CS5375 Computer Systems Organization and Architecture

Lecture 22

Guest Lecture by **Dr. John Leidel**

Chief Scientist, Tactical Computing Labs

HARDWARE MULTITHREADING

Dr. John Leidel
Chief Scientist, Tactical Computing Labs

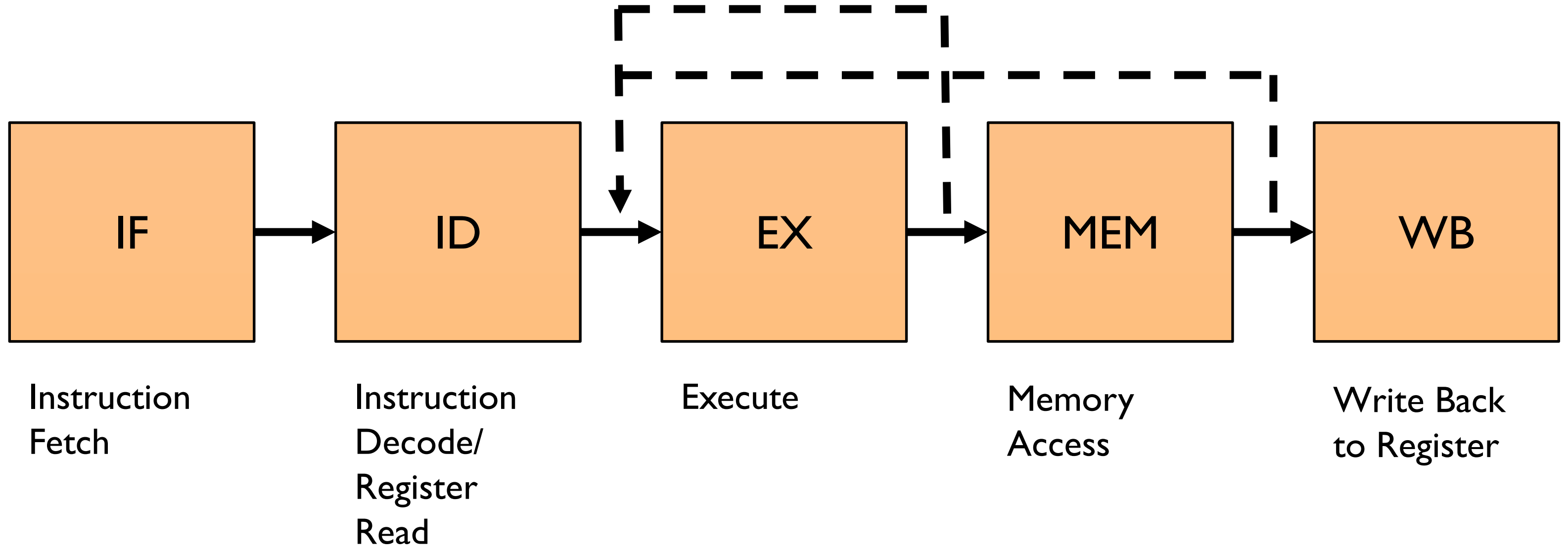
CS5375, Fall 2022

OUTLINE

- Life of a Thread
- Chip Multithreading
- Historic Architectures
- Research Challenges

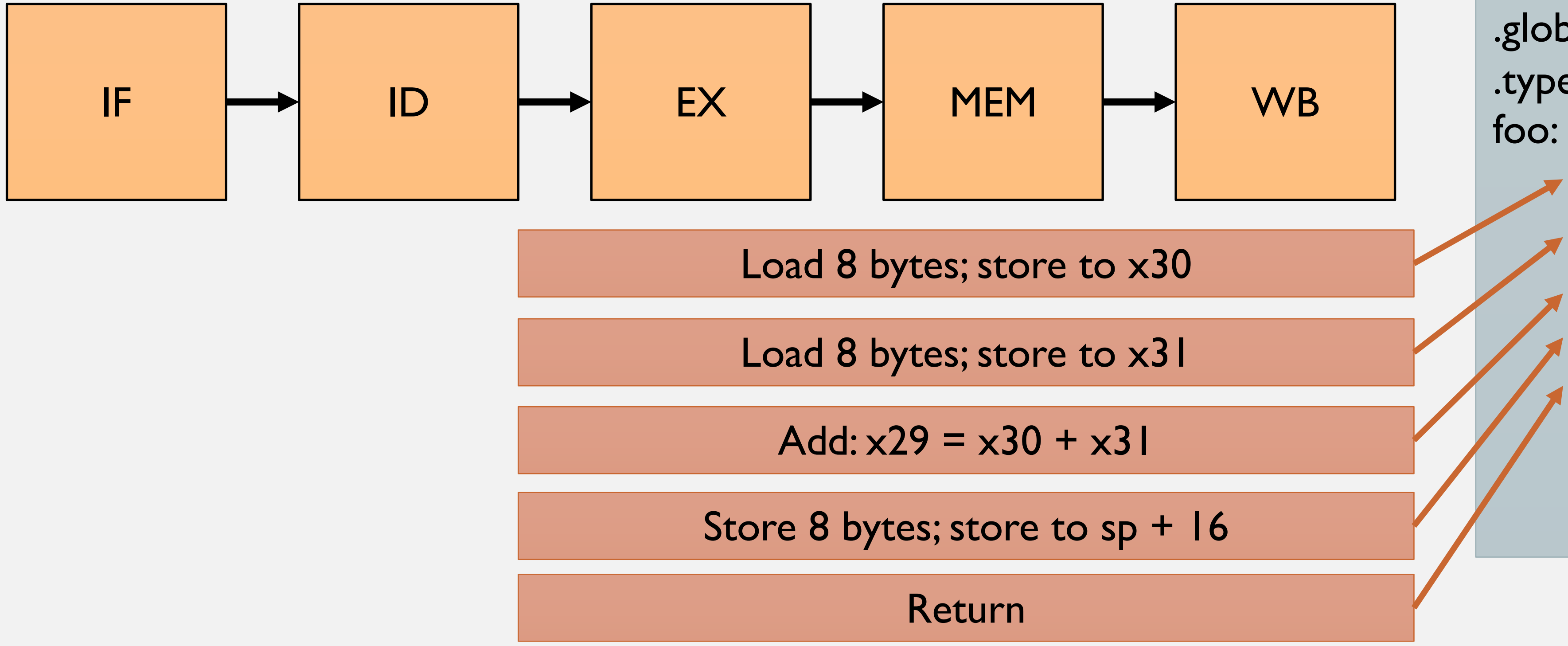
LIFE OF A THREAD

PIPELINED RISC



RISC-V “Rocket”

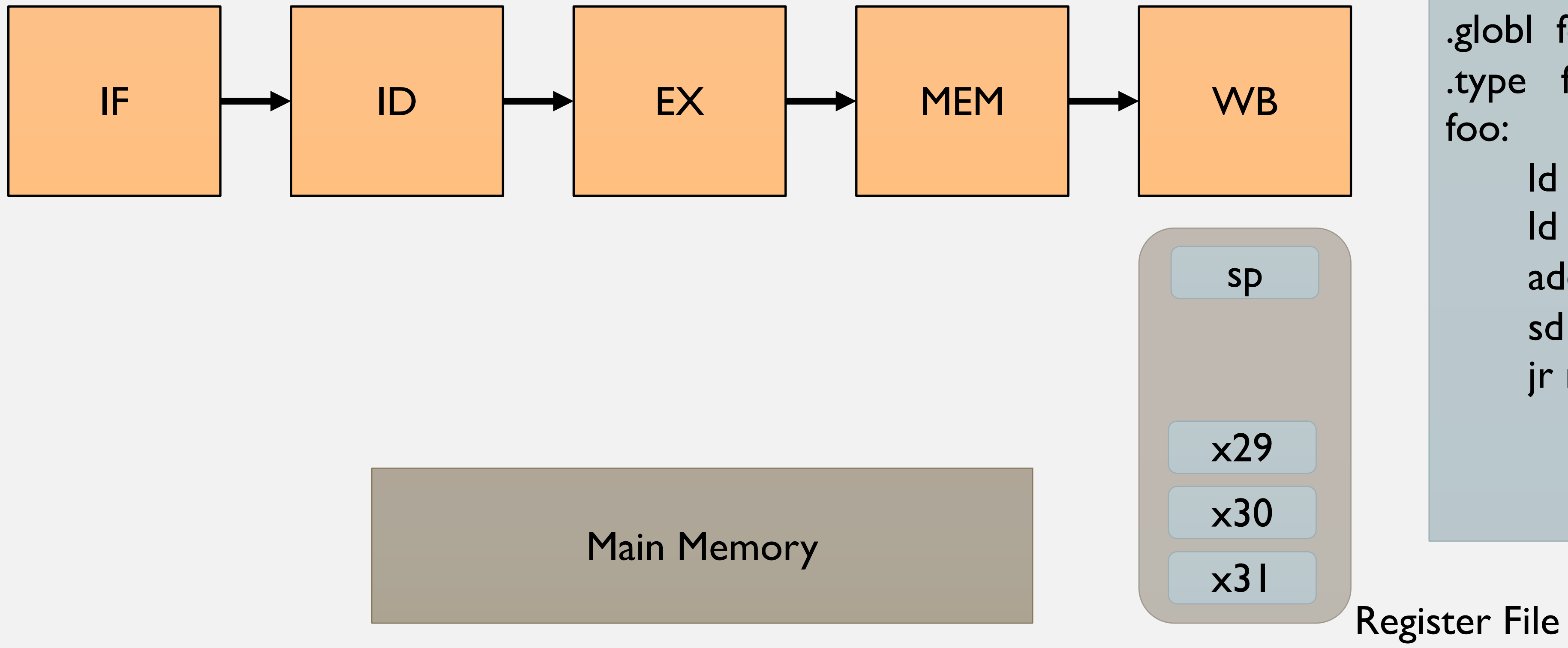
PIPELINED RISC EXECUTION


IF
ID
EX
MEM
WB
Load 8 bytes; store to x30
Load 8 bytes; store to x31
Add: x29 = x30 + x31
Store 8 bytes; store to sp + 16
Return

