IDF2012
INTEL DEVELOPER FORUM

Intel® Transactional Synchronization Extensions

Ravi Rajwar, CPU Architect, Intel
Martin Dixon, Principal Engineer, Intel

ARCS004

Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

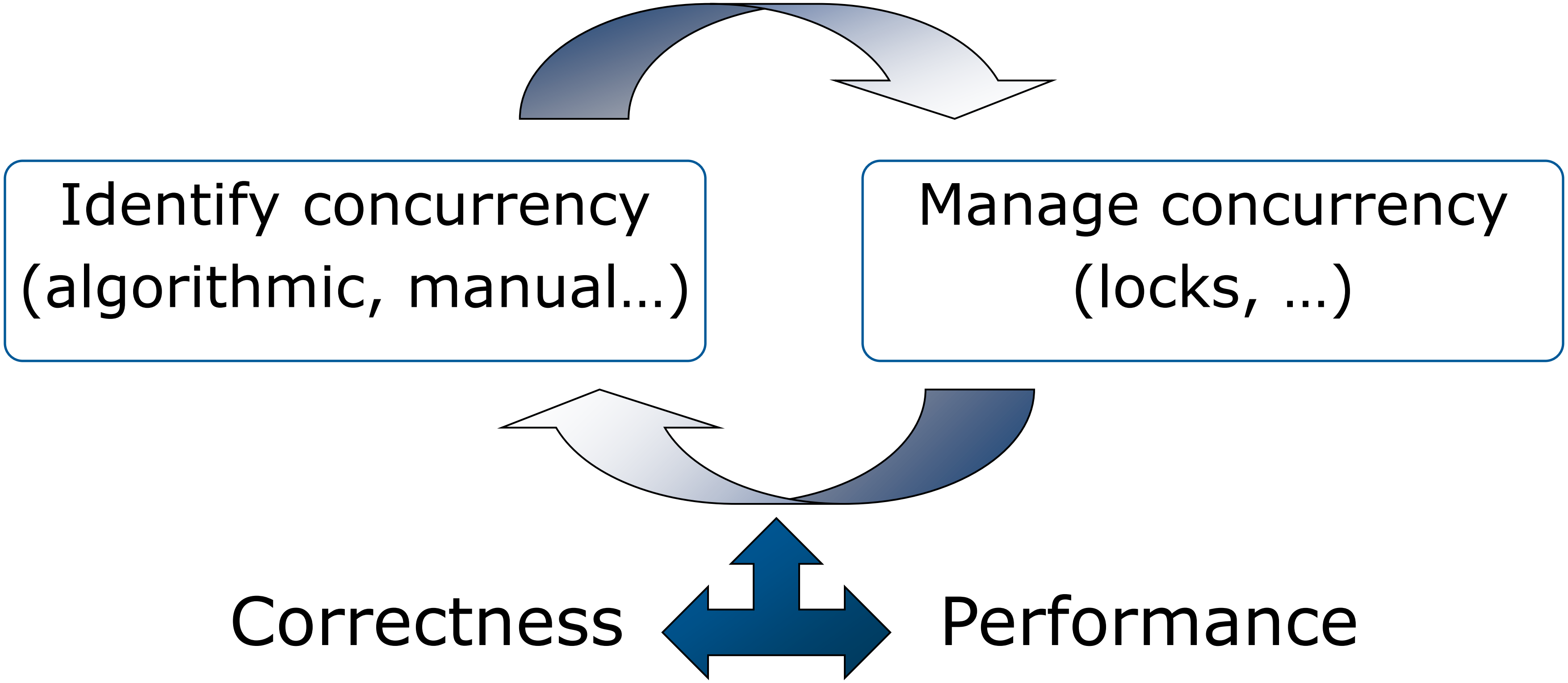
‡ For the next generation Intel® microarchitecture (Haswell)

Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

‡ For the next generation Intel® microarchitecture (Haswell)

Difficulty of Software Development



Identify concurrency (algorithmic, manual…)

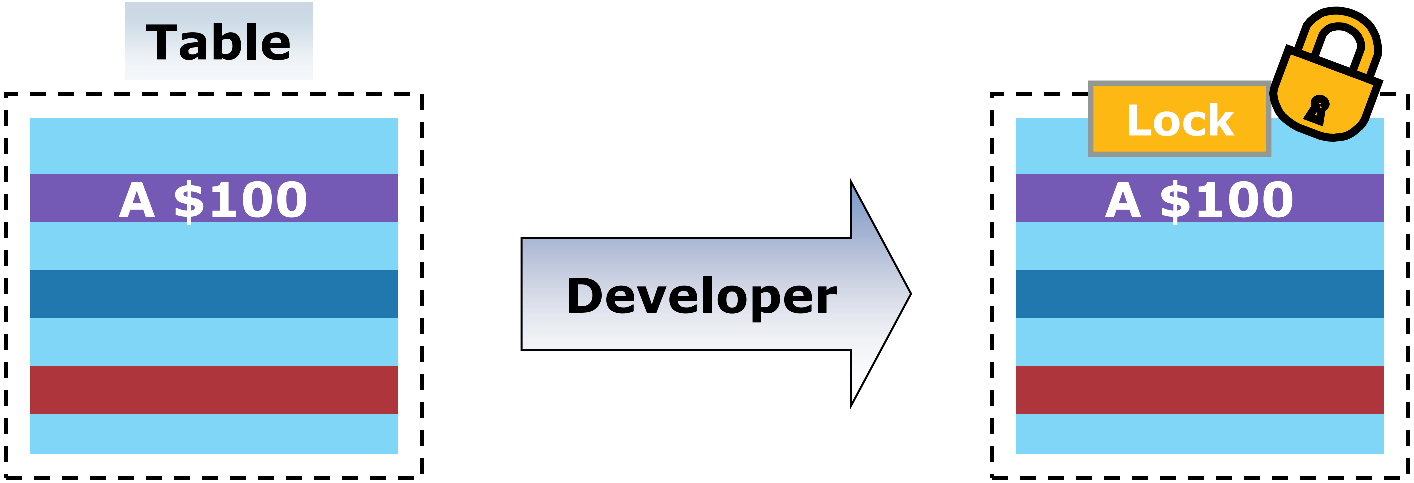
Manage concurrency (locks, …)

