



**PLEASE STAND BY**



# The Itanium® Architecture

## A Technical Overview

**Thomas Siebold**  
**Technical Consultant**  
**Transition Engineering & Consulting**  
**Business Critical Server Division**

**[thomas.siebold@hp.com](mailto:thomas.siebold@hp.com)**

**Rev. 6.5**

© 2004 Hewlett-Packard Development Company, L.P.  
The information contained herein is subject to change without notice





# The Itanium® Architecture

## A Technical Overview

**Thomas Siebold**  
**Technical Consultant**  
**Transition Engineering & Consulting**  
**Business Critical Server Division**

[thomas.siebold@hp.com](mailto:thomas.siebold@hp.com)

**Rev. 6.5**

© 2004 Hewlett-Packard Development Company, L.P.  
The information contained herein is subject to change without notice



# Language and cultural differences

This is a ,mobile phone'

...but in Germay it is called a  
,handy'

...but in other countries a  
,handy' is a .....



# Agenda

- **Terminology**
- **Itanium® Roadmap**
- **The Itanium® Architecture**



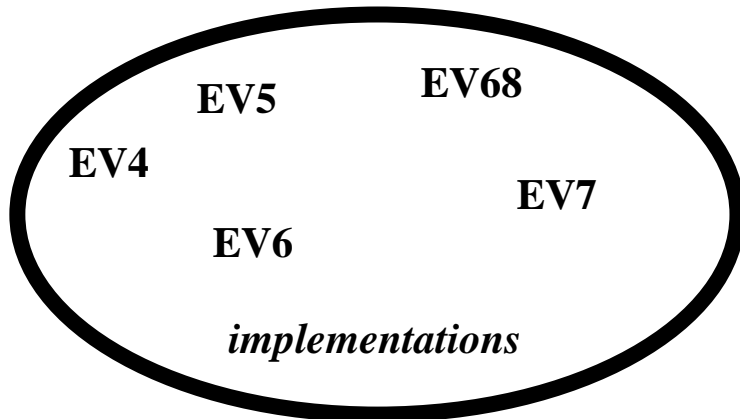
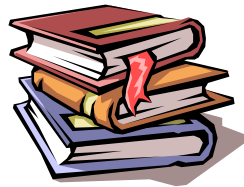


# Terminology



# Processor Architectures and Implementations

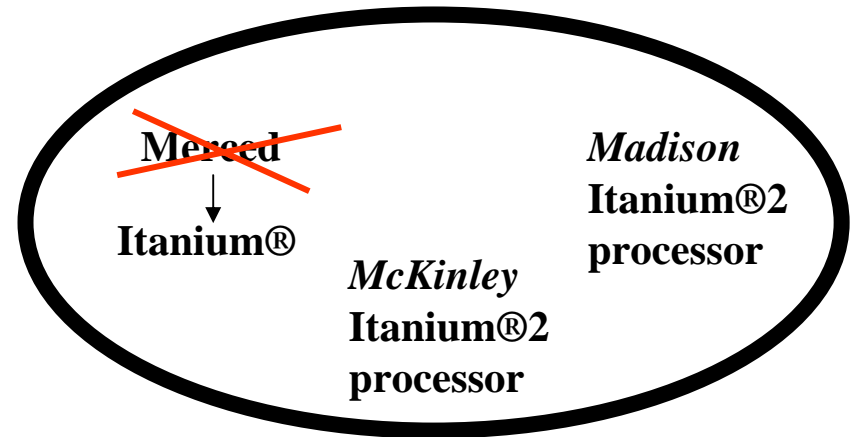
## Alpha Architecture



## Alpha® Processor Family

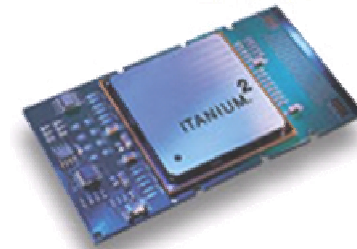
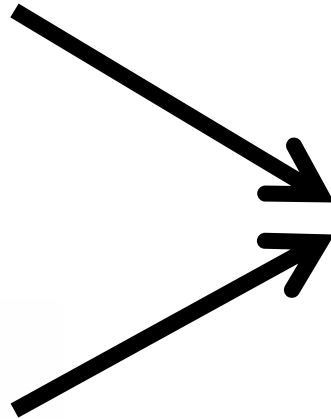
## ~~IA64 Architecture~~

## Intel Itanium® Architecture



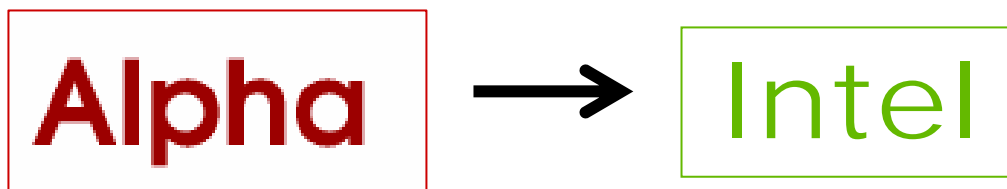
## Itanium® Processor Family

# Working Together





# Continue Working Together



- Alpha technology/resources enhance Itanium®-based **compilers/tools (SW)**
- Alpha technology/resources accelerate and enhance Itanium® Architecture **processors/platforms (HW)**

# Intel® Itanium® Processor Family Roadmap



**Performance**

**Lowest Cost of Ownership**

## Multi-Processor (MP) Capable *Leading Performance*

<p><b>Itanium® 2 Processor</b> 1.5GHz, 6M; 1.4GHz, 4M; 1.3GHz, 3M</p>	<p><b>Itanium® 2 Processor</b> (Madison 9M) &gt;1.5GHz, 9M</p>	<p><b>Montecito</b> <i>Dual Core, 24MB, 90nm Technology</i></p>	<p><b>Tukwila</b> <i>Multi Core, Developed with ex-Alpha team</i></p>
---	--	---	---

## Dual Processor (DP) Capable *Leading \$/FLOP*

<p><b>Itanium® 2 Processor</b> 1.4GHz, 1.5M, DP</p>	<p><b>Itanium® 2 Processor</b> &gt;1.4GHz, DP</p>	<p><b>Future DP</b></p>	<p><b>Future DP</b></p>
---	---	-------------------------	-------------------------

## Dual Processor (DP) Capable *Lower Power*

<p><b>LV Itanium® 2+ Processor</b> 1.0GHz, 1.5M, DP</p>	<p><b>LV Itanium® 2 Processor</b> &gt;1.0GHz, DP</p>	<p><b>Future DP, Low Voltage</b></p>	<p><b>Future DP, Low Voltage</b></p>
---	--	--------------------------------------	--------------------------------------

2003

2004

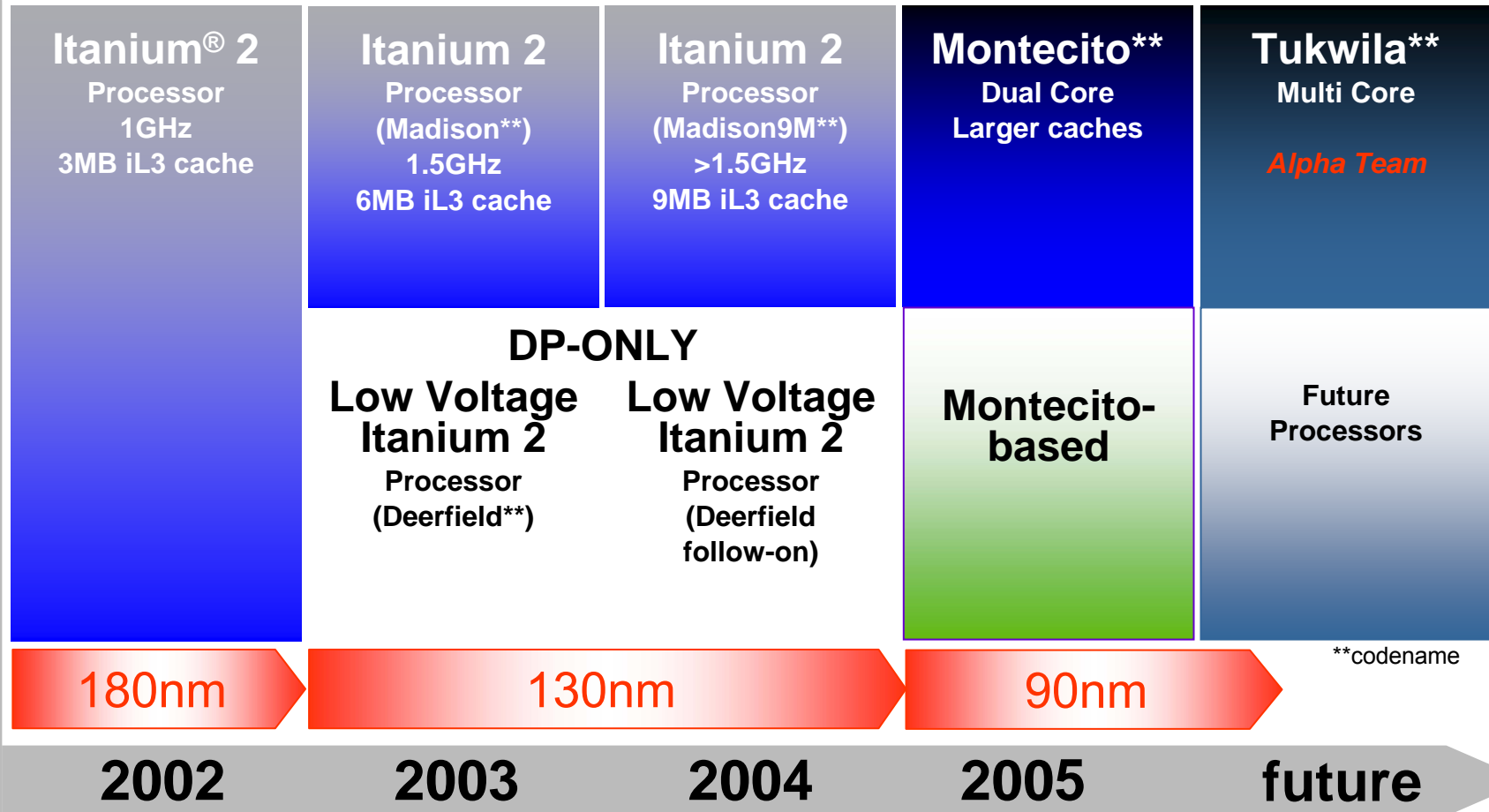
2005

Next Generation

**Long term Itanium® Roadmap Strength**

# Delivering on the Architecture

## MP/DP CAPABLE



All dates specified are target dates, are provided for planning purposes only and are subject to change.

# Itanium

## Itanium™ Processor

**System Bus**  
64 bits wide  
133MHz/266 MT/s  
2.1 GB/s

**Width**  
2 bundles per clock  
4 integer units  
2 load or stores per clock  
9 issue ports

**Caches**  
L1 – 2X16KB - 2 clock latency  
L2 – 96K – 12 clock latency  
L3 - 4MB external –20 clk  
11.7 GB/s bandwidth

**Addressing**  
44 bit physical addressing  
50 bit virtual addressing  
Maximum page size of 256MB

The diagram shows a central blue box labeled 'Core 800 MHz' connected to a 'System Bus' above it. Below the core are four yellow boxes representing 'L3 Cache' and 'BSB' (Branch Store Buffer) arranged in a 2x2 grid, connected to the core.

## Itanium2 - McKinley / **Madison**

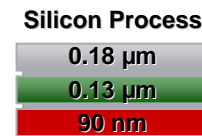
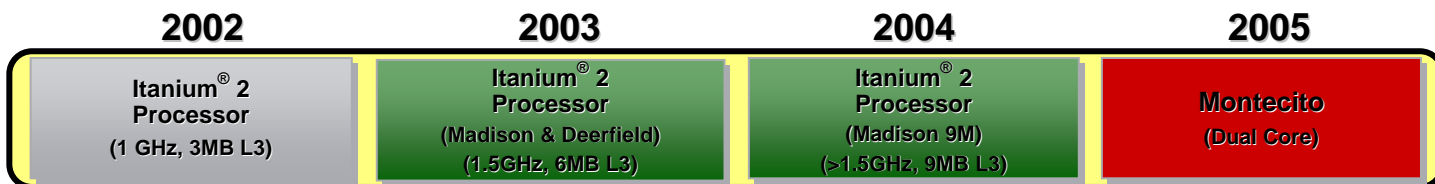
**System Bus**  
128 bits wide  
200MHz/400 MT/s  
6.4 GB/s

**Width**  
2 bundles per clock  
6 integer units  
2 loads and 2 stores per clock  
11 issue ports

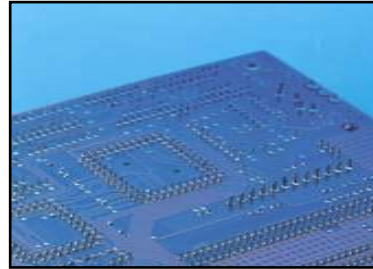
**Caches**  
L1 – 2X16KB - 1 clock latency  
L2 – 256K – 5 clock latency  
L3 - 3MB / 6MB – 12 clk  
32 GB/s bandwidth

**Addressing**  
50 bit physical addressing  
64 bit virtual addressing  
Maximum page size of 4GB

The diagram shows a blue box labeled 'Core 1 GHz' and a yellow box labeled 'L3 Cache' stacked vertically. They are connected to a 'System Bus' above them by two vertical arrows.