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

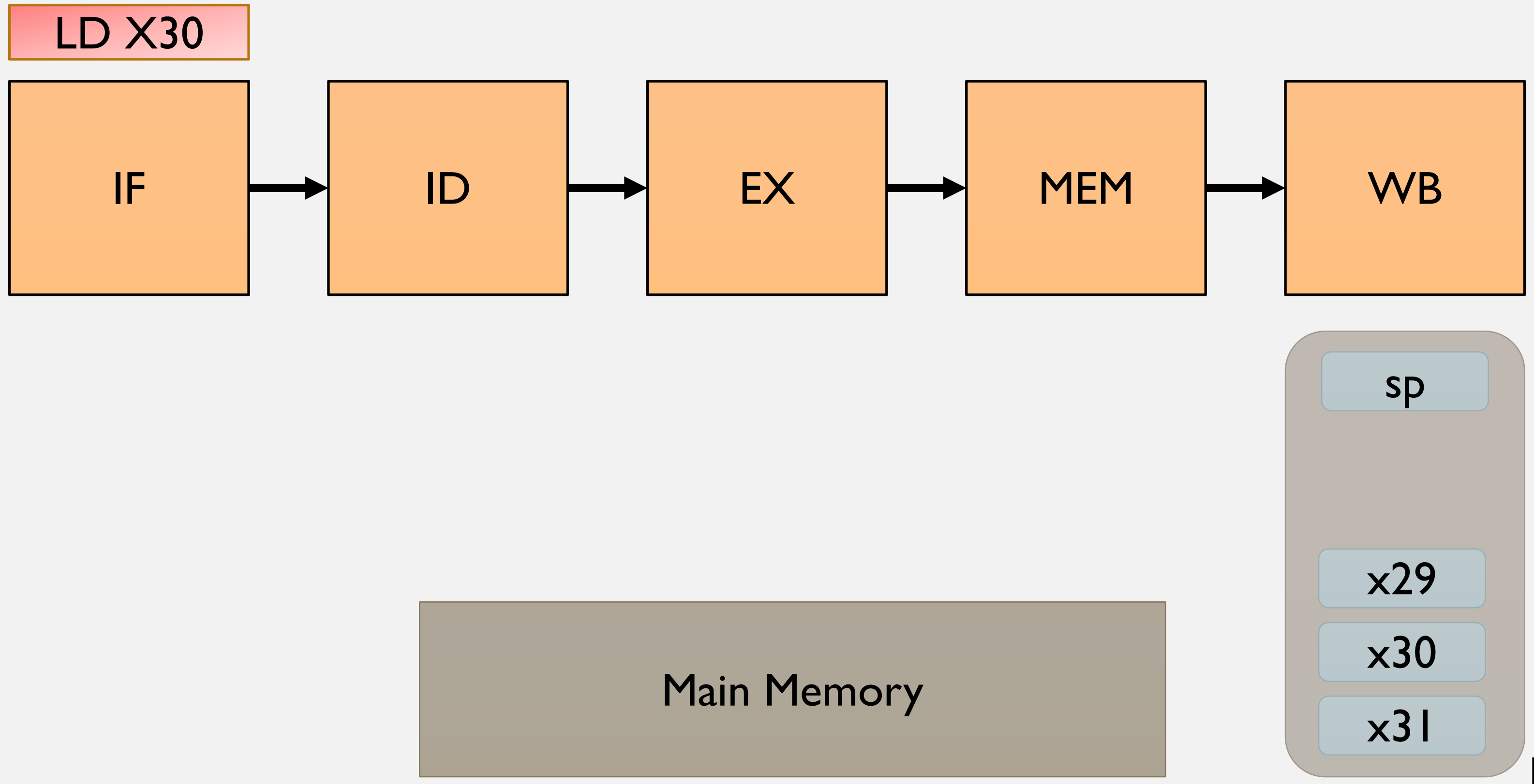
PIPELINED RISC EXECUTION



```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

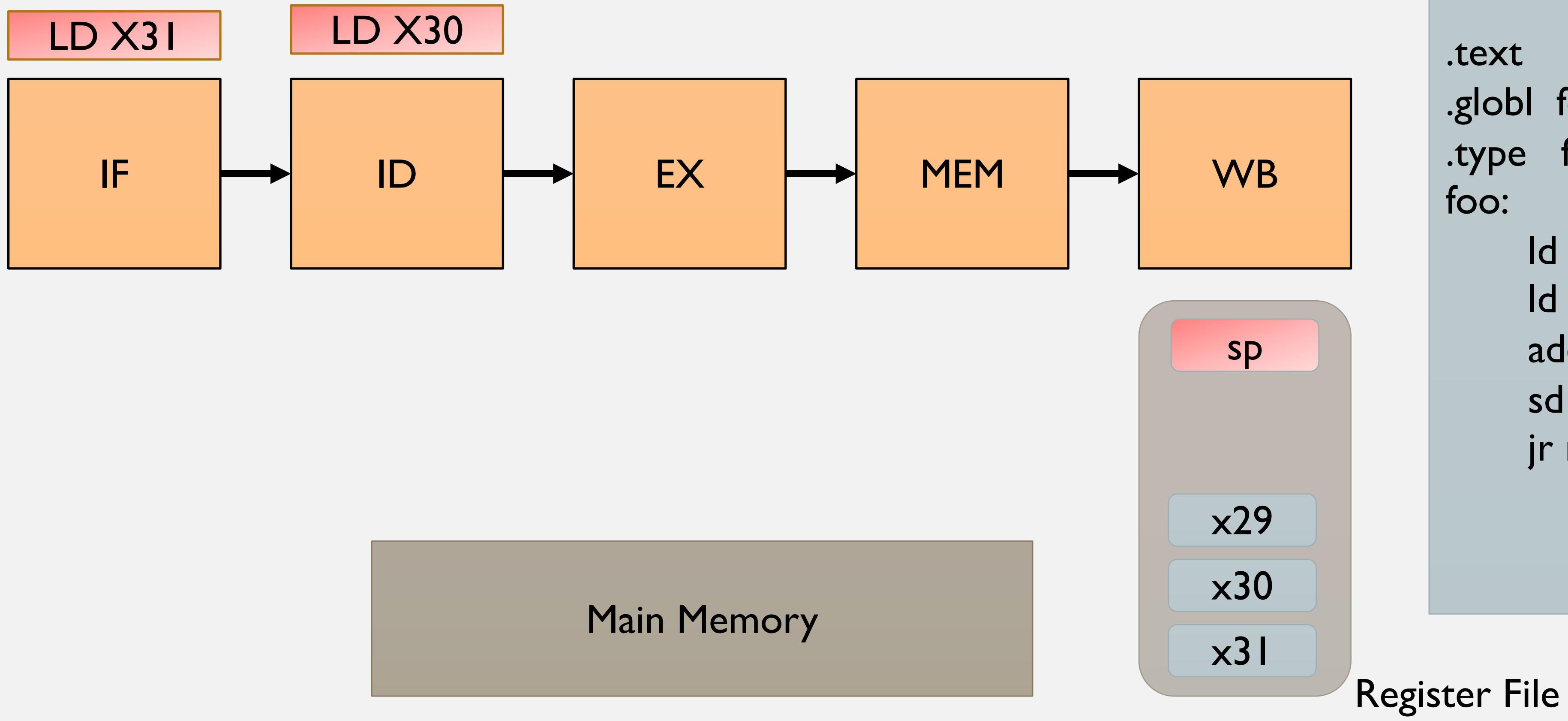
PIPELINED RISC EXECUTION



