Computing in the Mist:
Writing Applications for Unknown Machines

Thilo Kielmann

VU University, Amsterdam

kielmann@cs.vu.nl

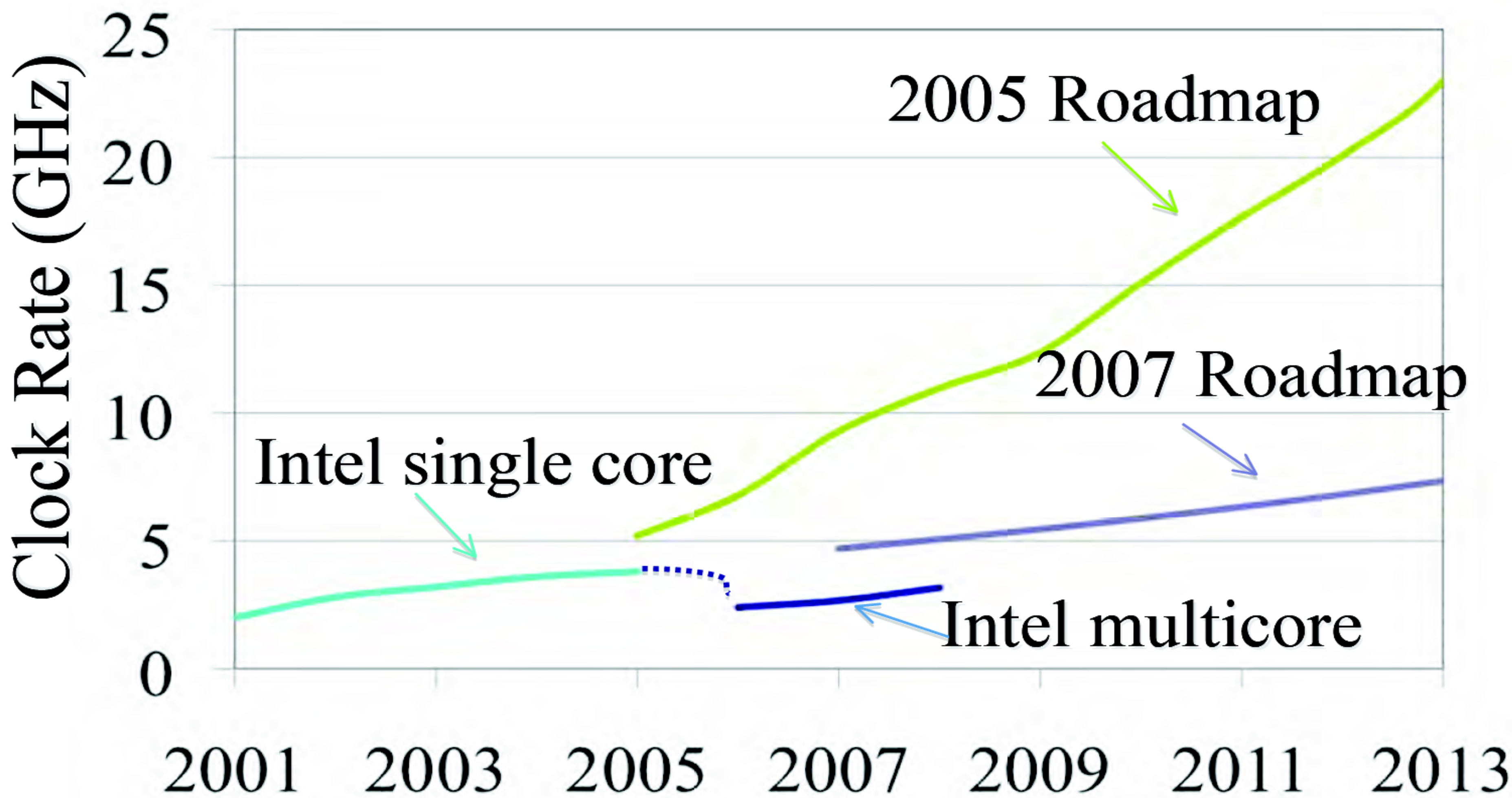
VU university amsterdam


THE MIST
NOVEMBER 21
THE WEINSTEIN COMPANY

- How many cores does your computer have?
- Where is your data?
- What has happened to job number 2342?

ITRS Roadmap 2005 and 2007

VU university amsterdam



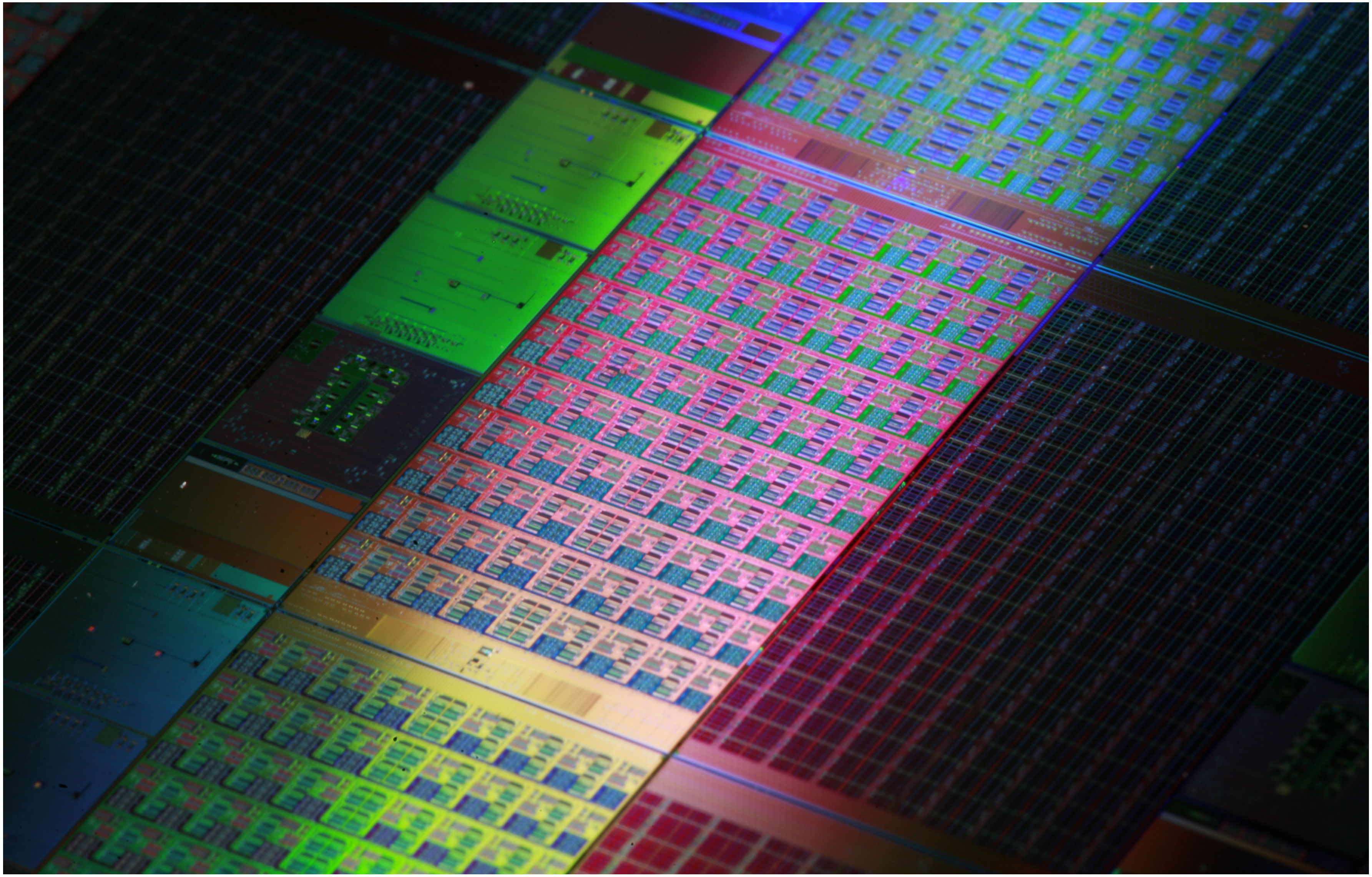
[D. Patterson, USENIX 2008 keynote]

