The Intel® Pentium® 4 Processor

Doug Carmean
Sr. Principal Architect
Intel Architecture Group

April 26, 2005
Cornell

Intel®
PDX

The Intel® Pentium® 4 Processor

Doug Carmean
Sr. Principal Architect
Intel Architecture Group

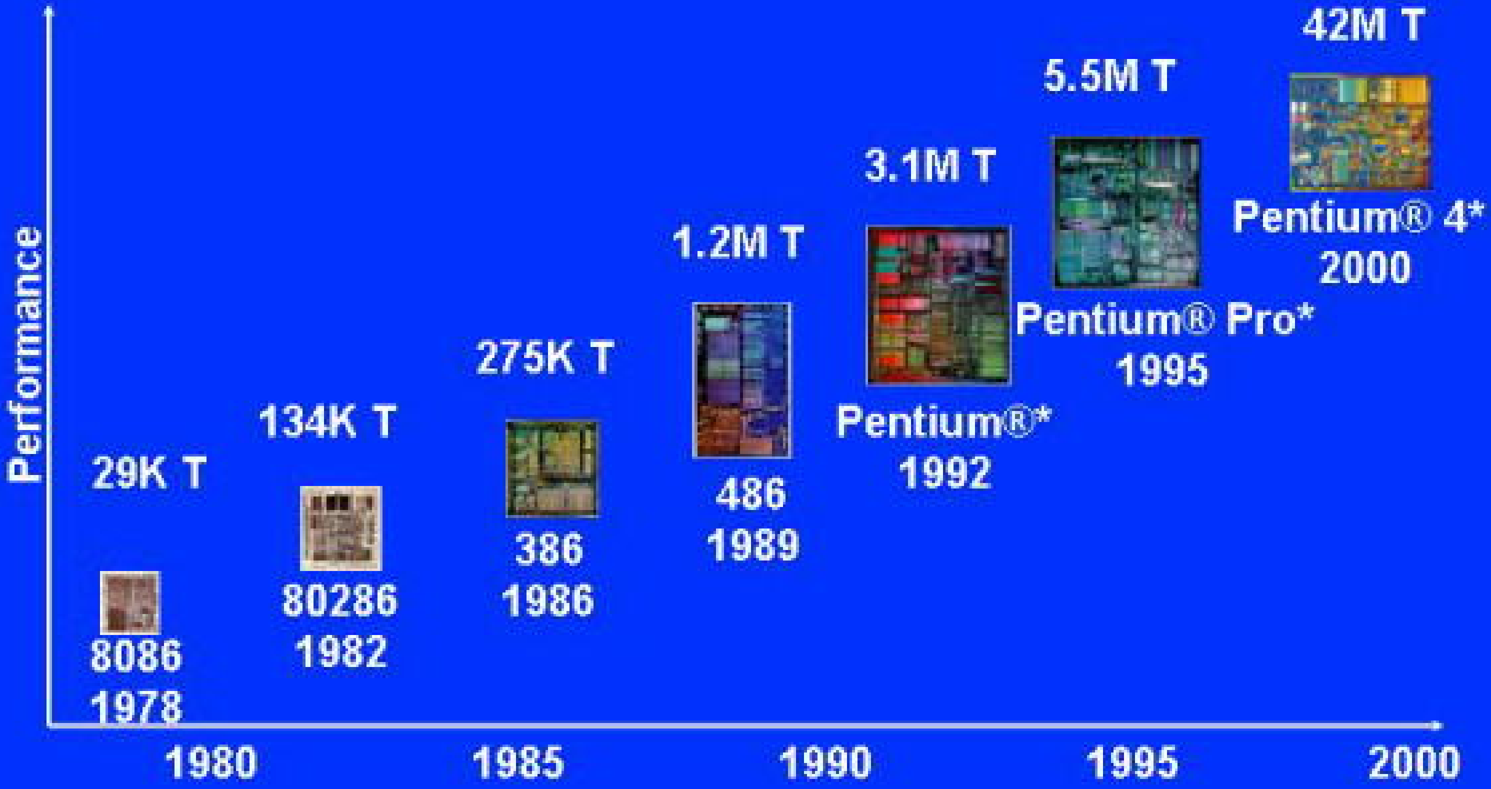
April 26, 2005
Cornell

Intel®
PDX

Agenda

- Review
- Pipeline Comparison
- Frequency Scaling
- Microarchitecture Description
- Analyzing Performance
- A Retrospective
- Summary

History



Intel®
PDX

Intel® Netburst™ Micro-architecture vs P6

Basic P6 Pipeline

1	2	3	4	5	6	7		10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sc	Exec

Intro at
733MHz
.18µ

Basic Pentium® 4 Processor Pip

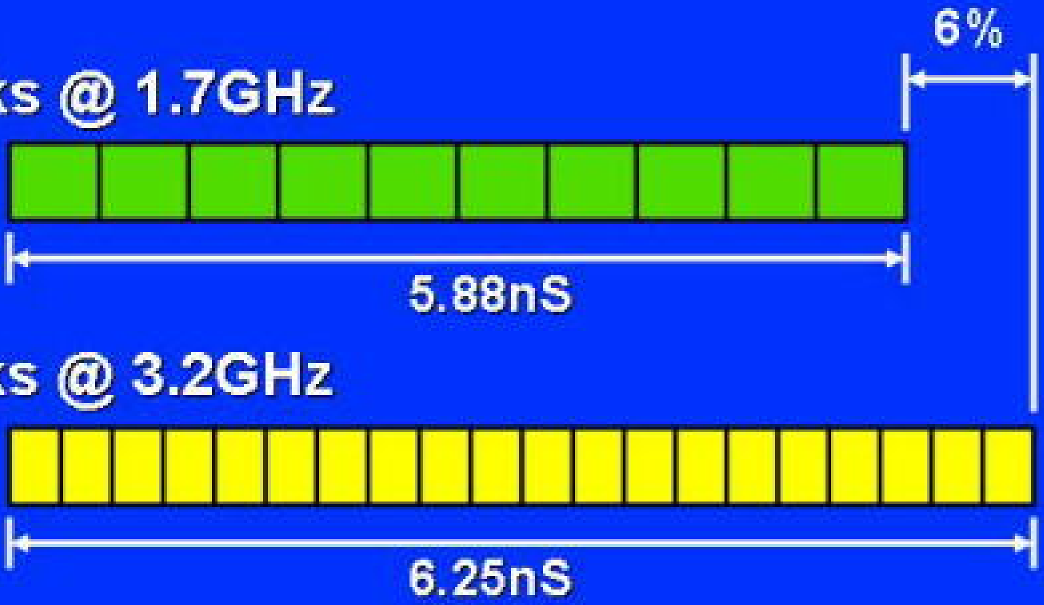
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TC Nxt IP		TC Fetch		Drive	Alloc	Rename		Que	Sch	Sch	Sch	Disp	Disp	R

