# From Instruction Level To Thread Level Parallelism

**John Paul Shen**
**Director of Microarchitecture Research**
**Intel Labs**

October 29, 2002
Microprocessor Research Forum

# "Iron Law" of Microprocessor Performance

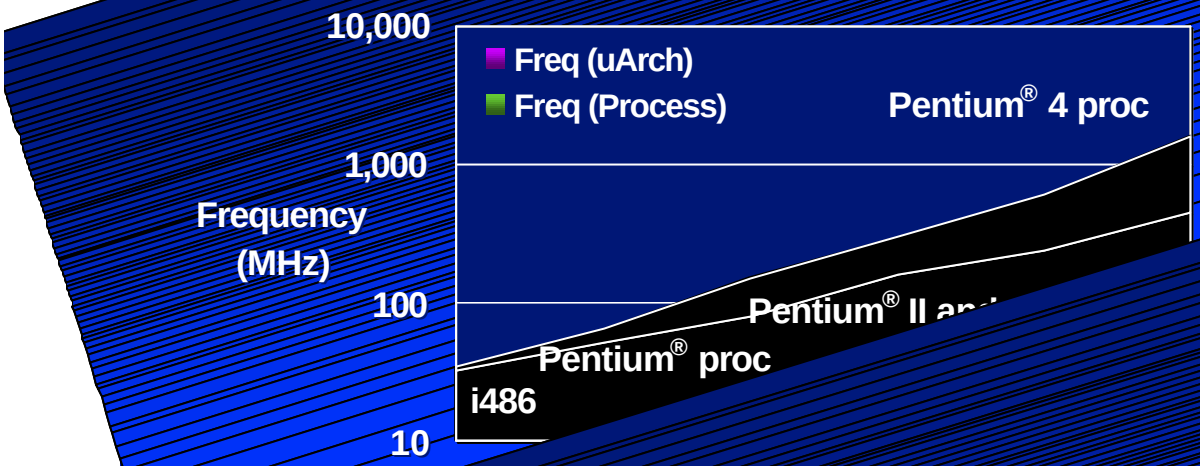
$$\text{1/Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(inst. count) (CPI) (cycle time)

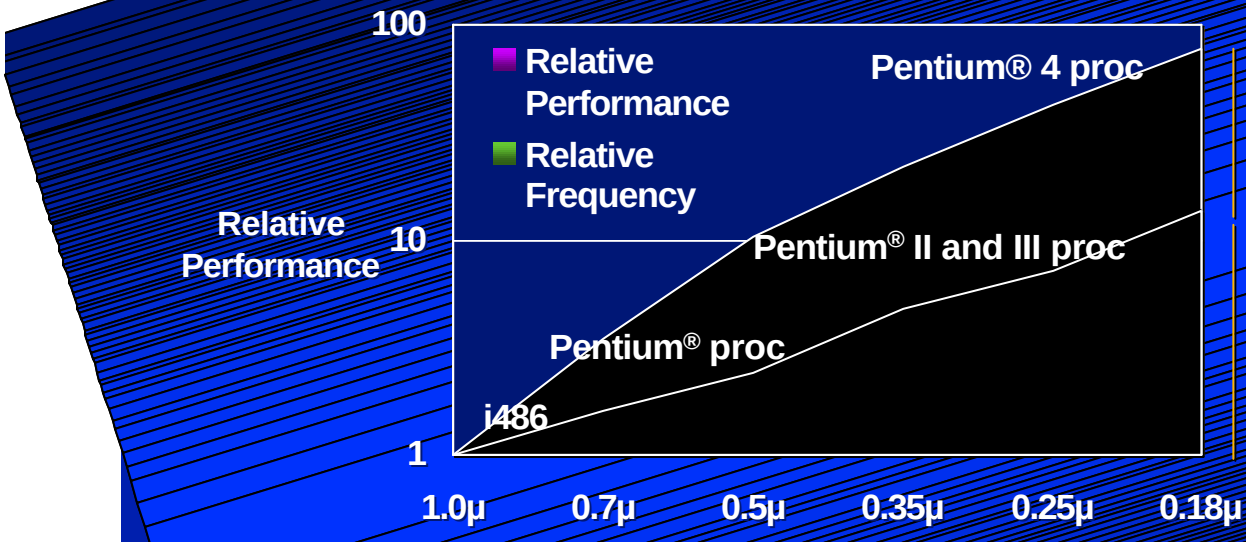
$$\text{Processor Performance} = \frac{\text{IPC x GHz}}{\text{inst. count}}$$

