STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Ira A. Fulton Schools of Engineering
Arizona State University

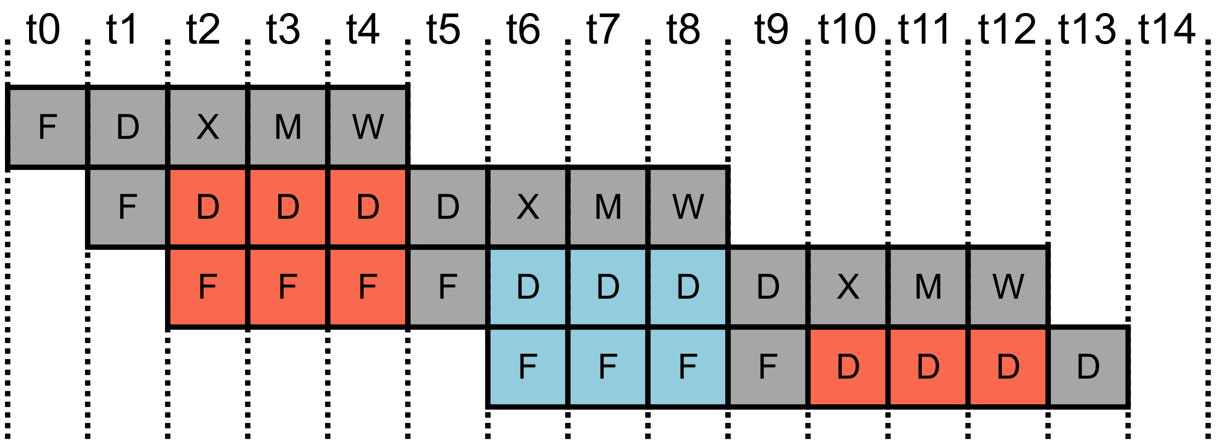
CSE 520
Computer Architecture II

Multithreading Architectures

Prof. Michel A. Kinsy

Pipeline Hazards

```
LW      x1, 0(x2)
LW      x5, 12(x1)
ADDI    x5, x5,#12
SW      12(x1), x5
```

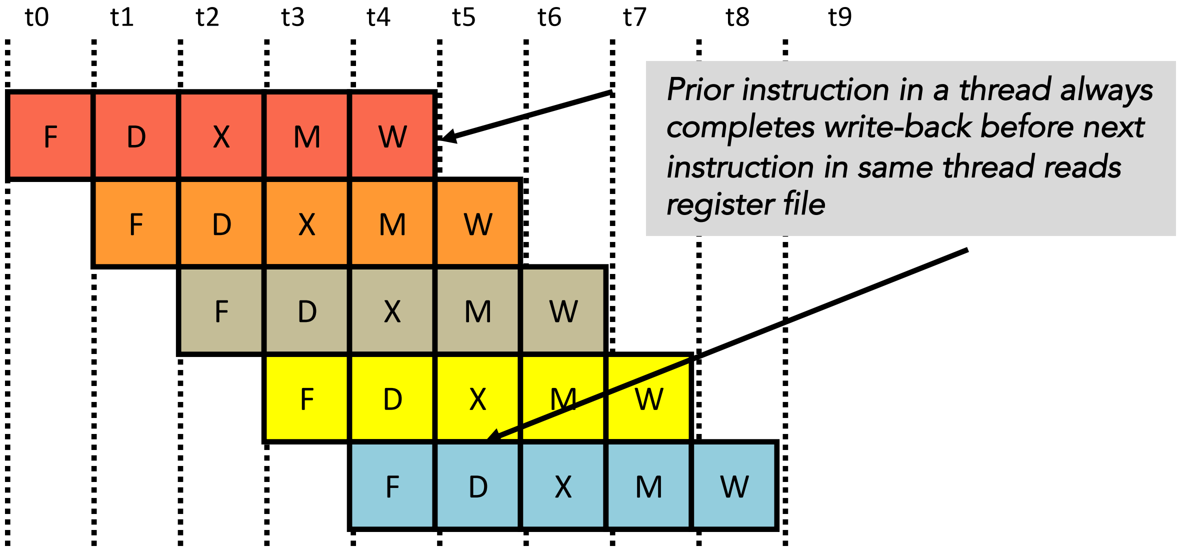


- Each instruction may depend on the next
  - What can be done to cope with this?
- Even bypassing does not eliminate all delays

