EXPLAINED:

MELTDOWN & SPECTRE

Jens Neuhalfen

https://github.com/neuhalje/presentation_meltdown_spectre

EXPLAINED:

MELTDOWN & SPECTRE

1E3 edition

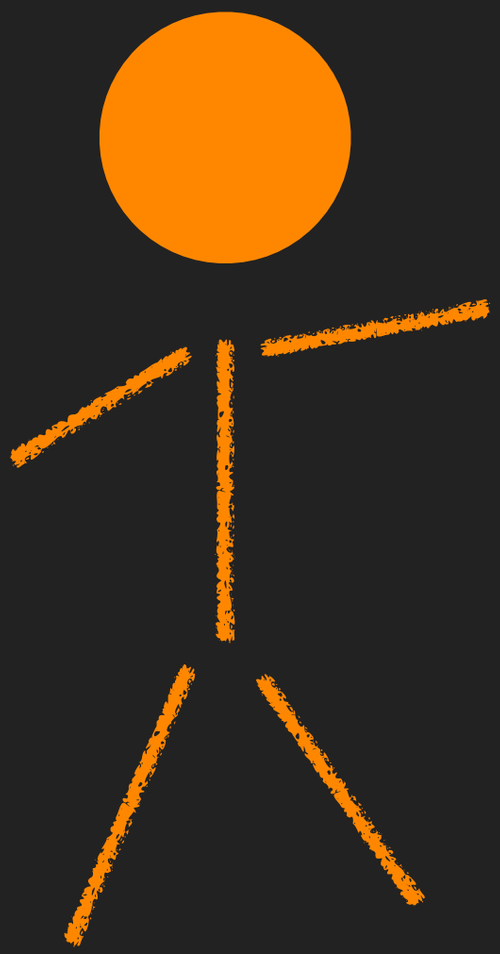
Jens Neuhalfen

https://github.com/neuhalje/presentation_meltdown_spectre

WHO AM I?

- Jens Neuhalfen
- Age: Forty something
- IT since: ever
- Skills: Bridge between IT and business, IT-Security Management, writing software
- https://github.com/neuhalje

SECURITY GOALS

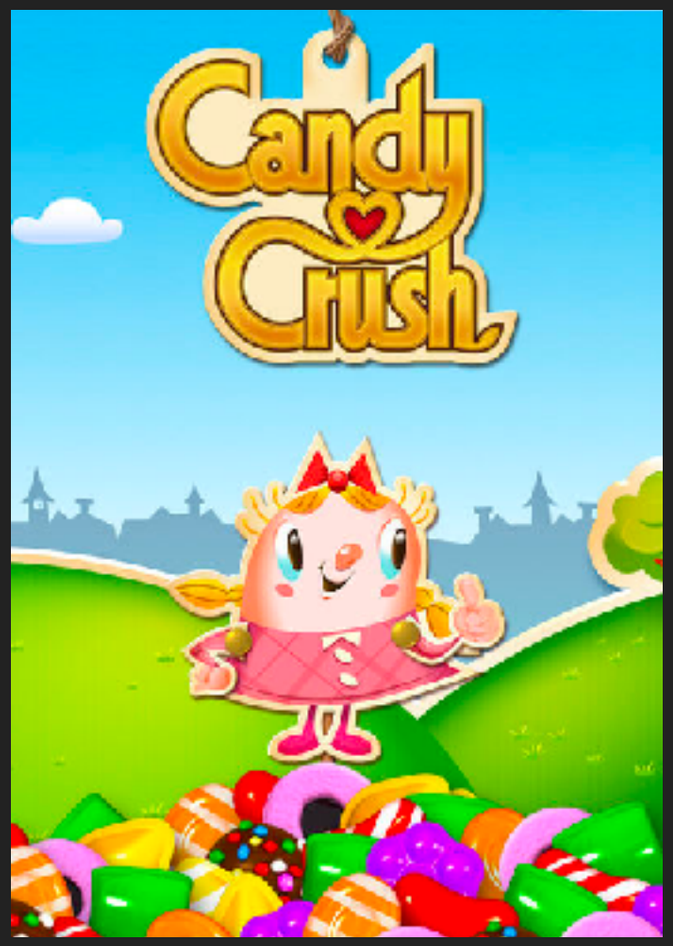


A PROCESS MUST BE ISOLATED (PROTECTED) FROM OTHER PROCESSES!

You

SECURITY GOALS: APP ISOLATION

SECURITY GOALS: APP ISOLATION

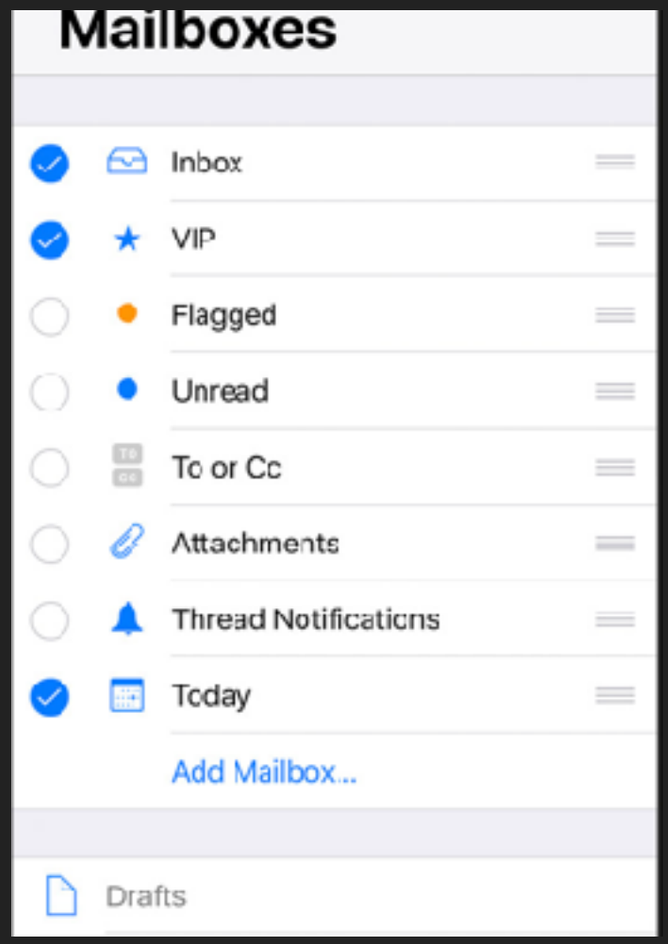


You don’t want

this

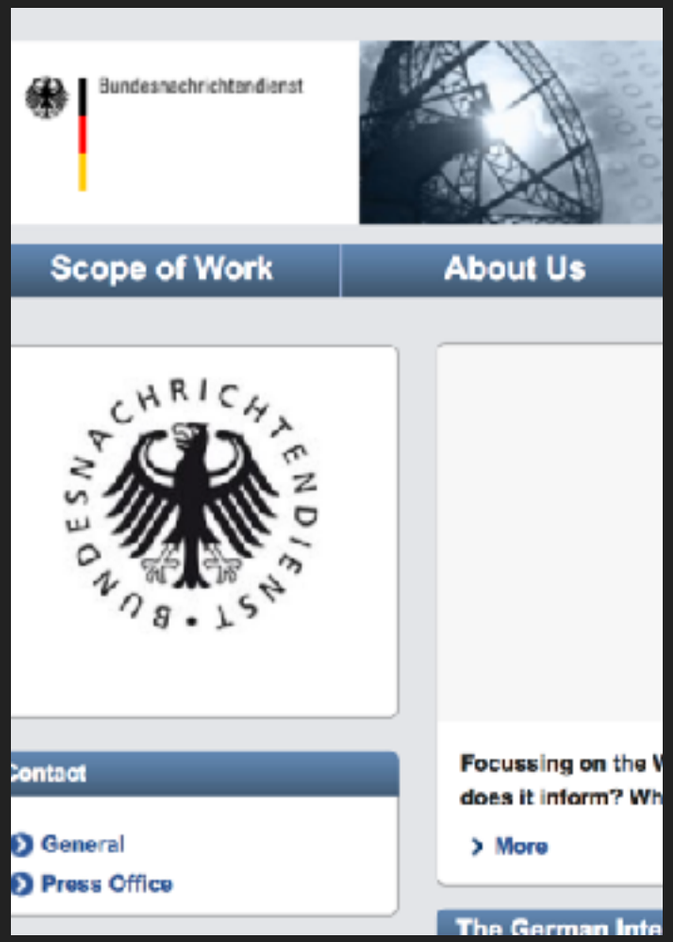
SECURITY GOALS: APP ISOLATION

You don't want
this



To read
that

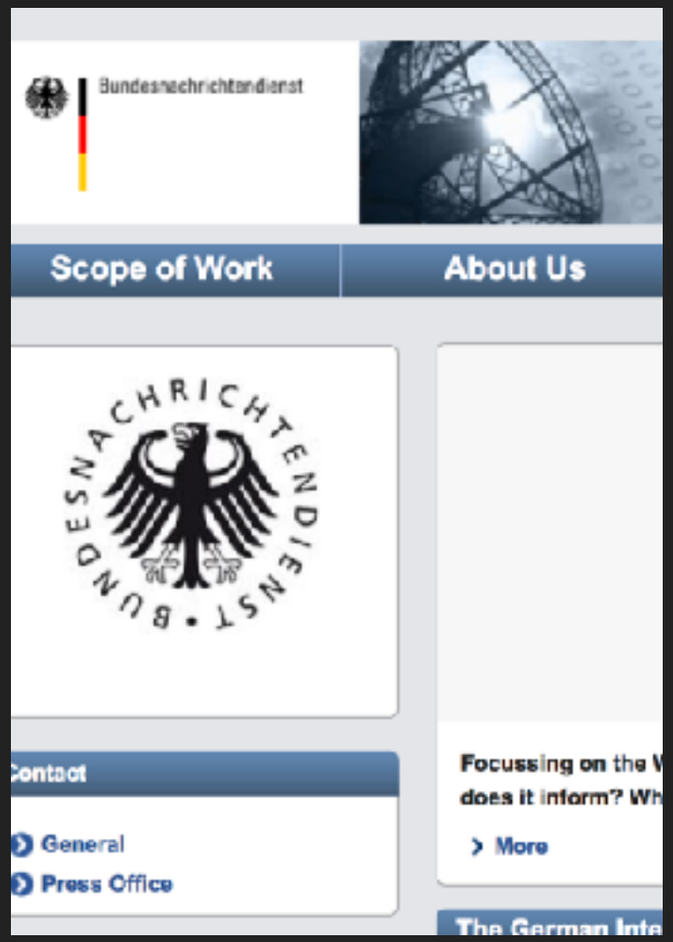
SECURITY GOALS: BROWSER TAB ISOLATION



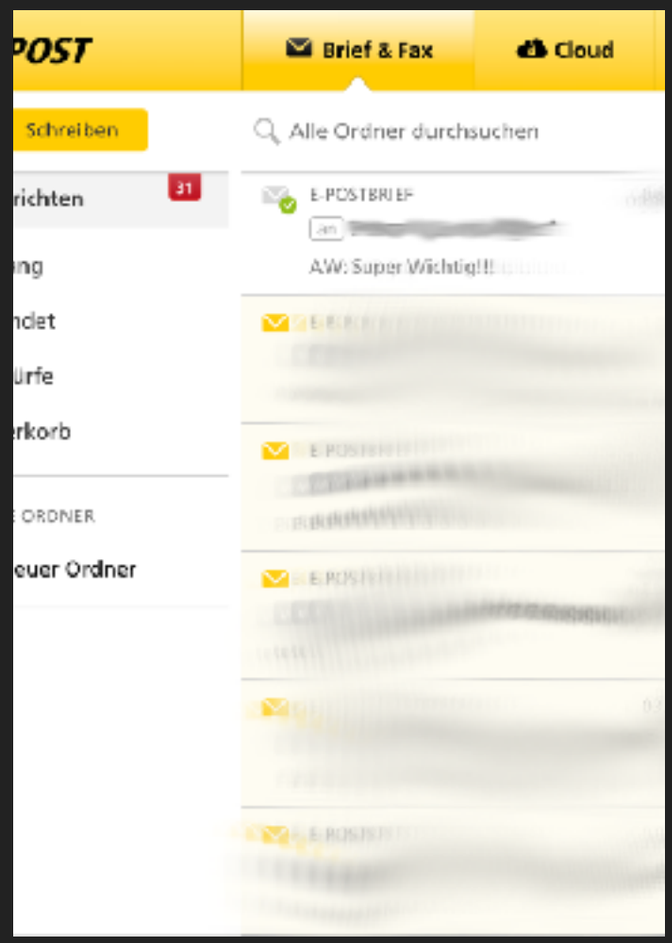
You don’t want

this

SECURITY GOALS: BROWSER TAB ISOLATION



You don't want
this



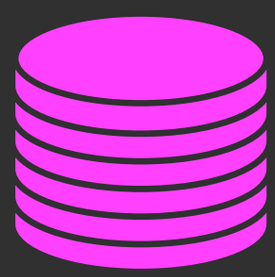
To read
that

SECURITY GOALS: CLOUD ISOLATION

You don’t want

this

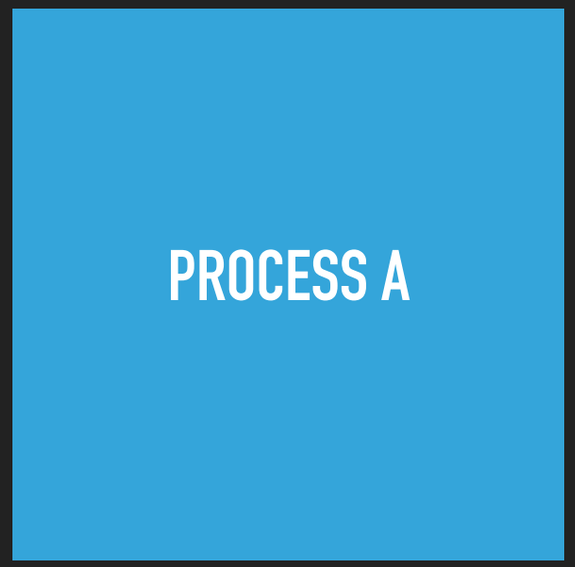
SECURITY GOALS: CLOUD ISOLATION



You don't want
this

To read
that

SECURITY GOALS: MEMORY ISOLATION



You don’t want

this

SECURITY GOALS: MEMORY ISOLATION

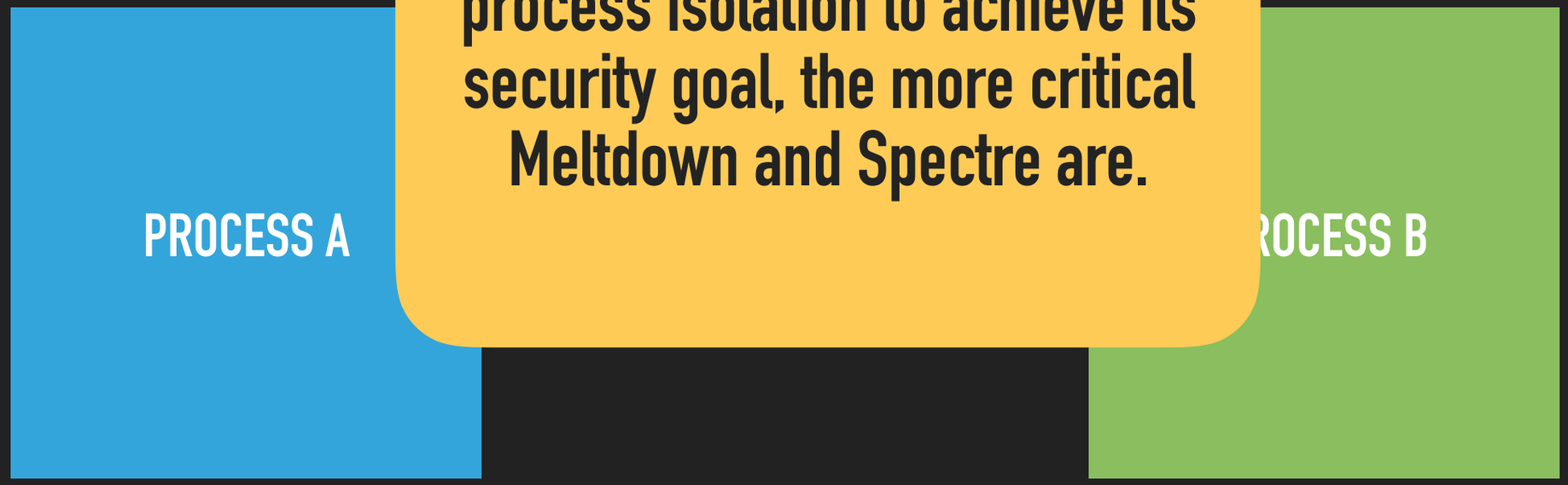
PROCESS A

PROCESS B

You don’t want *this*

To read *that*

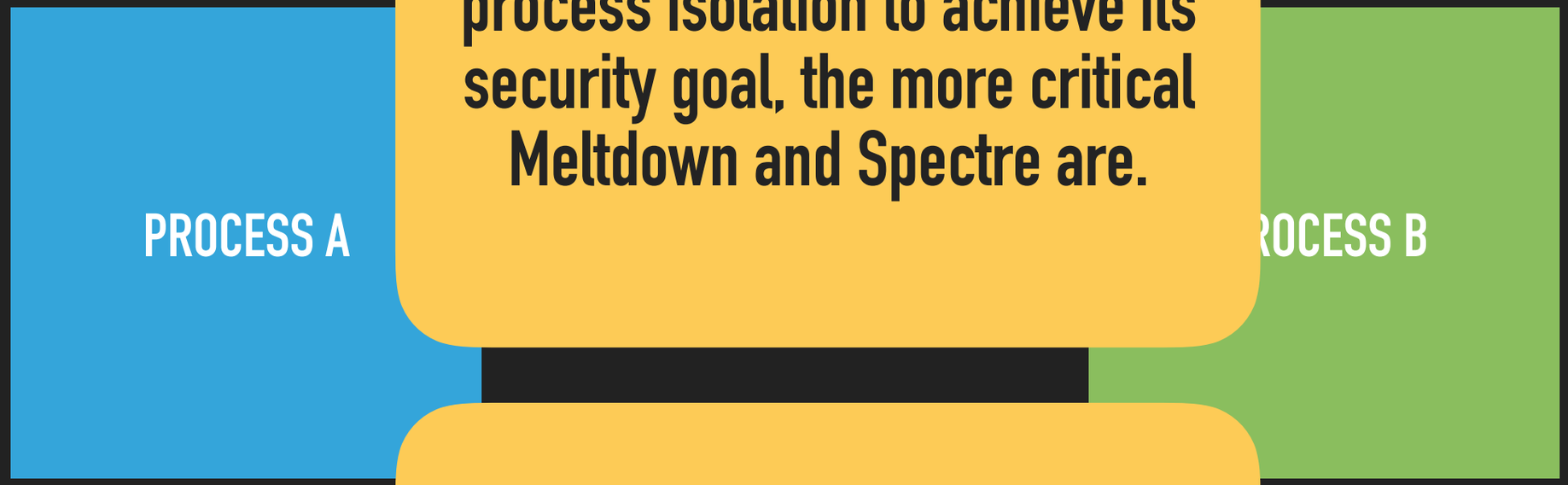
SECURITY GOALS: MEMORY ISOLATION



You don't want *this*

To read *that*

SECURITY GOALS: MEMORY ISOLATION



The more a system relies on process isolation to achieve its security goal, the more critical Meltdown and Spectre are.

Fortunately an attacker must be able to execute his code on a system to exploit the Meltdown and Spectre attacks.

You don't read *this* *hat*

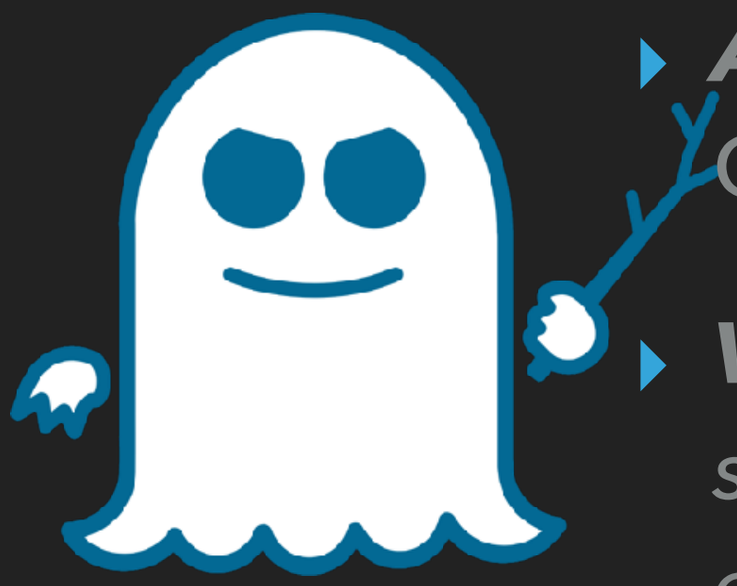
MELTDOWN AND SPECTRE

MANAGEMENT SUMMARY

MELTDOWN

- ***Result***:Programs can read memory it should not
- ***Affects***: All modern CPU/OS
- ***Vector***: Uses *out of order execution* to read forbidden memory and *cache timing* as side channel to exfiltrate data
- ***How bad***: Bad
- ***Fixes***: Needs changes in CPU and/or OS patches. Modest (X%) to severe (XX%) performance impact, higher on older CPU. Performance impact varies and depends on CPU and workload type.

SPECTRE



- ***Result***:Programs can read all memory
- ***Affects***: All modern CPU/OS
- ***Vector***: Uses *speculative execution* to read forbidden memory and *cache timing* to exfiltrate data
- ***How bad***: Very bad
- ***Fixes***: Needs changes in CPU and/or changes in programs. Performance impact varies and depends on CPU and workload type.

THREAT-O-METER

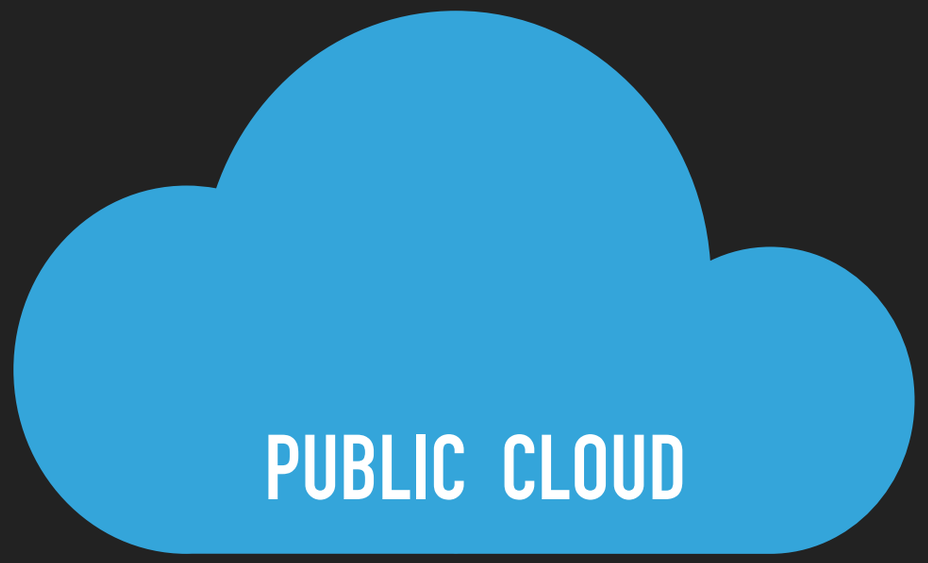
Threat - O - Meter

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Public clouds run code of many untrusted parties which makes them very vulnerable.

LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

DATABASE SERVER

PUBLIC CLOUD

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER

LOW RISK

Exploit unlikely or running untrusted code already worst case

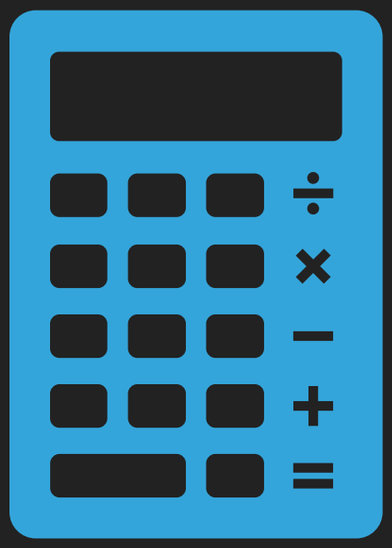
MEDIUM RISK

Exploit possibl needs anoth successful attack attackers co

Databases are often protected from the internet and are accessed only by application servers.

Running untrusted code on a database is often already the worst case scenario. Patching against Meltdown/ Spectre would only marginally increase security.

THREAT-O-METER



MAILSERVER

LOW RISK

Exploit unlikely or running untrusted code already worst case

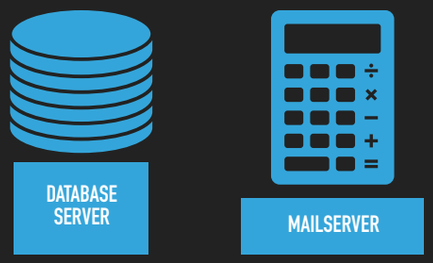
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possibl… needs anoth… successful attack to run attackers code

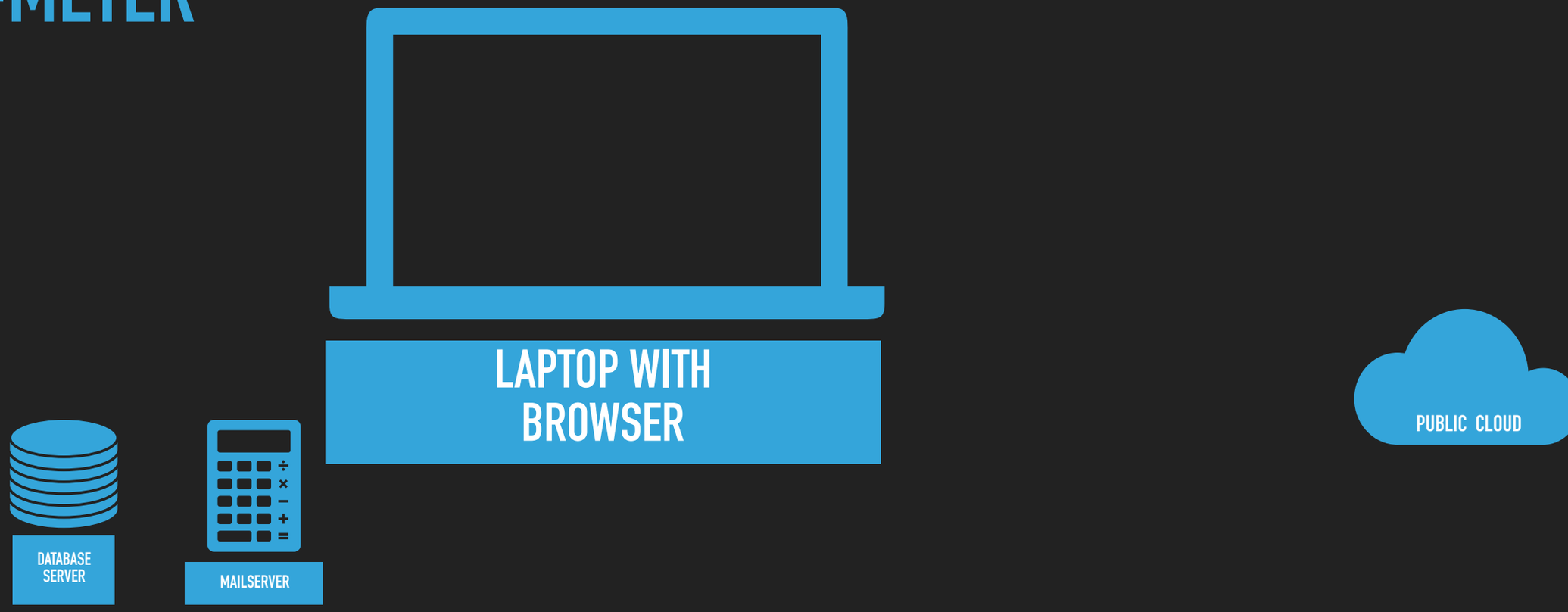
…design”

Mailserver are exposed to the internet but have been proven to be very robust to “remote code execution” attacks.

Also a code execution is already the worst case.

Arguably mail servers can be placed in “medium” due to their exposure to the internet.

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

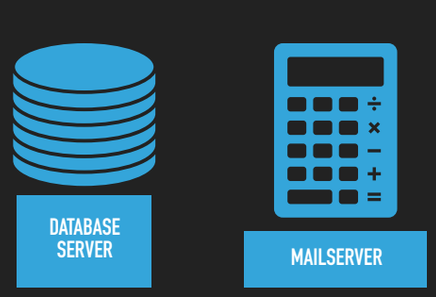
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Laptops/desktop systems with browsers are very vulnerable because they execute untrusted code in the form of JavaScript from websites.



LOW RISK

Exploit unlikely or running untrusted code already worst case

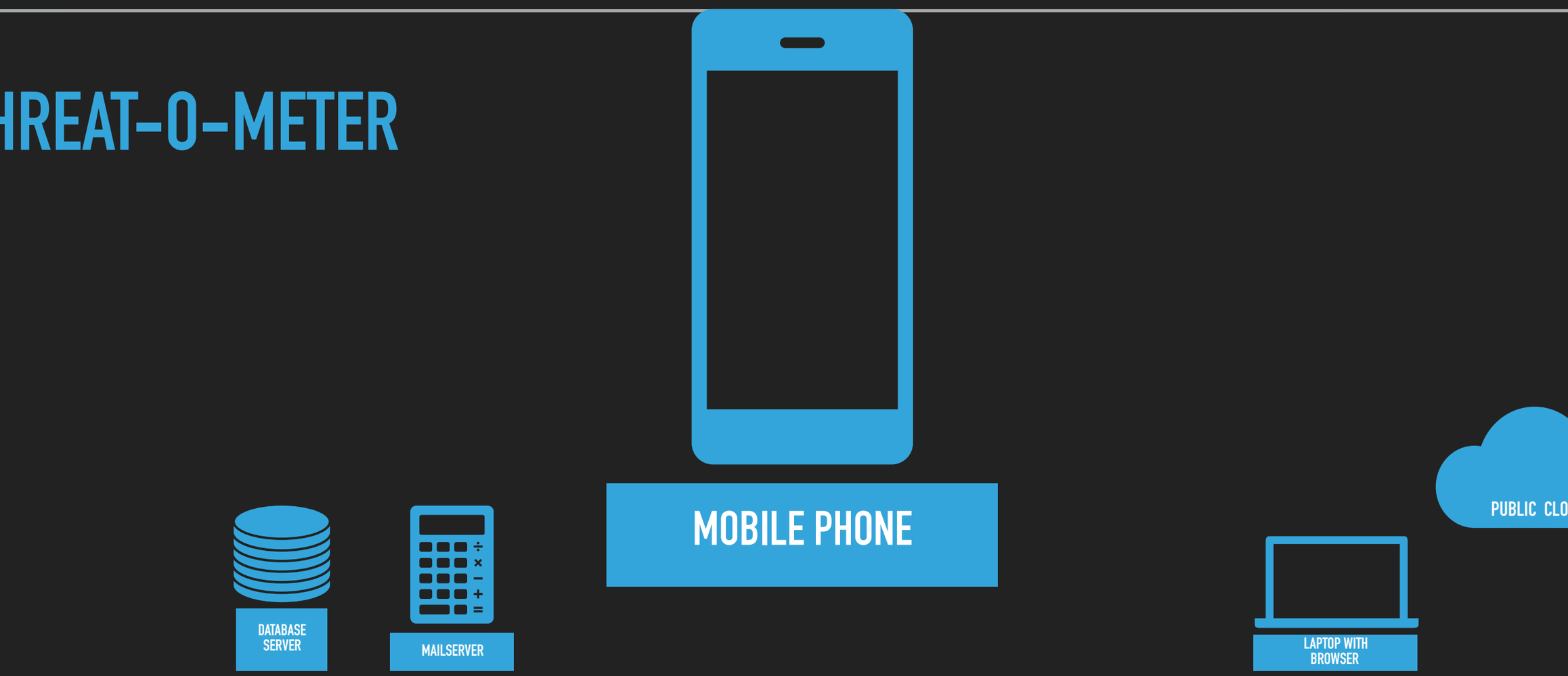
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Mobile phones run apps and websites (JavaScript).

