Pushing the limits of work-stealing

Anton Malakhov
Evgeny Fiksman
Intel, Russia and Israel

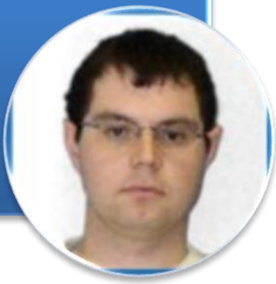
Authors

- Senior developer for Intel® Threading Building Blocks (Intel® TBB), since 2006, before the first Beta release
- Patent applications for algorithms of concurrent_hash_map and auto partitioner
- Currently responsible for Intel TBB task scheduler functionality and improvements for OpenCL* runtime team.
- linkedin.com/in/antonmalakhov

Anton Malakhov

- Lead developer of OpenCL* runtime for CPU and Intel® Many Integrated Core Architecture (Intel® MIC Architecture) since first line was added
- Currently main focal point between Intel's OpenCL and Intel TBB teams, closely working with Intel TBB team on definition, prototyping and deployment of new features

Evgeny Fiksman



Background

Intel® Xeon Phi™ coprocessor

- Intel® Many Integrated Core Architecture
- ~60 cores / ~240 threads
- Wide SIMD

OpenCL

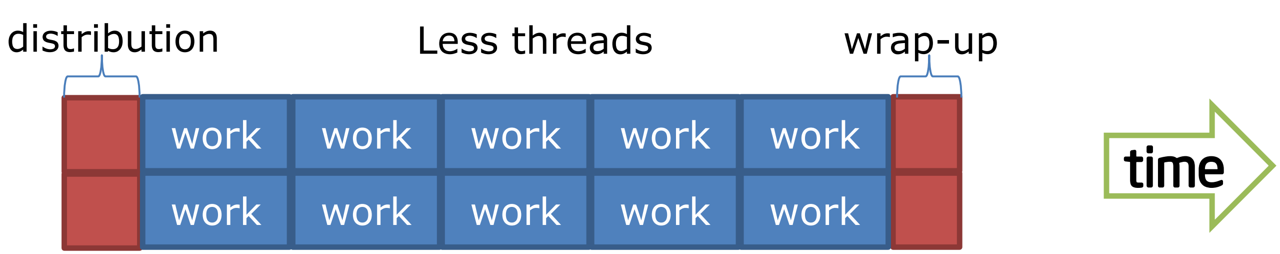
- Heterogeneous parallel programing on CPU, GPGPU, accelerators
- Host / Device model, C99 based
- Cross-platform, functional portability
- Challenge: Performance portability

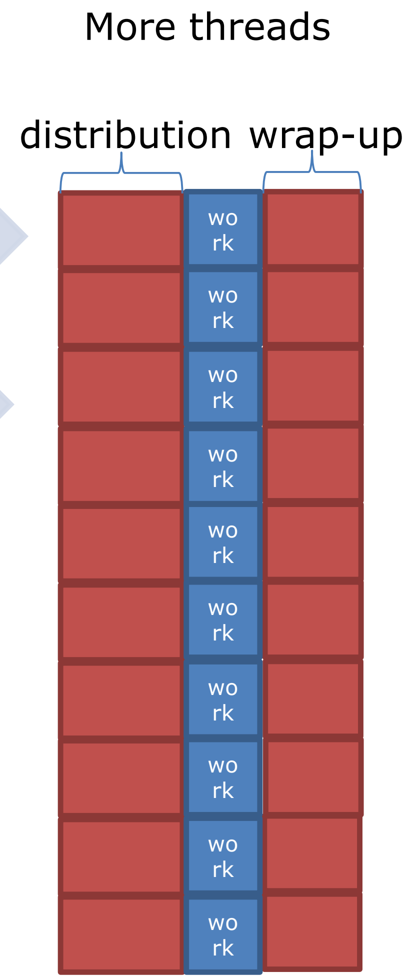
Intel TBB

- Work-stealing scheduler
- Efficient load balancing
- Nested parallelism
- Containers, algorithms, locks…
- www.threadingbuildingblocks.org

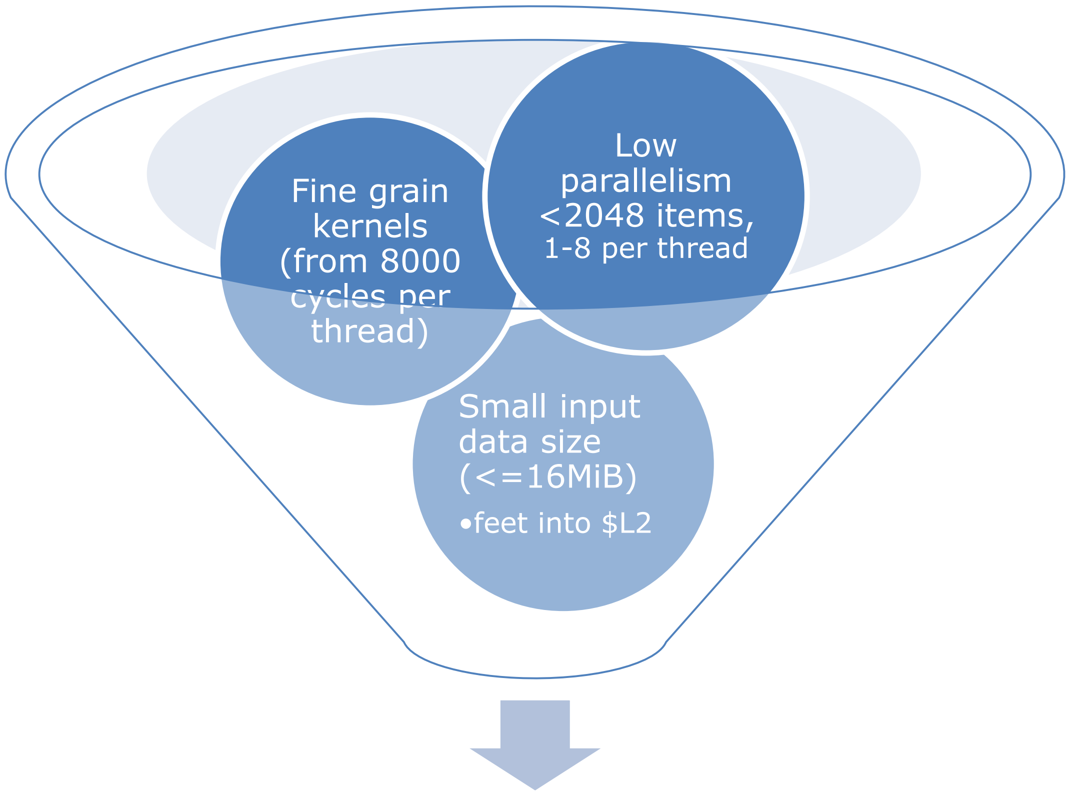
OpenCL on Intel Xeon Phi coprocessor: challenges

- **GPGPU applications** → Short/Lightweight kernels → Memory Bounded
- **Latency** → Host-Device communication → Memory Transfer and Compute overlap
- **Throughput** → Increased synchronization overheads → High memory latency → Need efficient threading
- **Intel MIC architecture** → More threads → Wider SIMD size → Increased overheads





Target Benchmark's Characteristics

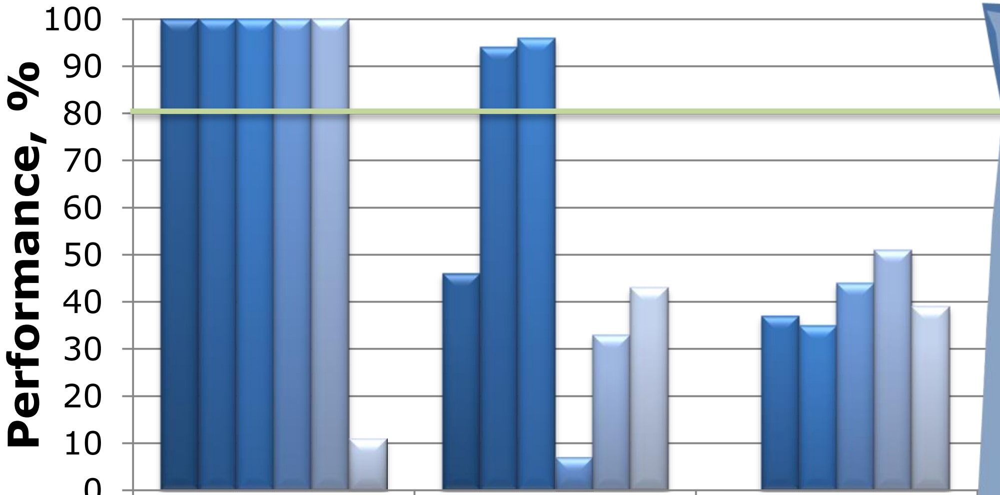


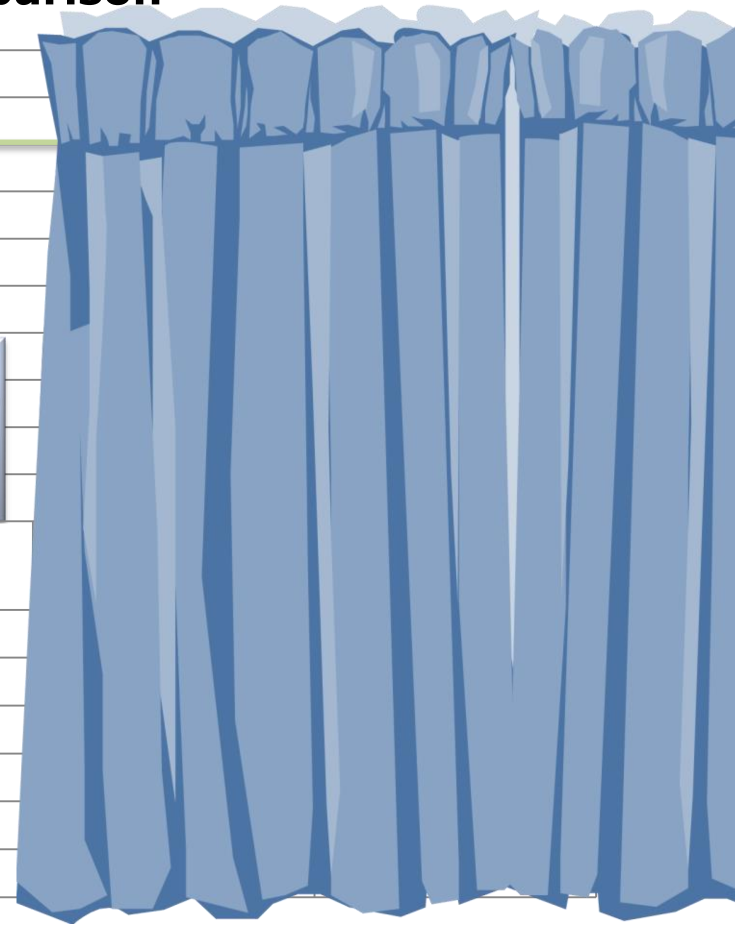
Native C++ reproducers on top of OpenMP* API and Intel TBB

Optimization Notice

Initial results

Normalized performance comparison





Threading: Benchmarks	omp static	omp dynamic	TBB 4.1 defaults
minimal	100	46	0
conformance	100	94	37
reduction	100	96	35
scan	100	7	44
spmv	100	33	51
spmv-unbalanced	11	43	39

