Stop the Leaks!

Towards Provable Information Security with seL4

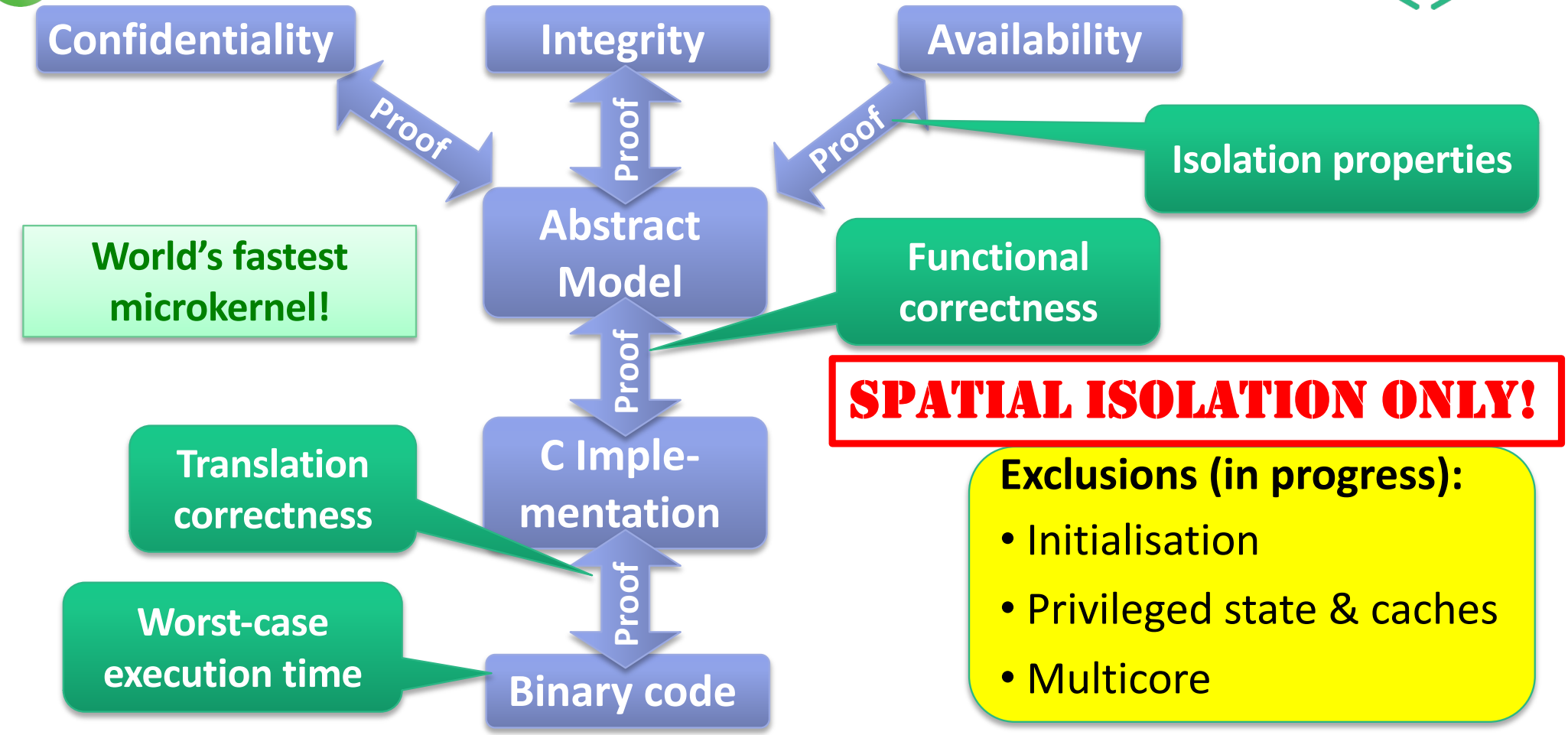
Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser
Trustworthy Systems | Data61

