Superscalar Design

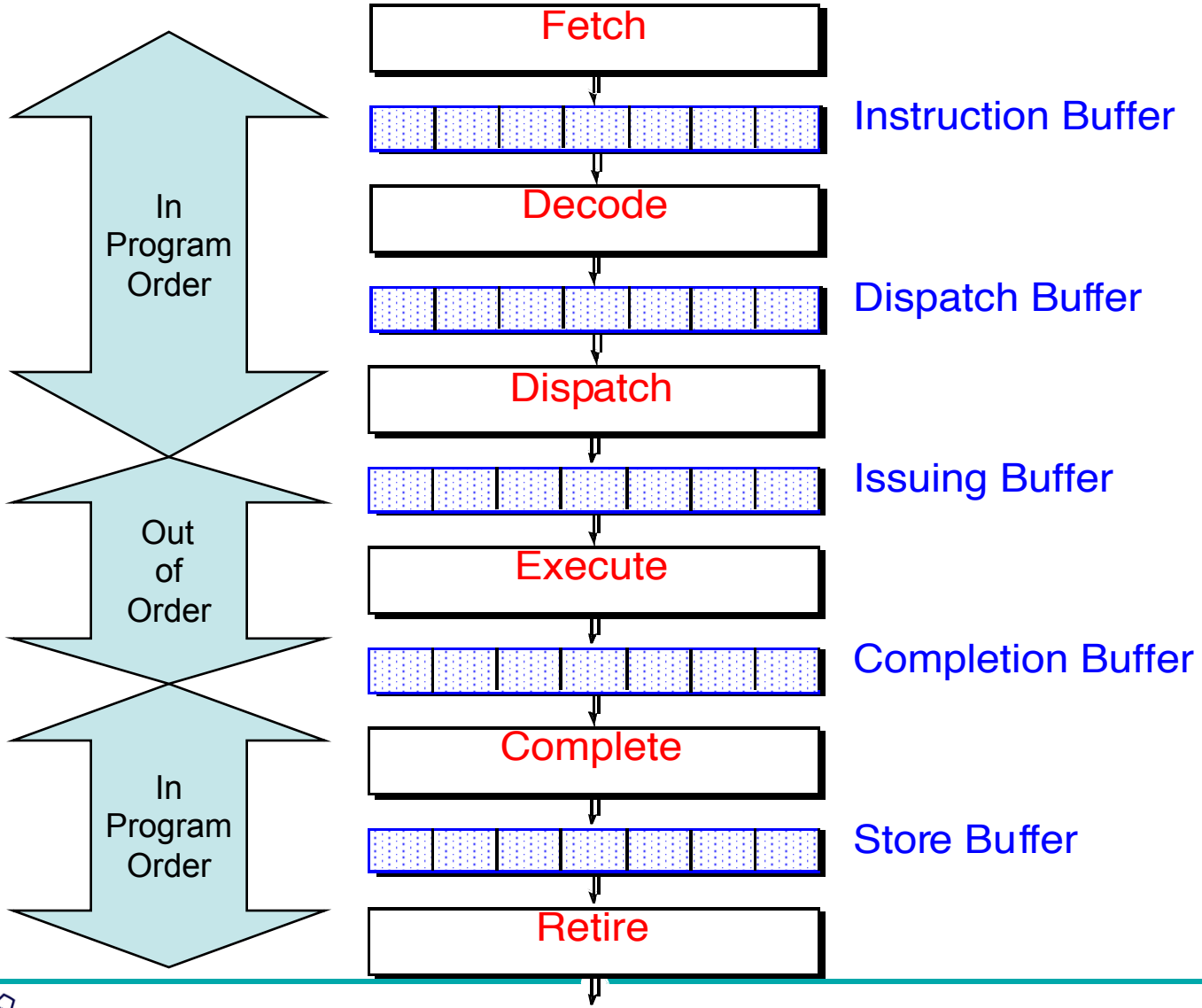
Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Lecture 3 (21 Jan 2015)

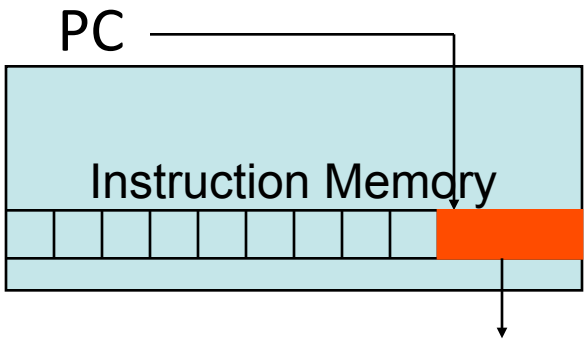
CADSL

Superscalar Pipeline Stages



Instruction Fetch

- Objective: Fetch multiple instructions per cycle
- Challenges:
  - Branches: control dependences
  - Branch target misalignment
  - Instruction cache misses

