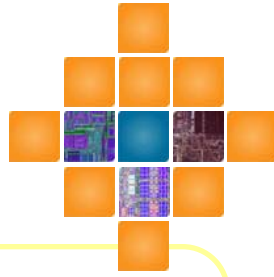


CS6290

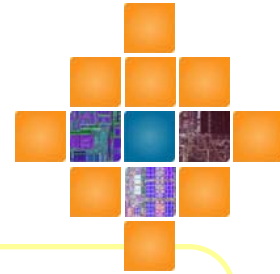
Pentiums



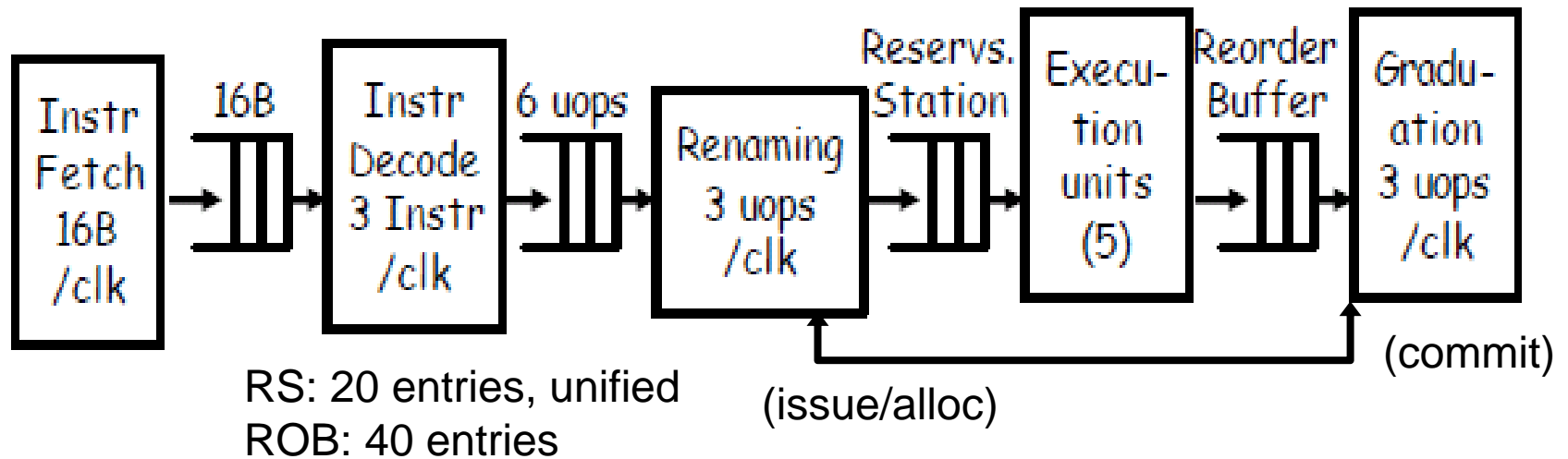
Case Study1 : Pentium-Pro



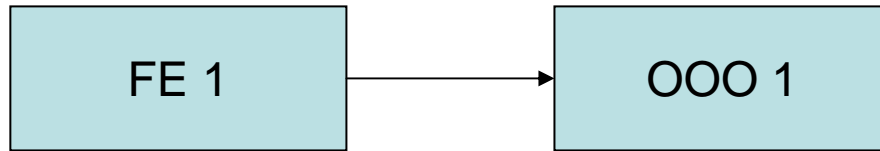
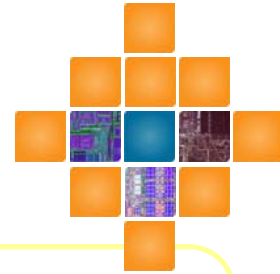
- Basis for Centrinos, Core, Core 2
- (We'll also look at P4 after this.)



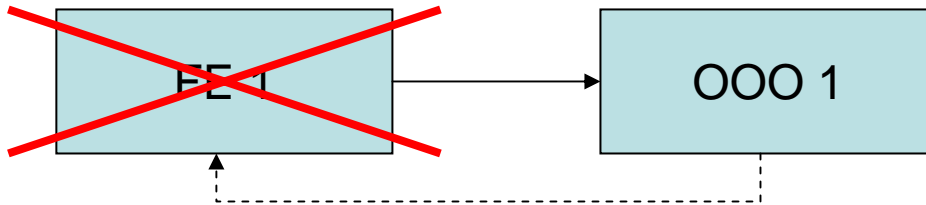
Hardware Overview



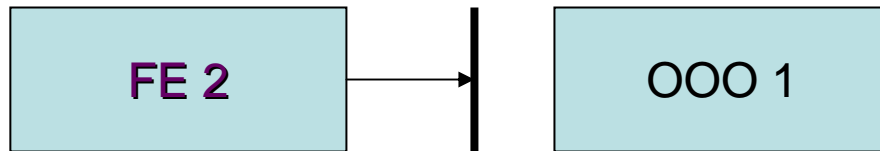
Speculative Execution & Recovery



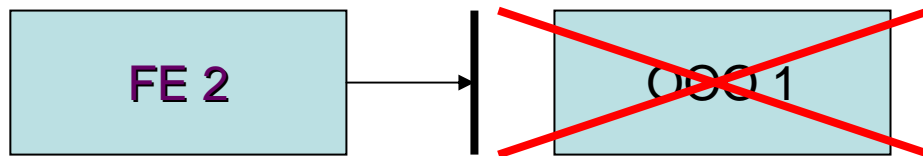
Normal execution: speculatively fetch and execute instructions



