Intel Itanium™ Porting Methodologies

Bill Chen

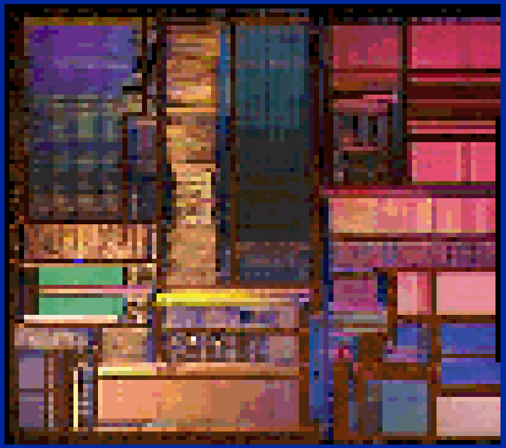
Intel China

Itanium™ Porting

- Why Port to Itanium™?
- Porting Process
- Porting Scenarios
- Porting Concerns
- Availability of porting tools by OS

Why Port to Itanium™?

- 64-bit virtual address space
- Addressing current architecture performance limitations
  - Inefficient parallelism
  - Branching
  - Procedure Calls
  - Memory latency



- Superior multimedia and FP performance

Intel Resources for Porting

- Application Engineer
  - Hands on technical assistance
- Training on SDV (Hardware and Software)
- Question and Answer Database (QuAD)
  - Web-based tech support by Intel experts

Intel Premier Support - Microsoft Internet Explorer provided by Intel Corp.
Intel Premier Support
Support home
Intel home
intel.
Intel Premier Support

Login

Welcome to Intel. Please enter your Intel Premier Support login and password below.

Login

Password

- Application Solution Center (ASC)
  - Lab-based technical assistance

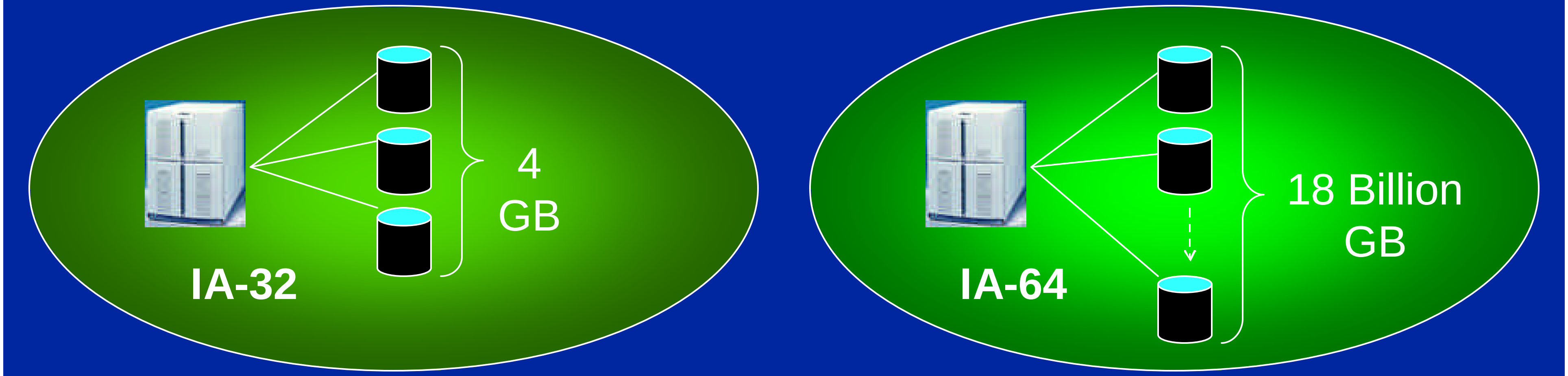
Itanium™ Porting Process

- Assess the Complexity of an Itanium™ Port
  - Identify dependencies
  - Analyze source code

- Develop a Porting Plan
  - Target platforms, training, resources
- Build Itanium™ executables
- Internal and External Testing

