System i Architecture – Part 1

Module 2

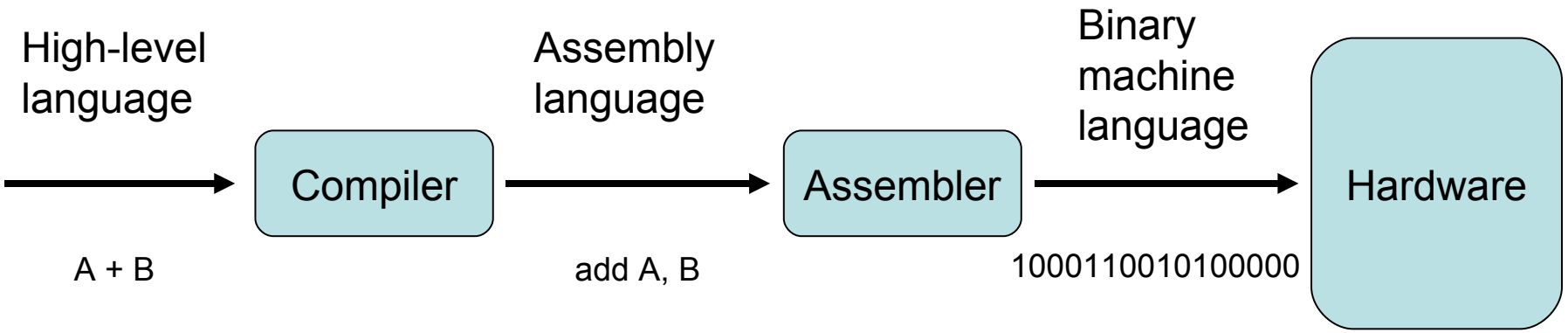
2.1 Impacts of Computer Design

If an instruction set architecture is to be successful, it must be designed to survive rapid changes in computer technology.*

An architect must plan for technology changes that can increase the lifetime of a successful computer.*

* Source: John Hennessy, David Patterson, Computer Architecture. A Quantitative Approach.

Hardware Dependencies



What happens by changing technology?

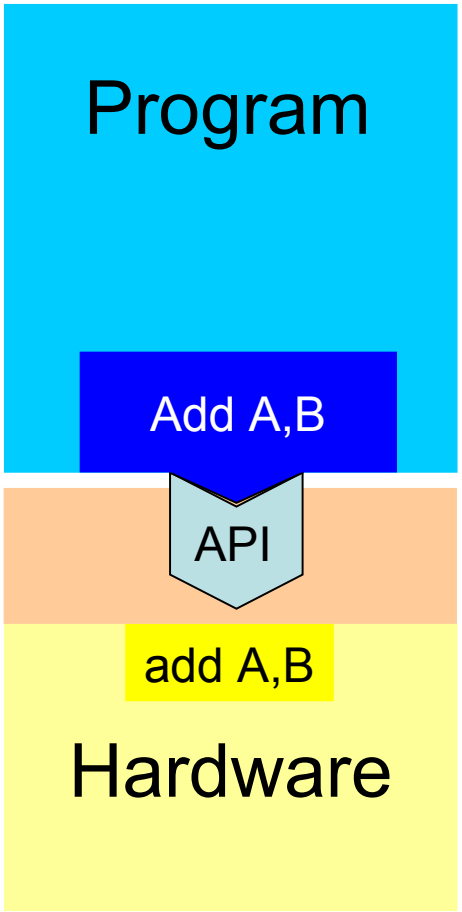
Recompile?

Rewrite?

Emulation?

Program sources?
Performance?
Usage advantages of new technology?
Costs?

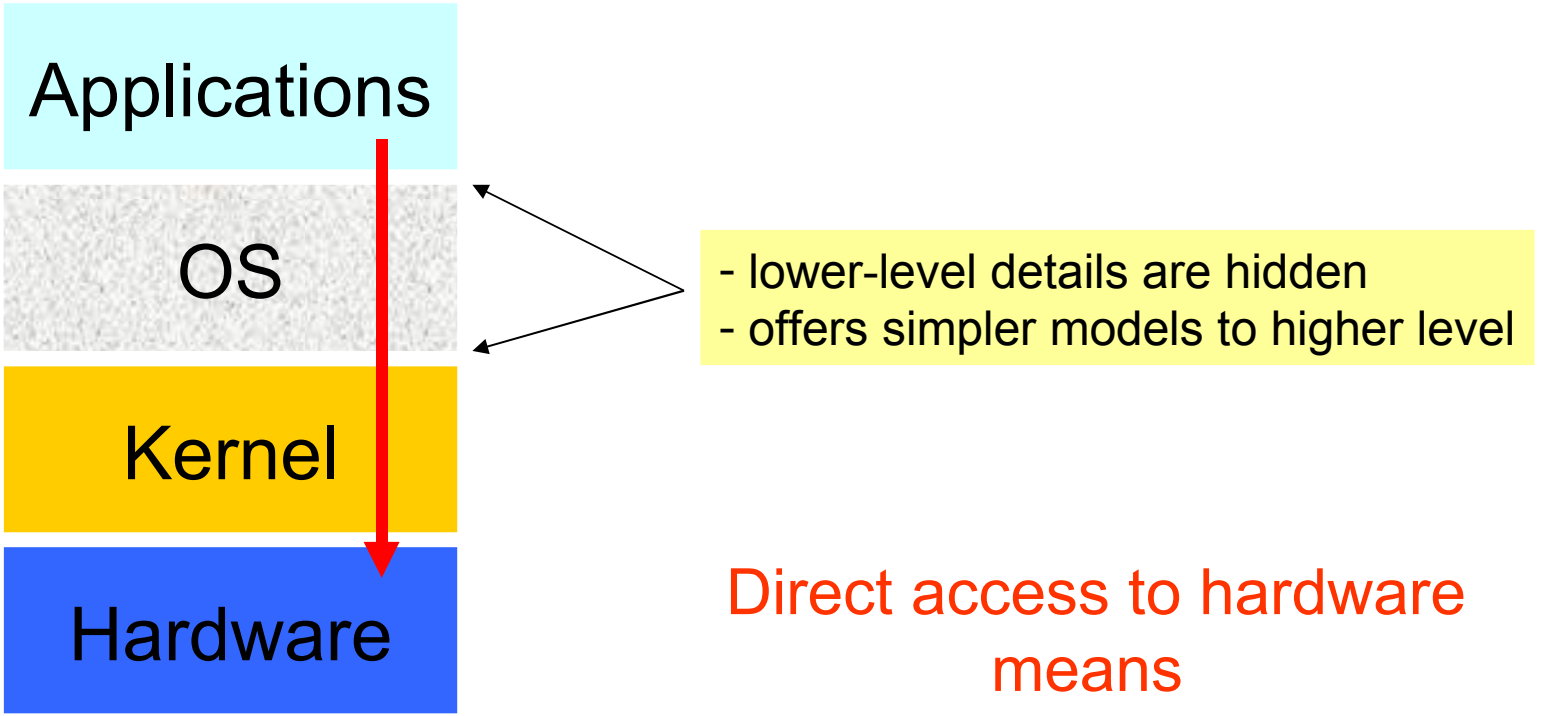
Application Programming Interfaces



- a set of definitions of the ways in which one piece of computer software communicates with another
- method of achieving abstraction