Configuration Info - SW Versions: Intel® C++ Intel® 64 Compiler, Version 13.1.1.163; Hardware: Intel® Xeon Phi™ Coprocessor 7120 (16GB, 1.238 GHz, 61C/244T); MPSS Version: 3.1; Flash Version: 2.1.03.0386; Host: 2x Intel® Xeon® CPU E5-2680 0 @ 2.70GHz (16C/32T); 64GB Main Memory; OS: Red Hat Enterprise Linux Server release 6.2 (Santiago), kernel 2.6.32-220.el6.x86_64; Benchmarks are measured only on Intel® Xeon Phi™ Coprocessor with power management disabled.
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

Optimization Notice

General optimizations

Improve inter-core communication

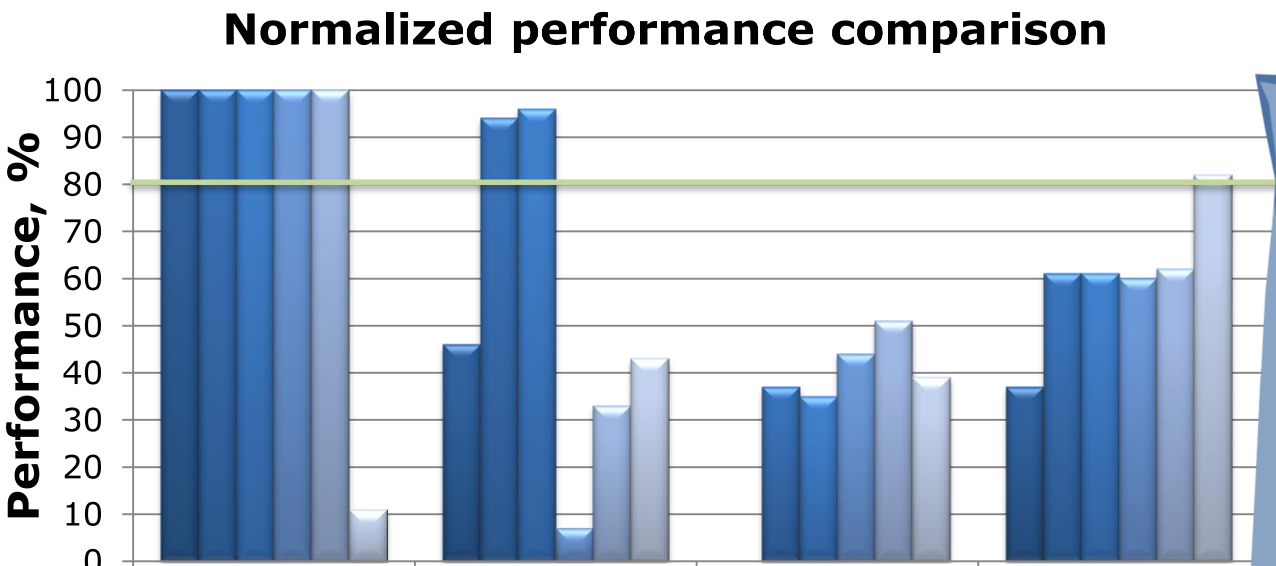
- Tuning pause times for spin-loops
- Optimized memory layouts for less shared cache
- Manual cache prefetching and eviction

Overcome multi-core overheads

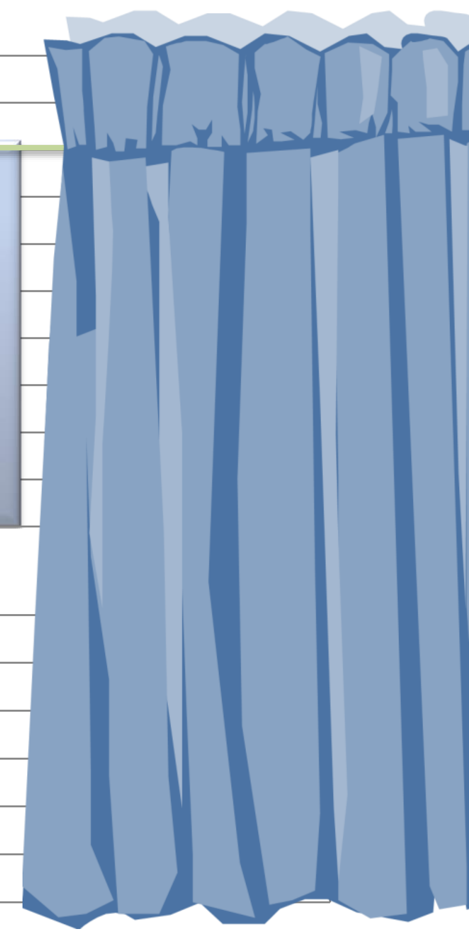
- Reduce thieves contention on a victim
- Reduce synchronization points on critical path
- Pin worker threads to remove outliers

Optimization Notice

Intermediate results



Threading: / Benchmarks	omp static	omp dynamic	TBB 4.1 defaults	TBB 4.2 U2 defaults
minimal	100	46	0	37
conformance	100	94	37	61
reduction	100	96	35	61
scan	100	7	44	60
spmv	100	33	51	62
spmv-unbalanced	11	43	39	82



Configuration Info - SW Versions: Intel® C++ Intel® 64 Compiler, Version 13.1.1.163; Hardware: Intel® Xeon Phi™ Coprocessor 7120 (16GB, 1.238 GHz, 61C/244T); MPSS Version: 3.1; Flash Version: 2.1.03.0386; Host: 2x Intel® Xeon® CPU E5-2680 0 @ 2.70GHz (16C/32T); 64GB Main Memory; OS: Red Hat Enterprise Linux Server release 6.2 (Santiago), kernel 2.6.32-220.el6.x86_64; Benchmarks are measured only on Intel® Xeon Phi™ Coprocessor with power management disabled.
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

Optimization Notice

Load balancing challenge

OpenMP* static schedule shines on well balanced work

- ✔ No load-balancing overheads, work is distributed equally across threads
- ✔ Deterministic distribution: cache locality on repeating workloads

↓

OpenCL needs load balancing without killing performance
Work-stealing works against things which help OpenMP* shine

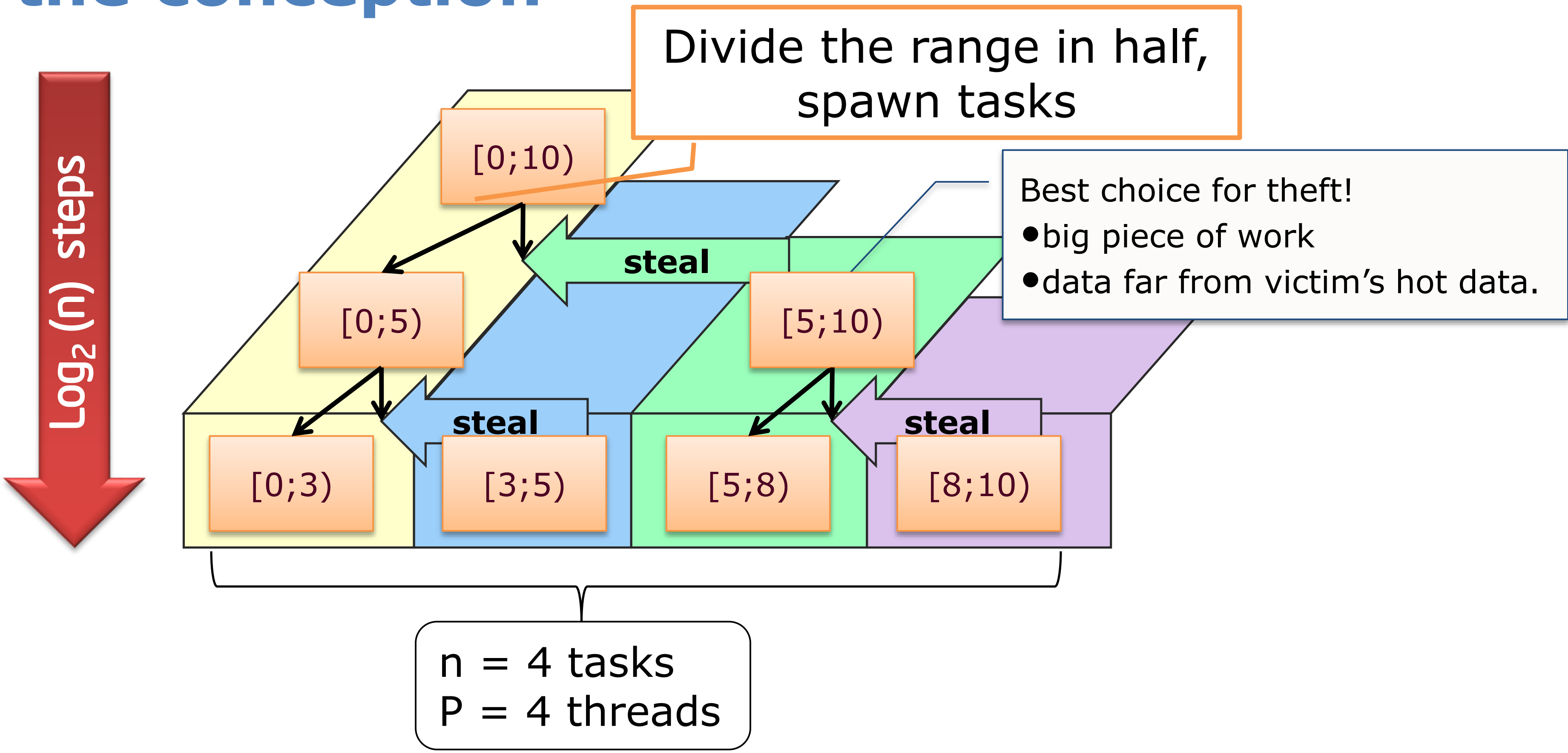
- ⊘ TBB assigns work to threads unevenly, Load-balancing breaks equal distribution
- ⊘ Random work-stealing disrupts cache locality

↓

One-fits-all approach needed

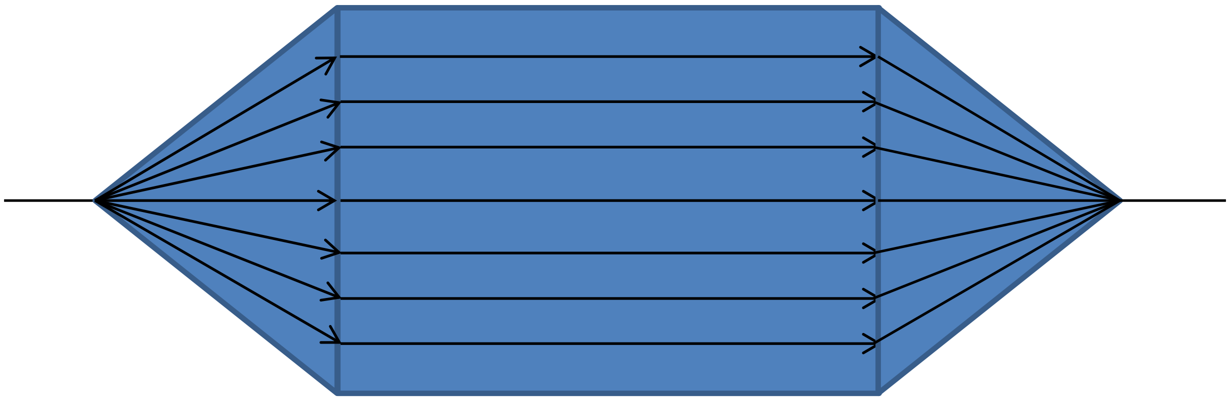
20% behind omp static is tolerable

Work distribution through stealing: the conception



Stealing is random → any thread can steal from any other thread → Non-determinism. Slow for few tasks across lots of threads