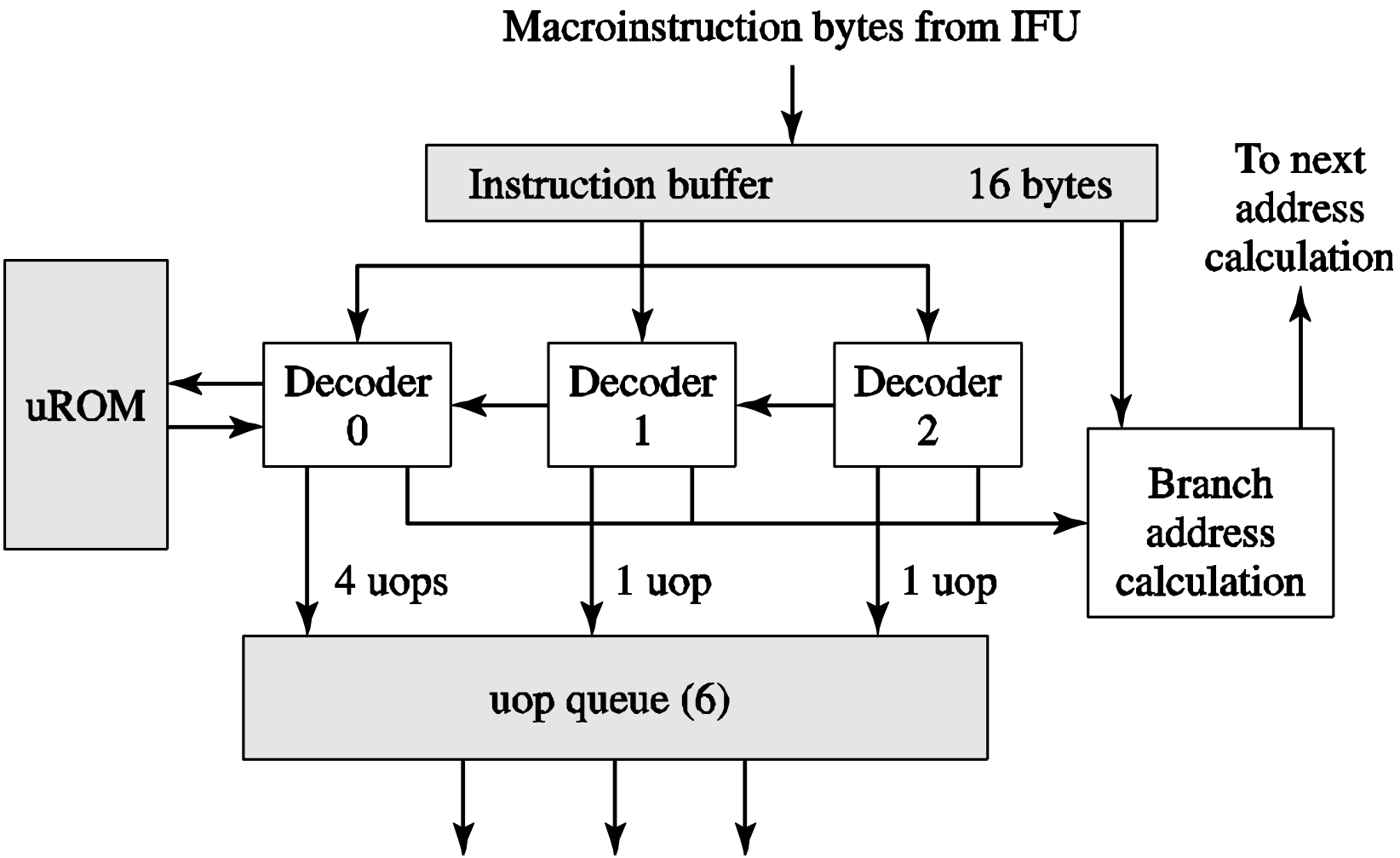


Issues in Decoding

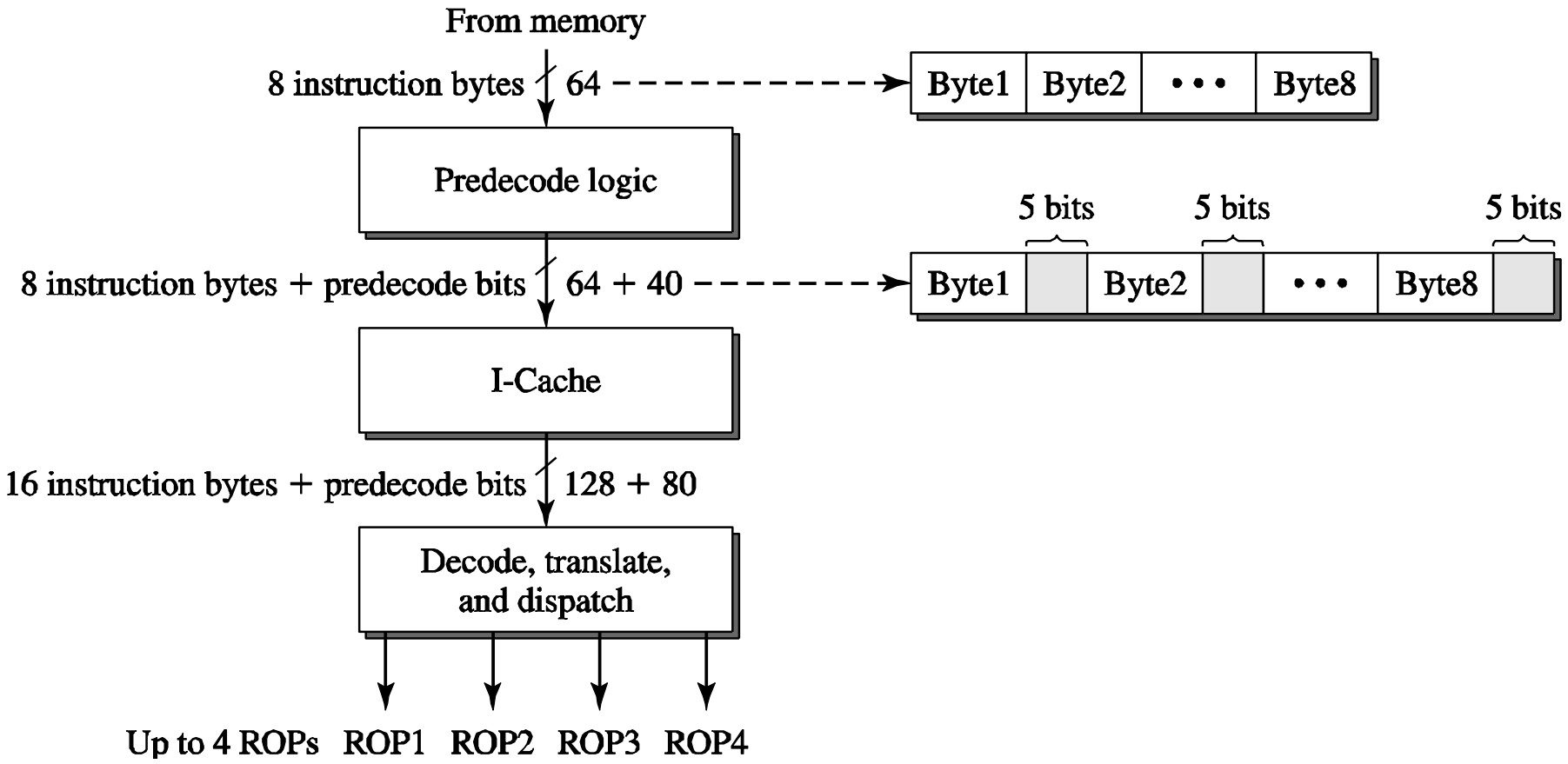
- Primary Tasks
  - Identify individual instructions (!)
  - Determine instruction types
  - Determine dependences between instructions

- Two important factors
  - Instruction set architecture
  - Pipeline width

Pentium Pro Fetch/Decode



Predecoding in the AMD K5



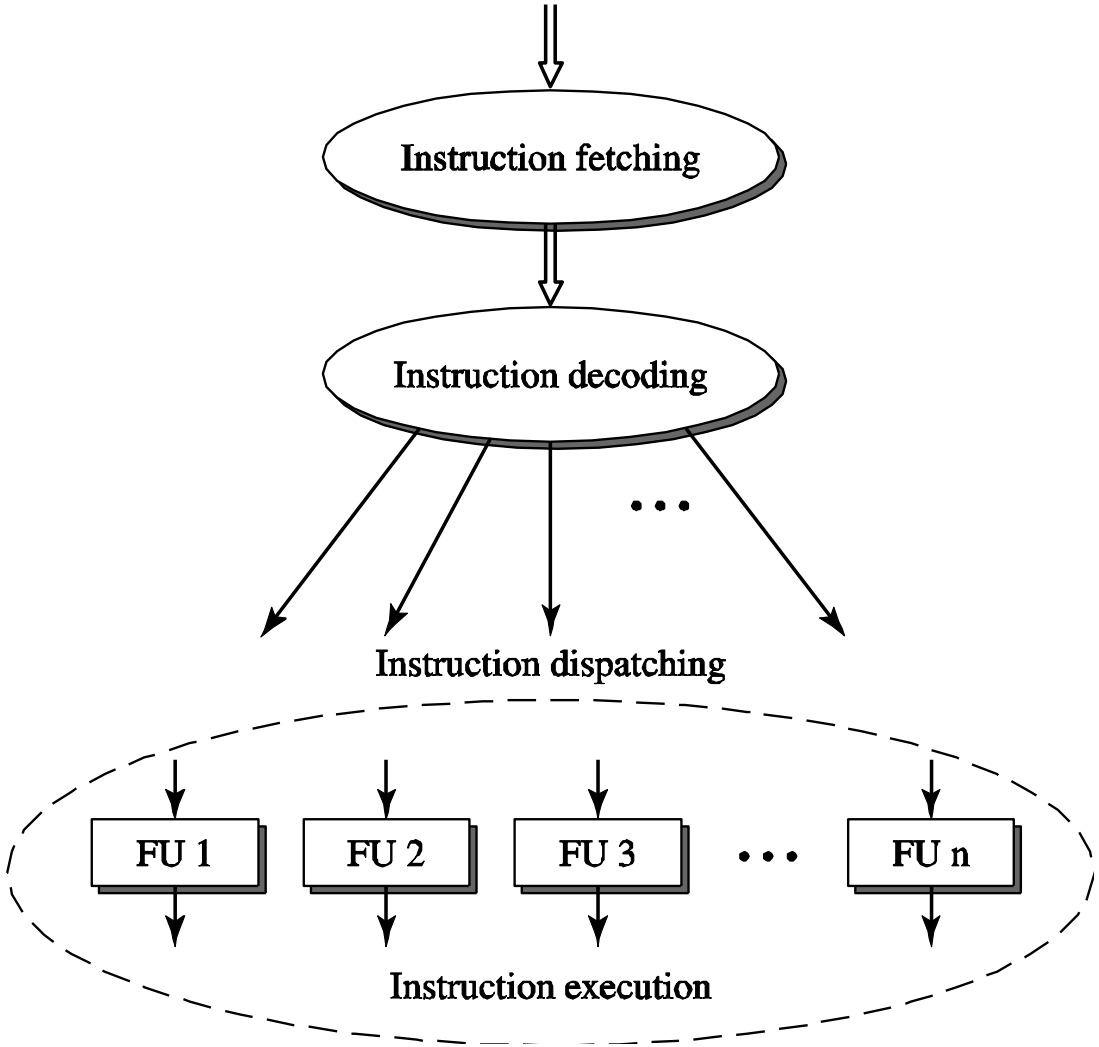
Instruction Dispatching

- Diversified pipeline
- Different type instructions executed by different FU in different pipelines
- Distributed control
- Operands are fetched from RF
- Operands may not be available
- Reservation station

