Leolvadtunk

*

biztonsági rések a processzorokban

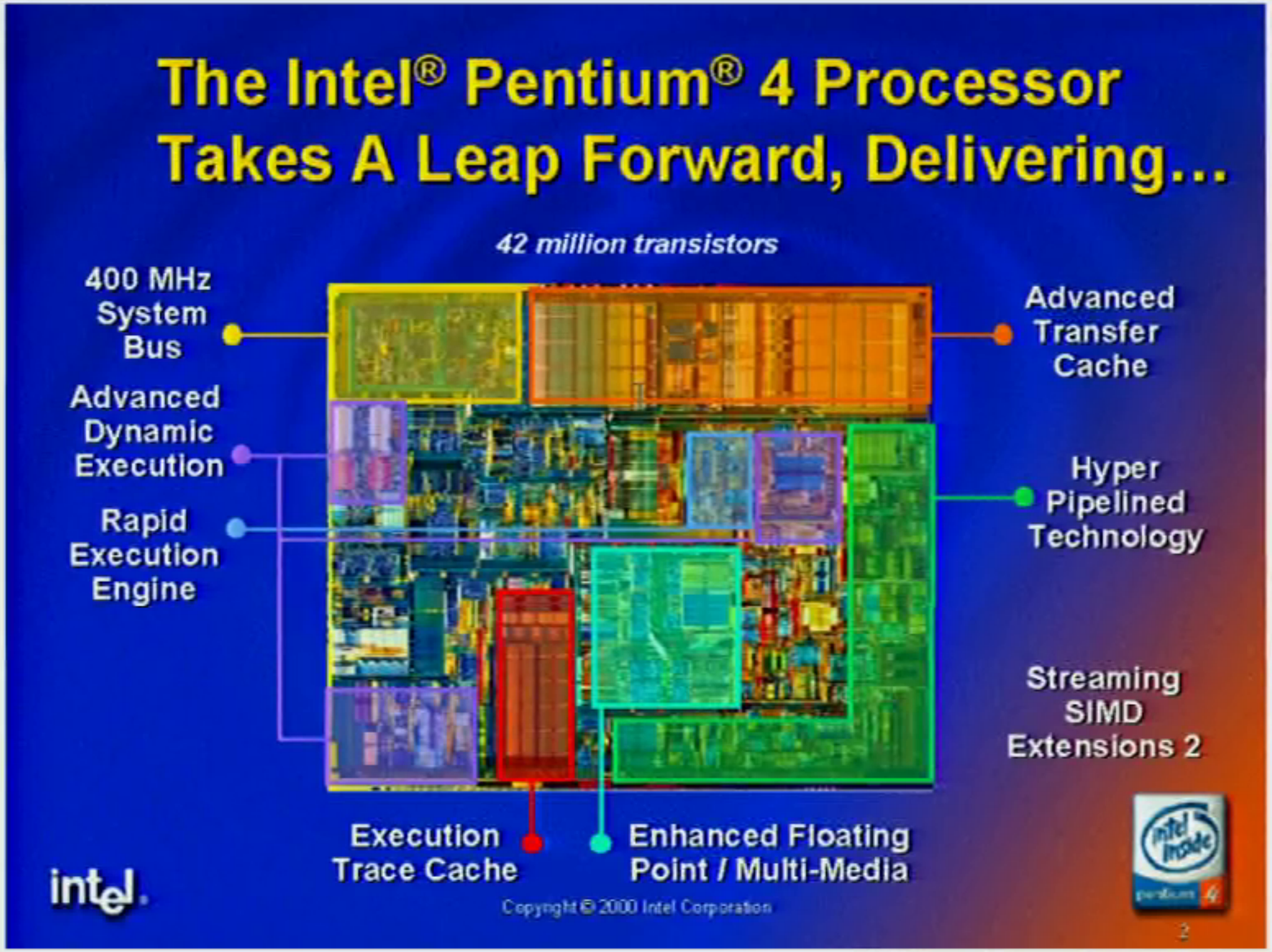
Csirmaz László

Közép-európai Egyetem
Debreceni Egyetem IK

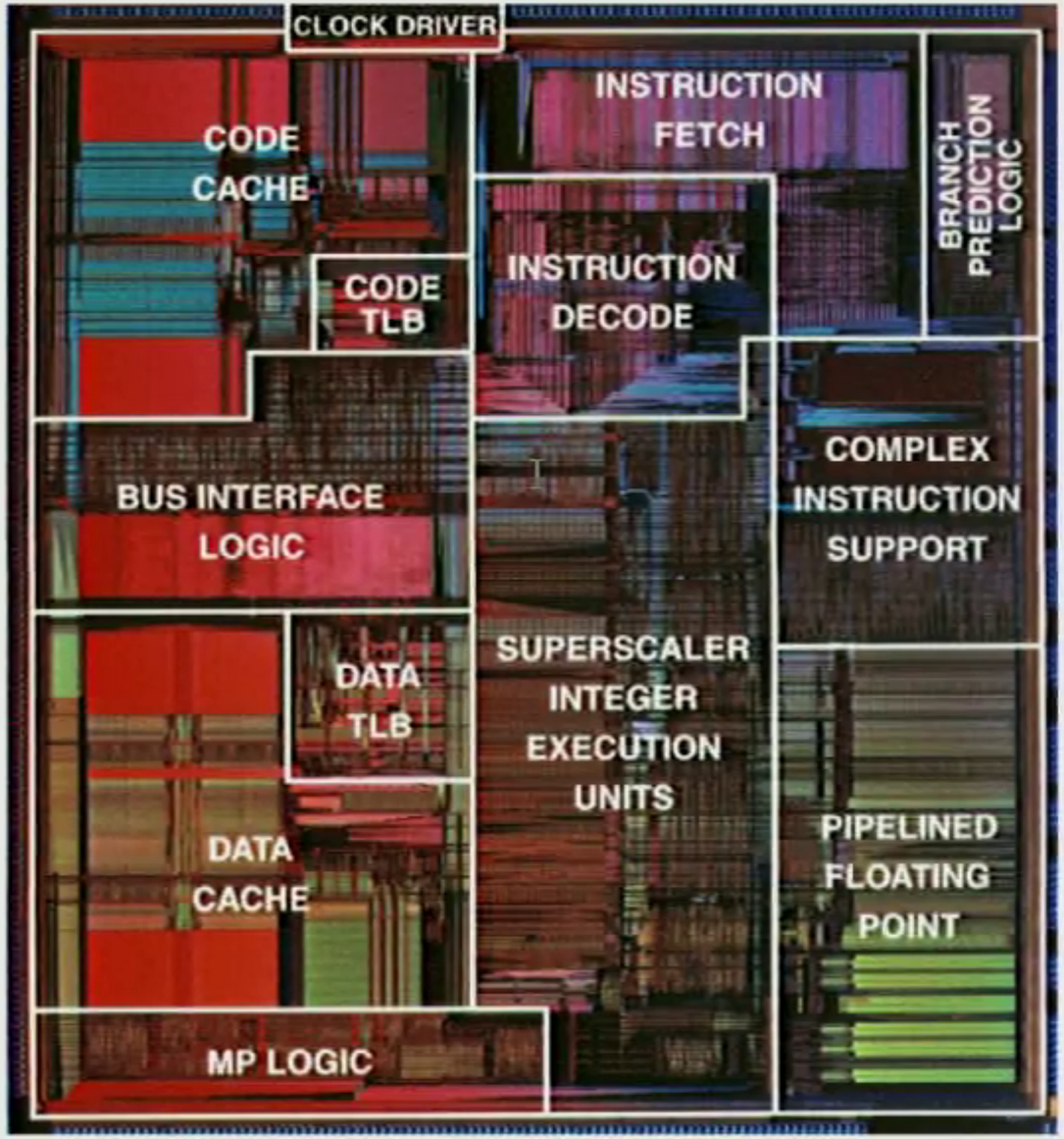
2018 április 5.