LOW RISK

Exploit unlikely or running untrusted code already worst case

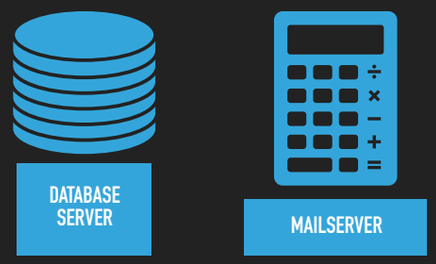
MEDIUM RISK

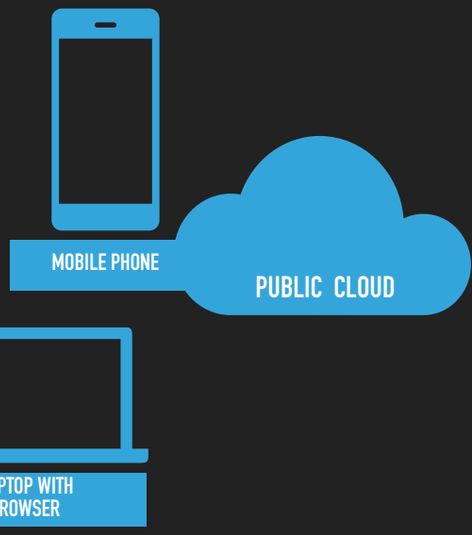
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER





LOW RISK

Exploit unlikely or running untrusted code already worst case

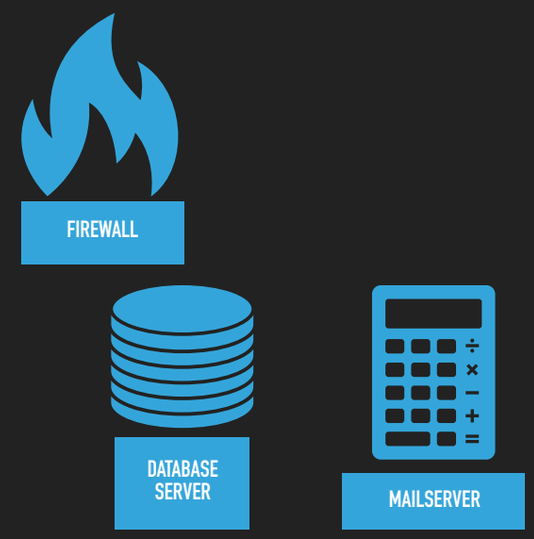
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possibl needs anoth successful attack attackers co

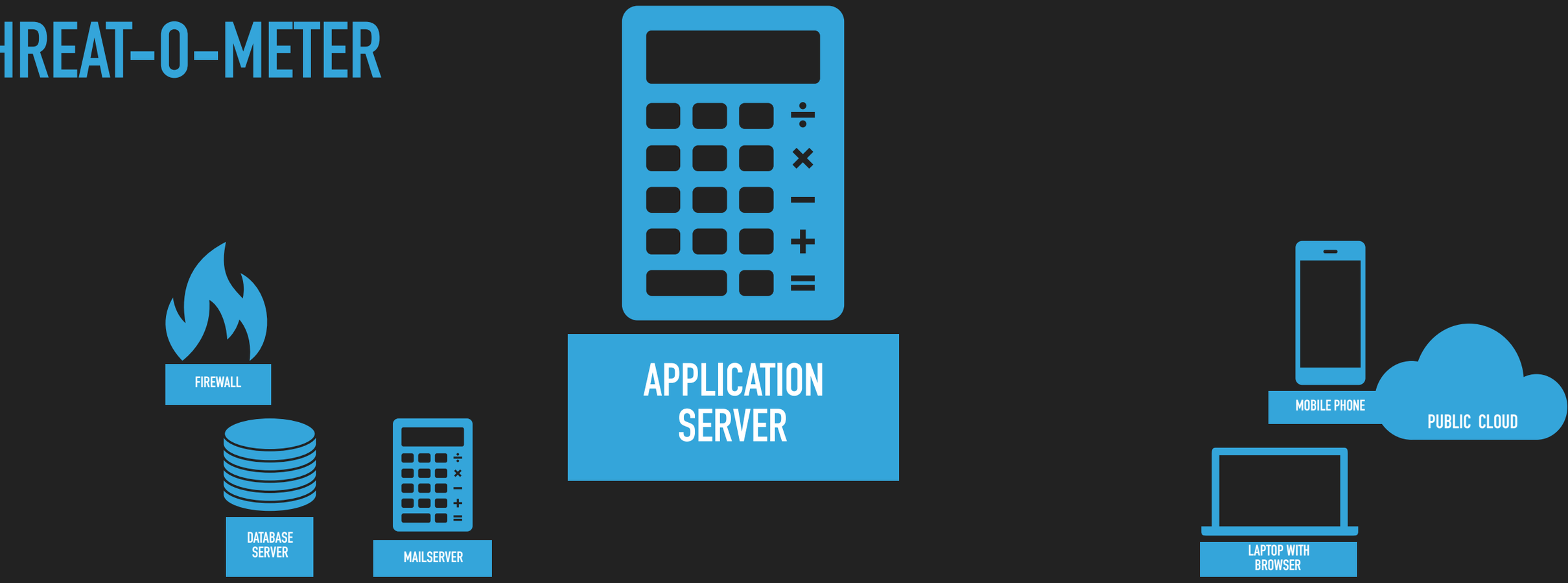
Firewalls and switches (normally) do not expose an attackable surface to the external network.

This greatly reduces the likelihood of attacks.

A code execution is already the worst case.

VPN gateways expose a complex interface and are more likely to be attacked.

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Application servers only run trusted code but attacks can lead to code execution.

How many Java (node, Ruby,...) **libraries does your software use? And transitively? Who audits all these?**

LOW RISK

Exploit unlikely or running untrusted code already worst case

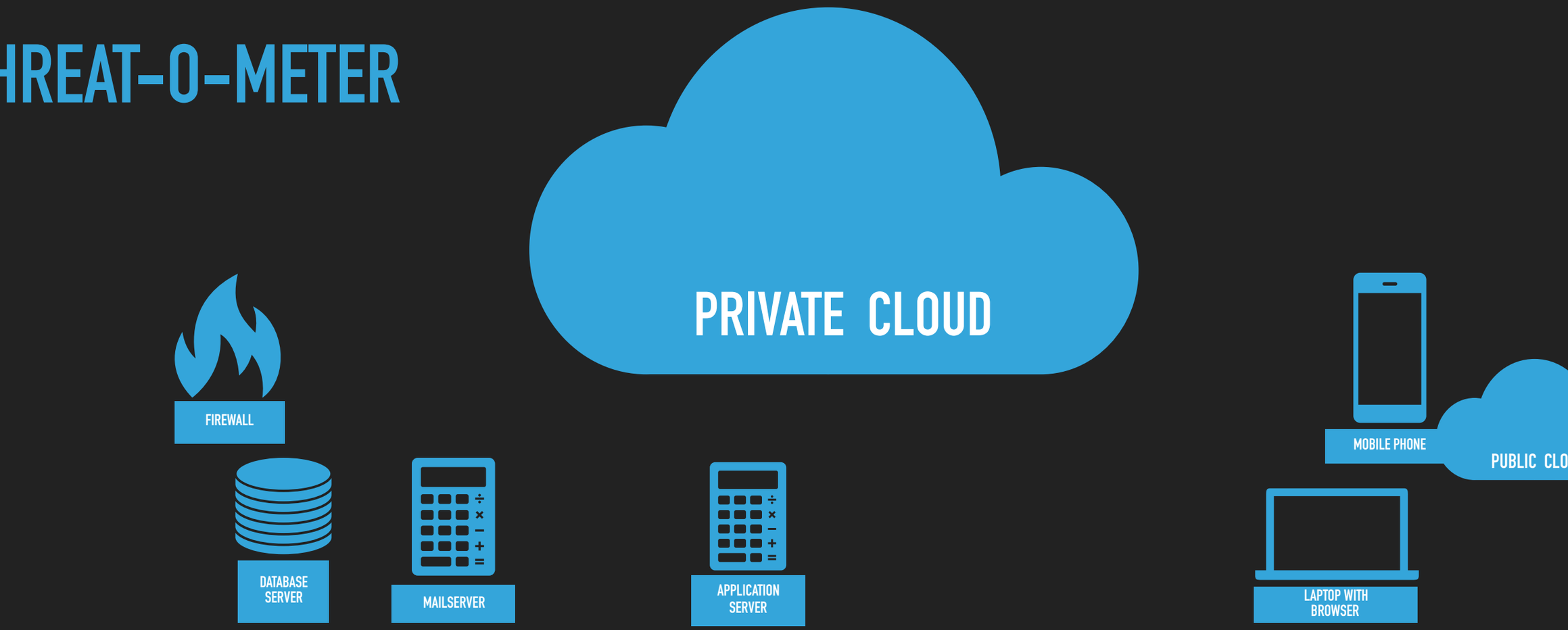
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

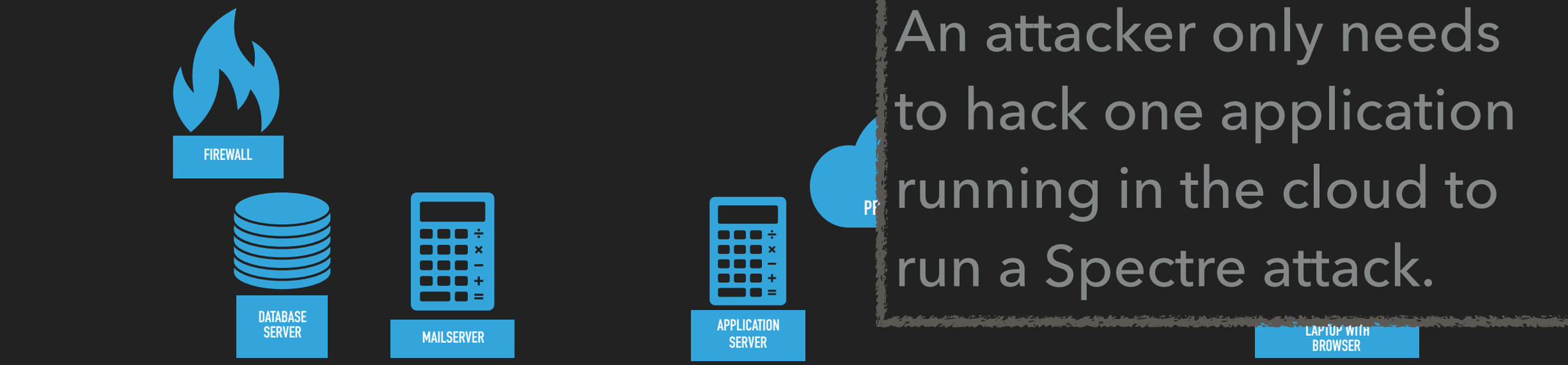
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



Private clouds run many different workloads but they are all trusted.

An attacker only needs to hack one application running in the cloud to run a Spectre attack.

LOW RISK

Exploit unlikely or running untrusted code already worst case

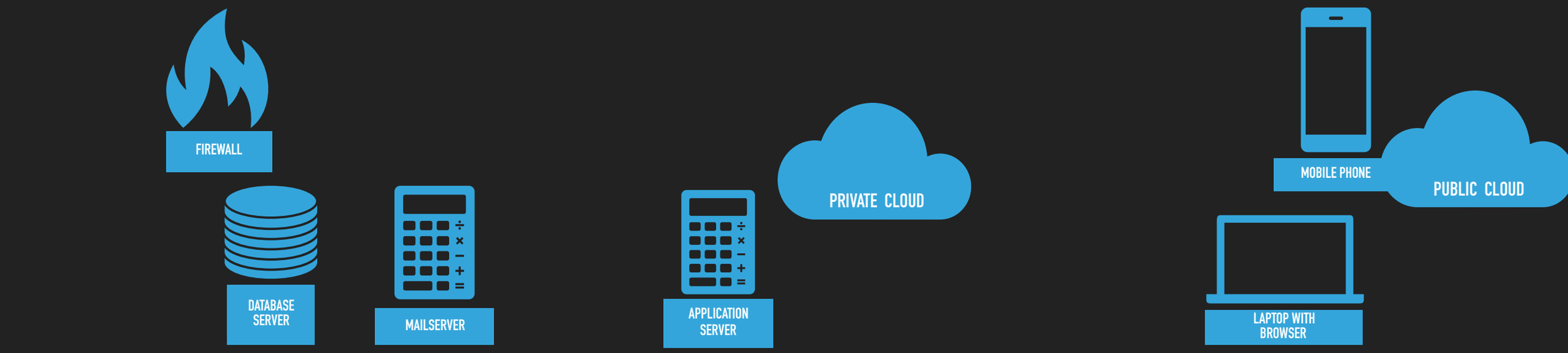
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

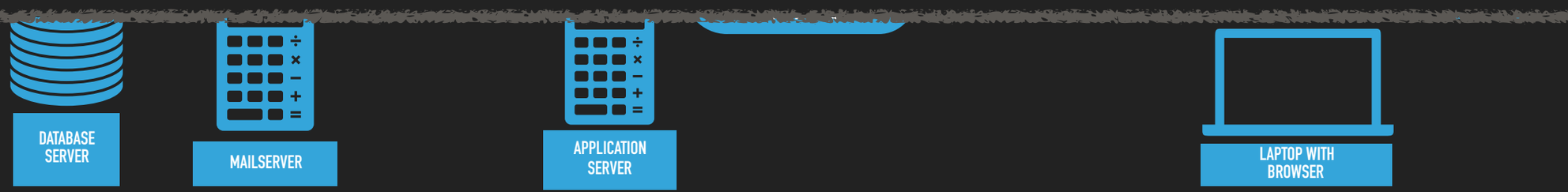
HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Given the patches are risky w. regards to performance and availability.

What would be your patching strategy for each risk class?



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

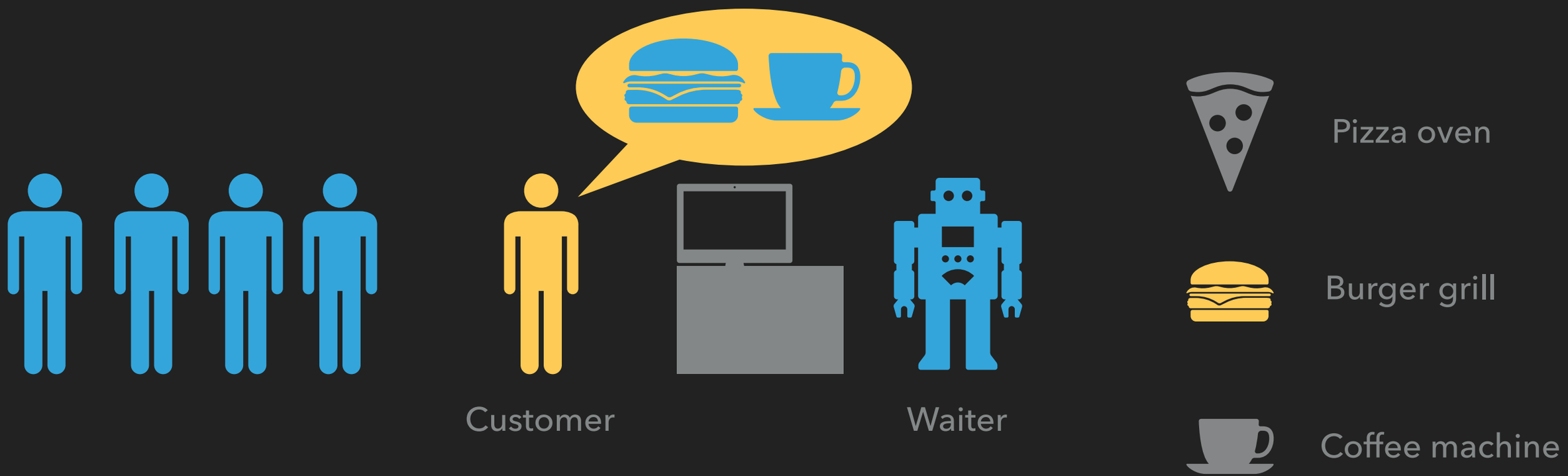
Exploit possible and runs untrusted code “by design”

ACCIDENT, MALICE, INCOMPETENCE?

WHY DID IT HAPPEN?

CONFIDENTIAL BURGERS INC.

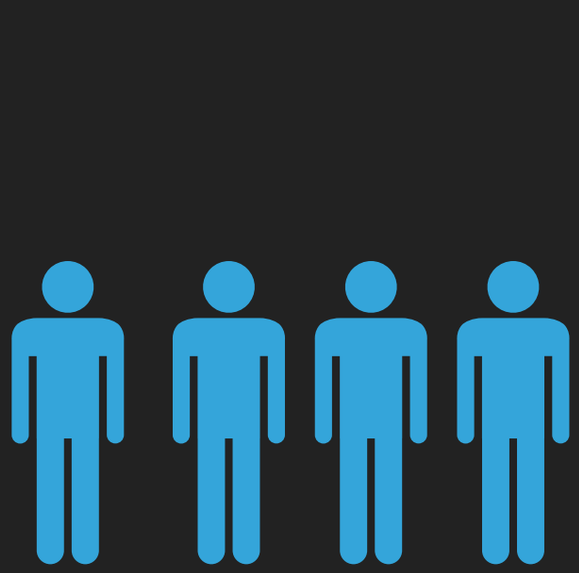
Confidential Burgers inc. sells burgers, pizza, and coffee.



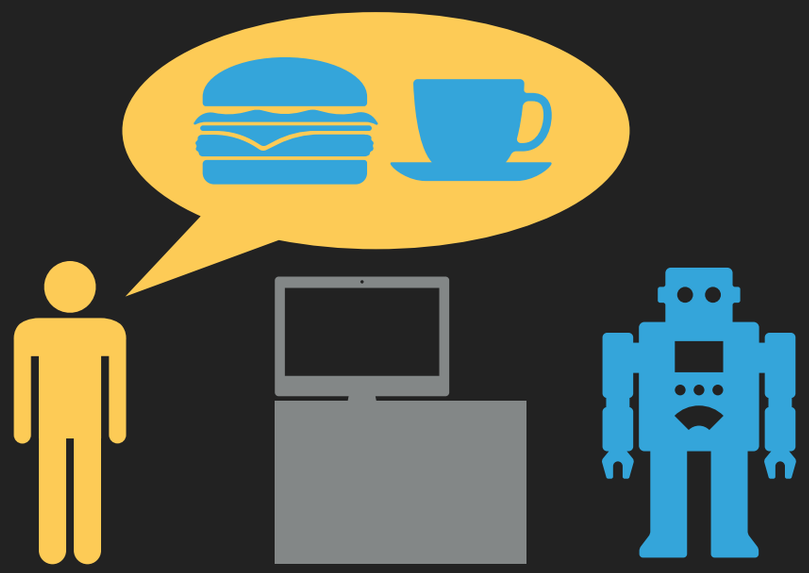
Grand Opening Today

CONFIDENTIAL BURGERS INC.

Confidential Burgers inc. sells burgers, pizza, and coffee.



The waiter (CPU) will



Customer

Waiter

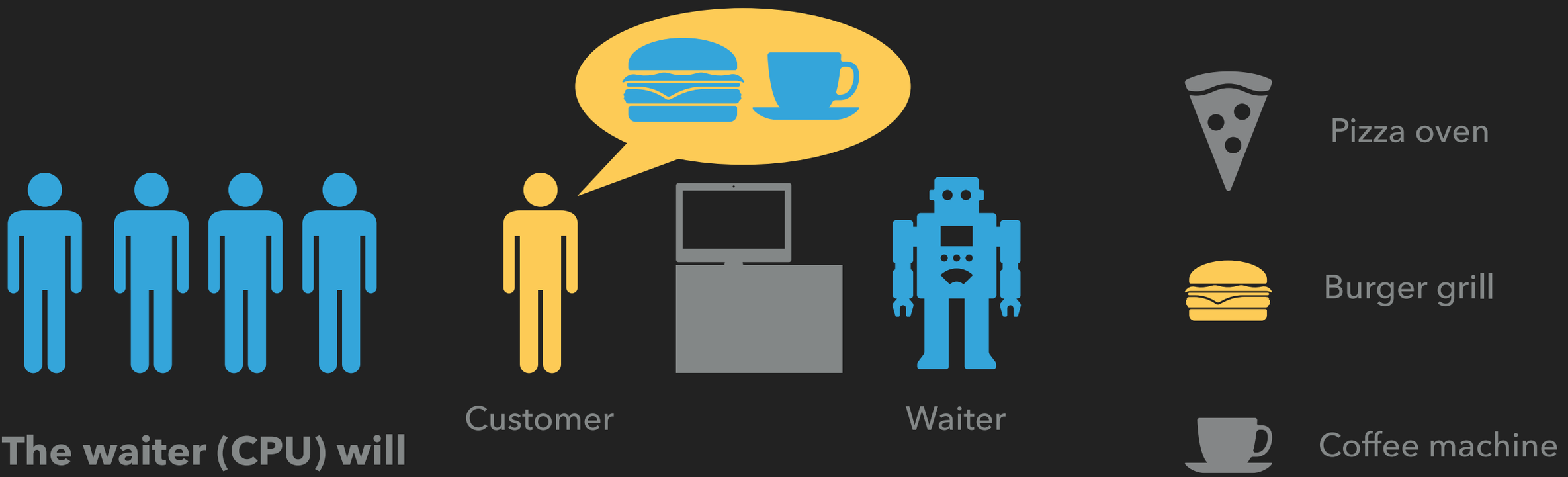
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC.

Confidential Burgers inc. sells burgers, pizza, and coffee.

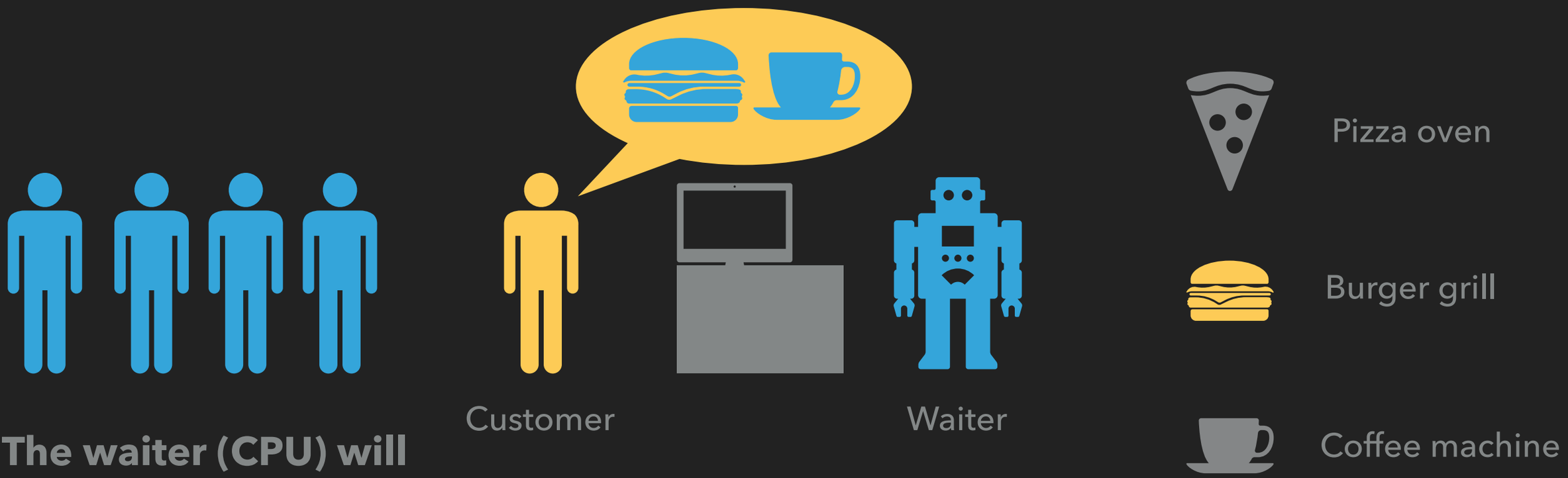


The waiter (CPU) will

1. **take an order from a customer (CPU instruction)**

CONFIDENTIAL BURGERS INC.

Confidential Burgers inc. sells burgers, pizza, and coffee.

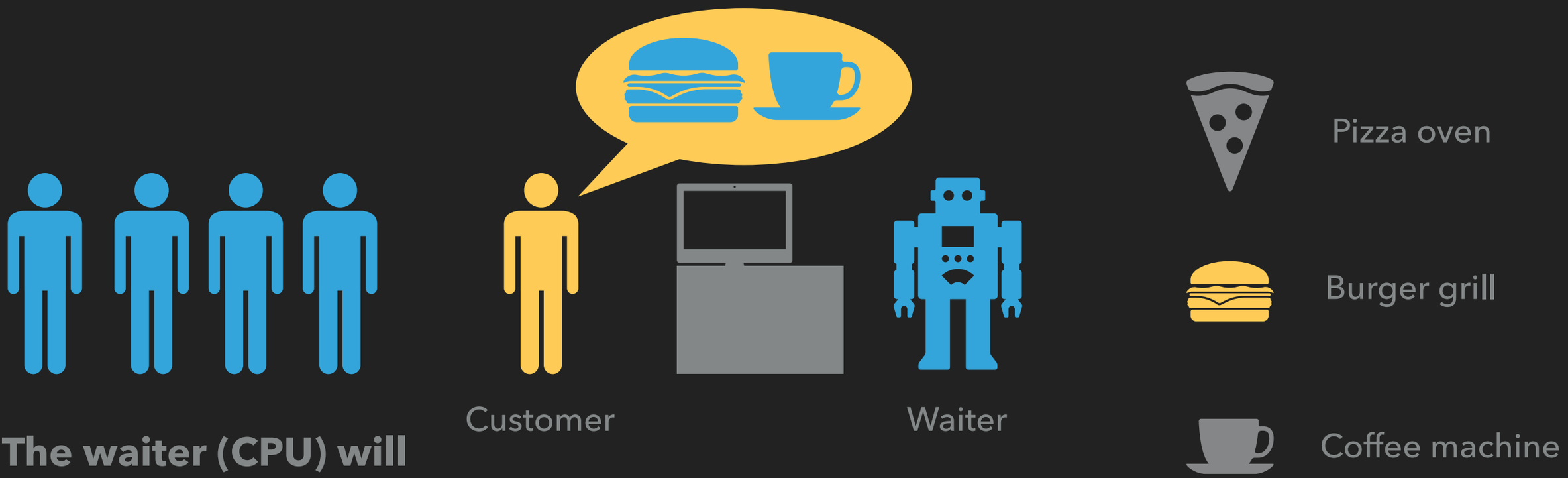


The waiter (CPU) will

1. **take an order from a customer (CPU instruction)**
2. **break the order (instruction) down into micro operations (µOPs - grilling a burger, baking a pizza, ...)**

CONFIDENTIAL BURGERS INC.

Confidential Burgers inc. sells burgers, pizza, and coffee.

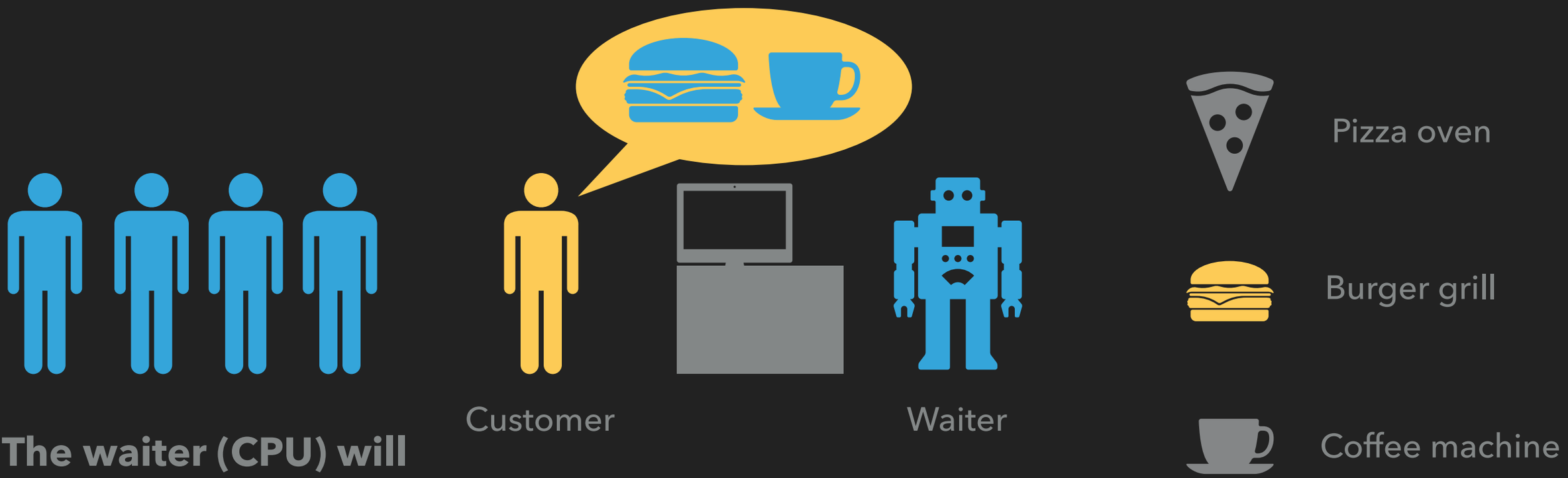


The waiter (CPU) will

1. **take an order from a customer (CPU instruction)**
2. **break the order (instruction) down into micro operations (µOPs - grilling a burger, baking a pizza, ...)**
3. **schedule and execute the µOPs**

CONFIDENTIAL BURGERS INC.

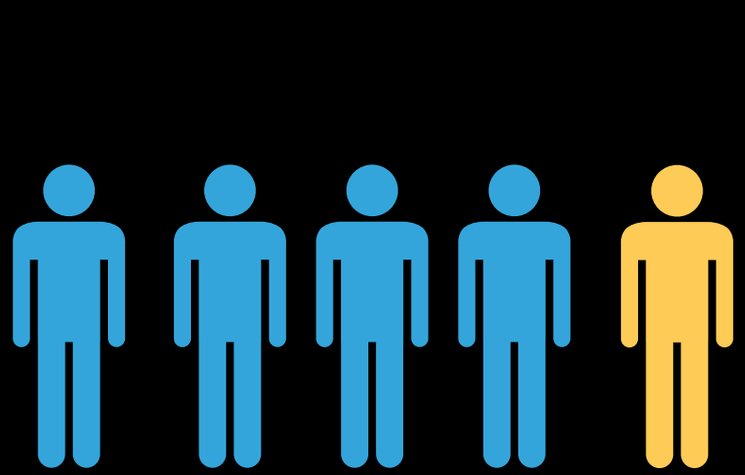
Confidential Burgers inc. sells burgers, pizza, and coffee.