# Freq




**Frequency Increa 50X**

- 12



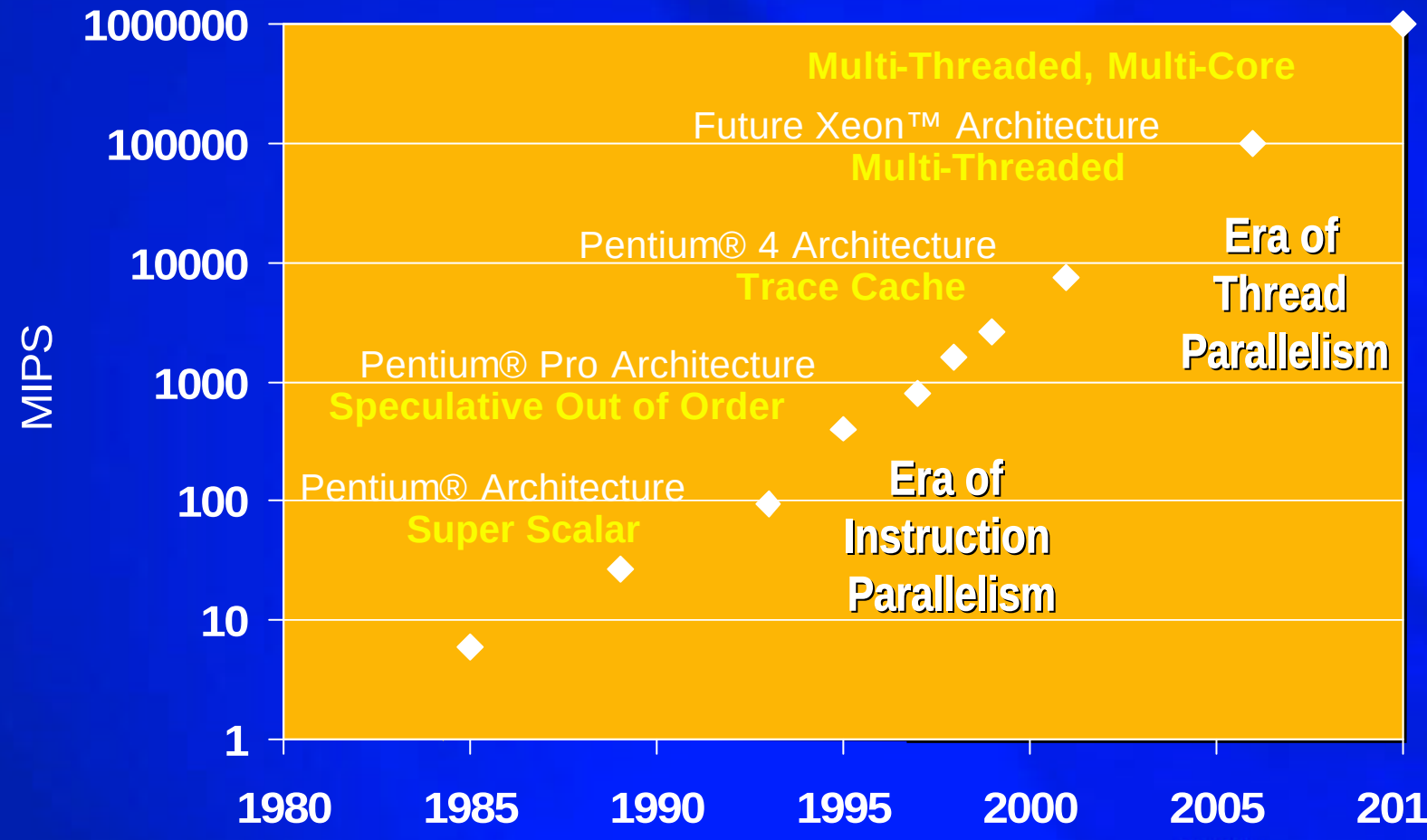

**Performance Increased >75X**

- 13X due to process technology
- Additional >6X due to microarchitecture

***Note: Performance measured using SpecINT and SpecFP**

Source: Intel Corporation

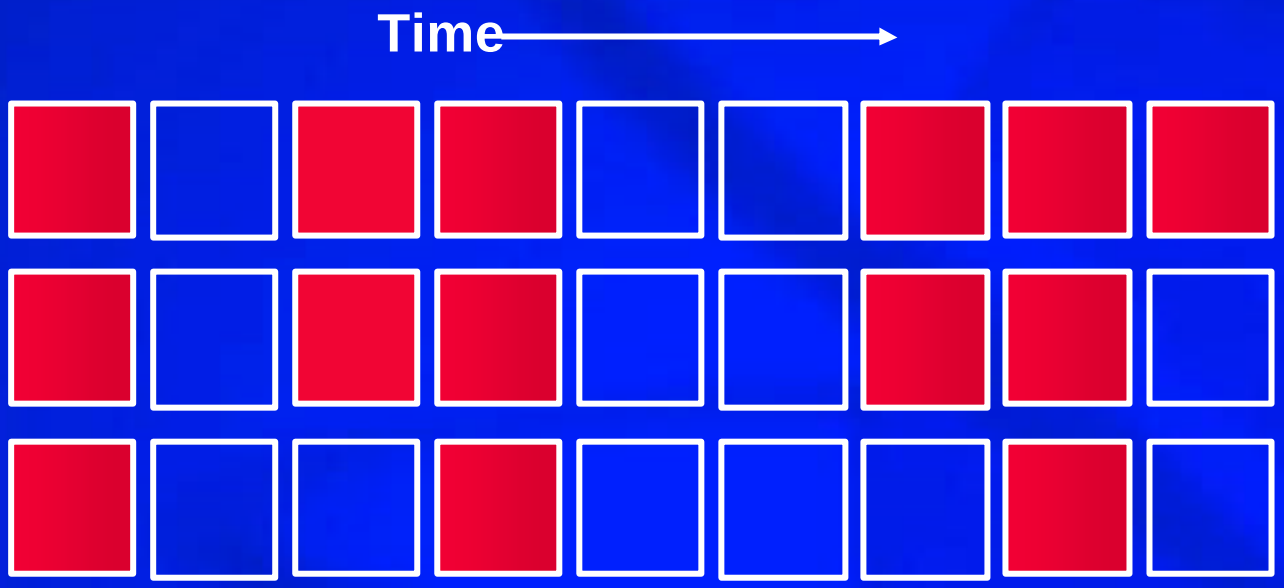
# Parallelism in Transition



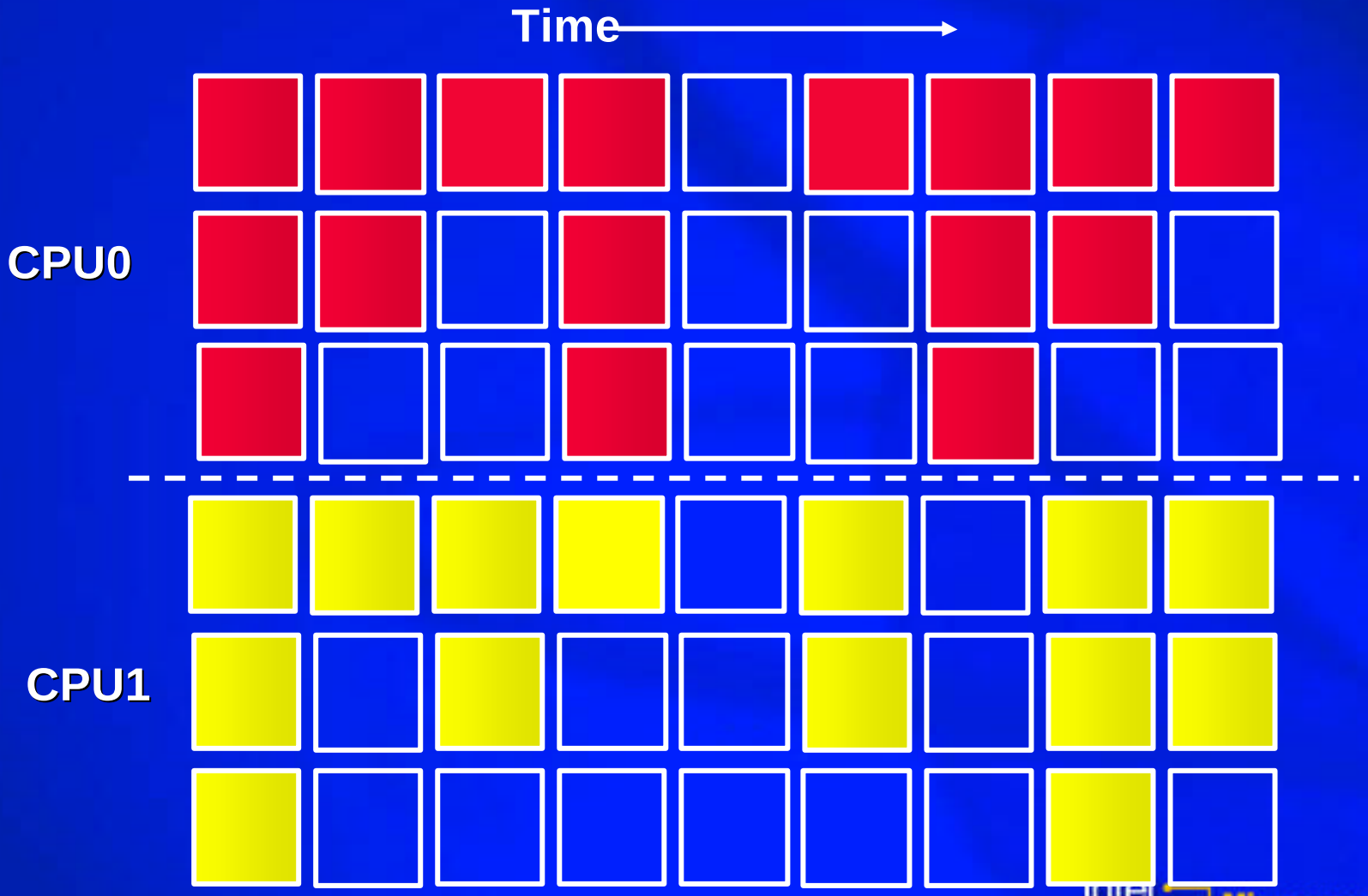

intel

Intel Microprocessor Research Forum 2002

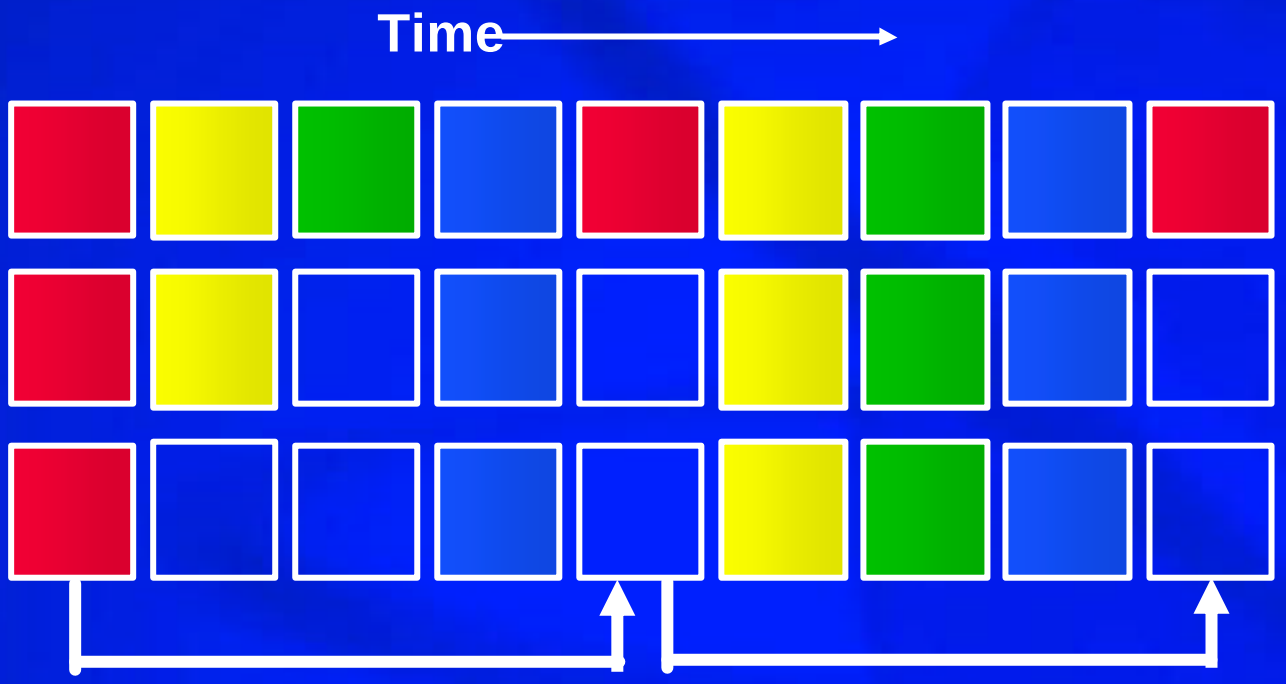
# Superscalar Issue



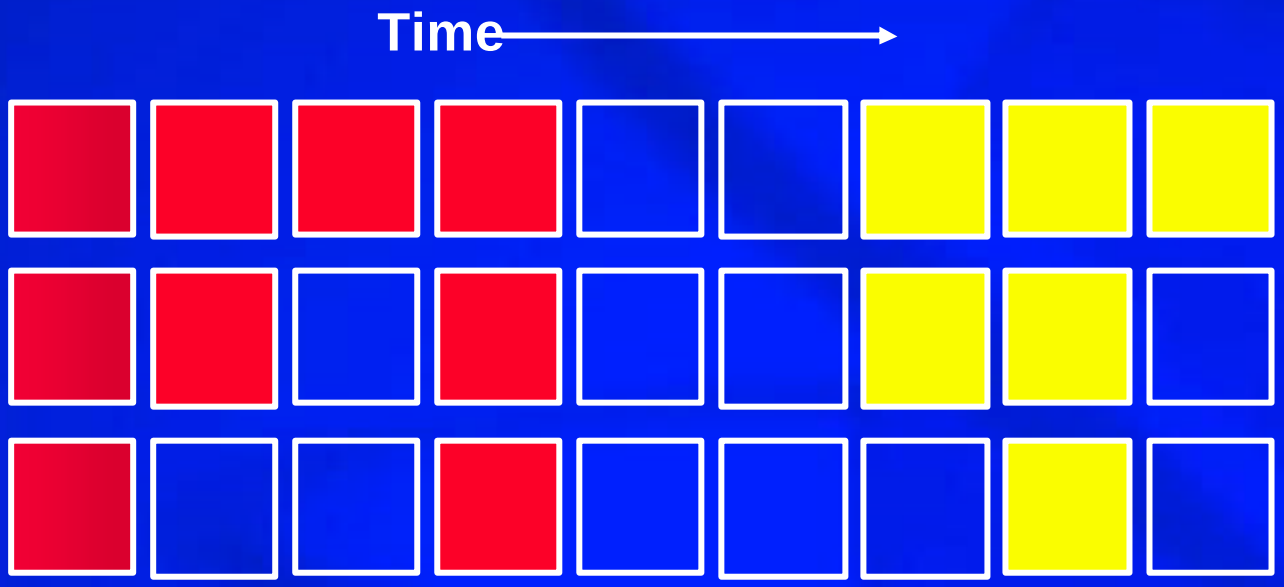
