Core-Scheduling for Virtualization: Where are We? (If We Want It!)

Dario Faggioli <*dfaggioli@suse.com*>
Software Engineer - Virtualization Specialist, **SUSE**
GPG: 4B9B 2C3A 3DD5 86BD 163E 738B 1642 7889 A5B8 73EE
https://about.me/dario.faggioli
https://www.linkedin.com/in/dfaggioli/
https://twitter.com/DarioFaggioli (@DarioFaggioli)

Dario Faggioli

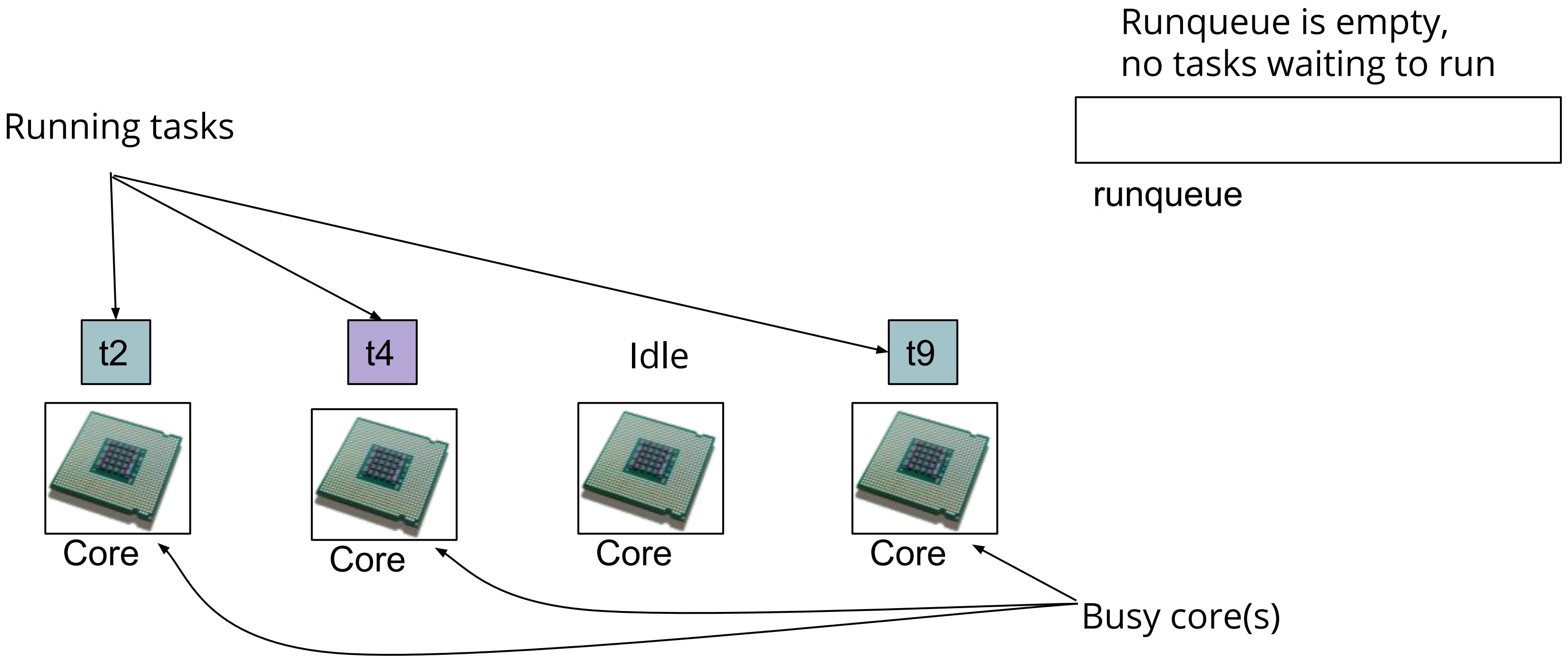
- Ph.D on Real-Time Scheduling, `SCHED_DEADLINE`
- 2011, Sr. Software Engineer @ Citrix
  The Xen-Project, hypervisor internals,
  NUMA-aware scheduler, Credit2 scheduler,
  Xen scheduler maintainer (still am)
- 2018, Virtualization Software Engineer @ SUSE
  Still Xen, but also KVM, QEMU, Libvirt;
  Scheduling, VM's virtual topology,
  performance evaluation & tuning

Scheduling & Core Scheduling

Scheduling

Could be vcpus of VMs

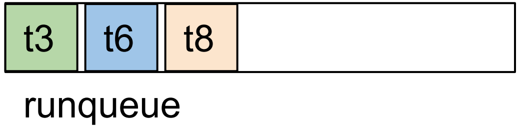
Letting tasks run on CPUs



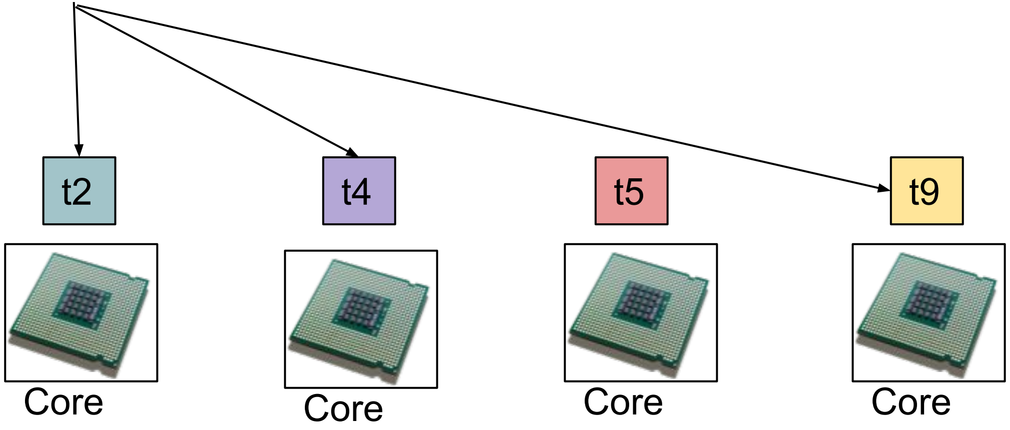
Scheduling

Letting as much tasks as possible to run on CPUs

Ready tasks. Would run,
but are waiting in runquque(s)
as there are not idle cores

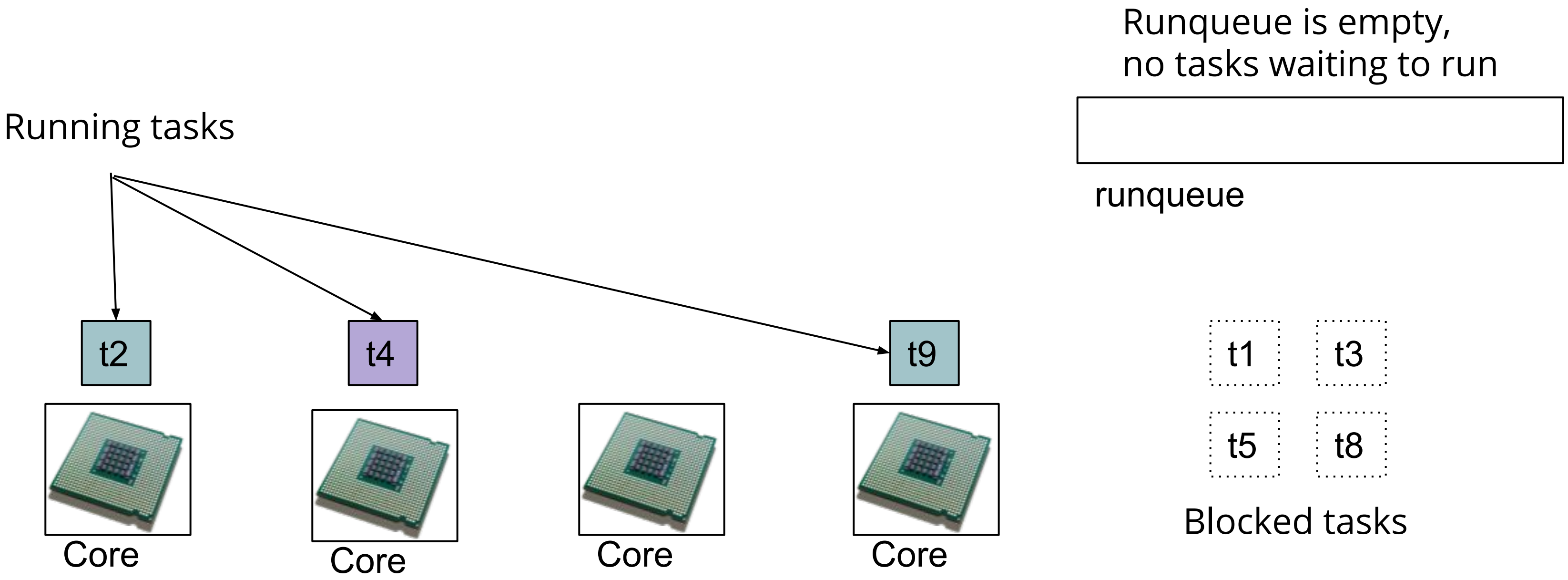


Running tasks



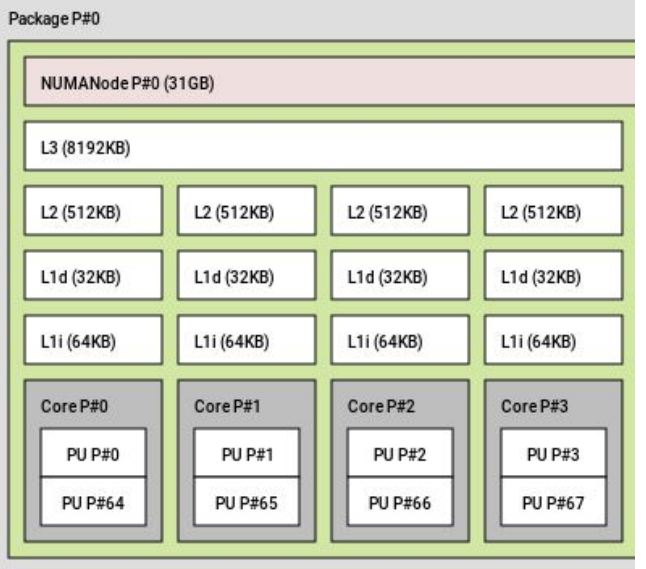
Scheduling

Letting runnable tasks run on CPUs



Simultaneous MultiThreading (SMT)

