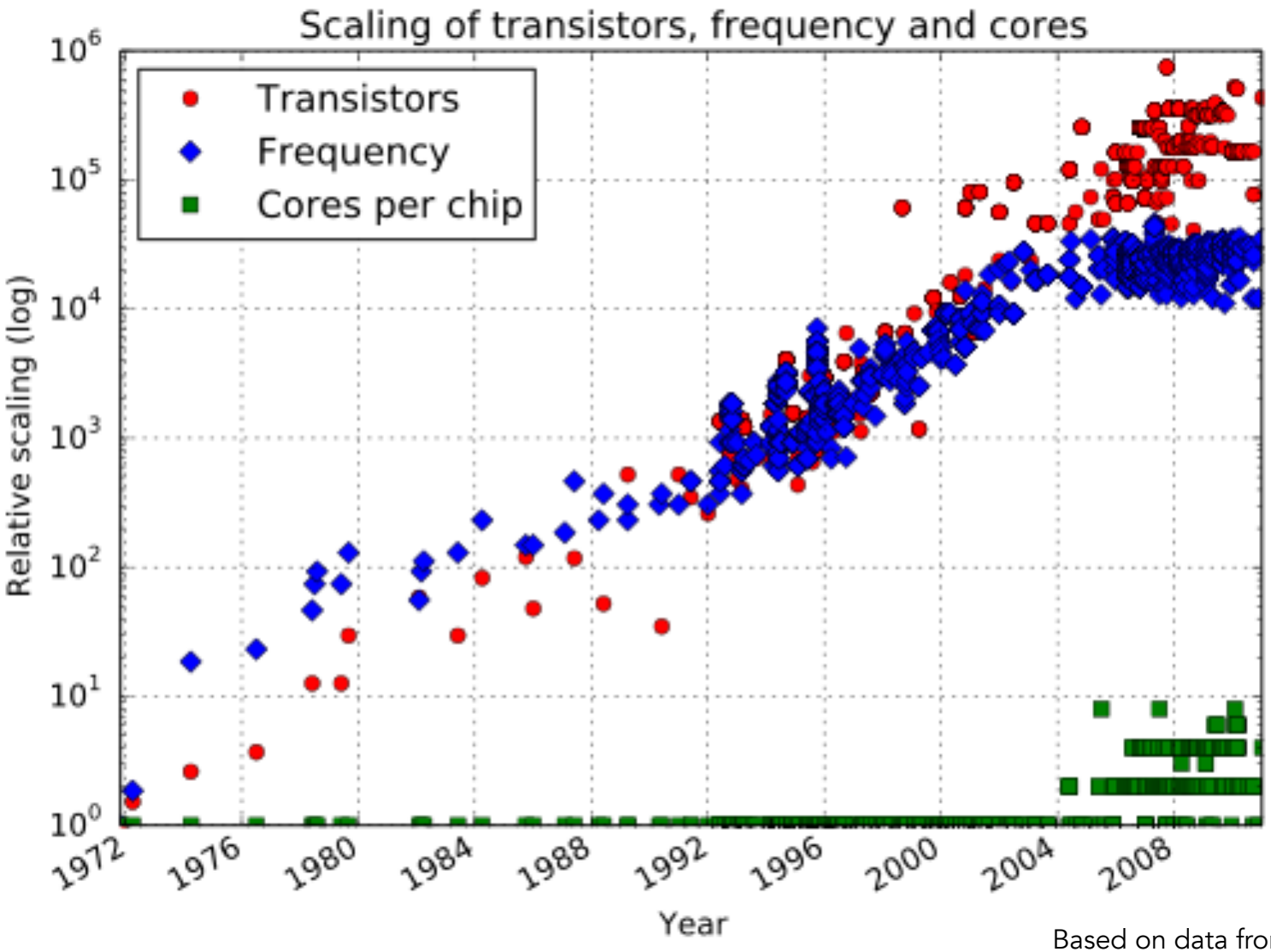
FCDS

Exploiting support for transactional memory in modern multi-core CPUs

Prof. Christof Fetzer

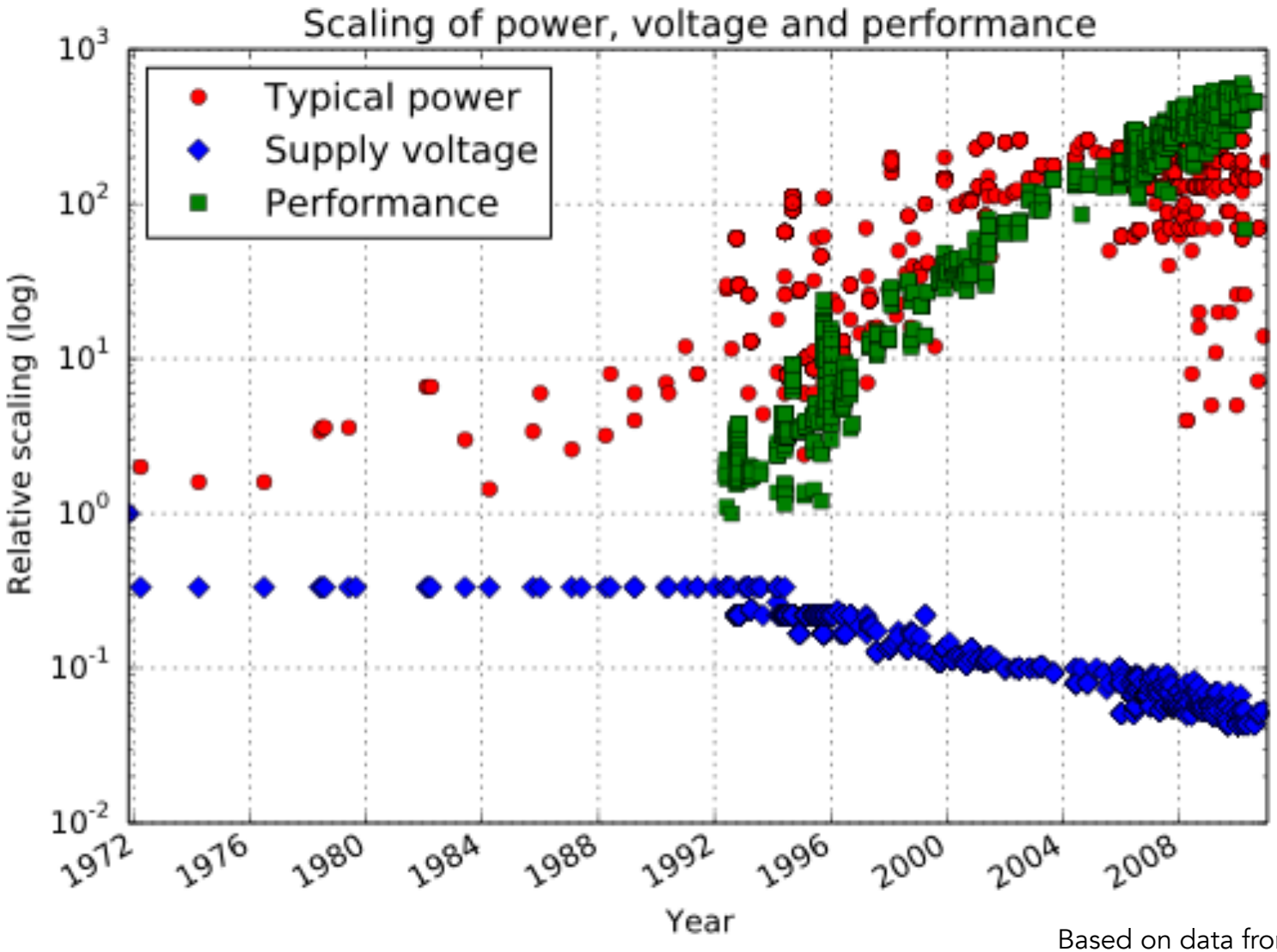
(slides by Pascal Felber)