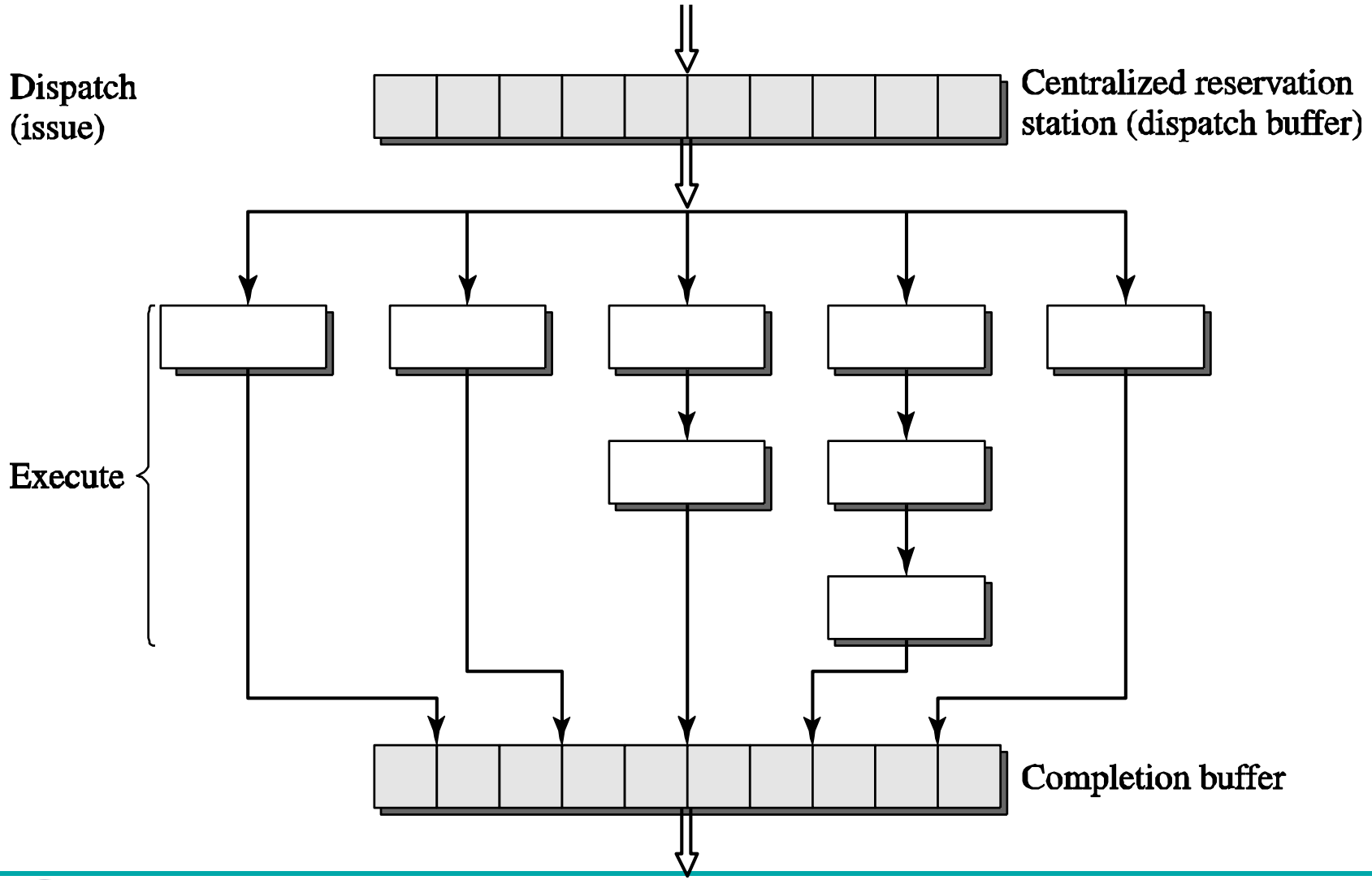
Instruction Dispatch and Issue

- Parallel pipeline
  - Centralized instruction fetch
  - Centralized instruction decode
- Diversified pipeline
  - Distributed instruction execution