Áttekintő

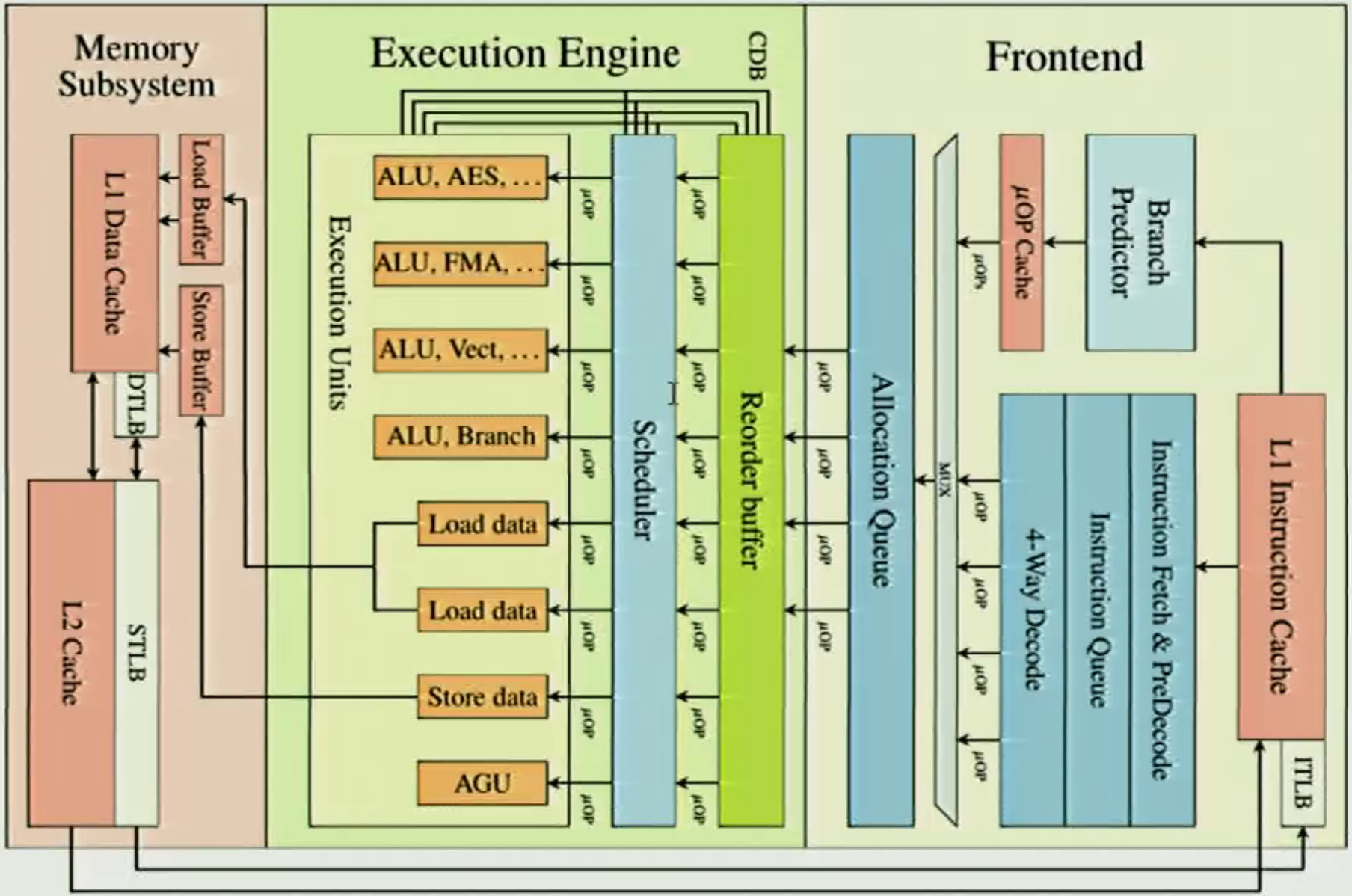
Intel pentium – 2000



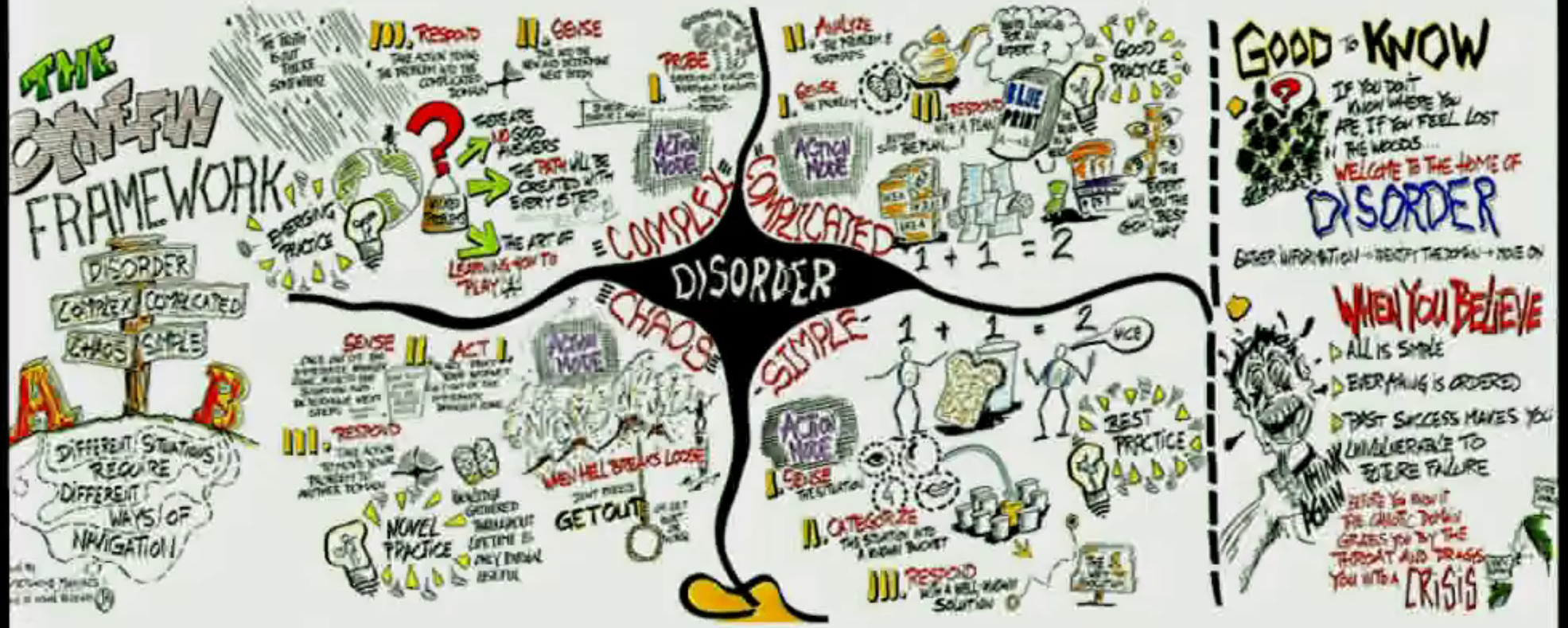
Aztán eltelt egy kis idő . . .



Nagyon vázlatosan . . .