A Fundamental Technology Change

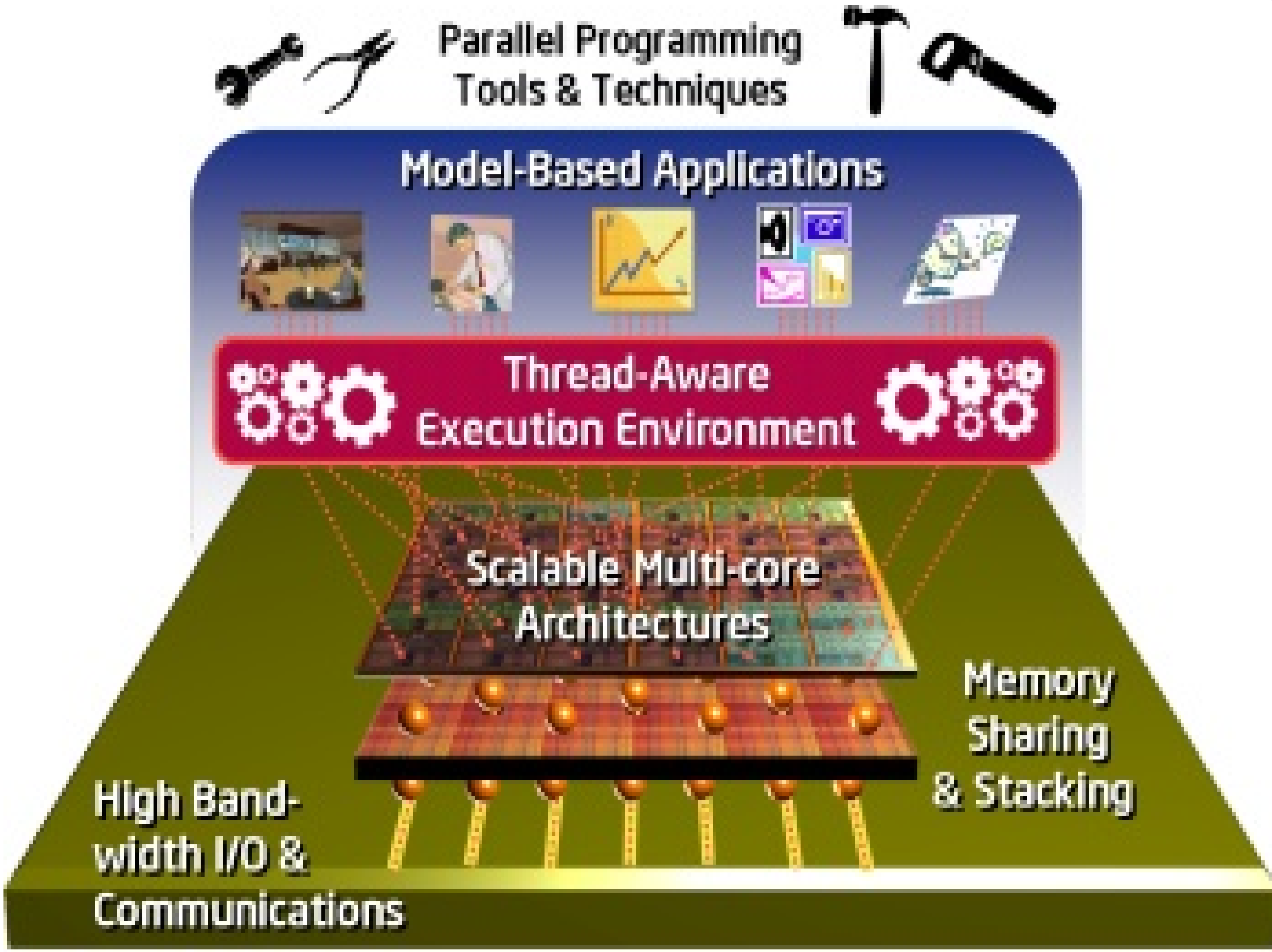
- CPU's will get faster only marginally:
  - limits of integration density
  - energy consumption (proportional to clock rate)
- If you want a faster computer, you need to use multiple CPU's:
  - In the past, the clock rate has doubled every 18 months
  - In the future, the number of cores will double every 18-24 months
- All programs must be parallel to use this new hardware!

