

# Next Generation Instruction Set Architecture

**John Crawford, Intel Fellow**

Director, Microprocessor Architecture  
Intel Corporation

**Jerry Huck**

Manager and Lead Architect  
Hewlett-Packard Company



# Objectives

- ◆ **Unveil the technology behind the next generation ISA**
  - Today's focus on architecture, not implementation
- ◆ **Context**
  - History
  - Motivation
- ◆ **ISA Preview**
  - A few key features
  - Benefits

# Intel and HP Technology Alliance

## ◆ Intel

- Microprocessor / platform technology
- 64-bit architecture definition

## ◆ HP

- Enterprise systems technology expertise
- Architecture research advancements

## ◆ Jointly defined next generation 64-bit instruction set

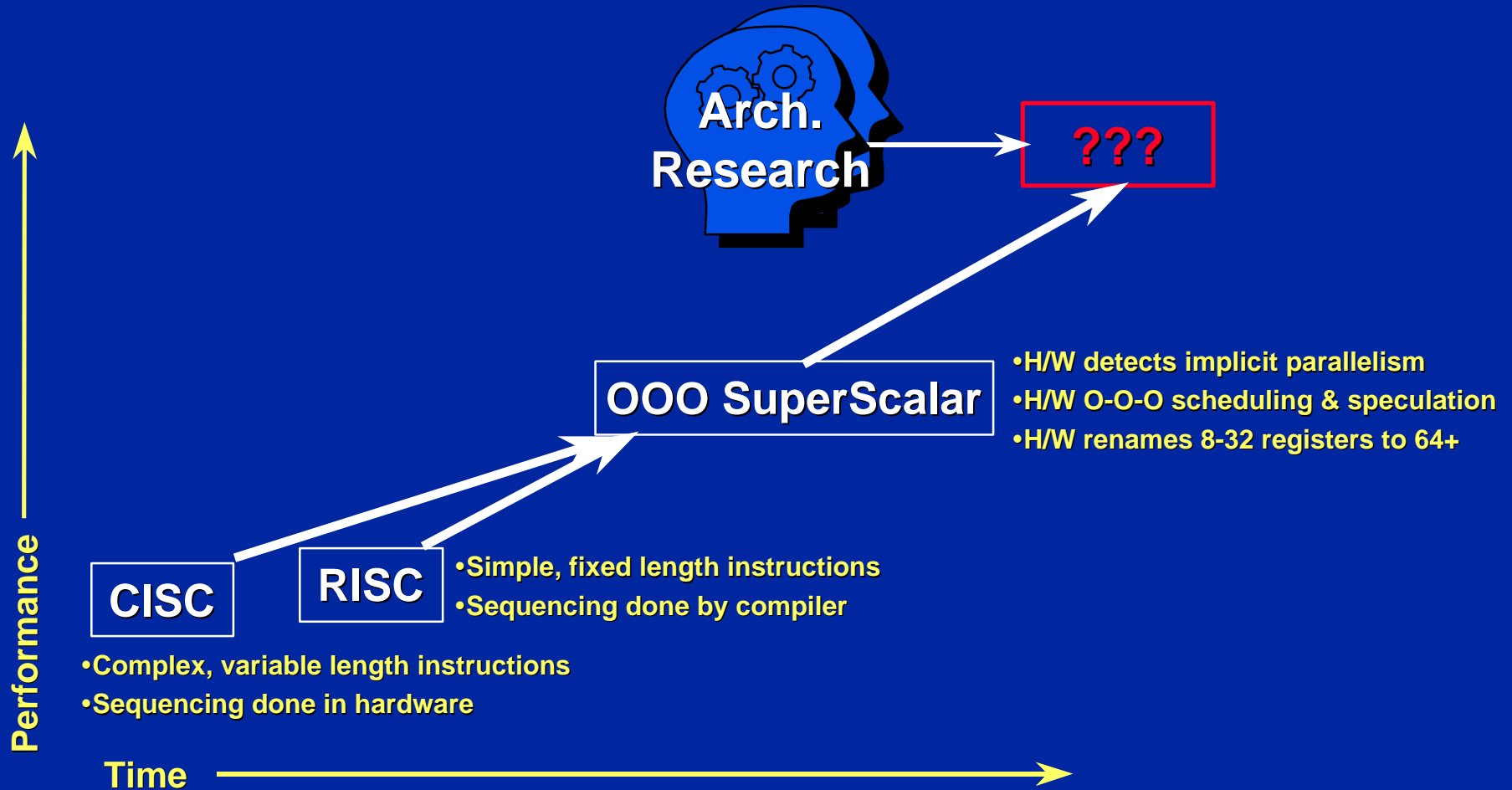
- Instruction set specification
- Compiler optimization
- Performance simulation and projection



# Instruction Set Architecture (ISA) Objectives

- ◆ **Enable industry leading system performance**
  - Breakthrough performance
  - Headroom
- ◆ **Enable compatibility with today's IA-32 software & PA-RISC software**
- ◆ **Allow scalability over a wide range of implementations**
- ◆ **Full 64-bit computing**