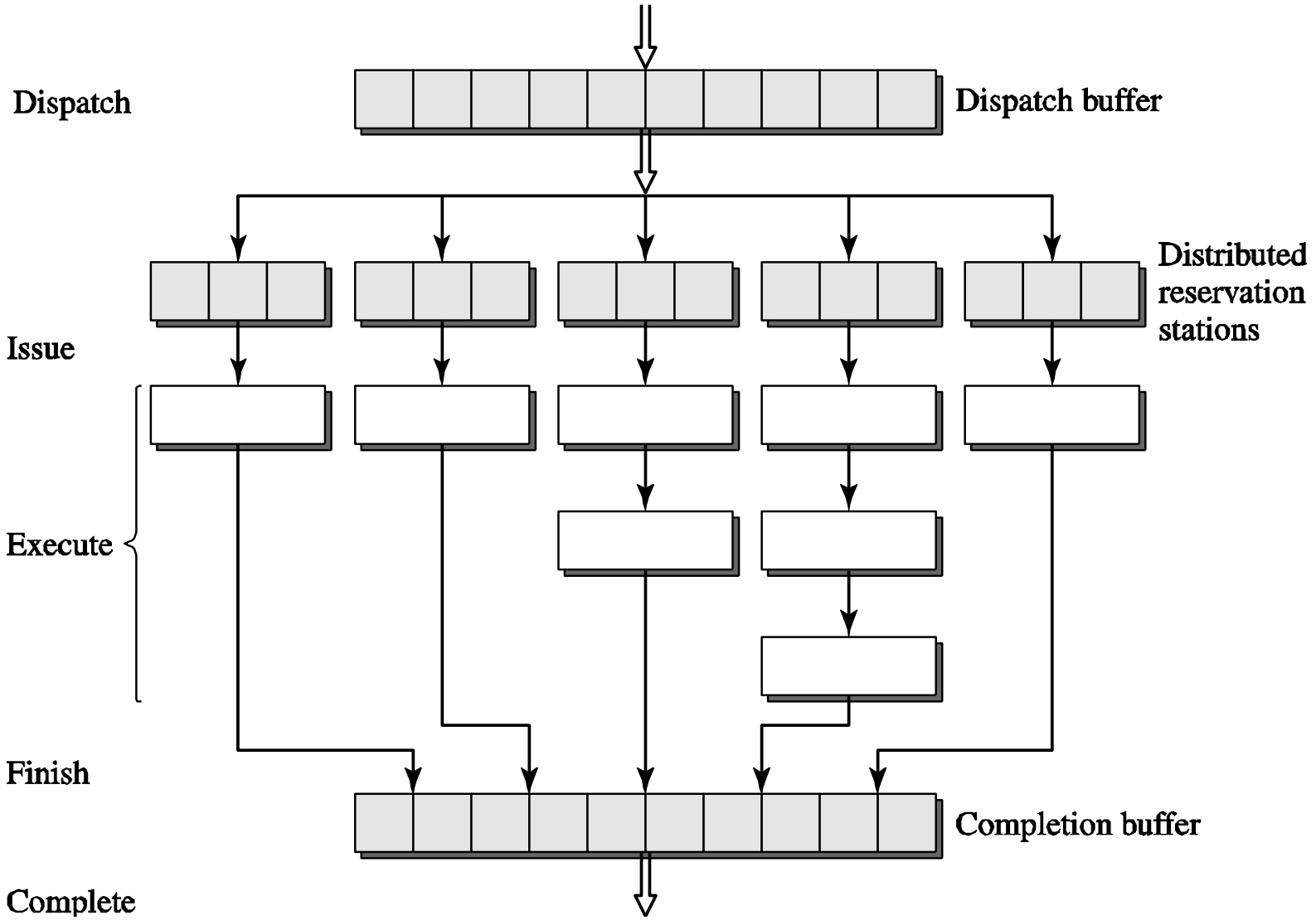
Necessity of Instruction Dispatch



Centralized Reservation Station



Distributed Reservation Station



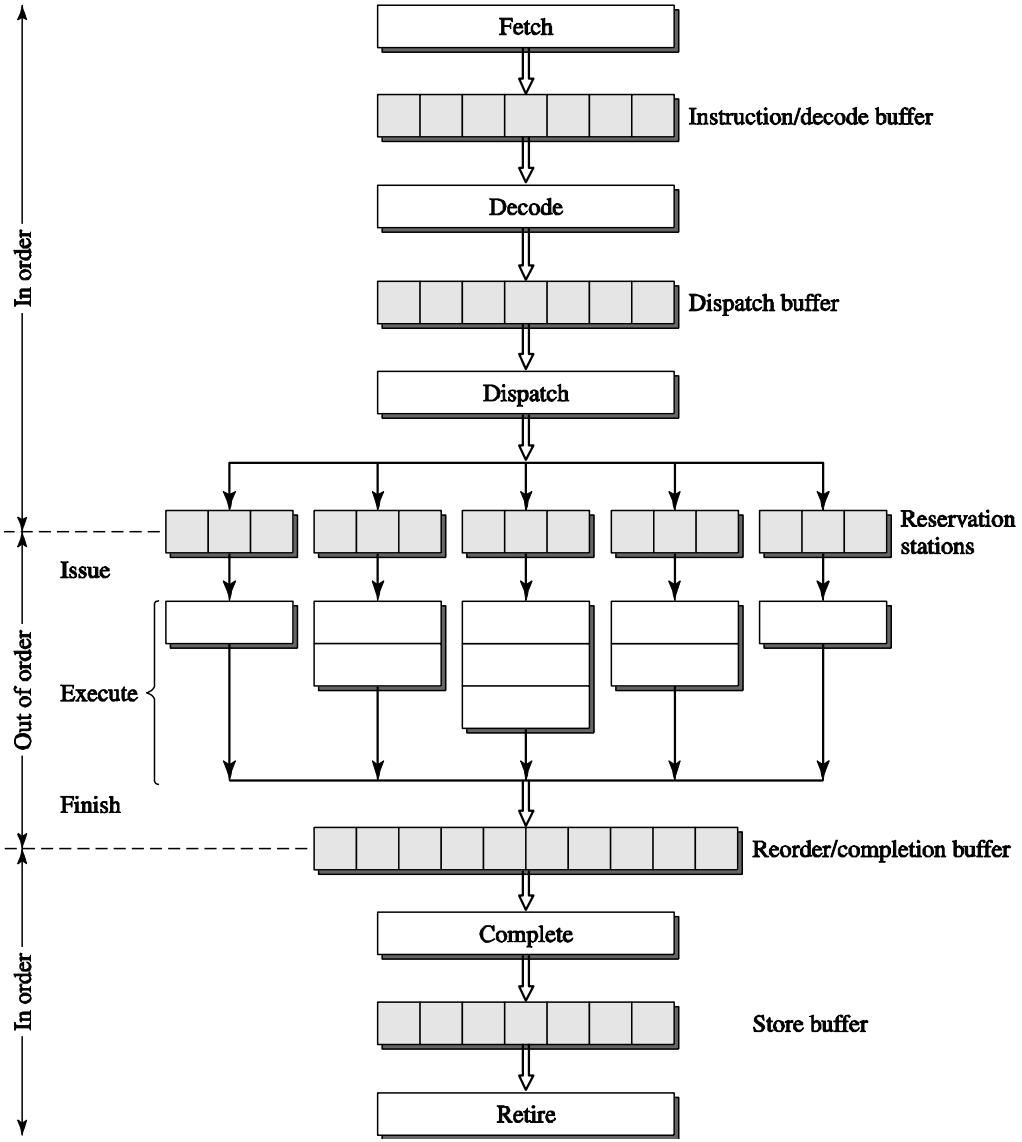
Issues in Instruction Execution

- Current trends
  - More parallelism ← bypassing very challenging
  - Deeper pipelines
  - More diversity
- Functional unit types
  - Integer
  - Floating point
  - Load/store ← most difficult to make parallel
  - Branch
  - Specialized units (media)
    - Very wide datapaths (256 bits/register or more)

Issues in Completion/Retirement

- Out-of-order execution
  - ALU instructions
  - Load/store instructions
- In-order completion/retirement
  - Precise exceptions
  - Memory coherence and consistency
- Solutions
  - Reorder buffer
  - Store buffer

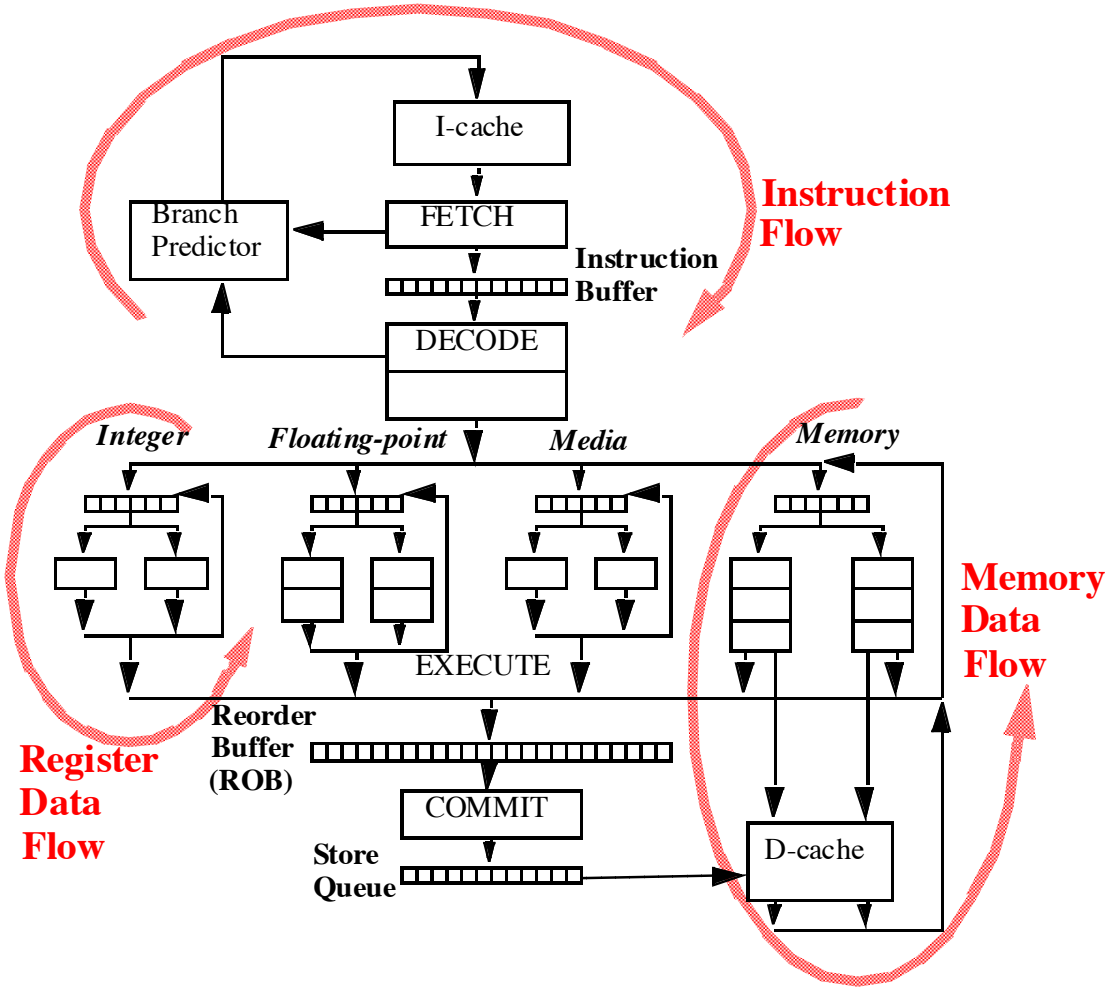
A Dynamic Superscalar Processor



Superscalar Techniques

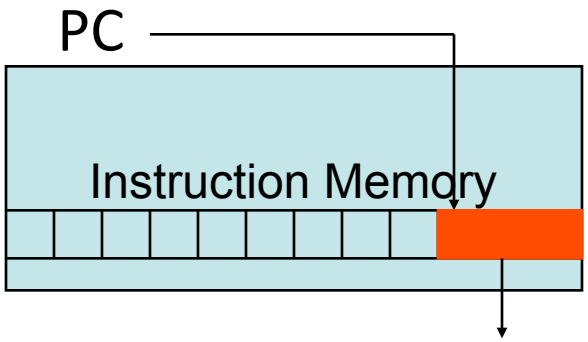
- The ultimate performance goal of a superscalar pipeline is to achieve maximum throughput of instruction processing
- Instruction processing involves
  - Instruction flow – Branch instructions
  - Register data flow – ALU instructions
  - Memory data flow – L/S instructions
- Max Throughput – Min (Branch, ALU, Load penalty)

Impediments to High IPC



Instruction Flow

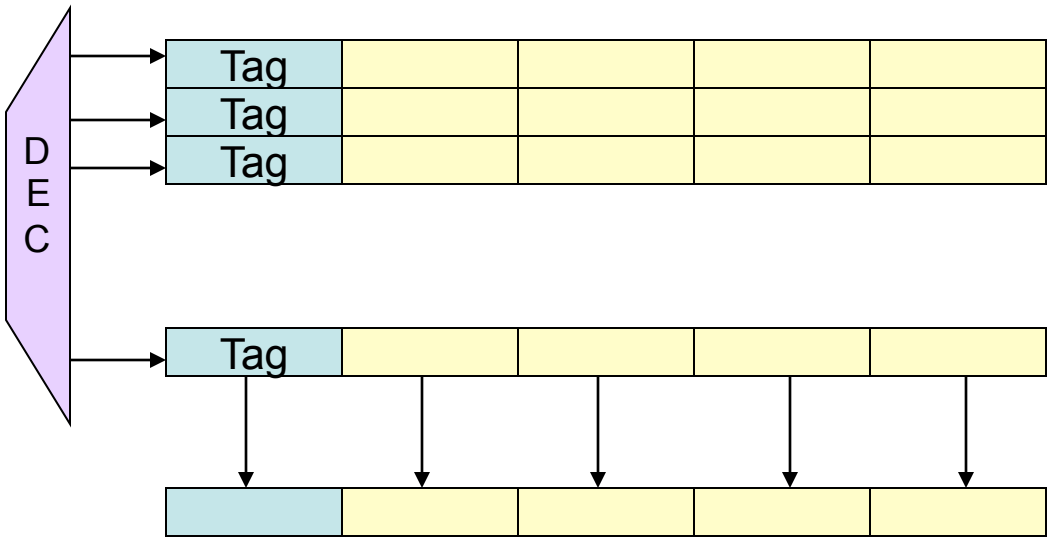
- Objective: Fetch multiple instructions per cycle
- Challenges:
  - Branches: control dependences
  - Branch target misalignment
  - Instruction cache misses