Porting Scenarios

- Full port with 64-bit pointers and 2^{64} address space
- Port with 64-bit pointers and <2 GB
- Unmodified IA-32 binaries

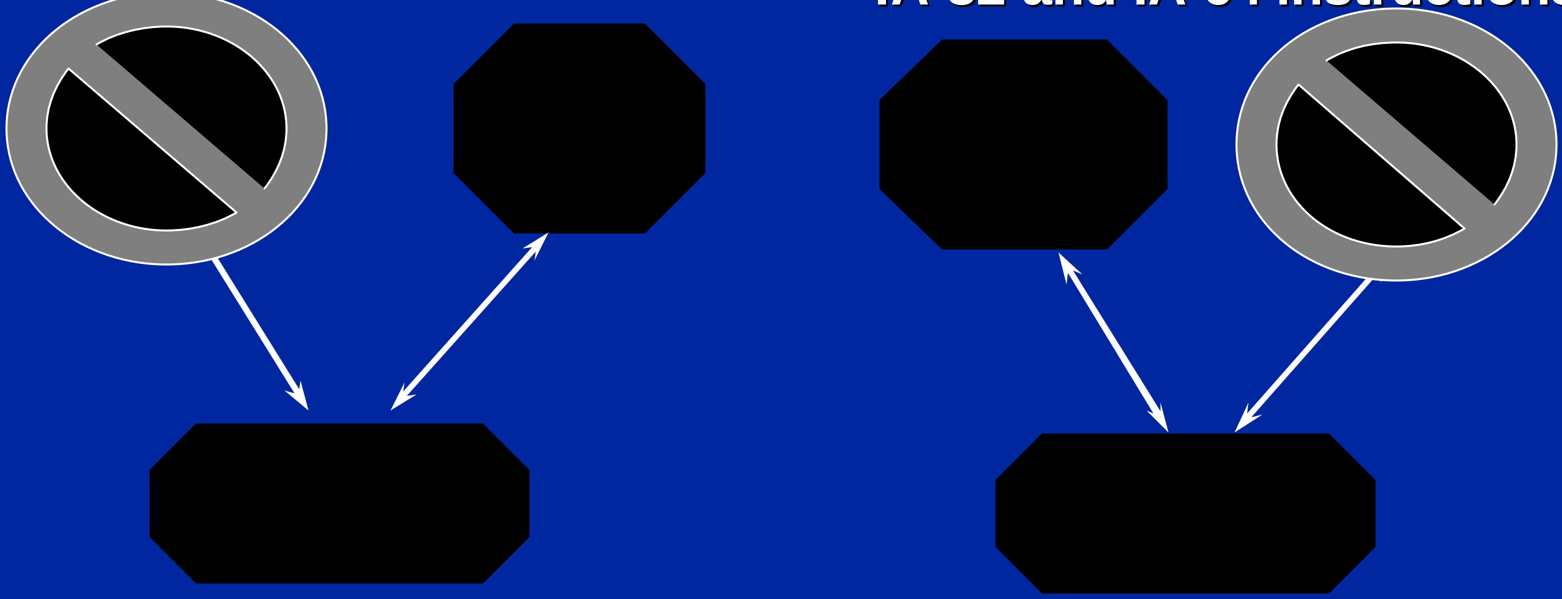


Complete full port for optimal performance on Itanium

* Third party names and brands are the property of their respective owners

Dynamic Library Interaction

- IA-64 library port
  - The library is being used by both IA-64 32-bit and 64-bit apps, but not IA-32 applications.
- IA-32 library
  - The library can only be used by IA-32 apps.
  - OS does not allow mixing of IA-32 and IA-64 instructions.



