calcul cu un singur procesor*
- Esecul (economic) al *supercalculatoarele traditionale*
- Lipsa unui *model standard* de calcul paralel (asemanator masinii Turing)
- Conceptul de *program memorat* (autonom) si dificultatea de a trata *interactia* intre procese

Performanta

Promovarea excesiva a *sistemelor cu un singur procesor*

- *Legea lui Amdahl:* Performanta maxima pentru o problema cu un factor serial f este

$$S(n) = \frac{t_s}{f t_s + \frac{(1-f)t_s}{n}} = \frac{n}{1+(n-1)f}$$

- *Consecinta:* Cu un numar infinit de procesoare, performanta maxima prin paralelizare este *limitata la* $1/f$.

  E.g., daca 5% este serial, performanta maxima este 20, indiferent cate procesoare folosim.

- *Prost folosita!* - Nu se refera la performanta paralelizarii unui *program*, ci a ... *unei executii*.

Hardware paralel:

In fundal, paralelismul a fost folosit masiv in hardware, dar mai mult pentru *sisteme cu un singur procesor*, spre exemplu:

- mai multe *"CPUs"*
  - "integer" ori "floating point"
- tehnici de *pipeline*
  - principala sursa de performanta a procesoarelor in anii '90
- facilitati pentru *hiper-threading*

Pipeline

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register \$2 :	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM :	X	X	X	–20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	–20	X	X	X	X

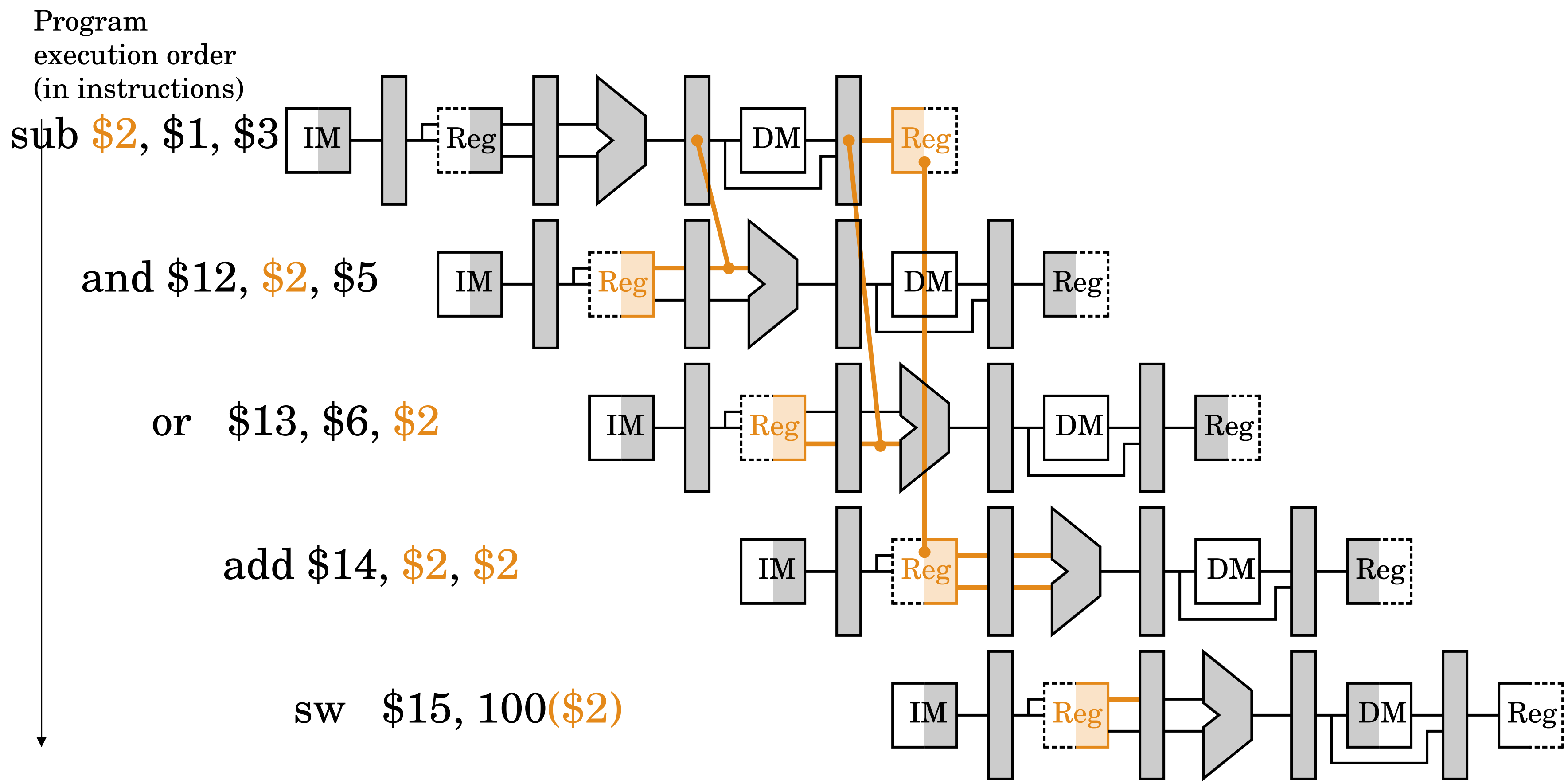


Figura ilustrează tehnica de pipeline (cu tehnica de *avansare* pentru a rezolva hazardurile de date).

Supercalculatoare dedicate: povestea unui *succes stiintific*, dar *esec economic*...

- supercomputere Cray (CDC) - anii ’70
- programare logica, generatia 5 de calculatoare
- arhitecturi dataflow architecture - anii ’80
  - Danny Hillis: *Connection Machine* at *Thinking Machines Corporation* (5 variante)

Supercalculatoare azi: (Cray, IBM, HP, etc.) - unicate, produse la cerere (“*clustere de PC-uri*”)

Jaguar is a petascale supercomputer built by *Cray* at Oak Ridge National Laboratory in Oak Ridge, Tennessee. As of November 2009, it was the *world’s fastest* computer with a peak performance of more than *1750 teraflops* (1.75 petaflops). A Cray XT5 system, Jaguar has *224,256 Opteron processor cores*, and operated with a version of Linux called Cray Linux Environment

Calcul performant “usor” accesibil:

- clustere de PC-uri
  - UB-Fizica, PUB, Timisoara, TU-Iasi
- calculatoare multi-core
- procesoare grafice (GPUs)

> Both nVidia and ATI have teamed with Stanford University to create a GPU-based client for the *Folding@Home* distributed computing project, for protein folding calculations. In certain circumstances the GPU calculates *forty times faster* than the conventional CPUs traditionally used by such applications.

Parallel computers I
- message passing computing

(multi-core)

Parallel programming options

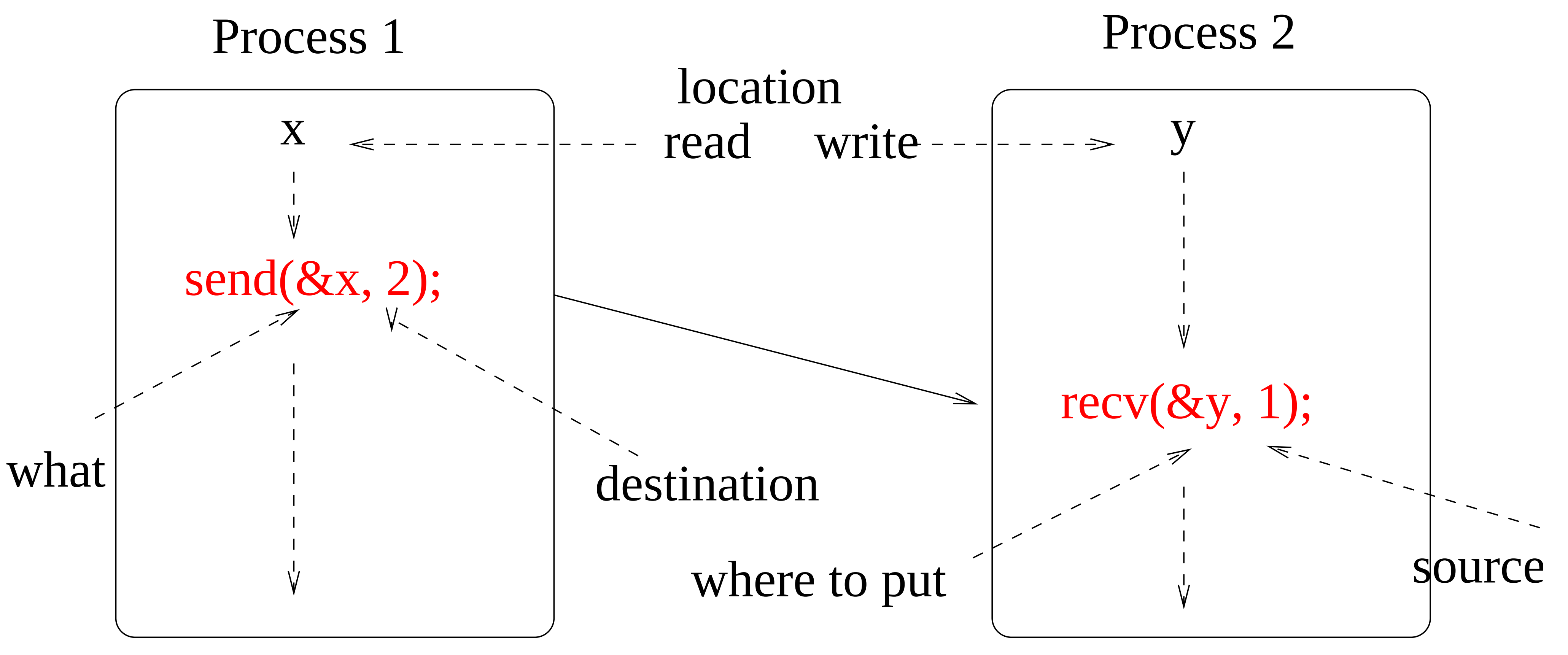
Programming a message-passing multicomputer can be achieved by

- Designing a *special* parallel programming language (e.g., OCCAM for transputers)
- *Extending* the syntax/reserved words of an existing sequential high-level language to handle message passing (e.g., CC+, FORTRAN M)
- Using an existing sequential high-level language and providing a *library* of external procedures for message passing (e.g., MPI, PVM)

Another option will be to write a sequential program and to use a *parallelizing compiler* to produce a parallel program to be executed by multicomputer.