Instruction Fetch

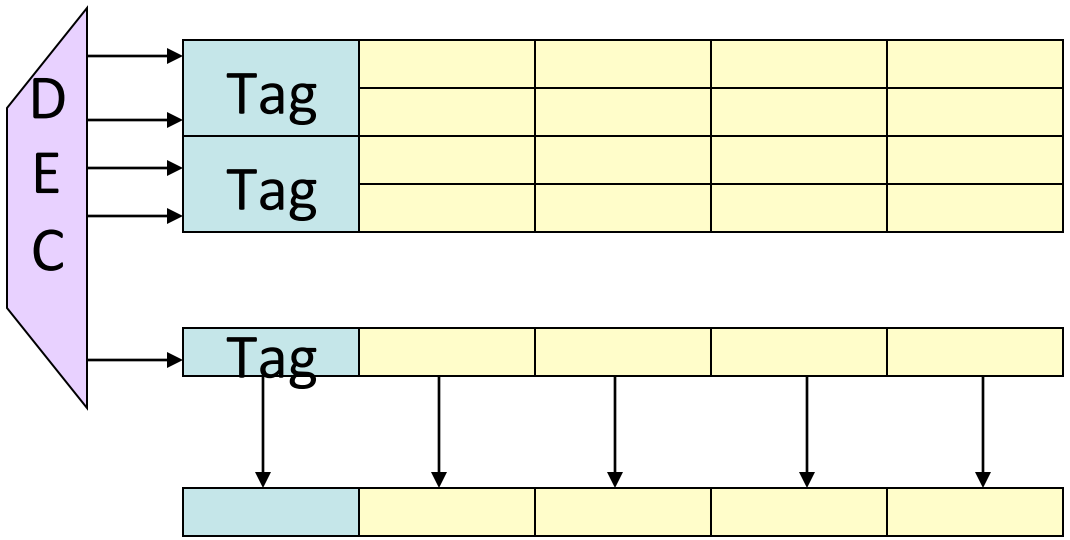
- Fetch s instructions from I-cache
- I-Cache must be wide enough that each row of the I-Cache array can store s instructions and that an entire row can be accessed
- Fetch width = Row width
- Assume access latency is 1 cycle

I-Cache Organization



1 cache line = 1 physical row

I-Cache Organization

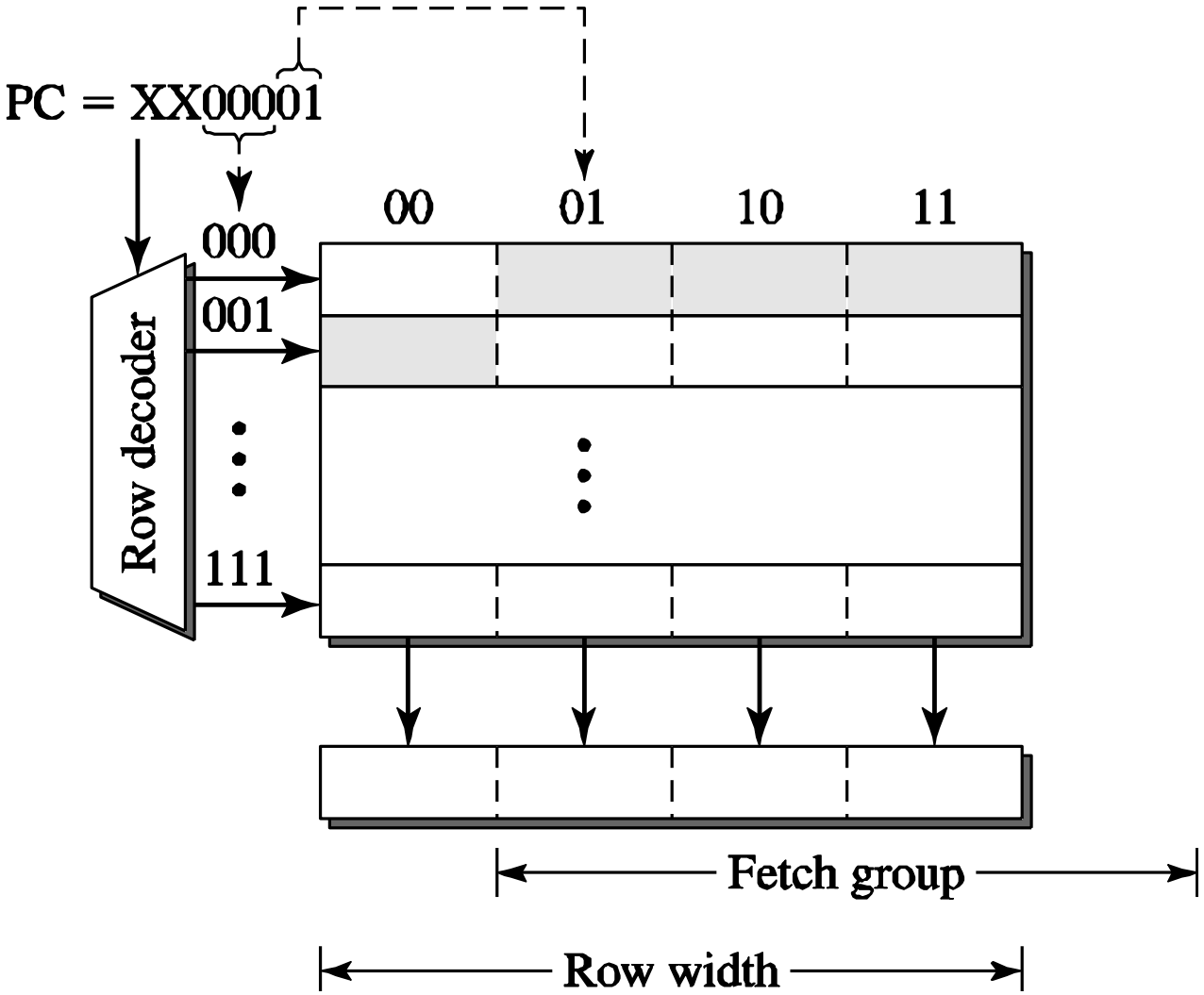


1 cache line = 2 physical rows

Instruction Flow

- Objective: Fetch multiple instructions per cycle
- Challenges:
  - Branches: control dependences
  - Branch target misalignment
  - Instruction cache misses
- Solutions
  - Code alignment (static vs. dynamic)
  - Prediction/speculation

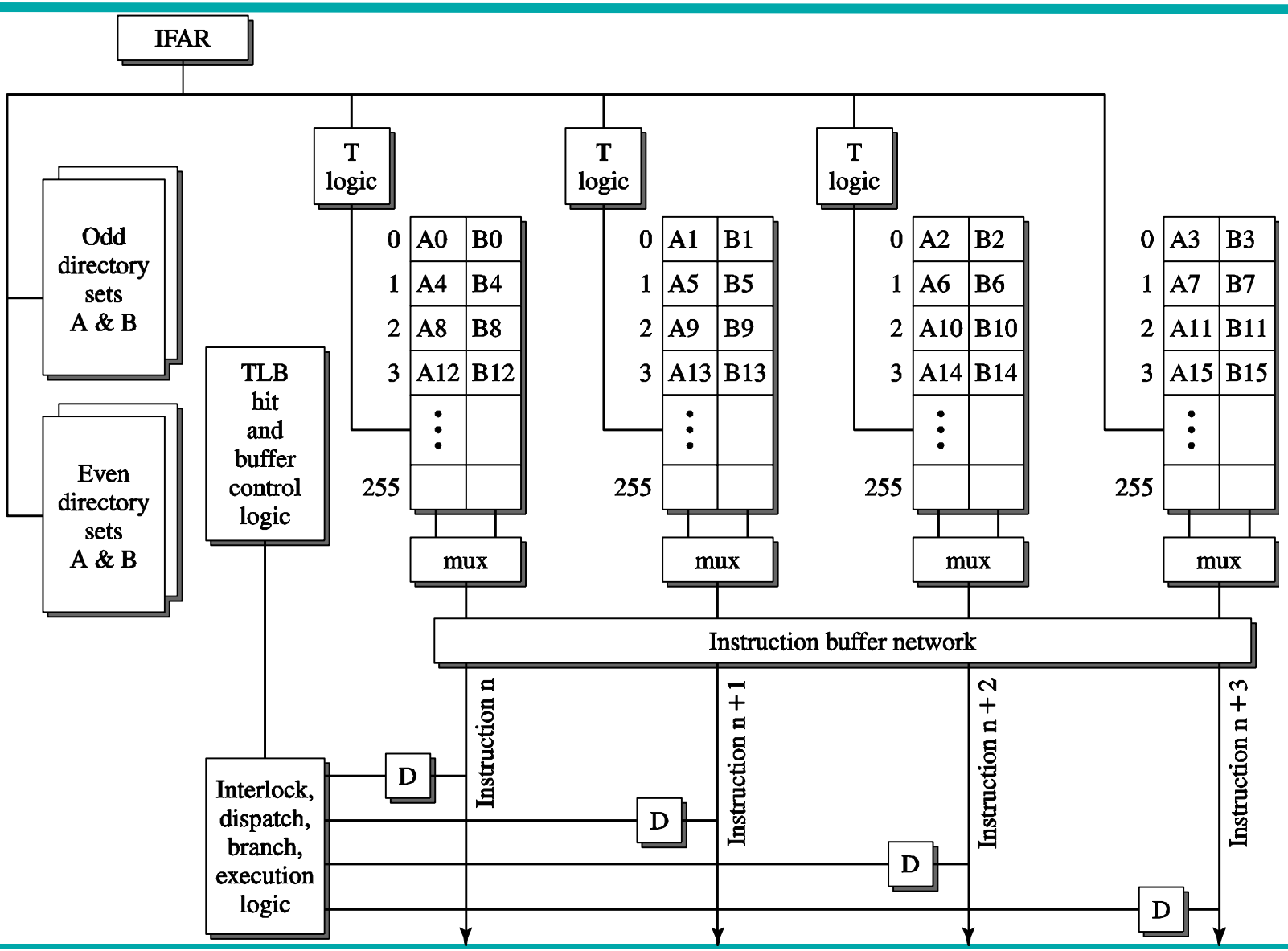
Fetch Alignment



Instruction Fetch

- 2 – way set associative I-Cache with a line size of 16 instructions (64 bytes)
- Each row of the I-Cache stores 4 associative sets 9two per set) of instructions
- Each line of I-cache spans four physical rows
- Physical I-cache array is actually composed of 4 independent sub-arrays
- One instruction can be accessed form one array

