Meltdown und Spectre
Was passiert da wirklich?

Prof. Dr.-Ing. habil. Daniel Lohmann
Fachgebiet System- und Rechnerarchitektur (SRA)
Leibniz Universität Hannover

Das Fachgebiet System- und Rechnerarchitektur

Prof. Dr.-Ing. habil. Daniel Lohmann
lohmann@sra.uni-hannover.de
Fachgebietsleiter

Betriebssysteme

apl. Prof. Dr.-Ing. habil. Jürgen Brehm
brehm@sra.uni-hannover.de

Rechnerarchitektur

Prof. Dr.-Ing. Christian Müller-Schloer
cms@sra.uni-hannover.de

Organic Computing

Das Fachgebiet System- und Rechnerarchitektur

Prof. Dr.-Ing. habil. Daniel Lohmann
lohmann@sra.uni-hannover.de
Fachgebietsleiter

Betriebssysteme

apl. Prof. Dr.-Ing. habil. Jürgen Brehm
brehm@sra.uni-hannover.de

Rechnerarchitektur

Prof. Dr.-Ing. Christian Müller-Schloer
cms@sra.uni-hannover.de

Organic Computing

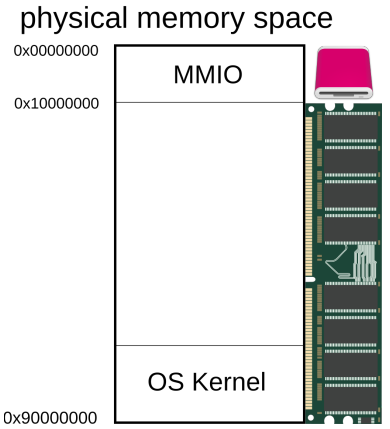
Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Worum geht es? Wissen, was dahinter steckt!

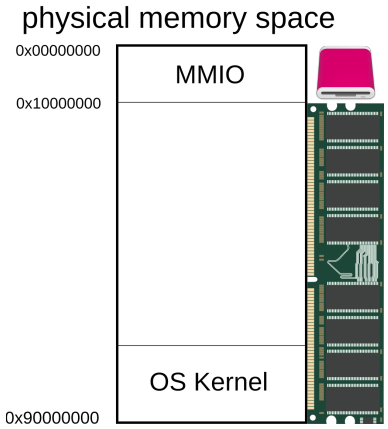
- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Speicherverwaltung 1: Realer (physischer) Adressraum



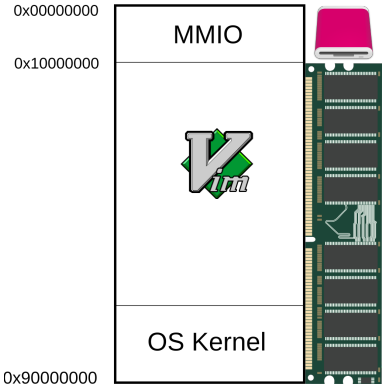
Speicherverwaltung 1: Realer (physischer) Adressraum

- Direkter Zugriff über den Adressbus
  - IO-Geräte
  - ROM (falls vorhanden)
  - **RAM**



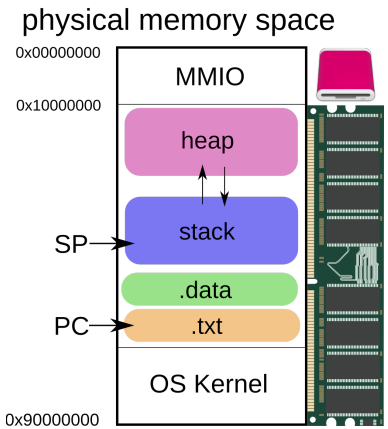
Speicherverwaltung 1: Realer (physischer) Adressraum

- Direkter Zugriff über den Adressbus
  - IO-Geräte
  - ROM (falls vorhanden)
  - **RAM**



Speicherverwaltung 1: Realer (physischer) Adressraum

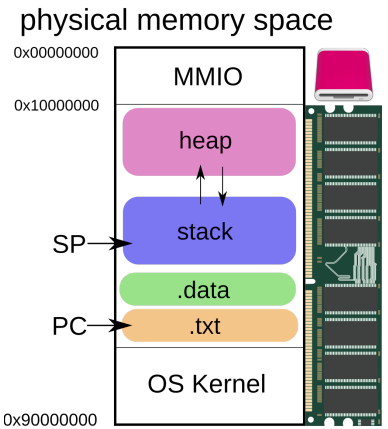
- Direkter Zugriff über den Adressbus
  - IO-Geräte
  - ROM (falls vorhanden)
  - **RAM**



Speicherverwaltung 1: Realer (physischer) Adressraum

- Direkter Zugriff über den Adressbus
  - IO-Geräte
  - ROM (falls vorhanden)
  - **RAM**

- „Rechner-Frühzeit“
  Programme arbeiten direkt
  mit diesen realen Adressen

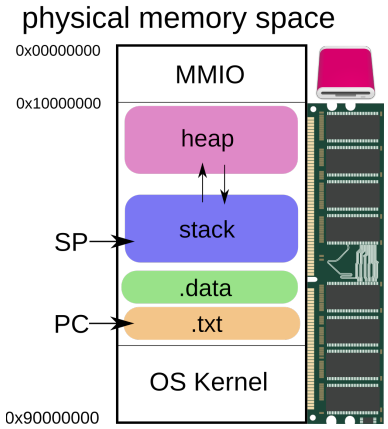


Speicherverwaltung 1: Realer (physischer) Adressraum

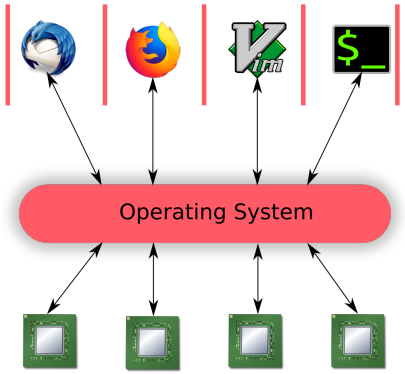
- Direkter Zugriff über den Adressbus
  - IO-Geräte
  - ROM (falls vorhanden)
  - **RAM**

- „Rechner-Frühzeit“
  Programme arbeiten direkt
  mit diesen realen Adressen

- Effizient, aber
  - keine Isolation zwischen Programm und Betriebssystem
  - kein Mehrprogrammbetrieb
  - kein Mehrbenutzerbetrieb