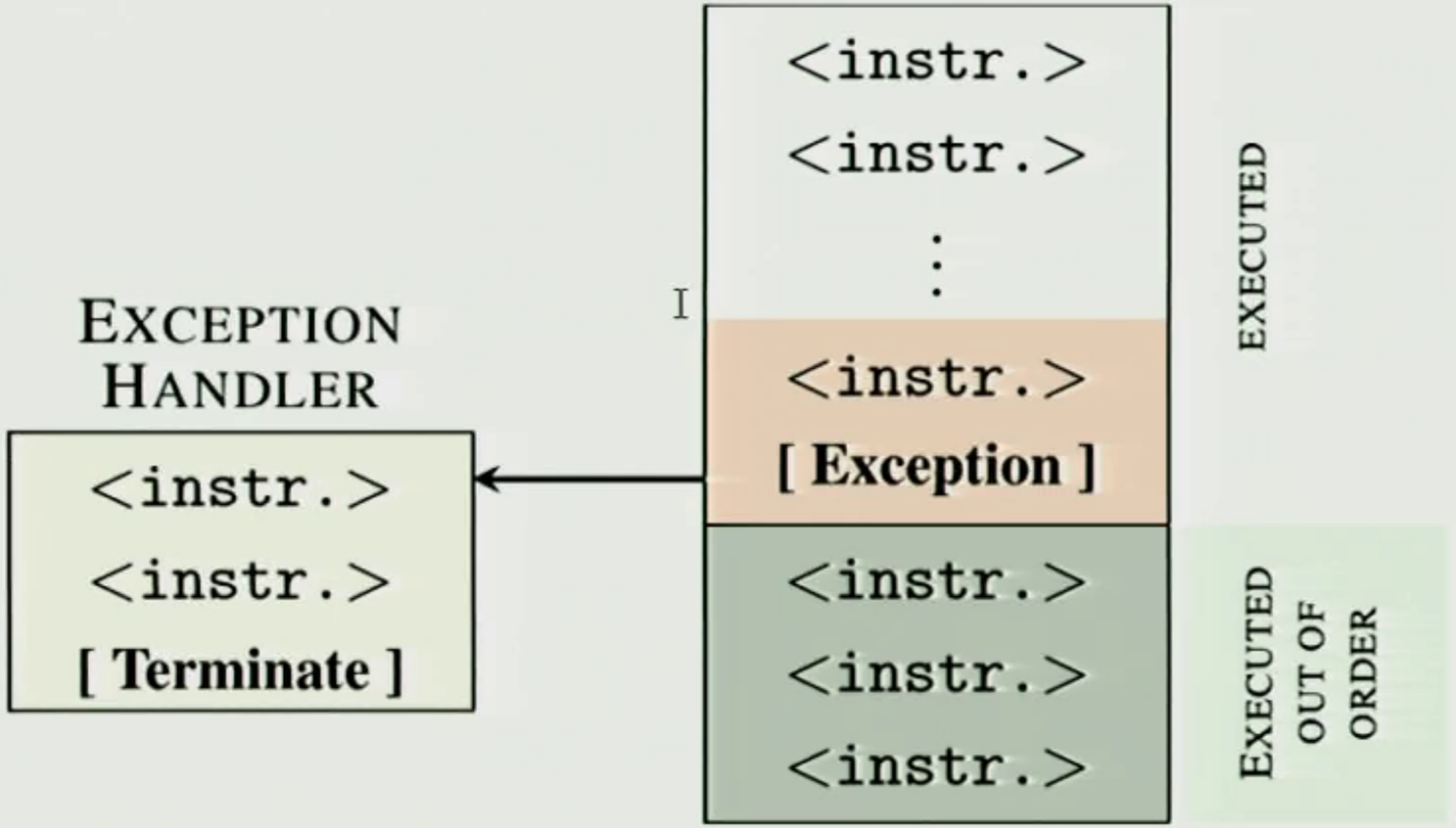
Ami bonyolult, nem lehet hibátlan



Áttekintő

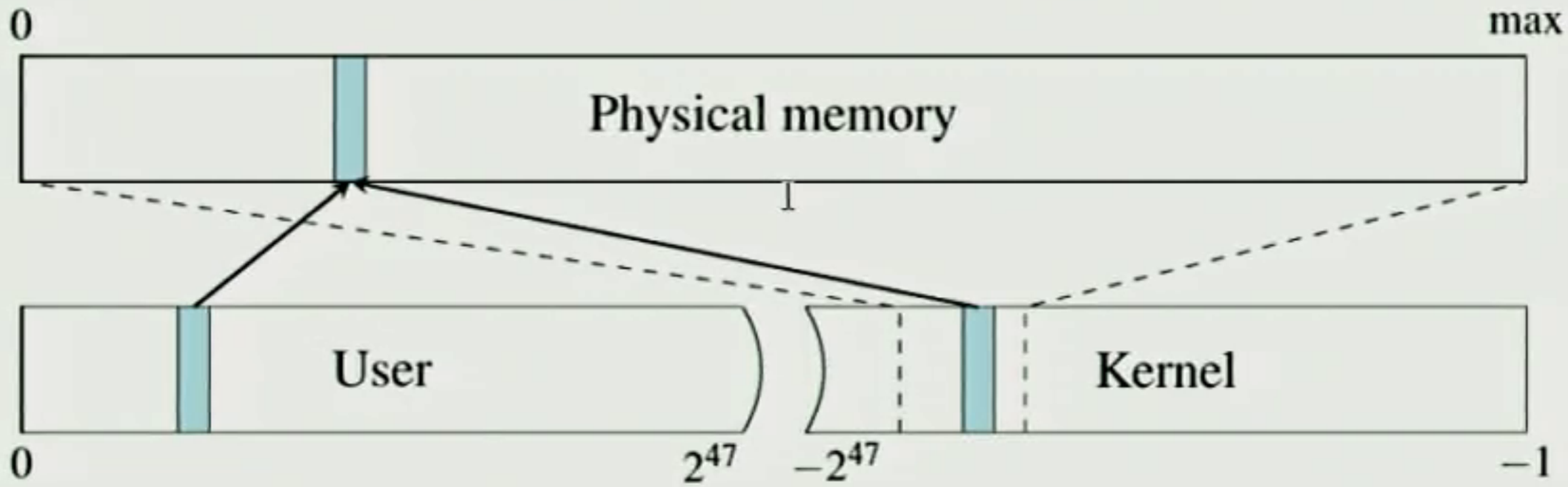
Out of order execution

A processzor gyors, a memória lassú. Dolgozzon előre, legfeljebb eldobja.



Memória felosztás

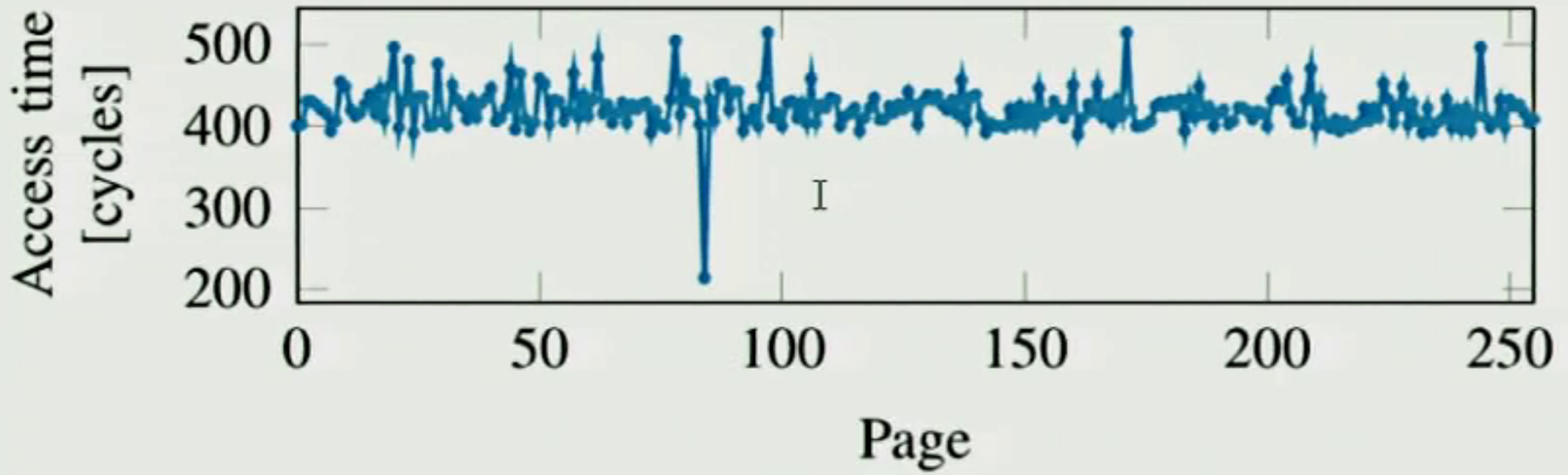
A processzor "kernel" módban látja a teljes fizikai memóriát
Ha felhasználó akar hozzáférni – megszakítás (exception)



Ám a processzor előre szalad és olvas a tiltott helyről.
Amikor kiderül, letiltja a memória módosítását, a regiszterek tartalmát visszaállítja. **Ám a mikroarchitektúra megváltozott!**

Benn van a gyorstárban?