- Reduce hardware dependence
- Negative performance impacts

Hardware Abstraction Layer of OS



2.2 Goals of the System i Architecture

Different Workloads - Different Requirements

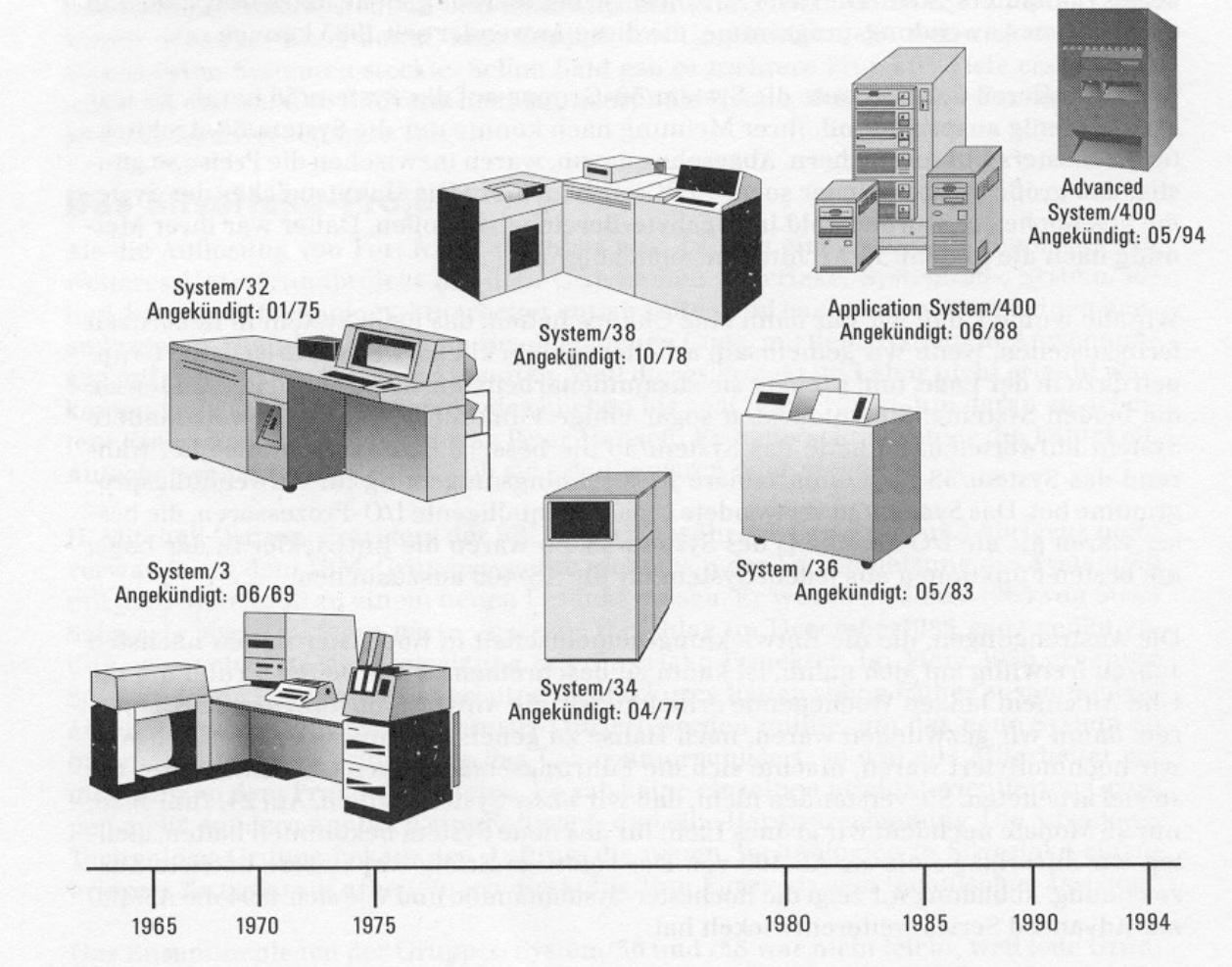
- Engineering/scientific Computing

  - Compute-intensive workload
  - Floating point processing
  - Using relatively small amounts of data
  - Many tight loops
  - I/O‘s more often sequential than random

- Commercial Applications

  - Many concurrent user
  - Primary using integer arithmetic, string comparison updates and inserts
  - Applications perform many calls to OS for services, such as I/O‘s
  - Fewer loops and more non-loop branches
  - Data often spread over a large amount of disk space

i5 Forerunner



Quelle: "AS/400 Blackbox geöffnet", Frank Soltis, Duke Press 1997

IBM Rochester (Minnesota)

The Secret of Success

Compact Cardless Computing

Advanced Architecture

Scaleable High Performance Computing

1969 1975 1977 1980 1982 1988 2000 2004

S/3 S/32 S/34 S/38 S/36 AS/400 iSeries i5

96 Col Card

Inexpensive Interactive Computing

Distributed Computing

Server Consolidation New Workloads

Virtualization

The Secret of Success

Compact
Cardless
Computing

Advanced
Architecture

Scaleable High
Performance Computing

1969 1975 1977 1980 1982 1988 2000 2004

S/3 S/32 S/34 S/38 S/36 AS/400 iSeries i5

96 Col Card

Inexpensive
Interactive
Computing

Server
Consolidation
New
Workloads

Virtualization

Note the secret to this success……

Gradual, not Radical Change……

Distributed
Computing

Design Requirements for System i

- **Goal: Business Computer**
  - Ease-of-use user interfaces
  - Ability to expand the system without an impact on business application software
  - Optimized for running business applications
    - IO intensive workload rather than computing intensive workload
  - Optimized for multi-user applications
  - High throughput (fast IO operations)

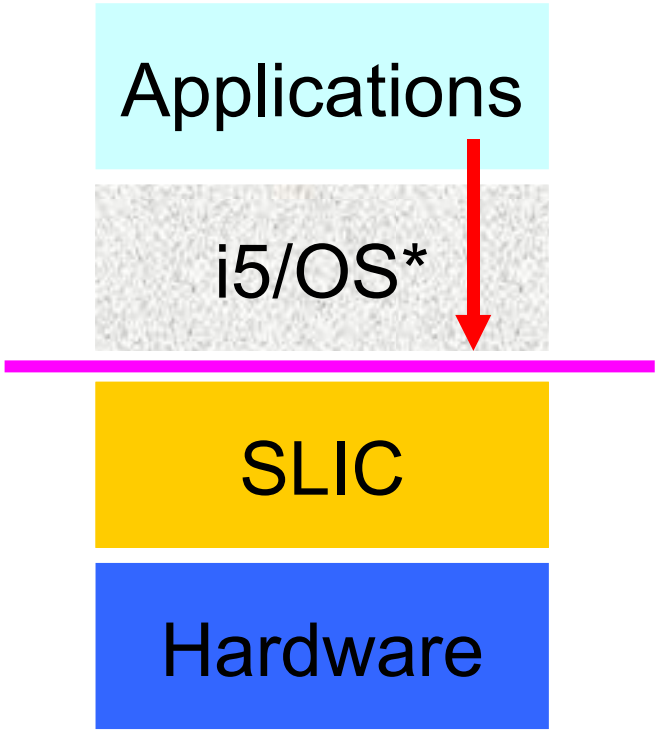
The Five Sacred Architecture Principles of System i

- Technology Independence
- Object-based Design
- Hardware Integration
- Software Integration
- Single-Level Store

2.3 Overview of the MI Architecture

* called Machine Interface on i5 Systems

Programmer's View on System i



Direct access to hardware is not allowed!

Machine Interface (MI)

- or Technology Independent Machine Interface (TIMI)
- logical not physical interface

System License Internal Code (SLIC)

- insulate applications from underlying hardware
- SLIC is hardware dependent!