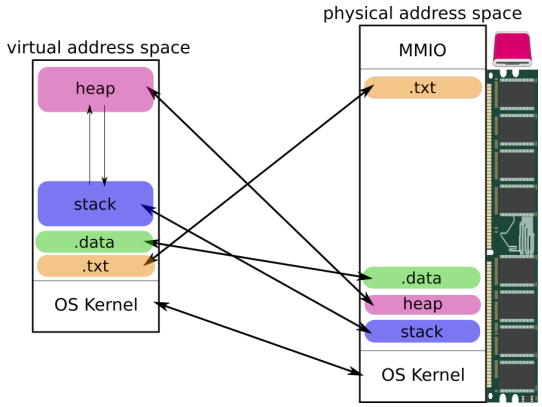


Multiplexing und Isolation durch das Betriebssystem



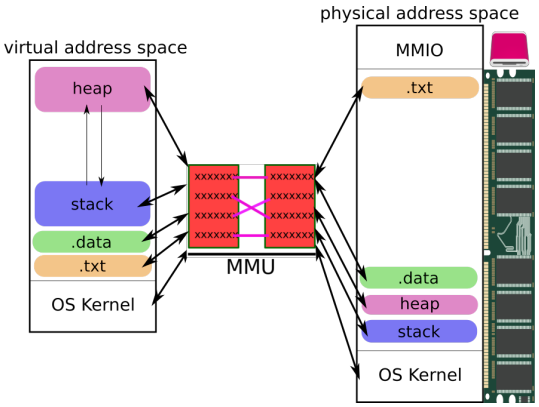
- Mehrprogrammbetrieb erfordert Koordinator –> Betriebssystem
  - Kernaufgabe eines Betriebssystems: **Multiplexing und Isolation**
  - Bereitstellen von „virtuellen Computern“
  - UNIX-Prozess, aber auch VM in einem Hypervisor

Speicherverwaltung 2: Virtueller Adressraum



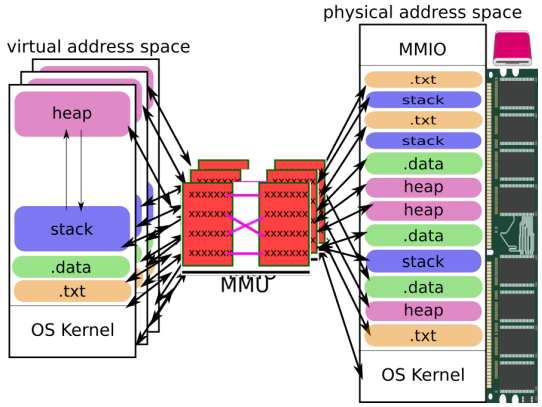
- Virtueller Speicher, durch Betriebssystem und Hardware (MMU) auf realen Speicher abgebildet
- –> Horizontale Isolation durch Einschränkung in der Abbildung

Speicherverwaltung 2: Virtueller Adressraum



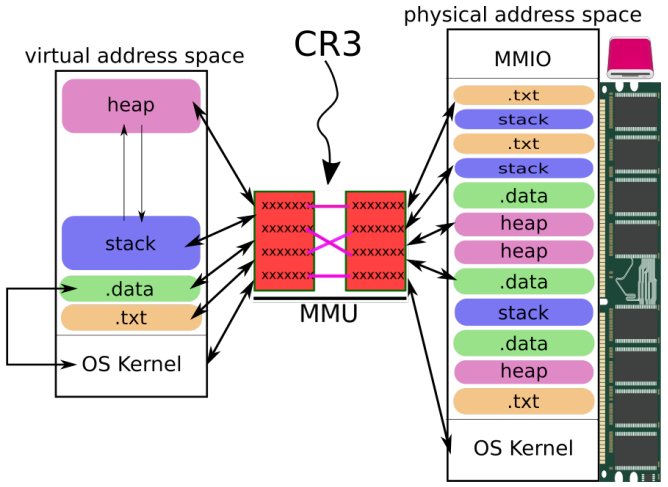
- Virtueller Speicher, durch Betriebssystem und Hardware (MMU) auf realen Speicher abgebildet
- –> Horizontale Isolation durch Einschränkung in der Abbildung

Speicherverwaltung 2: Virtueller Adressraum



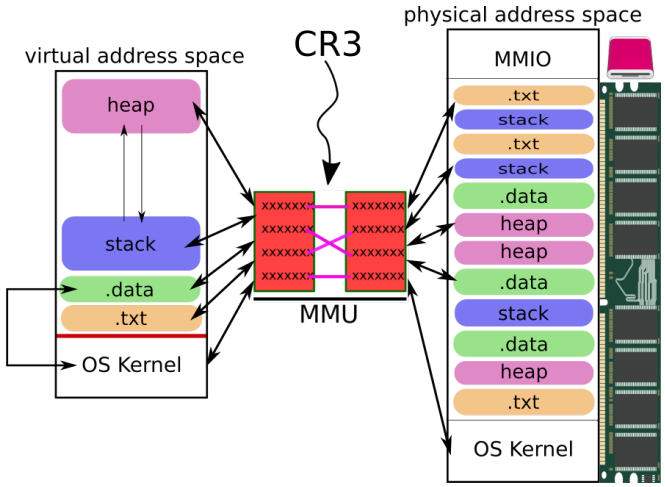
- Virtueller Speicher, durch Betriebssystem und Hardware (MMU) auf realen Speicher abgebildet
- –> Horizontale Isolation durch Einschränkung in der Abbildung

Speicherverwaltung 3: Privilegientrennung



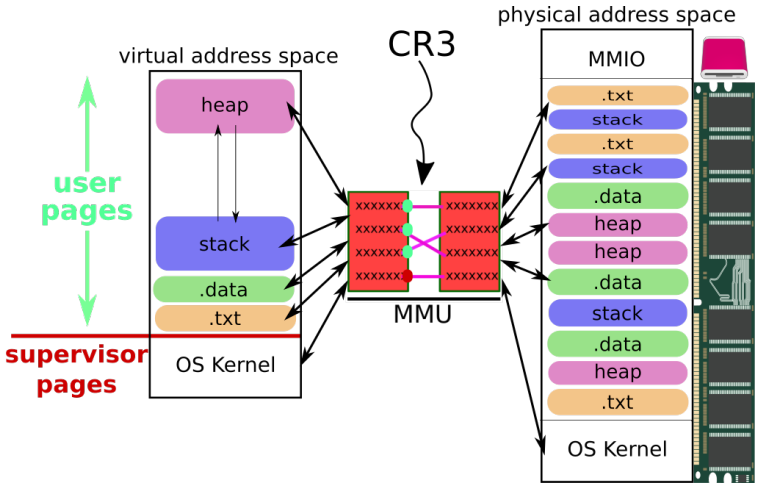
- Einteilung in Benutzer- und Systemmodus *(user vs. supervisor mode)*
- –> Vertikale Isolation durch Privilegebenen

Speicherverwaltung 3: Privilegientrennung



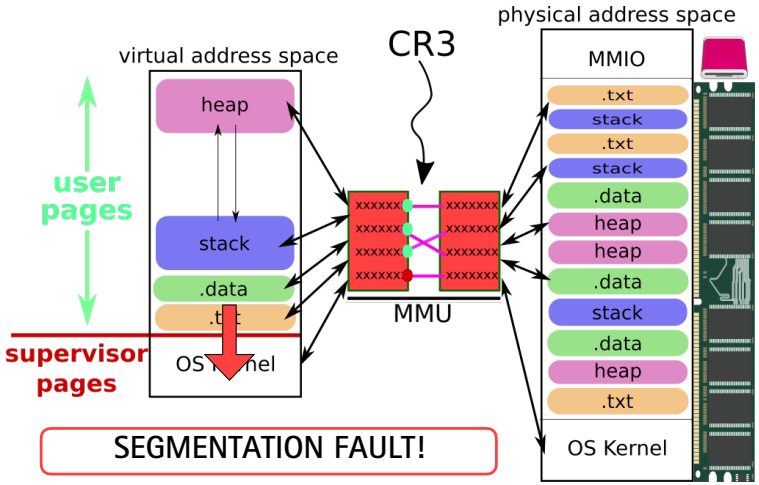
- Einteilung in Benutzer- und Systemmodus *(user vs. supervisor mode)*
- –> Vertikale Isolation durch Privilegebenen

