Introduction into Multicore Programming

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Topic – Multicore Designs

Karoly.Bosa@jku.at

Four multicore designs from some of the computer industry's leading chip manufacturers:

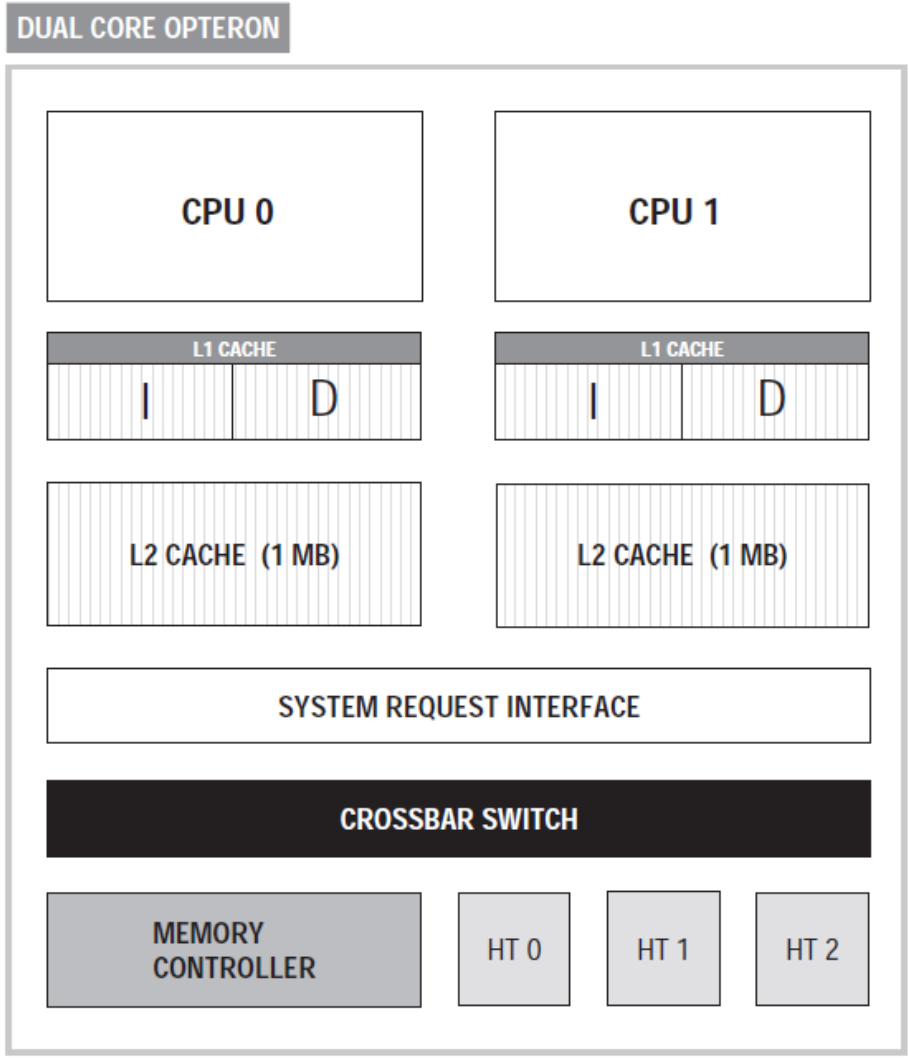
- AMD Multicore Opteron
- Sun UltraSparc T1
- IBM Cell Broadband Engine (CBE)
- The Intel Core 2 Duo

AMD Opteron

AMD Opteron Overview

Karoly.Bosa@jku.at

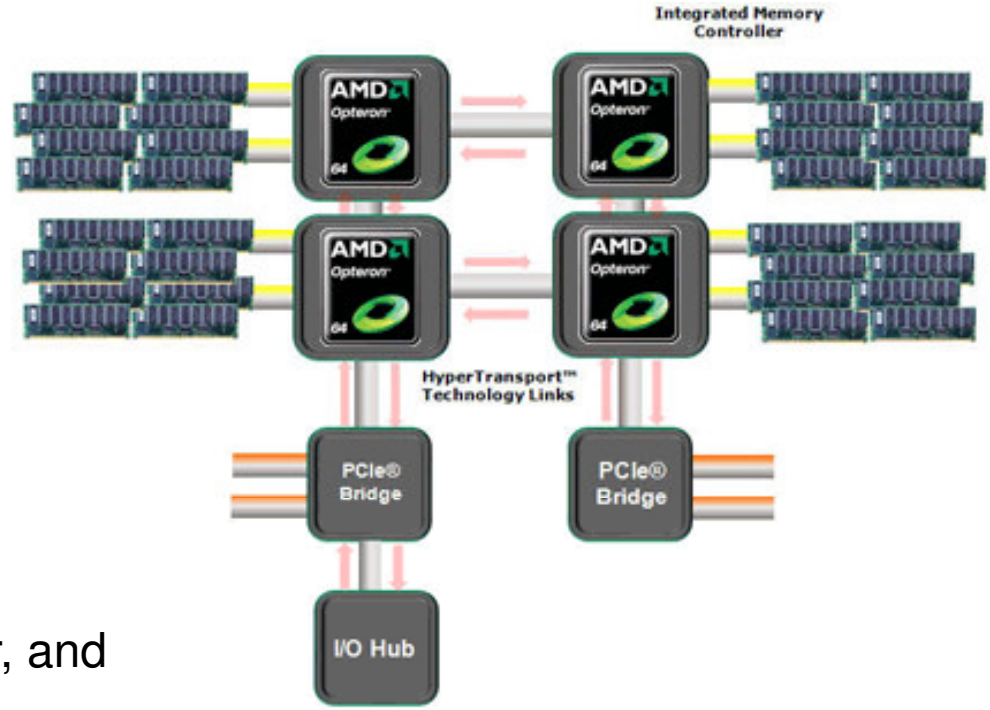
- entry level into AMD's multicore processor line,
- consists of two AMD 64 processors,
- compatible with Intel's family of processors,
- AMD's Direct Connect Architecture (DCA) with HyperTransport technology
- L1 cache can be logically divided to
  - I – Cache (for instructions) and
  - D - Cache (for data)



The Direct Connect Architecture in Opteron

Karoly.Bosa@jku.at

- AMD's DCA determines how the CPUs communicate with each other, memory and other I/O devices.



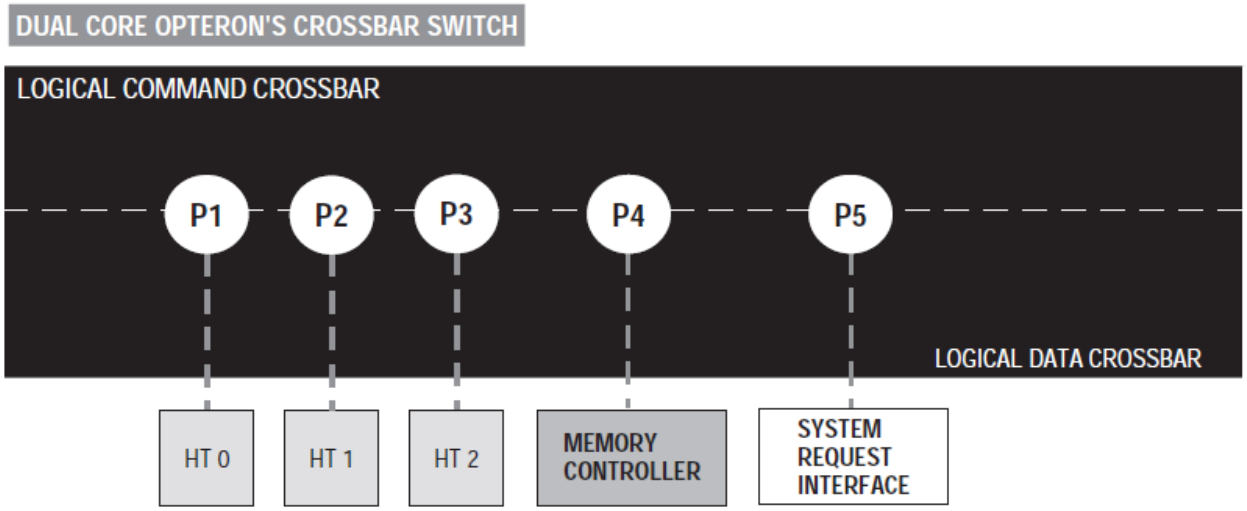
- The DCA is a point-to-point connection scheme, It does not use the FSB(!).
- The processors, memory controller, and I/O are directly connected.
- This dedicated link approach avoids the potential performance problems (bottlenecks) of the bus architectures.
- Each CPU can access the main memory of another processor, transparent to the programmer (HT ports).