Help us identify your 3rd party libraries early

i.e. DLL

Windows & UNIX have diverged

- The 32-bit world: one, happy “ILP32” family
  - `int`, `long`, `void *` (pointer): all 32 bits, UNIX *or* Windows
    - Same (base) types for UNIX and Windows
  - Both have named types derived from the base types
    - UNIX: `pid_t`, `size_t`, `time_t`, `off_t`, ...
    - Win32: `LONG`, `HANDLE`, `WPARAM`, `LPARAM`, ...
- The 64-bit world has differences

OS	Data Model	`int`	`long`	pointer
UNIX/64	I32,LP64	**32**	64	**64**
Windows (Win64)	IL32,P64	**32**	32	**64**

UNIX & Windows *both* tried hard to minimize changes needed in existing source code; different derived type models resulted in `long` being different

C Programming Data Models

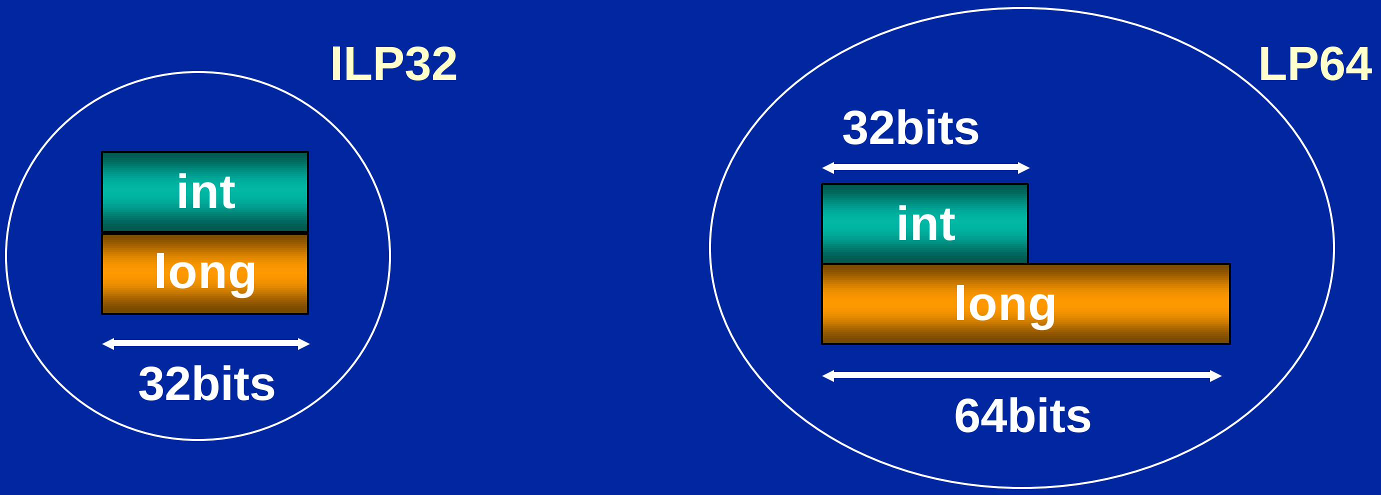
- OS Implements the Data Models
- ILP32
  - int, long and ptr are 32 bits
  - Used by 32-bit OSs
- LP64
  - int is 32 bits
  - long and pointer are 64 bits
  - Used by 64-bit UNIX OSs
- P64 (or LLP64)
  - int and long are 32 bits; pointer is 64 bits
  - Used by Win64* and Modesto*

	ILP32 size (bits)	LP64 size (bits)	P64 size (bits)
int	32	32	32
long	32	64	32
pointer	32	64	64