Speicherverwaltung 3: Privilegientrennung



- Einteilung in Benutzer- und Systemmodus *(user vs. supervisor mode)*
- –> Vertikale Isolation durch Privilegebenen

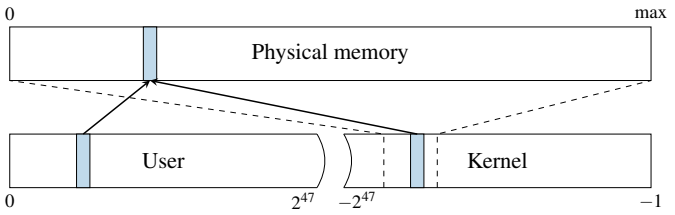
Speicherverwaltung 3: Privilegientrennung



- Einteilung in Benutzer- und Systemmodus *(user vs. supervisor mode)*
- –> Vertikale Isolation durch Privilegebenen

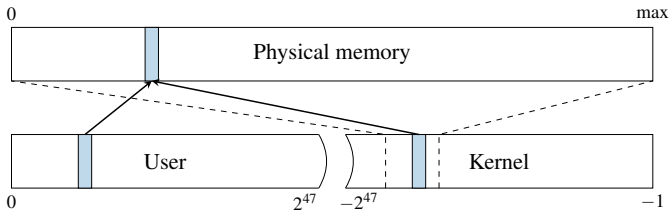
Zusammenfassung: Speicherverwaltung

- Betriebssysteme isolieren Prozesse durch Adressräume
  - Seitenbasierte Abbildung über die MMU auf realen Speicher
  - Zugriff auf „fremden" Speicher konstruktiv nicht möglich
- Kernadressraum wird (üblicherweise) in jeden Adressraum mit eingeblendet
  - ermöglicht effiziente Implementierung von Systemaufrufen
  - Zugriff aus Anwendung wird unterbunden über Supervisor-Bit
  - Versucht man es dennoch: SEGMENTATION FAULT
- Bei 64-Bit Systemen ist der komplette RAM Teil des Kernadressraums!



Zusammenfassung: Speicherverwaltung

- Betriebssysteme isolieren Prozesse durch Adressräume
  - Seitenbasierte Abbildung über die MMU auf realen Speicher
  - Zugriff auf „fremden" Speicher konstruktiv nicht möglich
- Kernadressraum wird (üblicherweise) in jeden Adressraum mit eingeblendet
  - ermöglicht effiziente Implementierung von Systemaufrufen
  - Zugriff aus Anwendung wird unterbunden über Supervisor-Bit
  - Versucht man es dennoch: SEGMENTATION FAULT
- Bei 64-Bit Systemen ist der komplette RAM Teil des Kernadressraums!



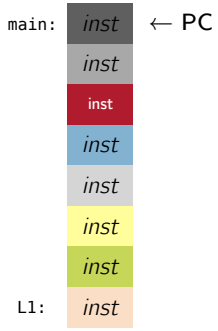
Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

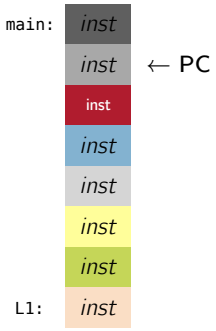
Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

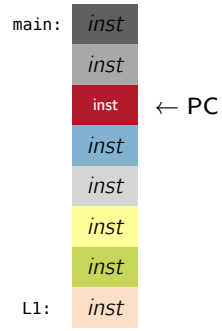
Befehlsabarbeitung eines Prozessors



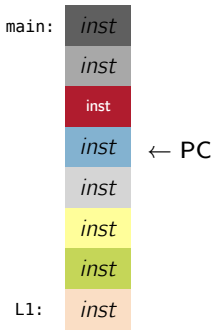
Befehlsabarbeitung eines Prozessors



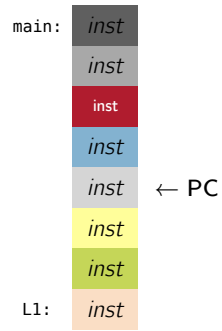
Befehlsabarbeitung eines Prozessors



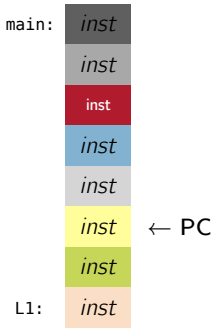
Befehlsabarbeitung eines Prozessors



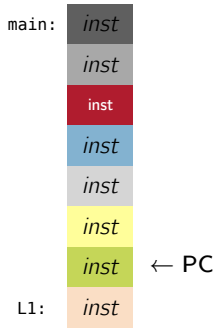
Befehlsabarbeitung eines Prozessors



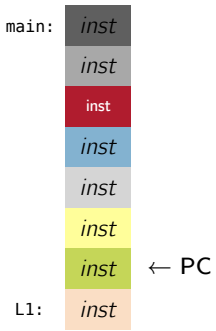
Befehlsabarbeitung eines Prozessors



Befehlsabarbeitung eines Prozessors



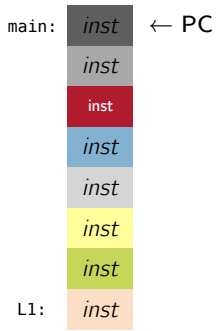
Befehlsabarbeitung eines Prozessors



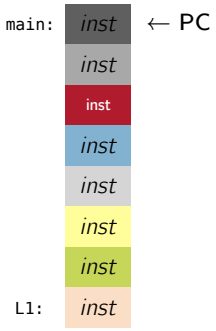
Problem: Befehlsabarbeitung ist aufwändig!

- Befehl laden
- Befehl verstehen
- Operanden laden und Berechnung durchführen
- Ergebnis speichern

Befehlsabarbeitung **mit Pipelining**

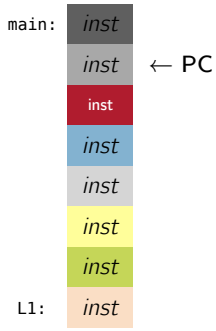


Befehlsabarbeitung **mit Pipelining**



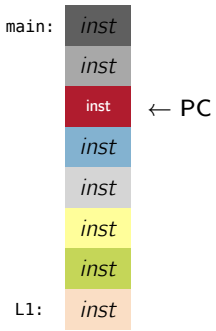
	IF	ID	EX	WB

Befehlsabarbeitung **mit Pipelining**



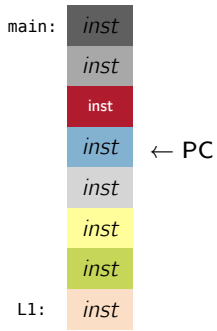
	IF	ID	EX	WB
1	inst			

Befehlsabarbeitung **mit Pipelining**



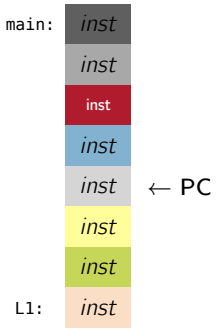
	IF	ID	EX	WB
1	inst			
2	inst	inst		

Befehlsabarbeitung **mit Pipelining**



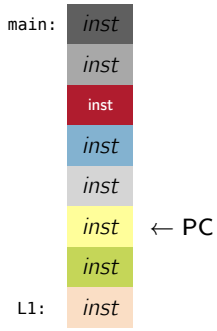
	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	