https://trustworthy.systems

CSIRO

Operating-System Security – An C

Provable Security Enforcement



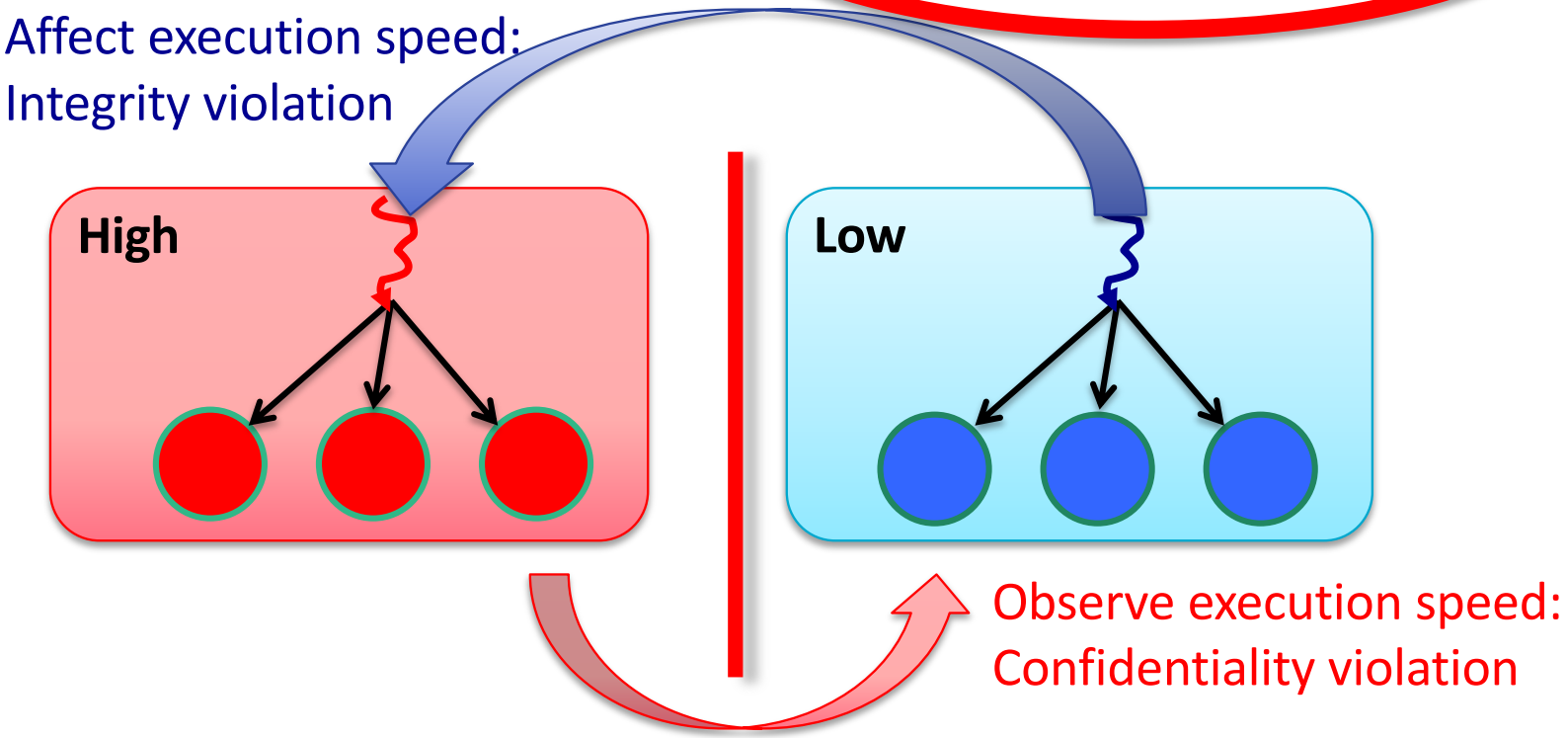
Background: Timing Channels

Safety: Timeliness

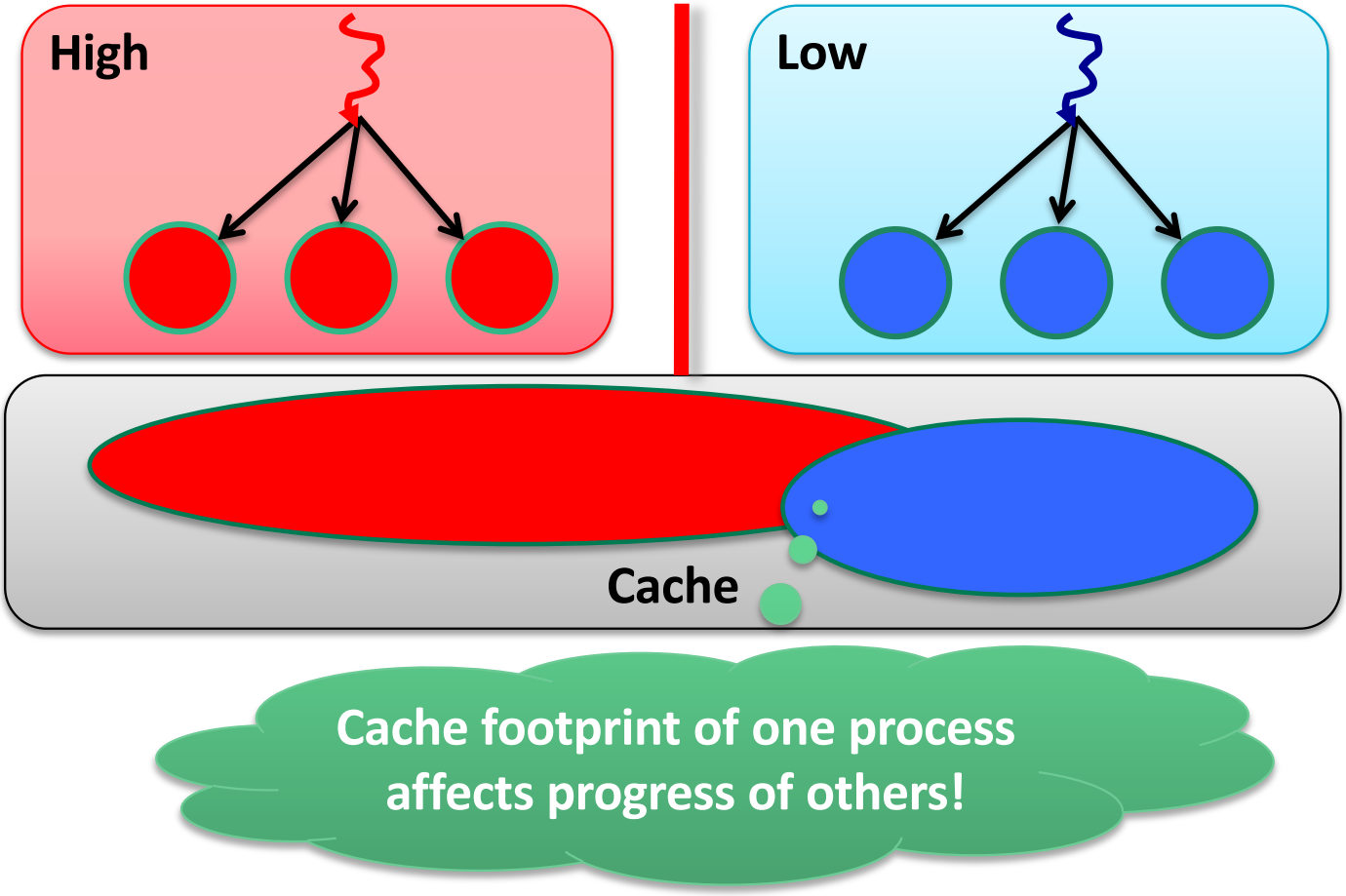
- Execution interference

Security: Confidentiality

- Leakage via timing channels



Timing Channels: Conflicts on Shared HW



Sharing can be:

- Concurrent multicore, HW thread
- Time-shared

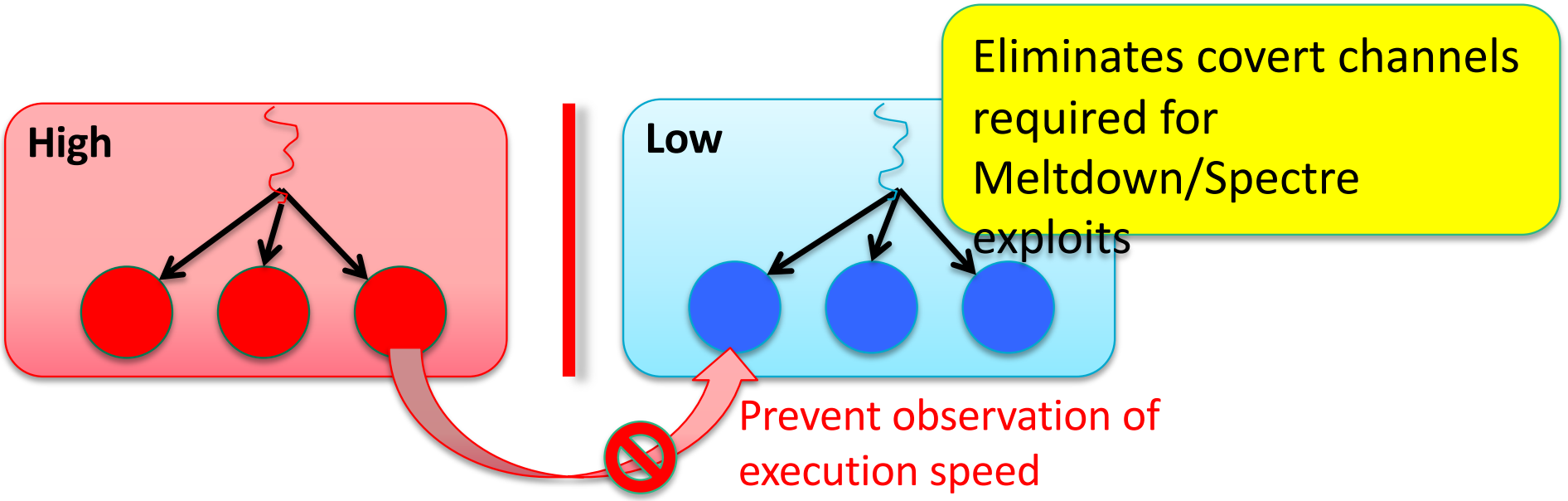
“Caches” include:

- L1 I-, D-cache
- TLB
- Branch predictor
- Instr. prefetcher
- Data prefetcher
- off-core caches & busses

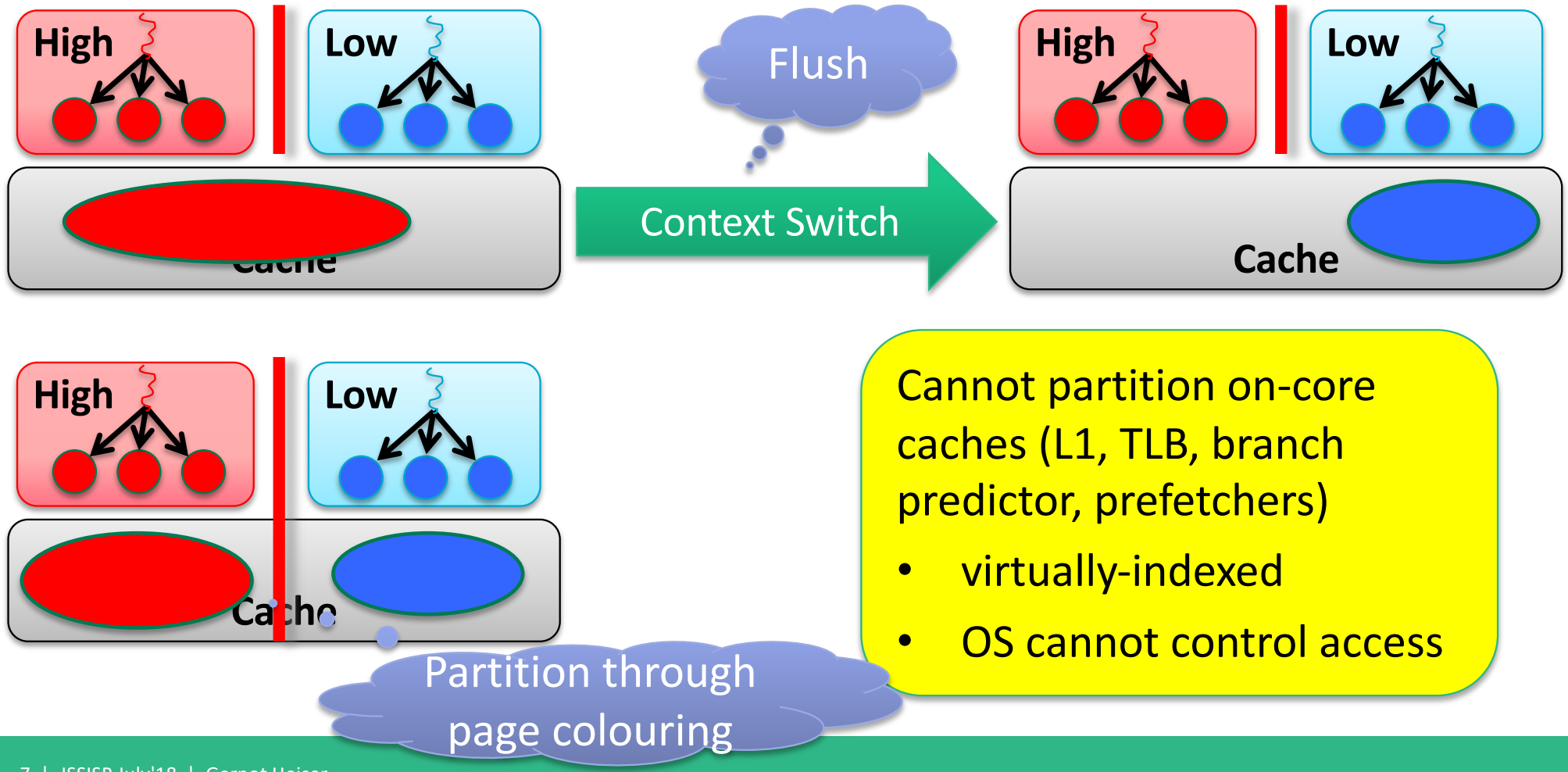
Aim: Black-Box Mitigation

Time protection, just like standard memory protection

- OS-enforced isolation
- No requirement for modifying user code
- High and Low code untrusted – mandatory confinement
- Should also protect against data-dependent execution time