System Request Interface and Crossbar

Karoly.Bosa@jku.at

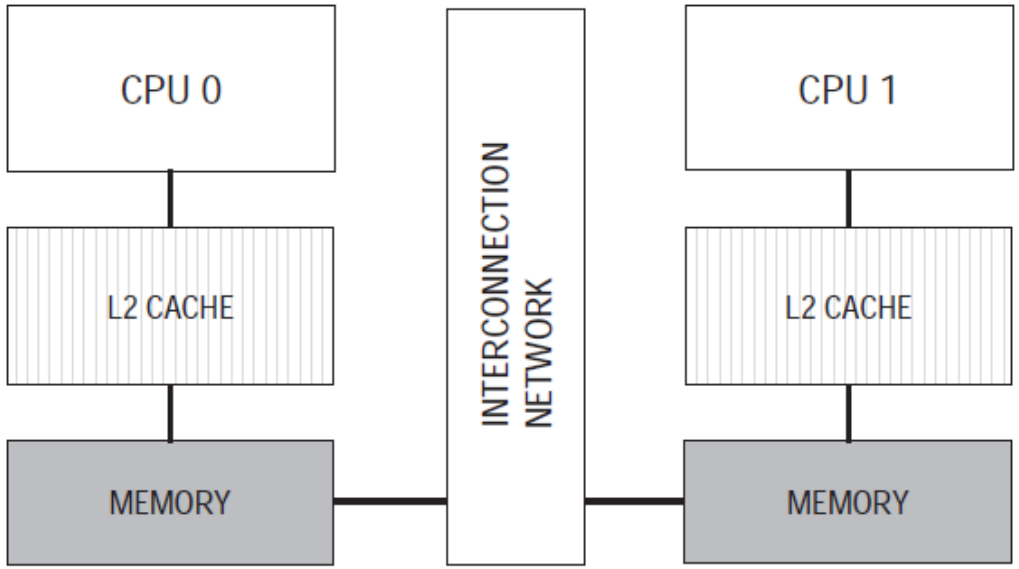
- The *System Request Interface (SRI)* manages and prioritizes the processor requests to the crossbar switch.
  - If the memory access is not local (off chip), then a routing table lookup sends it to a HT port.

- The *crossbar* has five ports: to memory controller, to SRI, and to the three HT ports.
  - It is responsible for packet routing.



The Opteron Is NUMA

- The network interconnection is accomplished by the Opteron HyperTransport technology.
- But using the HyperTransport technology, the CPUs are directly connected to each other and the I/O is directly connected to the CPU.



General NUMA Architecture

- This ultimately gives you a performance gain over the typical UMA and SMP configuration.

The Sun UltraSparc T1 Multiprocessor

Sun UltraSparc T1 Overview

- The UltraSparc T1 is an eight - core CMP.

- It has support for chip-level multithreading (CMT). Each core is capable of running four threads.

- By this T1 can handle up to 32 hardware threads → Eight cores with four threads presents itself to an application as 32 logical processors(!).

- Checking how many processors are apparently available for the operating system:

```
using namespace std;
#include < unistd.h >
#include < iostream >
int main(int argc,char *argv[])
{
        cout < < sysconf(_SC_NPROCESSORS_CONF) < < endl;
        return(0);
}
```

How many processors are available to the operating system

- ***Compile and Link Instructions:***
  gcc -o sysconfCPUTest sysconfCPUTest.cc

- ***Hardware:***
  AMD Opteron Core 2, UltraSparc T1, CBE, Intel Core 2 Duo

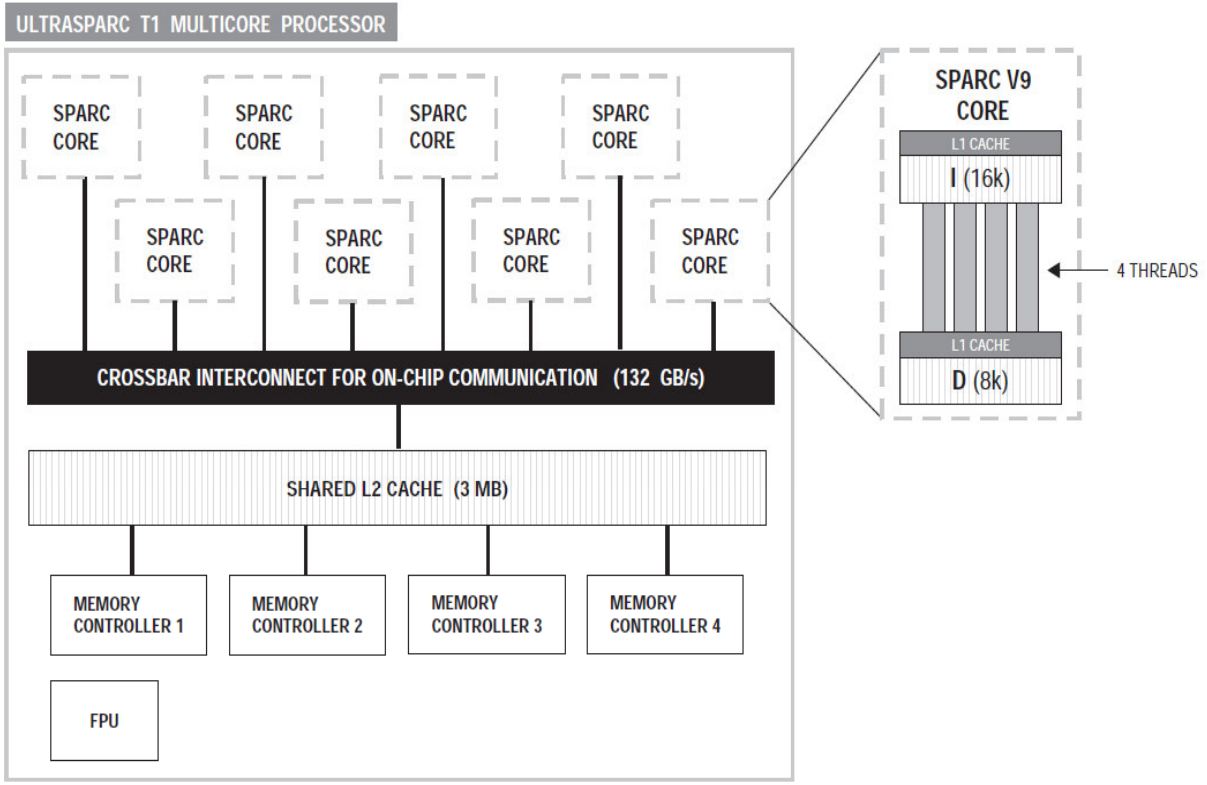
- sysconf() can query system variables and parameters:
  - _SC_NPROCESSORS_CONF → number of processors configured

  - _SC_NPROCESSORS_MAX →maximum number of processors supported

Sun UltraSparc T1 Overview II

Karoly.Bosa@jku.at

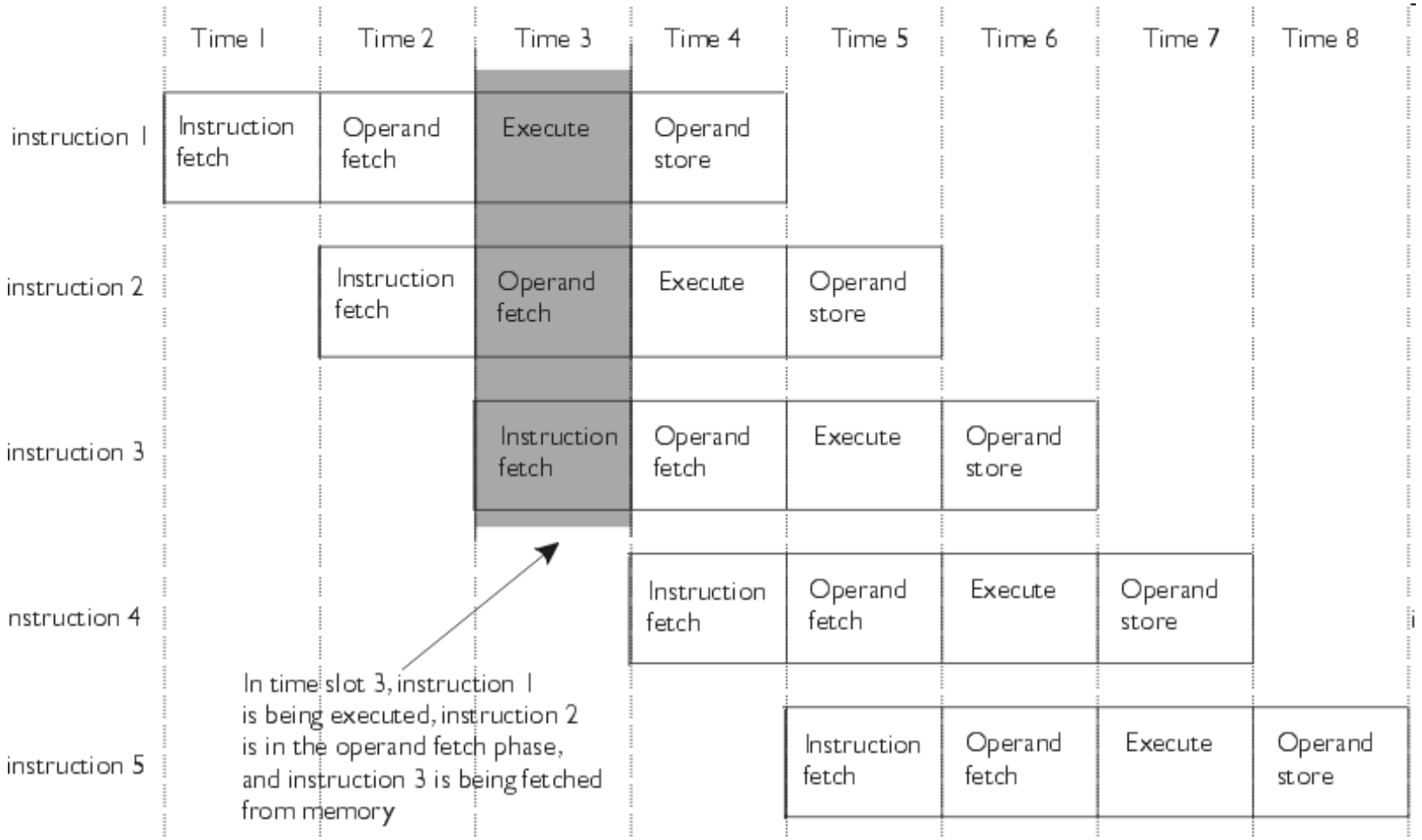
- Eight Sparc V9 cores.
- cores are 64-bit technology.
- Each core has L1 cache:
  - 16K L1 instruction cache and
  - an 8K L1 data cache.
- Shared L2 cache (3MB)