* Third party names and brands are the property of their respective owners

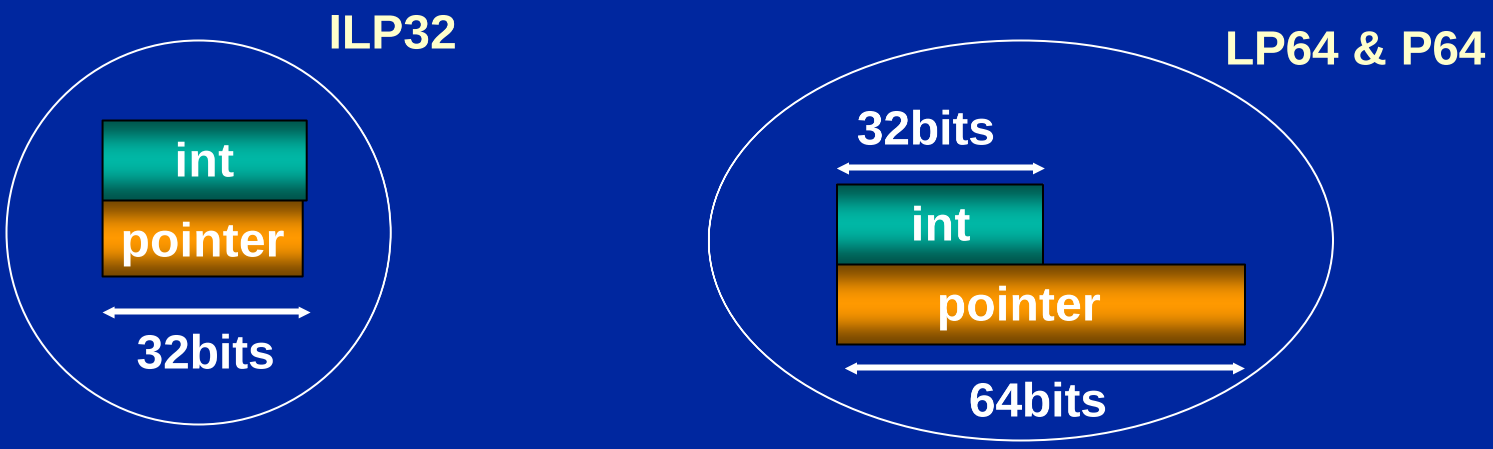
Porting Concerns (UNIX only)

- longs and ints are not the same size
  - Truncation of 64 bit value when assigned to a smaller type
  - Explicit cast improperly applied
  - A int pointer is not compatible with a long pointer
  - Lack of prototyped function declarations in scope of call statements
  - Untyped integral constants are int by default



Porting Concerns

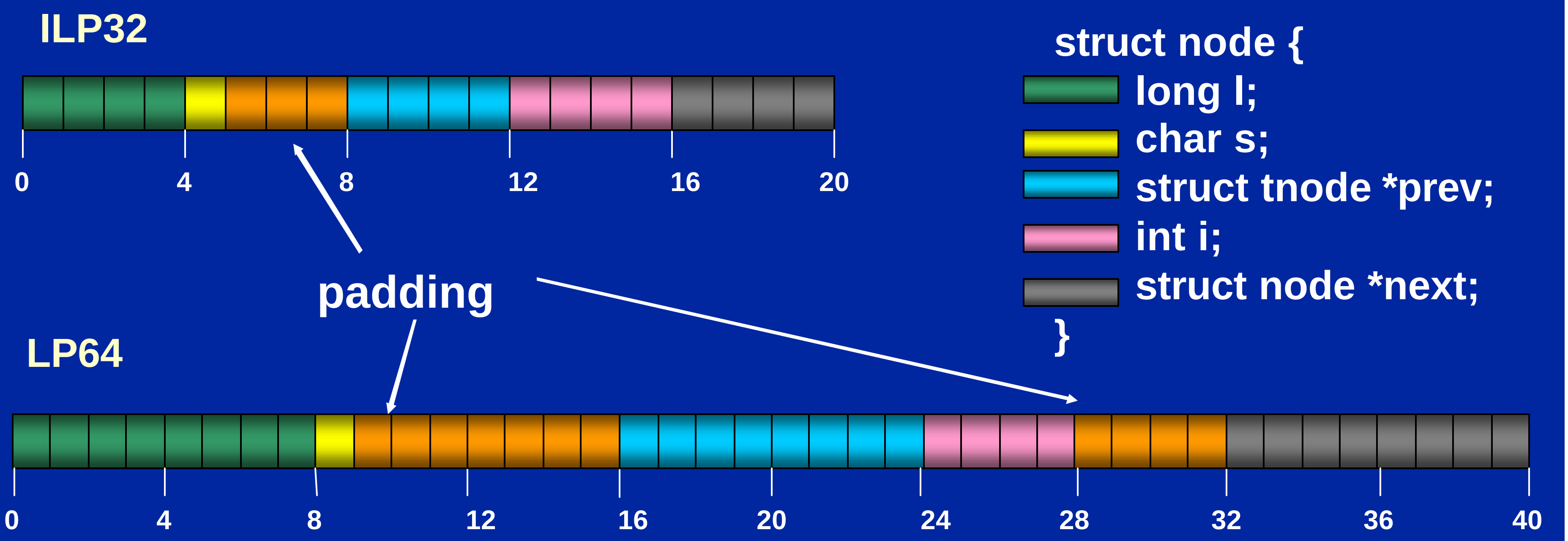
- pointers and ints are not the same size
  - Truncation of a 64 bit pointer when converted to a smaller type
  - Assumption that pointers and int are same size in arithmetic context
  - Pointer return types in the absence of a function prototype