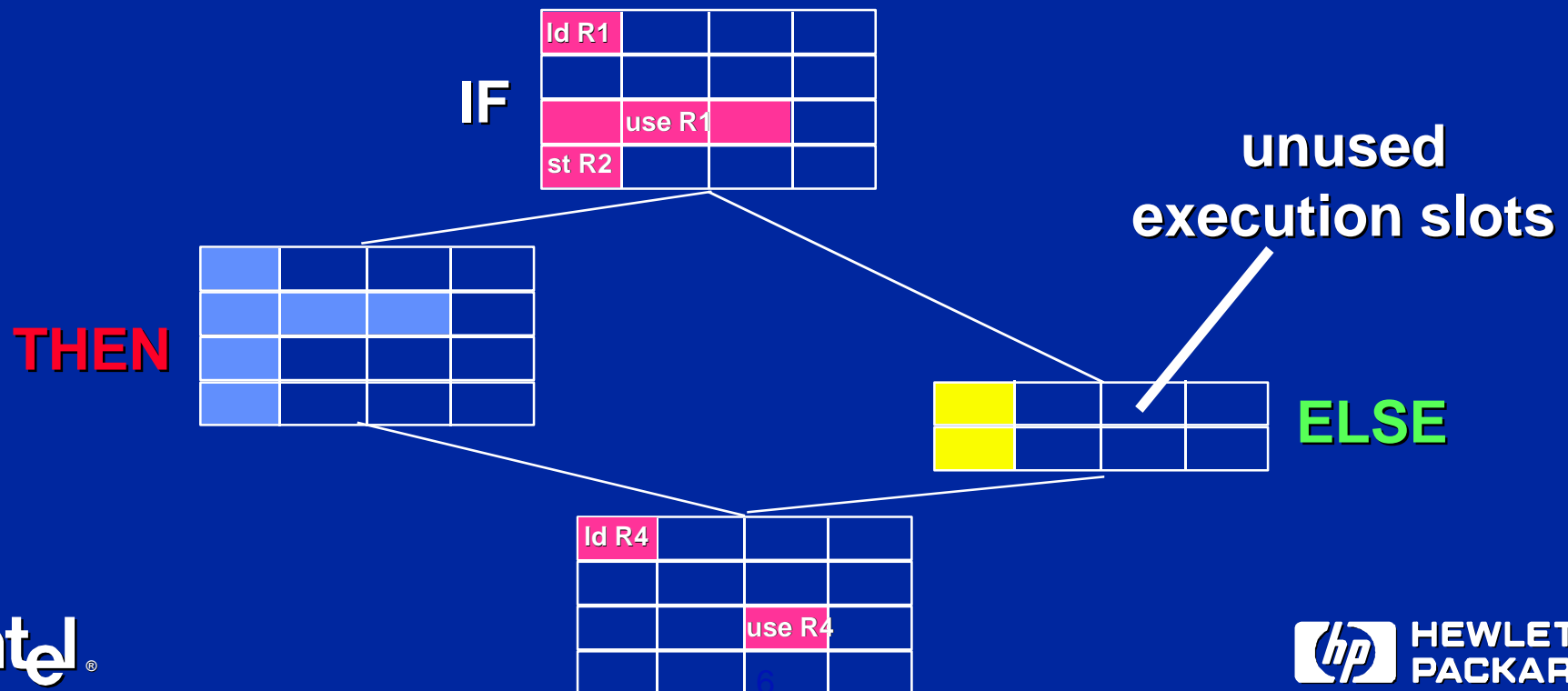
# Current State of The Art



*What's next, beyond traditional architectures?*

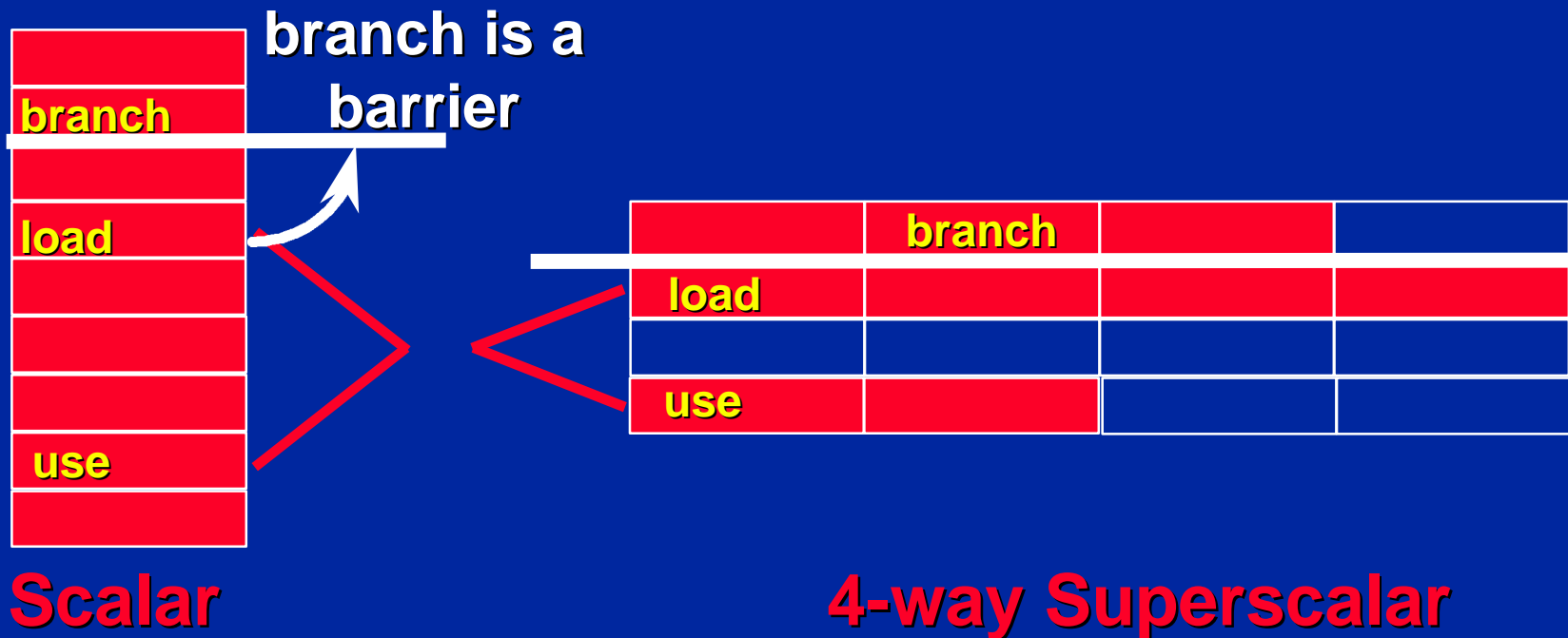
# Current Performance Limiters: *Branches*

- ◆ Mispredicts limit performance
- ◆ Small blocks restrict code scheduling freedom
  - Fragmentation
  - Poor utilization of wide machines



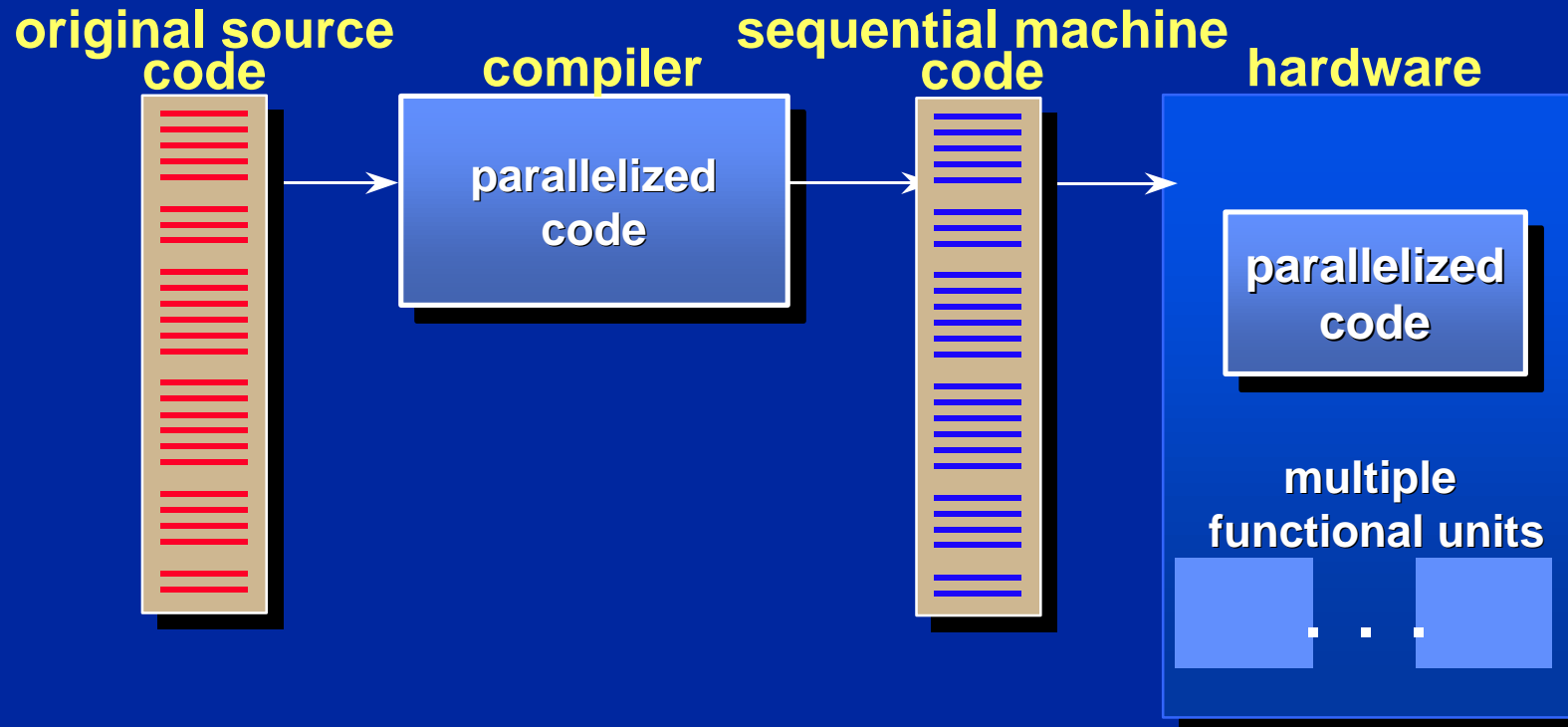
# Current Performance Limiters: *Latency to Memory*

