Unsichere Systeme trotz fehlerfreier Software?

Wie Hardware die Sicherheit von Software untergräbt

Michael Schwarz (@misc0110)

21.10.2021

CISPA Helmholtz-Zentrum für Informationssicherheit

Wer bin ich?

Michael Schwarz

Faculty @ CISPA

Fokus auf Seitenkanal Angriffe

@misc0110

michael.schwarz@cispa.de

CISPA Ziel

- Aktuell: Ursachen im Kern oft nicht verstanden
- Ziel: Gesamte Klasse von Angriffen verhindern
  - Kernursachen erforschen und verstehen
  - Grundlagenforschung mit anwendungsorientierter Forschung
  - Wissenschaftliche Exzellenz als Erfolgskriterium
  - Kaderschmiede für die nächste Generation

CISPA Forschungsfelder

Vertrauenswürdige
Informationsverarbeitung

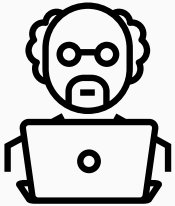
Zuverlässige
Sicherheitsgarantien

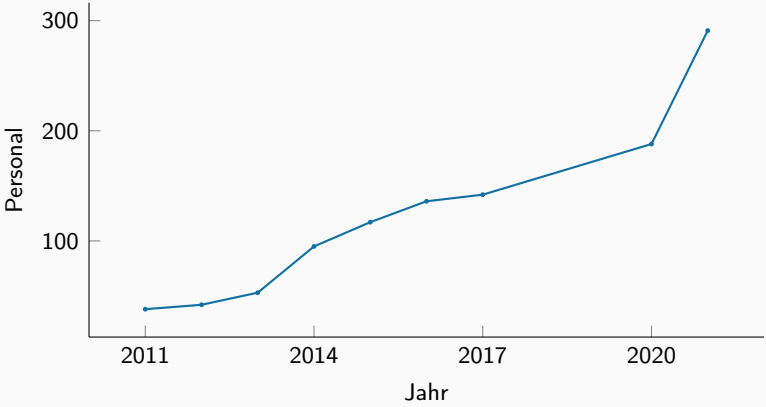
Erkennungs- und
Verteidigungsmechanismen
für Bedrohungen

Sichere Mobile und
Autonome Systeme

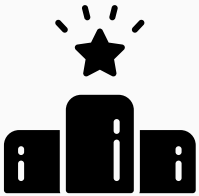
Empirische und
Verhaltensorientierte
Sicherheit

CISPA Personal





CISPA Ranking



CSRankings: Computer Science Rankings

CSRankings is a metrics-based ranking of top computer science institutions around the world. **Click on a triangle** (▶) to expand areas or institutions. **Click on a name** to go to a faculty member's home page. **Click on a chart icon** (the 📊 after a name or institution) to see the distribution of their publication areas as a bar chart. **Click on a Google Scholar icon** (🎓) to see publications, and **click on the DBLP logo** (🅓) to go to a DBLP entry.
Applying to grad school? Read this first.

Rank institutions in the world by publications from 2011 to 2021

All Areas [off | on]

AI [off | on]

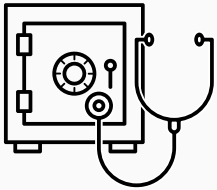
- ▶ Artificial intelligence
- ▶ Computer vision
- ▶ Machine learning & data mining
- ▶ Natural language processing
- ▶ The Web & information retrieval

Systems [off | on]

- ▶ Computer architecture
- ▶ Computer networks
- ▶ Computer security

#	Institution	Count	Faculty
1	▶ CISPA Helmholtz Center	48.6	19
2	▶ Georgia Institute of Technology	41.0	19
3	▶ Carnegie Mellon University	35.7	21
4	▶ Univ. of Illinois at Urbana-Champaign	31.3	21
5	▶ Purdue University	30.8	14
6	▶ Cornell University	29.7	14
7	▶ Northeastern University	29.2	18
8	▶ ETH Zurich	27.5	11
9	▶ Pennsylvania State University	27.1	15

Sichere Systeme



- Fehlerfreie Software bedeutet nicht sichere Systeme
- Hardware kann Geheimnisse preisgeben
- Unbeabsichtigt durch Seiteneffekte

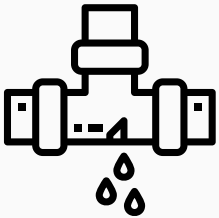
Stromverbrauch

Auführungszeit

CPU Caches

Seitenkanäle

Was ist ein Seitenkanal?

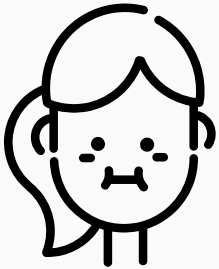


- Unbeabsichtigte Information über (geheime) Daten

→ Verraten Metadaten über die Geheimnisse

- Seitenkanalangriff: Geheimnisse aus den Metadaten schließen

Seitenkanäle in der Realität



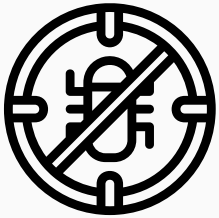
- Menschen verraten Seitenkanalinformationen
  - → Gesichtsausdruck, Mikromimik
  - → Gestik, Haltung
  - → Atmung, Schwitzen
  - → Artikulierung, Tonlage
- Intuitiv wahrnehmbar


2/3 + 3/4 =
26743 : 8 =
12986 x 3 =

Seitenkanäle in Software

- Seitenkanäle gibt es auch in Software
- Können auch für Attacken ausgenutzt werden
- Statt Geräusche misst man meistens Zeitunterschiede

No Bugs

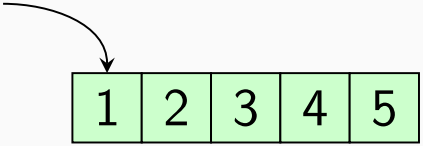


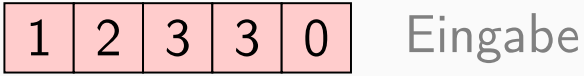
- Seitenkanäle nutzen keine Fehler aus
- Für diese Angriffe: fehlerfreie Software und Hardware
- Verwenden nur Seiteneffekte von Software oder Hardware

→ Fehlerfreie Software bedeutet nicht sichere Software

Beispiel: PIN Vergleich

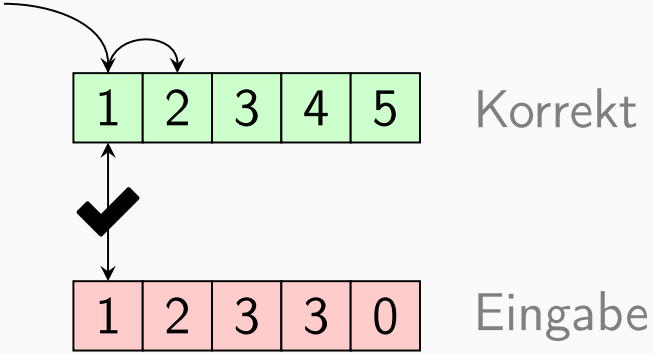
- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen





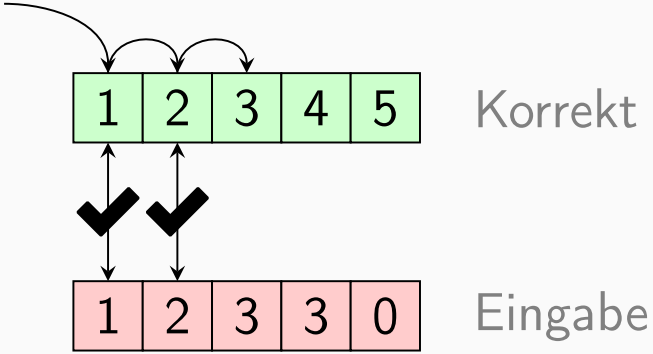
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



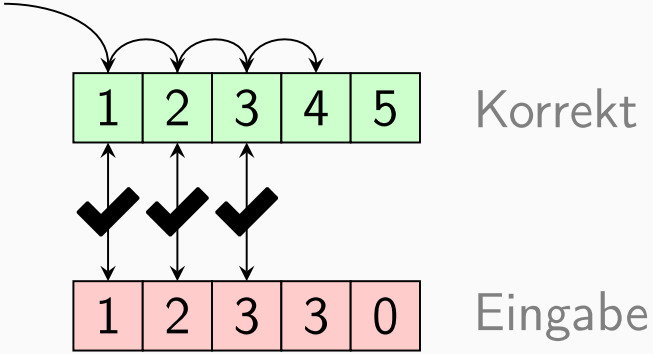
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



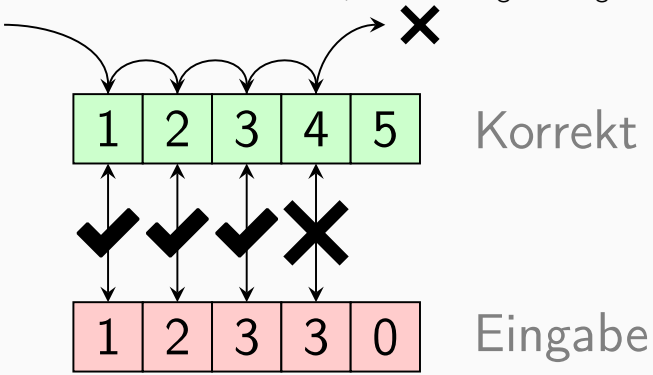
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



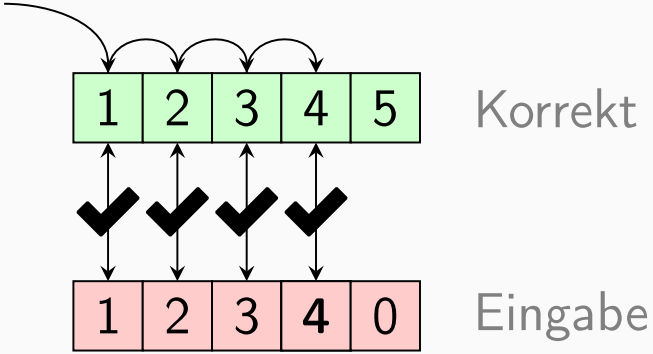
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



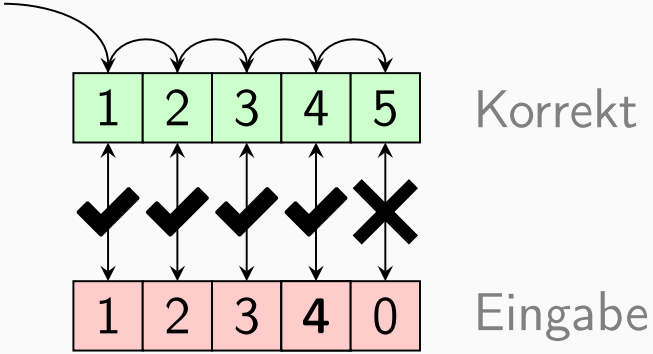
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



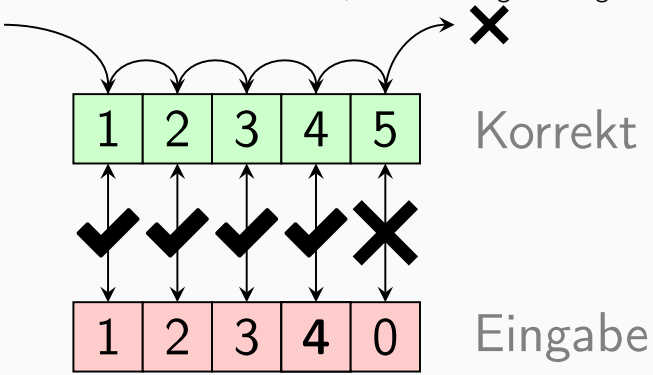
Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



Beispiel: PIN Vergleich

- Trivialer Ansatz:
  - Ziffer für Ziffer vergleichen
  - Sobald sich eine Ziffer unterscheidet, wird der Vergleich abgebrochen



Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	
2000	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	
2000	
3000	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	
2000	
3000	
...	...