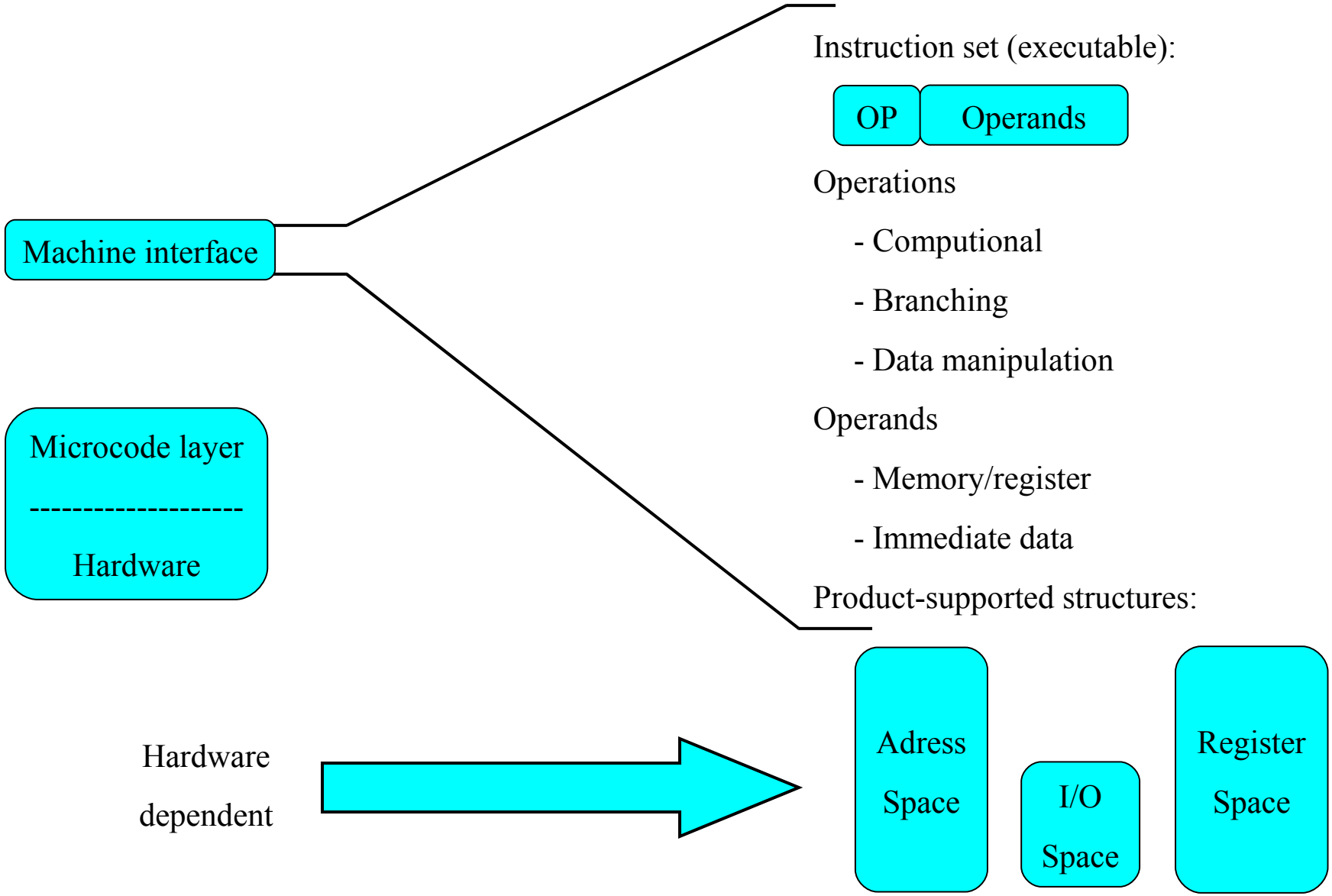
* Aka OS/400 on AS/400 and iSeries

System i Hardware Interface Architecture

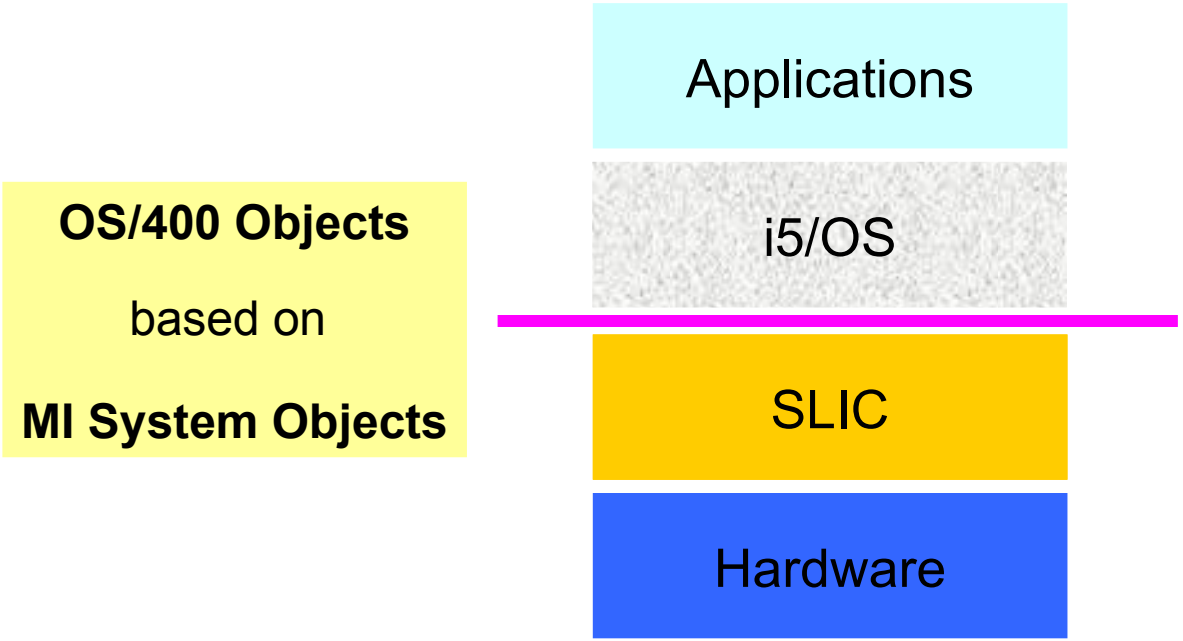
Designed for Software Investment Protection

- Technology Independent Machine Interface
- Software remains the same while
  - Processors change
  - Buses change
  - I/Os change
- Applications inherit advantages of new hardware enhancements
  - Processor speed and price/performance
  - Address space
  - Bus and I/O technology

Conventional Machine Interface



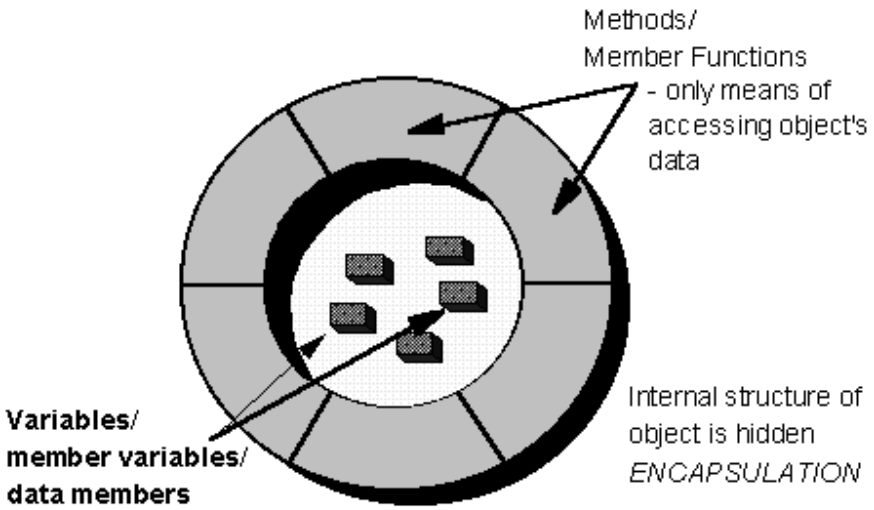
Use of Objects



- Objects are used for complex data structures

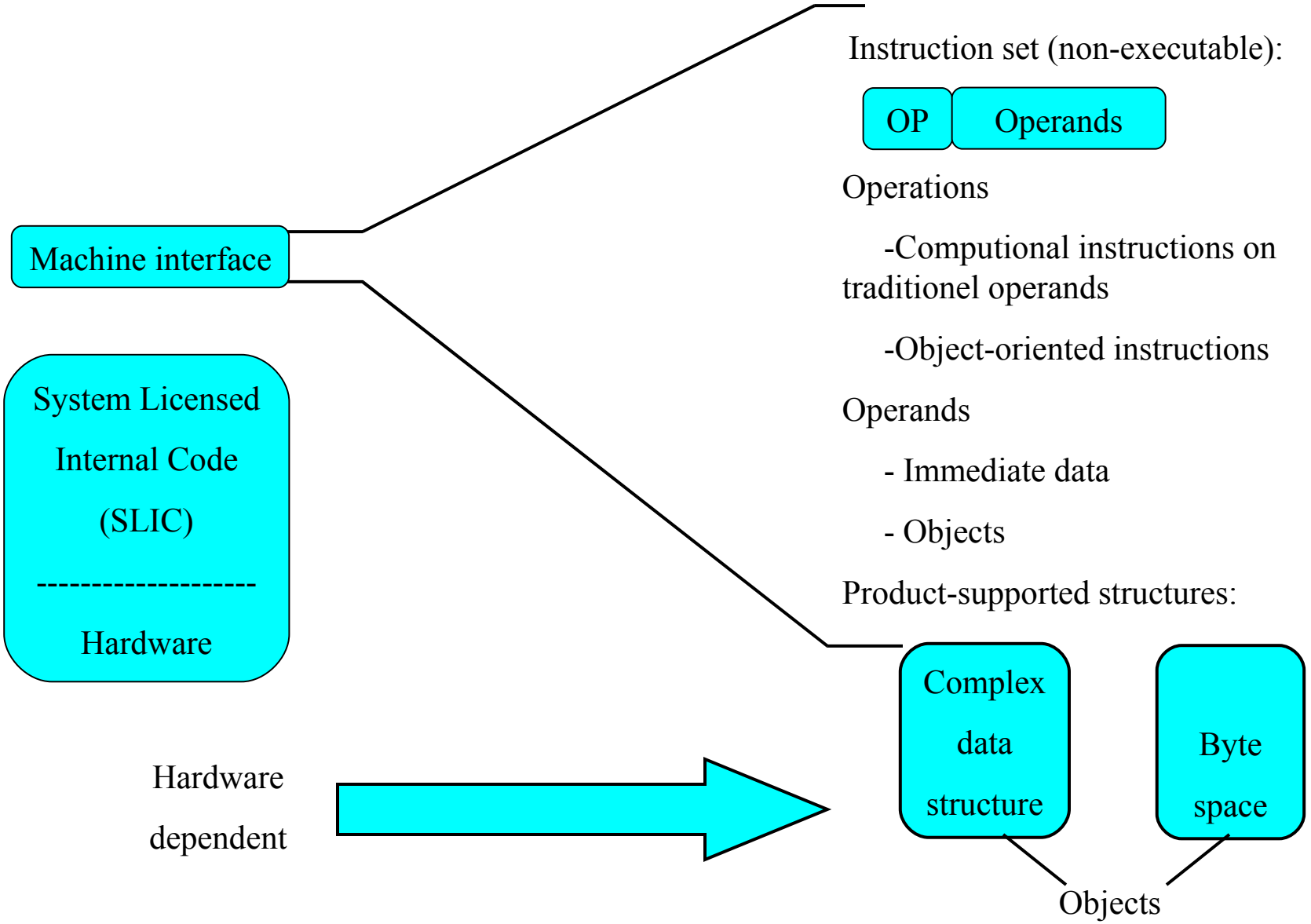
- Examples: Database file, User profile, Program, …