Correctness

Performance

Hard to Write Fast and Correct Multi-Threaded Code

The Need for Synchronization



Alice wants $50 from A

- A was $100, A is now $50

Bob wants $60 from A

- A was $100, A is now $40

A should be -10

Alice wants $50 from A

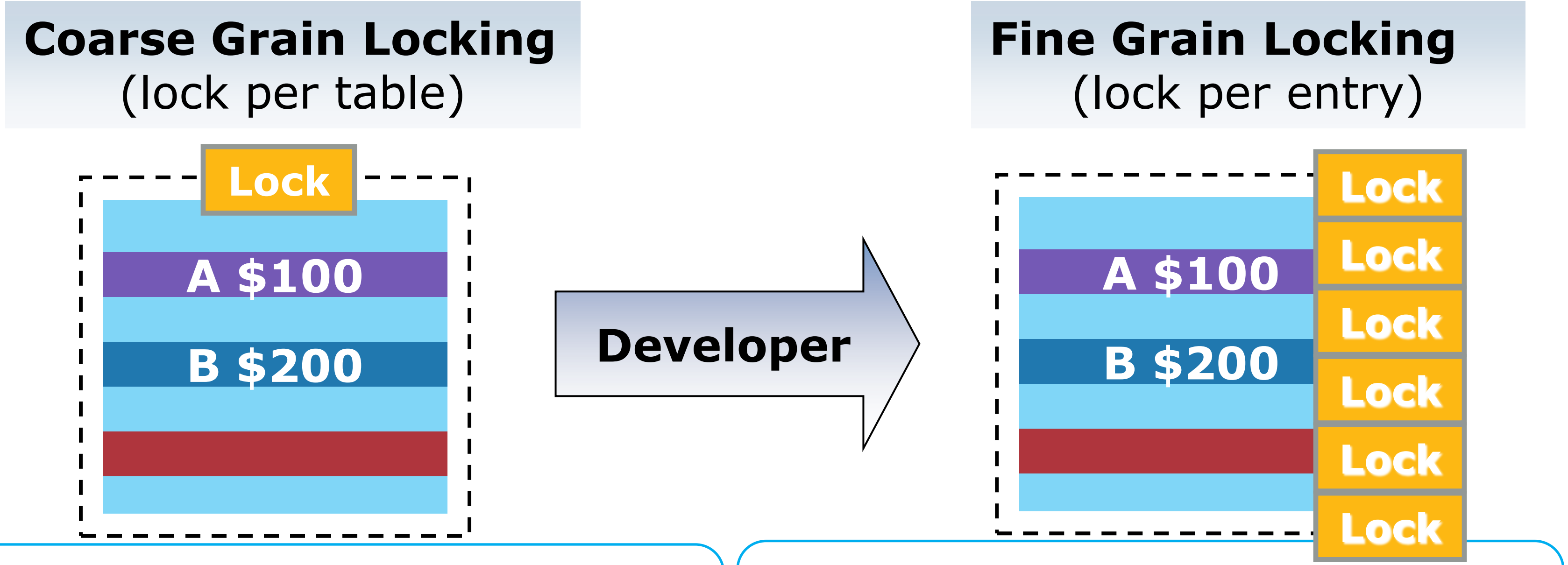
- Alice locks table
- A was $100, A is now $50

Bob wants $60 from A

- Bob waits till lock release
- A was $50, Insufficient funds

Bob and Alice saw A as $100. Locks prevent such data races

Lock Granularity Optimization



Alice withdraws $20 from A

- Alice locks table

Bob wants $30 from B

- Waits for Alice to free table

Alice withdraws $20 from A

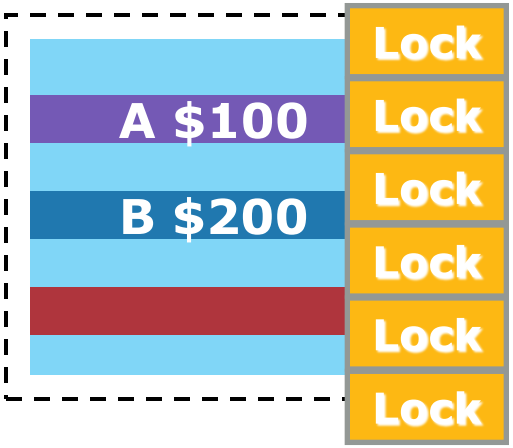
- Alice locks A

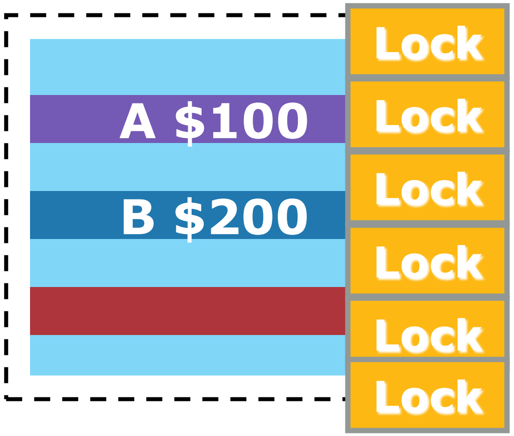
Bob wants $30 from B

- Bob locks B

Such Tuning is Time Consuming and Error Prone

Complexity of Fine Grain Locking





Alice transfers $20 from A to B

- Alice locks A and locks B

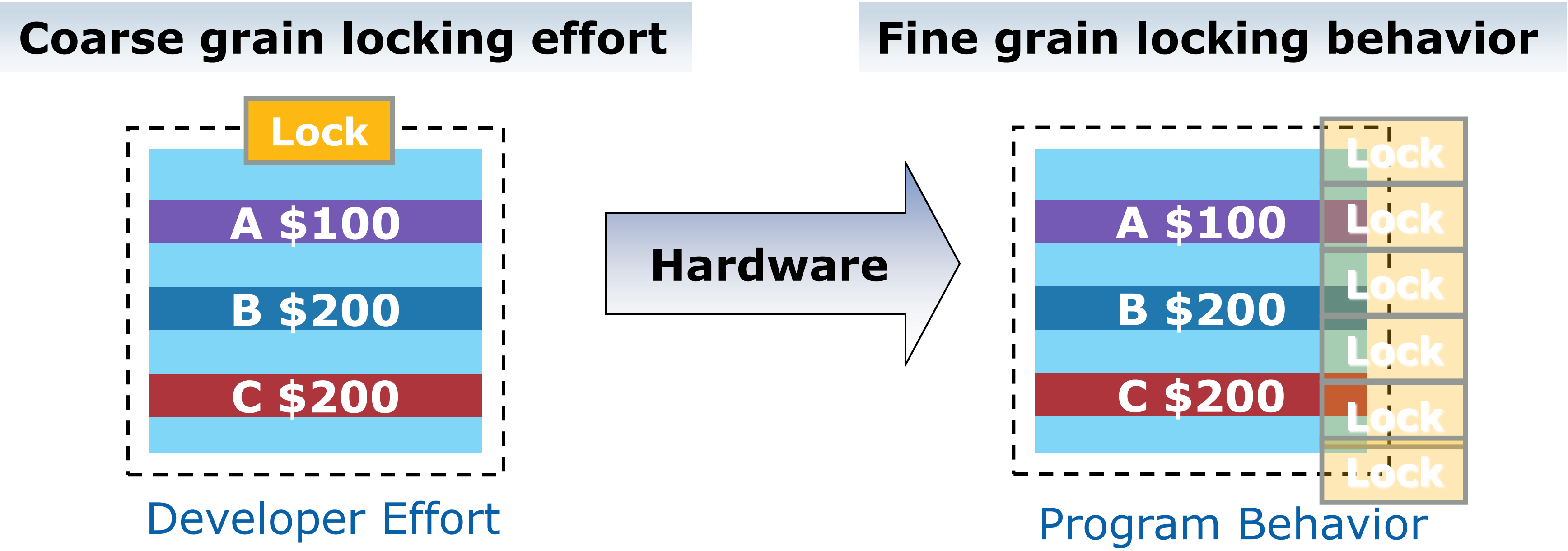
Performs transfer

- Alice unlocks A and unlocks B

Alice transfers $20 from A to B	Bob transfers $50 from B to A
Locks A	Locks B
Cannot lock B	Cannot lock A