OOO core detects misprediction, flush FE and start refetching



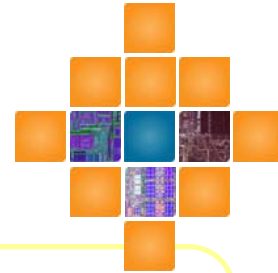
New insts fetched, but OOO core still contains wrong-path uops



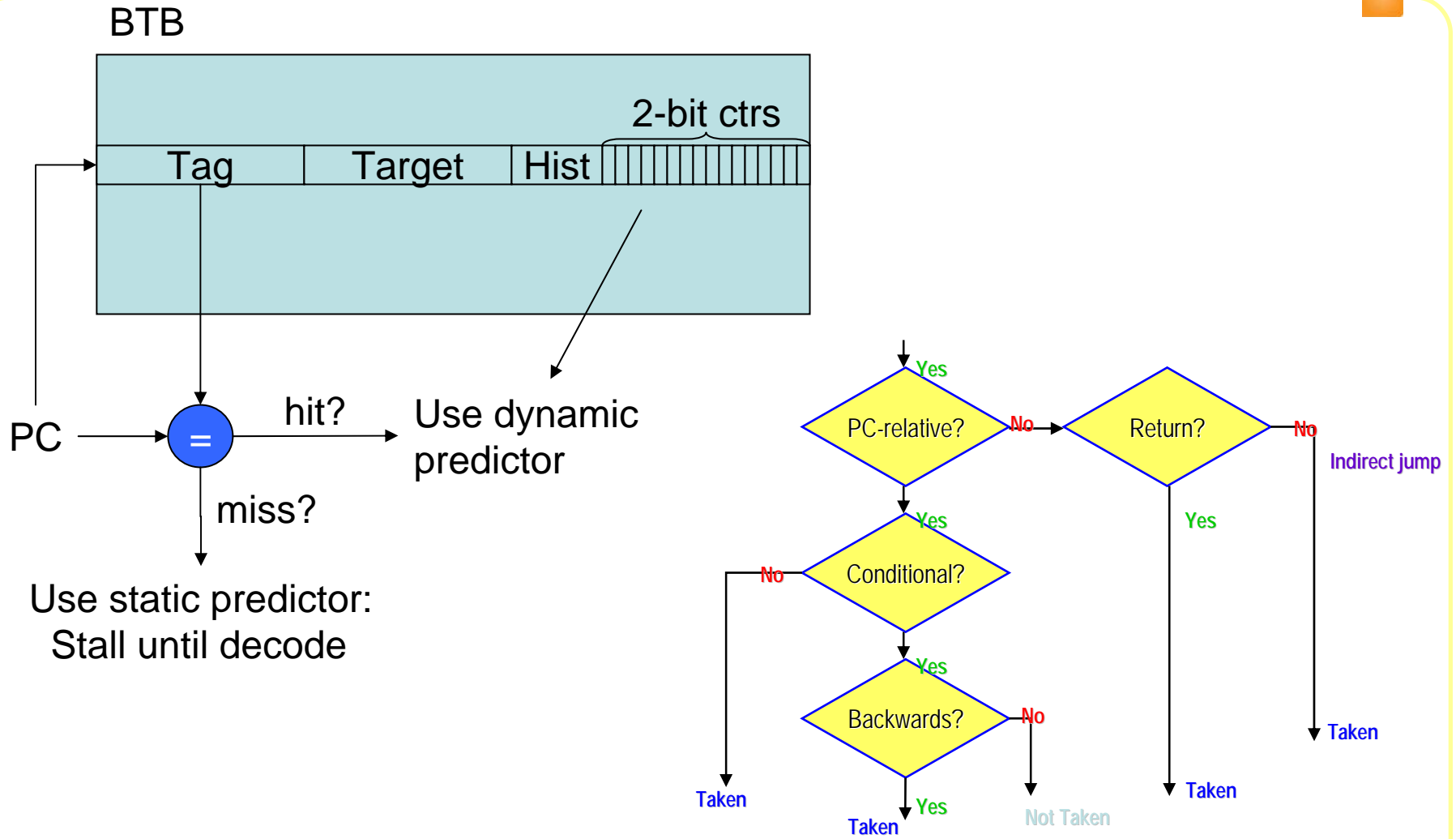
OOO core has drained, retire bad branch and flush rest of OOO core

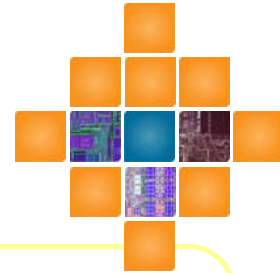


Normal execution: speculatively fetch and execute instructions



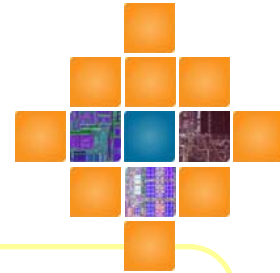
Branch Prediction





Micro-op Decomposition

- CISC → RISC
 - Simple x86 instructions map to single uop
 - Ex. INC, ADD (r-r), XOR, MOV (r-r, load)
 - Moderately complex insts map to a few uops
 - Ex. Store → STA/STD
 - ADD (r-m) → LOAD/ADD
 - ADD (m-r) → LOAD/ADD/STA/STD
 - More complex make use of UROM
 - PUSHA → STA/STD/ADD, STA/STD/ADD, ...