- ◆ Memory latency increasing relative to processor speed
- ◆ Load delay compounded by machine width
  - Latency hiding requires more parallelism in a wide machine



# Current Performance Limiters: *Extracting Parallelism*

- ◆ Sequential execution model

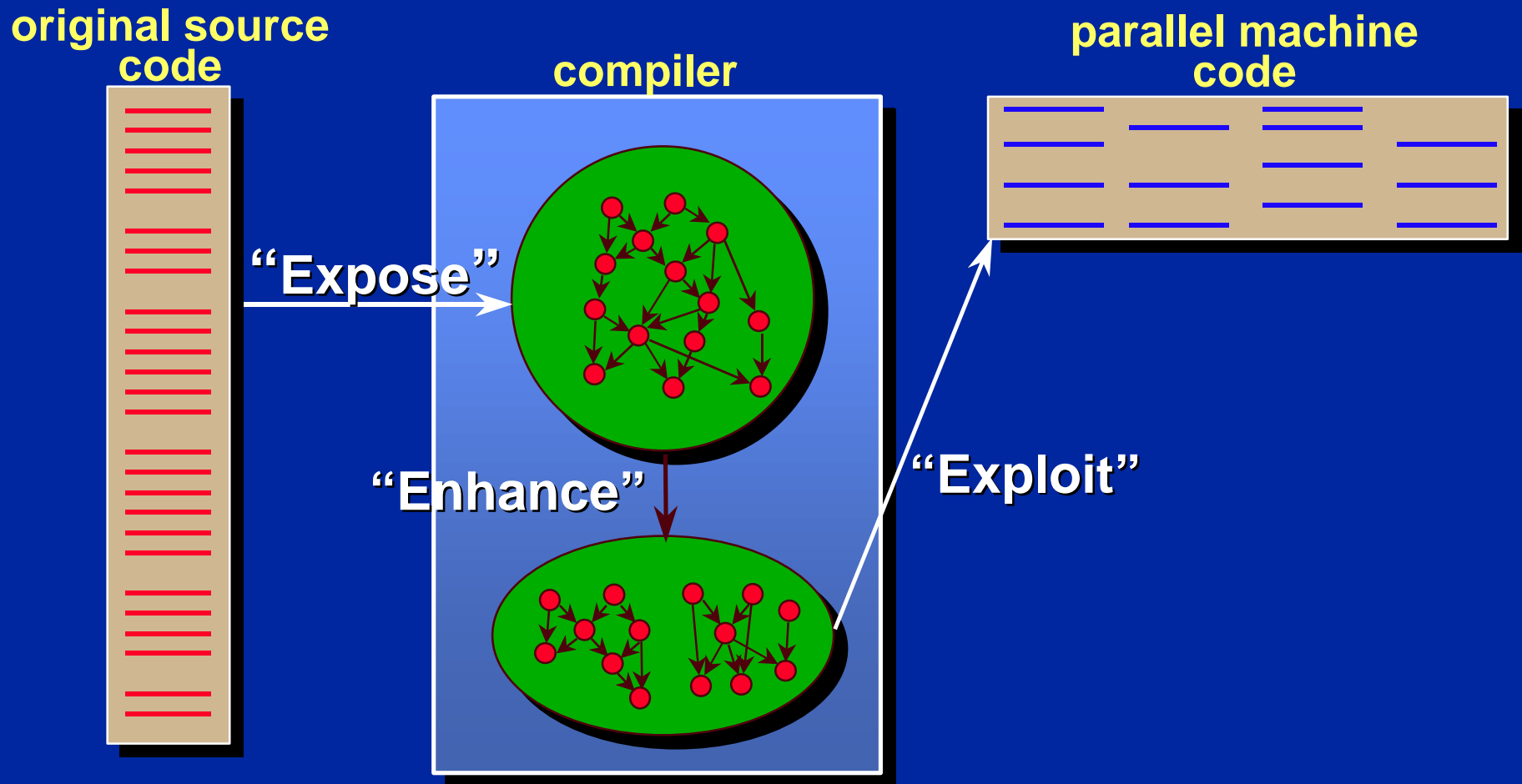


- ◆ Compiler has limited, indirect view of hardware

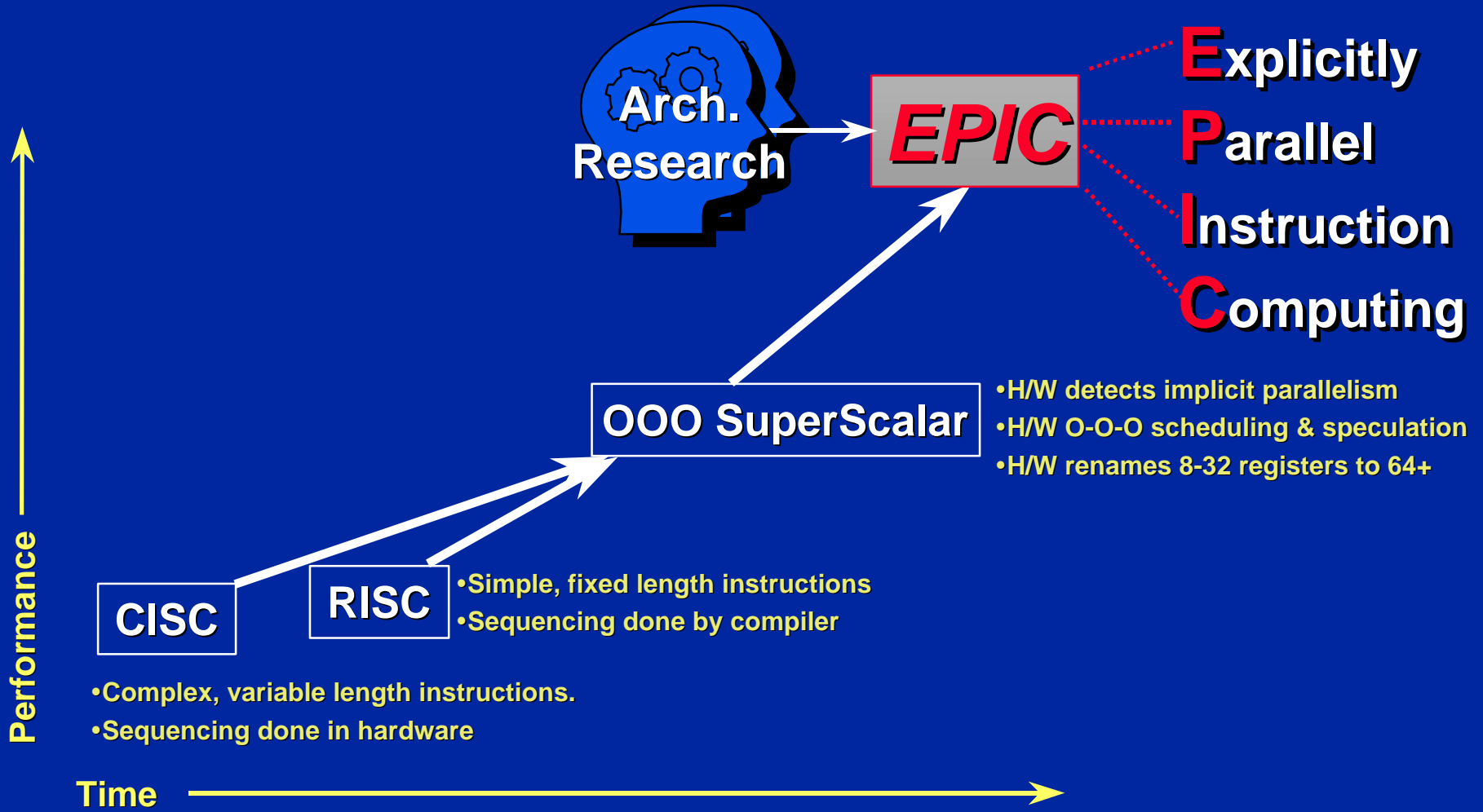
***Implicit parallelism limits performance***

# Better Strategy: Explicit Parallelism

Compiler