Expensive and Difficult to Debug Millions of Lines of Code

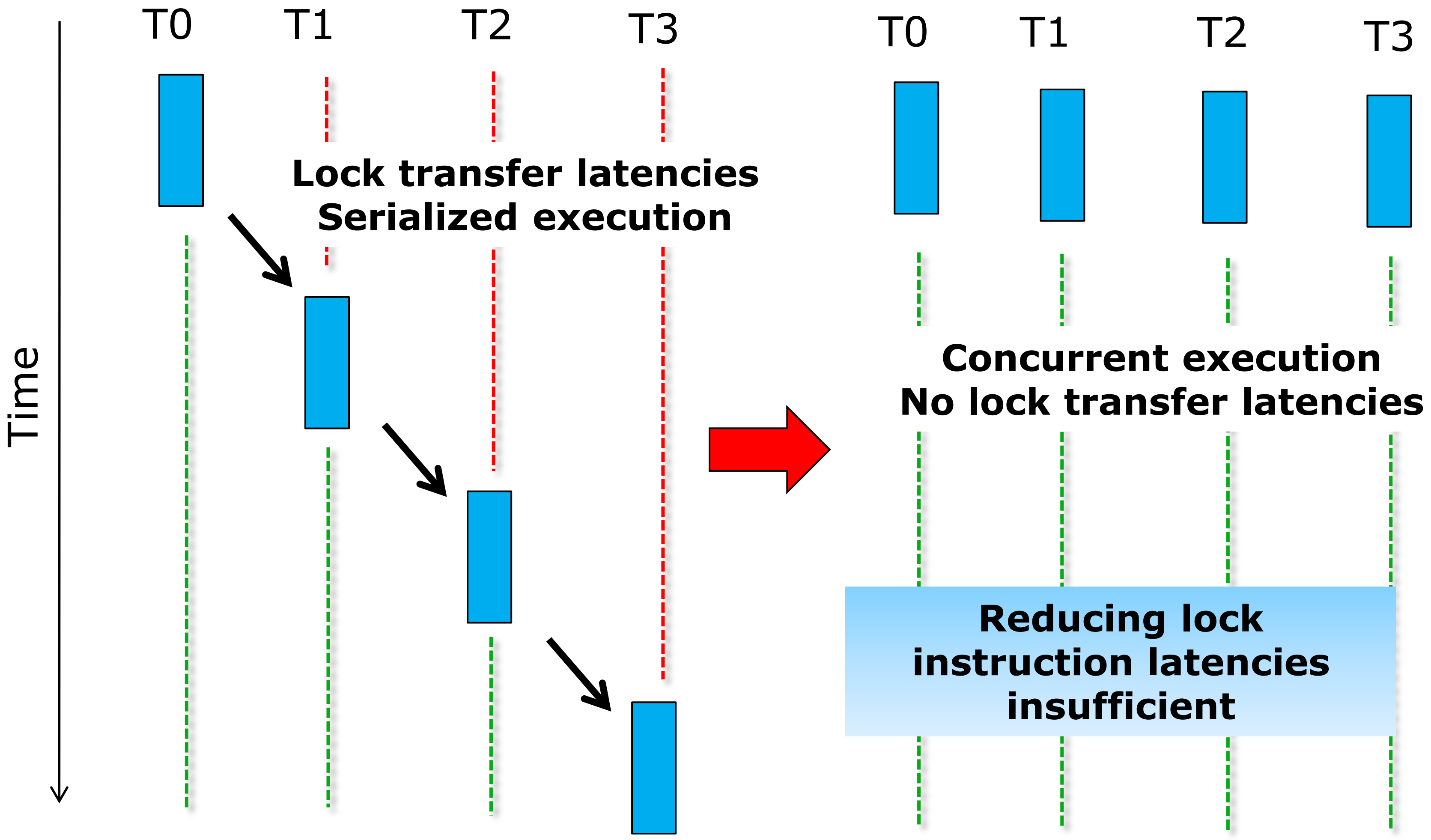
What We Really Want…



- Developer uses coarse grain lock
- Hardware elides the lock to expose concurrency
  - Alice and Bob don’t serialize on the lock
  - Hardware automatically detects real data conflicts

Lock Elision: Fine Grain Behavior at Coarse Grain Effort

Benefit of Lock Elision



Lock transfer latencies
Serialized execution

Concurrent execution
No lock transfer latencies

Reducing lock instruction latencies insufficient

Exposes Concurrency & Eliminates Unnecessary Communication

Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

‡ For the next generation Intel® microarchitecture (Haswell)