Intro at
1.5GHz
.18µ

Deeper Pipelines enable higher frequency and performance

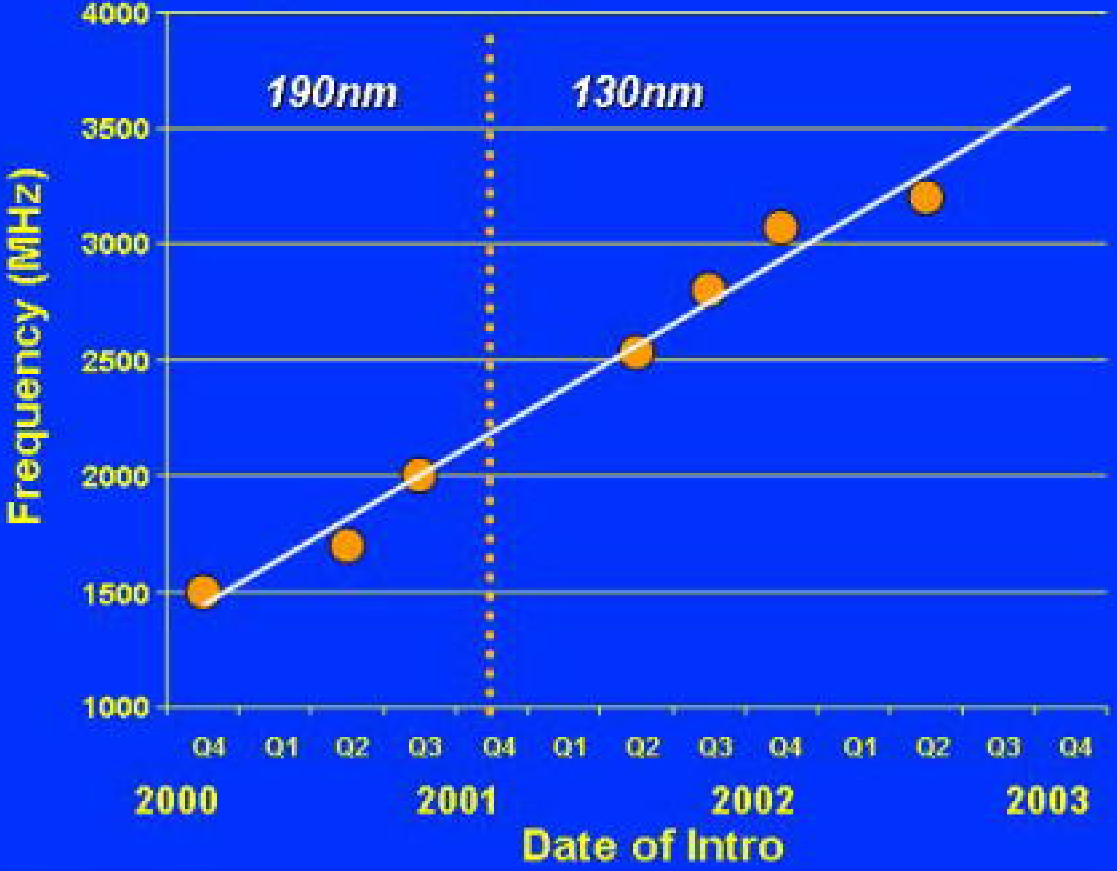
Pipeline Comparison

- P6:
  - 10 clocks @ 1.7GHz
- P4P:
  - 20 clocks @ 3.2GHz



Branch misprediction penalty remains similar

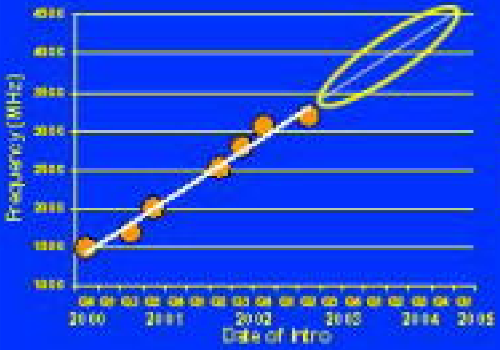
Frequency Scaling



~170MHz
per
Quarter

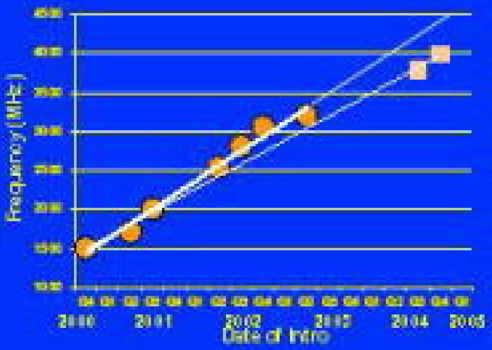
Solid Frequency Scaling Through 2003 (130nm process)

Frequency Scaling



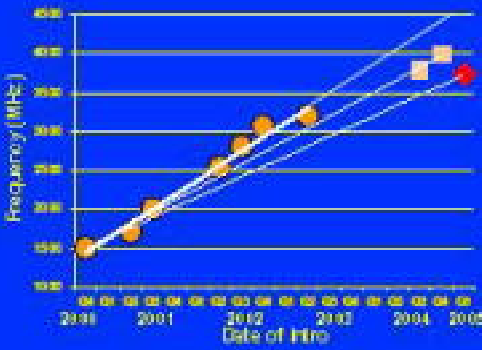
Original Ramp Rate

- ***170MHz per Q***
- ***>4.5GHz in 2005***
- ***Circuit Tweaking***
- ***Process Polishing***



Revised in 2003

- ***150MHz per Q***
- ***4GHz in 2005***
- ***Design Sched Slip***

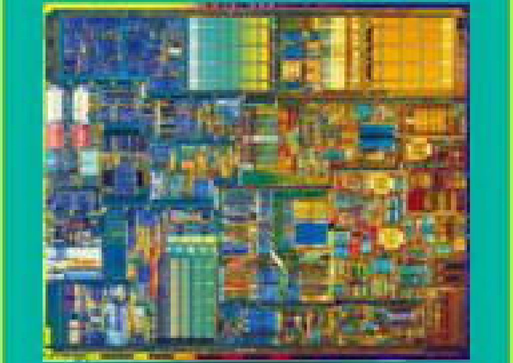


Revised in 2004

- ***130MHz per Q***
- ***3.7GHz in 2005***
- ***More Sched Slip***

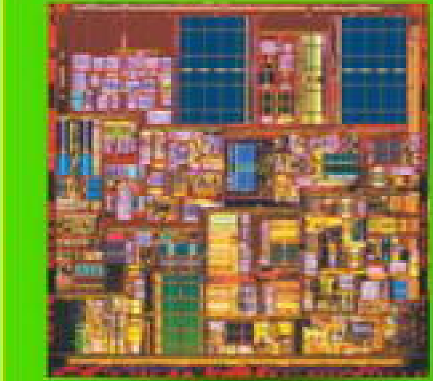
Schedule Slips Result in Significantly Lower Performance