Befehlsabarbeitung **mit Pipelining**



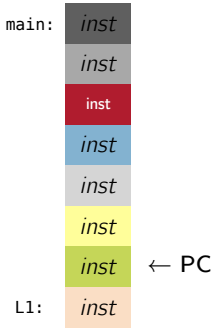
	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	
4	inst	inst	inst	inst

Befehlsabarbeitung **mit Pipelining**



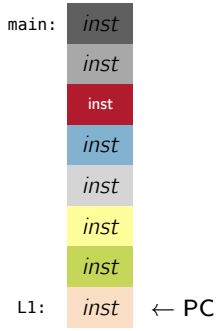
	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	
4	inst	inst	inst	inst
5	inst	inst	inst	inst

Befehlsabarbeitung **mit Pipelining**



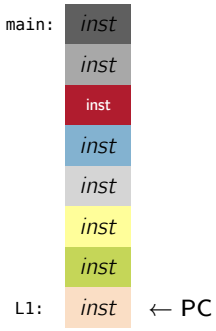
	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	
4	inst	inst	inst	inst
5	inst	inst	inst	inst
6	inst	inst	inst	inst

Befehlsabarbeitung **mit Pipelining**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	
4	inst	inst	inst	inst
5	inst	inst	inst	inst
6	inst	inst	inst	inst
7	inst	inst	inst	inst

Befehlsabarbeitung **mit Pipelining**



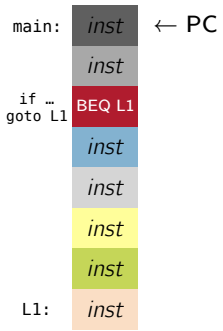
	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	inst	inst	inst	
4	inst	inst	inst	inst
5	inst	inst	inst	inst
6	inst	inst	inst	inst
7	inst	inst	inst	inst

Ergebnis: Höherer Befehlsdurchsatz!

- Im Idealfall immer mehrere Instruktionen gleichzeitig in Arbeit

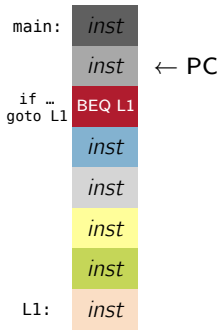
Befehlsabarbeitung **mit Pipelining** Hazard!

Befehlsabarbeitung **mit Pipelining** Hazard!



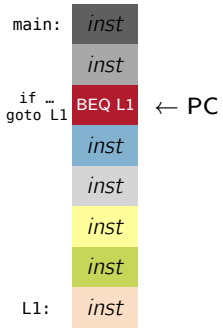
	IF	ID	EX	WB

Befehlsabarbeitung **mit Pipelining** Hazard!



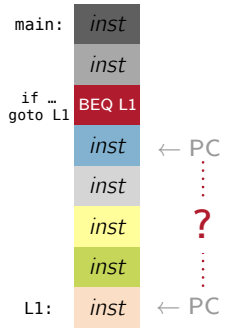
	IF	ID	EX	WB
1	inst			

Befehlsabarbeitung **mit Pipelining** **Hazard!**



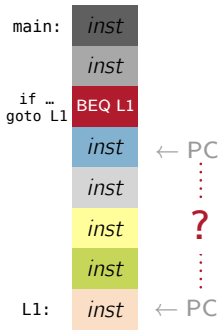
	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		

Befehlsabarbeitung **mit Pipelining** **Hazard!**



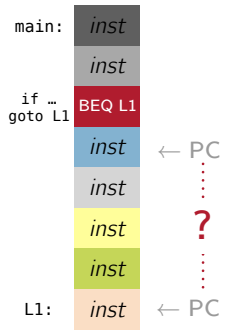
	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	

Befehlsabarbeitung **mit Pipelining** **Hazard!**



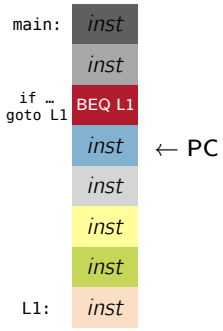
	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*

Befehlsabarbeitung **mit Pipelining** **Hazard!**



	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*

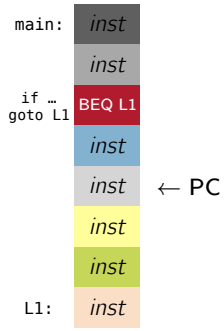
Befehlsabarbeitung **mit Pipelining** **Hazard!**



← PC

	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*
6	*o*	*o*	*o*	BEQ L1

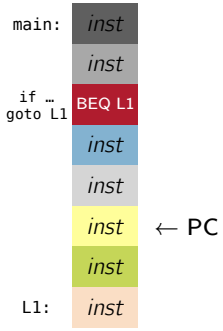
Befehlsabarbeitung **mit Pipelining** **Hazard!**



← PC

	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*
6	*o*	*o*	*o*	BEQ L1
7	*inst*	*o*	*o*	*o*

Befehlsabarbeitung **mit Pipelining** **Hazard!**



← PC

	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*
6	*o*	*o*	*o*	BEQ L1
7	*inst*	*o*	*o*	*o*
8	*inst*	*inst*	*o*	*o*

Befehlsabarbeitung **mit Pipelining** **Hazard!**

control hazard!
branch penalty: P_b = 3 Zyklen

	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*
6	*o*	*o*	*o*	BEQ L1
7	*inst*	*o*	*o*	*o*
8	*inst*	*inst*	*o*	*o*

Befehlsabarbeitung **mit Pipelining** **Hazard!**

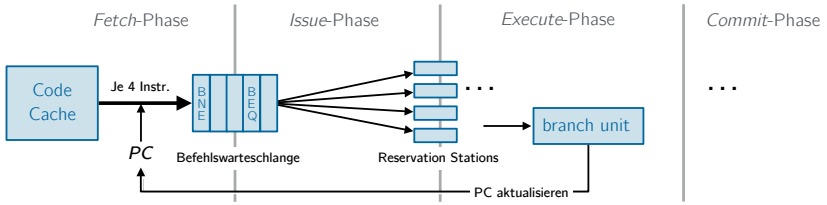
	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		
3	BEQ L1	*inst*	*inst*	
4	*o*	BEQ L1	*inst*	*inst*
5	*o*	*o*	BEQ L1	*inst*
6	*o*	*o*	*o*	BEQ L1
7	*inst*	*o*	*o*	*o*
8	*inst*	*inst*	*o*	*o*

control hazard!
branch penalty: P_b = 3 Zyklen

Problem: Pipeline Hazards
- Kann Vorteile des Pipelining wieder zunichte machen
- Problem vervielfacht sich mit Pipeline-Länge und Superskalarität

