ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Lecture 16: Shared Registers

CS 539 / ECE 526

Distributed Algorithms

Outline

- Types of Shared Registers
- Algorithms
  - SRSW Boolean Safe → SRSW Boolean Regular
  - SRSW Regular → SRSW Atomic
  - SRSW → MRSW
  - MRSW → MRMW

Types of Shared Registers

- Boolean vs. multi-value

- Single vs. multiple reader (SR, MR)
- Single vs. multiple writer (SW, MW)
  - SRSW: reader is different from writer (otherwise, not distributed)

- Operations (read/write) can overlap

Shared Registers

- If a read does not overlap with a write, return the most recent written value

- If a read overlaps with one or more writes
  - "Safe": can return any value
  - Regular: return the initial value or one of the written values
  - Atomic: provides an illusion that each op happens at some instant [start, end]
    - Operations never overlap

Types of Shared Registers

- Discussion: is safe register too weak?
  - (Note: a "safe" register is very unsafe)

- Without special treatment, a multi-valued register is only safe

– Old value:	011000
– Transient:	001000
– Transient:	001100
– New value:	001110

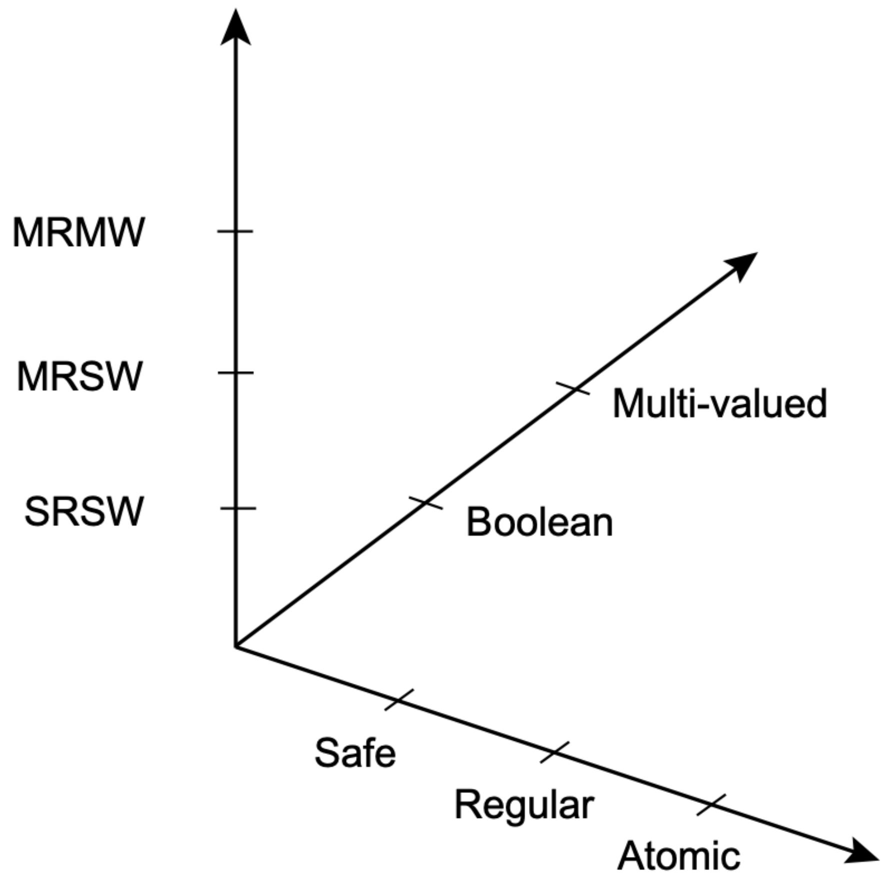
Types of Shared Registers

- Discussion: is safe register too weak?
  - (Note: a "safe" register is very unsafe)

- Without special treatment, a multi-valued register is only safe

– Old value:	011000	011000
– Transient:	001000	011010
– Transient:	001100	011110
– New value:	001110	001110

Space of Shared Registers



Connection to DSM

- Last lecture: distributed algorithms to build share memory with linearizability (atomic consistency)
  - Total order (atomic) broadcast
  - ABD algorithm

- Another way to view these algorithm:
  - Atomic broadcast: MRMW atomic register
  - ABD (so far): MRSW atomic register

Outline

- Types of Shared Registers
- Algorithms
  - SRSW Boolean Safe → Regular
  - SRSW Regular → Atomic
  - SRSW → MRSW
  - MRSW → MRMW

Main Question

- How to implement ”stronger” registers from “weaker” ones?