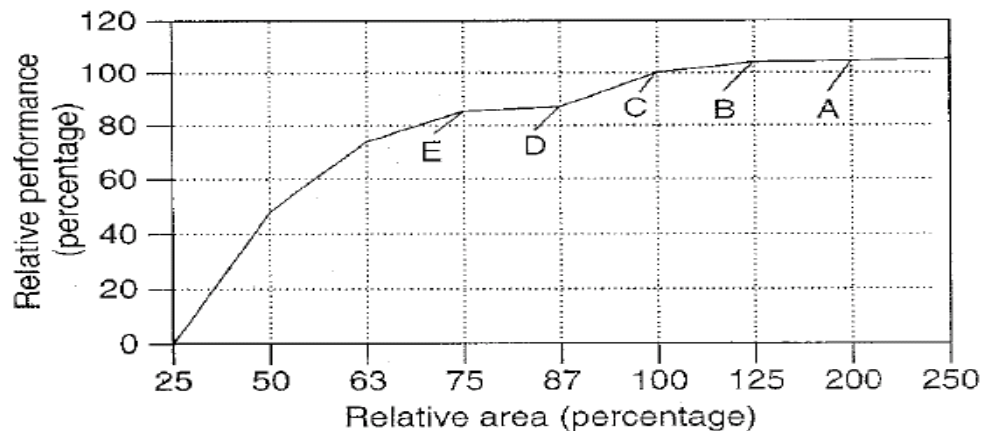
Decoder

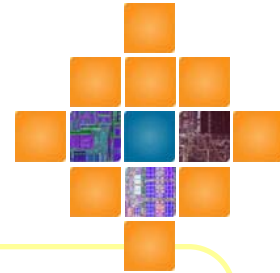
- 4-1-1 limitation

- Decode up to three instructions per cycle

- Three decoders, but asymmetric
- Only first decoder can handle moderately complex insts (those that can be encoded with up to 4 uops)
- If need more than 4 uops, go to UROM

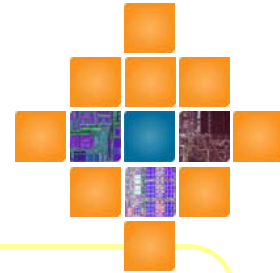
- A: 4-2-2-2
- B: 4-2-2
- C: 4-1-1
- D: 4-2
- E: 4-1





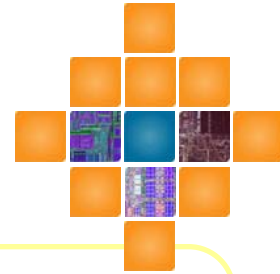
“Simple” Core

- After decode, the machine only deals with uops until commit
- Rename, RS, ROB, ...
 - Looks just like a RISC-based OOO core
 - A couple of changes to deal with x86
 - Flags
 - Partial register writes