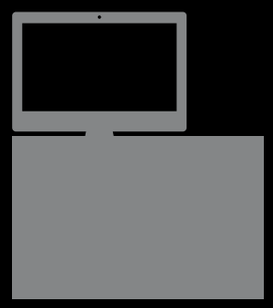
The waiter (CPU) will

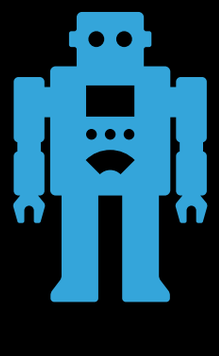
1. **take an order from a customer (CPU instruction)**
2. **break the order (instruction) down into micro operations (μOPs - grilling a burger, baking a pizza, ...)**
3. **schedule and execute the μOPs**
4. **complete the order (retire the instruction)**

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



Customer





Waiter

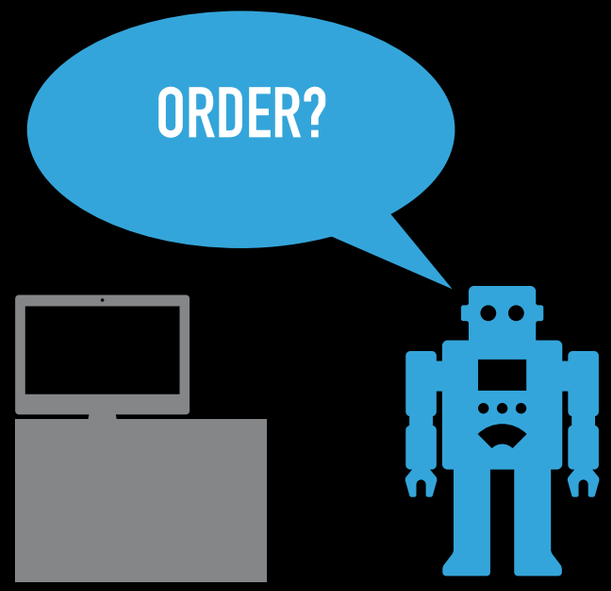
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION

Customer



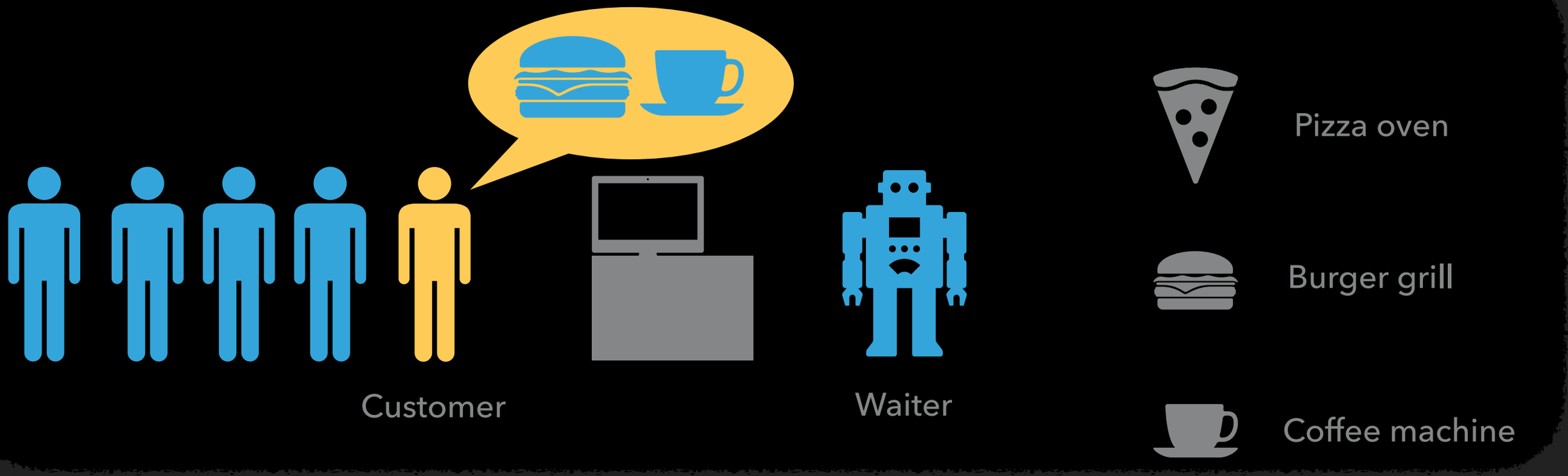
Waiter

Pizza oven

Burger grill

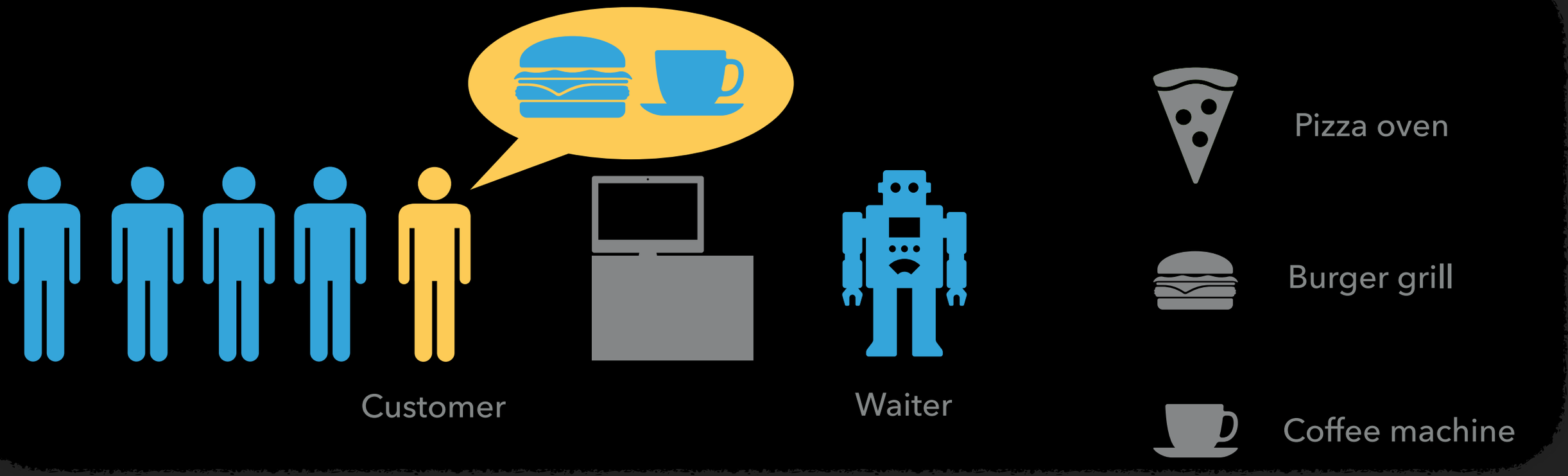
Coffee machine

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



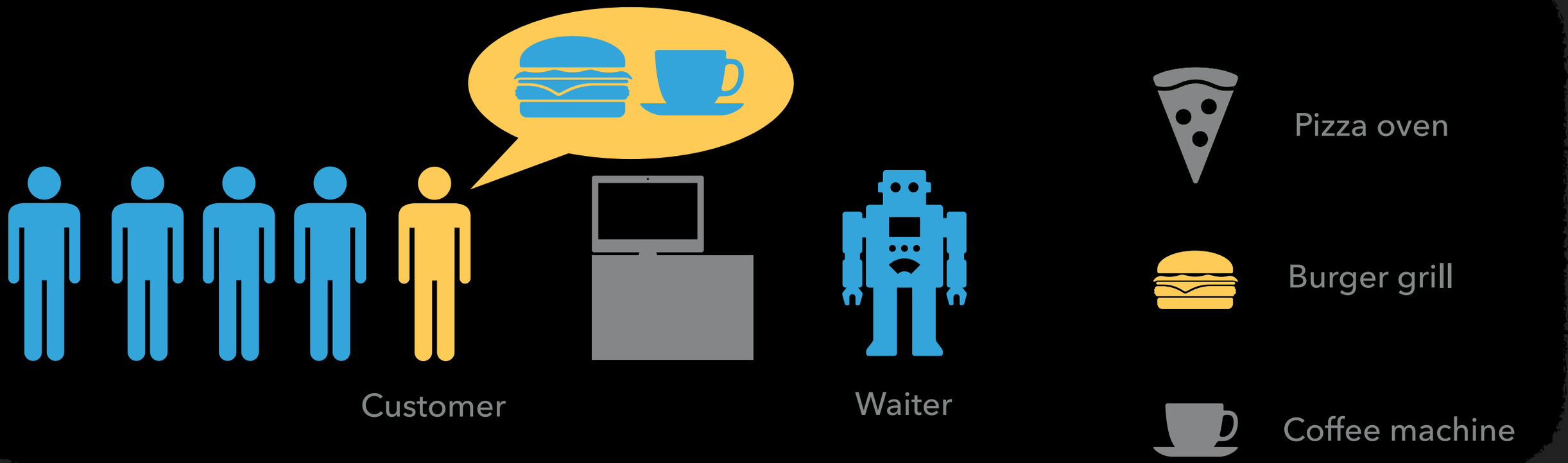
- Decode instruction into μOPs (“Burger”, “Coffee”)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



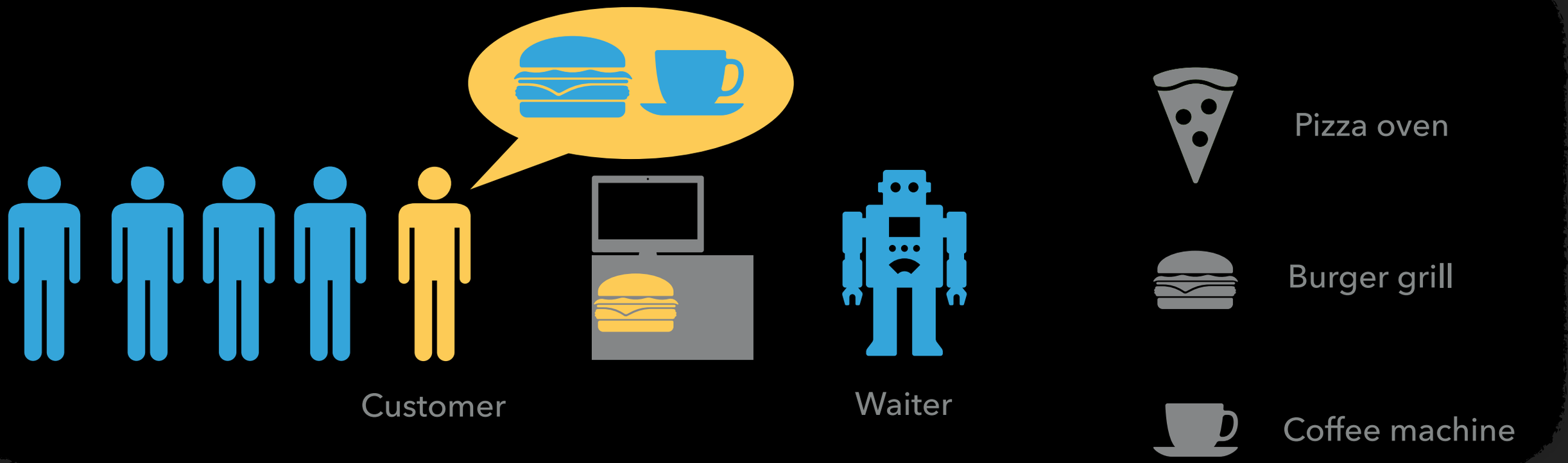
- Decode instruction into μOPs (“Burger”, “Coffee”)
- Schedule μOPs

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



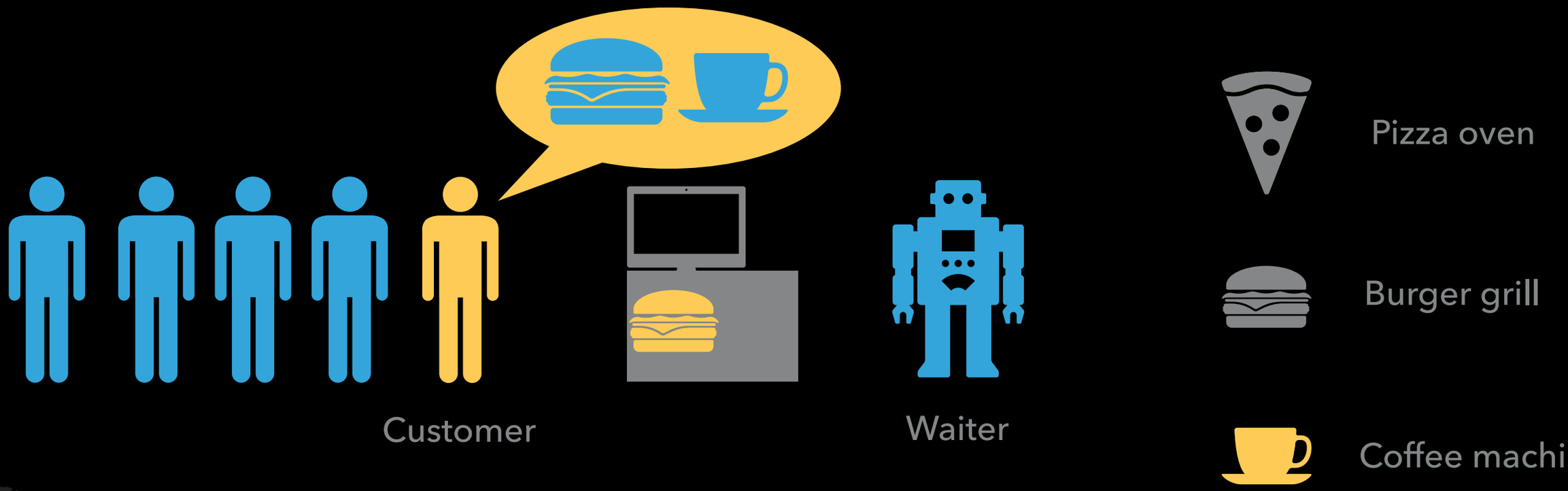
- Decode instruction into μOPs (“Burger”, “Coffee”)
- Schedule μOPs
  - run 1st μOP (grill the burger)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



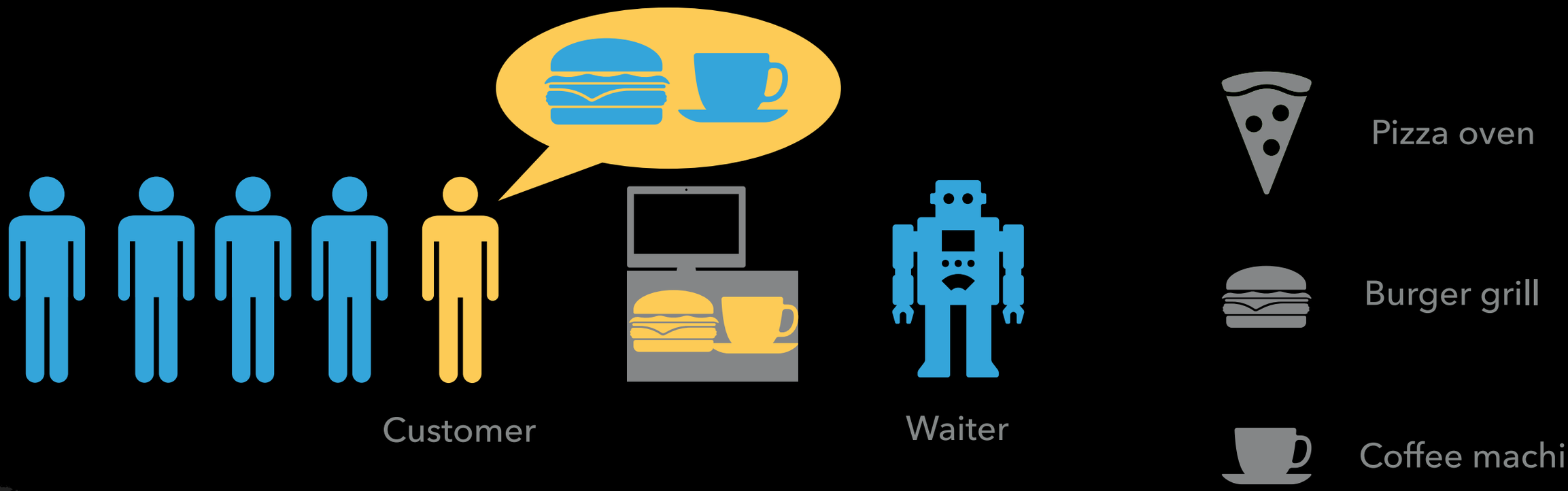
- Decode instruction into μOPs (“Burger”, “Coffee”)
- Schedule μOPs
  - run 1st μOP (grill the burger)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



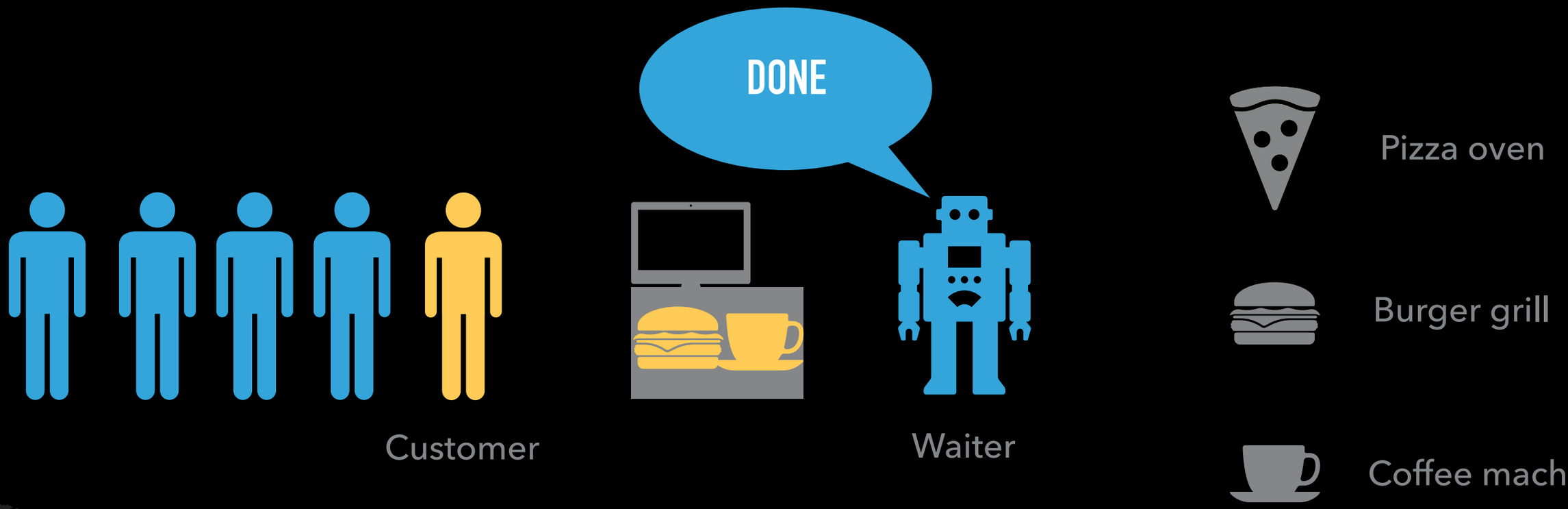
- Decode instruction into μOPs (“Burger”, “Coffee”)
- Schedule μOPs
  - run 1st μOP (grill the burger)
  - run 2nd μOP (brew coffee, serial execution)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



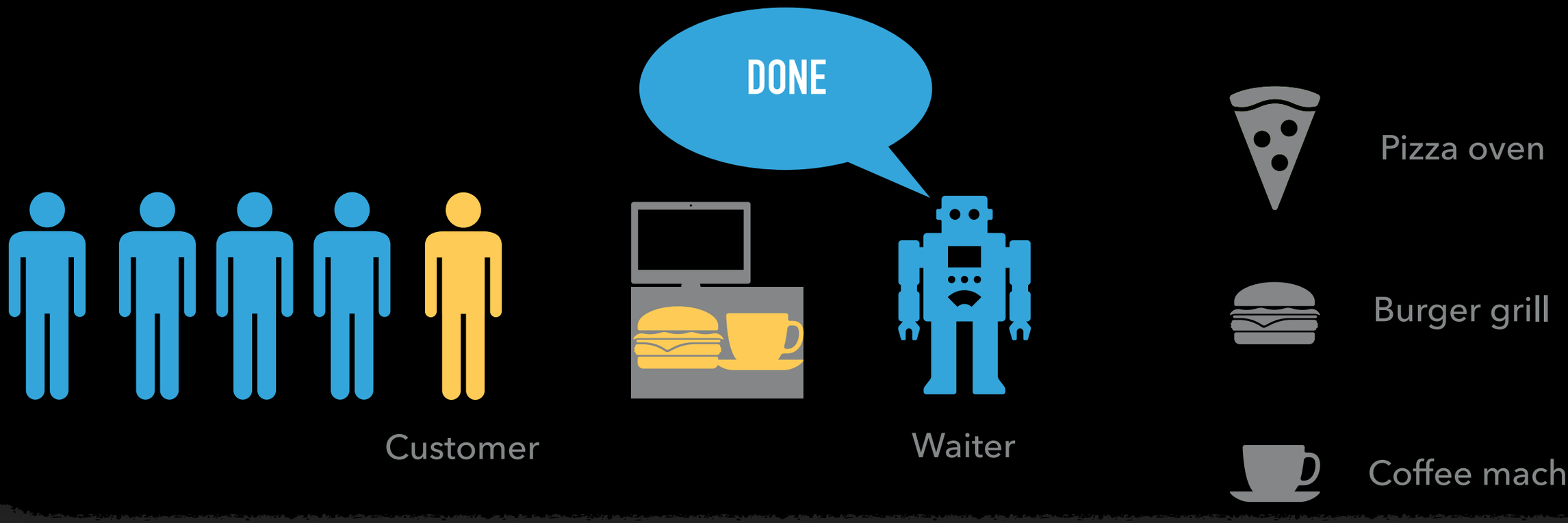
- Decode instruction into μOPs (“Burger”, “Coffee”)
- Schedule μOPs
  - run 1st μOP (grill the burger)
  - run 2nd μOP (brew coffee, serial execution)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



- Decode instruction into µOPs (“Burger”, “Coffee”)
- Schedule µOPs
  - run 1st µOP (grill the burger)
  - run 2nd µOP (brew coffee, serial execution)
- Retire instruction (customer)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



- One customer[1] after another (**in order**)
- Each part of the order [2] executed **serially**

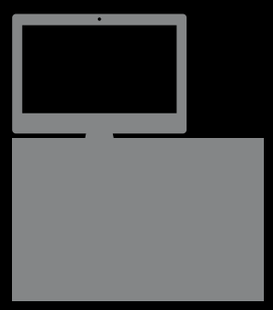
I.e. first the burger, then the coffee

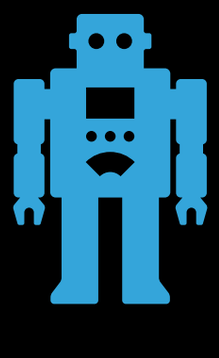
- PRO: Easy to implement and understand
- CON: Slow because resources[3] not utilised fully

[1] customer == CPU instruction [2] part == µOP - micro operation [3] oven, grill, coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION

Customer





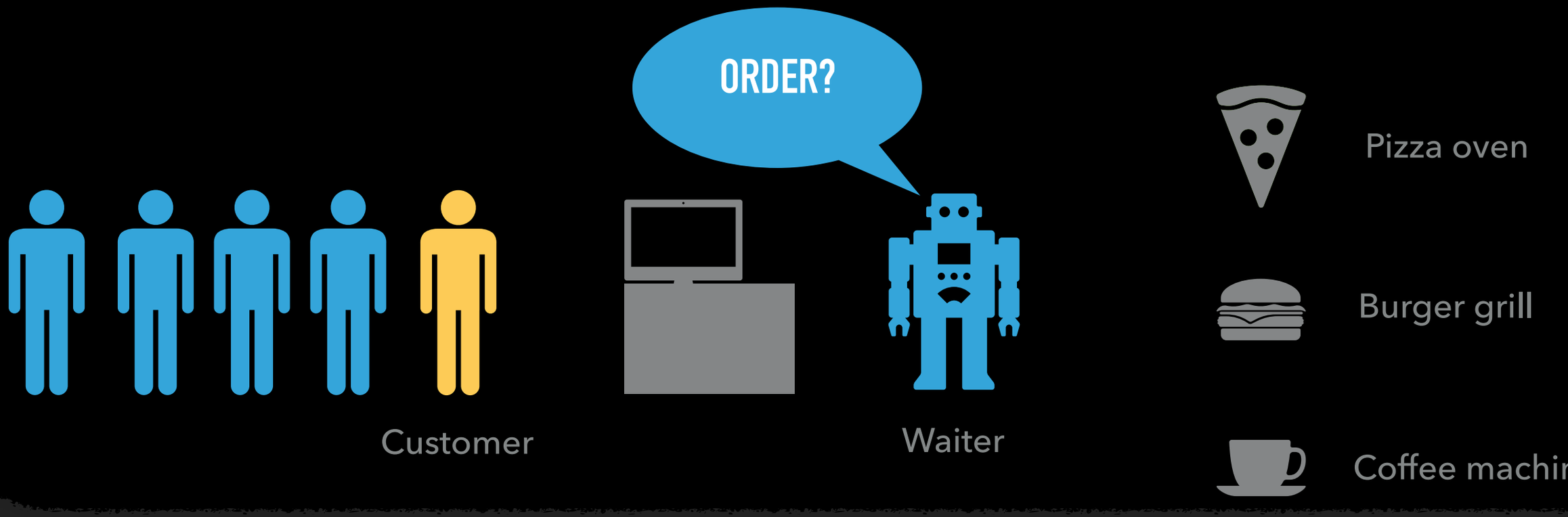
Waiter

Pizza oven

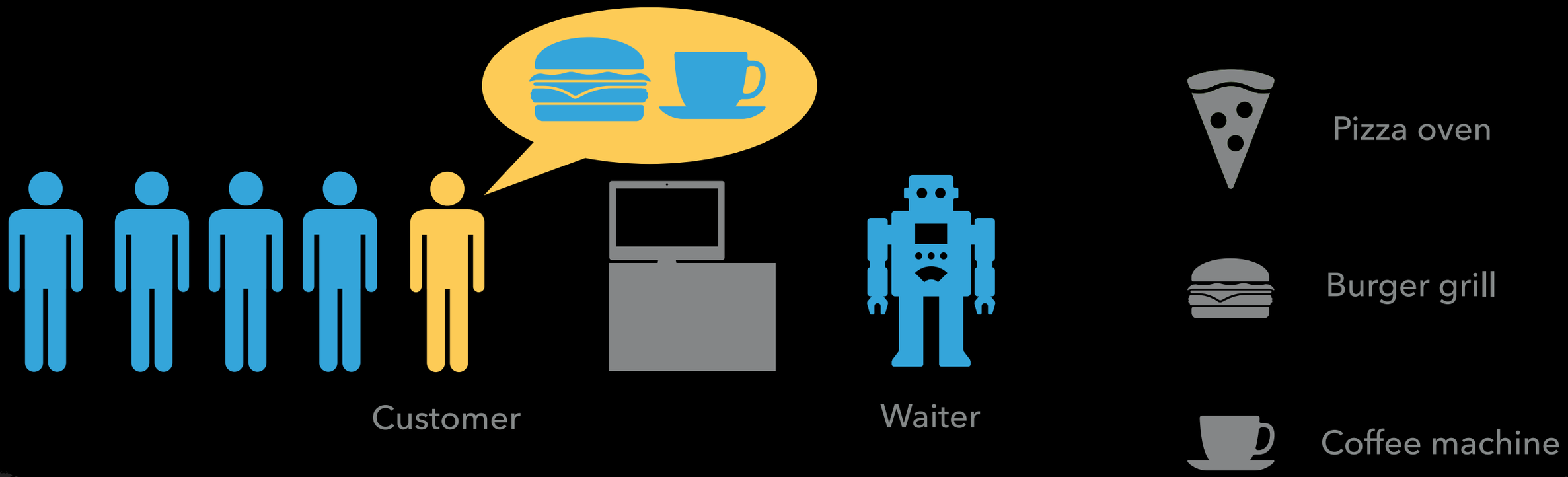
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION

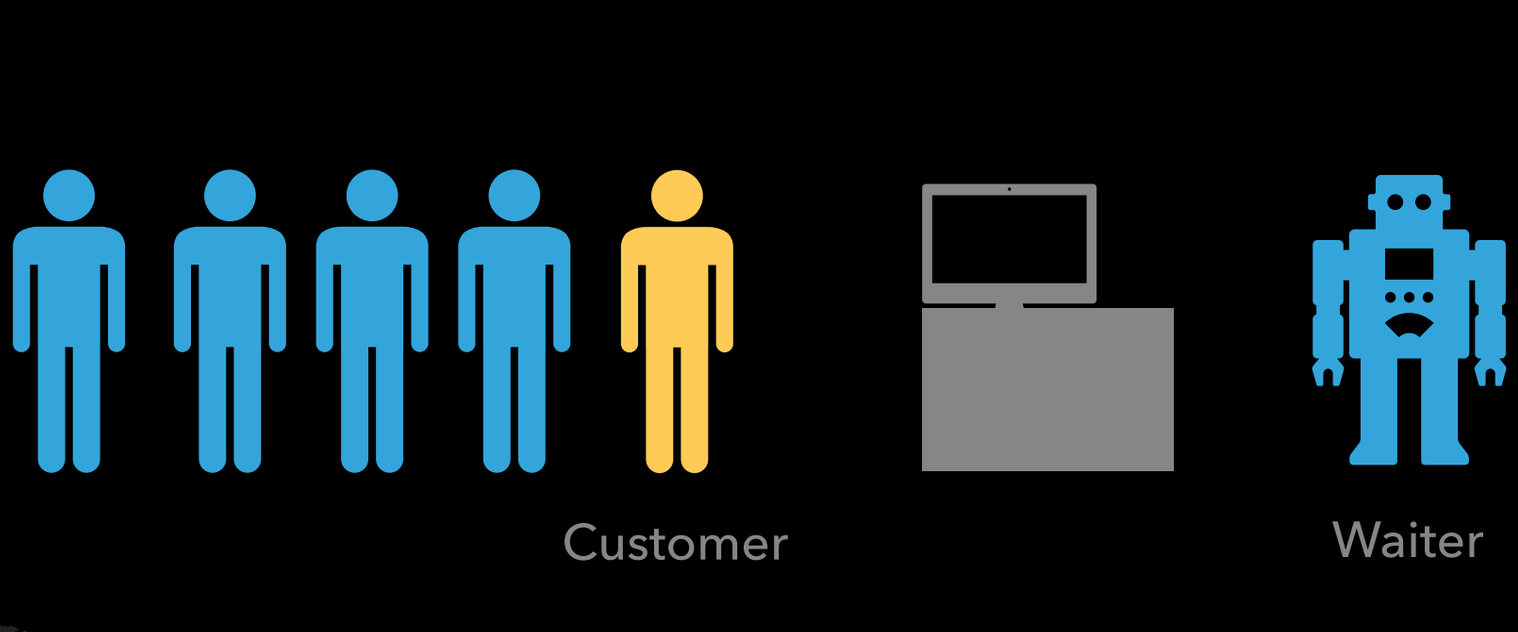


CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



- Decode instruction into μOPs

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



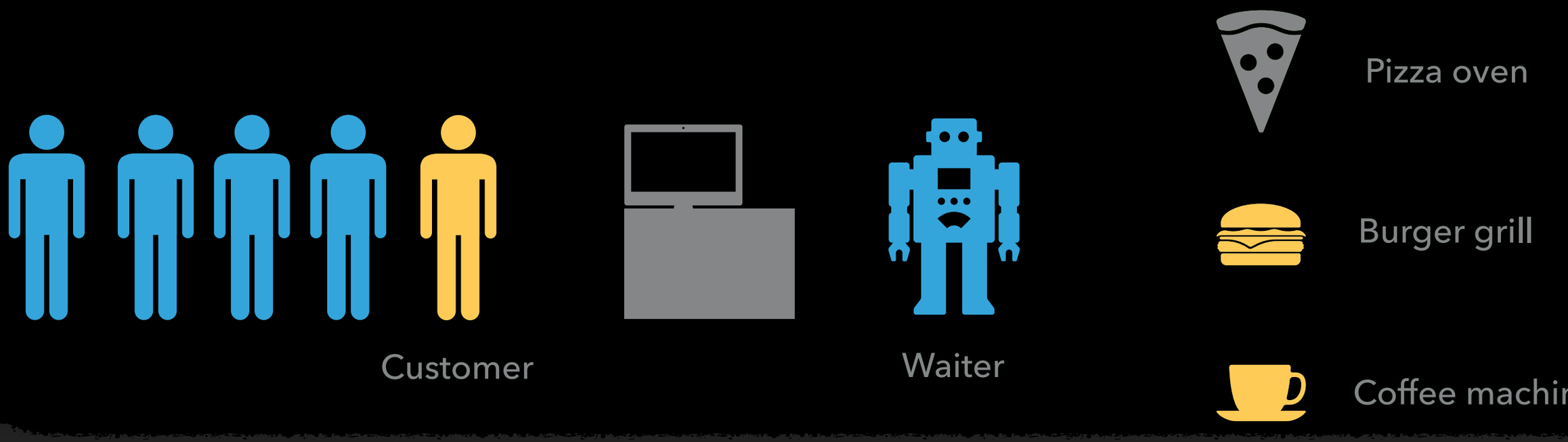

Pizza oven


Burger grill


Coffee machine

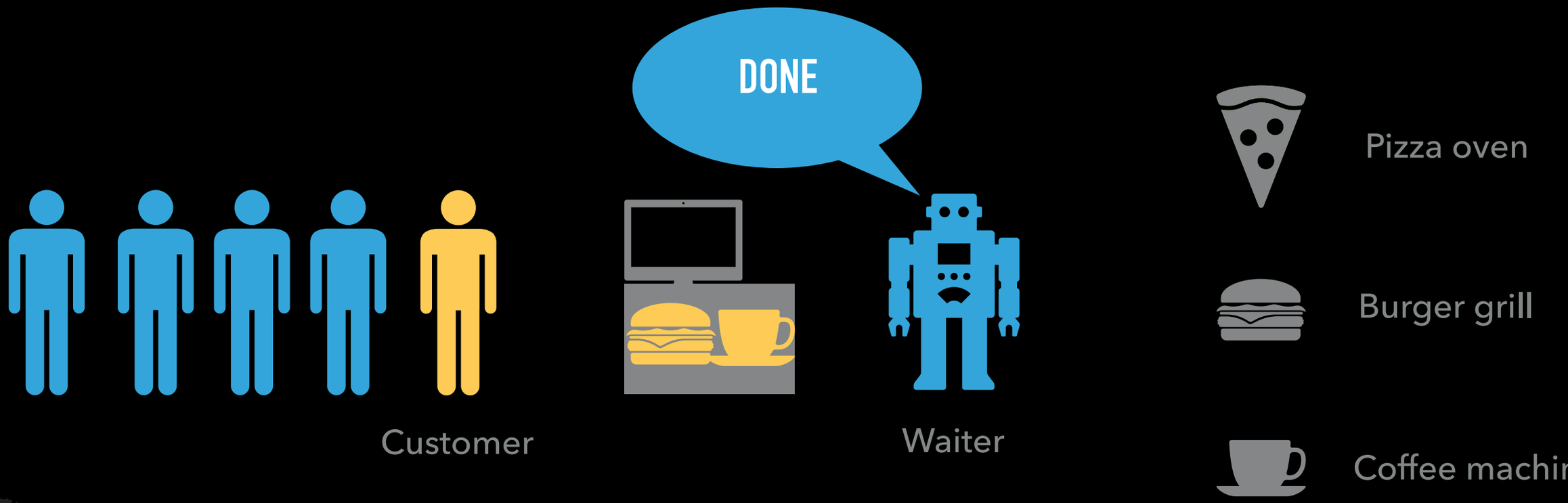
- Decode instruction into µOPs
- Schedule µOPs

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



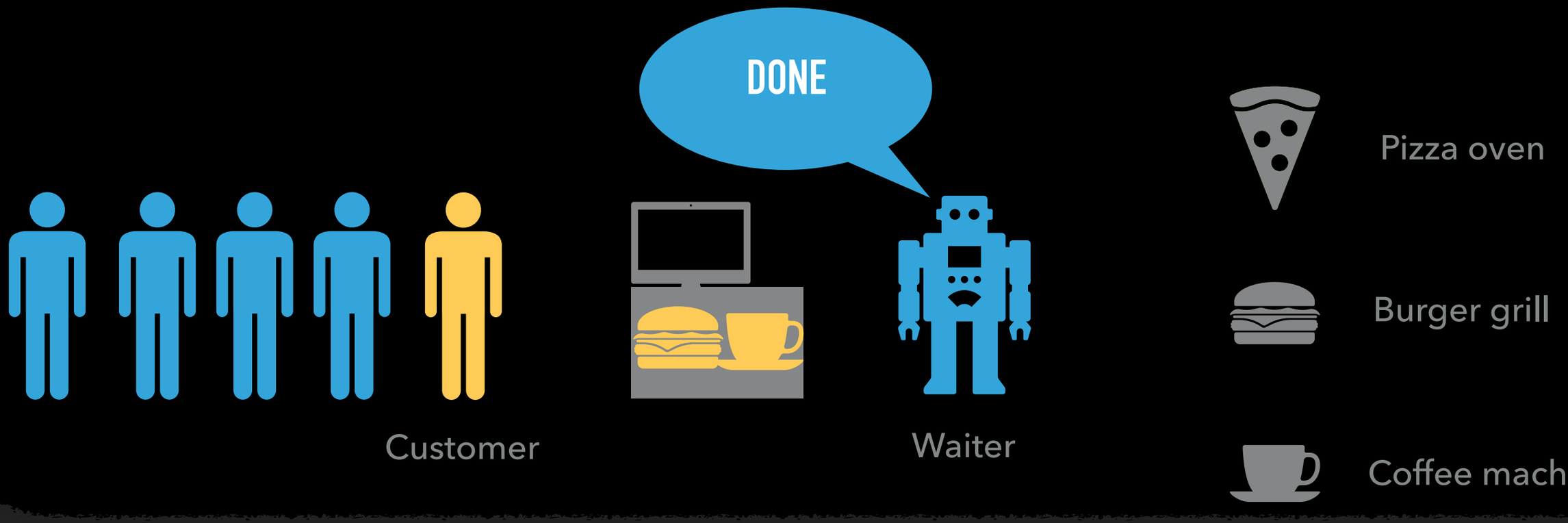
- Decode instruction into μOPs
- Schedule μOPs
  - run 1st μOP and 2nd μOP (parallel execution of μOPs)

