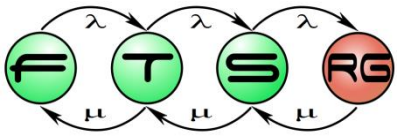
Meltdown: Egy sebezhetőség modellezése időzített automatákkal

Marussy Kristóf



Budapesti Műszaki és Gazdaságtudományi Egyetem
Hibatűrő Rendszerek Kutatócsoport

Meltdown és Spectre

- Kritikus sebezhetőségek a modern processzorokban
  - Kernel szintű izoláció megkerülése
  - Kihasználás akár JavaScript kódból?
- Meltdown
  - Kernel memória olvasása
  - Spekulatív utasításvégrehajtás
- Spectre (1-es és 2-es változat)
  - Bounds check bypass
  - Branch target injection

https://meltdownattack.com/

„Történelem”

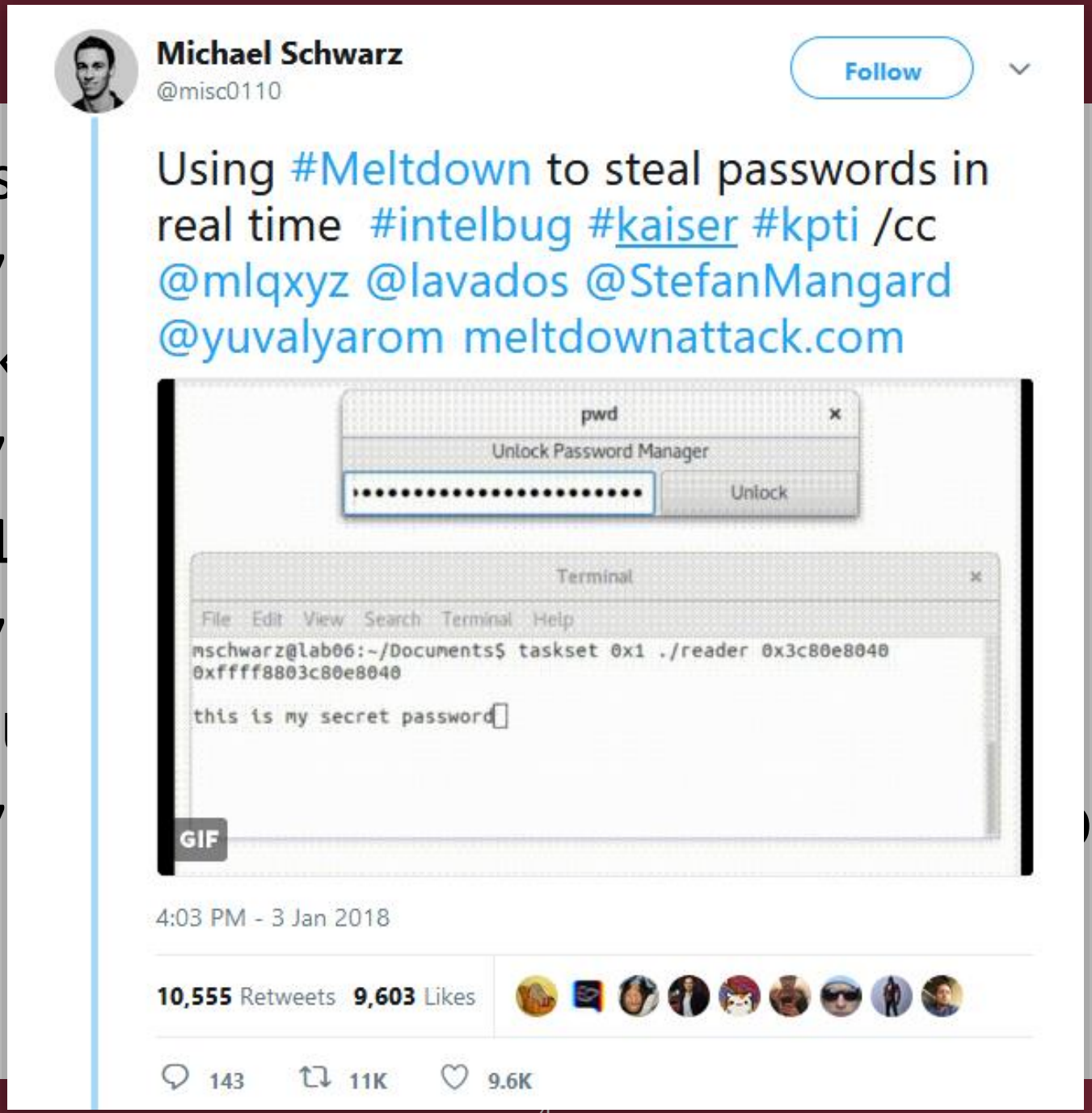
A teljesség igénye nélkül:

- 2017. jún. 1. – Google Project Zero jelenti a hibákat többek között az Intelnek, embargó
- 2017. nov. 7–27. – KAISER patchsorozat az LKML Linux kernelfejlesztői levelezőlistán
- 2017. dec. 4. – `kpti` patch az LKML-en
  - `CPU_BUG_INSECURE` – megindul a spekuláció
- 2017. dec. 27. – do not enable `pti` on AMD patch

A teljes

- 2017
  hibák
- 2017
  LKML
- 2017
  - CP
- 2017

patch



„Történelem”

A teljesség igénye nélkül:

- 2017. jún. 1. – Google Project Zero jelenti a hibákat többek között az Intelnek, embargó
- 2017. nov. 7–27. – KAISER patchsorozat az LKML Linux kernelfejlesztői levelezőlistán
- 2017. dec. 4. – `kpti` patch az LKML-en
  - `CPU_BUG_INSECURE` – megindul a spekuláció
- 2017. dec. 27. – do not enable `pti` on AMD patch
- 2017. jan. 3. – Michael Schwarz twitter üzenete, Graz-i Egyetem és Project Zero publikál