- The eight cores all share a single floating-point unit (FPU).
- Each core has a six-stage pipeline: **1) Fetch, 2) Thread selection, 3) Decode, 4) Execute, 5) Memory access, 6) Write back**

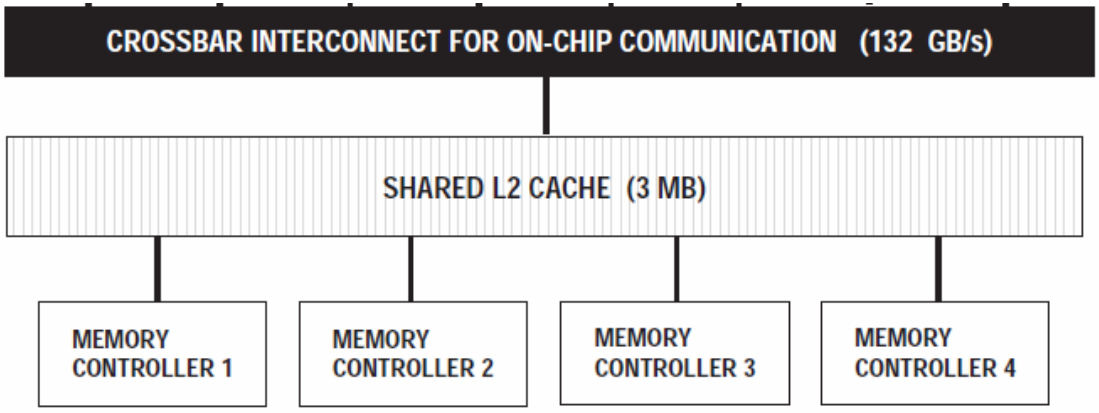
CPU Pipelining I.

Karoly.Bosa@jku.at



DDRAM Controller and L2 Cache

Karoly.Bosa@jku.at



- The UltraSparc T1 has four separate memory controllers.
- The L2 cache is divided on the T1 into four banks.
- Each controller is connected to one bank of L2 cache.
- The T1 can support up to 128GB of RAM.

The Sun and GNU gcc Compilers

- Two of the most commonly used compilers for the UltraSparc T1 are

  - the Sun C/C++ compiler (part of Sun Studio) and

  - the GNU gcc, the standard open source C/C++ compiler.

- Sun's compilers obviously have the best support for their processors,

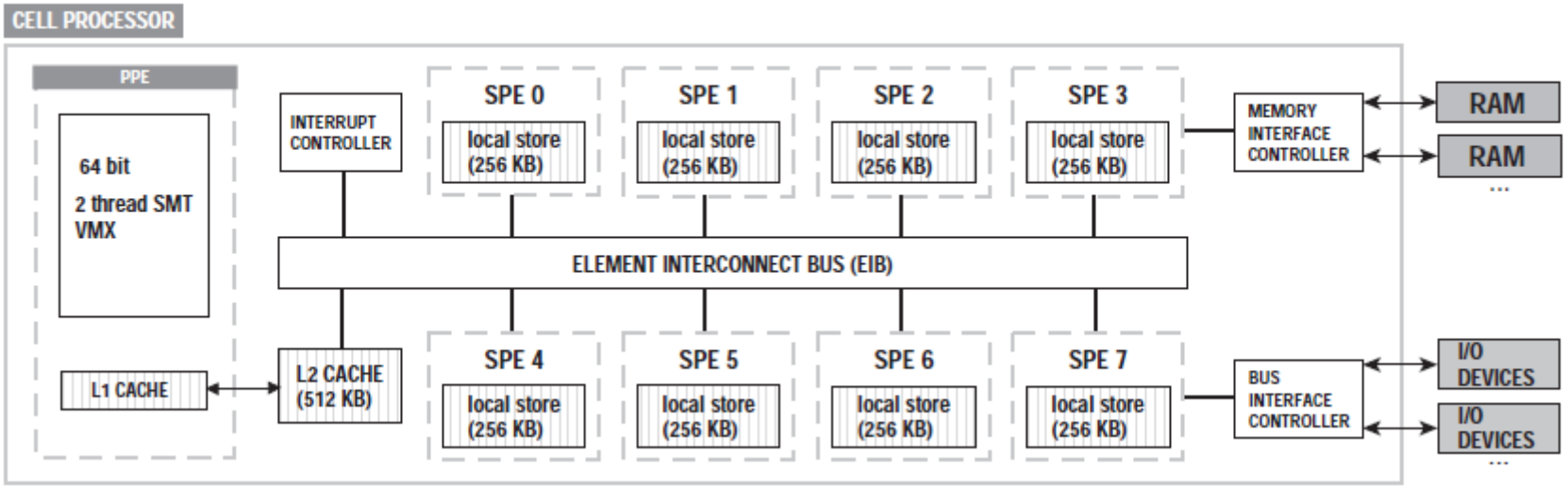
- GNU gcc has a great deal of support for T1, with options that take advantage of

  - thread loop unrolling,
  - vector operations,
  - branch prediction and
  - Sparc-specific platform options.

The IBM Cell Broadband Engine (CBE)

CBE Overview

Karoly.Bosa@jku.at

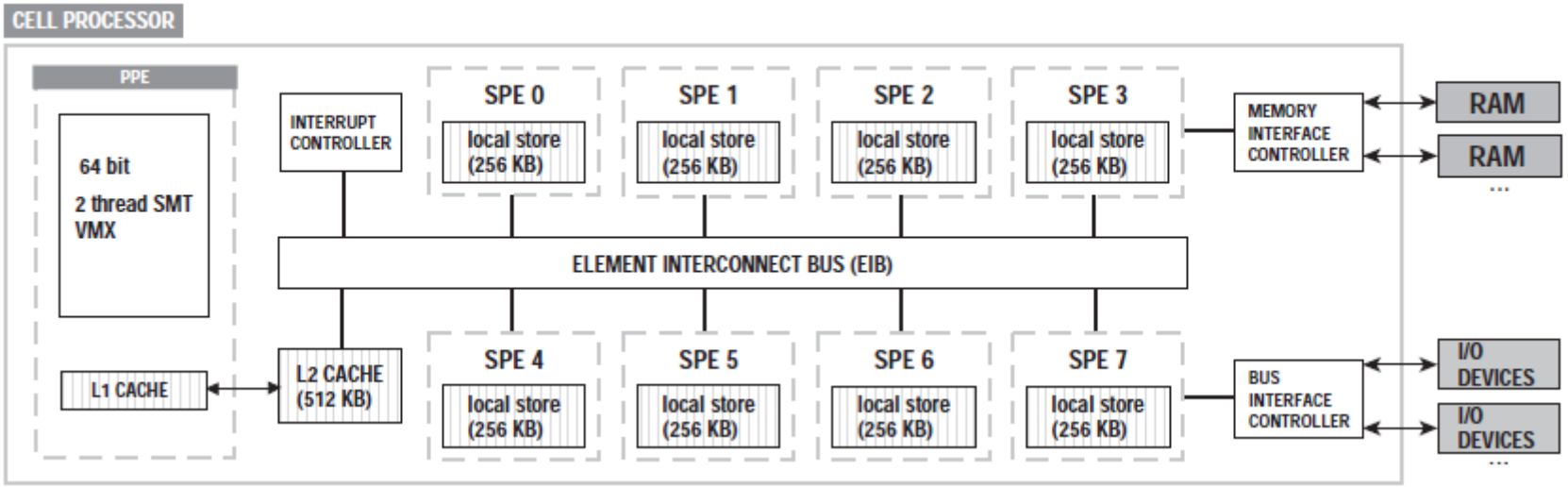


- The CBE is a **hybrid 9 core** processor (heterogeneous multicore chip).

- It consists of two different types of processors:
  - **1 PowerPC** Processing Element (PPE) and
  - **8 Synergistic** Processor Element (SPE), each has its own local memory.
- Additionally one high-speed memory controller, one high–bandwidth element interconnect bus, high-speed memory and I/O interfaces **are integrated on chip.**

CBE Overview

Karoly.Bosa@jku.at



- The CBE is a **hybrid 9 core** processor (heterogeneous multicore chip).

- It consists of two different types of processors:
  - **1 PowerPC** Processing Element (PPE) and
  - **8 Synergistic** Processor Element (SPE), each has its own local memory.
- Additionally one high-speed memory controller, one high–bandwidth element interconnect bus, high-speed memory and I/O interfaces **are integrated on chip.**

CBE and Linux

- CBE is very effective in a Linux environment (Playstation 3 comes with ready-to-install Linux).
- Currently, there is a Fedora and a Yellow Dog distribution of Linux for the CBE.
- The PPE element can be programmed using the standard GNU gcc compiler.
- If you would like to check the number of processors (with sysconf), the output is 2, because:
  - the SPEs are not directly accessible.
  - the PPE is a CMP; it is a dual thread processor.
- There is a CBE SDK available for downloading from IBM that includes tools necessary to compile the SPE code (It works x86, too).
    - http://www.ibm.com/developerworks/power/cell/downloads.html
    - http://librenix.com/?inode=7715 (another useful link)

IBM Full-System Simulator for the CBE

Karoly.Bosa@jku.at

- **Download link:**
  http://www.alphaworks.ibm.com/tech/cellsystemsim/download?open&S_TACT=105AGX59&S_CMP=GR

- **Hardware Requirements:**
  - The simulator is available for 32-bit and 64-bit x86 and 64-bit PowerPC systems.
  - Certain simulator features are only available on 64-bit architectures.
  - The system should have at least 1GB of memory available for use by the simulator.
- **Operating system:**
  - Fedora Linux Release 9.