Kimérhető, hogy a memória tartalom a gyorstárban van-e.



Itt a 84. lap a gyorstárban van, a többi 255 nincs (lassú módszer, de megbízható).

A módszer

1. Kiürítjük a gyorsítótárat például így:
   `for(i=0;i<256;i++) sum += dummy[i*4096];`
2. Kiolvassuk a `M[(*tiltott)*4096]` tömbelemet, ahol `M` egy 256*4096 méretű, általunk deklarált tömb; a `tiltott` változó értéke egy csak kernelből elérhető byte címe.
3. Megszakítást kapunk illegális hozzáférés miatt, amit lekezelünk.
4. Az előreszaladás miatt nem csak a `*tiltott` értéket olvasta ki, hanem kiszámolta az `M` indexét és kiadta az utasítást az `M[.]` tartalmának beolvasására, ami *bekerült a gyorsítótárba*.
5. Lemérjük, hogy az `M[i*4096]` elemek közül melyik van a gyorsítótárban – minthogy `M`-et mi deklarátunk, itt nincs biztonsági probléma.

Áttekintő

Leolvadtunk . . .

Ezt hajtjuk végre:

```
1 ; rcx = kernel address
2 ; rbx = probe array
3 retry:
4 mov al, byte [rcx]
5 shl rax, 0xc
6 jz retry
7 mov rbx, qword [rbx + rax]
```

4 okozza a megszakítást
5 szorozza az olvasott byte-ot 4096-tal
7 olvassa ki az `M[i*4096]`-t.
A megszakítás miatt az olvasott byte elvész, a kért tartalom nem kerül `rbx`-be – **de már benn van a gyorsítótárban!**

Olvasási sebesség

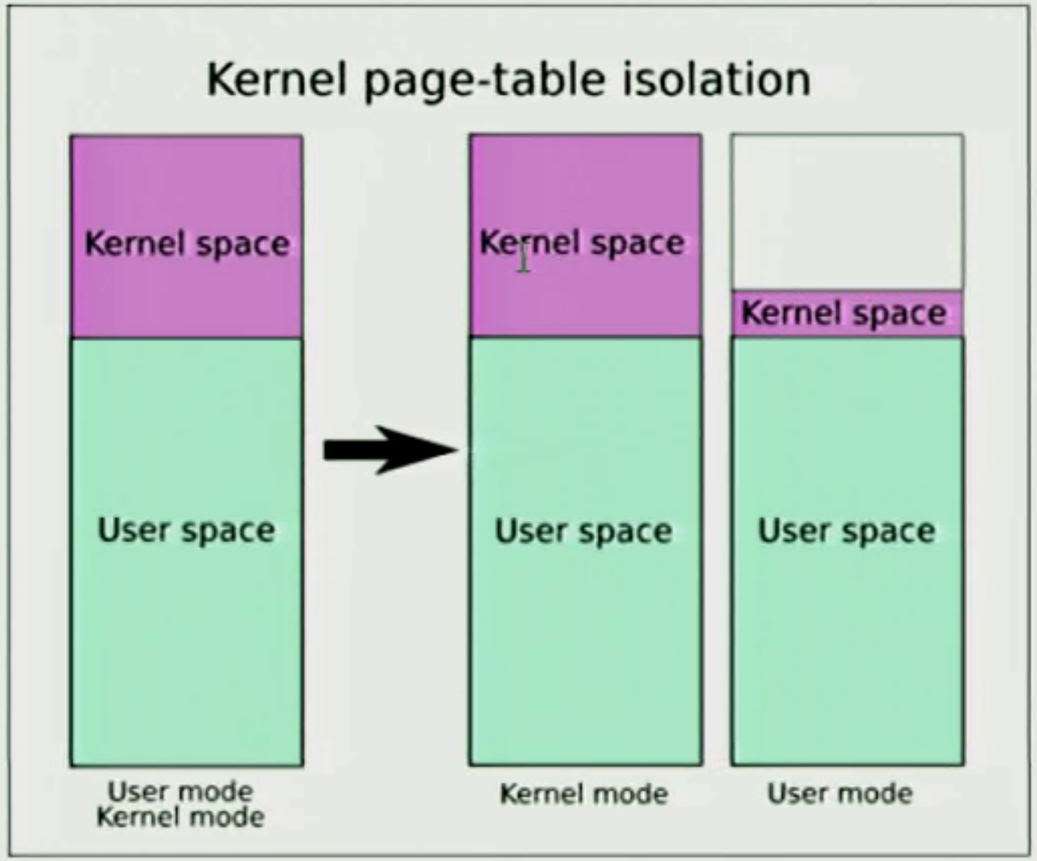
500 KB/sec a teljes fizikai memóriából,
ami a processzhez tartozó táblában szerepel

Gyakorlat: ez a tábla a *teljes memóriát tartalmazza* hogy a kernel gyors legyen.
Olvasható: pufferek, titkos kulcsok, passwordok, virtuális gép nem akadály
Különösen problémás felhőben: ki tudja ki mit futtat mellettem

Áttekintő

KAISER – Kernel Address Space Isolation

A kernel és a felhasználó másik táblát használ $\Longrightarrow$ akár 30% teljesítménycsökkenés is lehet.



Van másik . . .