Mitigation: Prevent Sharing of State



Issues

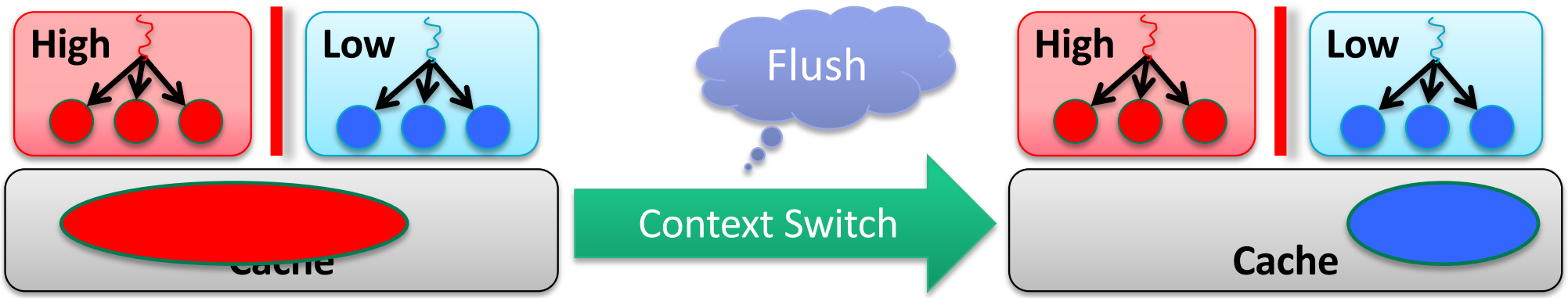
1. **Feasibility**: Can we close all channels?

2. **Efficiency**: Can we do this with bearable cost?

3. **Trustworthiness**: Can we *prove* there are no channels?

Feasibility

Methodology: Intra-Core Channels



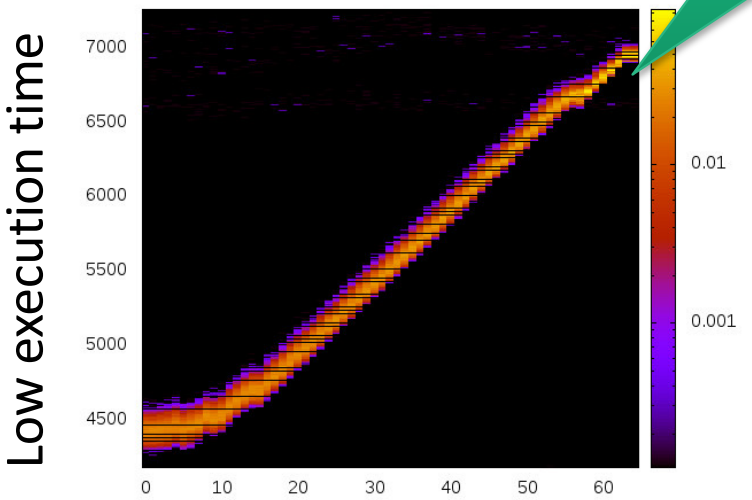
- Disable data prefetcher
- On context switch, perform all architected flush operations:
  - `wbinvd`
  - `invpcid` (Intel-64) or reload CR0, CR3 (IA-32)
- Optionally test (meanwhile revoked) Spectre microcode patch

