Basics

Introduced in 2011 C/C++ standard.

Concurrency primitives: non-atomic and atomic.

Consistency rules.

Provides platform independent abstraction
Shared Memory Accesses

Non-atomic accesses: Read (R), Write (W)
Atomic accesses = operation + memory order

Operations:
- Read (R)
- Write (W)
- Atomic update (U)
- Fence (F)

Memory orders:
- Relaxed (rlx)
- Release (rel)
- Acquire (acq)
- Acquire-Release (acq_rel)
- Sequentially consistent (sc)

Example:
- X.load(memory order)
- X.store(val, memory order)
- X.CAS(oldval, nwval, success mem order)
- atomic_thread_fence(memory order)
Access Types

Non-atomic read. e.g. \( t = X; \)
- \( R_{na} \)

Non-atomic write. e.g. \( X := v; \)
- \( W_{na} \)

Atomic read. e.g. \( X.load(memory\ order) \)
- \( R_{rlx}, R_{acq}, R_{sc} \)

Atomic write. e.g. \( X.store(val, memory\ order) \)
- \( W_{rlx}, W_{acq}, W_{sc} \)

Atomic update. e.g. \( X.CAS(oldval, nwval, success\ mem\ order) \)
- \( U_{rlx}, U_{rel}, U_{acq}, U_{acq\_rel}, U_{sc} \)

Fence. e.g. \( \text{atomic\_thread\_fence(memory\ order)} \)
- \( F_{rel}, F_{acq}, F_{acq\_rel}, F_{sc} \)
Examples

Initially $X = Y = 0$

$X_{sc} := 1; \quad \parallel \quad Y_{sc} := 1; \quad a := Y_{sc}; \quad // \quad 0 \quad b := Y_{sc}; \quad // \quad 0$

$a = b = 0$ **NOT** allowed.

$X_{rel} := 1; \quad \parallel \quad Y_{rel} := 1; \quad a := Y_{acq}; \quad // \quad 0 \quad b := Y_{acq}; \quad // \quad 0$

$a = b = 0$ is allowed.

$X_{rlx} := 1; \quad \parallel \quad Y_{rlx} := 1; \quad a := Y_{rlx}; \quad // \quad 0 \quad b := Y_{rlx}; \quad // \quad 0$

$a = b = 0$ is allowed.
Exercise

(1) Find out what would happen if we use $R_{rel}$, $W_{acq}$?

(2) Is $a = b = 0$ allowed in the following program?

\[
\begin{align*}
X_{na} & := 1; & Y_{na} & := 1; \\
a & := Y_{na}; & b & := Y_{na};
\end{align*}
\]