* Third party names and brands are the property of their respective owners

Porting Concerns

- Pointers/longs are 64 bits and 64-bits aligned
  - problems with data sharing because objects grow
  - data could be shared through IPC, network or disk



```
struct node {
🟩 long l;
🟨 char s;
🟦 struct tnode *prev;
🟪 int i;
⬛ struct node *next;
}
```

Porting Concerns

- Usage of undocumented/reserved bit fields
- Fix unguarded “#ifdefs" from defaulting to unwanted code generation
- Assembly code
- Self modifying code
- Portions of code utilizing data-packing

Examples

* Third party names and brands are the property of their respective owners

Microsoft WinXP* 64-bit Edition

- Porting Tools
  - Windows 2000* Platform SDK
  - Intel and Microsoft* C++ compilers
  - Intel Fortran 90 compiler
  - Intel Enhanced Debugger
  - MigraTEC Migration Workbench

- Links
  - http://www.microsoft.com/windows2000/future/64bit/64bit.asp
  - http://msdn.microsoft.com/library/default.asp?URL=/library/psdk/buildapp/64bitwin_410z.htm

* Third party names and brands are the property of their respective owners

Linux

- **Porting Tools**
  - HP/Intel: IA-64 Linux Simulator
  - VA Linux Systems: Sourceforge Itanium(tm) Processor Compile Farm
  - Red Hat: GNU tool chain
  - Red Hat Linux 7.1 for the Itanium^{TM} Processor
  - SGI: Pro64(tm) Compiler Development Tools
- **Links**
  - http://www.software.hp.com/ia64linux.htm
  - http://www.sourceforge.net/compilefarm
  - http://www.cygnus.com/ia64
  - http://www.linuxia64.org
  - http://www.redhat.com/products/software/linux/7-1_itanium.html
  - http://oss.sgi.com/projects/Pro64

HP-UX* 11i

- Refer to the other tracks at this event

* Third party names and brands are the property of their respective owners

Call to Action

- Identify 3rd party dependencies
- Develop porting plan
  - Get your developers trained on Itanium based tools
  - Select porting scenario/model for each application
- Get Software Ready for the hardware

Backup

Examples: Use #if defined appropriately

```
#if defined(_WIN32)
… //  stuff related to Win32
#if !defined(_WIN64)
… //  Win32 without Win64 (regular Win32)
#else /* is _WIN64 also */
… //  Win64 variant of Win32
#endif /* _WIN64 ? */
#elif defined(__unix) || …
… //  various UNIXes
#else /* some other OS */
#error Unhandled OS;
#endif
```

Enhance portability, readability: account for all OSs

Back

Examples: Do not #define constants

```
#define mask 0x37FFC;
```

```
const int mask= 0x37FFC;
```

Problem(s):

- using a #define that the compiler can’t type check

Remedy:

- use ANSI C’s “const”
- use a specific data type
- you’ll get warned if any misuse is attempted

Let the compiler check declarations for you

Back

Examples: Watch for Hex constants

```
0xFFFFFFFF
    // 32-bits: -1, 64-bits: 4,294,967,295
0x100000000
    // 32-bits: 0, 64-bits: 4,294,967,296

const int all1s= 0xFFFFFFFF;
```

Problem(s):

- generating "all 1s" in hex
- using a #define that the compiler *can't* type check
- "-1" in 32-bit system, 4,294967,295 in a 64-bit system

Remedy:

- use ANSI C's "const"
- use a specific data type
  - signed/unsigned
- use type suffixes – "L", "UL"

Count the digits!