Our work distribution issues



Distribution

- Non-determinism →cache locality
- Slow for first and last tasks
- Irregularity →final imbalance

Executing

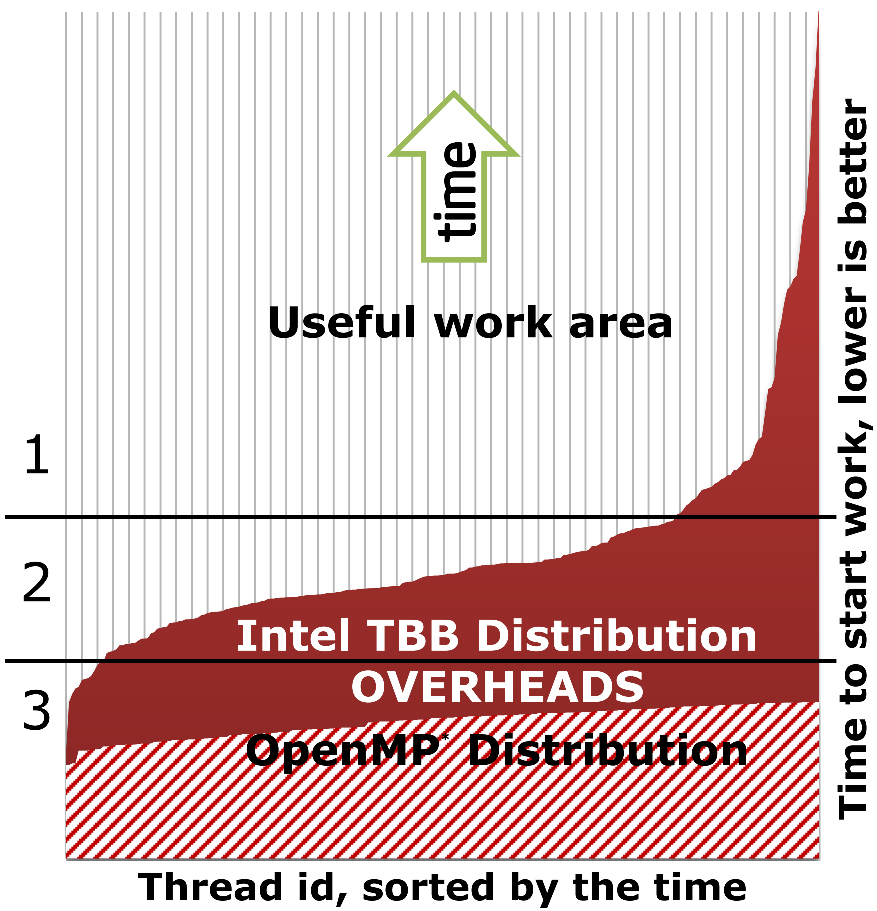
- Cache locality
- Granularity

Final load balancing

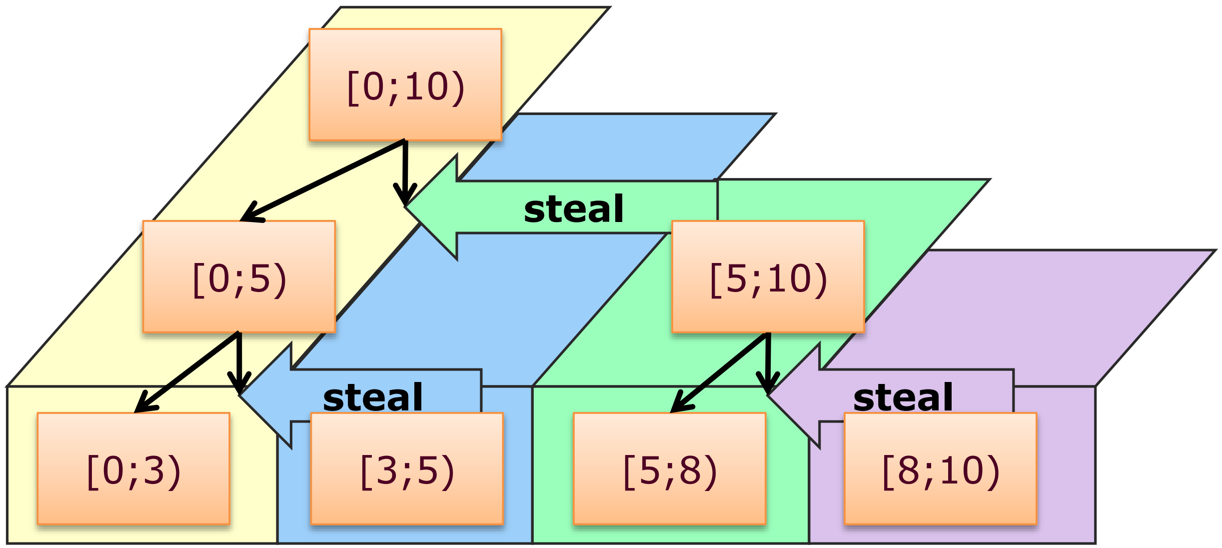
- Irregularity induced imbalance
- Or caused by too small granularity

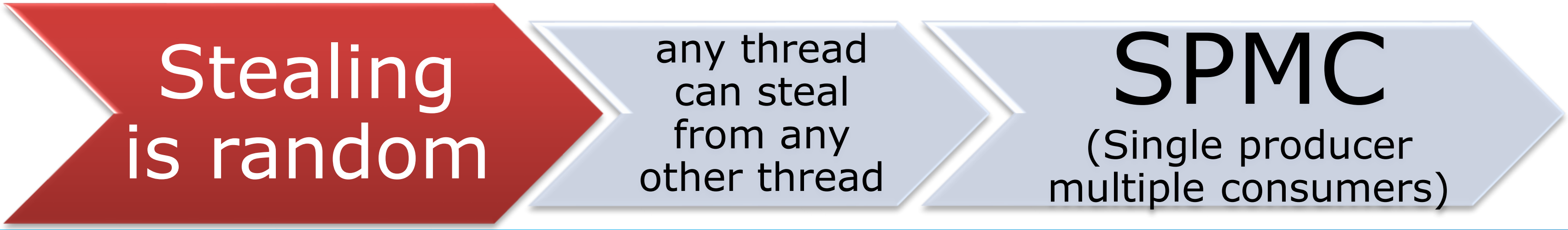
Work distribution through stealing: challenges of big concurrency

1. Hard to find last tasks across all task pools
   - The worst case is for n == P
2. Speed of signal propagation via binary task tree
   - Classical equal splits spread the work slower
3. Thieves contention on victim's lock
   - Try_lock helps for $P > \log_2 n$
   - But still hard to start the first tasks..

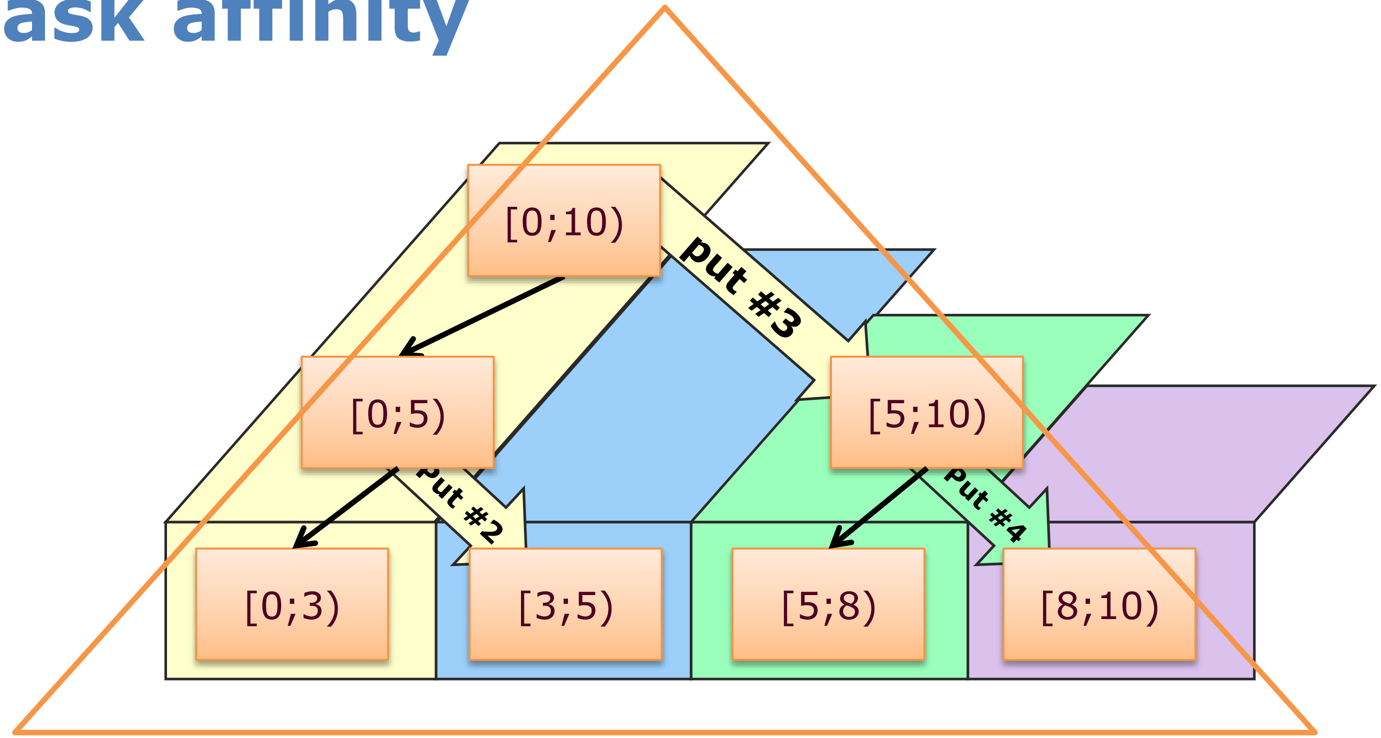


Work distribution through stealing: the conception (reminder)