Channel Analysis

Horizontal variation indicates channel

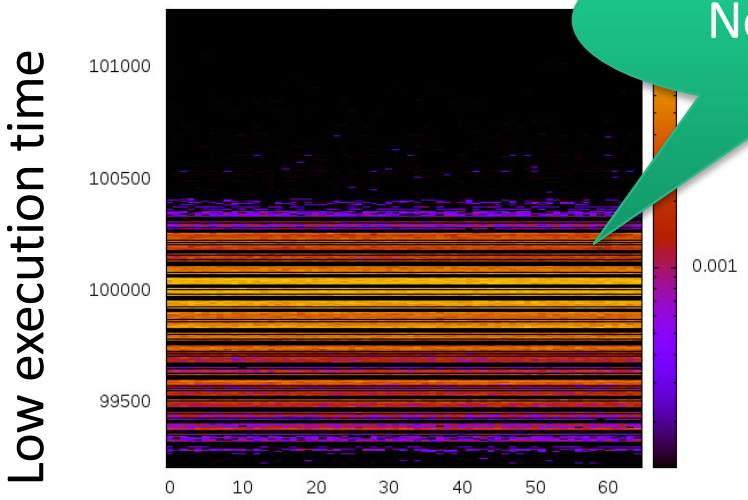
L1 D-cache channel

Intel Sandy Bridge 32-k



Unmitigated

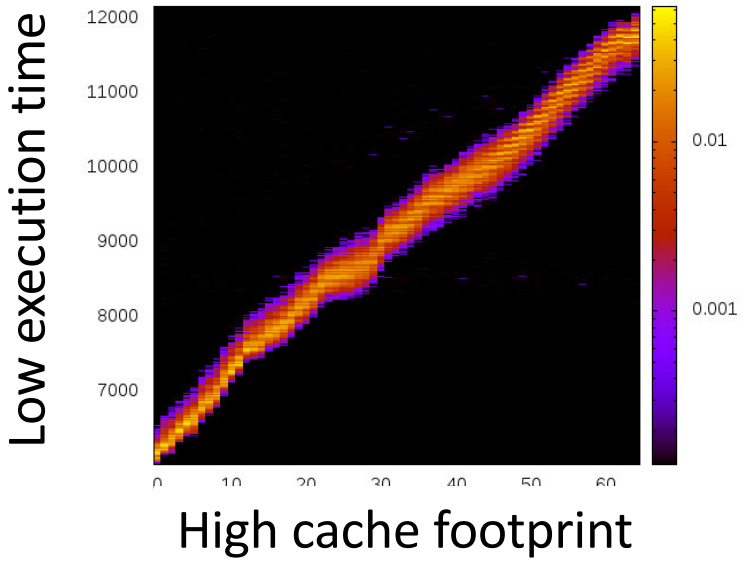
No channel



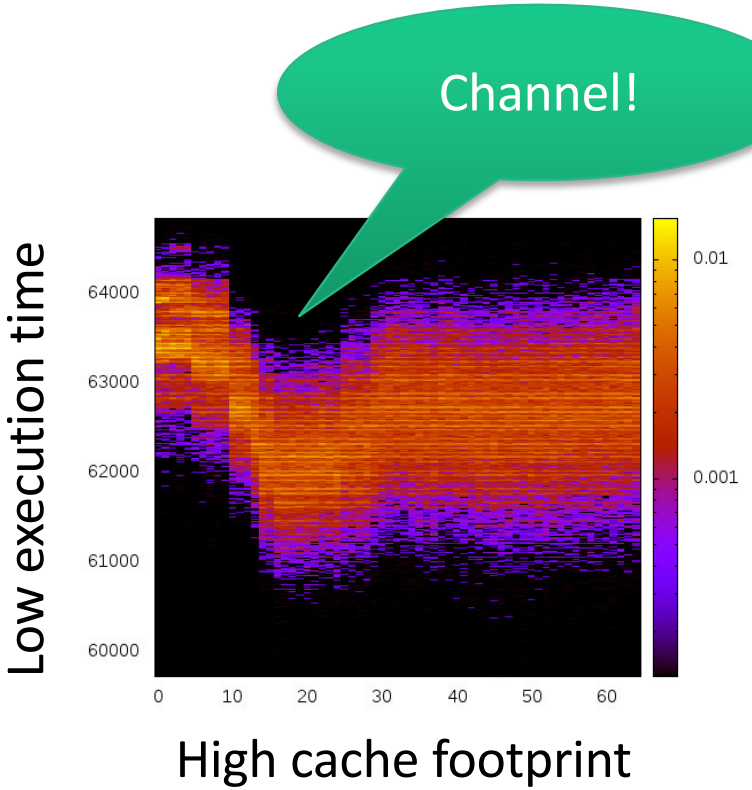
Maximally mitigated

Channel Analysis

L1 I-cache channel
Intel Sandy Bridge 32-bit



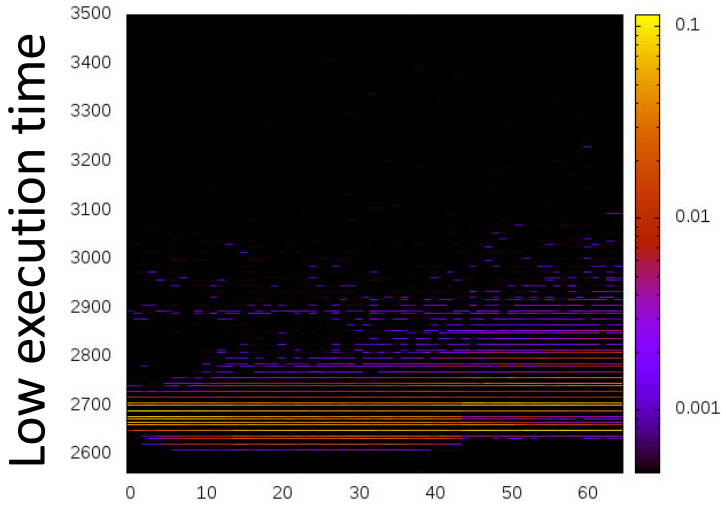
Unmitigated



Maximally mitigated

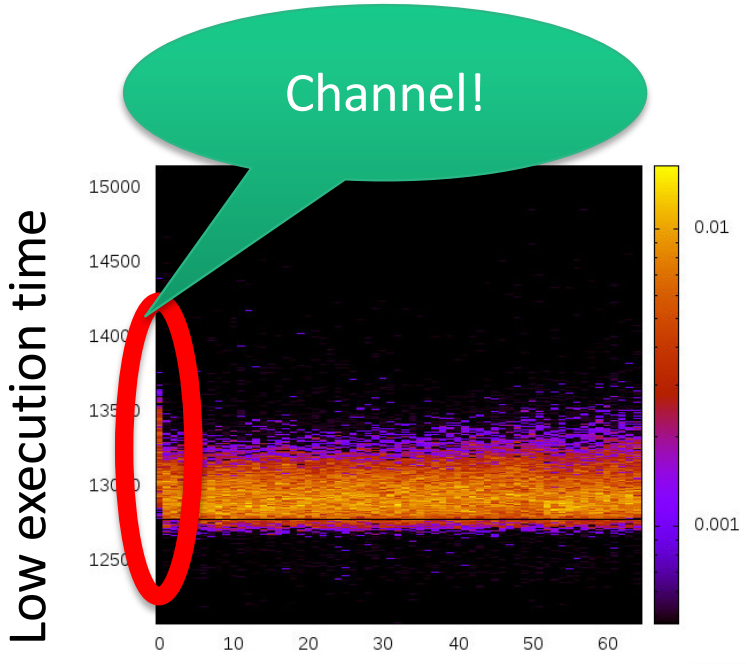
Channel Analysis

L1 I-cache channel
Intel Haswell 64-bit



High cache footprint

Unmitigated



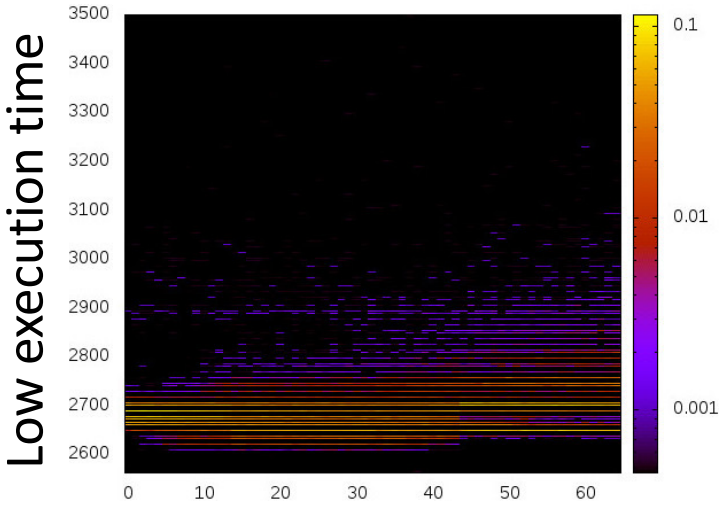
High cache footprint

Maximally mitigated

Reason unclear, suspect instruction prefetcher

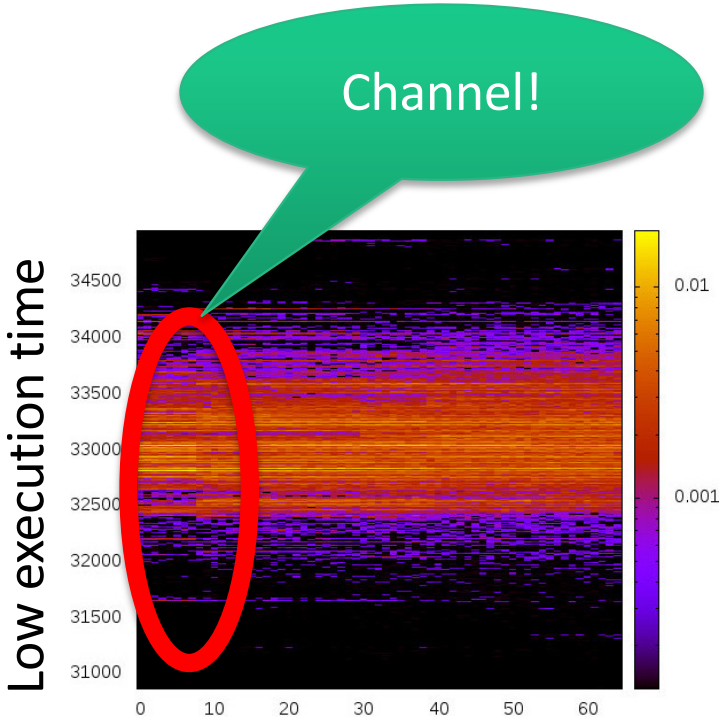
Channel Analysis

L1 I-cache channel
Intel Haswell 64-bit



High cache footprint

Unmitigated

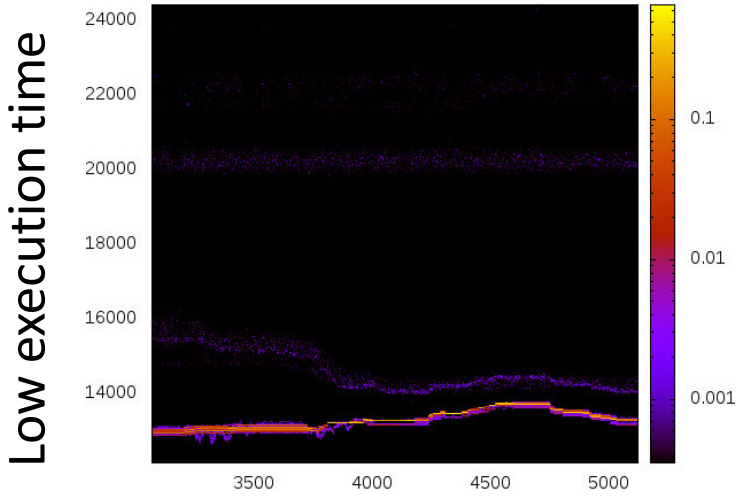


High cache footprint

Maximally mitigated
Including Spectre microcode patch

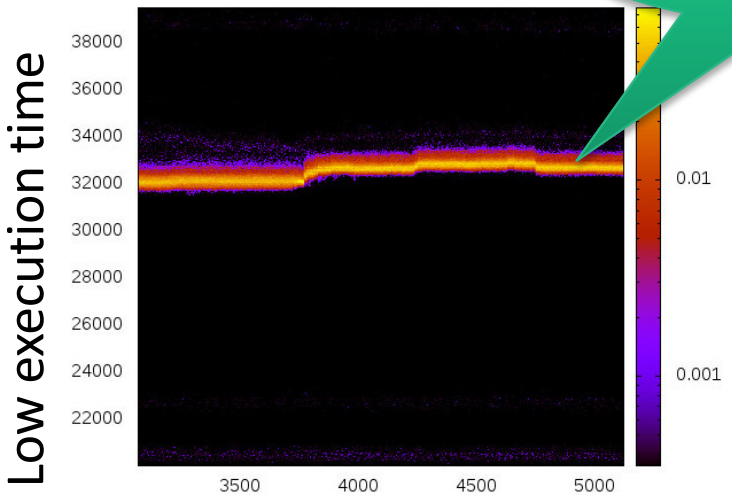
Channel Analysis

Branch target buffer channel
Intel Haswell 64-bit



High cache footprint

Unmitigated



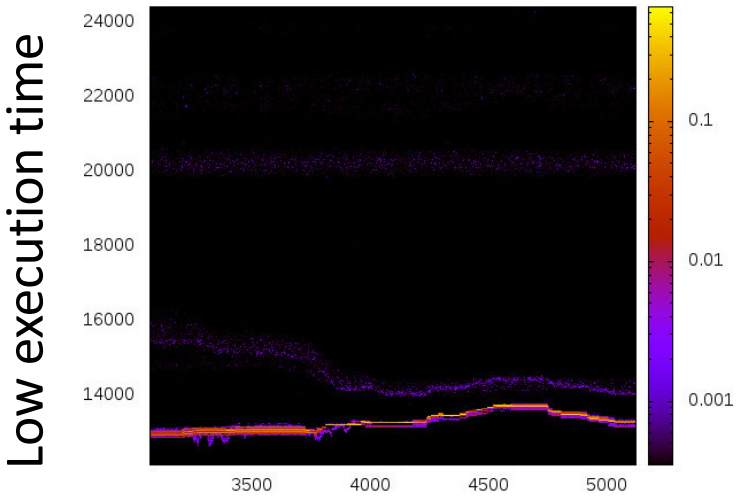
Channel!