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	
2000	
3000	
...	...

- Bei korrekter Ziffer wird die nächste Ziffer überprüft → längere Ausführungszeit

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
0000	
1000	
2000	
3000	
...	...

- Bei korrekter Ziffer wird die nächste Ziffer überprüft → längere Ausführungszeit
- Maximal 10 Versuche um die erste Stelle herauszufinden

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	
1200	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	
1200	
1300	

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	
1200	
1300	
...	...

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	
1200	
1300	
...	...

- Für jede Ziffer wiederholen

Beispiel: PIN Vergleich

- Messung der Ausführungszeit für verschiedene PINs

PIN	Zeit
1000	
1100	
1200	
1300	
...	...

- Für jede Ziffer wiederholen
- Längste Ausführungszeit verrät die Ziffer

Beispiel: PIN Vergleich

- Für jede Ziffer maximal 10 Messungen

Beispiel: PIN Vergleich

- Für jede Ziffer maximal 10 Messungen
- Bei 4 Stellen: 40 Versuche um PIN zu erraten

Beispiel: PIN Vergleich

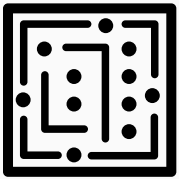
- Für jede Ziffer maximal 10 Messungen
- Bei 4 Stellen: 40 Versuche um PIN zu erraten
- Vergleich zu probieren: 10 000 Möglichkeiten

Beispiel: PIN Vergleich

- Für jede Ziffer maximal 10 Messungen
- Bei 4 Stellen: 40 Versuche um PIN zu erraten
- Vergleich zu probieren: 10 000 Möglichkeiten
- Seitenkanal reduziert Anzahl Versuche um Faktor 250

Mikroarchitekturelle Seitenkanäle

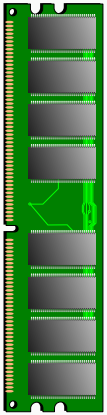
Mikroarchitektur



- Mikroarchitektur beschreibt die interne Art wie CPUs arbeiten
- Nicht sichtbar für Benutzer oder Programmierer
- Ist großteils nicht dokumentiert und kann nicht direkt beobachtet werden

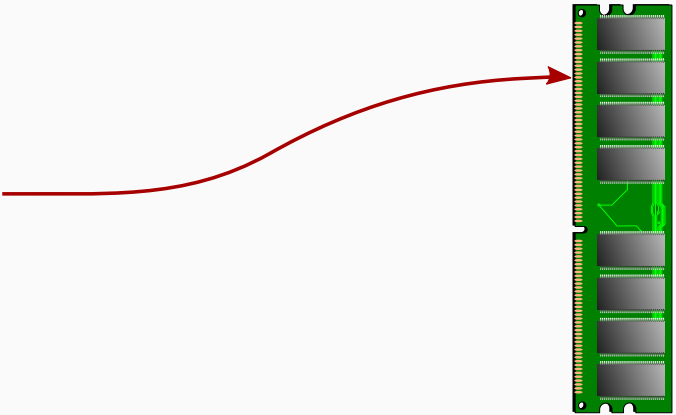
Beispiel: Daten im Speicher (Architektur)

```
val i = 42;
print i;
print i;
```



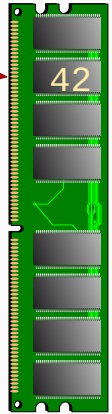
Beispiel: Daten im Speicher (Architektur)

```
val i = 42;
print i;
print i;
```



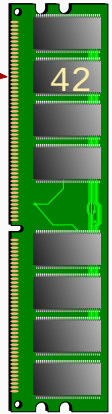
Beispiel: Daten im Speicher (Architektur)

```
val i = 42;
print i;
print i;
```



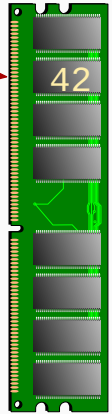
Beispiel: Daten im Speicher (Architektur)

```
val i = 42;
print i;
print i;
```

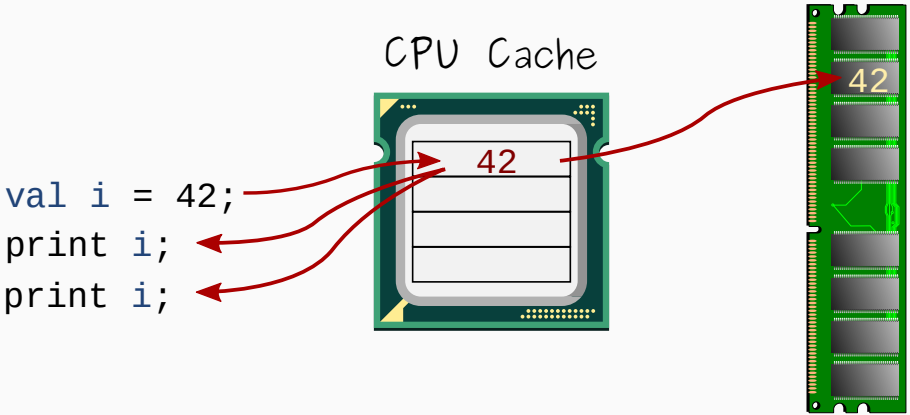


Beispiel: Daten im Speicher (Architektur)

```
val i = 42;
print i;
print i;
```

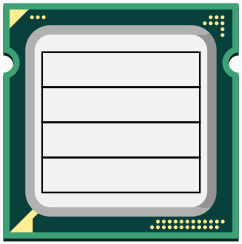


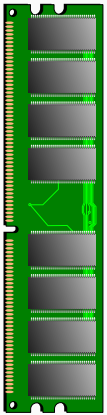
Beispiel: Daten im Speicher (Mikroarchitektur vereinfacht)