CPU scaling



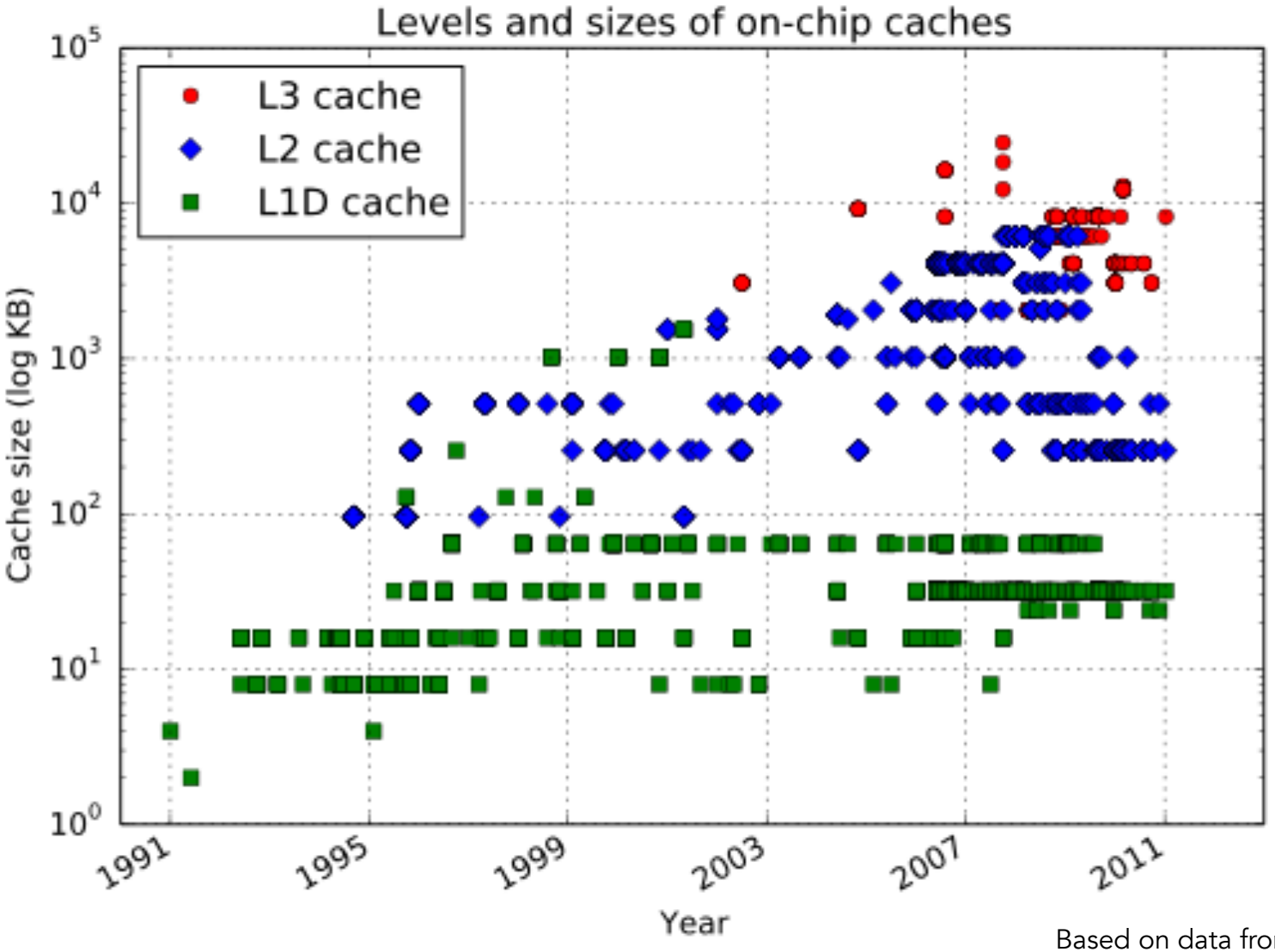
Based on data from the CPU DB [cpudb.stanford.edu]

Power scaling



Based on data from the CPU DB [cpudb.stanford.edu]

Cache scaling



Based on data from the CPU DB [cpudb.stanford.edu]