# Chip Multiprocessor (CMP)



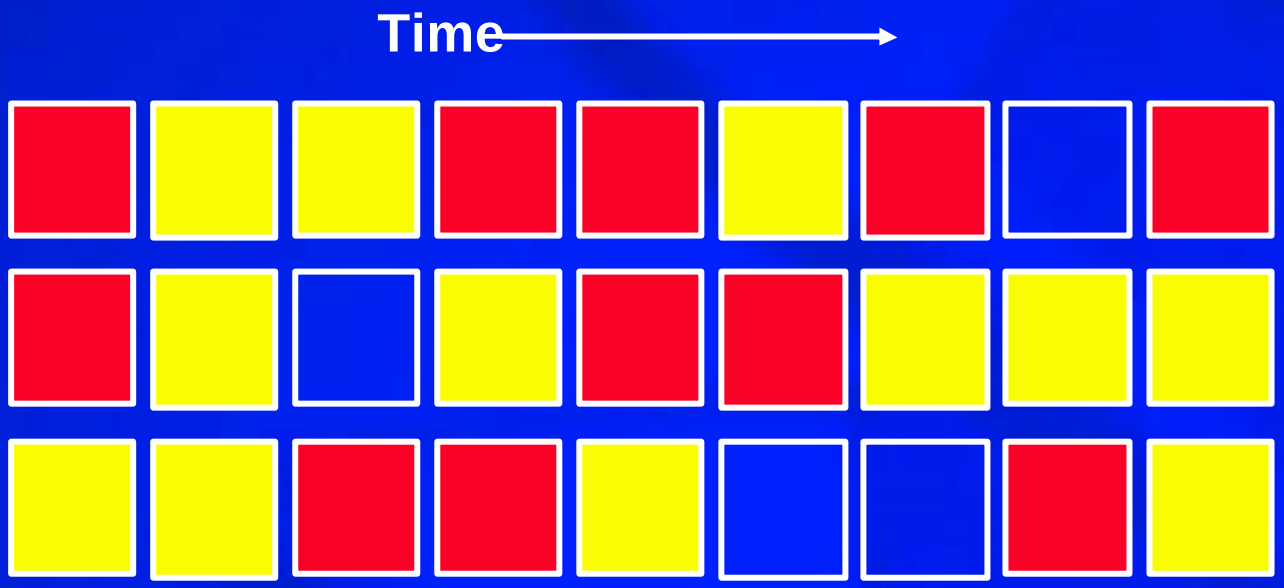
# Time-slicing Multi-Threading



# Switch-on-Event Multi-Threading

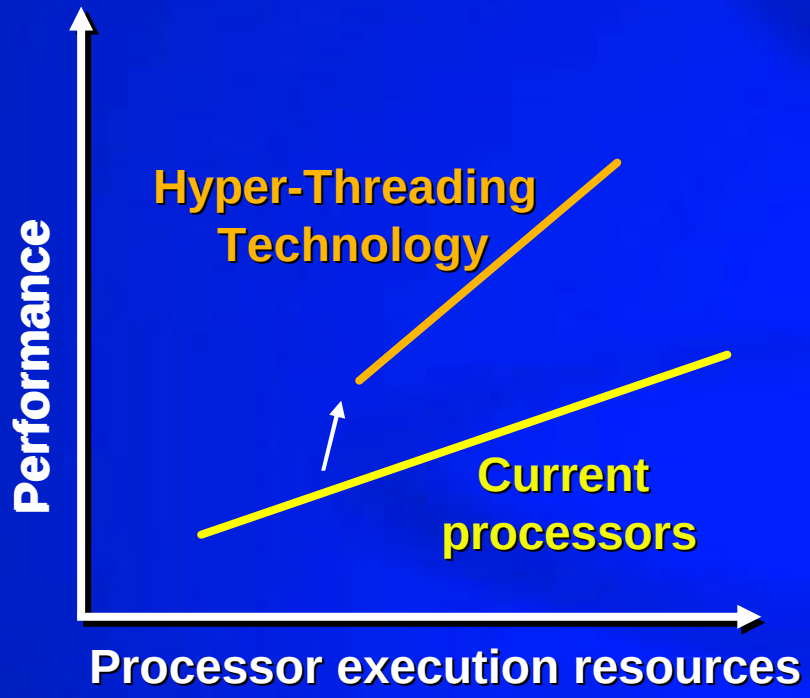


# Simultaneous Multi-Threading (SMT)




**Maximum utilization of function units by independent operations**

# Accelerate Performance of Threaded Applications



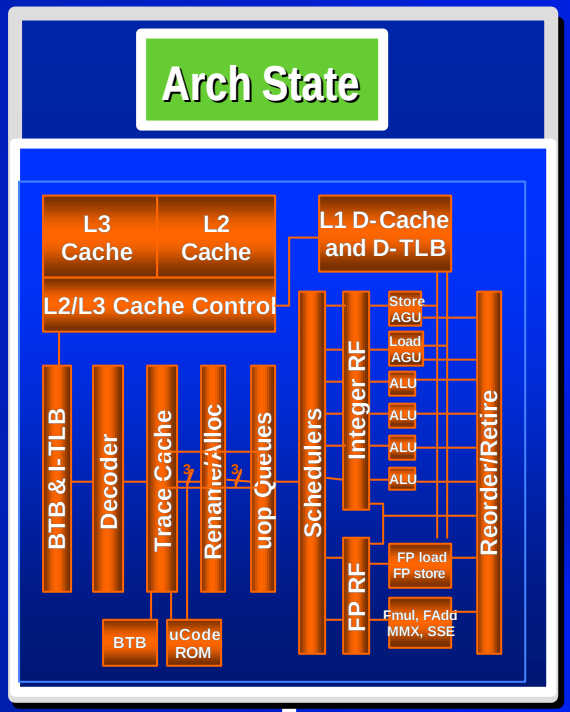

- SMT: most efficient/highest performance option
- More performance for CPU execution resources
  - Uses resources more fully
- Greater performance improvements from additional processor resources

