CSE 141:
Introduction to Computer Architecture

Branch Prediction 2 /
Hardware Multithreading

Jishen Zhao (https://cseweb.ucsd.edu/~jzhao/)

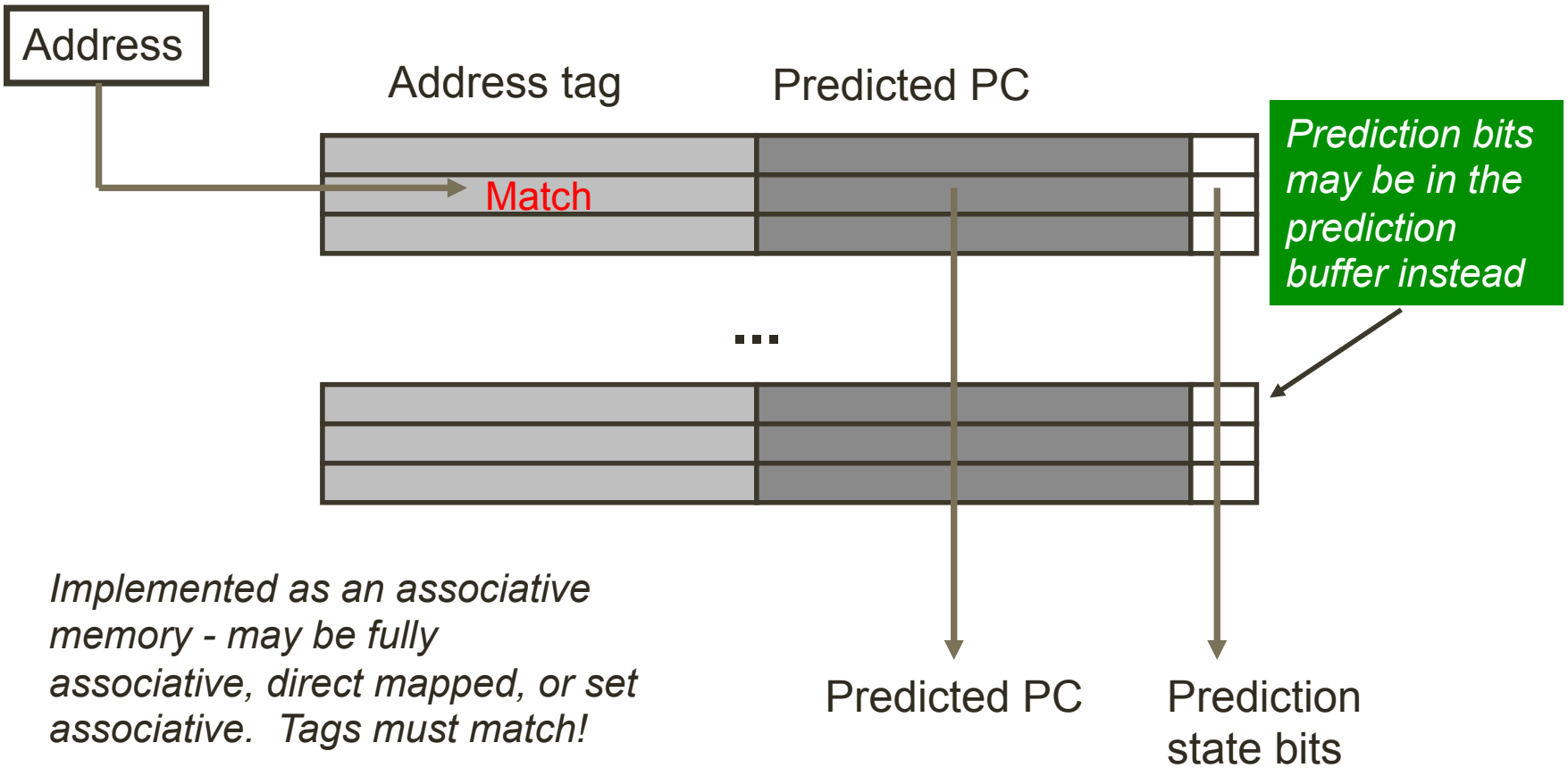
[Adapted in part from Dean Tullsen, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