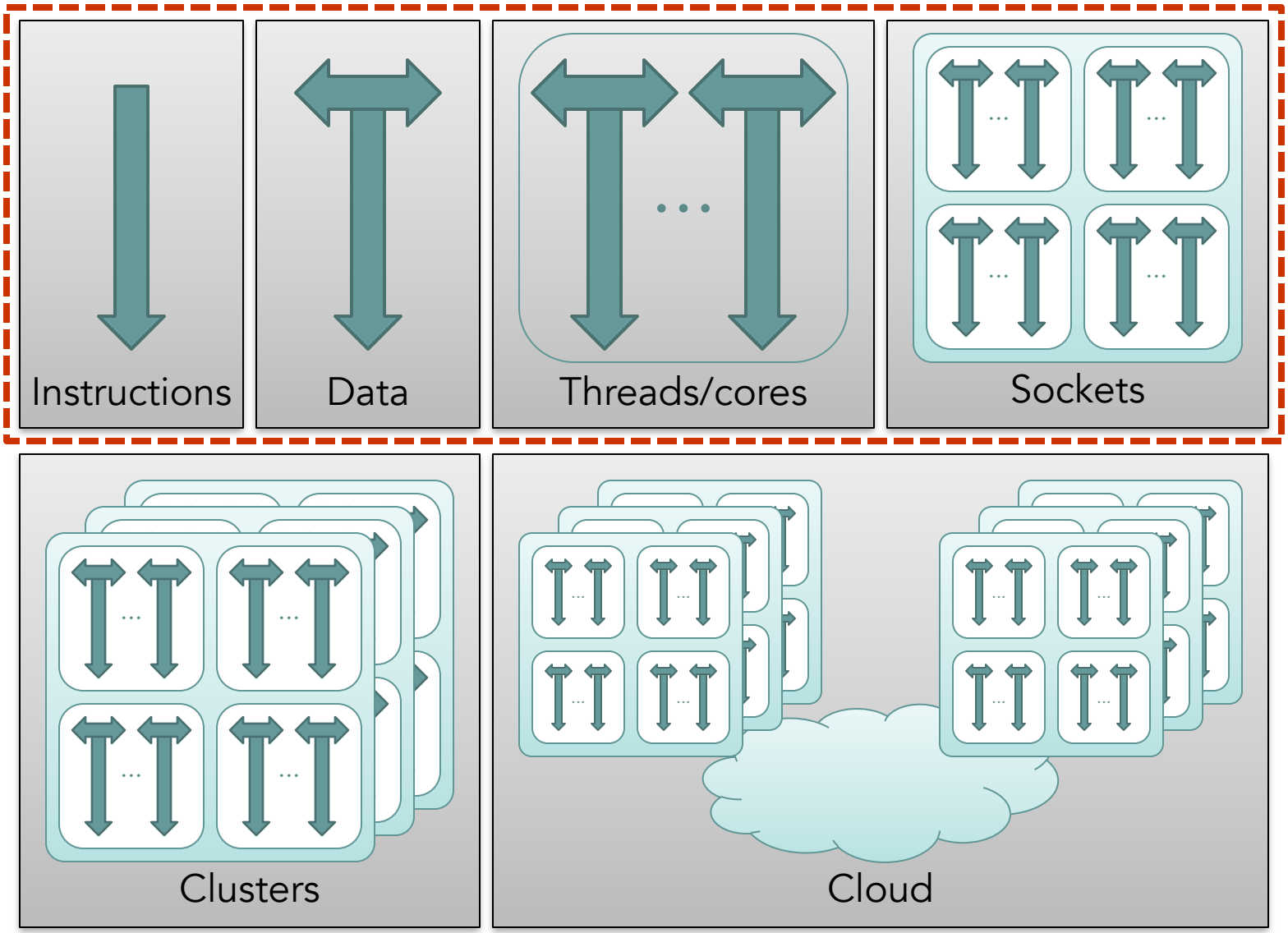
Multi-/many-cores are here

- Multi-cores are the answer to increasing CPU performance despite the "3 walls"
  - Power wall: higher clock speeds require more power and create thermal problems
  - Memory wall: gap between CPU and memory speeds
  - ILP wall: not enough instruction-level parallelism to keep the CPU busy
- Single-thread performance does not improve

  … but we can put more cores on a chip
- To take advantage of multi-cores we have to develop concurrent code

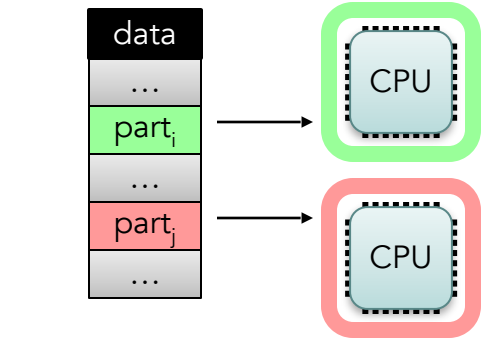
Based on slide by R. Rajwar (Intel)

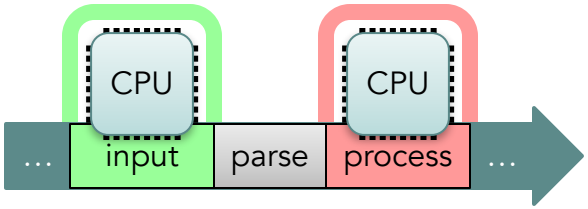
Parallelism: a HW perspective

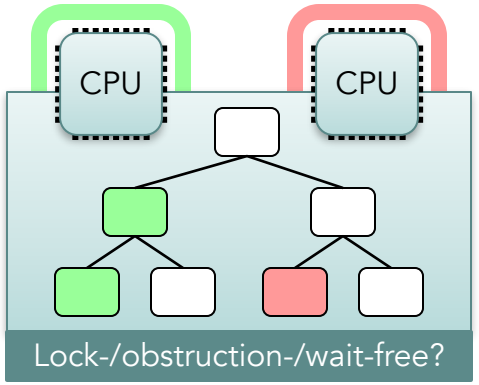


Workloads and parallelism

- Data can be partitioned
  → data parallelism
  - Split data, distribute to CPUs, process in parallel
- Work split in phases
  → pipeline parallelism
  - Each CPU responsible for some phase(s)
- Access to shared data
  → task parallelism
  - Critical sections for synchronization
  - Few conflicts ⇒ speculative/ optimistic parallelism