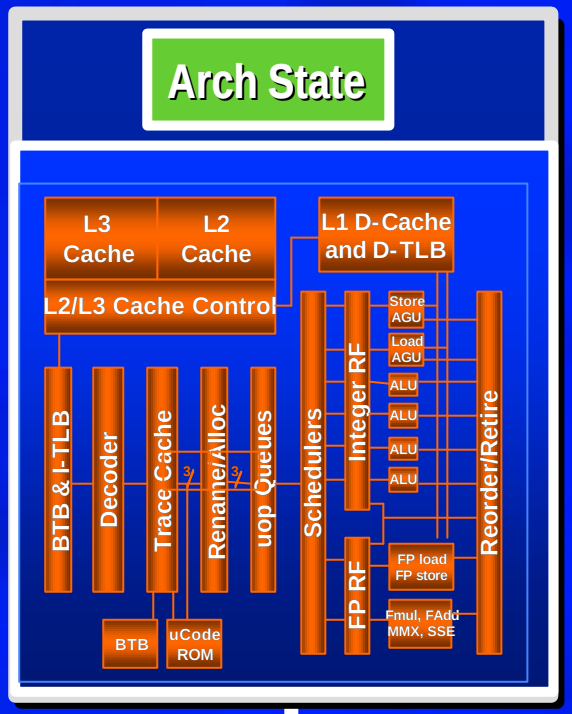
# Hyper-Threading Technology

- Executes two tasks simultaneously
  - Two different applications
  - Two threads of same application
- CPU maintains architecture state for two processors
  - Two logical processors per physical processor
- Demonstrated on prototype Intel® Xeon™ Processor MP
  - Two logical processors for < 5% additional die area
  - Power efficient performance gain
  - Result of significant research, design effort, and validation