exposes, enhances, and exploits parallelism in the source program and makes it explicit in the machine code.



# Next Generation Architecture Technology



# Next Generation Terminology

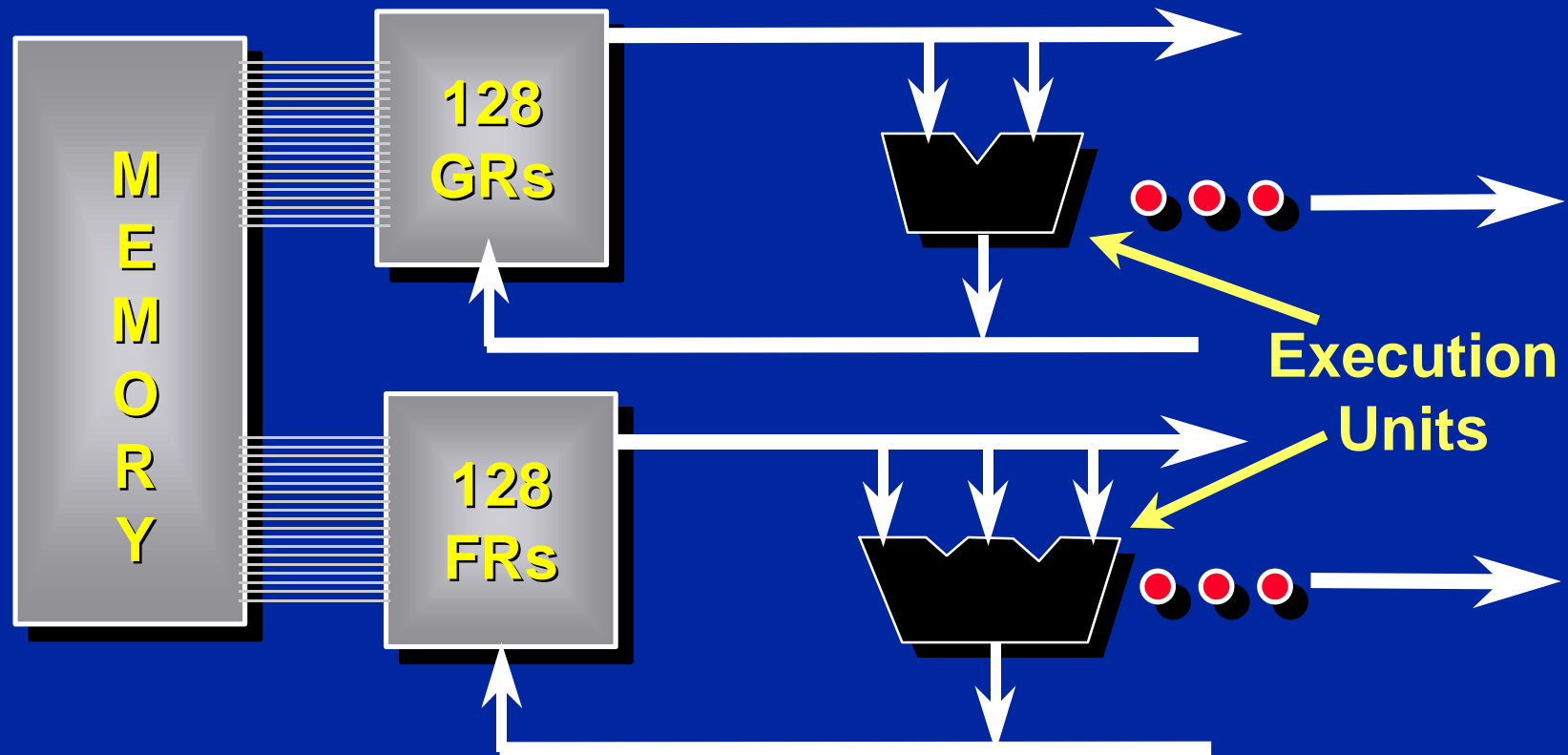
- ◆ **EPIC is the next generation technology**
  - e.g., RISC, CISC
- ◆ **IA-64 is the architecture that incorporates EPIC Technology**
  - e.g., IA-32, PA-RISC
- ◆ **Merced™ processor is the first IA-64-based implementation**
  - e.g., Pentium® II processor, PA-8500