Hiding the Internals: Encapsulation



- **Programs no longer know about precise format of the data structure**
  - any changes of data structures have no effect on application or system programs
- **Only instructions that treat the object as entity are allowed**
  - fields in the data structure can not manipulated with bit- or byte-level instructions

System i Machine Interface



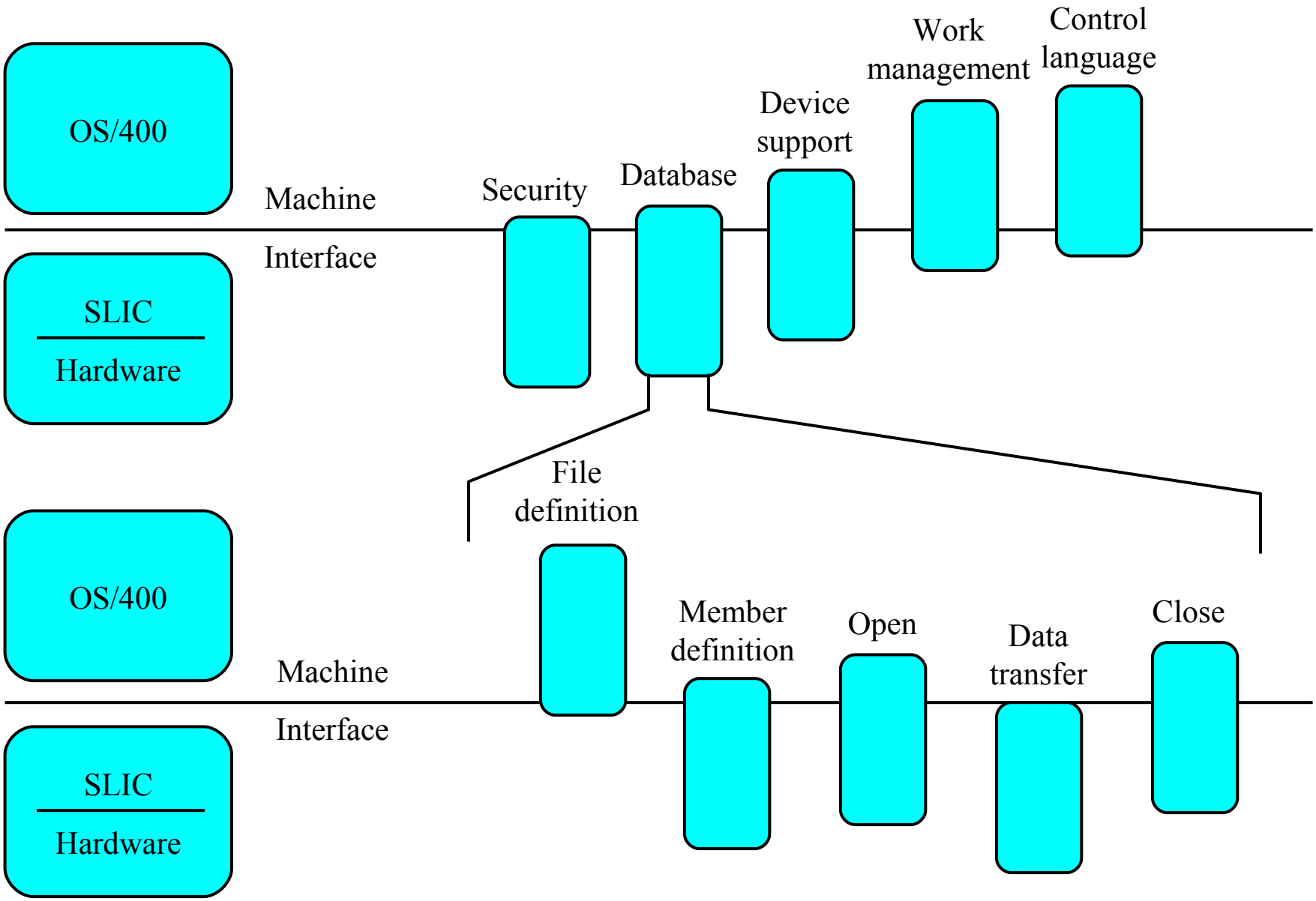
Pro‘s and Con‘s of using OO

- Advantages:
  - Technology Independance
    - Hardware changes have no impact on applications
  - Integrity
    - Programs at the MI layer do the right things
  - Security
    - Virus resistent

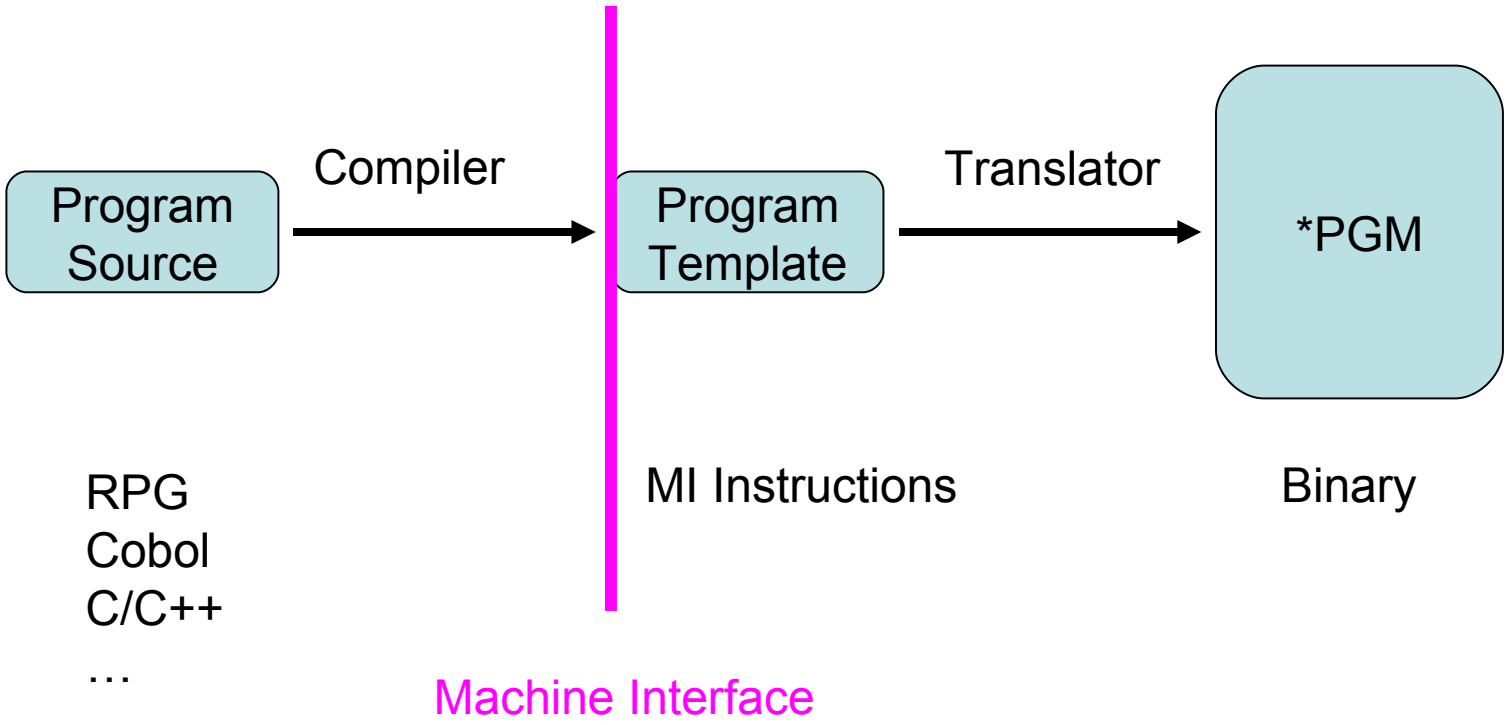
- Disadvantage
  - Performance
    - OO technology reuses many small modules
    - long instruction-path length
    - kernel performance has impact to the overall performance

Fine Tuning is important!