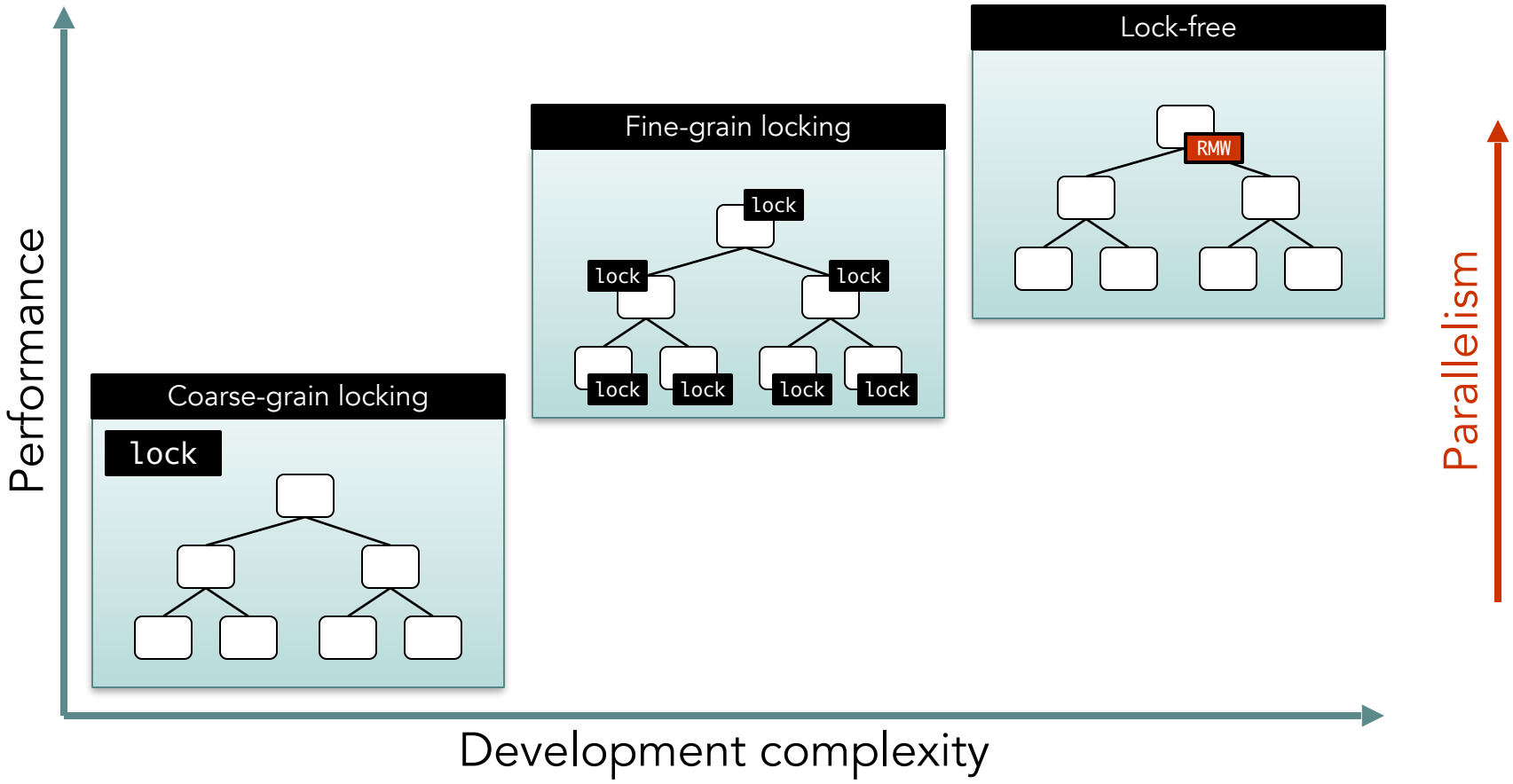
Concurrent programming is hard

- Hard to make **correct** and **efficient**
  - We need to exploit parallelism
  - Need to identify and manage concurrency
- The human mind tends to be sequential
  - Concurrent specifications
  - Non-deterministic executions
- What about races? deadlocks? livelocks? starvation? fairness?
  - Need synchronization (correctness)…

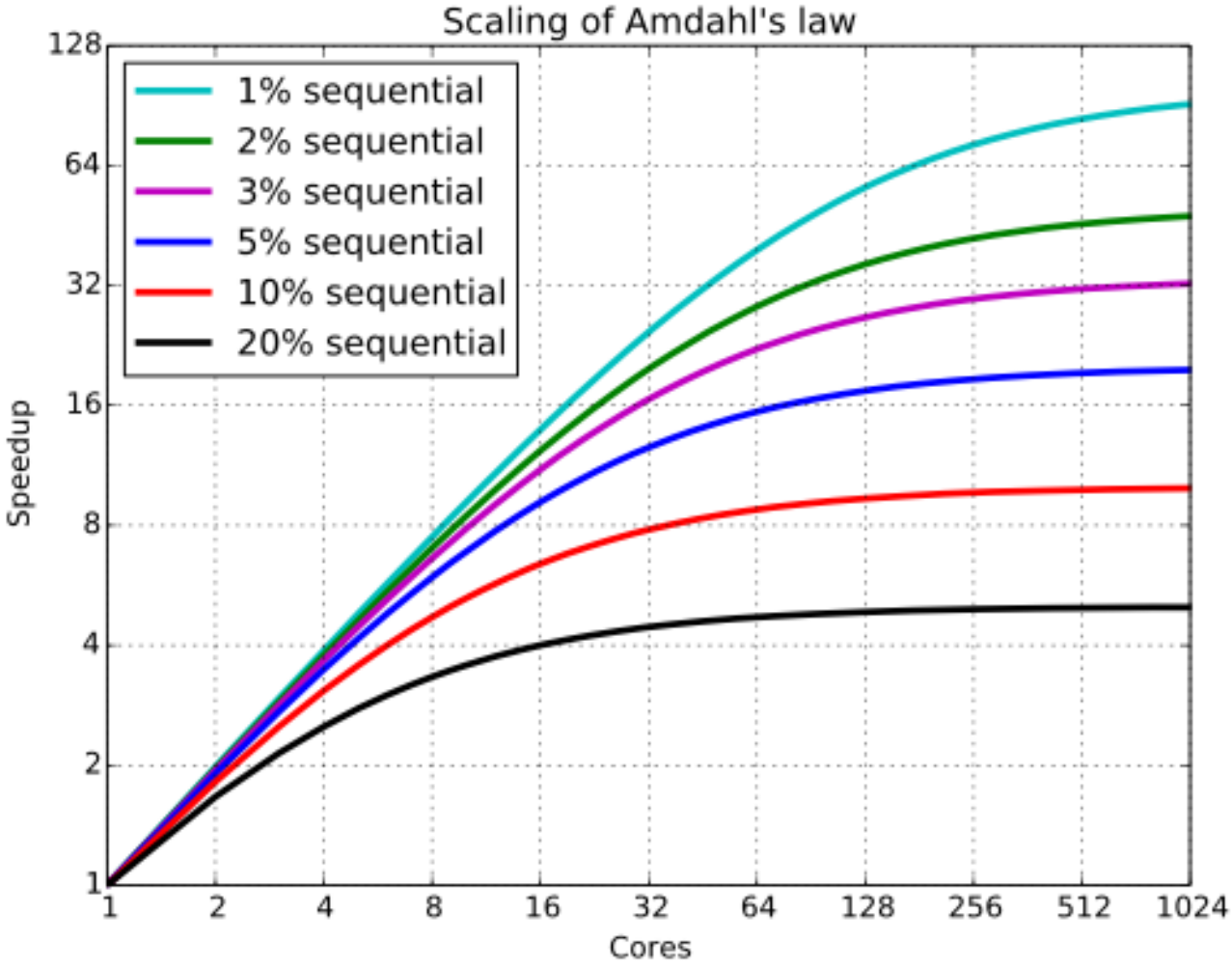
  … but not too much (performance)

Concurrency tradeoffs

Concurrent development approaches differ in complexity and performance



Why is that important?



$$Speedup = \frac{1}{1 - p + \frac{p}{n}}$$

Amdahl's law

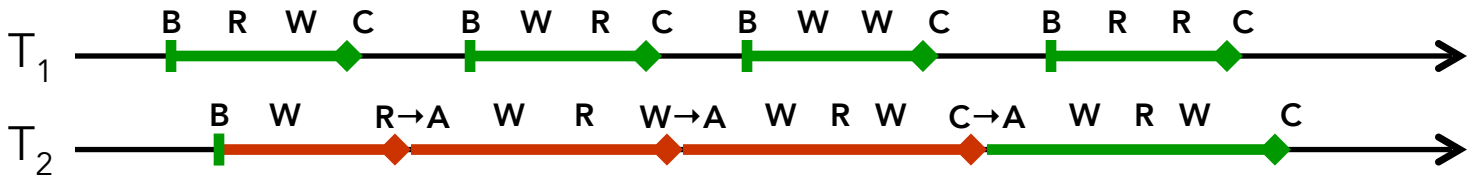
Relationship between the parallel part of a program and the speedup is not linear!