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



- Decode instruction into µOPs
- Schedule µOPs
  - run 1st µOP and 2nd µOP (parallel execution of µOPs)
- retire instruction (customer)

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION

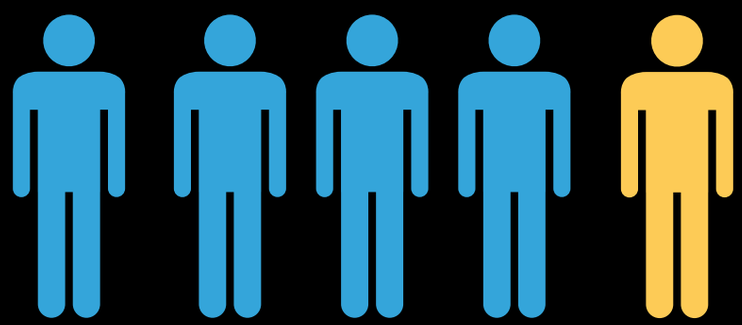


- One customer[1] after another (**in order**)
- Each part of the order [2] executed **in parallel**

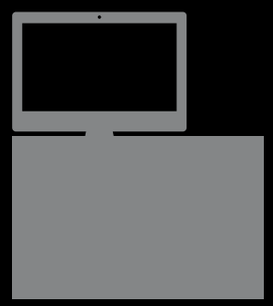
I.e. burger and coffee prepared at the same time

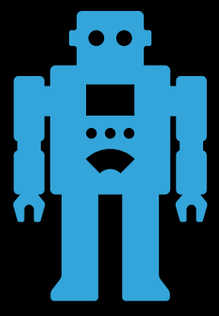
- PRO: **Faster bc. of better resource utilisation.**
- CON: Still not perfect, more complex

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer





Waiter

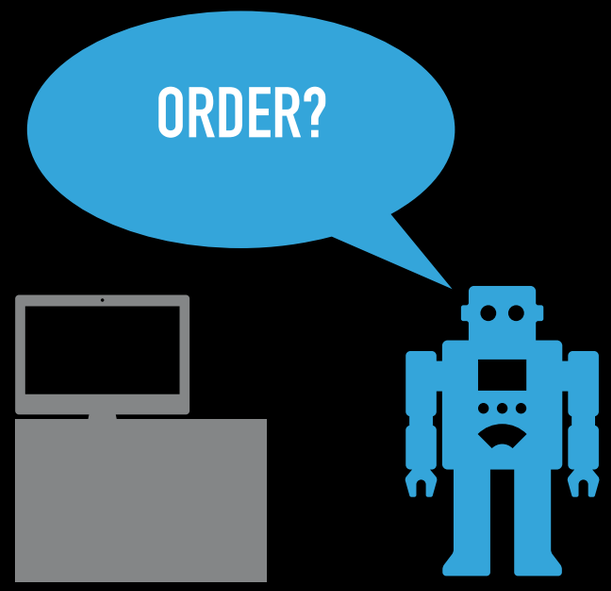
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

Customer



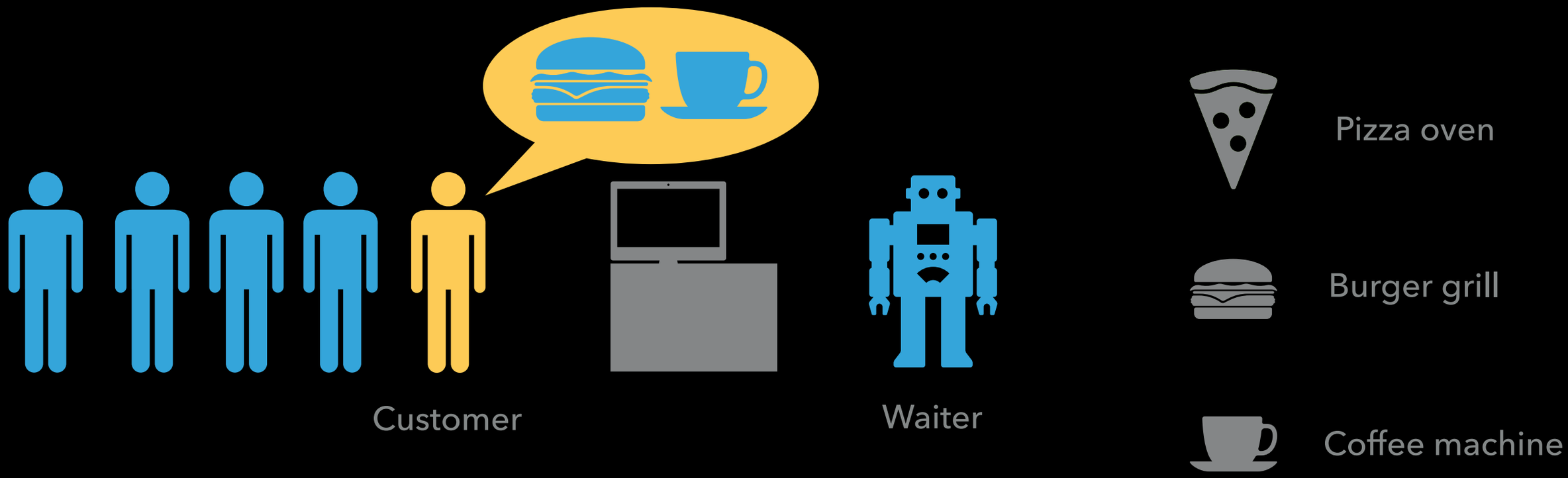
Waiter

Pizza oven

Burger grill

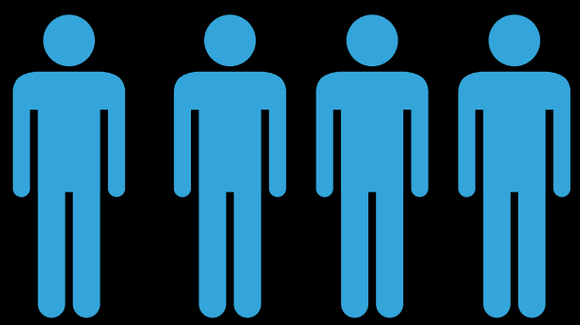
Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

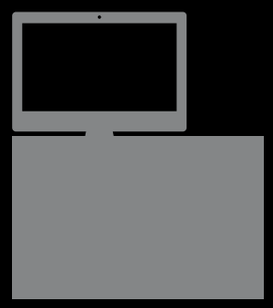


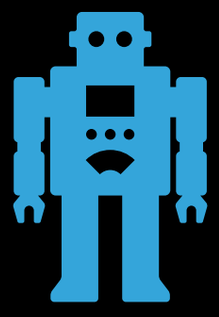
CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer

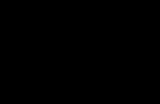




Waiter

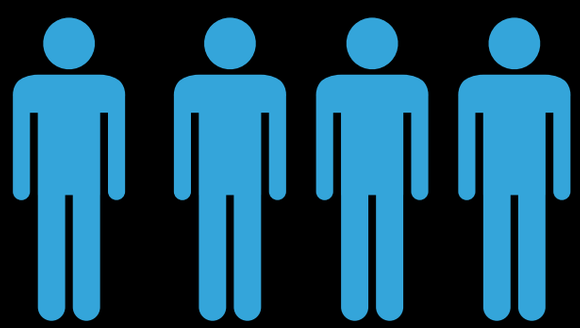
Pizza oven

Burger grill

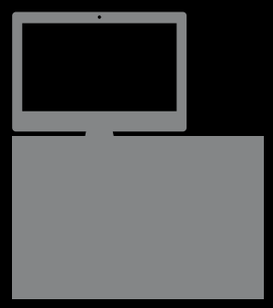


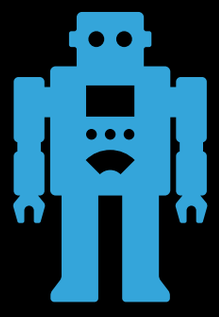
Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer





Waiter

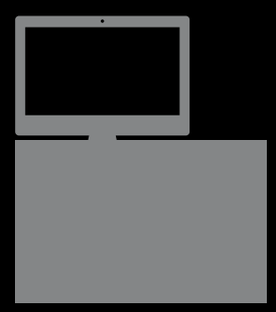
Pizza oven

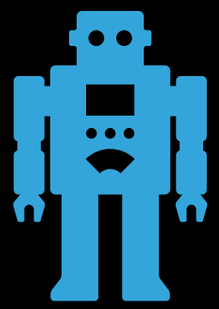
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

Customer





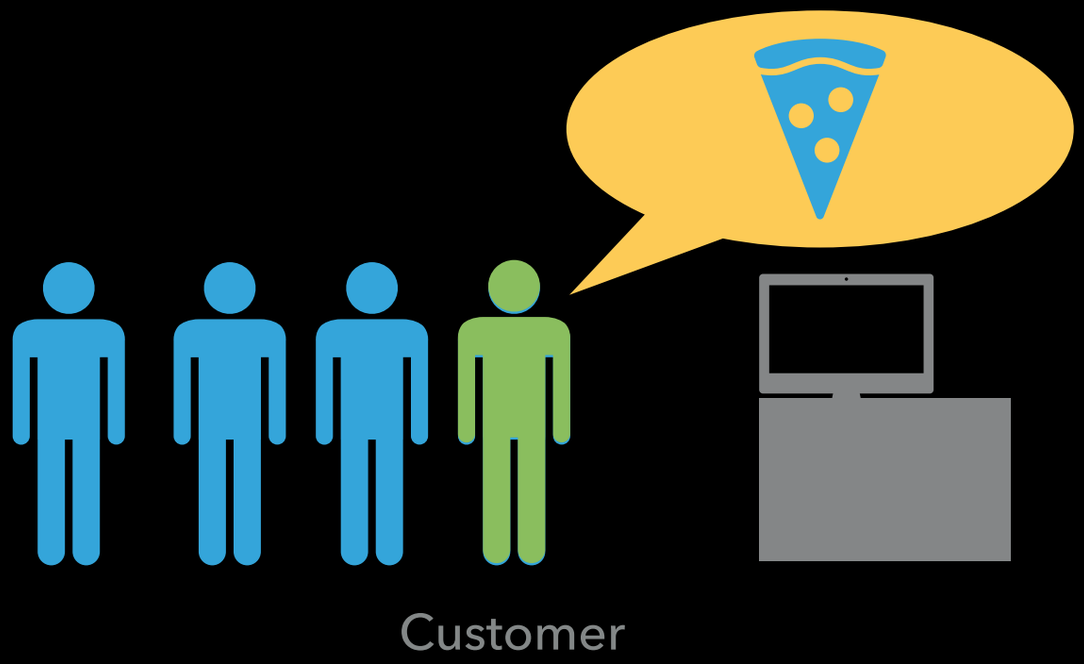
Waiter

Pizza oven

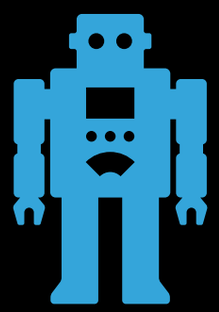
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer



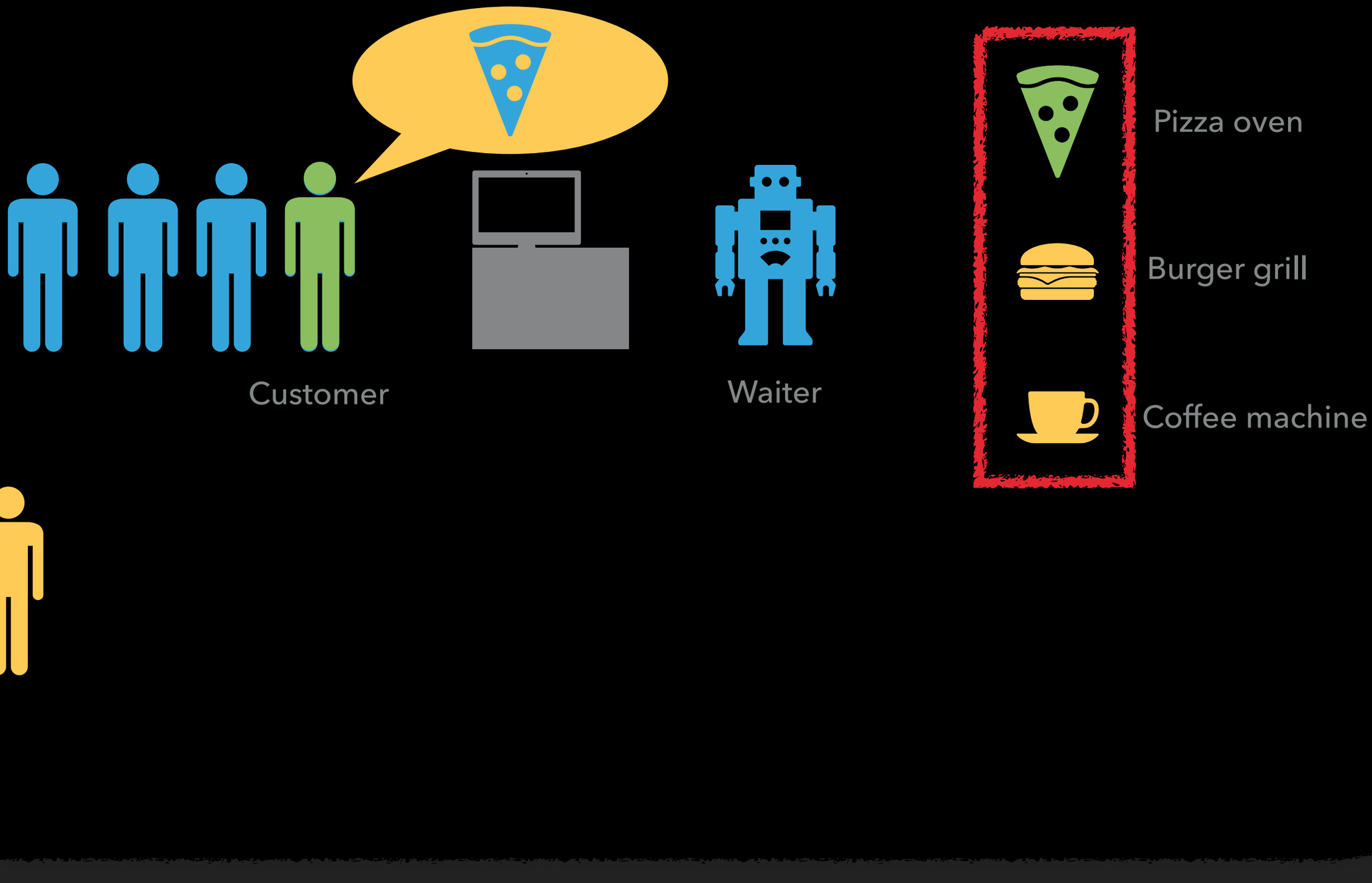
Waiter

Pizza oven

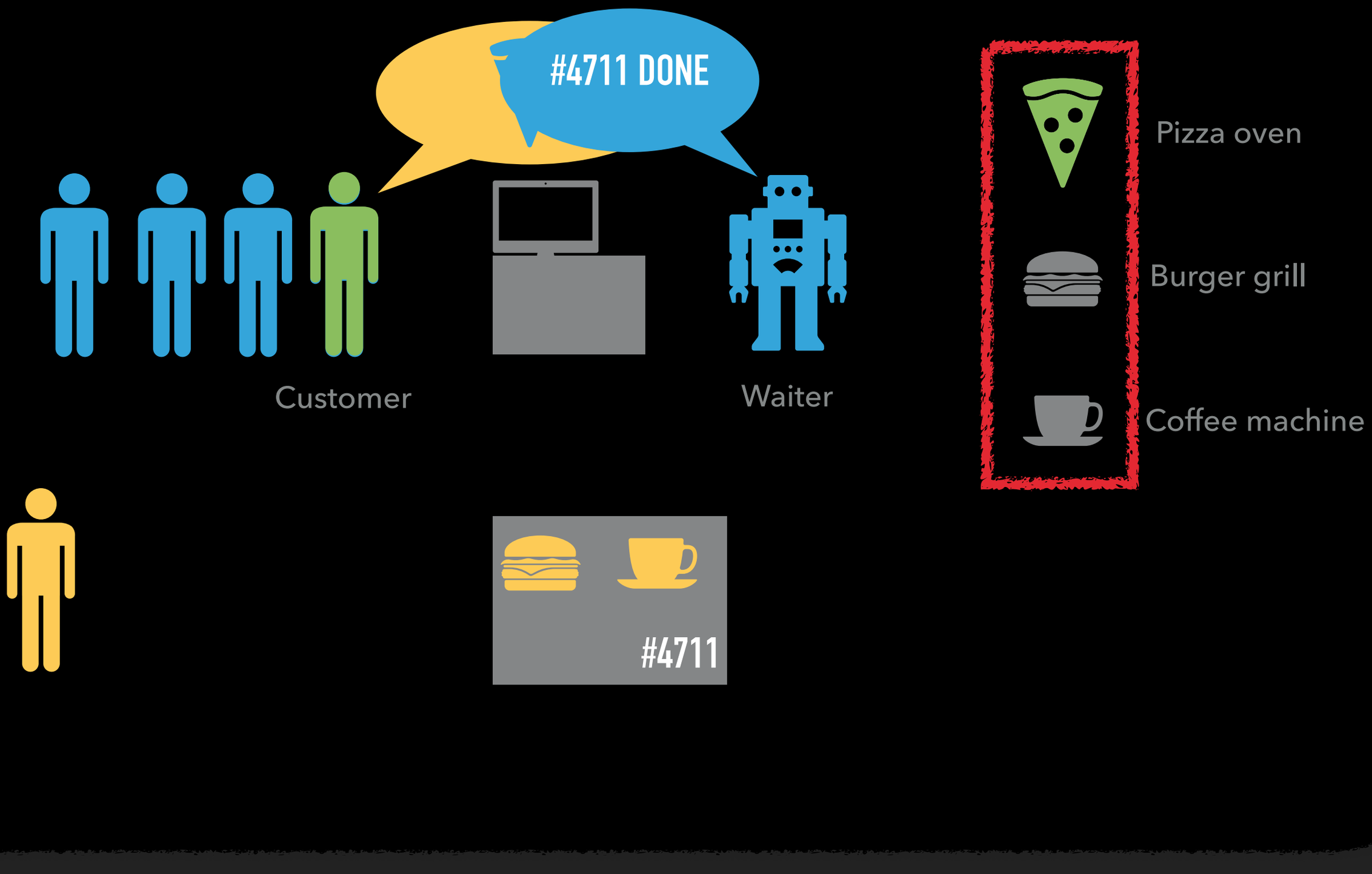
Burger grill

Coffee machine

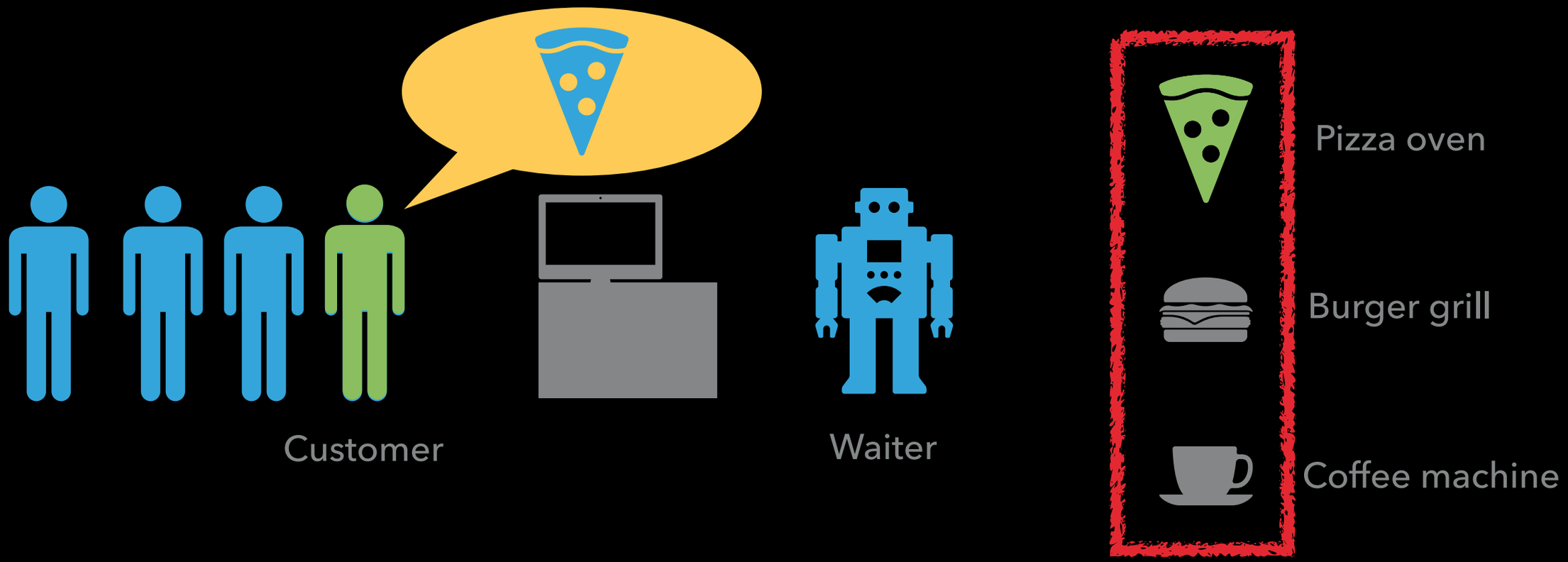
CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



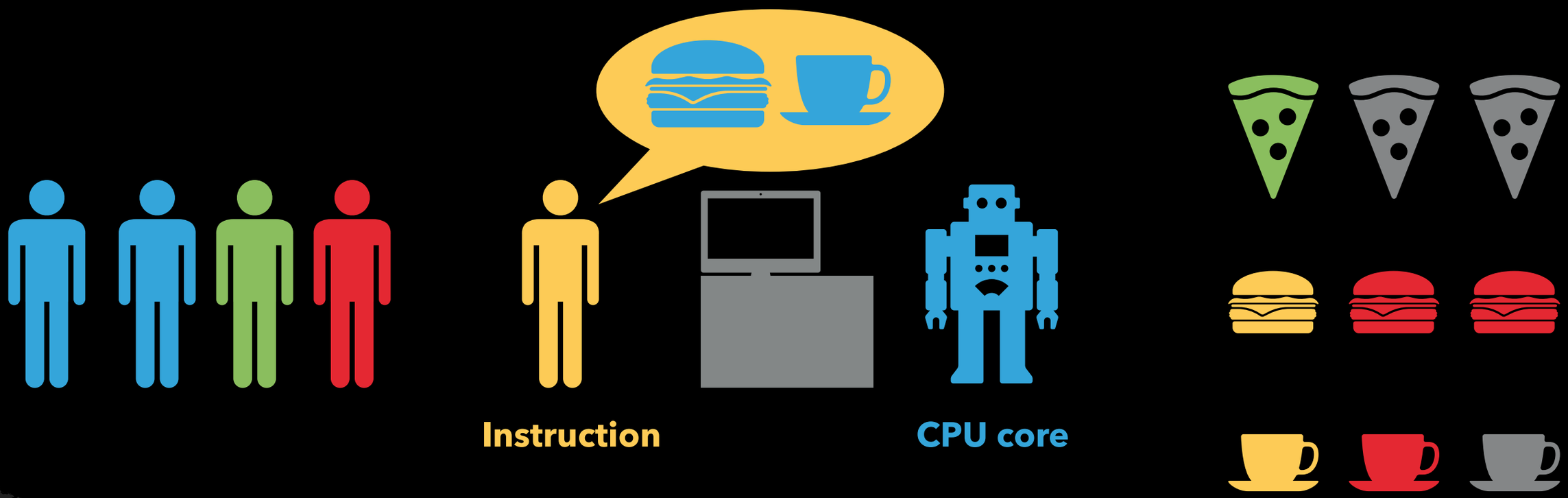
- Multiple customers' orders **executed in parallel**[1] and **delivered** (retired) **in order**

I.e. multiple orders prepared at the same time

- PRO: **Faster because resources are utilised even better**
- CON: More difficult to implement

[1] this is called *superscalar*

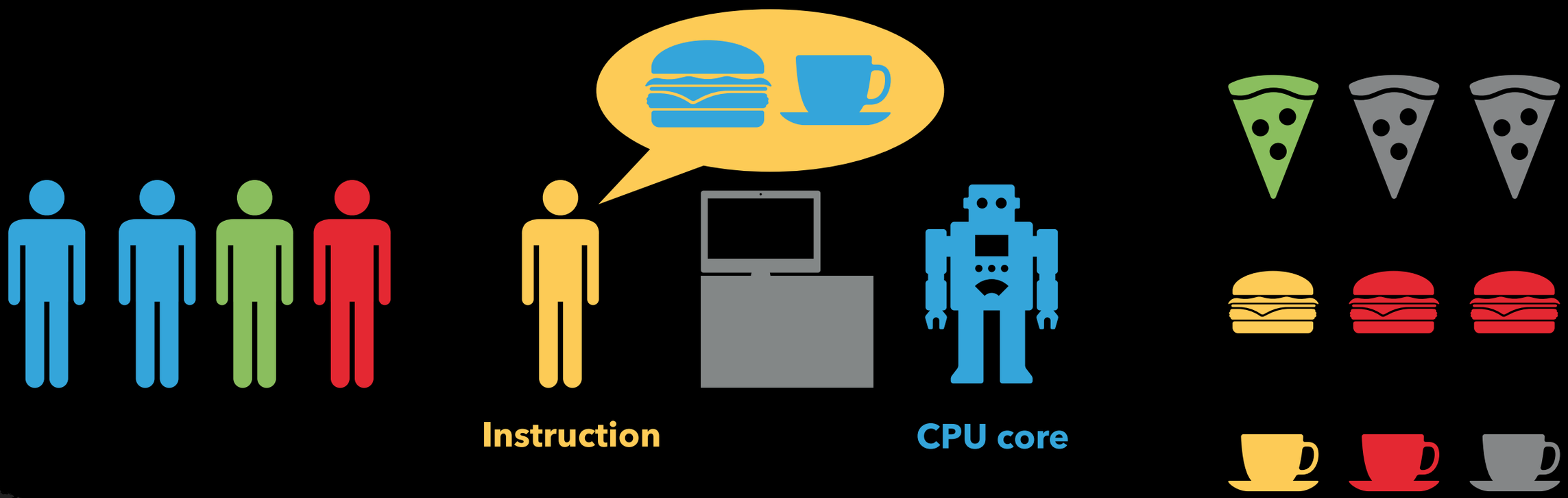
CONFIDENTIAL BURGERS INC.



Adding more resources increase parallelism & throughput.

This is all on one CPU core.

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

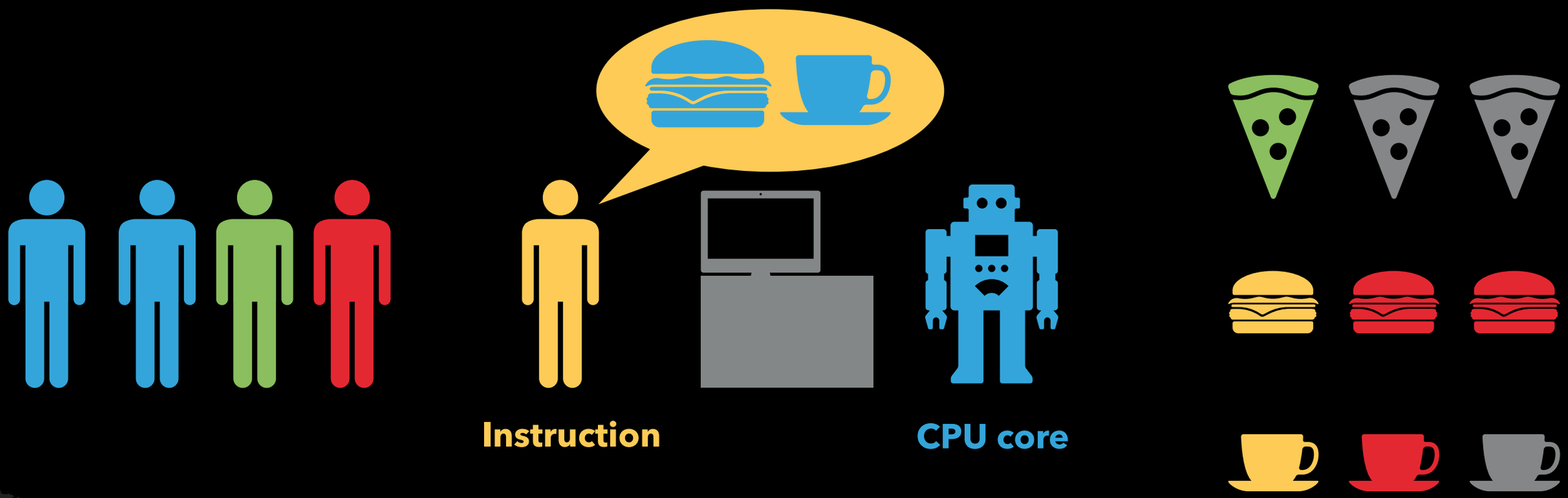


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

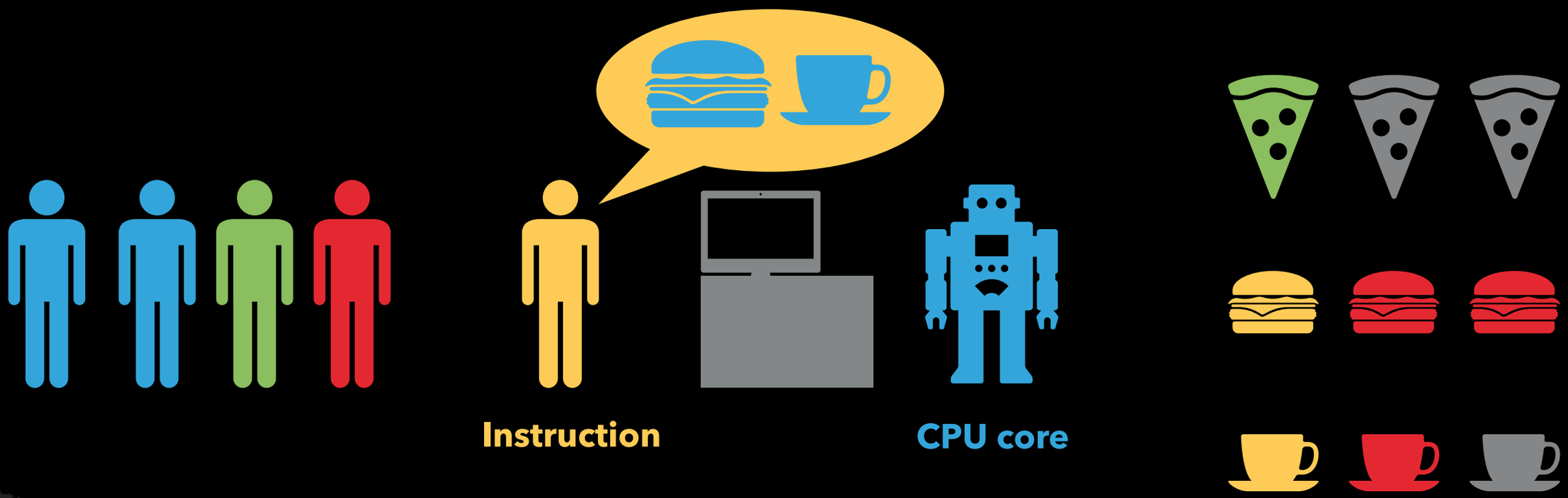


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

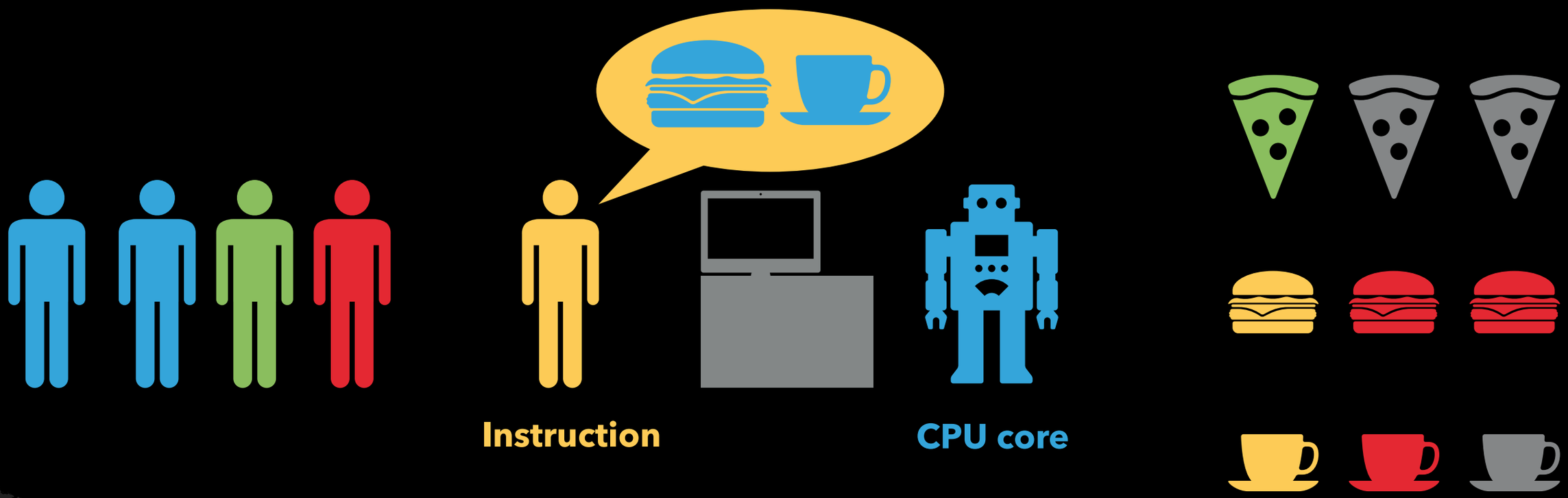


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

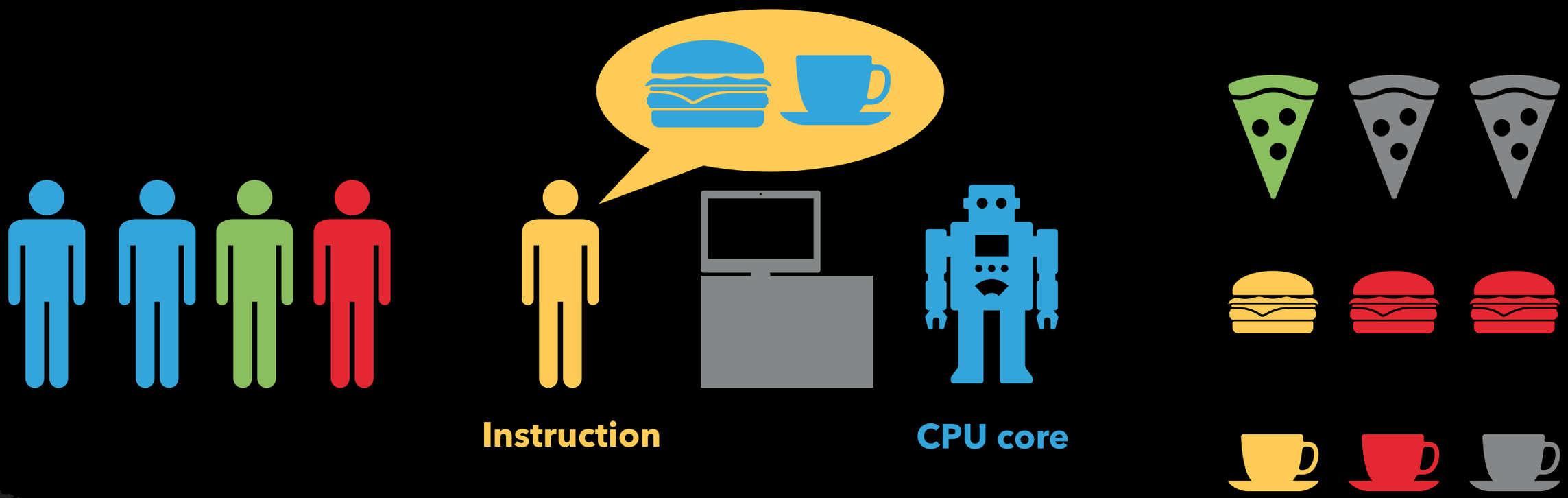


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

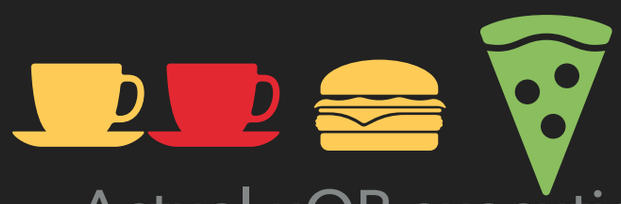
Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