Intel's 80-Core Teraflop Chip Prototype

VU university amsterdam

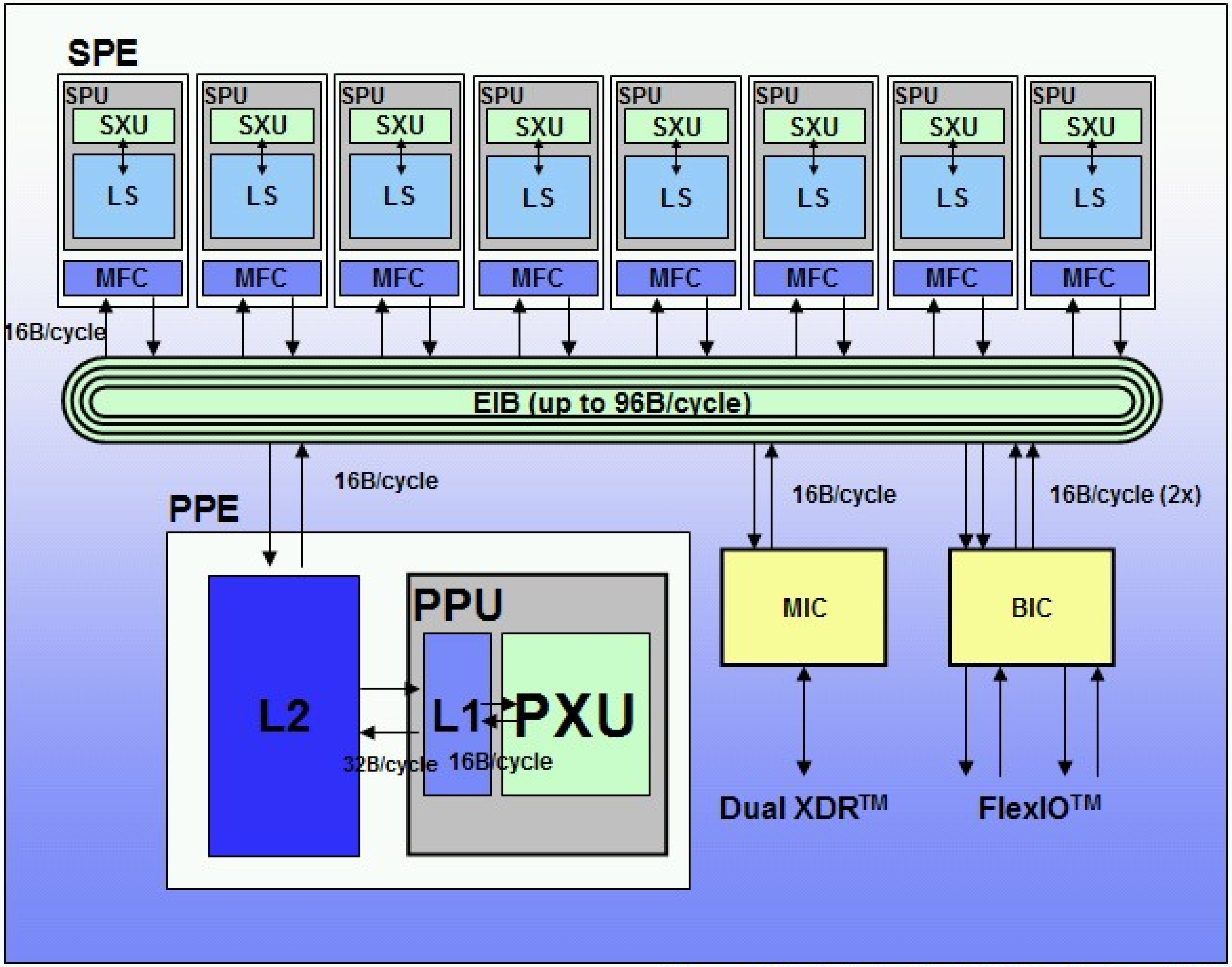


Intel's Tera Scale Vision



IBM's Cell: Heterogeneous Ensemble

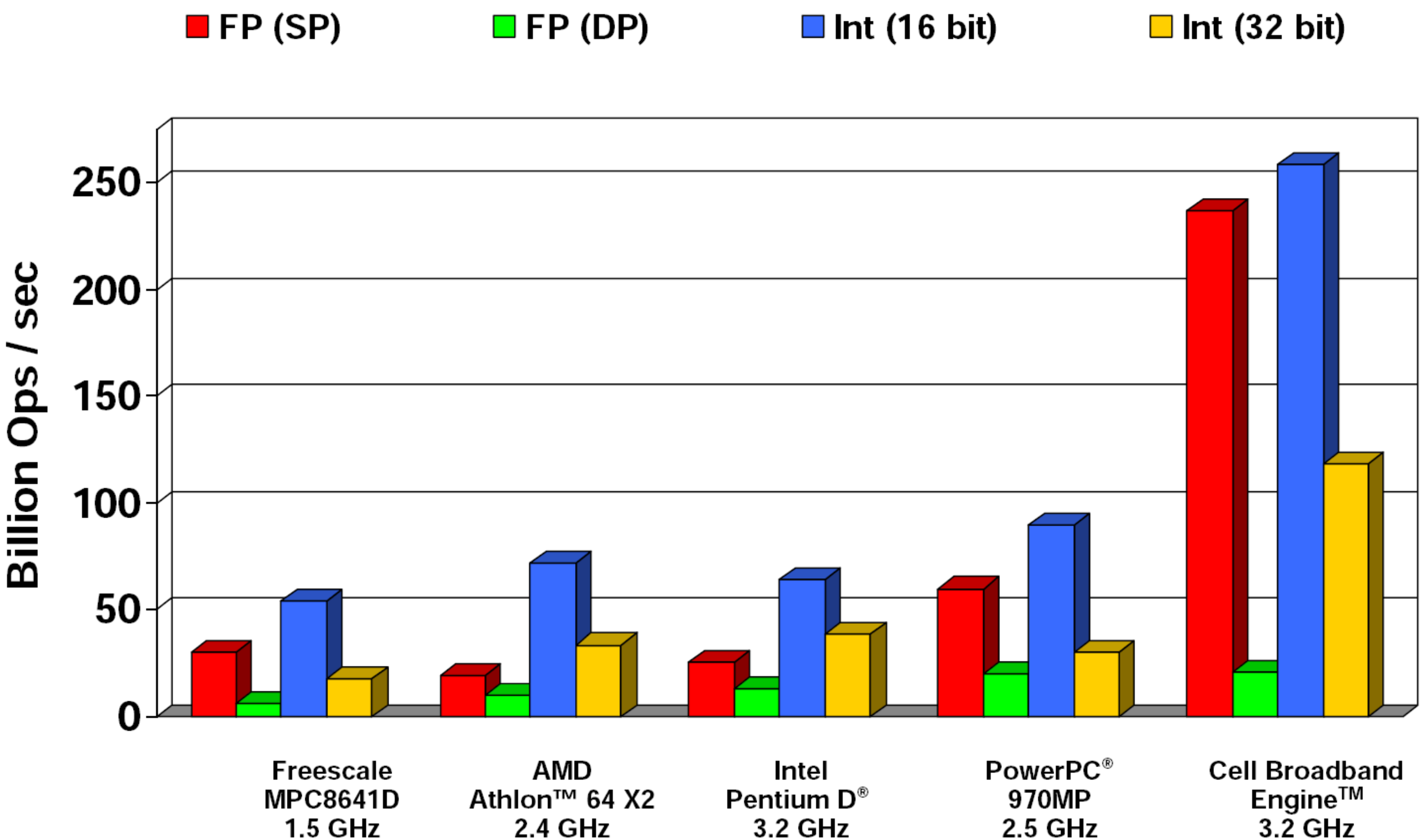
VU university amsterdam



Source: M. Gschwind et al., Hot Chips-17, August 2005

Cell Performance Compared

VU university amsterdam

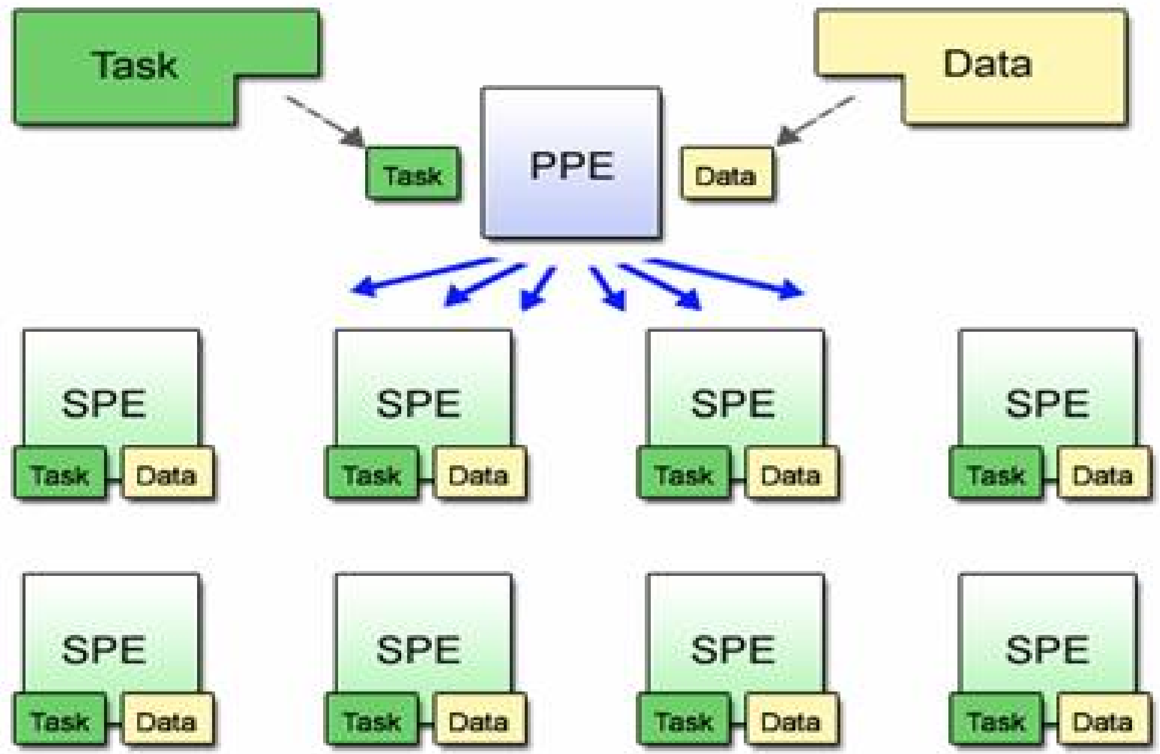


[Dr. Michael Perrone, IBM]

Cell Programming: Master/Worker

VU university amsterdam

- PPE executes main program
- SPE's execute sub tasks and return results
- all communication between PPE and SPE's must be programmed explicitly