- **Software:**
  - The simulator requires the tcl, tk, and xterm packages to be installed prior to installing the simulator.

POSIX Threads on CBE

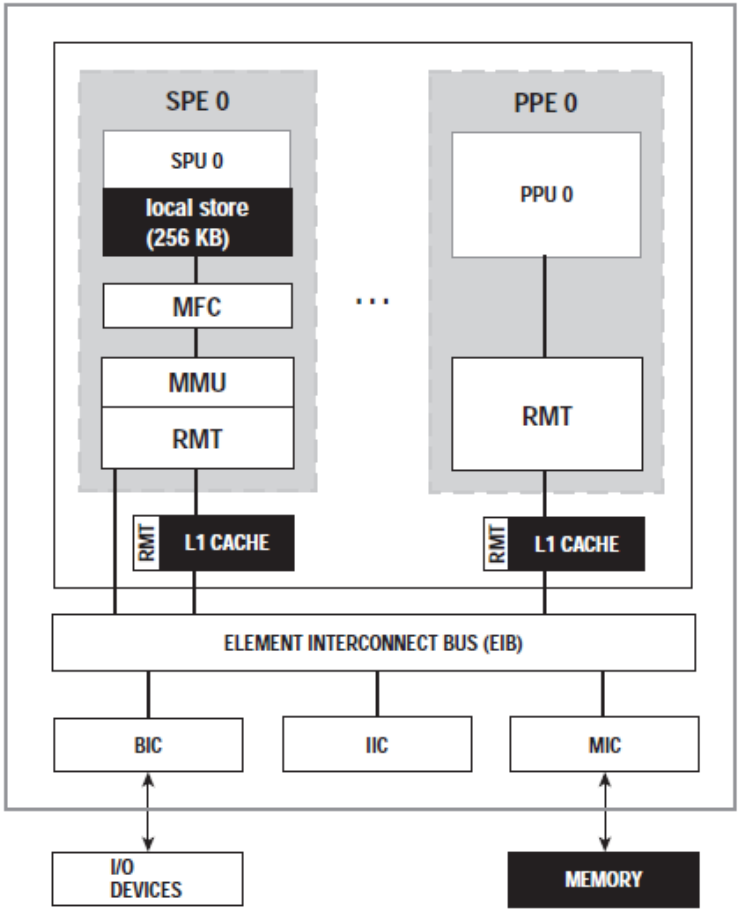
Karoly.Bosa@jku.at

- Standard POSIX threads (pthreads) and process management can be used with the PPE element,
- But the SPE has to be programmed using the thread library that ' s available as part of the CBE SDK.
- However the good news is the SPE thread calls are designed to be compatible with pthreads.

CBE Memory Models

Karoly.Bosa@jku.at

- The PPE accesses memory differently than the SPEs.
- CBE avoids the normal single bus bottleneck potentials because the SPEs each have their own local memory.
- Each SPE depends on DMAs to transfer data to and from the main memory and other SPEs' local memories.
- Simultaneous(!) data and code transfers between the SPE local stores and main storage.



SPE	Synergistic Processing Element	MMU	Memory Management Unit
SPU	Synergistic Processing Unit	RMT	Replacement Manangement Table
PPE	PowerPC Processing Element	BIC	Bus Interface Controller
PPU	PowerPC Processing Unit	IIC	Internal Interrupt Controller
MFC	Memory Flow Controller	MIC	Memory Interface Controller

Hidden from the Operating System

- CBE elements must be directly addressed to get the maximum performance.

- The standard Linux system calls can see the dual threads of the PPE,

- but they are not fully aware of the SPEs:
  - The developer must explicitly develop and compile code that works with the SPEs,
  - and then that code must be linked with the code from the PPE.

At the last point Linux knows how to handle the eight SPE processors.

Intel Core 2 Duo Processor

Intel Core 2 Duo Overview

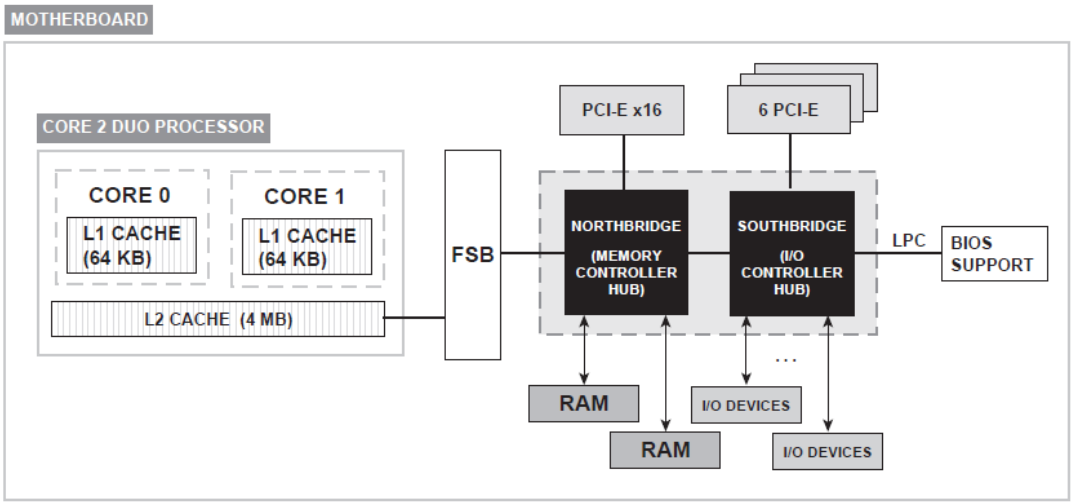
Karoly.Bosa@jku.at

- Some Intel multicore CPUs have dual cores, others have quad cores…

- Some multicore processors are enhanced with hyperthreading, giving each core two logical processors.

- Core 2 Duo was introduced in 2006, has dual cores; it has no hyperthreading but supports a 64 bit architecture.

- The *chipset* of the motherboard moves data back and forth from CPU to the various components of the motherboard, including
  - memory,
  - graphics card and
  - I/O devices.

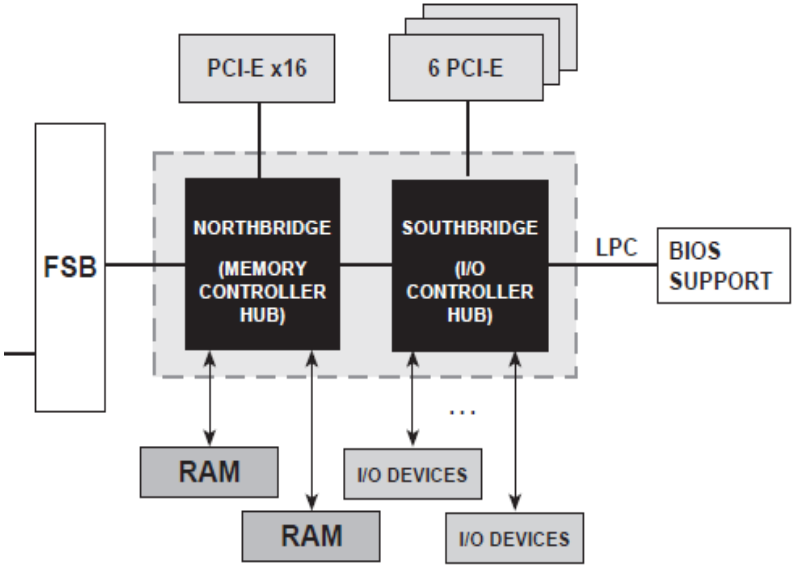


Some More Words about Chipset

Karoly.Bosa@jku.at

The chipset comprises two chips: Northbridge and Southbridge.

- The **Northbridge** (or **memory controller hub**):
  - Communicates directly with the CPU via the Front Side Bus.
  - Connects the CPUs with high-speed devices such as main memory.
  - Connects the CPUs with PCI-E slots and the Southbridge via an internal bus.



- The **Southbridge** (or **I/O controller):**
  - Slower than the Northbridge.
  - Responsible for the slower capabilities of the motherboard like the I/O devices such as audio, disk interfaces, and so on.
  - Connected to BIOS support via the Serial Peripheral Interface (SPI), six PCI-E slots, and other I/O devices.

Intel's PCI Express

- PCI-E or PCI Express is a computer expansion card interface.
- The usual slot serves as a serial connection for sound, video, and network cards on the motherboard.
- The PCI-E is a high-speed serial connection, which works more like a network than a bus.
- It uses a switch that controls many point-to-point full-duplex (simultaneous communication in both directions) serial connections called lanes.
- There can be 4, 8, of 16 lanes per slot. Each lane has two pairs of wires from the switch to the device — one pair sends data, and the other pair receives data.

Core 2 Duo's Instruction Set

Karoly.Bosa@jku.at

- The Core 2 Duo has increased performance of its processor by supporting Streaming SIMD Extensions (SSE).

- This speeds up applications that utilize SIMD operations such as highly intensive graphics, encryption and mathematical applications

- The processor has 16 registers used to execute SIMD instructions: 8 MMX and 8 XMM registers.