Sprungkosten bei superskalarer Architektur

- Beispiel: 4-fach superskalarer Aufbau mit *out-of-order execution*

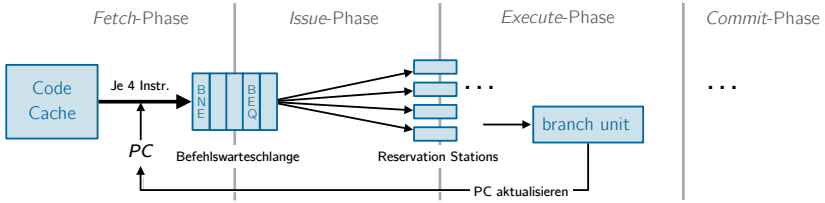


- 20 Prozent aller Instruktionen seien Sprünge, $P_b = 8$

$$IPC_{real} = \frac{IPC_{ideal}}{1+P_b \cdot 0{,}2}$$

Sprungkosten bei superskalarer Architektur

- Beispiel: 4-fach superskalarer Aufbau mit *out-of-order execution*

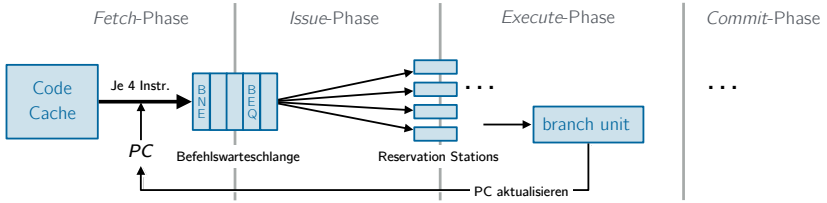


- 20 Prozent aller Instruktionen seien Sprünge, $P_b = 8$

$$IPC_{real} = \frac{IPC_{ideal}}{1+P_b \cdot 0{,}2} = \frac{4}{1+8 \cdot 0{,}2} = \mathbf{1,54}$$

Sprungkosten bei superskalarer Architektur

- Beispiel: 4-fach superskalarer Aufbau mit *out-of-order execution*

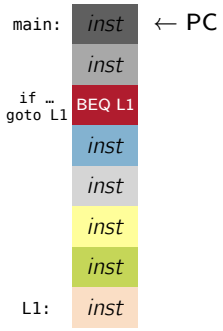


- 20 Prozent aller Instruktionen seien Sprünge, $P_b = 8$

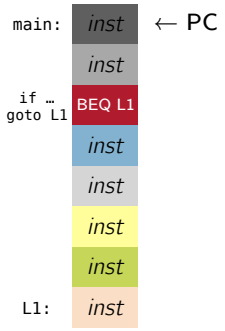
$$IPC_{real} = \frac{IPC_{ideal}}{1+P_b \cdot 0{,}2} = \frac{4}{1+8 \cdot 0{,}2} = \mathbf{1,54}$$

1,54 / 4 ➠ **weniger als 40%** der möglichen Leistung!

Befehlsabarbeitung **mit spekulativer Ausführung**

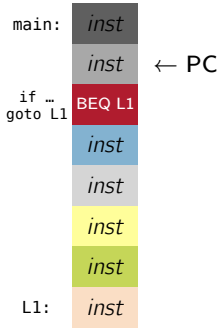


Befehlsabarbeitung **mit spekulativer Ausführung**



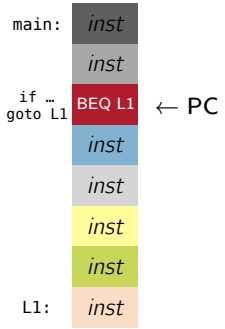
	IF	ID	EX	WB

Befehlsabarbeitung **mit spekulativer Ausführung**



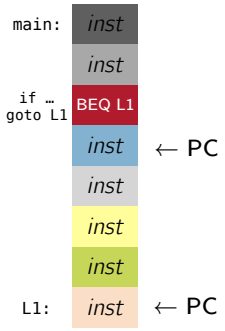
	IF	ID	EX	WB
1	*inst*			

Befehlsabarbeitung **mit spekulativer Ausführung**



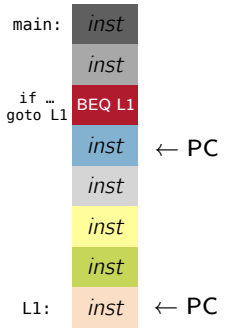
	IF	ID	EX	WB
1	*inst*			
2	*inst*	*inst*		

Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	

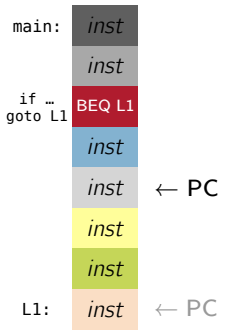
Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	

- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - Verwerfe ggfs. die Ergebnisse –> *rollback*

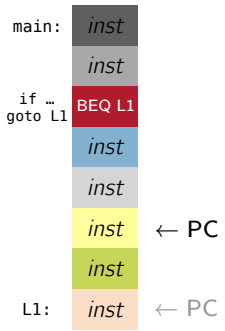
Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	inst	BEQ L1	inst	inst

- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - Verwerfe ggfs. die Ergebnisse –> *rollback*

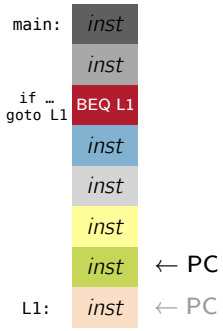
Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	inst	BEQ L1	inst	inst
5	inst	inst	BEQ L1	inst

- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - Verwerfe ggfs. die Ergebnisse –> *rollback*

Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	inst	BEQ L1	inst	inst
5	inst	inst	BEQ L1	inst
6	inst	inst	inst	BEQ L1

- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - Verwerfe ggfs. die Ergebnisse –> *rollback*

Befehlsabarbeitung **mit spekulativer Ausführung**

	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	inst	BEQ L1	inst	inst
5	inst	inst	BEQ L1	inst
6	inst	inst	inst	BEQ L1
7	inst	inst	inst	inst

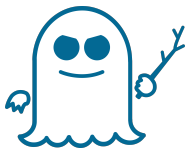
- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - Verwerfe ggfs. die Ergebnisse –> *rollback*

Befehlsabarbeitung **mit spekulativer Ausführung**

	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	~~inst~~	BEQ L1	inst	inst
5	~~inst~~	~~inst~~	BEQ L1	inst
6	~~inst~~	~~inst~~	~~inst~~	BEQ L1
7				

- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - **Verwerfe ggfs. die Ergebnisse** –> *rollback*

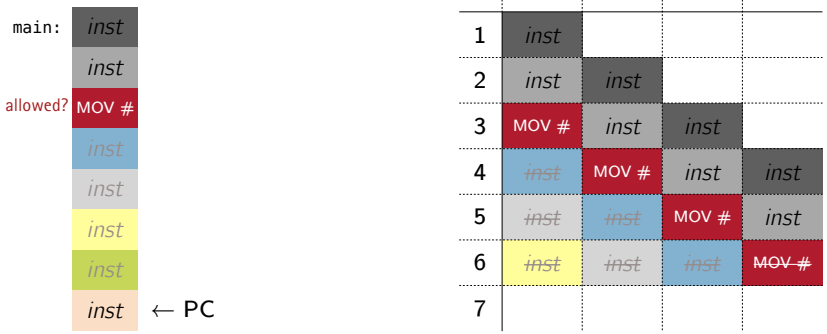
Befehlsabarbeitung **mit spekulativer Ausführung**