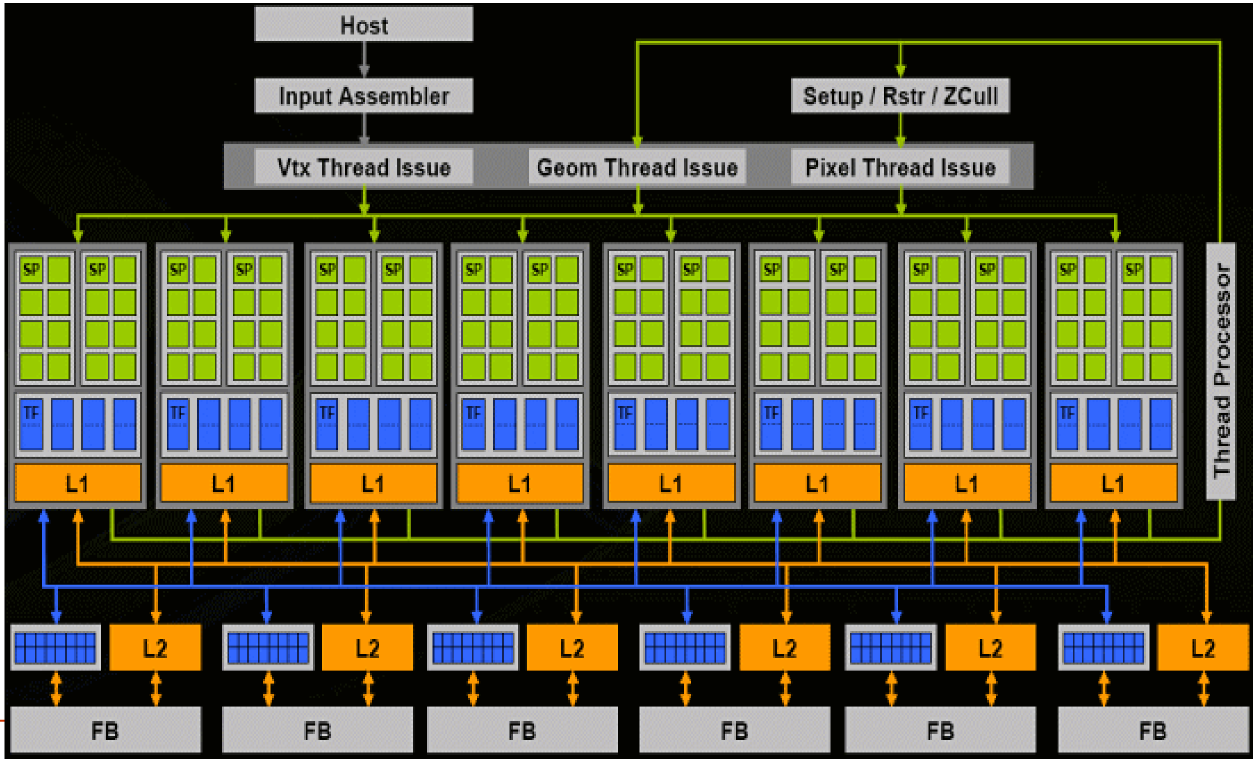


[Cell Programming Primer, 2006 Sony]

Graphics Processing Units (GPU's)

VU university amsterdam

- Example: Nvidia's GeForce8800
  - 8 x 16 blocks of stream processors
  - separate thread schedulers
  - crossbar-like access to graphics memory



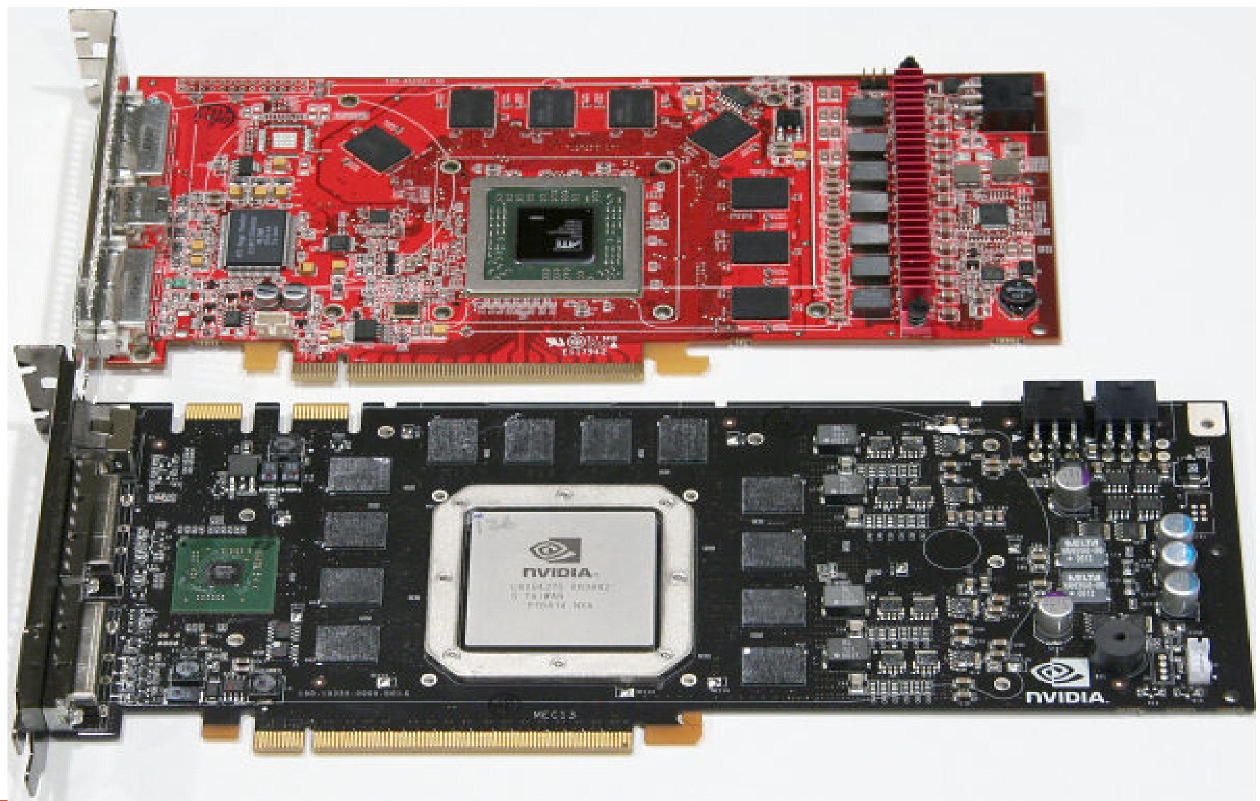
GPU vs. CPU

VU university amsterdam

- Nvidia GeForce 8800
  - clock speed 1.35 GHz
  - 681 million transistors
- Intel Pentium 4
  - clock speed 2.4 GHz
  - 55 million transistors

[Schellmann et.al., Euro-PAR 2008]:

2 Nvidia GeForce 8800GTX as fast as 16 Intel Xeon 3.2GHz



General Purpose GPU Programming

VU university amsterdam

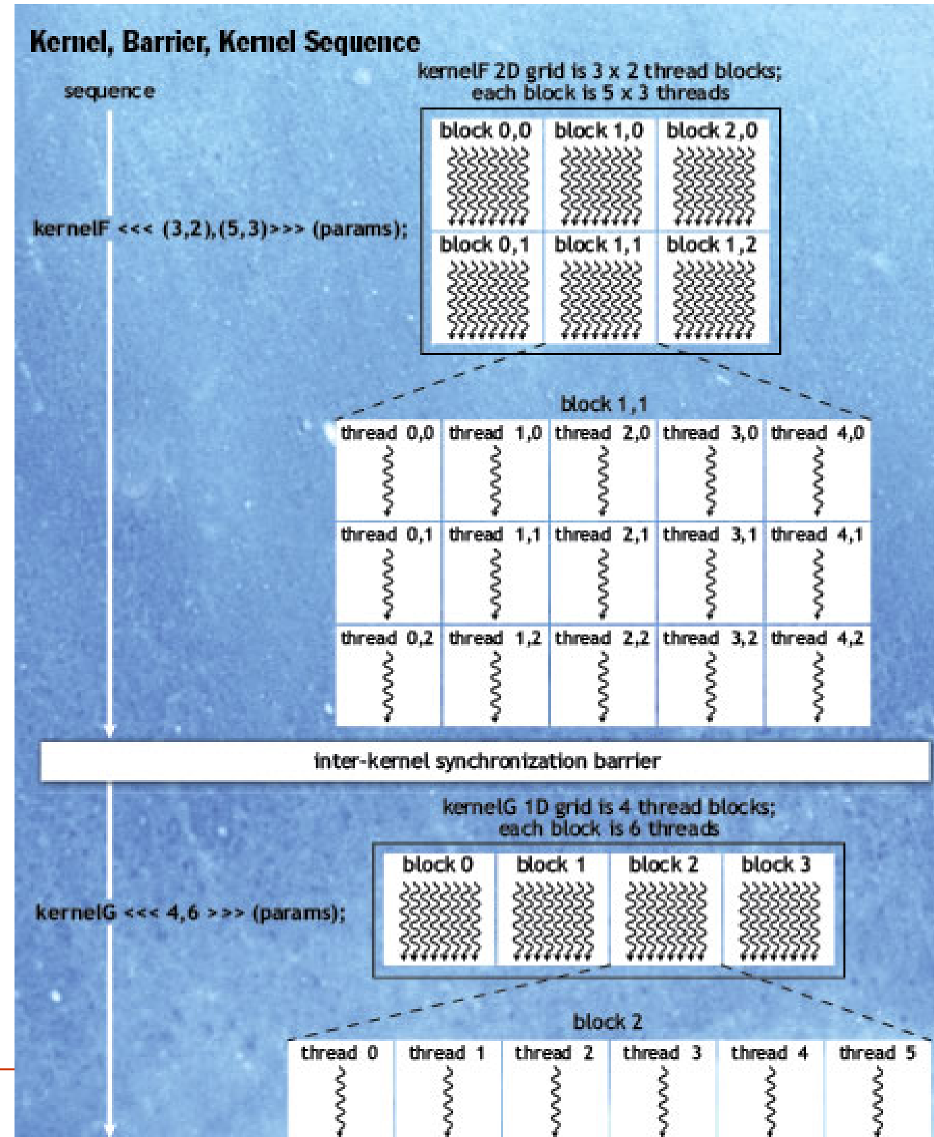
Nvidia's CUDA

Computing y ← ax + y with a Serial Loop

```
void saxpy_serial(int n, float alpha, float *x, float *y)
{
    for(int i = 0; i<n; ++i)
        y[i] = alpha*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Computing y ← ax + y in parallel using CUDA