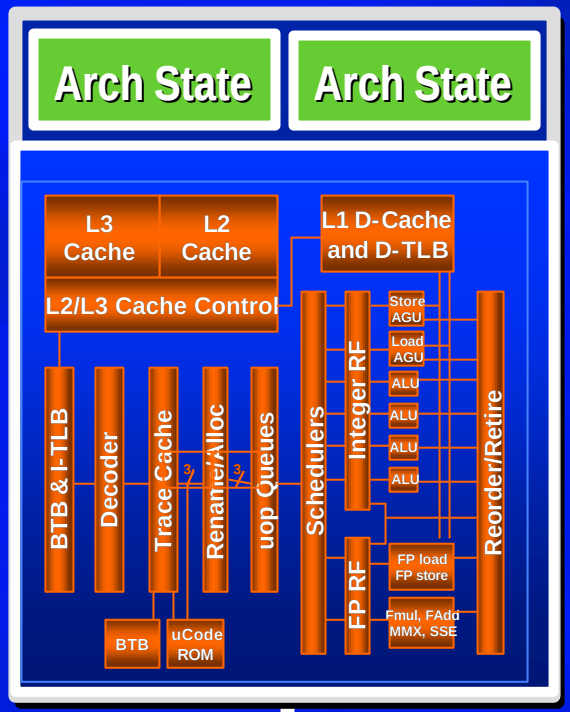
# Replicated vs. Shared Resources

## Multiprocessor




## Hyper-Threading




**Multi-Processors replicate execution resources**
**Hyper-Threading Technology shares resources**

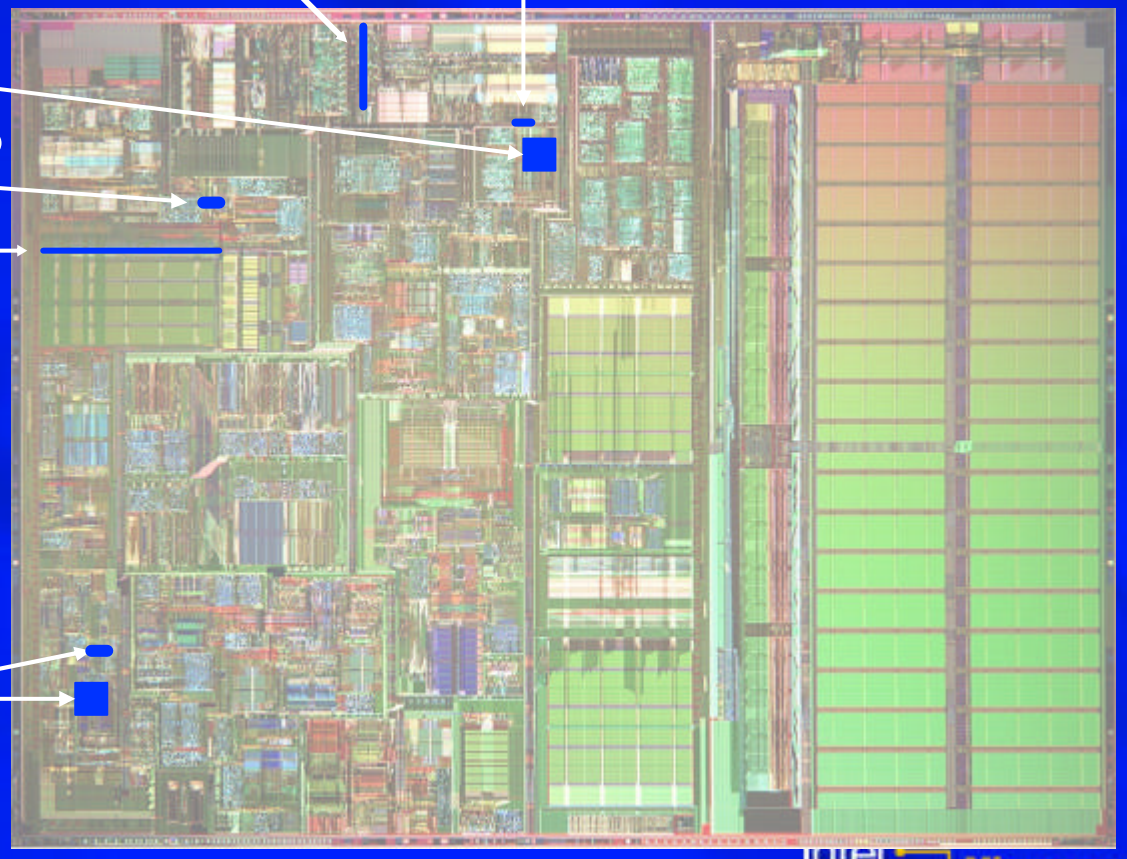
intel

Microprocessor Research Forum 2002

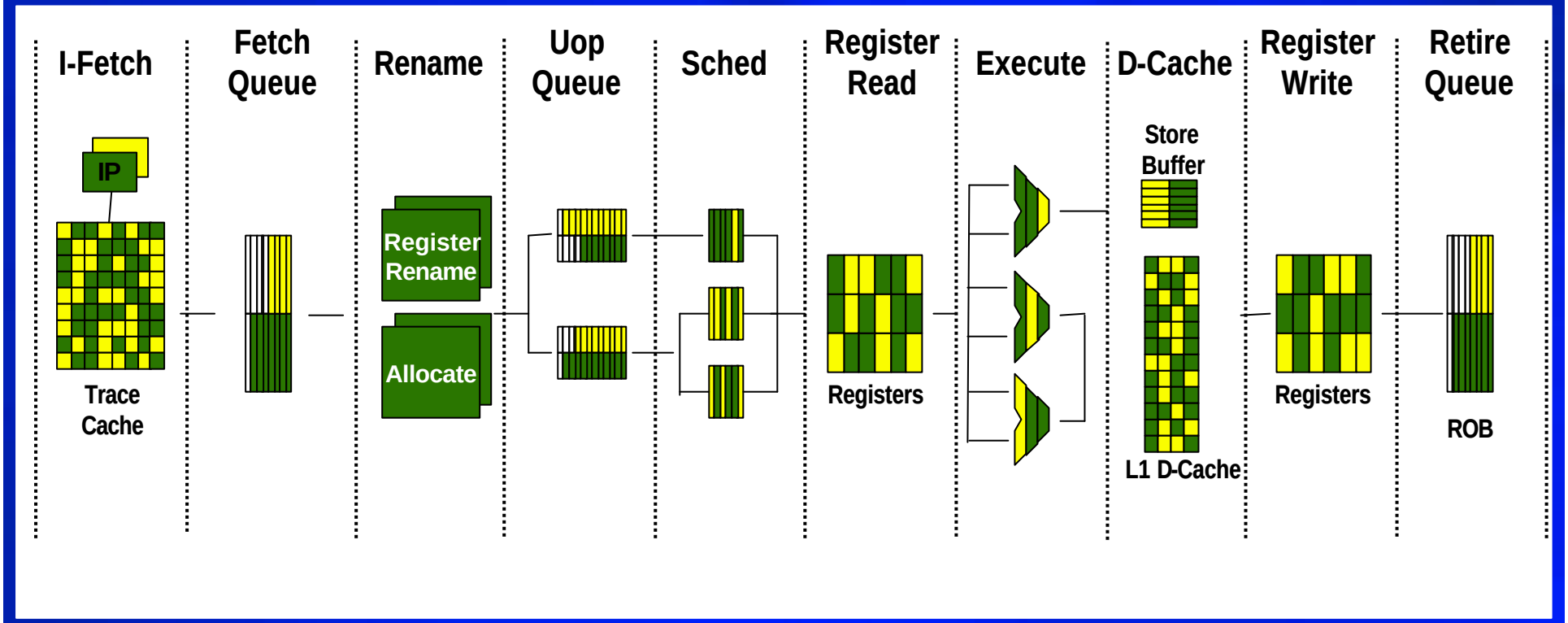
# Changes for Hyper-Threading

- Replicate resources
  - All per-CPU architectural state
  - Instruction Pointers, renaming logic
  - Some smaller resources
    - E.g, return stack predictor, ITLB, etc
- Partition resources
  - Several buffers (Re-order buffer, load/store suffers, queues,etc)
- Share most resources
  - Out-of-Order execution engine
  - Caches