	IF	ID	EX	WB
1	inst			
2	inst	inst		
3	BEQ L1	inst	inst	
4	~~inst~~	BEQ L1	inst	inst
5	~~inst~~	~~inst~~	BEQ L1	inst
6	~~inst~~	~~inst~~	~~inst~~	BEQ L1
7				

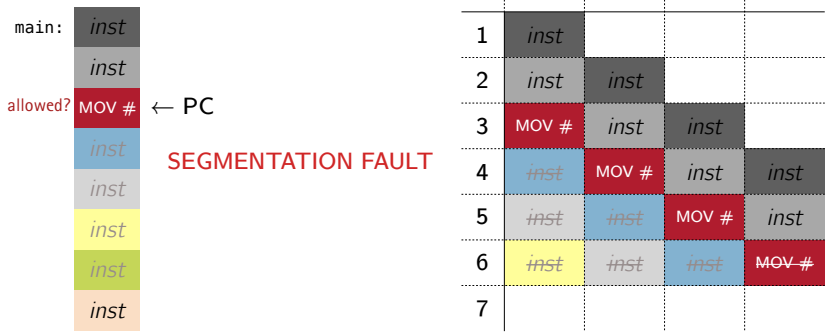
- Sprungziel unklar?
  - „Rate" das Sprungziel (Sprungvorhersage) –> *branch prediction*
  - Führe Instruktionen am Sprungziel „auf Verdacht" aus –> *speculative execution*
  - **Verwerfe ggfs. die Ergebnisse** –> *rollback*

Intel: Speicherzugriff **mit spekulativer Ausführung**



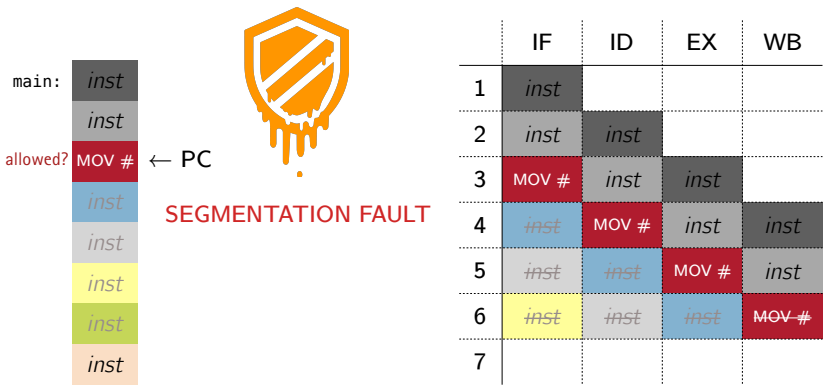
- Speicherzugriff erlaubt? (MMU fragen)
  - Gehe davon aus, dass Zugriff erlaubt —> *speculative execution*
  - **Falls nein: Verwerfe die Ergebnisse** —> *rollback*
  - Starte Fehlerbehandlung —> *trap*

Intel: Speicherzugriff **mit spekulativer Ausführung**



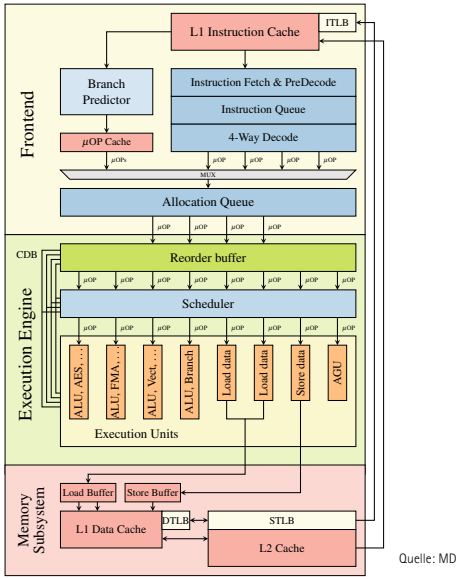
- Speicherzugriff erlaubt? (MMU fragen)
  - Gehe davon aus, dass Zugriff erlaubt —> *speculative execution*
  - **Falls nein: Verwerfe die Ergebnisse** —> *rollback*
  - Starte Fehlerbehandlung —> *trap*

Intel: Speicherzugriff **mit spekulativer Ausführung**



- Speicherzugriff erlaubt? (MMU fragen)
  - Gehe davon aus, dass Zugriff erlaubt —> *speculative execution*
  - **Falls nein: Verwerfe die Ergebnisse** —> *rollback*
  - Starte Fehlerbehandlung —> *trap*

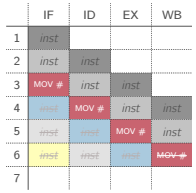
Architektur aktueller Intel-Prozessoren



Quelle: MD

Zusammenfassung: Spekulative Ausführung

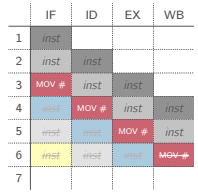
- Prozessor führt **bedingte Anweisungen spekulativ** aus
  - Grund: Bedingungstest benötigt mehrere Takte
  - Ziel: Anhalten der Pipeline verhindern
  - Benötigt: Guter Prediktor des Bedingungsergebnis
    Mechanismus zum Zurückrollen bei Misprediktion



- Bei **Sprungbefehlen** (Intel, AMD, ARM, ...) *–> Spectre*
  - Vorhersage anhand der letzten 2–4 Sprünge an dieser Stelle
- Bei **Speicheroperationen** (nur Intel) *–> Meltdown*
  - Statische Vorhersage: Gehe davon aus, dass Zugriff erlaubt ist

Zusammenfassung: Spekulative Ausführung

- Prozessor führt **bedingte Anweisungen spekulativ** aus
  - Grund: Bedingungstest benötigt mehrere Takte
  - Ziel: Anhalten der Pipeline verhindern
  - Benötigt: Guter Prediktor des Bedingungsergebnis
    Mechanismus zum Zurückrollen bei Misprediktion



- Bei **Sprungbefehlen** (Intel, AMD, ARM, ...) *–> Spectre*
  - Vorhersage anhand der letzten 2–4 Sprünge an dieser Stelle
- Bei **Speicheroperationen** (nur Intel) *–> Meltdown*
  - Statische Vorhersage: Gehe davon aus, dass Zugriff erlaubt ist

Wichtig: Spekulative Ausführung ist nicht *per se* „falsch"

- Ergebnisse werden **verworfen** bevor für sie die Software sichtbar werden
- Sie hinterlässt jedoch ein „messbares Geräusch" in der Mikroarchitektur

Worum geht es? Wissen, was dahinter steckt!

Leibniz Universität Hannover

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Worum geht es? Wissen, was dahinter steckt!

Leibniz Universität Hannover

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

19

Seitenkanal-Attacke *(covert channel attack)*

- Informationsverarbeitung verursacht immer auch **physikalische Effekte**
  - Energieverbrauch, Wärmeentwicklung, Zeitverhalten
  - Im Modell ist dieses unberücksichtigt
- Die Messung dieser physikalischen Eigenschaften ermöglicht oft (stochastisch) **Rückschlüsse** auf die verarbeitete Information!