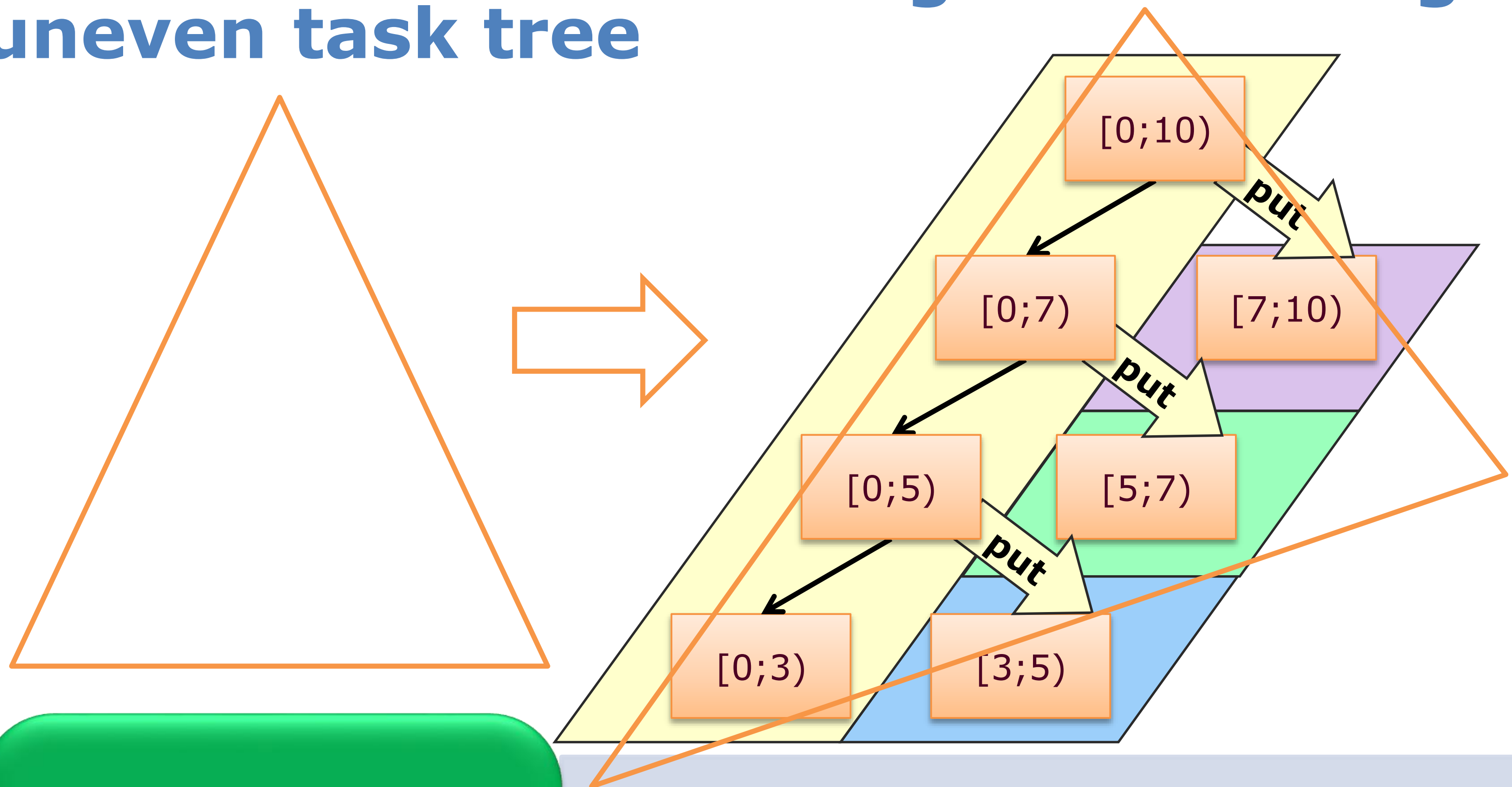
Work distribution through mailboxing: task affinity



Intel TBB task affinity

- Deterministic distribution via mailboxing
- No contention (SPSC communication)

Work distribution through mailboxing: uneven task tree

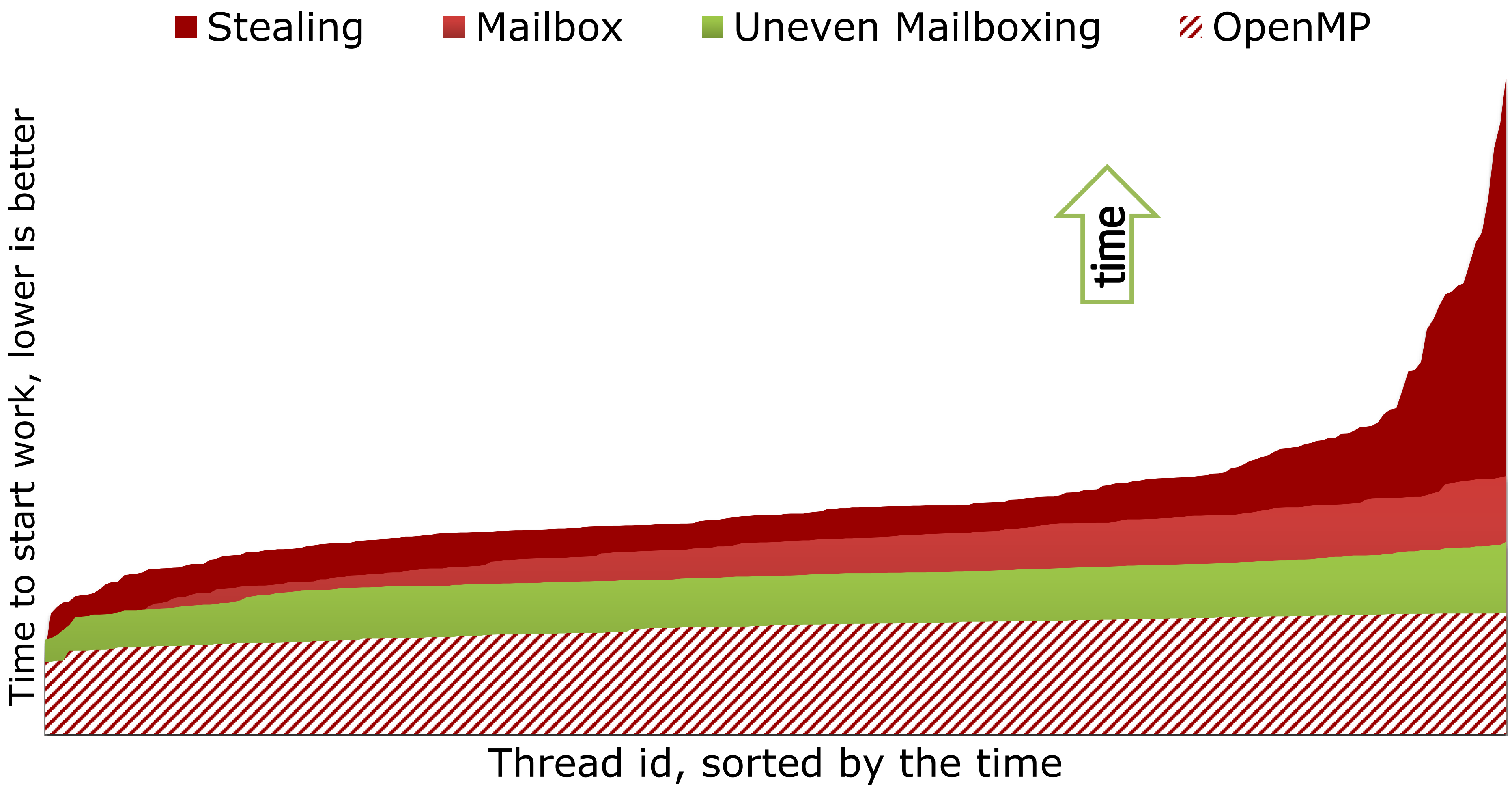


Unbalanced (lopsided) task tree

- Sending signal is faster than receiving it
- Earlier threads produce more tasks than later threads (not serialized!)
- Split in a proportion, not in half

Optimization Notice

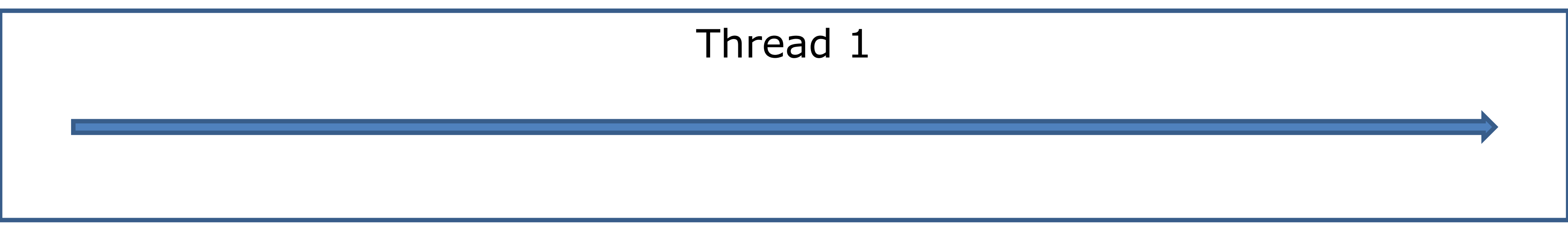
Start of 240 tasks, 1 per thread



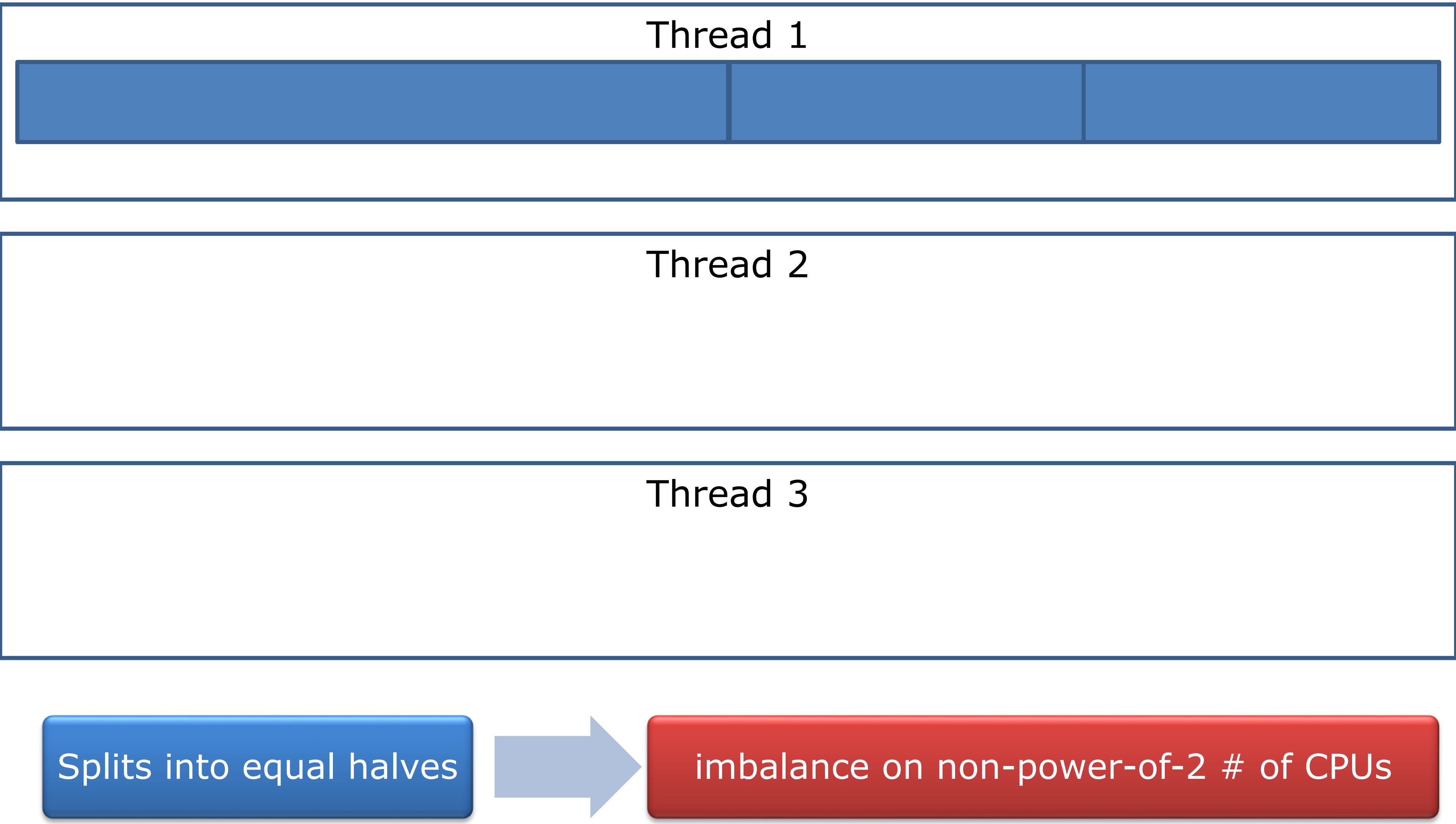
Configuration Info - SW Versions: Intel® C++ Intel® 64 Compiler, Version 13.1.1.163; Hardware: Intel® Xeon Phi™ Coprocessor 7120 (16GB, 1.238 GHz, 61C/244T); MPSS Version: 3.1; Flash Version: 2.1.03.0386; Host: 2x Intel® Xeon® CPU E5-2680 0 @ 2.70GHz (16C/32T); 64GB Main Memory; OS: Red Hat Enterprise Linux Server release 6.2 (Santiago), kernel 2.6.32-220.el6.x86_64; Benchmarks are measured only on Intel® Xeon Phi™ Coprocessor with power management disabled.
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