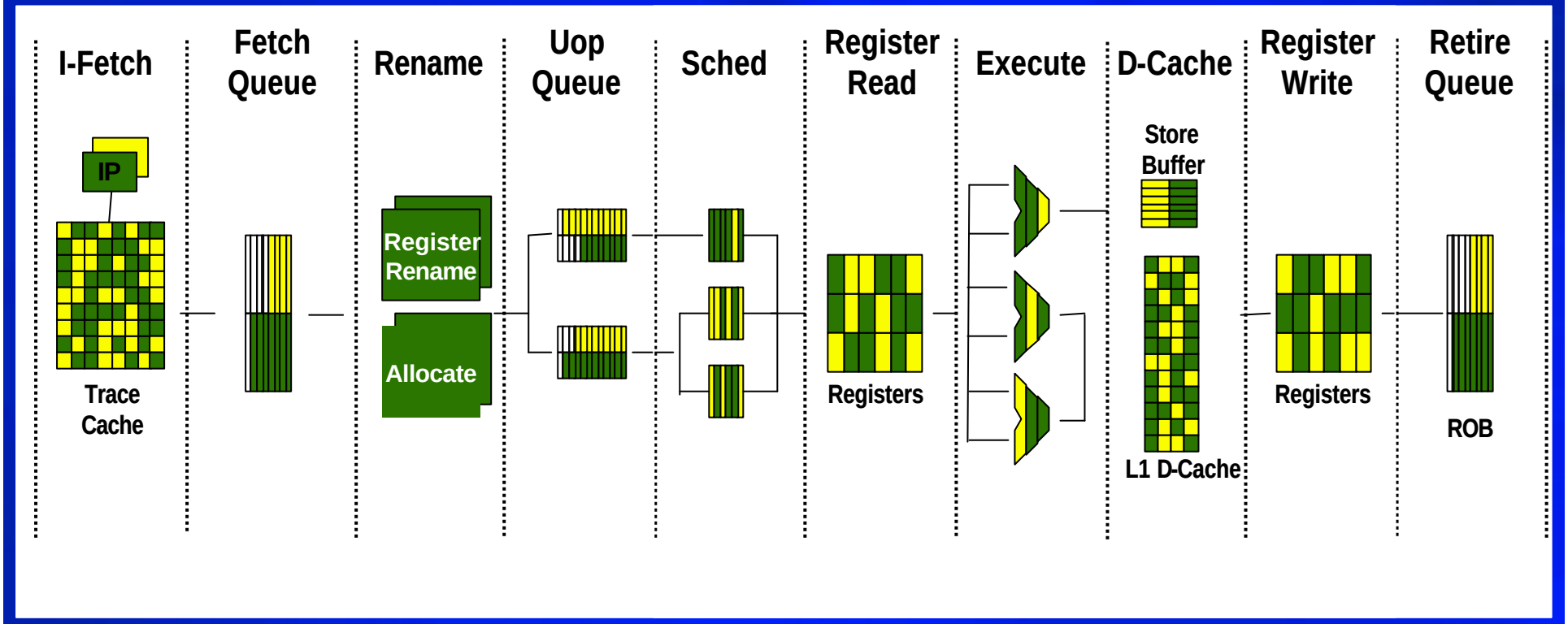
# What Was Added



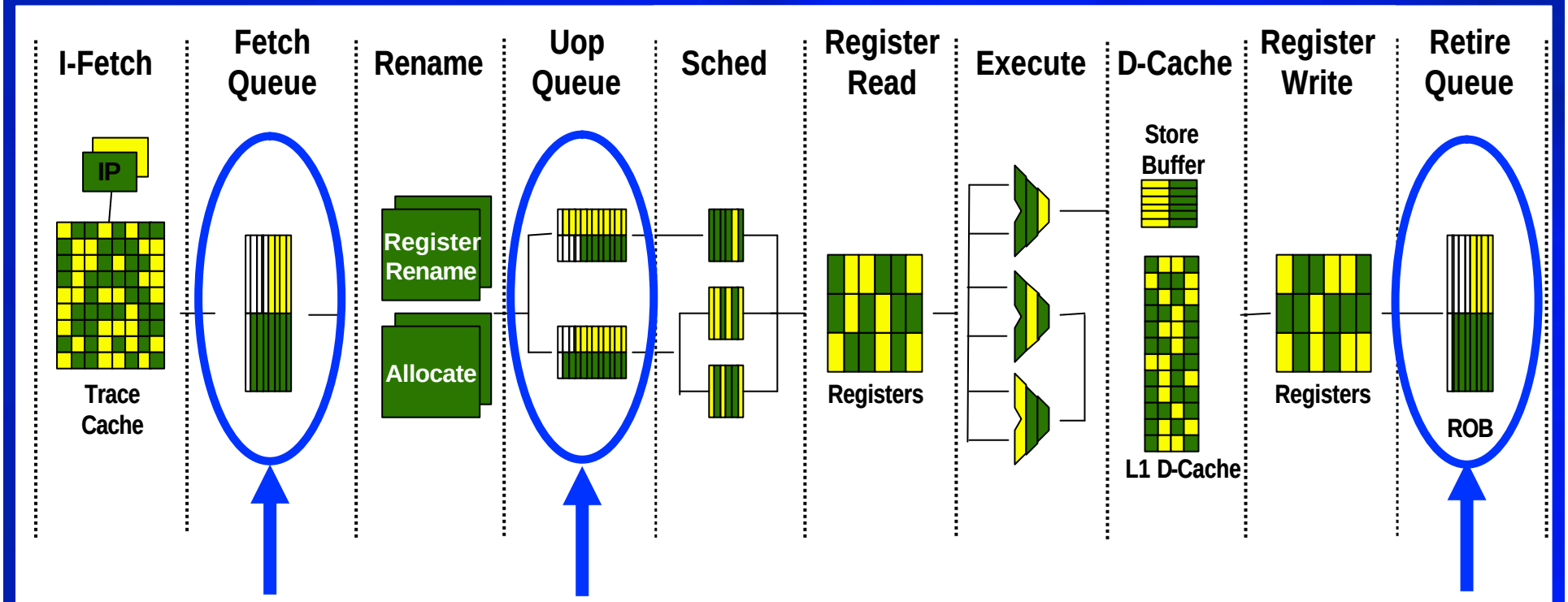
# Execution Pipeline



# Execution Pipeline

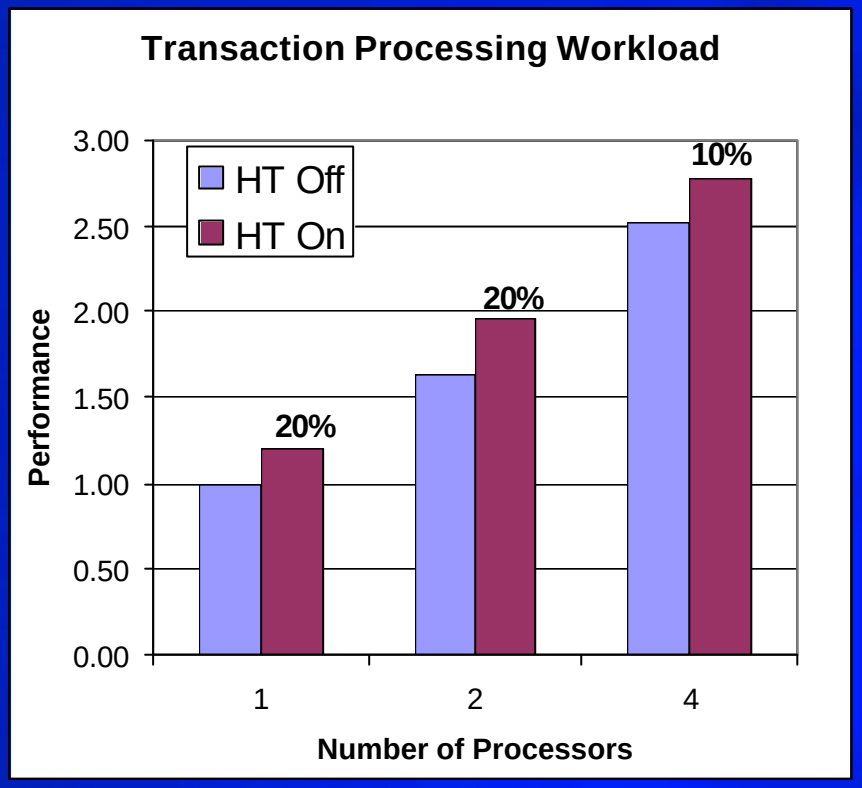


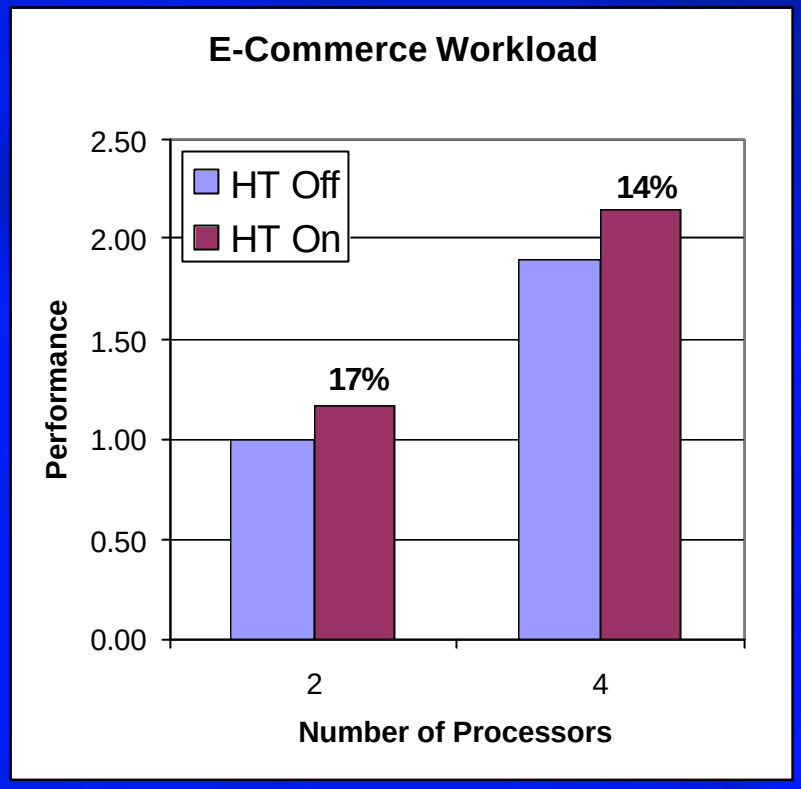
# Execution Pipeline




**Partition queues between major pipestages of pipeline**

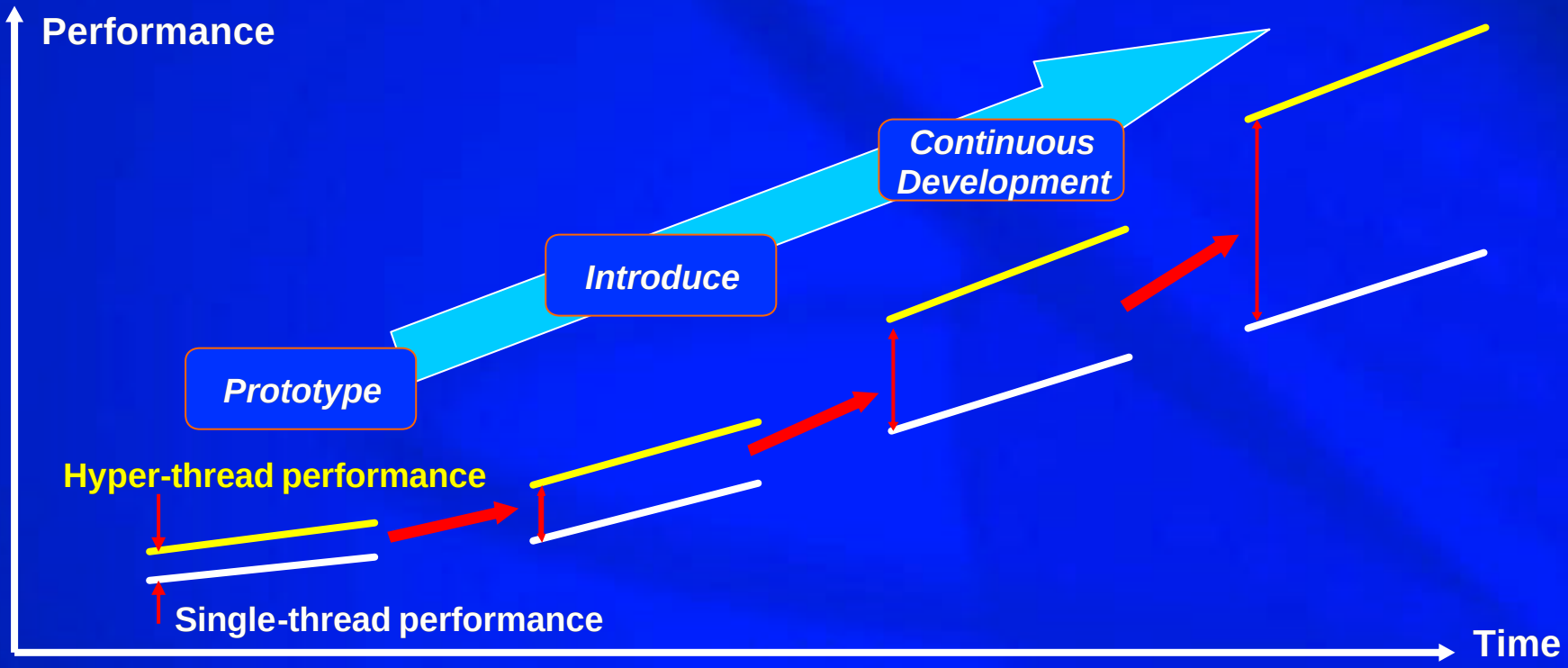
# Server Performance




**Good performance benefit from small die area investment**

# Intel's Long-Term Hyper-Threading Strategy




**Hyper-Threading Technology expected to deliver increasingly higher performance**

intel

Microprocessor Research Forum 2002

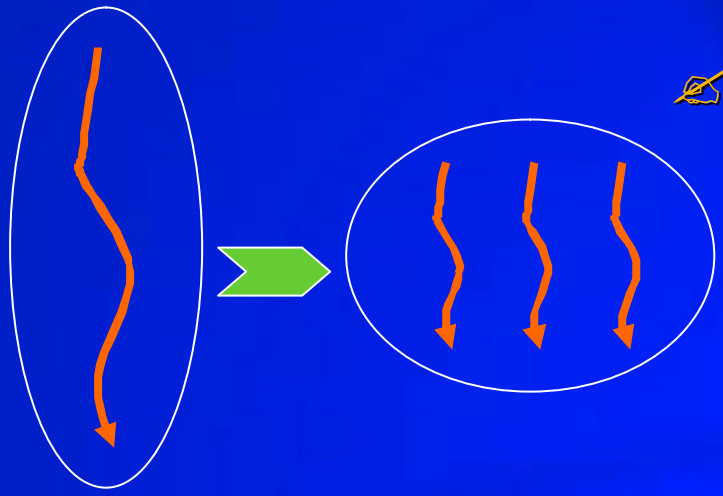
# Challenges for Multithreading

- Multithreaded Applications
  - Multithread Programming
  - Automatic Thread Partitioning
- Design Complexity
  - Additional Validation Overhead
  - Increasing Scalar Inefficiency
  - SMT vs. CMP Tradeoffs ?
- Latency of Single-Threaded Applications