RIOS-I Fetch Hardware



Goal and Impediments

- Goal of Instruction Flow
  - Supply processor with maximum number of useful instructions every clock cycle
- Impediments
  - Branches and jumps
  - Finite I-Cache
    - Capacity
    - Bandwidth restrictions

Branch Types and Implementation

1. Types of Branches
   A. Conditional or Unconditional
   B. Save PC?
   C. How is target computed?
      - Single target (immediate, PC+immediate)
      - Multiple targets (register)
2. Branch Architectures
   A. Condition code or condition registers
   B. Register

Branches – PowerPC

12 Types of Branches

Branch (unconditional, no save PC, PC+imm)
Branch absolute (uncond, no save PC, imm)
Branch and link (uncond, save PC, PC+imm)
Branch abs and link (uncond, save PC, imm)
Branch conditional (conditional, no save PC, PC+imm)
Branch cond abs (cond, no save PC, imm)
Branch cond and link (cond, save PC, PC+imm)
Branch cond abs and link (cond, save PC, imm)
Branch cond to link register (cond, don't save PC, reg)
Branch cond to link reg and link (cond, save PC, reg)
Branch cond to count reg (cond, don't save PC, reg)
Branch cond to count reg and link (cond, save PC, reg)

Branches – DEC Alpha

Alpha

3 Types of Branches

Conditional branch (cond, no save PC, PC+imm)

Bxx Ra, disp

Unconditional branch (uncond, Save PC, PC+imm)

Br Ra, disp

Jumps (uncond, save PC, Register)

J Ra

Branches – MIPS

MIPS

6 Types of Branches

Jump (uncond, no save PC, imm)

Jump and link (uncond, save PC, imm)

Jump register (uncond, no save PC, register)

Jump and link register (uncond, save PC, register)

Branch (conditional, no save PC, PC+imm)

Branch and link (conditional, save PC, PC+imm)

What's So Bad About Branches?

- Effects of Branches
  - Fragmentation of I-Cache lines
  - Need to determine branch direction
  - Need to determine branch target
  - Use up execution resources
    - Pipeline drain/fill

What's So Bad About Branches?

Problem: Fetch stalls until direction is determined

Solutions:

- Minimize delay
  - Move instructions determining branch condition away from branch (CC architecture)
- Make use of delay
  - Non-speculative:
    - Fill delay slots with useful safe instructions
    - Execute both paths (eager execution)
  - Speculative:
    - Predict branch direction

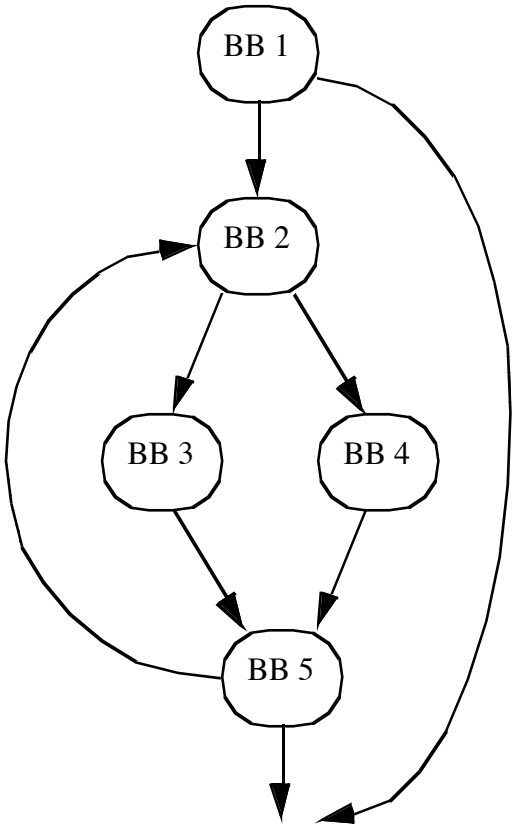
What's So Bad About Branches?

Problem: Fetch stalls until branch target is determined

Solutions:

- Minimize delay
  - Generate branch target early
- Make use of delay: Predict branch target
  - Single target
  - Multiple targets

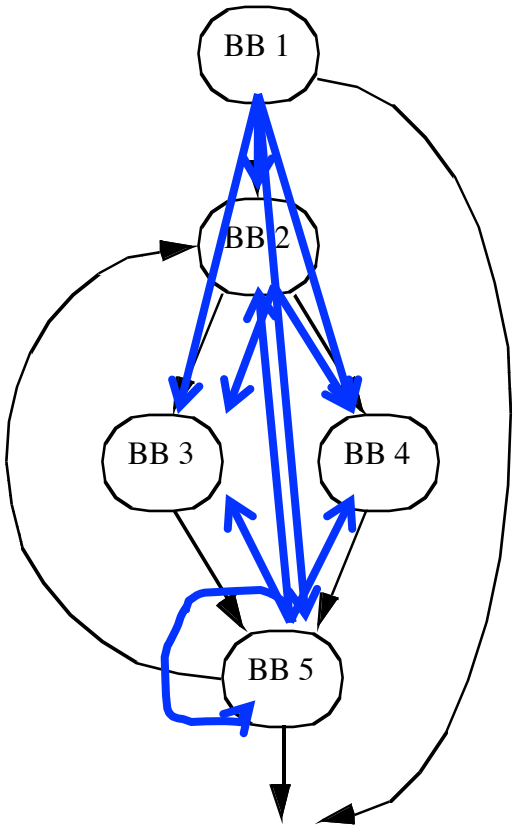
Control Dependences




```
main:
        addi r2, r0, A      |
        addi r3, r0, B      |
        addi r4, r0, C      |   BB 1
        addi r5, r0, N      |
        add  r10,r0, r0     |
        bge  r10,r5, end    |
loop:
        lw   r20, 0(r2)     |
        lw   r21, 0(r3)     |   BB 2
        bge  r20,r21,T1     |
        sw   r21, 0(r4)     |   BB 3
        b    T2             |
T1:
        sw   r20, 0(r4)     |   BB 4
T2:
        addi r10,r10,1      |
        addi r2, r2, 4      |
        addi r3, r3, 4      |   BB 5
        addi r4, r4, 4      |
        blt  r10,r5, loop   |
end:
```

- Control Flow Graph
  - Shows possible paths of control flow through basic blocks

Control Dependences




```
main:
        addi r2, r0, A      |
        addi r3, r0, B      |
        addi r4, r0, C      |   BB 1
        addi r5, r0, N      |
        add  r10,r0, r0     |
        bge  r10,r5, end    |
loop:                       |
        lw   r20, 0(r2)     |
        lw   r21, 0(r3)     |   BB 2
        bge  r20,r21,T1     |
        sw   r21, 0(r4)     |   BB 3
        b    T2             |
T1:                         |
        sw   r20, 0(r4)     |   BB 4
T2:                         |
        addi r10,r10,1      |
        addi r2, r2, 4      |
        addi r3, r3, 4      |   BB 5
        addi r4, r4, 4      |
        blt  r10,r5, loop   |
end:
```