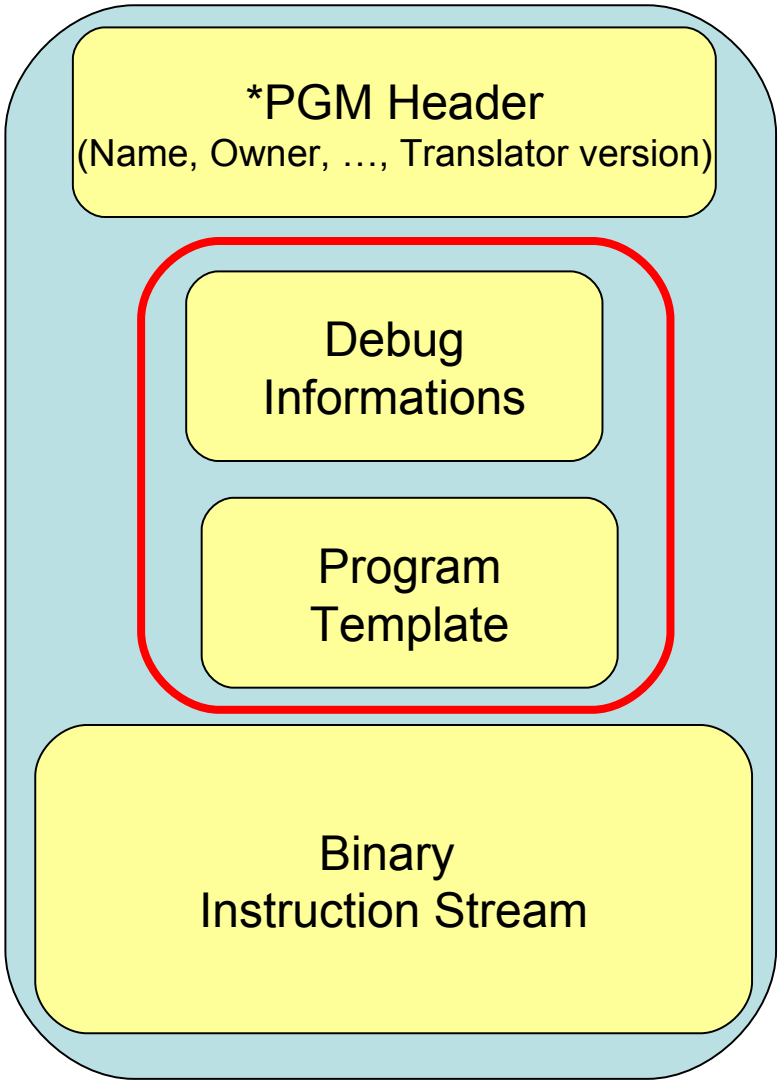
Operating System Functional Split



Creating a System i Program



Inside a *PGM Object



Observability

- Debug informations are not embedded in the code
- Template can be deleted

2.4 Examples

Using advantages of MI

2.4.1 From CISC to RISC

The way to a 64-bit Architecture

Instruction Set Classes

Example: C = A + B

Memory-memory	Register-memory	Register-register (load-store)
Add C, A, B	Load R1, A Add R3, R1, B Store R3, C	Load R1, A Load R2, B Add R3, R1, R2 Store R3, C

Data can be accessed directly

clocks/instruction vary by operand location

Source: John Hennessy, David Patterson, Computer Architecture. A Quantitative Approach.

CISC, RISC

- **Complex Instruction Set Computing**
  - Includes also complex instructions which were direct representations of high level functions
  - General goal: orthogonality of instruction set
  - Makes assembler programming easier

- **Reduced Instruction Set Computing**
  - Philosophy behind: Do everything in registers!
  - "Relegate Important Stuff to Compiler"

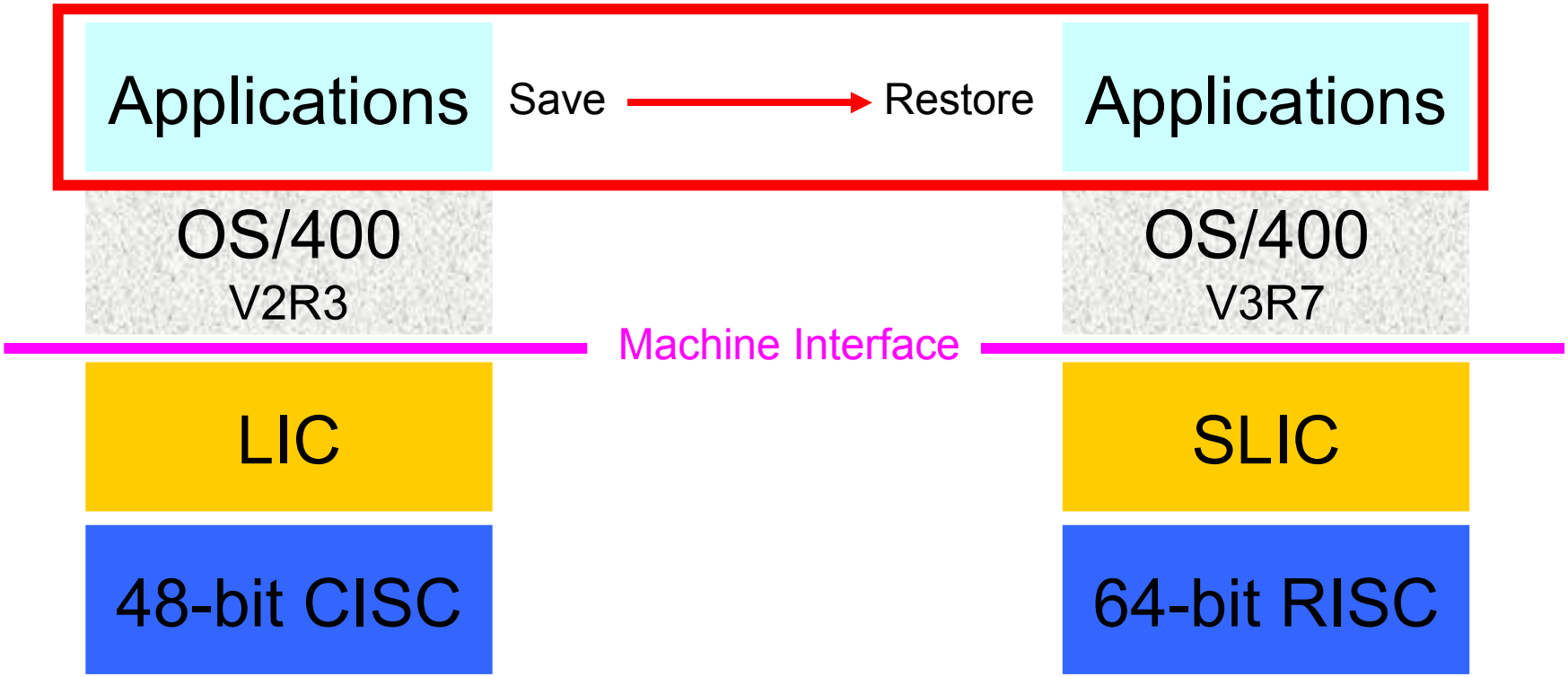
Exercise: CISC vs. RISC

Assume that a certain task needs *P* CISC instructions and *2P* RISC instructions, and that one CISC instruction takes 8T ns to complete, and one RISC instruction takes 2T ns.