CPU Cache

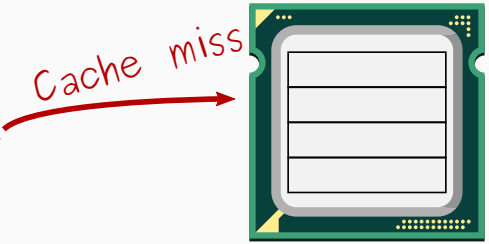
```
print i;
print i;
```





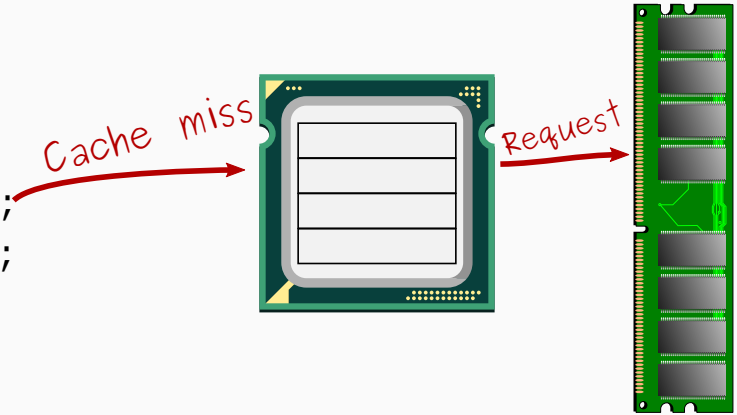
CPU Cache

```
print i;
print i;
```



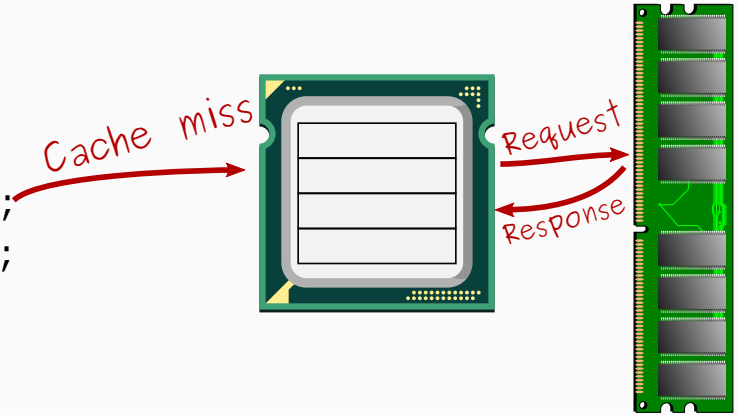
CPU Cache

```
print i;
print i;
```

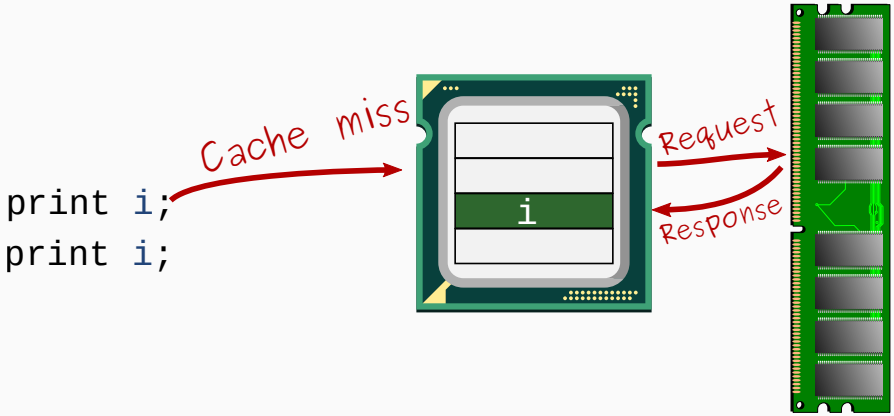


CPU Cache

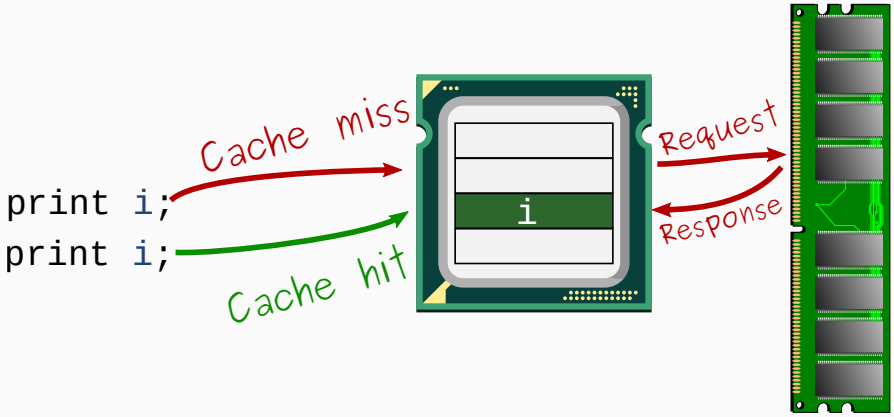
```
print i;
print i;
```



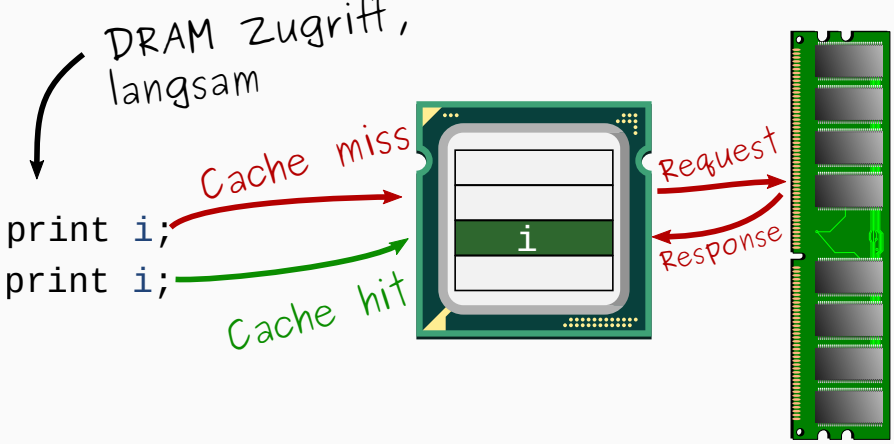
CPU Cache



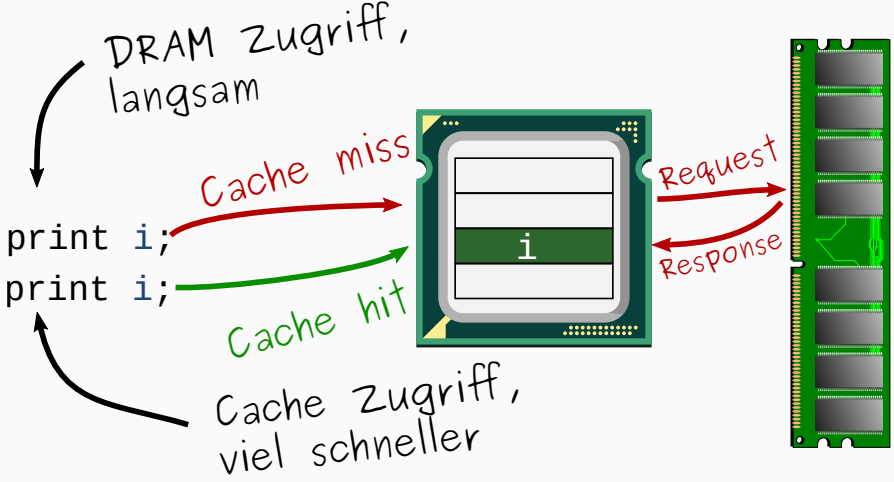
CPU Cache



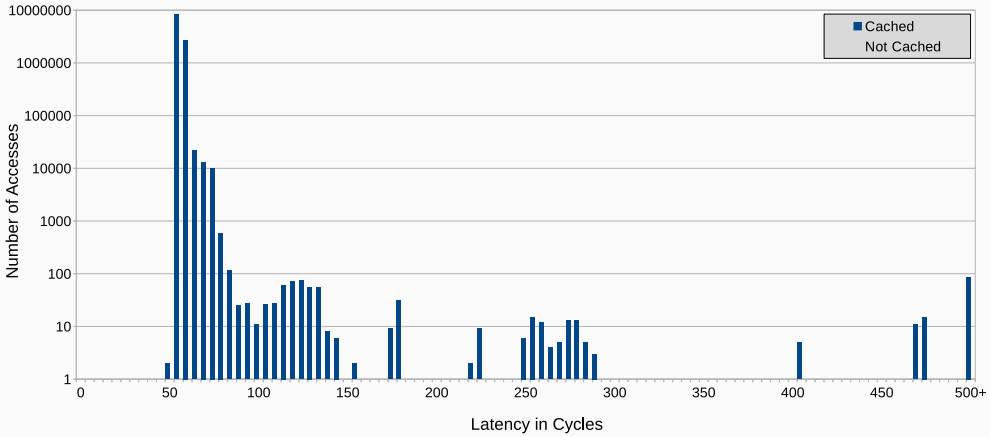
CPU Cache


DRAM Zugriff, langsam
Cache miss
print i;
print i;
Cache hit
i
Request
Response

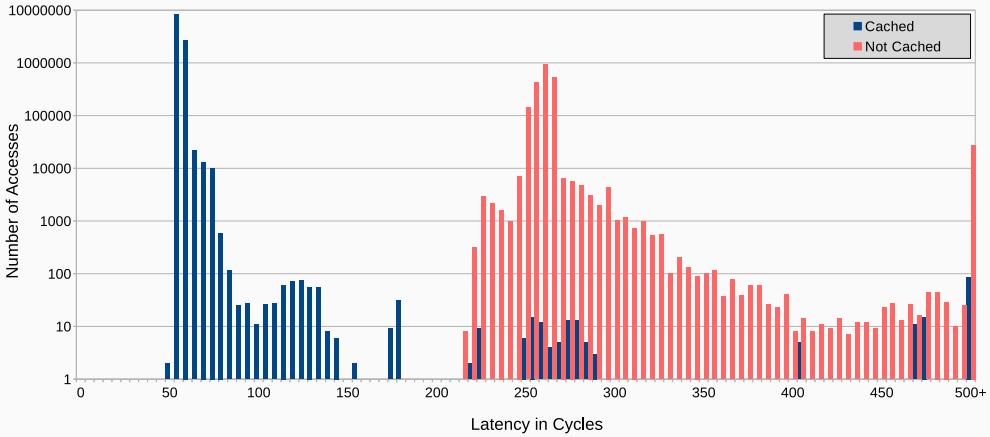
CPU Cache

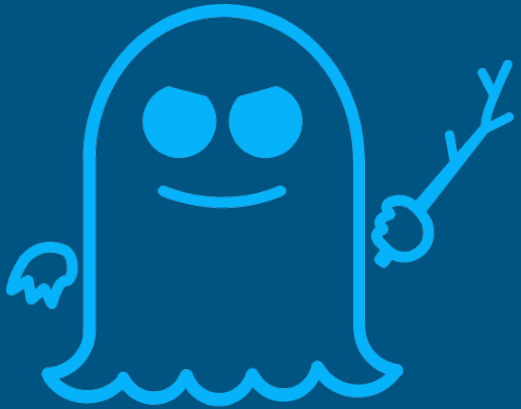

DRAM Zugriff,
langsam
print i;
print i;
Cache miss
Cache hit
i
Request
Response
Cache Zugriff,
viel schneller

CPU Cache


Cached
Not Cached
Number of Accesses
10000000
1000000
100000
10000
1000
100
10
1
0
50
100
150
200
250
300
350
400
450
500+
Latency in Cycles

CPU Cache

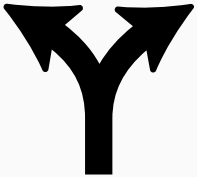

Cached
Not Cached
Number of Accesses
10000000
1000000
100000
10000
1000
100
10
1
0
50
100
150
200
250
300
350
400
450
500+
Latency in Cycles



SPECTRE

Spekulative Ausführung

Branch Prediction



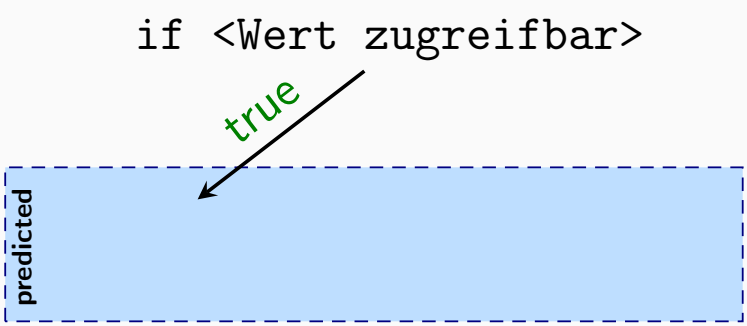
- Programme beinhalten oft Entscheidungen (`if`)
- Sind nicht immer sofort entscheidbar

$\rightarrow$ Entscheidung benötigt eventuell Werte aus dem Speicher

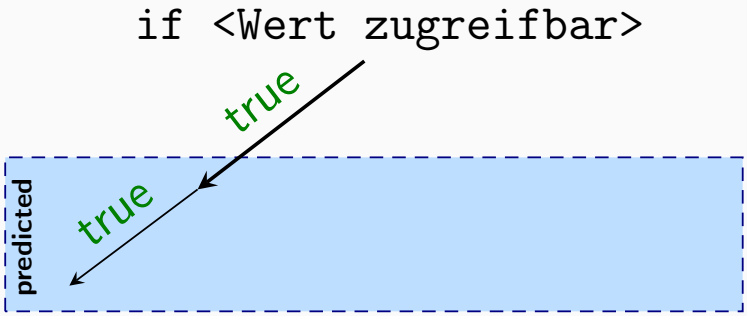
- Wartezeit verkürzen $\rightarrow$ Ergebnis vorhersagen

```
if <Wert zugreifbar>
```