The Family Tree



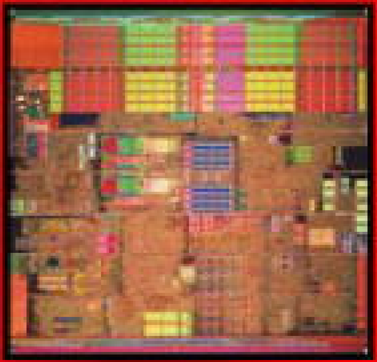
Willamette

- 190nm
- 8KB L1
- 256KB L2
- HT Capable



Northwood

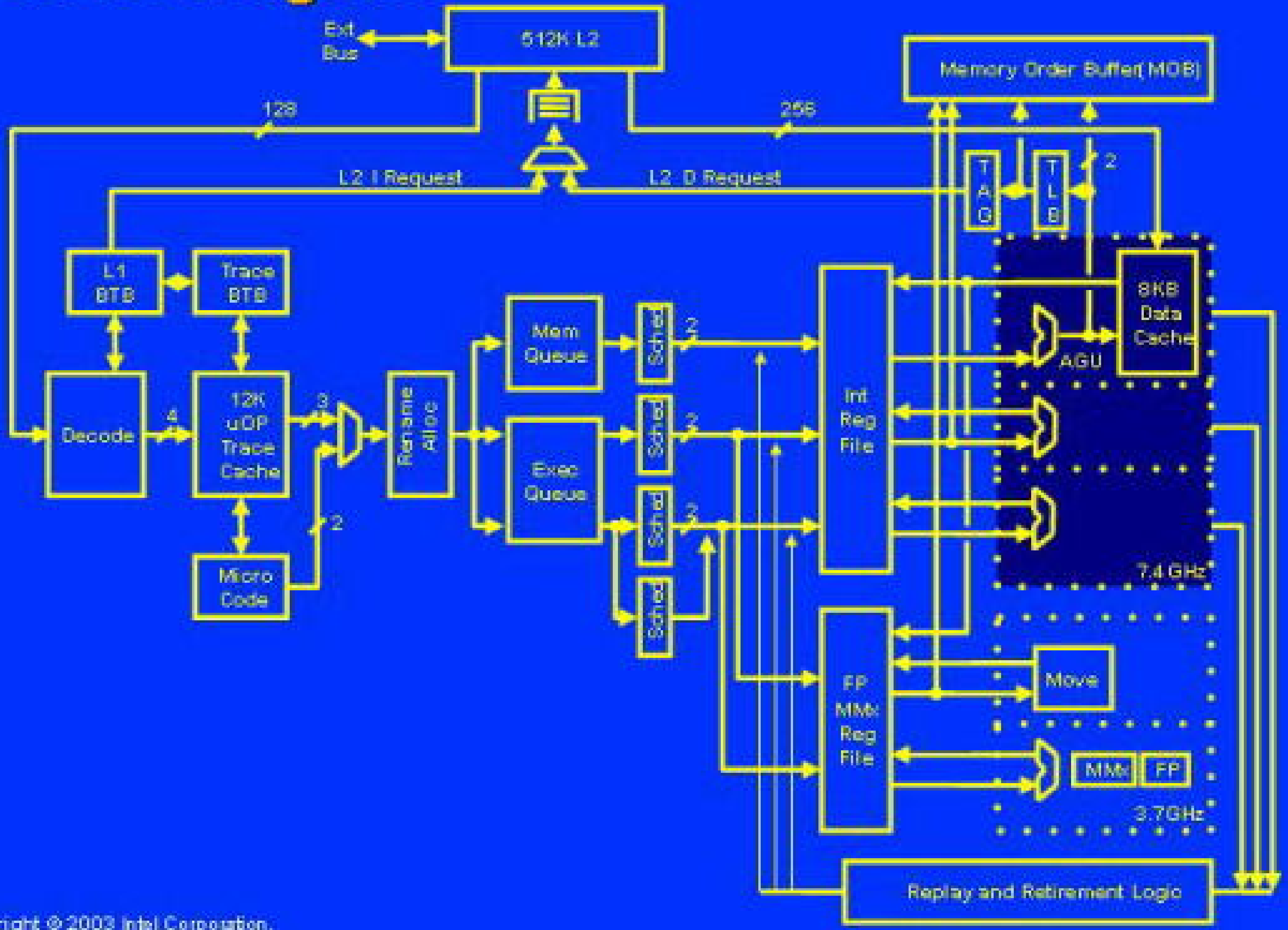
- 130nm
- 8KB L1
- 512KB L2
- HT Production
- Minor perf fixes



Prescott

- 90nm
- 16KB L1
- 1MB L2
- 64-bit
- Shift/Mul fixes
- Thread instrs

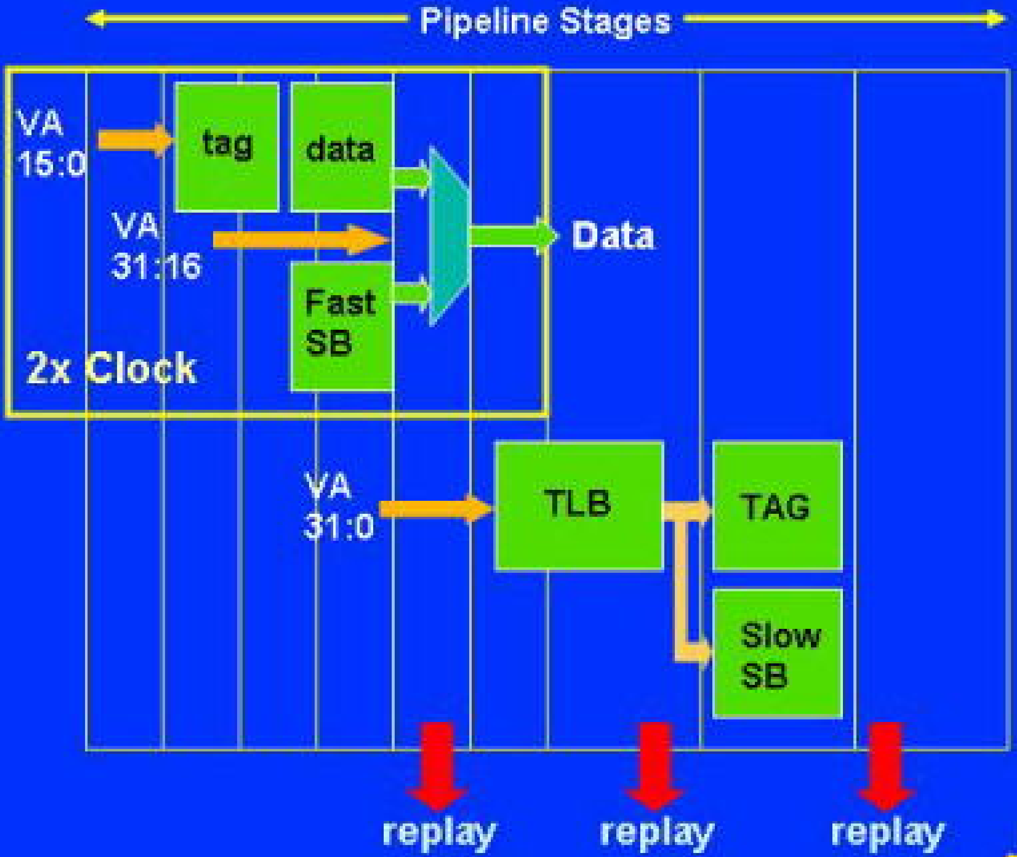
Block Diagram



Agenda

- Review
- Pipeline Comparison
- Frequency Scaling
- Microarchitecture Description
- Analyzing Performance
- A Retrospective
- Summary