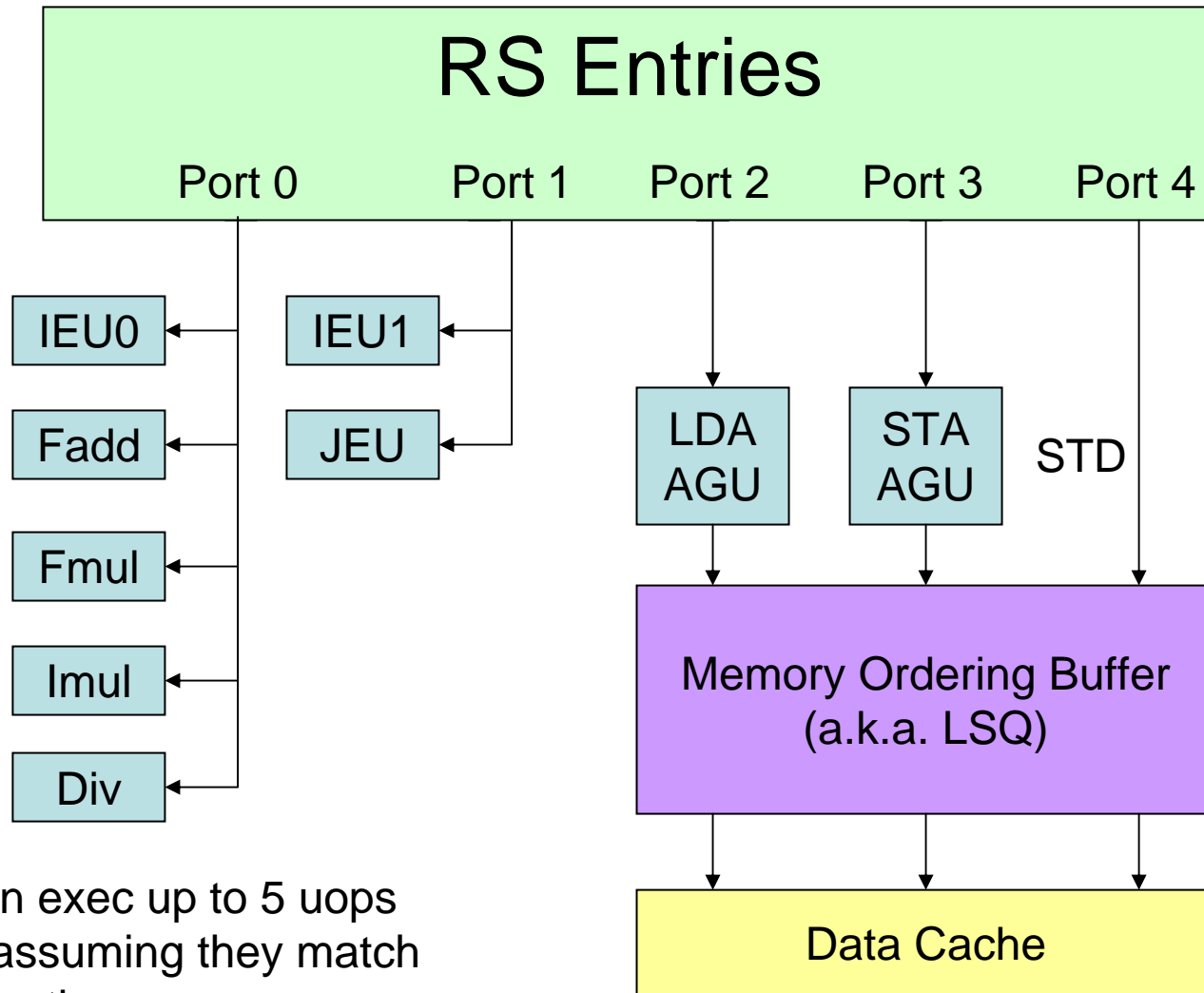


Execution Ports

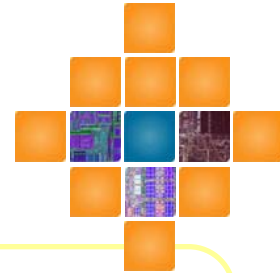
- Unified RS, multiple ALUs
 - Ex. Two Adders
 - What if multiple ADDs ready at the same time?
 - Need to choose 2-of-N and make assignments
 - To simplify, each ADD is assigned to an adder during Alloc stage
 - Each ADD can only attempt to execute on its assigned adder
 - If my assigned adder is busy, I can't go *even if the other adder is idle*
 - Reduce selection problem to choosing 1-of-N (easier logic)



Execution Ports (con't)

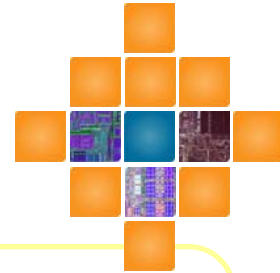


In theory, can exec up to 5 uops per cycle... assuming they match the ALUs exactly



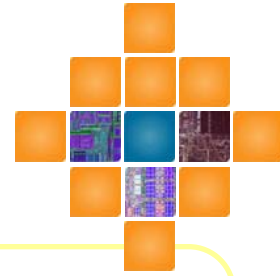
RISC→CISC Commit

- External world doesn't know about uops
- Instruction commit must be all-or-nothing
 - Either commit all uops from an inst or none
 - Ex. ADD [EBX], ECX
 - LOAD [EBX]
 - ADD tmp0 = EBX, ECX
 - STA tmp1 = EBX
 - STD tmp2 = tmp0
 - If load has page fault, if store has protection fault, if ...



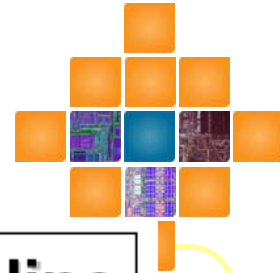
Case Study 2: Intel P4

- Primary Objectives
 - Clock speed
 - Implies performance
 - True if CPI not increases too much
 - Marketability (GHz sells!)
 - Clock speed
 - Clock speed



Faster Clock Speed

- Less work per cycle
- Traditional single-cycle tasks may be multi-cycle
 - More pipeline bubbles, idle resources
- More pipeline stages
 - More control logic (need to control each stage)
 - More circuits to design (more engineering effort)
- More critical paths
 - More timing paths are at or close to clock speed
 - Less benefit from tuning worst paths
- Higher power
 - $P = \frac{1}{2}CV^2f$



Extra Delays Needed

Basic Pentium III Processor Misprediction Pipeline

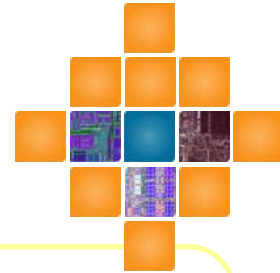
1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium 4 Processor Misprediction Pipeline

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC	Nxt IP	TC	Fetch	Drive	Alloc	Rename	Que	Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Figs	Br Ck	Drive	

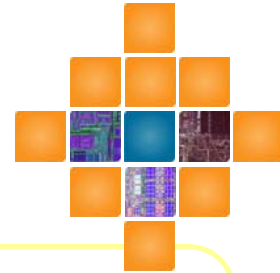


- Branch mispred pipeline has 2 “Drive” stages
 - Extra delay because P4 can't get from Point A to Point B in less than a cycle
- Side Note
 - P4 does not have a “20 stage pipeline” It's much longer!