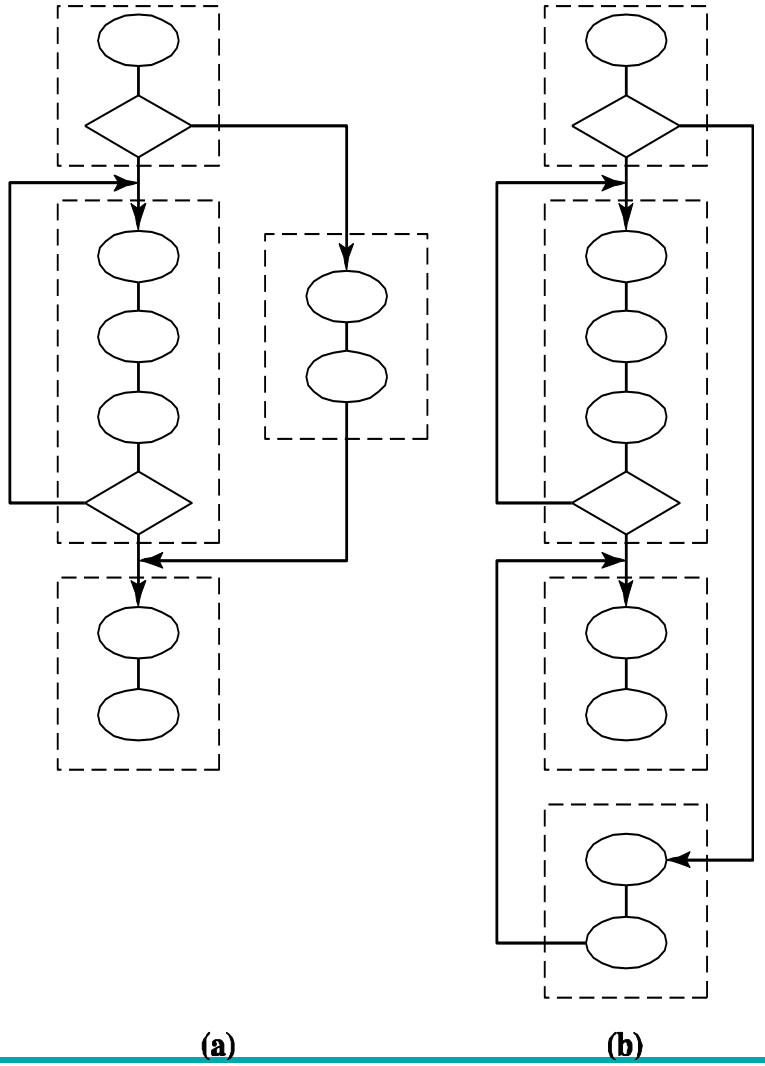
- Control Dependence
  - Node B is CD on Node A if A determines whether B executes
  - If path 1 from A to exit includes B, and path 2 does not, then B is control-dependent on A

Program Control Flow

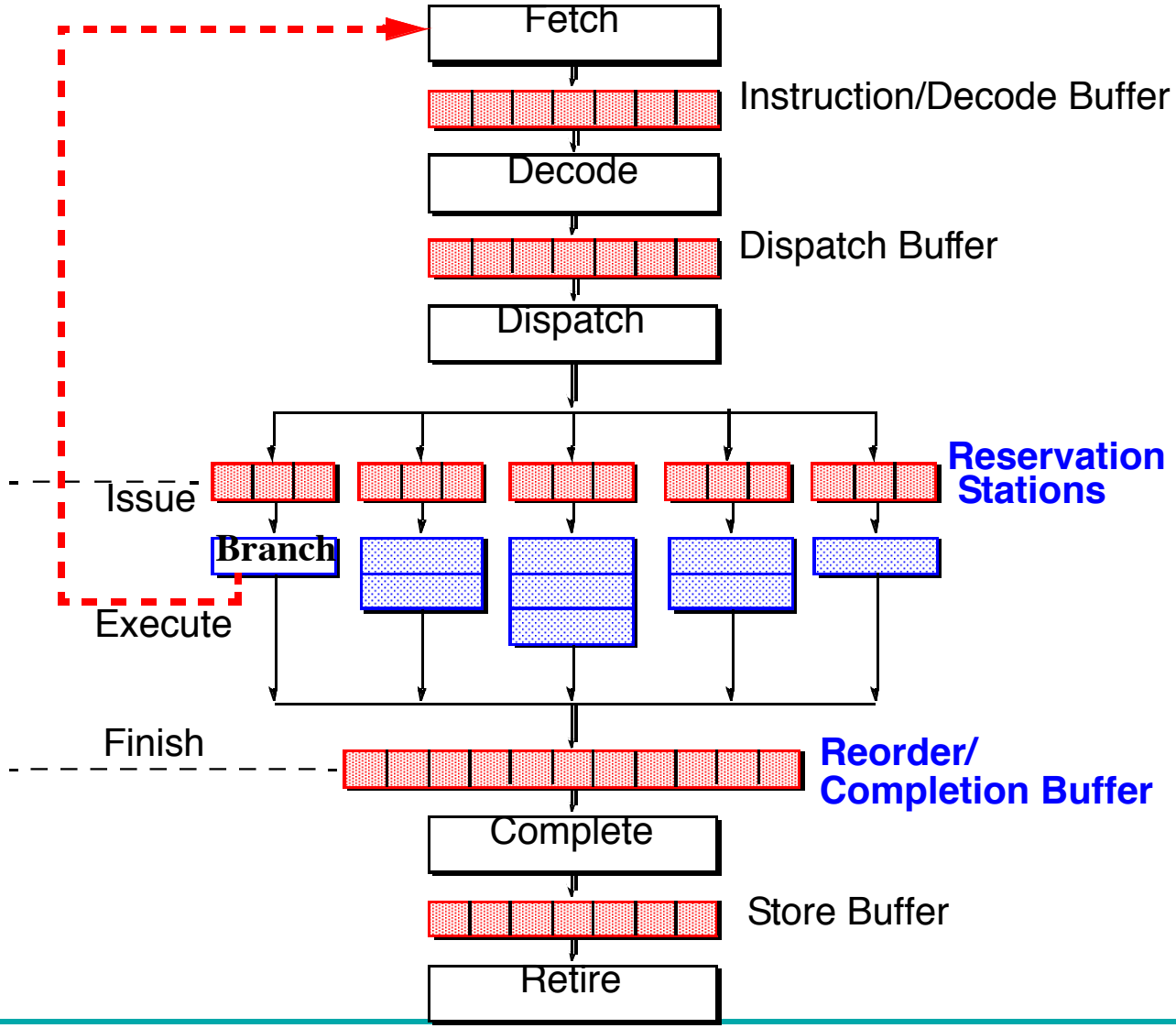
- Implicit Sequential Control Flow
  - Static Program Representation
    - Control Flow Graph (CFG)
    - Nodes = basic blocks
    - Edges = Control flow transfers
  - Physical Program Layout
    - Mapping of CFG to linear program memory
    - Implied sequential control flow
  - Dynamic Program Execution
    - Traversal of the CFG nodes and edges (e.g. loops)
    - Traversal dictated by branch conditions
  - Dynamic Control Flow
    - Deviates from sequential control flow
    - Disrupts sequential fetching
    - Can stall IF stage and reduce I-fetch bandwidth