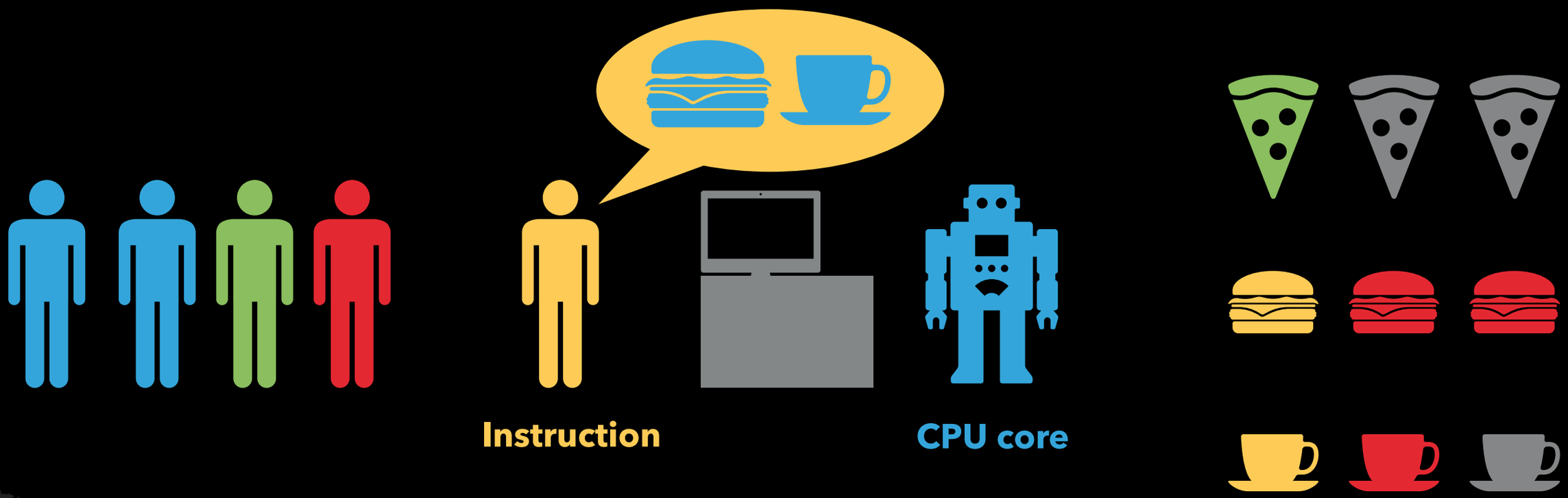
The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.



Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

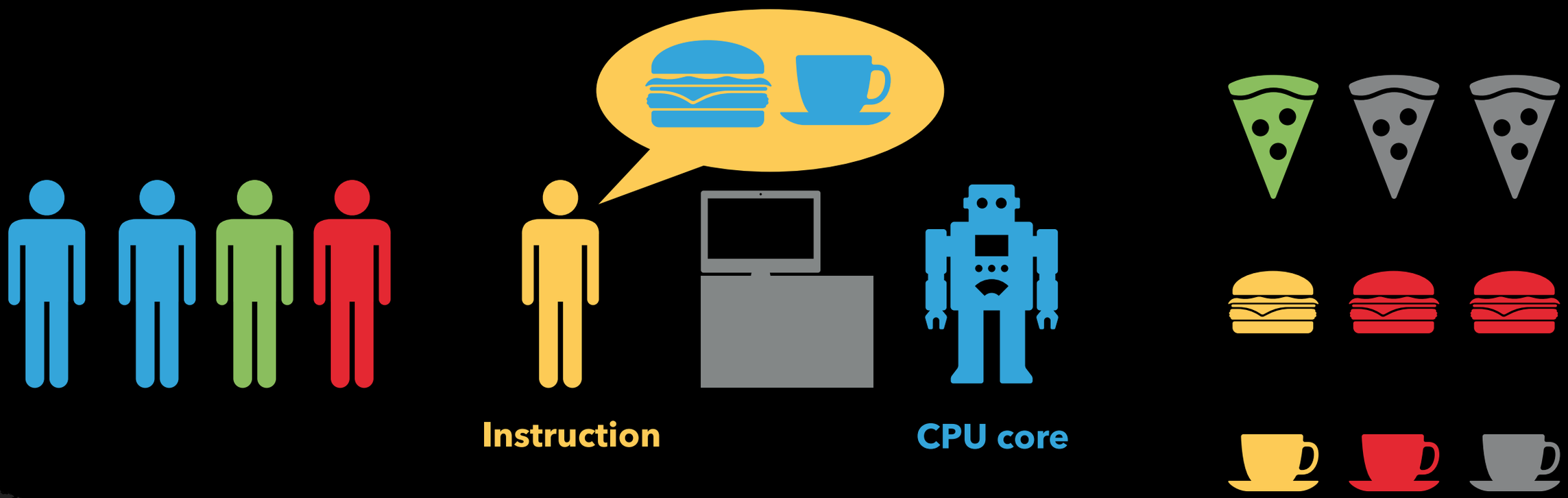


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

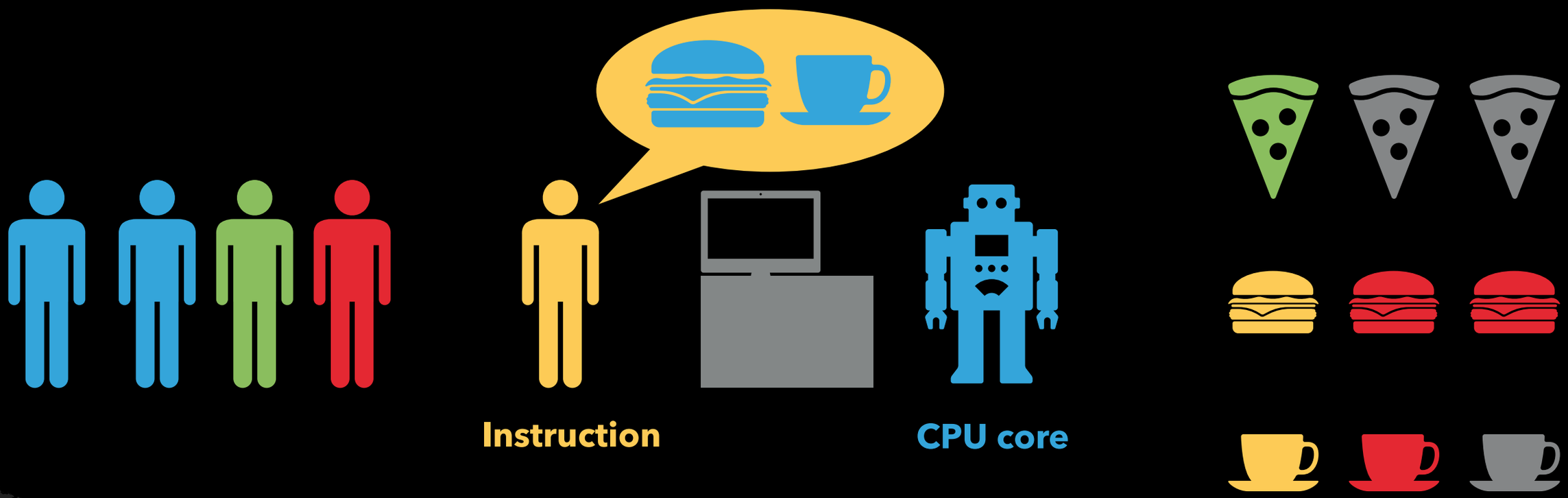


The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



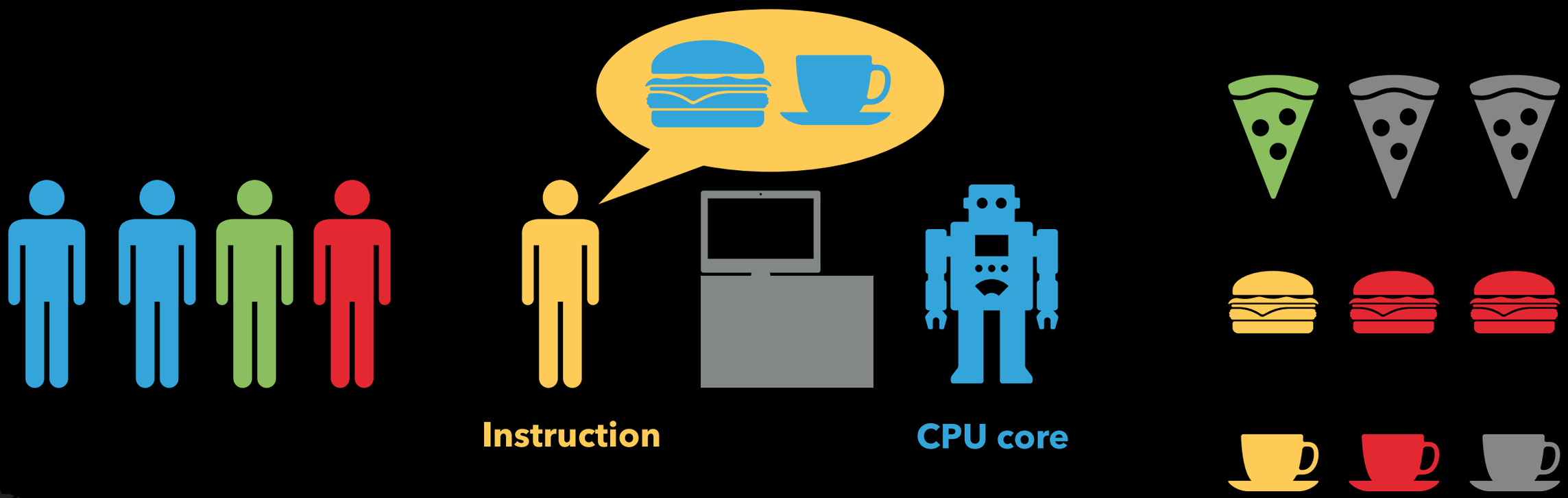
The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.

Actual µOP execution order

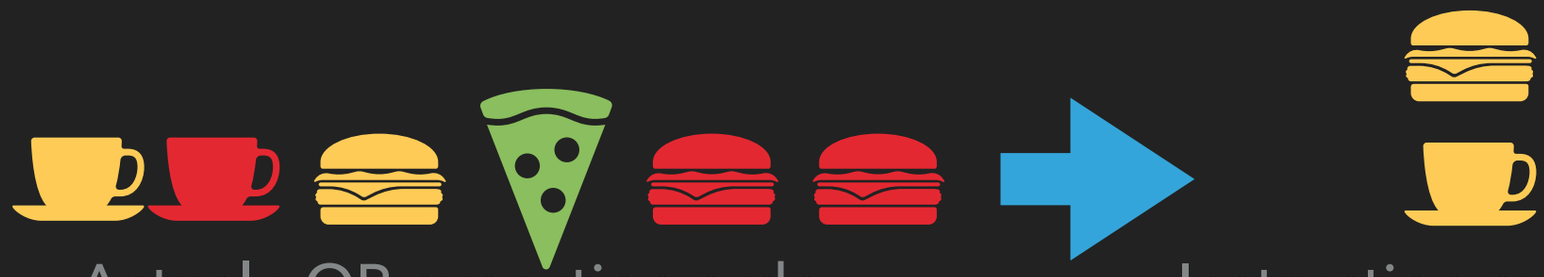
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction.

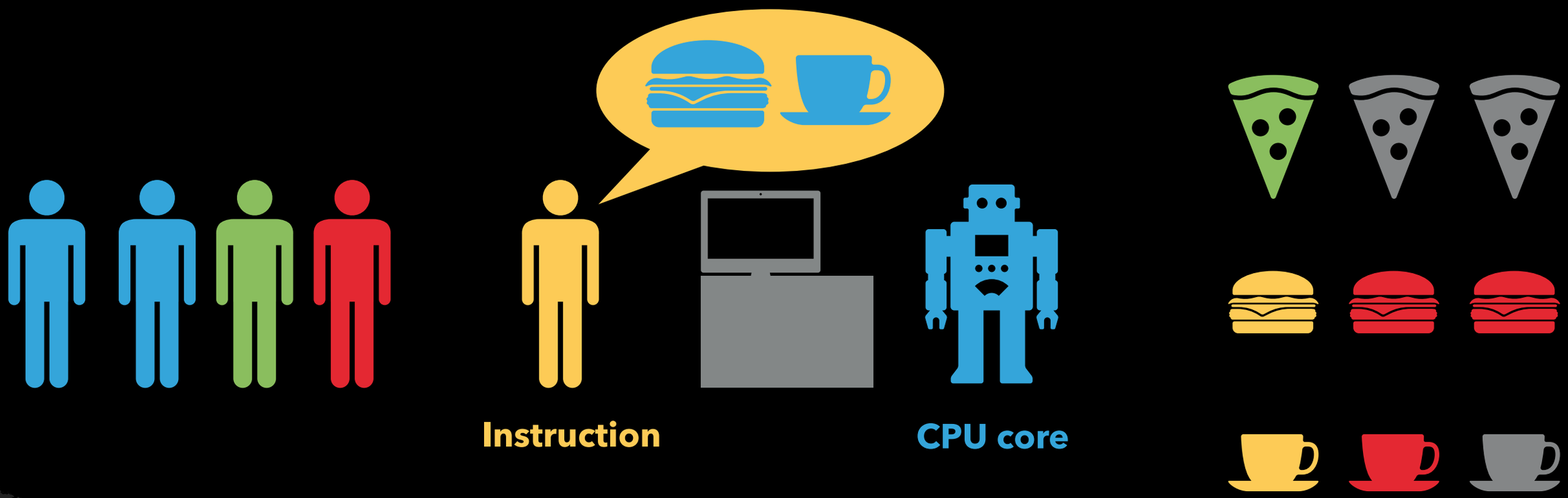
The CPU ensures that red is seen before green.



Actual µOP execution order

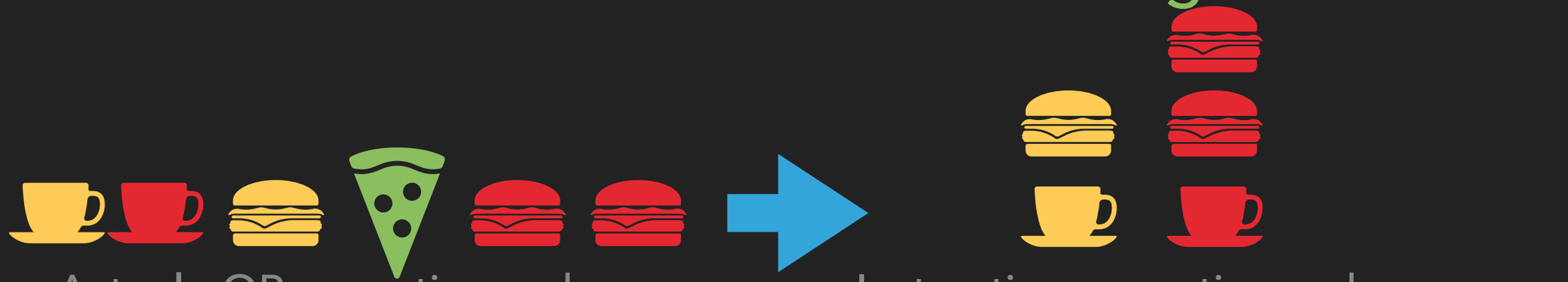
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction.

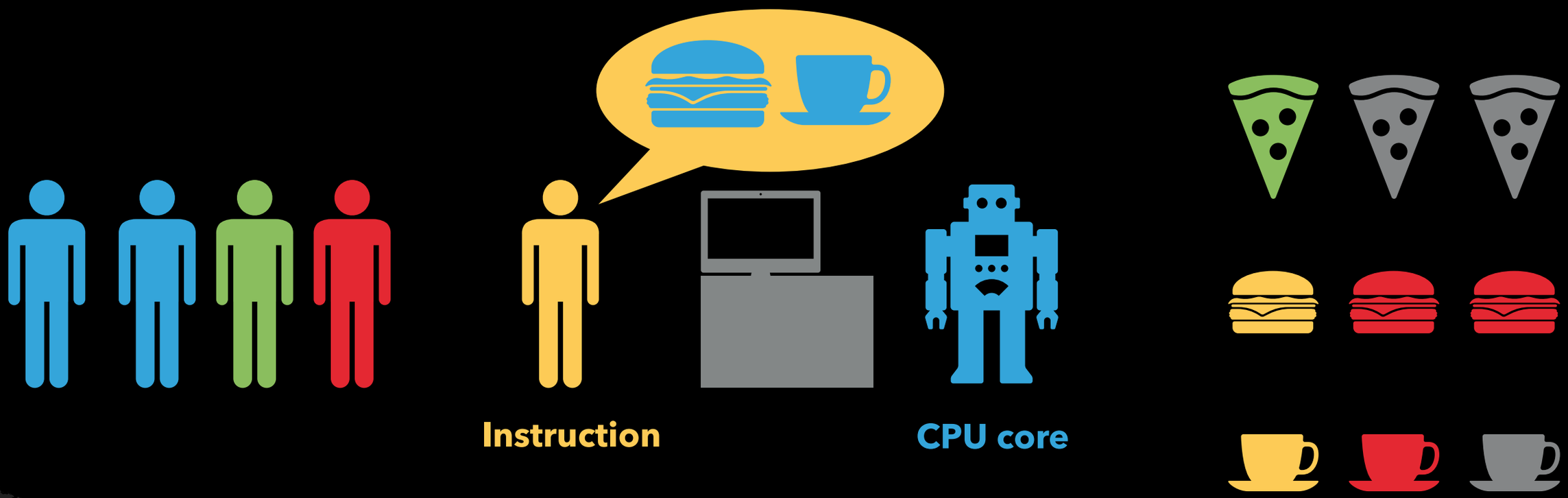
The CPU ensures that red is seen before green.



Actual µOP execution order

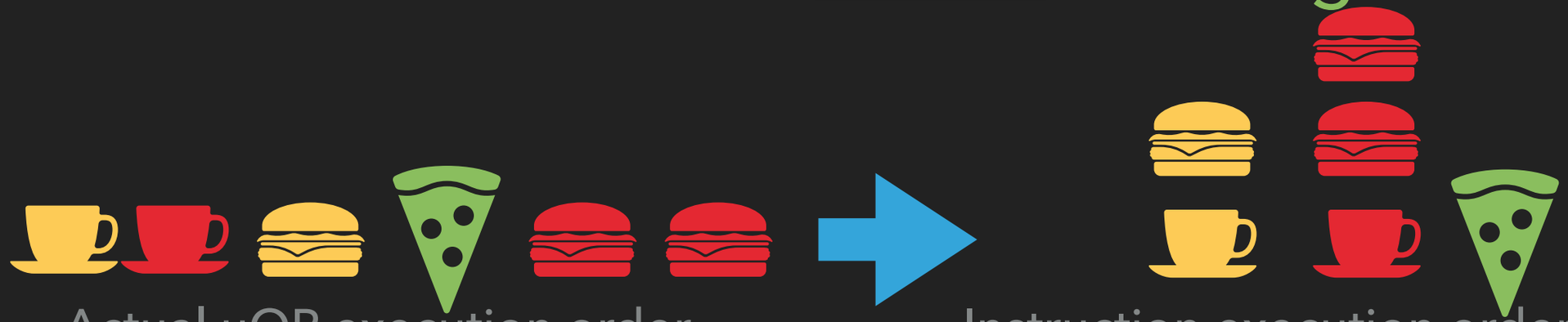
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction.

The CPU ensures that red is seen before green.



WHY DID MELTDOWN & SPECTRE HAPPEN?

WHY DID MELTDOWN & SPECTRE HAPPEN?

- Accident, malice, incompetence?

WHY DID MELTDOWN & SPECTRE HAPPEN?

- Accident, malice, incompetence?
- Or just a very advanced and complex system designed to be as fast as possible?

WHY DID MELTDOWN & SPECTRE HAPPEN?

- Accident, malice, incompetence?
- Or just a very advanced and complex system designed to be as fast as possible?
- Lessons learned: Complex systems have complex side effects!

WHY DID MELTDOWN & SPECTRE HAPPEN?

- Accident, malice, incompetence?
- Or just a very advanced and complex system designed to be as fast as possible?
- Lessons learned: Complex systems have complex side effects!

...

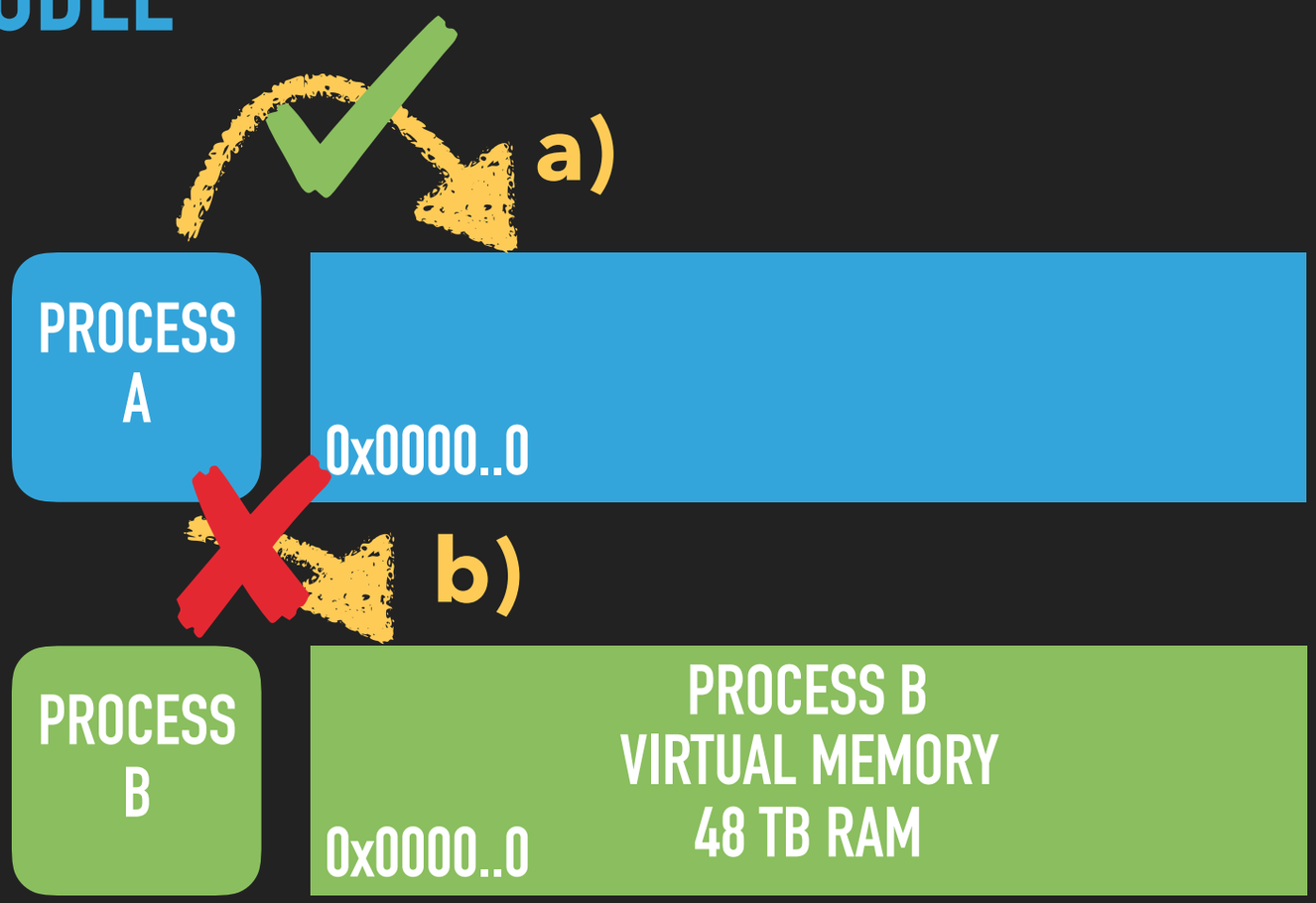
MISUNDERSTANDINGS AND NEGLECT CREATE MORE CONFUSION IN THIS WORLD THAN TRICKERY AND MALICE. AT ANY RATE, THE LAST TWO ARE CERTAINLY MUCH LESS FREQUENT.