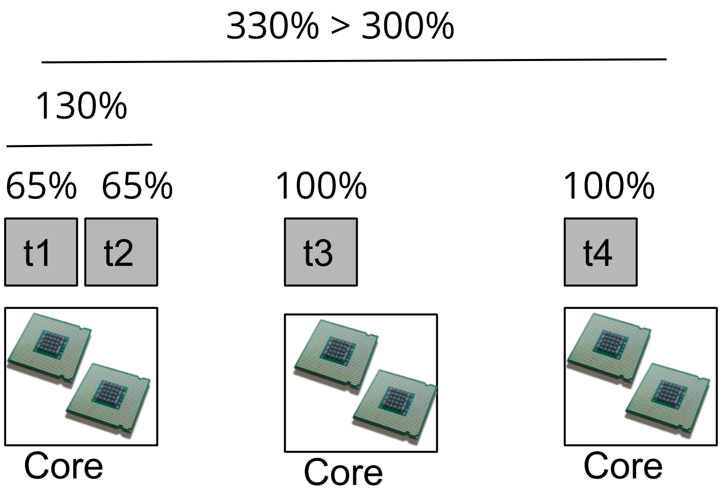
- Cores are split in Threads
- Multiple instruction streams at the same time
  - Increased parallelism
- Some CPU resources are shared
- Threads share caches (even L1s)
- Performance boost
  - Common knowledge: no more than +30%
  - Could be neuter or even be a slow down!
- x86:
  - Intel: since Pentium 4 (HyperThreading)
  - AMD: since Zen architecture



- 2 threads per core most common (but not necessarily)
- Different performance between 1 thread running vs. both
- Schedulers sees threads as CPUs
  - But they do deal with SMT already

SMT Execution

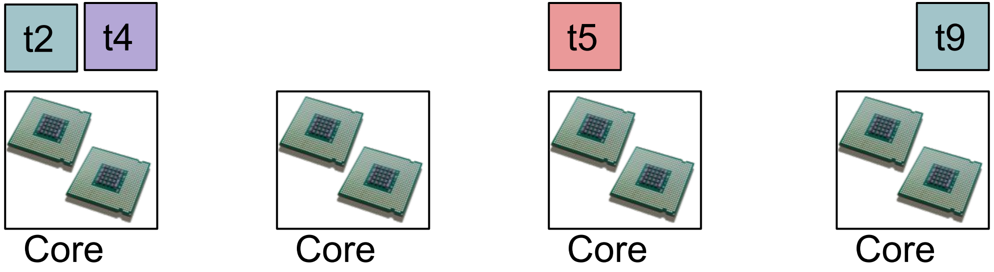
When all threads of a core are busy, tasks running on the core are slower:



- Overall: good. 330% speed is better than 300%, as it would be without SMT ~~(is it always?)~~
- Seen from t1 (or t2): it's slower! E.g., t3 and t4 run at 100% speed, t1 and t2 run at ~ 65%.

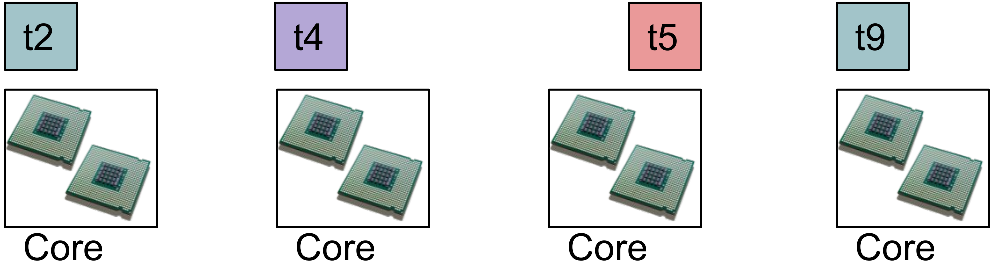
SMT Scheduling

Schedulers (should!) be SMT aware already



t2 + t4 + t5 + t9 =
65% + 65% + 100% + 100% =
330%

NO!



t2 + t4 + t5 + t9 =
100% + 100% + 100% + 100% =
400%

YES!

* if goal is performance

SMT: Is it Worth?

HyperThreading:
Intel implementation of SMT

(Intel) System with 4 Cores and HyperThreading (HT)

- 8 CPUs with HT enabled
- 4 CPUs with HT disabled

Stream	No HT vs. HT
copy	+3.93%
scale	+4.30%
add	+3.40%
triad	+3.63%

No!! No-HT is faster!

SMT: Is it Worth?

HyperThreading:
Intel implementation of SMT

(Intel) System with 4 Cores and HyperThreading (HT)

- 8 CPUs with HT enabled
- 4 CPUs with HT disabled

Stream	No HT vs. HT
copy	+3.93%
scale	+4.30%
add	+3.40%
triad	+3.63%

No!! No-HT is faster!

Kernbench	No HT vs. HT
-j2	+1.61%
-j4	+5.42%
-j8	-31.33%
-j16	-33.33%

Yes!! No-HT is 30% slower!

Core Scheduling: How it Works

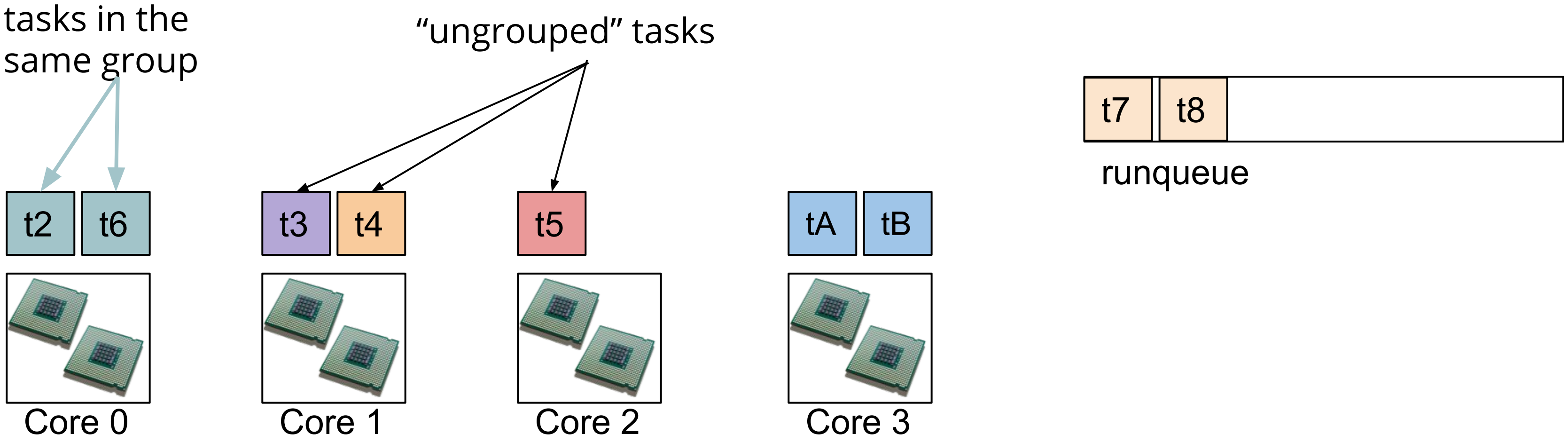
(Some) tasks are “grouped”

Tasks from same group ⇒ scheduled on same core

Never mix on same core tasks from different groups

Never mix on same core grouped and ungrouped tasks

Some CPUs (threads) may stay idle, even if runqueue is not empty



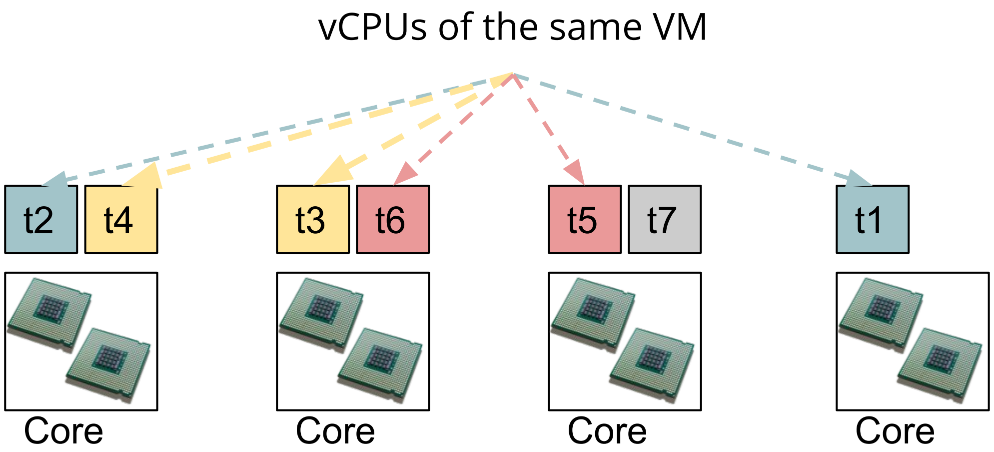
Motivations & Use Cases

Core Scheduling: Fairness of Accounting

Cloud, charging VMs for CPU time:

- t1, t2 are vCPUs of VM1 (from customer A)
- t3, t4 are vCPUs of VM2 (from customer B)
- T5 and t6 is vCPU of VM3 (from customer C)

Without Core Scheduling:



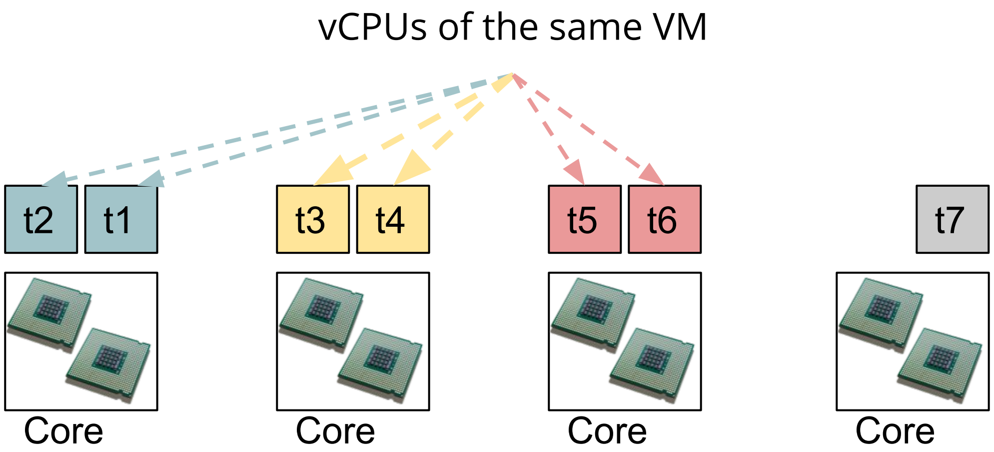
- t1 and t2, in VM1 run at different speeds
- t3 and t4, in VM2, run at different speeds
- Speed of VM1, and hence bill of customer A:
  - variable / not-consistent
  - influenced by VM2, and hence by customer B (and vice versa)

Core Scheduling: Fairness of Accounting

Cloud, charging VMs for CPU time:

- t2, t6 are vCPUs of VM1 (from customer A)
- t8, t7 are vCPUs of VM2 (from customer B)
- t5 is vCPU of VM3 (from customer C)

With Core Scheduling:



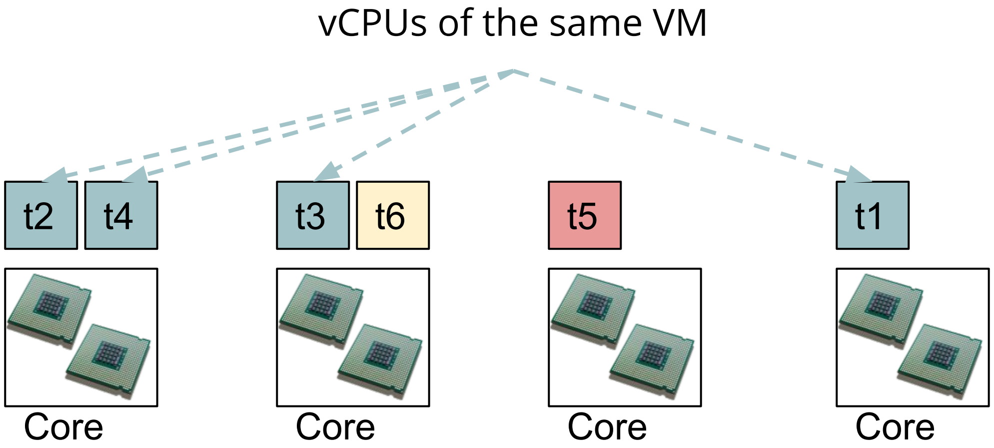
- Improved consistency
- No cross-VM (and cross-customer!!) side effects

Core Scheduling: in Guest Topology

Virtual Machines can have topologies:

- t1, t2, t3, t4 are vCPUs of VM1 (from customer A)
- VM1 ha a topology: 2 Core, with HT
  - t1 & t2 are “virtual HyperThread siblings”
  - t3 & t4 are “virtual HyperThread siblings”
- In-guest topology aware optimizations can be adopted (better perf.)

Without Core Scheduling:



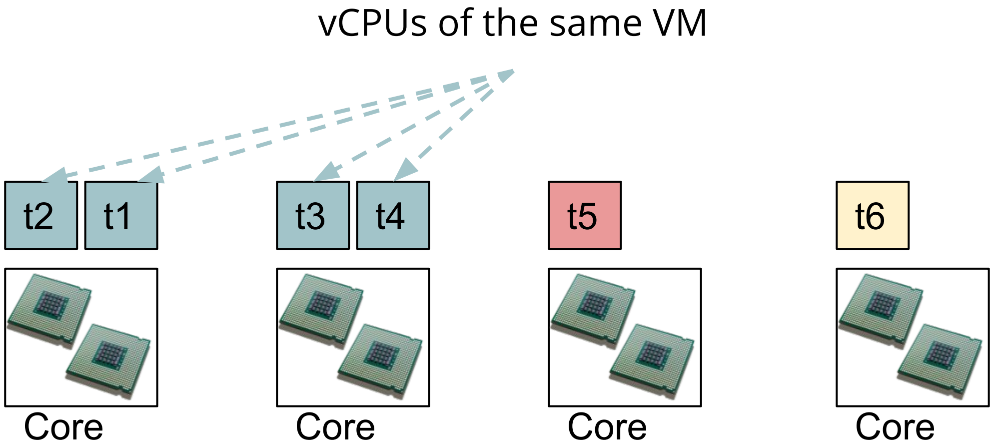
- t1, t2, t3 and t4 may run on any cores
- VM virtual topology not matching with where vCPUs run on host
- Guest scheduler will treat them as HyperThread siblings
- Suboptimal performance

Core Scheduling: in Guest Topology

Virtual Machines can have topologies:

- t1, t2, t3, t4 are vCPUs of VM1 (from customer A)
- VM1 ha a topology: 2 Core, with HT
  - t1 & t2 are "virtual HyperThread siblings"
  - t3 & t4 are "virtual HyperThread siblings"
- In-guest topology aware optimizations can be adopted (better perf.)

With Core Scheduling:



- t1 and t2 (t3 and t4) will always run together on a core the same core
- VM virtual topology will match with where vCPUs run on host
- Guest scheduler can safely treat them as HyperThread siblings
- Boost performance

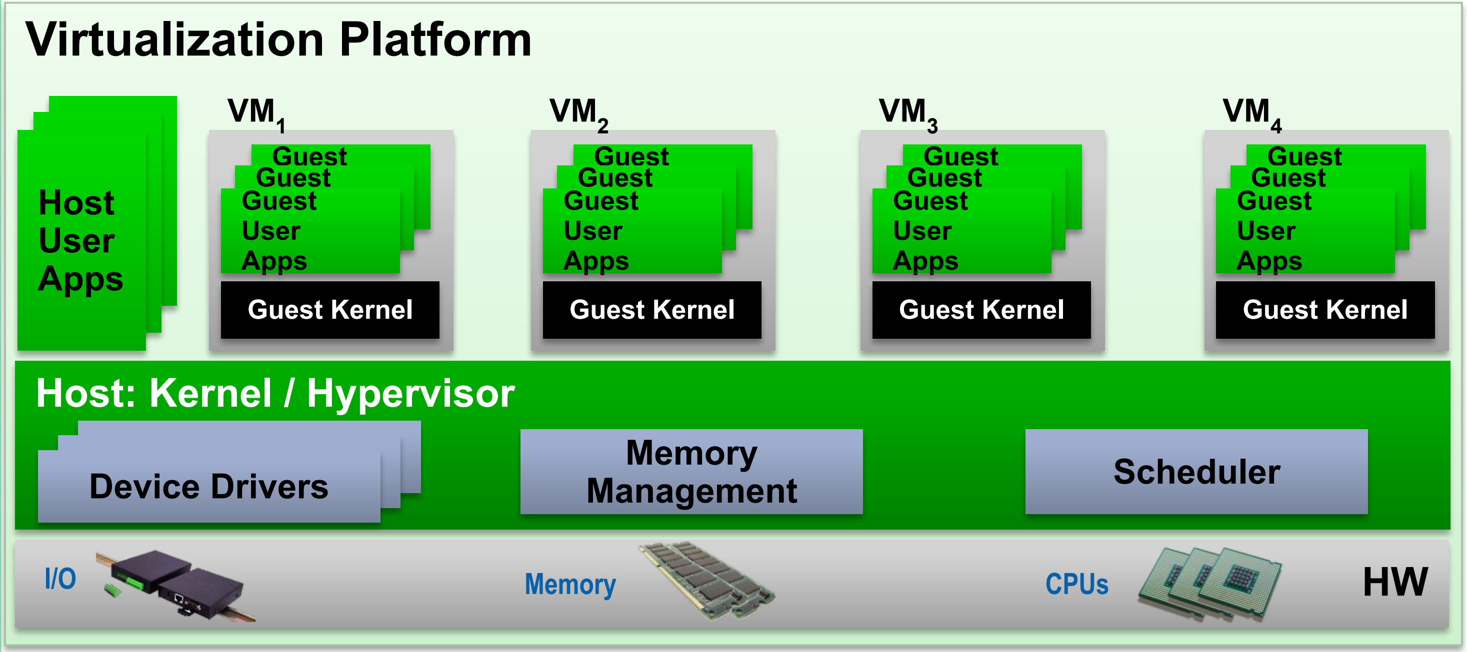
Core Scheduling: Security & Isolation

Spectre, Meltdown & Friends

- **Spectre v1** - Bounds Check Bypass
- **Spectre v2** - Branch Target Isolation
- **Meltdown** - Rogue Data Cash Load (a.k.a. Spectre v3)
- **Spectre v3a** - Rogue System Register Read
- **Spectre v4** - Speculative Store Bypass
- ...
- ...
- **LazyFPU** - Lazy Floating Point State Restore
- **L1TF** - L1 Terminal Fault (a.k.a. Foreshadow)
- **MDS** - Microarch. Data Sampling (a.k.a. Fallout, ZombieLoad, ...)
- ...

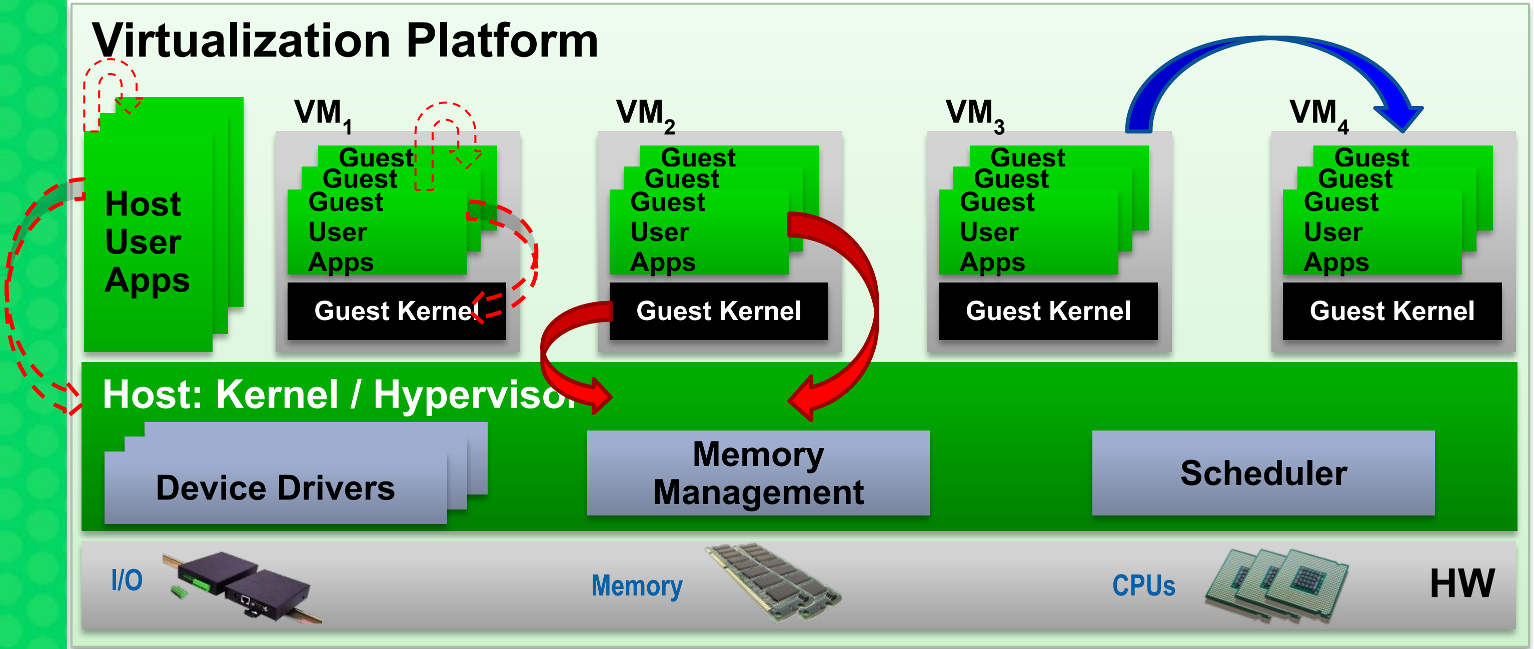
Virtualization, security, isolation ...