Transactional Synchronization

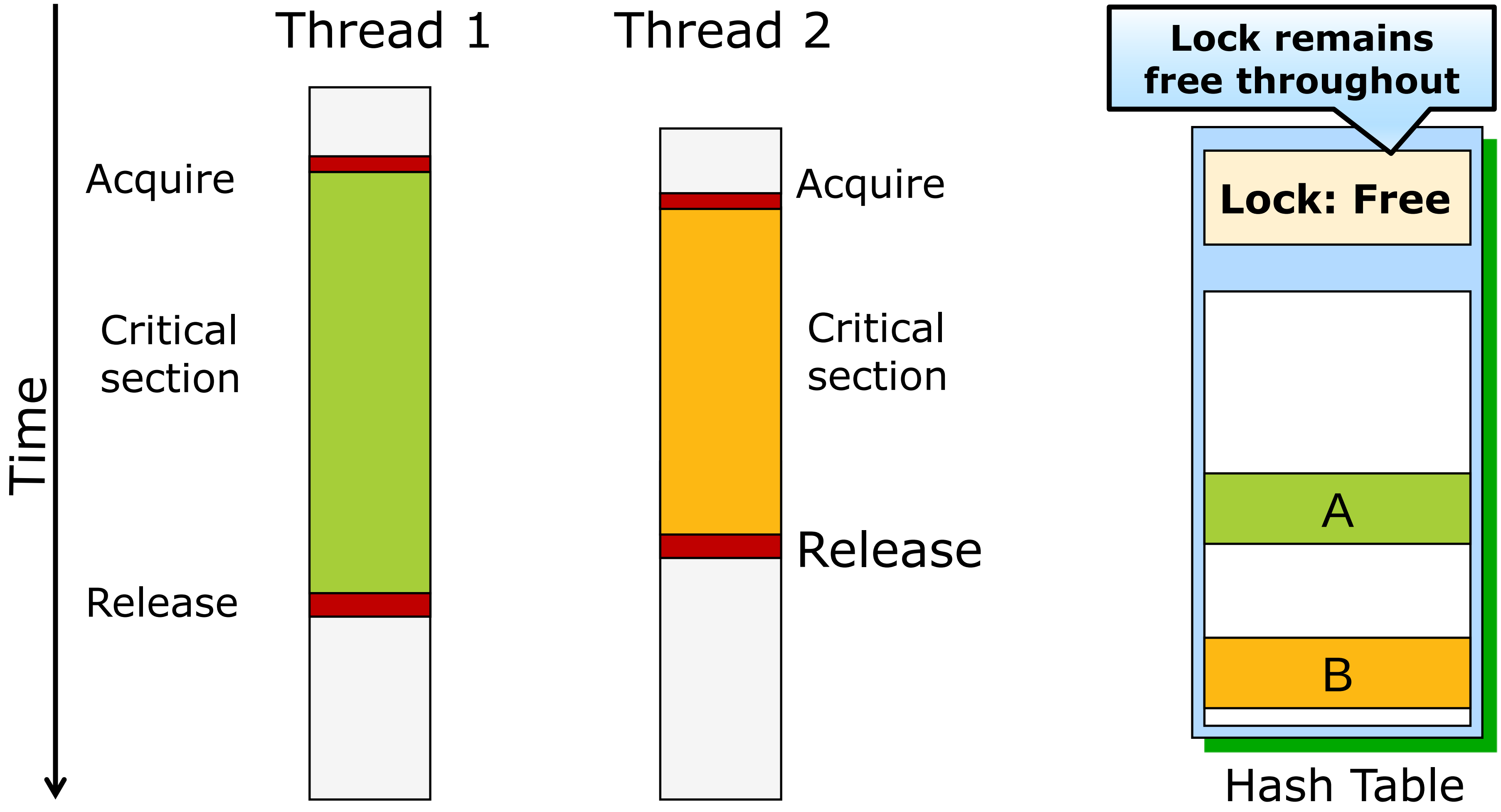
- **Hardware support to enable lock elision**
  - Focus on lock granularity optimizations
  - Fine grain performance at coarse grain effort
- **Intel® TSX: Instruction set extensions for IA‡**
  - Transactionally execute lock-protected critical sections
  - Execute without acquiring lock → expose hidden concurrency
  - Hardware manages transactional updates – All or None
    - Other threads can't observe intermediate transactional updates
    - If lock elision cannot succeed, restart execution & acquire lock

Intel® TSX Exposes Concurrency through Lock Elision

‡Intel® Architecture Instruction Set Extensions Programming Reference (http://software.intel.com/file/41604)

Intel® Transactional Synchronization Extensions (Intel® TSX)

A Canonical Intel® TSX Execution



Lock remains free throughout

No Serialization and No Communication if No Data Conflicts

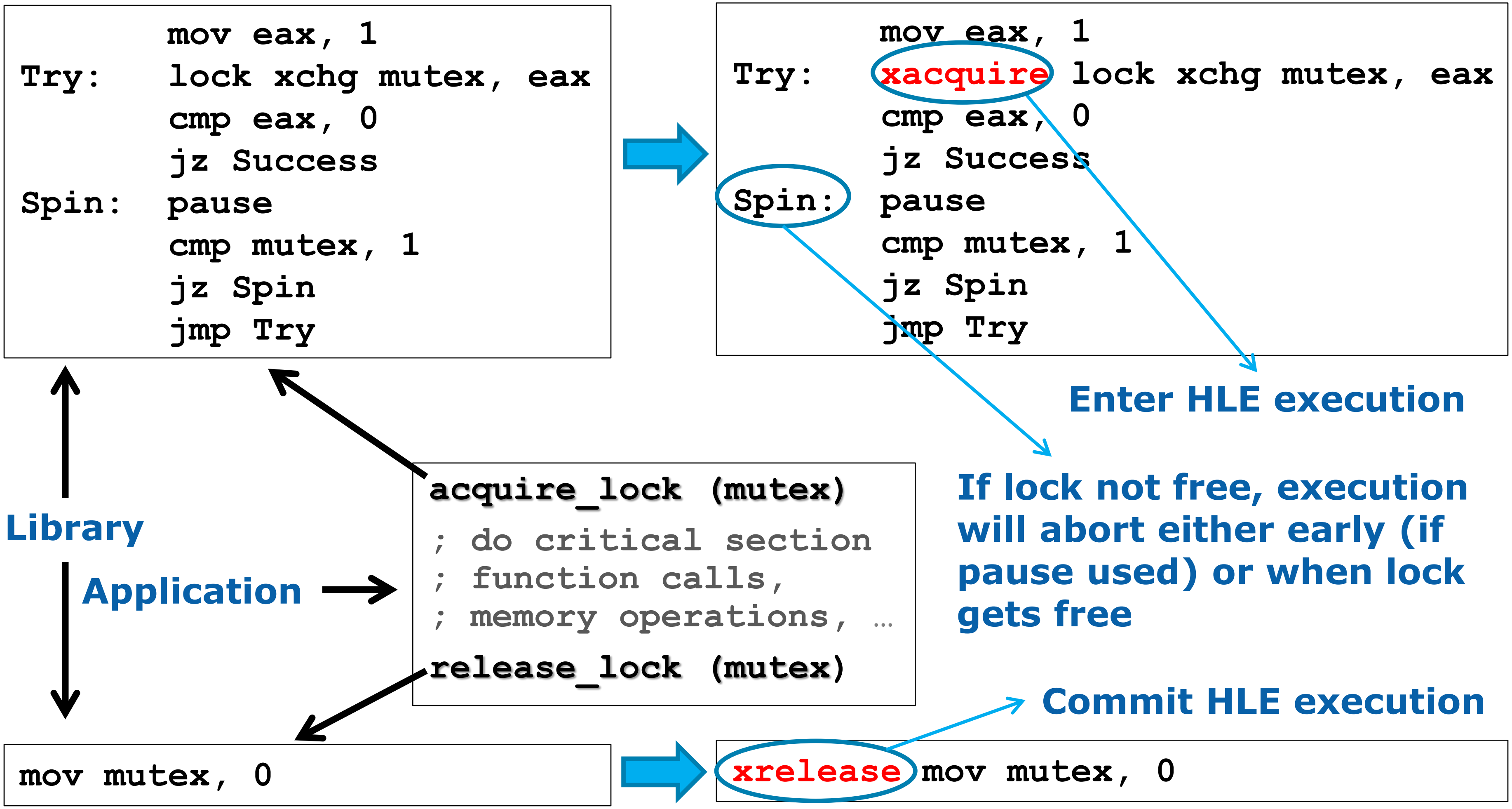
IDF2012 INTEL DEVELOPER FORUM

Intel® TSX Interfaces for Lock Elision

- **Hardware Lock Elision (HLE) – XACQUIRE/XRELEASE**
  - Software uses legacy compatible hints to identify critical section. Hints ignored on hardware without TSX
  - Hardware support to execute transactionally without acquiring lock
  - Abort causes a re-execution without elision
  - Hardware manages all architectural state
- **Restricted Transactional Memory (RTM) – XBEGIN/XEND**
  - Software uses new instructions to specify critical sections
  - Similar to HLE but flexible interface for software to do lock elision
  - Abort transfers control to target specified by XBEGIN operand
  - Abort information returned in a general purpose register (EAX)
- **XTEST and XABORT – Additional instructions**