Goethe's The Sorrows of Young Werther

INTERLUDE

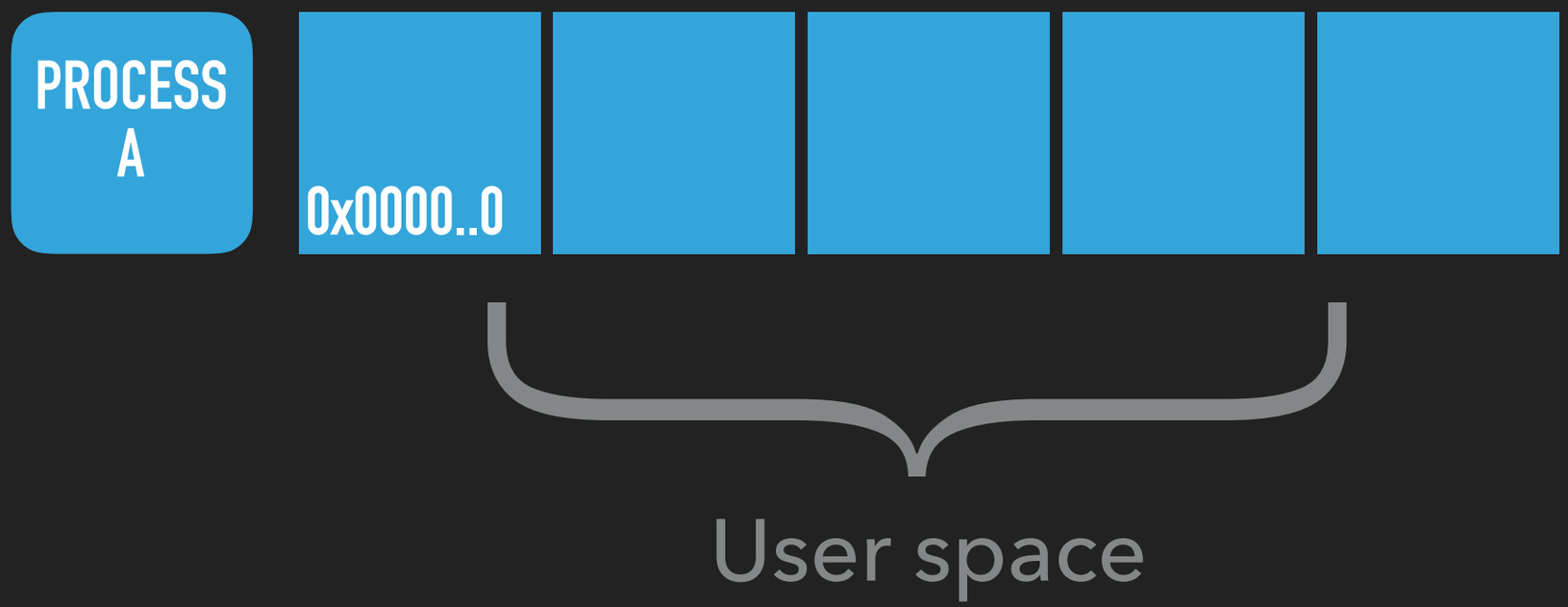
MEMORY MODEL

MEMORY MODEL

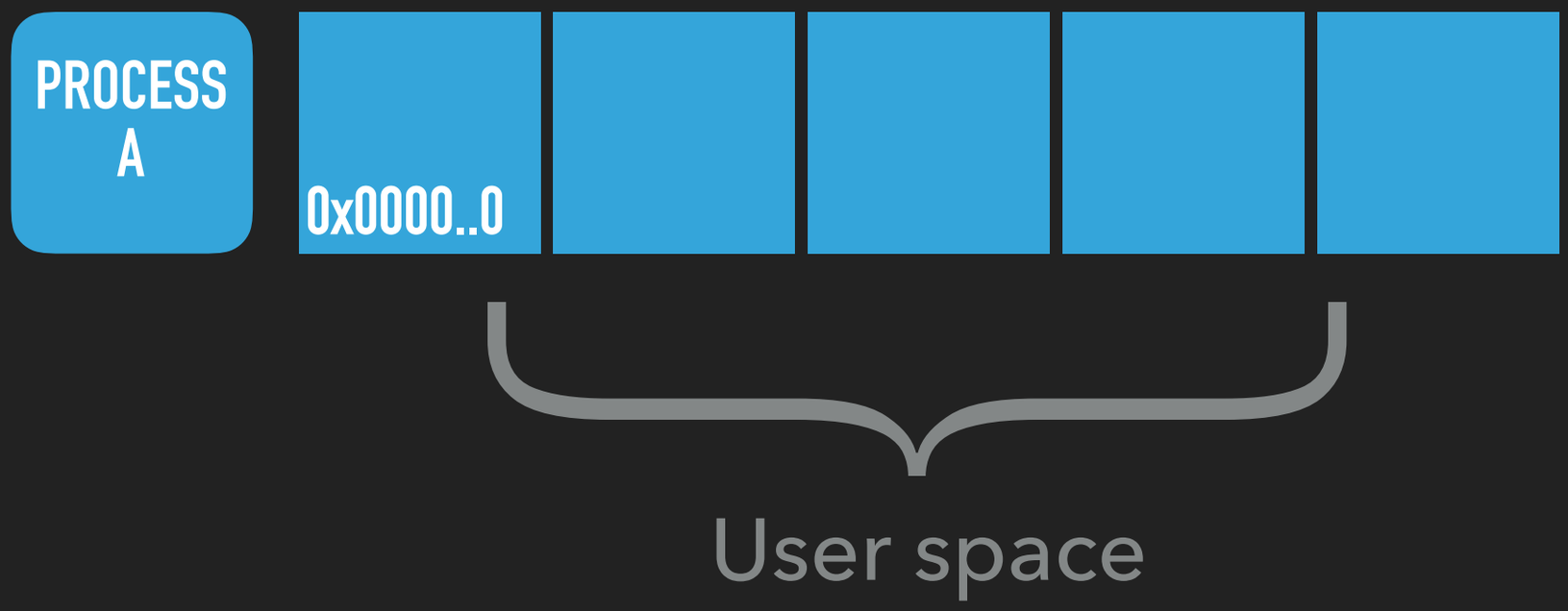


- CPU and OS isolate processes memory from each other
- *Virtual Memory* gives each process its own *address space*
- Each address space starts at “virtual address 0x000..0”

MEMORY MODEL

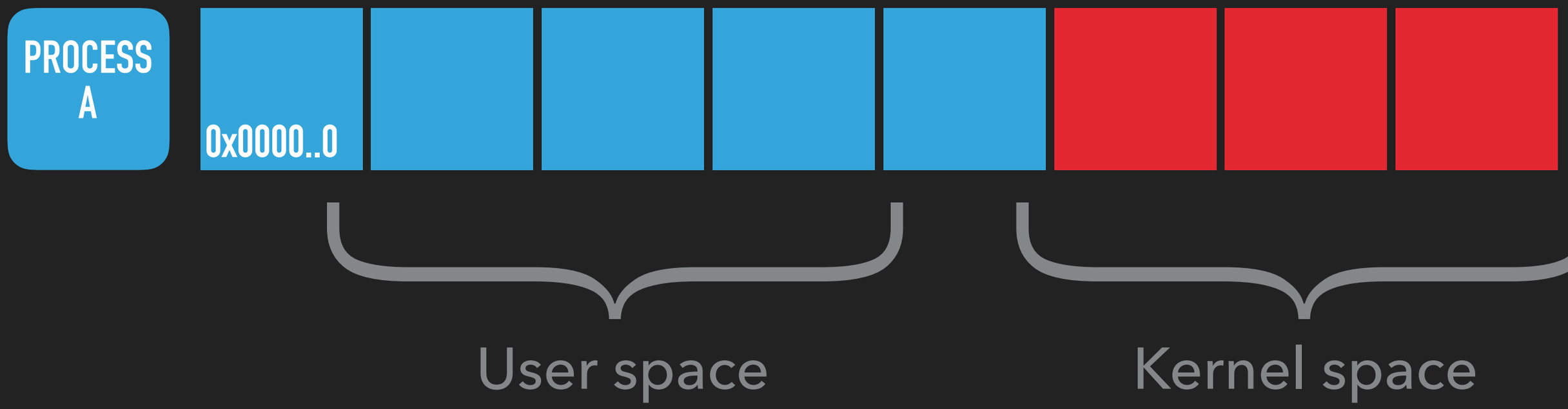


MEMORY MODEL



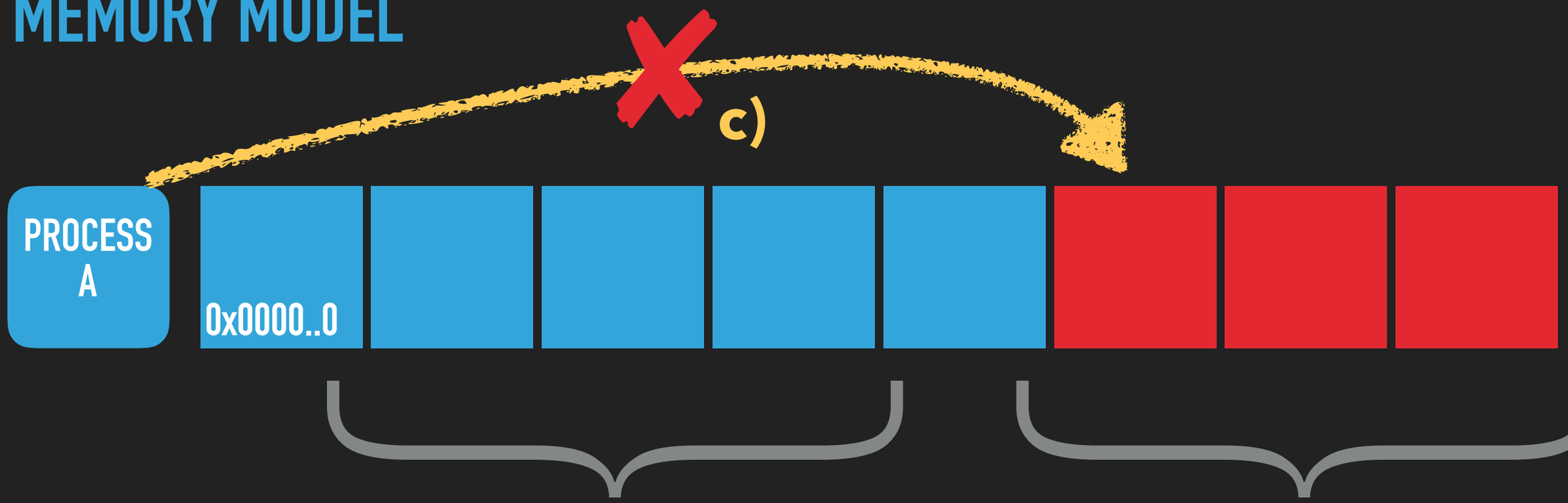
- Memory is split into *pages* (each 4KiB on x86)

MEMORY MODEL



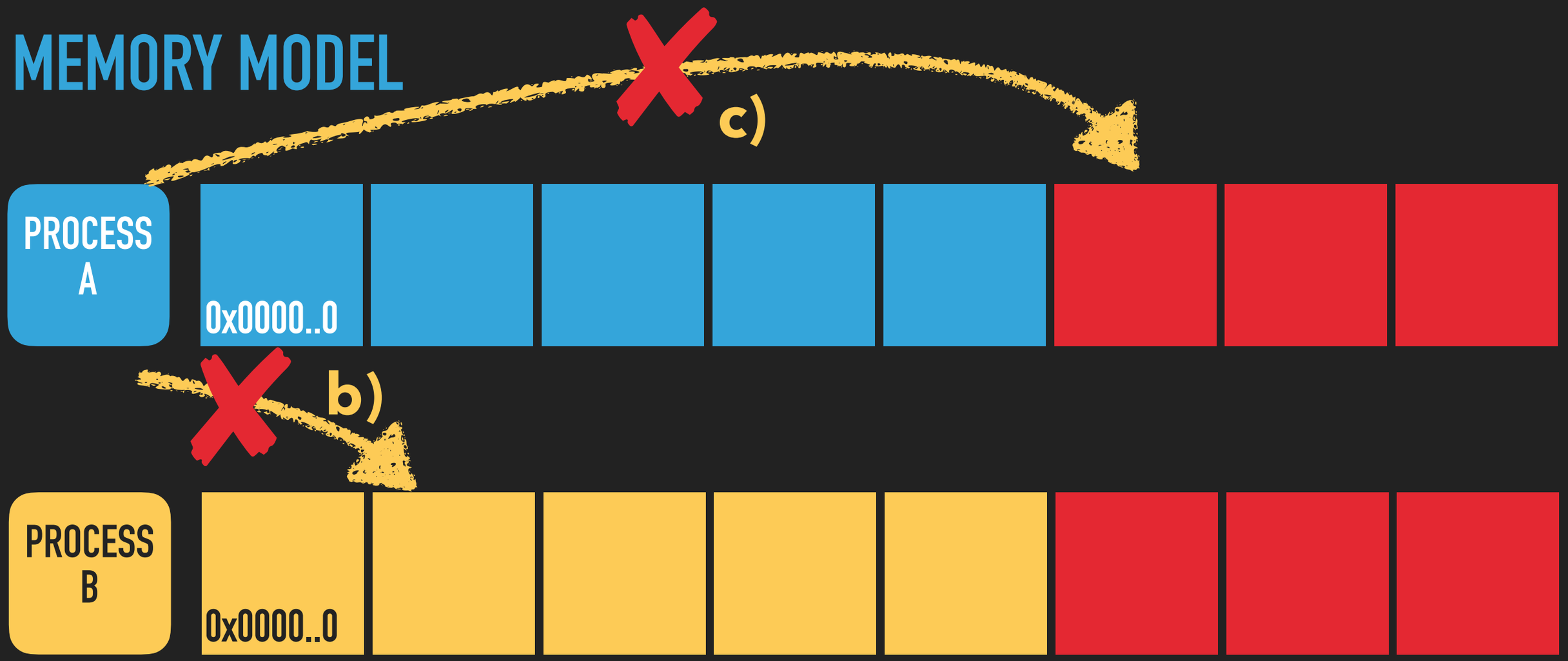
- Memory is split into *pages* (each 4KiB on x86)
- The kernel *maps* its own memory into each process

MEMORY MODEL



- Memory is split into *pages* (each 4KiB on x86)
- The kernel *maps* its own memory into each process
- This “kernel” memory is only accessible by the kernel

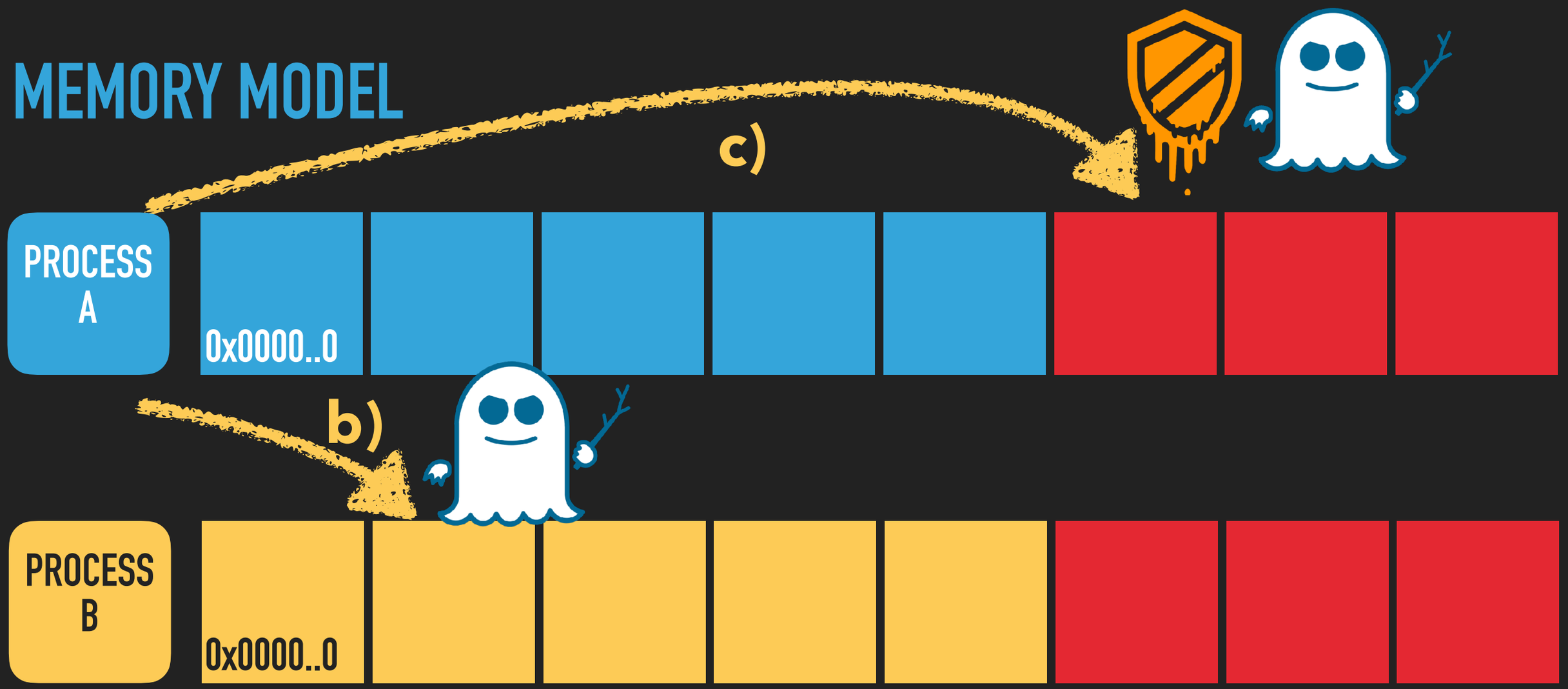
MEMORY MODEL



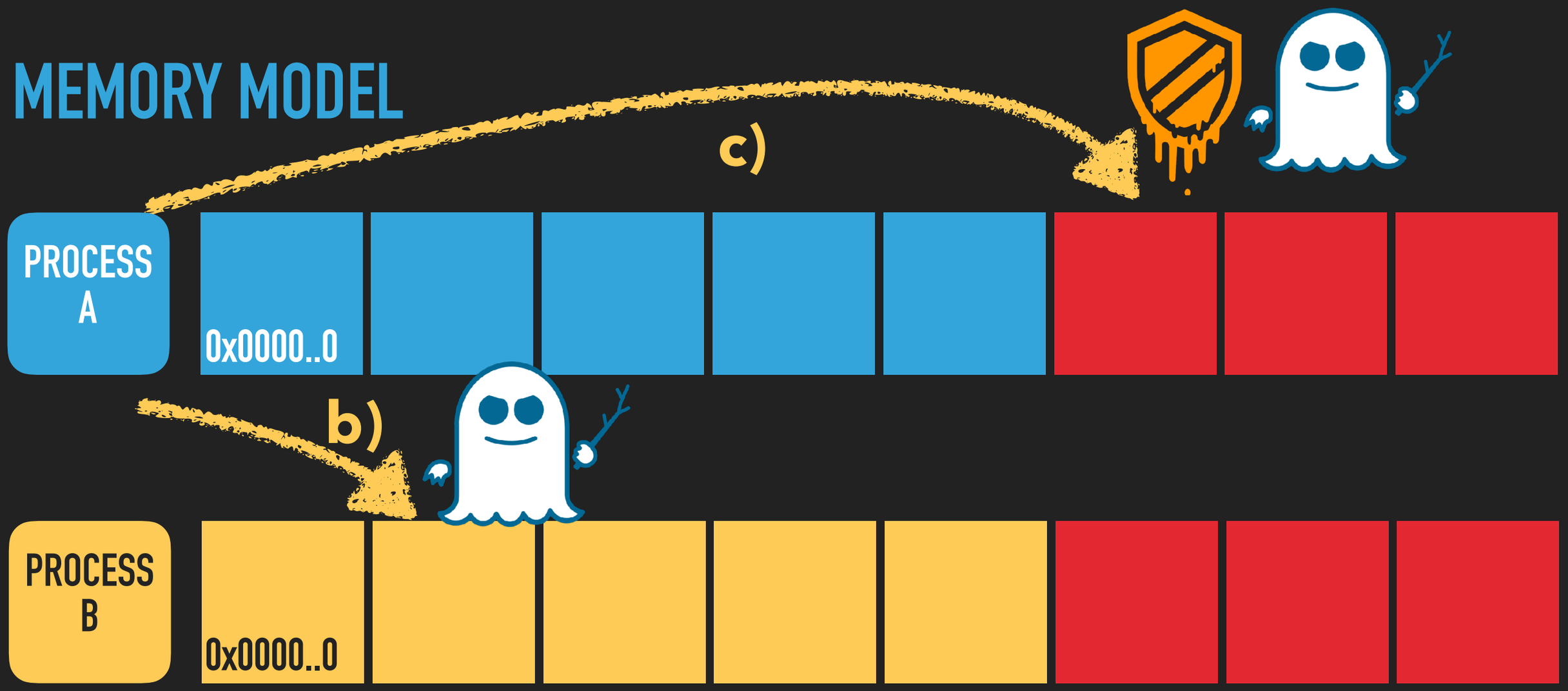
b) and c) are completely different error scenarios

- **c)** Kernel memory pages are marked “kernel only” but the process could try to access the pages via a pointer
- **b)** Process **B** has no possibility to even *describe* the address

MEMORY MODEL



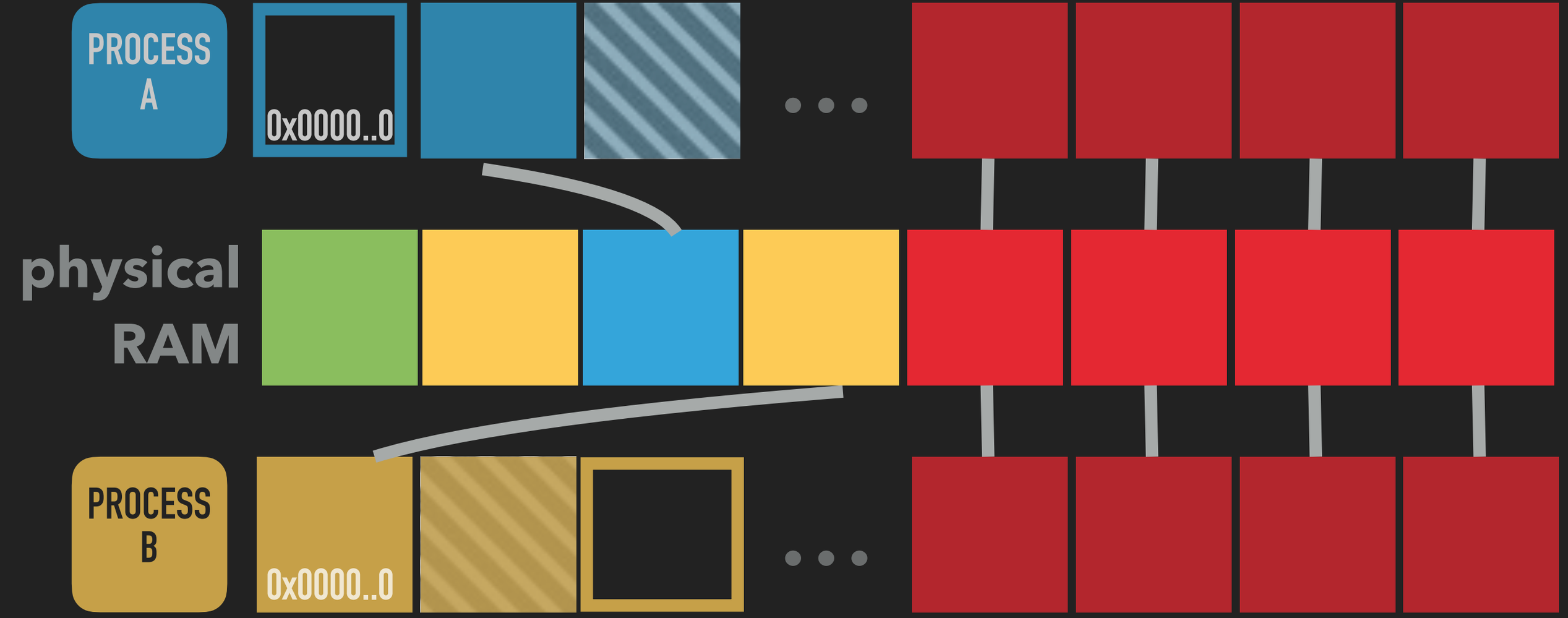
MEMORY MODEL



- **c)** is vulnerable to Meltdown and Spectre
- **b)** is vulnerable to Spectre

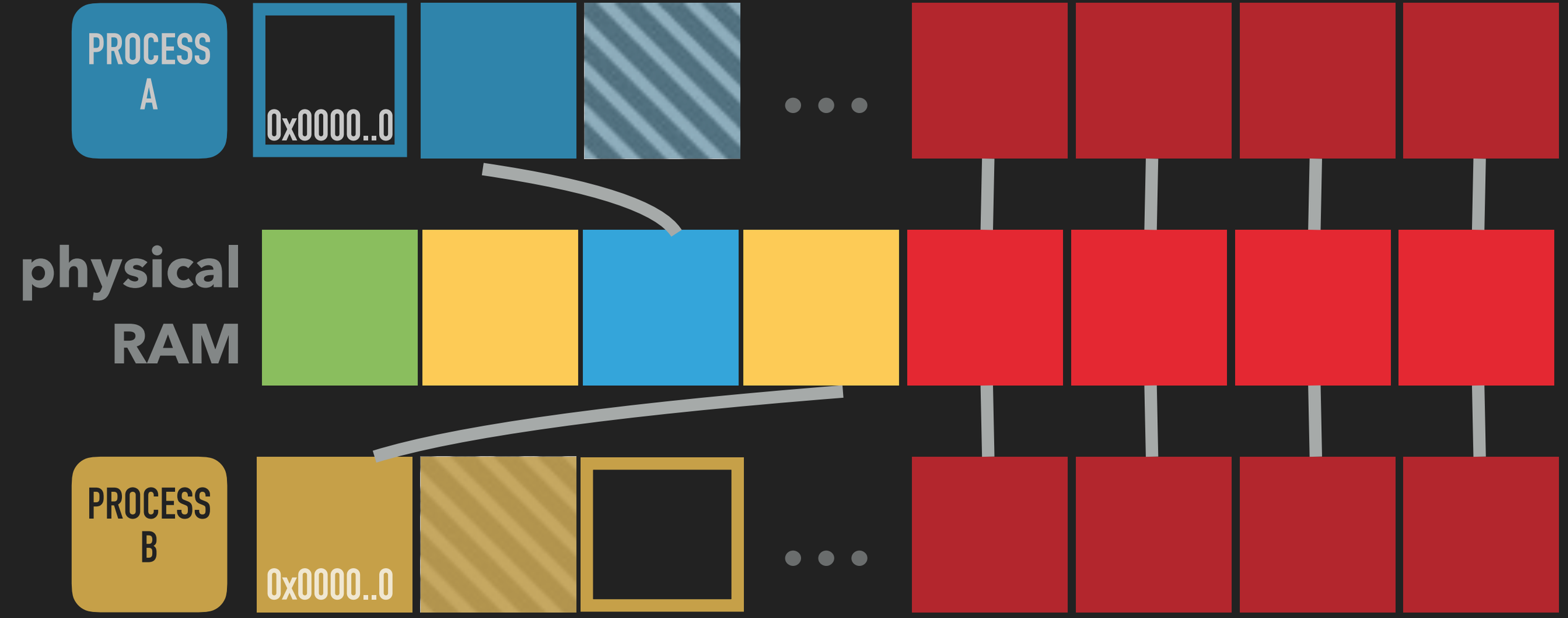
MEMORY MODEL

- Virtual memory is backed by physical RAM
- Virtual memory is much, much larger than physical RAM
- Not all virtual memory is backed by RAM, e.g.



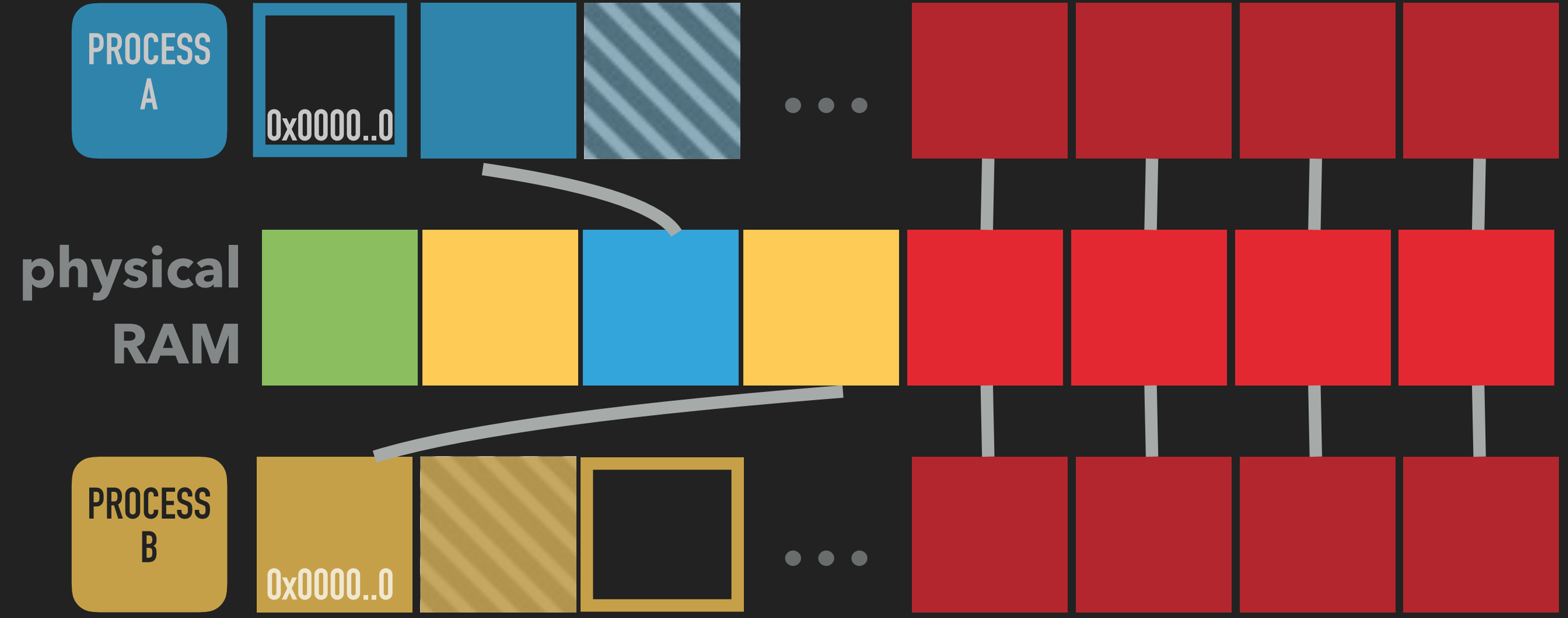
MEMORY MODEL

- Virtual memory is backed by physical RAM
- Virtual memory is much, much larger than physical RAM
- Not all virtual memory is backed by RAM, e.g.



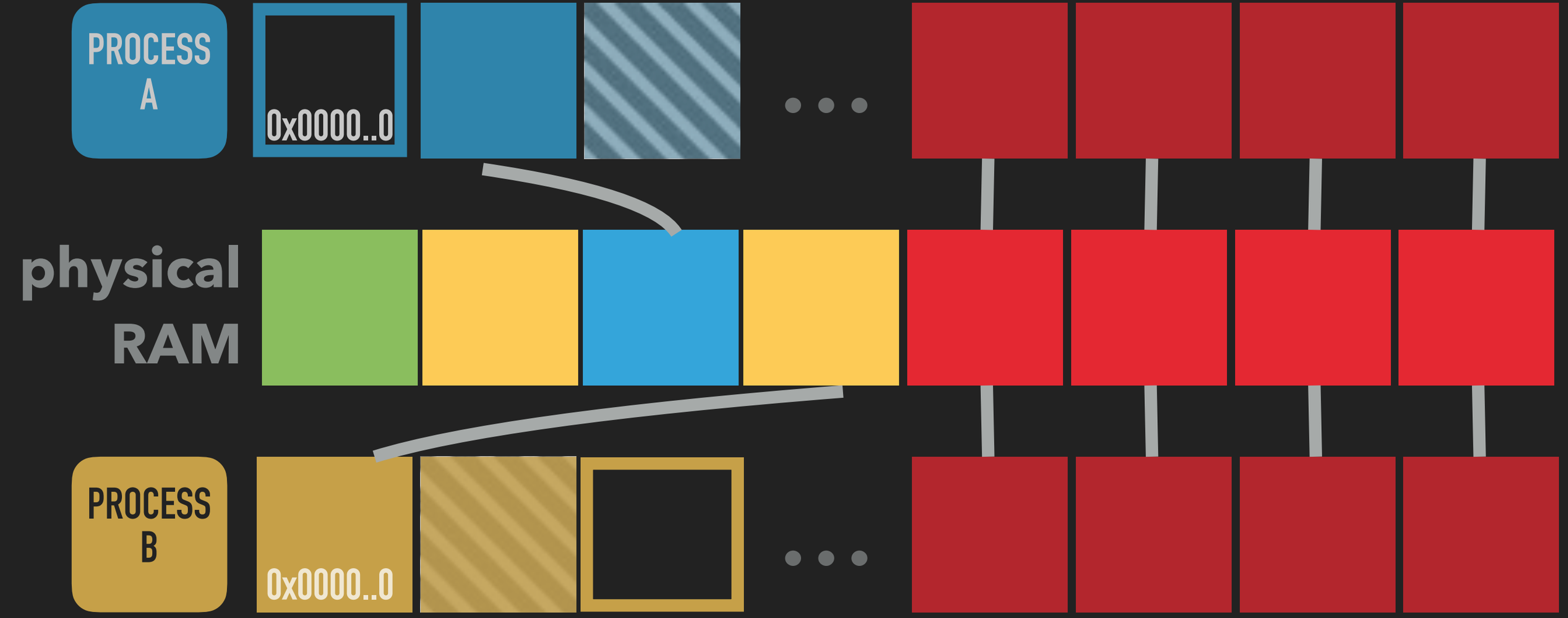
MEMORY MODEL

- Virtual memory is backed by physical RAM
- Virtual memory is much, much larger than physical RAM
- Not all virtual memory is backed by RAM, e.g.

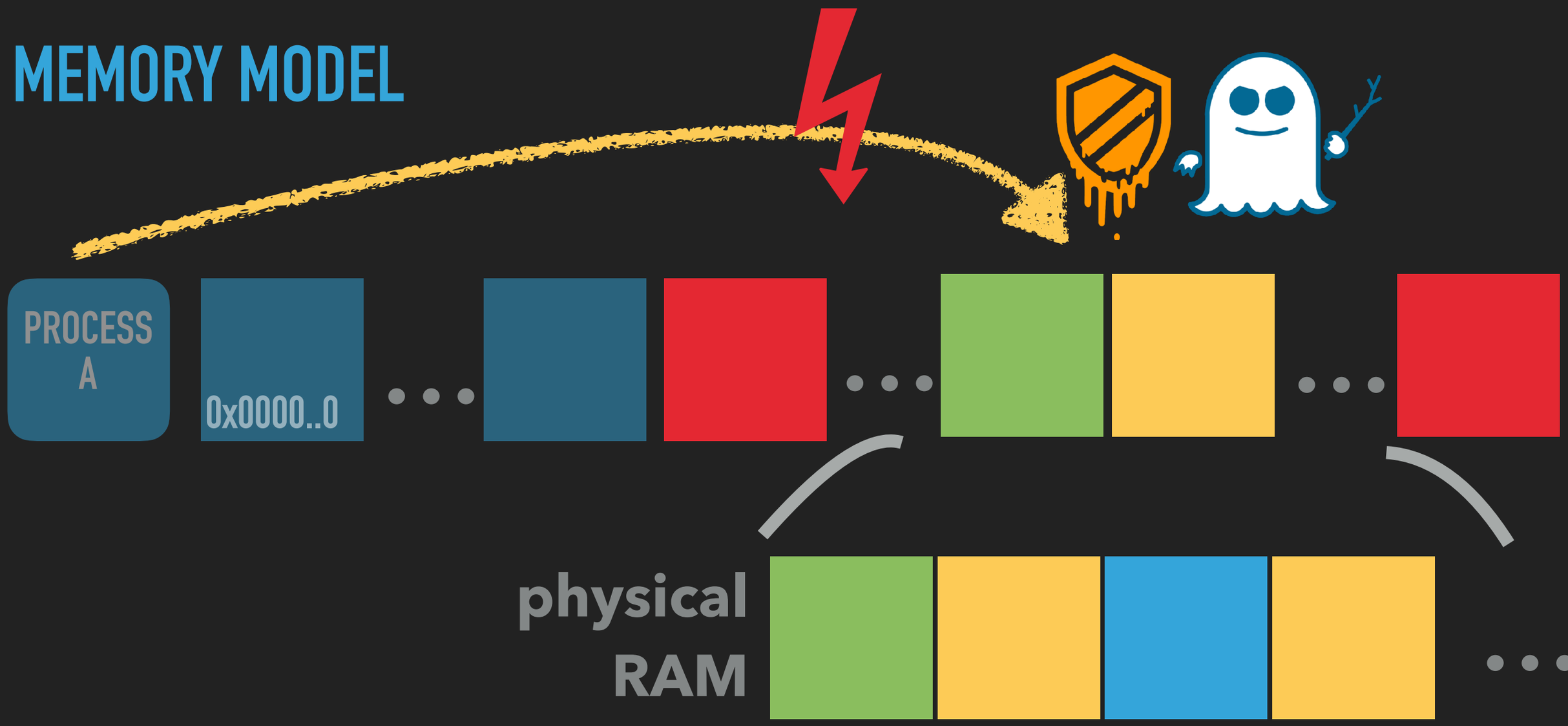


MEMORY MODEL

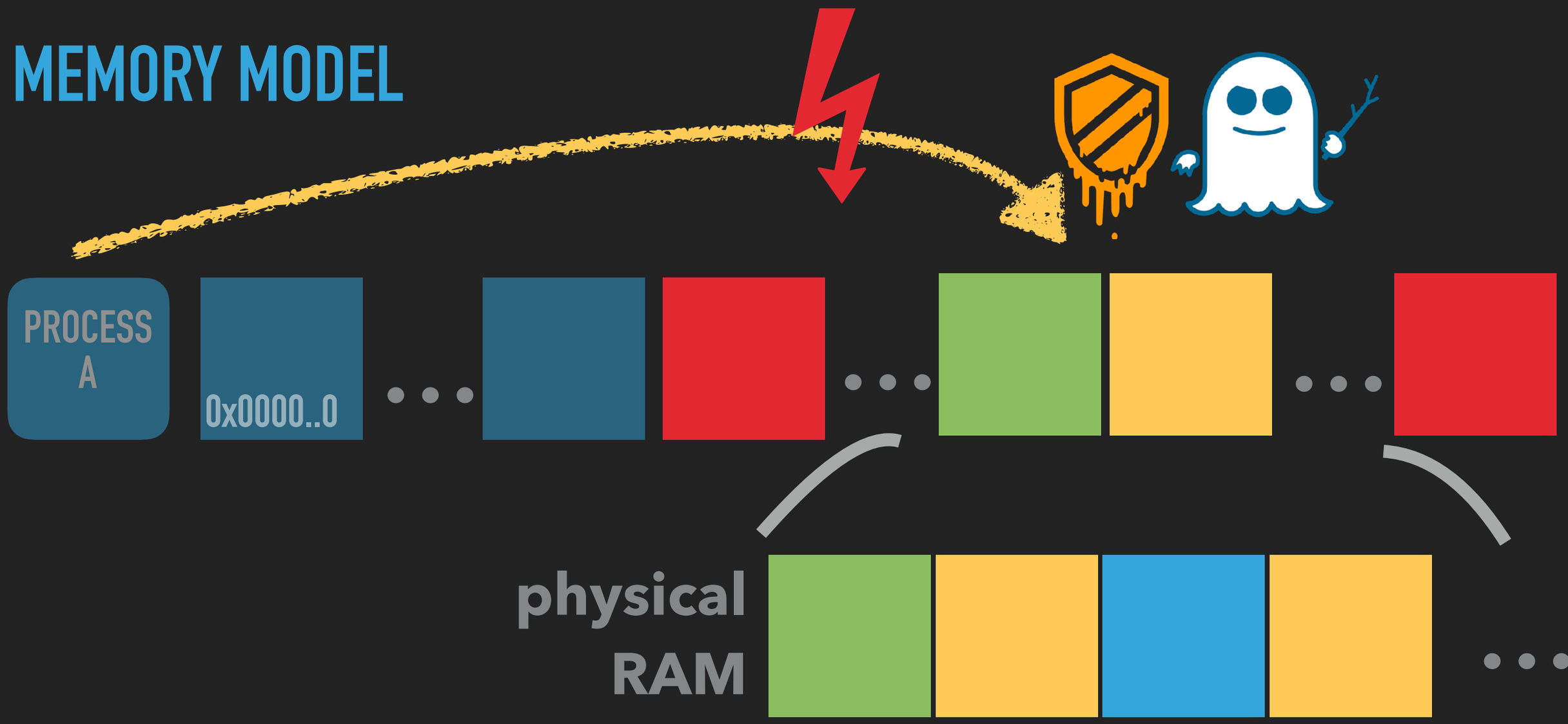
- Virtual memory is backed by physical RAM
- Virtual memory is much, much larger than physical RAM
- Not all virtual memory is backed by RAM, e.g.