Attack Scenarios:



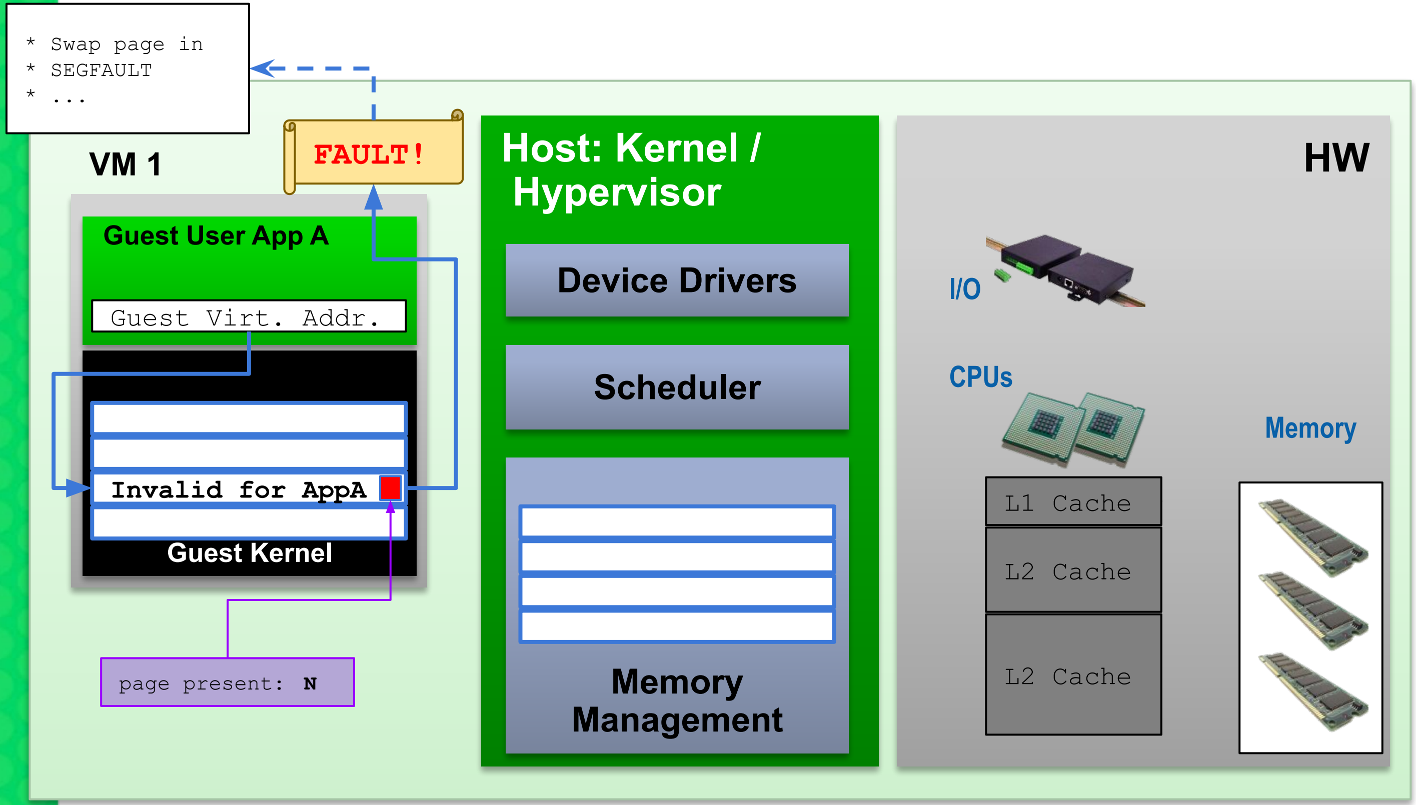
Virtualization, security, isolation ...

Attack Scenarios:



- Host User to Other Host User(s)
- Guest User to Other Guest User(s)
- Host User to Host Kernel
- Guest User to Guest Kernel
- Guest to Other Guest(s)
- Guest User to Hypervisor
- Guest Kernel to Hypervisor

L1TF - Virtualization (Foreshadow-NG, CVE-2018-3646)



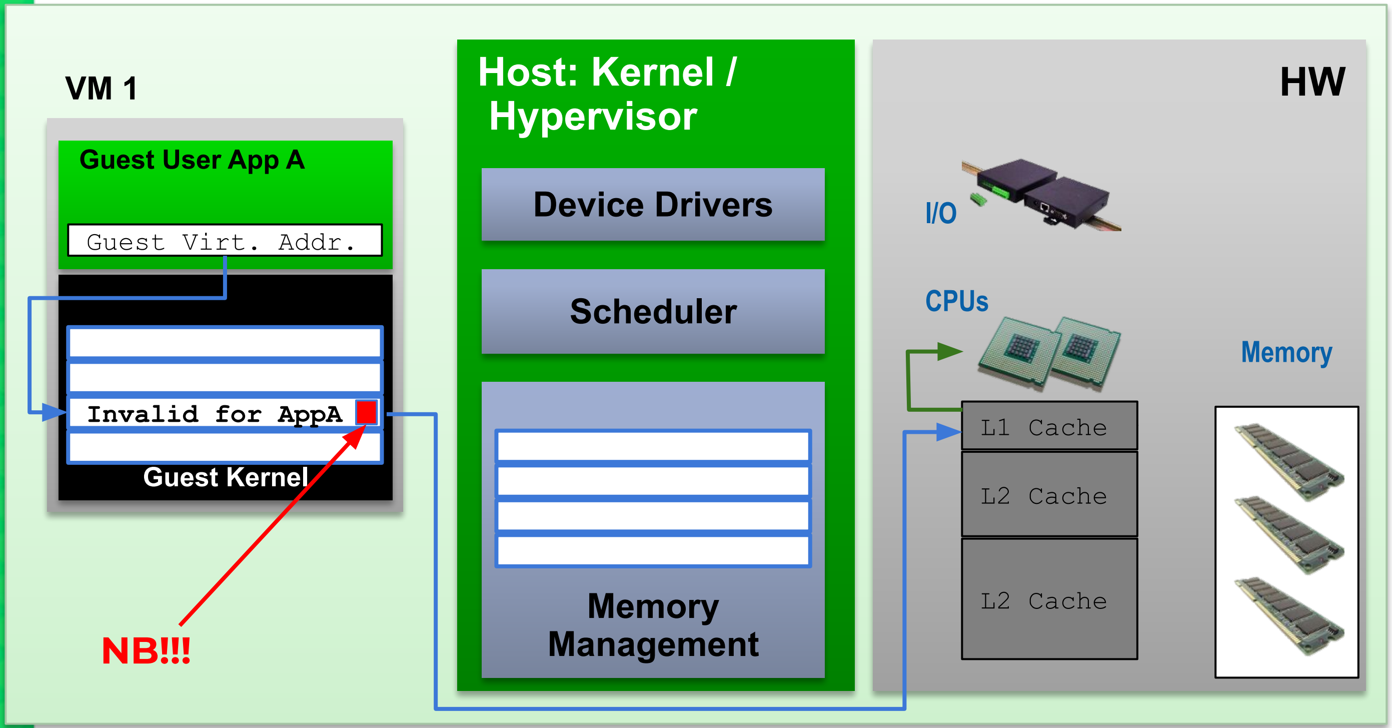
Regular execution

App accesses data in non present page:

1. Guest page tables
   page !present
2. Guest page fault

Potentially Malicious App A (e.g., trying to steal data within VM 1): **stopped!**

L1TF - Virtualization (Foreshadow-NG, CVE-2018-3646)



Speculative execution

App (speculatively) accesses data in non present page:

1. Guest page tables
   ~~page !present~~

~~2. Host page tables~~

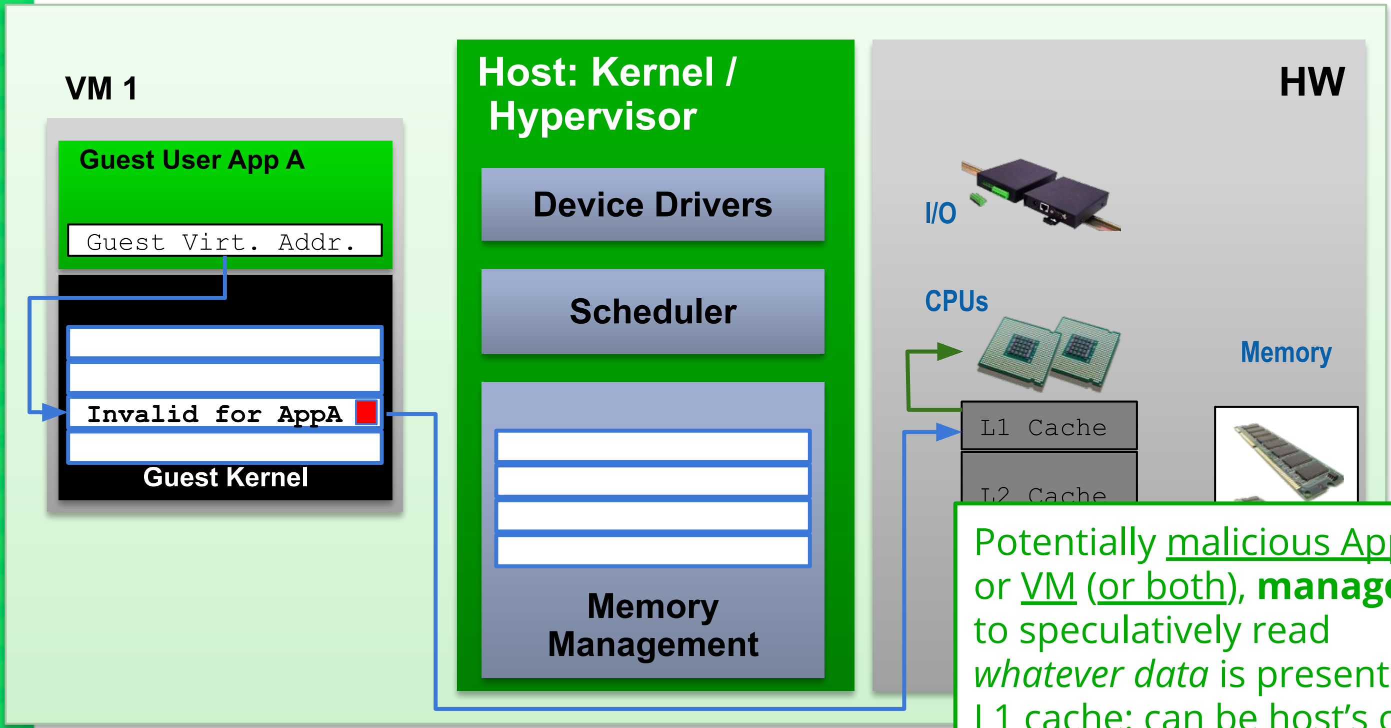
2. Check L1 cache

   ---

3. **Hit!** Load data in CPU

Wait... What?!?!