```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

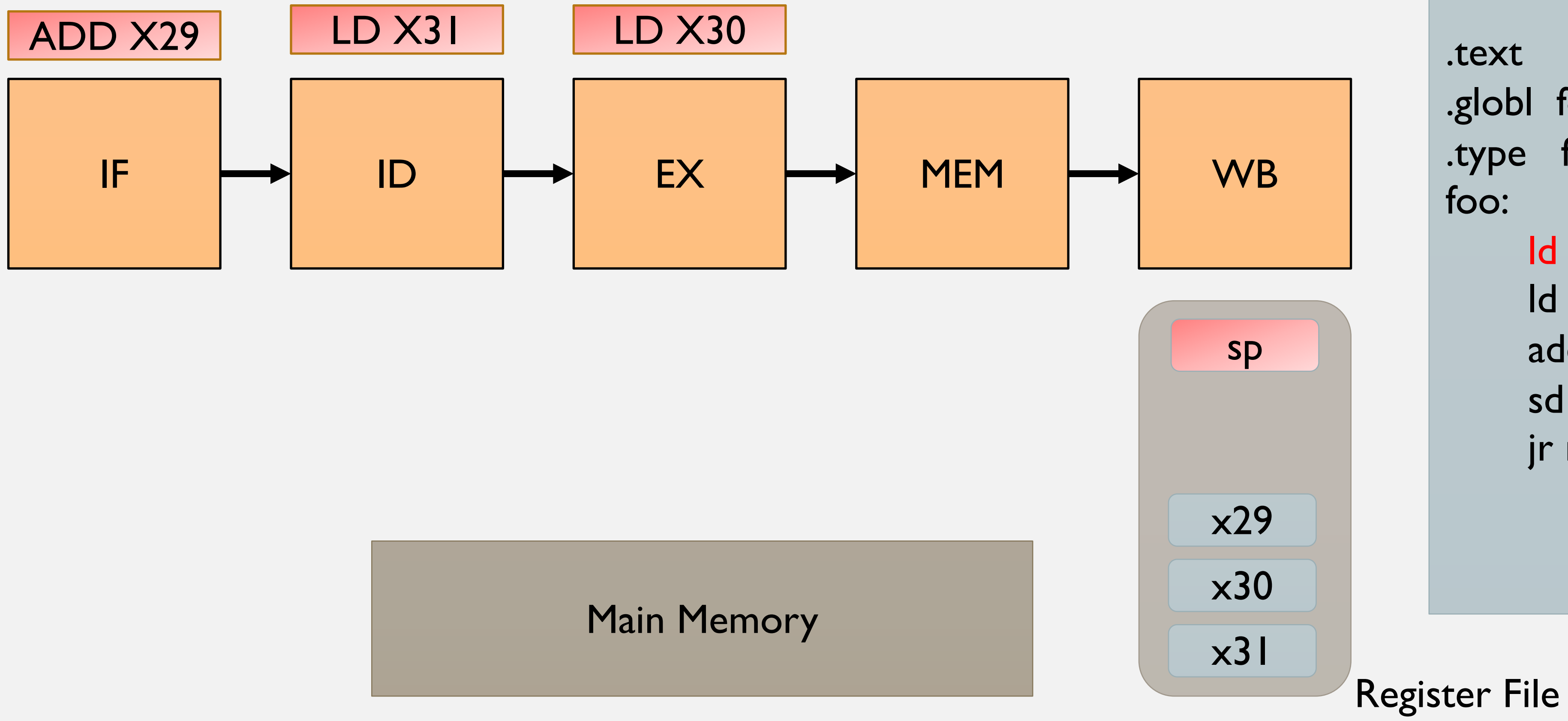
PIPELINED RISC EXECUTION



```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

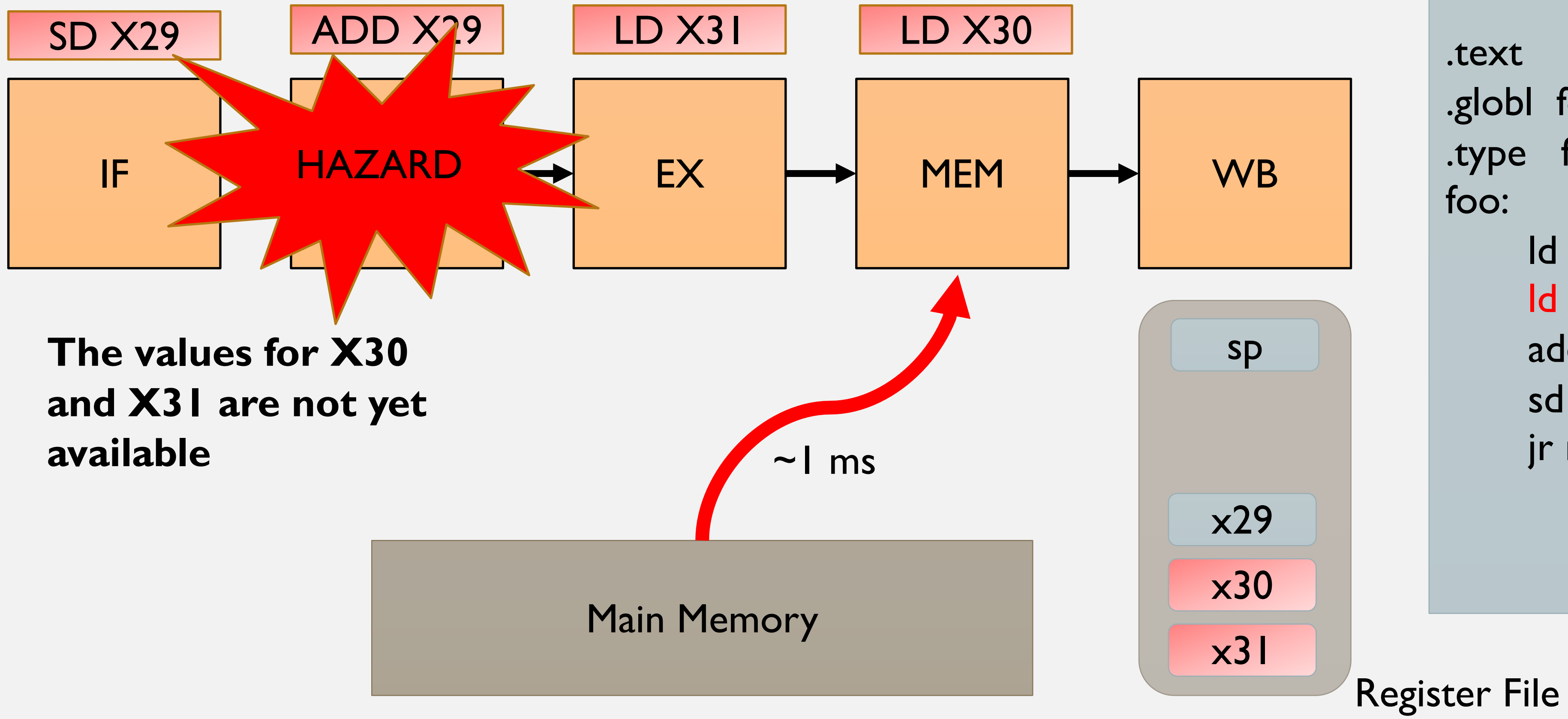
PIPELINED RISC EXECUTION