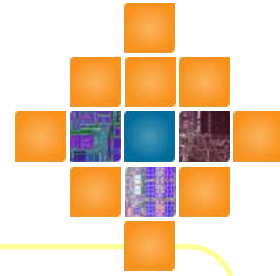
Make Common Case Fast

- Fetch:
 - Usually I\$ hit
 - Branches are frequent
 - Branches are often taken
 - Branch mispredictions are not that infrequent
 - Even if frequency is low, cost is high (pipe flush)
- P4 Uses a “Trace Cache”
 - Caches dynamic instruction stream
 - Contrast to I\$ which caches the static instruction image

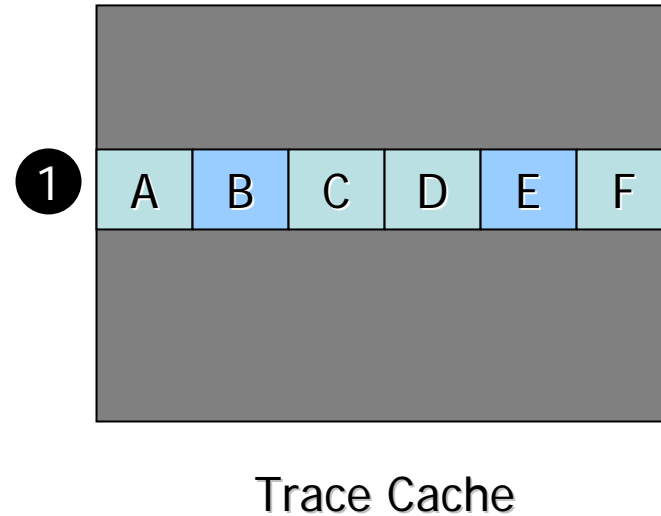
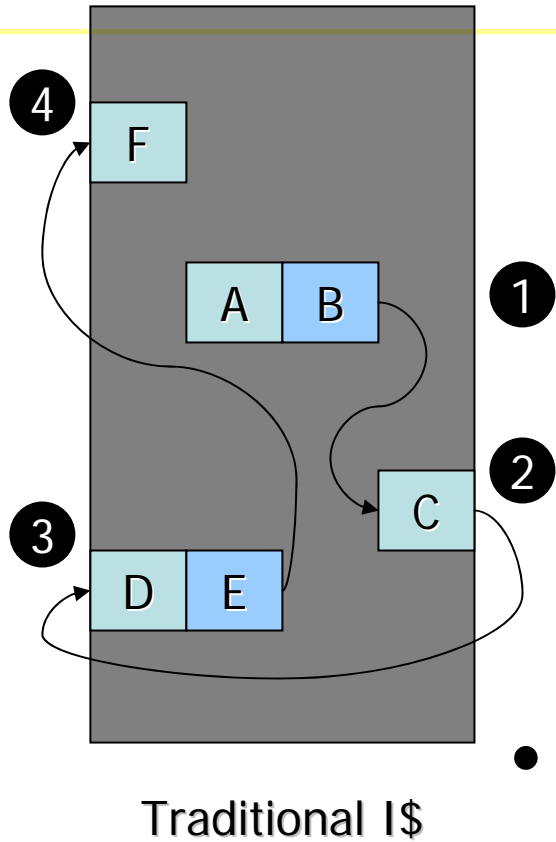


Traditional Fetch/I\$

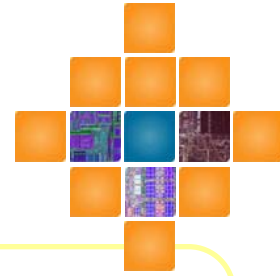
- Fetch from only one I\$ line per cycle
 - If fetch PC points to last instruction in a line, all you get is one instruction
 - Potentially worse for x86 since arbitrary byte-aligned instructions may straddle cache lines
 - Can only fetch instructions up to a taken branch
- Branch misprediction causes pipeline flush
 - Cost in cycles is roughly num-stages from fetch to branch execute