BHT implementation

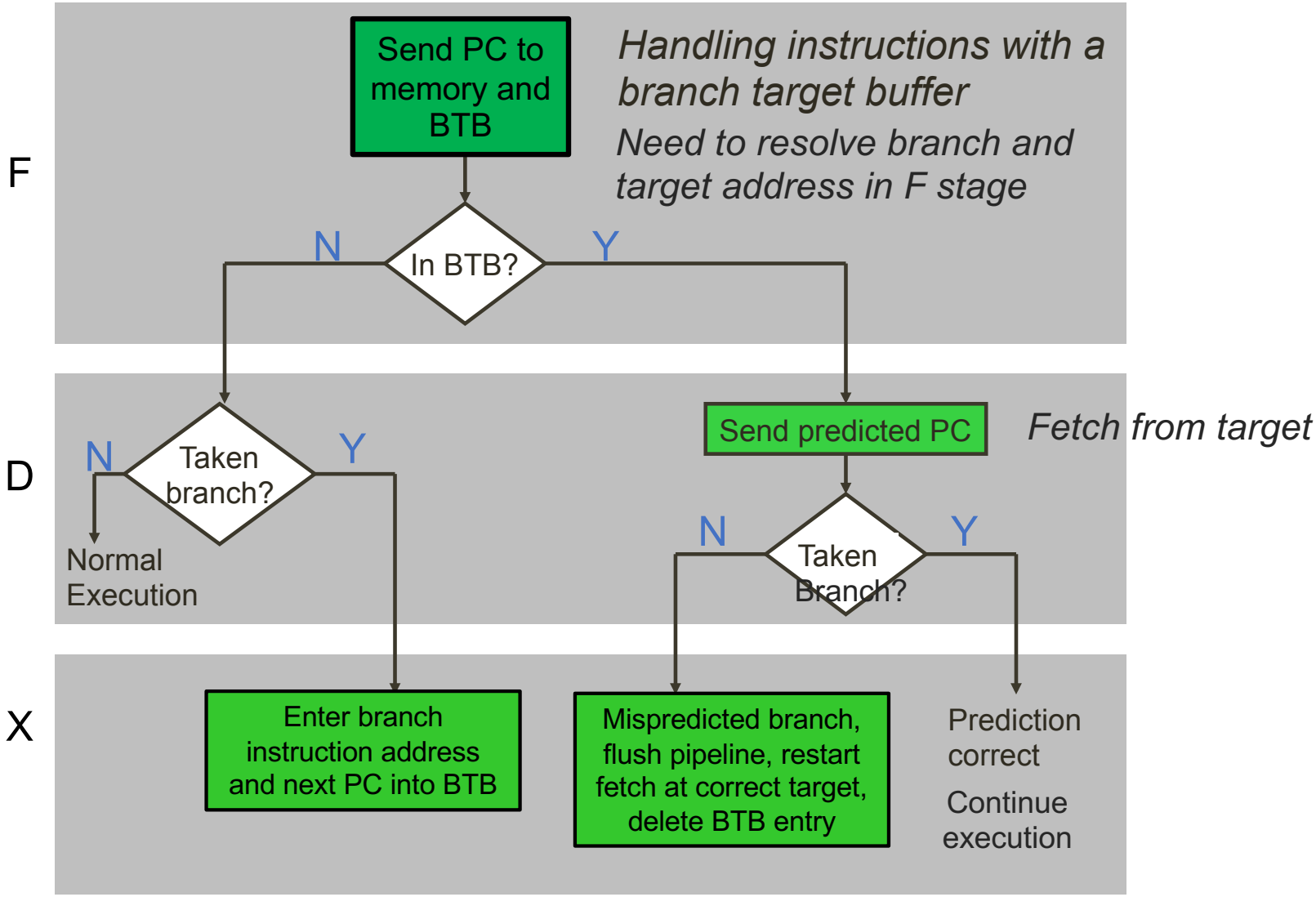
- A small cache accessed during F stage
- Counter (two bits) attached to each cache line

- If branch predicted taken, fetch begins from target *as soon as target PC known*