Flexible and Easy To Use

Intel® TSX Interface: HLE



```
        mov eax, 1
Try:    lock xchg mutex, eax
        cmp eax, 0
        jz Success
Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

```
        mov eax, 1
Try:    xacquire lock xchg mutex, eax
        cmp eax, 0
        jz Success
Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

Enter HLE execution

If lock not free, execution will abort either early (if pause used) or when lock gets free

Library

Application

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

Commit HLE execution

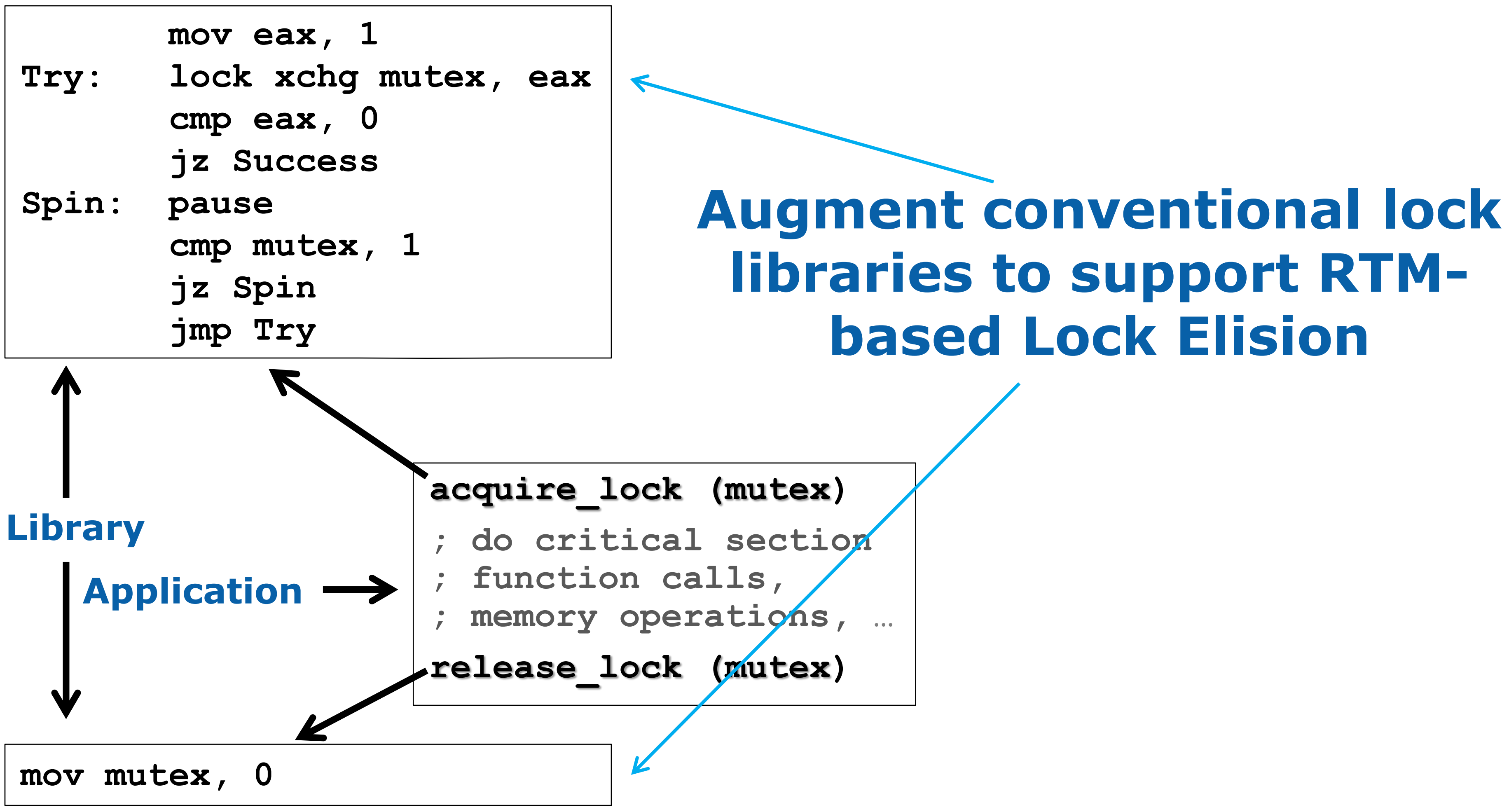
```
mov mutex, 0
```

```
xrelease mov mutex, 0
```