# Madison\*\*



\*\*codename

3<sup>rd</sup> Generation Itanium® Architecture Processor

130nm Process, 410M Transistors

1.5GHz Frequency

6 GFLOPS DP-F.P Peak

6MB integrated L3-Cache (48GB/s)

Pin-Compatible to Itanium® 2 Processor

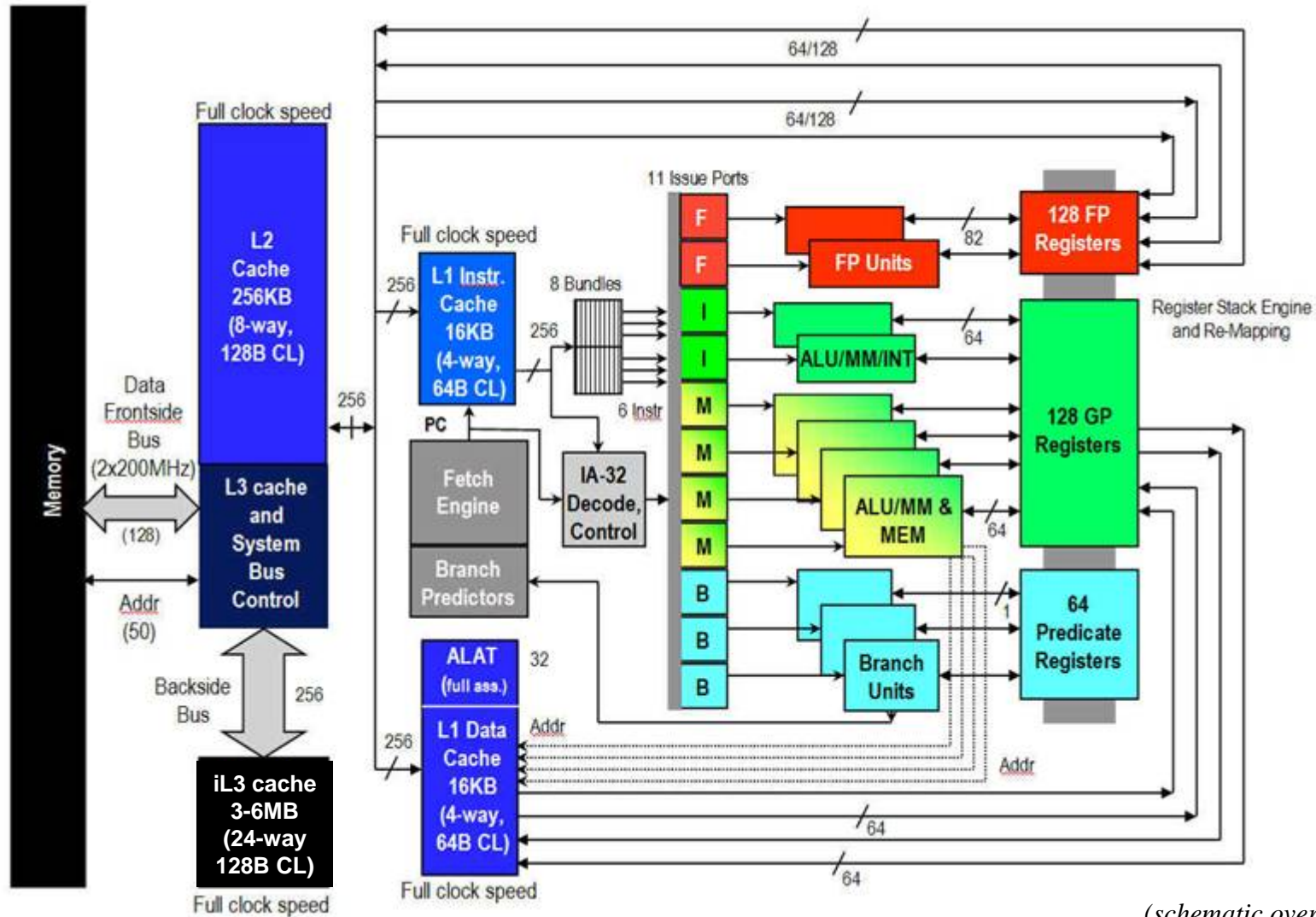
100% Binary Compatible

Same Thermal Envelope

Low-Voltage Version (Deerfield\*\*) in 2H2003

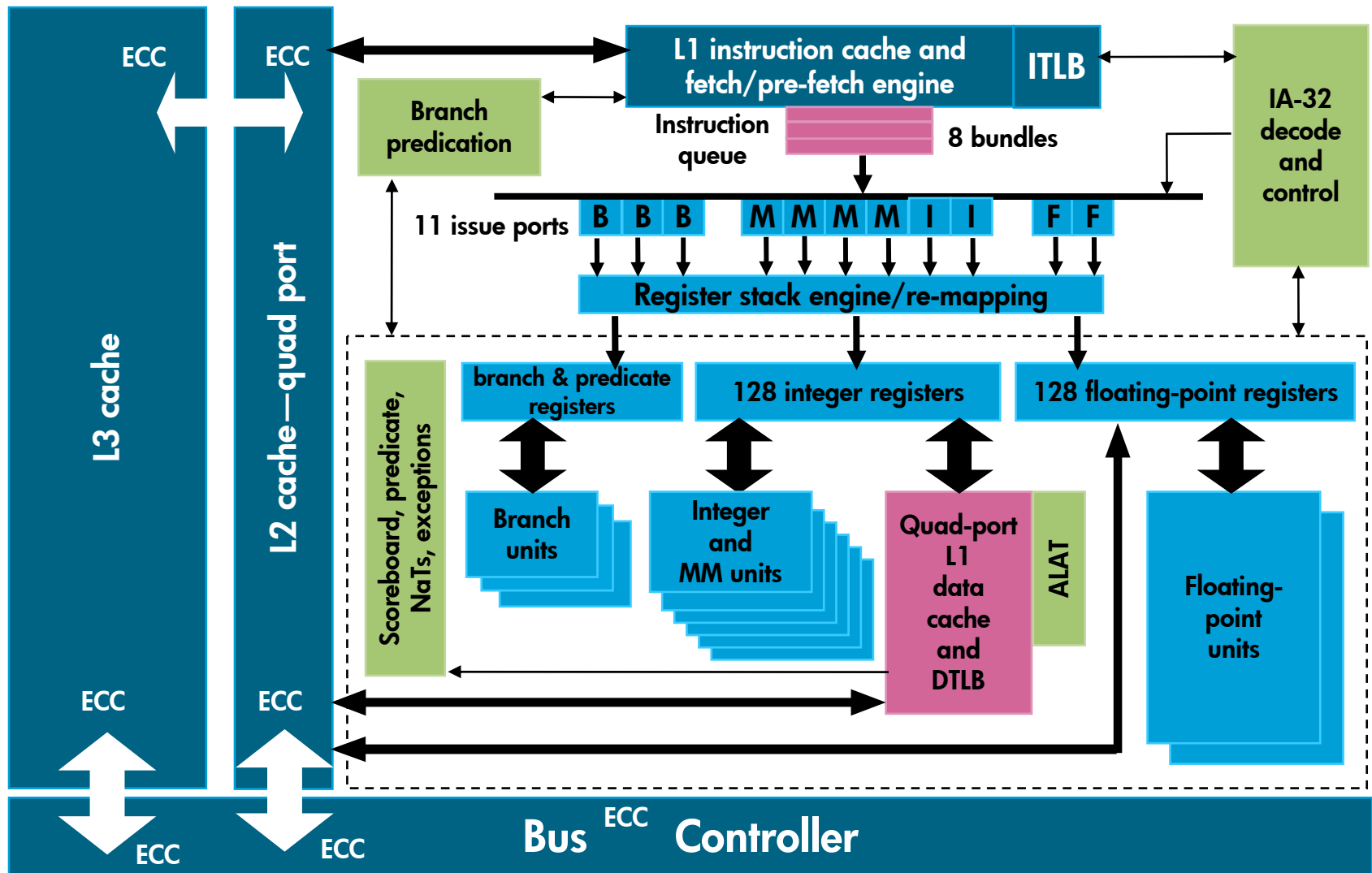
~1.3-1.5x faster than Itanium® 2

# Itanium® 2 Processor Block Diagram



(schematic overview)

# Intel® Itanium2®-based microarchitecture block diagram



# Montecito\*\*



\*\*codename

5<sup>th</sup> Generation Itanium® Architecture Processor

90nm Process

Dual Enhanced Core per Die

High Frequency

12MB integrated L3-Cache per Core

Multi-Threading Support

Some few new Instructions

Low-Voltage Version as well

Target in 2005

*All features and dates specified are targets provided for planning purposes only and are subject to change*



# Itanium2 Processor (“McKinley”)

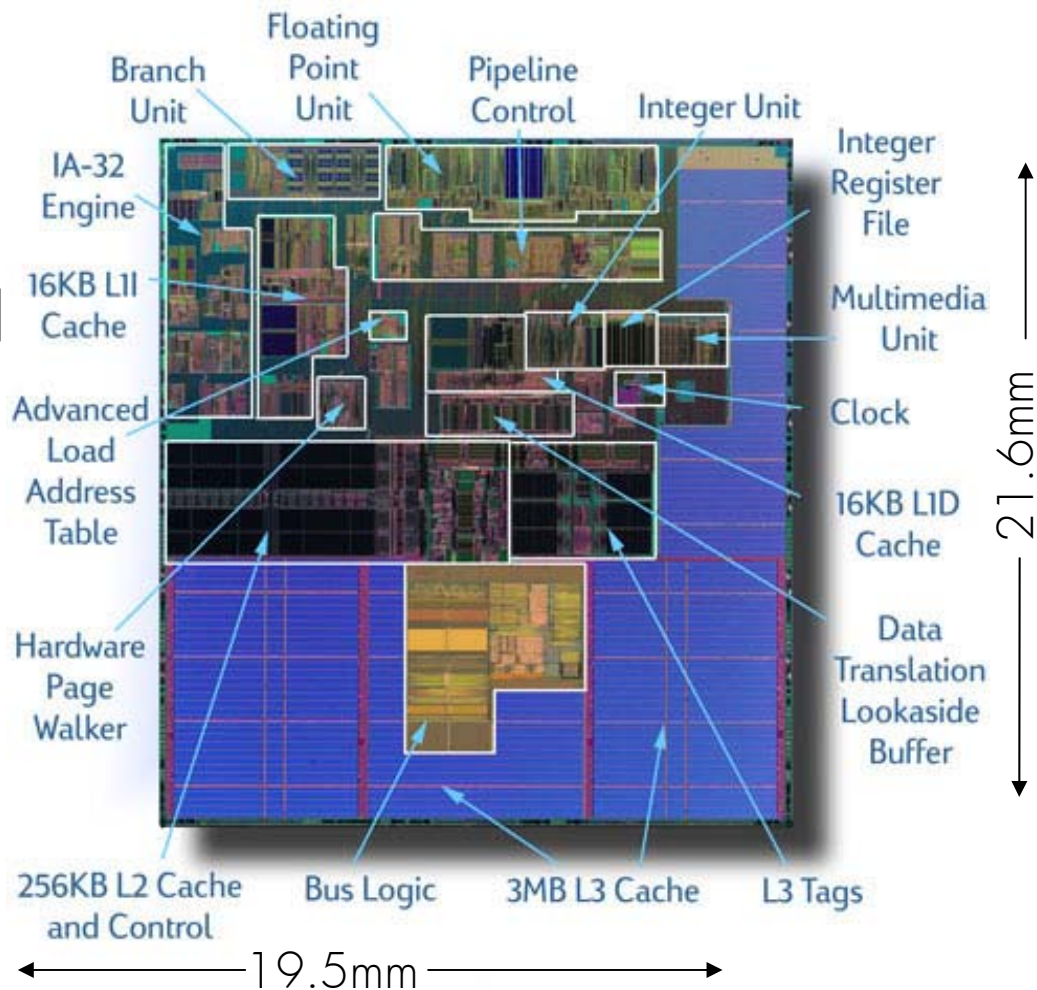
221M FETs

421mm<sup>2</sup>

90+% of the transistors and 50+% of the die area are devoted to cache and cache support logic