Speculative parallelism with TM

- Exploit parallelism across threads
  - Goal: serialize execution only when necessary
  - Approach: execute speculatively in parallel (optimistic non-blocking execution)
- Transactional memory (TM)
  - Critical sections execute in transactions (=units of work that modify data and either commit, or abort/restart)
  - Transactions run isolated from one another
  - Writes are buffered, become visible upon commit
  - The programmer only has to identify parallel regions and enclose them within transactions

TM in a nutshell

- TM can simplify concurrent programming
  - Sequence of instructions executed atomically
    - **BEGIN** … **READ** / **WRITE** … **COMMIT**
  - Optimistic CC: upon conflict, rollback & restart
  - Alleviates problems of locks: both safe and scalable



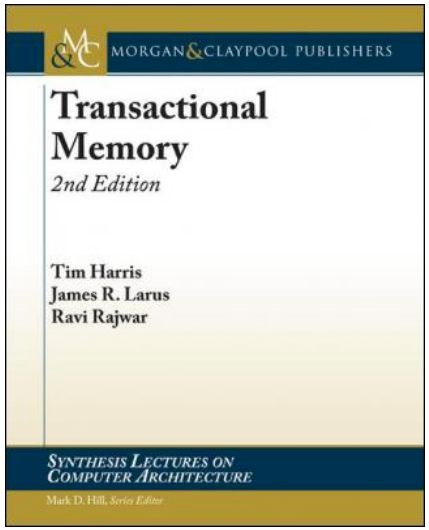
- Simple API
  - Language support (e.g., new keyword and attributes in C++)

```
__transaction_atomic
{
  x = map.remove(key);
  set.add(x);
}
```

Research on TM

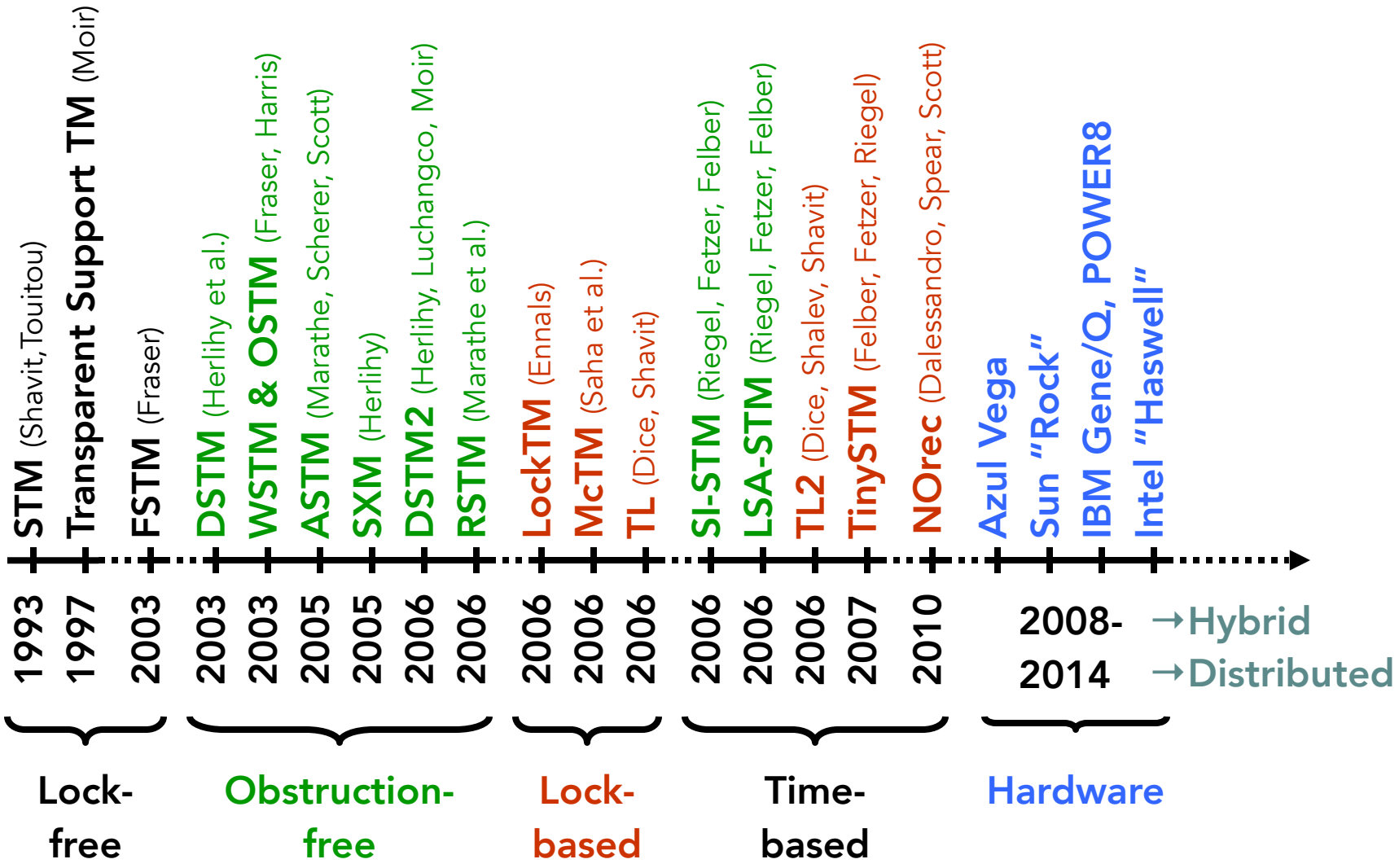
- Active field of research since 2003
  - Algorithms for TM
  - Hardware and hybrid designs
  - Language constructs
  - OS support, (RT) scheduling
  - Runtime and libraries, GC
  - TM for dependability
  - Applications (e.g., game/application servers, DB)
  - Contention management (liveness), theory
  - Distributed TM, cloud/cluster deployments