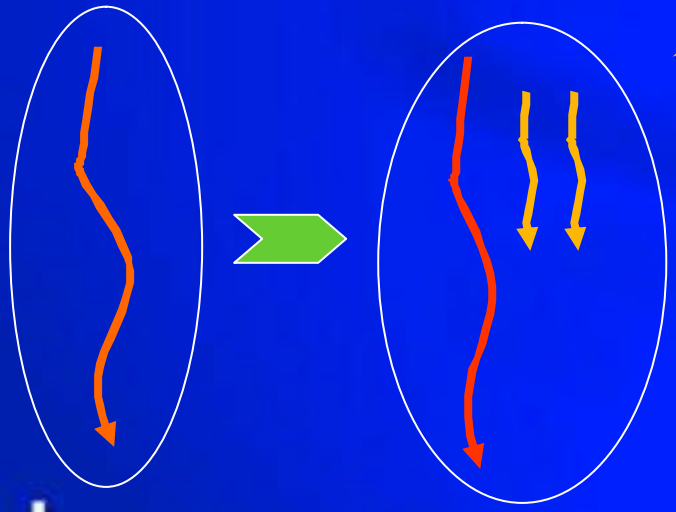
# Multi-Threaded Processors

- Targeting:
  - Throughput of Multi-tasking Workloads
  - Latency of Multi-threaded Applications
- Not Targeting:
  - Latency of Single-threaded Applications
- Research Challenge:
  - Leverage Multi-threaded CPU to Improve Latency of Single-threaded Applications

# Asymmetric Multi-Threading



- **Symmetric Multi-Threading**
  - Partition single thread into multiple threads
  - Achieve performance by parallel execution of multiple threads

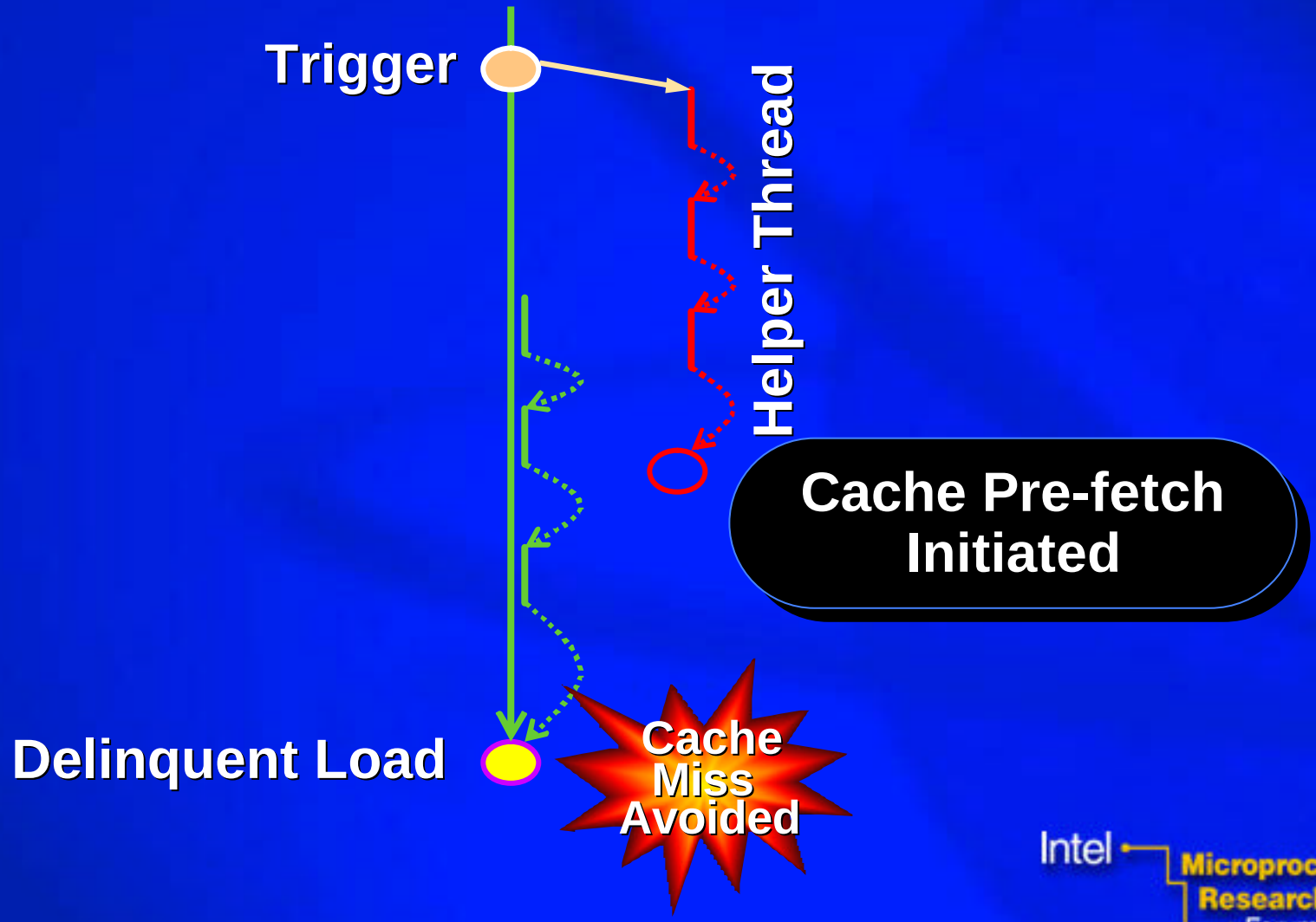


- **Asymmetric Multi-Threading**
  - Attach helper threads to original single-threaded code
  - Achieve speedup via memory prefetching by helper threads