Madison:  $\approx$  410M FET

Montecito:  $\approx$  1000M FET



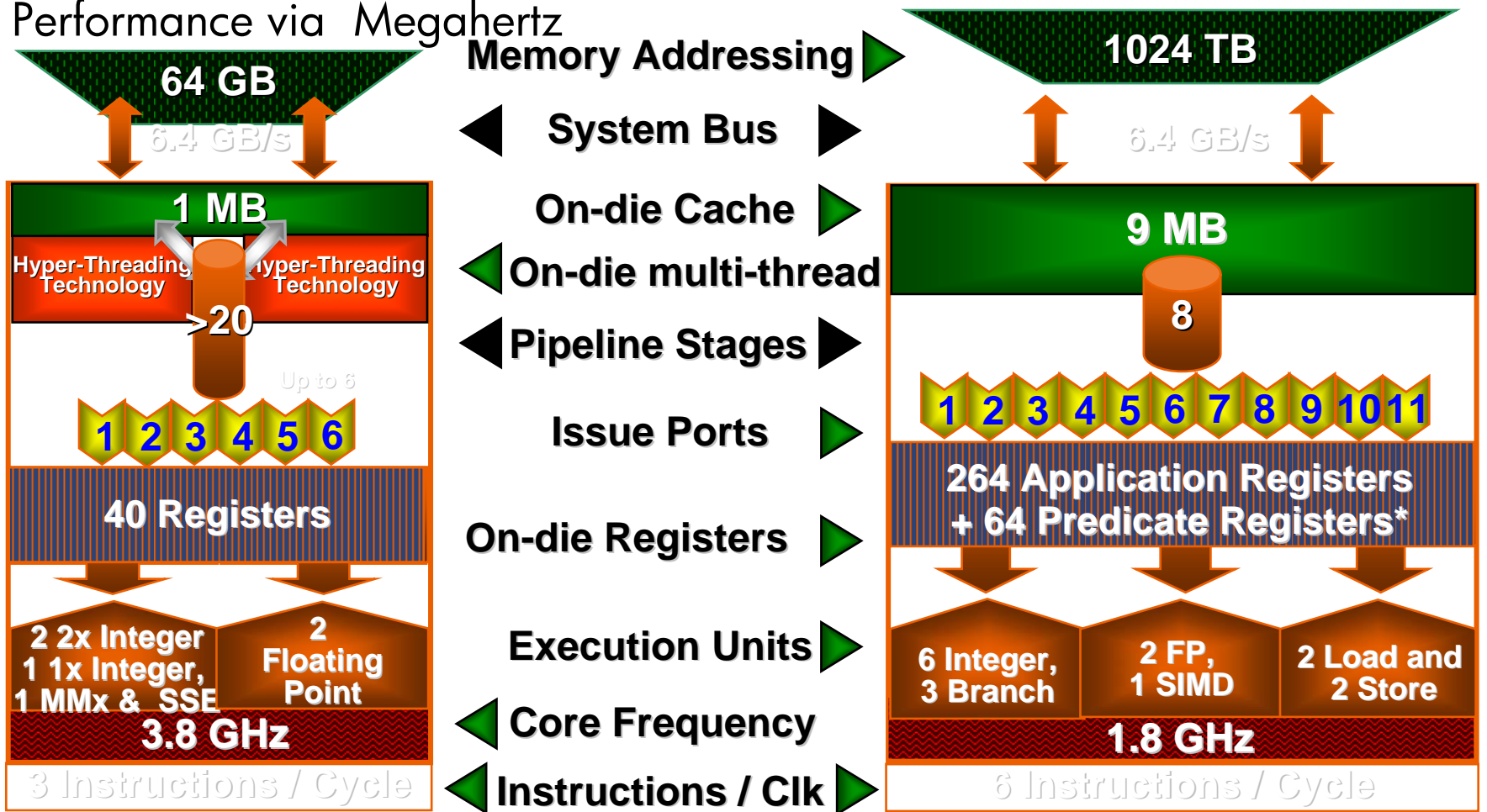
# Intel Enterprise Micro-Architectures



Xeon® Processor  
w/ 64-bit Extensions

Itanium® 2 Processor 9M

Performance via Megahertz



**Performance via Parallelism**

\* Intel's EPIC technology includes 64 single-bit predicate registers to accelerate loop unrolling and branch intensive code execution

# Itanium is uniquely architected for performance



Itanium integrates the best of IA32 performance technology with forward-looking architectural enhancements

## x86 32-bit/64-bit Xeon processor

- Optimized for cost/performance performance in small to medium scale application and databases

## EPIC 64-bit Itanium processor architecture

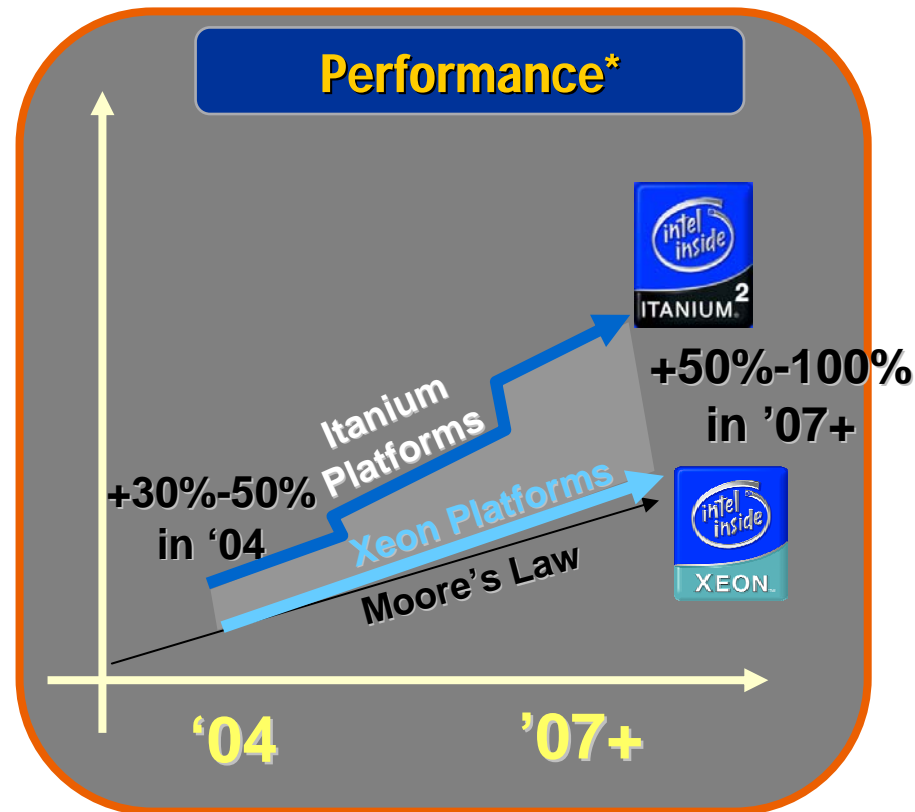
- Optimized for best throughput performance in large and complex technical and commercial workloads
- Performance is much more than 64-bits

<u>X86 32b/64b Xeon</u>	<u>Itanium System features</u>	<u>Itanium Customer benefits</u>
<ul style="list-style-type: none"> <li>• 24 to 40 general registers</li> <li>• Thread-level parallelism (Hyperthreads)</li> </ul>	<ul style="list-style-type: none"> <li>• 264 application registers + 64 predicate registers</li> <li>• Instruction-level parallelism + core-level parallelism*</li> </ul>	Efficient operation; high performance: <ul style="list-style-type: none"> <li>• Reduced context switching</li> <li>• Efficient workload management</li> <li>• Efficient clock-cycle utilization</li> </ul>
Hardware-based parallelism	Data and control speculation	Improve effective memory latency
Dual-core implementations*	Dual-core + multi-core implementations*	<ul style="list-style-type: none"> <li>• Higher performance density</li> <li>• Better system price/performance</li> </ul>
Performance driven by high-clock-rates (>3GHz)	Improved clock-cycle utilization	Sustained performance advantage for business critical applications
Mature development tools and compiler optimization	Core hardware performance improved by future compiler optimizations	Installed systems get faster, even without hardware upgrades

# Itanium® Architecture: Optimized for Multi-Core



- Parallel execution leadership: only Intel has all 3:
  - Multi cores on same die
  - Multi threads on same core
  - Explicit Parallelism in each core
- EPIC\*: inherent advantages for multi-core, multi-thread
  - Architecture: Parallelism + many registers to keep data on-chip
  - Core size: Smaller than IA-32, up to 2X more cores per die on Tukwila (than on IA-32)



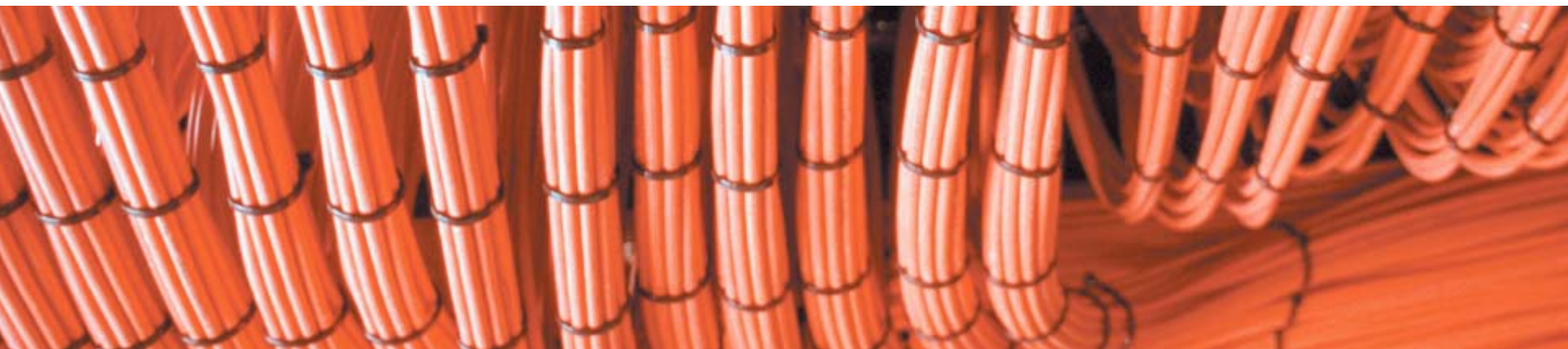
*\* For Enterprise & Technical Computing Application Segments*

**Itanium® Processor family delivers >2X Moore's Law performance**

\* EPIC is Itanium's architecture "Explicitly Parallel Instruction Set Computing"



# The Itanium® Architecture



# Explicitly Parallel Instruction Computing

## Basic Ideas

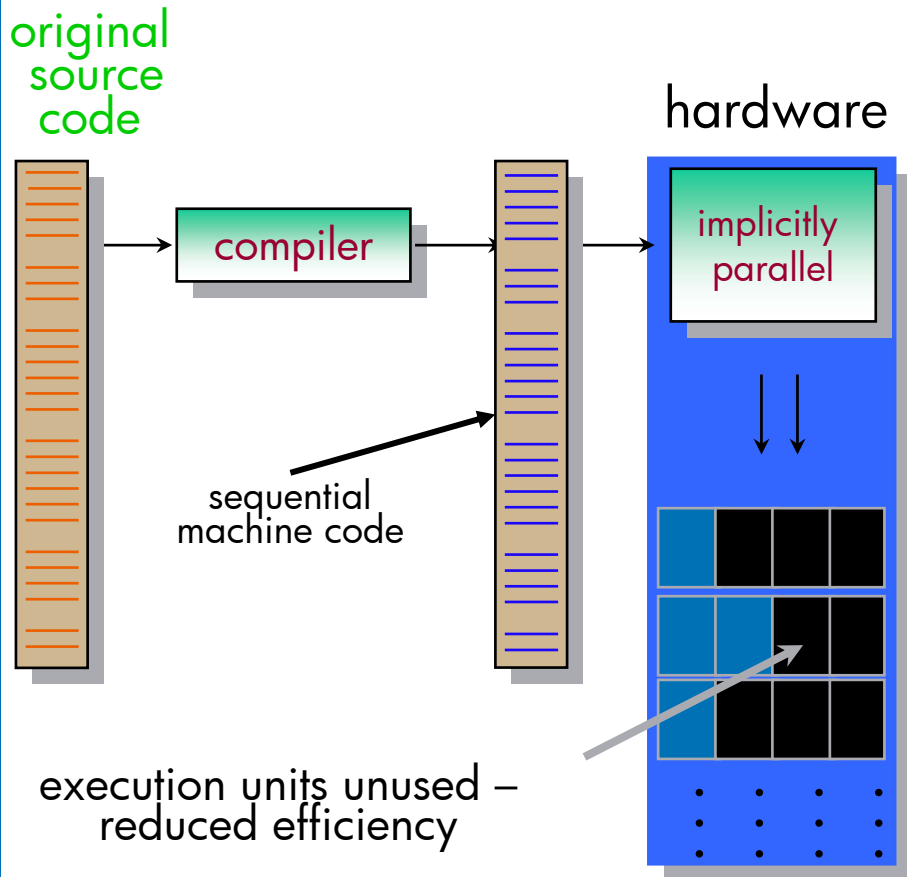


- **Static Hardware Design**
  - Compiler creates record of execution
    - Instructions in bundles
  - Machine plays record
    - Distribute among execution units
  - No runtime changes like *out-of-order-execution*
- **High Scalability of ,execution units'**
  - Very Large Instruction Word (VLIW) concept
  - Focus is parallelism
    - 6 instructions in parallel (2 bundles per cycle)
  - High number of execution units