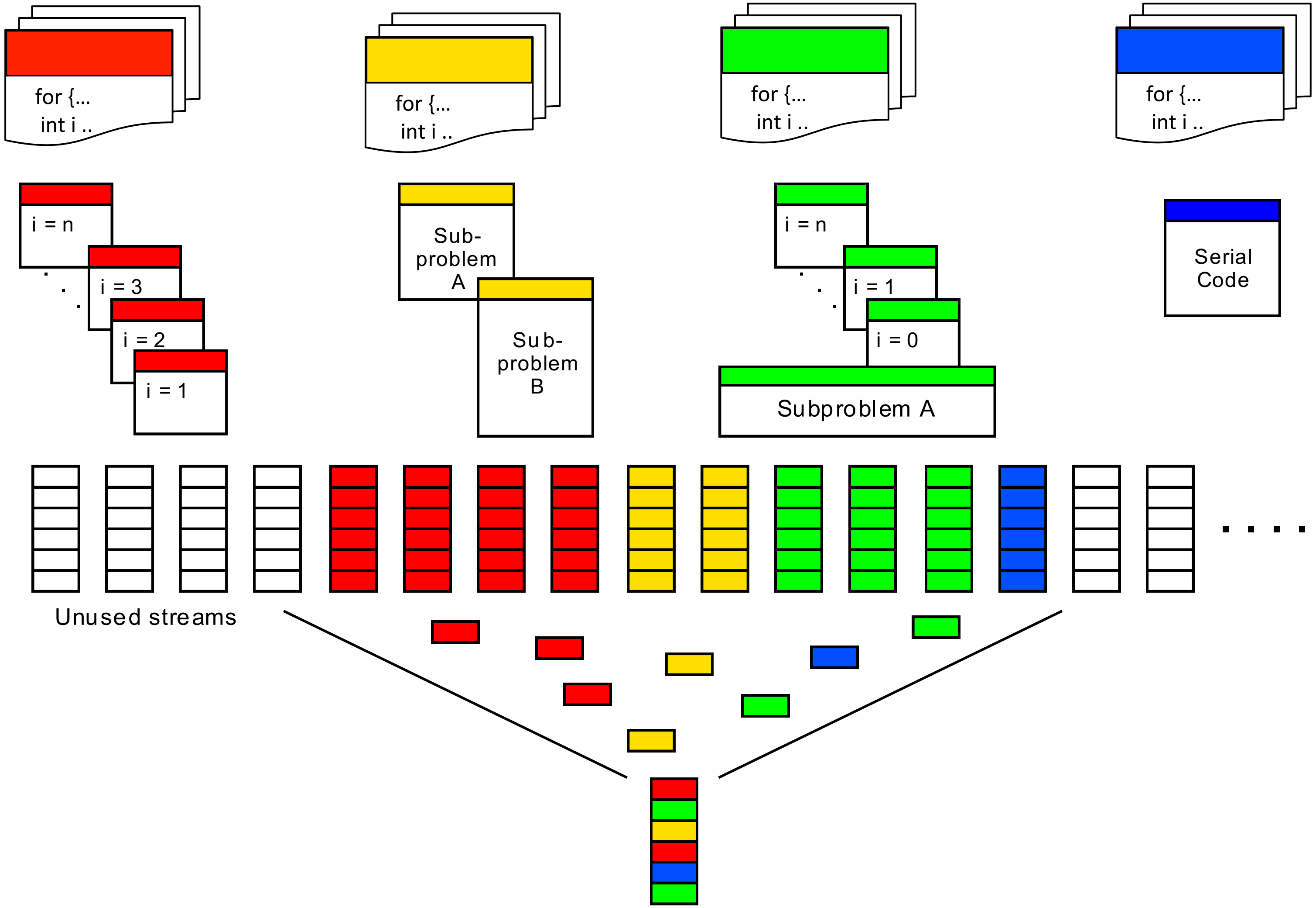
Multithreading

- How can we guarantee no dependencies between instructions in a pipeline?
- One way is to interleave execution of instructions from different program threads on same pipeline
  - *Interleave 4 threads, T1-T4, on non-bypassed 5-stage pipe*

T1: LW r1, 0(r2)
T2: ADD r7, r1, r4
T3: XORI r5, r4, #12
T4: SW 0(r7), r5
T1: LW r5, 12(r1)