- Why do we care if we already know how to achieve atomic (“strongest”) registers?
  - Because atomic registers are expensive and real-world systems may implement weaker registers and consistency models

Method 1: Mutual Exclusion

- Topic for next lecture
- Achieve atomicity by preventing overlapping operations altogether
- Downsides:
  - May be “blocked” by other processes for a long time
  - Not crash tolerant

Today: Method 2

- Build ”stronger” registers from “weaker” ones tolerating all but one (n-1) crashes

- An algorithm tolerating n-1 crashes is also said to be *wait-free*: no process waits for any other process
  - Algorithms today will clearly be wait-free

SRSW Boolean Safe → Regular

- Use a single Boolean safe register b

- Read(): return b;

- Write(x):

```
    if b != x        // use a read to check
        b = x;       // perform write only if old != new
```

SRSW Boolean Safe → Regular

- Proof of regularity:
  - Suppose a read overlaps with 1or more writes
  - If all those writes == original value, no actual write occurs → read returns original value
  - If one write != original value, OK for a Boolean regular register to return either 0 or 1

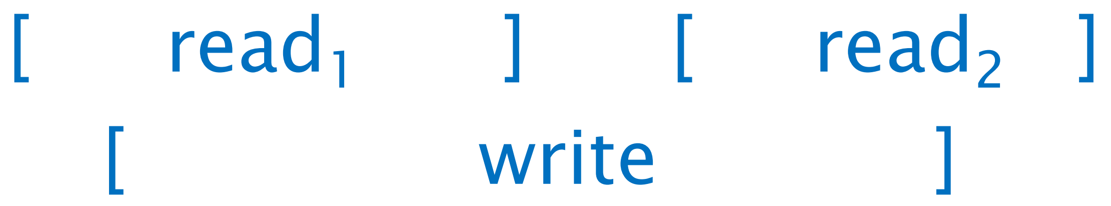
- Efficiency:
  - Memory cost: 1x (none)
  - Read cost: 1x (none)
  - Write cost: one read + one write

Outline

- Types of Shared Registers
- Algorithms
  - SRSW Boolean Safe → SRSW Boolean Regular
  - SRSW Regular → SRSW Atomic
  - SRSW → MRSW
  - MRSW → MRMW

SRSW Regular → Atomic

- First, what is a concrete example that a SRSW regular register fails to be atomic?
  - Recall again that writer != reader



  - $read_1$ returns new value & $read_2$ returns old allowed by regular, disallowed by atomic

SRSW Regular → Atomic

- Augment the value with a timestamp: reg = (ts, val)

- Writer maintains a timestamp ts

```
Write(x):
    ts = ts + 1;
    reg = (ts, x);
```

SRSW Regular → Atomic

- Reader (!= writer) keeps a local copy reg_local = (ts, val) in another regular reg
- Read():

```
    if reg.ts > reg_local.ts:
        reg_local = reg;
    return reg_local.val;
```

To Prove Atomicity/Linearizability

- Find a sequence S’ of operations s.t.
  - Each op is immediately followed by its response
  - Each read returns the preceding write value
  - If op1 ends before op2 starts, then op1 occurs before op2 in S’

SRSW Regular → Atomic

- Each write comes with a ts; Reader reads from local copy, updates local copy upon newer ts.

