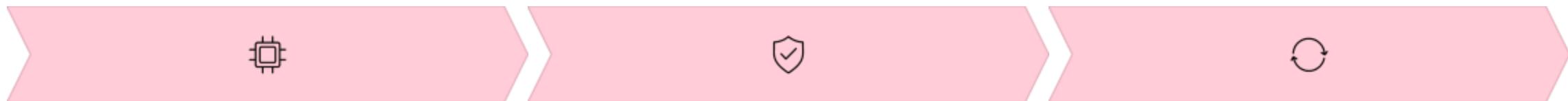


Compiler Optimization Challenges

- Memory consistency models specify the range of legal compiler optimizations on shared data
- In explicitly parallel programs, compilers cannot freely interchange reads and writes of different shared data items without clear synchronization points, as such transformations might affect program semantics
 - This constraint prevents even simple optimizations like register allocation of shared data
- In implicitly parallelized programs like High Performance FORTRAN (HPF), synchronization points are known, avoiding this issue
- Whether compilers can gain significant advantage from more relaxed consistency models remains an open research question.

Speculation Hides Latency in Strict Models



Dynamic Reordering

Processor uses dynamic scheduling to reorder memory references, executing them potentially out of order.

Violation Detection

If invalidation arrives before commit, processor detects potential sequential consistency violations.

Speculative Recovery

Processor backs out computation and restarts with invalidated memory reference, ensuring correct execution.

- Hill advocated sequential or processor consistency with speculative execution because:
 - Aggressive implementation gains most advantages of relaxed models
 - Adds little to implementation cost of speculative processor
 - Allows programmers to reason using simpler programming models
- The MIPS R10000 used out-of-order capability to support aggressive implementation of sequential consistency

Multilevel Cache Inclusion

- Multilevel inclusion means every cache level is a subset of the level further from the processor
 - This reduces contention between coherence traffic and processor traffic when snoops and cache accesses compete for resources
- Many multiprocessors enforce inclusion, though recent designs with smaller L1 caches and different block sizes sometimes choose not to
 - Intel i7 uses inclusion for L3, while AMD Opteron makes L2 inclusive of L1 but not L3.

Inclusion Challenges

Preserving inclusion becomes complex when L1 and L2 have different block sizes. A block in L2 represents multiple blocks in L1, and a miss in L2 causes replacement of data equivalent to multiple L1 blocks.

01

Problem Setup

L2 block size is four times L1. L1 contains blocks at addresses x and $x+b$, where $x \bmod 4b = 0$.

03

Replacement

L2 fetches 4b bytes and replaces blocks x , $x+b$, $x+2b$, $x+3b$. L1 replaces only block x .

02

Miss Occurs

Reference to block y maps to same location as x in both caches, causing miss in both L1 and L2.

04

Violation

L1 still contains $x+b$ but L2 does not, violating the inclusion property.

To maintain inclusion with multiple block sizes, higher cache levels must be probed during lower-level replacements to invalidate replaced words.