L1TF - Virtualization (Foreshadow-NG, CVE-2018-3646)



Potentially malicious App A, or VM (or both), **managed** to speculatively read *whatever data* is present in L1 cache: can be host's or other VMs' secrets!

Speculative execution

App (speculatively) accesses data in non present page:

1. Guest page tables
   ~~page !present~~
~~2. Host page tables~~
2. Check L1 cache

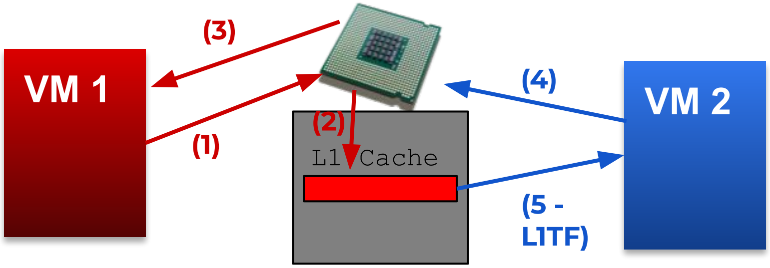
   ---

3. **Hit!** Load data in CPU

Wait… What?!?!

L1TF: VM-to-VM attack scenario

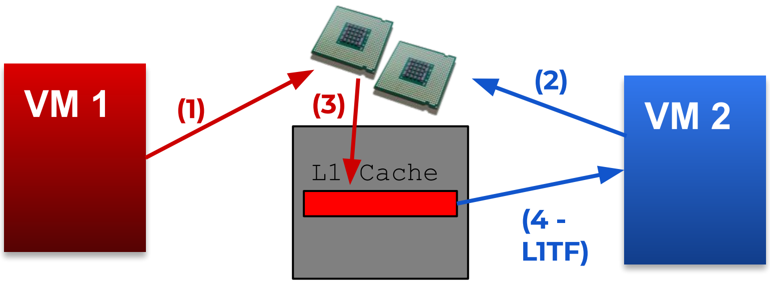
Sequential Context (no HyperThreading):



1. VM 1 runs on CPU
2. VM 1 puts secrets in L1 cache
3. VM 1 leaves CPU
4. VM 2 runs on CPU
5. **VM 2 reads VM 1's secrets!**

Context Switch

Concurrent Context (with HyperThreading):



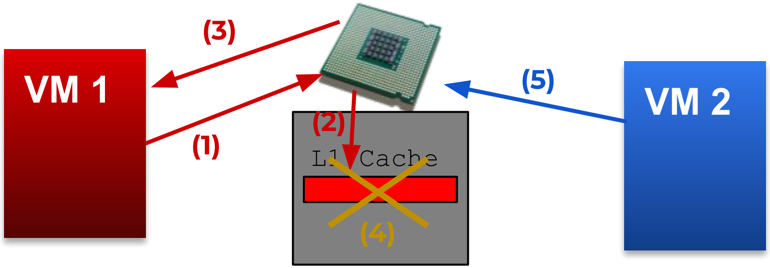
1. VM 1 runs on Thread A
2. VM 2 runs on Thread B
3. VM 1 puts secrets in L1 cache
4. **VM 2 reads VM 1's secret from L1 cache**

No context switch needed...

Guest (Kernel) to Other Guest(s) attack

L1TF: VM-to-VM attack scenario

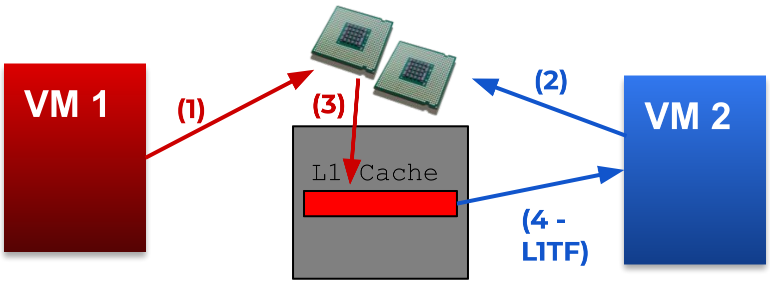
Sequential Context (no HyperThreading):



1. VM 1 runs on CPU
2. VM 1 puts secrets in L1 cache
3. VM 1 leaves CPU
4. Hypervisor: flush L1 cache
5. VM 2 runs on CPU
6. ~~VM 2 reads VM 1's secrets!~~

Context Switch

Concurrent Context (with HyperThreading):



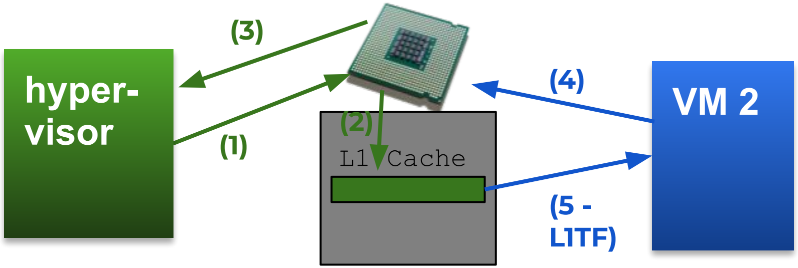
1. VM 1 runs on Thread A
2. VM 2 runs on Thread B
3. VM 1 puts secrets in L1 cache

   Hypervisor: THERE'S NOTHING I CAN DO !!!
4. **VM 2 reads VM 1's secret from L1 cache**

Guest (kernel) to Other Guest(s) attack

L1TF: VM-to-Hypervisor attack scenario

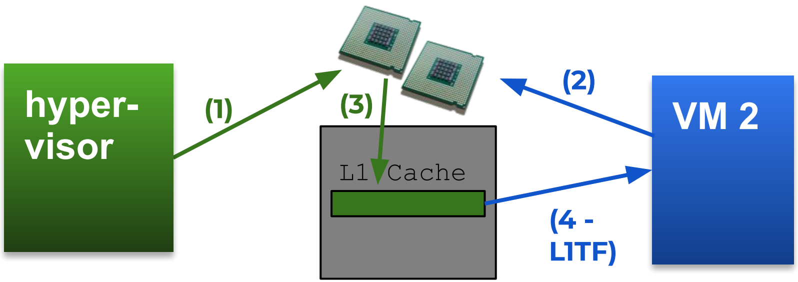
Sequential Context (no HyperThreading):



1. Hypervisor runs on CPU
2. Hypervisor puts secrets in L1
3. Hypervisor leaves CPU
4. VM 2 runs on CPU
5. **VM 2 reads hypervisor's secrets!**

VMEntry

Concurrent Context (with HyperThreading):



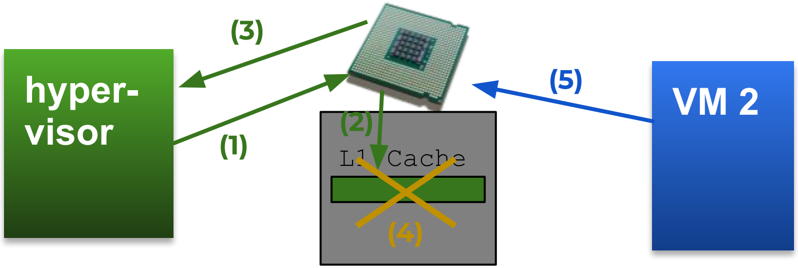
1. Hypervisor runs on Thread A
2. VM 2 runs on Thread B
3. Hypervisor puts secrets in L1
4. **VM 2 reads VM 1's secret from L1 cache**

No VMEntry needed...

Guest Kernel to Other Guest(s) attack

L1TF: VM-to-Hypervisor attack scenario

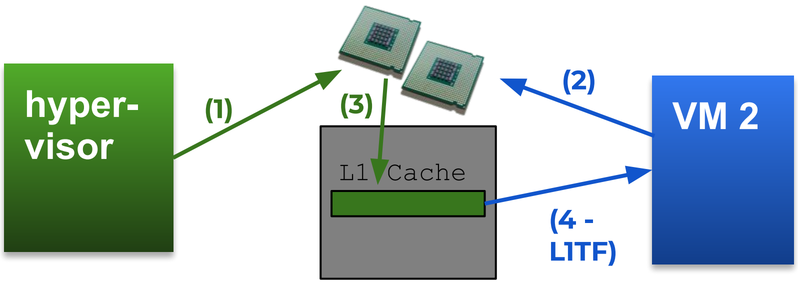
Sequential Context (no HyperThreading)



1. Hypervisor runs on CPU
2. Hypervisor puts secrets in L1
3. Hypervisor leaves CPU
4. Hypervisor: flush L1 cache
5. VM 2 runs on CPU
6. ~~VM 2 reads hypervisor's secrets!~~

VMEntry

Concurrent Context (with HyperTthreading)



1. Hypervisor runs on Thread A
2. VM 2 runs on Thread B
3. Hypervisor puts secrets in L1
   Hypervisor: THERE'S NOTHING I CAN DO !!!
4. **VM 2 reads Hypervisor's secret from L1 cache**

Guest kernel to Other Guest(s) attack

L1TF: Current Status

Mitigations:

- L1DFlush (Sequential Context), disable HyperThreading (Concurrent Context)

Still non-mitigated, *if HT on*

Almost impossible to detect (exp. with TSX) when attack is being performed

Attacker can (with TSX) scan physical memory with bandwidth of 1 gigabit/sec [*]

Ongoing efforts:

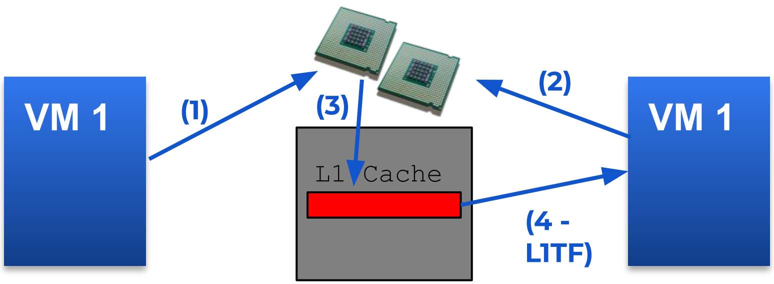
- Memory Isolation (e.g., guests|hypervisor)
  - "Kernel Page Table Isolation all the way down"
  - Can be effective in mitigating VM-to-Hypervisor concurrent contexts attacks
  - Not effective for VM-to-VM concurrent context attacks
- Core scheduling
  - **Needed** for mitigating concurrent contexts VM-to-VM attacks
  - Not effective for VM-to-hypervisor attacks

[*] Kernel Recipes 2019 - Kernel hacking behind closed doors

L1TF: Current Status

Mitigations:

- L1DFlush (Sequential Context), disable HyperThreading (Concurrent Context)



1. VM 1 runs on Thread A
2. VM 1 runs on Thread B
3. VM 1 puts secrets in L1 cache
   Hypervisor: THERE'S NOTHING I CAN DO !!!
4. **VM 1** reads **VM 1's** secret from L1 cache