- Proof: construct S' by ordering ops by their ts; W before R, earlier R before later R
  - Each op followed by its response – by construction
  - Each read returns preceding write – by construction
  - Respect real-time order of ops
    - [R]   or   [R]   ensured by ts of W

              [W]   [W]
    - [R_1] [R_2]: later read "sees" earlier read's ts

SRSW Regular → Atomic

- Each write comes with a ts; Reader reads from local copy, updates local copy upon newer ts.

- Efficiency:
  - Memory cost: 2x
  - Read cost: 2 read + 1 write
  - Write cost: 1x
  - (Ignoring the use of wider register |ts|+|val| vs |val|)

Outline

- Types of Shared Registers
- Algorithms
  - SRSW Boolean Safe → SRSW Boolean Regular
  - SRSW Regular → SRSW Atomic
  - SRSW → MRSW
  - MRSW → MRMW

SRSW → MRSW

- Allocate one SRSW register per reader

- Read(): // by reader i

```
        return Reg[i];
```

- Write(x):

```
        for each i in [1, n]:
                Reg[i] = x;
```

SRSW → MRSW

- Allocate one SRSW register per reader
- Reader reads own copy
- Writer updates all copies

- Works for safe and regular registers
- Efficiency:
  - Memory cost: n (# of reader)
  - Read cost: 1x (none)
  - Write cost: n

SRSW → MRSW

- Allocate one SRSW register per reader
- Reader reads own copy
- Writer updates all copies

- Does not work for atomic registers

```
[ read by j ]   [  read by  k ]
     [      j          write              k  ]
```

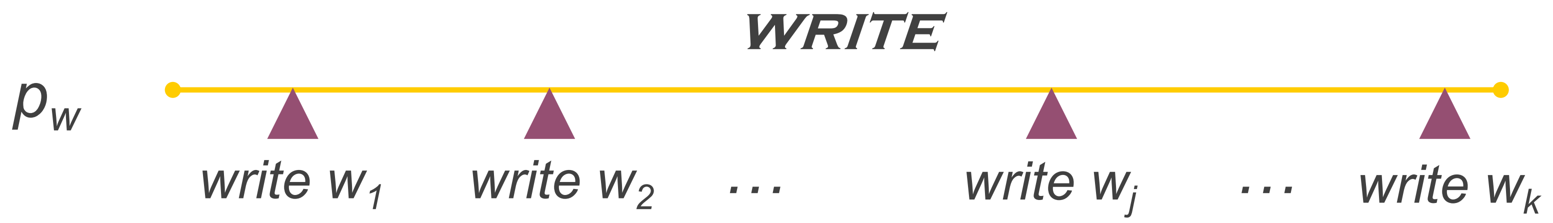
SRSW Atomic → MRSW Atomic

- Lesson: a read needs to ensure later reads see a value that is no older
  - Also the crux in SRSW regular → atomic

- In fact, we can prove the theorem below:
- In a wait-free implementation of MRSW atomic register using SRSW atomic registers, at least one reader must write