```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

PIPELINED RISC EXECUTION

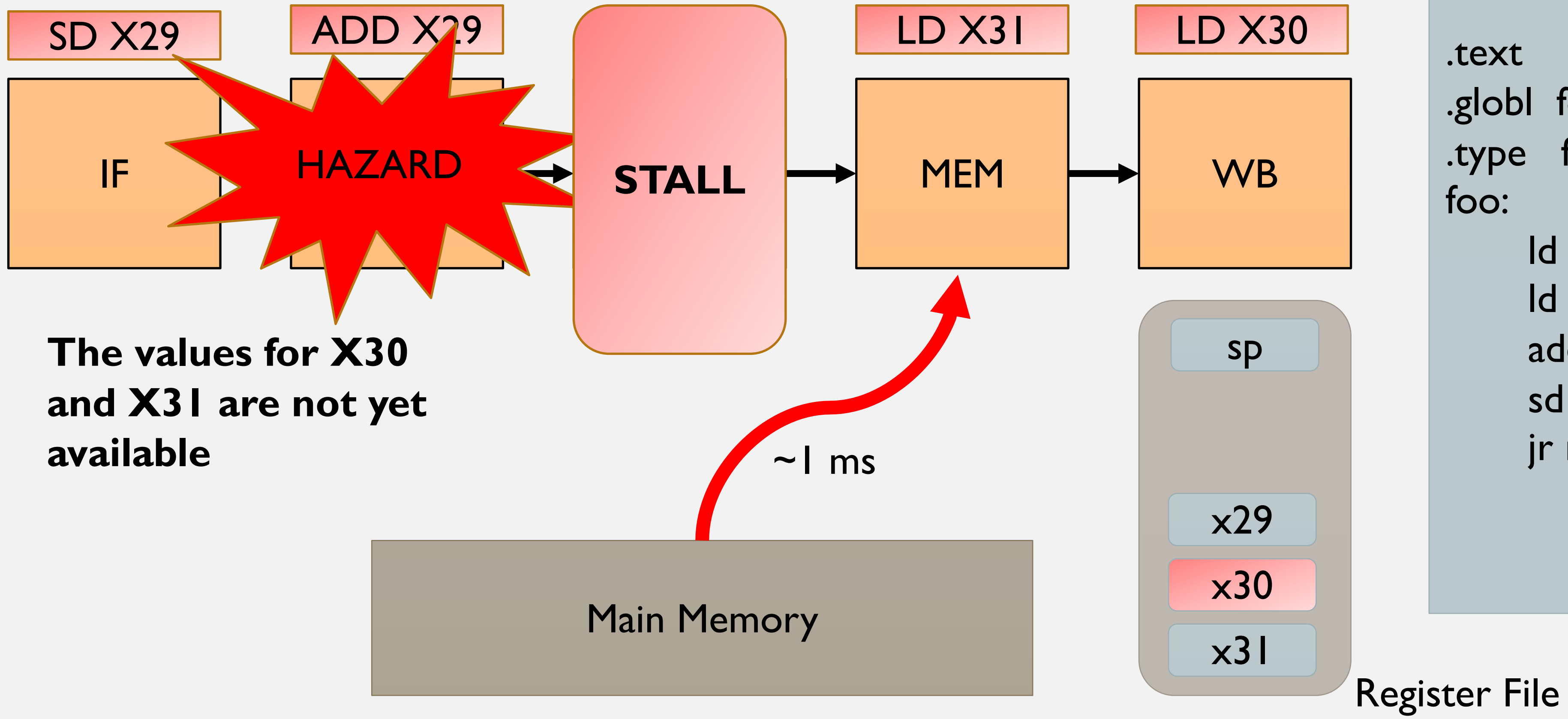


The values for X30 and X31 are not yet available

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

PIPELINED RISC EXECUTION

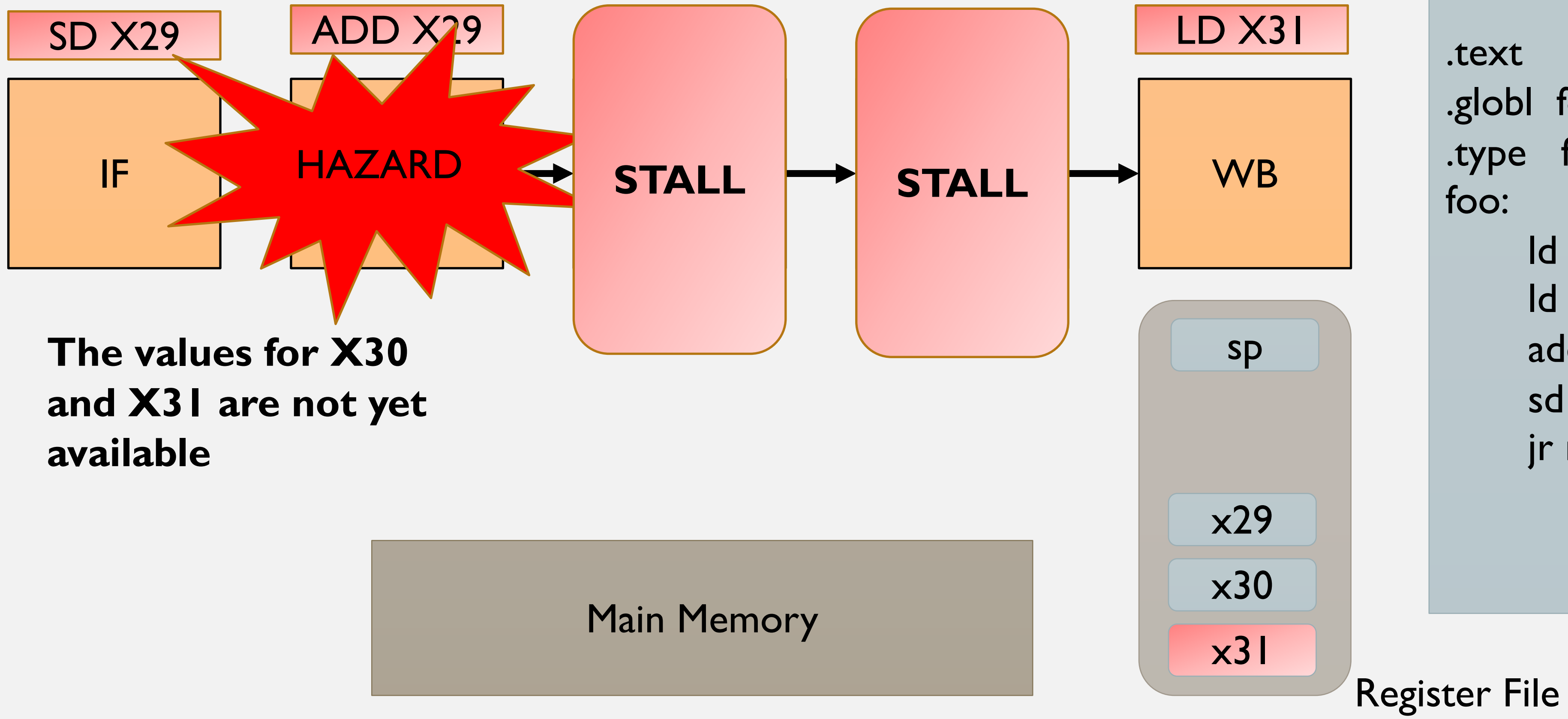


The values for X30 and X31 are not yet available

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

PIPELINED RISC EXECUTION



The values for X30 and X31 are not yet available

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

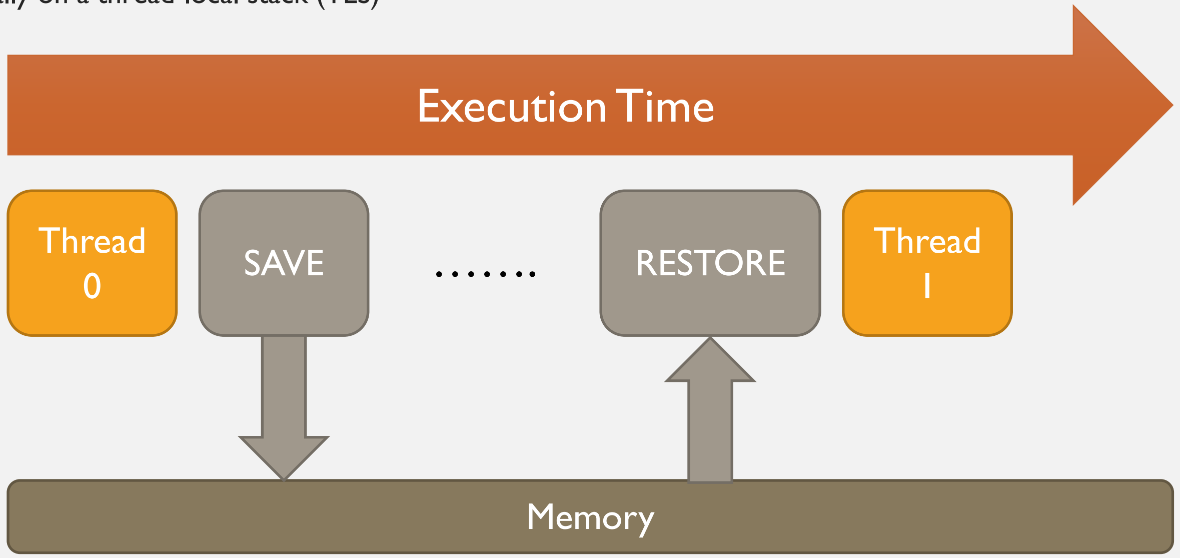
PIPELINE HAZARDS

- Our contrived example has a ~2ms pipeline hazard

- How do we avoid this?
  - Compiler modifications (instruction scheduling)
  - Multi-ported register files
  - Superscalar pipelines
  - **THREADING**

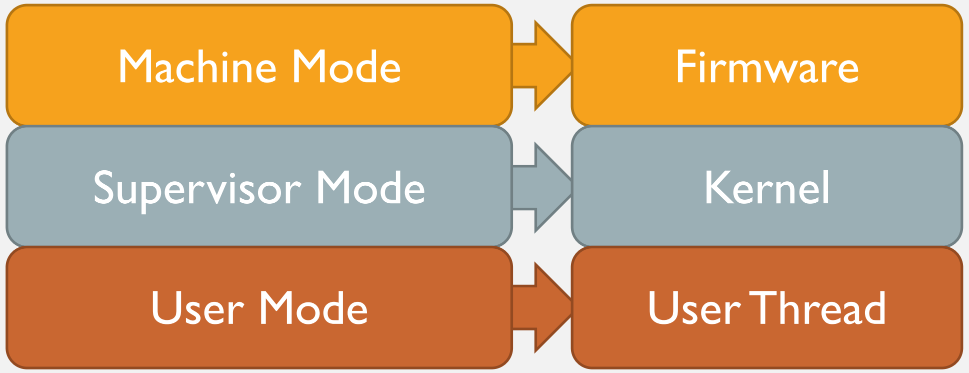
THREADING/CONTEXT SWITCHING

- The operating system participates with the hardware and enforces time sharing between tasks (threads, processes)
- Every time the kernel swaps threads/processes from utilizing the pipeline, a **context switch** is performed
- **Context switch**: the process of storing the state of the current thread of execution and then restoring the state of another thread
  - Contexts are stored to a **process control block**, generally on a thread-local stack (TLS)
- What does this include?
  - Register state
  - Program state (PC)
  - Interrupt state (if we're using thread-local interrupts)
  - Debug information
  - Any additional OS-specific state



PERFORMANCE

- The performance of standard context switch is gated by two major factors:
- Size of the context save region
  - RISC-V RV64G: 32 general registers, 32 floating point registers, PC
  - 65 registers x 64 bits = 4160 bits w/o TLS overhead
  - The larger the thread/register state, the slower the context switch
- Performance of the OS kernel mode switch
  - The kernel must actually perform the context switch such that the thread cannot poison the data
  - The kernel does this in ***Supervisor Mode*** such that it has access to all of the user thread's state
  - Switching to supervisor mode requires additional system calls

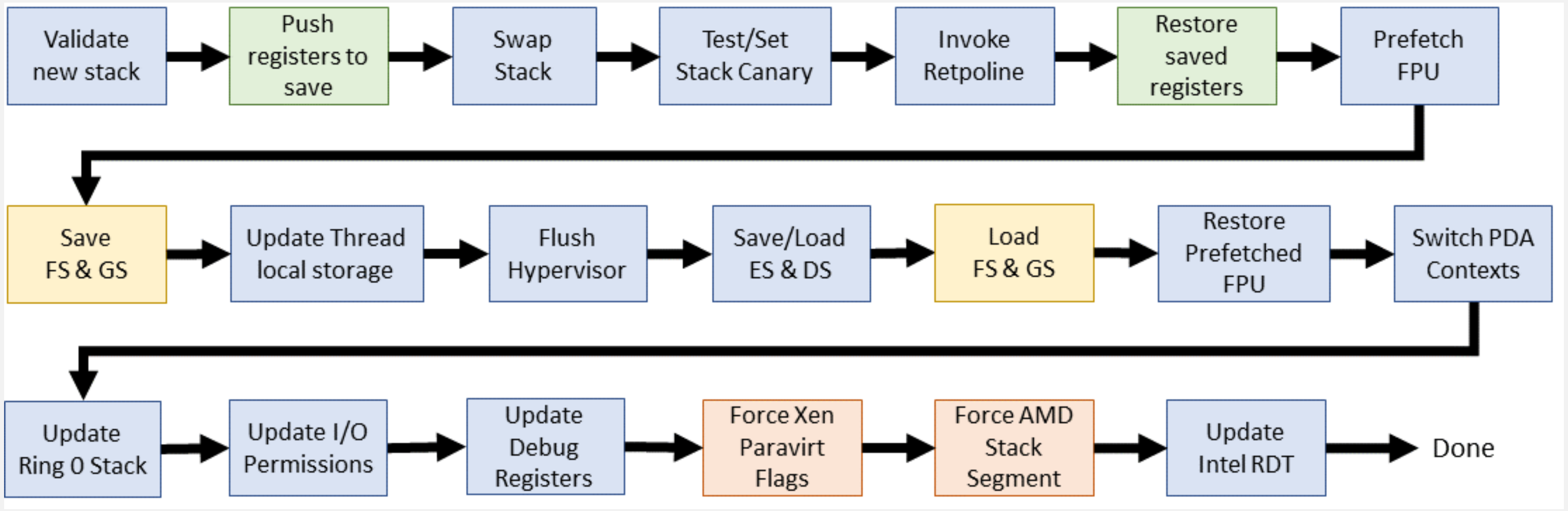


RISC-V Protection Rings

PERFORMANCE

- Additional transitionary operations may be required
  - TLB flushes: clearing the translated memory regions
  - I-Cache flushes: clear the threads cached instruction stack
  - D-Cache flushes: clear portions of the cached data (LRU)
  - Thread Local Storage (TLS) management: modern ELF binaries have thread local storage. Managing this often requires management of the stack, frame and memory pointers