…



http://research.cs.wisc.edu/trans-memory/biblio/index.html

A brief (partial) history of TM



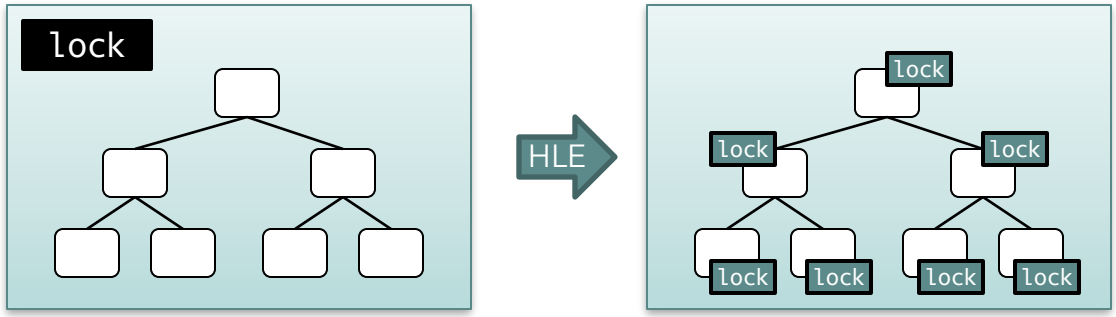
Intel's HTM (x86)

- Intel transactional synchronization extensions TSX has 2 interfaces

1. Hardware Lock Elision (HLE)
   - Lock-granularity optimizations: execute lock-protected critical sections transactionally without acquiring lock
   - Exploit hidden concurrency (automatic parallelization)
   - Binary is backward compatible
2. Restricted Transactional Memory (RTM)
   - More powerful/flexible for speculative execution
   - Can also be used for HLE
   - Requires checking CPU type in binary

HTM and HLE

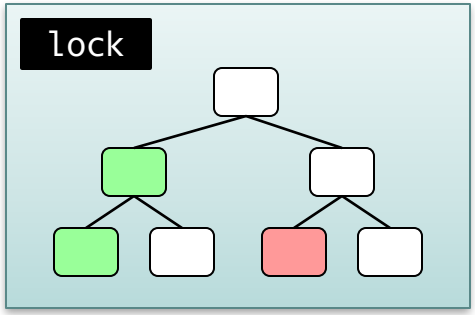
- HTM can support hardware lock elision (HLE)
- Developer uses "coarse-grain" locking
  - Easy to reason about and prove correct
- Hardware elides locks and provides "fine-grain" locking performance
  - Detects actual data conflicts, may abort and restart



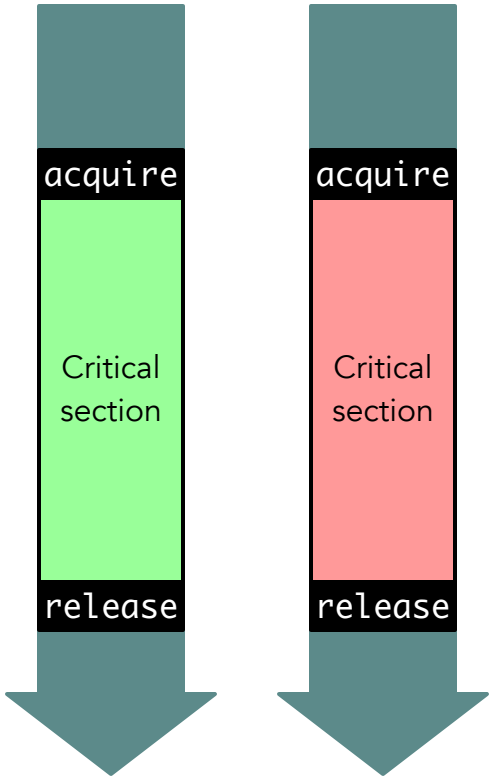
"Fine grain performance at coarse grain effort"

Speculative execution with HLE

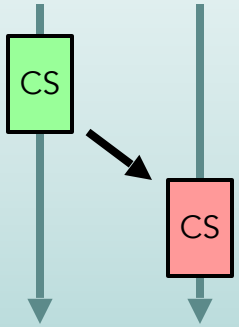
Data structure



- Lock remains free throughout execution
- Upon conflict, restart with lock

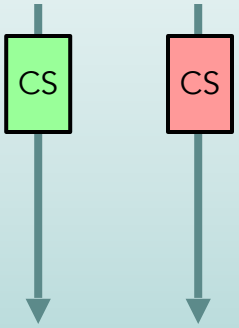


With locks



- Serial execution
- Lock transfer latencies

Without locks and conflicts



- Concurrent execution
- No lock transfer latencies

HLE

- Hint inserted in front of `LOCK` operation to identify a region candidate for lock elision
  - Prefixes `XACQUIRE` and `XRELEASE`
  - Hints ignored by non-TSX processors
- Region executed speculatively in transaction
  - Does not acquire lock (but watch for modifications)
  - Tracks load, buffers stores, checkpoints registers
  - Attempts to commit
  - If HW cannot commit (e.g., conflict), restart and execute non-speculatively by acquiring lock

Based on slide by R. Rajwar (Intel)

HLE example

Code example from Intel

Acquire lock (library)

```
      mov eax, 1
try:  lock xchg mutex, eax
      cmp eax, 0
      jz success
spin: pause
      cmp mutex, 1
      jz spin
      jmp try
```

HLE

Acquire lock with HLE (library)

```
      mov eax, 1
try:  xacquire lock xchg mutex, eax
      cmp eax, 0
      jz success
spin: pause
      cmp mutex, 1
      jz spin
      jmp try
```

Application

```
acquire_lock(mutex);
/* critical section */
release_lock(mutex);
```

```
mov mutex, 0
```

Release lock (library)

HLE

```
xrelease mov mutex, 0
```

Release lock with HLE (library)

RTM

- RTM introduces new instructions
  - Demarcate critical sections with XBEGIN/XEND
  - Speculative execution as HLE but no lock involved
  - If transaction cannot commit atomically, abort and execute handler specified by XBEGIN
  - Abort information returned in EAX register
  - Software can also explicitly abort with XABORT
  - XTEST instruction can be used by SW to determine if in active HLE or RTM region

Based on slide by R. Rajwar (Intel)

RTM example

Code example from Intel

Acquire lock with RTM (library)

```
retry:  xbegin abort
        cmp mutex, 0
        jz success
        xabort $0xff
abort:
      ; check EAX, maybe first
      ; retry speculatively,
      ; otherwise acquire lock  …
```

```
…       mov eax, 1
try:    lock xchg mutex, eax
        cmp eax, 0
        jz success
spin:   pause
        cmp mutex, 1
        jz spin
        jmp try
```

Application

```
acquire_lock(mutex);
/* critical section */
release_lock(mutex);
```

```
        cmp mutex, 0
        jnz unlock
        xend
                          …
```

```
…
unlock: mov mutex, 0
```