—> Prinzip des Seitenkanal-Angriffs

Seitenkanal-Attacke *(covert channel attack)*

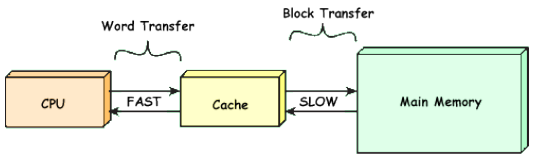
- Informationsverarbeitung verursacht immer auch **physikalische Effekte**
  - Energieverbrauch, Wärmeentwicklung, Zeitverhalten
  - Im Modell ist dieses unberücksichtigt
- Die Messung dieser physikalischen Eigenschaften ermöglicht oft (stochastisch) **Rückschlüsse** auf die verarbeitete Information!

—> Prinzip des Seitenkanal-Angriffs

Meltdown und Spectre sind Seitenkanal-Angriffe!

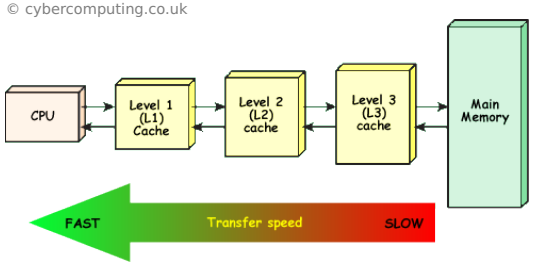
- „Geheimnis" wird in spekulativen Berechnungen benutzt (und verworfen)
- „Geheimnis" wird *indirekt* abgegriffen über das Zeitverhalten des Caches

Schnelle Zwischenspeicher: Caches



Cache overview

© cybercomputing.co.uk



Cache overview

© cybercomputing.co.uk

Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Worum geht es? Wissen, was dahinter steckt!

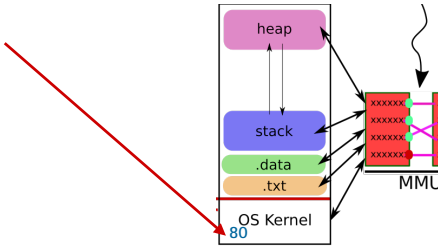
- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 ];
}
```

Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 ];
}
```

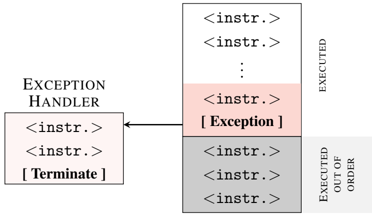


Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 ];
}
```

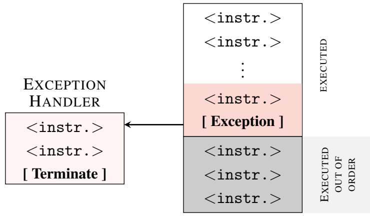
Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 );
}
```



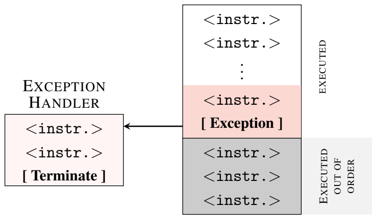
Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 );
}
```



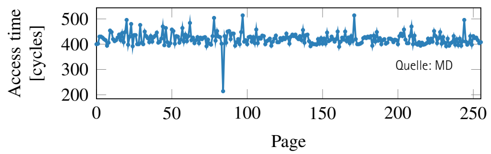
Meltdown im Detail

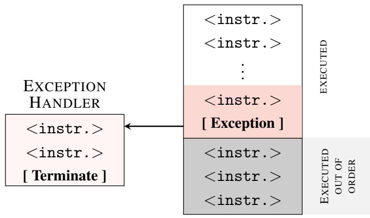
```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 );
}
```



Meltdown im Detail

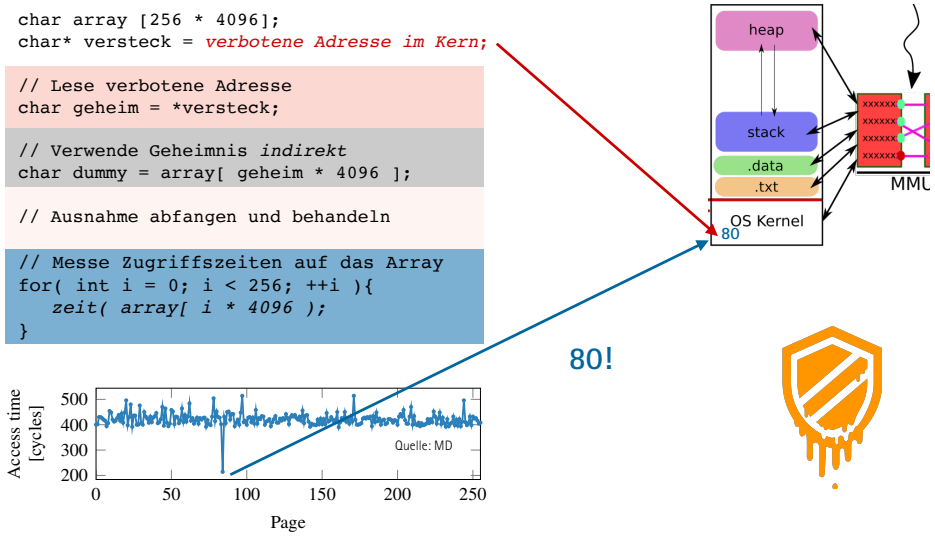
```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 );
}
```





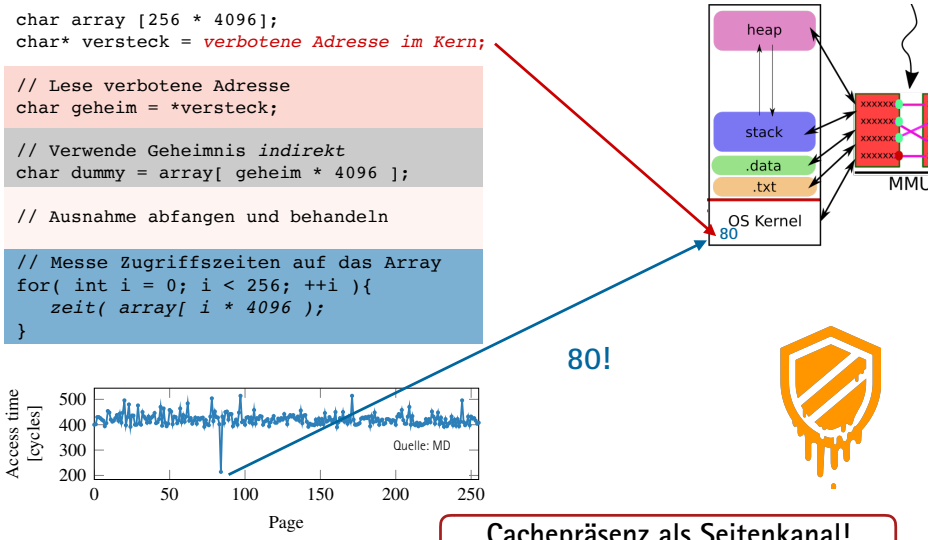
Meltdown im Detail

