COL380

Introduction to
Parallel & Distributed Programming

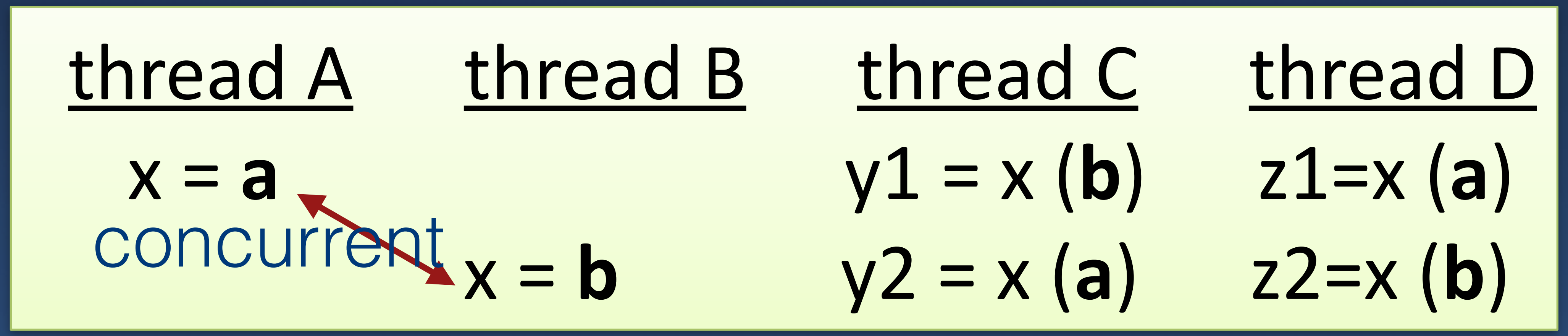
Causal Consistency

- Write is causally ordered after all earlier reads/writes in its thread
  - ➡ write may depends on the current complete ‘state’
- Read is causally ordered after its causative write
- Causality is transitive
- ∃ sequential order of causally related operations consistent with every thread’s view
  - ➡ Non-related writes may be seen in different order by different threads

Causal Consistency

- Write is causally ordered after all earlier reads/writes in its thread
  - ➡ write may depends on the current complete 'state'
- Read is causally ordered after its causative write
- Causality is transitive
- ∃ sequential order of causally related operations consistent with every thread's view
  - ➡ Non-related writes may be seen in different order by different threads

Causally Consistent



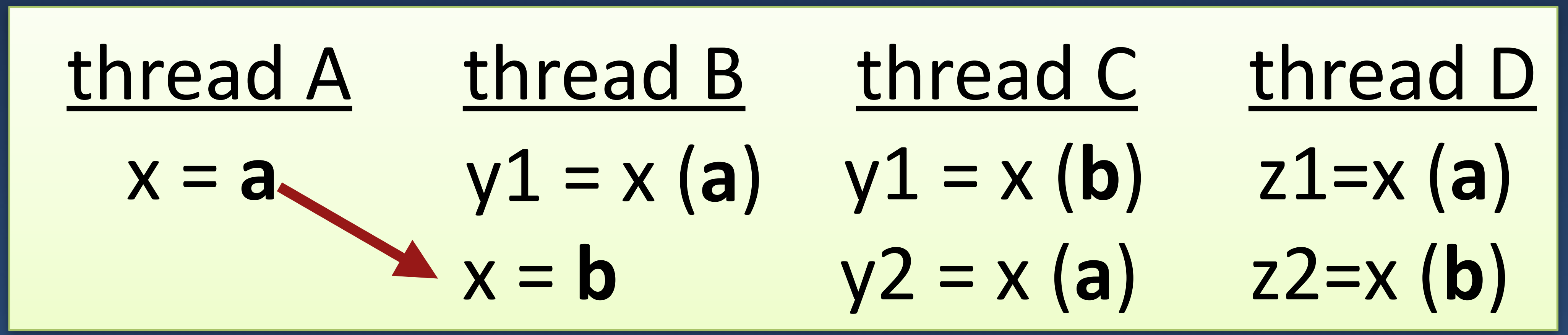
thread A	thread B	thread C	thread D
x = **a**		y1 = x (**b**)	z1=x (**a**)
	x = **b**	y2 = x (**a**)	z2=x (**b**)

concurrent

Causal Consistency

- Write is causally ordered after all earlier reads/writes in its thread
  - ➡ write may depends on the current complete ‘state’
- Read is causally ordered after its causative write
- Causality is transitive
- ∃ sequential order of causally related operations consistent with every thread’s view
  - ➡ Non-related writes may be seen in different order by different threads

Causally Inonsistent



thread A	thread B	thread C	thread D
x = **a**	y1 = x (**a**)	y1 = x (**b**)	z1=x (**a**)
	x = **b**	y2 = x (**a**)	z2=x (**b**)

Processor Consistency

- All threads see all writes by each thread in the order of that thread
  - ➡ all instances of write(**x**) are seen by each thread in the same order
  - ➡ No need to consistently order writes to different variables by different threads
- Easy to implement
  - ➡ Two or more writes from a single source must remain in order, as in a pipeline
  - ➡ All writes are through to the memory

Processor Consistency

- All threads see all writes by each thread in the order of that thread
  - ➡ all instances of write(**x**) are seen by each thread ir

    FIFO consistency relaxes this constraint

  - ➡ No need to consistently order writes to different variables by different threads
- Easy to implement
  - ➡ Two or more writes from a single source must remain in order, as in a pipeline
  - ➡ All writes are through to the memory

Processor Consistency

- All threads see all writes by each thread in the order of that thread
    - all instances of write(**x**) are seen by each thread in the same order
    - No need to consistently order writes to different variables by different threads
- Easy to implement
    - Two or more writes from a single source must remain in order, as in a pipeline
    - All writes are through to the memory

FIFO consistency is also known as **PRAM consistency**

Consistency Summary

Model	Description
Strict	Global time based atomic ordering of *all* shared accesses
Sequential	*All* threads see all shared accesses in the same order consistent with program order -- no centralized ordering
Causal	All threads see causally-related shared accesses in the same order
Processor	All threads see writes from each other in the order they were made. Writes to a variable must be seen in the same order by all threads
Weak	Special synchronization based reordering -- shared data consistent only after synchronization