Advanced Programming and Execution Models
for
Future Multi-Core Systems

Hans P. Zima

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA
and
Institute of Computational Science, University of Vienna, Austria

High Performance Embedded Computing (HPEC) Workshop

MIT Lincoln Laboratory, 18-20 September 2007

Contents

1 Introduction

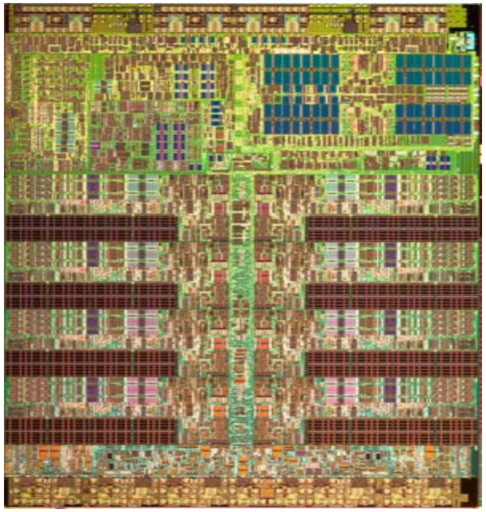
2 Towards High-Level Programming Models for Parallelism

3 Outline of a Generic Introspection Framework

4 Concluding Remarks

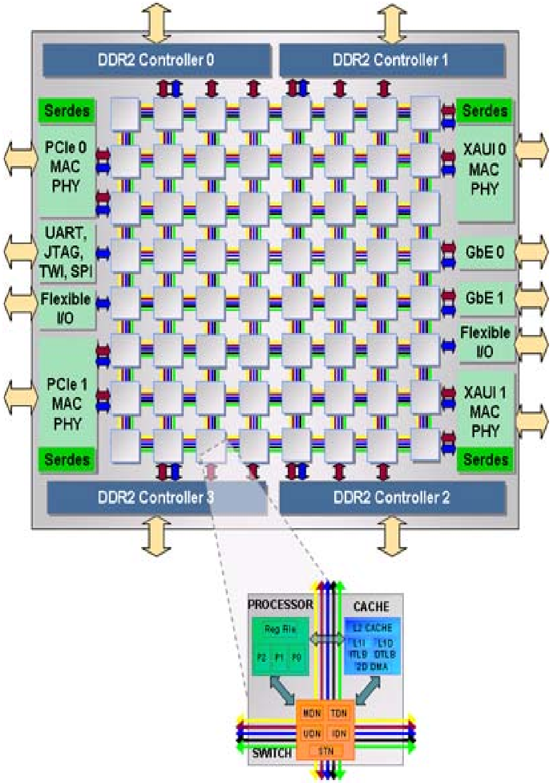
Multicore: An Emerging Technology

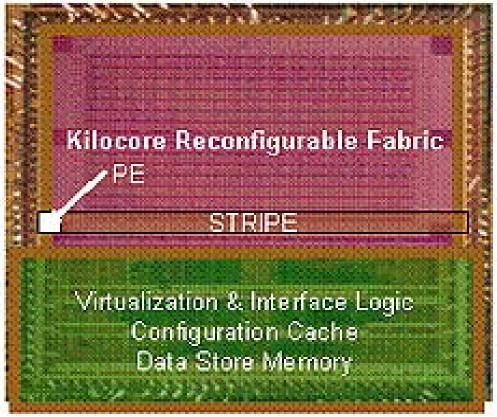
- **The Problem: CMOS manufacturing technology approaches physical limits**
  - *power wall, memory wall, ILP wall*
  - *Moore's Law still in force (number of transistors on a chip increasing)*
- **Solution: Multicore technology**
  - *improvements by multiple cores on a chip rather than higher frequency*
  - *on-chip resource sharing provides cost and performance benefits*
- **Multicore systems have been produced since 2000**
  - *IBM Power 4;Sun Niagara;AMD Opteron;Intel Xeon;…*
  - *Quadcore systems by AMD, Intel recently introduced*
  - *IBM/Sony/Toshiba: Cell Broadband Engine*
    - *Power Processor (PPE) and 8 Synergistic PEs (SPEs)*
    - *peak performance 230 GF (1 TF expected by 2010)*



Future Multicore Architectures: From 10s to 100s of Processors on a Chip

- **Tile64 (Tilera Corporation, 2007)**
  - *64 identical cores, arranged in an 8X8 grid*
  - *iMesh on-chip network, 27 Tb/sec bandwidth*
  - *170-300mW per core; 600 MHz – 1 GHz*
  - *192 GOPS (32 bit)*
- **Kilocore 1025 *(Rapport Inc. and IBM, 2008)***
  - *Power PC and1024 8-bit processing elements*
  - *125 MHz per processing element*
  - *32X32 “stripe” configuration*
  - *“stripes” dedicated to different tasks*
- **512-core SING chip (Alchip Technologies, 2008)**
  - *for GRAPE-DR, a Japanese supercomputer project expected to deliver 2PFLOPS in 2008*
- **80-core 1 TF research chip from Intel (2011)**