Basic send and receive routines

Passing a message between processes using `send()` and `recv()` library calls



MPI

Message Passing Interface (MPI):

- standard MPI 2.0

  http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

- implementation MPICH 2.0

  http://www.mcs.anl.gov/research/projects/mpich2/

- the “de facto” standard for
  - *commodity clusters*
  - *high-speed networks*
  - *supercomputers*

..Example (MPI program)

```
#include "mpi.h"
#includes <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv){
      int myid, nproc;
      int data[MAXSIZE], i, x, low, high, myresult, result;
      char fn[255];
      char *fp;
      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD,&nproc);
      MPI_Comm_rank(MPI_COMM_WORLD,&myid);
      if (myid == 0){
           strcpy(fn,getenv("HOME"));
           strcat(fn,"/MPI/rand_data.txt");
           if((fp = fopen(fn,"r")) == NULL){
                printf("Can't open the input file %s\n\n",fn);
                exit(1);
           }
```

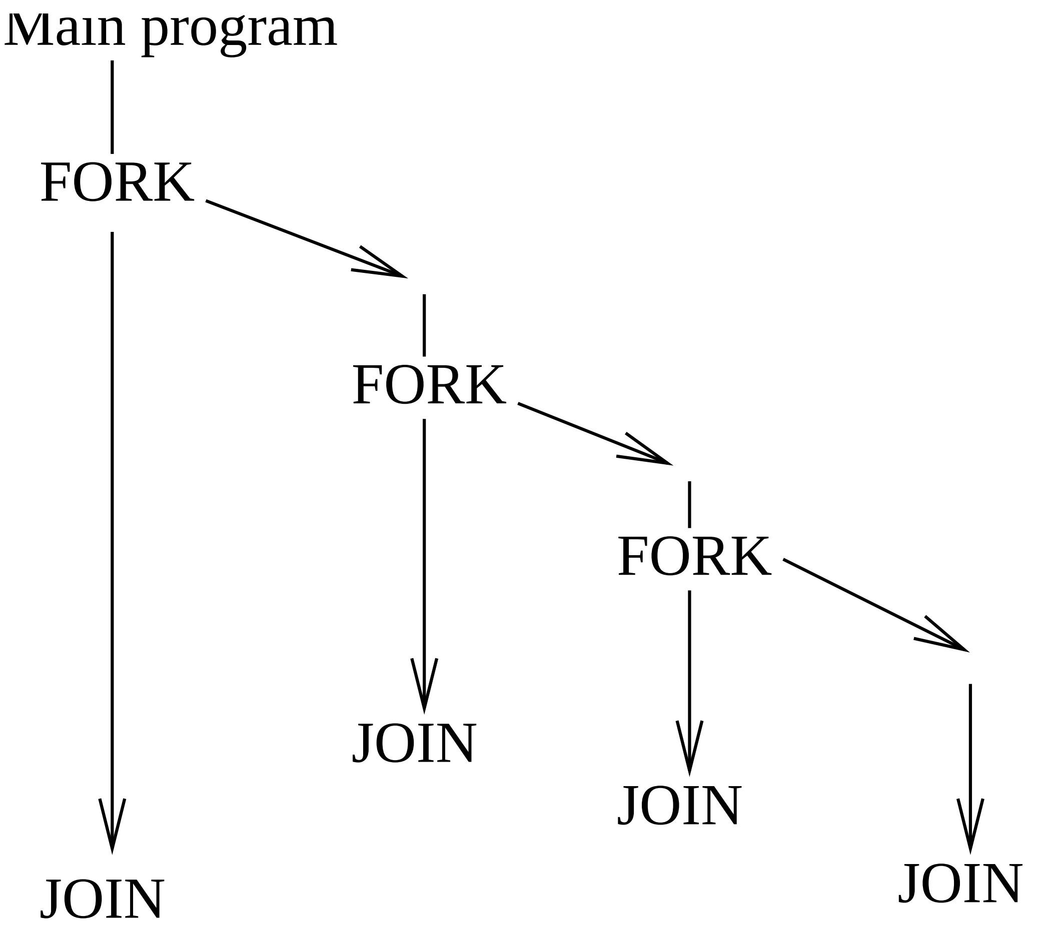
..Example (MPI program)

```
        for (i=0; i<MAXSIZE; i++) fscanf(fp,"%d",&data[i]);
    }
    /* Broadcast data */
    MPI_Bcast(data,MAXSIZE,MPI_INT,0,MPI_COMM_WORLD);
    /* Add my portion of data */
    x = n / nproc;
    low = myid * x;
    high = low + x;
    for (i=low; i<high; i++)
        myresult += data[i];
    printf("I got %d from %d\n", myresult,myid);
    /* Compute global sum */
    MPI_Reduce(&myresult,&result,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
    if (myid == 0) printf("The sum is %d.\n",result);
    MPI_Finalize();
}
```

Parallel computers II
- shared memory systems

(many-core)

Fork-Join construct



Threads

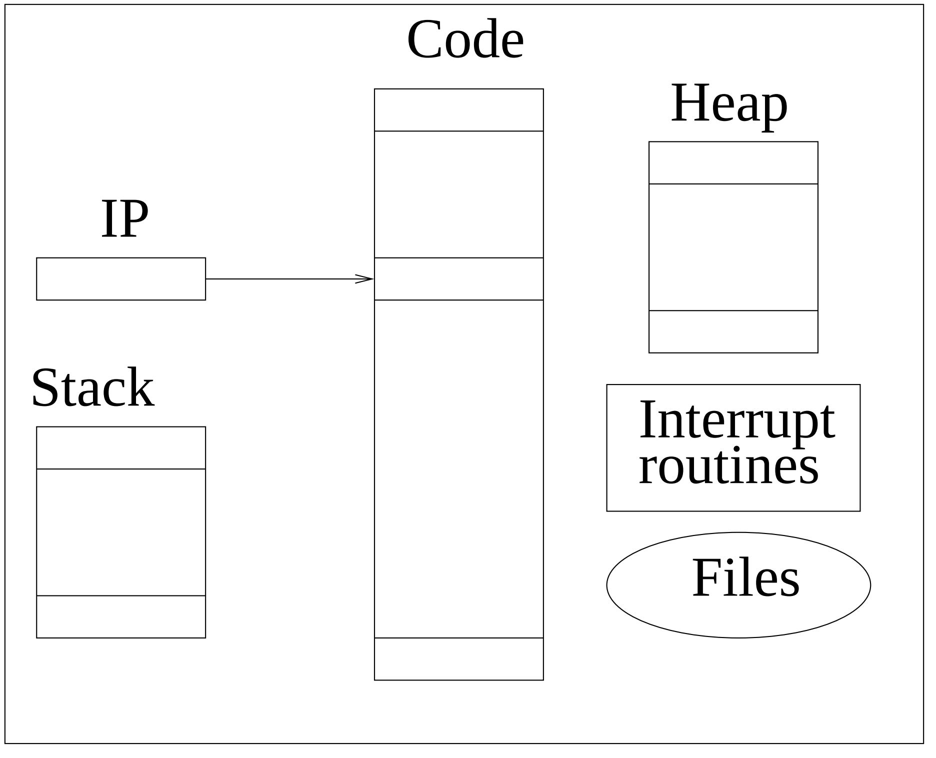
Threads vs. (heavyweight) processes:

- heavyweight *processes* are completely *separate* programs (with their own variables, stack, memory allocation)
- *threads* *share* the same memory space and global variables (but they have their own stack)