Under this assumption, which one has the better performance?

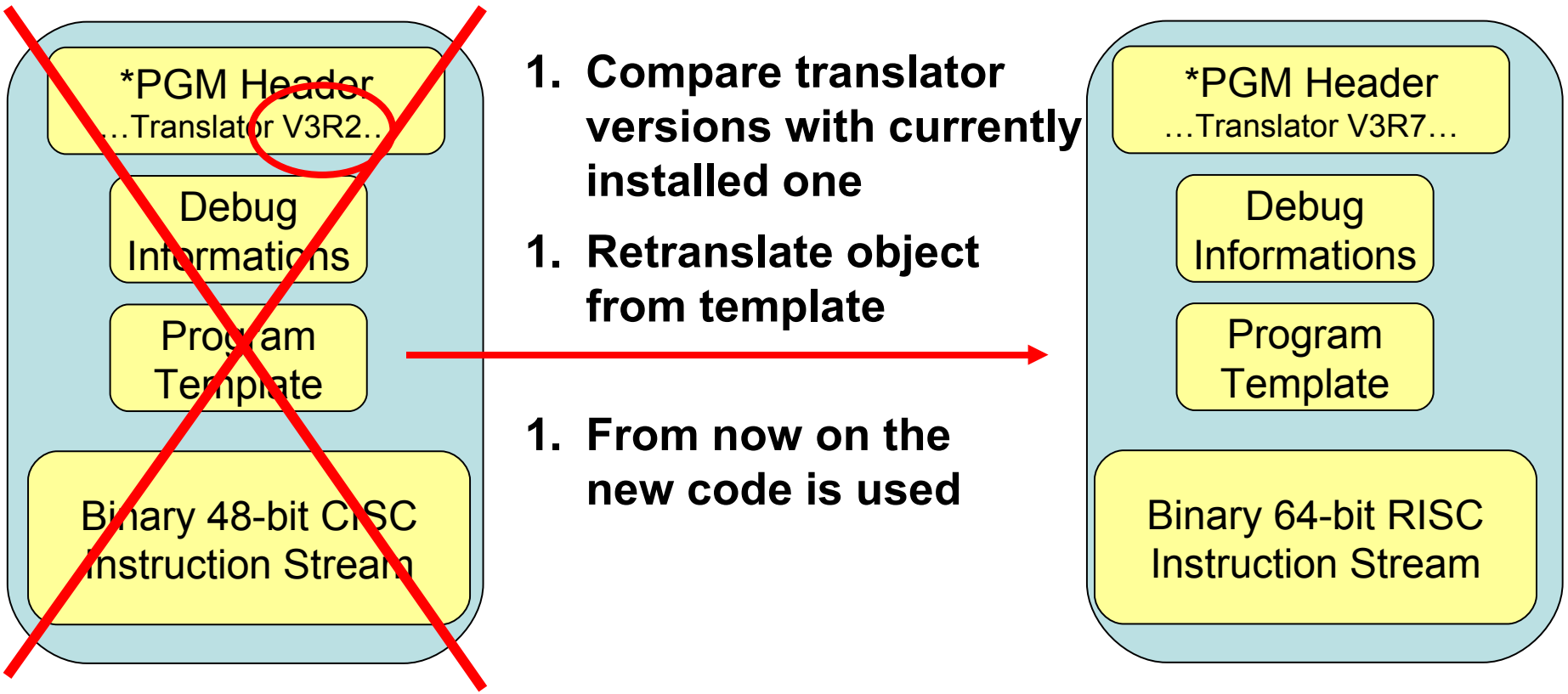
* Source: John Hennessy, David Patterson, Computer Organization and Design, Ex.1.52

A simple Way to 64 bit



How it was done?

Some customers called IBM and said:
“I just installed the new systems, and my applications run slower”.
What was wrong?



2.4.2 The Story of the Advanced S/36

IBM System/36

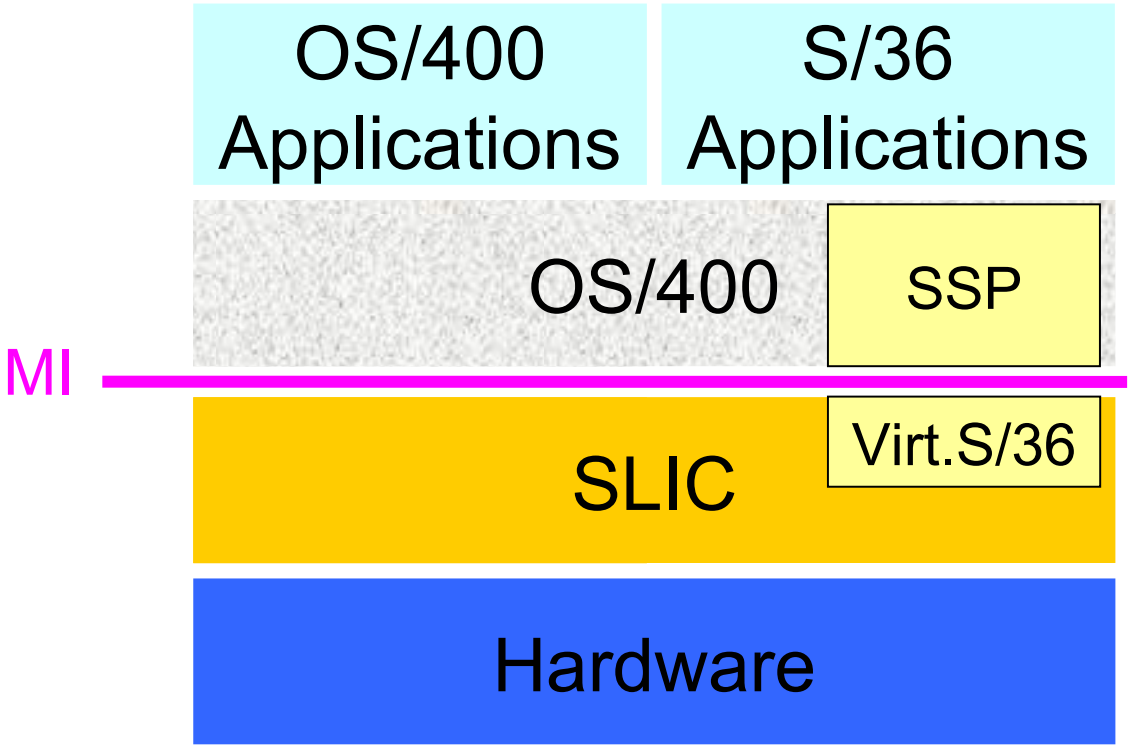
- **Was a simple and popular small business computer system,**
- **First shipped in 1983.**

The System/36 was flexible and powerful:

- It allowed **36 monitors and printers** to be connected together.
- **All users could access the system's hard drive or any printer**.
- It provided very good password security and resource security, allowing control over who was allowed to access any program or file.
- **Devices could be as far as a mile from the system unit**.
- Users could dial into a System/36 from anywhere in the world and get a **9600 baud connection**, which was very fast in the 1980s and very quick for connections which used only screen text and no graphics.
- It allowed the creation of databases of very large size. It supported up to about **8 million records**, and the largest 5360 with four hard drives in its extended cabinet could hold 1.453 gigabytes.
- The S/36 was "**bulletproof**," able to run for **six weeks or longer between restarts** (IPLs).