Legacy Compatible Enabling Within Libraries

Code example for illustration purpose only

Intel® TSX Interface: RTM



```
        mov eax, 1
Try:    lock xchg mutex, eax
        cmp eax, 0
        jz Success
Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

Augment conventional lock libraries to support RTM-based Lock Elision

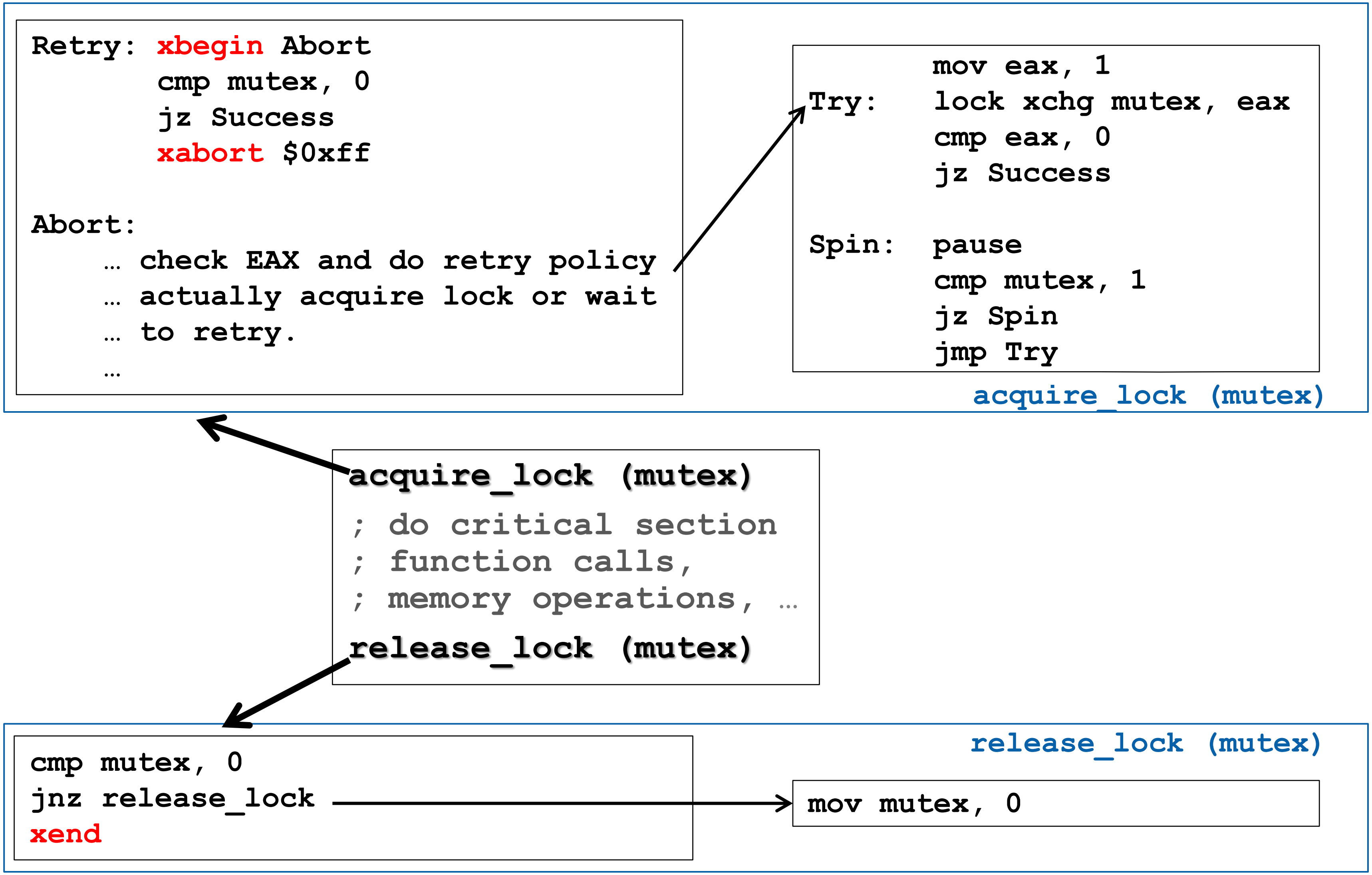
```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

```
mov mutex, 0
```

Lock Elision using RTM Also Enabled Inside Libraries

Code example for illustration purpose only

Intel® TSX Interface: RTM



```
Retry: xbegin Abort
       cmp mutex, 0
       jz Success
       xabort $0xff

Abort:
    … check EAX and do retry policy
    … actually acquire lock or wait
    … to retry.
    …
```

```
        mov eax, 1
Try:    lock xchg mutex, eax
        cmp eax, 0
        jz Success

Spin:   pause
        cmp mutex, 1
        jz Spin
        jmp Try
```

acquire_lock (mutex)

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

release_lock (mutex)

```
cmp mutex, 0
jnz release_lock
xend
```

```
mov mutex, 0
```

Code example for illustration purpose only

Intel® TSX Interface: RTM

```
Retry: xbegin Abort
       cmp mutex, 0
       jz Success
       xabort $0xff

Abort:
     … check EAX and do retry policy
     … actually acquire lock or wait
     … to retry.
     …
```

… Enter RTM execution, Abort is fallback path

… Check to see if mutex is free

… Abort transactional execution if mutex busy

… Fallback path in software

… Retry RTM or explicitly acquire mutex

```
acquire_lock (mutex)
; do critical section
; function calls,
; memory operations, …
release_lock (mutex)
```

```
cmp mutex, 0
jnz release_lock
xend
```

… Mutex not free →was not an RTM execution

… Commit RTM execution

Code example for illustration purpose only

Intel® TSX Usage Environment

- **Available in all x86 modes**
- **Some instructions and events may cause aborts**
  - Uncommon instructions, interrupts, faults, etc.
  - Always functionally safe to use any instruction
- **Software must provide a non-transactional path**
  - HLE: Same software code path executed without elision
  - RTM: Software fallback handler must provide alternate path

Architected for Typical Lock Elision Usage

Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

‡ For the next generation Intel® microarchitecture (Haswell)

Focus: Simplify Developer Enabling

- **Easy to start using Intel® TSX**
  - Simple and clean instruction set minimizes software changes
    - Can be hidden in software synchronization libraries
    - Supports nesting critical sections
  - Minimizes implementation-specific causes for aborts
    - Micro-architectural events such as branch mispredicts, cache misses, TLB misses, etc. do not cause aborts
    - No explicit limit on number of instructions inside critical section
- **Simplify decision process of when to use**
  - Designed to support typical critical sections
  - Competitive to typical uncontended critical sections

Developer Focused Architecture and Design

Implementation specific to the next general Intel® microarchitecture code name Haswell
Intel® Transactional Synchronization Extensions (Intel® TSX)

Intel® TSX Operational Aspects

1. **Identify and elide**
   - Identify critical section, start transactional execution
   - Elide locks, keep them available to other threads

2. **Execute transactionally**
   - Manage all transactional state updates

3. **Detect conflicting memory accesses**
   - Track data accesses, check for conflicts from other threads

4. **Abort or commit**
   - Abort discards all transactional updates
   - Commit makes transactional updates instantaneously visible

Implementation specific to the next general Intel® microarchitecture code name Haswell

Identify and Elide: HLE

mutex value

	self	*others*
mutex:	0	0
mutex:	1	0

```
xacquire lock cmpxchg mutex, ebx
```

- Hardware executes XACQUIRE hint
- Hardware elides acquire write to mutex
- Hardware starts transactional execution

```
mov ecx, mutex
```

- Reading mutex in critical section sees last value written (1)
- Other threads reading see free value (0)

mutex:	1	0
mutex:	0	0

```
xrelease mov mutex, 0
```

- Hardware executes XRELEASE hint
- Hardware elides release write to mutex
- Hardware commits transactional execution

Hardware Automatically Manages Elided Locks

Implementation specific to the next general Intel® microarchitecture code name Haswell

Identify and Elide: HLE

- **Hardware support to elide multiple locks**
  - Hardware elision buffer manages actively elided locks
  - XACQUIRE/XRELEASE allocate/free elision buffer entries
  - Skips elision without aborting if no free entry available
- **Hardware treats XACQUIRE/XRELEASE as hints**
  - Functionally correct even if hints used improperly
  - Hardware checks if locks meet requirements for elision
  - May expose latent bugs and incorrect timing assumptions

Hardware Management of Elision Enables Ease of Use

Implementation specific to the next general Intel® microarchitecture code name Haswell

Identify and Elide: RTM

mutex value

	self	*others*
mutex:	0	0
mutex:	0	0

`xbegin <fallback_path>`

- Hardware executes XBEGIN
- Hardware starts transactional execution
- Software checks for a free mutex, skips acquire