SAMPLE KERNEL CONTEXT SWITCH



4.14 Linux Kernel: https://www.maizure.org/projects/evolution_x86_context_switch_linux/

HARDWARE/CHIP MULTITHREADING

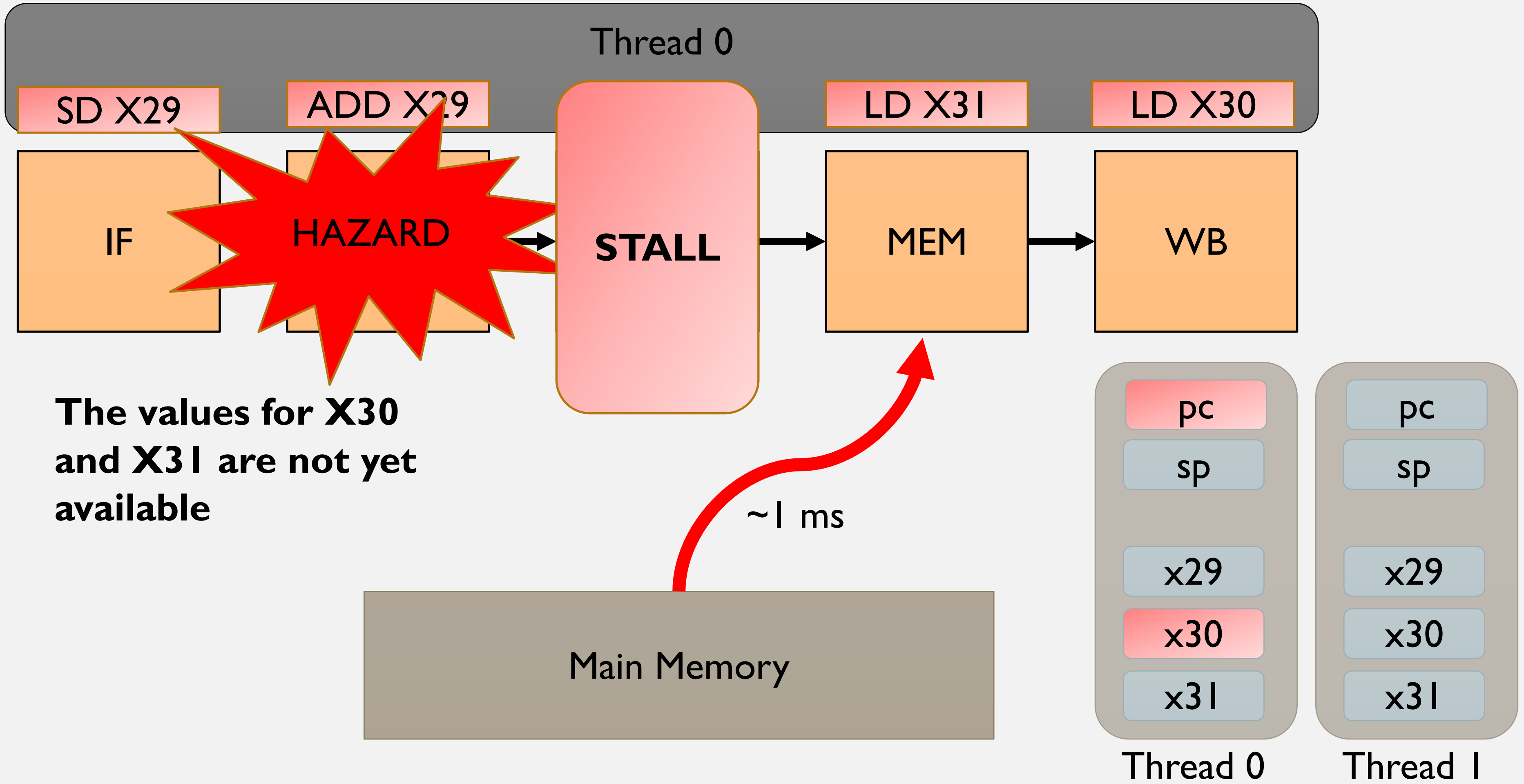
HARDWARE THREADING

- Hardware multi-threading is often employed to reduce the context switch overhead
  - Reduce the software and memory access penalties required to perform a context switch
  - Hardware resources/instructions that participate in software thread scheduling
- Additional hardware resources are provided for:
  - Storage of context save/restore state
  - Hardware driven context save/restore interrupts
- Goal:
  - ***Maximize the use of the hardware execution pipelines***

STEPS

1. Trigger the context switch
   - This may involve one or more mechanisms and/or interrupts
2. Pause the pipeline
   - The pipeline can no longer make forward progress
3. Save all the architectural state (Context Save)
4. Perform a ***pipeline flush***
   - Rewind any state that hasn't been fully executed in order to ensure that the pipeline is not poisoned for the next thread of execution
5. Restore the state from another thread (Context Restore)
6. Restart the pipeline