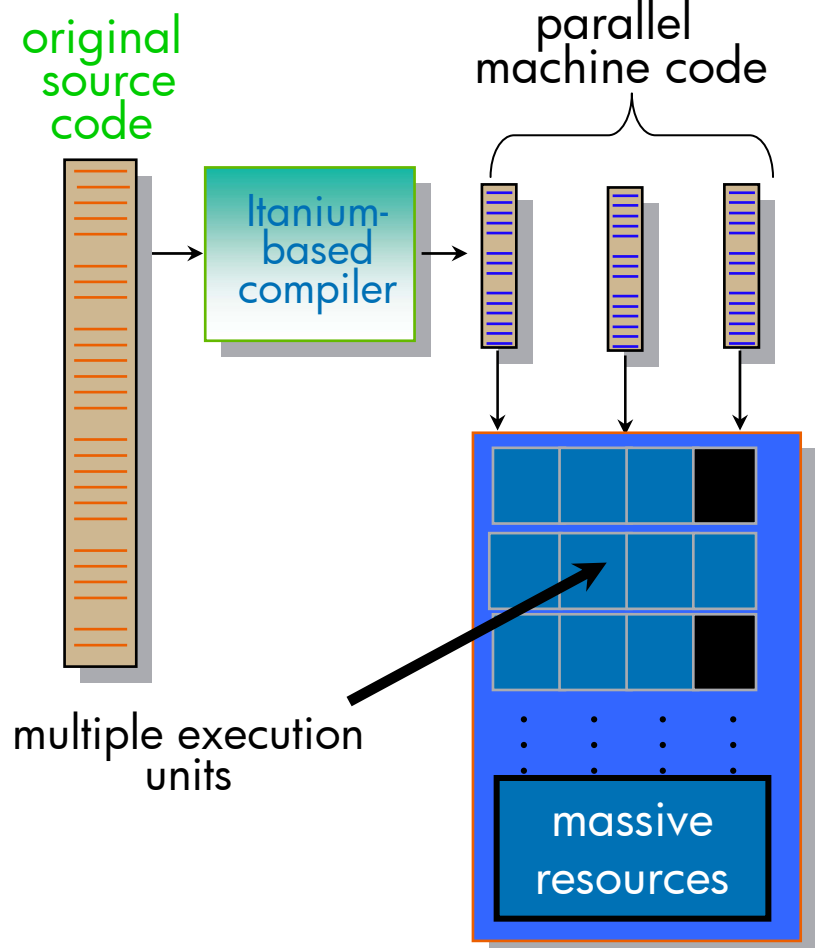
# Itanium Architecture – Basic Ideas



## Traditional architecture



## Itanium™ architecture



Increased parallelization – more throughput

# Traditional Architecture Limits EPIC Solutions



Today's Limits: complexity of multiple pipelines too great to allow effective on-chip scheduling for parallel operation

→Solution: explicit parallelism

Compiler handles Scheduling and communicates this to the chip

Today's Limit: number of registers on chip limits parallelism

→Solution: quadruple registers from 32 to 128

Today's Limit: Large (and growing) memory latency

→Solution: speculative loads

Today's Limit: conditional and/or unpredictable branches

→Solution: prediction and predication orchestrated by the compiler



# Architecture Limits – EPIC Solutions



Today's Limits: complexity of multiple pipelines too great to allow effective on-chip scheduling for parallel operation

→Solution: explicit parallelism

Compiler handles Scheduling and communicates this to the chip

Today's Limit: number of registers on chip limits parallelism

→Solution: quadruple registers from 32 to 128 and increasing addressing from 5 bits to 7

Today's Limit: Large (and growing) memory latency

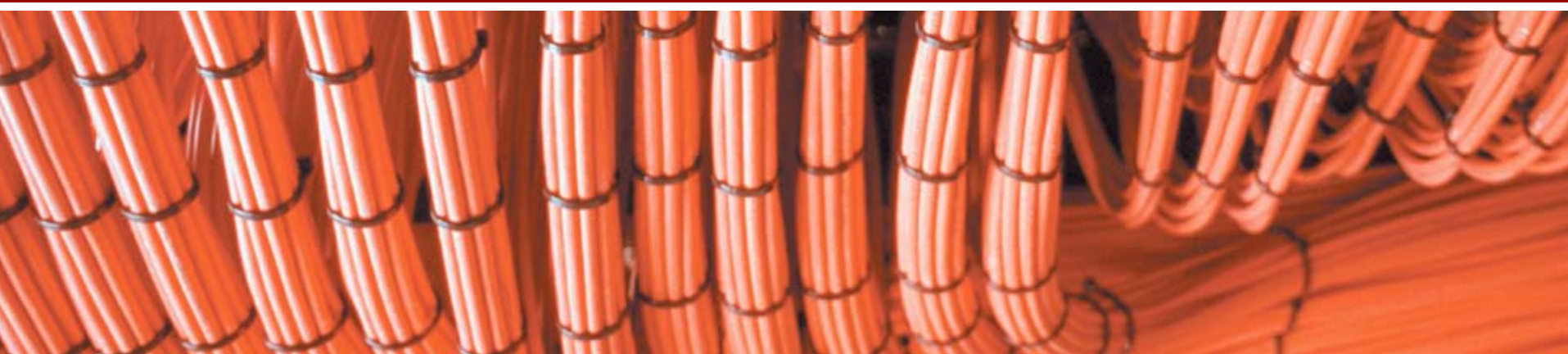
→Solution: speculative loads

Today's Limit: conditional and/or unpredictable branches

→Solution: prediction and predication orchestrated by the compiler



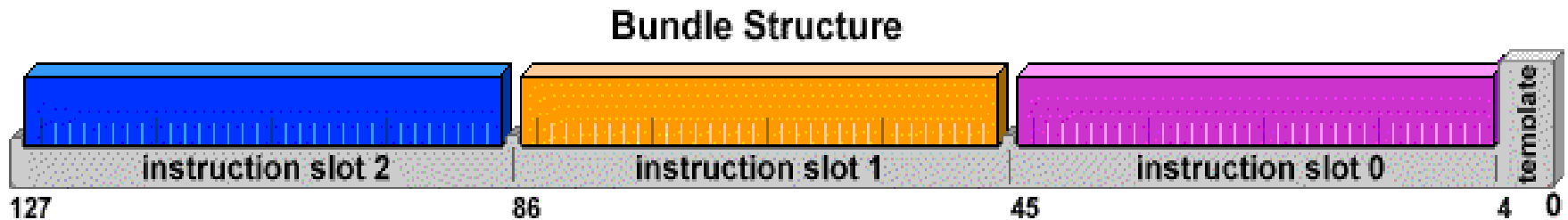
# Increasing Instruction Level Parallelism



# Explicit Parallelism

- Instruction Level Parallelism (**ILP**) is the ability to execute multiple instructions at the same time
- Explicitly Parallel Instruction Computing (**EPIC**) allows the compiler or assembler to specify the parallelism
- Compiler specifies **Instruction Groups**, a list of instructions with no dependencies that can be executed in parallel
- Instructions are packed in **bundles** of 3 instructions each
  - Instruction bundle
  - Two executed per cycle
- Massive resources on chip
  - Large number of registers to avoid register contention

# Instruction Format: Bundles & Templates



- Bundle (123 bits)
  - Set of three instructions (41 bits each)
- Template (5 bits)
  - Identifies types of instructions in bundle
    - One of Integer, Memory, Branch, Floating, eXtended
  - Identifies independent operations ("stops") -> MM\_F
  - Defines execution units to be invoked executing the bundle
  - Compiler can schedule functional units to avoid contention

# Instruction Format: Bundles & Templates



- Instruction types
  - M: Memory
  - I: Shifts and multimedia
  - A: Integer Arithmetic and Logical Unit
  - B: Branch
  - F: Floating point
  - L+X: Long (move, branch, ...)
- Template encodes types
  - MII, MLX, MMI, MFI, MMF
  - Branch: MIB, MMB, MFB, MBB, BBB
- Template encodes parallelism
  - All come in two flavors: with and without stop at end
  - Also, stop in middle: MI\_I M\_MI

# Explicitly Parallel Instruction Encoding



**Program:**



**Instruction Groups:**

- Explicit group stops
- No RAW or WAW dependencies

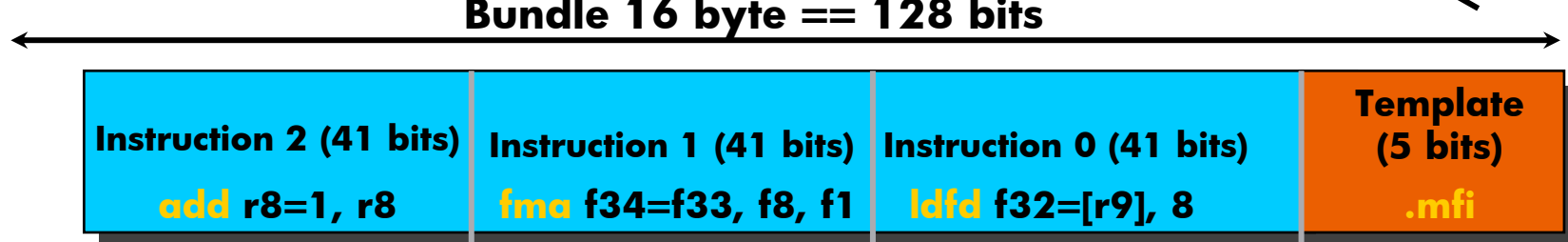


**Instruction Bundles:**

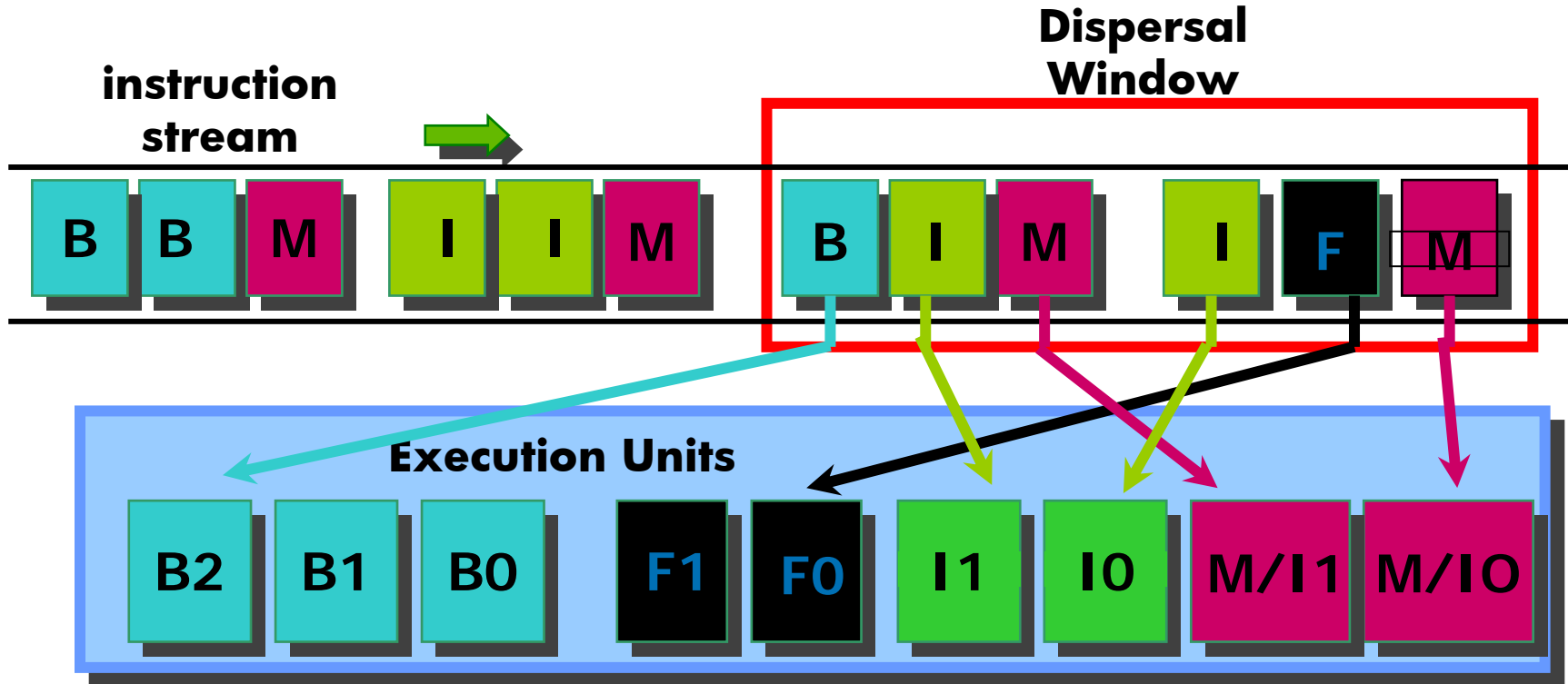
- 3 Instructions and template
- Stops at the end or within



**Bundle 16 byte == 128 bits**



# Instruction Dispersal, Itanium<sup>®</sup> Implementation



*Flexible Issue Capability*

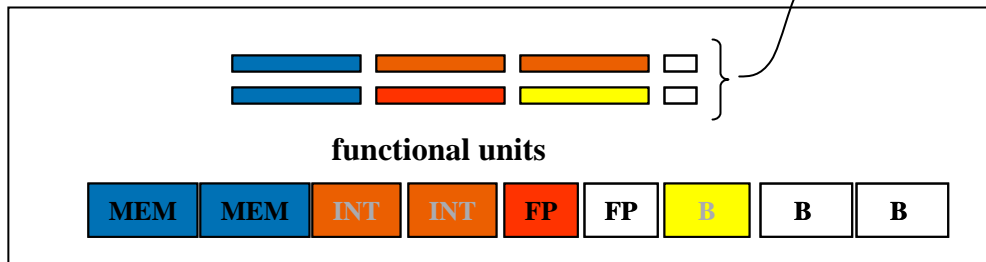
*Up to 6 instructions executed per clock*

# Explicitly Parallel Instruction Computing EPIC

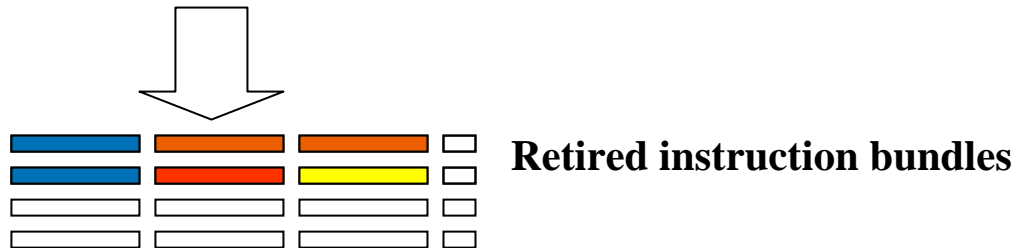


Fetch one or more bundles for execution  
(Implementation, Itanium® takes two.)