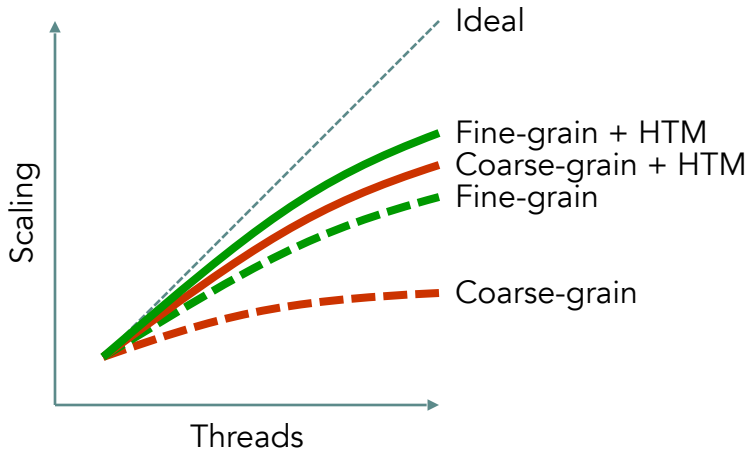
Release lock with RTM (library)

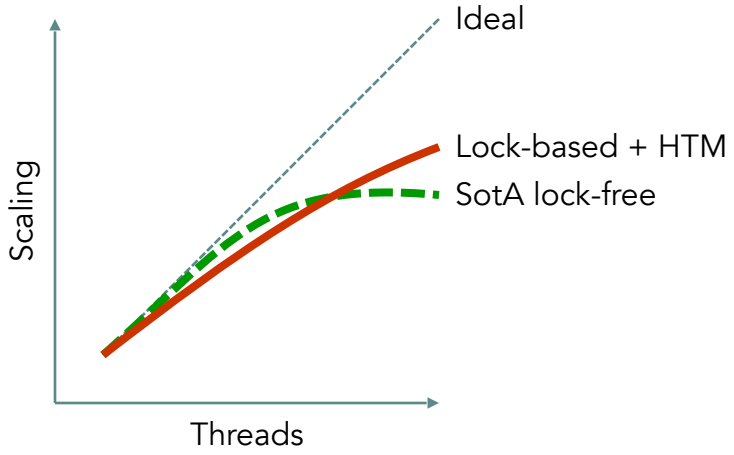
TSX implementation notes

- Write buffering using L1 cache (32KB)
  - Writes visible to other threads only after commit
  - Cache line granularity: eviction ⇒ abort, ⚠false sharing
  - Cache size/associativity can be a limit
- Conflict detection for reads and writes
  - L1 tracks addresses read/written in TSX region
  - Conflicts detected using cache coherence protocol
- Transactional commit
  - Transactional updates visible instantaneously
- Transactional abort discards all updates

HTM benefits

- Improve performance of lock-based code
  - Both coarse-grain and fine-grain applications
- Simplify programming
  - Lock-free algorithms hard to make/prove correct!
  - HTM provides lock-free behavior for lock-based code (if possible)
    - Use locks for critical sections
    - Let HW extract concurrency





Graphs not based on real data

More benefits

- Lock-free programming:
  - sometimes we would need a DCAS, etc
  - RTM helps to implement such DCAS
-

Progress?

- **Obstruction-free programming:**
  - difficult to achieve wait-free guarantees
  - guarantee to make progress while no other threads conflict
    - thread backs-off if it sees contention (i.e., transaction aborts)

IBM's HTM (POWER8)

- Similar API: `tbegin`, `tend`, `tabort`, `tcheck`
  - Transactions can be suspended: `tsuspend`, `tresume`
    - Inter-thread communication, stores that should not roll back, …
  - Conflicts management at cache block granularity
  - Transactions can be nested (flat nesting: inner transactions aborts all enclosing ones)
  - No guarantee of fairness/progress provided by HW

POWER8 HTM

```
start:  tbegin
        bne abort ; Aborted?

      ; execute transactionally

        tend
abort:
        b start ; Retry
```

HTM in practice

- Intel **TBB** provides
  - HLE-based `speculative_spin_mutex`
  - RTM-based `speculative_spin_rw_mutex`
- Intel **OpenMP** features RTM-based lock elision
- Experimental HLE-based lock elision support in glibc **pthreads**
- IBM's JVM exploits POWER8's HTM in JIT

…

When to use TM?

- TM is likely to help…

  …if the application has many threads/resources with lightweight locking requirements (short critical sections)

  …if read-only transactions rarely access the same data as concurrent update transactions (few conflicts)

  …if the application uses coarse-grain locks and does not scale well (much serialization)

- TM is unlikely to help…

  …if the application uses fine-grain locking and scales well (good parallelism)

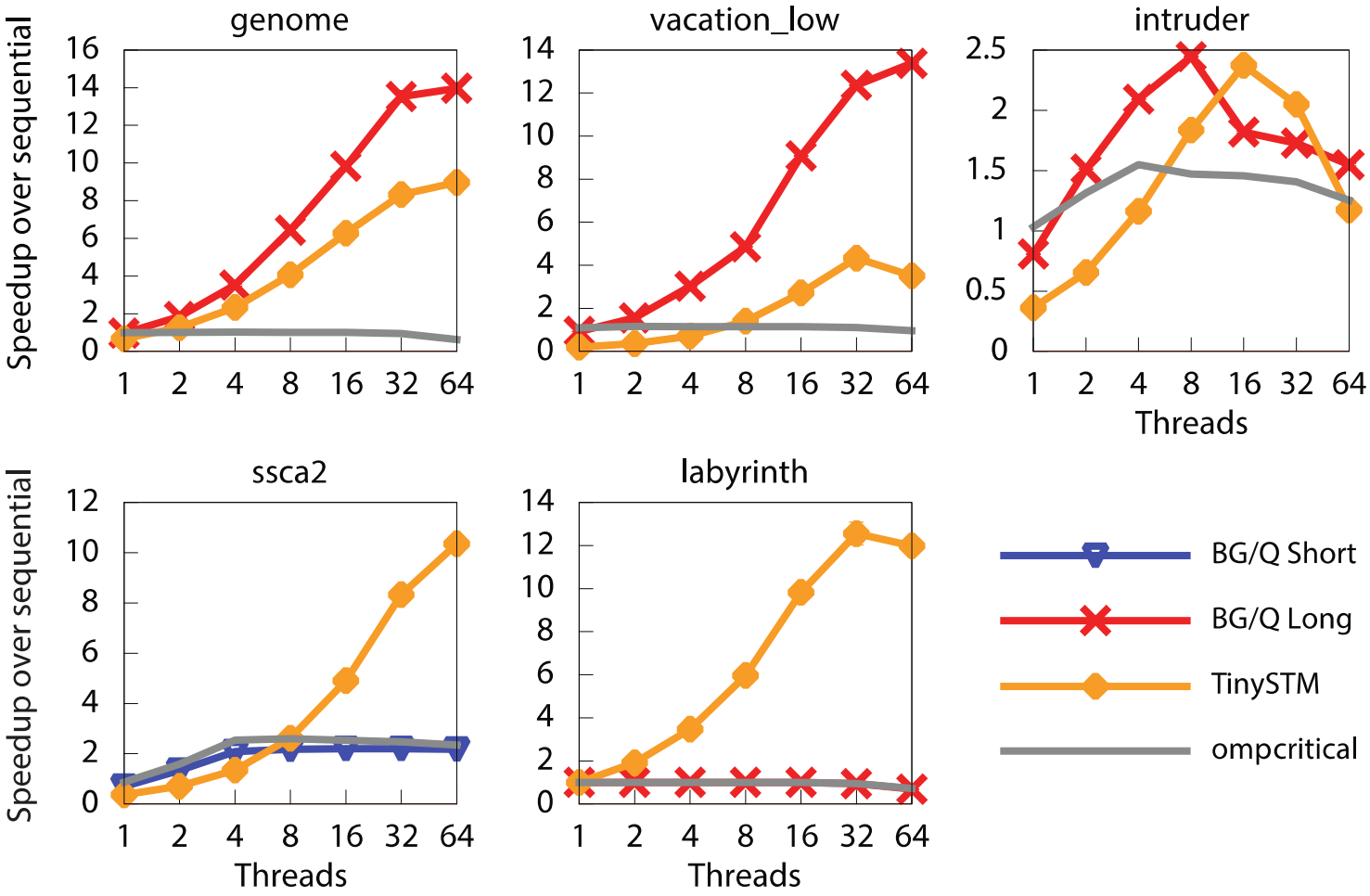
  …if transactions access much data (long transactions)