Multithreading



Program running in Parallel

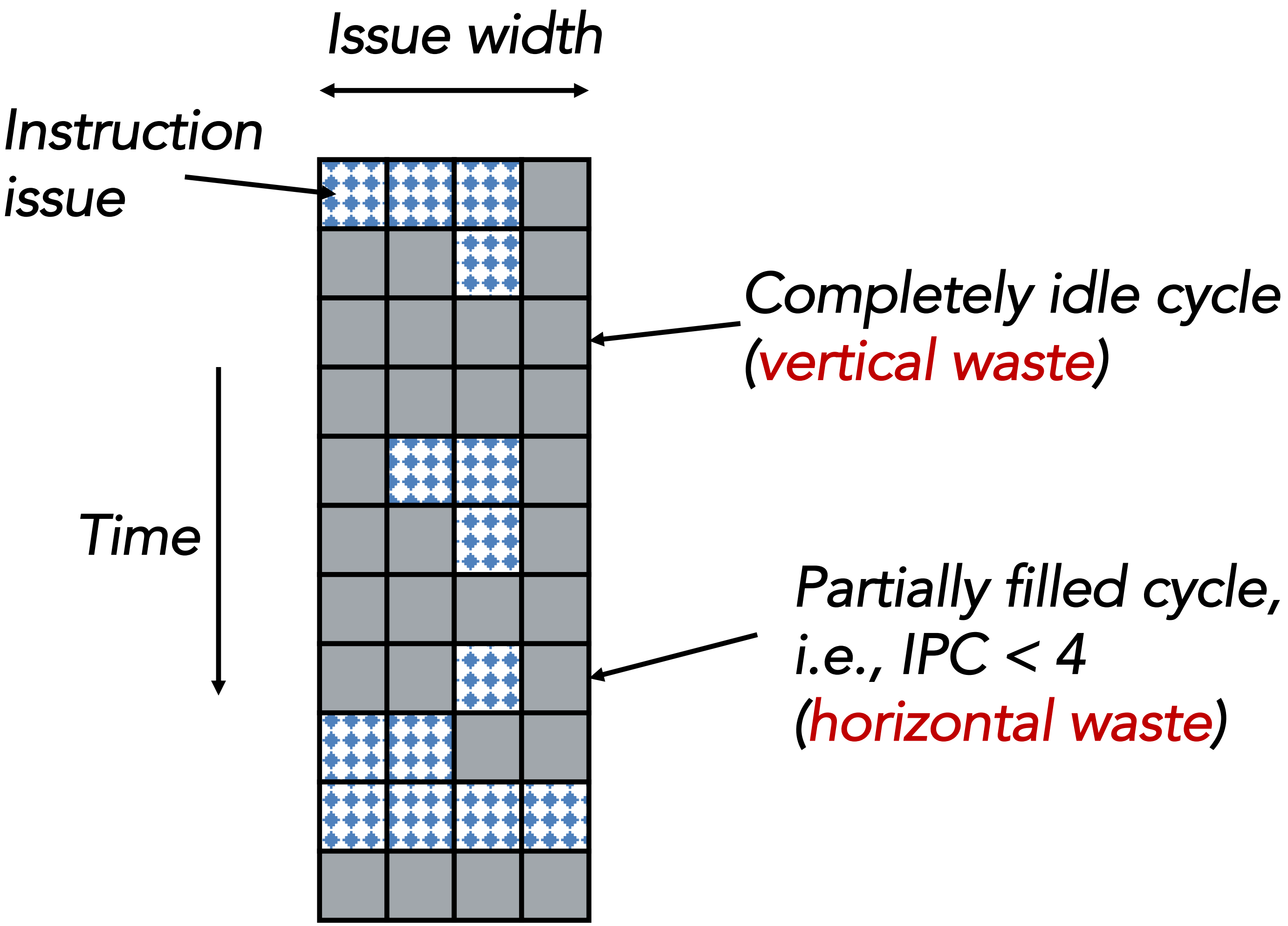
Threads of computation

Supported hardware threads (e.g., 16)

