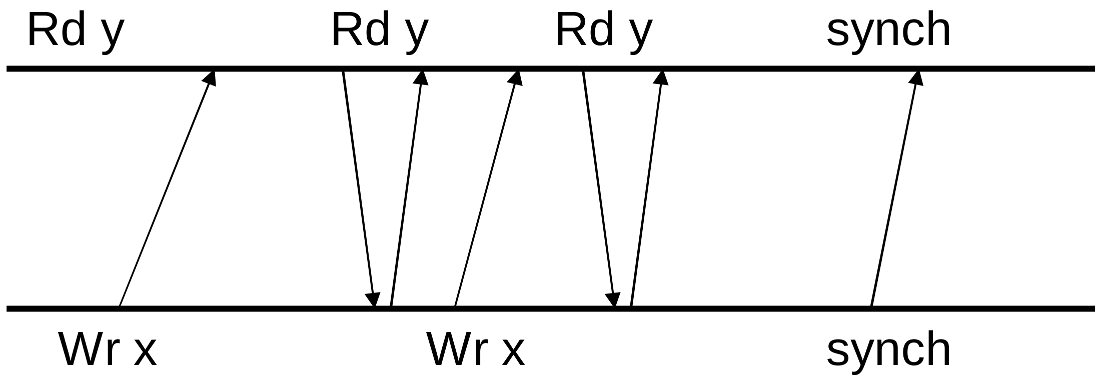
Lecture 13: LRC & Interconnection Networks