Back

Examples: Watch for Pointer truncation

```
mystruct *p;


unsigned int lowBits=
  (unsigned int)p;
  // truncation warning in Win64


unsigned int lowBits=
  PtrToInt(p);
  // truncation warning silenced


p= (mystruct *)lowBits;
```

- only do this if you *really* have to…

Problem(s):

- pointer truncations dirty your compilation listings

Remedy:

- Windows' `PtrToInt()` silences warnings
- see <`basetsd.h`>

Caution:

- Never, ever use data as pointer again; significant bits are *gone*

Be careful if you forcefully silence warnings

Back

Examples: Don’t cast Pointers to integer type

```
char *buf;
…
int i;
…
i= (int)buf;
```

```
uintptr_t ip;
…
ip= (uintptr_t)buf;
```

Problem(s):

- Pointers are bigger than ints in some architectures
- Using `long` won’t help in Win64
- Pointers *logically* unsigned

Remedy:

- Use `uintptr_t`; works on both UNIX and Windows

eliminate *all* cases of `(int)pointer casts`

Back

Examples: printf()format strings

```
long *Pl; // Win32 source
printf(“%08lX->%ld\n”,Pl,*Pl);

#ifdef _WIN64
#define FMTSZ3264 “I64”
#else /* Win32 or UNIX */
#define FMTSZ3264 “l”
#endif
```

Problem(s):

- “l” argument size specifier used with platform-scaled type
- don’t use “%X” for pointers
- “I64” does not scale – it is *not* polymorphic

Remedy:

- use “%p” to print a pointer
- use a macro and adjacent string catenation

```
__int3264 Pl;
printf(“%p->%” FMTSZ3264 “d\n”,Pl,*Pl);
```

Fix printing of pointers and “big” integers

Back

Examples: Don’t use `union`

```
union {
  long l;
  char bytes[4];
  };
...
for (i= 0; i<4; i++) …
```

```
union {
  __int3264 l; // chg w/ architecture
  char bytes[sizeof(__int3264)];
  };
...
for (i= 0; i<sizeof(bytes); i++) …
```

Problem(s):

- union for alternate access method incorrect for 64-bits

Remedy:

- 1st: avoid union if at all possible

else:

- fix primary data type size
- use C `sizeof( )` builtin for array

unions look ugly and cause lots of problems; don’t use them unless necessary

Back

Examples: Appropriate Field Indexing

```
struct S {
  void *pn;
  int ln;
};
S *Ps= new(S);
int i;
i= *(int *)((uintptr_t)Ps + 4);

i= *(int *)((uintptr_t)Ps +
  offsetof(S,ln));
```

padding varies with architecture

Problem(s):

- field offsets can vary across compilers
- any constant added to a pointer should be suspect
- natural alignment differs

Remedy:

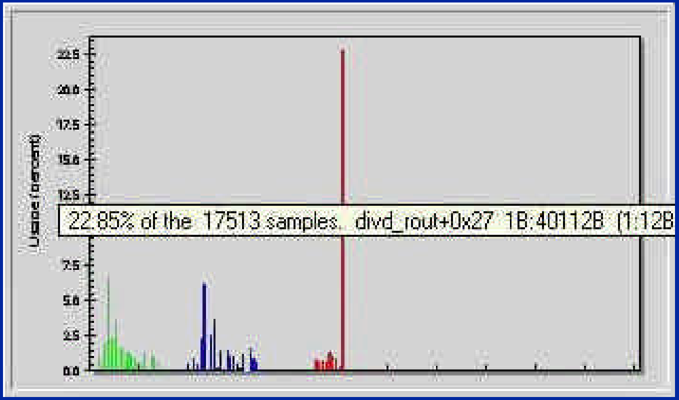
- use ANSI C `offsetof()` macro – ***not sizeof()***