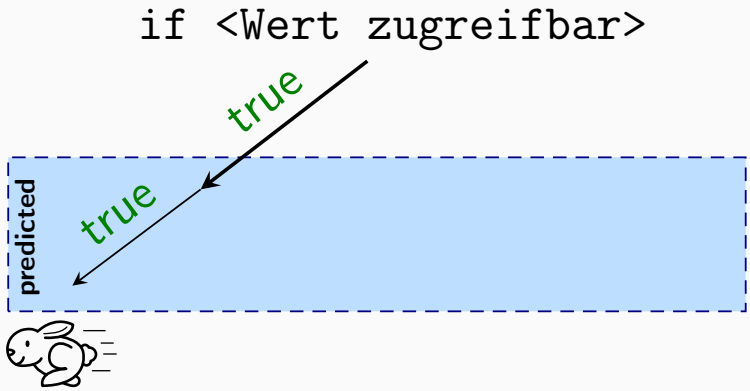
Branch Prediction



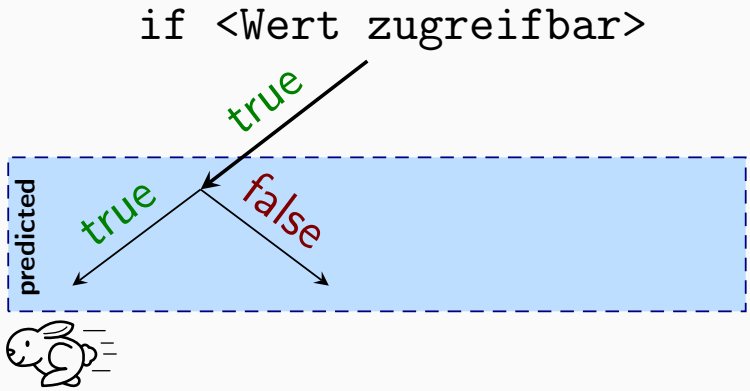
Branch Prediction



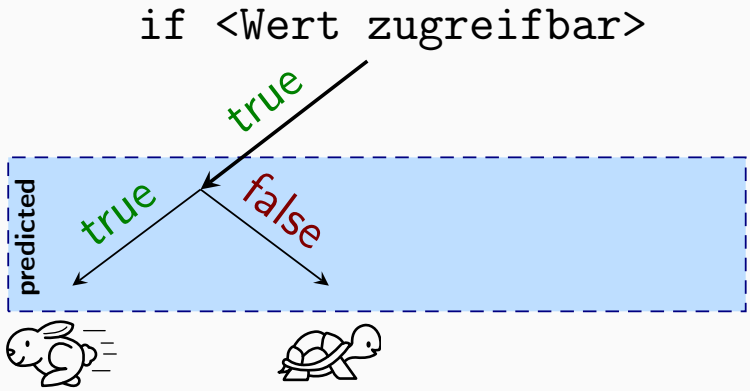
Branch Prediction



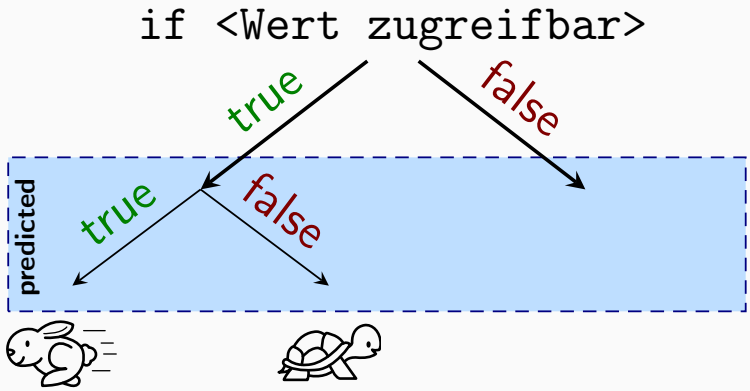
Branch Prediction



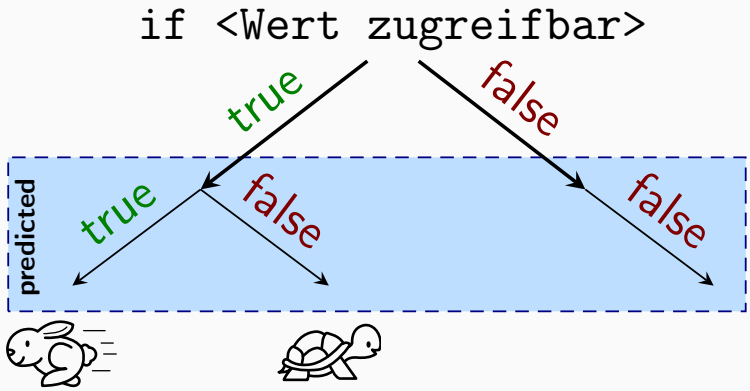
Branch Prediction



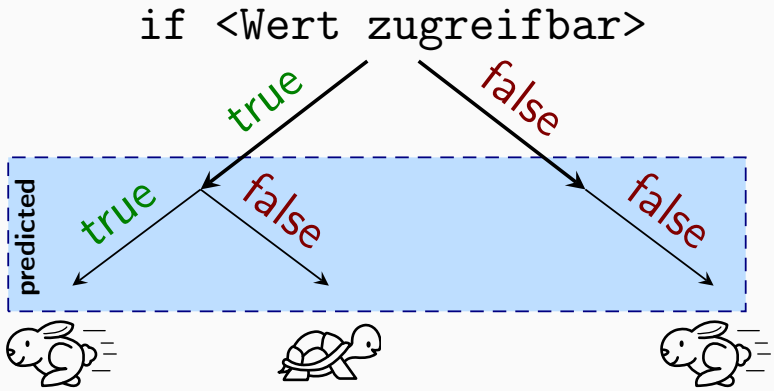
Branch Prediction



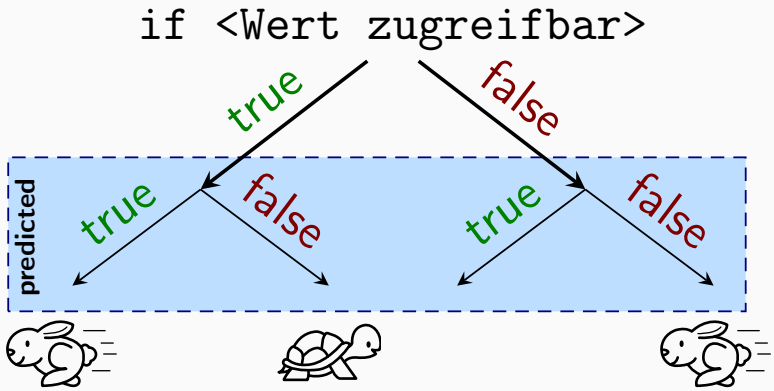
Branch Prediction



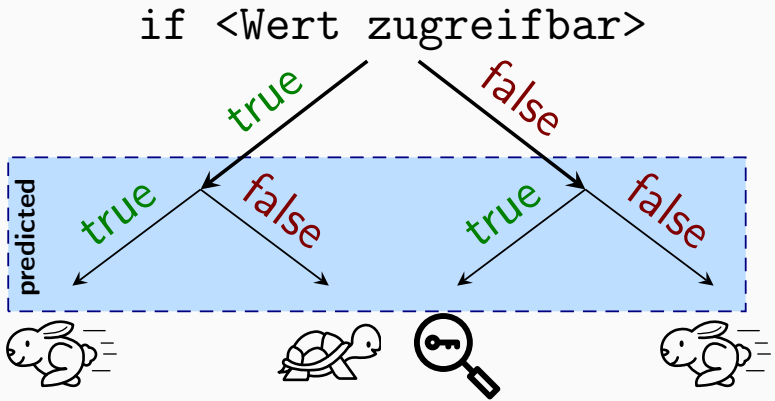
Branch Prediction



Branch Prediction



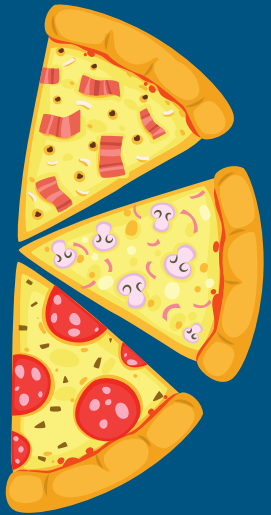
Branch Prediction



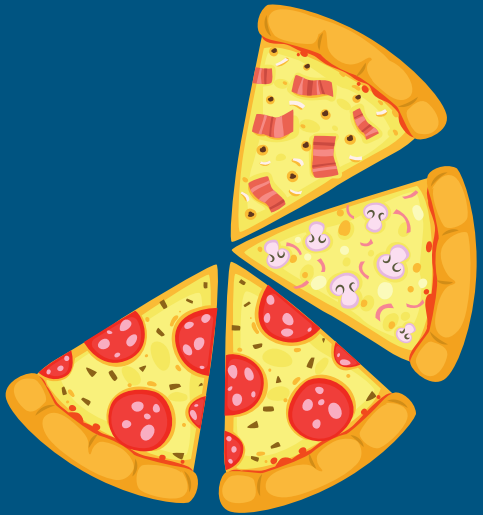

PIZZA
SPECIAL RECIPES

Prosciutto

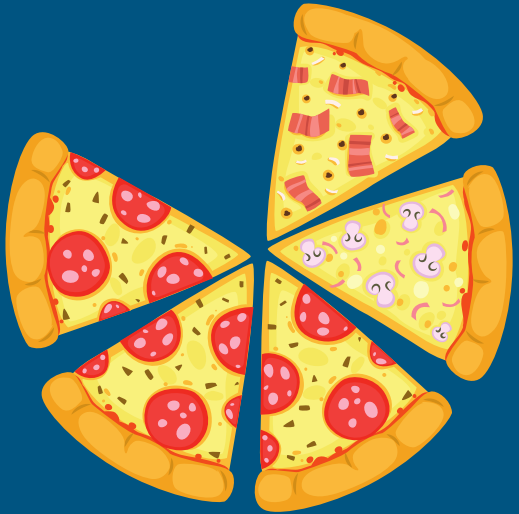
Funghi



Diavolo



Diavolo



Diavolo

Diavolo


»A table for 6 please«

Speculative Cooking


»A table for 6 please«


PIZZA
SPECIAL RECIPES


PIZZA
SPECIAL RECIPES


Pizza

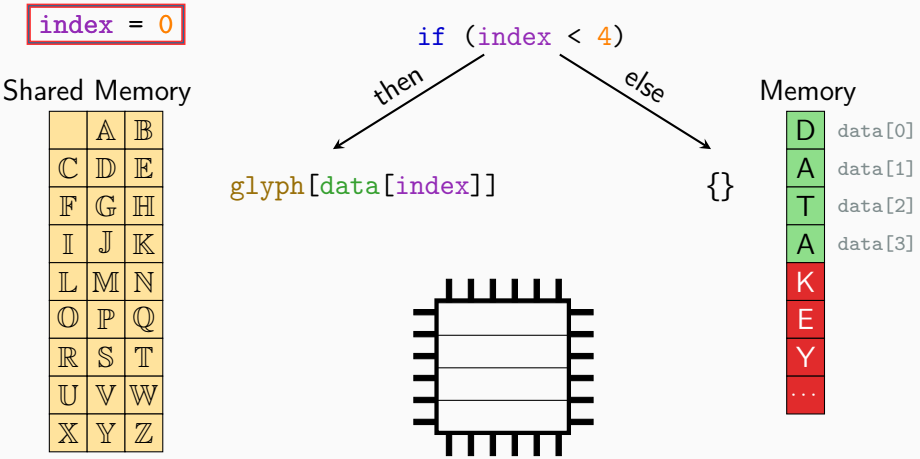

PIZZA
SPECIAL RECIPES


Pizza


PIZZA
SPECIAL RECIPES


Pizza

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 0
```

Shared Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
if (index < 4)
```