High cache footprint

Maximally mitigated

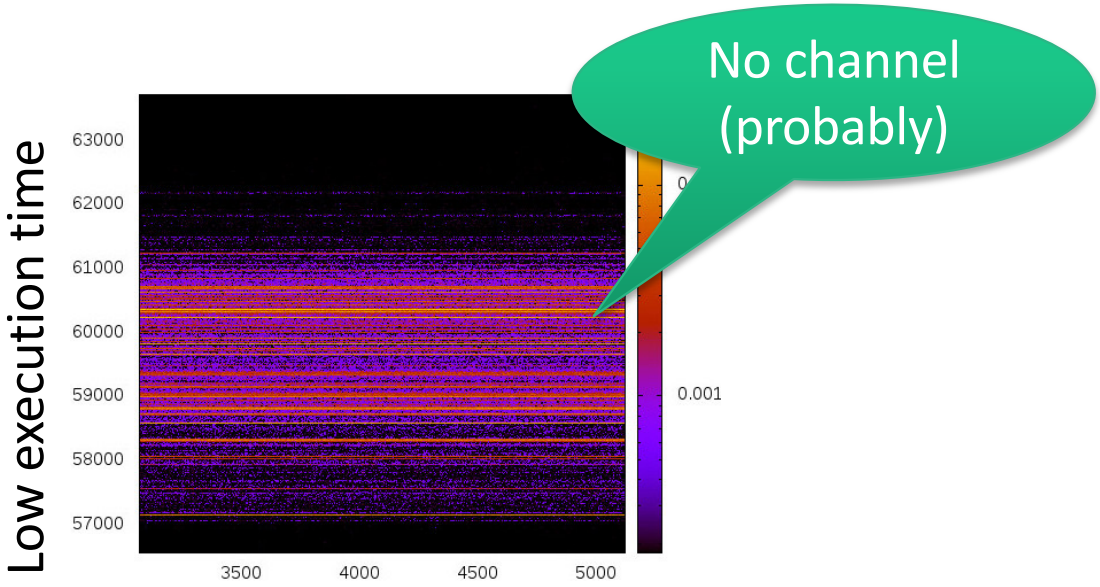
Channel Analysis

Branch target buffer channel
Intel Haswell 64-bit



High cache footprint

Unmitigated

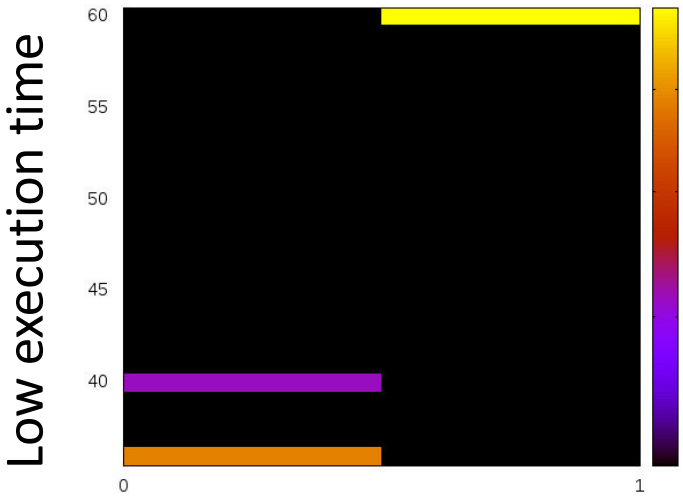


High cache footprint

Maximally mitigated
Including Spectre microcode patch

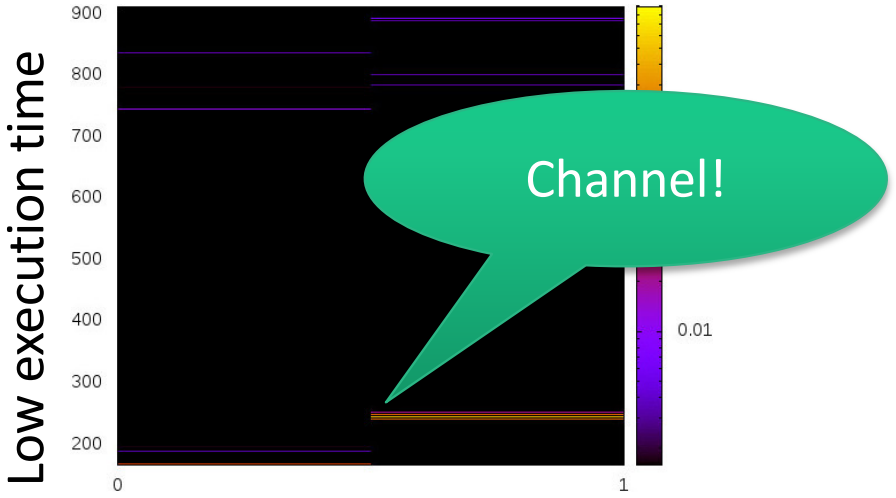
Channel Analysis

Branch history buffer channel
Intel Sandy Bridge 32-bit



High cache footprint

Unmitigated

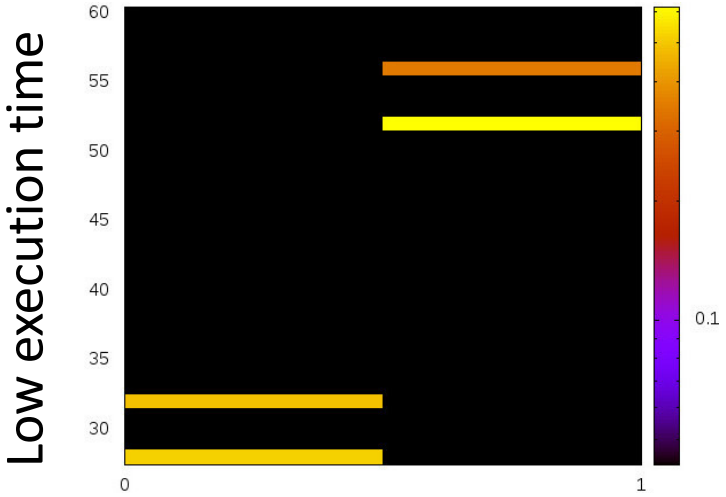


High cache footprint

Maximally mitigated

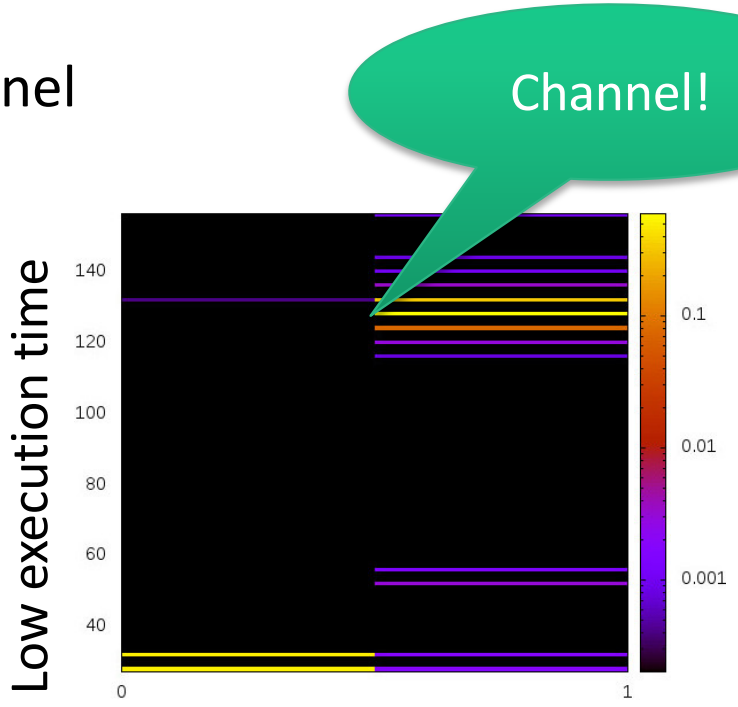
Channel Analysis

Branch history buffer channel
Intel Haswell 64-bit



High cache footprint

Unmitigated

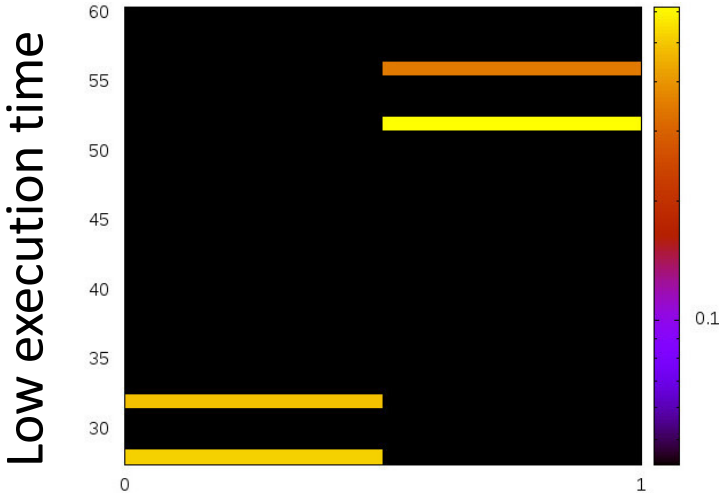


High cache footprint

Maximally mitigated

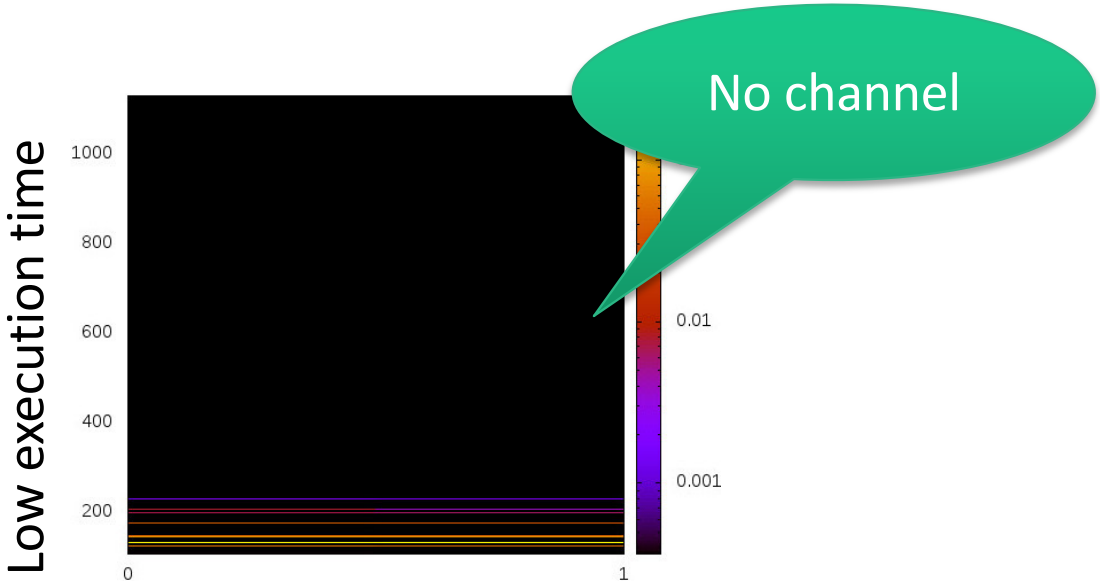
Channel Analysis

Branch history buffer channel
Intel Haswell 64-bit



High cache footprint

Unmitigated

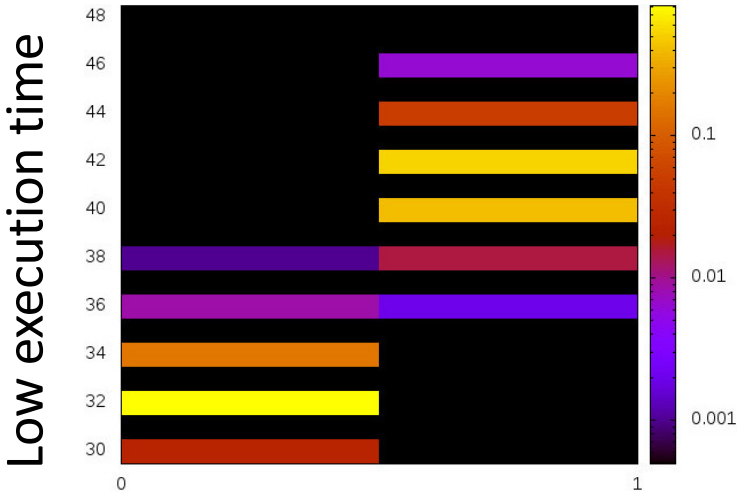


High cache footprint

Maximally mitigated
Including Spectre microcode patch

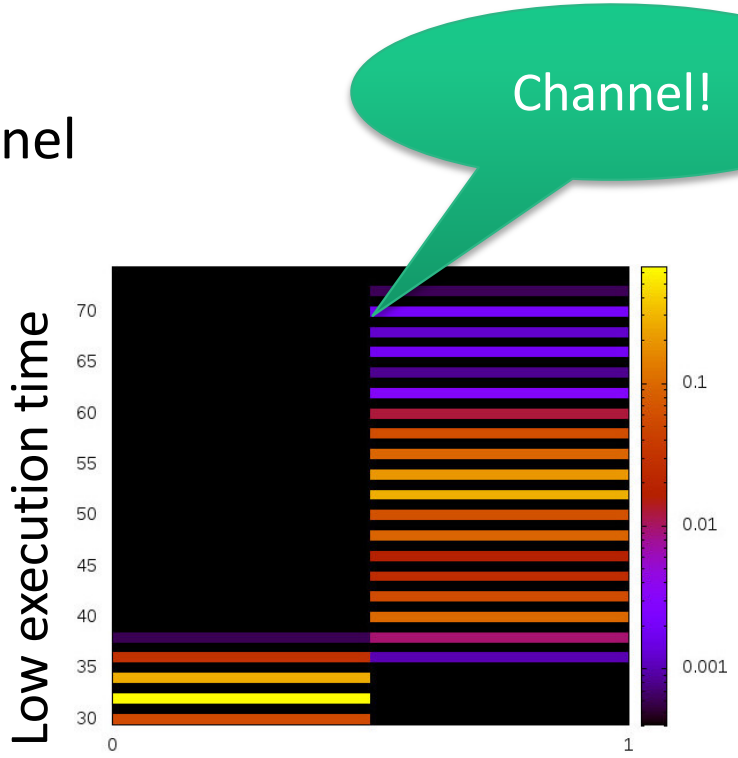
Channel Analysis

Branch history buffer channel
Intel Skylake 64-bit



High cache footprint

Unmitigated

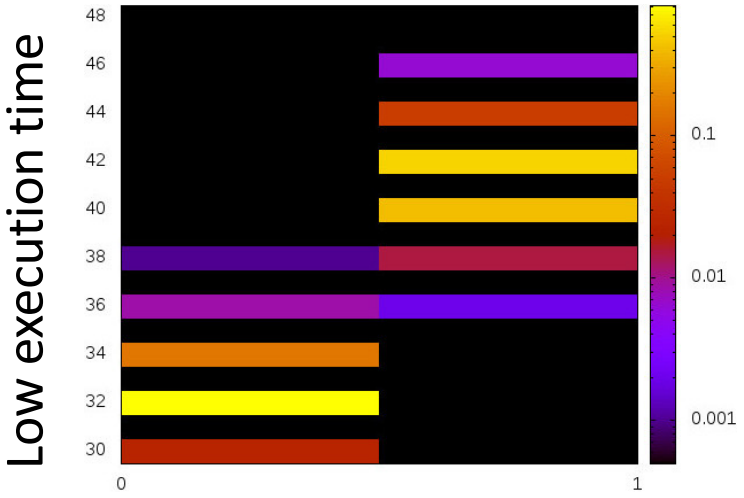


High cache footprint

Maximally mitigated

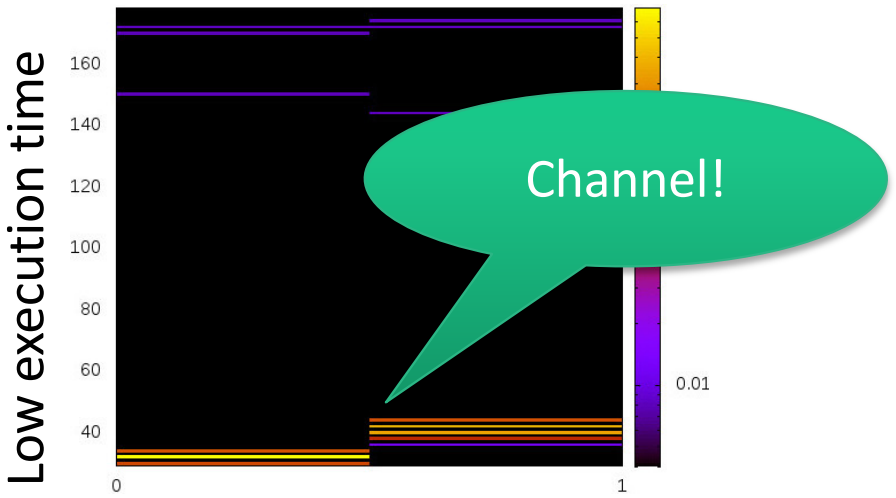
Channel Analysis

Branch history buffer channel
Intel Skylake 64-bit



High cache footprint

Unmitigated