TEMPORAL THREADED RISC EXECUTION

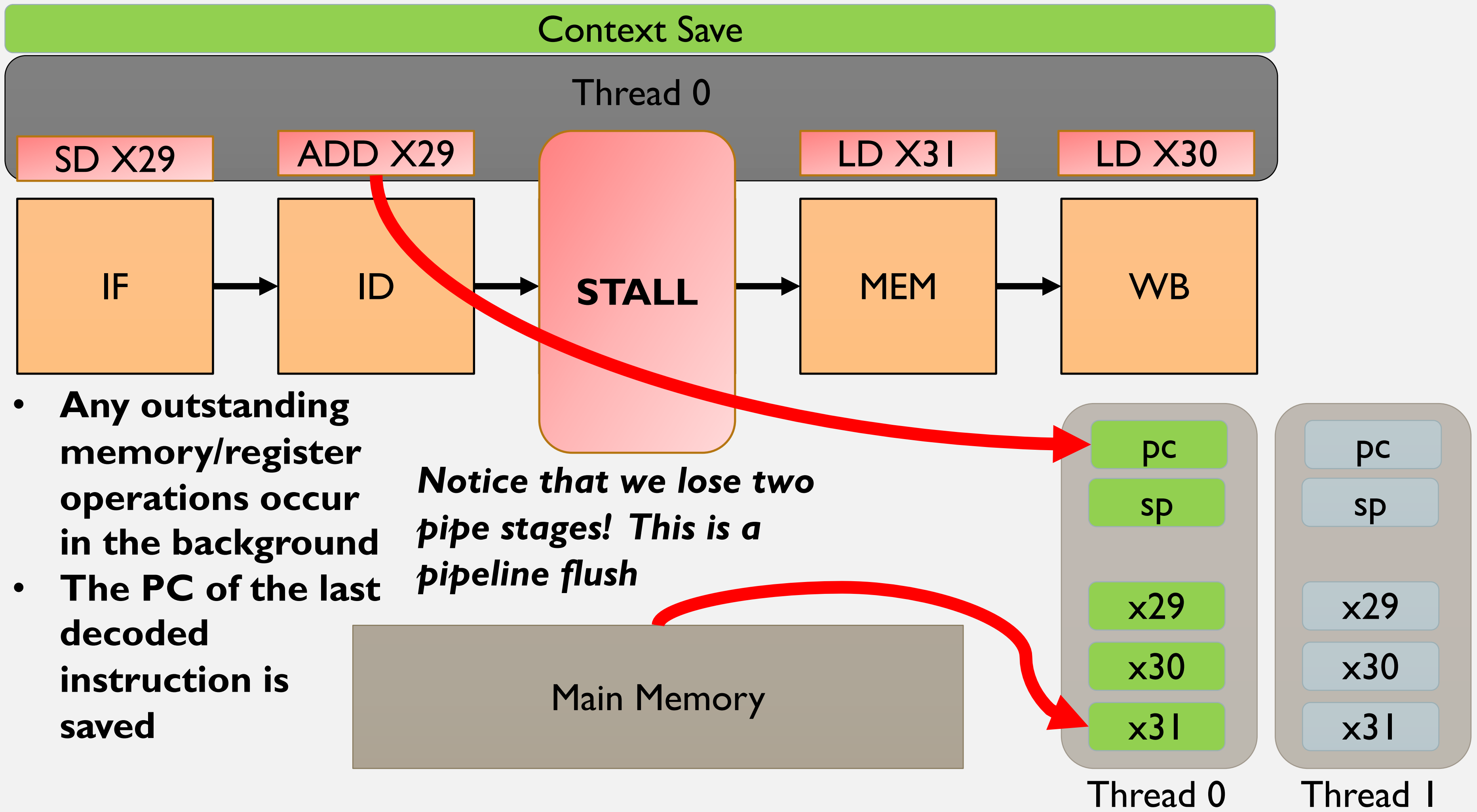


The values for X30 and X31 are not yet available

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

TEMPORAL THREADED RISC EXECUTION



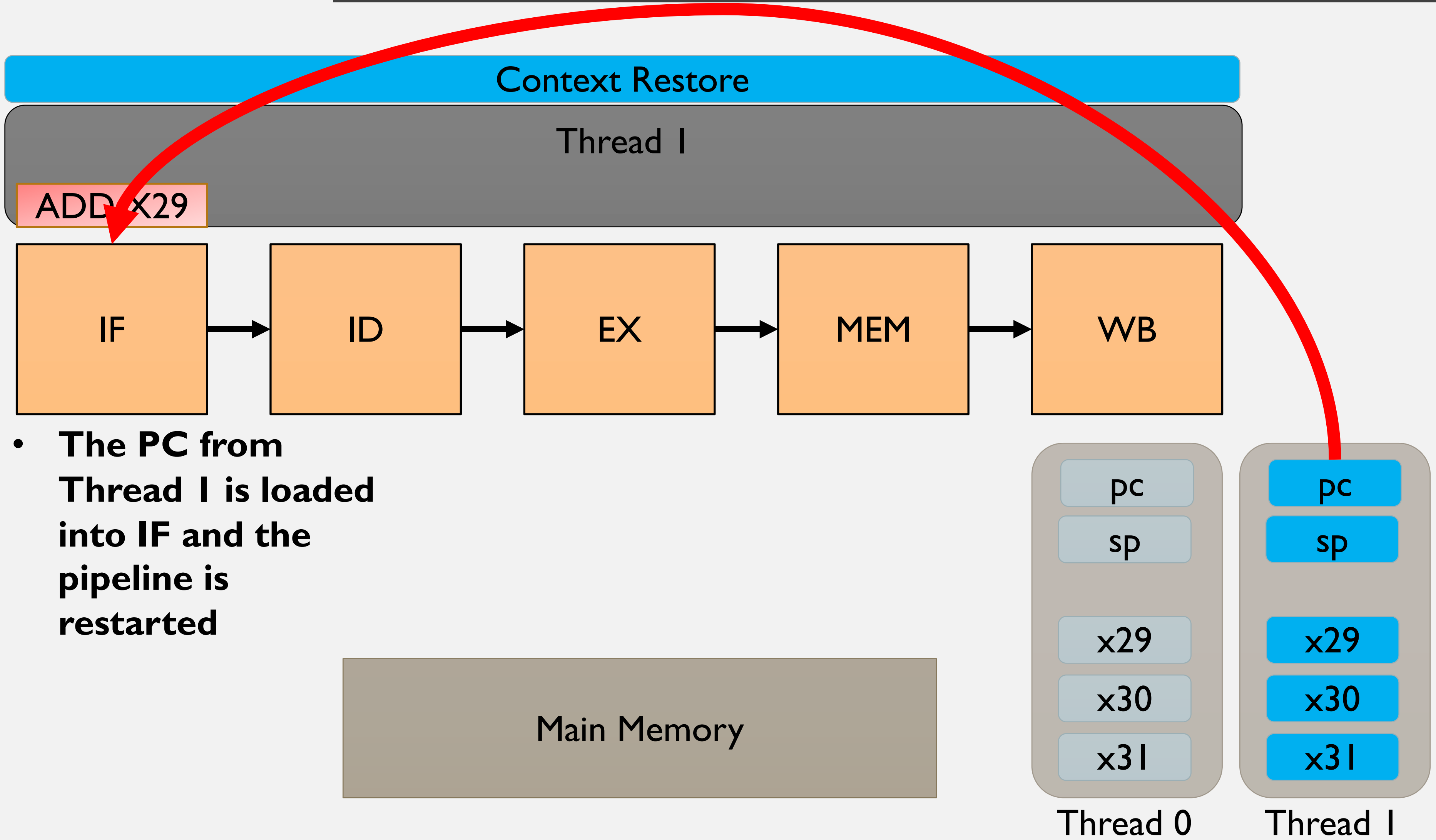
- **Any outstanding memory/register operations occur in the background**
- **The PC of the last decoded instruction is saved**

Notice that we lose two pipe stages! This is a pipeline flush

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

RV64-I

TEMPORAL THREADED RISC EXECUTION



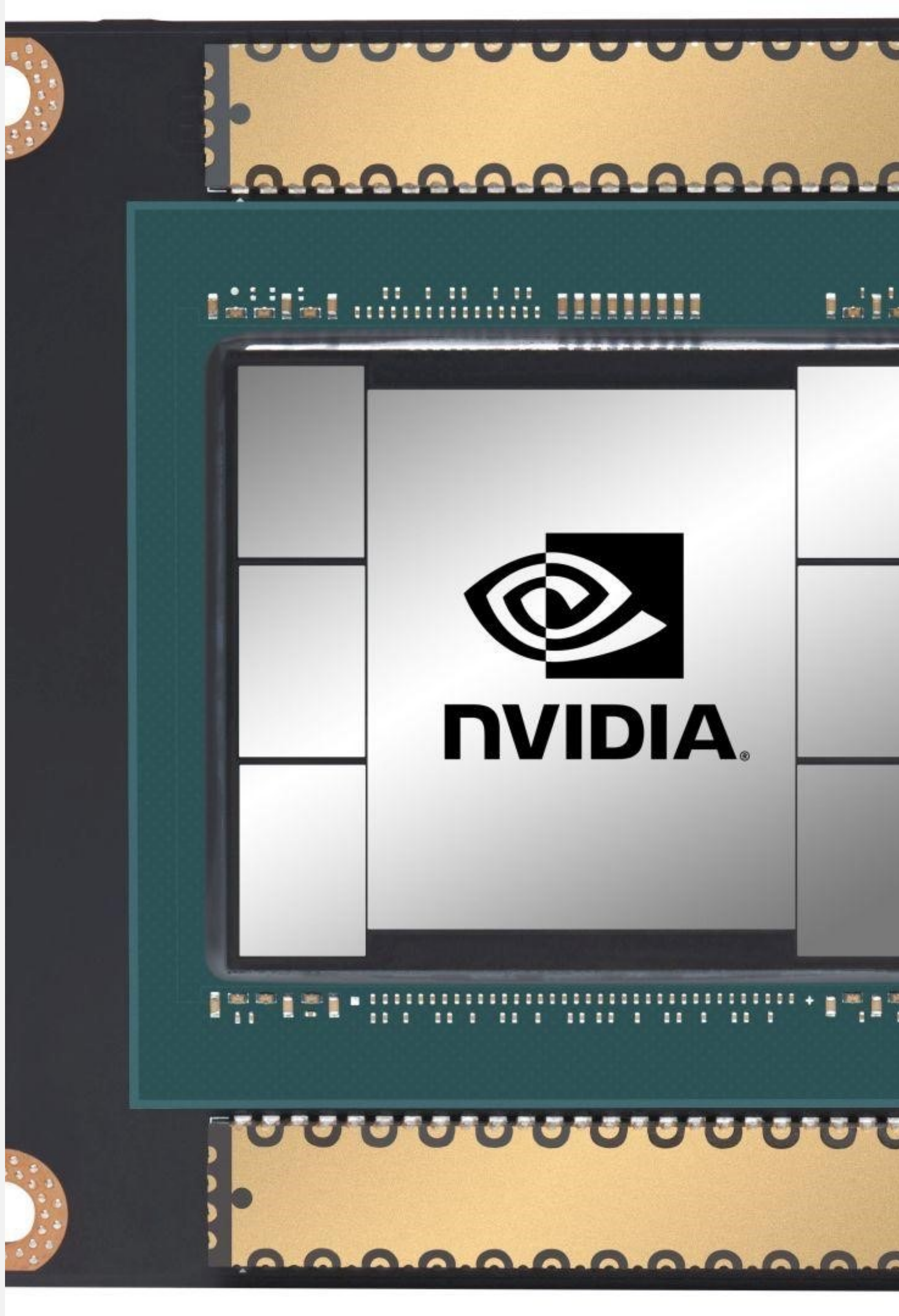
- **The PC from Thread 1 is loaded into IF and the pipeline is restarted**

```
.text
.globl  bar
.type   bar, @function
bar:
        add x29, x30, x31
        add x31, x31, sp
        jr ra
```

RV64-I

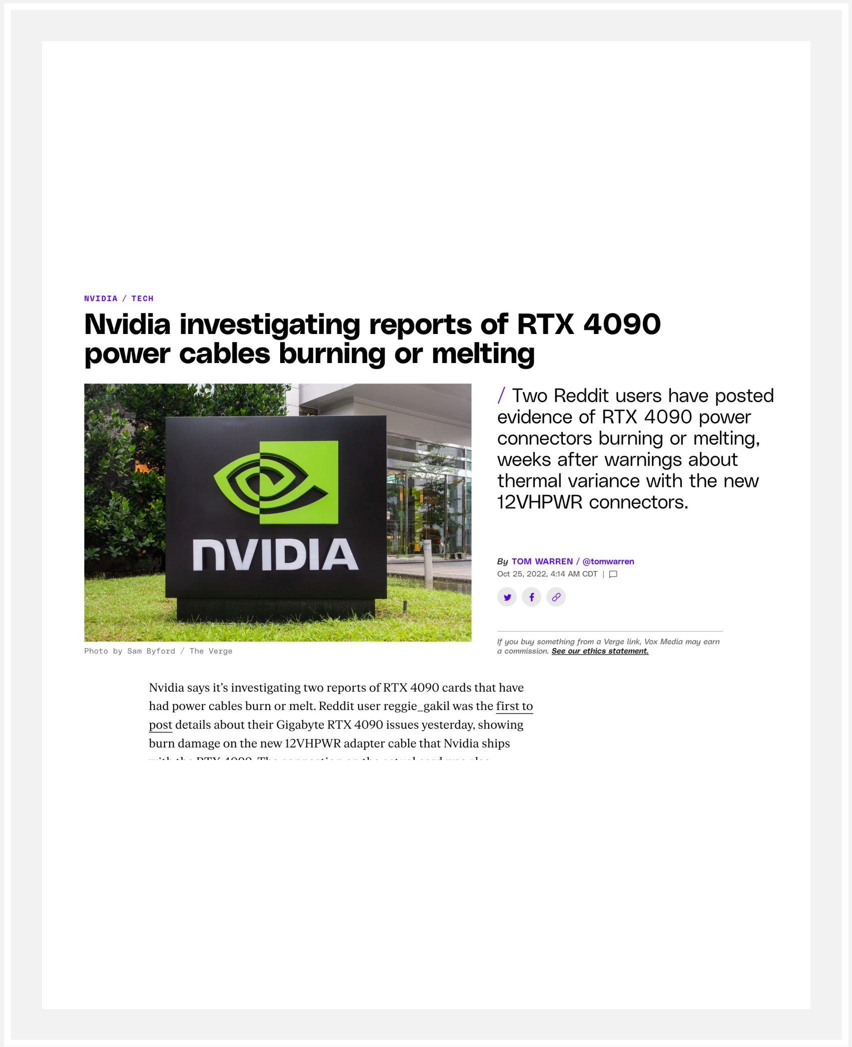
HARDWARE THREADING TYPES

- **Temporal Multithreading**:
  - Multiple hardware threads share access to pipeline resources, ***one*** thread at a time
  - Context switches involve two threads: One being saved, one being restored
  - Two types:
    - ***Coarse-grained***: context switches are performed by the kernel
    - ***Fine-grained***: context switches are performed between pipeline stages using hardware-driven mechanisms
- **Simultaneous Multithreading (SMT)**:
  - The maximum number of threads that can simultaneously execute in the pipeline is more than one
  - Context switches involve ***N*** threads, where {***N*** > 1}
  - GPU's utilize SMT mechanisms in order to maintain internal hardware concurrency
  - Question: why don't all hardware devices employ SMT techniques?