L1 Data Cache



Intel®
PDX

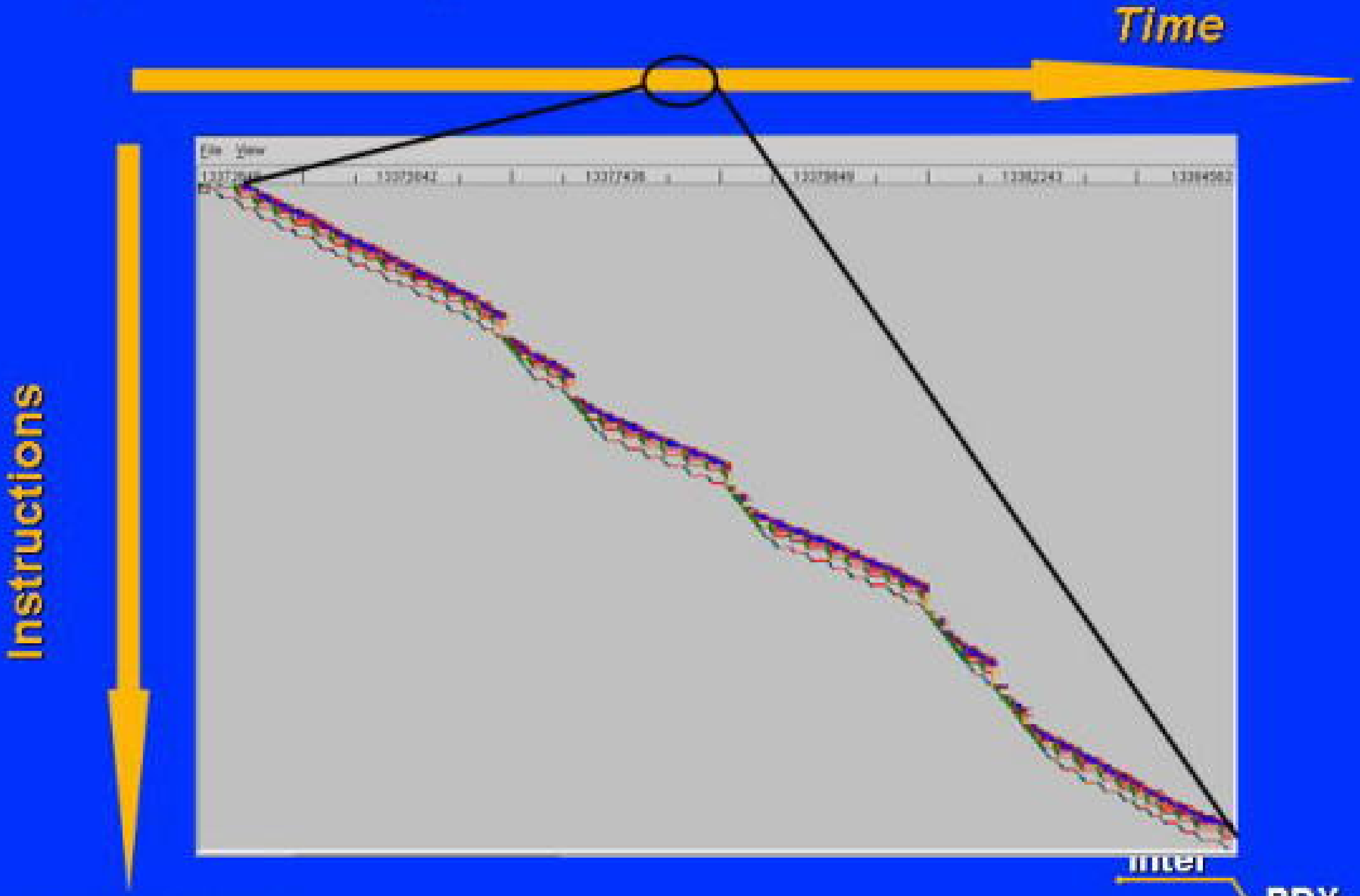
Agenda

- Review
- Pipeline Comparison
- Frequency Scaling
- Microarchitecture Description
- Analyzing Performance
- A Retrospective
- Summary

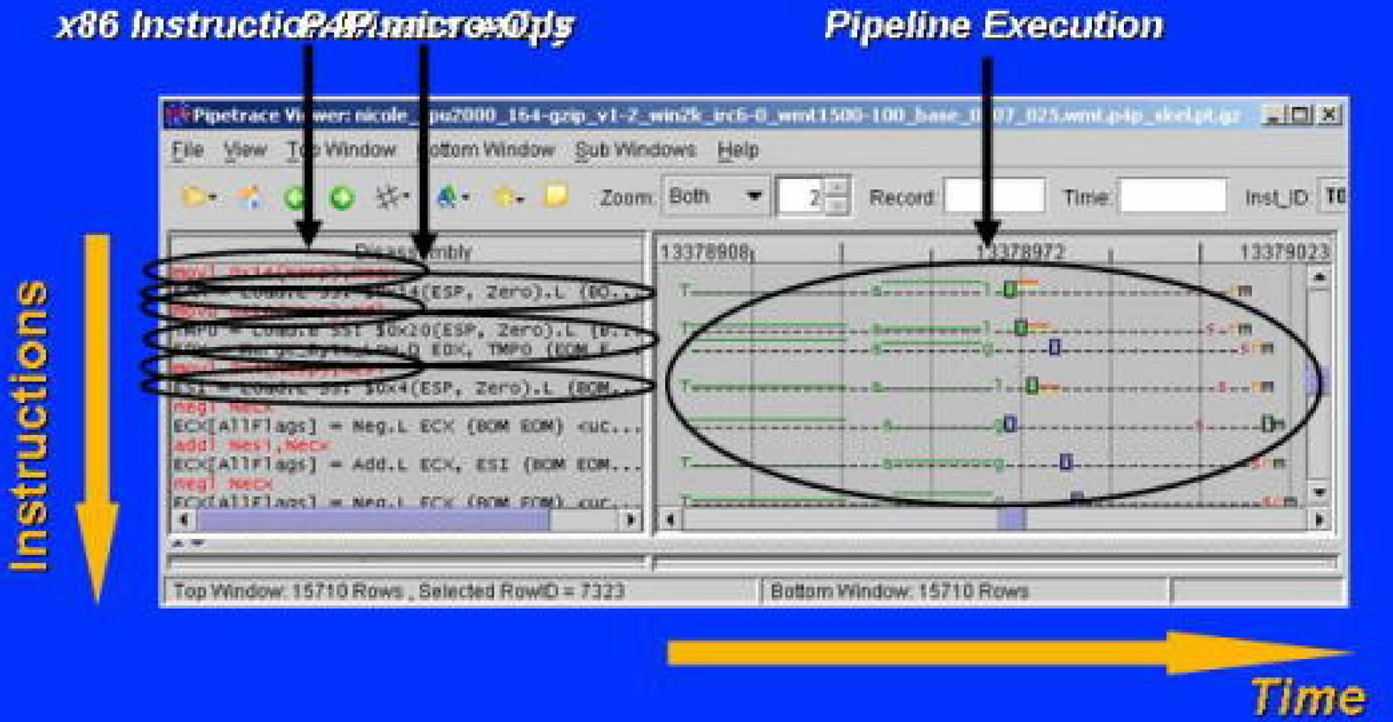
Analyzing Performance

- Performance simulator
  - Models low level microarchitecture details
  - Correlated to silicon
  - Tons of knobs (parameters) for various configurations
- Application "Traces"
  - Architectural state plus memory image
  - Multiple 30M instruction traces per application
  - A typical study runs on ~500 traces
  - An important study will run on ~1000 traces
- Pipeline events
  - Generates pipeline events (Fetch, Decode, ….)
  - Events are consistent between RTL & perf. Simulator
  - Graphical tool (ptv) used to analyze traces