High cache footprint

Maximally mitigated
Including Spectre microcode patch

Result Summary: Channel Capacities

Channel	Sandy Bridge		Haswell		Skylake		ARM A9		ARM A53	
L1 D-cache	4.0	0.04	4.7	0.43	3.3	0.18	5.0	0.11	2.8	0.15
L1 I-cache	3.7	0.85	0.46	0.36	0.37	0.18	4.0	1.0	4.5	0.5
TLB	3.2	0.47	3.2	0.18	2.5	0.11	0.33	0.16	3.4	0.14
BTB	2.0	1.7	4.1	1.6	1.8	1.9	1.1	0.07	1.3	0.64
BHB	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.01	1.0	0.5

Residual channels

Uncloseable channel on each processor studied!

Summary

- ***All*** evaluated processors have un-closable timing channels
- Hardware manufacturers can do something about this
  - Demonstrated by the (partially-effective) Spectre fix
- Need OS-controlled flush of ***all*** microarchitectural on-core state
  - L1 cache
  - TLB
  - BTB
  - BHB
  - prefetchers
  - whatever else they are hiding under the ~~hood~~ ISA

https://ts.data61.csiro.au/projects/TS/timingchannels/arch-mitigation.pml

Efficiency

Efficiency

- **Flushing on-core state** should not be a performance issue
  - no cost when not used
  - direct cost should be round 1 µs for dirty L1-D, far less for everything else
  - indirect cost should be negligible, if used on security-partition switch
    - eg VM switch, 10–100 Hz rate
    - no hot data in cache after other partition's execution
- **Partitioning off-core state** cost should be low
  - replacing dynamic (hardware) by static (software) partitioning