PERFORMANCE/POWER/AREA

- The performance of the hardware context switch is dependent upon:
  - The pipeline depth: *How many pipeline stages are cleared when a context switch occurs*
  - The size of the register context: *How much state is saved from the pipeline*
  - The size of the remaining bits: *How much system state must be stored*
  - The algorithm utilized to enforce context switching
  - ***Ideal situation***: context switches require a single cycle
- The increased hardware state (register files) requires additional area
- All the additional registered state, requires power
  - SMT architectures are notorious for maintaining power close to TDP



NVIDIA / TECH

Nvidia investigating reports of RTX 4090 power cables burning or melting

/ Two Reddit users have posted evidence of RTX 4090 power connectors burning or melting, weeks after warnings about thermal variance with the new 12VHPWR connectors.

Photo by Sam Byford / The Verge

By TOM WARREN / @tomwarren
Oct 25, 2022, 4:14 AM CDT |

If you buy something from a Verge link, Vox Media may earn a commission. ***See our ethics statement.***

Nvidia says it's investigating two reports of RTX 4090 cards that have had power cables burn or melt. Reddit user reggie_gakil was the first to post details about their Gigabyte RTX 4090 issues yesterday, showing burn damage on the new 12VHPWR adapter cable that Nvidia ships

https://www.theverge.com/2022/10/25/23422349/nvidia-rtx-4090-power-cables-connectors-melting-burning

CONTEXT SWITCH SCHEDULING

- One of the key aspects of of hardware multithreading is the mechanism by which to trigger a context switch

- Styles:
  - ***Manual threading***: a thread utilizes a special instruction/mechanism to manually trigger a context switch
  - ***Barrel threading***: counter driven mechanism to trigger context switches every N cycles
  - ***Hyperthreading***: Intel's virtualized core technology using superscalar pipelines where only portions of the required hardware state are duplicated
  - ***Managed threading***: technique by which a thread's execution progress/content determines when a context switch occurs
  - *Combinations thereof…*

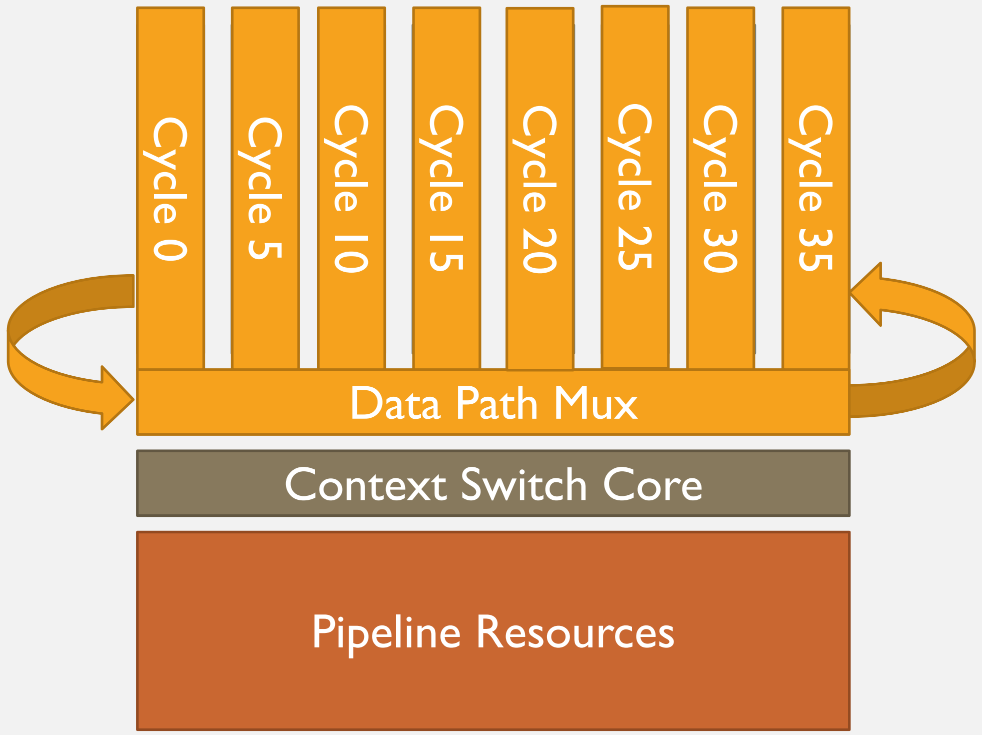
MANUAL THREADING

- Each thread manually yields pipeline access to another thread
- This generally requires specific instructions
- The only systems that generally implement this style are dataflow and/or highly embedded systems
- Context pipeline flushes will include all stages prior to the execution stage(s)
- VERY easy to deadlock!

```
.text
.globl  foo
.type   foo, @function
foo:
        ld x30, 0(sp)
        ld x31, 8(sp)
        yield
        add x29, x30, x31
        sd x29, 16(sp)
        jr ra
```

BARREL THREADING

- Barrel threading implicitly induces a context switch every ***N*** cycles
- Generally speaking, the threads DO NOT have any control over when/how the switches are induced
- This is often coupled to pipeline hazards (when selecting which thread to execute next)
  - Selecting threads is an entirely separate topic
  - Round robin scheduling is often utilized
- Pros:
  - Threads are guaranteed to execute some number of cycles at a deterministic cadence
- Cons:
  - If insufficient threads are available, the pipeline stalls
  - This can often generate significant cache contention (threads can diverge rather quickly)



MANAGED THREADING

- Managed threading induces a context switch based upon a predefined algorithm wired into the pipeline circuit
- Threads MAY have any control over when/how the switches are induced
- This is often coupled to instruction cost and/or pipeline hazards
- Pros:
  - More control from software is often provided
- Cons:
  - Threads are not guaranteed to share access to hardware
  - Often requires support from the runtime library

MANAGED THREADING

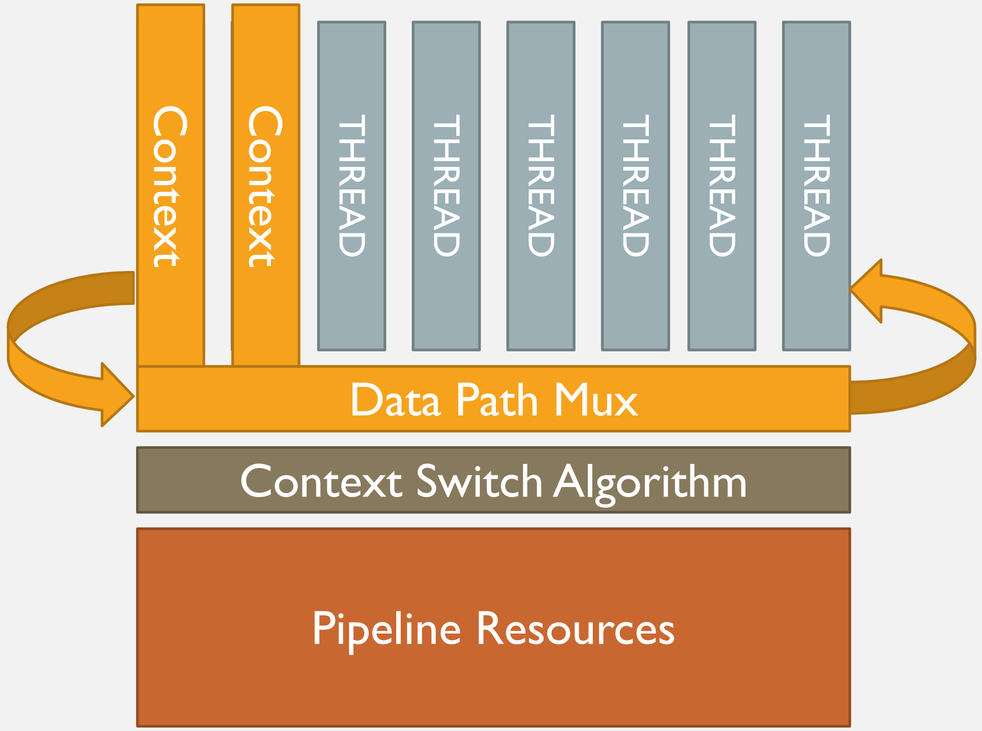
- Example: Context switch on every third load instruction

Thread 0

```
ld x29, 0(sp)
ld x28, 8(sp)
ld x27, 16(sp)
add x25, x29, x28
add x25, x25, x27
st x25, 24(sp)
```

Thread 1

```
ld x29, 0(sp)
ld x28, 8(sp)
add x25, x29, x28
ld x27, 16(sp)
add x25, x25, x27
st x25, 24(sp)
```