Register Set	Description
MMX	Set of eight registers used to perform operations on 64-bit packed integer data types
XMM	Set of eight registers used to perform operations on 128-bit packed single- and double-precision floating-point numbers
MXCSR	Register used with XMM registers for state management instructions

Topic – Roles of the Operating Systems

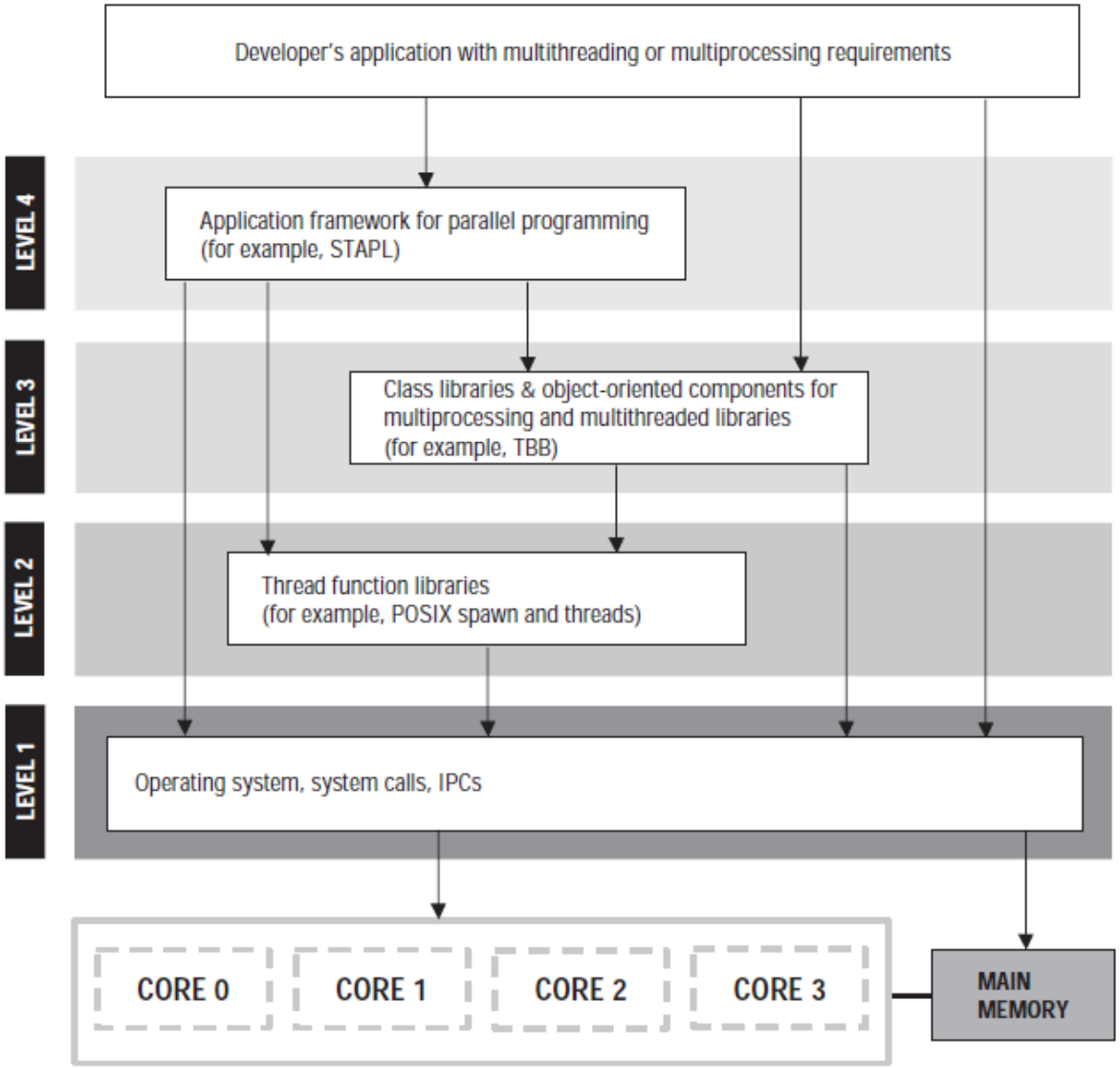
Roles of the operating systems:

- The Developer's Interaction with the Operating System
- Core Operating System Services
- Decomposition Choices of Parallel Programs
- Kernel Mode and User Mode
- Processor Affinity

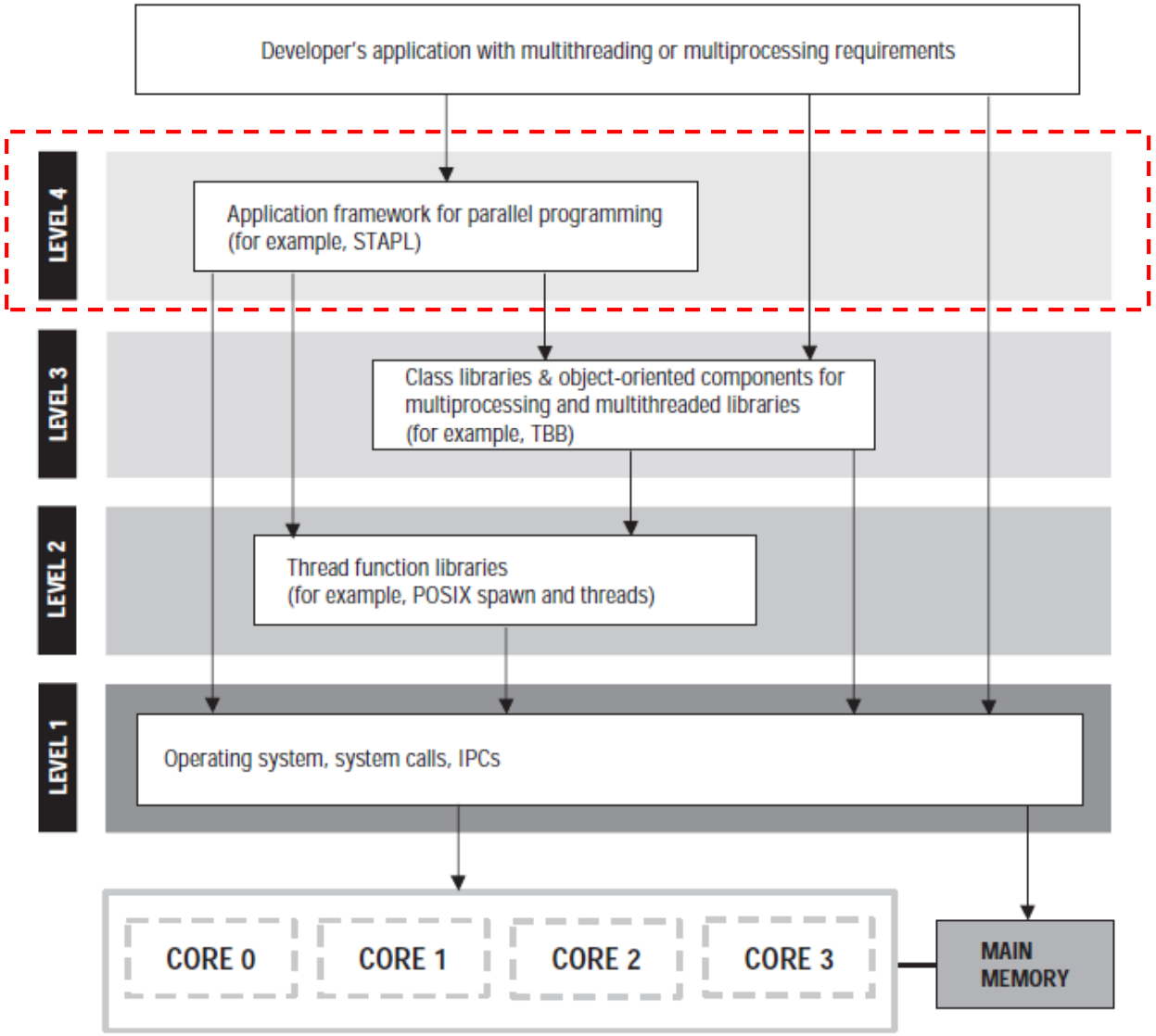
What Part Does the Operating System Play?

- Our focus on the operating system is the role it plays as a development tool:
    - **Software interface**: Providing a consistent and well defined interface to the hardware resources of the computer.
    - **Resource management** : Managing the hardware resources and other executing software applications, jobs, and programs
- the operating system provides a couple of software layers between the developer's program and the hardware resources connected to the computer.
- The two most important layers are called the *Application Program Interface (API)* and the *System Program Interface* (*SPI)* .