`mov ecx, mutex`

- Reading mutex in critical section sees 0
- Other threads reading see free value (0)

mutex:	0	0
mutex:	0	0

`xend`

- Hardware executes XEND
- Hardware commits transactional execution

RTM Provides Increased Flexibility for Software

Execute Transactionally

- **State updated during transactional execution**
  - State includes registers and memory
  - Hardware recovers register and memory state on aborts
- **Hardware manages all transactional updates**
  - Other threads cannot observe any intermediate updates
  - If lock elision cannot succeed, hardware restarts execution
  - Hardware discards all intermediate updates prior to restart

Software Does Not Worry About State Recovery

Implementation specific to the next general Intel® microarchitecture code name Haswell

Execute Transactionally – Memory

- **Buffering memory writes**
  - Hardware uses L1 cache to buffer transactional writes
    - Writes not visible to other threads until after commit
    - Eviction of transactionally written line causes abort
  - Buffering at cache line granularity

- **Sufficient buffering for typical critical sections**
  - Cache associativity can occasionally be a limit
  - Software always provides fallback path in case of aborts

Hardware Manages All Transactional Writes

IDF2012
INTEL DEVELOPER FORUM

Detect Conflicts

- **Read and write addresses for conflict checking**
  - Tracked at cache line granularity using physical address
  - L1 cache tracks addresses written to in transactional region
  - L1 cache tracks addresses read from in transactional region
    - Cache may evict address without loss of tracking
- **Data conflicts**
  - Occurs if at least one request is doing a write
  - Detected at cache line granularity
  - Detected using existing cache coherence protocol
  - Abort when conflicting access detected

Hardware Automatically Detects Conflicting Accesses

Implementation specific to the next general Intel® microarchitecture code name Haswell

Abort or Commit

- **Transactional abort**
  - Occurs when abort condition is detected
  - Hardware discards all transactional updates

- **Transactional commit**
  - Hardware makes transactional updates visible instantaneously
  - No cross-thread/core/socket coordination required

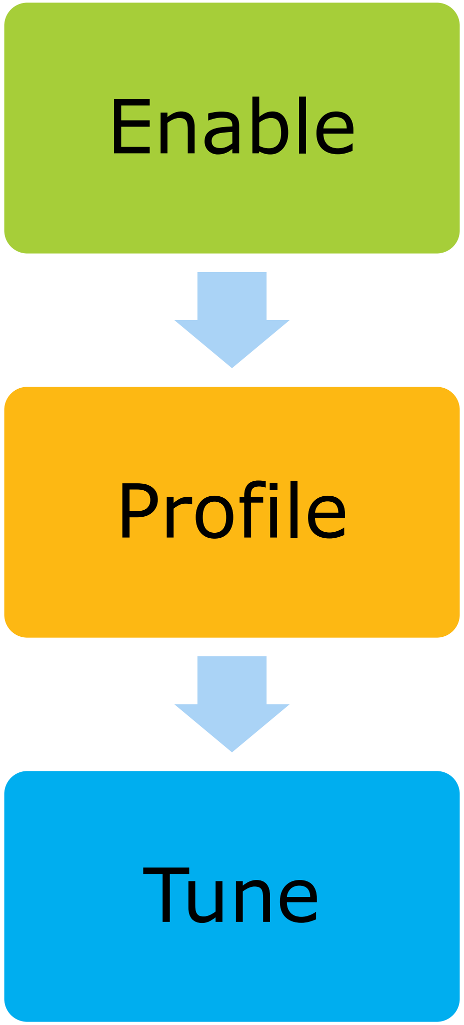
No Global Communication for Commit and Abort

Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

‡ For the next generation Intel® microarchitecture (Haswell)

Software



Enable

Architected for enabling ease

Profile

Extensive performance monitoring
and
profiling support

Tune

Easy to pin-point problem spots
Low touch changes

Software Enabling

- **Doesn't need operating system changes to use**
- **Compiler support through intrinsics and inline assembly**
  - Intel® Compiler (ICC) (v13.0)
  - GCC (v4.8)
  - Microsoft* VS2012
- **Various managed runtimes**
  - Enabling inside runtime, hidden from application developer
- **Changes can be localized to libraries**
  - Augment existing lock library to support Intel® TSX-based elision
  - Dynamic linking ➔ no need to recompile
    - Example: Linux GLIBC for pthreads (rtm-2.17 branch)

Easy to Get Started with Intel® TSX

Profiling

- **Extensive support for performance monitoring**
- **Performance Counters**
  - Count various Intel® TSX specific events
  - Count events within transactional regions
  - Gives first order look into transactional region characteristics
- **Performance Profiling**
  - Extensions to Precise Event Based Sampling (PEBS)
  - Allows detailed profiling of transactional aborts
    - Includes cycles in aborted transactional regions

See Intel Software Developer Manual for More Details

IDF2012
INTEL DEVELOPER FORUM

Software Considerations

- **Good coding practices will also help Intel® TSX**
  - Avoid false or inadvertent sharing
  - Avoid timing based synchronization
- **Most common locks are already elision friendly**
  - Some locks need effort to make them elision friendly
  - RTM provides improved flexibility
- **Not everything can or should use Intel® TSX**
- **Intel® TSX is not a magic bullet**

Watch for the Programmer Optimization Guide

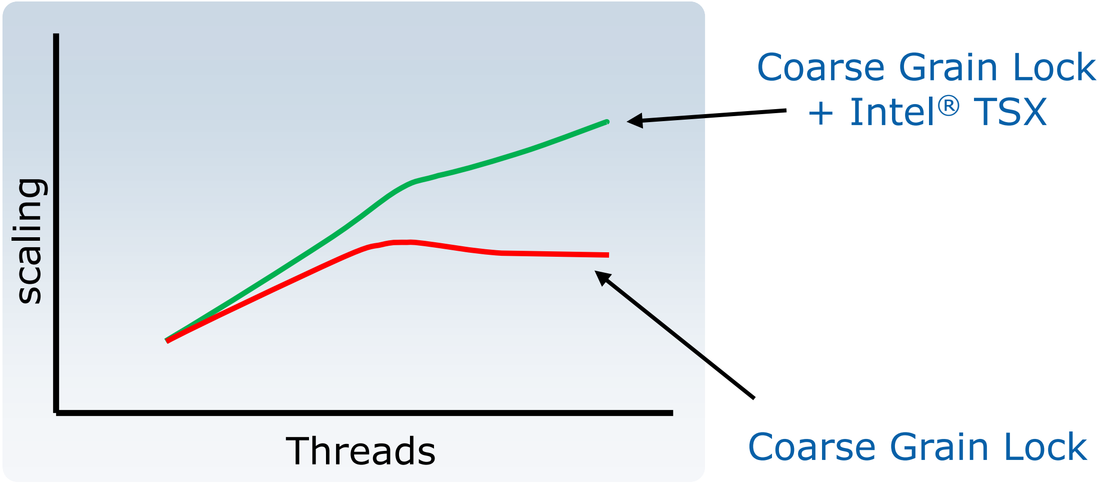
Agenda

- The Synchronization Problem
- Intel® Transactional Synchronization Extensions
- Implementation Insights‡
- Software Enabling and Considerations
- Summary