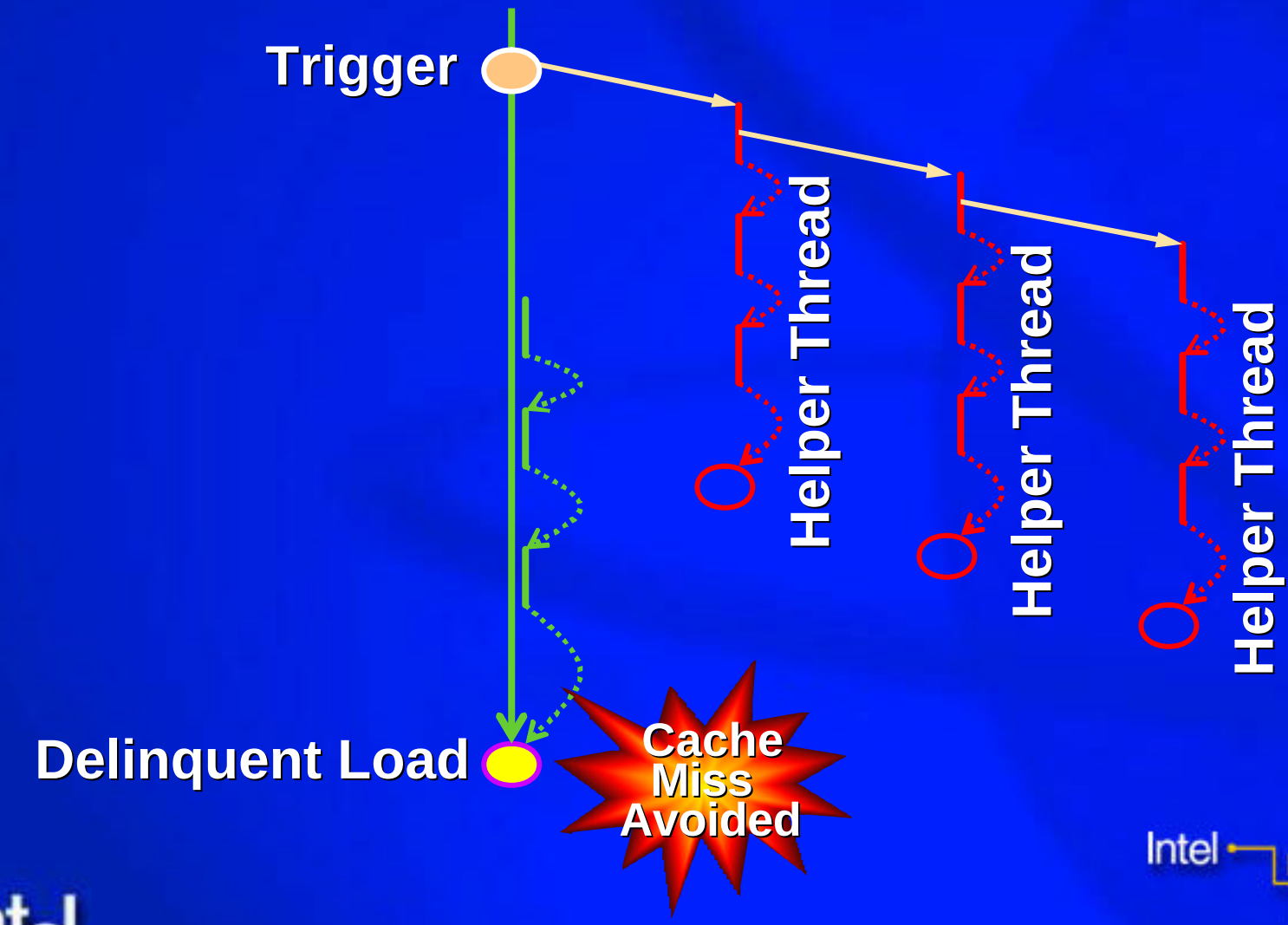
# Asymmetric Helper Threads

- Symmetric Multi-Threading Compiler
  - Partition single thread into multiple threads
  - Must ensure semantic correctness
  - Difficult for common and legacy code
- Asymmetric Multi-Threading Compiler
  - Attach helper threads to original code
  - Leverage side effect of helper threads
  - Can be dynamically invoked/controlled

# Pre-fetch via Helper Threads



# Chaining Triggers of Helper Threads



# Chaining Trigger Advantages

- Low-cost Thread Spawning:
  - Chaining triggers initiate helper threads without impacting main thread performance
- Long-range Prefetching:
  - Can target delinquent loads far ahead of the main thread
  - Helper threads make progress independent of main thread's lack of progress

# Prefetching Effectiveness of Helper Threads on HT Machine

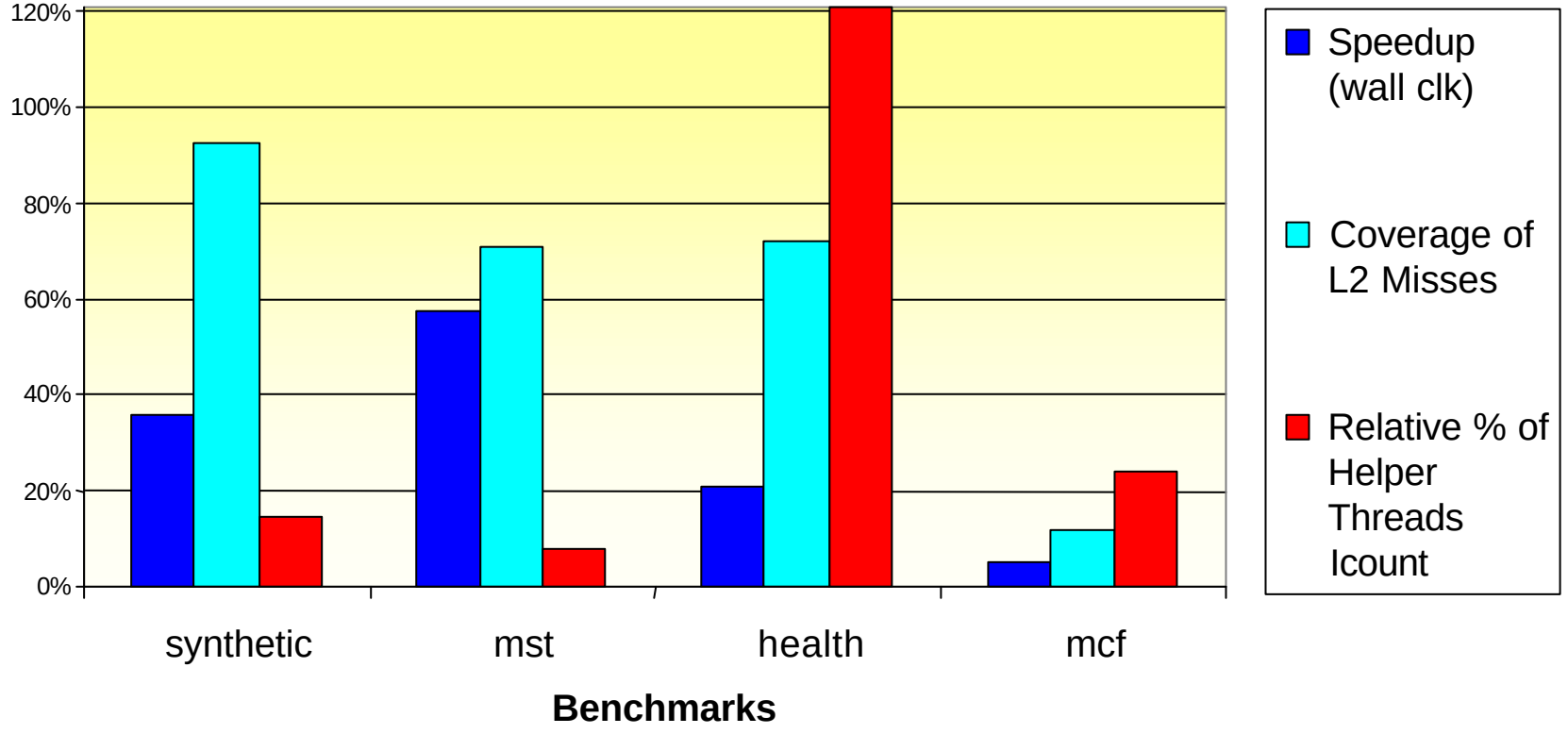
| Benchmark | Description | Speedup |
|---|---|---|
| ***Synthetic*** | Graph traversal in large random graph simulating large database retrieval | **22% - 45%** |
| ***MST*** (Olden) | Minimal Spanning Tree algorithm used for Data Clustering | **23% - 40%** |
| ***Health*** (Olden) | Hierarchical database modeling health care system | **11% - 24%** |
| ***MCF*** (SPECint) | Integer programming algorithm used for bus scheduling | 7.08% |

Intel Microprocessor Research

**Source: Intel Labs**

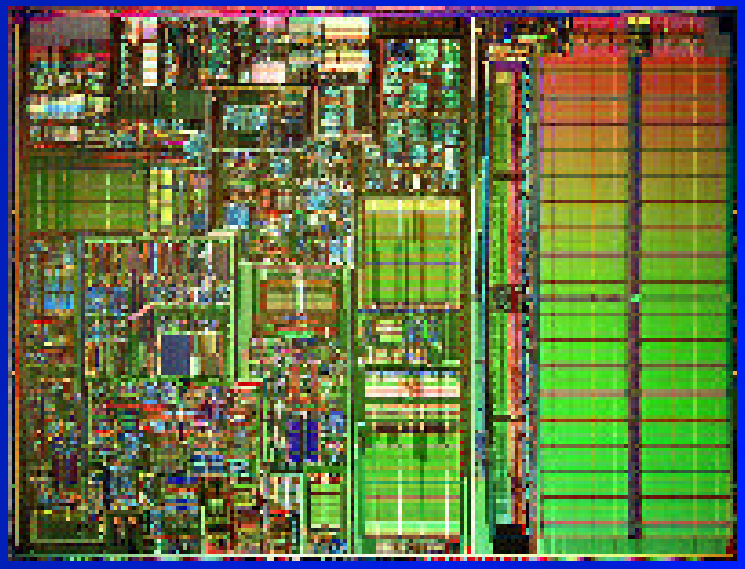
# Helper Threads on Hyper-Threading Machine

**Asymmetric MT effectiveness:**
**speedup, L2 miss coverage; % helper thread Icount**



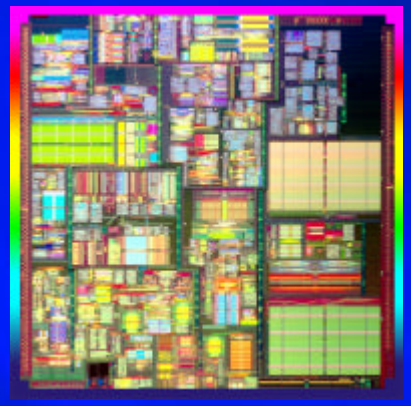

Source: Intel Labs

# Intel® Processors with Netburst™ Microarchitecture



**Intel® Xeon™ MP Processor**
256KB 2nd-Level Cache
1MB 3rd-Level Cache
.18u process

**Intel Xeon Processor**
256KB 2nd-Level Cache
.18u process



**Intel Xeon Processor**
512KB 2nd-Level Cache
.13u process

intel

Intel Microprocessor Research Forum 2002