Pipetrace: High Level View



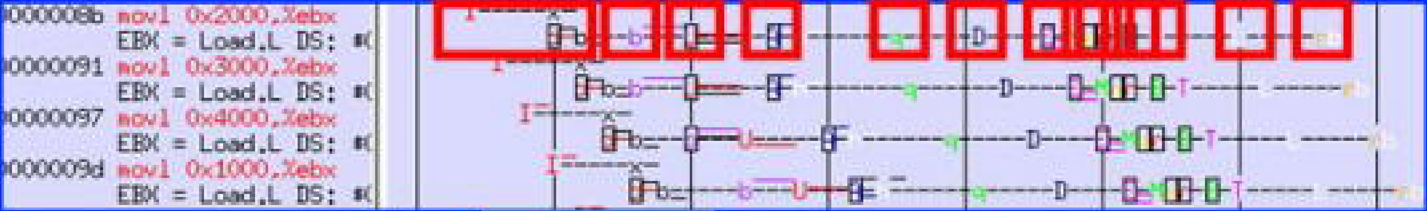
Pipetrace: uOP View Window


x86 Instructions
P4P micro-Ops
Pipeline Execution
Pipetrace Viewer: nicole_ pu2000_164-gzip_v1-2_win2k_icc6-0_wml1500-100_base_0 07_025.wml.p4p_skel.pt.gz
File
View
Top Window
Bottom Window
Sub Windows
Help
Zoom:
Both
2
Record:
Time:
Inst_ID:
Disassembly
13378908
13378972
13379023
Top Window: 15710 Rows , Selected RowID = 7323
Bottom Window: 15710 Rows
Instructions
Time

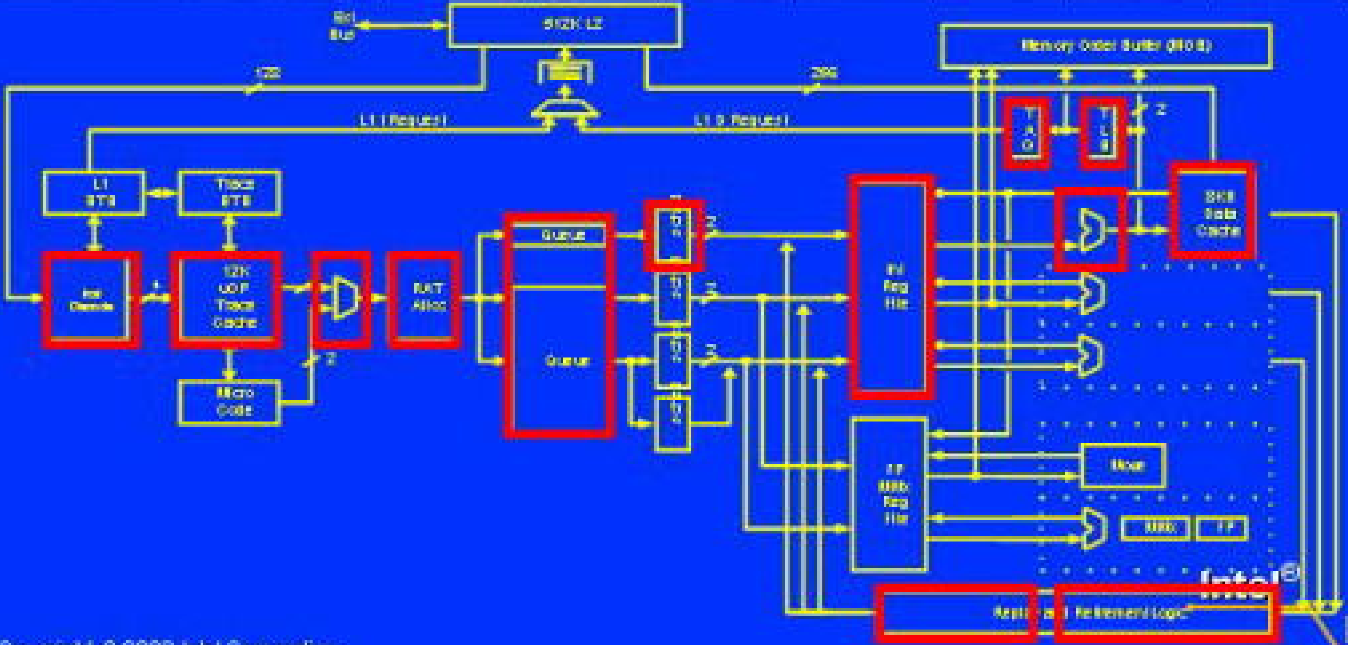
```
movl [illegible]
[illegible] = LoadLE SS: $0x14(ESP, Zero).L (BO...
movl [illegible]
TMP0 = Load.E SS: $0x20(ESP, Zero).L (B...
[illegible] Merge_Byte[illegible] EDX, TMP0 (BOM E...
movl [illegible]
ESI = LoadLE SS: $0x4(ESP, Zero).L (BOM...
negl %ecx
ECX[AllFlags] = Neg.L ECX (BOM EOM) <uc...
addl %esi,%ecx
ECX[AllFlags] = Add.L ECX, ESI (BOM EOM...
negl %ecx
ECX[AllFlags] = Neg.L ECX (BOM EOM) <uc...
```

Intel® PDX

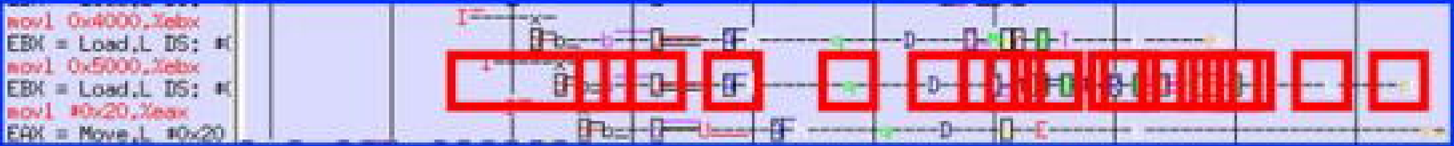
Simple Example



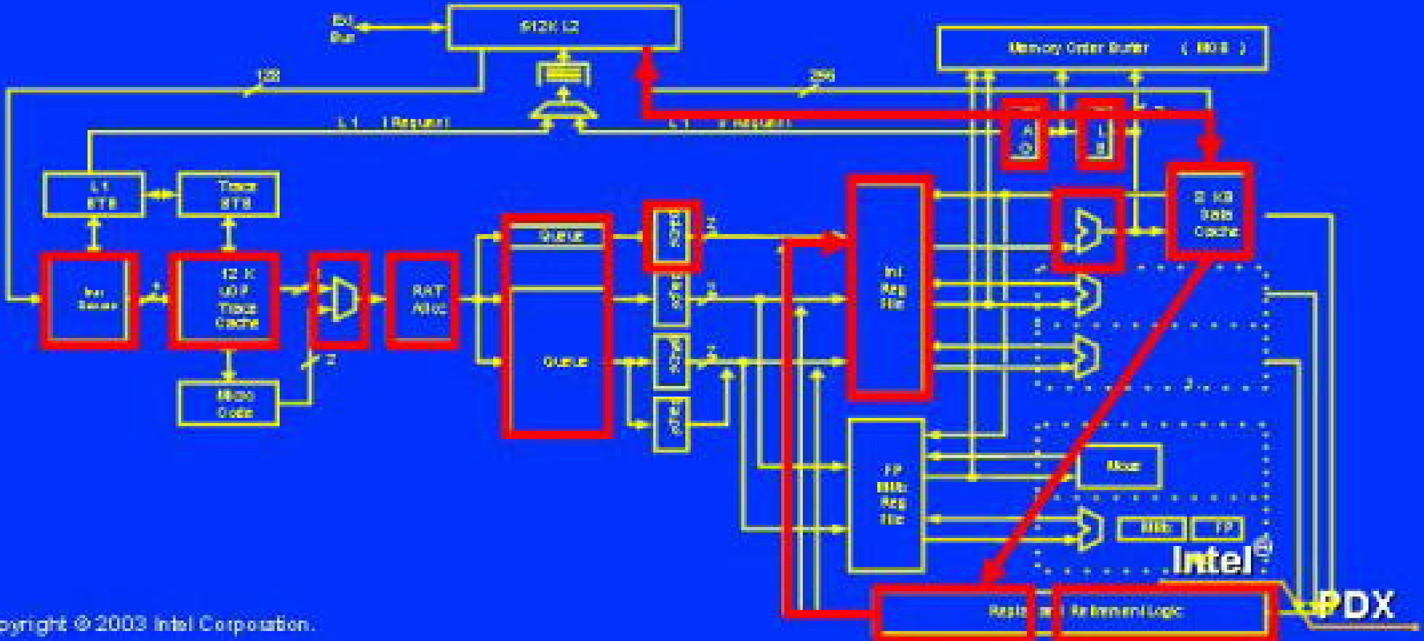
```
0000008b movl 0x2000,%ebx
         EBX = Load.L DS: #(
00000091 movl 0x3000,%ebx
         EBX = Load.L DS: #(
00000097 movl 0x4000,%ebx
         EBX = Load.L DS: #(
0000009d movl 0x1000,%ebx
         EBX = Load.L DS: #(
```