Publikációk

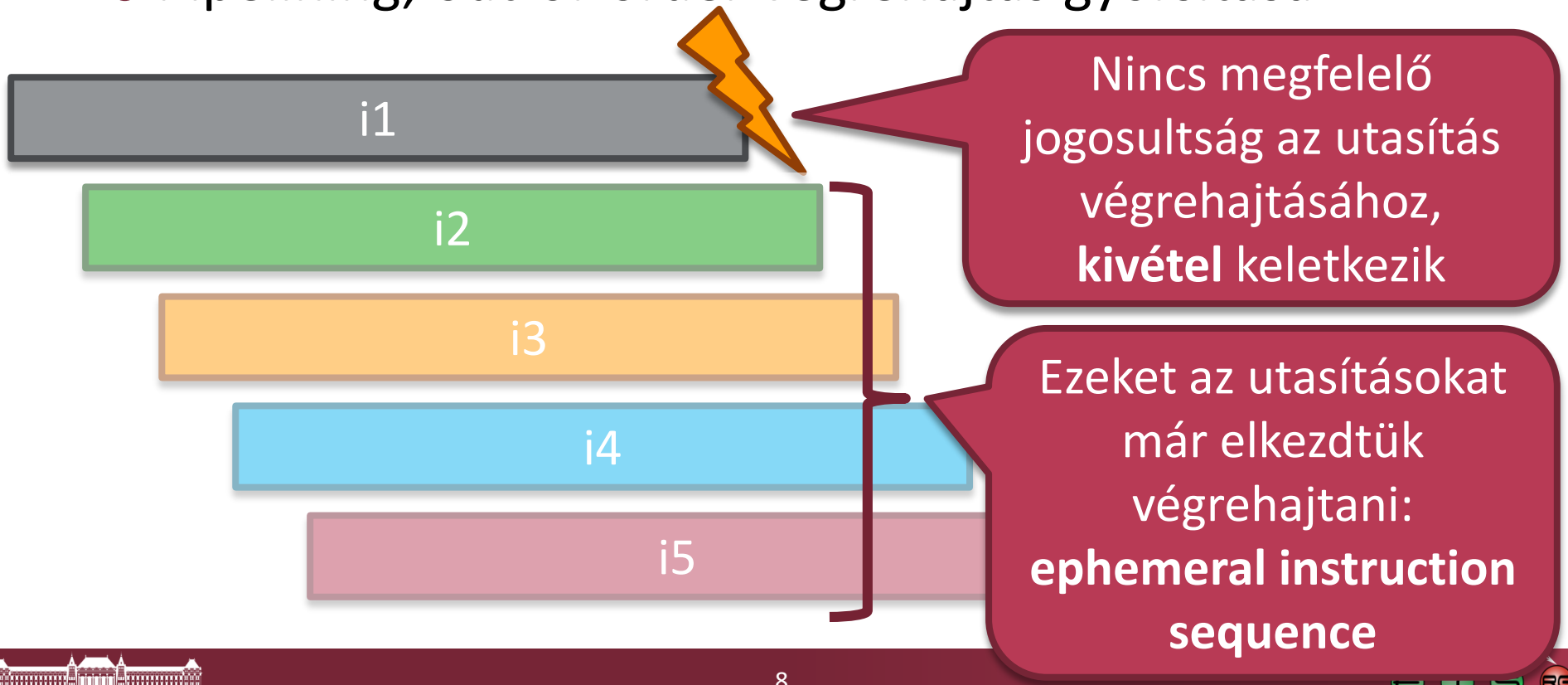
- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg (2018). *Meltdown.* arXiv:1801.01207
- Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom (2018). *Spectre Attacks: Exploiting Speculative Execution.* arXiv:1801.01203
- Jann Horn, Project Zero (2018). Reading privileged memory with a side-channel. [Online] URL: https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

A TÁMADÁS

Mit és miért szeretnénk modellezni?

Spekulatív utasításvégrehajtás

- Lásd: számítógép architektúrák
  - A CPU elkezd olyan utasításokat végrehajtani, amikre lehet, hogy nem lesz szükség
  - Pipelining, out-of-order végrehajtás gyorsítása



Fedett csatorna

- Az *ephemeral* utasításoknak nincs a regiszterekben vagy a memóriában megfigyelhető mellékhatása
  - Hiába tudjuk megkerülni a jogosultságkezelést, ha a jogosulatlanul olvasott adatot nem tudjuk kivinni

Fedett csatorna

- Az
  a
  m
  - a

Film: Steven Spielberg (2015). Bridge of Spies.

Fedett csatorna

- A
  a
  m
  - a

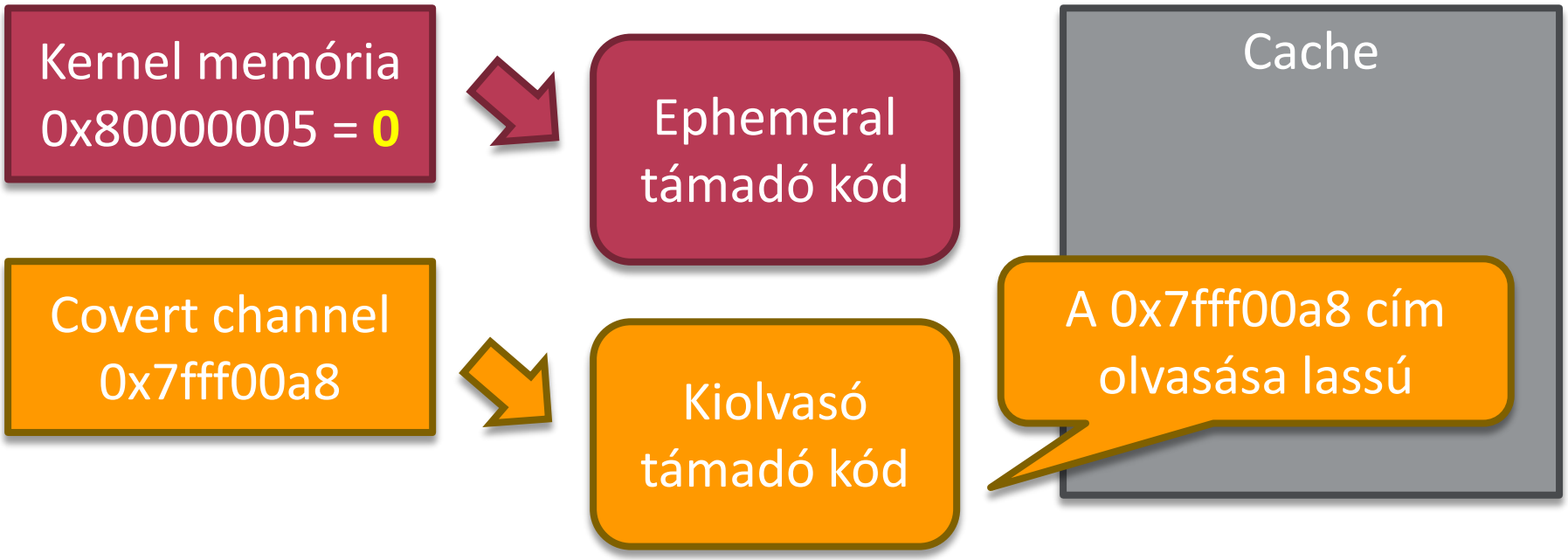
Film: Steven Spielberg (2015). Bridge of Spies.

Fedett csatorna

- Az *ephemeral* utasításoknak nincs a regiszterekben vagy a memóriában megfigyelhető mellékhatása
  - Hiába tudjuk megkerülni a jogosultságkezelést, ha a jogosulatlanul olvasott adatot nem tudjuk kivinni
- De megoldás: fedett csatorna (covert channel)
  - Információátviteli mód, ami kívülről nem érzékelhető
  - Eredetileg nem is információátvitelre szolgál
  - Processzorban: **mikroarchitekturális fedett csatorna**