"Hey, VM1, you're spying on yourself…"

- **Needed** for mitigating concurrent contexts VM-to-VM attacks
- Not effective for VM-to-hypervisor attacks

[*] Kernel Recipes 2019 - Kernel hacking behind closed doors

Core Scheduling Adoption Status

Core Scheduling in Hypervisors

- WMVware (ESX): they have
  - They have it: Side-Channel Aware Scheduler v2 (SCAv2)
  - Per-host (I think)

- Microsoft (Hyper-V):
  - They have it: The Hyper-V Core Scheduler
  - Basically per-host

- The Xen-Project (Xen hypervisor):
  - Will have it, next release (4.13), as “Experimental”
  - Core scheduling in the Xen hypervisor - SUSE Labs Conference 2019
  - Per-host (will become finer grained, but **not** per-VM)

Core Scheduling in Linux/KVM

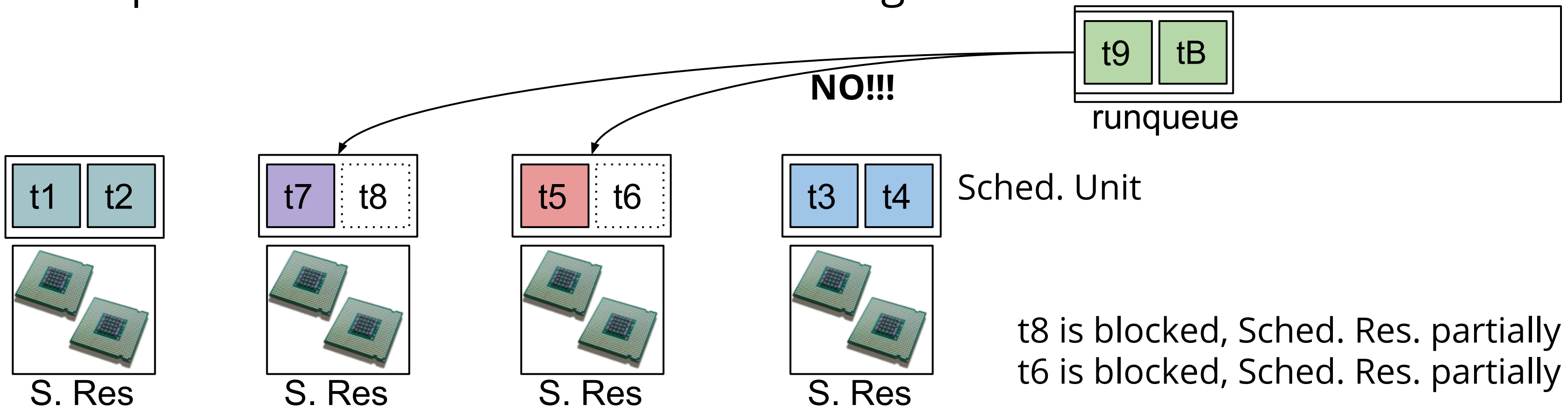
How it is being implemented:

- Co-Schedulable tasks are tagged ⇒ vCPUs of the same VM
  - Same tag (or no tag) ⇒ can be scheduled together on a core
- schedule() picks:
  - tagged task ⇒ task with the same tag on sibling(s); or idle
  - untagged task ⇒ untagged task on sibling(s); or idle
  - take priority into account
  - per-VM
- Challenges
  - How to quickly search for matching tagged task
  - task priority/vruntime weren't comparable across CPUs/runqueues
  - Fairness
  - Potential starvation
- Status: [RFC PATCH v3 00/16] Core scheduling v3

Core Scheduling: Xen Approach

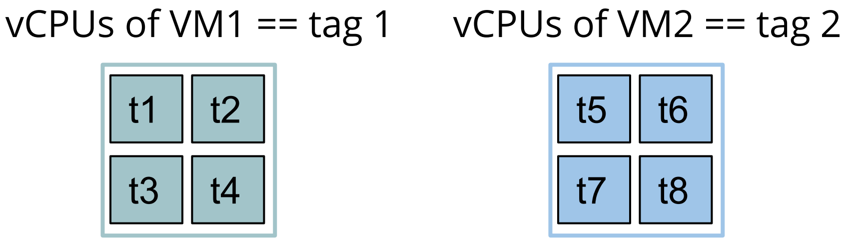
~~pCPUs and vCPUs~~ ⇒ Sched. Resources and Sched. Units

- Sched. Resource: a group of pCPUs (e.g., all pCPUs of a Core)
- Sched. Unit: a group of vCPUs (e.g., 2 on a system with SMT)
- Hypervisor scheduler schedules Units on Resources
- Which vCPUs are in which Sched. Unit never changes
- vCPUs within Sched. Units can block
- pCPUs within Sched. Resources can go idle



t8 is blocked, Sched. Res. partially idle
t6 is blocked, Sched. Res. partially idle

Core Scheduling: Linux (Patches) Approach

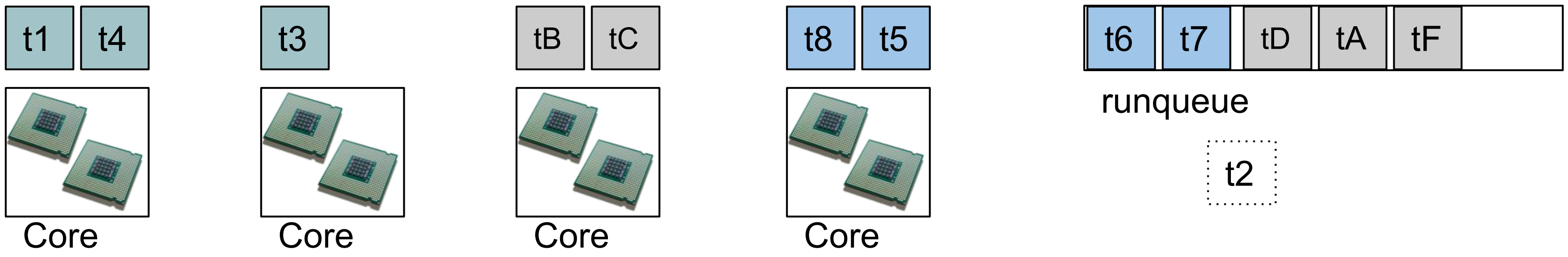


Flexible: works with any group of tasks

- +++ Very powerful
- --- Complex

E.g. per-VM tagging:

- Any vCPU of a VM is scheduled only together with other vCPUs of the VM
- time t=t_0



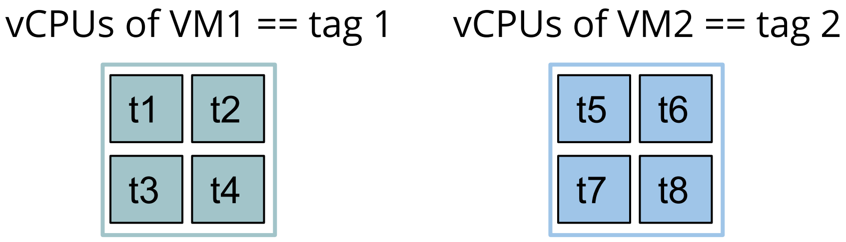
Core Scheduling: Linux (Patches) Approach

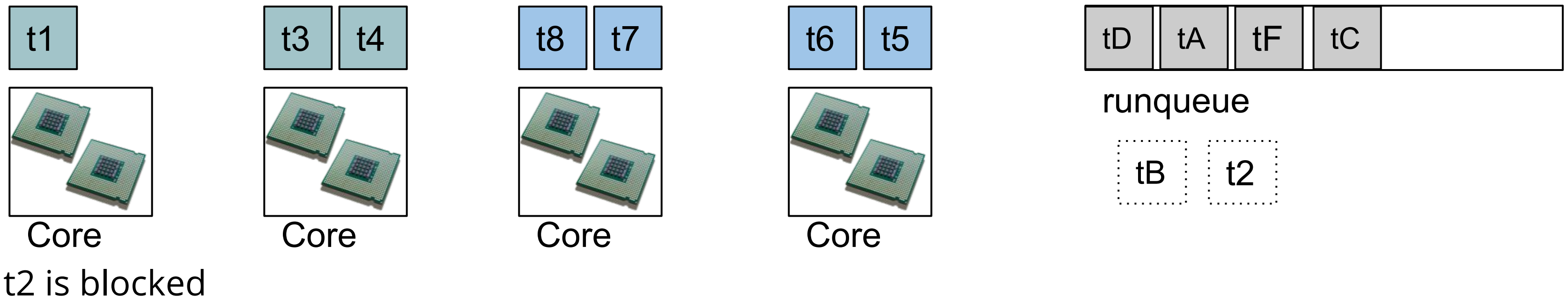
Flexible: works with any group of tasks

- +++ Very powerful
- --- Complex

E.g. per-VM tagging:

- Any vCPU of a VM is scheduled only together with other vCPUs of the VM
- later time t=t_0 + Dt



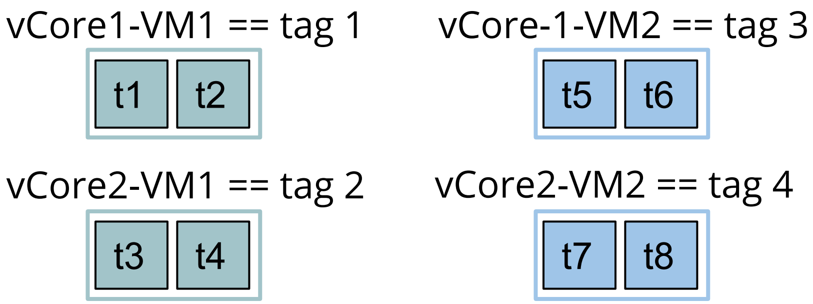


Core Scheduling: Linux (Patches) Approach

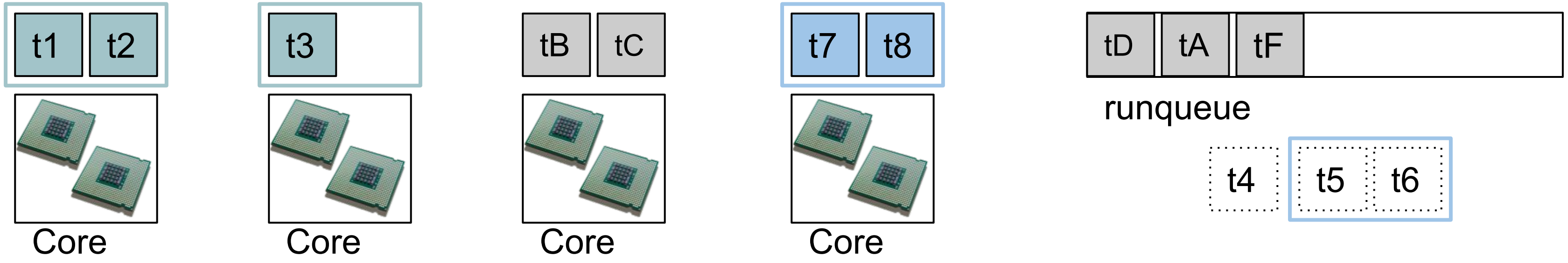
Flexible: works with any group of tasks

- +++ Very powerful
- --- Complex

E.g. virtual-Core tagging (as Xen does):