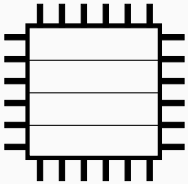
then

```
glyph[data[index]]
```

else

Speculate

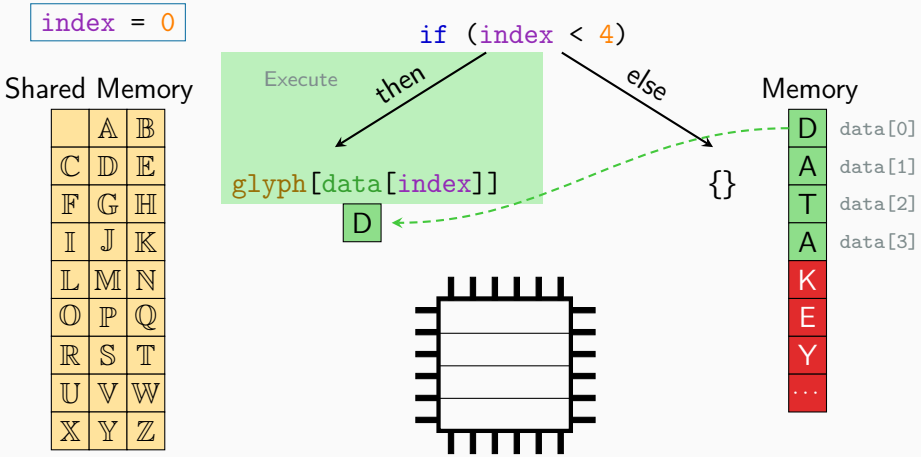
{}



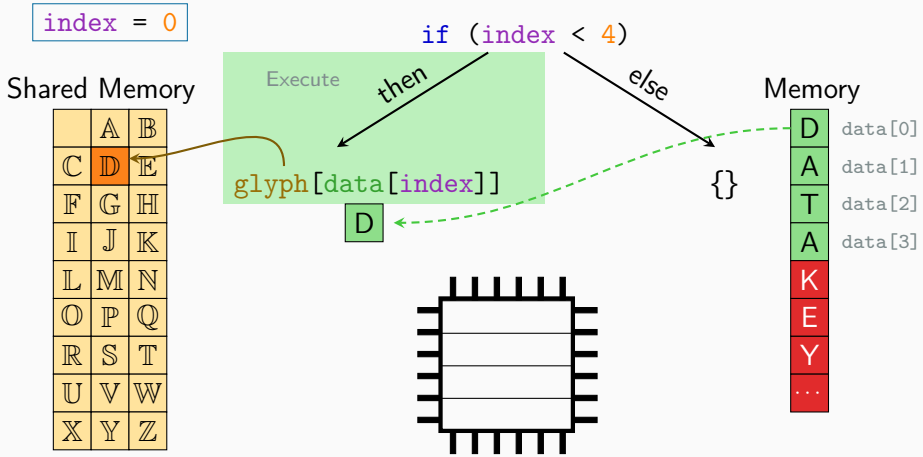
Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

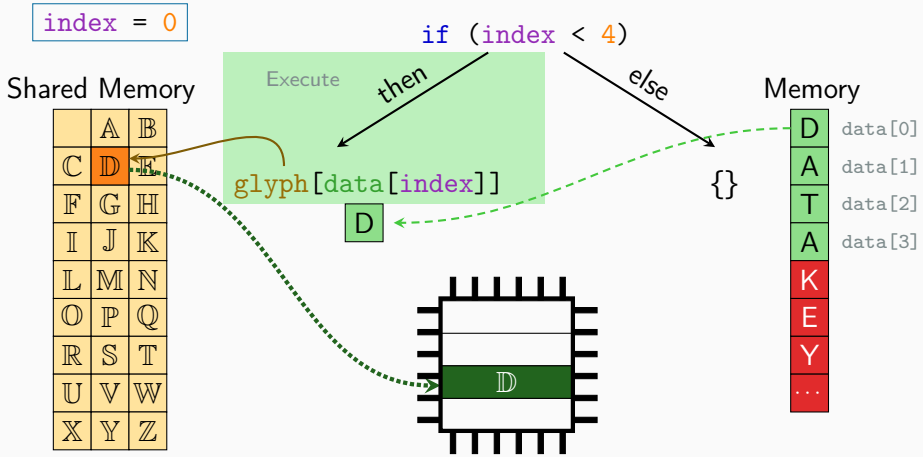
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

Shared Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

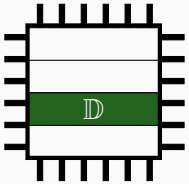
```
if (index < 4)
```

then

```
glyph[data[index]]
```

else

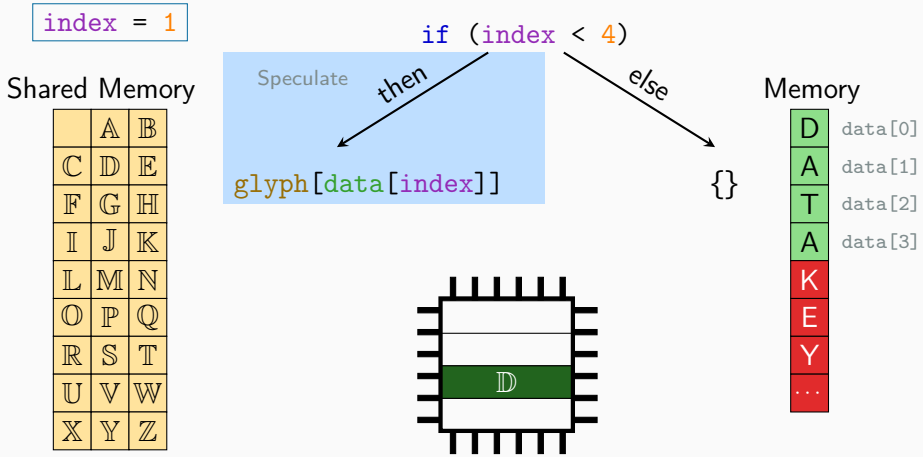
```
{}
```



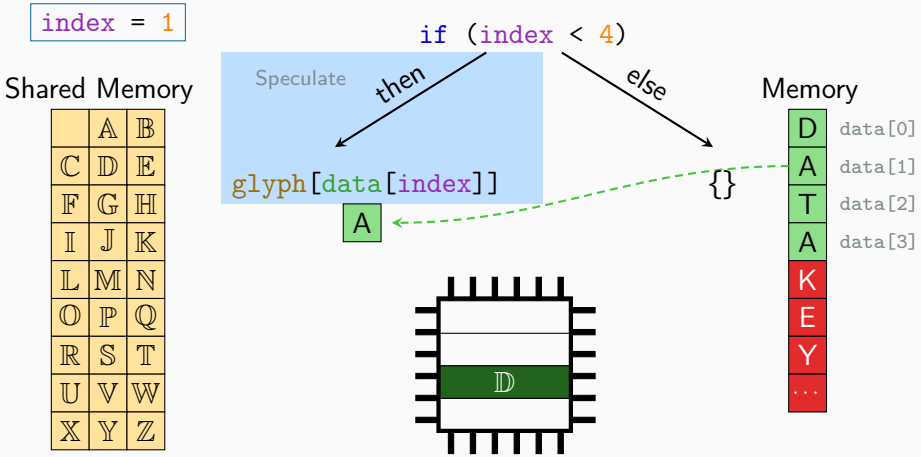
Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

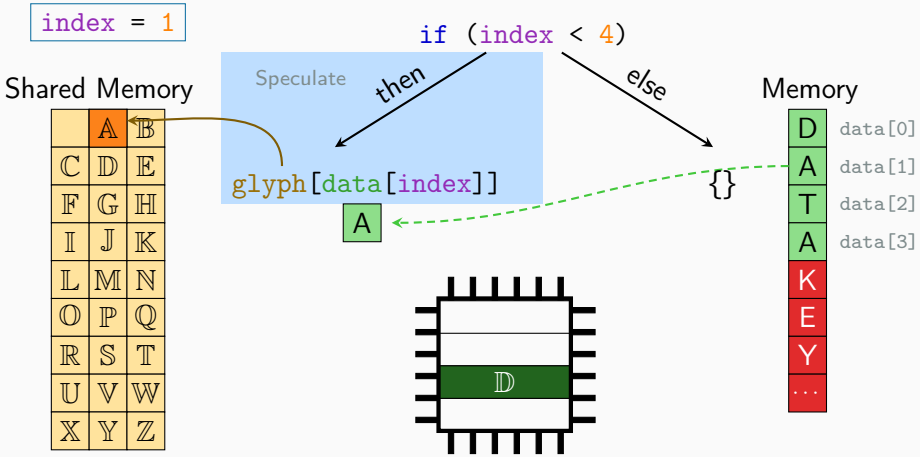
Spectre-PHT (aka Spectre Variant 1)



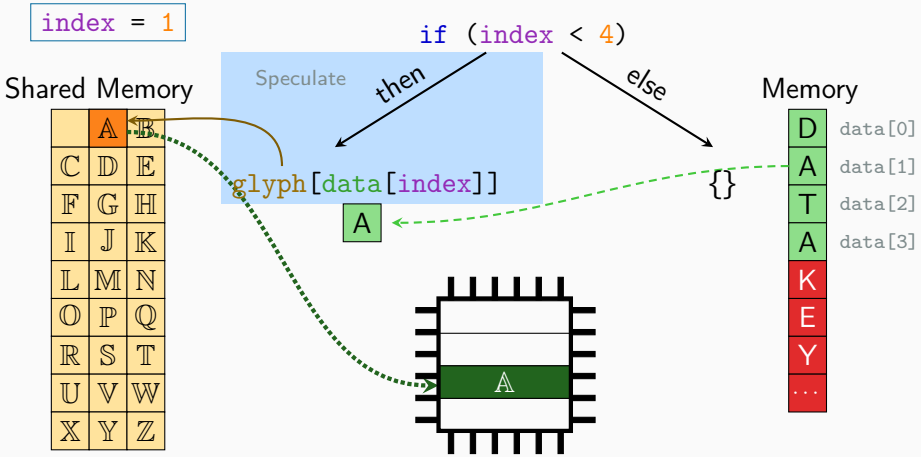
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

Shared Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
if (index < 4)
```

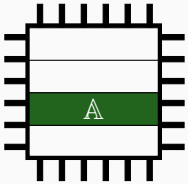
then

Execute