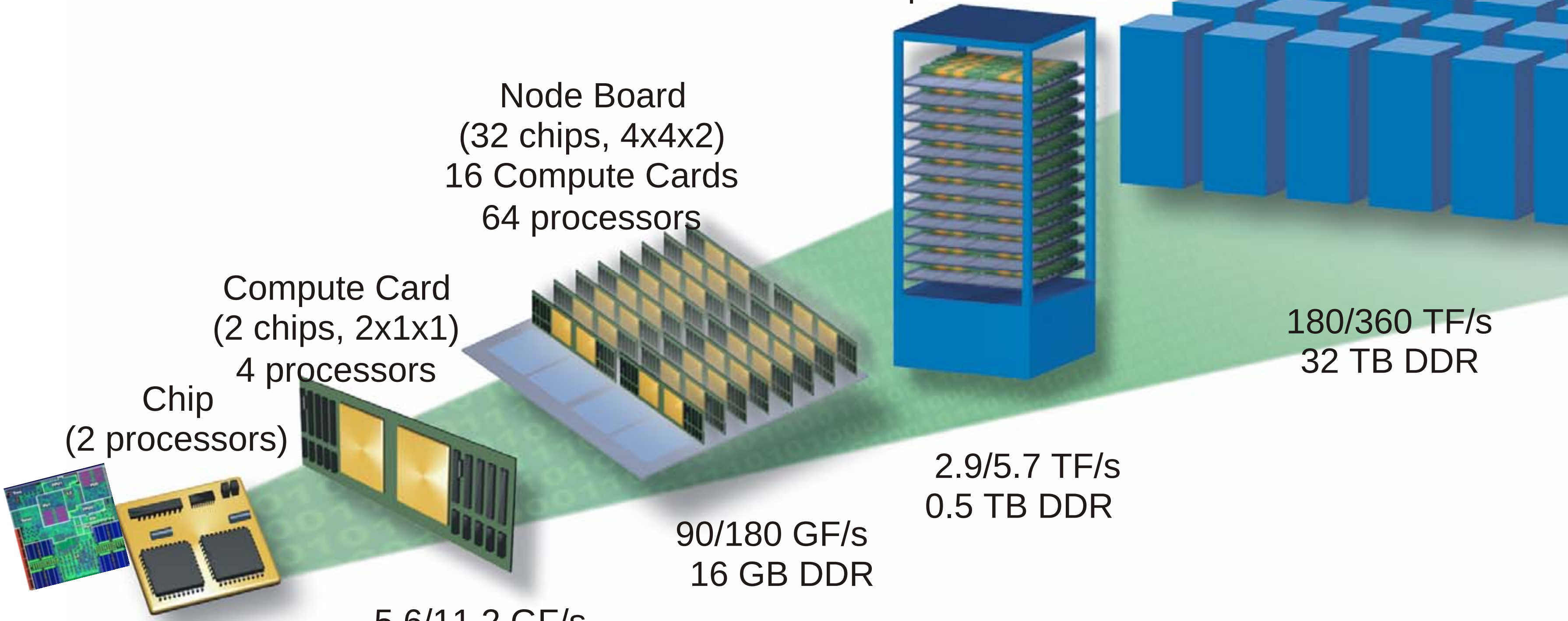



Kilocore Reconfigurable Fabric
PE
STRIPE
Virtualization & Interface Logic
Configuration Cache
Data Store Memory

HPC: Massive Parallelism Dominates the Path to Peta-Scale Machines

IBM BlueGene/L: 131,072 Processors 280 TF Linpack

Number 1 on TOP 500 List since 2006



System
(64 racks, 64x32x32)
131,072 processors

Rack
(32 Node boards, 8x8x16)
2048 processors

Node Board
(32 chips, 4x4x2)
16 Compute Cards
64 processors

Compute Card
(2 chips, 2x1x1)
4 processors

Chip
(2 processors)

180/360 TF/s
32 TB DDR

2.9/5.7 TF/s
0.5 TB DDR

90/180 GF/s
16 GB DDR

5.6/11.2 GF/s
1 GB DDR

2.8/5.6 GF/s
4 MB (cache)

Source: IBM Corporation

High Performance Computing and Embedded Computing: Common Issues

- High Performance Computing (HPC) and Embedded Computing (EC) have been traditionally at the extremes of the computational spectrum
- However, future HPC, EC, and HPEC systems will need to address many similar issues (at different scales):
  - *multicore as the underlying technology*
  - *massive parallelism at multiple levels*
  - *power consumption constraints*
  - *fault tolerance*
  - *high-productivity reusable software*

Software Issues for Future Parallel Systems

- **Provide high-productivity programming models and tools**
  - *support nested data and task parallelism*
  - *allow control of locality, power management, performance*
  - *provide intelligent tools for program development, debugging, tuning*
- **Address fault tolerance at multiple levels**
- **Exploit the abundance of low-cost processors for *introspection:***
  - *fault tolerance*
  - *performance tuning*
  - *power management*
  - *behavior analysis*

Can programming models for HPC provide a guideline?

Contents

1 Introduction

2 Towards High-Level Programming Models for Parallelism

3 Outline of a Generic Introspection Framework

4 Concluding Remarks

HPC Programming Paradigm: State-of-the-Art

The MPI Messing-Passing Model

- *a portable standard allowing full control of communication*
- *widely adopted as the dominating HPC programming paradigm*
- *A main reason for its success has been the capability to achieve performance on clusters and distributed-memory architectures*

Drawbacks of the MPI Model

- *wide gap between the scientific domain and the programming model*
- *conceptually simple problems can result in very complex programs; simple changes can require significant modifications of the source*
- *lack of separation between algorithm and communication management*

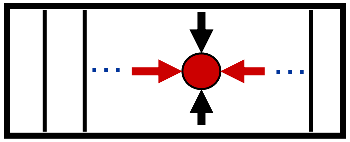
Higher-level Alternatives have been proposed since the 1990s

- *High Performance Fortran (HPF) language family, ZPL,…*
- *OpenMP*
- *PGAS Languages (CoArray Fortran, UPC, Titanium)*

The Key Idea of High Performance Fortran (HPF)

Message Passing Approach

local view of data, local control, explicit two-sided communication



HPF Approach

global view of data, global control, compiler-generated communication

initialize MPI

local **computation**

```
do while (.not. converged)
    do  J=1,M
        do  I=1,N
             B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                                  A(I,J-1)+A(I,J+1))
        end do
     end do
    A(1:N,1:N) = B(1:N,1:N)
```

```
 if (MOD(myrank,2) .eq. 1) then
    call MPI_SEND(B(1,1),N,…,myrank-1,..)
    call MPI_RCV(A(1,0),N,…,myrank-1,..)
    if (myrank .lt. s-1) then
        call MPI_SEND(B(1,M),N,…,myrank+1,..)
        call MPI_RCV(A(1,M+1),N,…,myrank+1,..)
    endif
                                   else  …
                    …
```

global **computation**

```
do while (.not. converged)
    do  J=1,N
        do  I=1,N
             B(I,J) = 0.25 * (A(I-1,J)+A(I+1,J)+
                                  A(I,J-1)+A(I,J+1))
        end do
     end do
    A(1:N,1:N) = B(1:N,1:N)
```

data distribution

```
processors  P(NUMBER_OF_PROCESSORS)
distribute(*,BLOCK) onto P :: A, B
```

communication compiler-generated

K. Kennedy, C. Koelbel, and H. Zima: The Rise and Fall of High Performance Fortran: An Historical Object Lesson

Proc. History of Programming Languages III (HOPL III), San Diego, June 2007

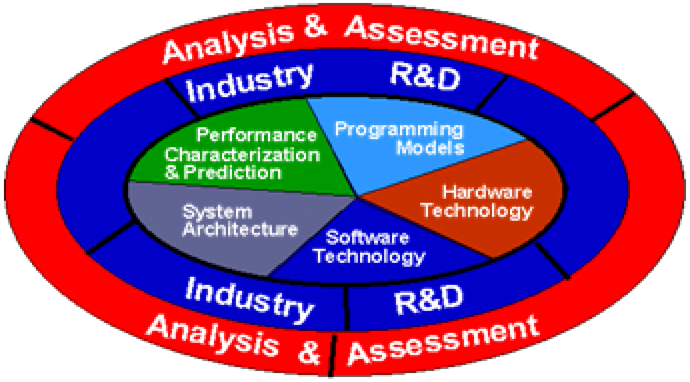
High Productivity Computing Systems

Goals:

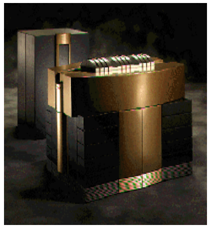
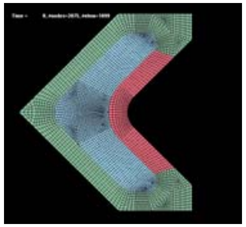
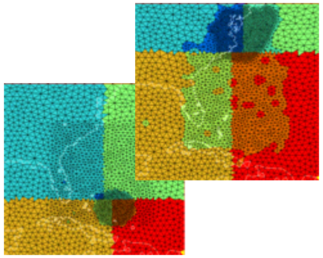
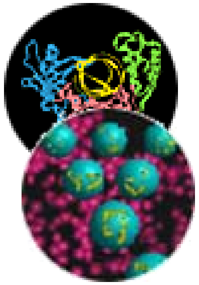
- ➢ ***Provide a new generation of economically viable high productivity computing systems for the national security and industrial user community (2007 – 2010)***

Impact:

- **Performance** (efficiency): critical national security applications by a factor of 10X to 40X
- **Productivity** (time-to-solution)
- **Portability** (transparency): insulate research and operational application software from system
- **Robustness** (reliability): apply all known techniques to **protect against outside attacks**, hardware faults, & programming errors



HPCS Program Focus Areas

Applications:

- Intelligence/surveillance, reconnaissance, cryptanalysis, airborne contaminant modeling and biotechnology

High Productivity Languages: Chapel (Cray), X10 (IBM), and Fortress (Sun)

Source: Bob Graybill (DARPA) et al.

Chapel – The Cascade HPCS Language Key Features

- Combination of ideas from HPF with modern language design concepts (OO, programming-in-the-large) and improved compilation technology
- Global view of data and computation
- Explicit specification of parallelism
  - *problem-oriented: forall, iterators, reductions*
  - *support for data and task parallelism*
- Explicit high-level specification of locality
  - *data distribution and alignment*
  - *data/thread affinity control*
  - *user-defined data distributions*
  - *NO explicit control of communication*

Chapel Webpage ***http://cs.chapel.washington.edu***

Example: Jacobi Relaxation in Chapel

```
const n= …, epsilon= …;
const DD: domain(2)      = [0..n+1, 0..n+1];
      D:  subdomain(DD) = [1..n, 1..n];
var delta: real;
var A, Temp: [DD] real;  /*array declarations over domain DD */

A(0,1..n) = 1.0;

do {
    forall (i,j) in D {  /* parallel iteration over domain D */
       Temp(i,j) = (A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))/4.0;
       delta = max reduce abs(A(D) – Temp(D));
       A(D) = Temp(D);
    } while (delta > epsilon);

writeln(A);
```

Example: Jacobi Relaxation in Chapel

```
const L:[1..p,1..q] locale = reshape(Locales);

const n= …, epsilon= …;
const DD:domain(2)distributed(block,block)on L=[0..n+1,0..n+1];
      D:  subdomain(DD) = [1..n, 1..n];
var delta: real;
var A, Temp: [DD] real; /*array declarations over domain DD */

A(0,1..n) = 1.0;

do {
    forall (i,j) in D {  /* parallel iteration over domain D */
       Temp(i,j) = (A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))/4.0;
       delta = max reduce abs(A(D) – Temp(D));
       A(D) = Temp(D);
    } while (delta > epsilon);

writeln(A);
```

Example: Jacobi Relaxation in Chapel

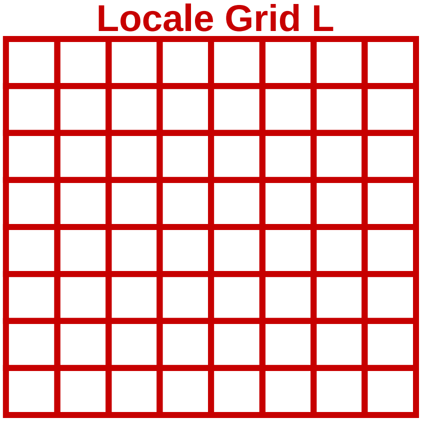
```
const L:[1..p,1..q] locale = reshape(Locales);

const n= …, epsilon= …;
const DD:domain(2)distributed(block,block)on L=[0..n+1,0..n+1];
      D:  subdomain(DD) = [1..n, 1..n];
var delta: real;
var A, Temp: [DD] real;

A(0,1..n) = 1.0;

do {
    forall (i,j) in D {
       Temp(i,j) = (A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))/4.0;
       delta = max reduce abs(A(D) – Temp(D));
       A(D) = Temp(D);
    } while (delta > epsilon);

writeln(A);
```