# Key 64-bit ISA Features within IA-64

- ◆ Architecture Resources
- ◆ Instruction Format
- ◆ Predication
- ◆ Speculation
- ◆ *(Branch Architecture)*
- ◆ *(Floating-Point Architecture)*
- ◆ *(Multimedia Architecture)*
- ◆ *(Memory Management & Protection)*
- ◆ *(Compatibility)*



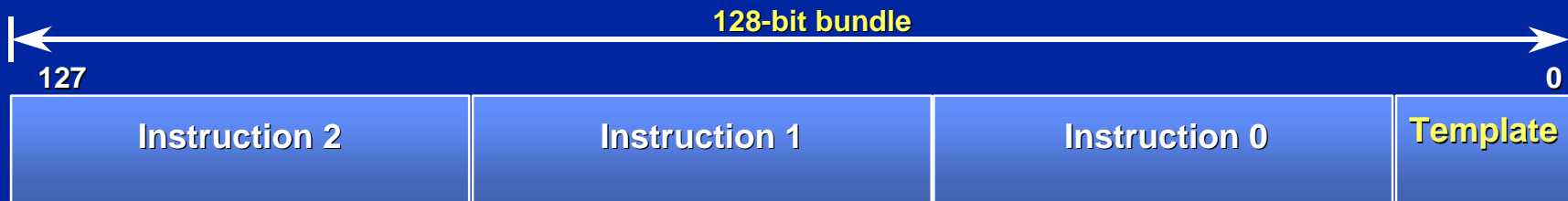
# Architecture Resources Provide for Parallel Execution & Scalability



- ◆ Massively resourced - large register files
  - Traditional architectures are forced to rename registers
- ◆ Inherently scalable - replicated function units
- ◆ Explicitly parallel - transistors used more effectively

# Instruction Format: Explicit Parallelism

- ◆ Breaking the sequential execution paradigm
  - Explicit instruction dependency: template
  - Flexibly groups any number of independent instructions
- ◆ Explicitly scheduled parallelism
  - Enables compiler to create greater parallelism
  - Simplifies hardware by removing dynamic mechanisms
  - Fully interlocked- hardware provides compatibility