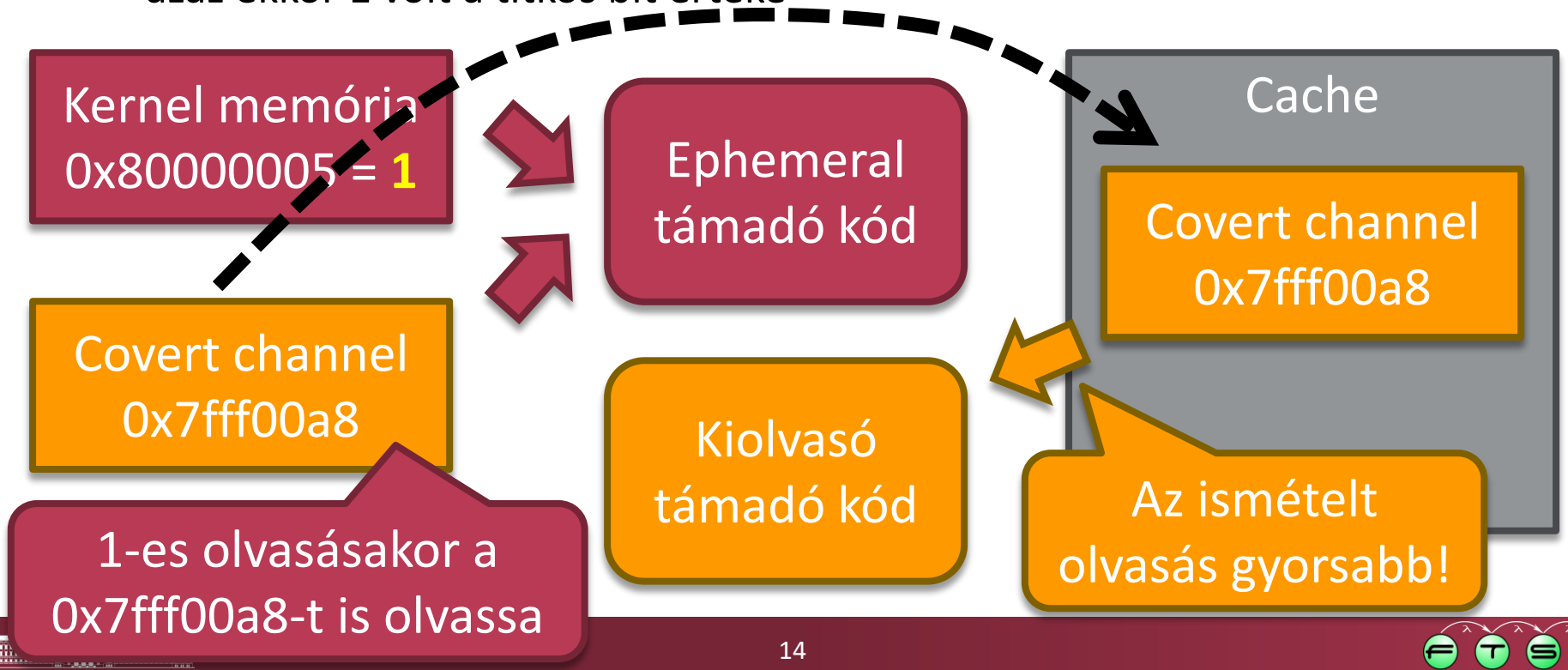
Adatátvitel a cache-ben

- Csatorna (covert channel): egy előre meghatározott memóriacím
  - Támadó kódnak van jogosultsága olvasni, kezdetben nincs a cache-ben
  - Ephemeral támadó kód olvassa, ha a titkos bit (kernel memória) értéke 1; ekkor a cache-be kerül (egyébként nem)
- Timing side-channel: a kiolvasó támadó kód időt mér
  - Ha a csatorna bit gyorsan olvasható, akkor a cache-ben van, azaz ekkor 1 volt a titkos bit értéke



Adatátvitel a cache-ben

- Csatorna (covert channel): egy előre meghatározott memóriacím
  - Támadó kódnak van jogosultsága olvasni, kezdetben nincs a cache-ben
  - Ephemeral támadó kód olvassa, ha a titkos bit (kernel memória) értéke 1; ekkor a cache-be kerül (egyébként nem)
- Timing side-channel: a kiolvasó támadó kód időt mér
  - Ha a csatorna bit gyorsan olvasható, akkor a cache-ben van, azaz ekkor 1 volt a titkos bit értéke



Mire elég az idő?

- **Kihívás:** Támadás megvalósítása ephemeral instruction sequence formájában

6.4 Limitations on ARM and AMD

We also tried to reproduce the Meltdown bug on several ARM and AMD CPUs. However, we did not manage to successfully leak kernel memory with the attack described in Section 5, neither on ARM nor on AMD. The reasons for this can be manifold. First of all, our implementation might simply be too slow and a more optimized version might succeed. For instance, a more shallow out-of-order execution pipeline could tip the race condition towards against the data leakage. Similarly, if the processor lacks certain features, e.g., no re-order
lementation might not be able to
or both ARM and AMD, the toy
in Section 3 works reliably, indi-
r execution generally occurs and
al memory accesses are also performed.

Vizsgáljuk meg, milyenek lehetnek egy alkalmas ephemeral támadó kód időzítési viszonyai **időzített automata** segítségével!

EGYSZERŰSÍTÉSEK, ABSZTRAKCIÓK

Mekkora részét modellezük a problémának?

Egyszerű processzormodell

- Csak egy 3 lépcsős pipeline-t tételezünk fel, out-of-order végrehajtás nélkül

- Egy utasítás csak ebben a sorrendben dolgozható fel, és minden műveletvégző egység sorban dolgozza fel az utasításokat
- Az utasítások közötti adatfüggőségeket elhanyagoljuk

Absztrakció

- Nem modellezük a processzor hardverszintű belső állapotát (nem hardver-modellellenőrzés)
- Óraváltozók
  - Hány órajele kezdődött el egy tevékenység?
  - Végrehajtási idők megadása invariánsokkal és őrfeltételekkel
  - UPPAAL időzített automata (XTA) formalizmus
- Cache helyett: csak logikai változó
  - Benne van-e a fedett csatorna memóriaterülete a cache-ben?

Támadó kód felépítése

	Utasítás	Dekódolás	Végrehajtás	Ellenőrzés
1.	Kernel memória olvasás	1 órajel	45-120 órajel	40-100 órajel
2 … N – 2.	Számítás	1 órajel	5-10 órajel	5 órajel
N – 1.	Fedett csatorna cache-be	1 órajel	45-120 órajel	10-25 órajel
Ide ugrik a végrehajtás kivétel keletkezése esetén:				
N.	Fedett csatorna olvasás	1 órajel	45-120 vagy 15-30 órajel	15-25 órajel

- Nagy végrehajtási idők,
  hogy könnyen szemléltethessük a jelenséget
- Változó hosszú utasítássorozat (N paraméter)
  - Vizsgálat: Mennyi számításra van elég idő?

Támadó kód felépítése

	Utasítás	Dekódolás	Végrehajtás	Ellenőrzés
1.	Kernel memória olvasás	1 órajel	45-120 órajel	40-100 órajel
2 … N – 2.	Számítás	1 órajel	5-10 órajel	5 órajel
N – 1.	Fedett csatorna cache-		jel	10-25 órajel
Ide ugrik a				
N.	Fedett csatorna olvasás	1 órajel	45-120 vagy 15-30 órajel	15-25 órajel

Privilegizált memóriát olvasó utasítás:
Az ellenőrzés kivételt dob

- Nagy végrehajtási idők,
  hogy könnyen szemléltethessük a jelenséget
- Változó hosszú utasítássorozat (N paraméter)
  - Vizsgálat: Mennyi számításra van elég idő?