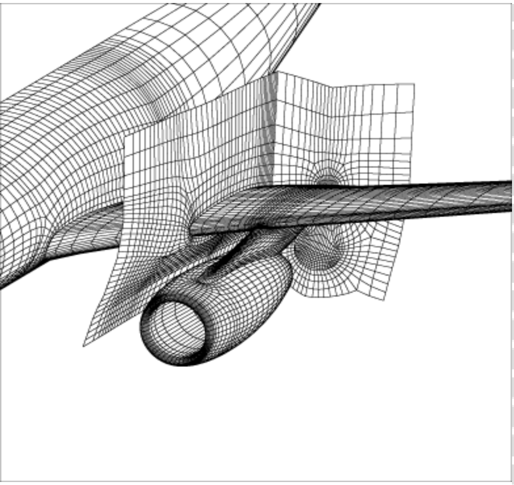
Key Features

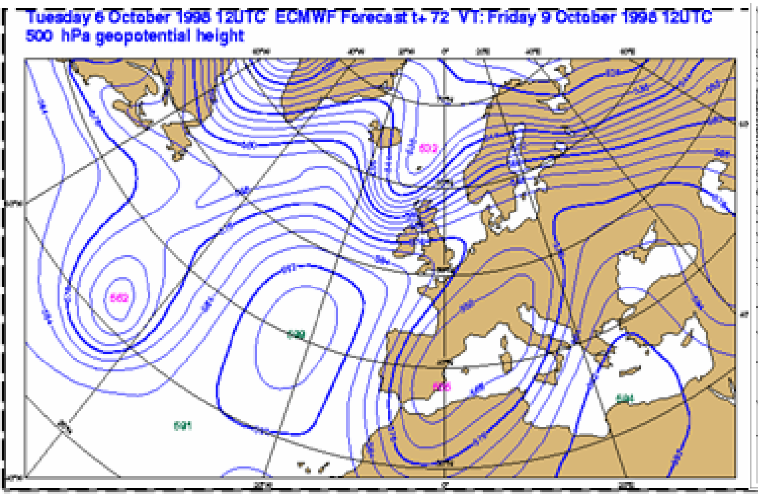
- global view of data/control
- explicit parallelism (forall)
- high-level locality control
- NO explicit communication
- NO local/remote distinction in source code

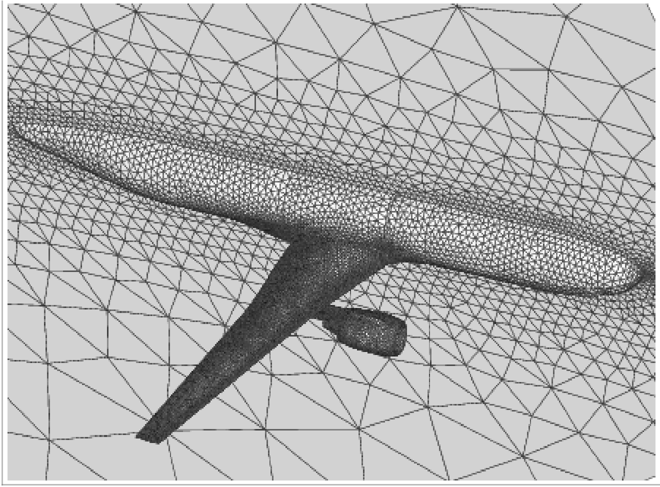
Chapel’s Framework for User-Defined Distributions

- **Complete flexibility for**
  - *distributing index sets across locales*
  - *arranging data within a locale*

- **Capability analogous to function specification**
  - *unstructured meshes*
  - *multi-block problems*
  - *multi-grid problems*
  - *distributed sparse matrices*




Tuesday 6 October 1998 12UTC ECMWF Forecast t+72 VT: Friday 9 October 1998 12UTC
500 hPa geopotential height



Example
Matrix-Vector Multiplication in Chapel

```
var A: [1..m,1..n] real;
var x: [1..n]      real;
var y: [1..m]      real;

y = sum reduce(dim=2) forall(i,j) in [1..m,1..n] A(i,j)*x(j);
```

Example
Matrix-Vector Multiply on the CELL: V1

```
  var A: [1..m,1..n] real;
  var x: [1..n]      real;
  var y: [1..m]      real;

  y = sum reduce(dim=2) forall (i,j) in [1..m,1..n] A(i,j)*x(j);
```

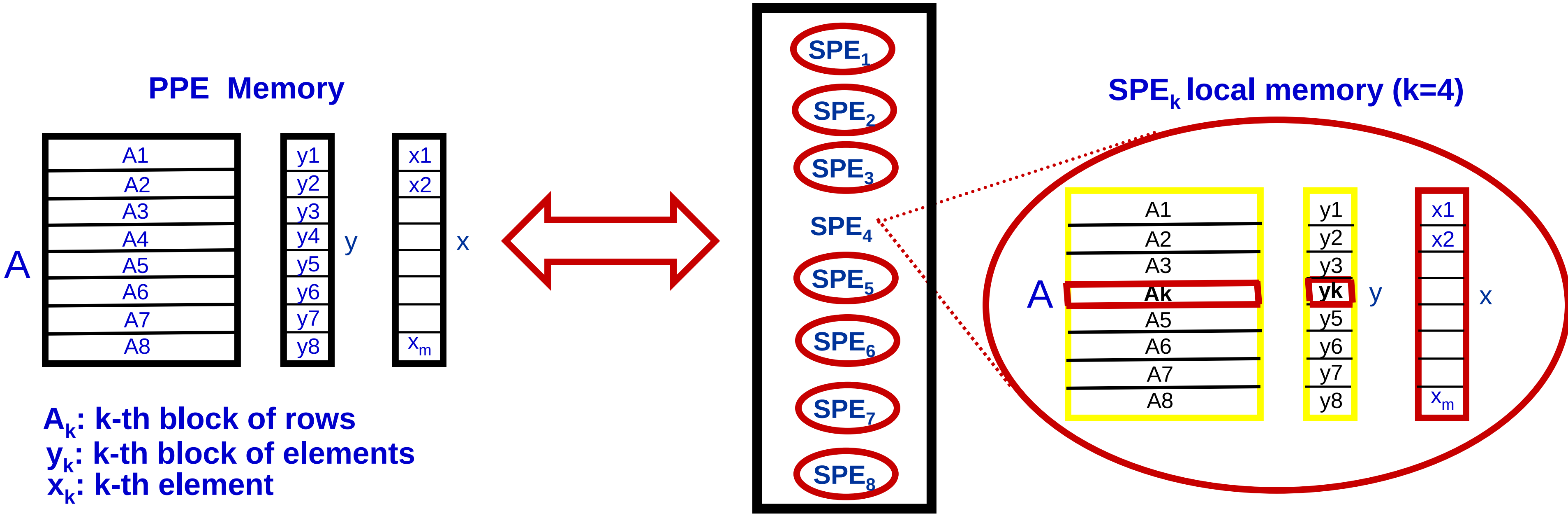
(original) Chapel version

```
param n_spe = 8;    /* number of synergistic processors (SPEs) */
const SPE:[1..n_spe] locale;      /* declaration of SPE array */

var A: [1..m,1..n] real distributed(block,*) on SPE;
var x: [1..n]      real replicated           on SPE;
var y: [1..m]      real distributed(block)   on SPE;

y = sum reduce(dim=2) forall (i,j) in [1..m,1..n] A(i,j)*x(j);
```