Replay loop



```
movl 0x4000,%ebx
EBX = Load.L DS: #(
movl 0x5000,%ebx
EBX = Load.L DS: #(
movl #0x20,%eax
EAX = Move.L #0x20
```



Copyright © 2003 Intel Corporation.

A Well Behaved Application

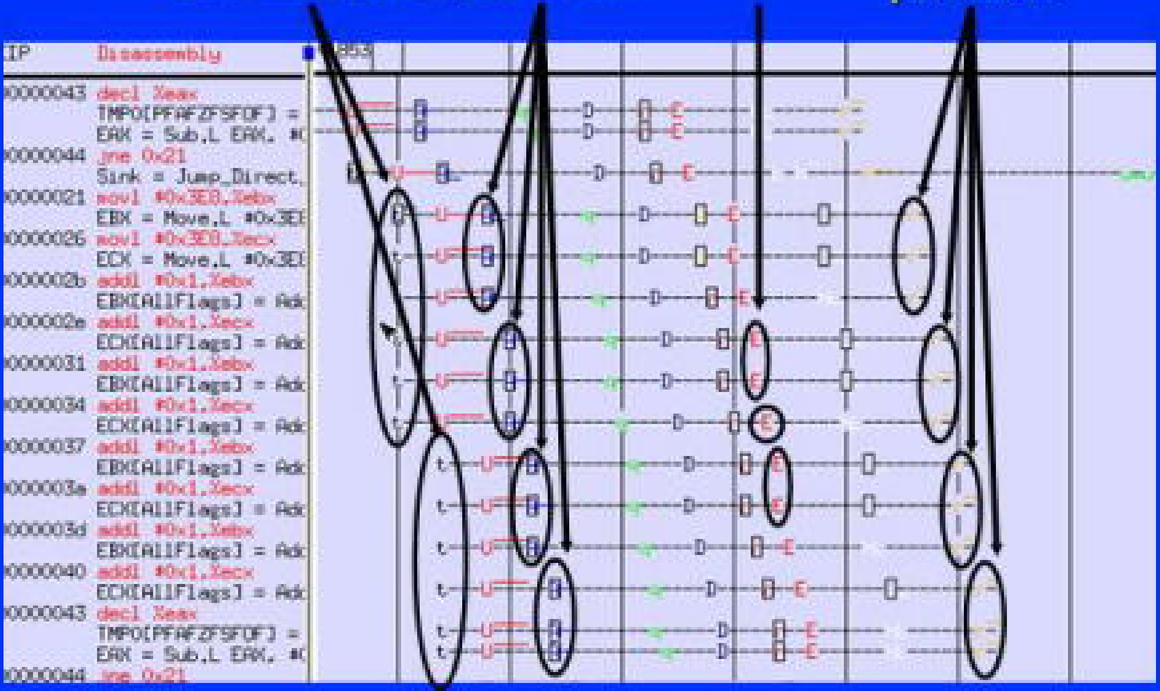
TC Hitting

Max Bandwidth per clock

Allocating 3 uOPs

Good Execution

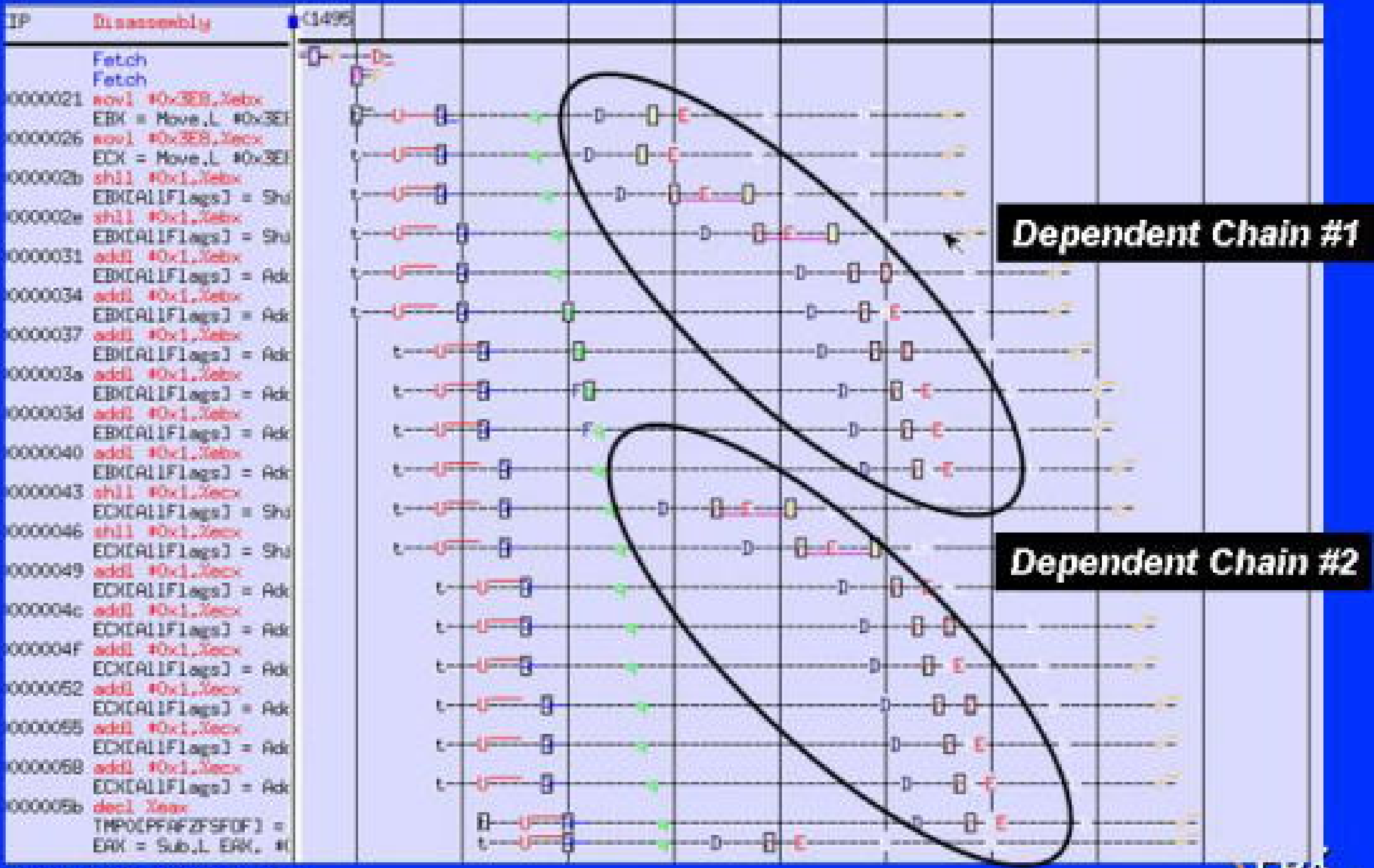
Retiring 3 uOPs per clock



```
IP        Disassembly
00000043 decl %eax
         TMPO[PFAFZFSFOF] =
         EAX = Sub.L EAX, #(
00000044 jne 0x21
         Sink = Jump_Direct
00000021 movl #0x3E8,%ebx
         EBX = Move.L #0x3E
00000026 movl #0x3E8,%ecx
         ECX = Move.L #0x3E
0000002b addl #0x1,%ebx
         EBX[AllFlags] = Ad
0000002e addl #0x1,%ecx
         ECX[AllFlags] = Ad
00000031 addl #0x1,%ebx
         EBX[AllFlags] = Ad
00000034 addl #0x1,%ecx
         ECX[AllFlags] = Ad
00000037 addl #0x1,%ebx
         EBX[AllFlags] = Ad
0000003a addl #0x1,%ecx
         ECX[AllFlags] = Ad
0000003d addl #0x1,%ebx
         EBX[AllFlags] = Ad
00000040 addl #0x1,%ecx
         ECX[AllFlags] = Ad
00000043 decl %eax
         TMPO[PFAFZFSFOF] =
         EAX = Sub.L EAX, #(
00000044 jne 0x21
```