Támadó kód felépítése

	Utasítás	Dekódolás	Végrehajtás	Ellenőrzés
1.	Kernel memória olvasás	1 órajel	45-120 órajel	40-100 órajel
2 … N – 2.	Számítás	1 órajel	5-10 órajel	5 órajel
N – 1.	Fed	1 órajel	45-120 órajel	10-25 órajel
		vétel keletkezése esetén:		
N.	Fe	1 órajel	45-120 vagy 15-30 órajel	15-25 órajel

Ephemeral utasítások:
A fedett csatornához szükséges számítások

- Nagy végrehajtási idők,
  hogy könnyen szemléltethessük a jelenséget
- Változó hosszú utasítássorozat (N paraméter)
  - Vizsgálat: Mennyi számításra van elég idő?

Támadó kód felépítése

	Utasítás	Dekódolás	Végrehajtás	Ellenőrzés
1.	Kernel memória olvasás	1 órajel	45-120 órajel	40-100 órajel
2 … N – 2.	Számítás	1 órajel	5-10 órajel	5 órajel
N – 1.	Fedett csatorna cache-be	1 órajel	45-120 órajel	10-25 órajel
Ide ugrik a vé kivétel keletkezése esetén:				
N.	Fedett csatorna		vagy ajel	15-25 órajel

Behozza a csatorna memóriaterületet a cache-be, ha a számítás szerint az olvasott bit 1

- Nagy végreha
  hogy könnyen szemléltethessük a jelenséget
- Változó hosszú utasítássorozat (N paraméter)
  - Vizsgálat: Mennyi számításra van elég idő?

Támadó kód felépítése

	Utasítás	Dekódolás	Végrehajtás	Ellenőrzés
1.	Kernel memória olvasás		120 órajel	40-100 órajel
2 … N – 2.	Számítás		10 órajel	5 órajel
N – 1.	Fedett csatorna cache-		20 órajel	10-25 órajel
Ide ugrik a végre… kivétel keletkezése esetén:				
N.	Fedett csatorna olvasás	1 órajel	45-120 vagy 15-30 órajel	15-25 órajel

Csatorna kiolvasása időméréssel a kivétel keletkezése után

- Nagy végrehajtási idők, hogy könnyen szemlélt
- Változó hosszú utasítás

Attól függ, benne van-e a csatorna memóriaterület a cache-ben

  - Vizsgálat: Mennyi számításra van elég idő?

A MODELL

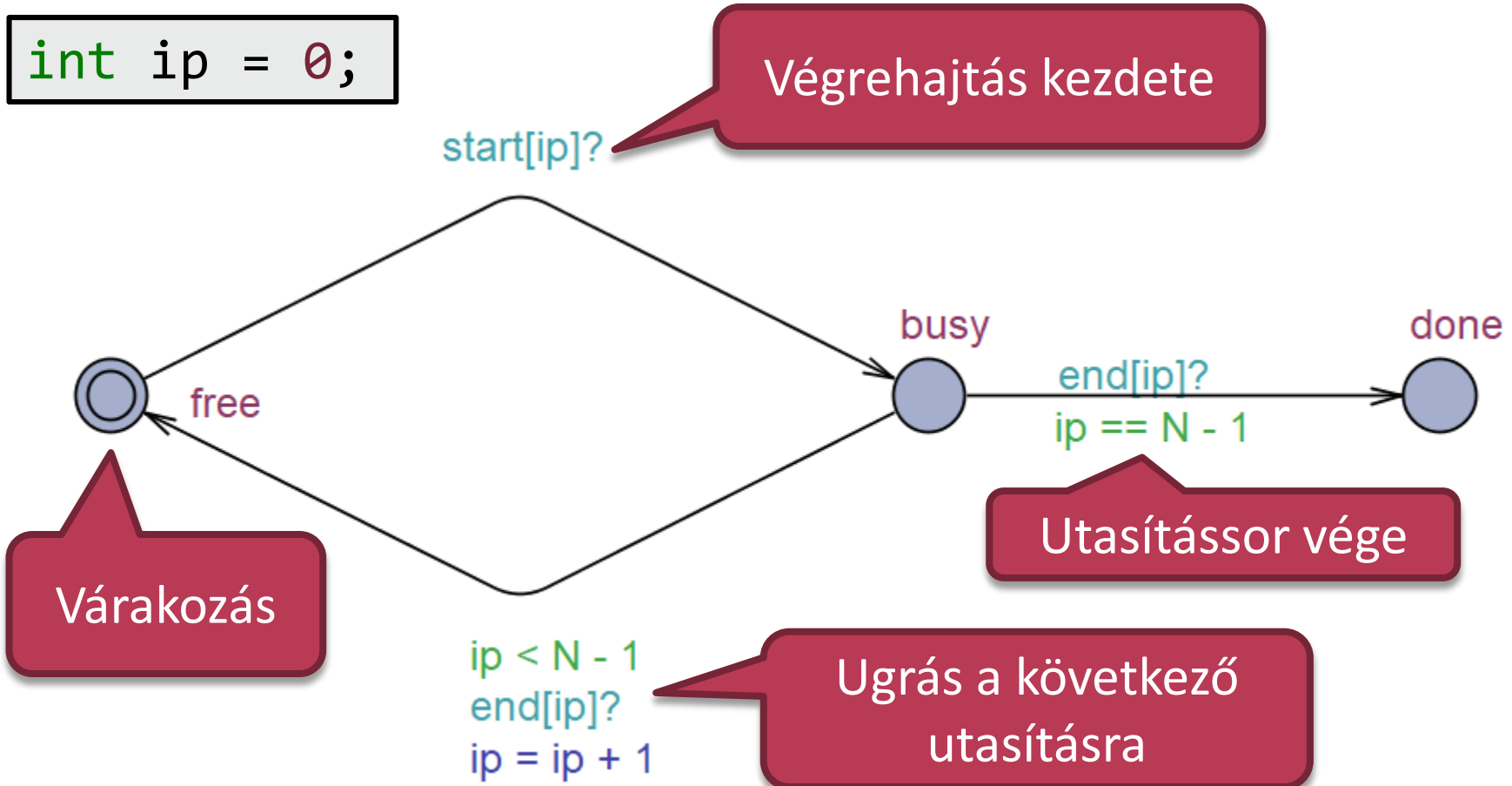
Hogyan modellezzük a támadó kódot?

Modell felépítése (áttekintés)

1. Végrehajtó egység modellezése
2. Utasítás életciklusának modellezése
3. Utasítások végrehajtása: Szinkronizáció a végrehajtó egységek és az utasítások között
4. Egyedi utasítások modellezése

Végrehajtó egység: Unit

```
int ip = 0;
```



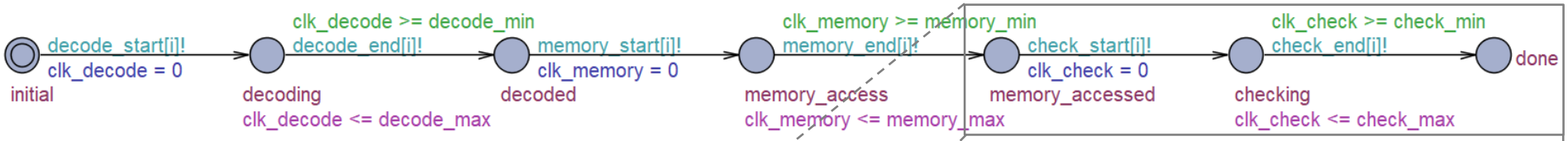
Nem foglalkozik azzal, meddig tart a végrehajtás