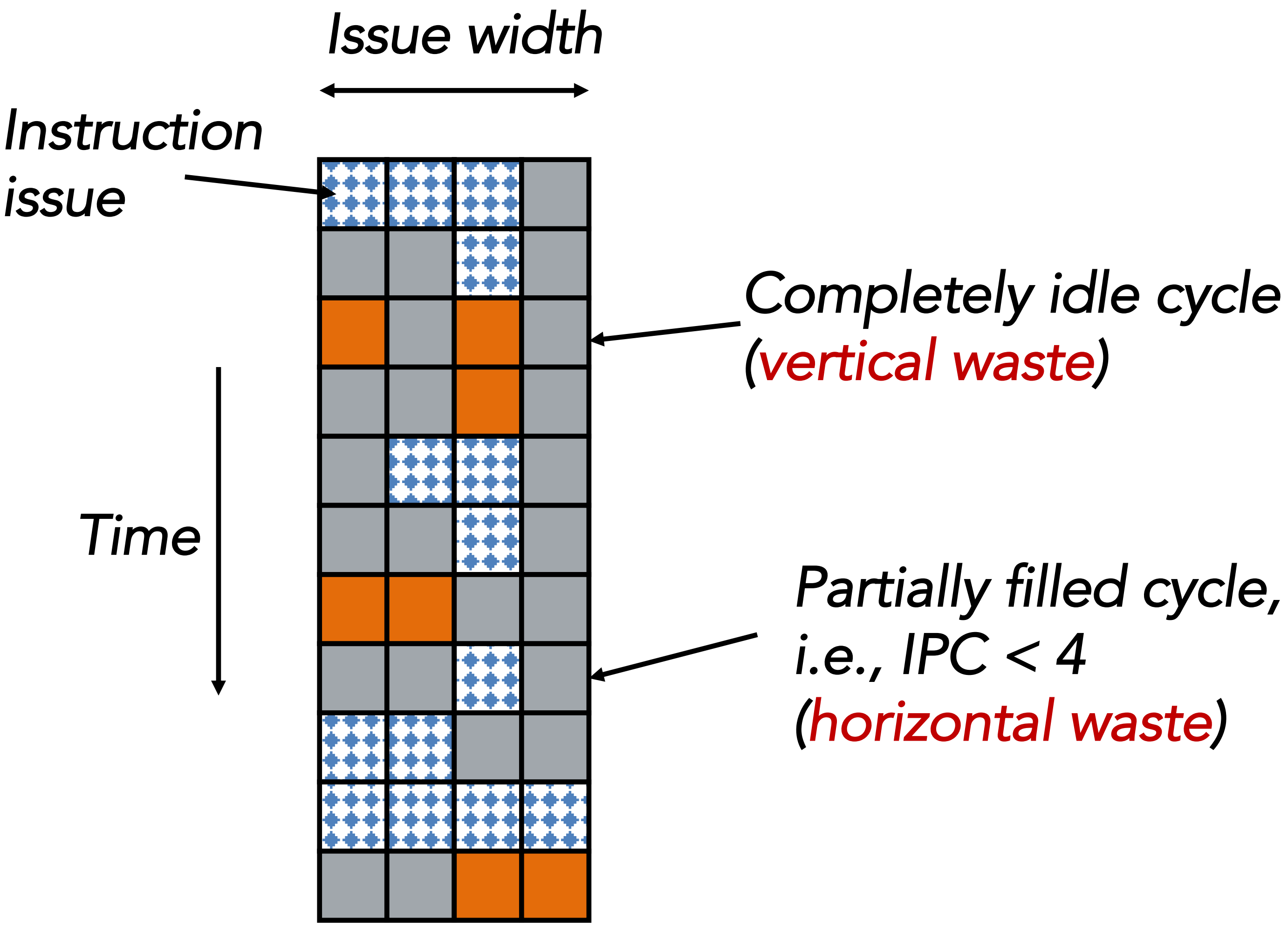
Instruction ready pool

Interleaving of instruction executions

Superscalar Machine Efficiency

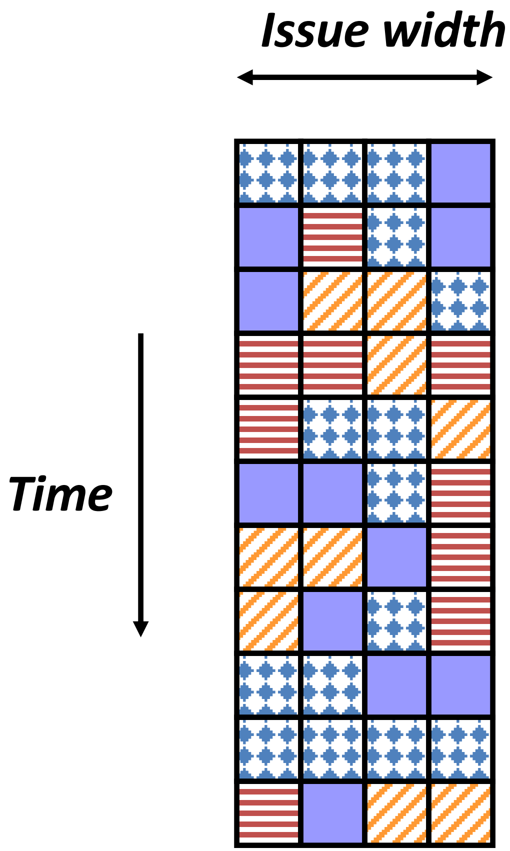


Vertical Multithreading