Distribution: work's point of view

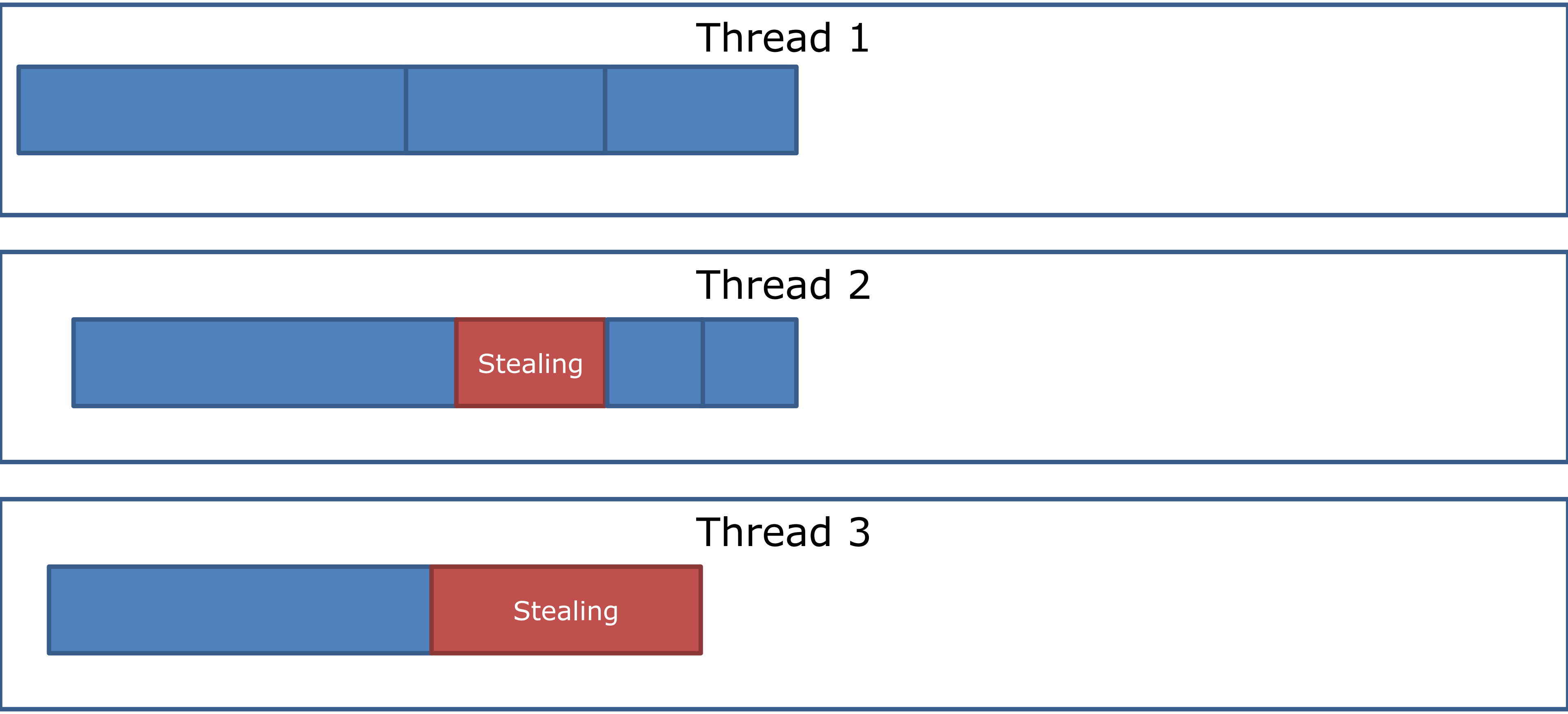
The thread, timeline



Simple work distribution: splitting

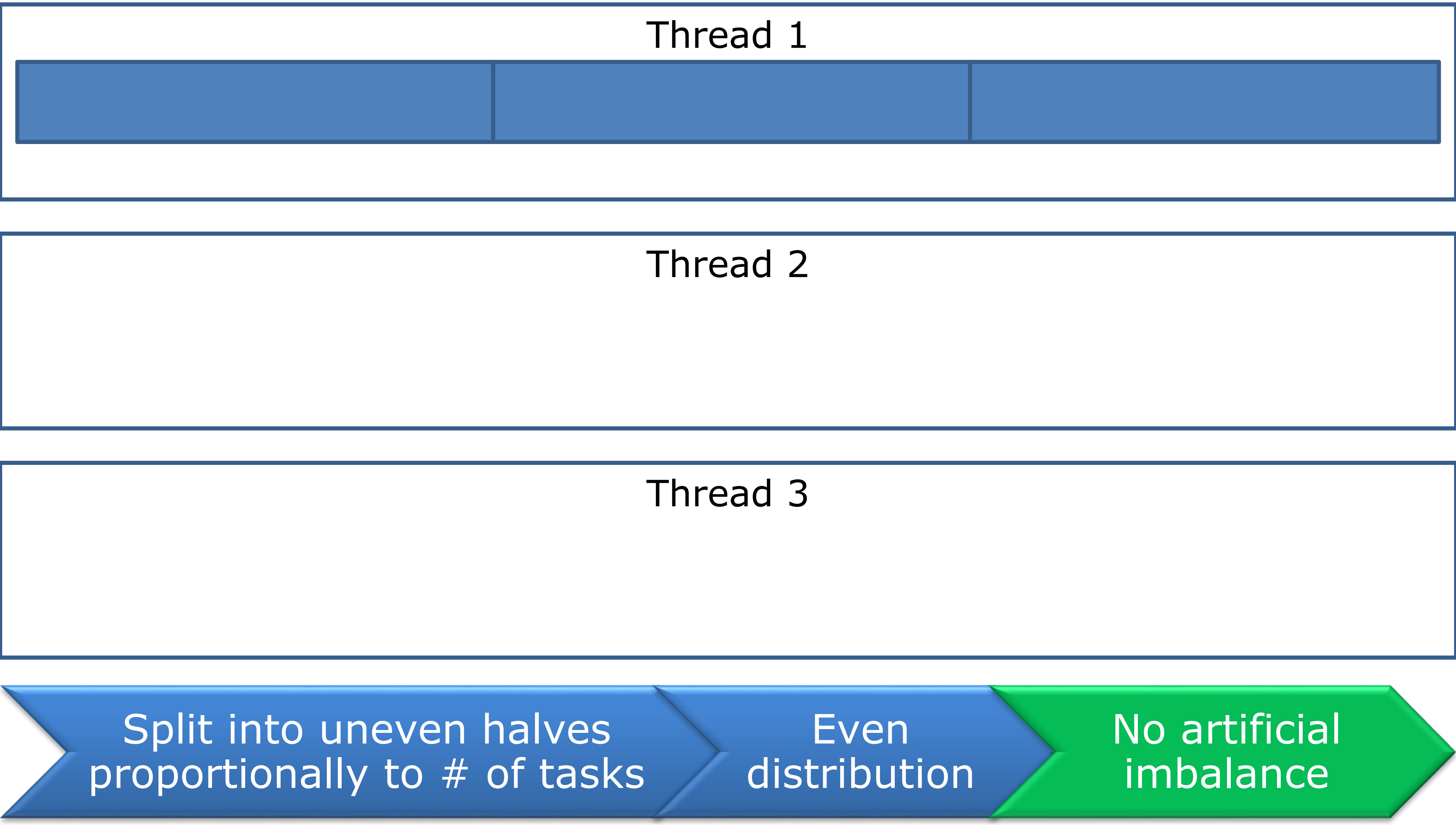


Simple work distribution: balancing

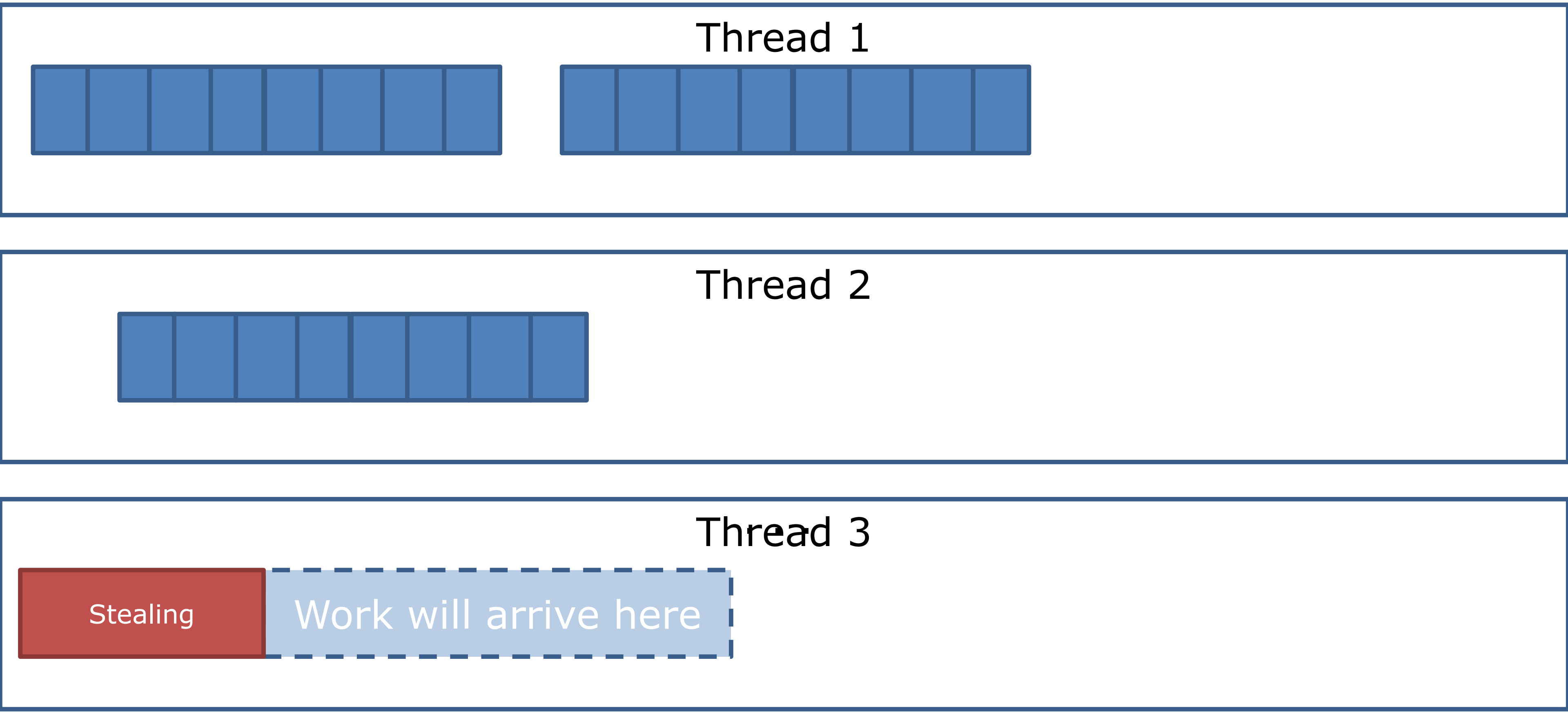


Can lead to inefficient stealing at the end, small grain-sizes

Proportional splitting

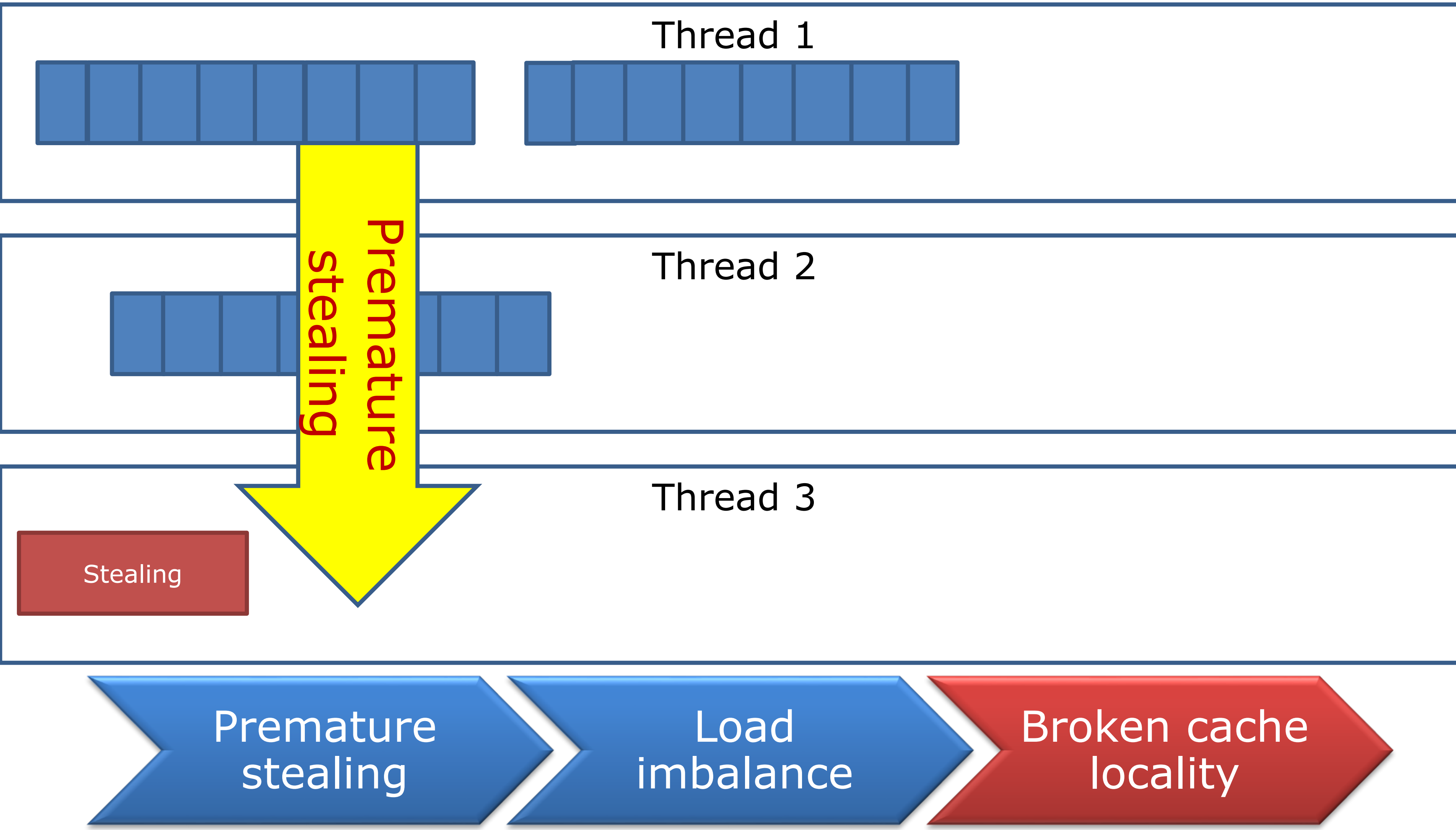