Modell felépítése (áttekintés)

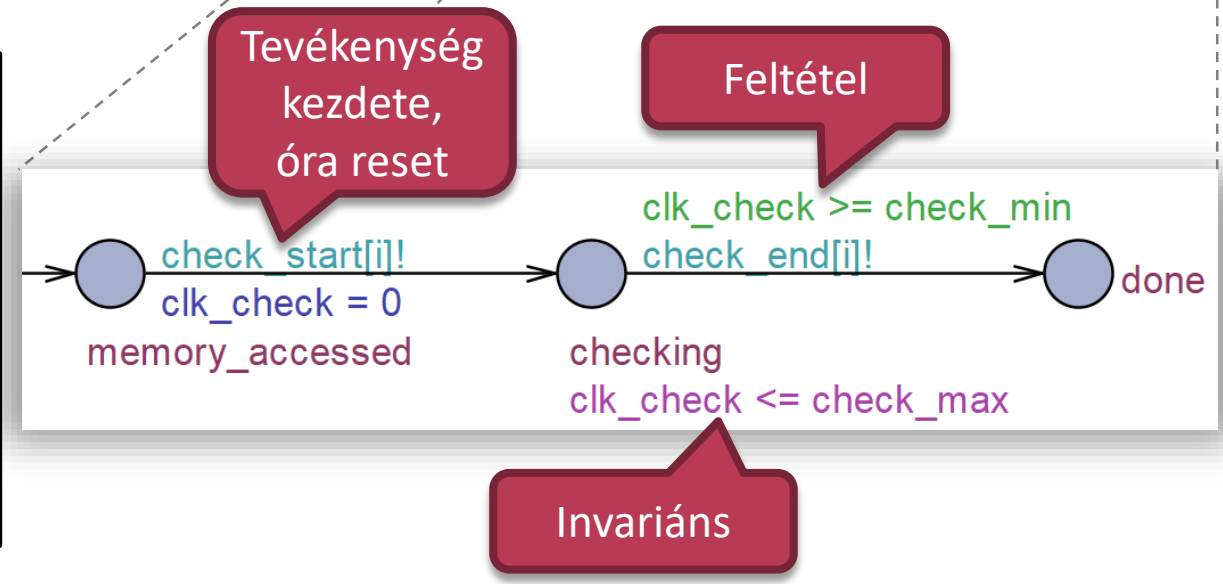
1. Végrehajtó egység modellezése
2. Utasítás életciklusának modellezése
3. Utasítások végrehajtása: Szinkronizáció a végrehajtó egységek és az utasítások között
4. Egyedi utasítások modellezése

Utasítások életciklusa (általános elv)

Dekódolás | Végrehajtás | Ellenőrzés




```
clock clk_decode;
clock clk_memory;
clock clk_check;
const int decode_min = 1;
const int decode_max = 1;
const int memory_min = 5;
const int memory_max = 10;
const int check_min = 5;
const int check_max = 5;
```



Modell felépítése (áttekintés)

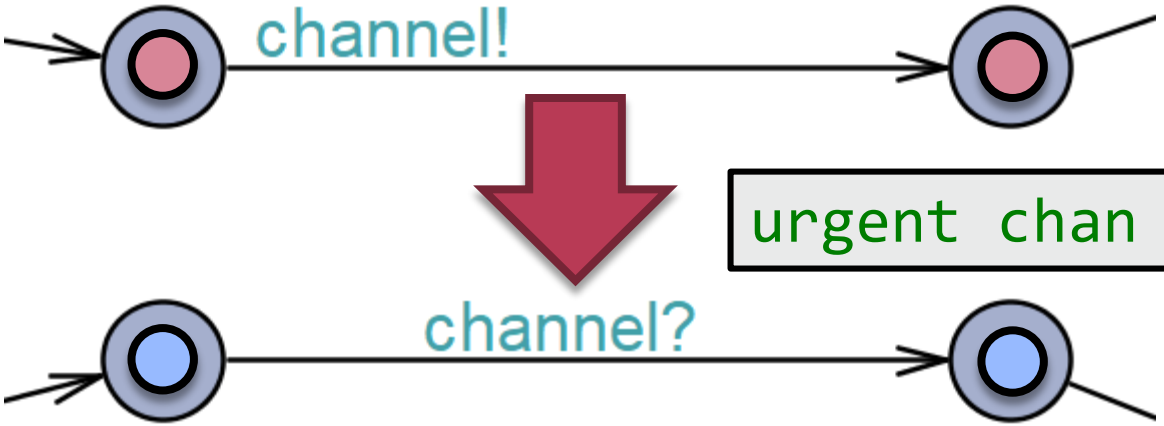
1. Végrehajtó egység modellezése
2. Utasítás életciklusának modellezése
3. Utasítások végrehajtása: Szinkronizáció a végrehajtó egységek és az utasítások között
4. Egyedi utasítások modellezése

Modellezendő viselkedés

- A feldolgozó egység (Unit) azonnal megkezdi a feldolgozást, amint feldolgozhatóvá válik a soron következő utasítás
- A feldolgozó egység sorban halad végig az utasításokon
- Az utasítás életciklusa határozza meg, hogy meddig tartanak a feldolgozási lépések

Utasítások végrehajtása

- Végrehajtás megkezdése késleltetés nélkül
  - Urgent channel: Azonnal lép, amint lehet



Ebben az állapotkonfigurációban nem telhet az idő!

Utasítások végrehajtása

- Végrehajtás megkezdése késleltetés nélkül
  - Urgent channel: Azonnal lép, amint lehet
- Utasítások elkülönítése: Csatornák tömbje

Global declarations

```
const int exploit_size = 1;
const int N = exploit_size + 3;
typedef int[0,N - 1] instr_t;

urgent chan decode_start[instr_t];
chan decode_end[instr_t];
urgent chan exec_start[instr_t];
chan exec_end[instr_t];
urgent chan check_start[instr_t];
chan check_end[instr_t];
```

Utasítások végrehajtása

- Végrehajtás megkezdése késleltetés nélkül
  - Urgent channel: Azonnal lép, amint lehet
- Utasítások elkülönítése: Csatornák tömbje

Felsorolt típus az utasítások azonosítására

Global declarations

```
const int exploit_size = 1;
const int N = exploit_size + 3;
typedef int[0,N - 1] instr_t;

urgent chan decode_start[instr_t];
chan decode_end[instr_t];
urgent chan exec_start[instr_t];
chan exec_end[instr_t];
urgent chan check_start[instr_t];
chan check_end[instr_t];
```

Utasítások végrehajtása

- Végrehajtás megkezdése késleltetés nélkül
  - Urgent channel: Azonnal lép, amint lehet
- Utasítások elkülönítése: Csatornák tömbje

Felsorolt típus az utasítások azonosítására

A feldolgozás azonnal indul

Global declarations

```
const int exploit_size = 1;
const int N = exploit_size + 3;
typedef int[0,N - 1] instr_t;

urgent chan decode_start[instr_t];
chan decode_end[instr_t];
urgent chan exec_start[instr_t];
chan exec_end[instr_t];
urgent chan check_start[instr_t];
chan check_end[instr_t];
```