MEMORY MODEL



MEMORY MODEL



- Like a matryoshka doll the kernel *maps all physical* memory into its address space
- Reading kernel memory allows reading of all (mapped) memory of all processes

MEMORY MODEL

```
Virtual memory map with 4 level page tables:

0000000000000000 - 00007fffffffffff (=47 bits) user space, different per mm
hole caused by [47:63] sign extension
ffff800000000000 - ffff87ffffffffff (=43 bits) guard hole, reserved for hypervisor
ffff880000000000 - ffffc7ffffffffff (=64 TB) direct mapping of all phys. memory
ffffc80000000000 - ffffc8ffffffffff (=40 bits) hole
ffffc90000000000 - ffffe8ffffffffff (=45 bits) vmalloc/ioremap space
ffffe90000000000 - ffffe9ffffffffff (=40 bits) hole
ffffea0000000000 - ffffeaffffffffff (=40 bits) virtual memory map (1TB)
... unused hole ...
ffffec0000000000 - fffffbffffffffff (=44 bits) kasan shadow memory (16TB)
... unused hole ...
                 vaddr_end for KASLR
fffffe0000000000 - fffffe7fffffffff (=39 bits) cpu_entry_area mapping
fffffe8000000000 - fffffeffffffffff (=39 bits) LDT remap for PTI
ffffff0000000000 - ffffff7fffffffff (=39 bits) %esp fixup stacks
... unused hole ...
ffffffef00000000 - ffffffffffffffff (=64 GB) EFI region mapping space
... unused hole ...
ffffffff80000000 - ffffffff9fffffff (=512 MB)  kernel text mapping, from phys 0
ffffffffa0000000 - ffffffffefffffff (1520 MB) module mapping space
[fixmap start]   - fffffffff5fffff kernel-internal fixmap range
fffffffff600000 - fffffffff600fff (=4 kB) legacy vsyscall ABI
fffffffffe00000 - ffffffffffffffff (=2 MB) unused hole
```

https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

OUT OF ORDER EXECUTION

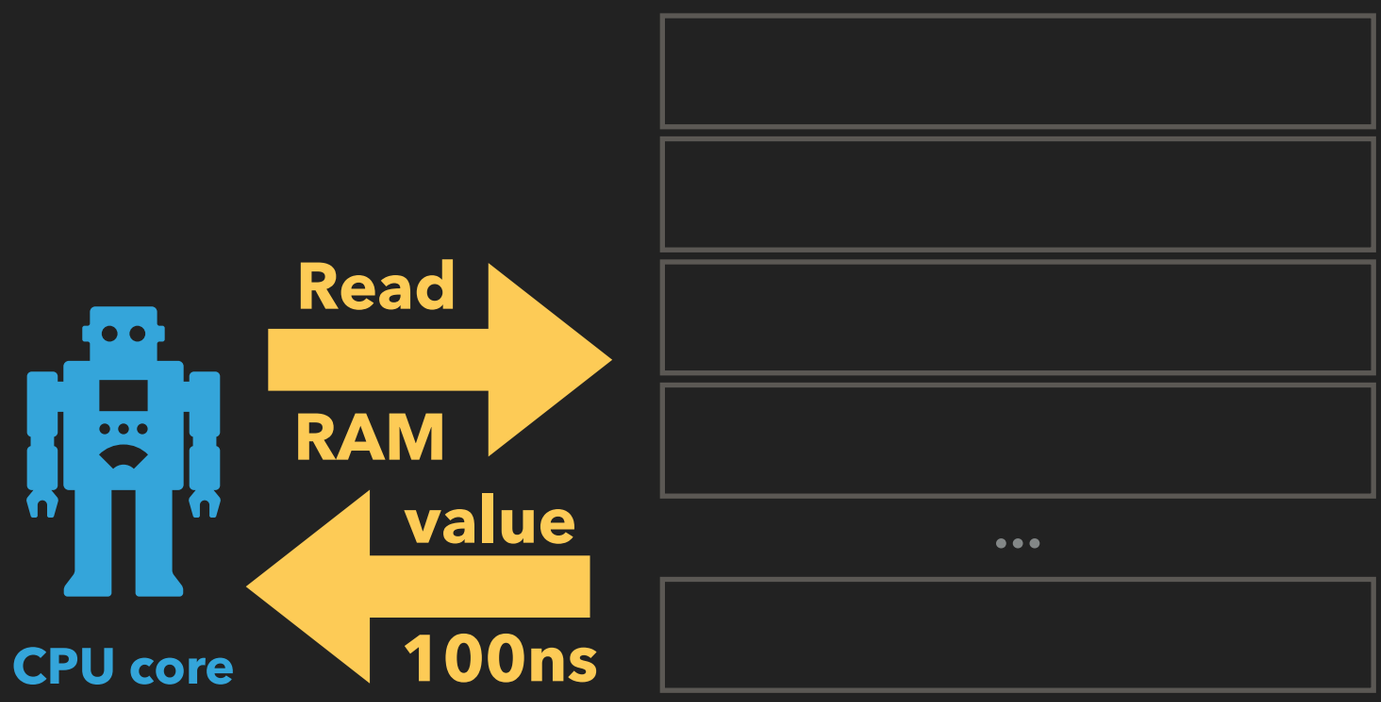
MELTDOWN

MELTDOWN

Meltdown basically works like this:

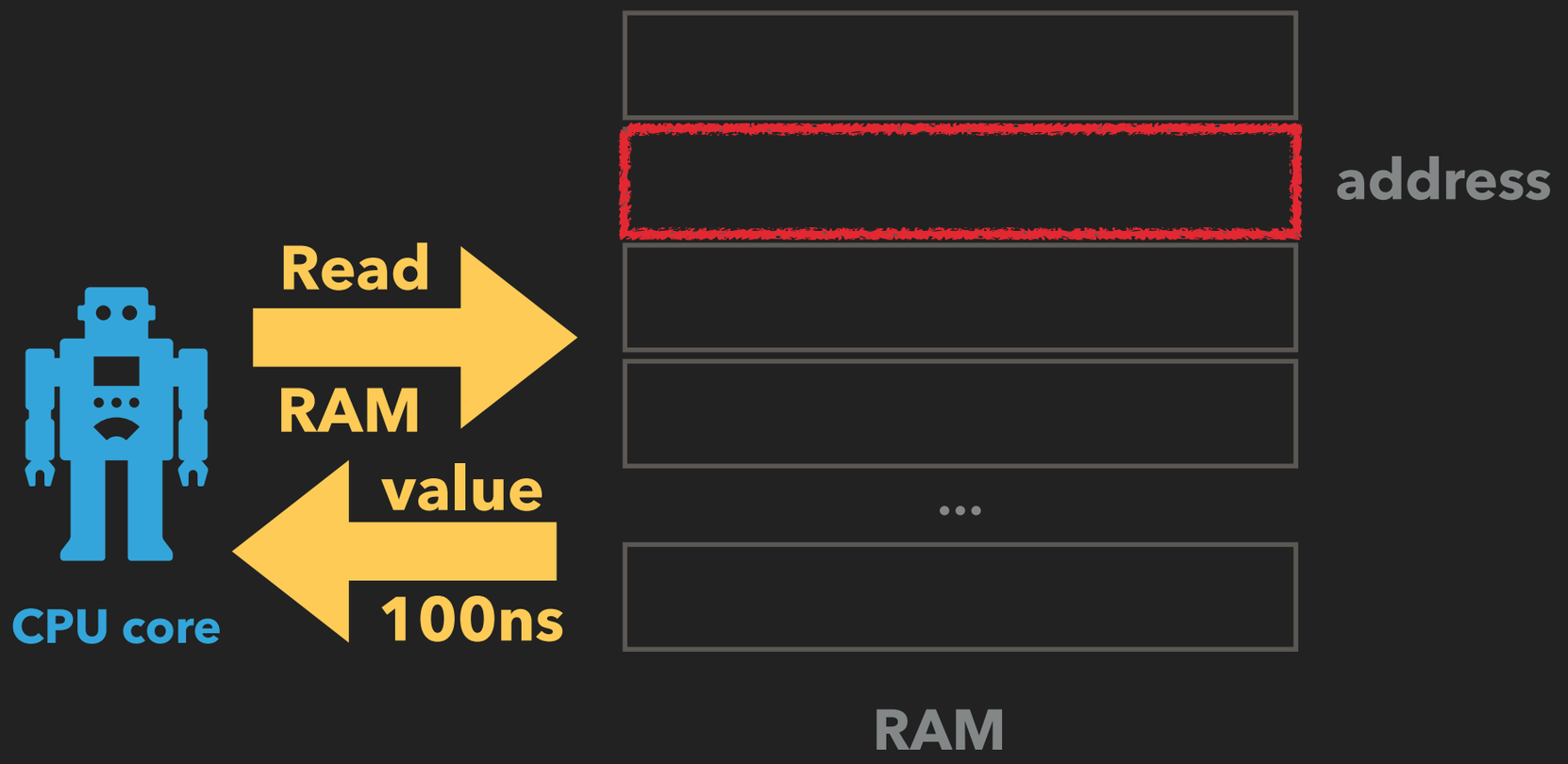
- READ secret from forbidden address
- Stash away secret before CPU detects wrongdoing
- Retrieve secret

MELTDOWN: STASHING AWAY – SIDECHANNEL



- Data is stored in RAM
- RAM is very slow
- Reading one byte stalls the CPU for hundreds of µOPs

MELTDOWN: STASHING AWAY – SIDECHANNEL

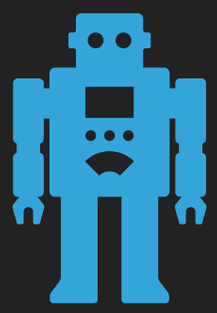


- Data is stored in RAM
- RAM is very slow
- Reading one byte stalls the CPU for hundreds of µOPs

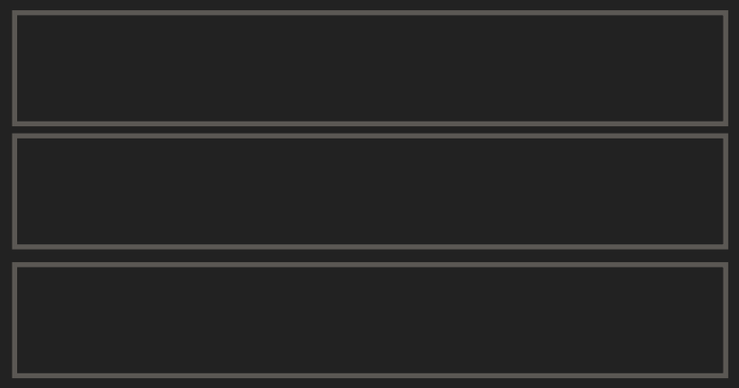
MELTDOWN: STASHING AWAY – SIDECHANNEL

- Data is stored in RAM
- RAM is very slow
- Reading one byte stalls the CPU for hundreds of µOPs

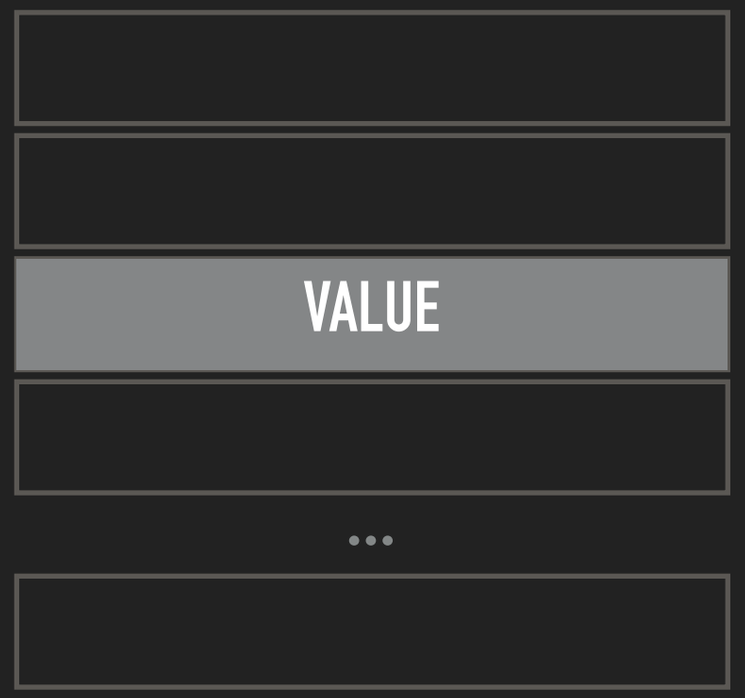
MELTDOWN: STASHING AWAY – SIDECHANNEL



CPU core



Cache

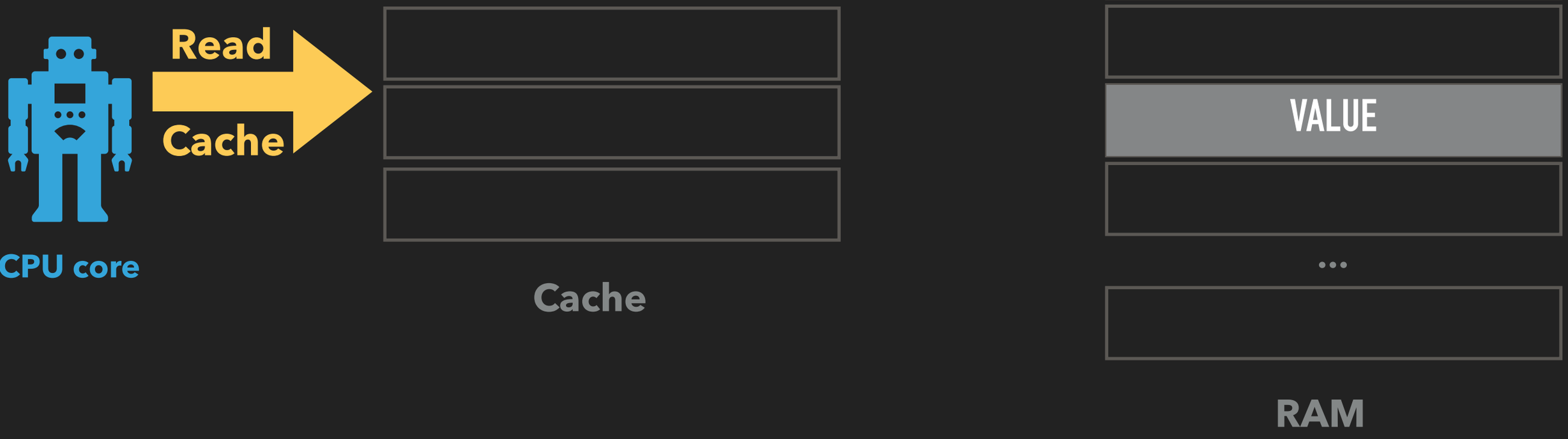


RAM

- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

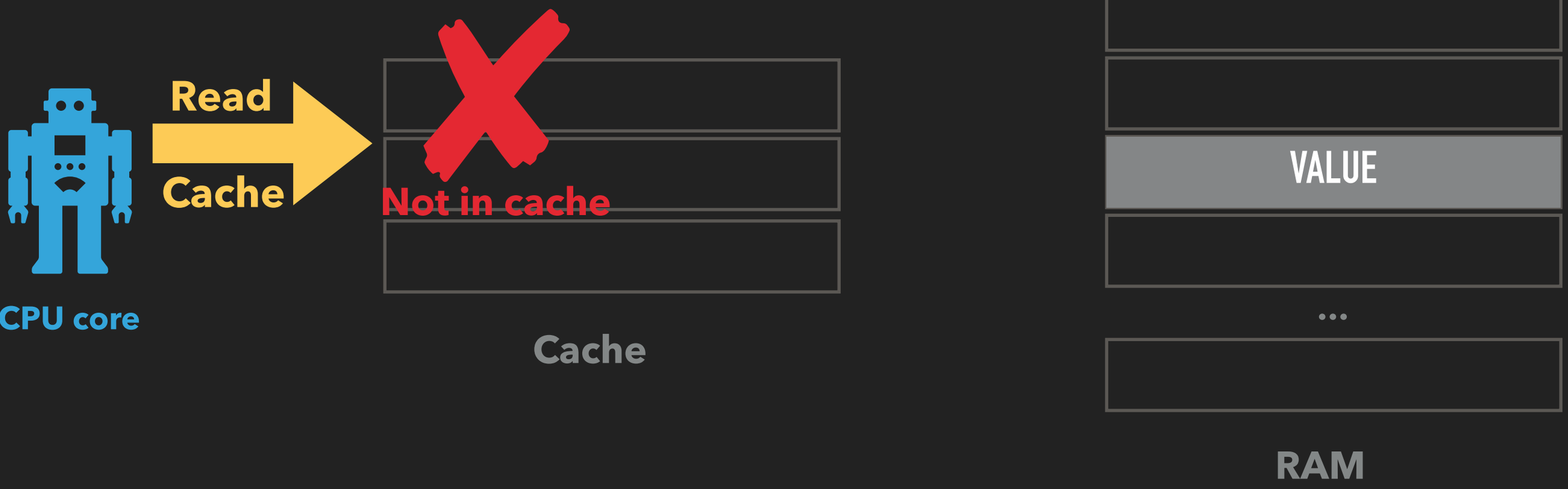
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

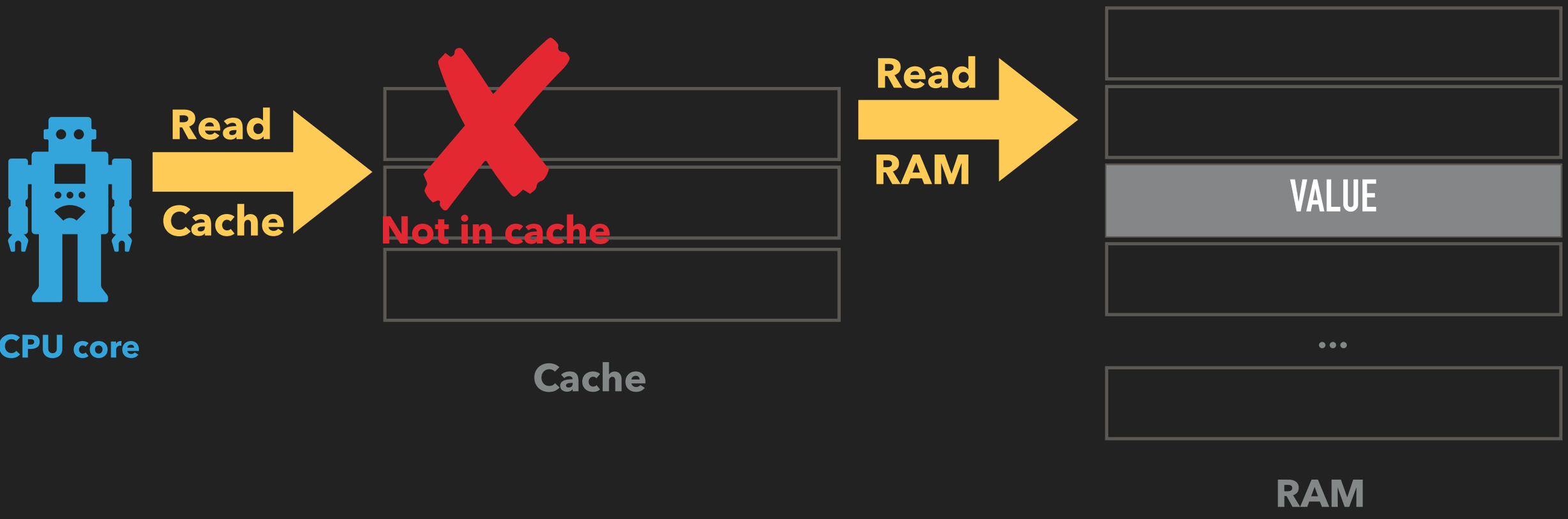
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

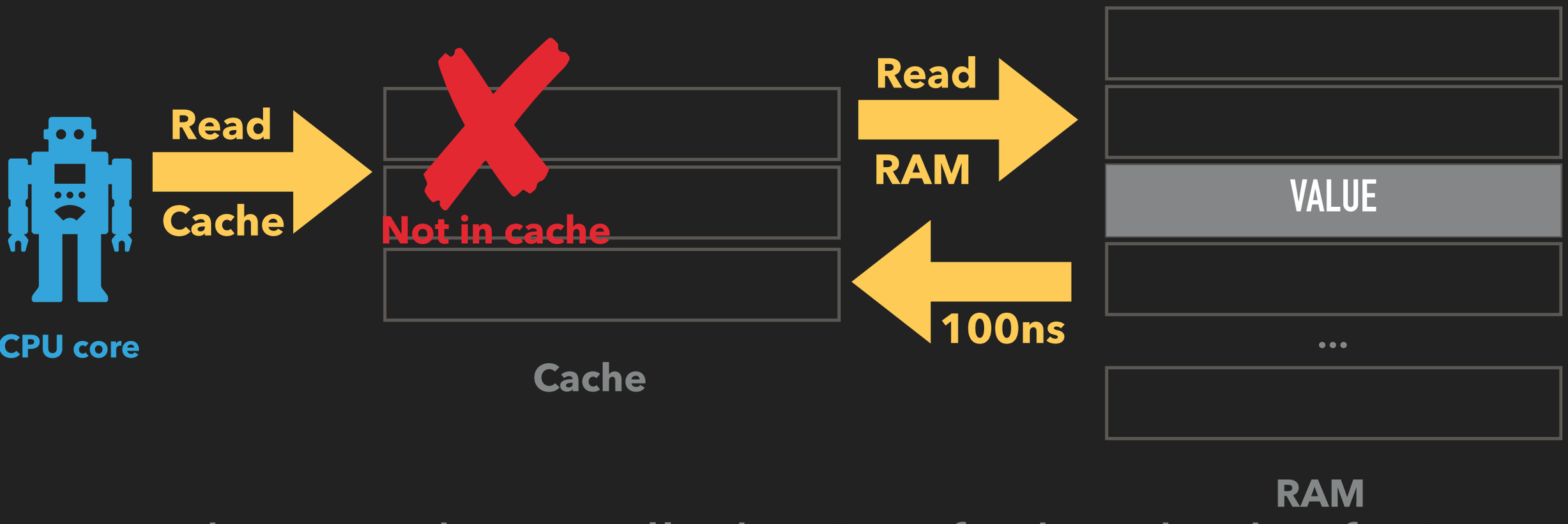
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

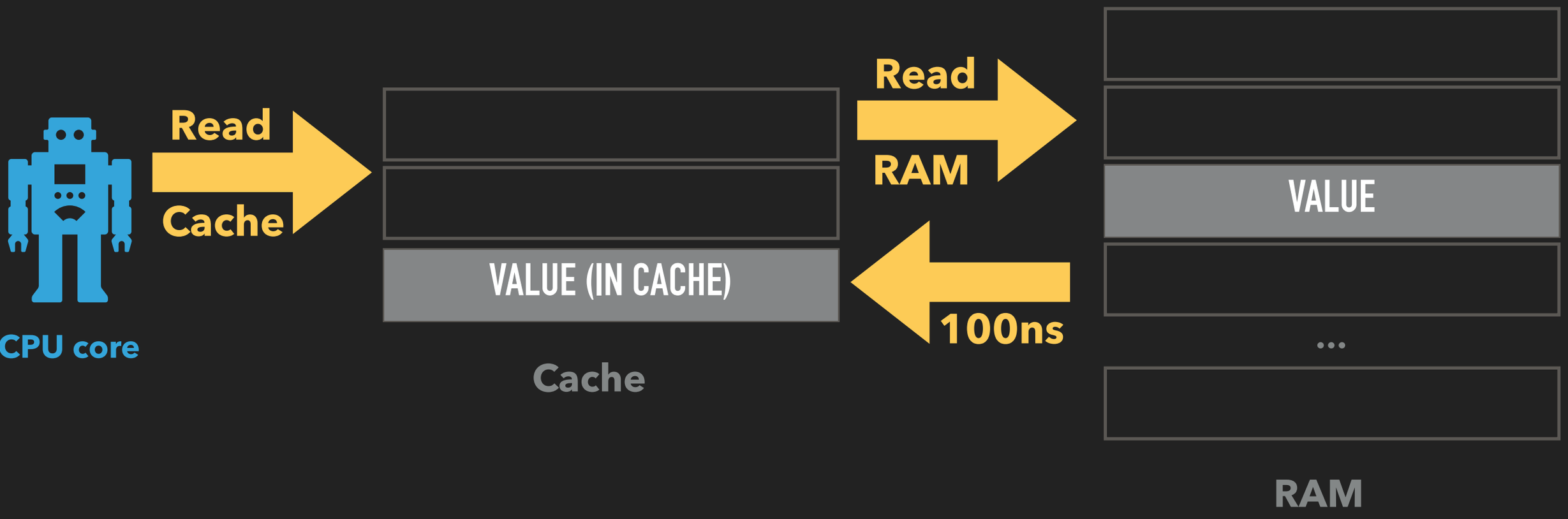
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

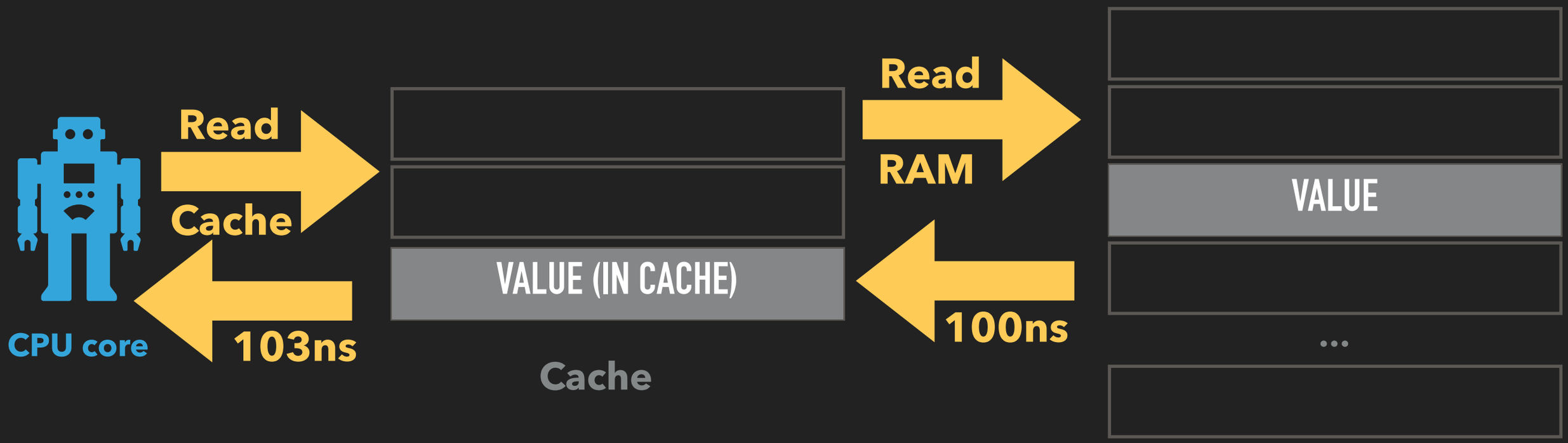
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

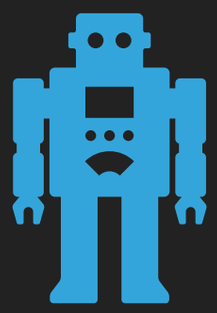
MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

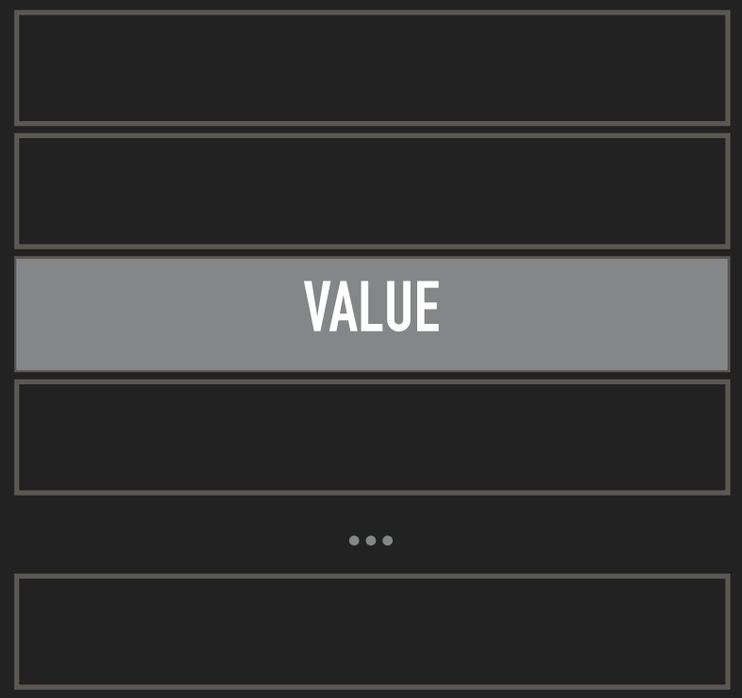
The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

MELTDOWN: STASHING AWAY – SIDECHANNEL



CPU core

Cache



RAM

- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

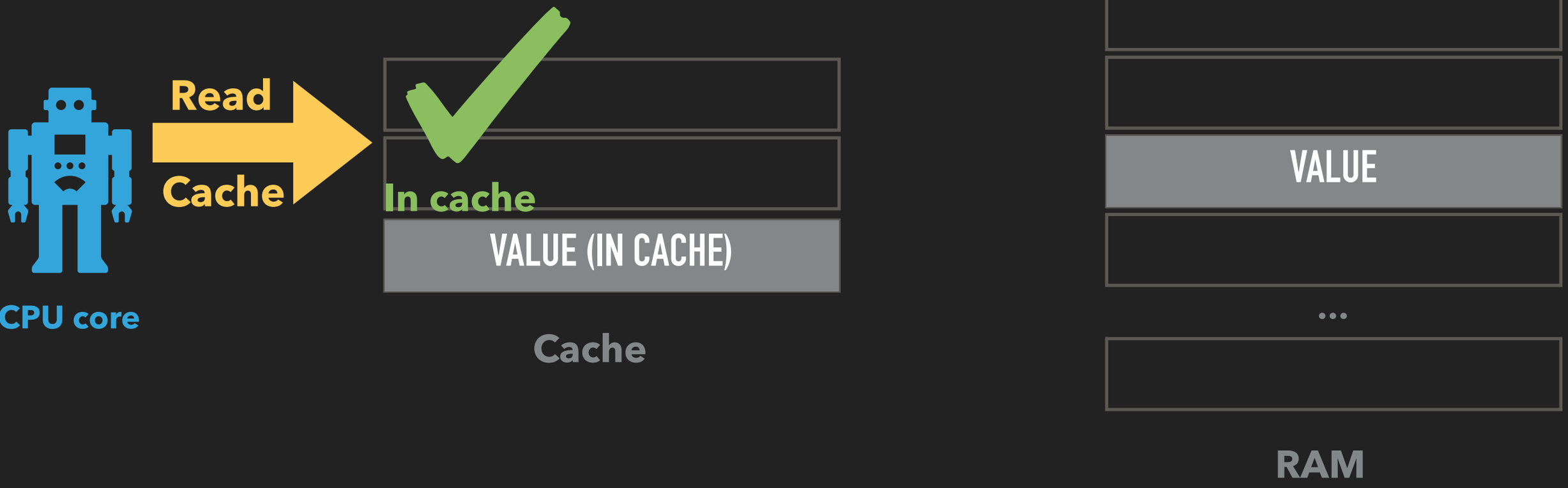
The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

MELTDOWN: STASHING AWAY – SIDECHANNEL

- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

MELTDOWN: STASHING AWAY – SIDECHANNEL



- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

MELTDOWN: STASHING AWAY – SIDECHANNEL

- Reading one byte stalls the CPU for hundreds of µOPs
- CPU caches considerably speed this up
- E.g. reading cached takes 3ns, reading uncached 103ns

The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

“READ” INSTRUCTION

For a CPU the “READ value from memory at 4711” instruction looks like this (μOPs):

1. Check that program may read from address 1
2. Store the value at address in register[1] 2

If 1 fails the program is aborted.