```
glyph[data[index]]
```

else

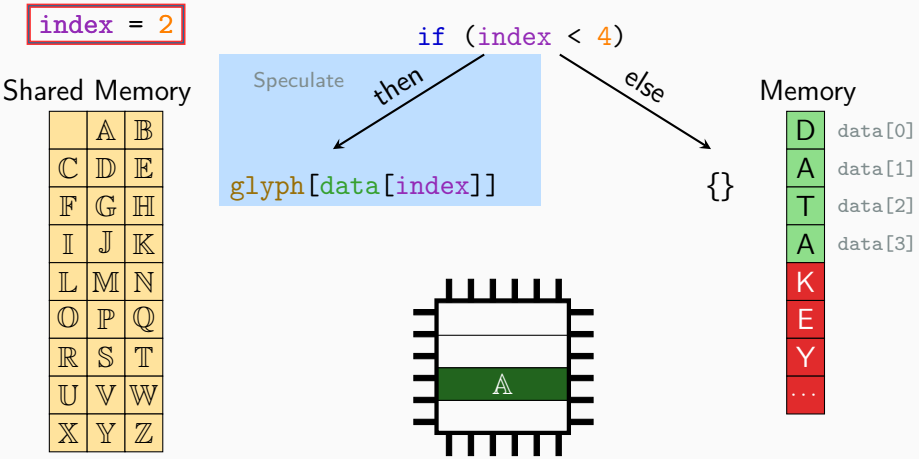
{}



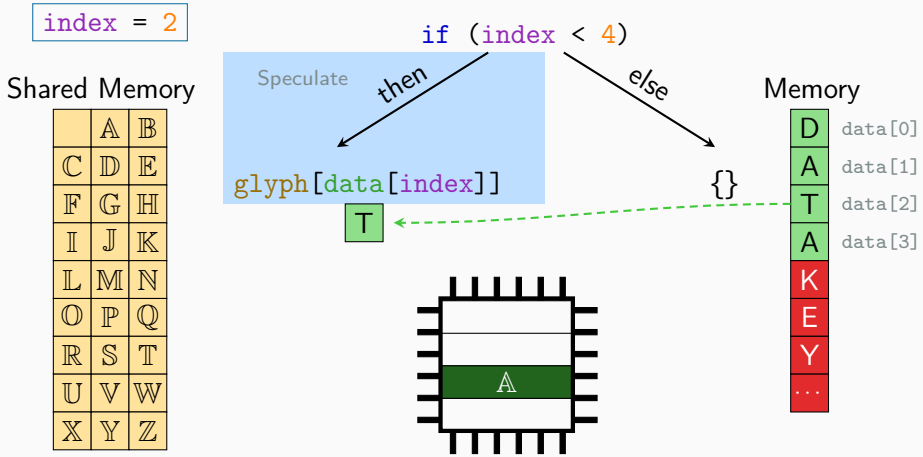
Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

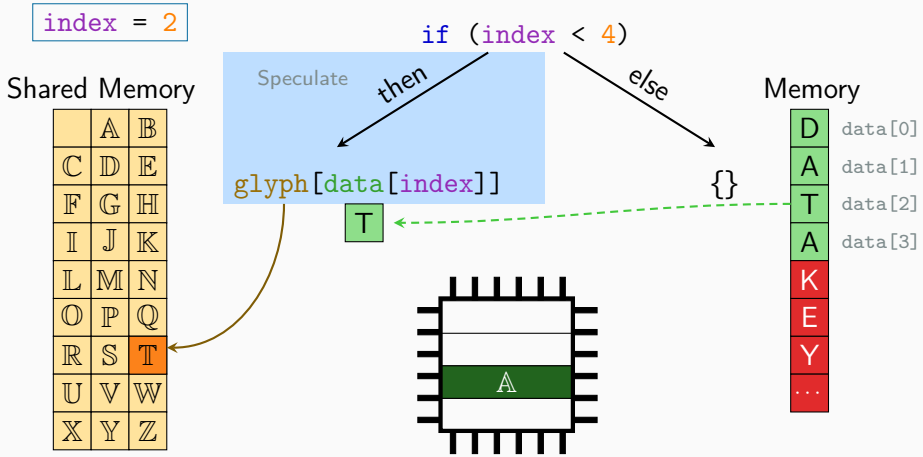
Spectre-PHT (aka Spectre Variant 1)



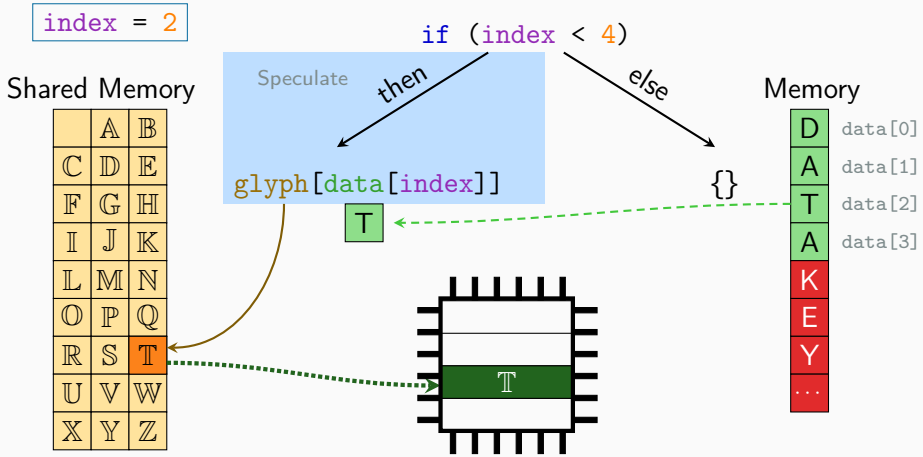
Spectre-PHT (aka Spectre Variant 1)



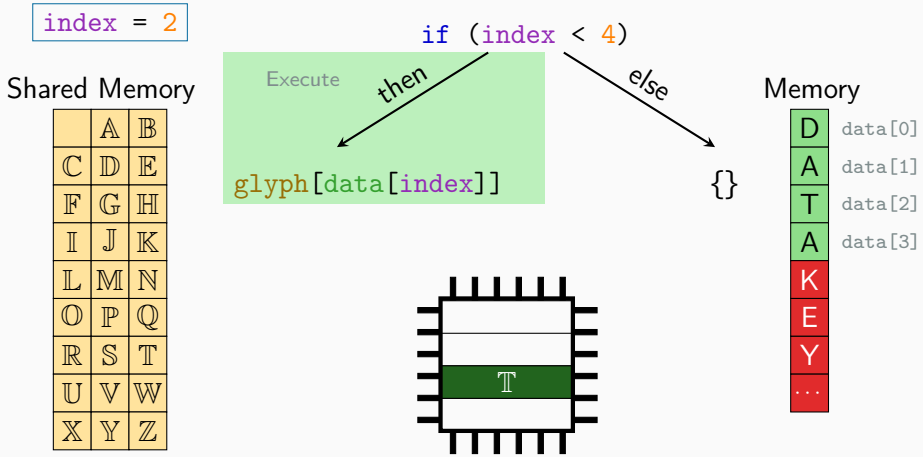
Spectre-PHT (aka Spectre Variant 1)



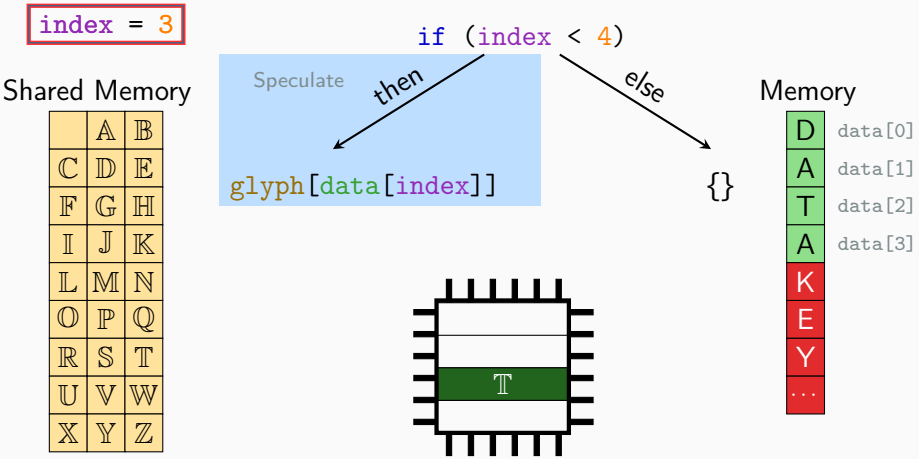
Spectre-PHT (aka Spectre Variant 1)



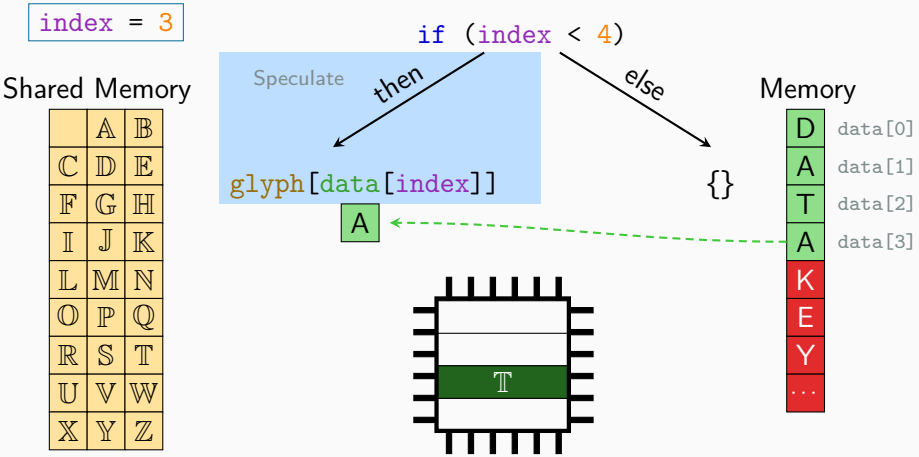
Spectre-PHT (aka Spectre Variant 1)



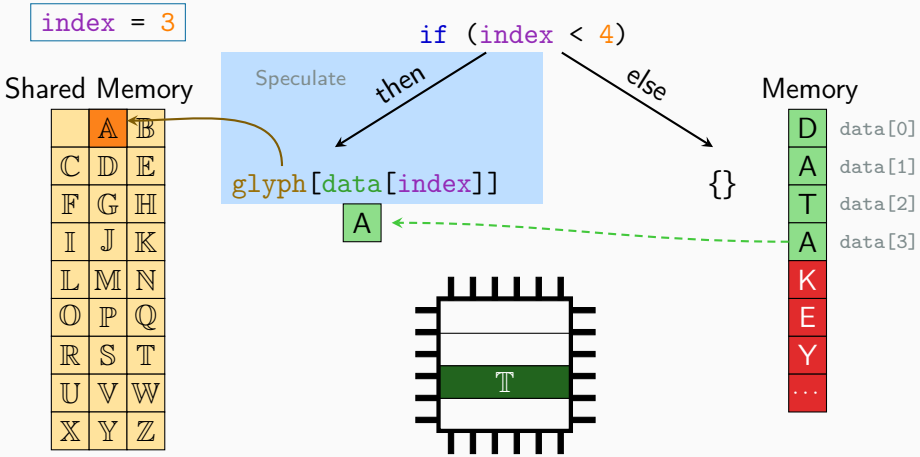
Spectre-PHT (aka Spectre Variant 1)



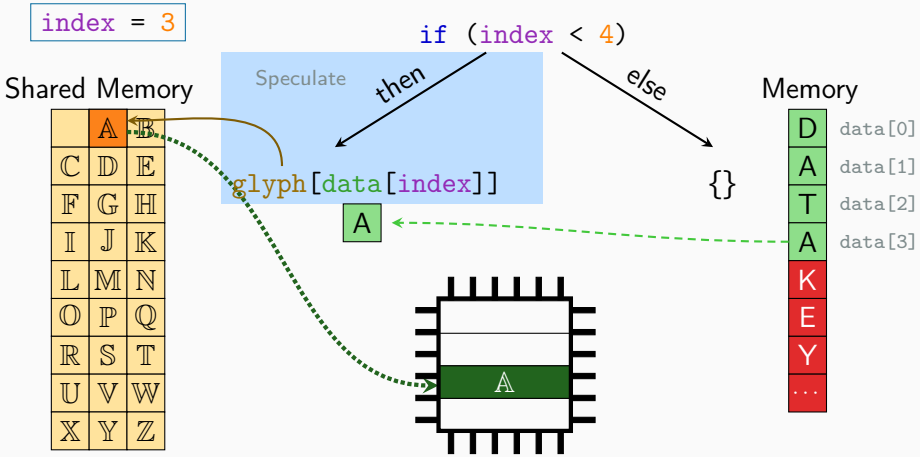
Spectre-PHT (aka Spectre Variant 1)



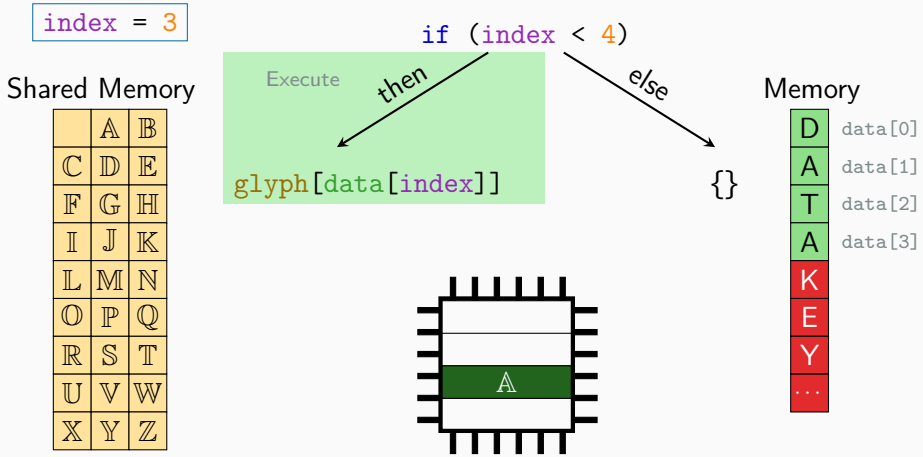
Spectre-PHT (aka Spectre Variant 1)



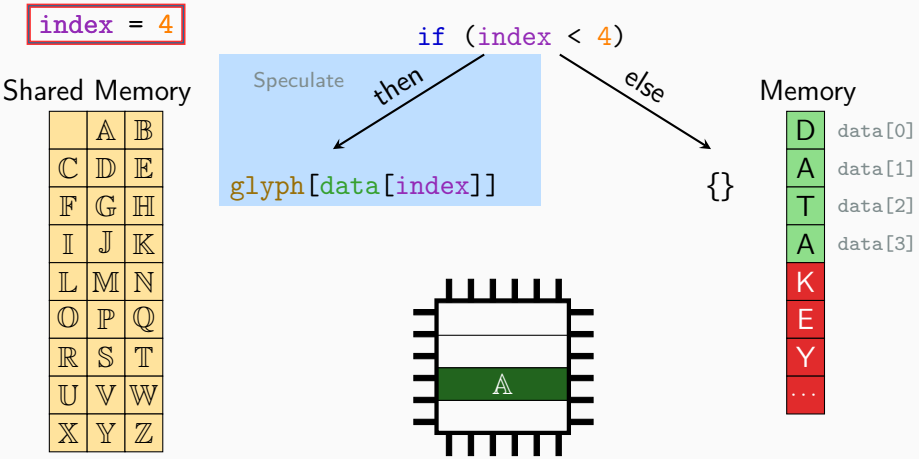
Spectre-PHT (aka Spectre Variant 1)



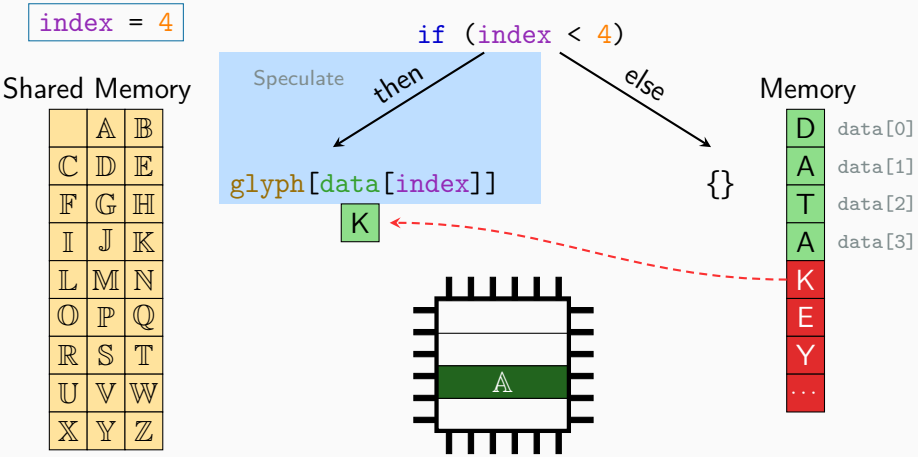
Spectre-PHT (aka Spectre Variant 1)



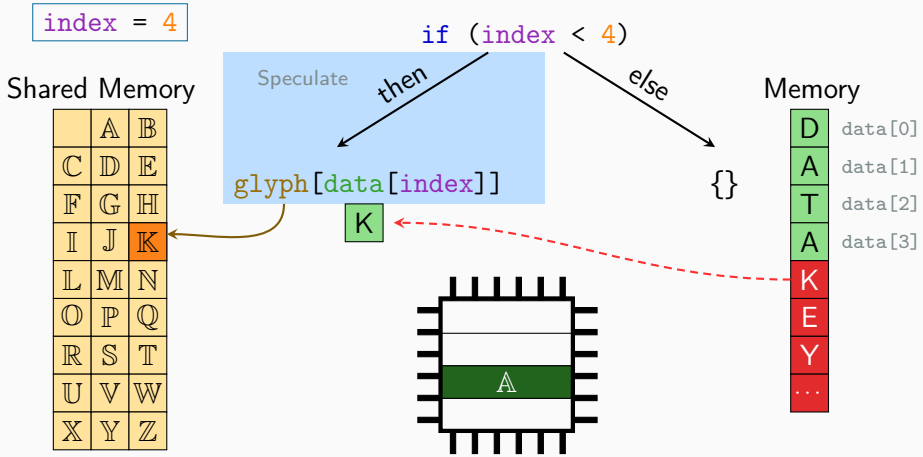
Spectre-PHT (aka Spectre Variant 1)



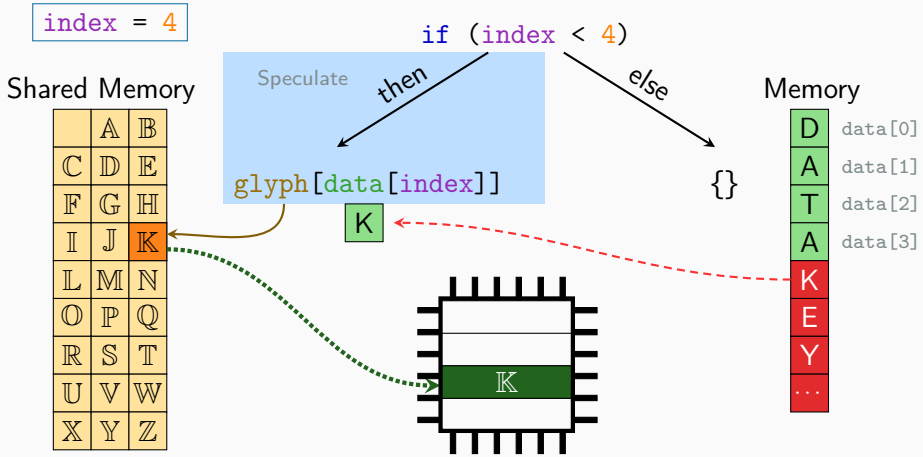
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 4
```