Processor



Try to execute all instructions in parallel, depending on available units.



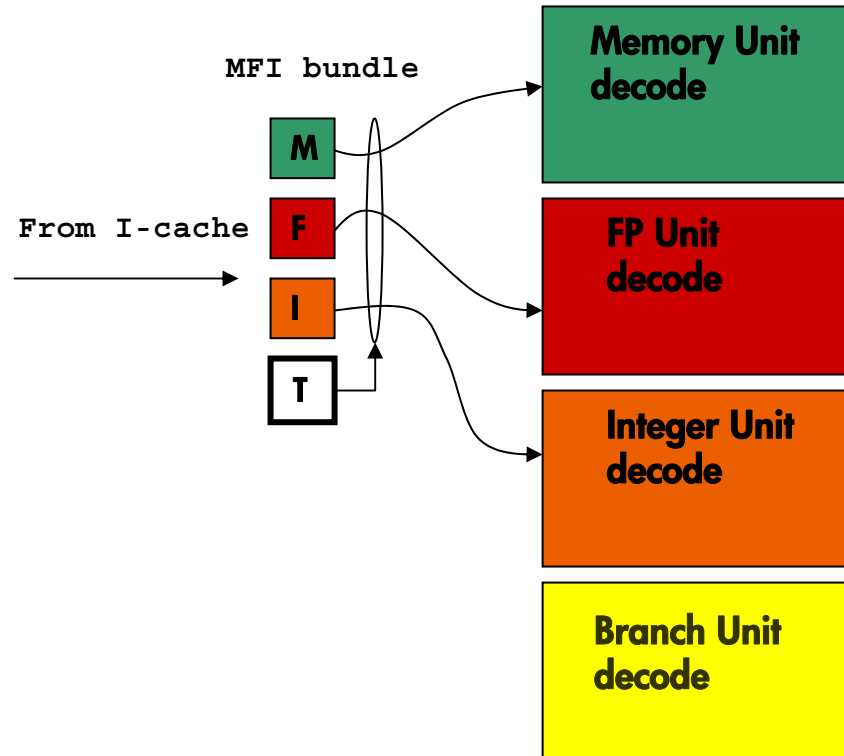


# Defined templates

## Execution Units

Memory  
Integer  
FP  
Branch

0	MII	M	I	I
1	MII;	M	I	I
2	MI;I	M	I	I
3	MI;I;	M	I	I
4	MLX	M	I	I
5	MLX;	M	I	I
8	MMI	M	M	I
9	MMI;	M	M	I
10	M;MI	M	M	I
11	M;MI;	M	M	I
12	MFI	M	F	I
13	MFI;	M	F	I
14	MMF	M	M	F
15	MMF;	M	M	F
16	MIB	M	I	B
17	MIB;	M	I	B
18	MBB	M	B	B
19	MBB;	M	B	B
22	BBB	B	B	B
23	BBB;	B	B	B
24	MMB	M	M	B
25	MMB;	M	M	B
28	MFB	M	F	B
29	MFB;	M	F	B



# Itanium® 2 Dispersal Matrix



	MII	MLI	MMI	MFI	MMF	MIB	MBB	BBB	MBB	MFM
MII	Green	White	Green	Green	Green	Green	Green	Red	Green	Red
MLI	Green	Green	Green	Red	Green	Red	Green	Red	Green	Red
MMI	Green	Green	Green	Green	Green	Green	Green	Red	Green	Green
MFI	Green	Red	Green	Red	Green	Red	Red	Red	Green	Red
MMF	Green	Green	Green	Green	Green	Green	Green	Red	Green	Green
MIB*	Green	Red	Green	Red	Green	Red	Red	White	Green	Red
MBB	White	White	White	White	White	White	White	White	White	White
BBB	White	White	White	White	White	White	White	White	White	White
MMB*	Green	Green	Green	Green	Green	Green	Green	White	Green	Green
MFB*	Red	Red	Green	Red	Green	Red	Red	White	Green	Red

\* hint in first bundle



Possible Itanium® 2 full issue



Possible Itanium® processor and Itanium® 2 full issue

Itanium® 2 allows more compiler dispersal options

# Instruction Groups

- *Instruction groups:*
- Set of instructions
- No dependencies (raw, waw) within group
- May execute in parallel
- The processor executes as many instructions per instruction group as possible, based on its resources
- Must contain at least one instruction (no upper limit)
- Instruction groups are indicated by cycle breaks (;;)

# Instruction groups and bundles

```
ld8    r5 = [r7]
sub    r1 = r2, r3
add    r10 = r20, r21 ;;
add    r1 = r1, r5 ;;
st8    [r7] = r1
```

**Instructions within a group may not have any register dependencies within the group.**

**;; indicates the end of a group.**

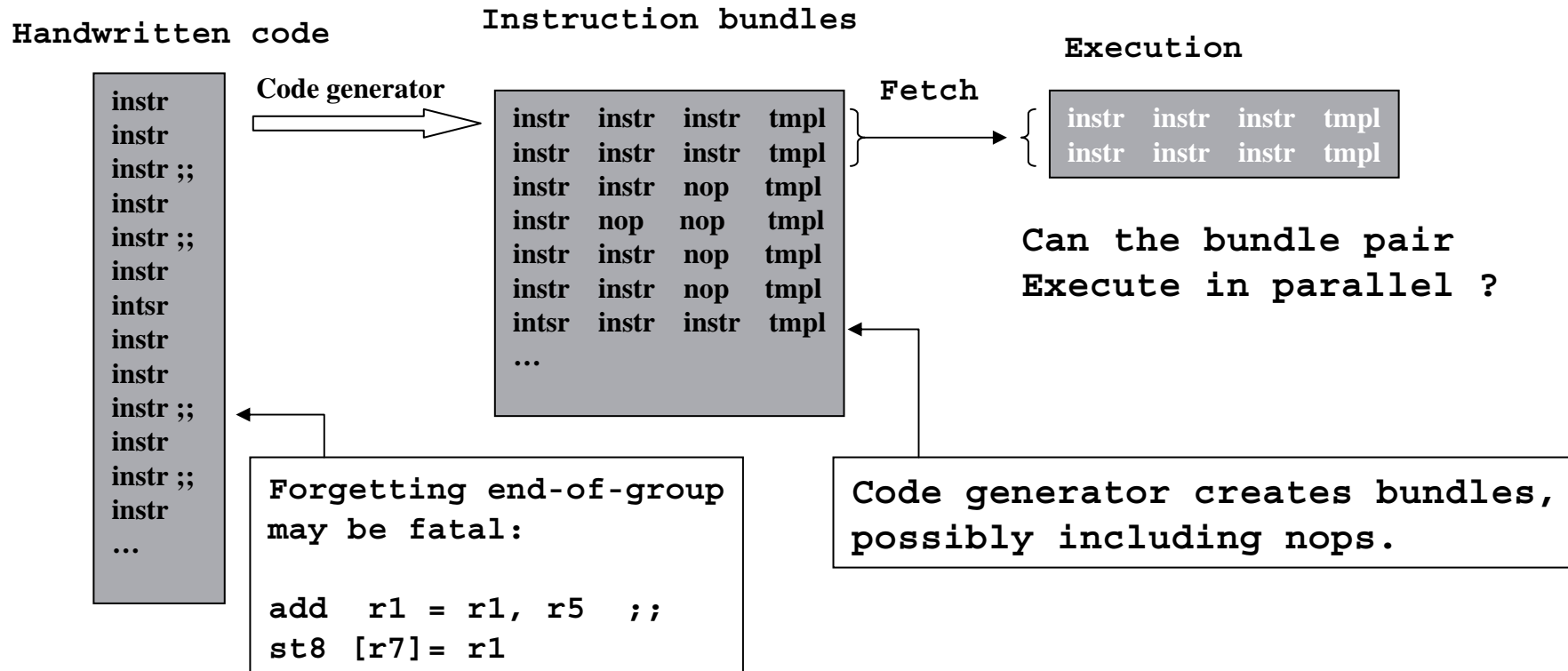
## Instruction bundles

```
{
    .mii                // template
    ld8 r10, [r5]       // slot 0, Memory
    add r1 = r2, r3     // slot 1, Integer
    add r4 = r5,r6      // slot 2, Integer
}
```

**Instructions are fetched and executed in bundles.**

# Instruction groups and bundles

Itanium® and Itanium2® fetch 2 bundles at a time for execution.  
They may or may not execute in parallel.



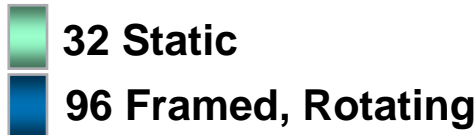
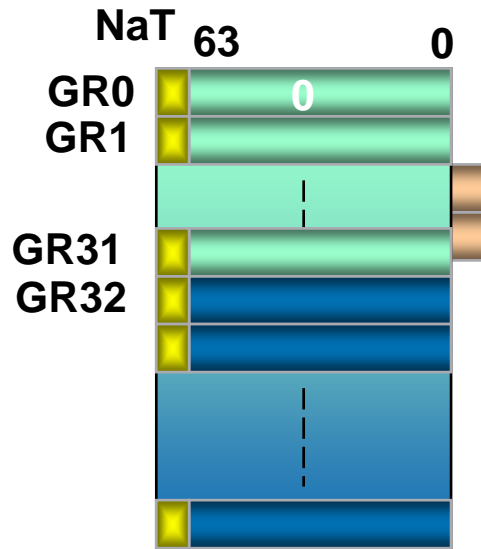
*There are two difficulties:*

- 1) *Finding instruction triplets matching the defined templates.*
- 2) *Matching pairs of bundles that can execute in parallel.*

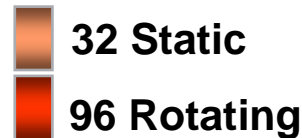
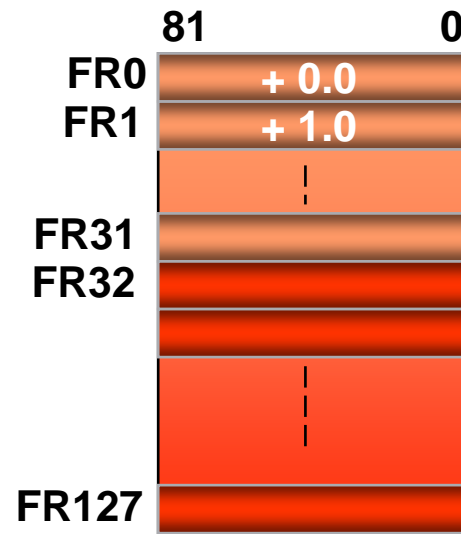
# Massive On Chip Resources

- Several register files visible to the programmer:

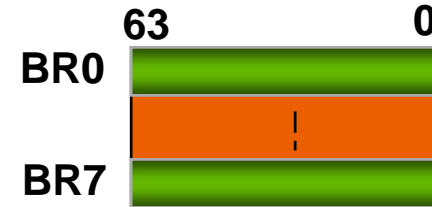
## Integer Registers



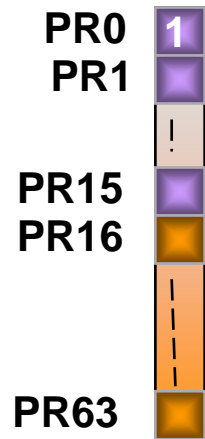
## F.P. Registers



## Branch Registers

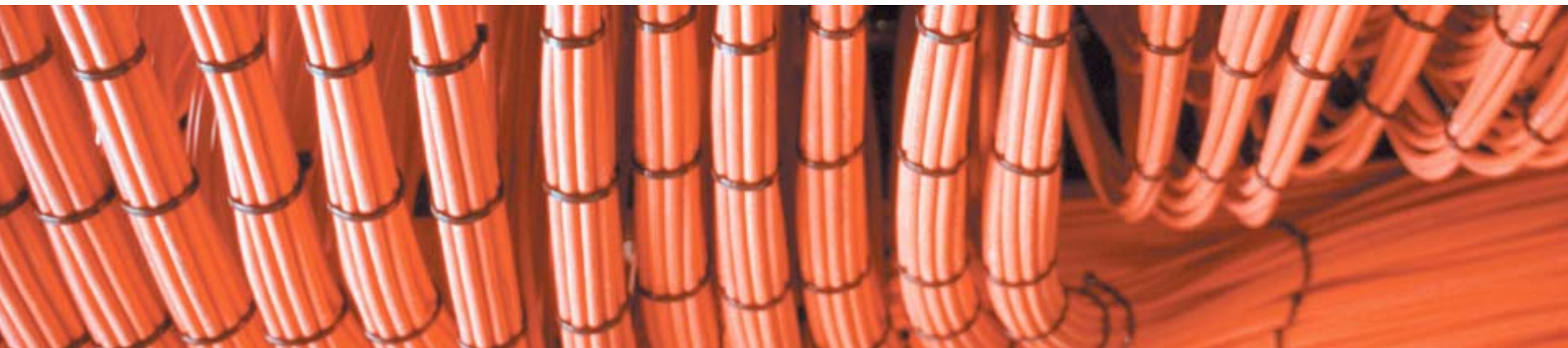


## Predicate Registers





# Improving Branch Handling



# What is the problem ?

- Traditional CPUs:
  - Branch-prediction is used to predict the most likely set of instructions
  - Correct branch prediction keeps the execution pipelines full
  - A mispredicted branch flushes the pipeline with a large penalty
- Itanium® architecture improves branch handling:
  - Provide a way to minimize branches using predicates
  - Provide support for special branch instructions
    - counted loop: br.ctop, br.exit
    - While loop: br.wtop, br.wexit
    - ....



# Branch Handling

- Predication
  - Conditional execution of instructions
  - When the predicate is true, the instruction is executed
  - When it is false, the instruction is treated as a NOP
- Predication converts a control dependency into a data dependency
- Predication eliminates branches in the code

# Predication

- Traditional code:

```
if (a>b)
    c = c + 1
else
    d = d * e + f
```

- Avoid branch by using predicated code

```
p1, p2 = compare(a>b)
```

```
if (p1) c = c + 1
```

```
if (p2) d = d * e + f
```

- Predicate  $p1$  set to 1 if compare is true, and to 0 if it evaluates to false
- $p2$  is the complement of  $p1$

# Predication

Before:

- Instructions  $c = c + 1$  and  $d = d * e + f$  are control dependant on  $a < b$

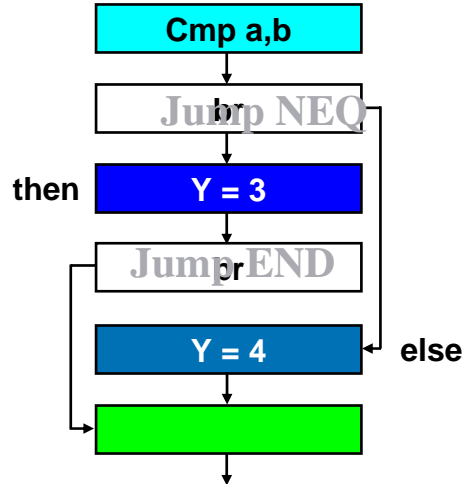
After:

- Instruction are data dependant:
  - Values of  $p1$  and  $p2$
  - They determine execution
  - The branch is eliminated

# Predication



## Traditional Architecture



## Itanium® Architecture

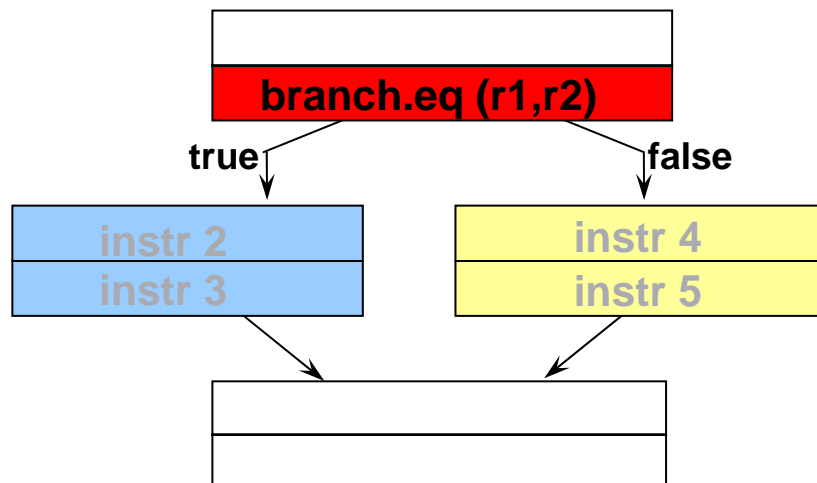


Only one 'branch' will have a valid predicate and be executed.

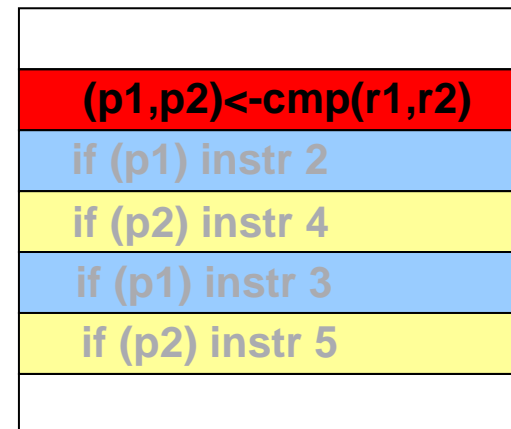
# Predication

- predication provides the ability to conditionally execute instructions based on computed true/false conditions
  - avoids branches
  - predicated instruction either completes or is dismissed (no ops)
  - predicate registers are set by compare/test instructions

## Typical



## Optimized IPF



# IPF Instructions (cont)

- Instruction style is “(Pn) opcode target(s)=source(s)”

– Example:

```
(p4)      cmp.eq      p7,p12 = r37, r52
(p7)      br          label1
(p12)     br          label2
```

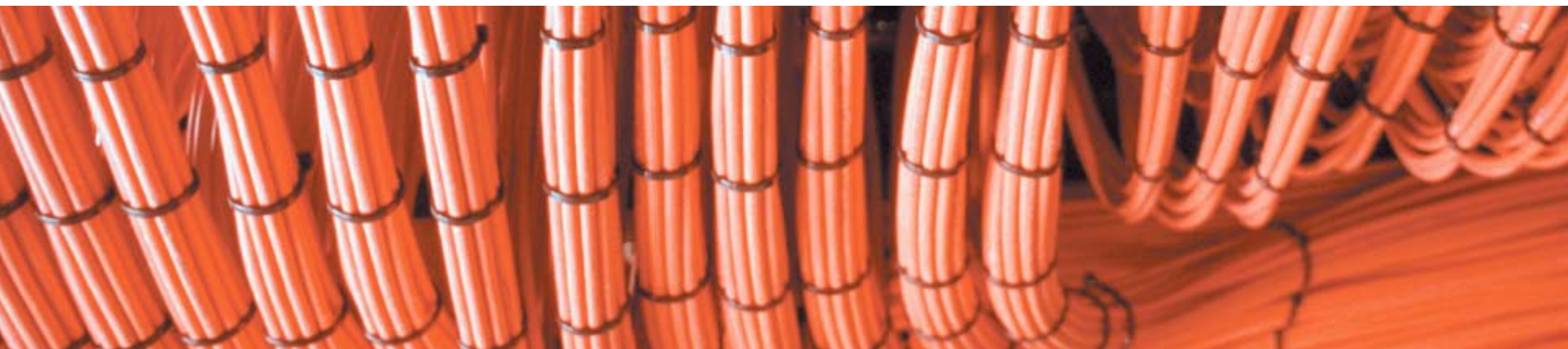
– First instruction only:

- P4 controls whether or not the results are kept or discarded
- the result registers are predicate registers P7 and P12
- R37 is compared for equality with R52
  - If equal: P7 is set to 1 and P12 is set to 0.
  - If not equal: P7 is set to 0 and P12 is set to 1.

– Combination of three instructions show how an if-then-else might be coded.



# Reducing Memory Access Cost



# Reducing Memory Access Cost

- Itanium® architecture eliminates many memory accesses through:
  - large register files to manage work in progress
  - better control of the memory hierarchy (cache hints)
  
- Itanium® architecture reduces remaining memory accesses by:
  - moving load instructions earlier in the code
    - *Data speculation* - advance a load before a possible data dependency
    - *Control speculation* – speculative load before its guarding branch
  - -> allows early execution of loads to hide latency
  - -> enables the processor to bring in the data in time
  - -> avoids stalling the processor

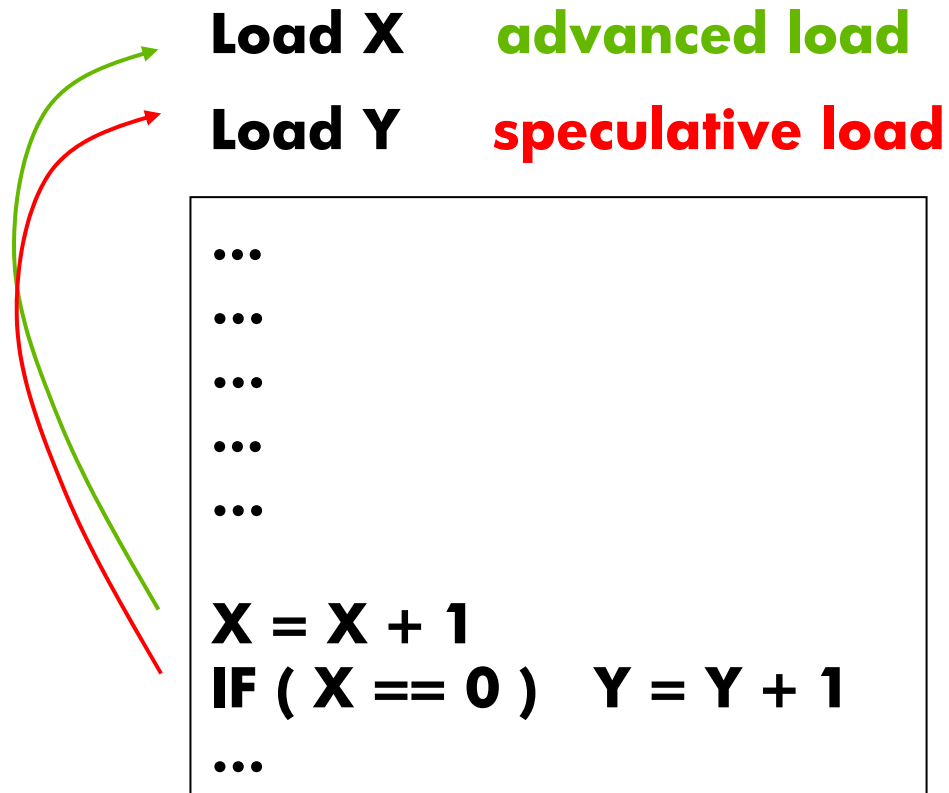


# Data Speculation

- allows early execution of loads to hide latency
- advance load before a possible data dependency (load before store)
- speculative load before a branch that guards it