  - less efficient, in average few % performance degradation
  - advantage: performance isolation/predicatbility

Hardware Requirements

New Hardware-Software Contract Needed!

- The ISA is a purely functional contract
  - sufficient to ensure functional correctness
  - abstracts away time
  - insufficient for ensuring either timing safety or security
- For security need an abstraction of microarchitectural state
  - essential for letting OS provide time protection

Timing channels can be closed *iff* all shared hardware state can be

- **Partitioned or**
- **Flushed**

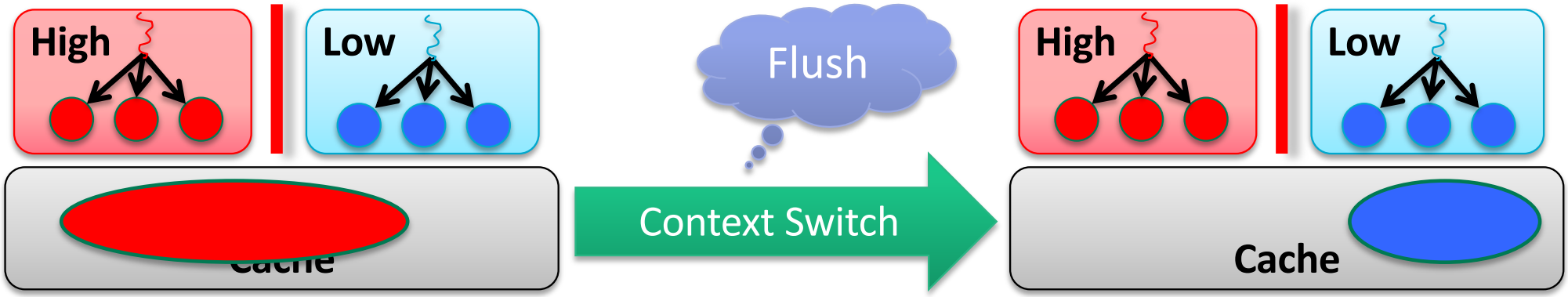
New Hardware-Software Contract: AISA

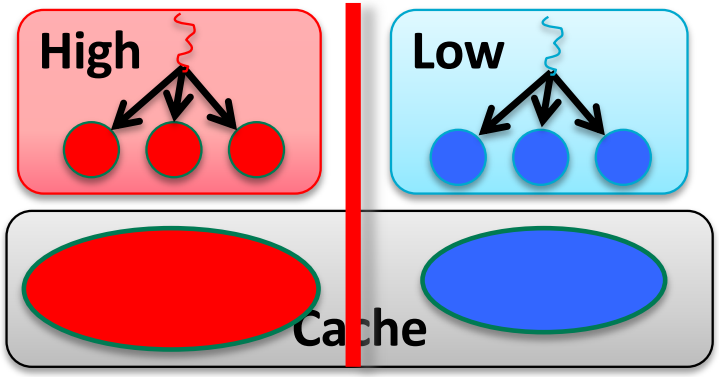
Augmented ISA must provide abstractions that support time protection:

1. Identify partitionable state and how to partition
   - Generally physically-addressed caches, memory interfaces
   - Mostly there, just make it part of the contract
2. Identify existence of non-partitionable state and how it can be flushed
   - Can probably lump all on-core state into single abstraction
   - A single flush-on-core-state operation may be sufficient

OS-Level Time Protection

Recall: Mitigation Approaches

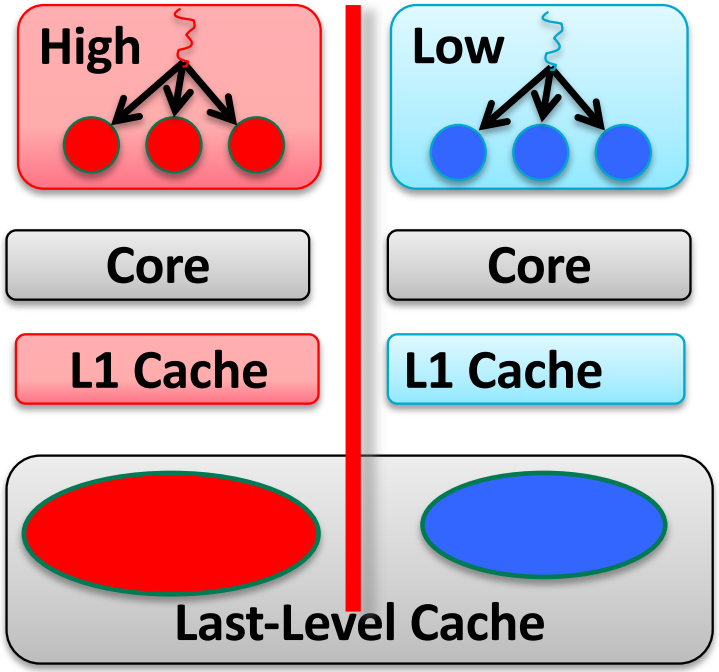




Cannot partition on-core caches (L1, TLB, branch predictor, prefetchers)

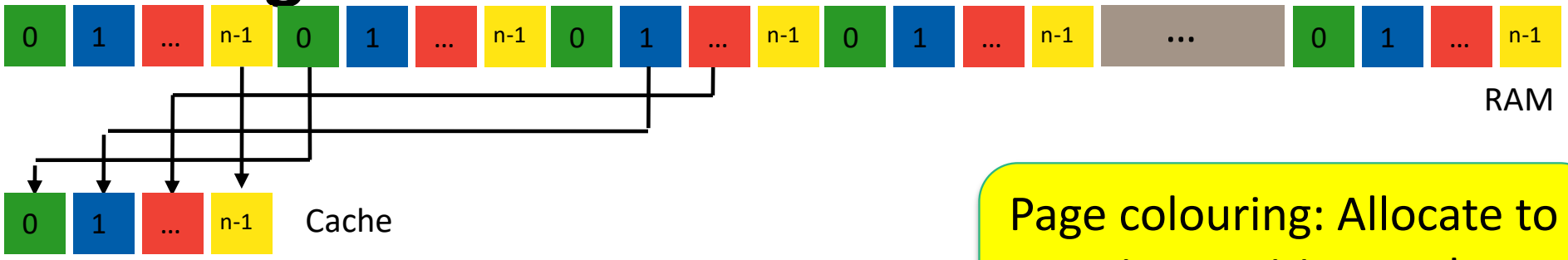
- virtually-indexed
- OS cannot control access