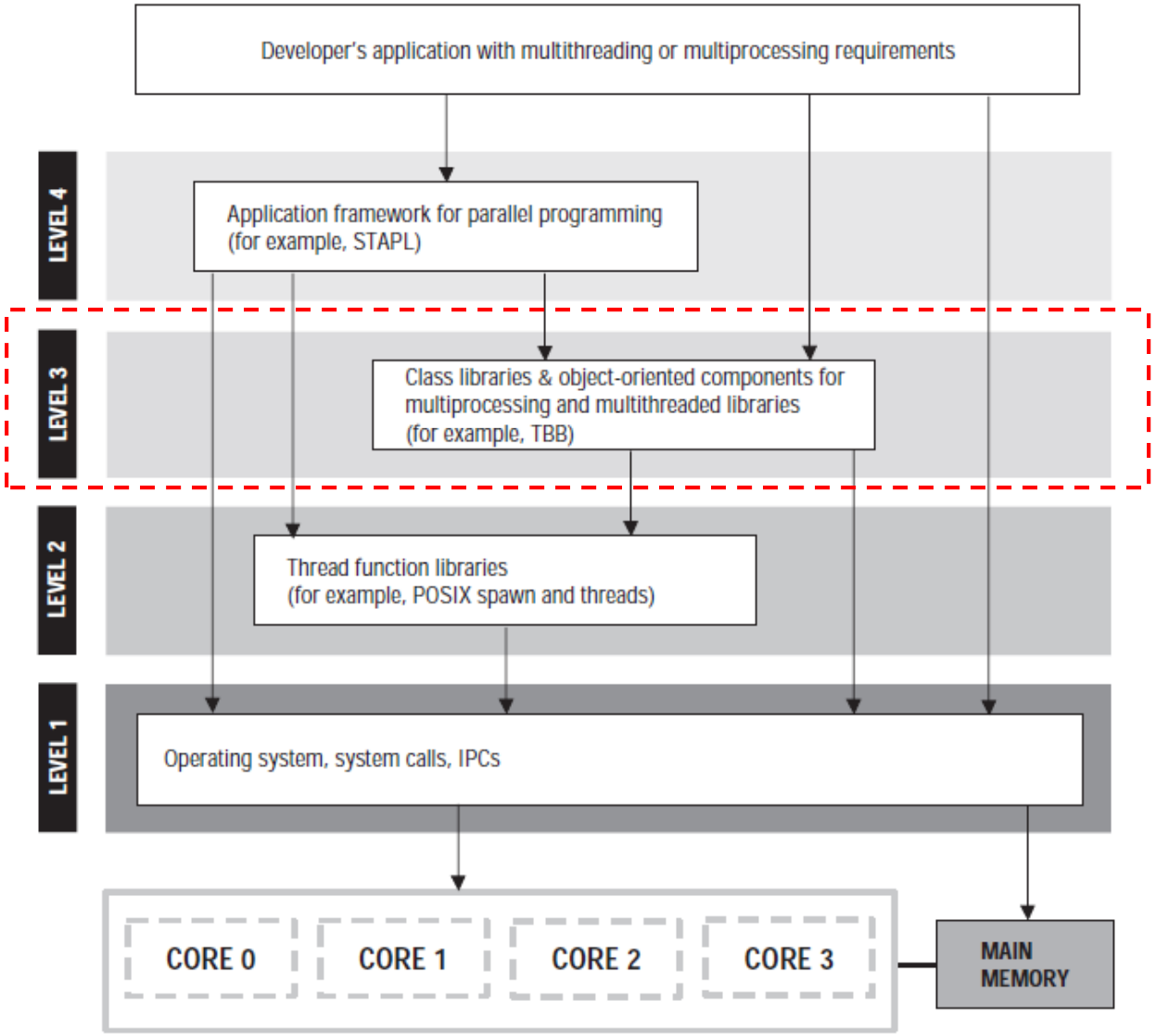
The Developer ' s Interaction with the Operating System



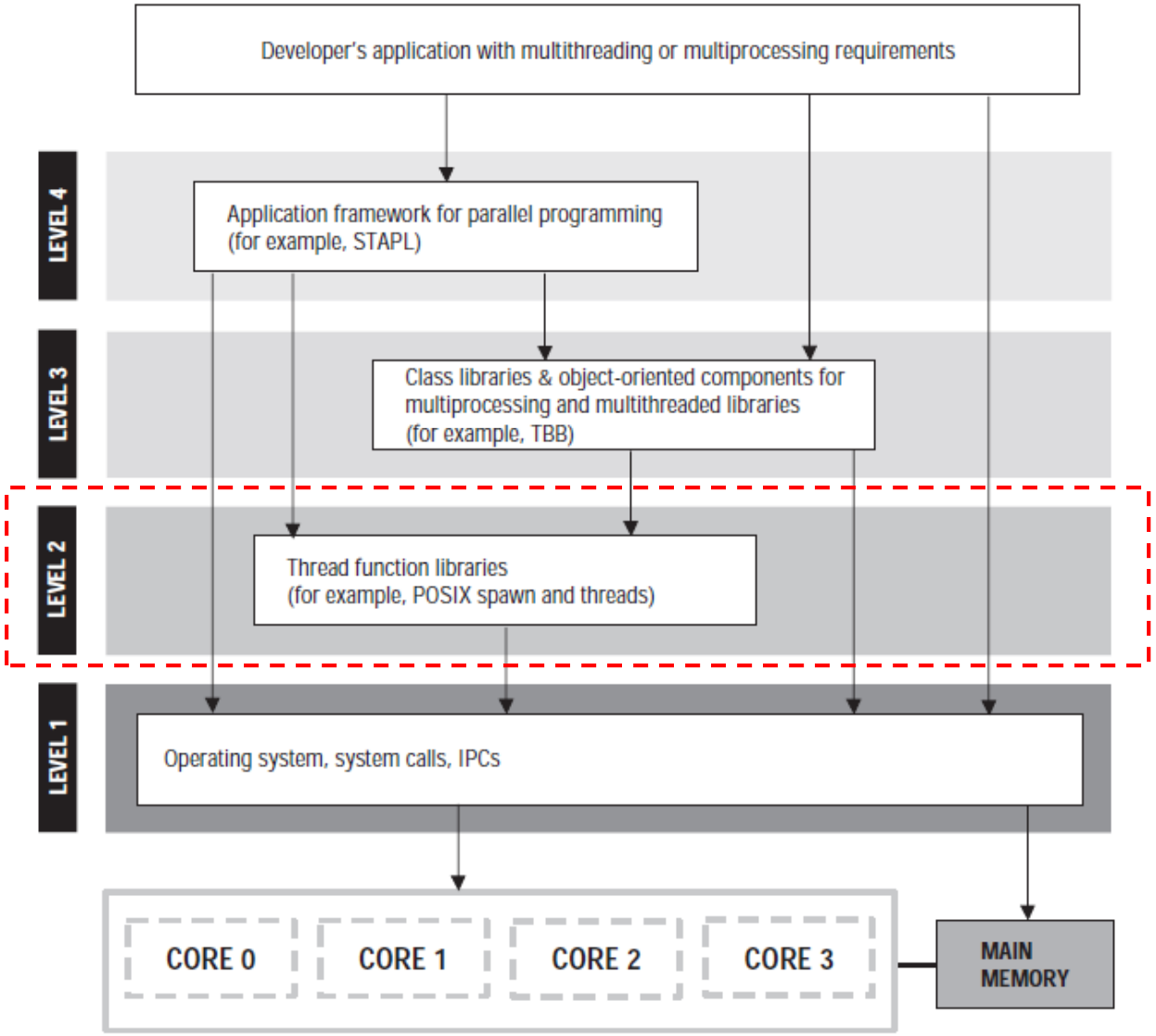
The Developer ' s Interaction with the Operating System



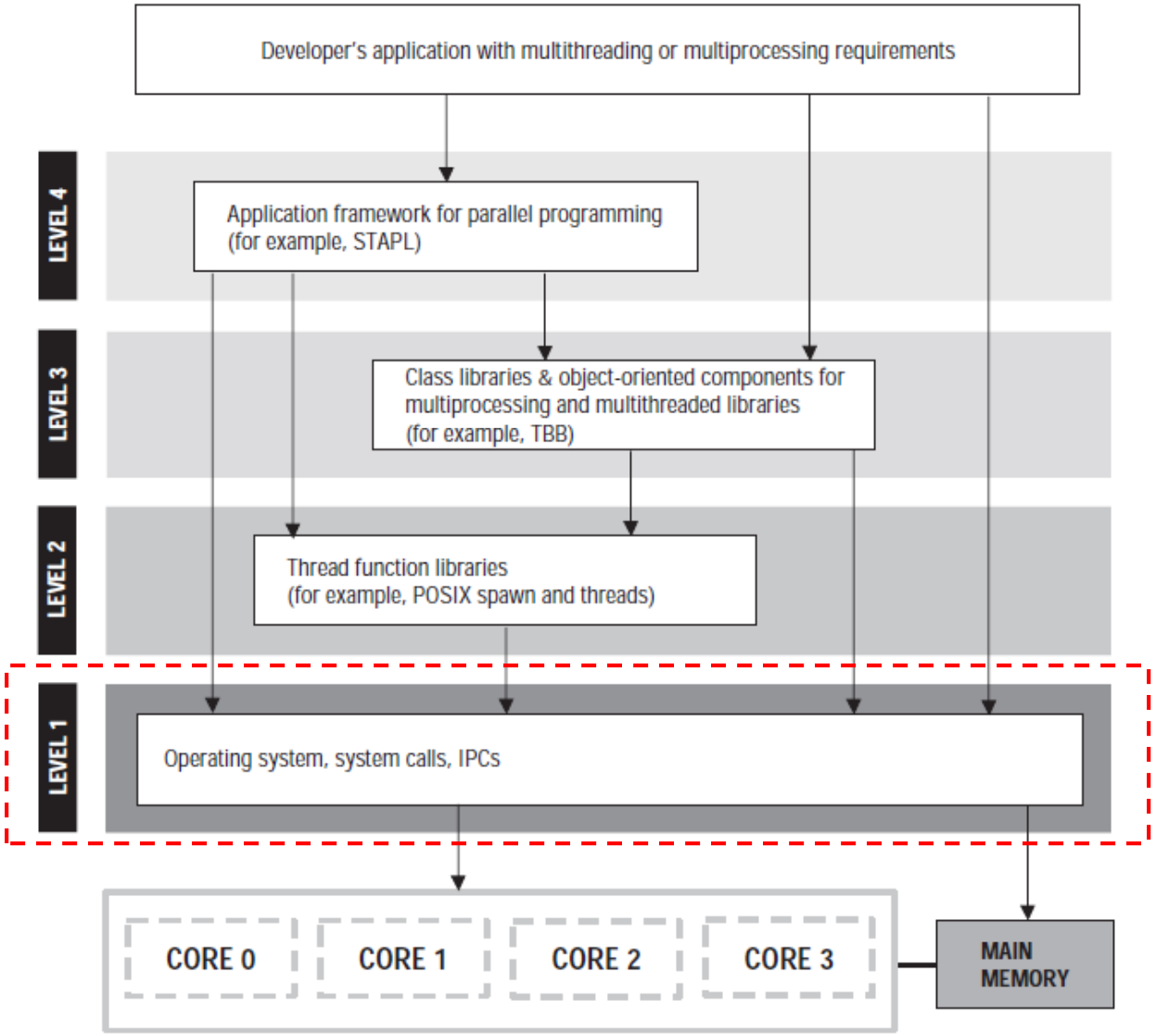
The Developer ' s Interaction with the Operating System