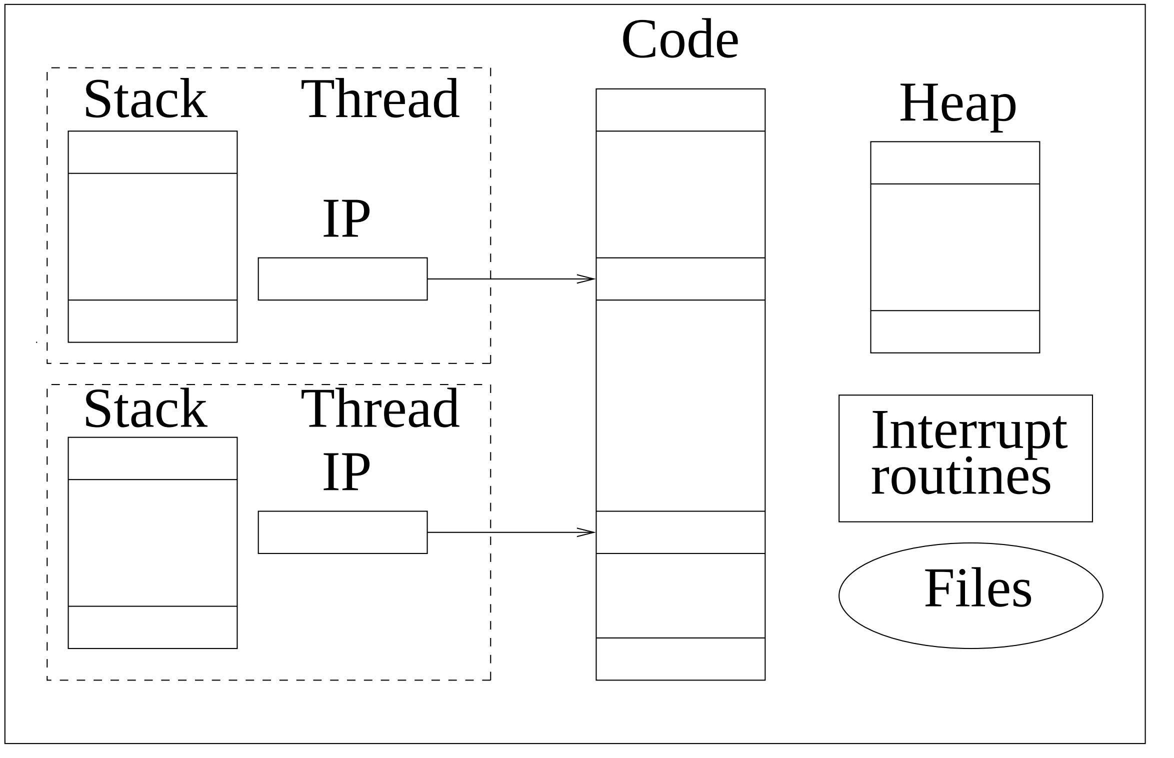
Threads' advantages:

- thread *creation* can take three orders of magnitude less time than process creation
- *communication* (via shared memory) and *synchronization* are also faster than in the case of processes

Threads



(a) Process



(b) Threads

Pthreads

Pthreads:

- library over C/C++
- low-level parallel programming with shared memory

Example - adding numbers

```
#include <stdio.h>
#include <pthread.h>
#define array_size 1000
#define no_threads 10

int a[array_size];
int global_index = 0;
int sum = 0;
pthread_mutex_t mutex1;
```

```
main()
{
  int i;
  pthread_t thread[10];
  pthread_mutex_init(&mutex1, NULL);

  for (i = 0; i < array_size; i++)
    a[i] = i+1;

  for (i = 0; i < no_threads; i++)
    if (pthread_create(&thread[i], NULL, slave, NULL) != 0)
      perror("Pthread_create fails");

  for (i = 0; i < no_threads; i++)
    if (pthread_join(thread[i], NULL) != 0)
      perror("Pthread_join fails");

  printf("The sum of 1 to %i is %d\n", array_size, sum);
}
```

```
void *slave(void *ignored)
{
  int local_index, partial_sum = 0;
  do {
    pthread_mutex_lock(&mutex1);
      local_index = global_index;
      global_index++;
    pthread_mutex_unlock(&mutex1);
    if (local_index < array_size)
      partial_sum += *(a + local_index);
  } while (local_index < array_size);
  pthread_mutex_lock(&mutex1);
    sum += partial_sum;
  pthread_mutex_unlock(&mutex1);
  return();
}
```

High-level shared memory programming languages:

- recent effort to make *"parallel programming easier"*
- basic approach:
  - write *sequential programs*
    ... and *inform the compiler* on parts of the programs that can be run (and how to run) in parallel

Examples:

- OpenMP and Cilk

OpenMP

- Standard OpenMP 3.0; adopted by Intel, Microsoft
  - http://openmp.org/

Example:

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char **argv) {
   const int N = 100000;
   int i, a[N];
   #pragma omp parallel for
   for (i = 0; i < N; i++)
      a[i] = 2 * i;
   return 0;
}
```

Limitation: Limited support for nested parallelism, recursion.

Cilk

- introduced by C Leiserson (former Chief Architect at CM-5); recently adopted by Intel
- language, implementation, running environment (scheduler)
  - http://www.cilk.com/

Example:

```
cilk int fib (int n) {
  if (n < 2) return n;
  else {
    int x, y;

    x = spawn fib (n-1);
    y = spawn fib (n-2);

    sync;