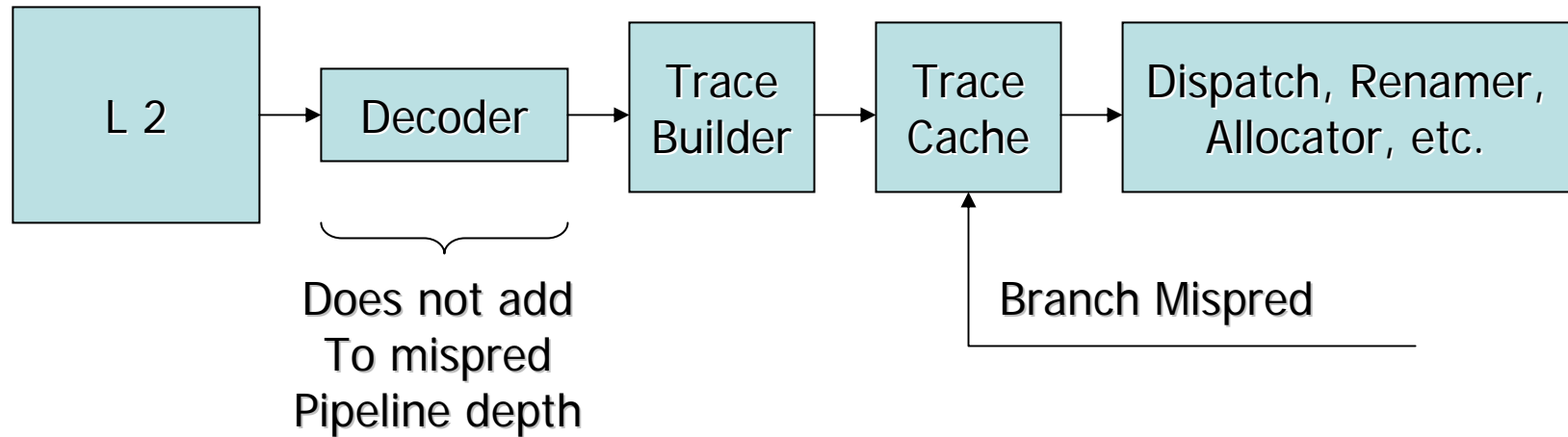
Trace Cache



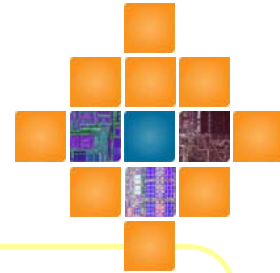
- Multiple “I\$ Lines” per cycle
- Can fetch past taken branch
 - And even multiple taken branches



Decoded Trace Cache

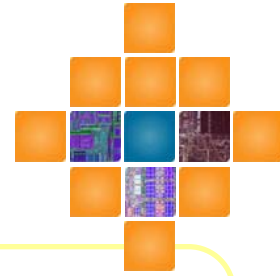


- Trace cache holds decoded x86 instructions instead of raw bytes
- On branch mispred, decode stage not exposed in pipeline depth



Less Common Case Slower

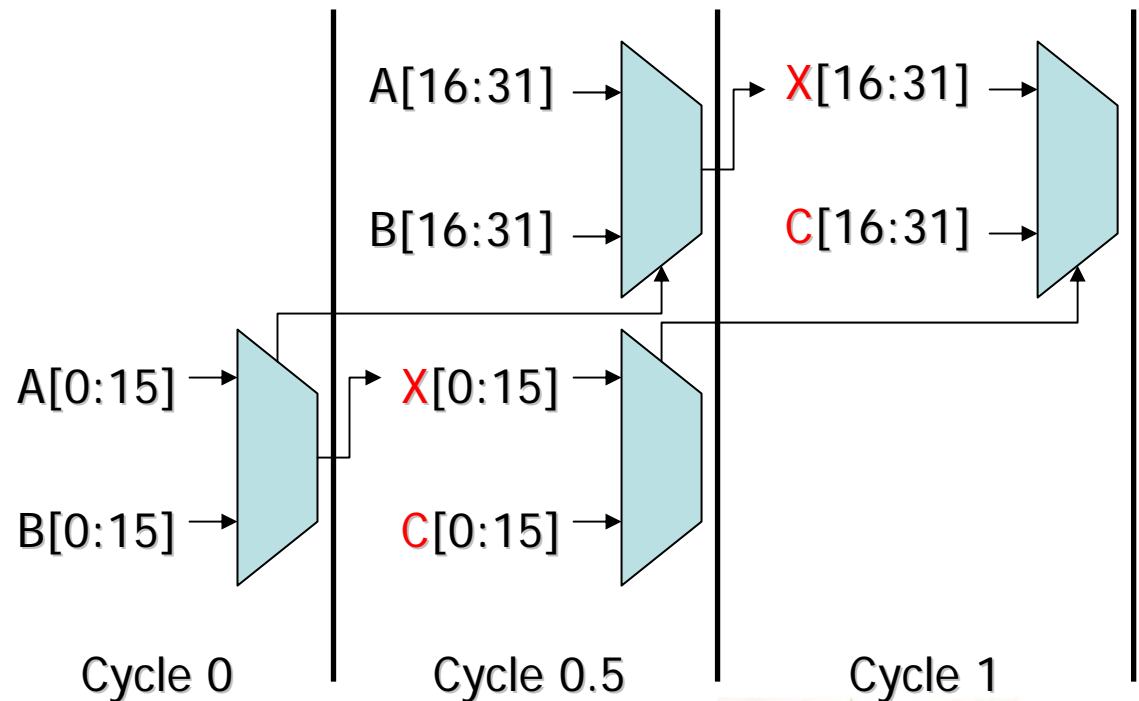
- Trace Cache is Big
 - Decoded instructions take more room
 - X86 instructions may take 1-15 bytes raw
 - All decoded uops take same amount of space
 - Instruction duplication
 - Instruction “X” may be redundantly stored
 - ABX, CDX, XYZ, EXY
- Tradeoffs
 - No I\$
 - Trace\$ miss requires going to L2
 - Decoder width = 1
 - Trace\$ hit = 3 ops fetched per cycle
 - Trace\$ miss = 1 op decoded (therefore fetched) per cycle