The Developer ' s Interaction with the Operating System



The Developer ' s Interaction with the Operating System

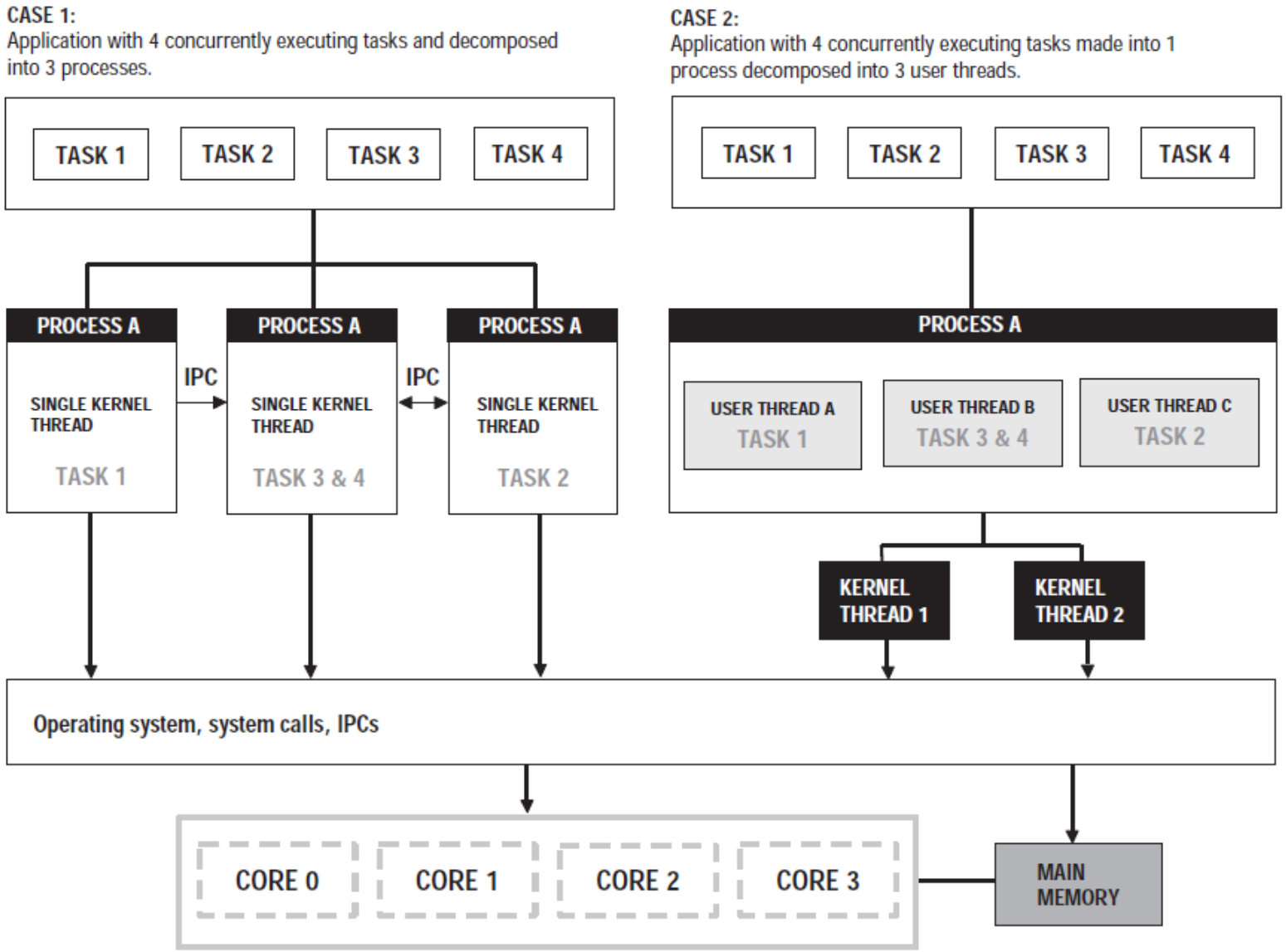


Core Operating System Services

- Process management
- Memory management
- Filesystem management
- I/O management
- Interprocess Communication Manager

Example for Decomposition Choices

Karoly.Bosa@jku.at

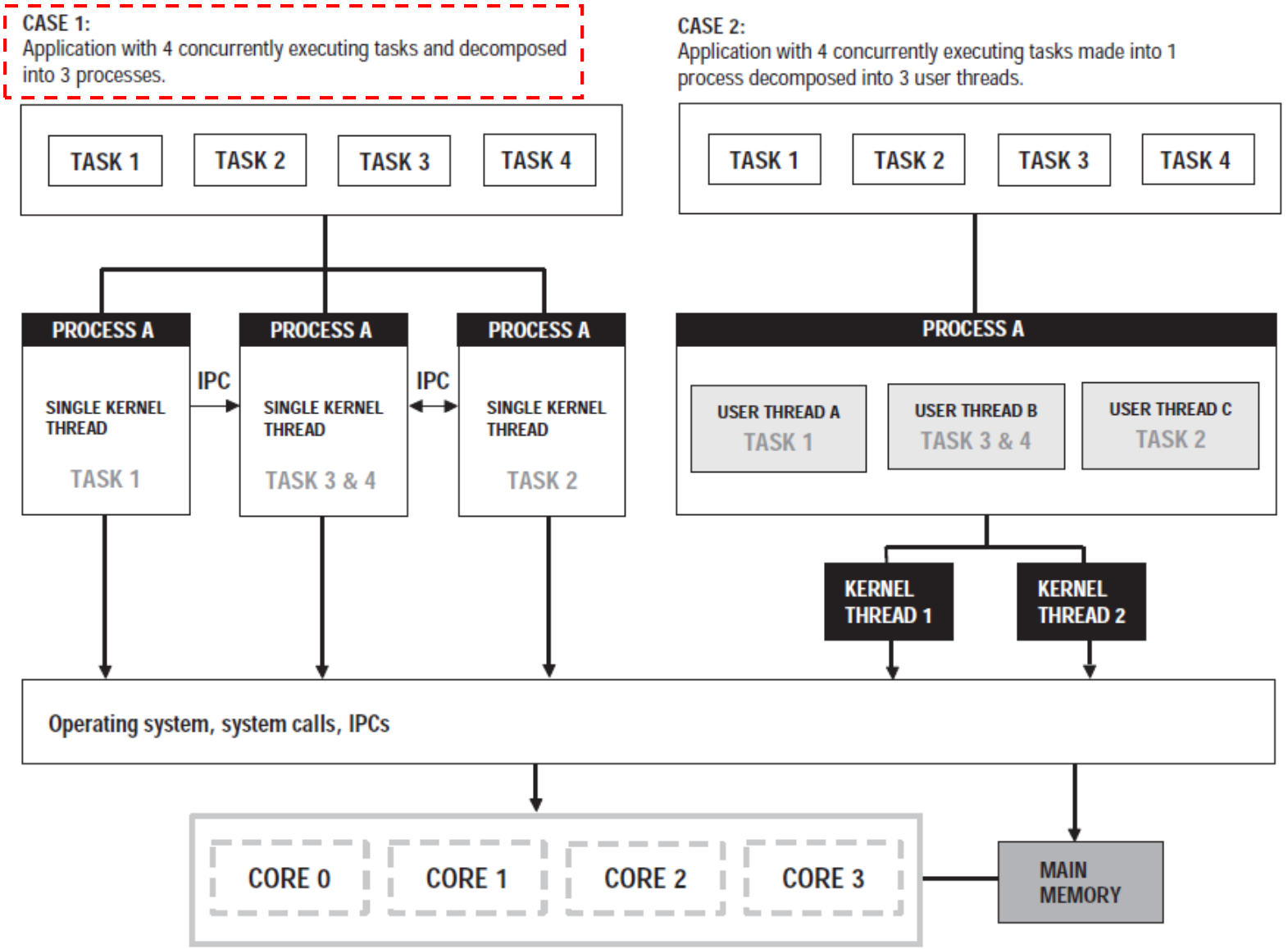


User Thread vs. Kernel Thread

- Kernel thread:
  - it is sometimes called a LWP (Lightweight Process) because it has less overhead than a process.
  - it is created and scheduled by the kernel.
  - it is the "lightest" unit of kernel scheduling.
  - at least one kernel thread exists within each process.
  - the allowable number of kernel threads per process is limited.
- User thread:
  - it is normally created by a threading library and scheduling is managed by the threading library itself.
  - All user threads belong to process that created them.
  - The advantage of user threads is that they are portable.
- Kernel threads are often more expensive to create than user threads and the system calls to directly create kernel threads are very platform specific.