‡ For the next generation Intel® microarchitecture (Haswell)

Applying Intel® TSX

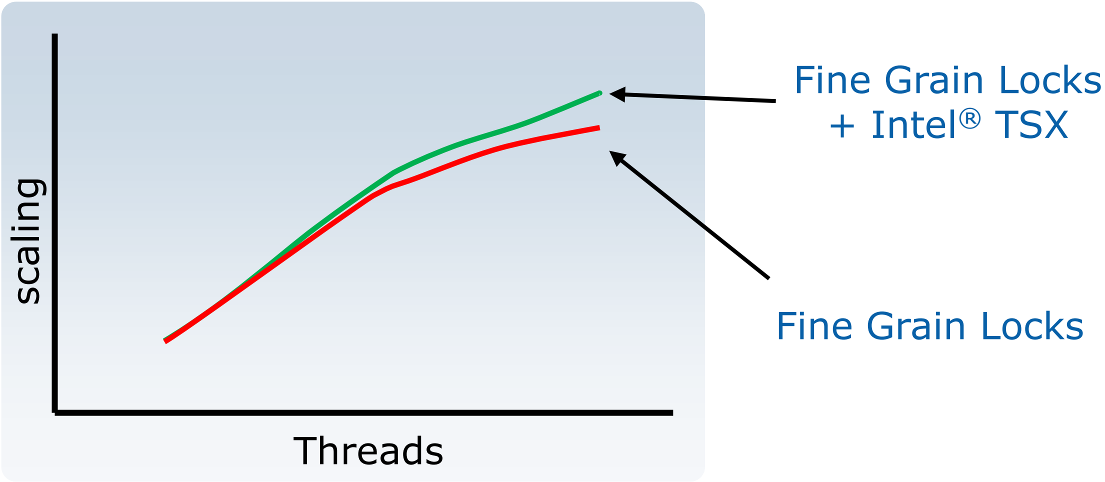
Application with Coarse Grain Lock



Coarse Grain Lock + Intel® TSX

Coarse Grain Lock

Application re-written with Finer Grain Locks



Fine Grain Locks + Intel® TSX

Fine Grain Locks

An example of secondary benefits of Intel® TSX

Fine Grain Behavior at Coarse Grain Effort

Enabling Simpler Algorithms

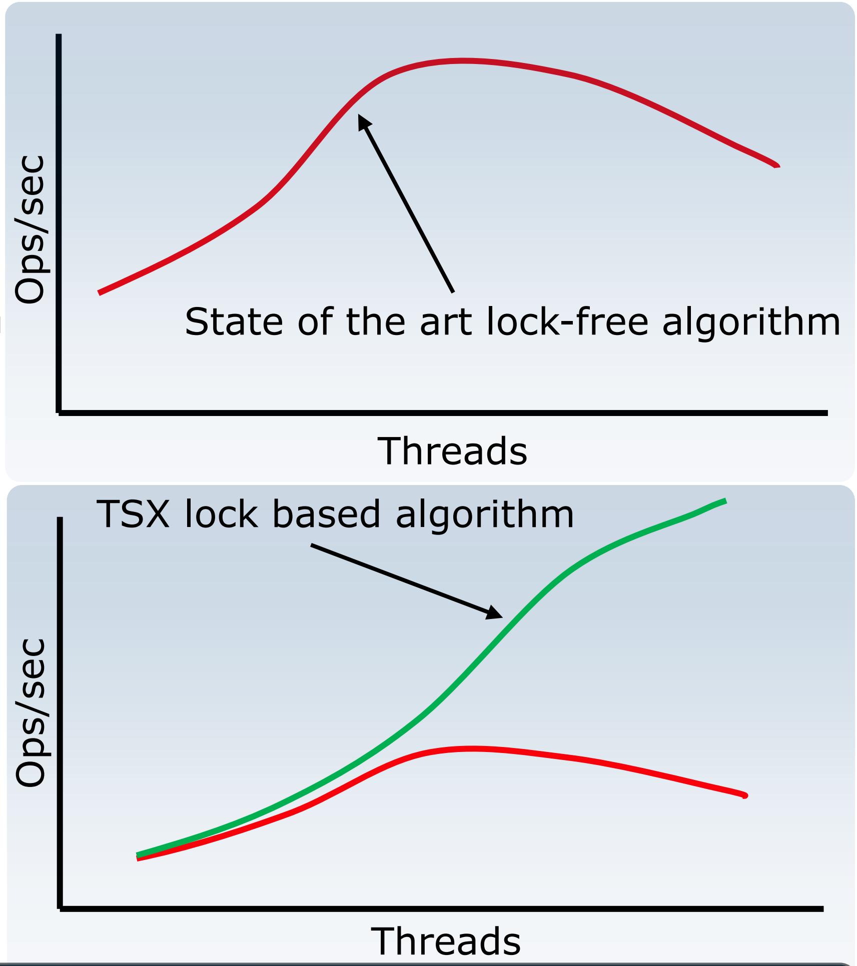
Lock-Free Algorithm

- Don’t use critical section locks
- Developer manages concurrency
- Very difficult to get correct & optimize
  - **Constrain data structure selection**
  - Highly contended atomic operations

Lock-Based + Intel® TSX

- Use critical section locks for ease
- Let hardware extract concurrency
- Enables algorithm simplification
  - **Flexible data structure selection**
  - Equivalent data structure lock-free algorithm very hard to verify

Real World Example



Intel® TSX Can Enable Simpler Scalable Algorithms

Intel® TSX Summary

- **Improves existing synchronization**
  - Lock-based critical sections
  - Goes beyond latency reduction and focuses on serialization
- **Exposes hidden concurrency**
  - Coarse grain effort by developer
  - Finer grain behavior by hardware
- **Architected for**
  - Typical concurrency use-case with lock elision
  - Simple enabling and ease of use

Think About How It Helps You and Other Novel Usages

Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

- A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.
- Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.
- The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.
- Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps.
- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number.
- Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.
- Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm
- Haswell and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user
- Intel, Sponsors of Tomorrow, Ultrabook and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

- *Other names and brands may be claimed as the property of others.
-

Optimization Disclaimer

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the second quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions, including supply constraints and other disruptions affecting customers; customer acceptance of Intel’s and competitors’ products; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. Intel is in the process of transitioning to its next generation of products on 22nm process technology, and there could be execution and timing issues associated with these changes, including products defects and errata and lower than anticipated manufacturing yields. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. The majority of Intel’s non-marketable equity investment portfolio balance is concentrated in companies in the flash memory market segment, and declines in this market segment or changes in management’s plans with respect to Intel’s investments in this market segment could result in significant impairment charges, impacting restructuring charges as well as gains/losses on equity investments and interest and other. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent Form 10-Q, Form 10-K and earnings release.

Rev. 5/4/12