Let the compiler calculate field offsets

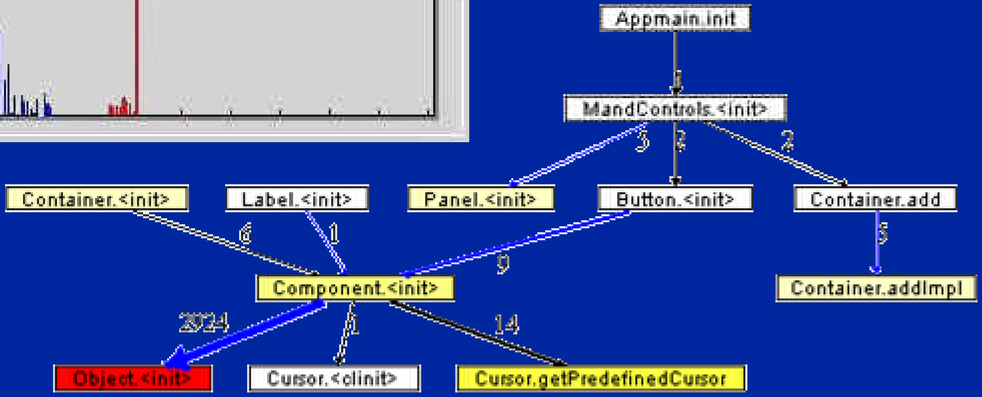
Back

ASC Performance analysis

- Runtime analysis gives an accurate picture
  - application “hotspots”, call graph
- Tools:
  - VTune
  - Quantify
  - APIMON
  - …



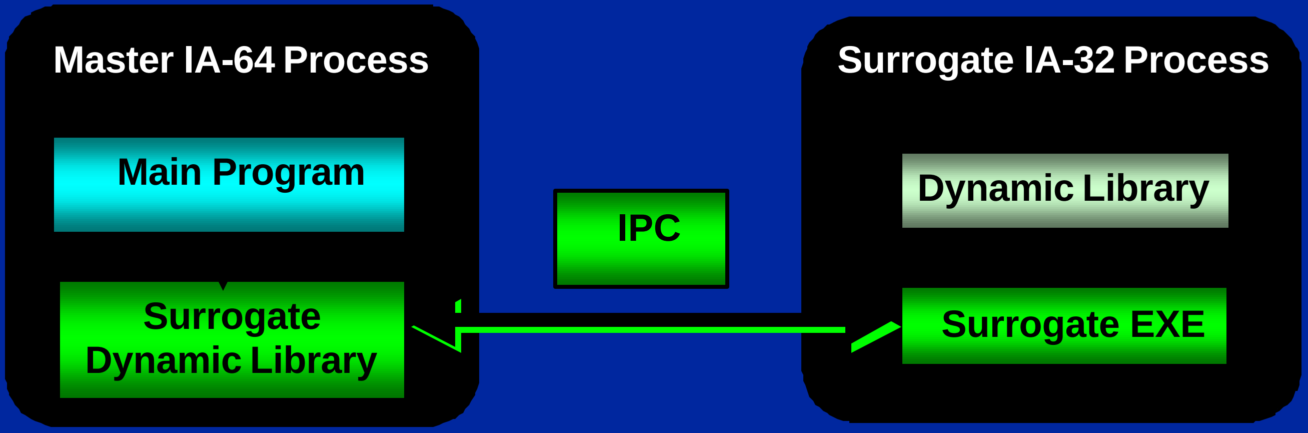
22.85% of the 17513 samples. divd_rout+0x27 1B:40112B (1:12B



Tools can help you pinpoint *best-benefit* spots

Dynamic Library Interaction

- 32-bit library access to 64-bit process through IPC
  - Surrogate binaries can be used to manage the IPC translation with no changes to existing code



* Third party names and brands are the property of their respective owners

Back