```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 ];
}
```



Meltdown im Detail

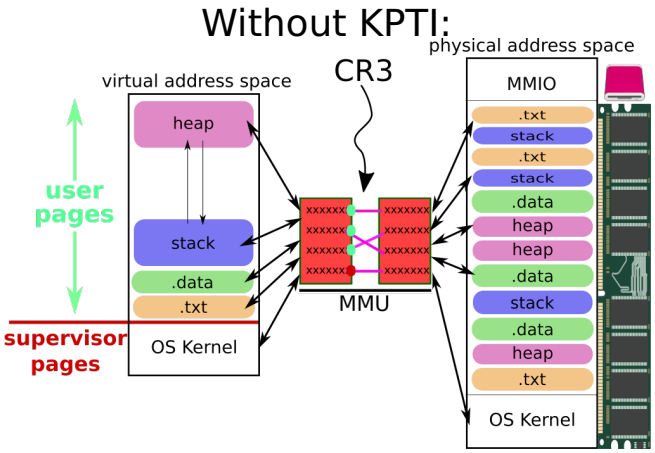
```
char array [256 * 4096];
char* versteck = verbotene Adresse im Kern;

// Lese verbotene Adresse
char geheim = *versteck;

// Verwende Geheimnis indirekt
char dummy = array[ geheim * 4096 ];

// Ausnahme abfangen und behandeln

// Messe Zugriffszeiten auf das Array
for( int i = 0; i < 256; ++i ){
   zeit( array[ i * 4096 ];
}
```



Cachepräsenz als Seitenkanal!

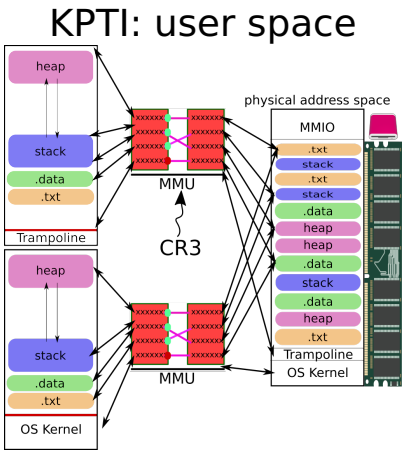
Abhilfe: Kernspeicher im *user mode* ausblenden: KPTI-Patch

- KPTI: Kernel Page Table Isolation – Extra Seitentabelle für Kern (pro Prozess!)



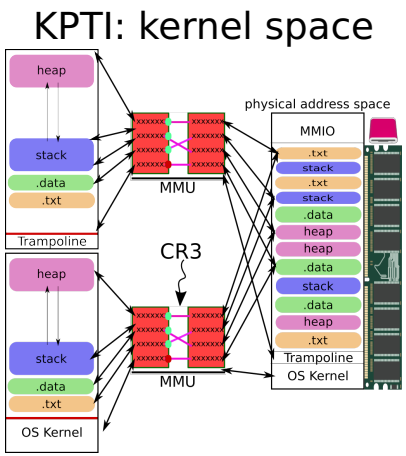
Abhilfe: Kernspeicher im *user mode* ausblenden: KPTI-Patch

- KPTI: Kernel Page Table Isolation – Extra Seitentabelle für Kern (pro Prozess!)



Abhilfe: Kernspeicher im *user mode* ausblenden: KPTI-Patch

- KPTI: Kernel Page Table Isolation – Extra Seitentabelle für Kern (pro Prozess!)



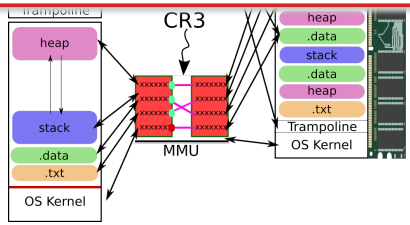
Abhilfe: Kernspeicher im *user mode* ausblenden: KPTI-Patch

- KPTI: Kernel Page Table Isolation – Extra Seitentabelle für Kern (pro Prozess!)

Die Laufzeitkosten können erheblich sein!

- Intel: 2% (auf aktueller Hardware!)
- Linux-Developers: 5% (auf aktueller Hardware!)
- IO-Intensiver Betrieb: bis zu 30% (auf aktueller Hardware!)

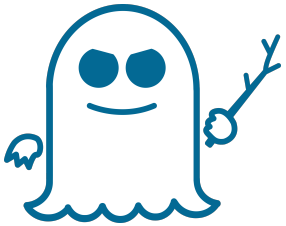
Dennoch gibt es keine Alternative!



Spectre im Überblick

- Spectre verwendet ebenfalls Cache-Präsenz als Seitenkanal
- Spekulative Ausführung durch gezieltes Misstraining der Sprungvorhersage
- Angriff rein im *user mode*, ohne Beteiligung des Kerns – **kein Kernel Patch möglich!**
- Erfordert **sehr detailliertes Wissen** über den Zielprozess und seinen Programmcode

```
if (offset < arr1->length) {
  char val = arr1->data[offset];
  long idx = ((val&1)*0x100)+0x200;
  if (idx < arr2->length)
    char val2 = arr2->data[idx];
}
```



Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Worum geht es? Wissen, was dahinter steckt!

- Betriebssystem-Grundlagen
  - Speicher-/Adressraumverwaltung
- Rechnerarchitektur-Grundlagen
  - Pipelining
  - Spekulative Ausführung
  - Sprungvorhersage
- IT-Sicherheits-Grundlagen
  - Seitenkanäle
  - Caching
- Meltdown und Spectre
  - Meltdown im Detail
  - Spectre im Überblick
- Zusammenfassung

Zusammenfassung

- Meltdown und Spectre sind **Seitenkanalangriffe**
  - „Geheimnis" wird indirekt abgegriffen
  - aus dem Zeitverhalten des Caches
- Ansatz: Sorge dafür, das CPU „Geheimnis" spekulativ liest
  - wird später verworfen
  - verwende es geschickt als Arrayindex – und „markiere" so „Geheimnis" im Cache
- Meltdown ist wie ein **Generalschlüssel**
  - Ermöglicht Lesezugriff auf das **gesamten RAM**
  - Patchen auf Betriebssystemebene möglich (und sinnvoll!)
- Spectre ist eine Anleitung zum Erstellen von **Spezialschlüsseln**
  - Ermöglicht Lesezugriffe auf „nahestehenden" Adressraum
  - Detaillierte Kenntnisse des Zielprozesses erforderlich

Vielen Dank für Ihre Aufmerksamkeit :-)

Prof. Dr.-Ing. habil. Daniel Lohmann
lohmann@sra.uni-hannover.de
Fachgebietsleiter

Betriebssysteme

Zeit für Fragen

Referenz


```
@article{MD,
      author = {Lipp, Moritz and Schwarz, Michael and Gruss, Daniel and Prescher,
     Thomas and Haas, Werner and Mangard, Stefan and Kocher, Paul and Genkin,
     Daniel and Yarom, Yuval and Hamburg, Mike},
      title = {Meltdown},
      journal = {ArXiv e-prints},
      archivePrefix = "arXiv",
      eprint = {1801.01207},
      year = 2018,
      month = jan,
}
```