Chapel with (implicit) heterogeneous semantics



A_k: k-th block of rows
y_k: k-th block of elements
x_k: k-th element

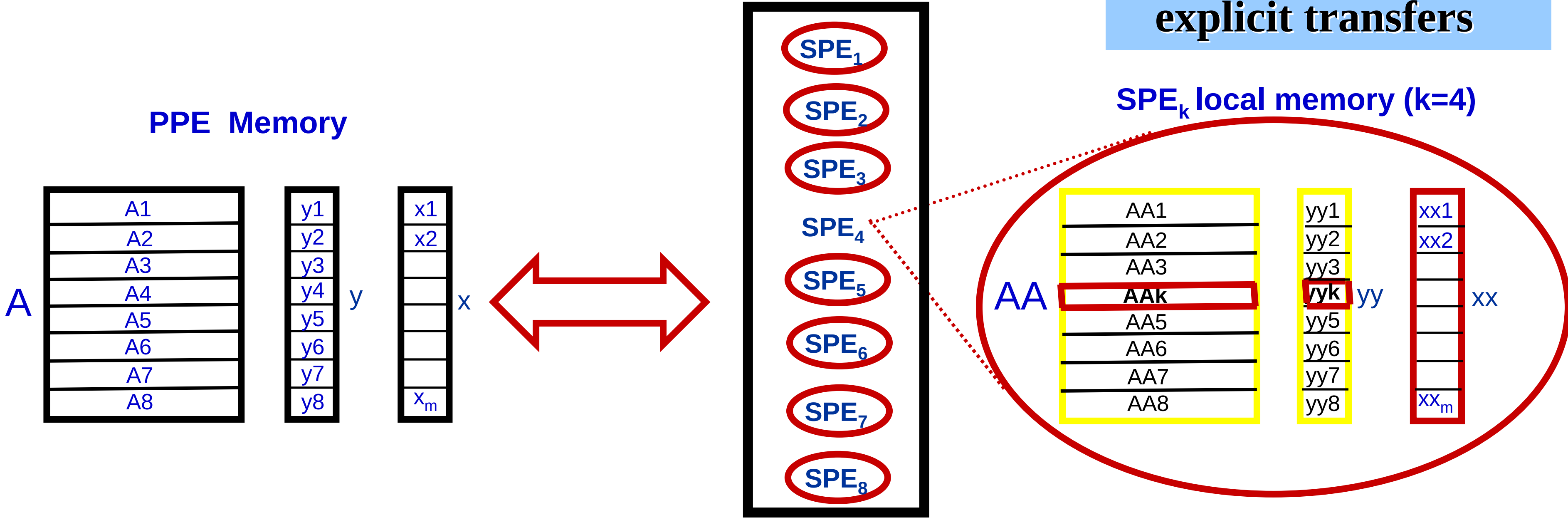
Example
Matrix-Vector Multiply on the CELL: V2

```
param n_spe = 8;                    /* number of synergistic processors (SPEs) */
const SPE:[1..n_spe] locale;                /* declaration of SPE locale array */
const PPE: locale                                 /* declaration of PPE locale */

var A: [1..m,1..n] real on PPE linked(AA) distributed(block,*) on SPE;
var x: [1..n]      real on PPE linked(xx) replicated           on SPE;
var y: [1..m]      real on PPE linked(yy) distributed(block)   on SPE;

AA=A; xx=x;                                /* copy and distribute A, x to SPEs */
y=sum reduce(dim=2) forall (i,j) in [1..m,1..n] on locale(xx(j)) A(i,j)*x(j);
y=yy;                                                 /* copy yy back to PPE */
```

Chapel/HETMC with explicit transfers



Chapel Summary

- Explicit support for nested data and task parallelism
- Locality awareness via user-defined data distributions
- Separation of computation from data organization
- Special support for high-level management of communication (halos, locality assertions, etc.)
- Natural framework for dealing with heterogeneous multicore architectures and real-time computation
- Also: The high-level approach represented by Chapel makes a significant contribution to *system reliability*

Contents

1 Introduction

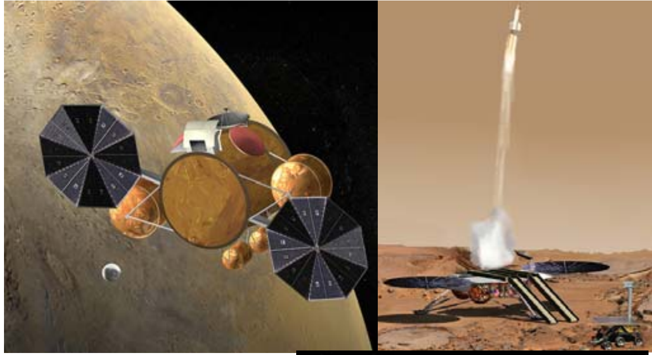
2 Towards High-Level Programming Models for Parallelism

3 Outline of a Generic Introspection Framework

4 Concluding Remarks

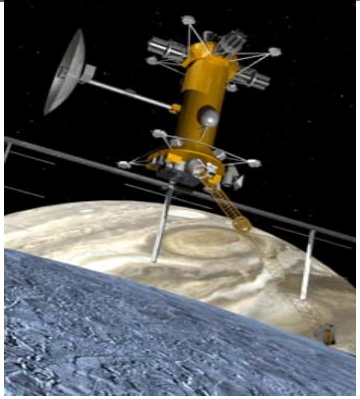
Requirements for Future Deep Space Missions

- **High-capability on-board computing**
  - *autonomy*
  - *science processing*
- **Radiation-hardened processor capability is insufficient**
  - *lags commercial products by 100X-1000X and two generations*
- **COTS-based multicore systems will be able to provide the required capability**
- **Fault Tolerance is a major issue**
  - *focus on dealing with Single Event Upsets (SEUs)*
  - *Total Ionization Dose (TID) is less of a problem*