Premature stealing in Intel TBB



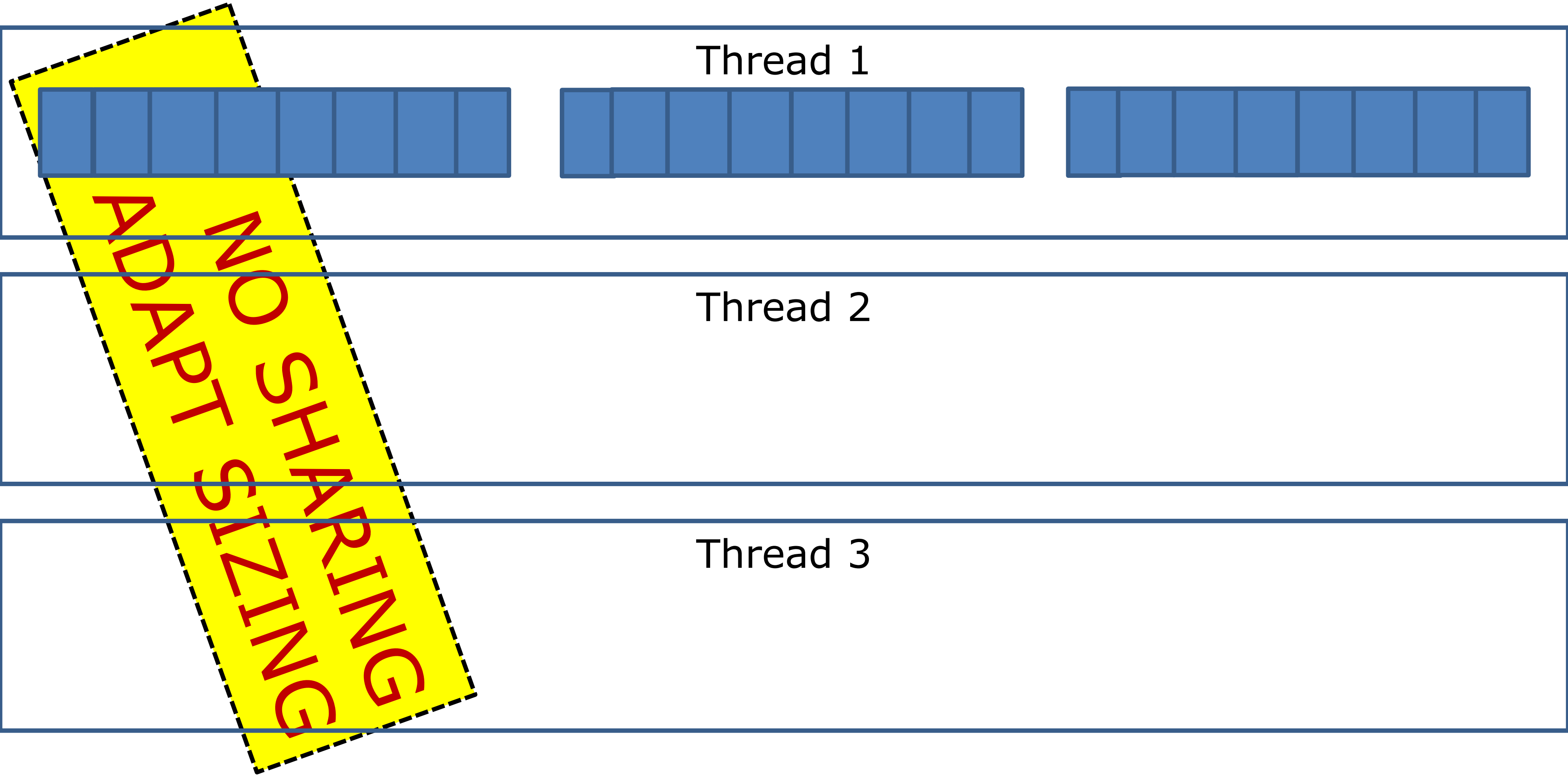
Stealing can occur earlier than the work supposed for the thread arrives

Premature stealing in Intel TBB



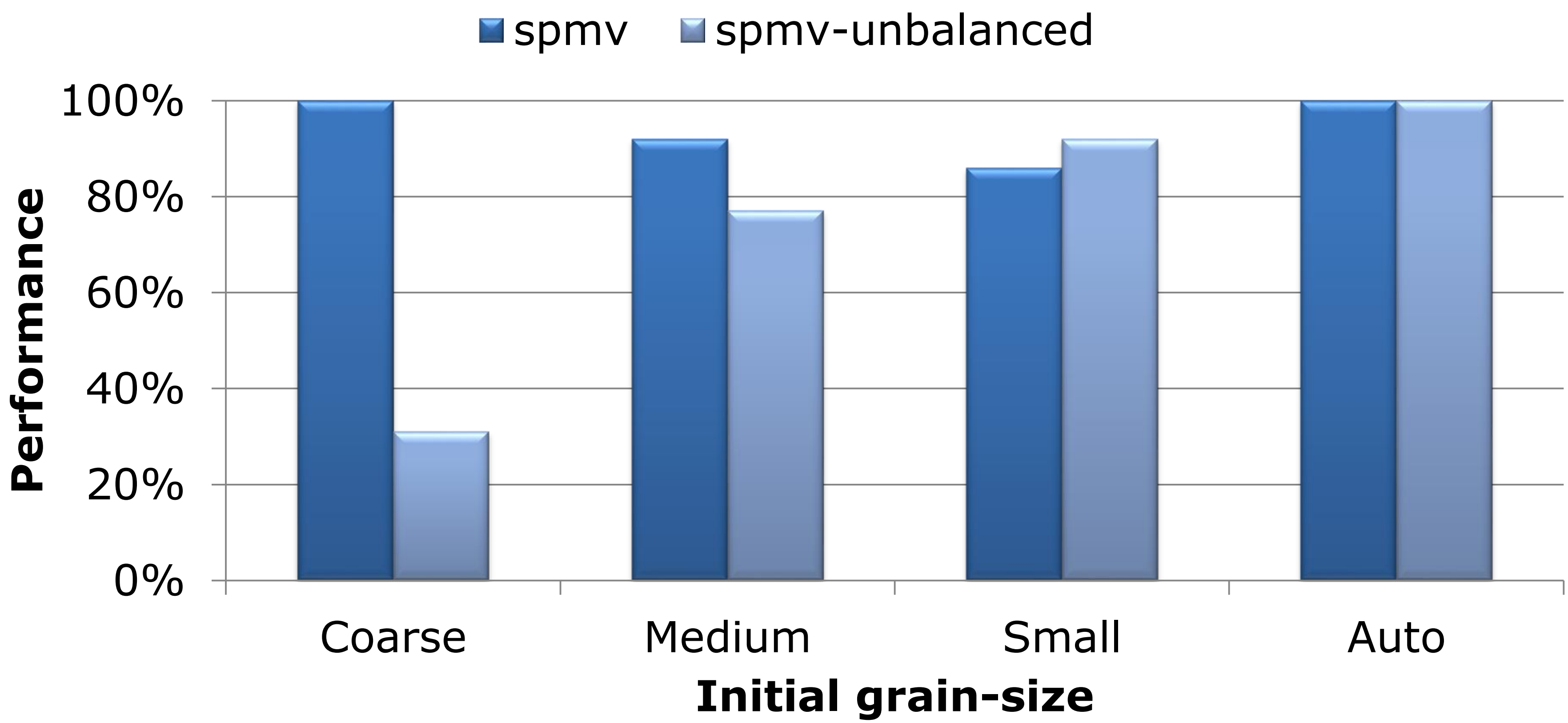
Optimization Notice

Solving last problems



RDTSC is used to put a ban on sharing the work → Also coarsen grain-size during this interval