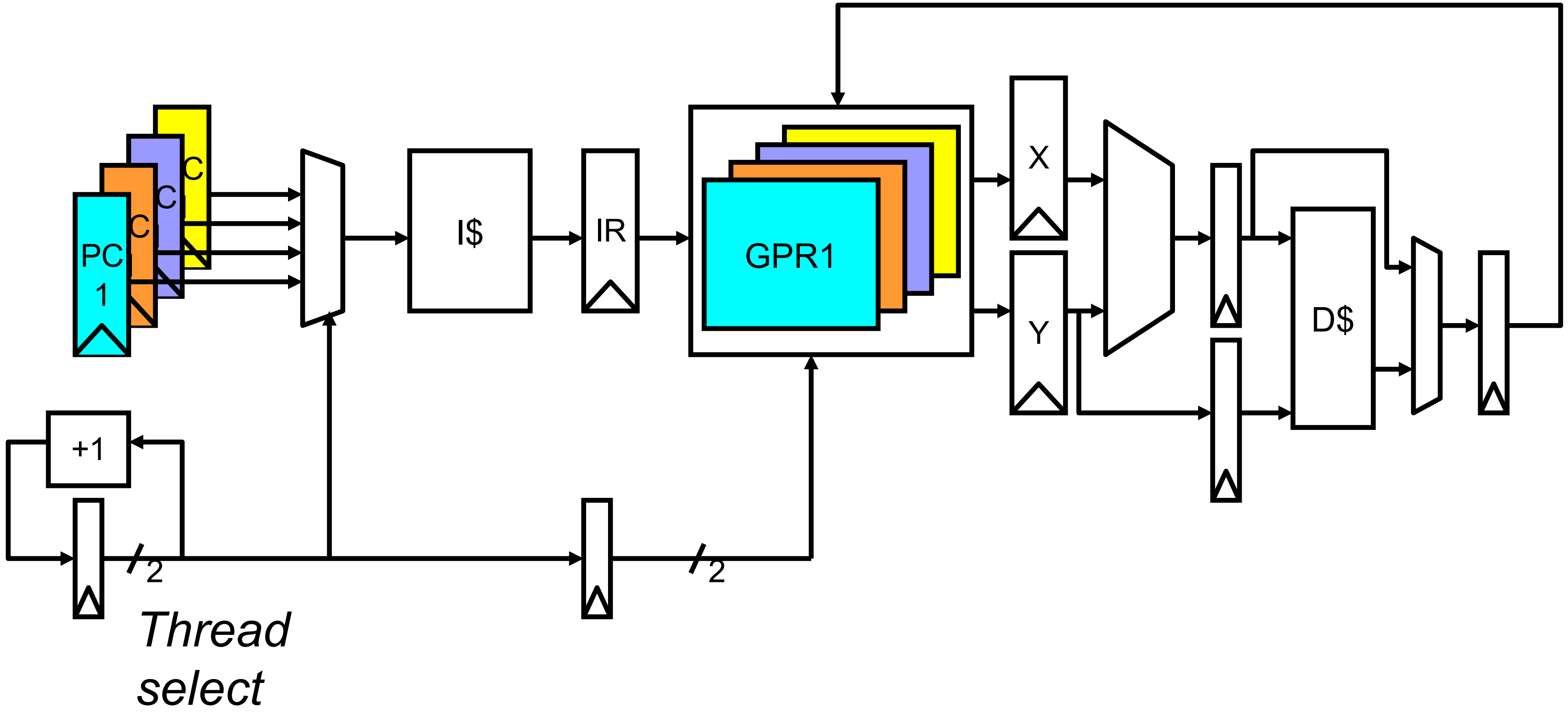
Ideal Superscalar Multithreading

- Have multiple thread contexts in a single processor
- Interleave multiple threads to multiple issue slots with no restrictions
- Tullsen, Eggers, Levy, UW, 1995