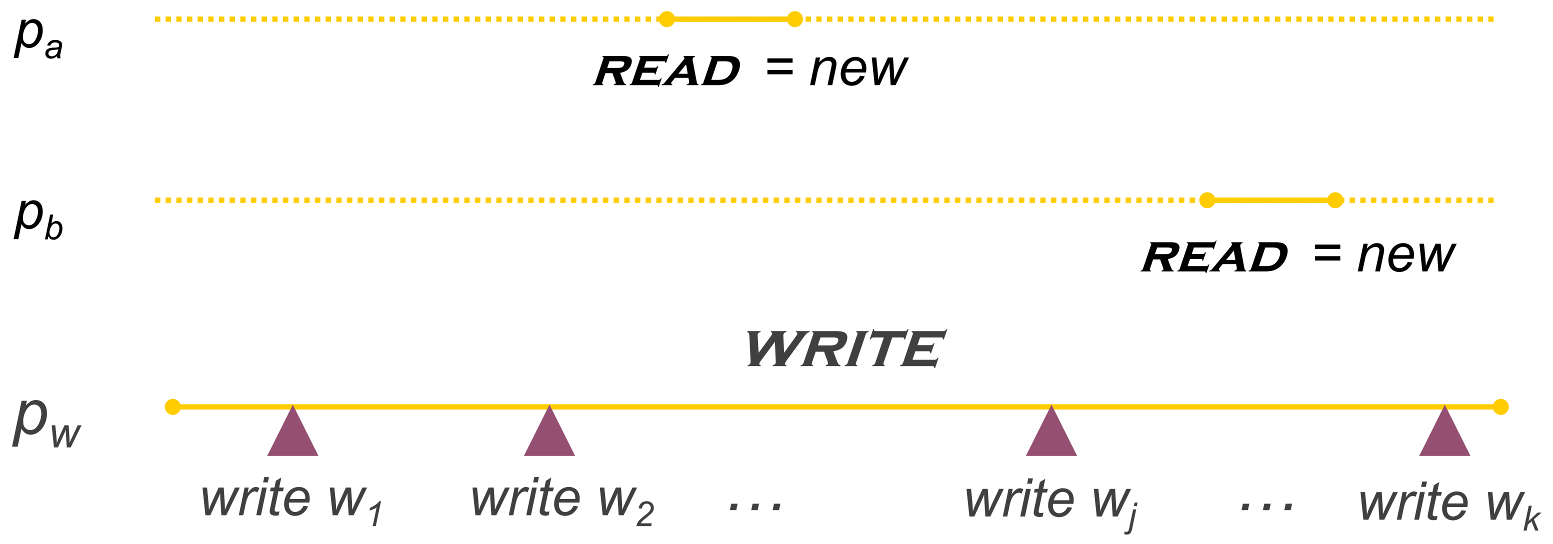
SRSW Atomic → MRSW Atomic

- One reader must write. Proof:
  - Consider writer p_w and two readers p_a, p_b
  - Suppose for contradiction no reader writes
  - A write by p_w performs many low-level writes
  - Each of which *visible to* either p_a or p_b
    - p_a and p_b read from disjoint SR registers