Effect of automatic initial grain-size

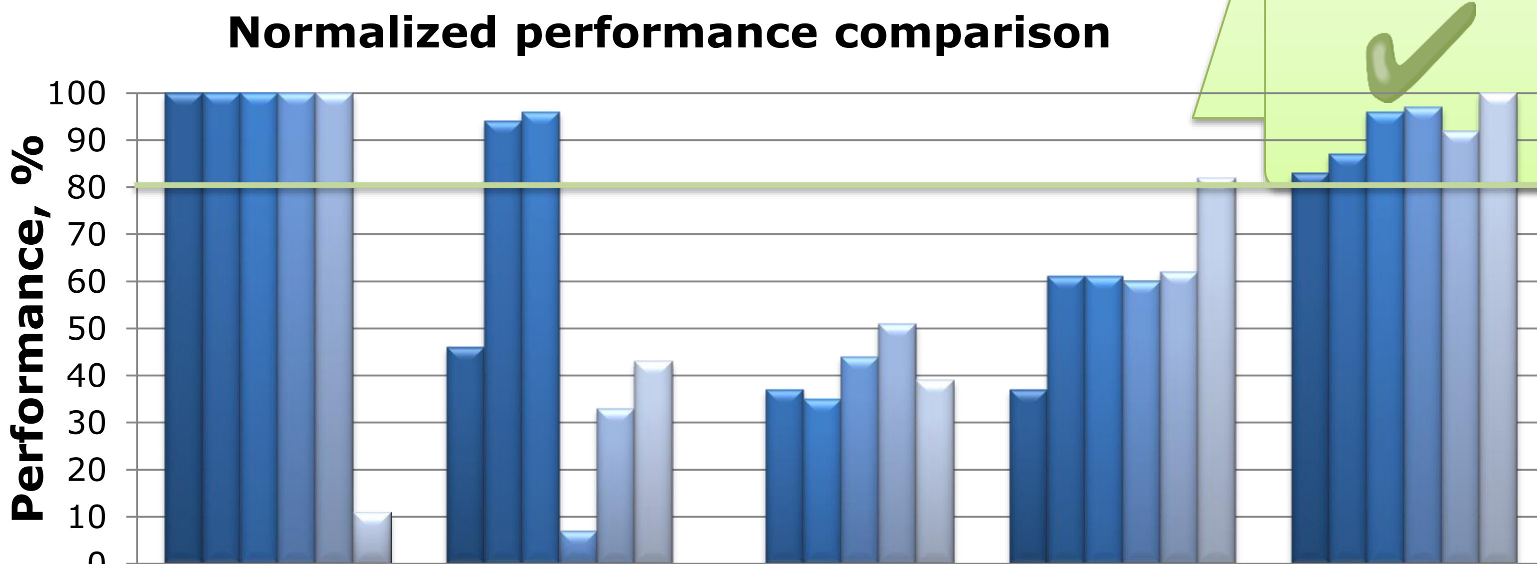


Universal solution provides good performance for both balanced and unbalanced workloads

Configuration Info - SW ... GHz, 61C/244T); MPSS Version: 3.1; Flas ... prise Linux Server release 6.2 (Santiago), ... Performance tests and ... ts as measured by those tests. Any differe ... s of information to evaluate the performanc ... of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

Optimization Notice

Final results



Threading: / Benchmarks	omp static	omp dynamic	TBB 4.1 defaults	TBB 4.2 U2 defaults	opencl partitioner
minimal	100	46	0	37	83
conformance	100	94	37	61	87
reduction	100	96	35	61	96
scan	100	7	44	60	97
spmv	100	33	51	62	92
spmv-unbalanced	11	43	39	82	100

Configuration Info - SW Versions: Intel® C++ Intel® 64 Compiler, Version 13.1.1.163; Hardware: Intel® Xeon Phi™ Coprocessor 7120 (16GB, 1.238 GHz, 61C/244T); MPSS Version: 3.1; Flash Version: 2.1.03.0386; Host: 2x Intel® Xeon® CPU E5-2680 0 @ 2.70GHz (16C/32T); 64GB Main Memory; OS: Red Hat Enterprise Linux Server release 6.2 (Santiago), kernel 2.6.32-220.el6.x86_64; Benchmarks are measured only on Intel® Xeon Phi™ Coprocessor with power management disabled. Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

Summary

Started from up to 3x gap with OpenMP* static

Major solutions:

Using task affinity	Unbalanced task tree	Even work distribution	Adaptive delay	Dynamic grain-size

Almost closed gaps with OpenMP* static

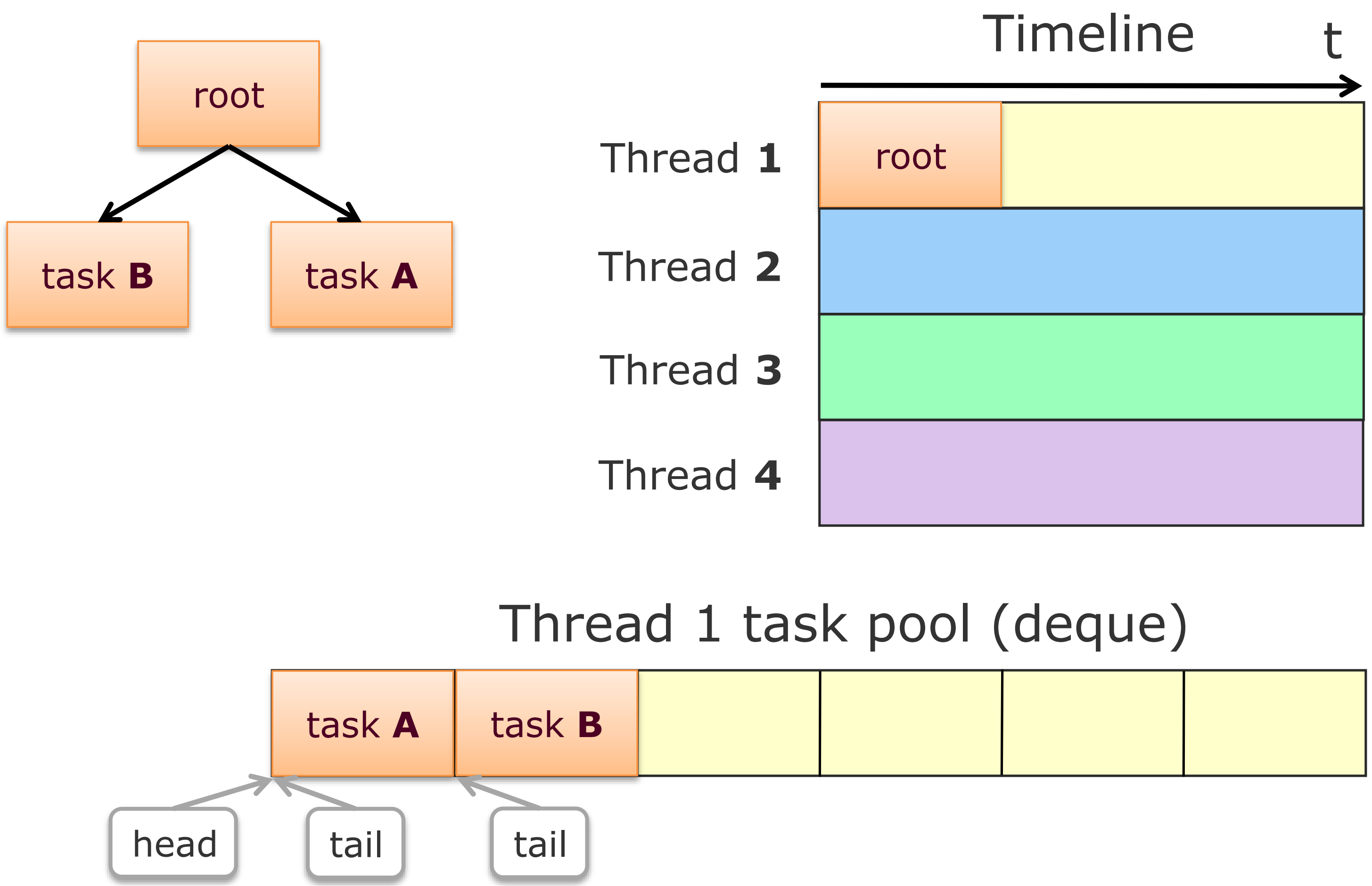
Legal Disclaimer & Optimization Notice

Optimization Notice

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 SSSE3 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.