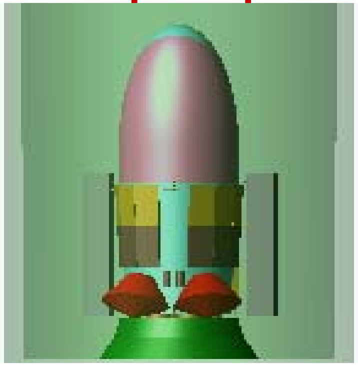


Mars Sample Return

VenusMobile Explorer

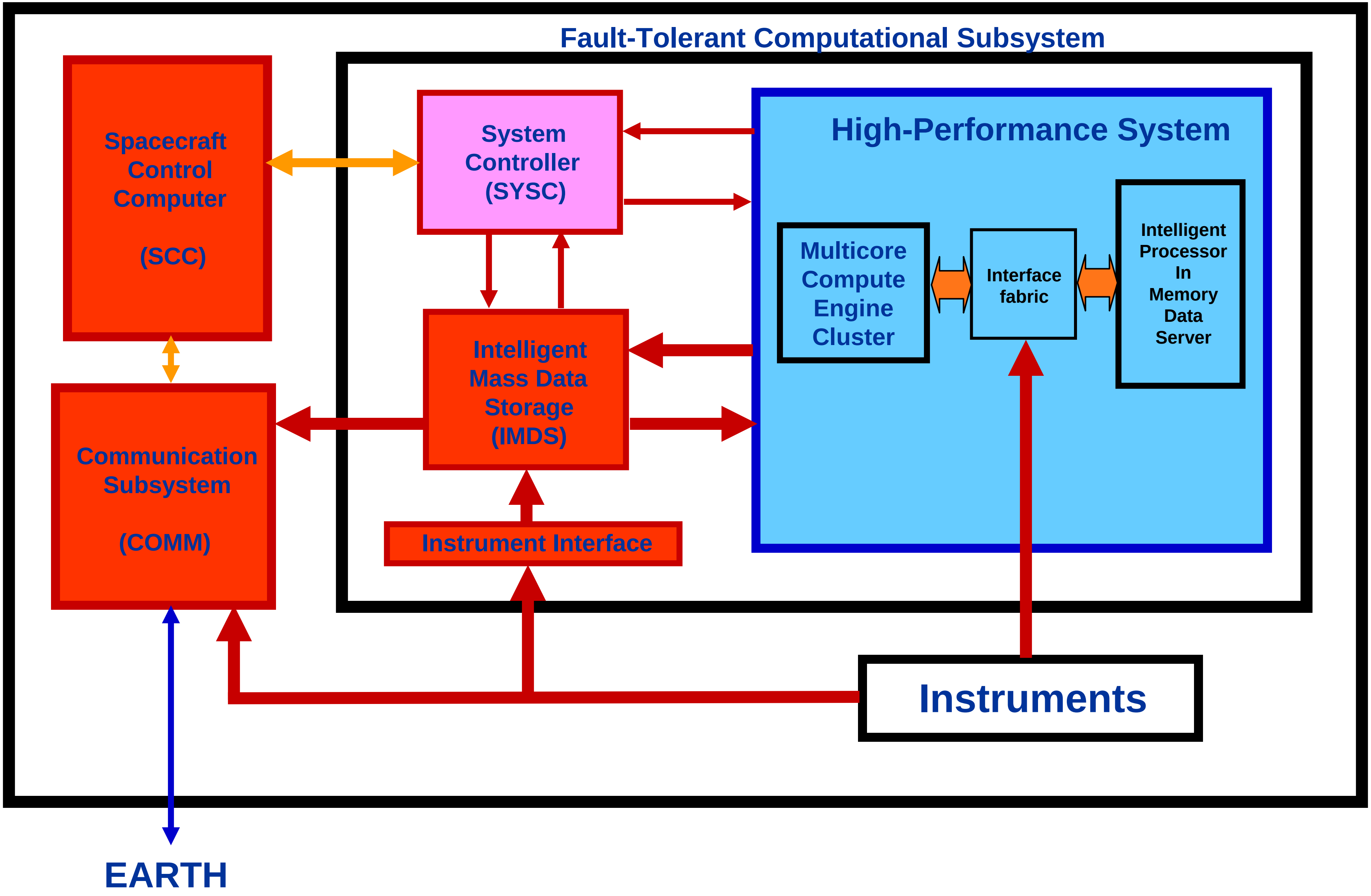


Europa Explorer



NeptuneTriton Explorer

High-Capability On-Board System: Global View


Fault-Tolerant Computational Subsystem
Spacecraft Control Computer (SCC)
System Controller (SYSC)
High-Performance System
Multicore Compute Engine Cluster
Interface fabric
Intelligent Processor In Memory Data Server
Intelligent Mass Data Storage (IMDS)
Communication Subsystem (COMM)
Instrument Interface
Instruments
EARTH