- Topics: LRC implementation, interconnection characteristics

Shared Virtual Memory

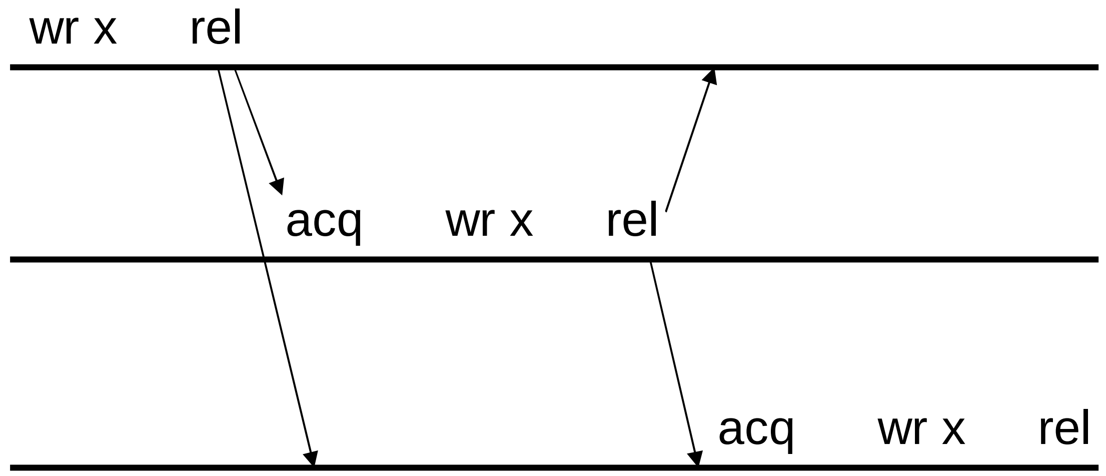


Traffic with hardware CC

Traffic with software CC

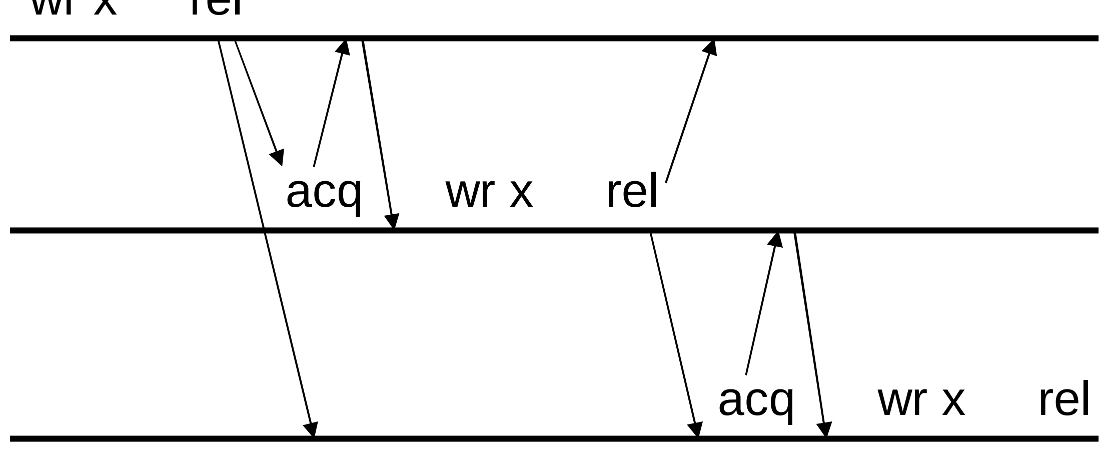
Eager Release Consistency

- Invalidates/Updates are sent out to the list of sharers when a processor executes a release



Lazy Release Consistency

- Invalidates/Updates are sought when a processor executes an acquire – fewer messages, higher implementation complexity



Causality

- Acquires and releases pertain to specific lock variables
- When a process executes an acquire, it should receive all updates that were seen before the corresponding release by the releasing processor
- Therefore, each process must keep track of all write notices (modifications to each shared page) that were applied at every synchronization point

Example

P1	P2	P3	P4
A1			
			A4
R1			
			R4
	A1		
A2			
			A3
	R1		
			R3
R2			
	A3		
		A5	
	R3		
		R5	
	A1		
	R1		
		A1	

Example

$$\begin{tikzcd}
P1 & P2 & P3 & P4 \\
A1 \arrow[dd] & & & \\
 & & & A4 \arrow[dd] \\
R1 \arrow[ddr] & & & \\
 & & & R4 \arrow[ddd] \\
 & A1 \arrow[ddd] & & \\
A2 & & & \\
 & & & A3 \arrow[dd] \\
 & R1 \arrow[ddd] & & \\
 & & & R3 \arrow[ddll] \\
R2 & & & \\
 & A3 \arrow[dd] & & \\
 & & A5 \arrow[dd] & \\
 & R3 \arrow[dd] & & \\
 & & R5 \arrow[ddd] & \\
 & A1 \arrow[d] & & \\
 & R1 \arrow[dr] & & \\
 & & A1 &
\end{tikzcd}$$

Implementation

- Each pair of synch operations in a process defines an *interval*
- A partial order is defined on intervals based on release-acquire pairs
- For each interval, a process maintains a vector timestamp of “preceding” intervals: the vector stores the last preceding interval for each process
- On an acquire, the acquiring process sends its vector timestamp to the releasing process – the releasing process sends all write notices that have not been seen by acquirer

LRC Performance

- LRC can reduce traffic by more than a factor of two for many applications (compared to ERC)
- Programmers have to think harder (causality!)
- High memory overheads at each node (keep track of vector timestamps, write notices) – garbage collection helps significantly
- Memory overheads can be reduced by eagerly propagating write notices to processors or a home node – will change the memory model again!

Interconnection Networks

- Recall: fully connected network, arrays/rings, meshes/tori, trees, butterflies, hypercubes
- Consider a k-ary d-cube: a d-dimension array with k elements in each dimension, there are links between elements that differ in one dimension by 1 (mod k)
- Number of nodes N = k^d

Number of switches :	N	Avg. routing distance:	d(k-1)/2
Switch degree :	2d	Diameter :	d(k-1)
Number of links :	Nd	Bisection bandwidth :	$2wk^{d-1}$
Pins per node :	2wd		

Should we minimize or maximize dimension?

Dimension

- For a fixed machine size N, low-dimension networks have significantly higher latencies for a packet – scalable machines should employ high dimensionality (high cost!)
- For a fixed number of pins, message latency decreases at first, then increases (as we increase dimensionality)
- What if we keep constant bisection bandwidth?
- Lower dimensions also reduce wire length

Number of switches	:	N	Avg. routing distance	:	$d(k-1)/2$
Switch degree	:	2d	Diameter	:	$d(k-1)$
Number of links	:	Nd	Bisection bandwidth	:	$2wk^{d-1}$
Pins per node	:	2wd			

Title

- Bullet