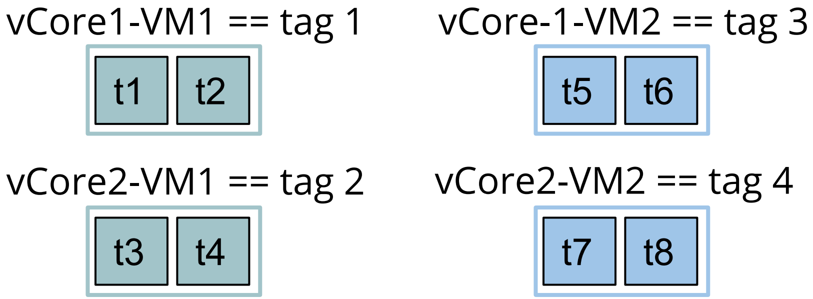
- Two vCPUs (of the same VM) are always scheduled together
- time t=t_0



Core Scheduling: Linux (Patches) Approach

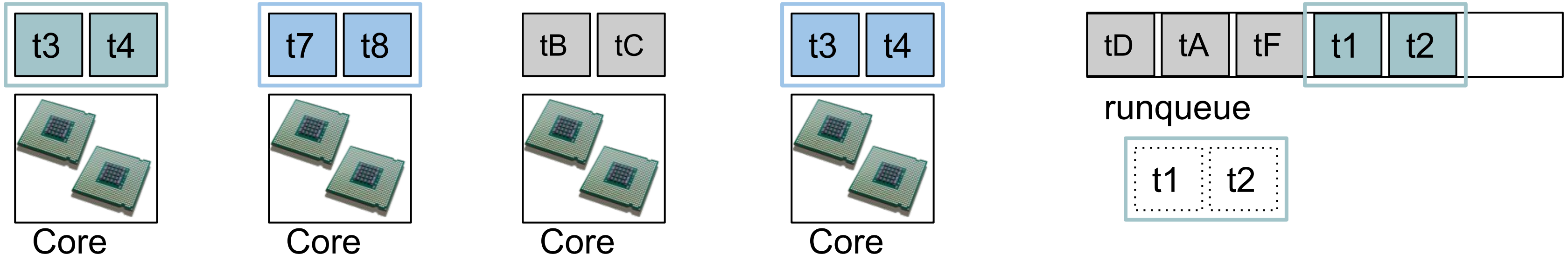
Flexible: works with any group of tasks

- +++ Very powerful
- --- Complex



E.g. virtual-Core tagging (as Xen does):

- Two vCPUs (of the same VM) are always scheduled together
- later time t=t_0 + Dt



Benchmarks

MMTests

- Historically for Memory Management testing, general now
- Fetching, building, configuring, running, collecting results, comparing
  - config-file == env. variables
  - shellpacks == wrappers!
- Monitors: perf, ftrace, ...
- Dashboards for comparing results
- Being enhanced for virt: dfaggioli/mmtests/tree/bench-virt
- **CPU/Memory benchmarks:** Hackbench, STREAM, NAS, Libmicro (syscall & glibc microbenchmarks), Speccpu2016, ...
- **IO benchmarks:** Iozone, Bonnie, Postmark, Reaim, Dbench4, ...
- Networking: Sockperf, Netperf, Netpipe, Siege, ...
- **Structured benchmarks:** Kernbench, Specjvm, Pgbench, Sqlite, Postgres & MariaDB OLTP benchmarks, ...

MMTests

```
./run-mmtests.sh BASELINE --config configs/config-netperf-unbound
./run-mmtests.sh PTI-ON --config configs/config-netperf-unbound

./bin/compare-mmtests.pl --directory work/log --benchmark netperf-tcp \
 --names BASELINE,PTI-ON
                               BASELINE                PTI-ON
    Hmean     64         1205.33 (   0.00%)      2451.01 ( 103.35%)
    Hmean     128        2275.90 (   0.00%)      4406.26 (  93.61%)
    … … …
    Hmean     8192      36768.43 (   0.00%)     43695.93 (  18.84%)
    Hmean     16384     42795.57 (   0.00%)     48929.16 (  14.33%)
```

Benchmarking Setup

- Test machine: Intel Xeon, 4 Cores, with HT (8 CPUs)
- (for Xen: Dom0 always with 8 vCPUs)
- VMs:
  - 1 VM with 8 vCPUs, or
  - 1 VM with 4 vCPUs, or
  - 2 VMs with 8 vCPUs each (overcommit)
- Scenarios (all results compared to “without patches, HT on”, positive numbers are better):
  - No HT
  - Patch overhead
  - With Core Scheduling

Benchmarking Setup

Benchmarks run inside VMs:

- STREAM: pure memory benchmark (various kind of mem-ops done in parallel, with parallelism NR_CPUS/2 tasks)
- Kernbench : builds a kernel, with varying number of compile jobs
- Hackbench : communication via pipes between group of processes
- mutilate : load generator for memcached, with high request rate
- netperf-unix : two communicating tasks, no pinning
- sysbenchcpu : the process-based CPU stressing workload of sysbench
- sysbenchthread : the thread-based CPU stressing workload of sysbench
- sysbench : the database workload

Full report:

https://lore.kernel.org/lkml/277737d6034b3da072d3b0b808d2fa6e110038b0.camel@suse.com/

DISCLAIMER:

These are the results of an ongoing effort.
If some of the numbers appear weird and difficult to understand or explain… It’s because they actually are!!

VIEWER DISCRETION ADVISED

Analysis of results and related data is still being carried on.

Stay tuned for updates.

Benchmarks Results

Means that without HT we are slower than with HT, by 12.34%.This tells us how sensitive to HT a benchmark is

Overhead: how slower we are, just with the patches applied and core scheduling not being used, wrt without any patches. Ideally, this would be 0%. If positive, means just applying the patches improves performance.

Baremetal	No HT	Patch applied	Core-sched
1 thread	-12.34%	-3.45%	-2.34%
4 threads			
7 threads			
[illegible]hreads	-1567.8%		

Means that with core scheduling, we are slower than with HT and without core scheduling by 2.34%. Ideally, this would be 0%, and higher than the ‘no HT’ column.

This would mean that core scheduling has “less worse” performance than HT disabled, so it is a good thing (for core scheduling)

NB: Large negative (< -100%) values mean that performance are worse by 1567.8% (non that we are going back in time very fast!)

Core-Scheduling in Xen: Performance

1x VM, 8 vCPU

	No HT	Patch Overhead	Coresched
Stream	+2.82% … +6.84%	+0.47% … +5.07%	-14.91% … -9.55%
Kernbench	-46.41% ... +6.04%	+0.46% … +1.70%	-6.91% … +0.19%
Hackbench	-50.23% … +4.17%	-14.08% … +14.06%	-16.51% … +11.06%
Mutilate	-68.33% … -6.48%	-1.11% … +2.33%	-45.50% … -6.17%
Netperf	-11.87% … +25.95%	-15.48% … +14.57%	-18.00% … +1.81%

Core-Scheduling in Xen: Performance

Overhead not too bad, except than for Hackbench and Netperf, where it varies

1x VM, 8 vCPU

		Patch erhead	Coresched
Stream	+2.82% … +6.84%	+0.47% … +5.07%	-14.91% … -9.55%
Kernbench	-46.41% … +6.04%	+0.46% … +1.70%	-6.91% … +0.19%
Hackbench	-50.23% … +4.17%	-14.08% … +14.06%	-16.51% . +11.06%
Mutilate	-68.33% … -6.48%	-1.11% … +2.33%	-45.50% . -6.17%
Netperf	-11.87% … +25.95%	-15.48% … +14.57%	-18.00% … +1.81%

Core scheduling does better than disabling HT (not in Stream and netperf, though)

Core-Scheduling in Xen: Performance

2x VMs, 8 vCPUs each (overload)

	No HT	Patch Overhead	Coresched
Stream	-26.13% … -22.94%	-0.87% … +1.45%	-13.34% … -6.37%
Kernbench	-50.26% ... -48.38%	-0.24% … -0.13%	-23.98% … -17.84%
Hackbench	+15.02% … +35.59%	-2.28% … +5.42%	-12.19% … +16.91%
Mutilate	-93.85% … -56.82%	-2.19% … +8.57%	-83.70% … -13.03%
Netperf	-50.48% … -15.77%	-16.39% … +7.61%	-36.22% … +4.41%

Core-Scheduling in Xen: Performance

Even with overcommit, overhead stays low enoug (except with Netperf)

2x VMs, 8 vCPUs each (overload)

	[illegible]	Patch …erhead	Coresched
Stream	-26.13% … -22.94%	-0.87% … +1.45%	-13.34% … -6.37%
Kernbench	-50.26% ... -48.38%	-0.24% … -0.13%	-23.98% … -17.84%
Hackbench	+15.02% … +35.59%	-2.28% … +5.42%	-12.19% … +16.91%
Mutilate	-93.85% … -56.82%	-2.19% … +8.57%	-83.70% … -13.03%
Netperf	-50.48% … -15.77%	-16.39% … +7.61%	-36.22% … +4.41%

Under overcommit, core scheduling is always better than disabling HT

Core Scheduling Performance: hackbench

Baremetal	No HT	Patch applied	Core-sched
1 group	-53.57%	2.97%	-162.95%
5 groups	-38.20%	0.12%	-768.32%
24 groups	-16.45%	-1.54%	-1372.10%
32 groups	-27.71%	-0.63%	-1597.64%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 group	-148.80%	1.95%	-0.27%
5 groups	2.23%	9.79%	14.94%
24 groups	-24.67%	8.15%	-11.92%
32 groups	-8.64%	6.71%	10.72%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 group	1.80%	15.47%	6.58%
5 groups	-0.21%	16.45%	4.06%
7 groups	5.69%	2.48%	10.10%
16 groups	-1.82%	7.65%	14.75%

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 group	-217.43%	-147.64%	-205.78%
5 groups	-48.96%	-13.78%	-19.03%
24 groups	-55.35%	-33.21%	-30.90%
32 groups	-62.32%	-44.62%	-43.27%

Core Scheduling Performance: hackbench

Baremetal	No HT	Patch applied	Core-sched
1 group	-53.57%	2.97%	-162.95%
5 groups	-38.20%	0.12%	-768.32%
24 groups	-16.45%	-1.54%	-1372.10%
32 groups	-27.71%	-0.63%	-1597.64%

Whaaat?!?!