Utasítások végrehajtása

- Végrehajtás megkezdése késleltetés nélkül
  - Urgent channel: Azonnal lép, amint lehet
- Utasítások elkülönítése: Csatornák tömbje

Felsorolt típus az utasítások azonosítására

A feldolgozás azonnal indul

A feldolgozás során telhet az idő, amíg az invariáns engedi

Global declarations

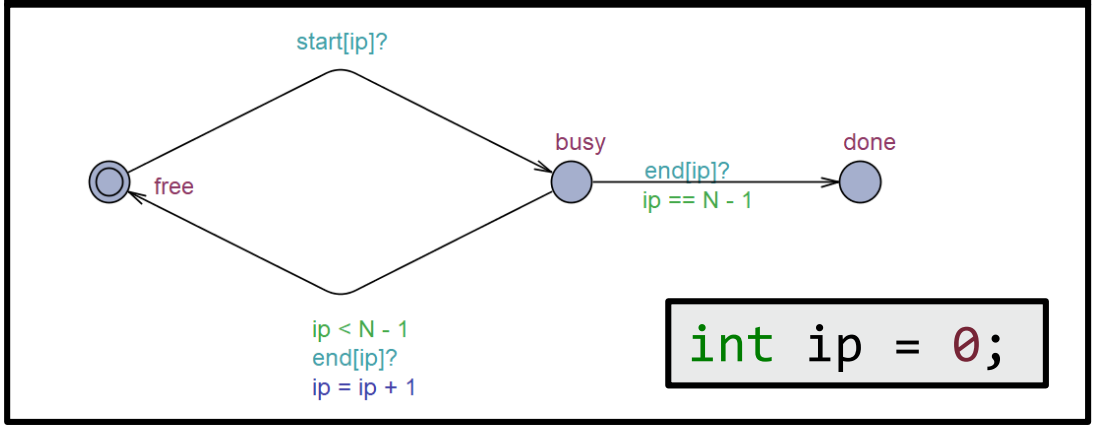
```
const int exploit_size = 1;
const int N = exploit_size + 3;
typedef int[0,N - 1] instr_t;

urgent chan decode_start[instr_t];
chan decode_end[instr_t];
urgent chan exec_start[instr_t];
chan exec_end[instr_t];
urgent chan check_start[instr_t];
chan check_end[instr_t];
```

Utasítások végrehajtása

- Végrehajtó egységek: Egy Unit template alapján
  - Megkülönböztetés: Paraméterezés csatornatömb *referenciával*

```
Unit(urgent chan &start[instr_t],
     chan &end[instr_t])
```



Tömb referencia paraméter

Példányosítás a megfelelő tömbökkel

System declarations

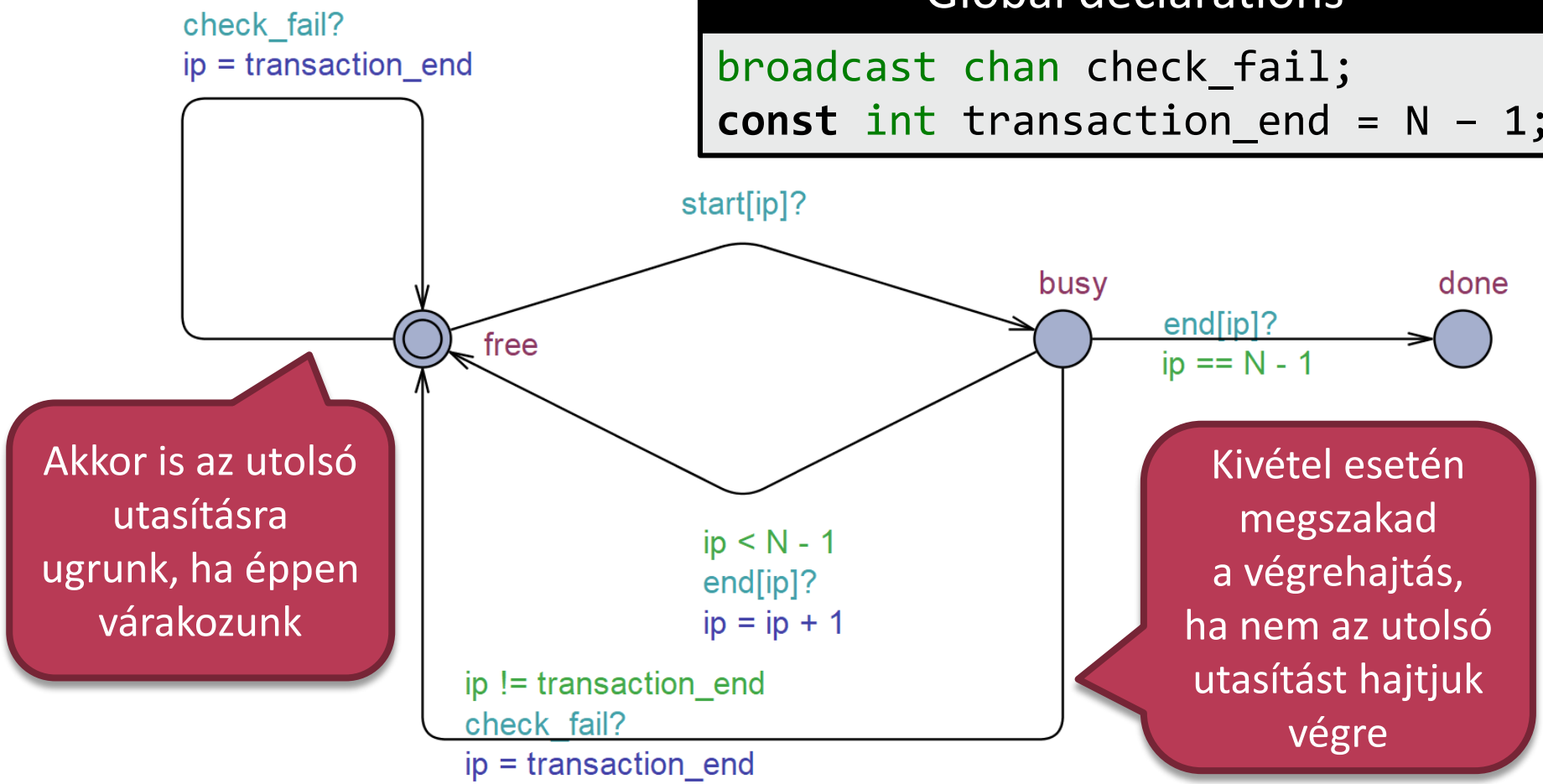
```
DecodeUnit = Unit(decode_start, decode_end);
MemoryUnit = Unit(memory_start, memory_end);
CheckUnit = Unit(check_start, check_end);
system DecodeUnit, MemoryUnit, CheckUnit, Instruction;
```

Kivétel keletkezése

- Kivétel: Minden ephemeral utasítás végrehajtását érinti
  - Broadcast channel kell

Global declarations

```
broadcast chan check_fail;
const int transaction_end = N - 1;
```



Akkor is az utolsó utasításra ugrunk, ha éppen várakozunk

Kivétel esetén megszakad a végrehajtás, ha nem az utolsó utasítást hajtjuk végre

Modell felépítése (áttekintés)