```
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;

    if( i<n )  y[i] = alpha*x[i] + y[i];
}

// Invoke parallel SAXPY kernel (256 threads per block)
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```



Clusters: Beyond Single Computers

VU university amsterdam

DAS3, VU Amsterdam:

Programming for Clusters

VU university amsterdam

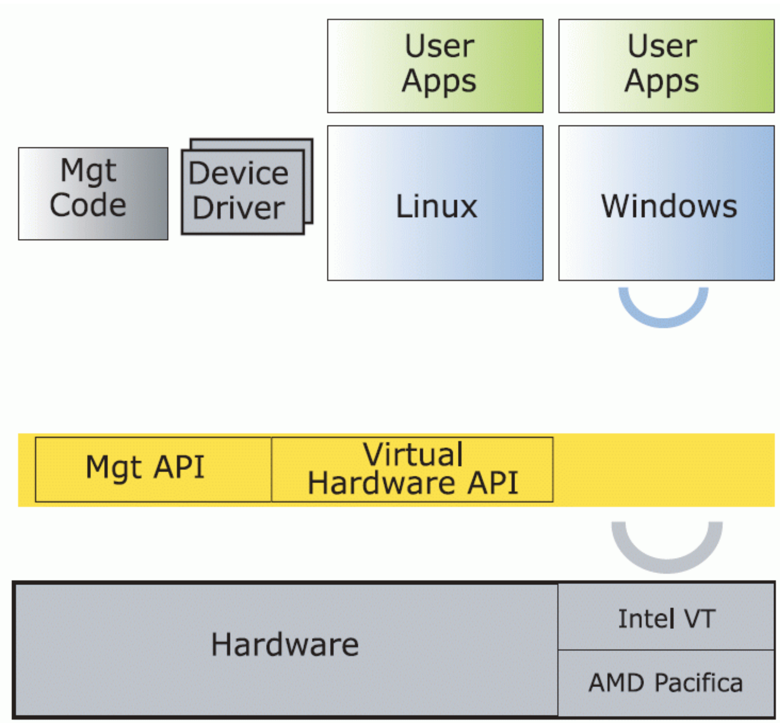
- Distributed memory, high-speed networks
- commonly: message passing (MPI), C and Fortran
- academic: Java remote method invocation (Ibis)

- deployment via shared file system and batch queue scheduling

Clouds: Data Centers + Virtualization

VU university amsterdam

(OK, a bit too simplified...)



Cloud Computing: Platform as a Service (PaaS)

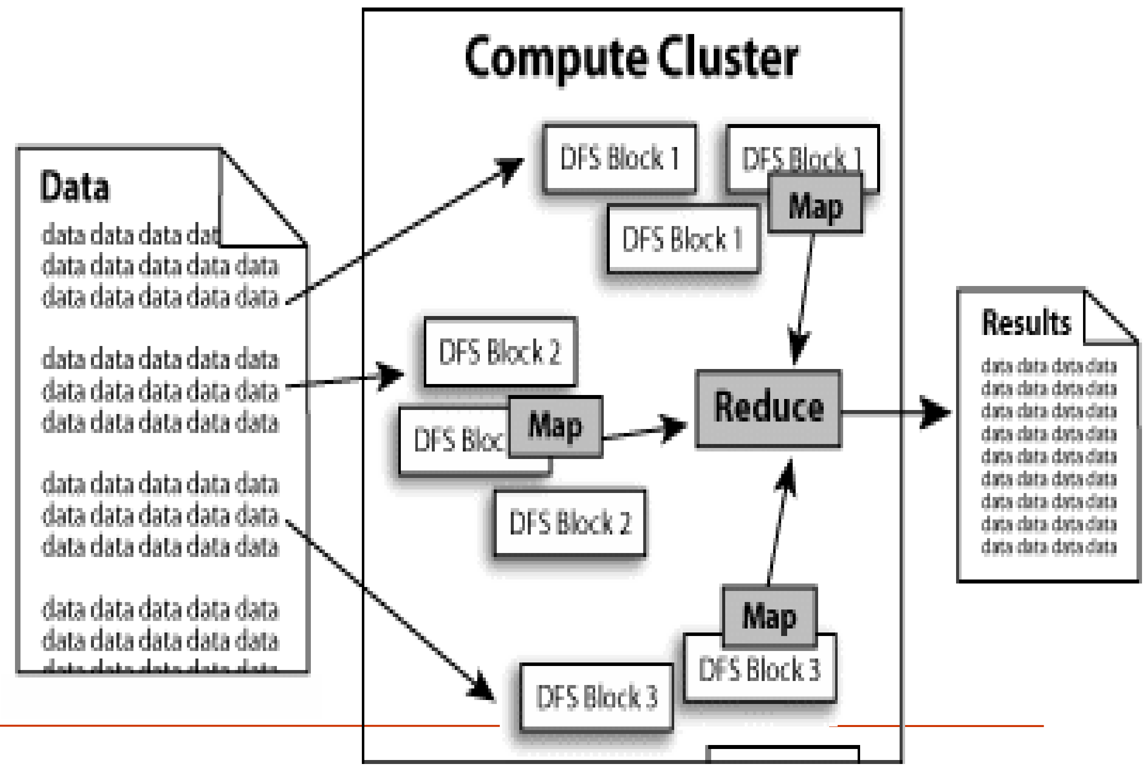
VU university amsterdam

- Amazon Web Services:
  - Elastic Compute Cloud (EC2)
    - allows to dynamically create/remove virtual machines with user-defined image (OS + application)
    - payment for CPU per hour
  - Simple storage Service (S3)
    - provides persistent object storage, write-once objects
    - payment for storage volume and transfer volume
- Highly dynamic service provider for compute and storage capacities

Programming Clouds: Hadoop

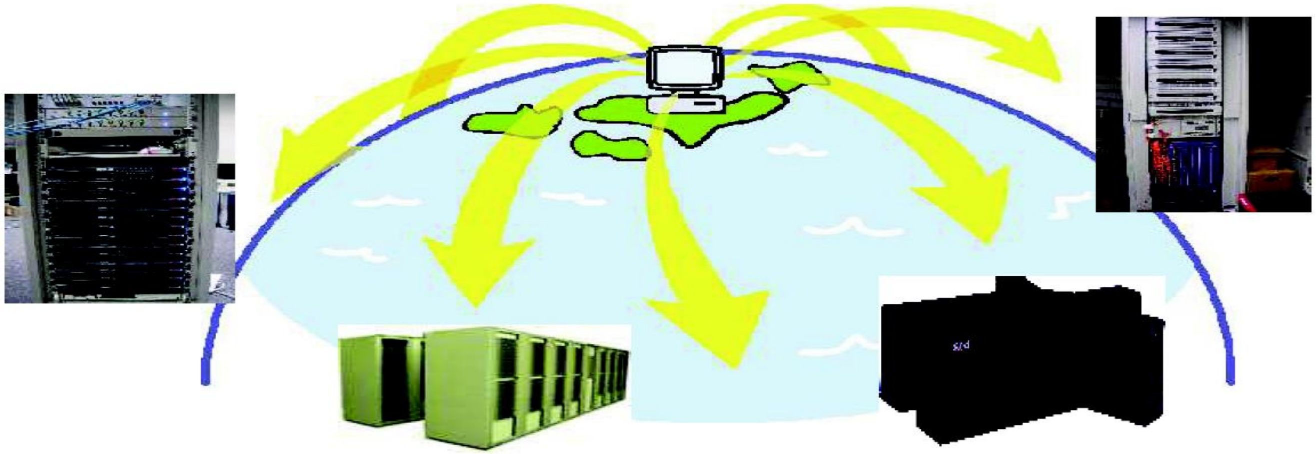
VU university amsterdam

- implements the map-reduce paradigm
- allows processing of large data sets
- user defined map and reduce functions
- Hadoop distributed file system (HDFS) built on top of Amazon S3
- Got popular due to fault-tolerant, file-based implementation



Grid Computing

VU university amsterdam



Grid Programming: Globus GASS copies a file...