Shared Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

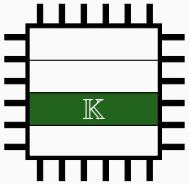
```
if (index < 4)
```

then

```
glyph[data[index]]
```

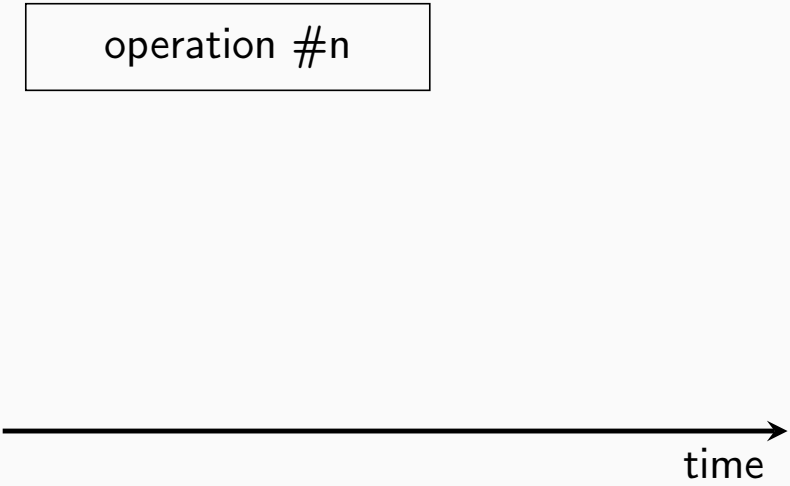
else

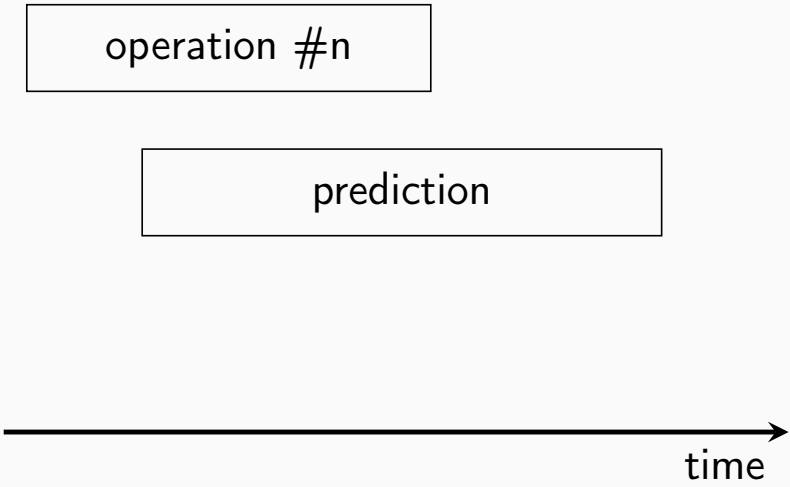
Execute {}

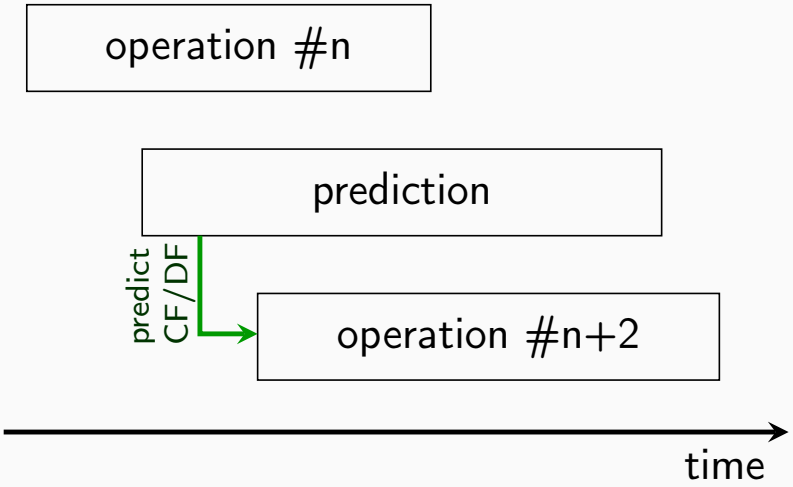


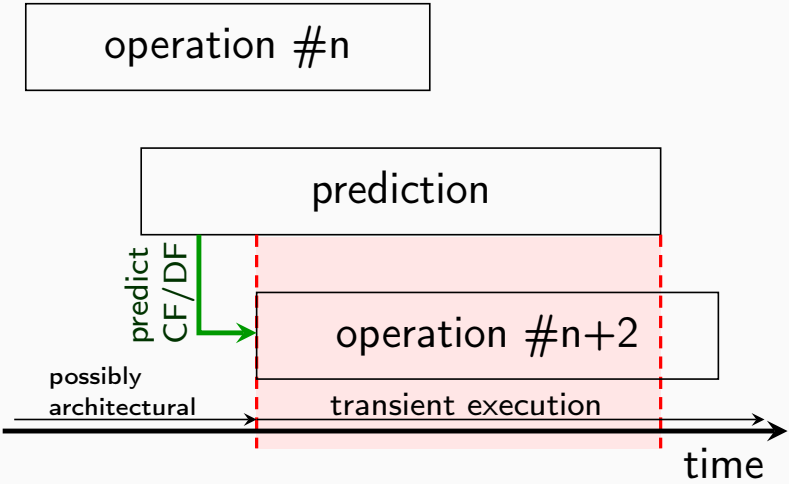
Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