  …if transactions conflict often (high contention)

HTM performance (IBM)

Evaluation of Blue Gene/Q Hardware Support for Transactional Memories. Wang et al. [PACT 2012]

STAMP benchmarks



HTM performance (Intel)

Exploiting Hardware Transactional Memory in Main-Memory Databases. Leis, Kemper, Neumann. [ICDE 2014]

Intel's TSX for in-memory databases

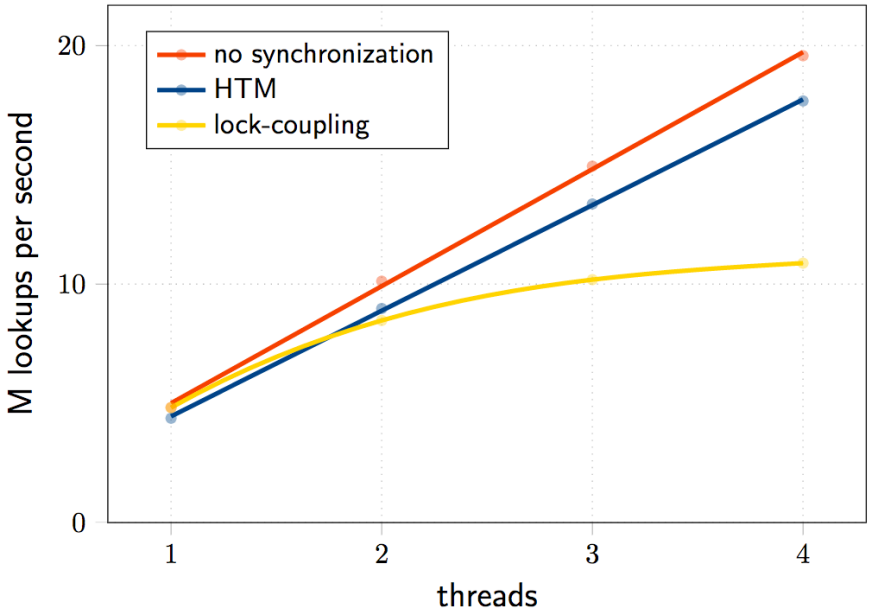


Figure 11. Synchronized index lookup, 16M sparse 4 byte keys

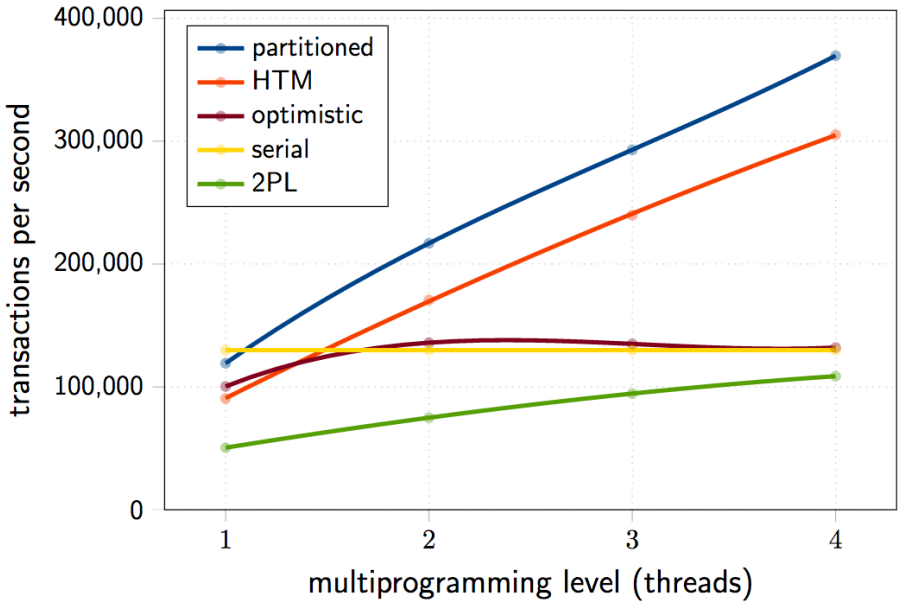


Figure 12. Scalability with TPC-C on desktop system

Summary

- Concurrent programming necessary to take advantage of multi-/many-cores
  - Complex to write correct and efficient code
- HTM provides powerful tools to solve locking and synchronization problems
  - HLE is trivial to use, binary-compatible
  - RTM and other HTM APIs are flexible and powerful
- Tools/libraries with HTM optimizations
- Useful in combination with other paradigms
  - Cloud/clusters, MapReduce, ESP, MP, etc.

TSX bug?

- **HSW136. Software Using Intel® TSX May Result in Unpredictable System**
  - **Problem:** Under a complex set of internal timing conditions and system events, software using the Intel TSX (Transactional Synchronization Extensions) instructions may result in unpredictable system behavior.
  - **Implication:** This erratum may result in unpredictable system behavior.
  - **Workaround:** It is possible for the BIOS to contain a workaround for this erratum.

- **HSW1. TSX instruction**
  - Due to Erratum HSW136, TSX instructions are disabled and are only supported for software development. See your Intel representative for details.