Advanced S/36

- IBM introduced the AS/400 Advanced/36 with PowerPC technology in 1994

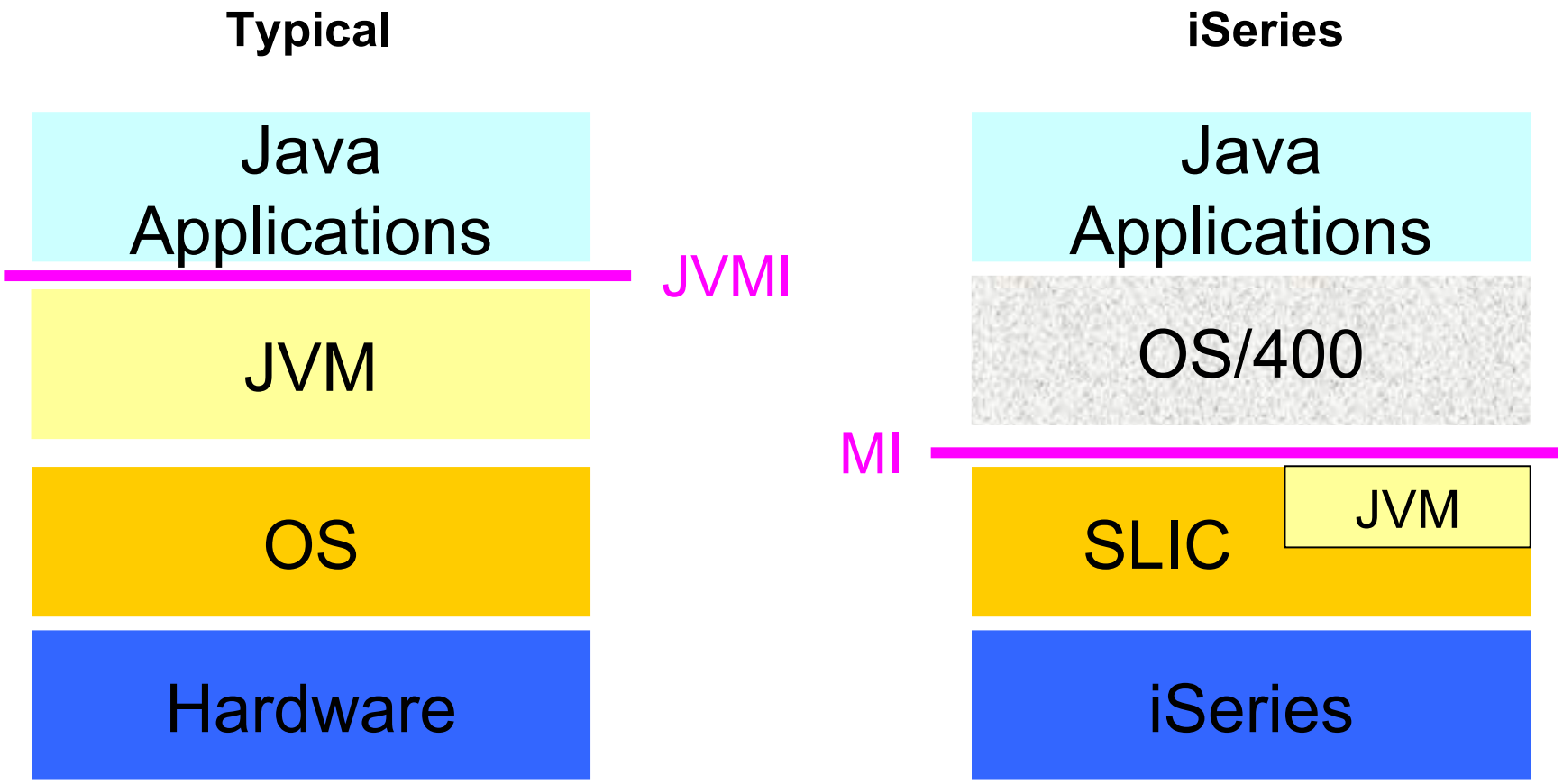


System Support Program
(S/36 Operating system)

- was available for all RISC Models with OS/400 V3R6 to V4R5
- not included in i5/OS

2.4.3 Java Implementation

Java Implementation



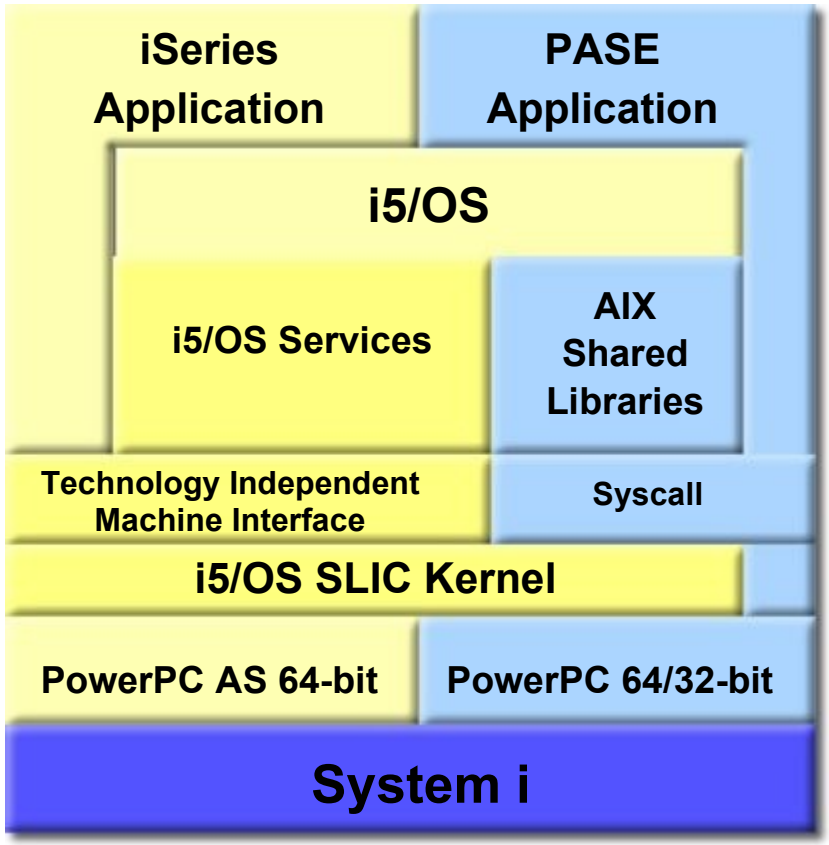
- Introduced with OS/400 V4R2
- New C++ implementation

2.4.4 AIX Runtime (without LPAR)

Portable Application Solution Environment

Portable Application Solution Environment (PASE)

- An integrated i5/OS[7] runtime for porting selected UNIX applications
  - –Supports AIX 5L 32/64-bit application model
  - –Exploits PowerPC's ability to switch runtime modes
- Applications
  - –Integrated with iSeries file systems and work management
  - –Can call DB2/400[7], Java™ and ILE programs
  - –Exploit all aspects of iSeries operations environment



2.5 MI Programming Example

Program Models

- Programs come in two flavors
  - original program model (OPM)
  - Integrated Language Environment (ILE)

  MI programs can be created only for the OPM environment!
  - If you require ILE support use ILE C and its built-in MI support
  - For OPM use Create Program QPRCRTPG API

Example: Return the larger of two packed arguments

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

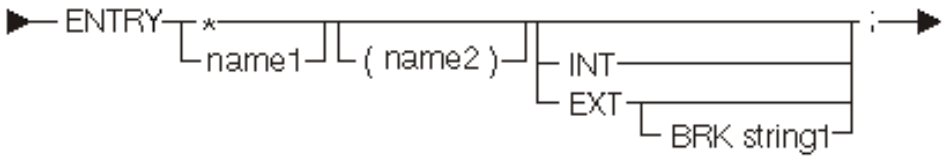
Define an Entry Point

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

First the program needs an ENTRY directive statement to designate its external entry point. The following directive declares an unnamed (the *) external (the EXT) entry point, which is called with a parameter list corresponding to PARM_LIST (defined later in the source code).

Entry Directive Statement

The following diagram and tables show the entry directive statement:



Keyword	Description
INT	Internal entry point.
EXT	External entry point.
BRK	Symbolic breakpoint is associated with the entry point.
*	Entry point defined has no name or is associated with the next MI instruction.

Constant	Range	Description
name1	Any	Entry point name being defined
name2	Any	Parameter list name for this entry point
string1	1-10 bytes	Breakpoint name

The default scope is internal (INT).

The entry statement defines entry point program objects. The next instruction number is associated with this entry point. The entry statement is to be the definition point for this object, so the ODT number assigned to this object is the next available ODT number.

Declare Arguments

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

OS/400 programs typically pass parameters by reference as part of the high-level language (HLL) calling convention. Because OS/400 programs pass by reference (that is, address and not value), the program also needs to define three space pointers (how storage is referenced) to represent the three parameters being passed.

To associate these three space pointers with the parameters being passed to the program, the following operand list (OL) is declared

Declare Paramater Types

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

Instruction Stream

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

The program then branches (the (B) extender to CMPNV) to label ITS2 if ARG1 is less than ARG2 (the /LO branch target). Other target keywords could be LO, HI, EQ, …
If ARG2 was greater than ARG1, the CPYNV instruction at label ITS2 is run, setting RESULT to the value of ARG2.