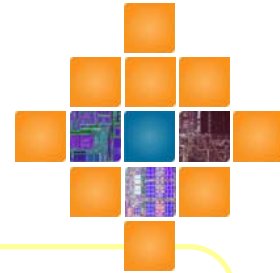


Addition

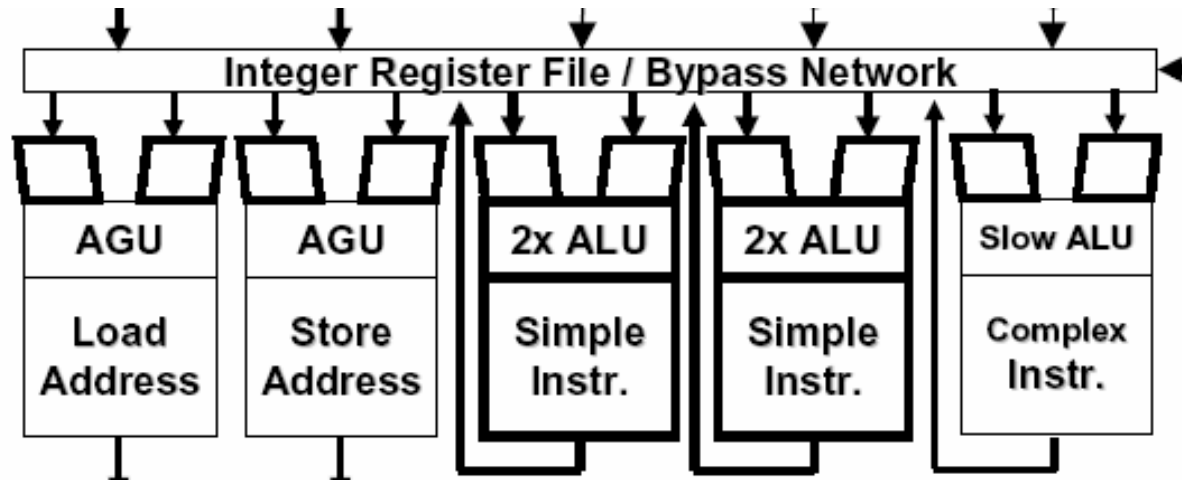
- Common Case: Adds, Simple ALU Insts
- Typically an add must occur in a single cycle
- P4 “double-pumps” adders for 2 adds/cycle!
 - 2.0 GHz P4 has 4.0 GHz adders

$$X = A + B$$
$$Y = X + C$$

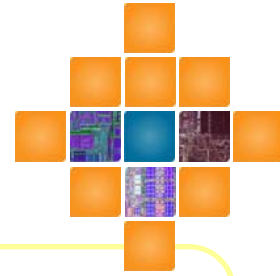




Common Case Fast

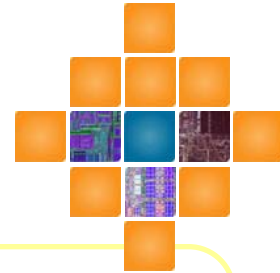


- So long as only executing simple ALU ops, can execute two dependent ops per cycle
- 2 ALUs, so peak = 4 simple ALU ops per cycle
 - Can't sustain since T\$ only delivers 3 ops per cycle
 - Still useful (e.g., after D\$ miss returns)



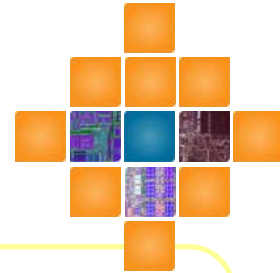
Less Common Case Slower

- Requires extra cycle of bypass when not doing only simple ALU ops
 - Operation may need extra half-cycle to finish
- Shifts are relatively slower in P4 (compared to previous latencies in P3)
 - Can reduce performance of code optimized for older machines