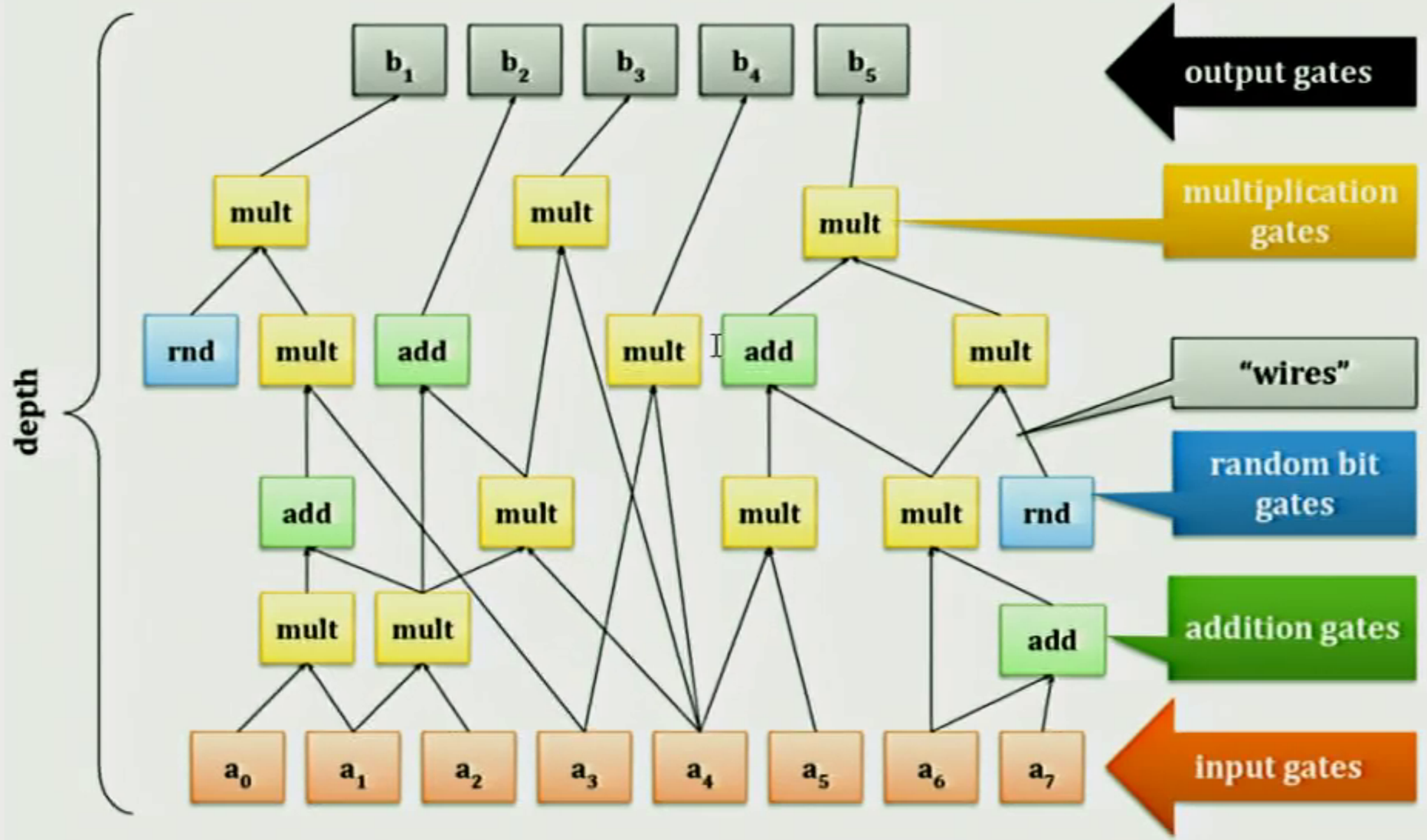
Mi van a hasonló támadásokkal, mint például a SPECTRE?

Áttekintő

Nem tudom mi a támadás, mégis tudok védekezni?

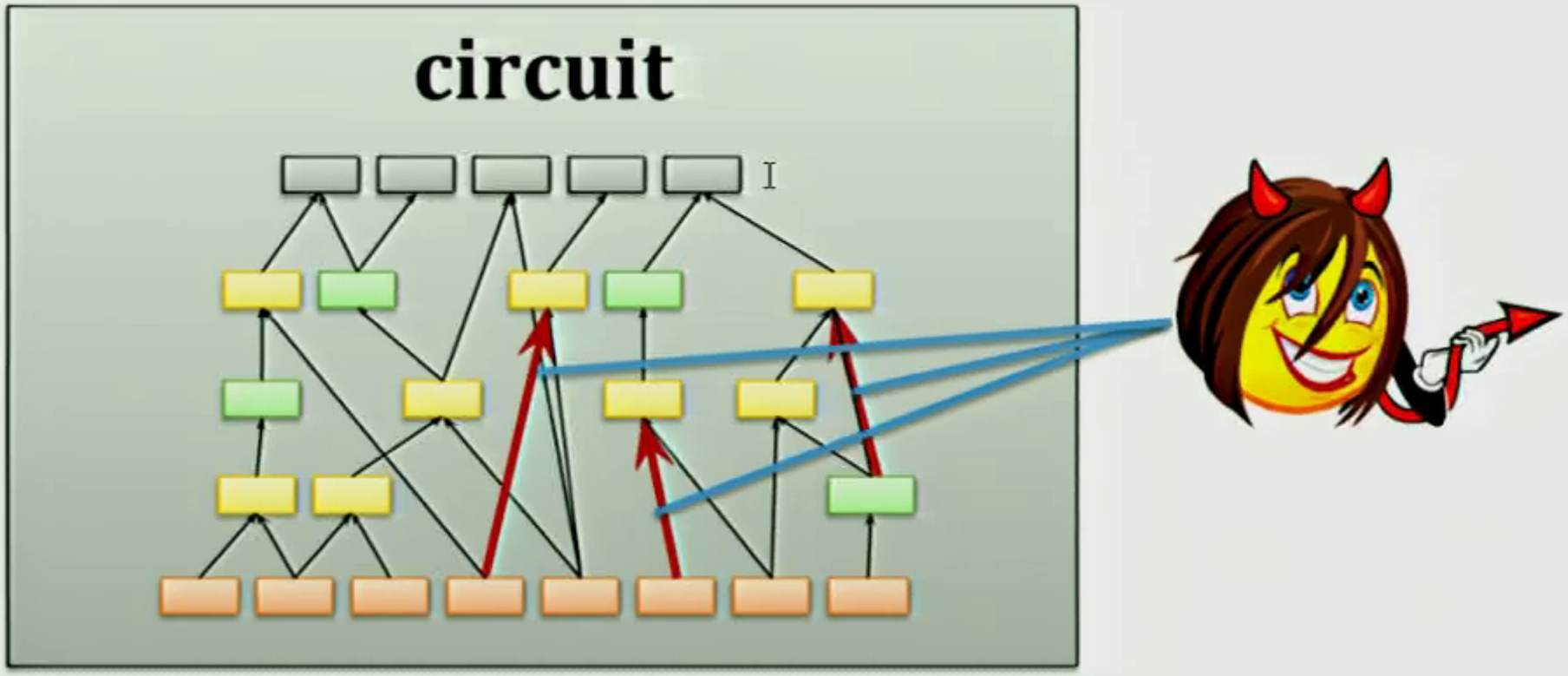
IGEN!