    return (x+y);
  }
}
```

Interactive computing
- a new computing paradigm

Turing:

- *On computable numbers, with an application to the Entscheidungsproblem* (written in 1936)
- *"Automatic"* vs. *"interactive machines"*

> Automatic machines.
>
> If at each stage the motion of a machine (in the sense of 1) is completely determined by the configuration, we shall call the machine an *"automatic machine"* (or *a-machine*). For some purposes we might use machines (*"choice machines"* or *c-machines*) whose motion is only partially determined by the configuration (hence the use of the word possible in 1). When such a machine reaches one of these ambiguous configurations, it cannot go on until some *arbitrary choice has been made by an external operator*. This would be the case if we were using machines to deal with axiomatic systems. In this paper I deal only with automatic machines, and will therefore often omit the prefix a-.
>
> (Turing, 1936)

Recent approaches:

- *coordination languages*
  - LINDA, REO, AGAPIA, etc.
- *orchestration languages*
  - ORC, BPEL (WSBPEL), etc.
- Theory & applications
  - *Interactive Computation: The New Paradigm*, D. Goldin, S. Smolka, P. Wagner (Eds.), Springer, 2006.

Agapia

- a programming language for interactive, parallel systems

Contents

Agapia

- *Generalities*
- A glimpse on AGAPIA programming
- Finite interactive systems $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- Compiling srv-programs
- Floyd-Hoare logics for (s)rv-programs

History

History

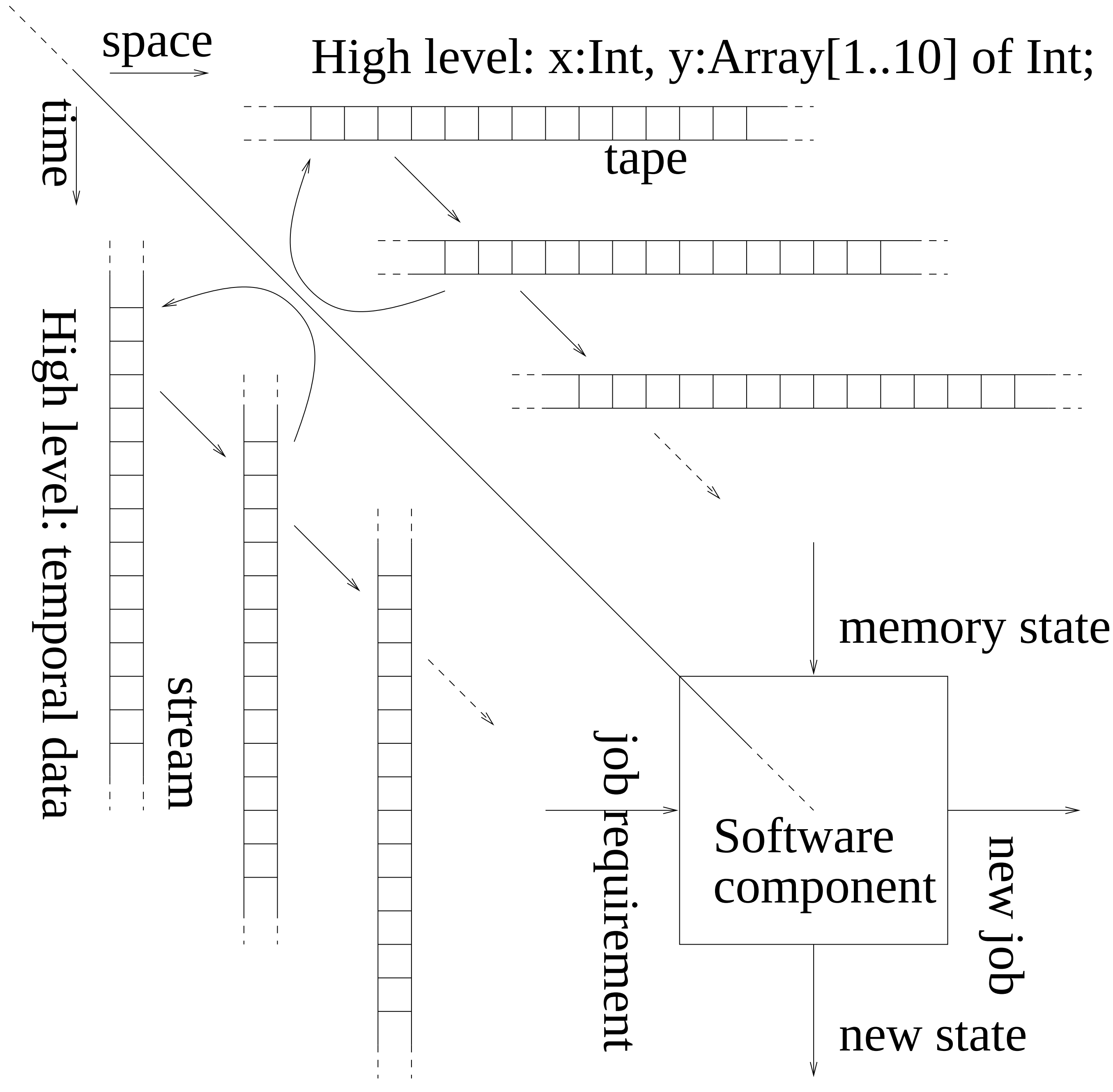
- *space-time duality "thesis"*
  - Stefanescu, *Network algebra*, Springer 2000
- *finite interactive systems*
  - Stefanescu, Marktoberdorf Summer School 2001
- *rv-systems* (interactive systems with registers and voices)
  - Stefanescu, NUS, Singapore, summer 2004
- *structured rv-systems*
  - Stefanescu, Dragoi, Popa, Sofronia, etc. - since 2006

RV-Systems and Agapia Programming - link UIUC

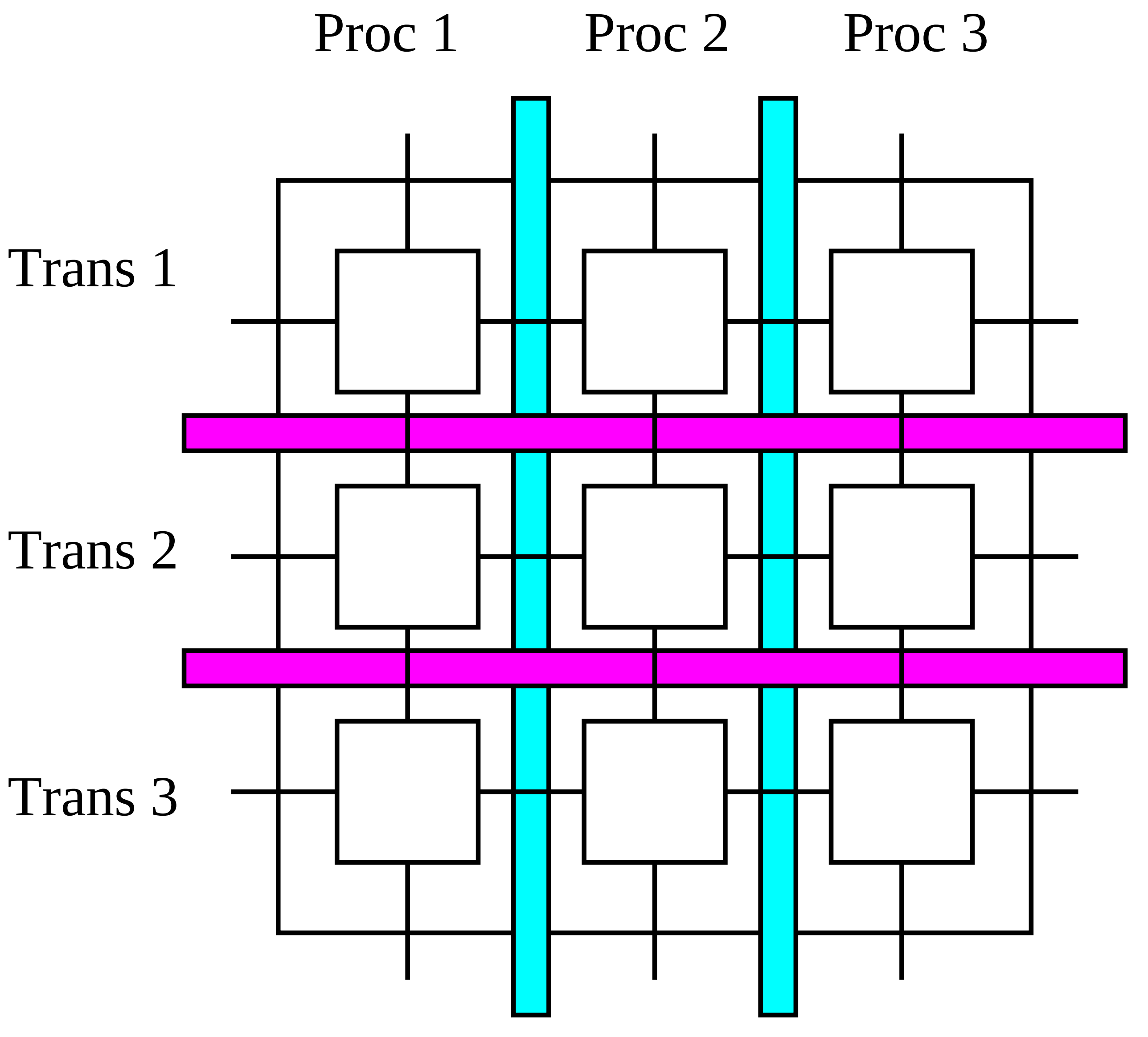
http://fsl.cs.uiuc.edu/index.php/RV_Systems_and_Agapia_Programming

ST-Dual picture

ST-Dual picture



Processes and transactions



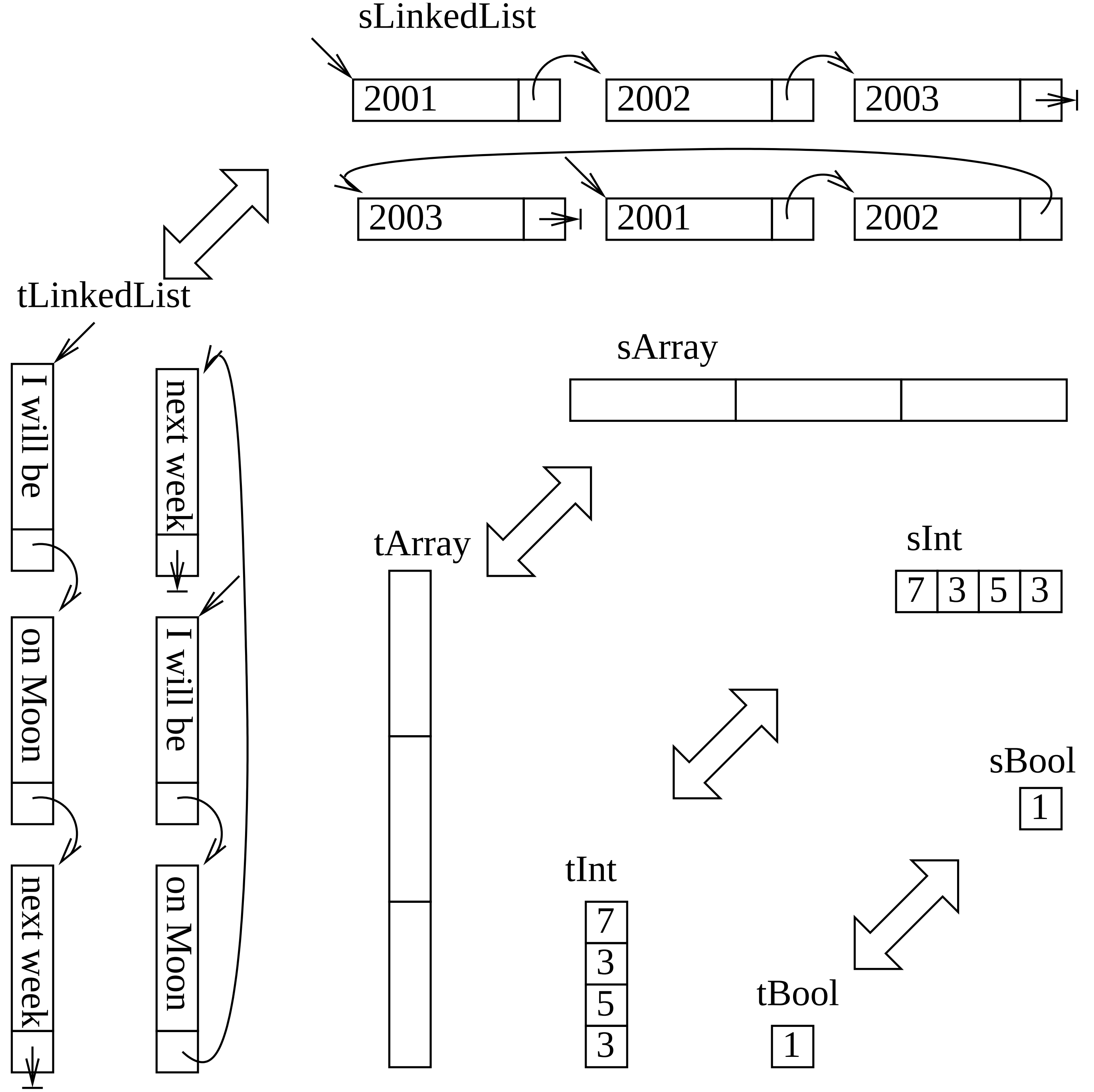
High level temporal structures

data with usual (*spatial*) representation:

sBool, sInt, sArray, sLinkedList, etc.

and their *time dual* (i.e., data with *temporal representation*):

tBool, tInt, tArray, tLinkedList, etc.



Rv-systems

Agapia

- Generalities
- *A glimpse on AGAPIA programming*
- Finite interactive systems $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- Compiling srv-programs
- Floyd-Hoare logics for (s)rv-programs

Perfect numbers

A specification for perfect numbers:

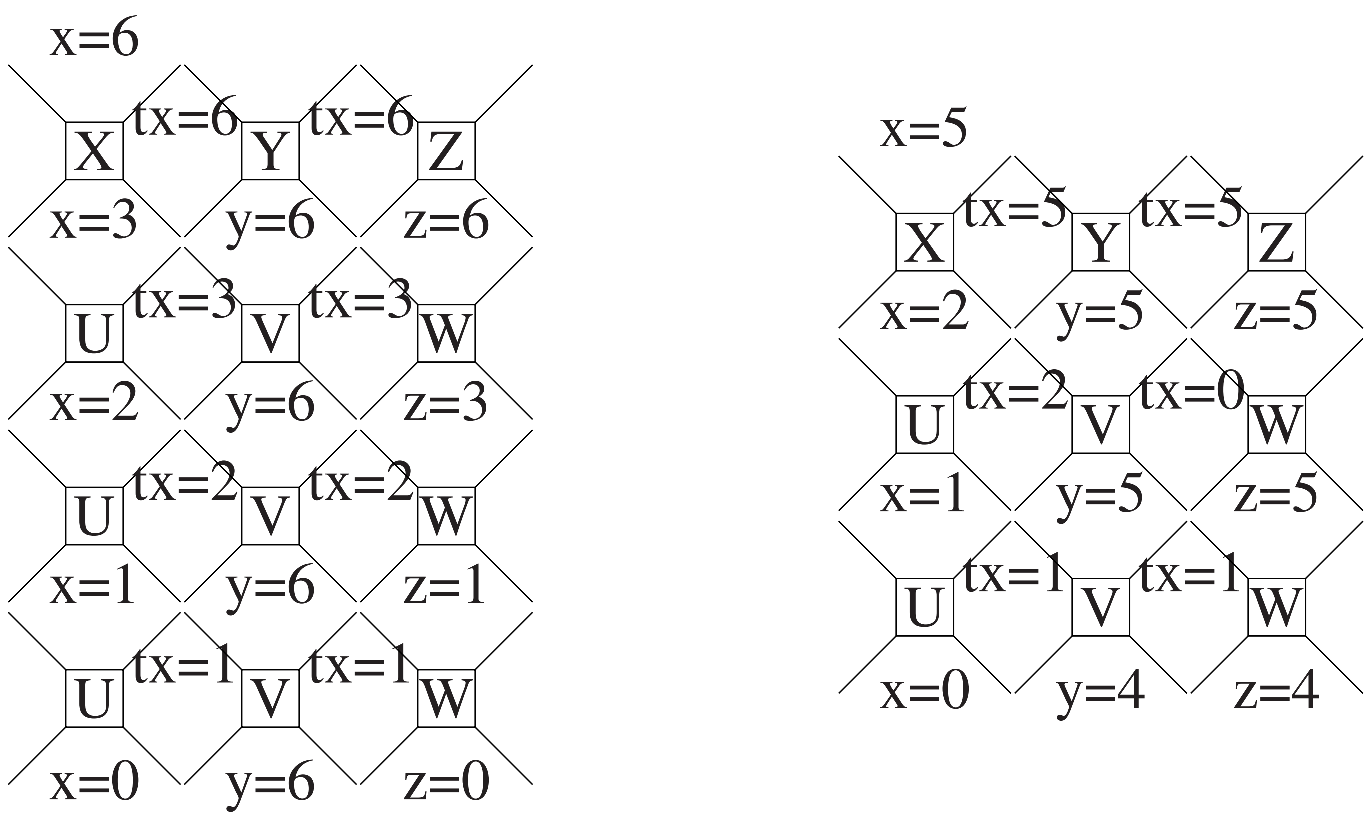
3 components C_x, C_y, C_z where:

- C_x: read n from north and write $n \frown \lfloor n/2 \rfloor \frown (\lfloor n/2 \rfloor - 1) \frown \ldots \frown 2 \frown 1$ on east;
- C_y: read $n \frown \lfloor n/2 \rfloor \frown (\lfloor n/2 \rfloor - 1) \frown \ldots \frown 2 \frown 1$ from west and write $n \frown \phi(\lfloor n/2 \rfloor) \frown \ldots \frown \phi(2) \frown \phi(1)$ on east [$\phi(k) =$ "if k divides n then k else 0"];
- C_z: read $n \frown \phi(\lfloor n/2 \rfloor) \frown \ldots \frown \phi(2) \frown \phi(1)$ from west and subtract from the first the other numbers.

These components are composed *horizontally*. The global input-output specification: *if the input number in C_x is n, then the output number in C_z is 0 iff n is perfect*.

..Perfect numbers

Two scenarios for perfect numbers:



(a) (b)

Types are denoted as $\langle west|north\rangle \to \langle east|south\rangle$

Our (s)rv-scenarios are similar with the tiles of Bruni-Gadducci-Montanari, et.al.

..Perfect numbers

The 1st AGAPIA program Perfect1 (construction by rows):