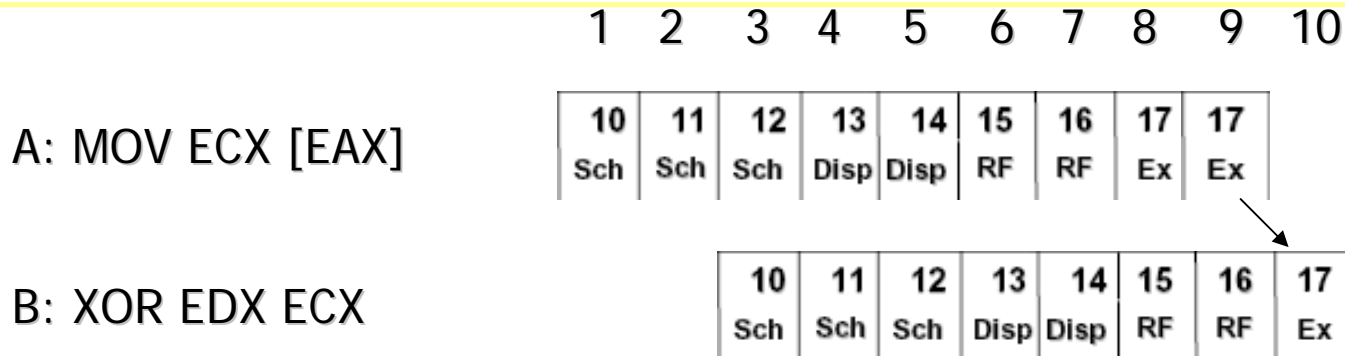


Common Case: Cache Hit

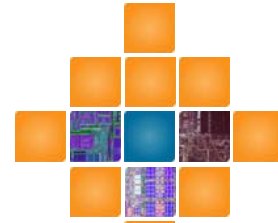
- Cache hit/miss complicates dynamic scheduler
- Need to know instruction latency to schedule dependent instructions
- Common case is cache hit
 - To make pipelined scheduler, just assume loads always hit



Pipelined Scheduling



- In cycle 3, start scheduling B assuming A hits in cache
- At cycle 10, A's result bypasses to B, and B executes



Less Common Case is Slower

1 2 3 4 5 6 7 8 9 10 11 12 13 14

A: MOV ECX [EAX]

10	11	12	13	14	15	16	17	17	17	17	17	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Ex	Ex	Ex	Ex	Ex

B: XOR EDX ECX

10	11	12	13	14	15	16	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex

C: SUB EAX ECX

10	11	12	13	14	15	16	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex

D: ADD EBX EAX

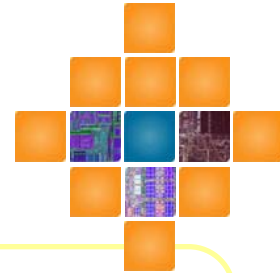
10	11	12	13	14	15	16	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex

E: NOR EAX EDX

10	11	12	13	14	15	16	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex

F: ADD EBX EAX

10	11	12	13	14	15	16	17
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex

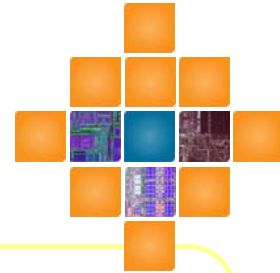


Replay

- On cache miss, dependents are speculatively misscheduled
 - Wastes execution slots
 - Other “useful” work could have executed instead
 - Wastes a lot of power
 - Adds latency
 - Miss not known until cycle 9
 - Start rescheduling dependents at cycle 10
 - Could have executed faster if miss was known

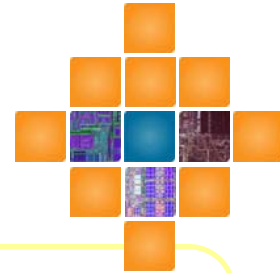
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

10	11	12	13	14	15	16	17	17	17	17	17	17				
Sch	Sch	Sch	Disp	Disp	RF	RF	Ex	Ex	Ex	Ex	Ex	Ex				
									10	11	12	13	14	15	16	17
									Sch	Sch	Sch	Disp	Disp	RF	RF	Ex
									10	11	12	13	14	15	16	17
									Sch	Sch	Sch	Disp	Disp	RF	RF	Ex



P4 Philosophy Overview

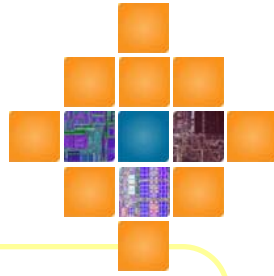
- Amdahl's Law...
- Make the common case fast!!!
 - Trace Cache
 - Double-Pumped ALUs
 - Cache Hits
 - There are other examples...
- Resulted in very high frequency



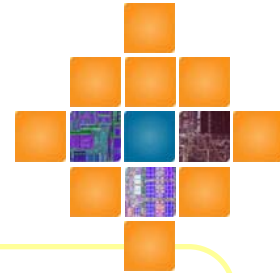
P4 Pitfall

- Making the less common case too slow
 - Performance determined by both the common case and uncommon case
 - If uncommon case *too* slow, can cancel out gains of making common case faster
 - common by what metric? (should be time)
- Lesson: Beware of Shhadma
 - Don't screw over the less common case

Tejas Lessons



- Next-Gen P4 (P5?)
- Cancelled spring 2004
 - Complexity of super-duper-pipelined processor
 - Time-to-Market slipping
 - Performance Goals slipping
 - Complexity became unmanageable
 - Power and thermals out of control
- “Performance at all costs” no longer true



Lessons to Carry Forward

- Performance is still King
- But restricted by power, thermals, complexity, design time, cost, etc.
- Future processors are more balanced
 - Centrino, Core, Core 2
 - Opteron