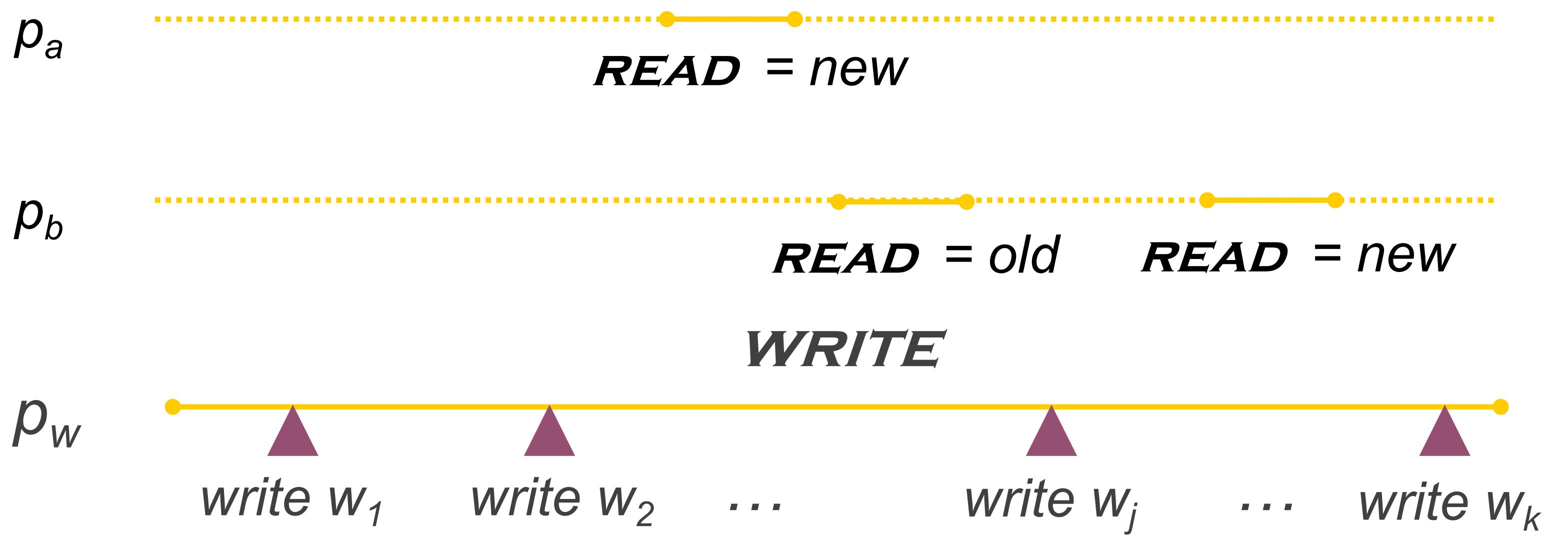
SRSW Atomic → MRSW Atomic

- One reader must write. Proof:
  - Some w_a (w_b) causes p_a (p_b) to see new value
  - a ≠ b
    - Because each write is visible to only 1 reader



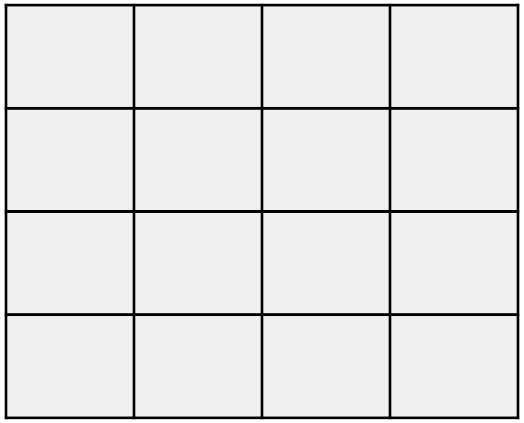
SRSW Atomic → MRSW Atomic

- One reader must write. Proof:
  - Some w_a (w_b) causes p_a (p_b) to see new value
  - $a \neq b$; WLOG $a < b$
  - New-old inversion after w_a → not linearizable



SRSW Atomic → MRSW Atomic

- Lesson: a read needs to ensure later reads see a value that is no older
- Allocate n*n matrix of SRSW atomic
  - (i, j): value reported by reader i to reader j
  - The single writer writes to the diagonal
  - A reader reads a column and writes a row (other than diagonal)



SRSW Atomic → MRSW Atomic

- Write(y):

```
    ts = ts + 1;
    for each i in [1, n]:
        Reg[i][i] = (y, ts);
```

(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)

(y, t+1)	(x, t)	(x, t)	(x, t)
(x, t)	(y, t+1)	(x, t)	(x, t)
(x, t)	(x, t)	(y, t+1)	(x, t)
(x, t)	(x, t)	(x, t)	(y, t+1)

SRSW Atomic → MRSW Atomic

- Read(): // by reader j (j=2 in example)

```
    read entire column j
    write row j with highest ts pair
    return the value with highest ts
```

(y, t+1)	(x, t)	(x, t)	(x, t)
(x, t)	(y, t+1)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)

(y, t+1)	(x, t)	(x, t)	(x, t)
(y, t+1)	(y, t+1)	(y, t+1)	(y, t+1)
(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)

SRSW Atomic → MRSW Atomic

- Read(): // by reader j (j=3 in example)

```
    read entire column j
    write row j with highest ts pair
    return the value with highest ts
```

(y, t+1)	(x, t)	(x, t)	(x, t)
(y, t+1)	(y, t+1)	(y, t+1)	(y, t+1)
(x, t)	(x, t)	(x, t)	(x, t)
(x, t)	(x, t)	(x, t)	(x, t)

(y, t+1)	(x, t)	(x, t)	(x, t)
(y, t+1)	(y, t+1)	(y, t+1)	(y, t+1)
(y, t+1)	(y, t+1)	(x, t)	(y, t+1)
(x, t)	(x, t)	(x, t)	(x, t)