**Memory latency can be responsible for 60% or more of processor stalls**

# Advanced and Speculative loads



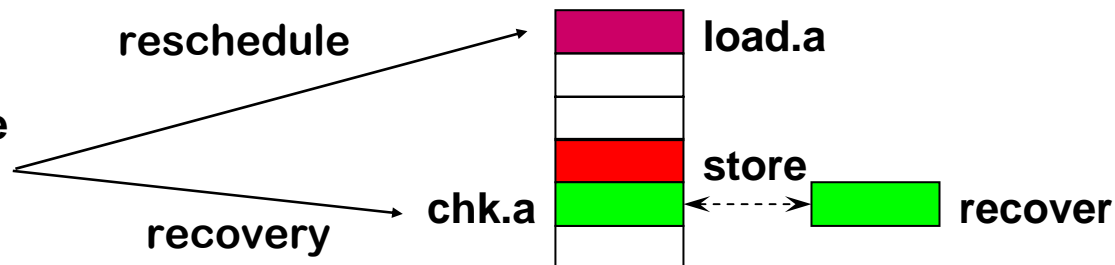
# Data Speculation

- allows early execution of loads to hide latency
- advance load before a possible data dependency (load before store)

typical



optimized IPF



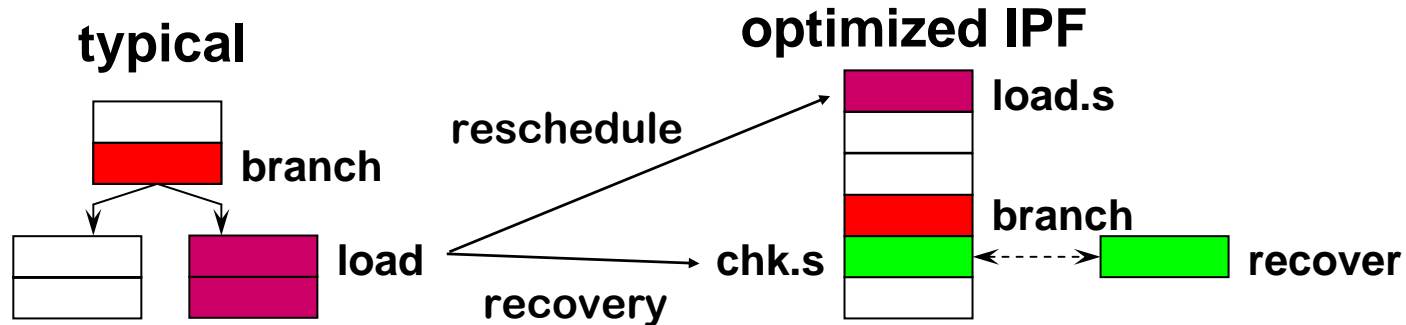
support for data speculation

- ALAT (advanced load address table) – hardware structure that contains information about outstanding advanced loads
- advanced loads: ld.a
- check loads: ld.c
- advance load checks: chk.a
- speculative advanced loads: ld.sa

**Latency can be responsible for 60% or more of processor stalls**

# Control Speculation

- allows early execution of loads to hide latency
- speculative load before a branch that guards it



support for control speculation

- NaT (Not a Thing) bit – 65<sup>th</sup> bit of GR, set on incorrect speculation instead of faulting
- NaT bit propagated in computations
- speculation check: `chk.s`
- speculative load: `ld.s`

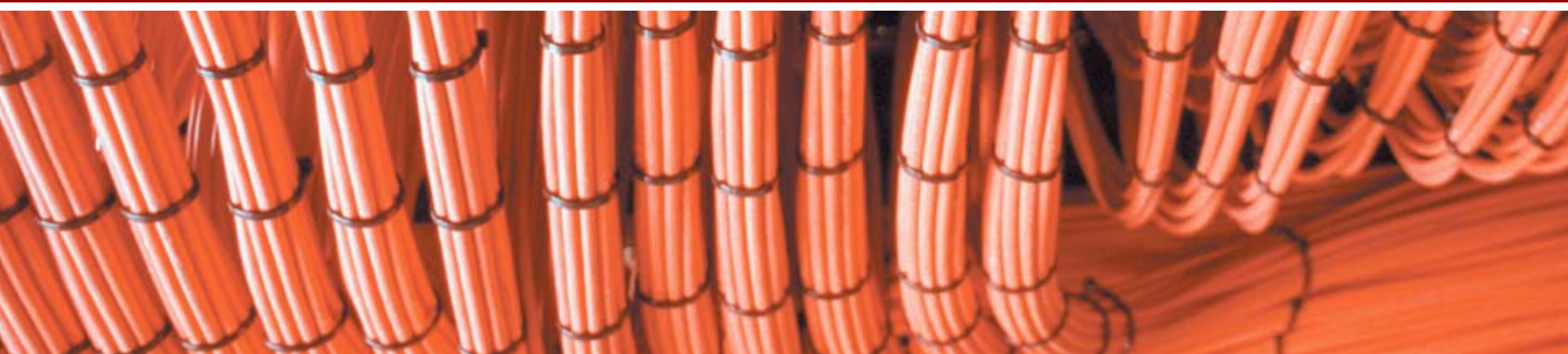
***speculation hides memory latency***

# Massive Memory Resources

- Physical memory
  - Full implementation will address 16 EB of physical memory ( $2^{64}$ )
    - 16,000,000,000GB
  - Itanium® architecture microprocessor has 44-bit address bus
    - 16TB (16,000GB) physical memory addressable
  - Itanium2® architecture microprocessors have 50-bit address bus
- Virtual memory
  - Itanium® architecture microprocessor uses 50-bits
  - Itanium2® architecture microprocessors uses 64-bits



# Supporting Modular Code

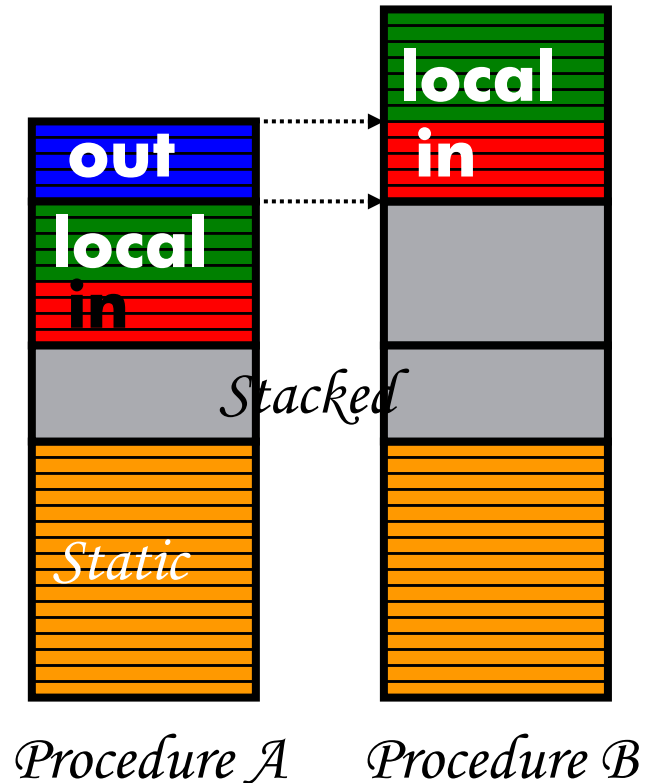


# Procedure Call Overhead

- Modular programs create more overhead
  - Programs tend to be call intensive
  - Register space shared by caller and callee
  - Call>Returns require register save/restores
  - Frequent memory access
  - Limitations due to resource shortage
- Itanium® solution
  - Massive register resources
    - Renaming, rotating
    - Integer registers stackable
  - Register Stack Engine (RSE)
  - Eliminates memory accesses
  - Allows to allocate local registers dynamically

# Register Stack

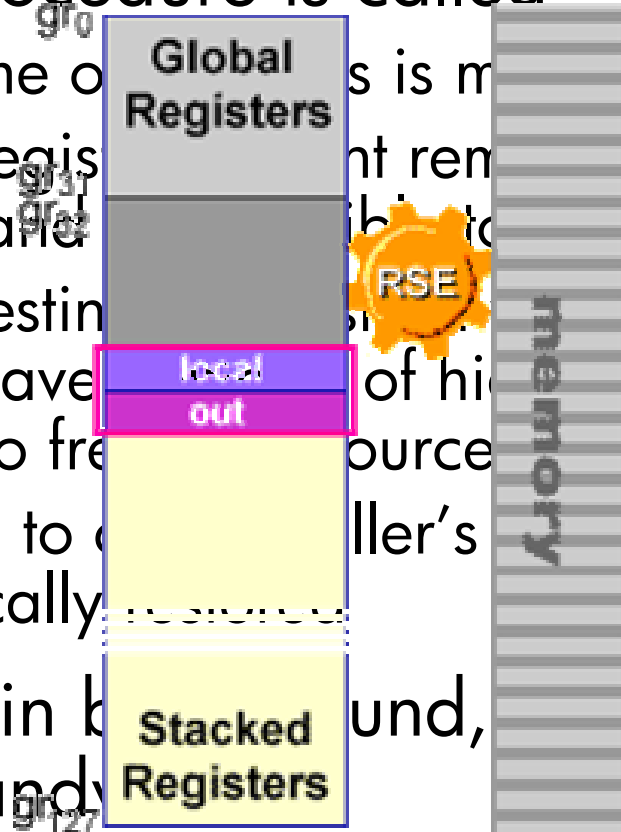
- The general register stack is divided into two subsets:
- Static: 32 permanent registers (r0-r31)
  - visible to all procedures
  - Used for global variables
- Stacked: 96 other registers are like a stack
  - procedure code allocates up to 96 registers for a frame
  - previous frame is hidden
  - first register is renamed to logical register r32
  - small frames eliminate/reduce saving/restoring registers to/from memory





# Register Stack Engine (RSE)

- When a procedure is called
  - New frame of global registers is made available
  - Caller's registers are not removed, but are made invisible and the registers of the called procedure are made visible
  - If deep nesting occurs, the RSE will save the registers of higher-level procedures to free up registers to use for the current procedure
  - On return to the caller, the RSE automatically restores the caller's registers to their original content
- RSE works in kernel mode, managing unused memory bandwidth
- Activity not visible to application programs



# Procedure Call Overhead

## **Traditional**

*Procedure A*

call B

*Procedure B*

save current register state

...

restore previous register state

return...

## **Itanium® Architecture**

*Procedure A*

call B

*Procedure B*

alloc, no save!

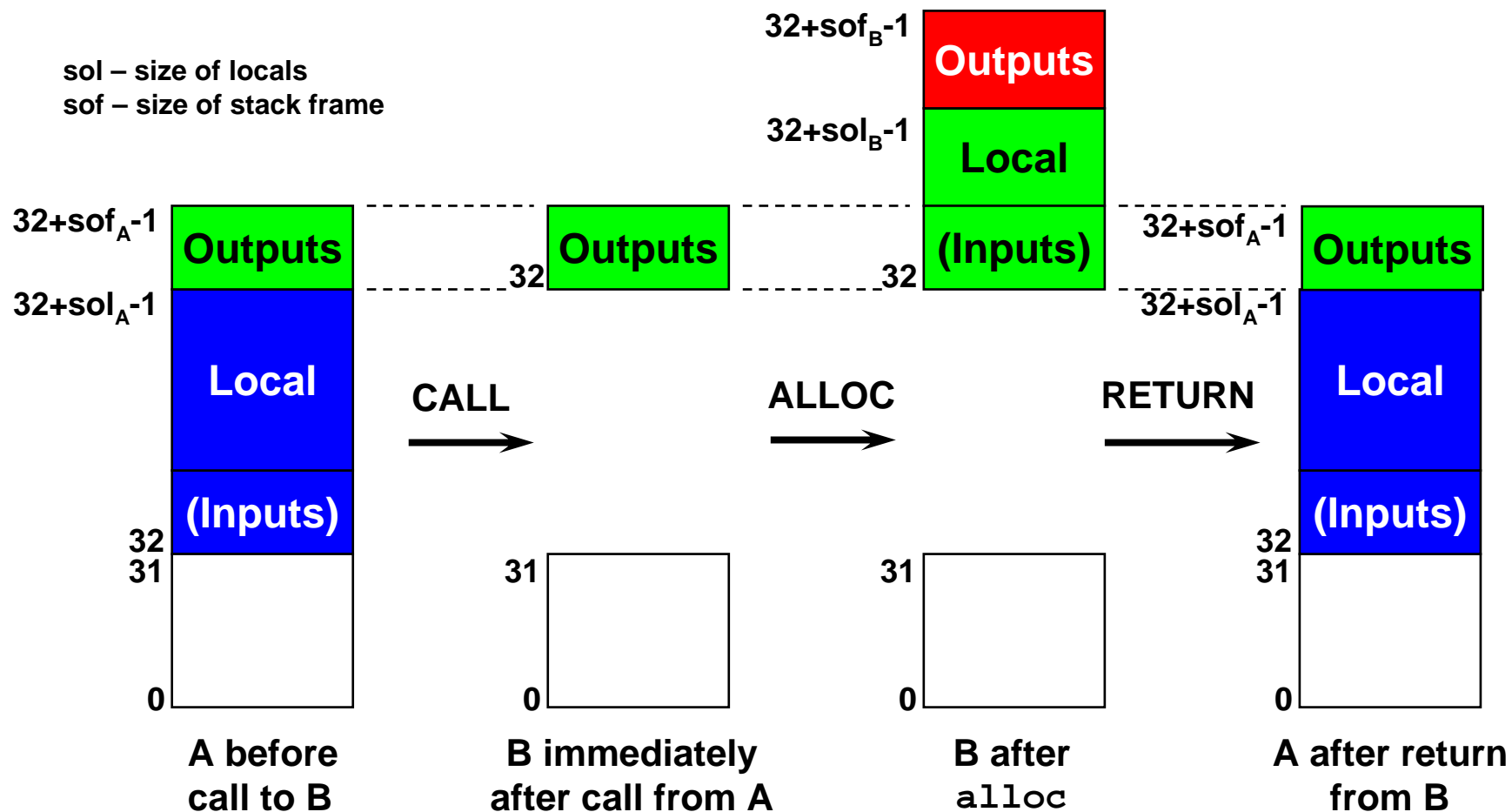
...