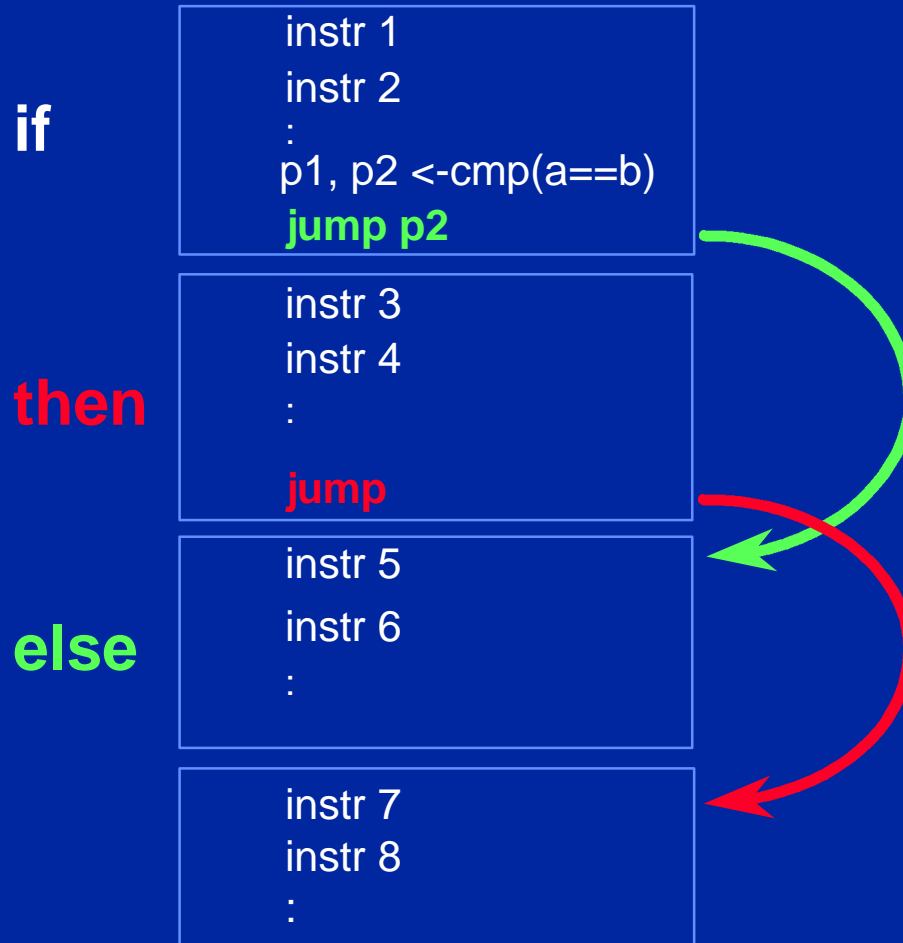


- ◆ Modest code size expansion

*The new instruction format enables scalability w/ compatibility*

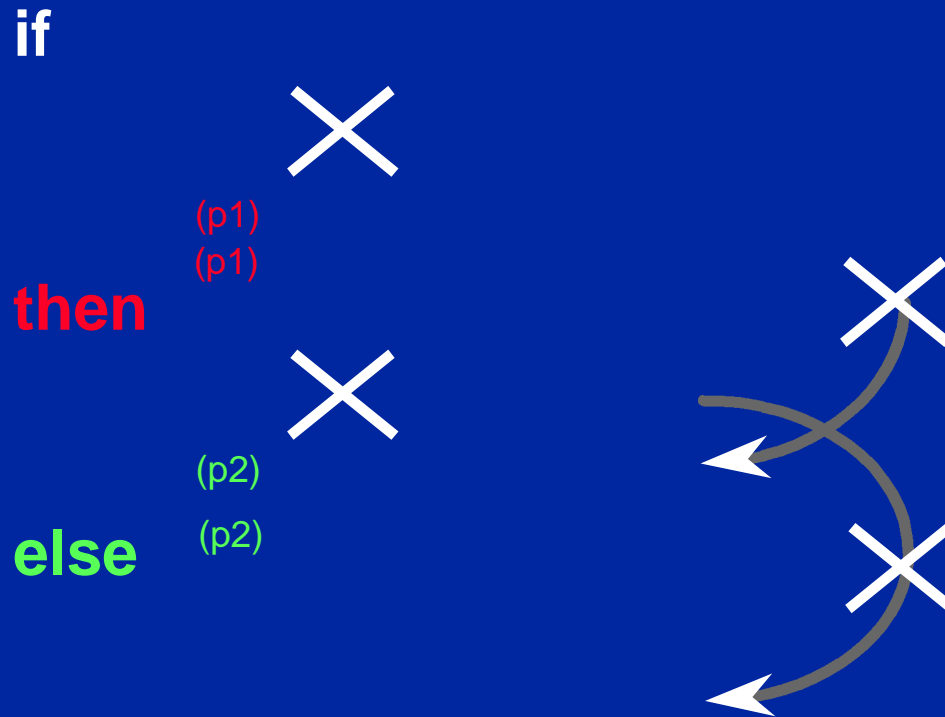
# Branches Limit Performance

Traditional Architectures: 4 basic blocks



***Control flow introduces branches***

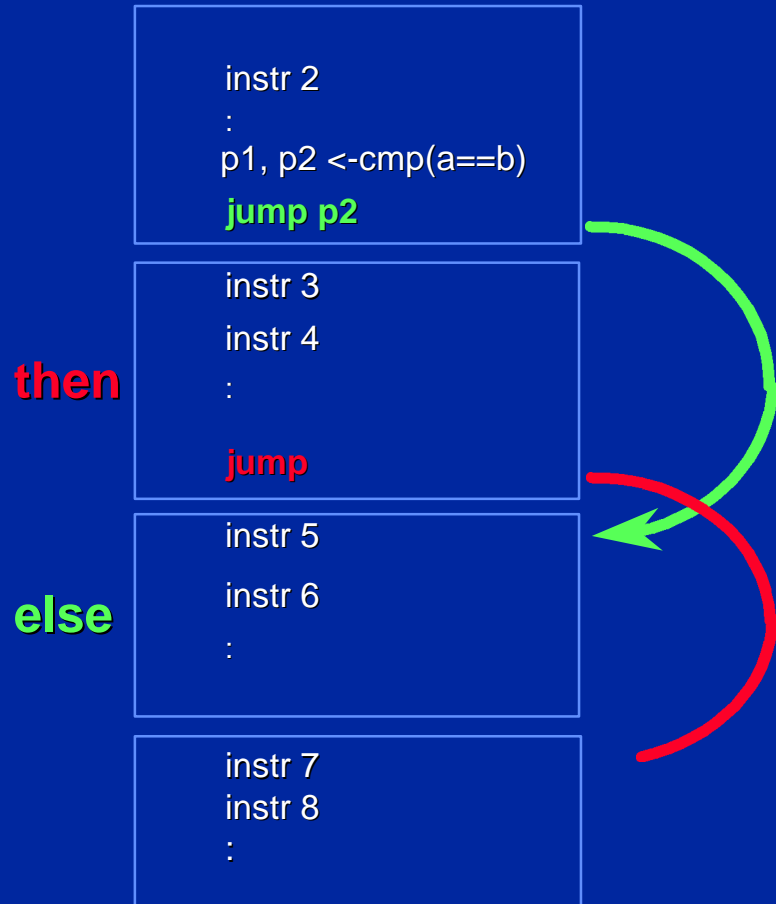
# Predication



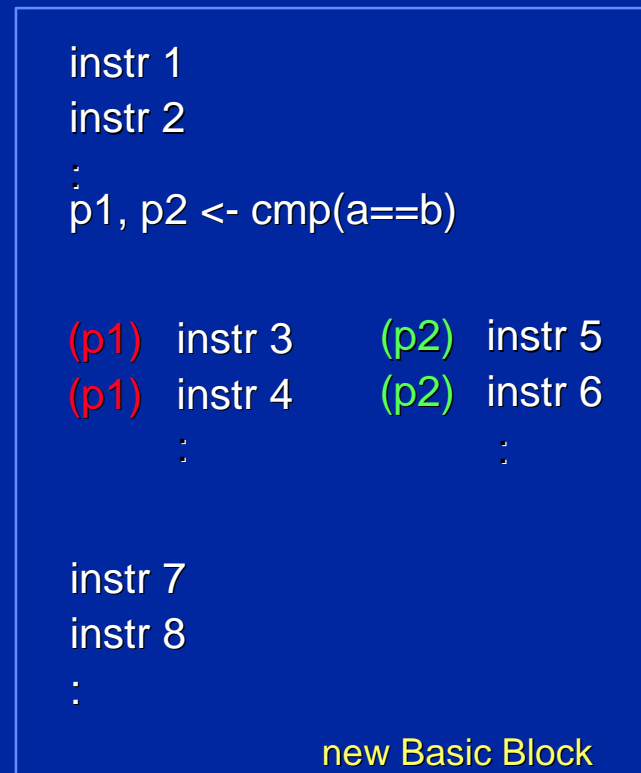
*The predicate can remove branches*

# Predication Enhances Parallelism

Traditional Architectures: 4 basic blocks



EPIC Architectures: 1 basic block

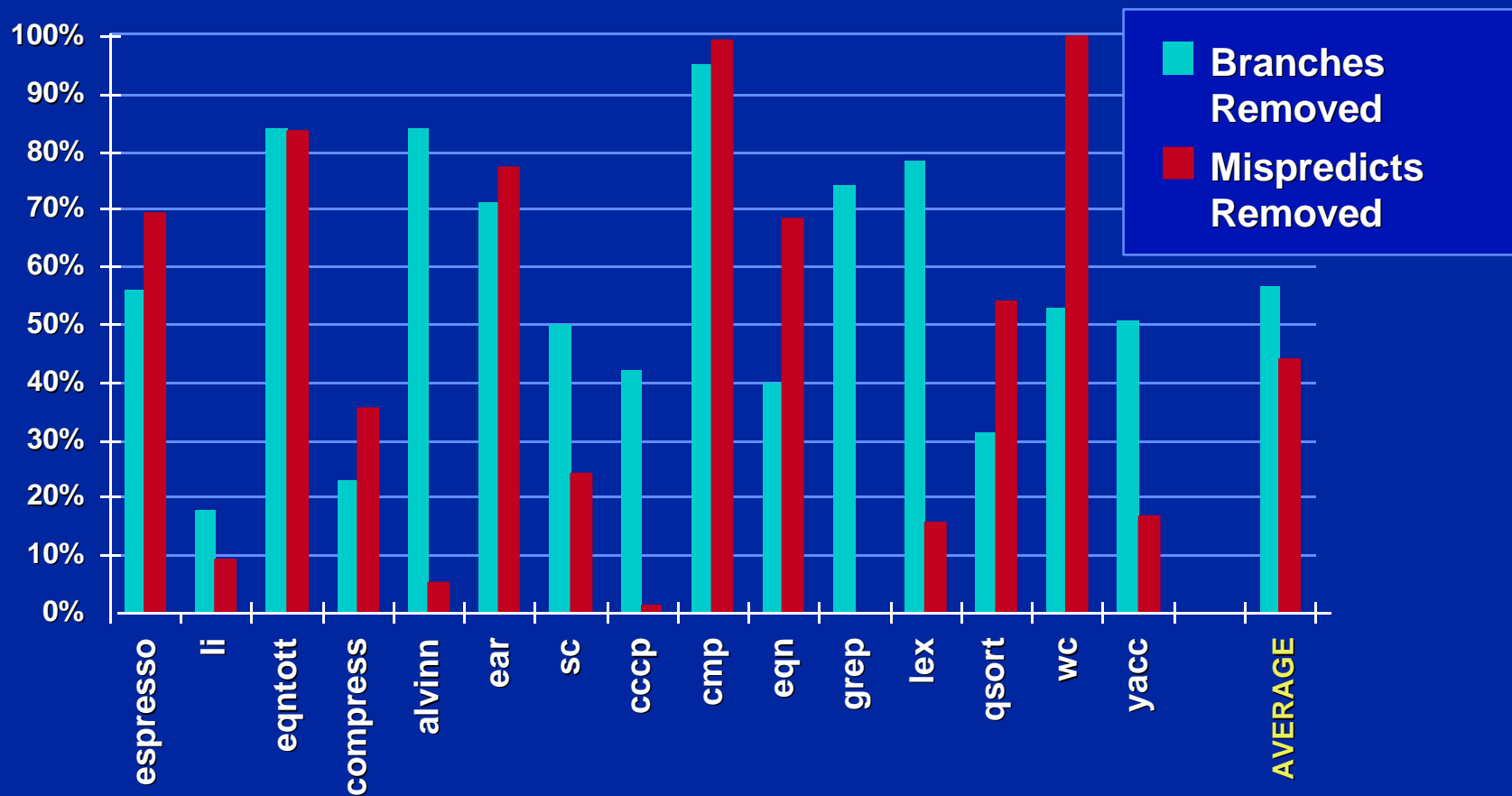


*Predication enables more effective use of parallel hardware*

# Predication: Features and Benefits

- ◆ **Compiler given larger scheduling scope**
  - Nearly all instructions can be predicated
  - State updated if an instruction's predicate is true, otherwise acts as a NOP
  - Compiler assigns predicates, compare instructions set them
  - Architecture provides 64 1-bit predicate registers (PR)
- ◆ **Predicated execution removes branches**
  - Convert a control dependence to a data dependence
  - Reduce mispredict penalties
- ◆ **Parallel execution through larger basic blocks**
  - Effective use of parallel hardware

# Predication Increases Performance



***On average, over half of all branches are removed***



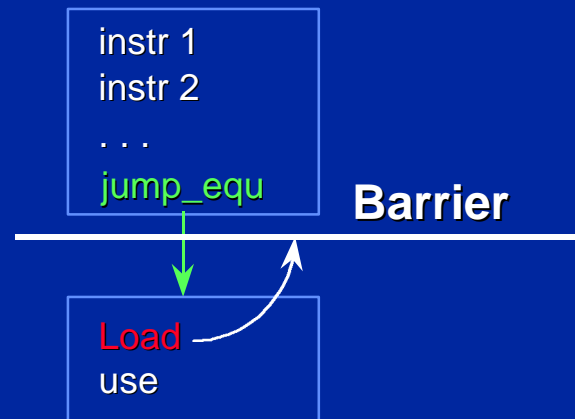