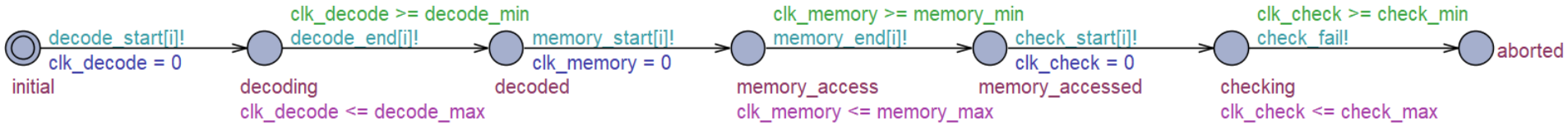
1. Végrehajtó egység modellezése
2. Utasítás életciklusának modellezése
3. Utasítások végrehajtása: Szinkronizáció a végrehajtó egységek és az utasítások között
4. Egyedi utasítások modellezése

Támadó kód modellezése

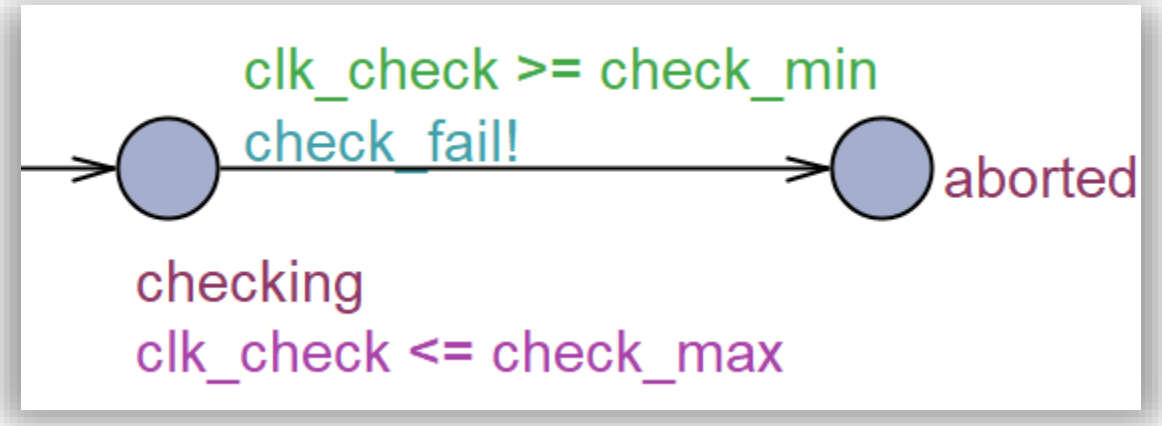
i	Utasítás	Hatás
0.	ReadKernelInstruction	Nem megy át a privilégium ellenőrzésen, kivétel keletkezik
1 … N – 3	Instruction	A támadáshoz szükséges számításokat `exploit_size` darab (global declaration-ben állítható) utasítás példánnyal modellezük, `i` template paraméter
N – 2	WriteSCInstruction	Fedett csatorna cache-be, ha az olvasott bit 1
Ide ugrik a végrehajtás kivétel keletkezése esetén:		
N - 1	ReadSCInstruction	Attól függ a futásideje, hogy a fedett csatorna bekerült-e a cache-be

- Több template-et készítünk, amik az eredeti Instruction template másolatai
  - Módosítások az egyes utasítások jellegzetességeinek modellezéséhez

Kernel bit olvasás: ReadKernelInstruction

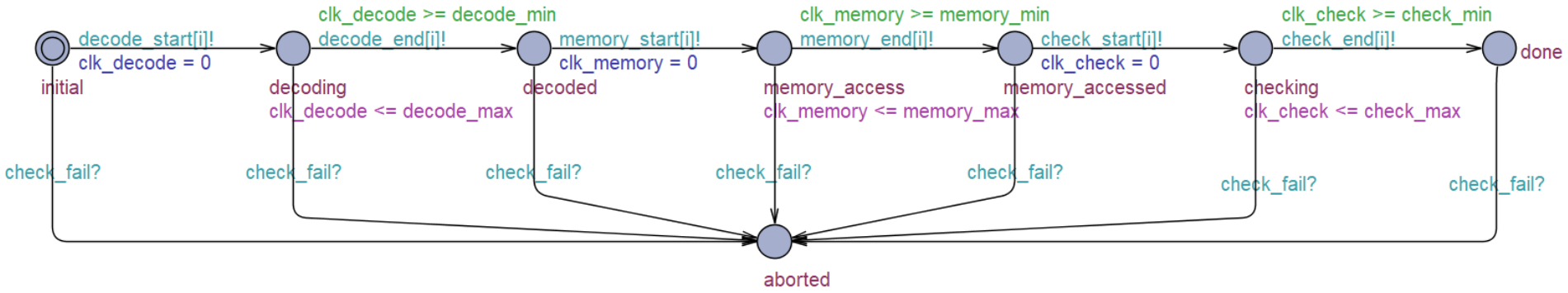



```
int i = 0;
```



- Az első utasítás ellenőrzése kivételt okoz

Ephemeral számítás: Instruction(exploit_t i)

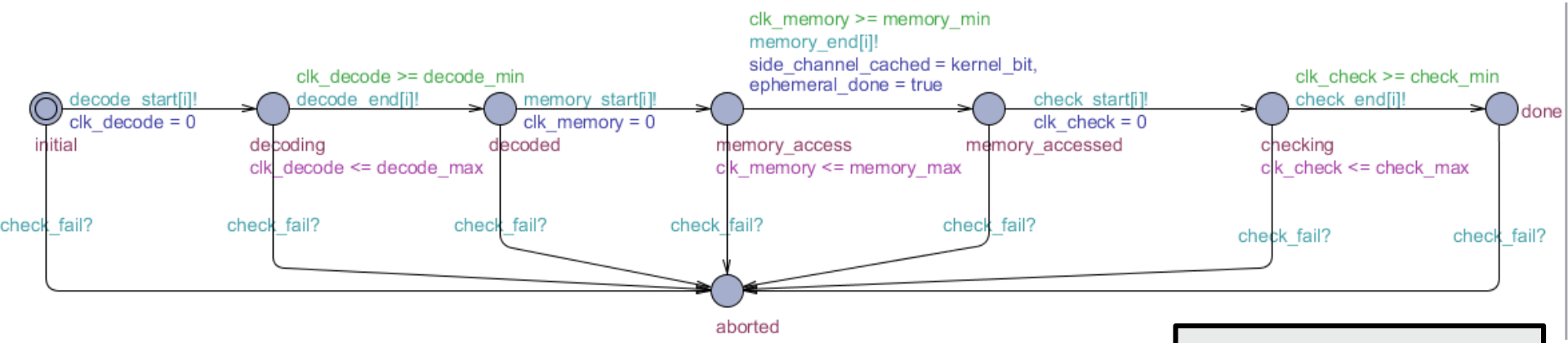


- 1, …, N – 3. utasítás: számítások
- Kivétel esetén ugrás az aborted helyre
- `exploit_t i` paraméter
- Példányosítás **`system`** `Instruction;` formában az `exploit_t` összes értékére