Concurrently Shared Hardware



Flushing shared cache doesn't help, must partition

Partition Cache Through Memory Colouring

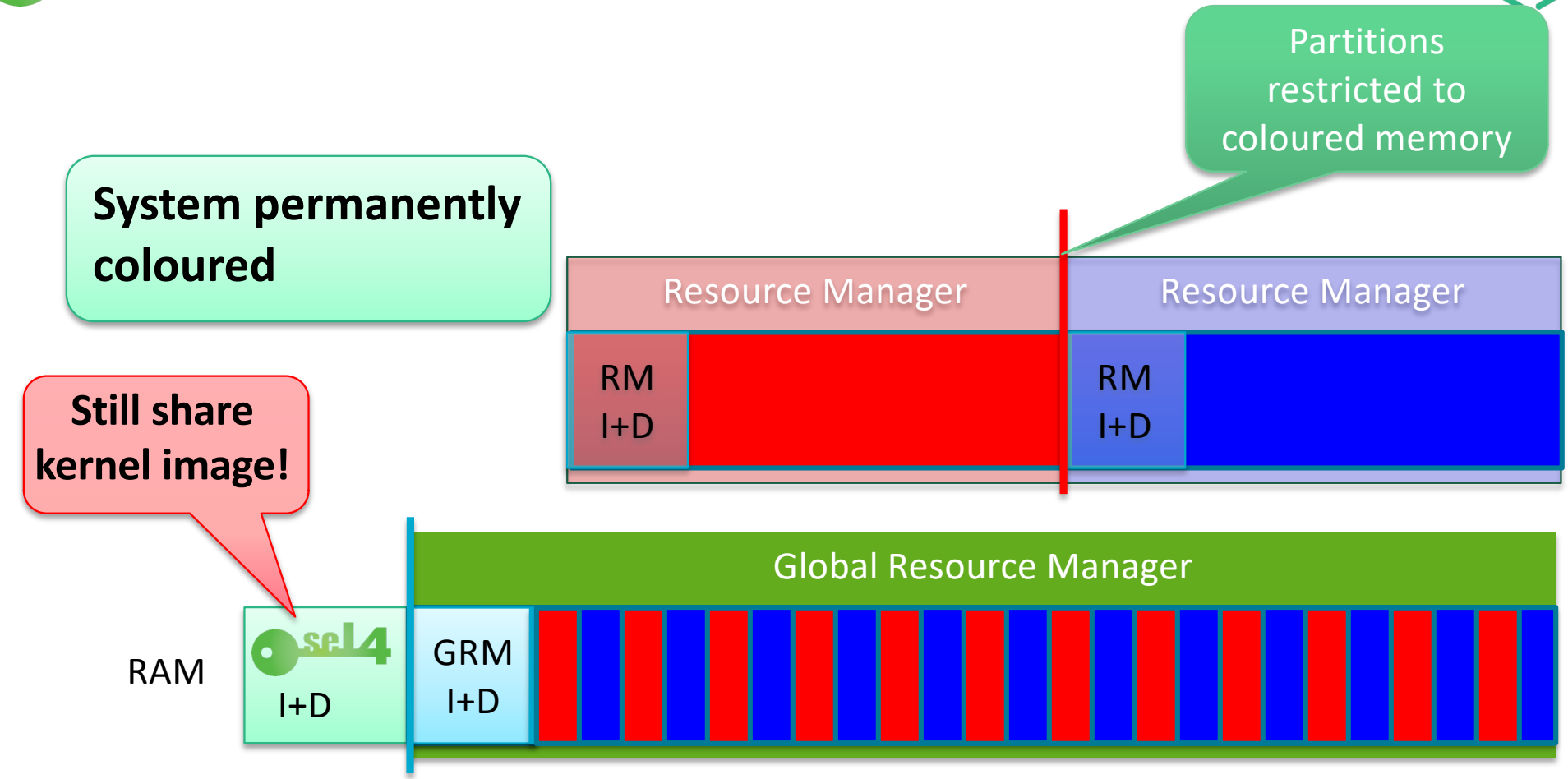


Page colouring: Allocate to security partitions only memory of disjoint colours

Exploit associative cache lookup:

- Particular address maps to specific cache subset, called *cache colour*
- # colours = cache size / (page size * associativity)

Colouring User Memory Is Easy

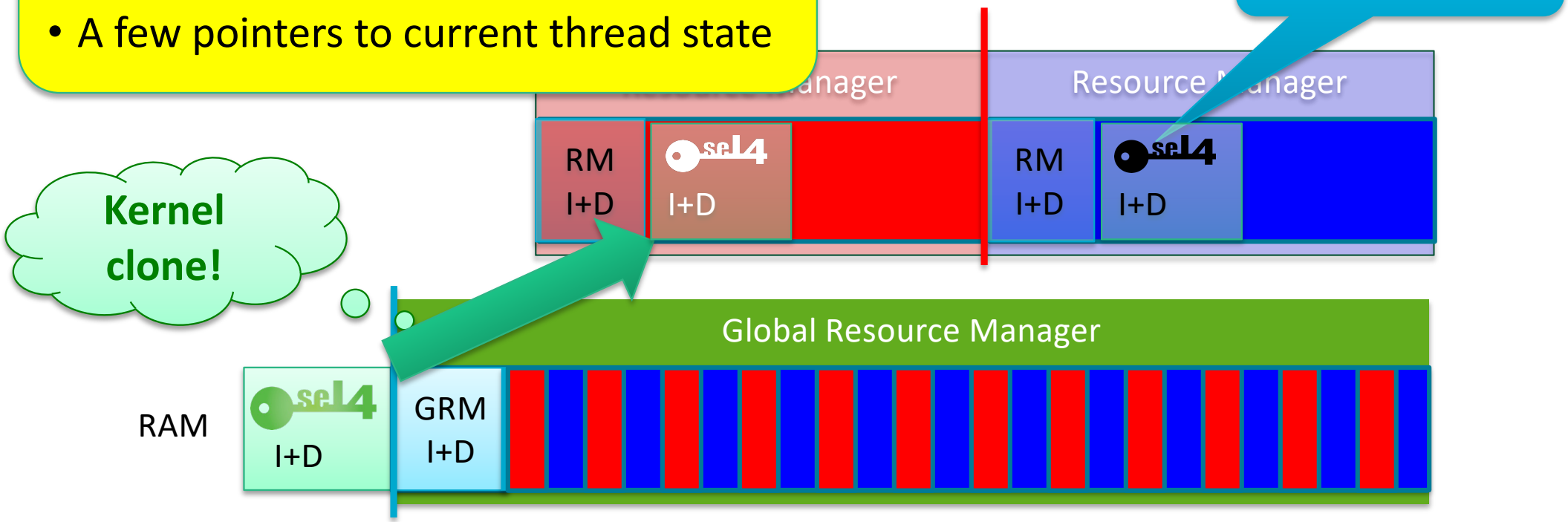


Colouring the Kernel

Remaining shared kernel data:

- Scheduler queue array & bitmap
- A few pointers to current thread state

Each partition has own kernel image



Flushing on Context Switch

Domain switch:

1. T_0 = current_time()
2. Switch context
3. Flush caches
4. Touch all code/data needed for return
5. Reprogram timer
6. while (T_0+WCET < current_time()) ;
7. return

Latency depends on prior execution!

Ensure deterministic timing

Remove dependency

Can We Verify Time Protection?

What Needs To Be Proved?

1. Correct treatment of partitionable state:
   - Need hardware model that identifies all such state (i.e. AISA)
   - Enables *functional correctness* argument:
     No two domains can access the same physical state

Transforms timing channels into storage channels!

2. Correct flushing of non-partitionable state
   - Not trivial: eg proving all cleanup code/data are forced into cache after flush
     – Needs an actual cache model
   - Even trickier: need to prove padding is correct
     – … without explicitly reasoning about time!

How Can We Prove Time Padding?

- Idea: Minimal formalisation of hardware clocks
  - Monotonically-increasing counter
  - Can add constants to time values
  - Can compare time values

To prove: padding loop terminates as soon as timer value ≥ T_0+WCET

Functional property

Summary

- Time protection is *doable*
  - **But** requires manufacturers to adhere to more detailed HW/SW contract
  - They seem to have started listening
- Time protection seems *provable*
  - Core insight: Explicitly (& abstractly) represent state exploited by channels
  - Converts timing into storage channels
  - Reduces to functional correctness argument

Thank You

Gernot Heiser | gernot.heiser@data61.csiro.au | @GernotHeiser

https://trustworthy.systems

CSIRO