Hogyan modellezünk egy (kripto) processzort?



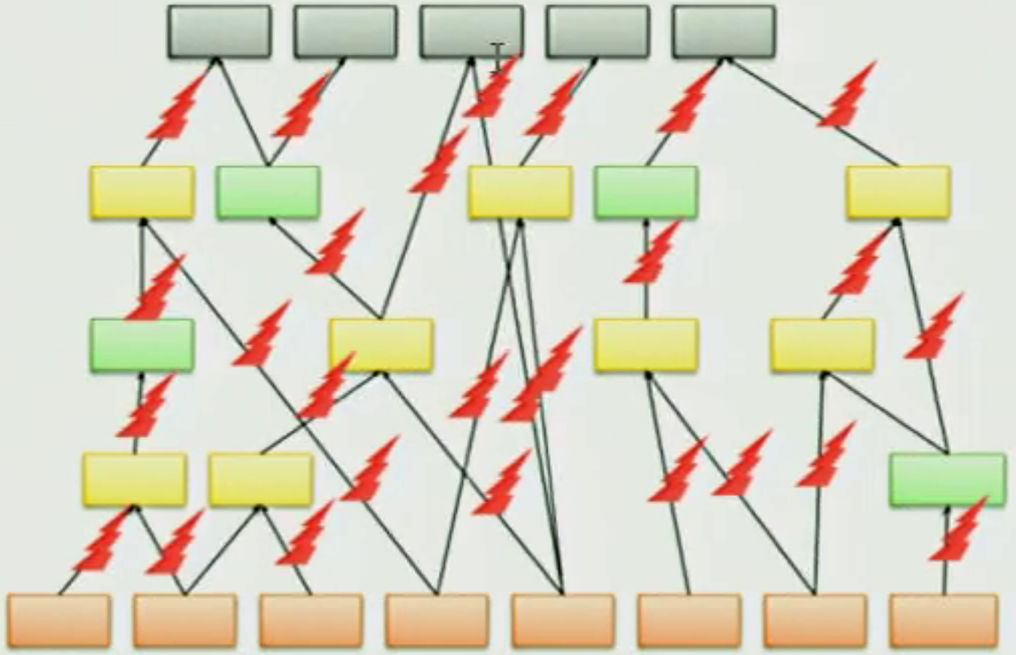
Támadás fajták – I

Minden lépésben k összeköttetést megnézhetünk, és megtudjuk, hogy folyik-e rajta áram. Minden lépésben mást választhatunk.