Program Control Flow

- Dynamic traversal of static CFG
- Mapping CFG to linear memory



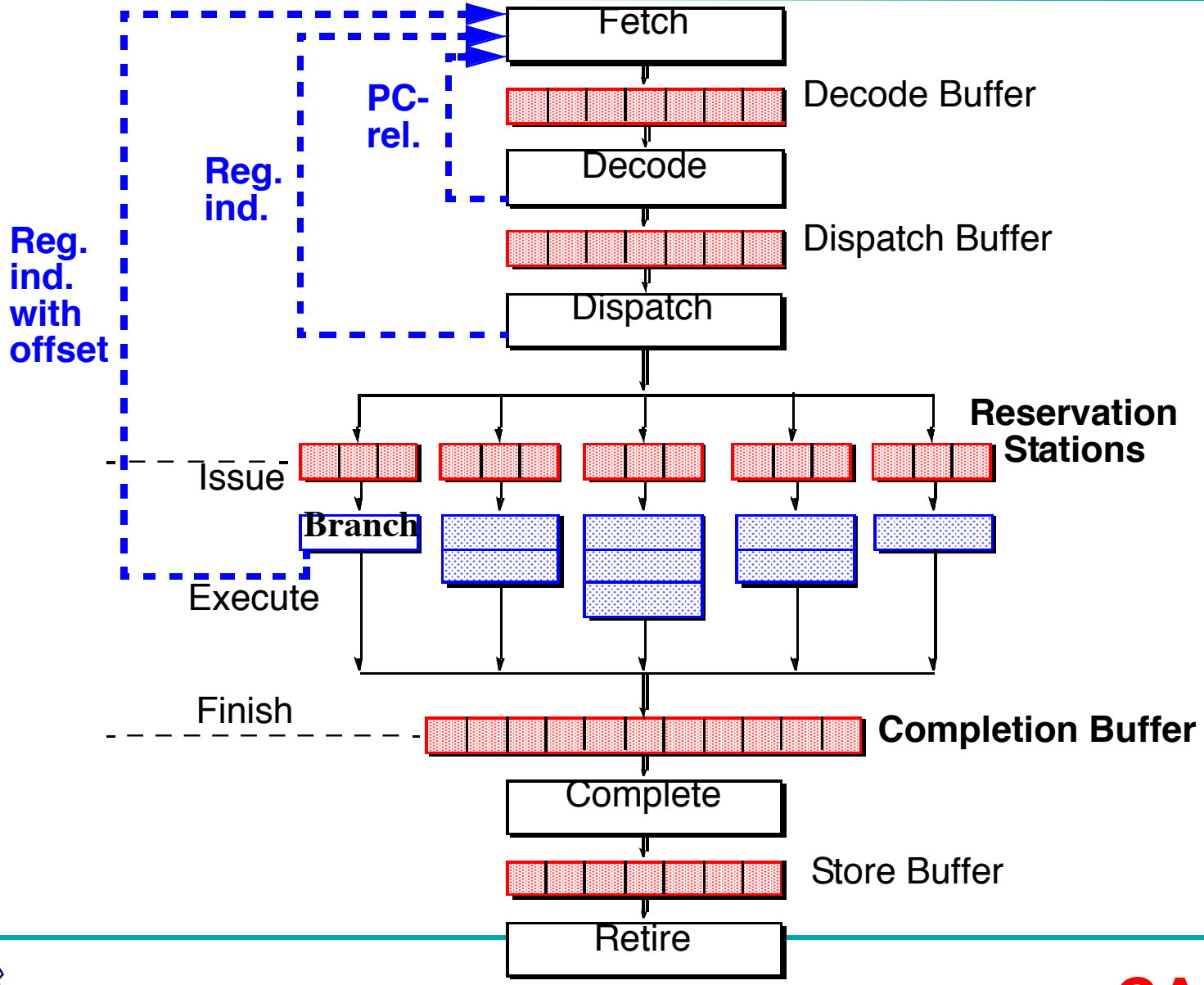
Disruption of Sequential Control Flow



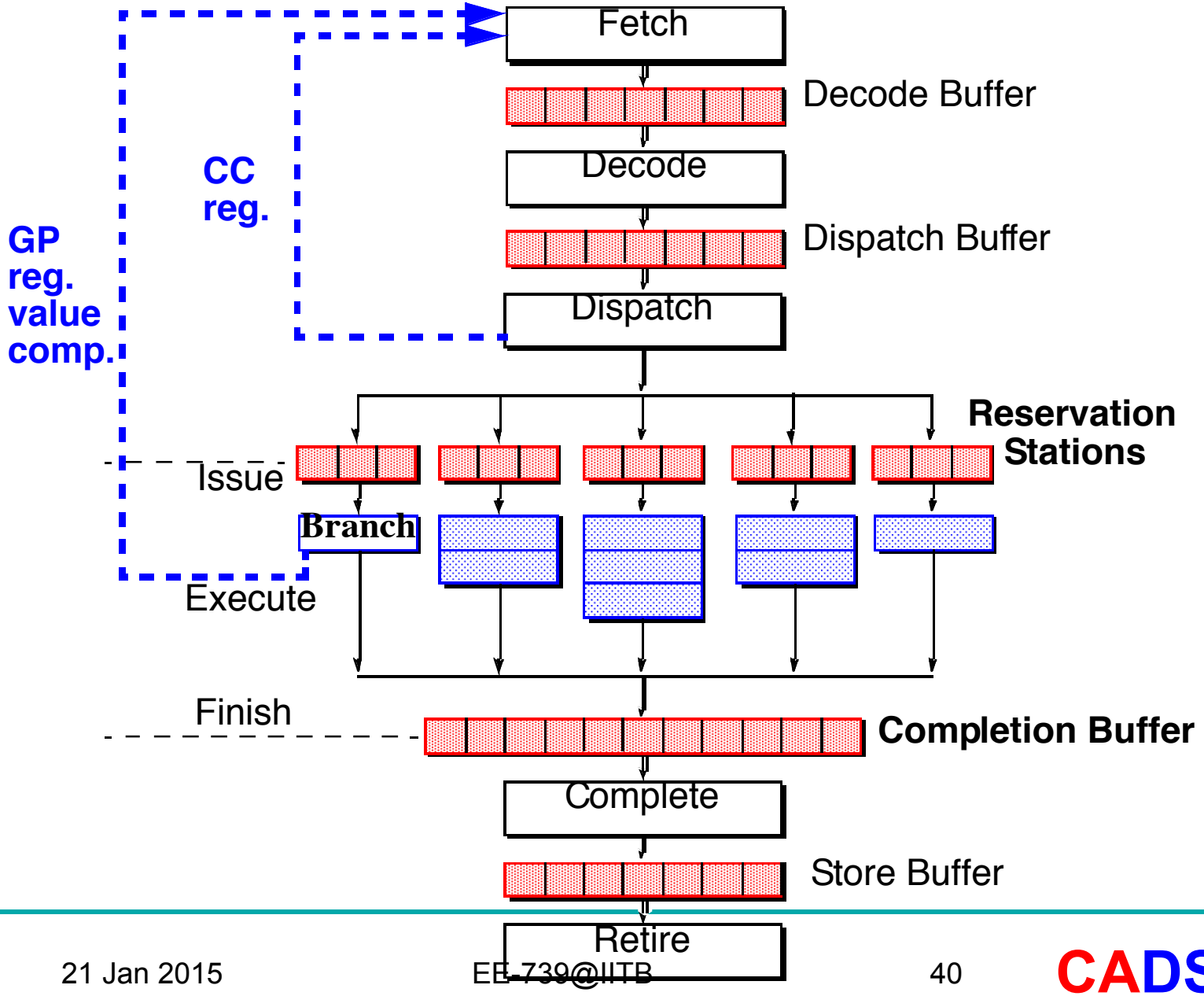
Branch Prediction

- Target address generation → Target Speculation
  - Access register:
    - PC, General purpose register, Link register
  - Perform calculation:
    - +/- offset, autoincrement, autodecrement
- Condition resolution → Condition speculation
  - Access register:
    - Condition code register, General purpose register
  - Perform calculation:
    - Comparison of data register(s)

Target Address Generation



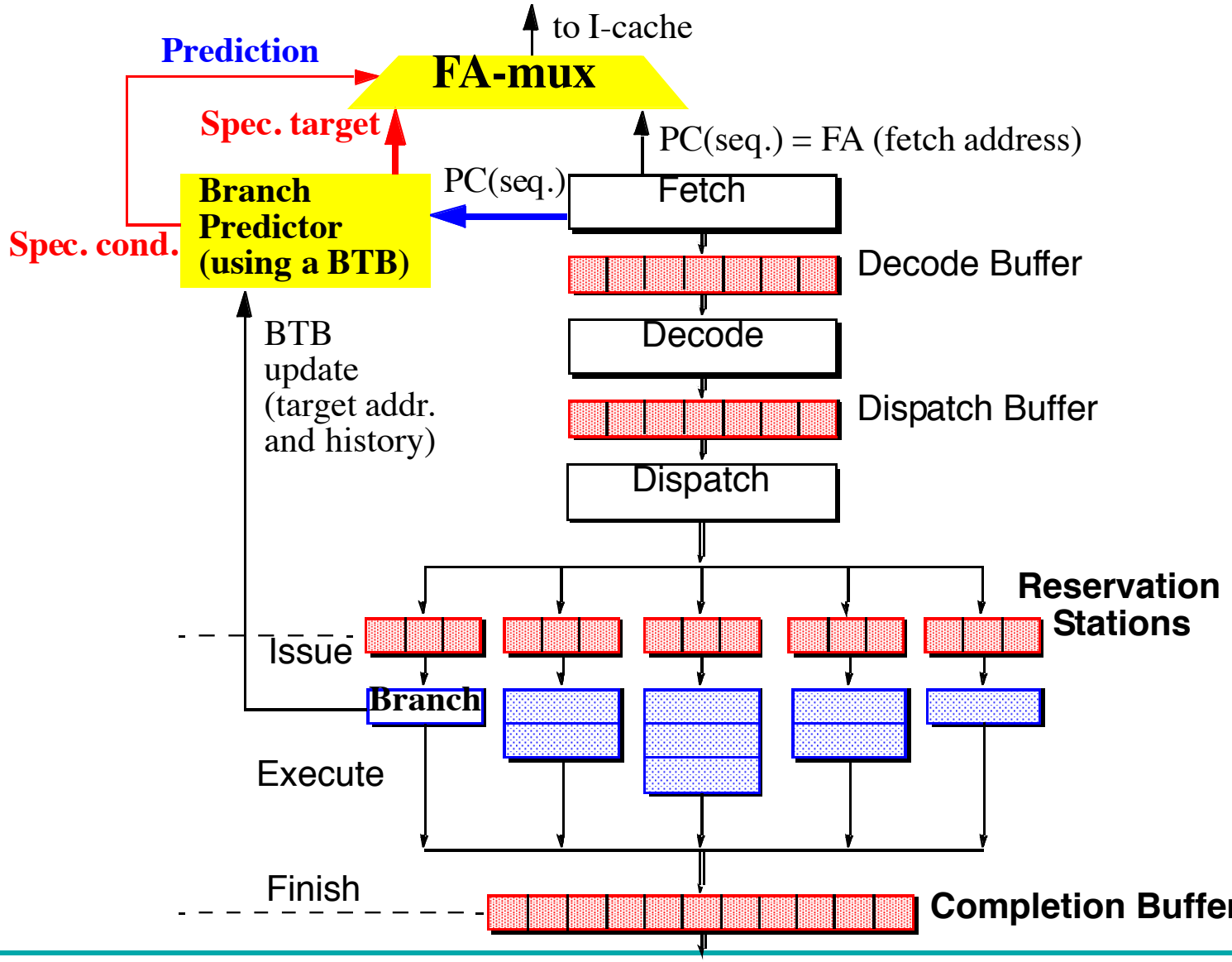
Condition Resolution



Dynamic Branch Prediction

- Main advantages:
  - Learn branch behavior autonomously
    - No compiler analysis, heuristics, or profiling
  - Adapt to changing branch behavior
    - Program phase changes branch behavior
- First proposed in 1979-1980
  - US Patent #4,370,711, Branch predictor using random access memory, James. E. Smith
- Continually refined since then

Branch Instruction Speculation



Thank You