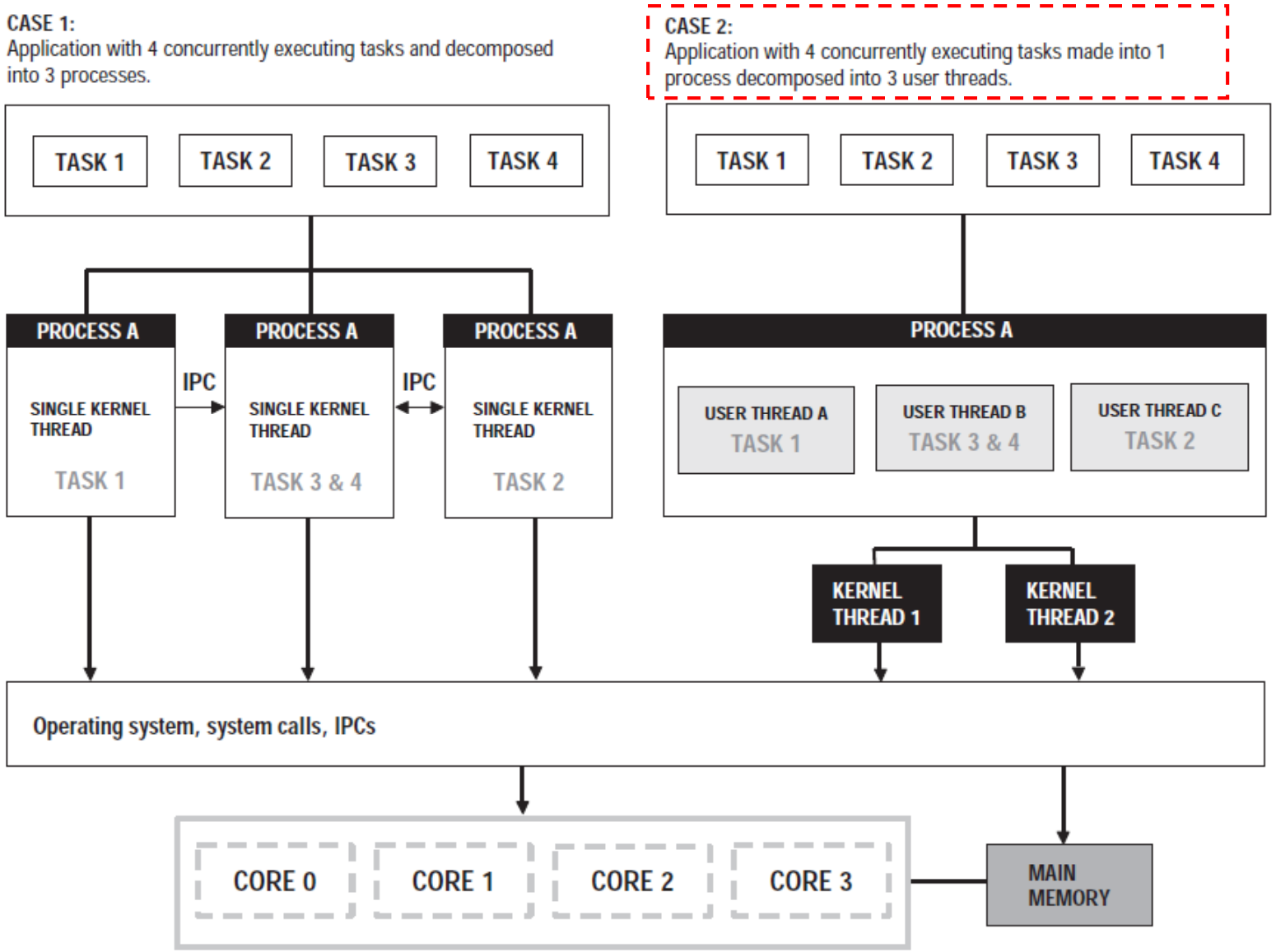
Example for Decomposition Choices

Karoly.Bosa@jku.at



Example for Decomposition Choices

Karoly.Bosa@jku.at



What Is POSIX and Why Use It?

- Portable Operating System Interface (POSIX) is a standard.

- It defines a standard operating system interface and environment, including:

  - a command interpreter (or “ shell ”) and

  - common utility programs

  to support applications portability at the source code level.

- The major operating system environments e.g.: ZOS, Solaris, AIX, Windows, Mac OS X, Linux, HP-UX, IRIX, all claim basic support for the POSIX standard.

Process Management

- For our purposes, the process lifecycle is summarized as:
  - Process creation
  - Process scheduling/execution
  - Process termination

- The operating system has to multitask the processes. Each process executes until some amount of time has expired or until some event has occurred.

- The interval of time a process is given to execute on a core is called a *quantum*.

- Each process is controlled by an associated scheduling policy and priority.

- We will use to our purposes four basic scheduling policies supported by the POSIX standard:

SCHED_FIFO
SCHED_RR
SCHED_SPORADIC
SCHED_OTHER

User Mode vs. Kernel Mode

- There are two distinct execution modes for the CPU in Linux:
  - Kernel mode: it is assumed to be executing trusted software (the kernel itself), and thus it can execute any instructions and reference any memory addresses.
  - User mode: a non-privileged mode in which each process starts out. It is forbidden for processes in this mode to access those portions of memory (i.e., RAM) that have been allocated to the kernel or to other programs.
- When a user mode process wants to use a service that is provided by the kernel (i.e., access system resources), it must switch temporarily into kernel mode, which has root privileges.
- When the kernel has satisfied the process's request, it restores the process to user mode.
- The standard procedure to switch from user mode to kernel mode is to call the 0x80 software interrupt.

SMP Support in Linux

- SMP support was introduced with kernel version 2.0, and has improved steadily ever since:

- The way Linux implements threads is to treat them at scheduling the same way as any process.

- Some Linux distributions don't provide a ready–made SMP–aware kernel, which means that you'll have to make one yourself.

- You can easily check the SMP support by the statement:

  cat /proc/cpuinfo

- For detailed hardware specific support information see Enkh Tumenbayar, Linux SMP HOWTO (http://tldp.org/HOWTO/SMP-HOWTO.html).

Processor Affinity

Karoly.Bosa@jku.at

- Processor affinity is the ability to direct a specific task, or process, to use a specified core.

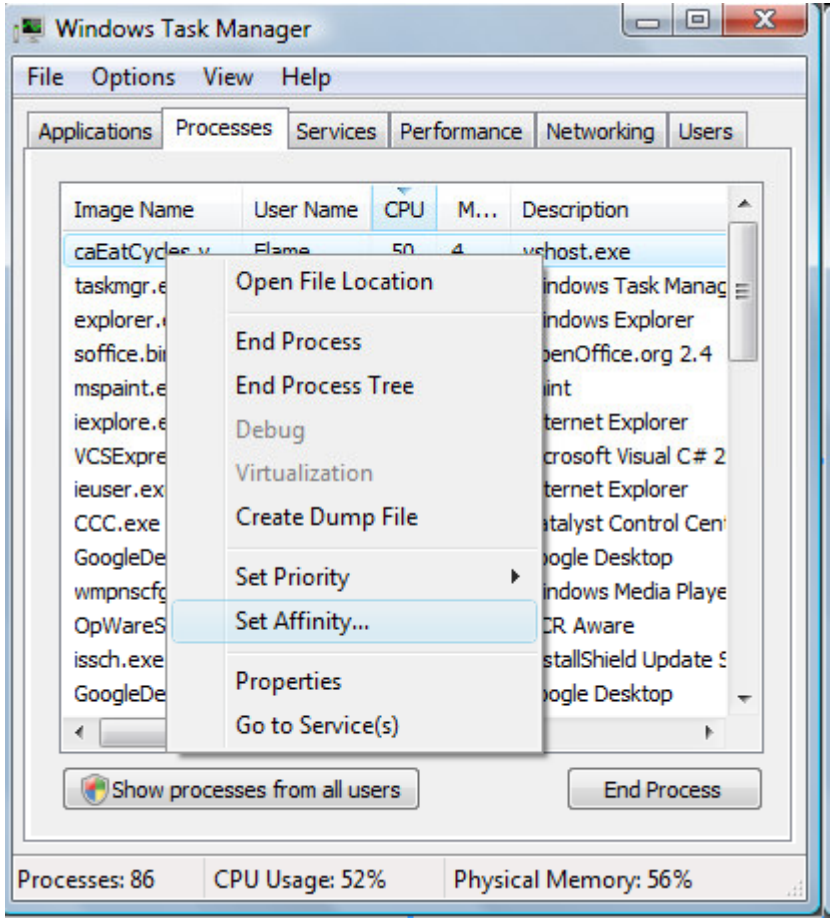
- Why is this useful?

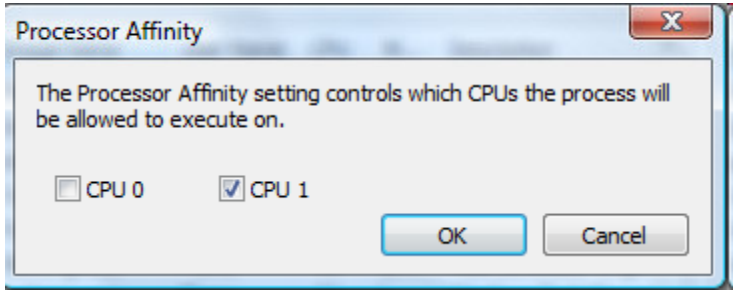
    - If the process is directed to always use the same core it is possible that the process will run more efficiently because of the cache re-use.

    - If one or two performance critical processes direct only to one core while all other processes are directed to other cores.

Setting Processor Affinity in Vista

Karoly.Bosa@jku.at





Processor Affinity on Linux

Karoly.Bosa@jku.at

- Command **taskset** is used to set or retrieve the CPU affinity of a running process given its PID or to launch a new executable/command with a given CPU affinity.

- Install schedutils :
  - Debian: apt-get install util-linux
  - Red Hat Enterprise: up2date schedutils or rpm -ivh schedutils*

- The CPU affinity is represented as a bitmask e.g.:
  - 0x00000001 is processor #0 (1st processor)
  - 0x00000003 is processors #0 and #1
  - 0x00000004 is processors #2 (3rd processor)

- For instance:
  - taskset 0x00000001 -p 13545

- Using –c flag instead of bitmask:
  - taskset -c 1 -p 13545
  - taskset -c 3,4 -p 13545