```
(X # Y # Z) % while_t(x>0){U # V # W}
```

Its type is **Perfect1** : $\langle nil|sn;nil;nil\rangle \rightarrow \langle nil|sn;sn;sn\rangle$.

Modules:

Classical, imperative program

```
X:: module{listen nil;}{read x:sn;}
        {tx:tn; tx=x; x=x/2;}{speak tx;}{write x;}
Y:: module{listen tx:tn;}{read nil;}
        {y:sn; y=tx;}{speak tx;}{write y;}
Z:: module{listen tx:tn;}{read nil;}
        {z:sn; z=tx;}{speak nil;}{write z;}
U:: module{listen nil;}{read x:sn;}
        {tx:tn; tx=x; x=x-1;}{speak tx;}{write x;}
V:: module{listen tx:tn;}{read y:sn;}
        {if(y%tx != 0) tx=0;}{speak tx;}{write y;}
W:: module{listen tx:tn;}{read z:sn}
        {z=z-tx;}{speak nil;}{write z;}
```

..Perfect numbers

The 2nd AGAPIA program Perfect2 (construction by columns):

```
(X % while_t(x>0){U} % U1)
# (Y % while_t(tx>-1){V} % V1)
# (Z % while_t(tx>-1){W} % W1)
```

Its type is **Perfect2** : $\langle nil|sn;nil;nil\rangle \rightarrow \langle nil|nil;nil;sn\rangle$.

New modules: *Dataflow program*

```
U1:: module{listen nil;}{read x:sn;}
         {tx:tn; tx=-1;}{speak tx;}{write nil;}
V1:: module{listen tx:tn;}{read y:sn;}
         {null;}{speak tx;}{write nil;}
W1:: module{listen tx:tn;}{read z:sn}
         {null;}{speak nil;}{write z;}
```

Contents

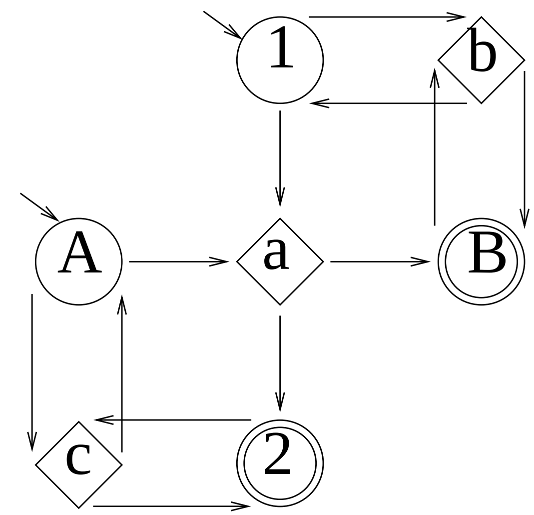
Agapia

- Generalities
- A glimpse on AGAPIA programming
- *Finite interactive systems* $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- Compiling srv-programs
- Floyd-Hoare logics for (s)rv-programs

Finite interactive systems

Finite interactive systems:

- *states*: 1,2 [1-initial; 2-final]
- *classes*: A, B [A-initial; B-final]
- *transitions*: a, b, c



Parsing procedure (to recognize grids):

A parssing for
```
abb
cab
cca
```
:

```
 1 1 1    1 1 1    1 1 1    1 1 1       1 1 1
Aa b b   AaBb b   AaBbBb   AaBbBb ...  AaBbBbB
          2        2 1      2 1         2 1 1
Ac a b   Ac a b   Ac a b   AcAa b      AcAaBbB
                            2           2 2 1
Ac c a   Ac c a   Ac c a   Ac c a      AcAcAaB
                                        2 2 2
```

FIS vs. 2-dimensional languages

Theorem:

The following are equivalent for a 2-dimensional language L (called recognizable two-dimensional language*; their class is denoted by REC):*

1. *L is recognized by a* on-line tessellation automaton*;*
2. *L is defined by a* tile systems *(i.e., local lattice languages closed to letter-to-letter homomorphisms);*
3. *L is defined by an* existential monadic second order formula*; etc.*

See: Giammarresi-Restivo (1997), or Lindgren-Moore-Nordahl (1998); a useful web-page is B.Borchert's page at

`http://math.uni-heidelberg.de/logic/bb/2dpapers.html`

Notice: 2-dimensional languages are also known as "picture" languages.

..FIS vs. 2-dimensional languages

Theorem:

> *A set of grids is recognizable by a finite interactive system iff it is recognizable by a tiling system.*

This shows that the class of FIS recognizable grid languages coincides with REC, so we may *inherit many results known for 2-dimensional languages*. Two important ones are:

Corollaries:

1. *Context-sensitive word languages coincide with the projection on the 1st row of the* FIS *recognizable grid languages.*

2. *The emptiness problem for* FIS*'s is undecidable.*

Contents

Agapia

- Generalities
- A glimpse on AGAPIA programming
- Finite interactive systems $\leftarrow$ *[nfa]*
- *Rv-programs* $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- Compiling srv-programs
- Floyd-Hoare logics for (s)rv-programs

RV-programs

RV-systems:

- An *rv-system* (*interactive system with registers and voices*) is a FIS enriched with:
  - *registers* associated to its *states* and *voices* associated to its *classes*;
  - appropriate *spatio-temporal transformations for actions*.

  We study rv-systems specified by *rv-programs* (see below)
- A *computation* is described by a scenario like in a FIS, but with concrete data around each action.

..RV-programs

An rv-program (for perfect numbers):

in: A,1; out: D,2

X::

(A,1)	x : sInt

	tx : tInt;
	tx = x;
	x = x/2;
	goto [B,3];

U::

(A,3)	x : sInt

	tx : tInt;
	tx = x;
	x = x − 1;
	if (x > 0) goto [B,3]
	else goto [B,2];

Y::

(B,1)	y : sInt
tx : tInt	y = tx; goto [C,2];

V::

(B,2)	y : sInt
tx : tInt	if(y%tx != 0) tx = 0; goto [C,2];

Z::

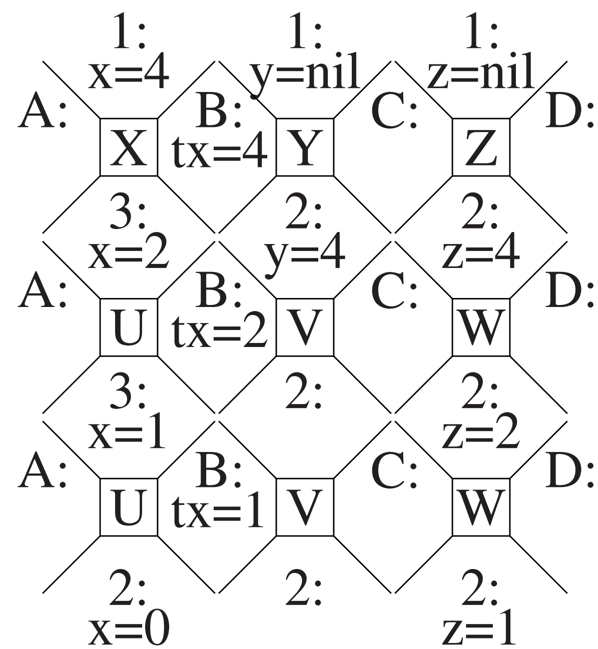
(C,1)	z : sInt
tx : tInt	z = tx; goto [D,2];

W::

(C,2)	z : sInt
tx : tInt	z = z − tx; goto [D,2];

..RV-programs

Scenario:



Operational semantics:

- defined in terms of scenarious

Relational semantics:

- input-output relation generated by all possible scenarious

Contents

Agapia

- Generalities
- A glimpse on AGAPIA programming
- Finite interactive systems $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- *Structured rv-programs* $\leftarrow$ *[while programs]*
- Compiling srv-programs
- Floyd-Hoare logics for (s)rv-programs

Basic characteristics of AGAPIA

- *space-time invariant*
- *high-level temporal data* structures
- *computation extends* both in *time* and *space*
- a *structural, compositional model*
- simple *operational semantics* (using *scenarios*)
- simple *relational semantics*
- a *name-free* interaction calculus

AGAPIA v0.1: Syntax

Syntax of AGAPIA v0.1:

Interfaces

$SST ::= nil \mid sn \mid sb$
$\quad \mid (SST \cup SST) \mid (SST, SST) \mid (SST)^*$
$ST ::= (SST)$
$\quad \mid (ST \cup ST) \mid (ST; ST) \mid (ST;)^*$
$STT ::= nil \mid tn \mid tb$
$\quad \mid (STT \cup STT) \mid (STT, STT) \mid (STT)^*$
$TT ::= (STT)$
$\quad \mid (TT \cup TT) \mid (TT; TT) \mid (TT;)^*$

Expressions

$V ::= x : ST \mid x : TT$
$\quad \mid V(k) \mid V.k \mid V.[k] \mid V@k \mid V@[k]$
$E ::= n \mid V \mid E + E \mid E * E \mid E - E \mid E/E$
$B ::= b \mid V \mid B \&\& B \mid B || B \mid !B \mid E < E$

Programs

$W ::= null \mid new\ x : SST \mid new\ x : STT$
$\quad \mid x := E \mid if(B)\{W\}else\{W\}$
$\quad \mid W; W \mid while(B)\{W\}$
$M ::= module\{listen\ x : STT\}\{read\ x : SST\}$
$\quad \{\ W\ \}\{speak\ x : STT\}\{write\ x : SST\}$
$P ::= null \mid M \mid if(B)\{P\}else\{P\}$
$\quad \mid P\%P \mid P\#P \mid P\$P$
$\quad \mid while\_t(B)\{P\} \mid while\_s(B)\{P\}$
$\quad \mid while\_st(B)\{P\}$

Example: Termination detection

Example: A program for distributed termination detection

```
P= I1# for_s(tid=0;tid<tm;tid++){I2}#
   $ while_st(!(token.col==white && token.pos==0)){
     for_s(tid=0;tid<tm;tid++){R}}
```

where:

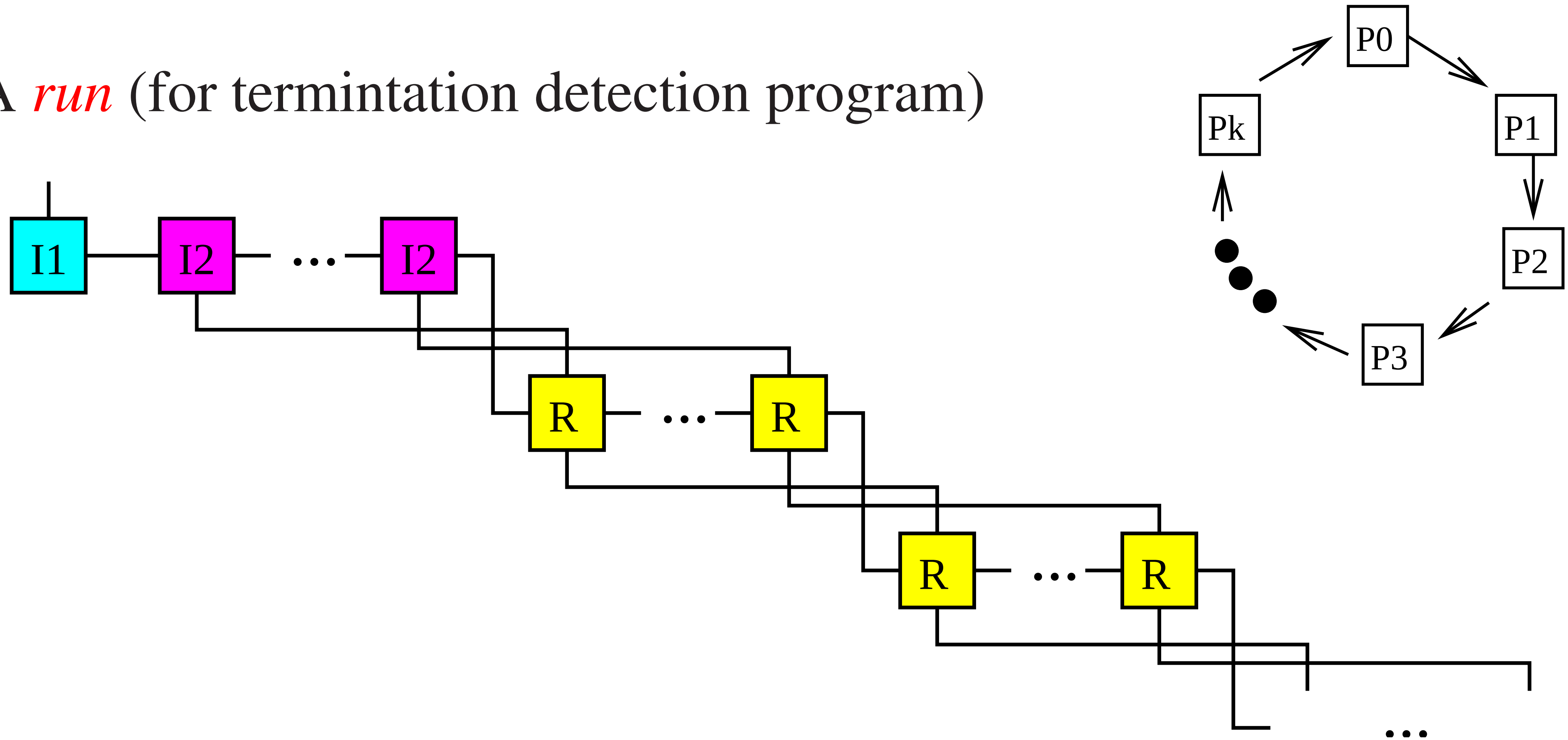
```
I1= module{listen nil}{read m}{
    tm=m; token.col=black; token.pos=0;
    }{speak tm,tid,msg[ ],token(col,pos)}{write nil}