SRSW Atomic → MRSW Atomic

- Proof of atomicity similar:
  - Construct S' by ordering ops by ts, put W before R, earlier R before later R
  - Argue the three conditions for atomicity hold
    - Key step: a read ensures later reads see a value that is no older
- Efficiency:
  - Memory cost: n^2
  - Read cost: n reads + n writes
  - Write cost: n writes

Outline

- Types of Shared Registers
- Algorithms
  - SRSW Boolean Safe → SRSW Boolean Regular
  - SRSW Regular → SRSW Atomic
  - SRSW → MRSW
  - MRSW → MRMW

MRSW Atomic → MRMW Atomic

- Allocate one MRSW atomic register per writer
- Augment with ts

```
Write(x): // by writer i
        Read all registers to find max_ts
        reg[i] = (max_ts+1, x);
Read():
        Read all registers
        Return value with max ts
```

MRSW Atomic → MRMW Atomic

- Can two writes have the same ts?
- Yes! Break ties deterministically using proc ID

```
Write(x): // by writer i
        Read all registers to find max_ts
        reg[i] = (max_ts+1, x);
Read():
        Read all registers
        Return value with max (ts, writer proc ID)
```

MRSW Atomic → MRMW Atomic

- Proof: construct S’ by ordering all writes by (ts, writer proc ID)
- Put each read after the write it reads from, earlier read before later read

- By construction, every op followed by response, and read returns preceding write
- Remains to prove real-time order respected

MRSW Atomic → MRMW Atomic

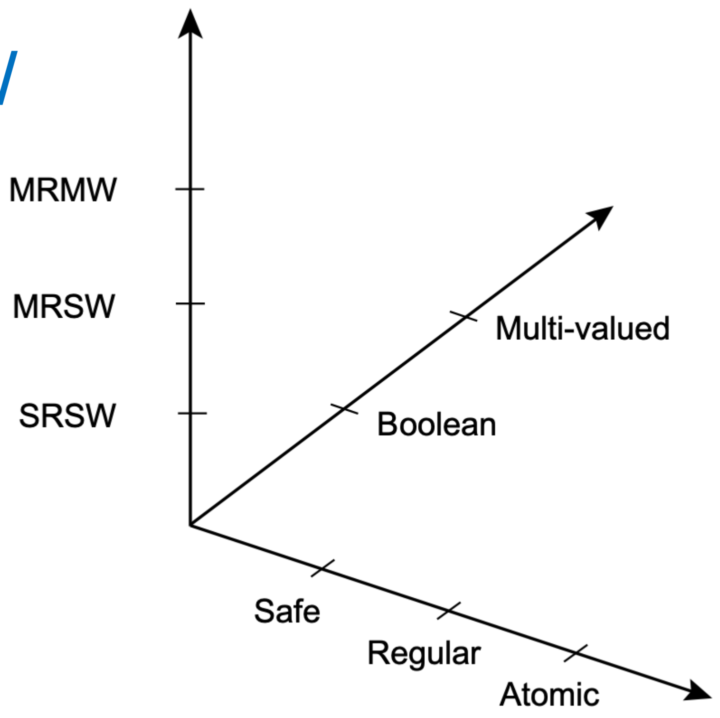
- Remains to prove real-time order respected
  - Case 1: W_1 ends before W_2 begins

  → W_2 reads local copy updated by W_1 & increments ts

  → ts of W_1 < ts of W_2 → W_1 before W_2 in S'
  - Case 2: W ends before R begins → R sees (ts, WID) of W or higher → R after W in S'
  - Case 3: R ends before W begins → W sees (ts, WID) that R reads from or higher, and then increments ts → ts of R < ts of W → R before W in S'
  - Case 4: R_1 ends before R_2 begins → R_2 sees (ts, WID) that R_1 reads from or higher → R_2 after R_1 in S'

Summary

- The strongest register can be built from the weakest with wait-freedom
  - SRSW safe →* regular → atomic
    - *only showed binary
  - SRSW → MRSW → MRMW
  - … also with high costs


MRMW
MRSW
SRSW
Multi-valued
Boolean
Safe
Regular
Atomic