VU university amsterdam

```
int copy_file (char const* source,   char const* target)
{
globus_url_t                        source_url;
globus_io_handle_t                  dest_io_handle;
globus_ftp_client_operationattr_t   source_ftp_attr;
globus_result_t                     result;
globus_gass_transfer_requestattr_t  source_gass_attr;
globus_gass_copy_attr_t             source_gass_copy_attr;
globus_gass_copy_handle_t           gass_copy_handle;
globus_gass_copy_handleattr_t       gass_copy_handleattr;
globus_ftp_client_handleattr_t      ftp_handleattr;
globus_io_attr_t                    io_attr;
int                                 output_file = -1;

if ( globus_url_parse (source_URL, &source_url) != GLOBUS_SUCCESS ) {
  printf ("can not parse source_URL \"%s\"\n", source_URL);
  return (-1);
}

if ( source_url.scheme_type != GLOBUS_URL_SCHEME_GSIFTP  &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_FTP     &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_HTTP    &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_HTTPS   ) {
  printf ("can not copy from %s - wrong prot\n", source_URL);
  return (-1);
}
globus_gass_copy_handleattr_init  (&gass_copy_handleattr);
globus_gass_copy_attr_init        (&source_gass_copy_attr);

globus_ftp_client_handleattr_init (&ftp_handleattr);
globus_io_fileattr_init           (&io_attr);

globus_gass_copy_attr_set_io      (&source_gass_copy_attr, &io_attr);
                                   &io_attr);
globus_gass_copy_handleattr_set_ftp_attr
                                  (&gass_copy_handleattr,
                                   &ftp_handleattr);
globus_gass_copy_handle_init      (&gass_copy_handle,
                                   &gass_copy_handleattr);

if (source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ||
    source_url.scheme_type == GLOBUS_URL_SCHEME_FTP    ) {
  globus_ftp_client_operationattr_init (&source_ftp_attr);
  globus_gass_copy_attr_set_ftp (&source_gass_copy_attr,
                                 &source_ftp_attr);
}
else {
  globus_gass_transfer_requestattr_init (&source_gass_attr,
                                         source_url.scheme);
  globus_gass_copy_attr_set_gass(&source_gass_copy_attr,
            &source_gass_attr);
}

output_file = globus_libc_open ((char*) target,
              O_WRONLY | O_TRUNC | O_CREAT,
              S_IRUSR  | S_IWUSR | S_IRGRP |
              S_IWGRP);
if ( output_file == -1 ) {
  printf ("could not open the file \"%s\"\n", target);
  return (-1);
}
/* convert stdout to be a globus_io_handle */
if ( globus_io_file_posix_convert (output_file, 0,
                                   &dest_io_handle)
     != GLOBUS_SUCCESS) {
  printf ("Error converting the file handle\n");
  return (-1);
}

result = globus_gass_copy_register_url_to_handle (
         &gass_copy_handle, (char*)source_URL,
         &source_gass_copy_attr, &dest_io_handle,
         my_callback, NULL);
if ( result != GLOBUS_SUCCESS ) {
  printf ("error: %s\n", globus_object_printable_to_string
          (globus_error_get (result)));
  return (-1);
}
globus_url_destroy  (&source_url);
return (0);
}
```

The Grid Application Toolkit (JavaGAT)

VU university amsterdam

HPC wire

The Leading Source for Global News and Information Covering the Ecosystem of High Productivity Computing / November 14, 2007

Home Page | Free Subscription | Advertising | About HPCwire

Features:

JavaGAT -- A Kindler Gentler Grid Interface

by Rob van Nieuwpoort and Thilo Kielmann

Vrije Universiteit, Amsterdam

[SC'07]

ACM TechNews

November 16, 2007

JavaGAT Example: Copy a File

High-level, uniform

```
import org.gridlab.gat.*;
import org.gridlab.gat.io.File;

public class Copy {
  public static void main(String[] args)
      throws Exception {
    GATContext context = new GATContext();
    URI source = new URI(args[0]);
    URI dest = new URI(args[1]);
    // Create a GAT File object
    File file = GAT.createFile(context, source);
    file.copy(dest); // The actual file copy.
    GAT.end(); // Shutdown the JavaGAT.
  }
}
```

- Provides the high level abstraction, that application programmers need; will work across different systems
- Shields gory details of lower-level middleware system

The Simple API for Grid Applications (SAGA): Towards a Standard

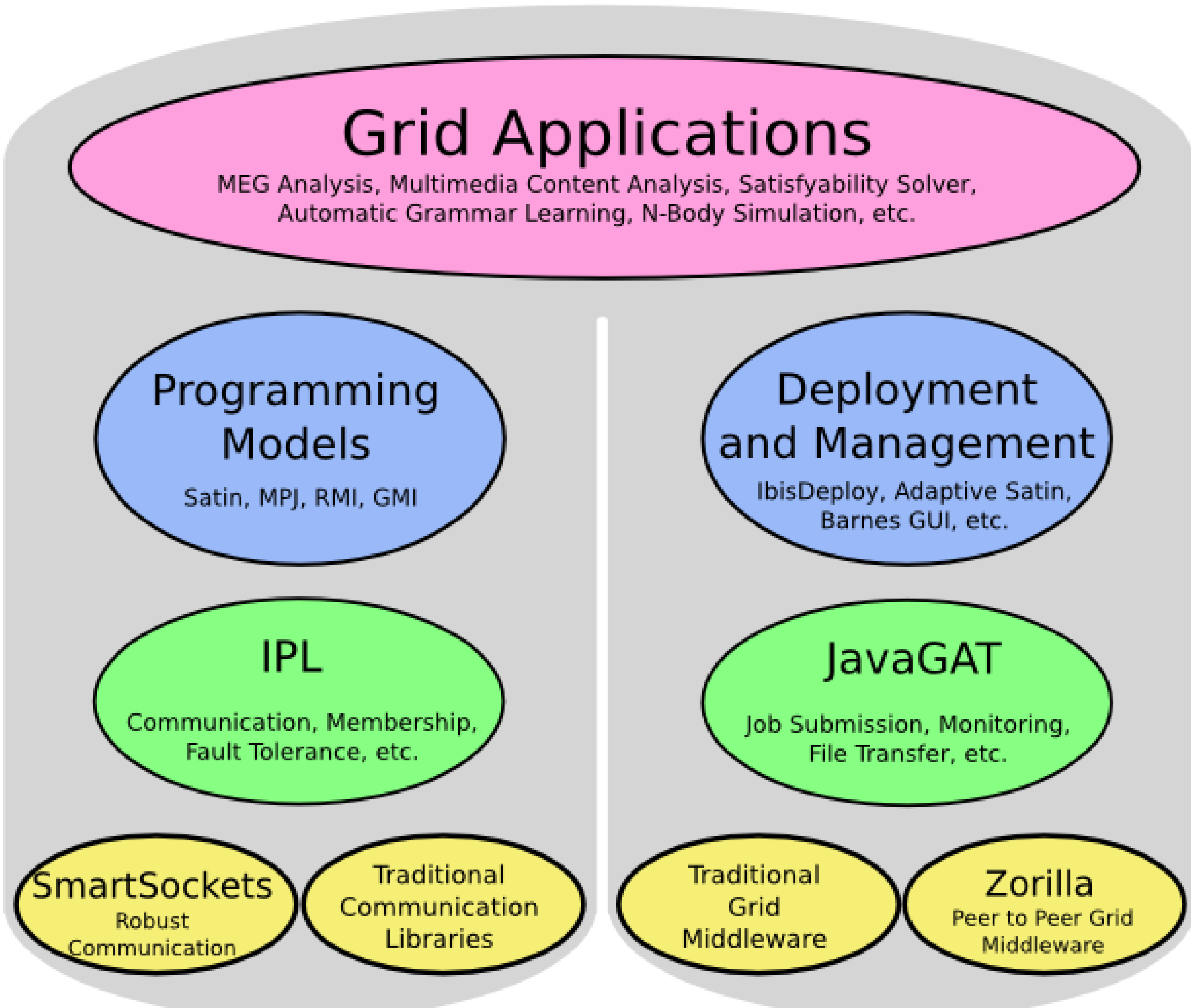
VU university amsterdam

- The need for a standard programming interface
  - Projects keep reinventing the wheel again, yet again, and again
  - MPI as a useful analogy of community standard
  - OGF as the natural choice; established the SAGA-RG
- Community process
  - Design and requirements derived from 23 use cases
  - SAGA Design Team (OGF, Berkeley, VU, LSU, NEC)

