SLIDE 6: ACTIVITY

GIVEN SEQUENCE OF THREE MEMORY TRANSACTIONS

READ
WRITE
READ

DRAW PIPELINE DIAGRAM SHOWING HOW THESE TRANSACTIONS EXECUTE ON THREE MICROARCHITECTURES

1) SINGLE-CYCLE
2) TWO CYCLE UNPIPELINED
3) TWO CYCLE PIPELINED

STAGE 1: (MT) MEMORY TAG CHECK
STAGE 2: (MD) MEMORY DATA ACCESS

EXAMPLE: SINGLE-CYCLE

READ \( \rightarrow \downarrow \text{MT} \) \( \rightarrow \downarrow \text{MD} \) \( \rightarrow \downarrow \text{AD} \)
WRITE \( \rightarrow \downarrow \text{MT} \) \( \rightarrow \downarrow \text{MD} \) \( \rightarrow \downarrow \text{AD} \)
READ \( \rightarrow \downarrow \text{MT} \) \( \rightarrow \downarrow \text{MD} \) \( \rightarrow \downarrow \text{AD} \)

MT AND MD HAPPEN IN A SINGLE CYCLE
STALL?

\[ \text{STALL}_Y = (\text{TYPE}_Y = \text{RD}) \land (\text{TYPE}_M = \text{WR}) \]

DUPLICATE?

\[ \text{STALL}_Y = (\text{TYPE}_Y = \text{RD}) \land (\text{TYPE}_M = \text{WR}) \]

WHAT IF WRITING + READING SAME ADDRESS? RAW HAZARD!

SCALING?

Hazard REQUIRES Extra MEMORY ADDRESS!!
DIFFICULT TO KNOW AT COMPILE TIME

STALL?

\[ \text{STALL}_Y = (\text{TYPE}_Y = \text{RD}) \land (\text{TYPE}_M = \text{WR}) \land (\text{ADD}_Y = \text{ADD}_M) \]

BYPASS?

* ACTIVITY: DRAW BYPASS PATH ON SLIDE 8
SLIDE 11: SCREW

no y, comb!

SLIDE 12: PIPELINES

"P" STAGE

SLIDE 13: PARALLEL READ / PIPELINES

RAW HAZARD

Between DATA + INSTR ?
Activity: Synchronization, Consistency, Concurrency

Initialize
\[ x = 0 \]
\[ y = 1 \]

Synchronization

\begin{align*}
\text{Process 0} & \quad \text{Process 1} \\
1 \text{ \text{LW} r1, x} & \quad 5 \text{ \text{LW} r1, x} \\
2 \text{ \text{LW} r2, y} & \quad 6 \text{ \text{LW} r2, y} \\
3 \text{ \text{SW} r2, x} & \quad 7 \text{ \text{SW} r2, x} \\
4 \text{ \text{SW} r1, y} & \quad 8 \text{ \text{SW} r1, y}
\end{align*}

Which are valid final values for \( x \) and \( y \)?
\( \{00, 10, 11, 01\} \)

Derive dynamic instructions to verify that each final value is possible
\[ x = 0 \quad y = 1 \]

Diagram:
\[ \begin{array}{c}
\text{P0} \\
1 \text{ \text{LW} r1, x} \\
2 \text{ \text{LW} r2, y} \\
3 \text{ \text{SW} r2, x} \\
4 \text{ \text{SW} r1, y}
\end{array} \quad \begin{array}{c}
\text{P1} \\
5 \text{ \text{LW} r1, x} \\
6 \text{ \text{LW} r2, y} \\
7 \text{ \text{SW} r1, x} \\
8 \text{ \text{SW} r1, y}
\end{array} \]
ILLUSTRATES NEED FOR SIMPLIFICATION

AHO. SWAP
AHO. OR
AHO. AND
AHO. ADD
ATOMIC MEMORY OPTIONS

```
AMO. OP  GET, ADD, SUB

TEMP = M[R[ADD]]
M[R[ADD]] = TEMP + R[SRC]
R[GET] = TEMP

loop:
AMO. ON  r1, r2, 1
  loop  r1, r0, loop
w  r1, x
w  r2, y
sw  r2, x
sw  r1, y
sw  r0, z
```

GET LOCK

RELEASE LOCK

CONSISTENCY

ASSUME PROCESSES USE LOCKS, WHICH ARE VALID VALUES FOR X AND Y.