Intel® PDX

Parallelism with dependencies



```
IP        Disassembly
          Fetch
          Fetch
0000021   movl  #0x3E8,%ebx
          EBX = Move.L  #0x3E
0000026   movl  #0x3E8,%ecx
          ECX = Move.L  #0x3E
000002b   shll  #0x1,%ebx
          EBX[AllFlags] = Sh
000002e   shll  #0x1,%ebx
          EBX[AllFlags] = Sh
0000031   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
0000034   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
0000037   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
000003a   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
000003d   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
0000040   addl  #0x1,%ebx
          EBX[AllFlags] = Ad
0000043   shll  #0x1,%ecx
          ECX[AllFlags] = Sh
0000046   shll  #0x1,%ecx
          ECX[AllFlags] = Sh
0000049   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
000004c   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
000004F   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
0000052   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
0000055   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
0000058   addl  #0x1,%ecx
          ECX[AllFlags] = Ad
000005b   decl  %eax
          TMP0[PFAFZFSFOF] =
          EAX = Sub.L EAX, #
```

Agenda

- Review
- Pipeline Comparison
- Frequency Scaling
- Microarchitecture Description
- Analyzing Performance
- A Retrospective
- Summary

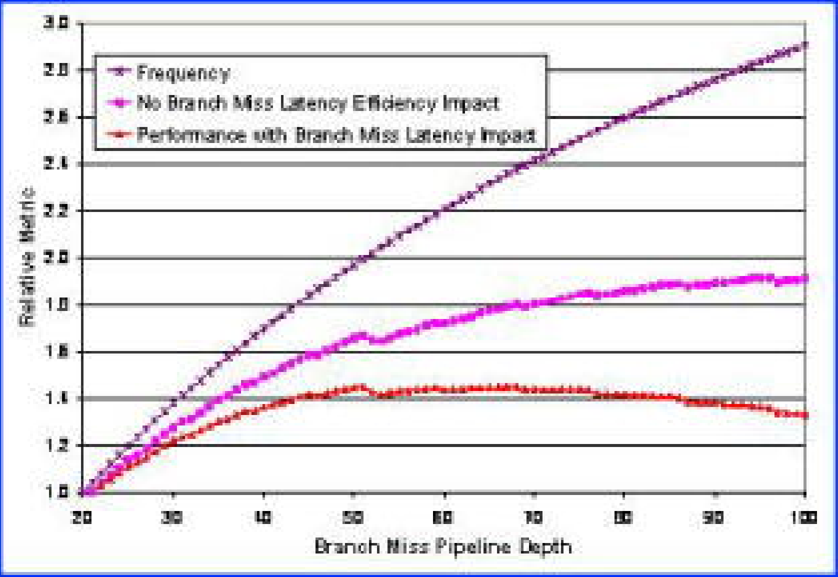
What *Were* We Thinking?

- Need to improve traditional performance
- Start working on threaded performance
- The P6 family is out of gas
- Performance ~ Frequency
- Frequency Sells
- We can pipeline *anything*

Improve Performance: Push IPC, Push Frequency,
Invent the *new uArch for a family of processors,*
Start down the SMT/CMP path

Increasing Performance by Implementing Deeper Pipelines and Larger Caches*

- Performance vs. pipeline depth
  - Cycle time = overhead time + useful time
  - Overhead time (latch + skew + jitter) is constant vs. pipeline depth
  - Useful time can be sliced up arbitrarily and is constant vs. pipeline depth
  - Use a simulator to calculate IPC vs. pipeline depth
  - The impact of increasing multiple pipelines is the product of impact of individual pipeline increases
  - Performance = Freq * IPC