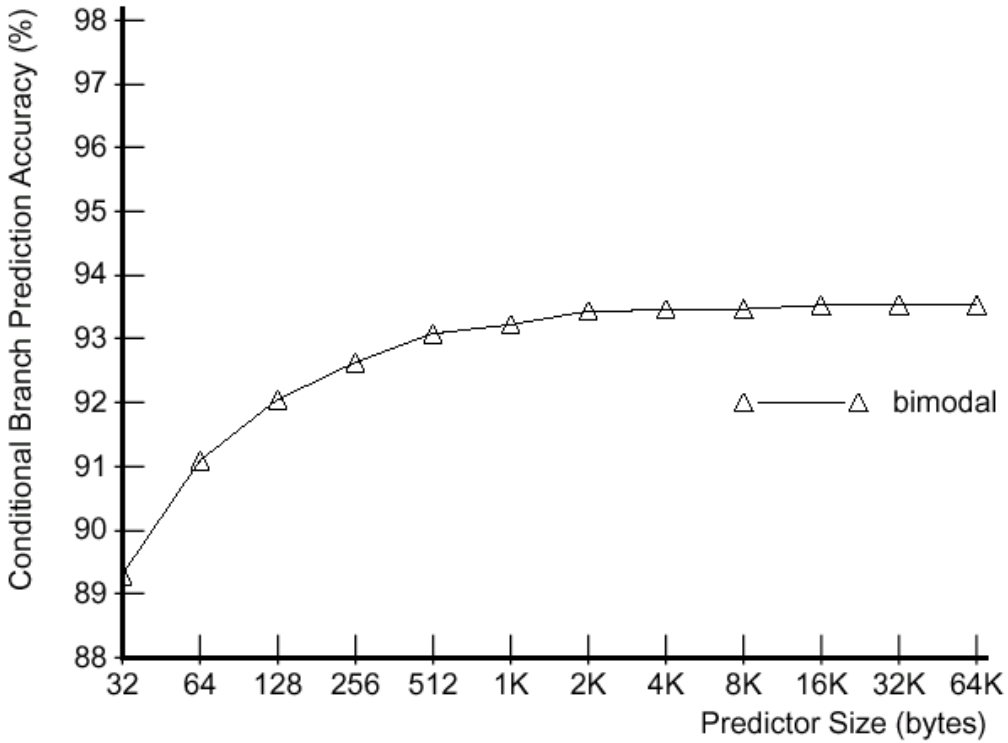
BTB implementation



BTB implementation

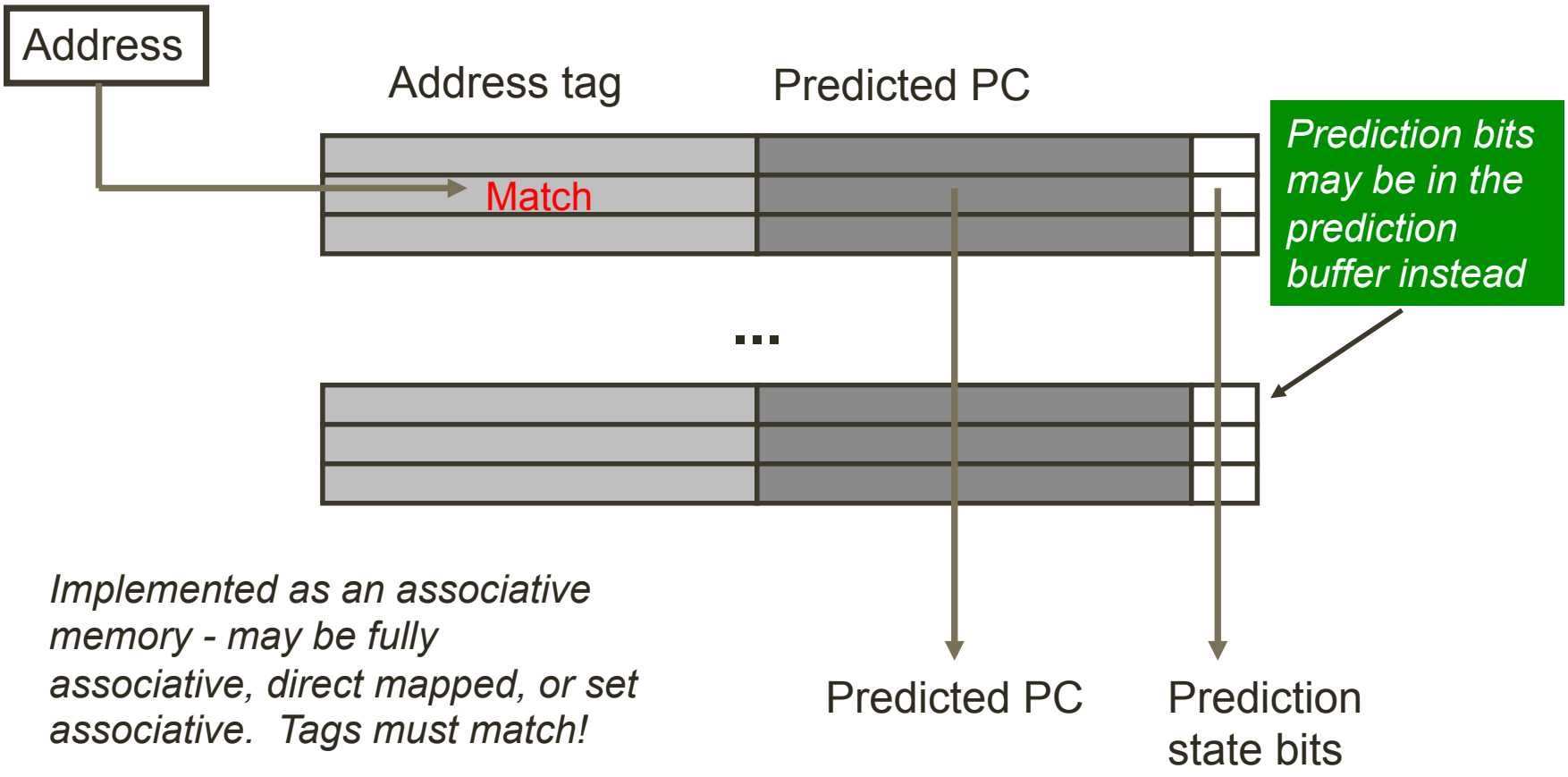


Two-bit prediction accuracy

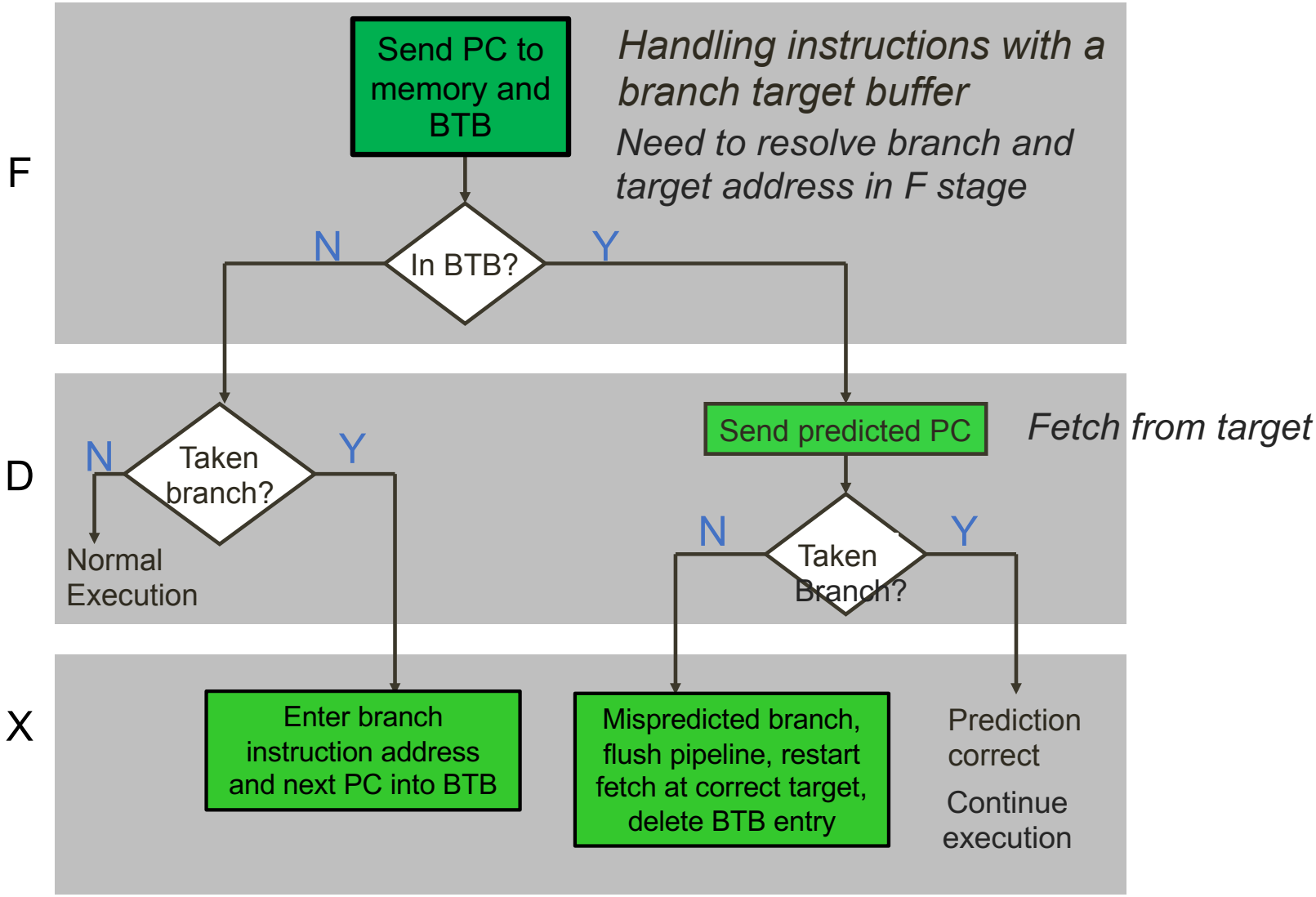


Prediction accuracy for SPEC'89

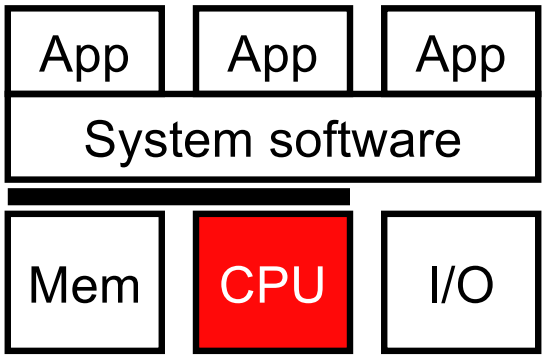
Review: BTB implementation



Review: BTB implementation



Summary: Pipelining



- Single-cycle datapaths ✓
- Latency vs. throughput & performance ✓
- Basic pipelining ✓
- Data hazards ✓
  - Bypassing
  - Load-use stalling
- Pipelined multi-cycle operations ✓
- Control hazards ✓
  - Branch prediction

Any other methods on handling data and control hazards?

Do something else: Hardware multithreading

An example: Fine-grained Multithreading

Thread 1

```
     bne R3, R6, L3
     addi R1<- R1, #1
L3:  add R1<- R2, R4
```

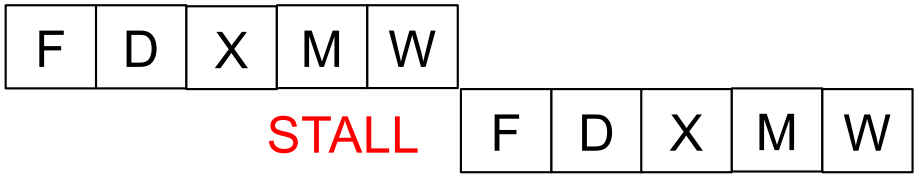
Thread 2

```
add R10<- R5, R8
addi R10<- R10, #1
```