no restore! (remap)

return

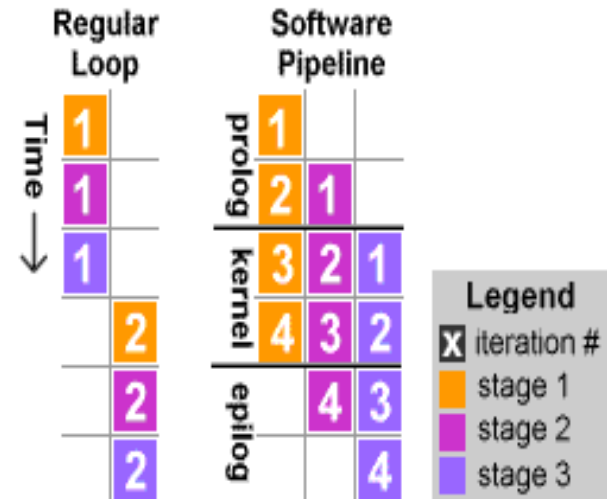
# Register Stack Engine (RSE)

sol – size of locals  
 sof – size of stack frame

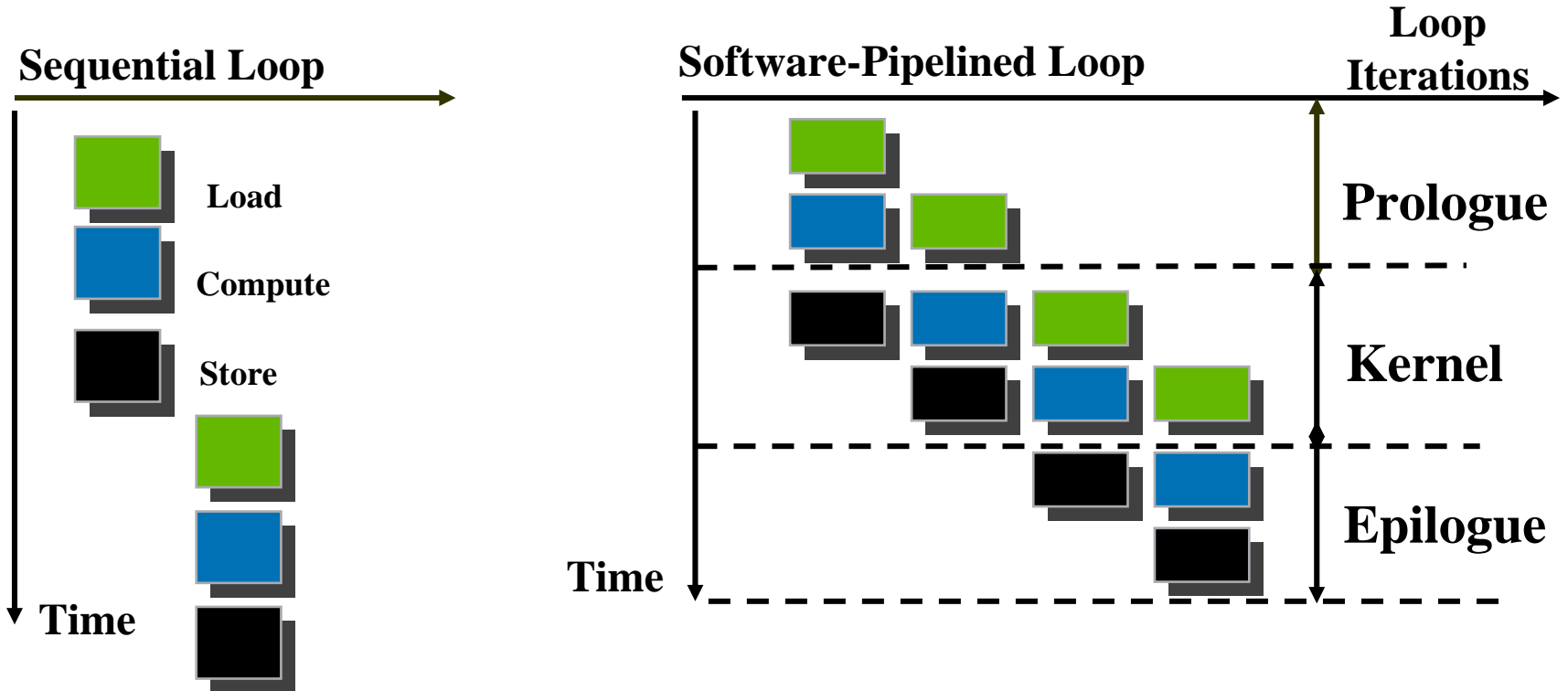


# Loop Optimization Overhead

- Enhance loop performance:
  - Done by unrolling loops
  - Causes code expansion
  - Prologue/epilogue add to code size
- Itanium® solution
  - Software pipelining
  - Architecture support
    - **Minimal prologue/epilogue code**
    - **Predication**
    - **Loop control registers (LC, EC)**
    - **Loop branches (br.ctop, br.wtop)**



# Software Pipelining



- **Multiple iterations execute in parallel**
- **ILP Maximized**
- **Different iteration stages execute in parallel**
- **Execution load is balanced**

# Software Pipelining

iteration 1	iteration 2	iteration 3	iteration 4	iteration 5	cycle
ld4					X
	ld4				X+1
add		ld4			X+2
st4	add		ld4		X+3
	st4	add		ld4	X+4
		st4	add		X+5
			st4	add	X+6
				st4	X+7

*Prolog*

*Epilog*

# Architecture Limits – EPIC Solutions



Today's Limits: complexity of multiple pipelines too great to allow effective on-chip scheduling for parallel operation

→Solution: explicit parallelism

Compiler handles Scheduling and communicates this to the chip

Today's Limit: number of registers on chip limits parallelism

→Solution: quadruple registers from 32 to 128 and increasing addressing from 5 bits to 7

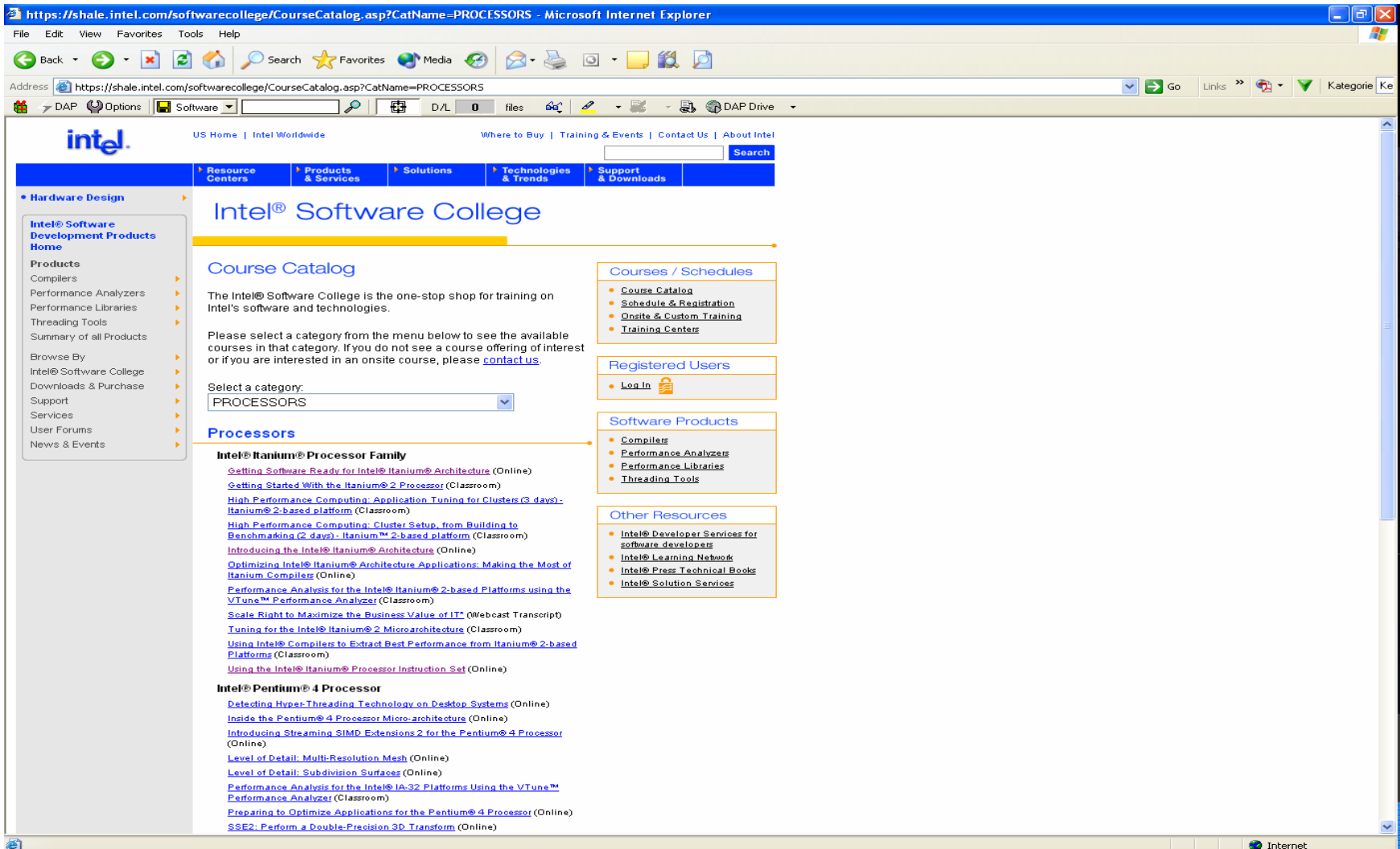
Today's Limit: Large (and growing) memory latency

→Solution: speculative loads

Today's Limit: conditional and/or unpredictable branches

→Solution: prediction and predication orchestrated by the compiler

# Itanium(r) Architecture Training



The screenshot shows a Microsoft Internet Explorer browser window displaying the Intel Software College Course Catalog for Processors. The page features a navigation menu on the left, a main content area with a search bar and course listings, and several sidebars for additional resources.

**Navigation Menu:** US Home | Intel Worldwide | Where to Buy | Training & Events | Contact Us | About Intel

**Course Catalog:** The Intel® Software College is the one-stop shop for training on Intel's software and technologies. Please select a category from the menu below to see the available courses in that category. If you do not see a course offering of interest or if you are interested in an onsite course, please [contact us](#).

**Processors**

**Intel® Itanium® Processor Family**

- [Getting Software Ready for Intel® Itanium® Architecture](#) (Online)
- [Getting Started With the Itanium® 2 Processor](#) (Classroom)
- [High Performance Computing: Application Tuning for Clusters \(3 days\)- Itanium® 2-based platform](#) (Classroom)
- [High Performance Computing: Cluster Setup, from Building to Benchmarking \(2 days\)- Itanium™ 2-based platform](#) (Classroom)
- [Introducing the Intel® Itanium® Architecture](#) (Online)
- [Optimizing Intel® Itanium® Architecture Applications: Making the Most of Itanium Compiler](#) (Online)
- [Performance Analysis for the Intel® Itanium® 2-based Platforms using the VTune™ Performance Analyzer](#) (Classroom)
- [Scale Right to Maximize the Business Value of IT™](#) (Webcast Transcript)
- [Tuning for the Intel® Itanium® 2 Microarchitecture](#) (Classroom)
- [Using Intel® Compilers to Extract Best Performance from Itanium® 2-based Platforms](#) (Classroom)
- [Using the Intel® Itanium® Processor Instruction Set](#) (Online)

**Intel® Pentium® 4 Processor**

- [Detecting Hyper-Threading Technology on Desktop Systems](#) (Online)
- [Inside the Pentium® 4 Processor Micro-architecture](#) (Online)
- [Introducing Streaming SIMD Extensions 2 for the Pentium® 4 Processor](#) (Online)
- [Level of Detail: Multi-Resolution Mesh](#) (Online)
- [Level of Detail: Subdivision Surfaces](#) (Online)
- [Performance Analysis for the Intel® IA-32 Platforms Using the VTune™ Performance Analyzer](#) (Classroom)
- [Preparing to Optimize Applications for the Pentium® 4 Processor](#) (Online)
- [SSE2: Perform a Double-Precision 3D Transform](#) (Online)

**Sidebars:**

- Hardware Design:** Intel® Software Development Products Home, Products (Compilers, Performance Analyzers, Performance Libraries, Threading Tools, Summary of all Products, Browse By, Intel® Software College, Downloads & Purchase, Support, Services, User Forums, News & Events).
- Courses / Schedules:** Course Catalog, Schedule & Registration, Onsite & Custom Training, Training Centers.
- Registered Users:** Log In.
- Software Products:** Compilers, Performance Analyzers, Performance Libraries, Threading Tools.
- Other Resources:** Intel® Developer Services for software developers, Intel® Learning Network, Intel® Press Technical Books, Intel® Solution Services.



# Itanium® Architecture Training

The following classes can be taken online or can be downloaded:

- **Getting Software Ready for Intel® Itanium® Architecture**
- **Introducing the Intel® Itanium® Architecture**
- **Using the Intel® Itanium® Processor Instruction Set**
- Intel(r) Software College
  - <https://shale.intel.com/softwarecollege/CourseCatalog.asp?CatName=PROCESSORS>

Questions ?





i n v e n t