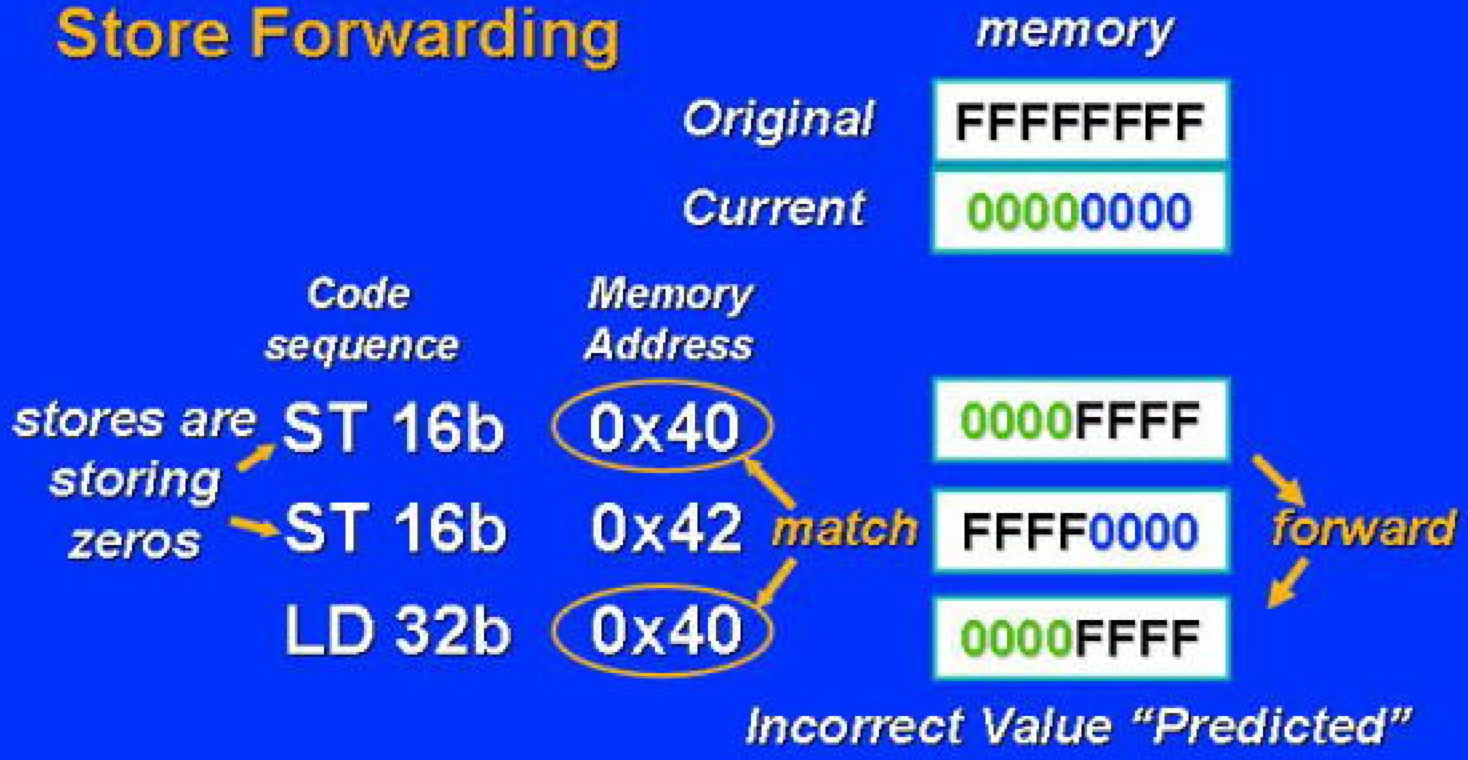


*Eric Sprangle, Doug Carmean, ISCA 2002

But. . . as Pipelines get Deeper, Algorithms Change Fundamentally

- Goal: Hold critical latencies to a constant number of *cycles* as frequency increases
  - ALUs become staggered (16-bits per clock)
    - P4P ALUs are ½ cycle and running 2x frequency
    - Non critical functions moved - shifts take longer
  - Scheduler algorithms completely change
  - L1 data cache becomes smaller, virtually indexed
  - Data consumed on partial address matches
    - L1 data cache uses way predictors
    - Store forwarding uses subset of virtual addresses

Store Forwarding



memory

Original FFFFFFFF

Current 00000000

Code sequence	Memory Address	
ST 16b	0x40	0000FFFF
ST 16b	0x42	FFFF0000
LD 32b	0x40	0000FFFF

stores are storing zeros

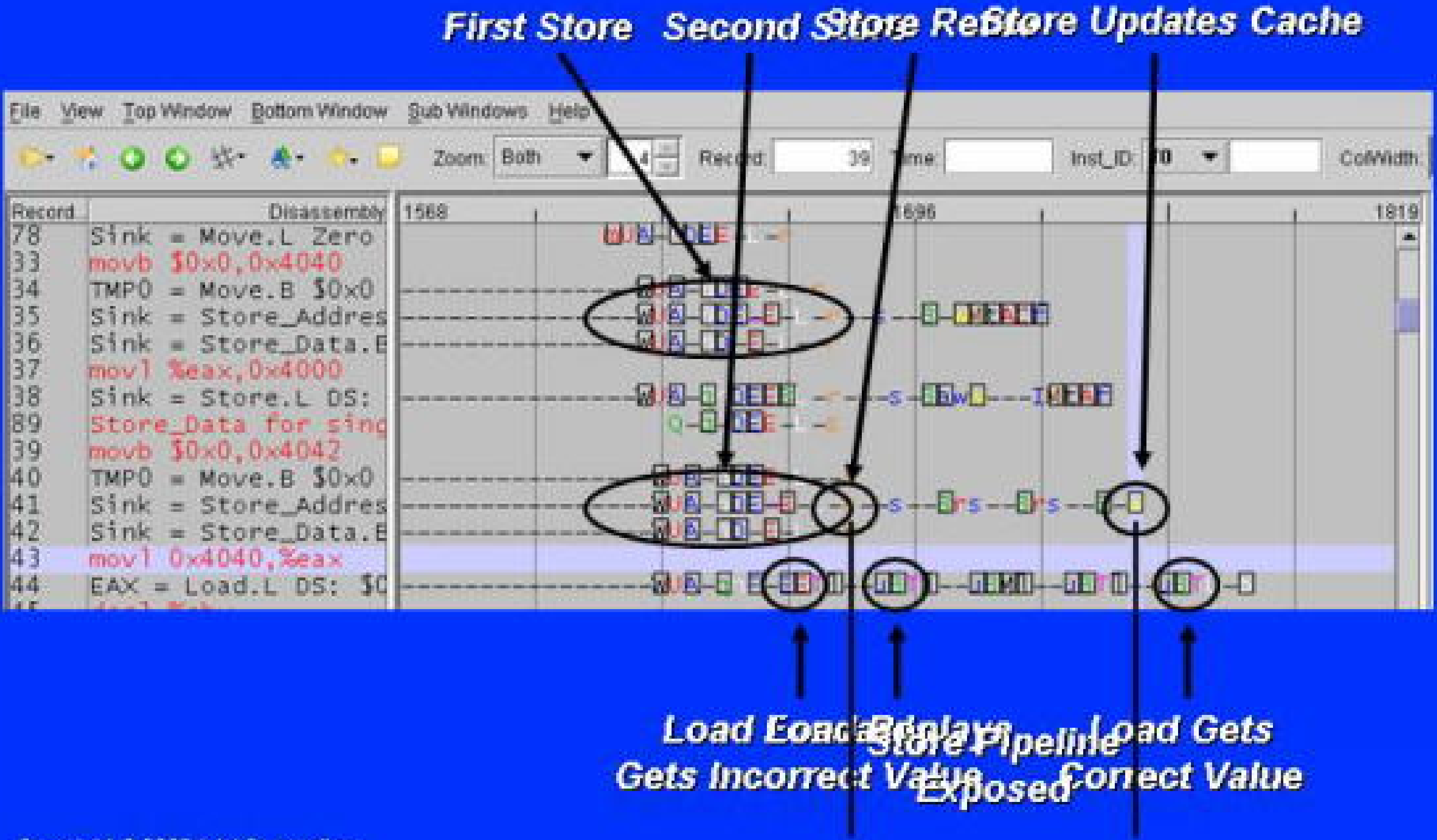
match

forward

Incorrect Value “Predicted”

Both stores have to commit the data to the cache before the load gets the right data

Store forwarding



```
Record  Disassembly
78      Sink = Move.L Zero
33      movb $0x0,0x4040
34      TMP0 = Move.B $0x0
35      Sink = Store_Addres
36      Sink = Store_Data.B
37      movl %eax,0x4000
38      Sink = Store.L DS:
89      Store_Data for sing
39      movb $0x0,0x4042
40      TMP0 = Move.B $0x0
41      Sink = Store_Addres
42      Sink = Store_Data.B
43      movl 0x4040,%eax
44      EAX = Load.L DS: $0
```

First Store

Second Store

Store Restore

Store Updates Cache

Load Gets Incorrect Value

Load Replays

Store Pipeline Exposed

Load Gets Correct Value

Pipeline Evolution

Intel® PDX

Debunking Deeper Pipelines

- Deeper pipelines enable higher frequency, yielding higher performance
- But, additional algorithms introduce complexity which reduce potential performance gain
- But, engineering effort ~~can~~ *should(?)* overcome challenges represented by complexity increases

Summary

- The Pentium® 4 Processor's deep pipelines provide high performance by enabling high frequency
- Introduced with, and maintained a 2x frequency advantage over the P6 pipeline
- Innovative algorithms introduced significant complexity, requiring huge effort to tune
- Evolution vs. Revolution – Evolution won this chapter