Simple Multithreaded Pipeline

- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage



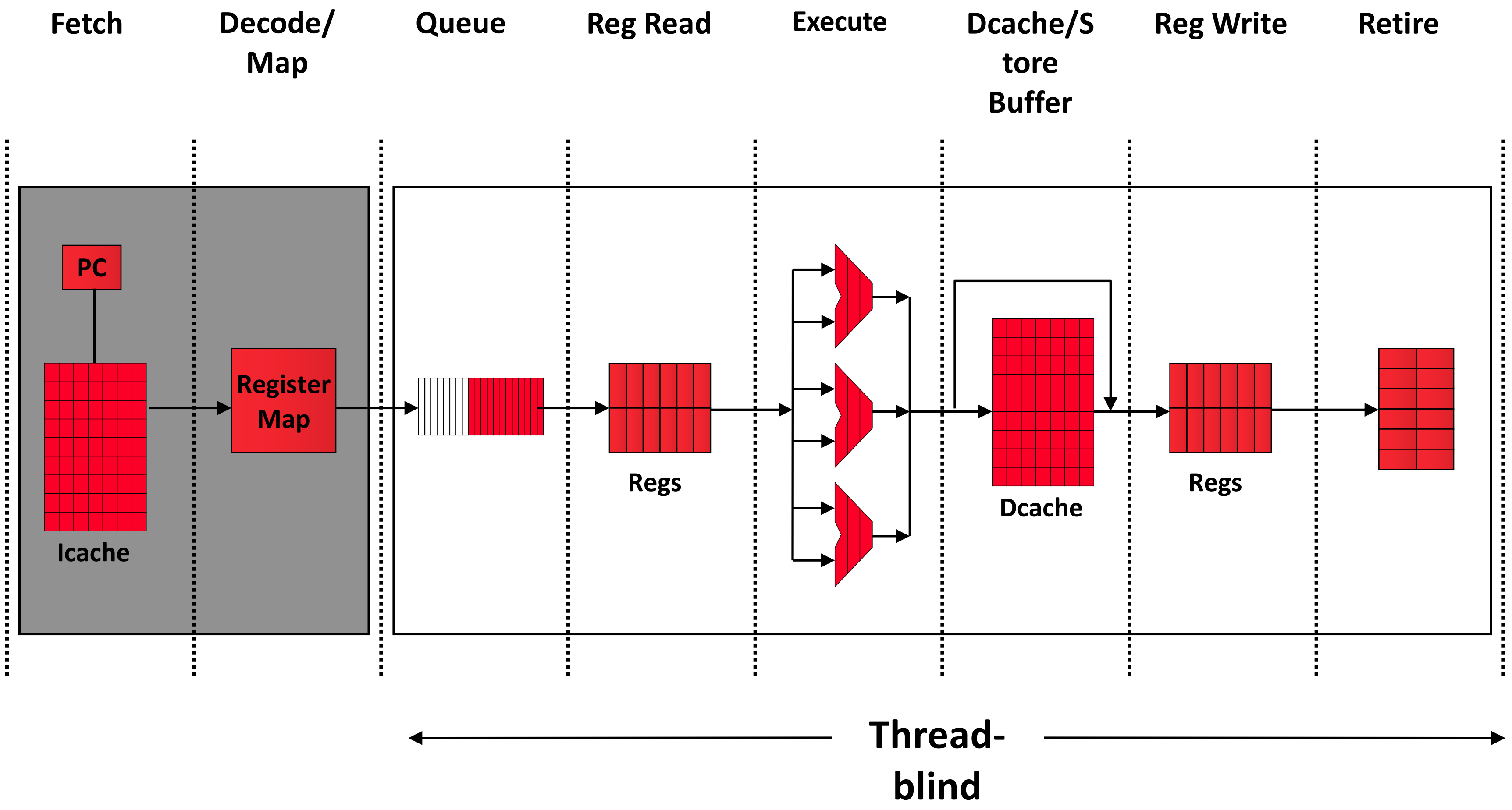
Multithreading Implementation

- Each thread requires its own user state
  - PC
  - GPRs
- Each thread needs its own system state
  - Virtual memory page table base register
  - Exception handling registers
- Transparency
  - Appears to software (including OS) as multiple, albeit slower, CPUs

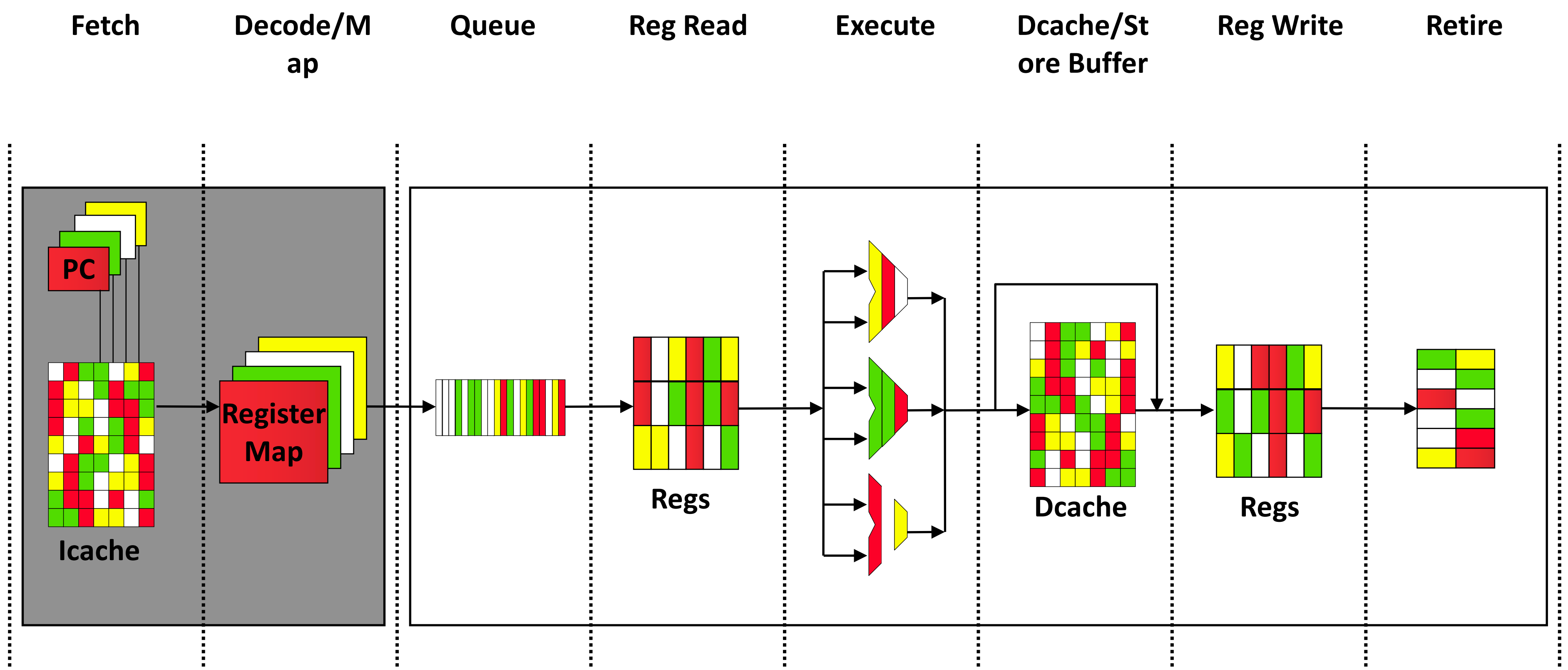
Hardware Multithreading

- Thread
  - Instruction stream with state (registers and memory)
  - Register state is also called "thread context"
- Threads could be part of the same process (program) or from different programs
  - Threads in the same program share the same address space
- Number of supported threads could be greater than number of hardware threads
  - When a new thread needs to be executed, old thread's context in hardware written back to memory and new thread's context loaded