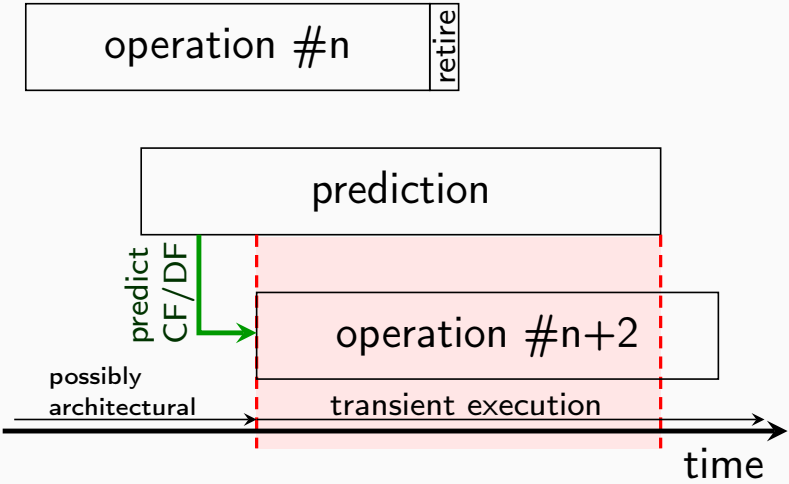

operation #n
time


operation #n
prediction
time

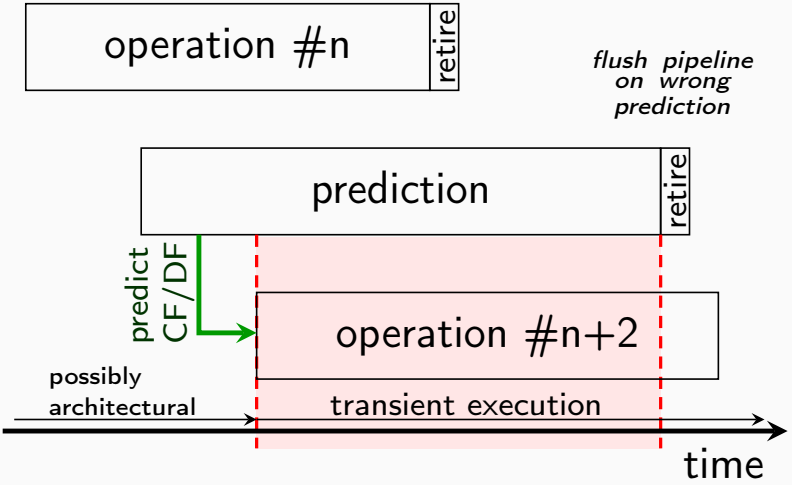

operation #n
prediction
predict CF/DF
operation #n+2
time


operation #n
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time

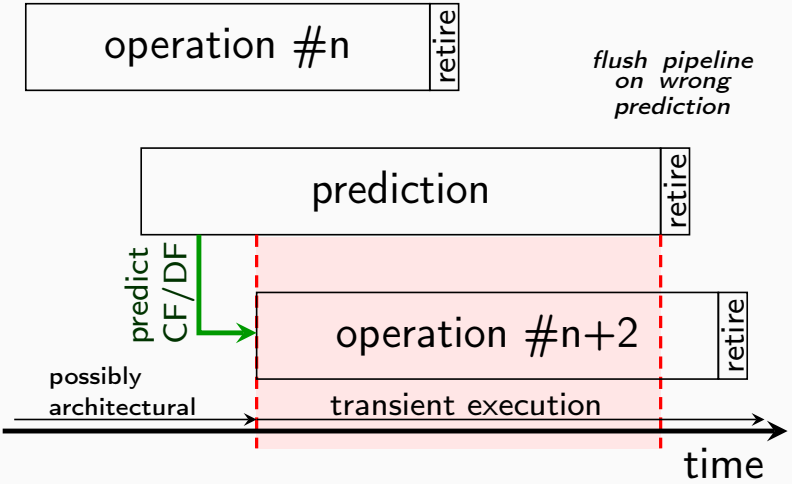
Spectre Root Cause



Spectre Root Cause



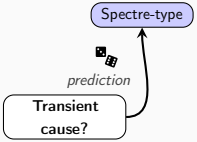
Spectre Root Cause



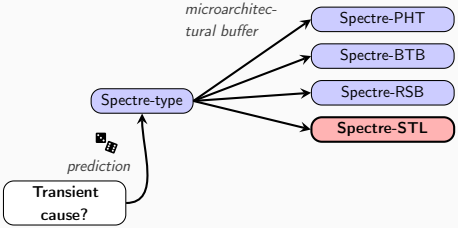
Spectre Varianten

Transient cause?

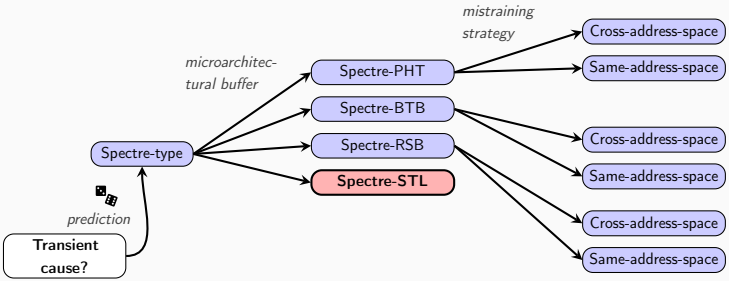
Spectre Varianten



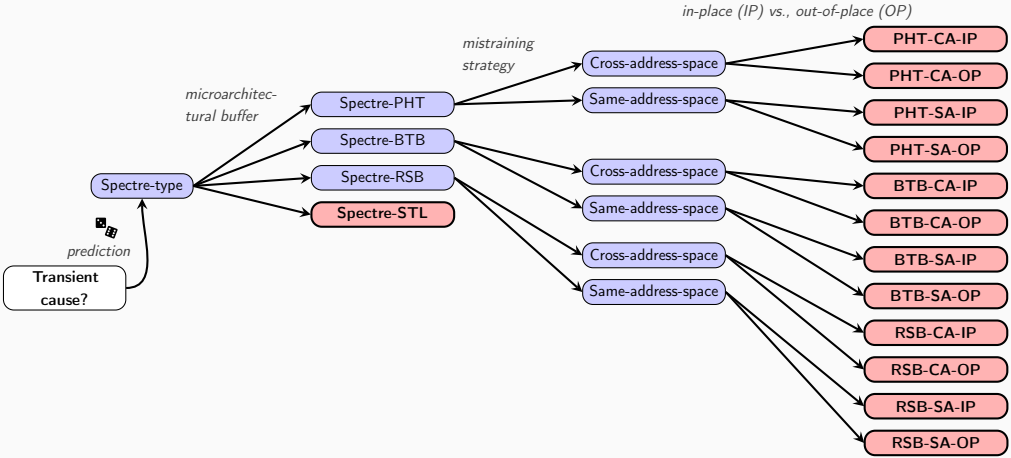
Spectre Varianten



Spectre Varianten



Spectre Varianten



Was nicht funktioniert

- Spekulative Ausführung entfernen

→ Extremer Performanceverlust

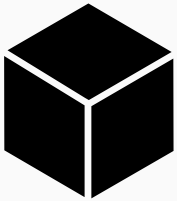
- Caches entfernen

→ Performanceverlust von Faktor 100

- Zeitmessung aus der CPU entfernen

→ Kann man sich selbst bauen

CPU als Black Box

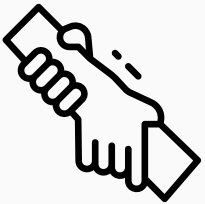


- CPU führt transparent Code aus und optimiert

→ Funktioniert für eine Sicherheitsdomäne

- Heute: verschiedene Benutzer und Sicherheitsdomänen
  - Hypervisor, Betriebssystem, (Browser) Sandbox, Trusted Execution Environment, ...
- CPU macht keine Unterscheidung in der Mikroarchitektur

Zukunft: Hardware/Software Co-Design



- Sichere Optimierungen durch Kontext
- Software kommuniziert Meta-Information mit Hardware
- Erste Ideen bereits im Einsatz: Core Scheduling
- Weitere Eregbnis in zukünftigen Prozessoren:
  - Sichere Cachearchitekturen
  - Nicht-spekulierbarer Speicher für sensible Daten

Ewiger Kreislauf?

- Katz und Maus Spiel von Angriffen und Lösungen?
- Nicht wenn wir Klassen von Problemen lösen

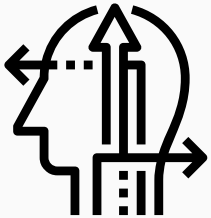
→ Spirale statt ewiger Kreis

- Angriffe wird es immer geben...
- ...aber Systeme werden insgesamt sicherer

Was lernen wir daraus?

- Software Seitenkanäle gibt es seit vielen Jahren
- Waren nie “interessant”
- Erst als wir gezeigt haben, dass man damit Daten lesen kann ($\rightarrow$ Meltdown, Spectre)
- Optimierungen führen neue Seitenkanäle ein

Eine Chance



Die Entdeckung gibt uns die Chance

- neue Prozessorarchitekturen zu entwickeln
- mehr an Sicherheit zu denken
- Kompromisse zwischen Sicherheit und Performance zu finden

Fazit

- Seitenkanal-Angriffe wurden zu lange unterschätzt
  - Grundlegende Konzepte gibt es schon lange (Zeitmessung)
- Es wird zu wenig Wert auf Sicherheit gelegt
  - Wir brauchen mehr Fokus auf Sicherheit
  - Performance darf nicht mehr das einzige Kriterium bei Prozessoren sein
- Noch viel Forschung notwendig

Unsichere Systeme trotz fehlerfreier Software?

Wie Hardware die Sicherheit von Software untergräbt

Michael Schwarz (@misc0110)

21.10.2021

CISPA Helmholtz-Zentrum für Informationssicherheit