Source: ISCA '95 S.Mahlke, et.al.



# Memory Latency Causes Delays

- ◆ Loads significantly affect performance
  - Often first instruction in dependency chain of instructions
  - Can incur high latencies

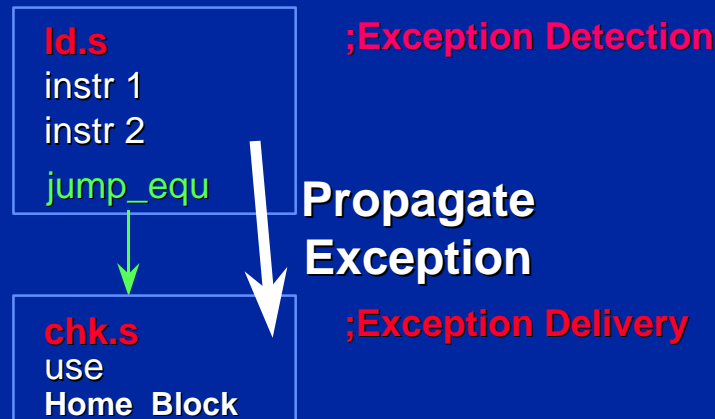
## Traditional Architectures



- ◆ Loads can cause exceptions

# Speculation

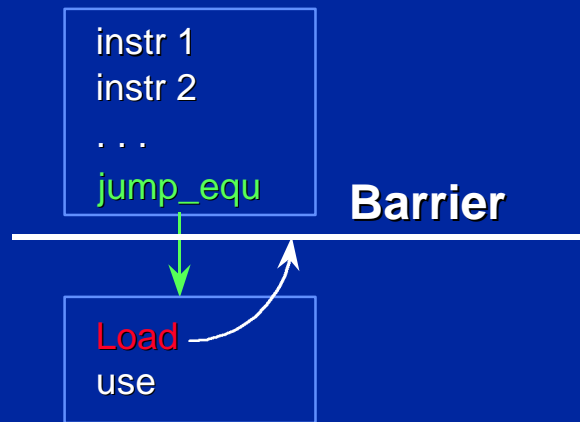
## EPIC Architectures



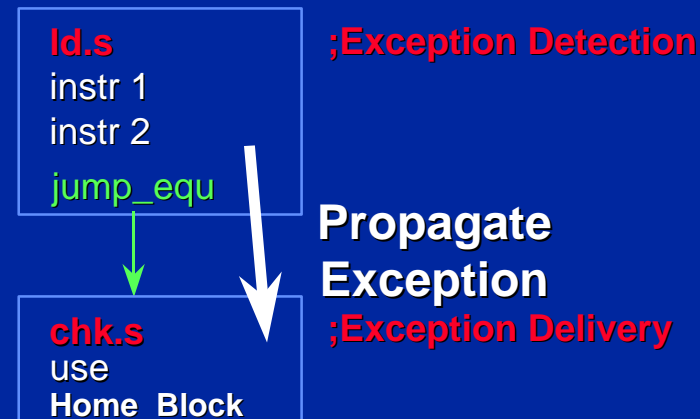
- ◆ **Separate load behavior from exception behavior**
  - Speculative load instruction (**ld.s**) initiates a load operation and detects exceptions
  - Propagate an exception “**token**” (stored with destination register) from **ld.s** to **chk.s**
  - Speculative check instruction (**chk.s**) delivers any exceptions detected by ld.s

# Speculation Minimizes the Effect of Memory Latency

## Traditional Architectures



## EPIC Architectures



- ◆ Give scheduling freedom to the compiler
  - Allows **ld.s** to be scheduled above branches
  - **chk.s** remains in home block, branches to fixup code if an exception is propagated

# Example: 8 Queens Loop

if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))

## Original Code

```

1  R1=&b[j]
   R3=&a[i+j]
   R5=&c[i-j+7]
2  ld R2=[R1]
4  P1,P2 <-cmp(R2==true)
5  <P2> br exit

```

True 38%  
Mispred 43%

```

6  ld R4=[R3]
8  P3,P4 <-cmp(R4==true)
9  <P4> br exit