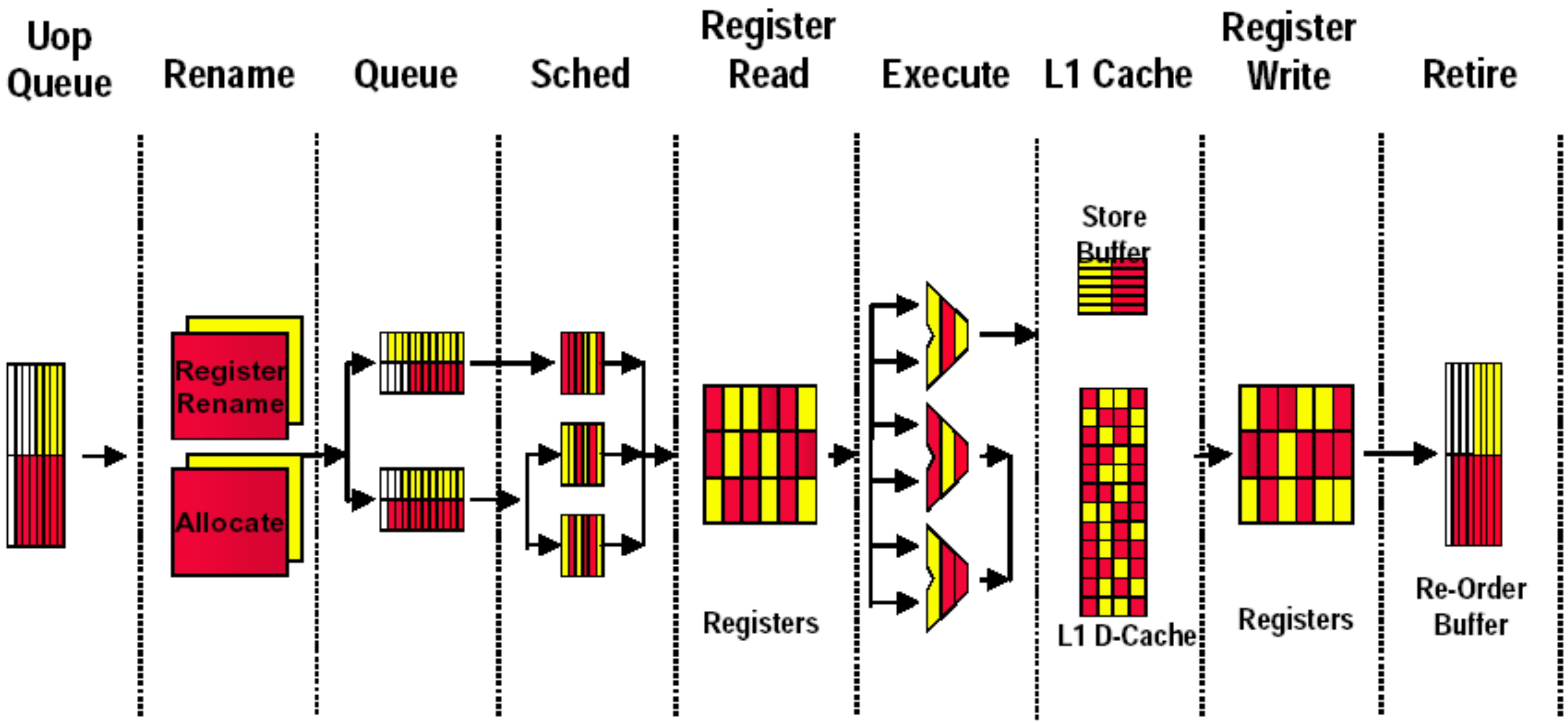
Basic Out-of-order Pipeline



Hardware Multithread Pipeline



Pentium-4 Hyperthreading (2002)



Pentium-4 Hyperthreading (2002)

- First commercial SMT design (2-way SMT)
  - Hyperthreading == simultaneous multithreading (**SMT**)
  - Logical processors share nearly all resources of the physical processor
  - Caches, execution units, branch predictors
- Die area overhead of hyperthreading ~ 5%
- When one logical processor is stalled, the other can make progress
  - No logical processor can use all entries in queues when two threads are active

Hardware Multithreading

- Benefit
  - Latency tolerance
  - Better hardware utilization
  - Better than software-level context switch
- Cost
  - Requires multiple thread contexts to be implemented in hardware (area, power, latency cost)
  - Usually reduced single-thread performance
  - Resource sharing contention
  - Switching penalty
    - Can be reduced with additional hardware

Thread Scheduling Policies

- Software-controlled interleave (TI ASC PPUs, 1971)
  - OS allocates S pipeline slots amongst N threads
  - Hardware performs fixed interleave over S slots, executing whichever thread is in that slot
- Hardware-controlled thread scheduling (HEP, 1982)
  - Hardware keeps track of which threads are ready to go
  - Picks next thread to execute based on hardware priority scheme
- Fixed interleave (CDC 6600 PPUs, 1965)
  - Each of N threads executes one instruction every N cycles
  - If thread not ready to go in its slot, insert pipeline bubble