Global declarations

```
typedef int[1,exploit_size] exploit_t;
```

Fedett csatorna: WriteSCInstruction



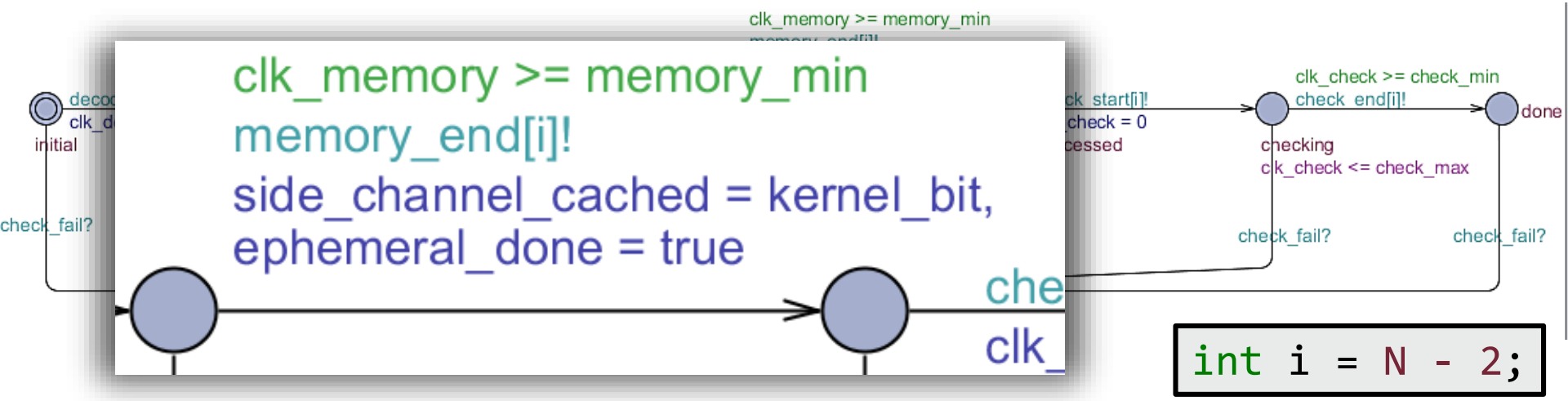

```
int i = N - 2;
```

- Behozza a cache-be a fedett csatorna memóriaterületet, ha a kernel memóriából olvasott bit 1

Global declarations

```
const bool kernel_bit = true;
bool side_channel_cached = false;
```

Fedett csatorna: WriteSCInstruction

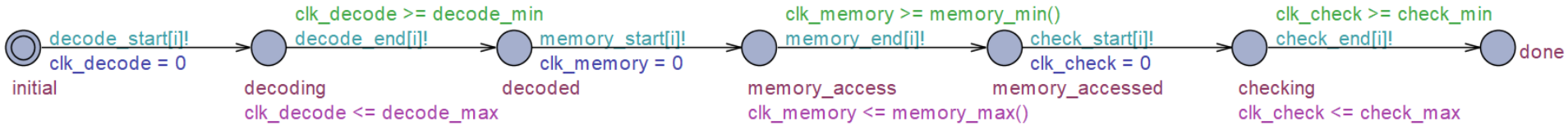


- Behozza a cache-be a fedett csatorna memóriaterületet, ha a kernel memóriából olvasott bit 1

Global declarations

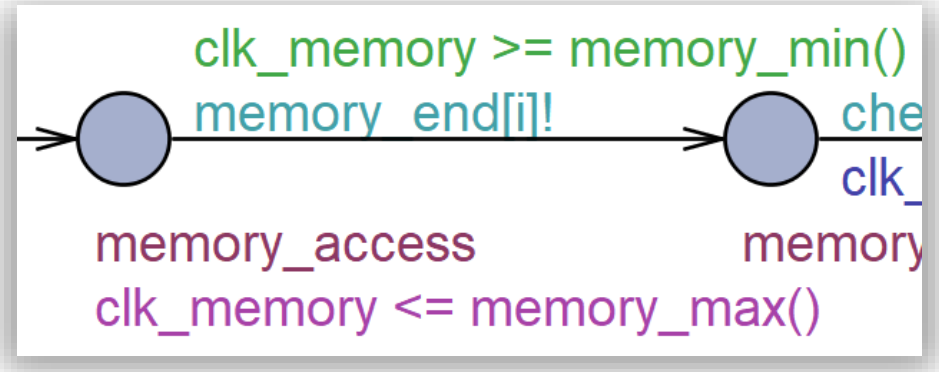
```
const bool kernel_bit = true;
bool side_channel_cached = false;
```

Fedett csatorna olvasás: ReadSCInstruction



- Fedett csatorna állapotától függő végrehajtási idő
- Nem kell felkészülni a megszakításra

```
int i = N - 1;
```




```
int memory_min() {
  if (side_channel_cached) {
    return cached_memory_min;
  } else {
    return uncached_memory_min;
  }
}
int memory_max() {
  if (side_channel_cached) {
    return cached_memory_max;
  } else {
    return uncached_memory_max;
  }
}
```

Rakjunk mindent össze

- Időzítési értékek globális konstansok
- Processzek példányosítása

Global declarations

```
const int decode_min = 1;
const int decode_max = 1;
const int uncached_memory_min = 45;
const int uncached_memory_max = 120;
const int cached_memory_min = 15;
const int cached_memory_max = 30;
const int kernel_check_min = 40;
const int kernel_check_max = 100;
const int user_check_min = 10;
const int user_check_max = 25;
```

System declarations

```
DecodeUnit = Unit(decode_start, decode_end);
MemoryUnit = Unit(memory_start, memory_end);
CheckUnit = Unit(check_start, check_end);
system DecodeUnit, MemoryUnit, CheckUnit, ReadKernelInstruction,
Instruction, WriteSCInstruction, ReadSCInstruction;
```

DEMO Tulajdonságok ellenőrzése

- Az utolsó utasítás (ReadSCInstruction) mindig ki tudja olvasni a kernel memória tartalmát a fedett csatornából
- Legfeljebb hány utasításból állhat a számítás (`exploit_size`), hogy még sikeres legyen a támadás?
- Hány órajelig tart az utasítássor végrehajtása, ha…
  …a kernel memória tartalma 1-es bit?
  …a kernel memória tartalma 0-s bit?
- Az utolsó előtti utasítás (WriteSCInstruction) sohasem hajtódik végre teljes egészében

Tulajdonságok ellenőrzése

- Az utolsó utasítás (ReadSCInstruction) mindig ki tudja olvasni a kernel memória tartalmát a fedett csatornából
  - A<> ReadSCInstruction.done && side_channel_cached verifikálása
- Legfeljebb hány utasításból állhat a számítás (`exploit_size`), hogy még sikeres legyen a támadás?
  - E<> ReadSCInstruction.done && ephemeral_done verifikálása
  - `exploit_size` értékének növelése
- Hány órajelig tart az utasítássor végrehajtása, ha…
  …a kernel memória tartalma 1-es bit?
  …a kernel memória tartalma 0-s bit?
  - Legrövidebb diagnosztikai trace keresésének beállítása
  - `kernel_bit` állítása és `total_time` megfigyelése (a szimulátorban)
- Az utolsó előtti utasítás (WriteSCInstruction) sohasem hajtódik végre teljes egészében
  - A[] !WriteSCInstruction.done verifikálása