OpenGridForum
OPEN FORUM | OPEN STANDARDS

Ibis: Grid Programming and Deployment Simplified

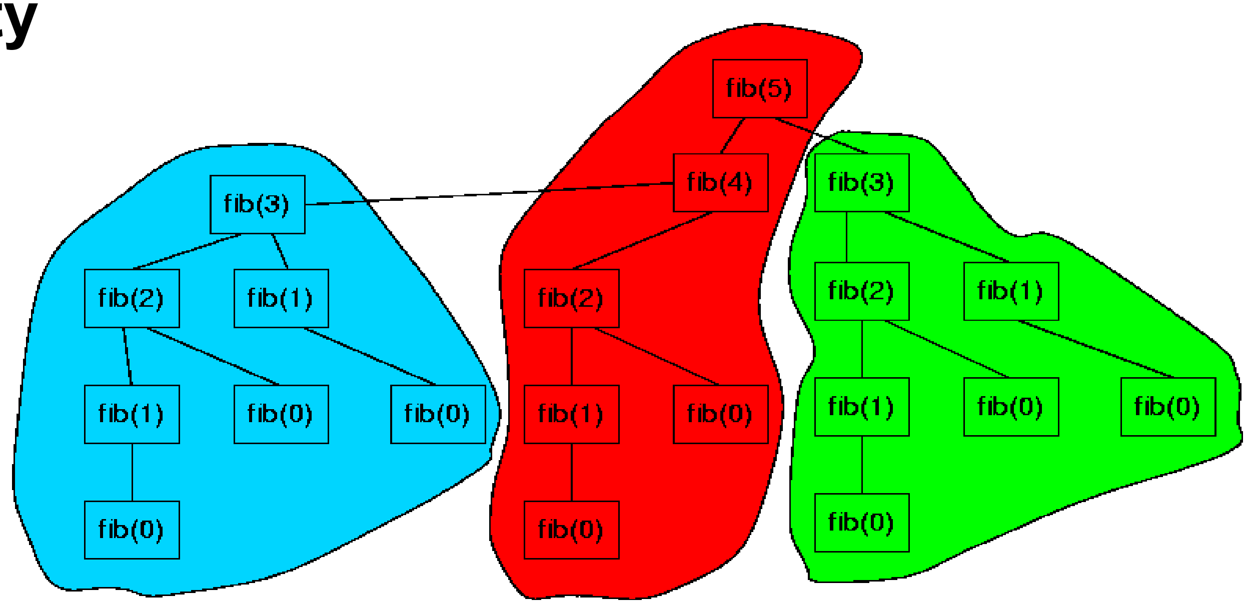
VU university amsterdam



Satin: Divide-and-Conquer for the Grid

VU university amsterdam

- Effective paradigm for Grid applications (hierarchical)
- Satin: Grid-aware load balancing (work stealing)
- Also support for
  - Fault tolerance
  - Malleability
  - Migration



Parallel Sudoku Solver with ADLB

1	2				9			7
		3				6	1	
				7		8		
					5	3		
7		9	1		8	2		6
		5	6					
		1		9				
	6	7				1		
2			5				3	8

Work-package =
partially completed "board"

```
Program:
  if (rank = 0)
        ADLB_Put initial board

  ADLB_Get board
  while success   (else done)
     ooh
     find first blank square
     if failure   (problem solved!)
        print solution
        ADLB_Set_Done
     else
        for each valid value
           set blank square to value
           ADLB_Put new board
  end while ☐
```

More: Master-Worker Parallelism

- "Embarrassingly parallel" problems
- minimal communication
- no dependence on numbers/types of computers

- Popular e.g. by Seti-at-home, BOINC, etc.
- only(?) applicable to very simple problems

More: Parallel Skeletons / Higher-Order Components

VU university amsterdam

- **Abstract parallelism and communication from the application logic**
- **Highly useful approach to implement adaptive (autonomic) applications**

- **(If this all reminds you of what you were doing while you were still young, you are getting my point...)**

Challenges of Near-future Platforms

- Scalability
  - applications have to run on widely different numbers of CPU's
  - if your program can not use twice the number of CPU's, you won't be able to utilize next year's computer
- Heterogeneity
  - applications will have to run on many core and multi core, and special-purpose CPU's (like Cell and GPU's)
  - think of clusters of multi core, clusters of Cell's, clusters of clusters of...

Challenges (2)

VU university amsterdam

- **Performance portability**
  - applications must run *efficiently* on different types of machines (one of the hard problems of parallel computing)
  - I mean, both on Tsubame with GPGPU's *and* on Roadrunner with Cell's...
- **Malleability**
  - applications must be able to run with changing numbers of processors, at run time
    - adapt to changing environments
- **Fault tolerance**
  - simple statistics: with a large number of parts involved, failure probability raises towards 1

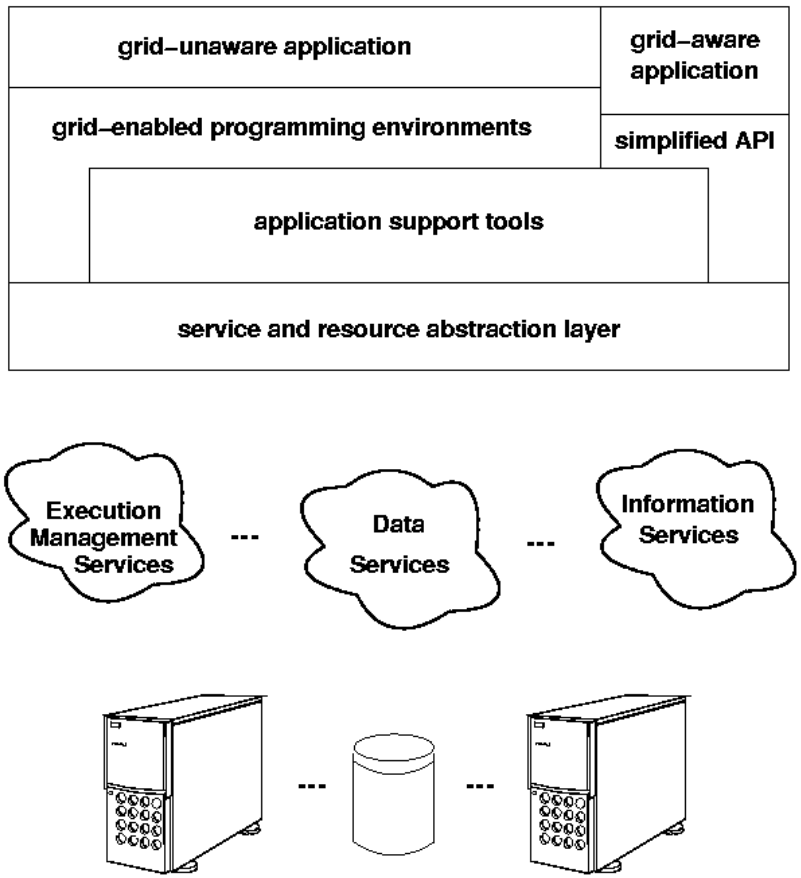
Programming for the Mist of Architectures

VU university amsterdam

- Lots of heroic efforts squeezing out performance:
  - CUDA
  - Cell
  - Astron writing Assembler for the Blue Gene/L ...
- We are back to the (Transputer) times where codes were written for specific parallel machines
  - Not what we want (except for researching machines)
- The opposite on clouds and grids: (lazy guys)
  - Map/Reduce and Hadoop abstract from machine
  - but add fault tolerance and malleability

Approach seen by CoreGRID folks

VU university amsterdam



Application + runtime env.