Types of Multithreading

- When the hardware executes from those hardware contexts determines the granularity of multithreading
  - Fine-grained
    - Cycle by cycle
  - Coarse-grained
    - Switch on event (e.g., cache miss)
    - Schedule-based switching
  - Simultaneous
    - Instructions from multiple threads executed concurrently in the same cycle

Fine-grained Multithreading

- Advantages
  - Simpler to implement, can eliminate dependency checking, branch prediction logic completely
  - Switching need not have any performance overhead (i.e., dead cycles)
  - Coarse-grained requires a pipeline flush or a lot of hardware to save pipeline state
    - Higher performance overhead with deep pipelines and large windows
- Disadvantages
  - Low single thread performance: each thread gets 1/Nth of the bandwidth of the pipeline

Fine-grained Multithreading

- Switch to another thread every cycle such that no two instructions from the thread are in the pipeline concurrently
- Advantages
  - Simpler to implement, can eliminate dependency checking, branch prediction logic completely
  - Switching need not have any performance overhead (i.e., dead cycles)
- Disadvantages
  - Low single thread performance: each thread gets 1/Nth of the bandwidth of the pipeline

Tera MTA Fine-grained Multithreading

- Up to 256 processors
- Up to 128 active threads per processor
- Processors and memory modules populate a sparse 3D torus interconnection fabric
- Flat, shared main memory
  - No data cache
  - Sustains one main memory access per cycle per processor
- GaAs logic in prototype, 1KW/processor @ 260MHz
  - CMOS version, MTA-2, 50W/processor

Coarse-grained Multithreading

- When a thread is stalled due to some event, switch to a different hardware context
  - Switch-on-event multithreading
- Possible stall events
  - Cache misses
  - Synchronization events (e.g., load an empty location)
  - FP operations
- Requires a pipeline flush or a lot of hardware to save pipeline state
  - Higher performance overhead with deep pipelines and large windows
- Resource sharing in space and time always causes fairness considerations
  - How much progress each thread makes
    - When and for how long to switch?
    - What is the switching overhead vs. benefit?
    - Where to store the contexts?
    - How does the hardware scheduler interact with the software scheduler for fairness?

IBM RS64-IV

- 4-way superscalar, in-order, 5-stage pipeline
- Two hardware contexts
- On an L2 cache miss
  - Flush pipeline
  - Switch to the other thread

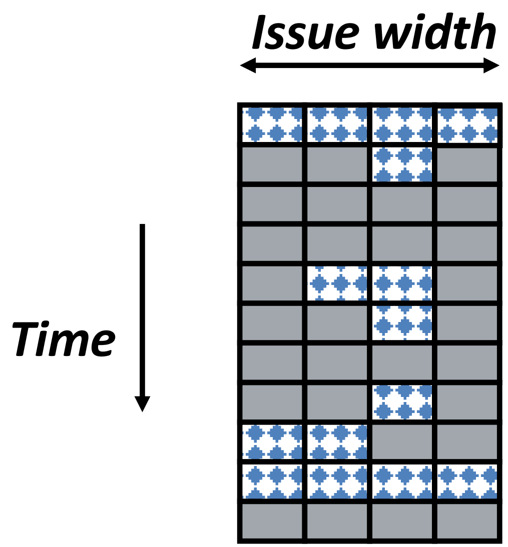
- Considerations
  - Memory latency vs. thread switch overhead
  - Short pipeline, in-order execution (small instruction window) reduces the overhead of switching

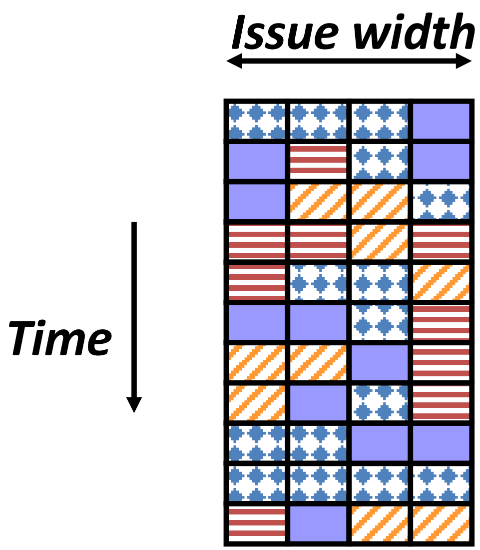
Simultaneous Multithreading

- Fine-grained and coarse-grained multithreading can start execution of instructions from only a single thread at a given cycle
- Execution unit (or pipeline stage) utilization can be low if there are not enough instructions from a thread to "dispatch" in one cycle
- In a machine with multiple execution units (i.e., superscalar)
  - Have multiple thread contexts in a single processor
  - When the hardware executes from those hardware contexts determines the granularity of multithreading

Simultaneous Multithreading

- Dynamic adaptation to different parallelism types
  - For regions with low thread level parallelism (TLP) entire machine width is available for instruction level parallelism (ILP)
  - For regions with high thread level parallelism (TLP) entire machine width is shared by all threads





Next Class

- Graphics Processing Units