Fault-Tolerance in a High-Capability On-Board System

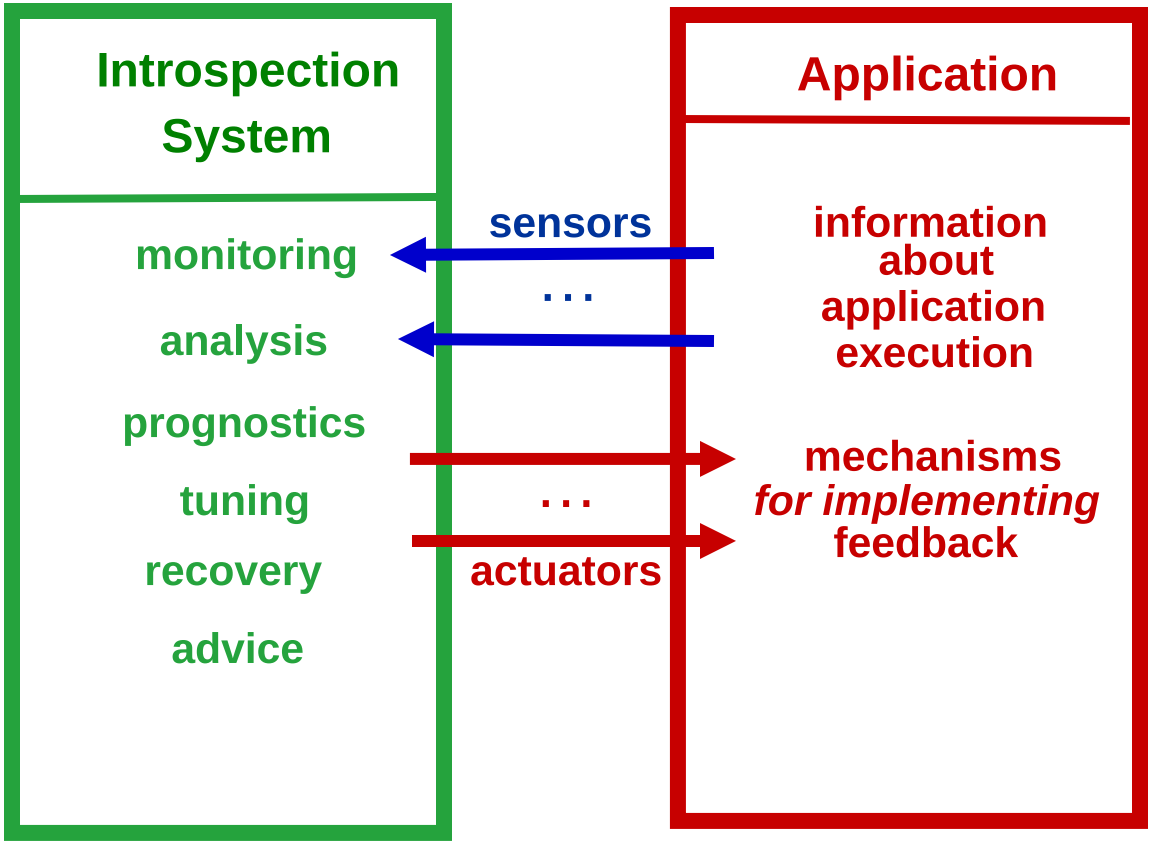
- **Deep hierarchy of hardware and software layers**
- **Fault tolerance must be addressed at each layer**
- **Approaches include**
  - ***hardware fault tolerance***
    - ***for example, spacecraft control computer***
  - ***combination of hardware and software fault tolerance, e.g.:***
    - ***system controller in the Space Technology 8 (ST-8) mission***
    - ***isolation of cores in a multicore chip***
  - ***software-implemented adaptive fault tolerance***
    - ***adjusting degree of fault tolerance to application requirements***
    - ***exploiting knowledge about the domain or the algorithm***
    - ***introspection can effectively support software fault tolerance***

Introspection…

- Exploits the abundance of processors in future systems
- Enables a system to become self-aware and context-aware:
  - *monitoring execution behavior*
  - *reasoning about its internal state*
  - *changing the system or system state when necessary*
- Can be implemented via a hierarchical system of agents
- Can be applied to many different scenarios, including:
  - *fault tolerance*
  - *performance tuning*
  - *energy management*
  - *behavior analysis*

A prototype system will be implemented at the Jet Propulsion Laboratory

Introspection: A Simplified Global View



Introspection System

monitoring

analysis

prognostics

tuning

recovery

advice

sensors

. . .

. . .