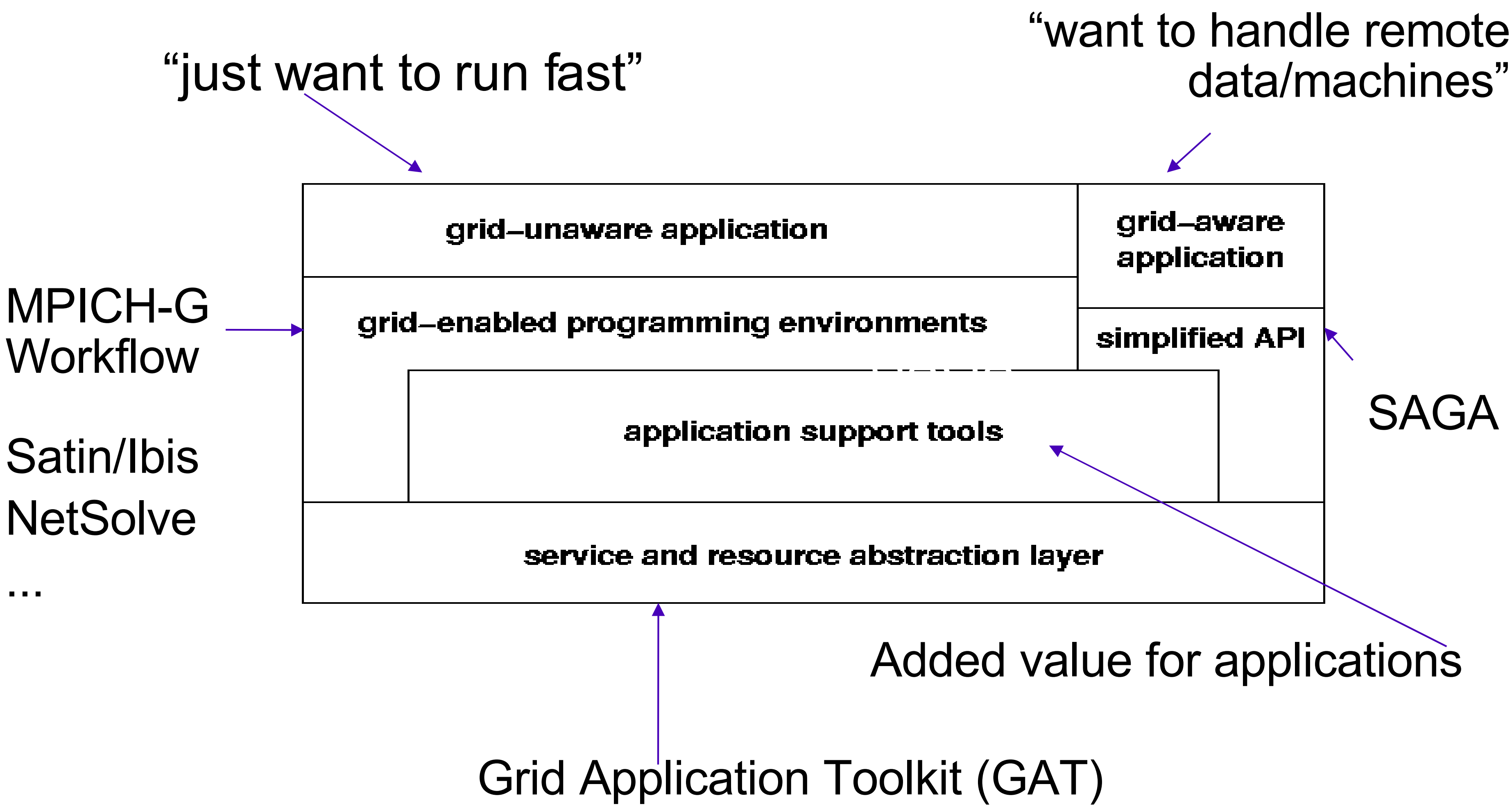
Middleware

Resources

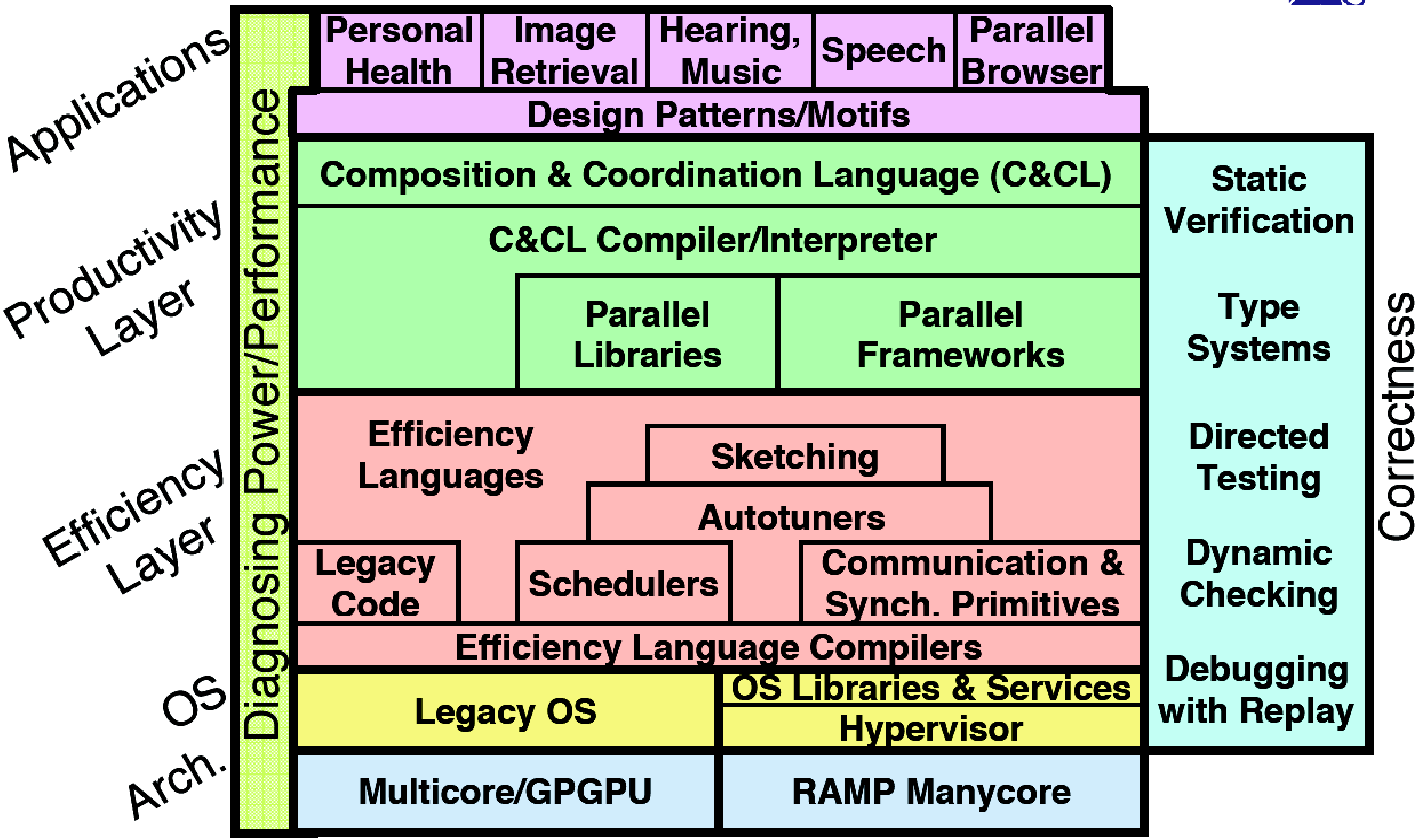
CoreGRID

Grid Application Runtime Stack

VU university amsterdam



Approach seen by Berkeley's PAR Lab



[D. Patterson, USENIX 2008 keynote]

Programming Models for the Mist

- We need a lower layer of efficiency primitives that handle certain platforms the respectively “best” way
  - this is for the CCI's and CCR's
  - users should not see this, only tool writers or compilers
- A higher “coordination” layer has to describe available concurrency, in a declarative manner (?)
- This means, we should reconsider the works from the 80s and 90s and see why they failed and what we could use today
  - map-reduce as the perfect example for an old idea, re-animated
  - There is hope for your Ph.D. work, after all ;-)

Summary / Conclusions

VU university amsterdam

- The future is parallel
- Parallel programming is hard
- This is a big chance for Computer Science to get it right, finally...
- My personal take:
  - The solution will be a combination of declarative parallelism, combined with *MUCH* systems work on getting the plumbing right
  - We might have to step back from getting the last bit of performance in favour of a more sustainable approach

Think different.