```

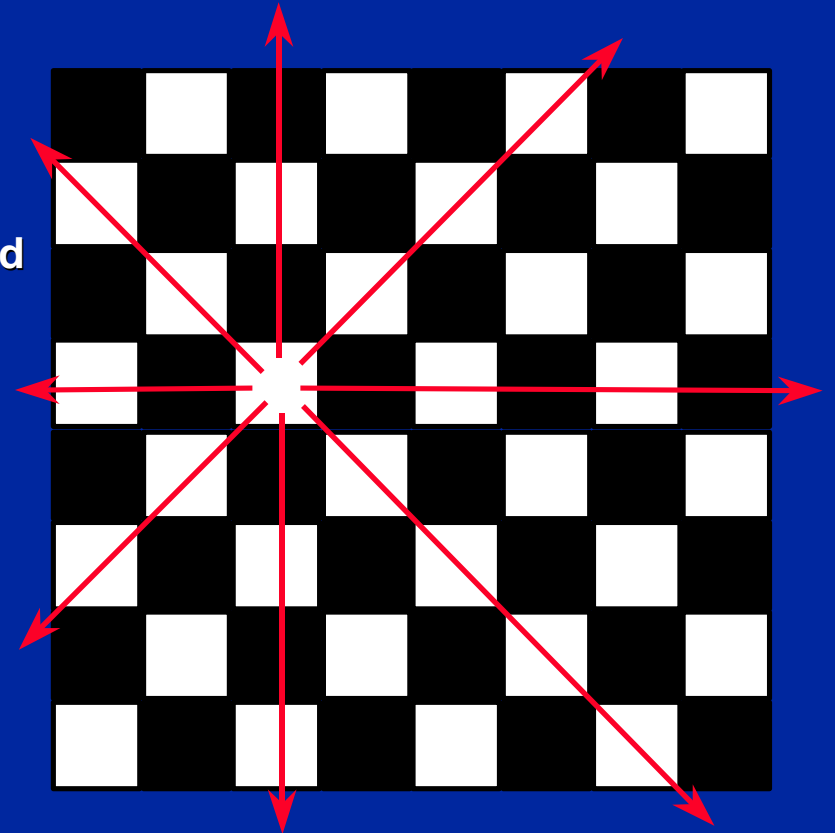
72% 33%

```

10 ld R6=[R5]
12 P5,P6 <-cmp(R5==true)
13 <P5> br then
    else

```

47% 39%



13 cycles

intel® 3 potential mispredicts

# Example: 8 Queens Loop

if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))

## Original Code

1	R1=&b[j] R3=&a[i+j] R5=&c[i-j+7]
2	ld R2=[R1]
4	P1,P2 <-cmp(R2==true)
5	<P2> br exit
6	ld R4=[R3]
8	P3,P4 <-cmp(R4==true)
9	<P4> br exit
10	ld R6=[R5]
12	P5,P6 <-cmp(R5==true)
13	<P5> br then else

13 cycles

intel® 3 potential mispredicts

## Speculation

1	R1=&b[j] R3=&a[i+j] R5=&c[i-j+7] ld R2=[R1] ld.s R4=[R3] ld.s R6=[R5]
2	
4	P1,P2 <-cmp(R2==true)
5	<P2> br exit
6	chk.s R4 P3,P4 <-cmp(R4==true)
7	<P4> br exit
8	chk.s R6 P5,P6 <-cmp(R5==true)
9	<P5> br then else

9 cycles

3 potential mispredicts



# Example: 8 Queens Loop

if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))

## Speculation

1	R1=&b[j] R3=&a[i+j] R5=&c[i-j+7]
2	ld R2=[R1] ld.s R4=[R3] ld.s R6=[R5]
4	P1,P2 <-cmp(R2==true)
5	<P2> br exit
6	chk.s R4 P3,P4 <-cmp(R4==true)
7	<P4> br exit
8	chk.s R6 P5,P6 <-cmp(R5==true)
9	<P5> br then else

9 cycles

3 potential mispredicts

## Predication

1	R1=&b[j] R3=&a[i+j] R5=&c[i-j+7]
2	ld R2=[R1] ld.s R4=[R3] ld.s R6=[R5]
4	P1,P2 <-cmp(R2==true) <P2> br exit
5	<p1> chk.s R4 <p1> P3,P4 <-cmp(R4==true) <P4> br exit
6	<p3> chk.s R6 <p3> P5,P6 <-cmp(R5==true)
7	<P5> br then else

7 cycles

1 potential mispredict

True Mispred  
12% 16%



# Example: 8 Queens Loop

if ((b[j] == true) && (a[i+j] == true) && (c[i-j+7] == true))

## Original Code

```

1  R1=&b[j]
   R3=&a[i+j]

```

## Predication

```

1  R1=&b[j]
   R3=&a[i+j]

```

**RESULT: Almost half the required cycles are reduced and 2/3 of the potential mispredicts are eliminated.**

```

10  ld R6=[R5]
12  P5,P6 <-cmp(R5==true)
13  <P5> br then
    else

```

13 cycles

3 potential mispredicts

```

6   <p3> chk.s R6
   <p3> P5,P6 <-cmp(R5==true)
7   <P5> br then
    else

```

7 cycles

1 potential mispredict



# EPIC is the Next Generation Technology

## *Explicitly Parallel Instruction Computing*

### ◆ Explicit parallelism

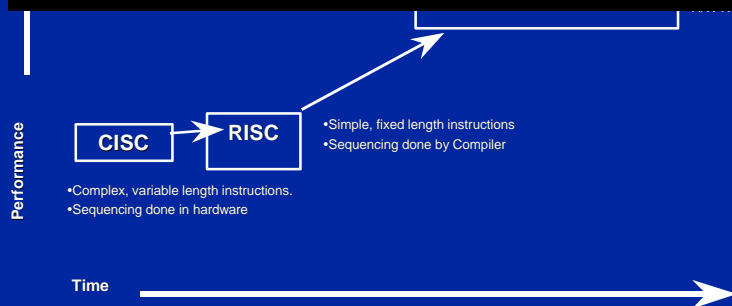
- ILP is explicit in machine code
- Compiler schedules across a wide scope
- Binary compatibility across all family members

### ◆ Features that enhance ILP

- Predication
- Speculation
- Others...

### ◆ Resources for parallel execution

- Many registers
- Many functional units
- Inherently scalable



# IA-64: EPIC Technology Applied

- ◆ **Enables industry leading performance and capability**
  - Explicitly parallel: Beyond the limitations of current architectures
  - Inherently scalable, massively resourced: Provides headroom for future market requirements
  - Fully compatible: For existing applications and the future
- ◆ **Addresses server and workstation market requirements**
  - Enterprise transaction processing
  - Decision support
  - Graphical imaging
  - Volume rendering
  - Many others

***The Next Generation in Computer Architecture***