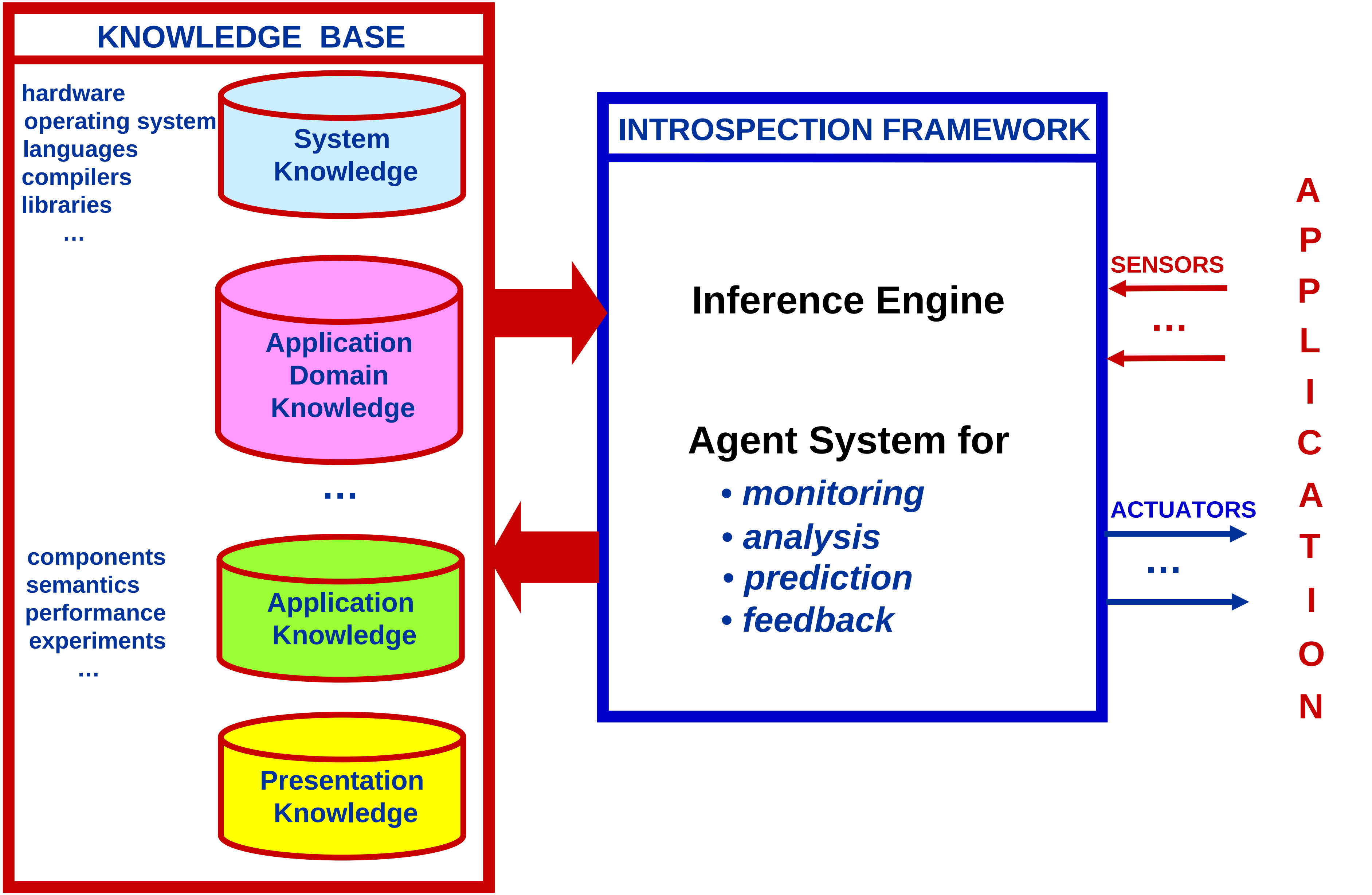
actuators

Application

information about application execution

mechanisms *for implementing* feedback

Introspection Framework Overview



KNOWLEDGE BASE

hardware
operating system
languages
compilers
libraries
…

System Knowledge

Application Domain Knowledge

…

components
semantics
performance
experiments
…

Application Knowledge

Presentation Knowledge

INTROSPECTION FRAMEWORK

Inference Engine

Agent System for

- *monitoring*
- *analysis*
- *prediction*
- *feedback*

SENSORS

…

ACTUATORS

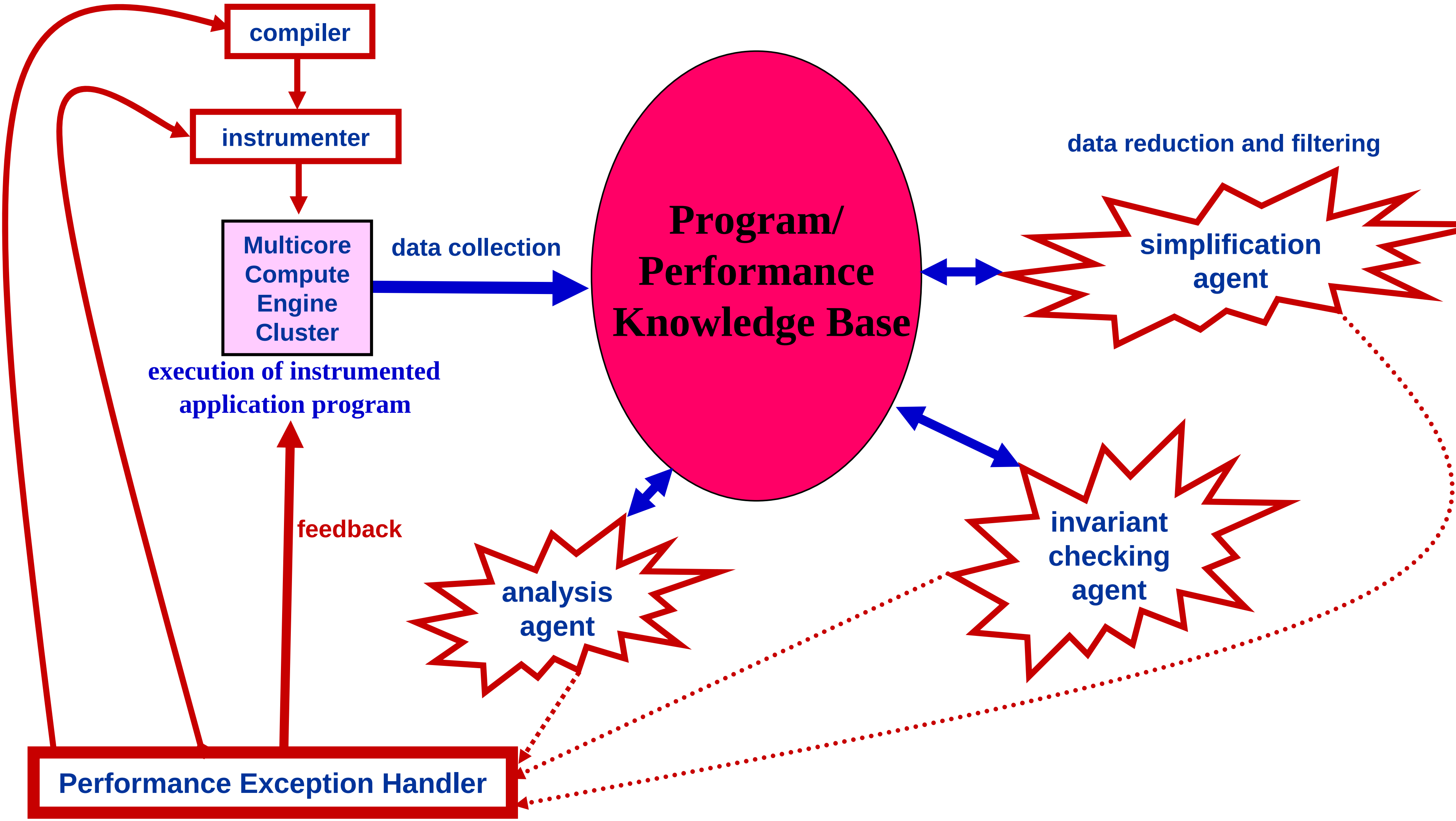
…

APPLICATION

Case Study: Introspection Sensors for Performance Tuning

- **Introspection *sensors* yield information about the execution of the application:**
  - *hardware monitors: accumulators, timers, programmable events*
  - *low-level software monitors (e.g., at the message-passing level)*
  - *high-level software monitors (e.g., at a high-productivity language level)*
- **Introspection *actuators* provide mechanisms, data, and control paths for implementing feedback to the application:**
  - *instrumentation and measurement retargeting*
  - *resource reallocation*
  - *computational steering*
  - *program restructuring and recompilation (offline)*

Agent System for Online Performance Analysis


compiler
instrumenter
Multicore Compute Engine Cluster
data collection
execution of instrumented application program
Program/ Performance Knowledge Base
data reduction and filtering
simplification agent
feedback
analysis agent
invariant checking agent
Performance Exception Handler

Concluding Remarks

- Future HPC and EC systems will be based on multicore technology providing low-cost high-capability processing
- Key software challenges
  - *programming and execution models combining high productivity with sufficient control for satisfying system requirements*
  - *intelligent tools supporting program development, debugging, and tuning*
  - *generic frameworks for introspection supporting fault tolerance, performance tuning, power management, and behavior analysis*
- All these developments are currently in flow
  - *architectures are a moving target*
  - *promising initial steps have been taken in many areas*
  - *successful high-productivity software solutions will take years to reach industrial strength*