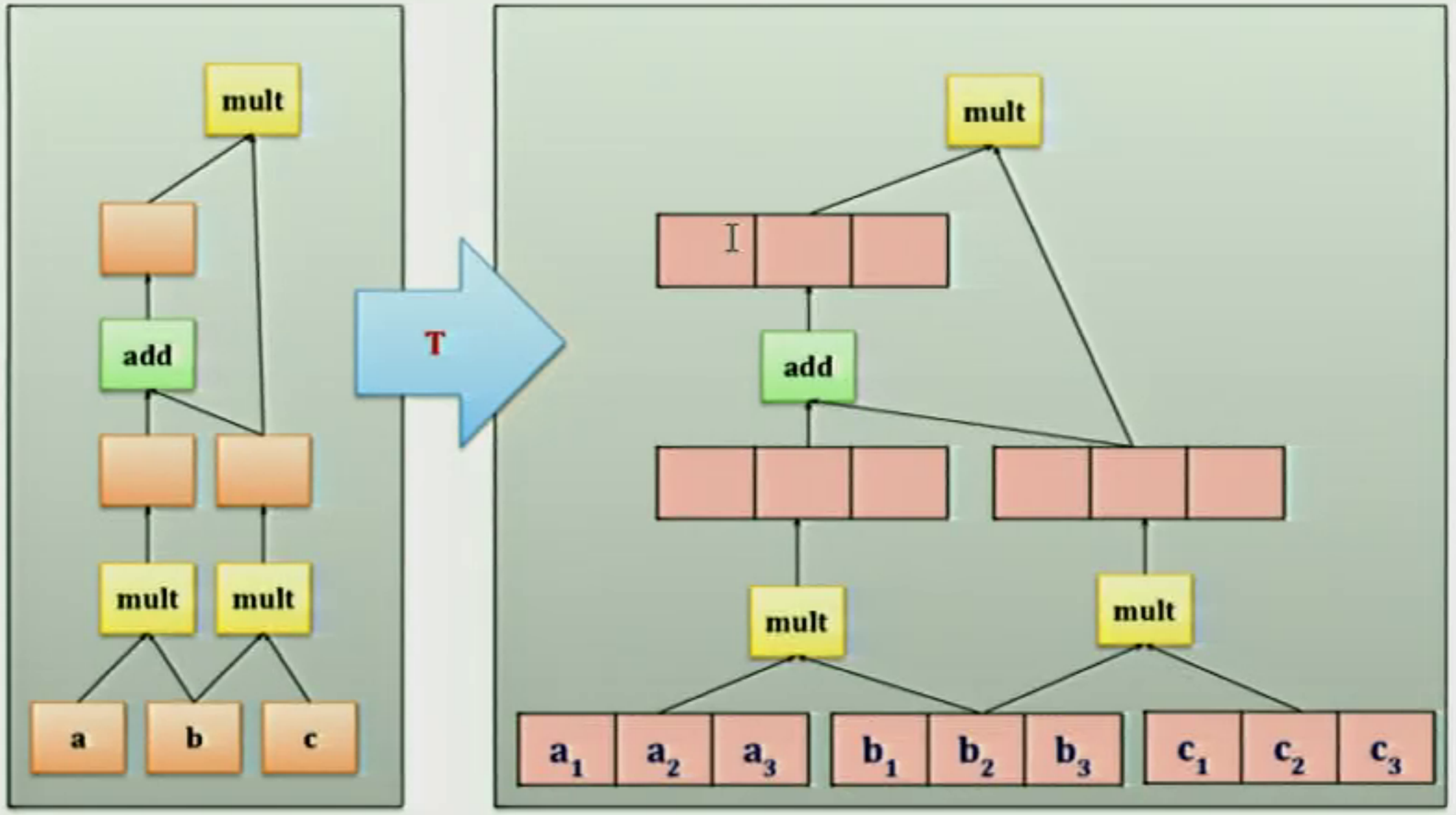
Támadás fajták – II és III

- Minden lépésben az összes összeköttetésről kapunk *zajos* információt.
- Lépésenként legljebb λ bit tetszőleges információt kaphatunk a *működő* egységekről.



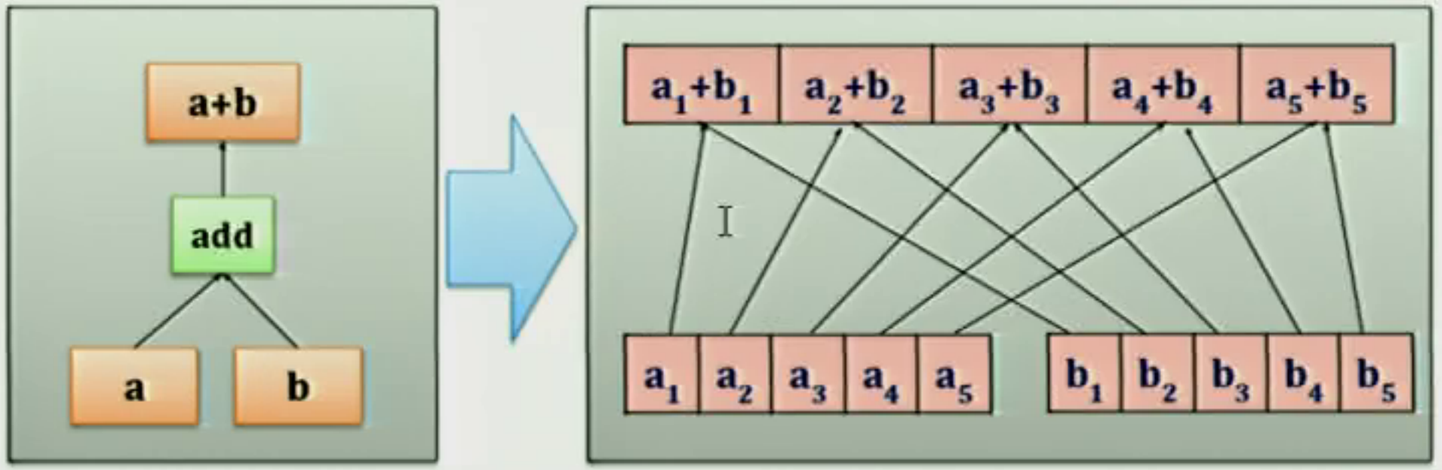
A megoldás: titokmegosztás & több résztvevős számítás

Minden értéket szétszedünk és a titokrészeken végezzük a műveleteket.



Az összeadás egyszerű

Az elrejtett érték a titokrészek mod 2 **összege** (XOR-ja).



mivel ez a titokomegosztás *lineáris*: a részeket külön-külön összeadjuk (XOR-oljuk) és kész.

Hogyan szorzunk?

Szorzásnál a disztributivitást használjuk, és véletlen értékekkel módosítjuk a részletszorzatokat.

	a_1	a_2	a_3	a_4
b_1	a_1b_1	a_2b_1	a_3b_1	a_4b_1
b_2	a_1b_2	a_2b_2	a_3b_2	a_4b_2
b_3	a_1b_3	a_2b_3	a_3b_3	a_4b_3
b_4	a_1b_4	a_2b_4	a_3b_4	a_4b_4

véletlen mátrix:

0	$-z_{12}$	$-z_{13}$	$-z_{14}$
z_{12}	0	$-z_{23}$	$-z_{24}$
z_{13}	z_{23}	0	$-z_{34}$
z_{14}	z_{24}	z_{34}	0

+

a_1b_1	a_1b_2 $-z_{12}$	a_1b_3 $-z_{13}$	a_1b_4 $-z_{14}$
a_1b_2 $+z_{12}$	a_2b_2	a_2b_3 $-z_{23}$	a_2b_4 $-z_{24}$
a_1b_3 $+z_{13}$	a_2b_3 $+z_{23}$	a_3b_3	a_3b_4 $-z_{34}$
a_1b_4 $+z_{14}$	a_2b_4 $+z_{24}$	a_3b_4 $+z_{34}$	a_4b_4
c_1	c_2	c_3	c_4

a szorzat titokrészei az **oszlopösszegek** lesznek.

Áttekintő

Ahonnan a képek is vannak

Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Baniel Genkin, Yuval Yarom, Mike Hamburg:
Meltdown, `https://arxiv.org/pdf/1801.01207`

Specter attacks, `https://arxiv.org/pdf/1801.01203`

Meltdown and Spectre, `https://meltdownattack.com/`

Stefan Dziembowski: **Modelling Side-Channel Leakage**,
`http://www.crypto.edu.pl/Dziembowski/talks`

Krzysztof Pietrzak: **A leakage-resilient mode of operation**
`https://users-cs.au.dk/stm/local-cache/leak-mode.pdf`

Köszönöm a figyelmet!