INITIALIZE X = 0, Y = 1

<table>
<thead>
<tr>
<th>Procession 0</th>
<th>Procession 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMO. ON</td>
<td>AMO. ON</td>
</tr>
<tr>
<td>GET, ADD, SUB</td>
<td>GET, ADD, SUB</td>
</tr>
<tr>
<td>r1, r2, 1</td>
<td>r1, r2, 1</td>
</tr>
<tr>
<td>on</td>
<td>on</td>
</tr>
<tr>
<td>loop</td>
<td>loop</td>
</tr>
<tr>
<td>w r1, x</td>
<td>w r1, x</td>
</tr>
<tr>
<td>w r2, y</td>
<td>w r2, y</td>
</tr>
<tr>
<td>sw r2, x</td>
<td>sw r2, x</td>
</tr>
<tr>
<td>sw r1, y</td>
<td>sw r1, y</td>
</tr>
<tr>
<td>sw r0, z</td>
<td>sw r1, z</td>
</tr>
<tr>
<td></td>
<td>sw r0, z</td>
</tr>
</tbody>
</table>
ALL FOUR VALUES STILL POSSIBLE! LOADS AND STONES MIGHT TAKE EFFECT OUT-OF-ORDER

SEQUENTIAL CONSISTENCY.
WHAT IF WE ASSUME EACH PROCESSOR EXECUTES
ATOMICALLY AND COMPLETELY IN ORDER? IE, WE
USE SYNCHRONIZATION & ASSUME SEQ CONSISTENT. WHAT VALUE
ACCEPTABLE?

Convergence

Evict!

Illustrates need for convergence

Synchronization, consistency, convergence

How does our mutex avoid any issues?
Short Answer Problem

Consider two cache designs:

Baseline Design
- Direct Mapped
- 8 Cache Lines
- 16 B / Cache Line
- 128 B Capacity

Alternative Design
- Direct Mapped
- 4 Cache Lines
- 32 B / Cache Line
- 128 B Capacity

Calculate miss rate for two transaction sequences:

A) 0x0000, 0x0004, 0x0008, 0x000c, 0x0010, ...

B) 0x0000, 0x1014, 0x1018, 0x0000, 0x1014, ...
**First Determine Address Mapping**

**Baseline**

<table>
<thead>
<tr>
<th>25</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>10x off</td>
<td>10x? off</td>
</tr>
</tbody>
</table>

**Alternative**

<table>
<thead>
<tr>
<th>24</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>10x off</td>
<td>10x? off</td>
</tr>
</tbody>
</table>

**Then Determine Miss Rate**

A) First access to first word of cache line will always miss. Remaining accesses to that cache line will always hit.

**Baseline Miss Rate**: $1/4 = 25\%$

**Alternative Miss Rate**: $1/8 = 12.5\%$

B) Know first 3 accesses are compulsory misses, know there will be no capacity misses.

Do these 3 lines conflict in cache or not?

**Baseline**

```
0x0000
0x1014
0x1018
0x0000
0x1014
0x1018
```

All 3 map to same index! BUT DIFFERENT TAGS! MUST CONFLICT IN DIRECT MAPPED CACHE

**Baseline Miss Rate**: 0%

**Alternative Miss Rate**: $2/3 = 66\%$
**ITANiUM 2**

**L1 I#**
- 16 KB, 64 B line size
- 4-way set assoc.
- 1 RD port, 32 B/cycle
- 1 cycle hit latency
- Mostly blocking

**L2 UNIFIED $**
- 256 KB, 128 B line size
- 8-way set assoc.
- 4 RD ports, 16 B/cycle
- 5+ cycle hit latency
- Write back, war alloc.
- 16 BANKS
- Exclusive

**L1 D$**
- 16 KB, 64 B line size
- 4-way set assoc.
- 2 RD ports, 8 B/cycle
- 2 WR ports, 64 B/cycle
- 1 cycle hit latency
- Non-blocking
- Write through

**L3 UNIFIED $**
- 9 MB, 128 B line size
- 18-way set assoc.
- 1 RD port, 32 B/cycle
- 128 B access takes 4 cycles
- 14+ cycle hit latency

---

**Diagram:**
- Blocks labeled with different sizes and access times.
- Connections indicating flow from L1 to L2, L1 to L3, and L3 to memory.

*Rough approximations. Actual parameters much more complicated.*
IBM Power 7

L1 D$
- 32 KB
- 8 way set assoc
- 1 RD port, 16 B/cycle
- 1 cycle hit latency

L2 unified $ (private)
- 256 KB, 128 B line size
- 8 way set assoc
- 8 cycle hit latency
- Write back

L3 unified $ (shared)
- Embedded DRAM
- 32 MB, 128 B line size
- 8 way set assoc
- 8 banks
- Advanced caching line logic/mapping algorithms

Rough approximations
Actual parameters
Much more complicated