Program End

```
/* Program Name: MI01 */

ENTRY * (PARM_LIST) EXT;

DCL SPCPTR ARG1@ PARM;
DCL SPCPTR ARG2@ PARM;
DCL SPCPTR RESULT@ PARM;

DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT;

DCL DD ARG1 PKD(15,5) BAS(ARG1@);
DCL DD ARG2 PKD(15,5) BAS(ARG2@);
DCL DD RESULT PKD(15,5) BAS(RESULT@);

CMPNV(B) ARG1,ARG2 / LO(ITS2);
CPYNV RESULT,ARG1;
B RETURN;
ITS2: CPYNV RESULT,ARG2;

RETURN: RTX *;
PEND;
```

The program has now finished processing and ends.
The previous return external (RTX) instruction is not needed because it is implied by the PEND directive. The RTX instruction is included to add clarity to the program flow.

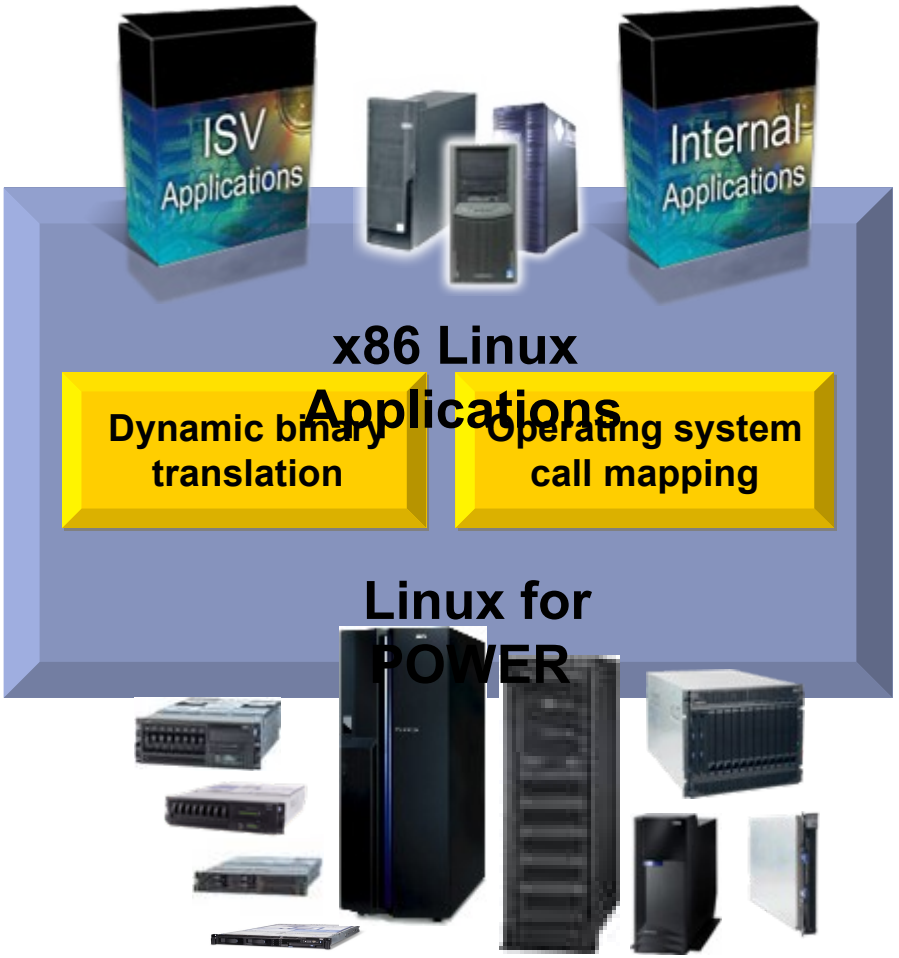
Lx86

Transforming x86 Linux applications without porting

Helps enable x86 Linux apps to "just run" on Power Systems servers with Linux OS

PowerVM Lx86 for x86 Linux applications

- Supports installation and running of most existing 32-bit x86 Linux applications[1]
- Creates an x86 Linux application environment running on Linux for System p
- Extends values of IBM Power Systems platforms to x86 Linux applications

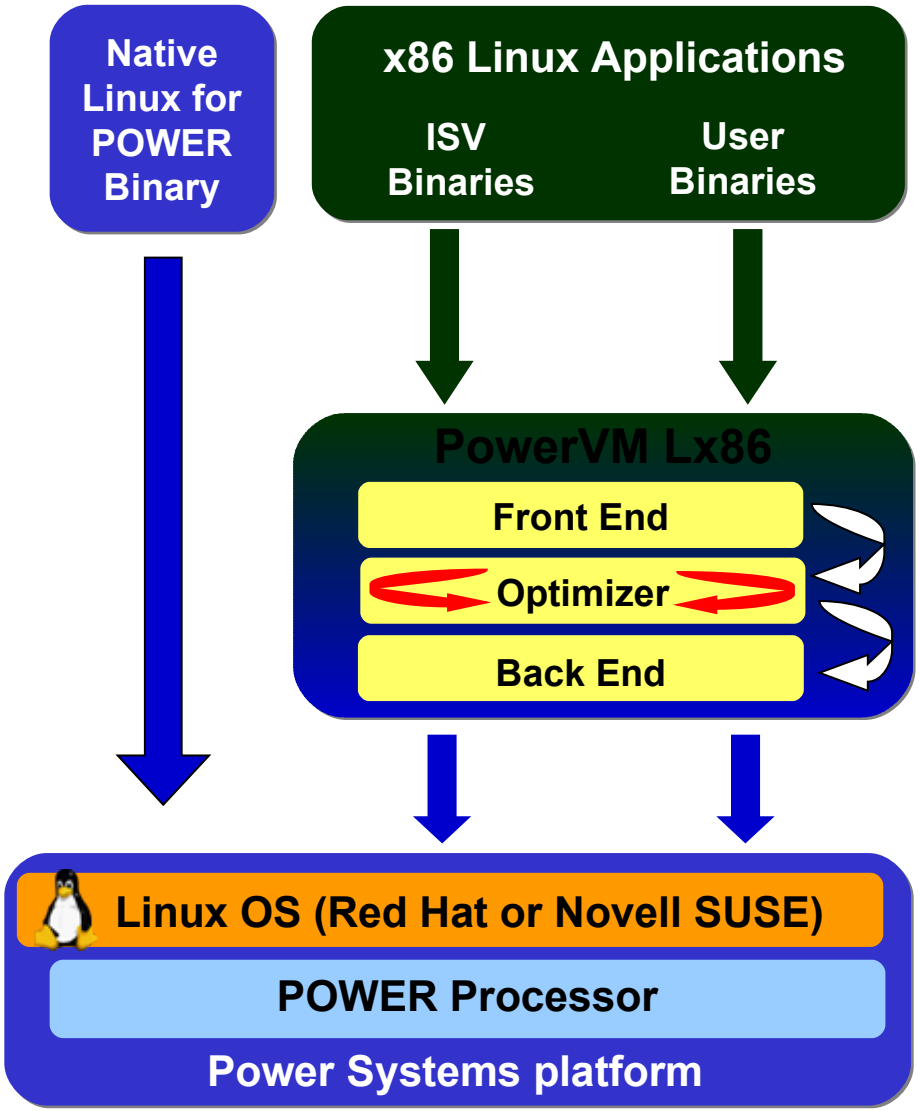


(1) System p Application Virtual Environment ("System p AVE")
runs most x86 Linux applications, but System p AVE cannot run applications that:

Directly access hardware (for example, 3D graphics adapters); Require nonstandard kernel module access or use kernel modules not provided by the Linux for POWER operating system distribution; Do not use only the Intel IA-32 instruction set architecture as defined by the 1997 Intel Architecture Software Developer's Manual consisting of Basic Architecture (Order Number 243190), Instruction Set Reference Manual (Order Number 243191) and the System Programming Guide (Order Number 243192) ll dated 1997; Are Linux/x86 specific system administration or configuration tools; Require x86 real-mode.

Visit ibm.com/systems/p/linux/qual.html for detailed qualifications.

What does PowerVM Lx86 do?



- **Dynamically translates and maps x86 Linux instructions to POWER**
- **Translation process**
  - Translates blocks of code into intermediate representation
  - Performs optimizations
  - Stores optimized, frequently used blocks of code in cache
  - Handles Linux OS call mapping
  - Encodes binary for target POWER processor platform
- **Best for certain applications and usage scenarios**
  - Power architecture can provide many advantages
  - But these make our architecture very different from x86 architecture
  - Translation can be resource intensive