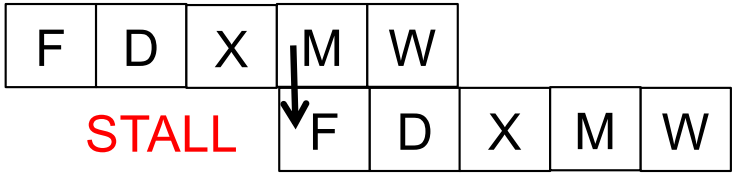
Other threads…

Has no dependency

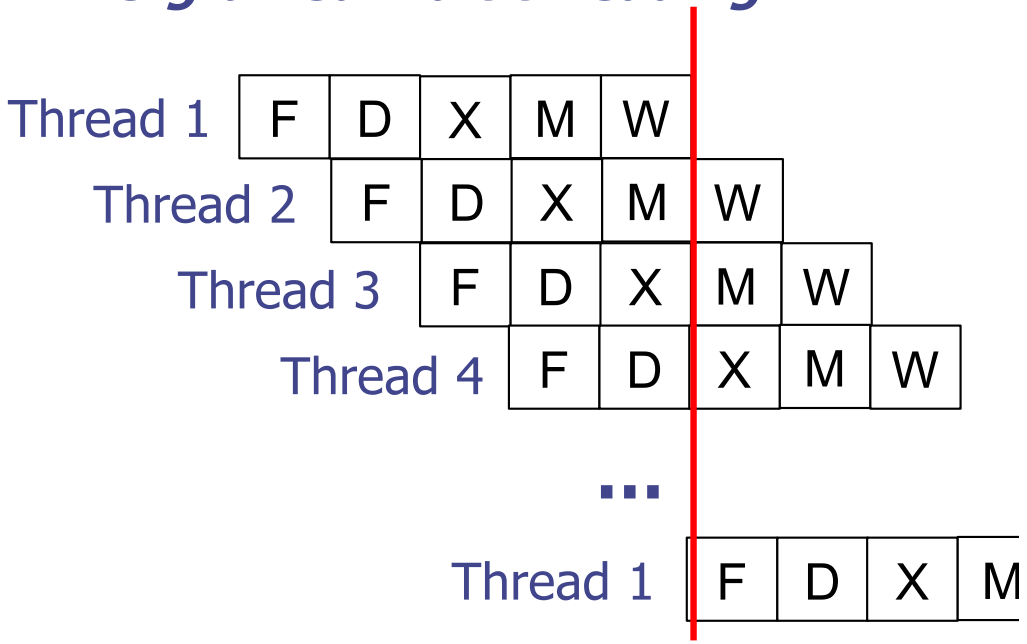
Thread 1



Even if MF forwarding:



Fine-grained multithreading:



Pros and cons

- Advantages
  - + No need for dependency checking between instructions
    (only one instruction in pipeline from a single thread)
  - + No need for branch prediction logic
  - + Otherwise-stall cycles used for executing useful instructions from different threads
  - + Improved system throughput and pipeline utilization

- Disadvantages
  - - Extra hardware complexity: multiple hardware contexts (PCs, register files, …), thread selection logic
  - - Reduced single thread performance (one instruction fetched every N cycles from the same thread)
  - - Resource contention between threads in caches and memory
  - - Some dependency checking logic *between* threads remains (load/store)

Uniprocessor Concurrency

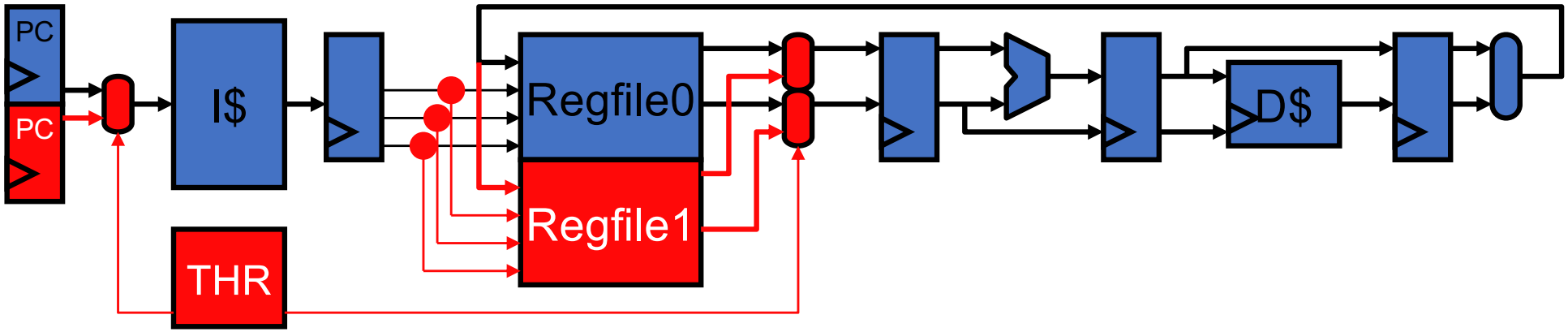
- Programmer explicitly creates multiple threads

- A "thread switch" can occur at any time
  - Pre-emptive multithreading by OS

- Common uses:
  - Handling user interaction (GUI programming)
  - Handling I/O latency (send network message, wait for response)
  - **Expressing parallel work via Thread-Level Parallelism (TLP)**

Hardware Multithreading

- **Not** the same as software multithreading!
- A **hardware thread** is a sequential stream of insns
  - could be a software thread or a single-threaded process



- **Hardware Multithreading (MT)**
  - Multiple hardware threads dynamically share a single pipeline
  - Replicate only per-thread structures: program counter & registers
  - Hardware interleaves instructions

Hardware Multithreading

- Why use hw multithreading?
  - + **Multithreading improves utilization and throughput**
    - Single programs utilize <50% of pipeline (branch, cache miss)
    - allow insns from different hw threads in pipeline at once
  - **Multithreading does not improve single-thread performance**
    - Individual threads run as fast or even slower
  - **Coarse-grain MT**: switch on cache misses Why (will discuss later)?
  - **Simultaneous MT**: no explicit switching, fine-grain interleaving
    - Simultaneous multithreading (SMT): Issue multiple instructions from multiple threads in one cycle. The processor must be superscalar to do so.
    - **Intel's "hyperthreading"**

process A
thread 1
thread 2

process B
(just one thread)

Hardware Multithreading