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 group	-148.80%	1.95%	-0.27%
5 groups	2.23%	9.79%	14.94%
24 groups	-24.67%	8.15%	-11.92%
32 groups	-8.64%	6.71%	

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 group	1.80%	15.47%	6.58%
5 groups	-0.21%	16.45%	4.06%
7 groups	5.69%	2.48%	10.10%
16 groups	-1.82%	7.65%	14.75%

2x VM, 8 vCPU		atch plied	
1 group	-217.43%	-147.64%	-205.78%
5 groups	-48.96%	-13.78%	-19.03%
24 groups	-55.35%	-33.21%	-30.90%
32 groups	-62.32%	-44.62%	-43.27%

Under overcommit, overhead is high

At least this is good (compare with first column, i.e., 'no HT')

Core Scheduling Performance: sysbench

Baremetal	No HT	Patch applied	Core-sched
1 thread	-6.07%	4.01%	-4.37%
4 threads	5.93%	6.83%	0.16%
7 threads	-8.95%	-0.35%	2.62%
8 threads	3.08%	19.19%	14.72%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 thread	4.95%	-4.38%	-26.91%
4 threads	16.14%	0.67%	-20.76%
7 threads	14.17%	30.14%	-20.11%
8 threads	-19.96%	-19.70%	-37.34%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 thread	43.63%	-21.90%	-18.30%
2 threads	-0.69%	-0.97%	-14.80%
3 threads	25.45%	5.60%	11.88%
-			

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 thread	-8.34%	12.68%	-50.73%
4 threads	-46.29%	-27.06%	-66.91%
7 threads	-50.93%	-18.41%	-68.46%
8 threads	-52.64%	-28.65%	-59.74%

Core Scheduling Performance: sysbench

Baremetal	No HT	Patc appli	d
1 thread	-6.07%	4.01%	-4.37%
4 threads	5.93%	6.83%	0.16%
7 threads	-8.95%	-0.35%	2.62%
8 threads	3.08%	19.19%	14.72%

So, baremetal is fine

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 thread	4.95%	-4.38%	-26.91%
4 threads	16.14%	0.67%	-20.76%
7 threads	14.17%	30.14%	-20.11%
8 threads	-19.96%	-19.70%	-37.34%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 thread	43.63%	-21.90%	-18.30%
2 threads	-0.69%	-0.97%	-14.80%
3 threads	25.45%	5.60%	11.88%
-			

Virtualization, not so much!

2x VM, 8 vCPUs	No HT		ched
1 thread	-8.34%	12.68%	-50.73%
4 threads	-46.29%	-27.06%	-66.91%
7 threads	-50.93%	-18.41%	-68.46%
8 threads	-52.64%	-28.65%	-59.74%

Core Scheduling Performance: STREAM

Baremetal	No HT	Patch applied	Core-sched
copy	-0.51%	-0.43%	-0.75%
scale	-1.05%	-1.52%	-1.32%
add	0.38%	1.60%	-0.09%
triad	0.12%	-0.09%	-0.06%

VM, 8 vCPUs	No HT	Patch applied	Core-sched
copy	-1.17%	-0.93%	-3.15%
scale	1.24%	0.78%	0.89%
add	1.64%	1.84%	1.82%
triad	-0.12%	0.29%	0.33%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
copy	0.76%	0.37%	-1.03%
scale	2.40%	2.32%	0.47%
add	1.12%	0.03%	-1.58%
triad	0.23%	-0.03%	-0.66%

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
copy	-26.18%	-9.26%	-14.31%
scale	-31.55%	-15.72%	-17.07%
add	-29.29%	-19.45%	-20.46%
triad	-26.69%	-21.74%	-20.33%

Core Scheduling Performance: STREAM

Not particularly HT sensitive benchmark…

Bareme			Core-sched
copy	-0.51%	-0.43%	-0.75%
scale	-1.05%	-1.52%	-1.32%
add	0.38%	1.60%	-0.09%
triad	0.12%	-0.09%	-0.06%

VM, 8 vCPUs	No HT	Patch applied	Core-sched
copy	-1.17%	-0.93%	-3.15%
scale	1.24%	0.78%	0.89%
add	1.64%	1.84%	1.82%
triad	-0.12%	0.29%	0.33%

… still core scheduling does no harm under baremetal and VM normal load, and helps a bit under overcommit

VM, 4 vCPUs	No HT	Patch applied	
copy	0.76%	0.37%	-1.03%
scale	2.40%	2.32%	0.47%
add	1.12%	0.03%	-1.58%
triad	0.23%	-0.03%	-0.66%

			-sched
copy	-26.18%	-9.26%	-14.31%
scale	-31.55%	-15.72%	-17.07%
add	-29.29%	-19.45%	-20.46%
triad	-26.69%	-21.74%	-20.33%

Core Scheduling Performance: mutilate

Baremetal	No HT	Patch applied	Core-sched
1 thread	-0.32%	1.04%	-6.41%
3 threads	-12.43%	0.74%	-8.54%
5 threads	-8.53%	-1.12%	-18.16%
8 threads	21.22%	1.67%	-12.74%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 thread	25.50%	27.51%	26.19%
3 threads	-38.95%	8.47%	9.98%
5 threads	-87.35%	3.10%	-0.92%
8 threads	-66.04%	1.66%	-1.84%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 thread	22.06%	22.30%	22.05%
5 thread	14.89%	15.54%	17.14%
4 threads	15.50%	23.72%	24.26%
-			

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 thread	-33.43%	-26.57%	-23.54%
3 threads	-93.59%	-54.19%	-58.99%
5 threads	-97.21%	-82.43%	-81.25%
8 threads	-86.24%	-61.20%	-61.65%

Core Scheduling Performance: mutilate

Baremetal	No HT	Patch applied	Core-
1 thread	-0.32%	1.04%	-6.41%
3 threads	-12.43%	0.74%	-8.54%
5 threads	-8.53%	-1.12%	-18.16%
8 threads	21.22%	1.67%	-12.74%

Baremetal is regressing

	HT	Patch applied	Core-sched
1 thread	25.50%	27.51%	26.19%
3 threads	-38.95%	8.47%	9.98%
5 threads	-87.35%	3.10%	-0.92%
8 threads	-66.04%	1.66%	-1.84%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 thread	22.06%	22.30%	22.05%
5 thread	14.89%	15.54%	17.14%
4 threads	15.50%	23.72%	24.26%
-			

Virt, both normal load and overcommitted, improves

2x VM, 8 vCPUs			re-sched
1 thread	-33.43%	-26.57%	-23.54%
3 threads	-93.59%	-54.19%	-58.99%
5 threads	-97.21%	-82.43%	-81.25%
8 threads	-86.24%	-61.20%	-61.65%

Core Scheduling Performance: Kernbench

Baremetal	No HT	Patch applied	Core-sched
-j2	1.60%	0.07%	0.30%
-j4	5.99%	0.39%	-0.31%
-j8	-30.75%	0.62%	-5.16%
-j16	-33.59%	-0.32%	-6.04%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
-j2	11.91%	10.59%	10.32%
-j4	3.11%	8.95%	7.83%
-j8	-35.63%	2.07%	-1.55%
-j16	-33.52%	0.68%	-2.17%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
-j2	10.85%	11.42%	10.27%
-j4	-3.44%	10.79%	9.93%
-j8	10.32%	10.82%	10.18%
-			

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
-j2	-64.26%	-43.70%	-53.90%
-j4	-127.19%	-48.62%	-64.77%
-j8	-162.31%	-70.14%	-79.98%
-j16	-154.92%	-63.02%	-65.59%

Core Scheduling Per… : Kernbench

Both baremetal and in VM, when reaching saturation, core scheduling is very effective

Baremetal	No HT	Patch applied	Core-sche…
-j2	1.60%	0.07%	0.30%
-j4	5.99%	0.39%	-0.31%
-j8	-30.75%	0.62%	-5.16%
-j16	-33.59%	-0.32%	-6.04%

	…o HT	Patch applied	Core-sched
	11.91%	10.59%	10.32%
	3.11%	8.95%	7.83%
-j…	-35.63%	2.07%	-1.55%
-j16	-33.52%	0.68%	-2.17%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
-j2	10.85%	11.42%	10.27%
-j4	-3.44%	10.79%	9.93%
-j8	10.32%	10.82%	10.18%
-			

Significant overhead, but overcommit results are quite good

2… 8…			
-j2	-64.26%	-43.70%	-53.90%
-j4	-127.19%	-48.62%	-64.77%
-j8	-162.31%	-70.14%	-79.98%
-j16	-154.92%	-63.02%	-65.59%

Core Scheduling Performance: sysbench-cpu

Baremetal	No HT	Patch applied	Core-sched
1 task	-0.03%	0.00%	0.01%
5 tasks	-20.64%	0.00%	-0.24%
7 tasks	-61.24%	0.00%	-0.92%
16 tasks	-81.29%	-0.13%	-2.51%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 task	7.41%	7.45%	7.44%
5 tasks	-17.88%	3.43%	2.94%
7 tasks	-61.72%	0.91%	0.12%
16 tasks	-83.89%	0.04%	-3.28%

VM, 4 vCPUs	No HT	Patch applied	Core-sched
1 task	6.62%	7.43%	7.44%
3 tasks	4.82%	5.40%	5.42%
5 tasks	3.53%	5.35%	5.44%
8 tasks	3.30%	5.33%	5.45%

2x VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 task	-3.67%	-11.70%	-40.93%
5 tasks	-130.24%	-17.96%	-30.65%
7 tasks	-205.34%	-51.79%	-57.70%
16 tasks	-248.65%	-70.56%	-72.76%

Core Scheduling Performance: sysbench-cpu

Baremetal	No HT	Patch applied	Core-sched
1 task	-0.03%	0.00%	0.01%
5 tasks	-20.64%	0.00%	-0.24%
7 tasks	-61.24%	0.00%	-0.92%
16 tasks	-81.29%	-0.13%	-2.51%

1 VM, 8 vCPUs	No HT	Patch applied	Core-sched
1 task	7.41%	7.45%	7.44%
5 tasks	-17.88%	3.43%	2.94%
7 tasks	-61.72%	0.91%	0.12%
16 tasks	-83.89%	0.04%	-3.28%

VM, 4 vCPUs	No HT	Patch	Core-sched
1 task	6.62%		
3 tasks	4.82%	5.40%	5.42%
5 tasks	3.53%	5.35%	5.44%
8 tasks	3.30%	5.33%	5.45%

2x VM	No HT	Patch applied	Core-sched
	-3.67%	-11.70%	-40.93%
5 tasks	-130.24%	-17.96%	-30.65%
7 tasks	-205.34%	-51.79%	-57.70%
16 tasks	-248.65%	-70.56%	-72.76%

Baremetal, VM normal load and VM overcommit all doing great... Am I dreaming or what?!?!

Conclusions

Conclusions

- Core scheduling is necessary, if we want to be able to mitigate some vulnerabilities (which badly affect virtualization, e.g., L1TF)
- Mitigating vulnerabilities is not the only use case for Core Scheduling in Virtualization
- Core Scheduling performs better than disabling HyperThreading in overcommitted scenarios
- Efficiently implementing Core Scheduling in Linux is complex, and the current patches still need some work

Thanks!

Questions?