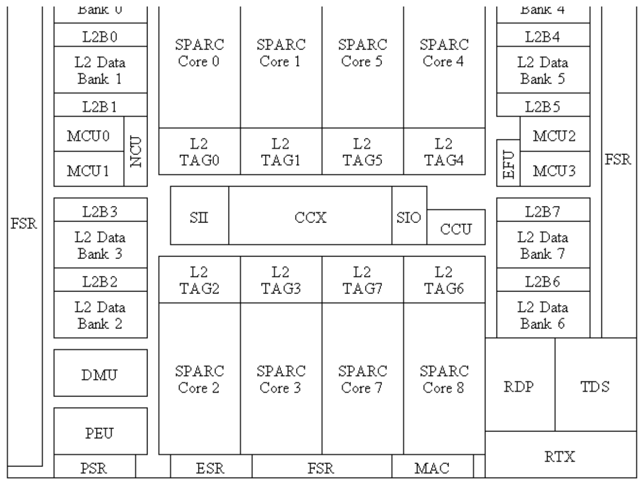
PITFALLS

- Additional hardware state requires additional chip area, power
- Deeper pipelines induce larger pipeline flushes
- Context switch algorithms can potentially deadlock a system if not carefully crafted
- Increased design complexity requires additional design verification
- Executing *too* concurrently can/will require additional power resources
- If your threads diverge too far, you will experience issues with I-Cache misses
- You can induce malicious timing attack vectors
- The ***software*** will need additional work to support the hardware concurrency

HISTORIC ARCHITECTURES

SUN SPARC/OPENSPARC

- T2 Series introduced scalable chip multithreading
  - 8 threads per core; 8 cores per CPU; 64 concurrent threads
  - Superscalar pipeline with 2 integer ALU's per core
  - One of the first major commercial success for CMT technology
  - Initial research was largely completed by Kunle Olukotun (Stanford)
  - T3 series increased sequential throughput
- Now open source via the OpenSPARC project

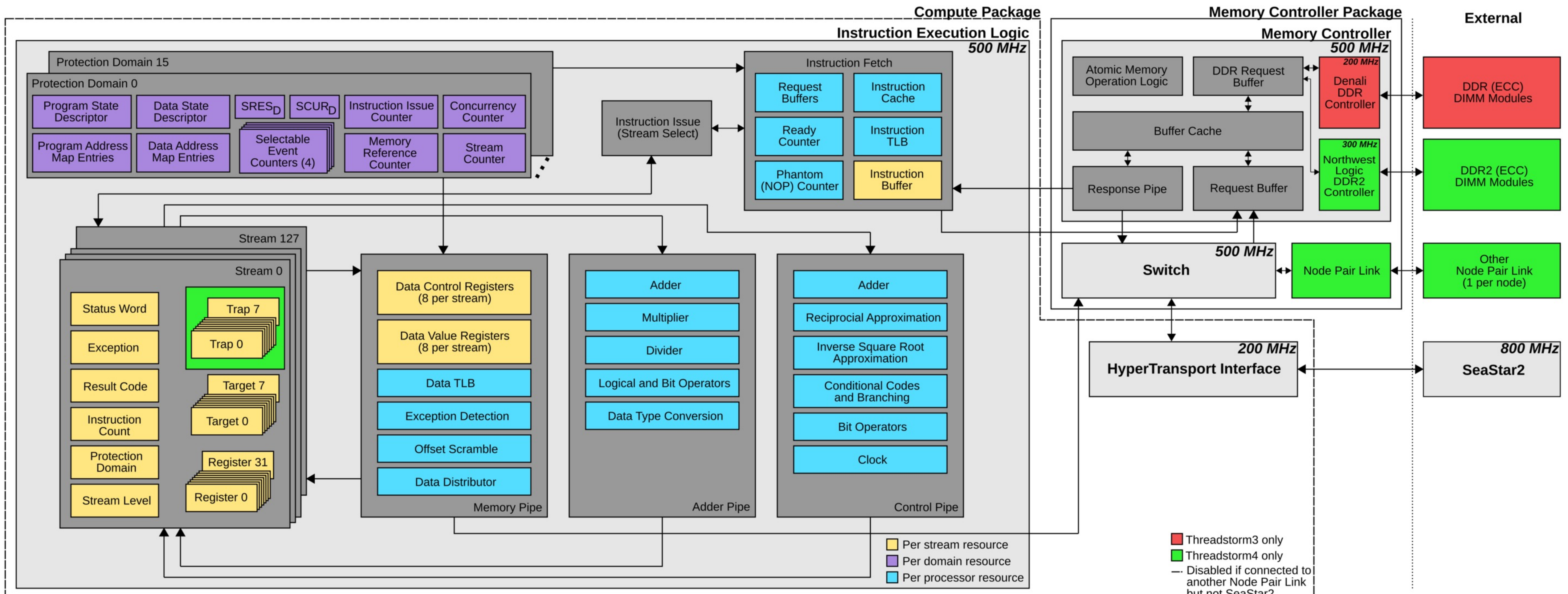


Niagra 2 / UltraSPARC T2 / OpenSPARC T2 - Die Micrograph Diagram (davidhalko)

TERA MTA/CRAY XMT2

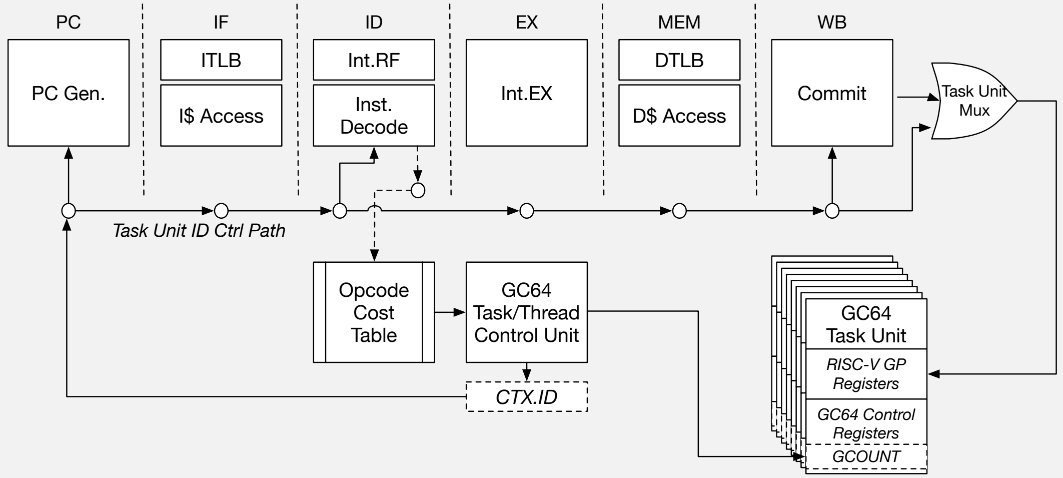
- Originally designed by Burton Smith (Seymour Cray winner) (MTA)
- Revised after merger with Cray.
- Released in 2005
- 128 threads per core @ 500Mhz
- 64-bit barrel thread configuration set to switch every cycle
- 21 stage pipeline (threads would never execute until at least 21 cycles in the future)
- Designed for executing graph algorithms or other sparse problems

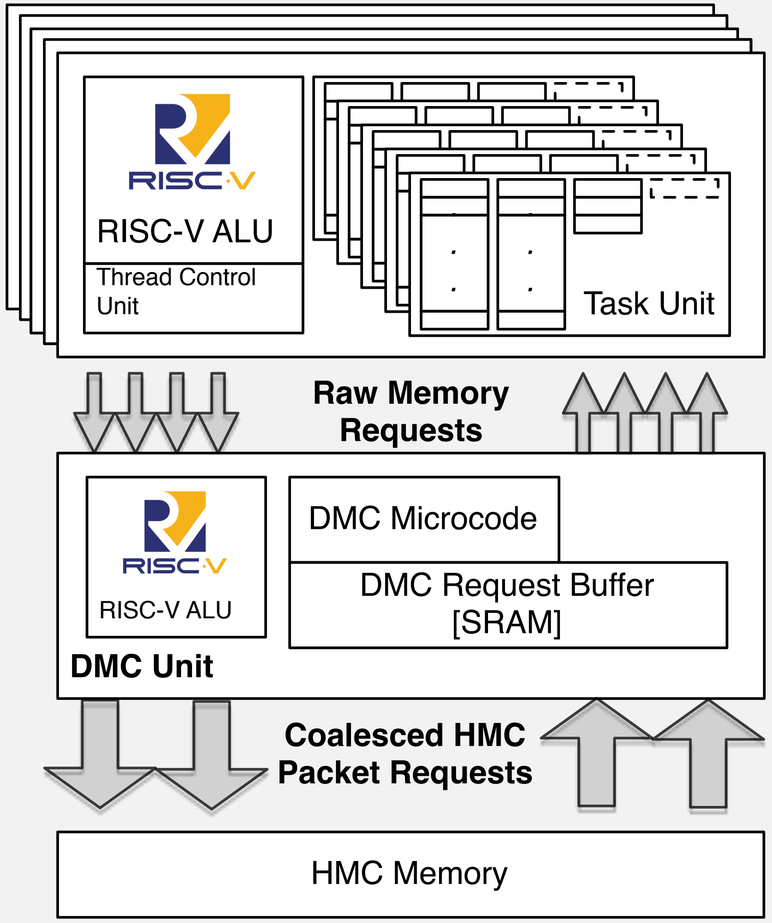
TERA MTA/CRAY XMT2



GOBLINCORE-64

- RISC-V RV64G microarchitecture built for sparse solvers
- Temporal context switching using pressure-driven algorithm
  - Threads exert *pressure* on the pipeline.
  - Pressure is accumulated for each instruction until we reach saturation and context switch





RESEARCH CHALLENGES

RESEARCH CHALLENGES

- What is the correct context switch algorithm?
    - How do we derive the mechanism utilized for existing hardware? (Hardware vendors often hide this)
- How do we derive the ***best*** number of threads per core?
- How do we couple the context switching to our runtime library?
- How do we couple the context switching mechanisms to the compiler?

TACTICAL COMPUTING LABS

- ***We're hiring!***
  - Positions are fully remote!

- ***Hardware Research Engineers***
  - Verilog, Chisel, C++ development experience
  - ASIC/FPGA design/layout experience
- ***HPC Simulation Developer***
  - HPC/parallel programming experience
  - C, C++, Python experience
  - Experience in hardware simulation highly desired

https://tactcomplabs.com/job-openings/