I2= module{listen tm,tid,msg[ ],token(col,pos)}
    {read nil}{
    id=tid; c=white; active=true; msg[id]=null;
    }{speak tm,tid,msg[ ],token(col,pos)}
    {write id,c,active}
```

..Example: Termination detection

```
R=module{listen tm,tid,msg[ ],token(col,pos)}
  {read id,c,active}{
  if(msg[id]!=emptyset){ //take my jobs
     msg[id]=emptyset;
     active=true;}
  if(active){ //execute code, send jobs, update color
     delay(random_time);
     r=random(tm-1);
     for(i=0;i<r;i++){ k=random(tm-1);
       if(k!=id){msg[k]=msg[k]∪{id}};
       if(k<id){c=black};}
     active=random(true,false);}
  if(!active && token.pos==id){ //termination
     if(id==0)token.col=white;
     if(id!=0 && c==black){token.col=black;c=white};
     token.pos=token.pos+1[mod tm];}
  }{speak tm,tid,msg[ ],token(col,pos)}
  {write id,c,active}
```

..Example: Termination detection

A *run* (for termintation detection program)

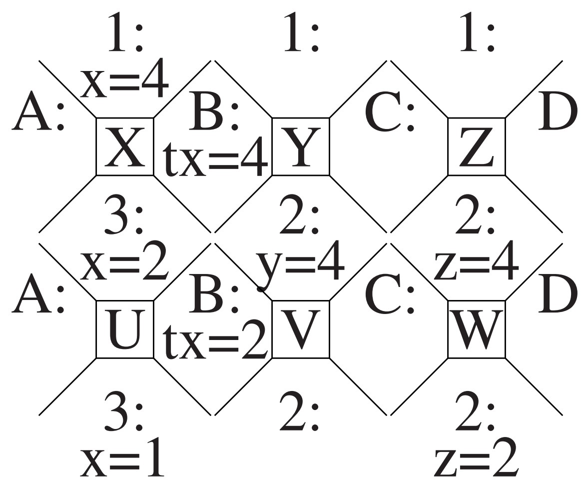


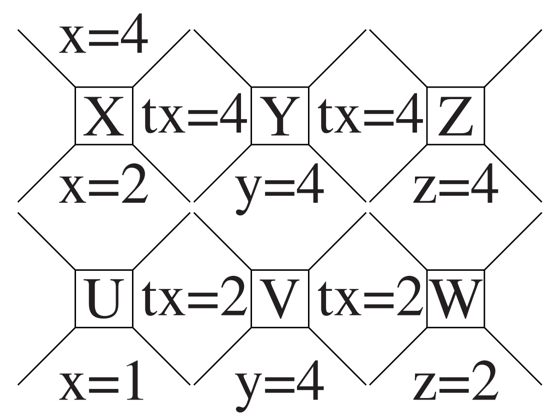

```
I1# for_s(tid=0;tid<tm;tid++){I2}#
$ while_st(!(token.col==white && token.pos==0)){
   for_s(tid=0;tid<tm;tid++){R}}
```


Scenarios

Scenarios:

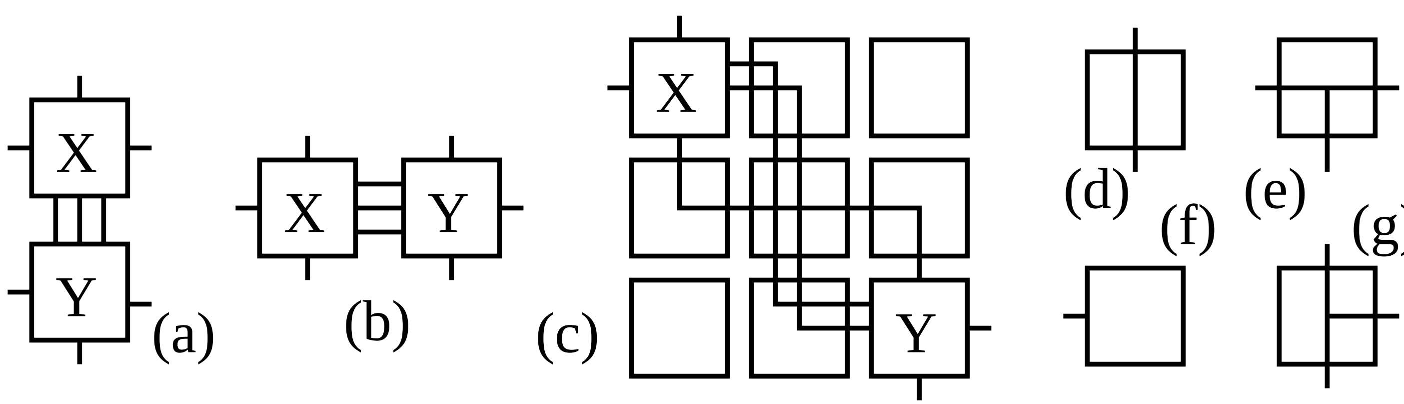
```
 1  1  1
AaBbBbB
 2  1  1
AcAaBbB
 2  2  1
AcAcAaB
 2  2  2
```





(1) FIS's scenario (2) rv-scenario (3) srv-scenario

Srv-scenario operations:



(4)

Contents

Agapia

- Generalities
- A glimpse on AGAPIA programming
- Finite interactive systems $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- *Compiling srv-programs*
- Floyd-Hoare logics for (s)rv-programs

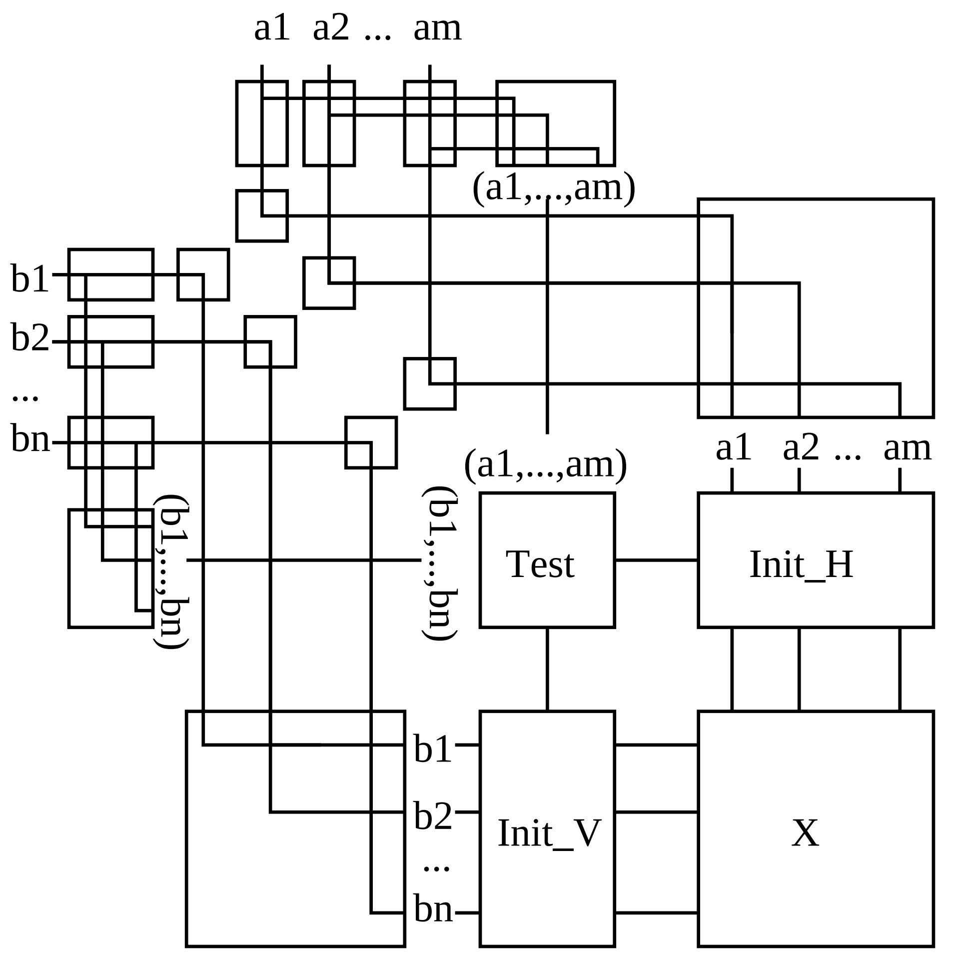
Compiling srv-programs

Implementation: Currently, *we have*

- a simulator for running rv-programs
- a translation from srv- to rv-programs and o proof of its correctness
- a mechanical procedure based on the above translation
- a typing procedure
- a partial implementation in K (a programming framework developed at UIUC)

..Compiling srv-programs

Example: The translation of *if* is based on the following component



whose implementation as a rv-program is rather tedious.

..Compiling srv-programs

A much more challenging task:

- extend assembly language like MIPS with interactive features (voices)
- design interactive processors *TRIPS architectures*
- use such a setting as the target for compiling high-level interactive programming languages including features from AGAPIA

Intermediary step: Add srv-programming features to certain mature programming languages as Eifel, Real Time Java, etc.

Contents

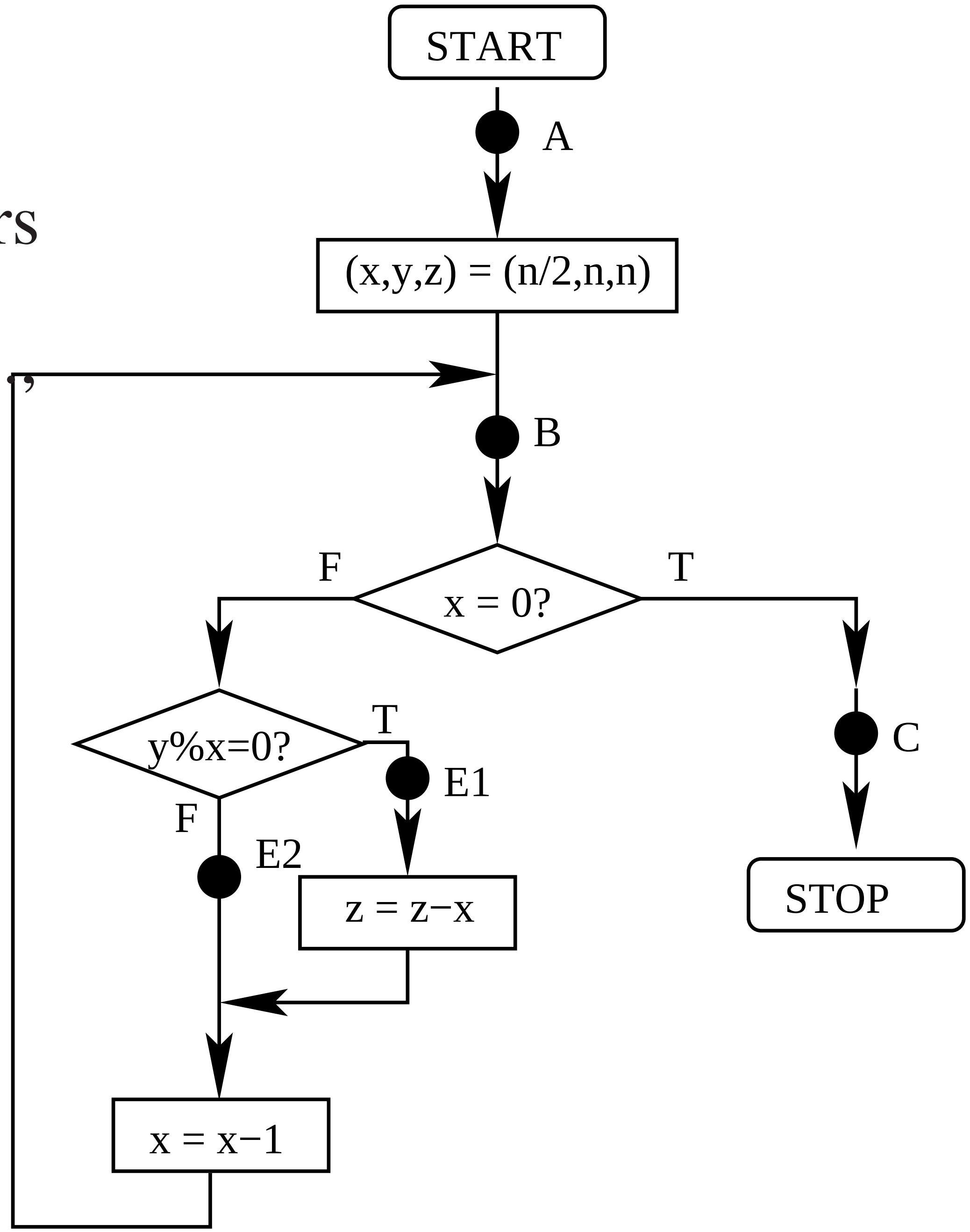
Agapia

- Generalities
- A glimpse on AGAPIA programming
- Finite interactive systems $\leftarrow$ *[nfa]*
- Rv-programs $\leftarrow$ *[flowchart programs]*
- Structured rv-programs $\leftarrow$ *[while programs]*
- Compiling srv-programs
- *Floyd-Hoare logics for (s)rv-programs*

Floyd's method for flowchart programs

Floyd's method for flowcharts:

- a program for perfect numbers
- *cut-points* and *assertions*, e.g.,
  ϕ_B : "$0 \leq x \wedge y = n \geq 2$
  $\wedge z = n - \sum_{d|n, x<d<n} d$"
- *invariance conditions*, e.g.,
  $\phi_B \wedge C_{p(B,E1,B)} \Rightarrow \sigma_2(\phi_B)$
- *termination*: no infinite computation

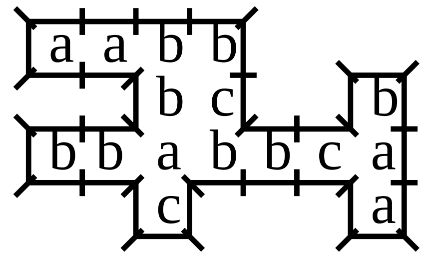


Grids and scenarios

Grids:

```
aabbabb
abbcdbb
bbabbca
ccccaaa
```

(a)



(b)

$$\begin{tikzcd} a \arrow[r] \arrow[d] & b \arrow[r] \arrow[d] & c \arrow[r] \arrow[d] & d \arrow[d] \\ e \arrow[r] & f \arrow[r] & g \arrow[r] & h \end{tikzcd}$$

(c)

Standard interpretation:

- *columns* - processes
- *rows* - process interactions (nonblocking message passing)
- left-to-rigth and top-to-bottom causality

Contour-and-contents representation of grids:

The grid in (b) is represented as:

- Contour: $e^4s^2e^2n^1e^1s^3w^1n^1w^3s^1w^1n^1w^2n^1e^2n^1w^2n^1$
- Contents: $a^2b^3cb^3ab^2caca$.

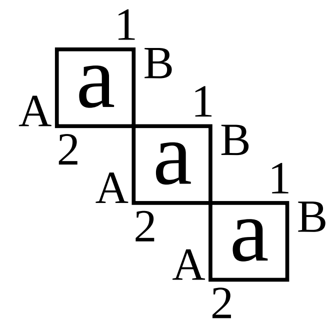
..Grids and scenarios

Scenarios:

(a)

```
 1 1 1
AaBbBbB
 2 1 1
AcAaBbB
 2 2 1
AcAcAaB
 2 2 2
```

(b)



Scenario = Grid + Data [around its letters]

Contour-and-contents representation of scenarios:

The scenario in (b) is represented as:

- Contour: $e_1s_Be_1s_Be_1s_Bw_2n_Aw_2n_Aw_2n_A$
  (or, shortly, $(e_1s_B)^3(w_2n_A)^3$)
- Contents: aaa.

Verification of rv-programs

A framework for rv-program verification:

Three steps:

- find an appropriate set of *contours* and *assertions* (it should be a *finite* and *complete* set);

  [complete = all scenarious of the associated FIS may be decomposed into such contours]
- fill in the contours with all *possible scenarios*; and
- prove the *invariance condition*, i.e., these scenarios respect the border assertions.

Except for the guess of assertions, the proof is finite and fully automatic.

Verification of structured rv-programs

Hoare logics for *structured rv-programs*:

- it has been *partially developed*
- it was used to verify the *correctness of the termination detection protocol*
- its rules are *sound*, but we have *no claim on thier completeness*...

Thank you !

(end)

References

RV-Systems and Agapia Programming:

- Link UIUC

  http://fsl.cs.uiuc.edu/index.php/RV_Systems_and_Agapia_Programming

- A few references:

– G. Stefanescu. Interactive Systems with Registers and Voices. Fundamenta Informaticae, 73(1-2): 285-305 (2006).

– Dragoi, C., and G. Stefanescu. Implementation and verification of ring termination detection protocols using structured rv-programs. Annals of University of Bucharest, Mathematics-Informatics Series, 55(2006), 129-138.

– G. Stefanescu. Towards a Floyd logic for interactive rv-systems. In: Proc. 2nd IEEE Conference on Intelligent Computer Communication and Processing (Ed. A.I. Letia). Technical University of Cluj-Napoca, September 1-2, 2006, 169-178.

– Alexandru Popa, Alexandru Sofronia, Gheorghe Stefanescu: High-level Structured Interactive Programs with Registers and Voices. J. UCS 13(11)(2007): 1722-1754.

– Cezara Dragoi, Gheorghe Stefanescu: On Compiling Structured Interactive Programs with Registers and Voices. SOFSEM 2008, LNCS 4910, Springer, 2008: 259-270.

References

(cont.)

- Cezara Dragoi, Gheorghe Stefanescu: AGAPIA v0.1: A Programming Language for Interactive Systems and Its Typing System. Electr. Notes Theor. Comput. Sci. 203(3): 69-94 (2008).
- Cezara Dragoi, Gheorghe Stefanescu: A sound spatio-temporal Hoare logic for the verification of structured interactive programs with registers and voices. WADT'08. Also in: CoRR abs/0810.3332: (2008).
- Gheorghe Stefanescu and Camelia Chira, "New parallel programming language design: a bridge between brain models and multi-core/many-core computers?", in "From Natural Language to Soft Computing: New Paradigms in Artificial Intelligence", L.A. Zadeh et.al (Eds.), Editing House of the Romanian Academy, 2008, pg. 196-210. Also in: CoRR abs/0812.2926: (2008)
- Alexandru Sofronia, Alexandru Popa, and Gheorghe Stefanescu, "Undecidability Results for Finite Interactive Systems", Romanian Journal of Information Science and Technology, Vol. 12, no. 2, 2009 pg. 265-279.