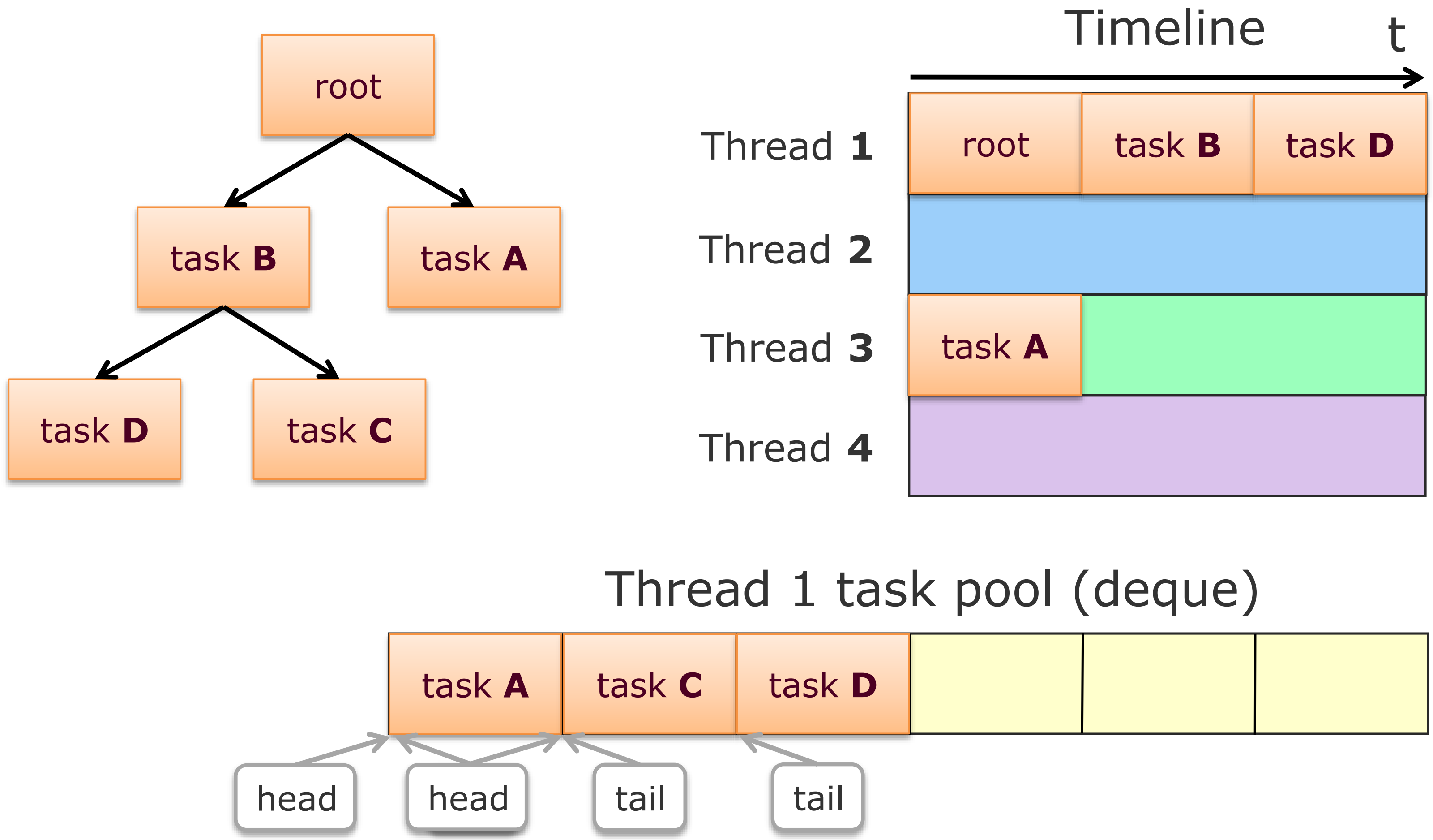
Notice revision #20110804

Executing and stealing tasks

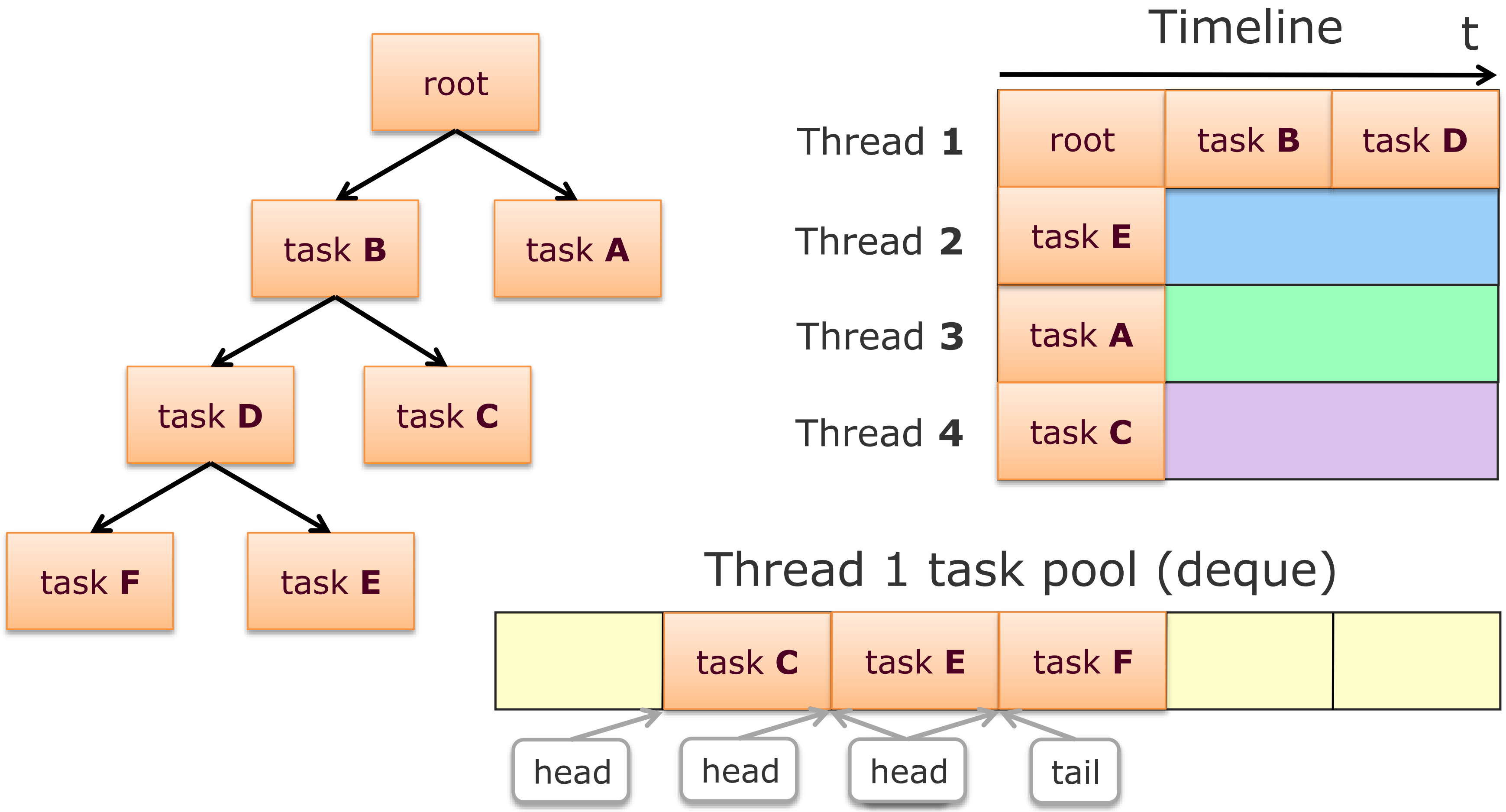


Optimization Notice

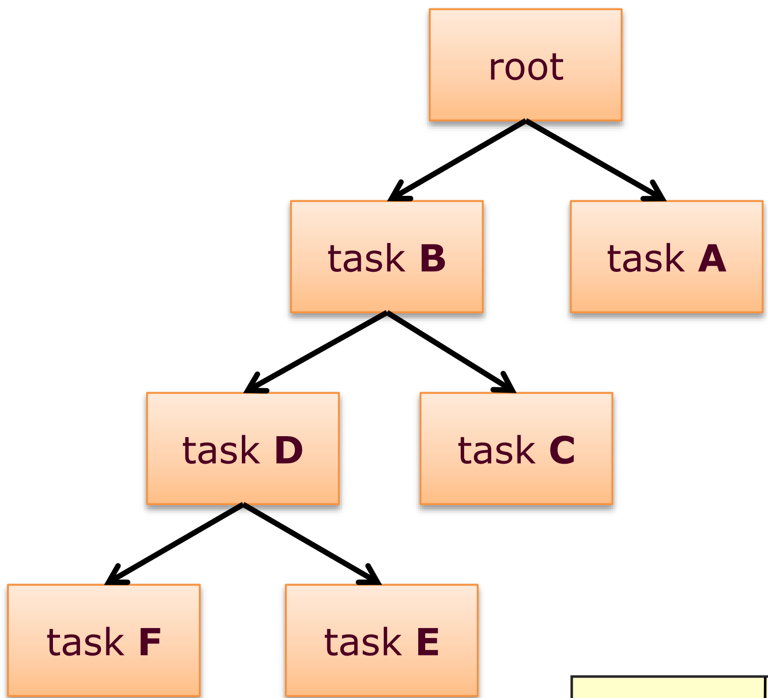
Executing and stealing tasks

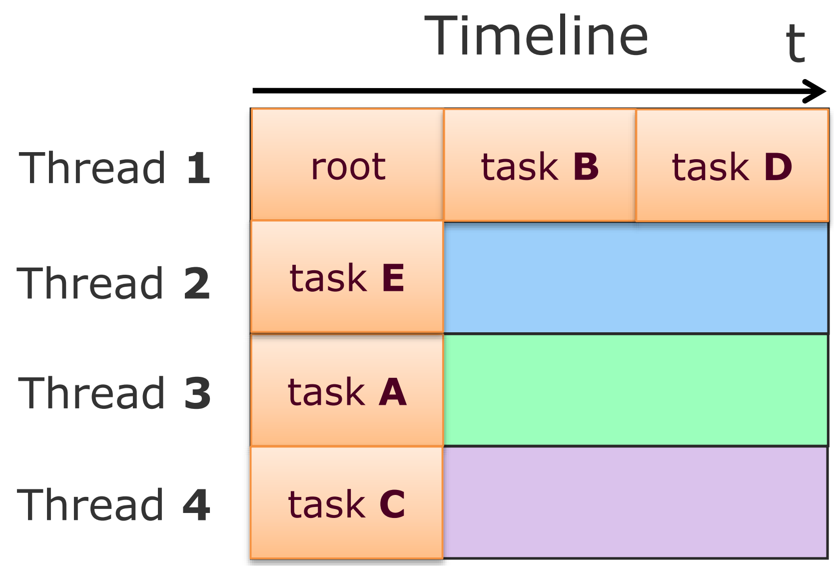


Executing and stealing tasks

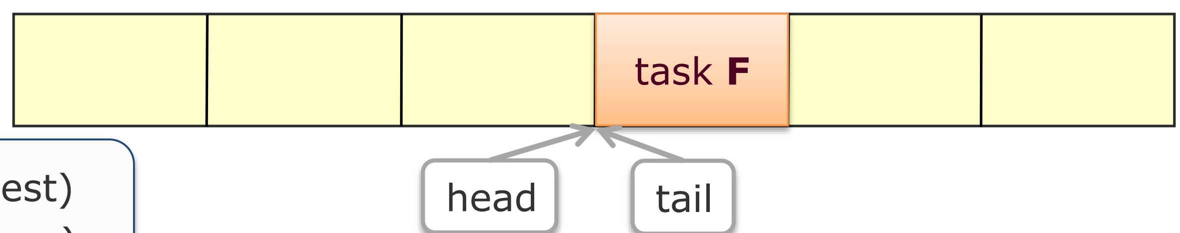


Executing and stealing tasks





Thread 1 task pool (deque)



executing from the back (newest)
stealing from the front (old/large)

when (tail – head) > 1, adding or taking tasks by local owner from a task pool is lock free

Optimization Notice

Distribution of 240 tasks to its threads

Configuration Info - SW Versions: Intel® C++ Intel® 64 Compiler, Version 13.1.1.163; Hardware: Intel® Xeon Phi™ Coprocessor 7120 (16GB, 1.238 GHz, 61C/244T); MPSS Version: 3.1; Flash Version: 2.1.03.0386; Host: 2x Intel® Xeon® CPU E5-2680 0 @ 2.70GHz (16C/32T); 64GB Main Memory; OS: Red Hat Enterprise Linux Server release 6.2 (Santiago), kernel 2.6.32-220.el6.x86_64; Benchmarks are measured only on Intel® Xeon Phi™ Coprocessor with power management disabled.
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.