PhD defense? Why not a PhD attack?

A starting point for those wishing to ask the many questions left unanswered by Alyssa's thesis.

General topics

Does anyone use these vulnerabilities in real attacks?
How are these vulnerabilities relevant for exploit chains?
What is the impact of AI in this research area?
What exactly is process isolation and where is it used?
Isn't all data potentially sensitive and worth protecting?
Should we just disable speculation entirely?
Can we make a fully secure side-channel resistant processor?
Does it help to migrate to newer architectures such as RISC-V?
Aren't unintended information flows stemming from how *software* is implemented also side-channels?
Aren't 'new' side channels just variants on known ones?
What is the future of side-channel attacks, e.g., 10 years from now?
Aren't side channel attacks a solved problem?
. . . okay, but isn't Spectre a solved problem?
. . . okay, but isn't Spectre variant 1 a solved problem?
Why not just rewrite all software in a modern language like Rust[1]?
How has the research area changed since you started this research?
How could formal verification be used to solve these problems?
What would be good future topics for other researchers in this area?
Is it written as 'side channel' or 'side-channel'?

Fault Correlation Analysis

Can you distinguish mute vs corruption without output (§2.4.3)?
Can FCA be used despite countermeasures such as masking?
§2.6 claims FCA works despite random delays – did you verify this?
Could you use FCA to perform second-order attacks?
What does 'squaring the samples' mean in §2.6?
How many fault attempts are required per trace point?
How practical is the assumption made about repeatability of encryption? (i.e., no variation between runs)
How long did it take to gather traces? Why so long?
Why do you need to apply the 'bucketing'?
Is the Hamming weight leakage model appropriate for your targets?
Why develop a custom glitching hardware platform?
How were the post-processing parameters chosen?
Why didn't you get all the key bytes from the HW engine?
Could FCA be used against SoCs, or only MCUs?
How will safer languages such as Rust[1] make this paper irrelevant?
Isn't this impractical? Why not just use an oscilloscope?

RIDL

Can you really control which data is in-flight in the pipeline? If not, isn't this just a statistical attack?
What do all the columns in Table 1 refer to?
Isn't the kernel attack from §3.6.4 irrelevant, since all modern OSes use SMAP? Can it be done despite SMAP?
Why didn't you implement an end-to-end JavaScript attack in a real web browser rather than only in SpiderMonkey?
How does the *mask-sub-rotate* technique work exactly?
Why is the covert channel so low-bandwidth?
Is this research relevant for other CPU vendors?
You spent a significant amount of effort reverse-engineering the behavior of Intel's μarch; is this a good use of researcher time?
How do these issues differ from Meltdown?
Does this apply to modern software, such as code written in Rust[1]?
Aren't both RIDL and CrossTalk just yet more instances of the widely-documented security concerns about shared resources?
Isn't this just a rehash of the ZombieLoad paper?

CrossTalk

Are performance counters the best way to analyze processor behavior and potentially find similar issues?
. . . what other methods could be applied?
Why didn't you cover more instructions? Or more contexts?
Are cross-core attacks actually practical? Or relevant?
Aren't cache (e.g., LLC) attacks already cross-core?
Isn't the covert channel impractical? Why bother?
Isn't it overly simplistic to summarize the attack as just 'invert the ECDSA equation and solve for the private key'?
Does mixing private key bits (or message bits) into the ECDSA nonce fully mitigate the attack?
Can you perform these attacks without SGX?
Isn't the impact limited to the exposure of RNG output?
Should we be avoiding using RDRAND and RDSEED?
If the RNG issue doesn't affect server parts, is it relevant?
Can we mitigate similar issues by rewriting RNG code in Rust[1]?
Does this work really matter, given the issue was mitigated?

Type-based Data Isolation

What happens when there are types inside types?
How much attack surface is left with same-type overflows?
Is there a way to verify that all arithmetic is instrumented?
If the mitigation isn't 100% complete/sound, can't it be bypassed?
. . . doesn't that make the mitigation useless?
Why is memory overhead so high?
Did you cherry-pick your benchmarks? What about *X*?
Why include CPU2000 results? Isn't it irrelevant?
Why not include [other benchmark]? Isn't it critical?
Why measure SPECspeed and not SPECrate? Why not use OpenMP?
Isn't all pointer arithmetic done with GetElementPtr?
Couldn't this be done using points-to analysis?
Doesn't the 4GB limit for individual allocations make this impractical?
Why can't SCEV be used to determine bounds in code paths which may be executed speculatively? Can we overcome these obstacles?
Why isn't this upstreamed? Do you plan to upstream it?
Does hardware memory tagging make TDI obsolete?
What are your thoughts about perlbench?
Why not just mitigate these issues by porting software to Rust[1]?
CPU2017 has considerably higher overhead than CPU2006; which is more representative of modern software?
Does this really mitigate any real-world vulnerabilities?

'Meta' topics

Why didn't you choose other papers?
. . . with a focus on μarch attacks and defenses?
. . . with a focus on mitigations?
. . . papers for which you're the first author?
. . . papers in a completely different subject area?
Why are none of your recent papers included in the thesis?
Can you summarize your research, i.e., make an elevator pitch?
Why didn't you consolidate the acknowledgements?
Why are the references so chaotic and inconsistent?
Why didn't you cite paper *X*?
What happened to the line numbers in the listings?
Do memory-safe languages like Rust[1] make this thesis irrelevant?
Did you at least learn to stop trying to use TikZ? ♀♡

.. wait, didn't you finish writing this thesis in May 2021?
What the heck took you so long to defend?

[1]Rust! Rust! Rust! Rust ensures memory safety! Memory safety means no vulnerabilities! No vulnerabilities makes software strong! Rust!