This can be handled by the program.

[1] Register: The CPUs scratchpad

“READ” INSTRUCTION

For a CPU the “READ value from memory at 4711” instruction looks like this (μOPs):

1. Check that program may read from address 1
2. Store the value at address in register[1] 2

If 1 fails the program is aborted.

This can be handled by the program.

In our burger example:

1. Customer orders a burger & coffee
2. Burger is ready, coffee machine breaks
3. Customer does not get his burger

[1] Register: The CPUs scratchpad

MELTDOWN: READING FORBIDDEN DATA

Meltdown basically works like this:

- READ secret from forbidden address
  1. Check that program may read from address
  2. Store the read value in register
- Stash away secret
  1. *Magic*
- *Retrieve secret (later)*

µOPs: 1 2 1

MELTDOWN: READING FORBIDDEN DATA

μOPs ordered by *instruction*	μOPs ordered by *execution*
1 Check access	2 Read into register
2 Read into register	1 *Magic*
1 *Magic*	1 Check access

The re-ordering on the right happens, when the “forbidden data” is already cached (because cache access is so fast).

MELTDOWN: READING FORBIDDEN DATA

μOPs ordered by *instruction*	μOPs ordered by *execution*
1 Check access	2 Read into register
2 Read into register	1 *Magic*
1 *Magic*	1 Check access

The re-ordering on the right happens, when the “forbidden data” is already cached (because cache access is so fast).

Reordering is not a problem because the CPU will ensure that 1 is only seen *iff* 1 succeeds.

MELTDOWN: READING FORBIDDEN DATA

μOPs ordered by *instruction*	μOPs ordered by *execution*
1 Check access	2 Read into register
2 Read into register	1 *Magic*
1 *Magic*	1 Check access

The re-ordering on the right happens, when the “forbidden data” is already cached (because cache access is so fast).

Reordering is not a problem because the CPU will ensure that 🟦 is only seen *iff* 🟨 succeeds.

Unless 🟦 is able to hide the secret in such a way that the attacker can find it later.

MELTDOWN: READING FORBIDDEN DATA

μOPs ordered by *instruction*	μOPs ordered by *execution*
1 Check access	2 Read into register
2 Read into register	1 *Magic*
1 *Magic*	1 Check access

The re-ordering on the right happens, when the "forbidden data" is already cached (because cache access is so fast).

Reordering is not a problem bec
that is only seen *iff* succe

Unless is able to hide the secr
attacker can find it later.

In our burger example:

1. **Customer orders a burger & coffee**
2. **Customer gets his burger**
3. **Coffee machine breaks**
4. **Customer runs away with burger**

MELTDOWN

For Meltdown two actors are needed

The **spy** and a **collector**.

110011010
010111010
111100100
000101101
100110010
Spy

The **spy** will “steal” the secret and stash it away.

The CPU will kill him for accessing the secret information.

110011010
010111010
111100100
000101101
100110010
Collector

The **collector** will find the stashed away secret.

MELTDOWN: THE SIDECHANNEL (IDEA)

```
110011010
010111010
111100100
000101101
100110010
```

Spy

```
110011010
010111010
111100100
000101101
100110010
```

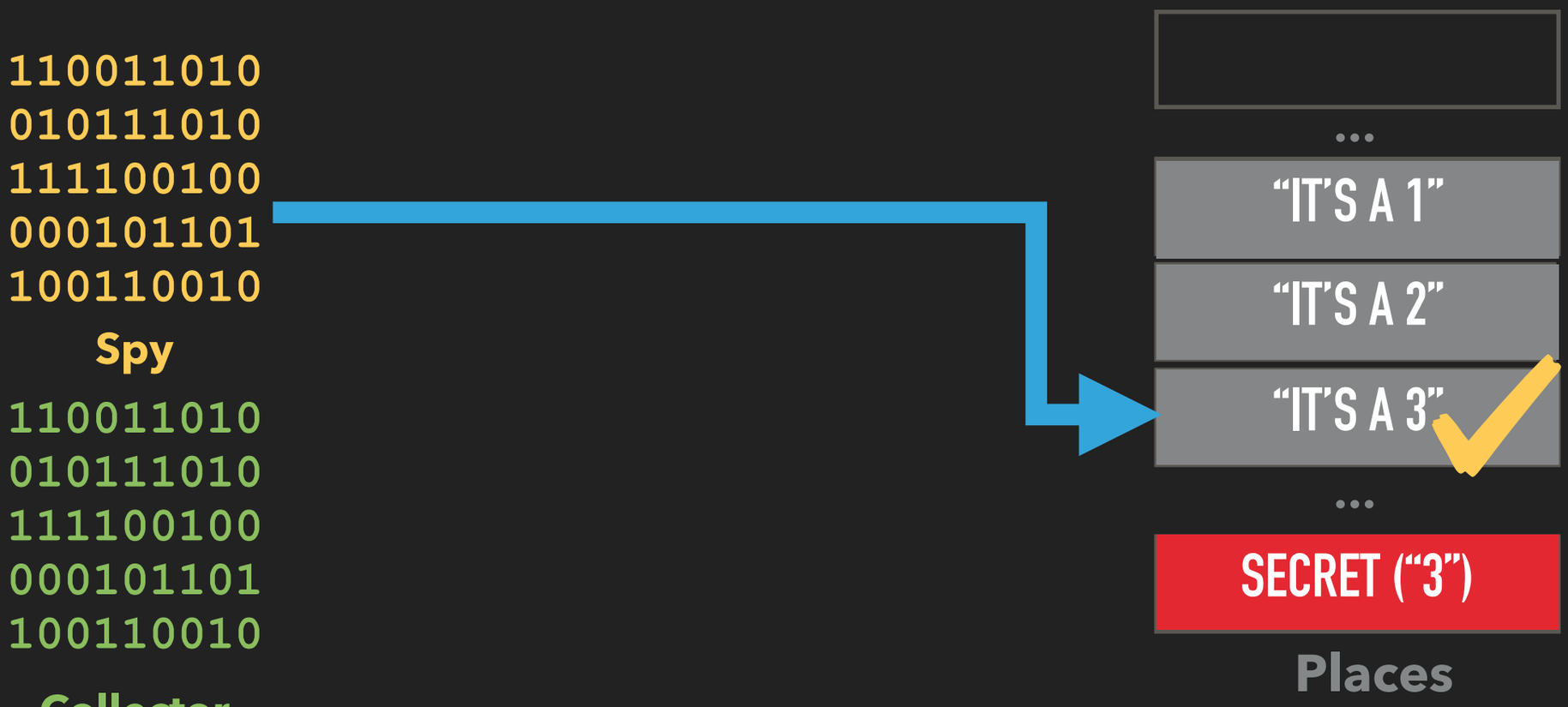
Collector

Places

MELTDOWN: THE SIDECHANNEL (IDEA)

Spy will read the **secret**

MELTDOWN: THE SIDECHANNEL (IDEA)



- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block

MELTDOWN: THE SIDECHANNEL (IDEA)

- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block
- CPU detects **Spys** access validation and terminates **Spy**

MELTDOWN: THE SIDECHANNEL (IDEA)

- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block
- CPU detects **Spys** access validation and terminates **Spy**
- **Collector** now looks for **Spys** mark in all grey blocks

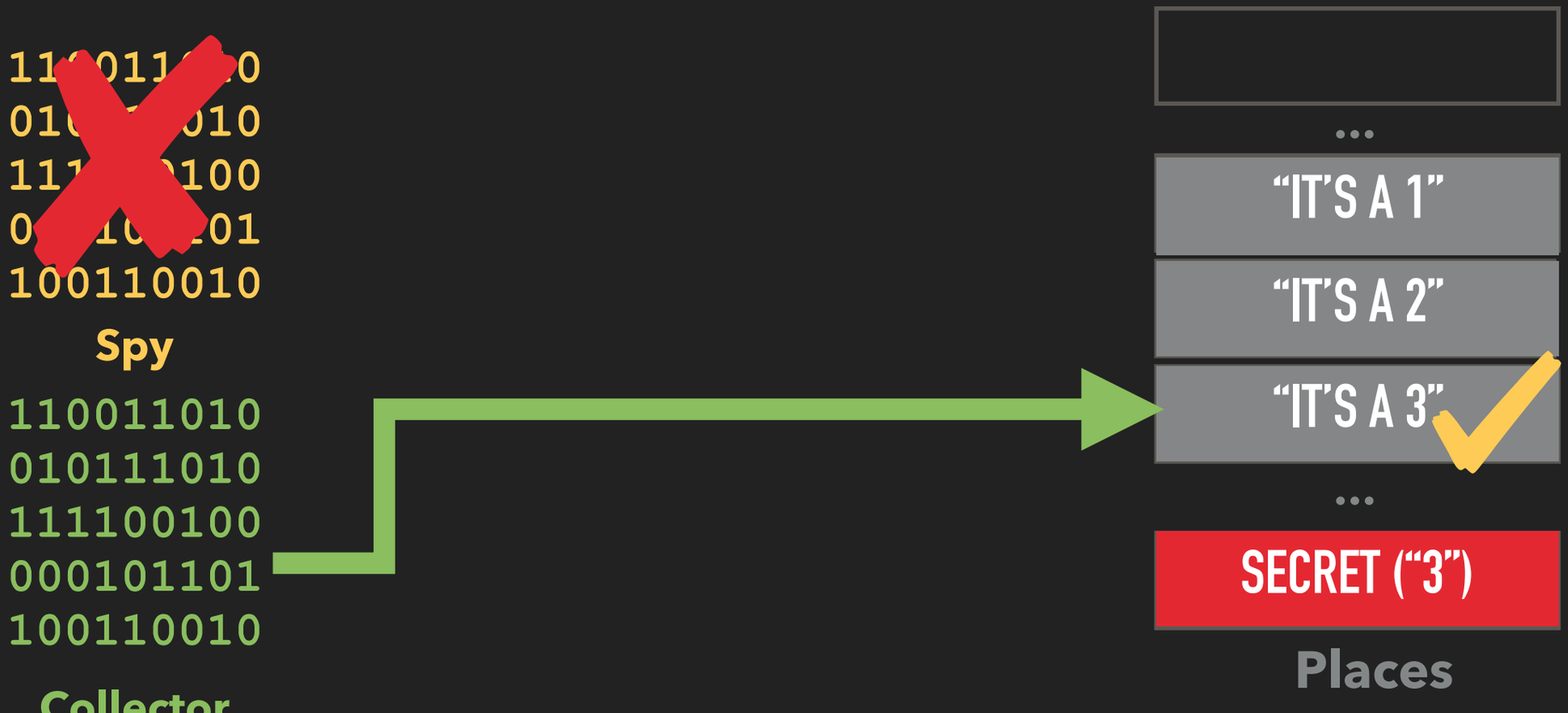
MELTDOWN: THE SIDECHANNEL (IDEA)

- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block
- CPU detects **Spys** access validation and terminates **Spy**
- **Collector** now looks for **Spys** mark in all grey blocks

MELTDOWN: THE SIDECHANNEL (IDEA)

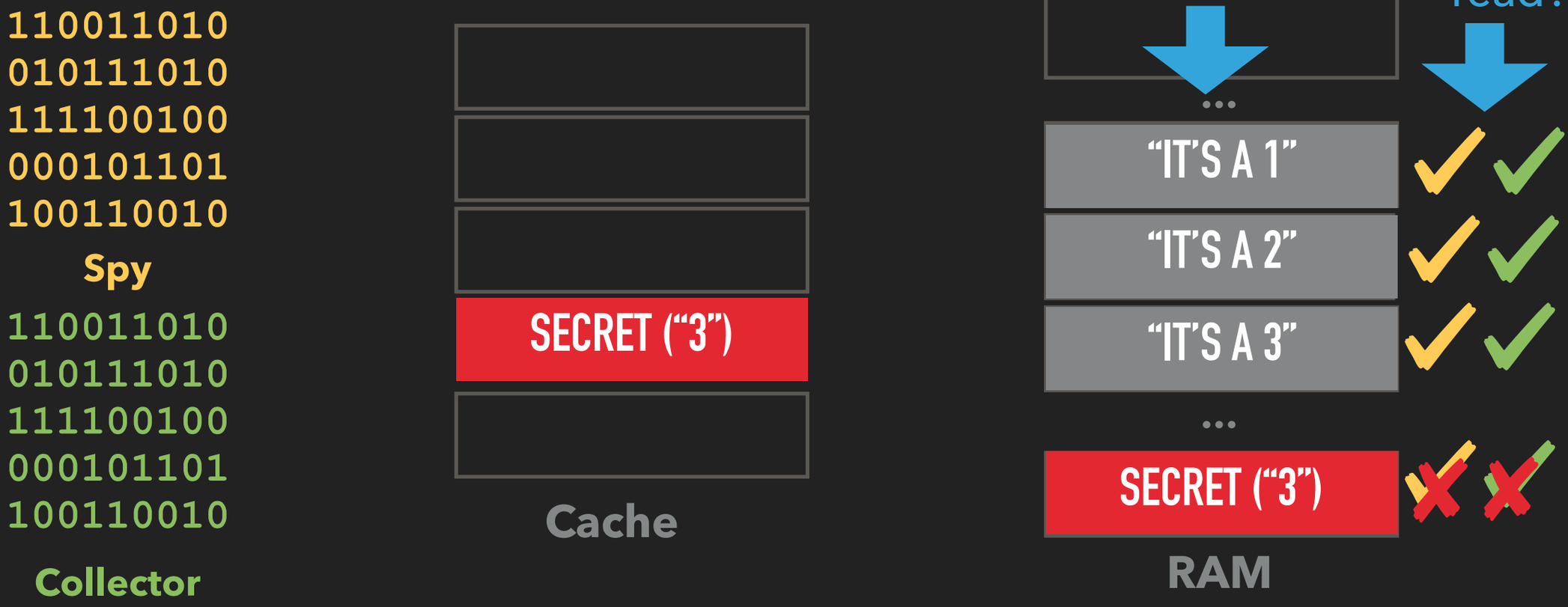
- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block
- CPU detects **Spys** access validation and terminates **Spy**
- **Collector** now looks for **Spys** mark in all grey blocks

MELTDOWN: THE SIDECHANNEL (IDEA)



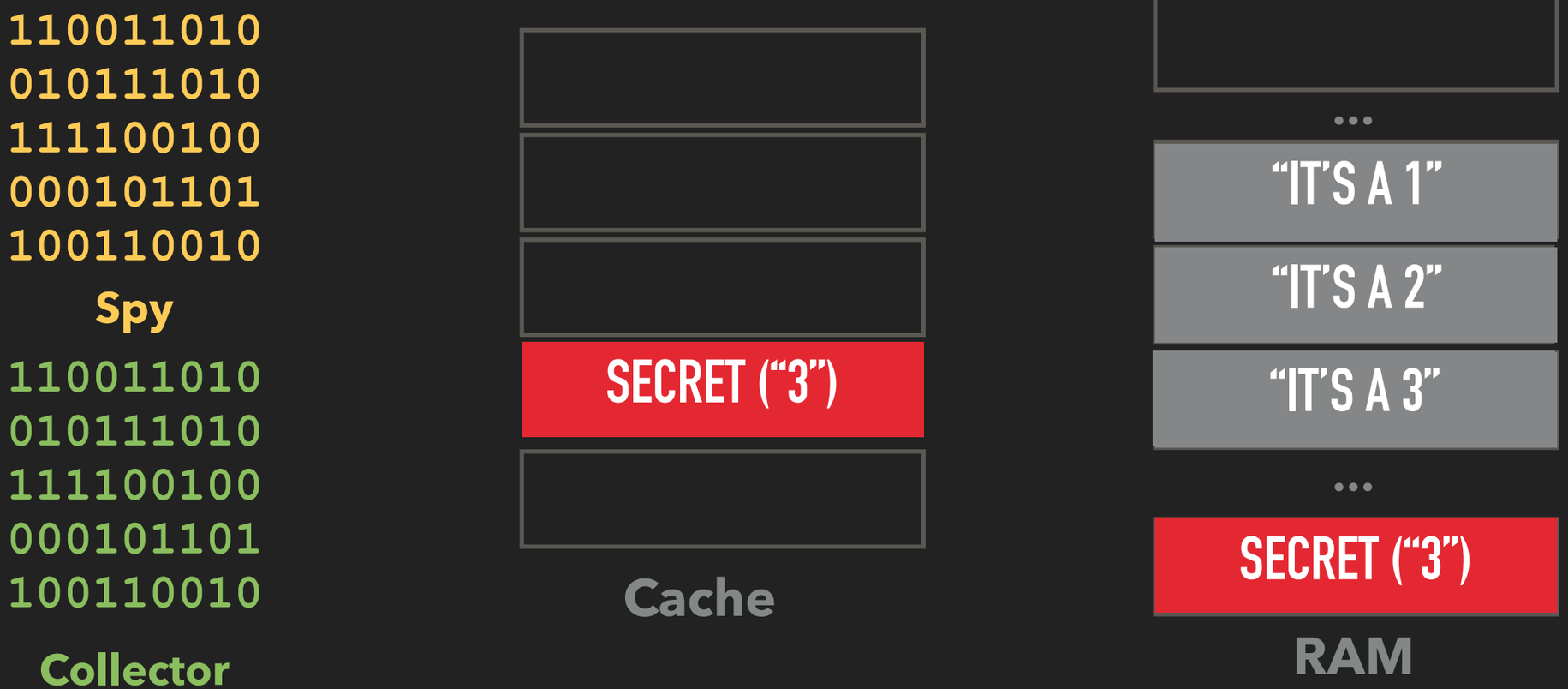
- **Spy** will read the **secret**
- Depending on the **value**, **Spy** will mark a grey block
- CPU detects **Spys** access validation and terminates **Spy**
- **Collector** now looks for **Spys** mark in all grey blocks

MELTDOWN: THE ATTACK



- Meltdown needs some preconditions
- The **secret** is in the cache (value: 3)
- Both **Spy** and **Collector** can read grey memory blocks

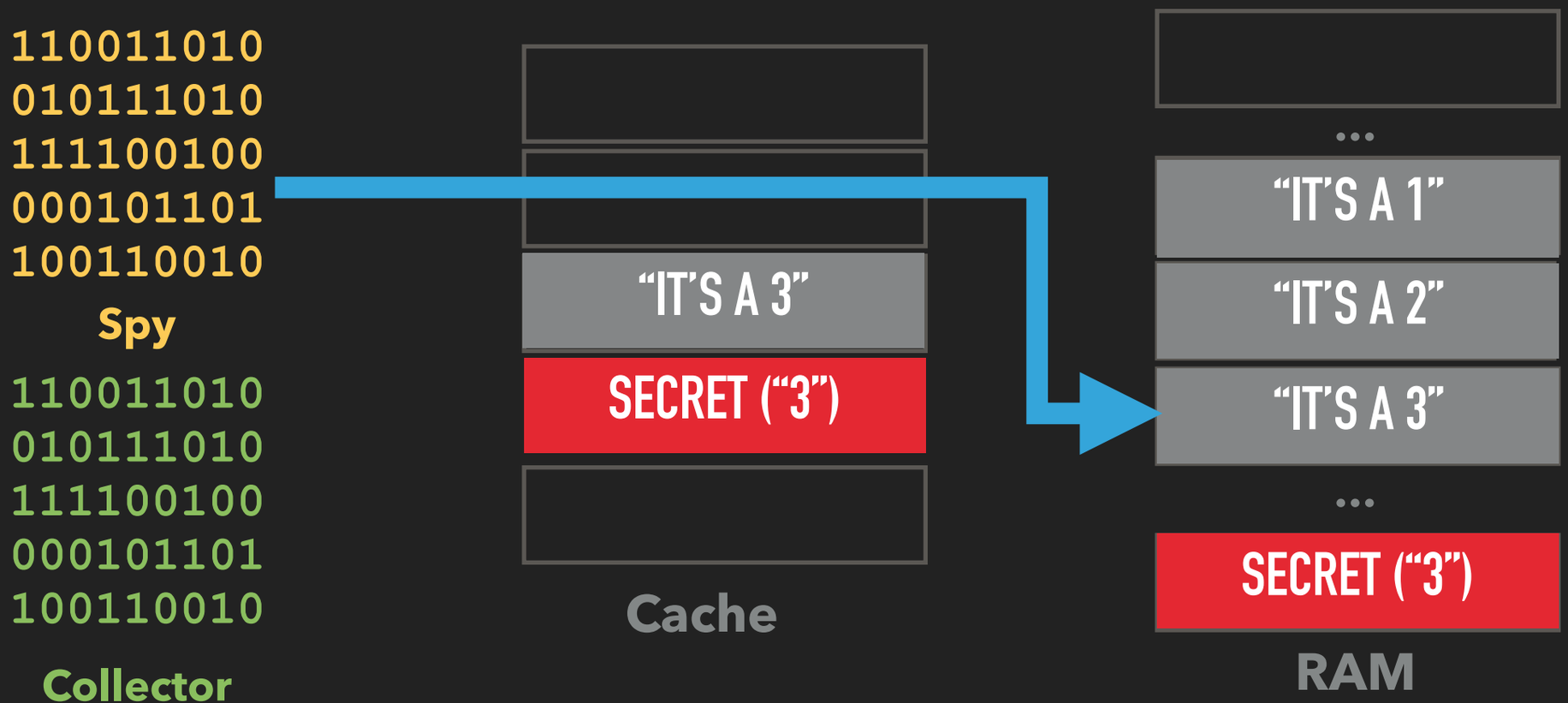
MELTDOWN: THE ATTACK



MELTDOWN: THE ATTACK

2 1. **Spy** will read the **secret**

MELTDOWN: THE ATTACK



2

1. **Spy** will read the **secret**
2. Depending on the **value**, **Spy** will cache a grey block[1]

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK

2 1. **Spy** will read the **secret**

2. Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK

2 1. **Spy** will read the **secret**

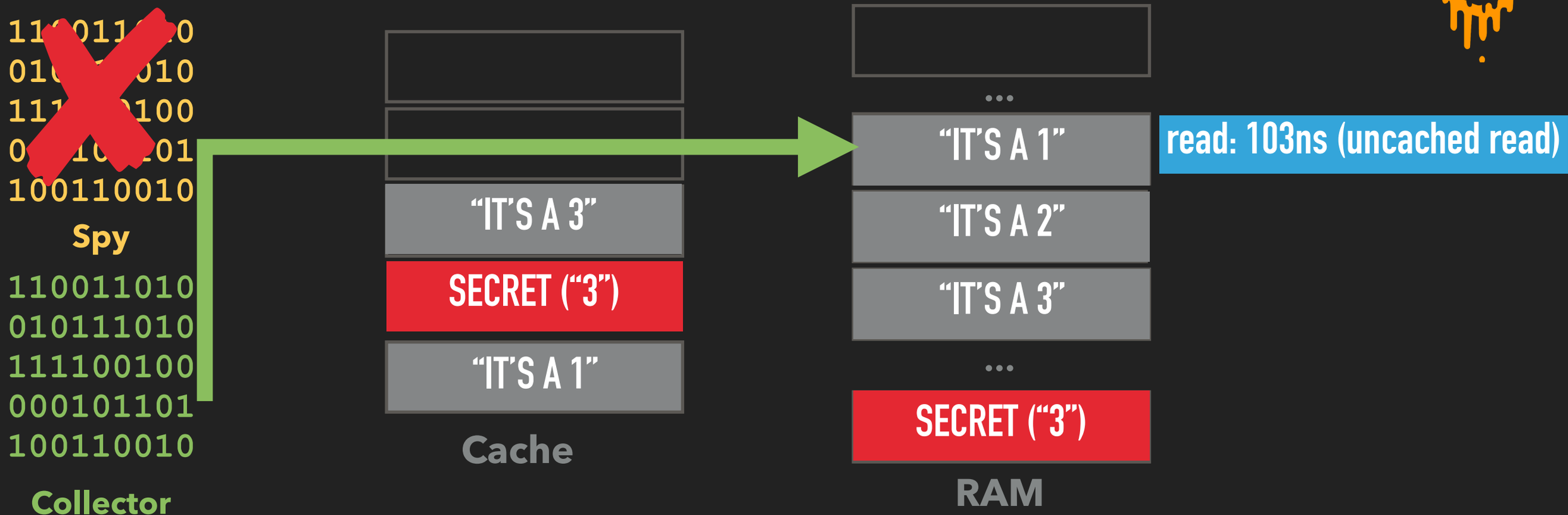
2. Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK



2 1. **Spy** will read the **secret**

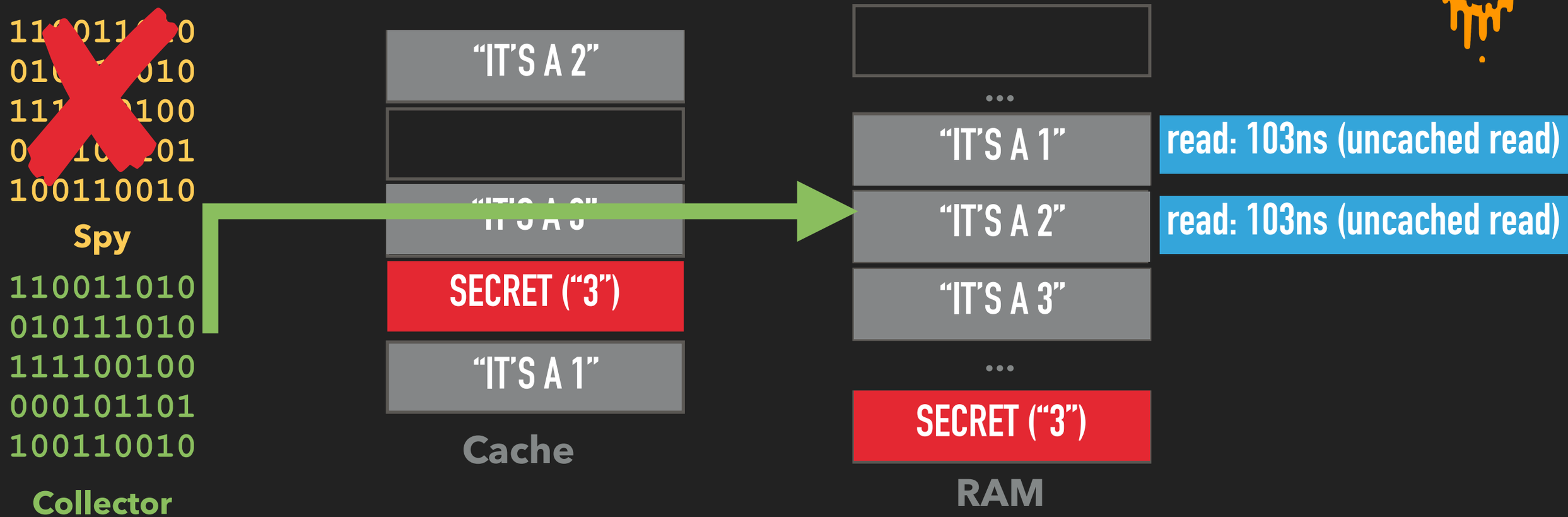
2. Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK



2 1. **Spy** will read the **secret**

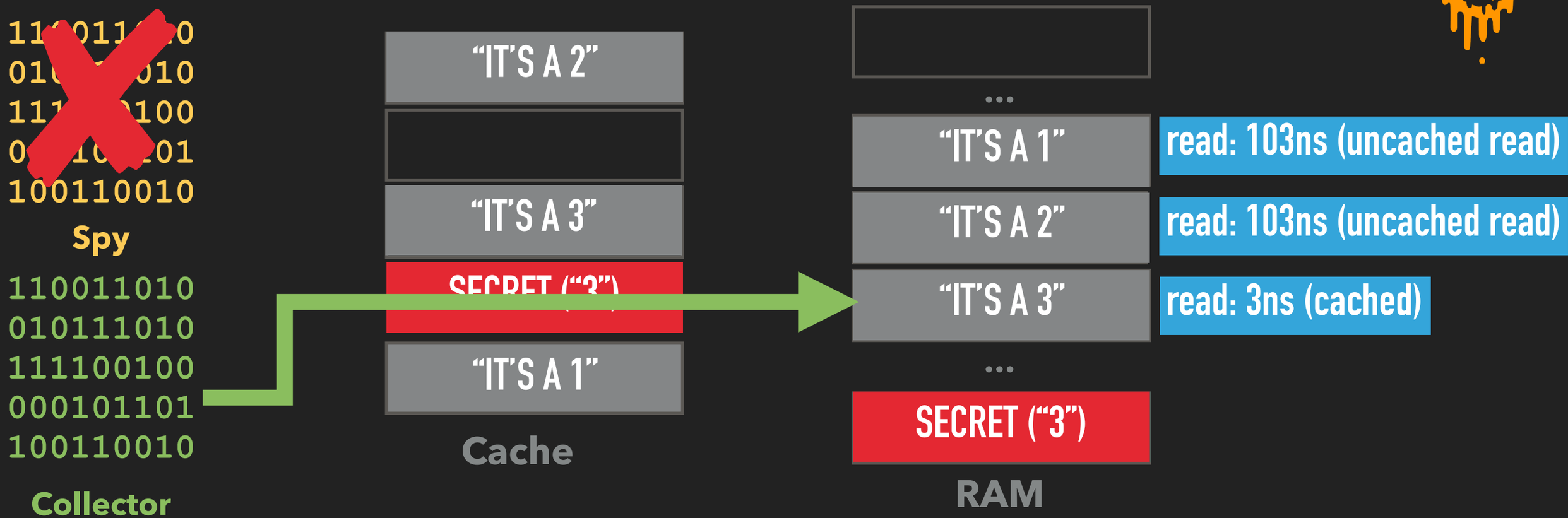
2. Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK



2 1. **Spy** will read the **secret**

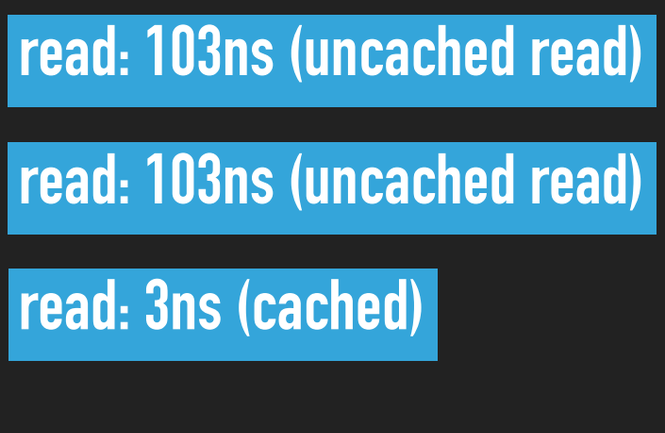
2. Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK



2 1. **Spy** will read the **secret**

2. Depending on the **value**, **Spy** will cache a grey block[1]

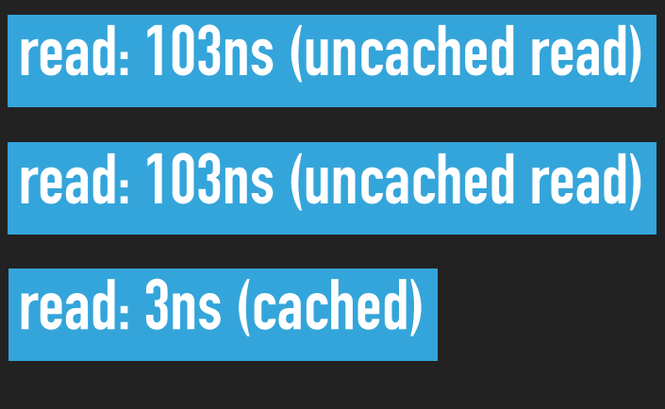
1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

1. Block "It's a 3" will be the block read the fastest

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK



2 1. **Spy** will read the **secret**

2 2 Depending on the **value**, **Spy** will cache a grey block[1]

1 3. CPU detects **Spys** access validation and terminates **Spy**

4. **Collector** now reads all grey blocks and stops the time

1. Block "It's a 3" will be the block read the fastest

[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN

Meltdown exploits two properties of modern CPUs

- *Out of order execution* of OPs and µOPs
- Timing side channels for the cache

This allows an attacker to

- Read all memory mapped[1] in a process
- This often includes all other processes memory
- This does NOT allow reading “outside of a VM[2]”

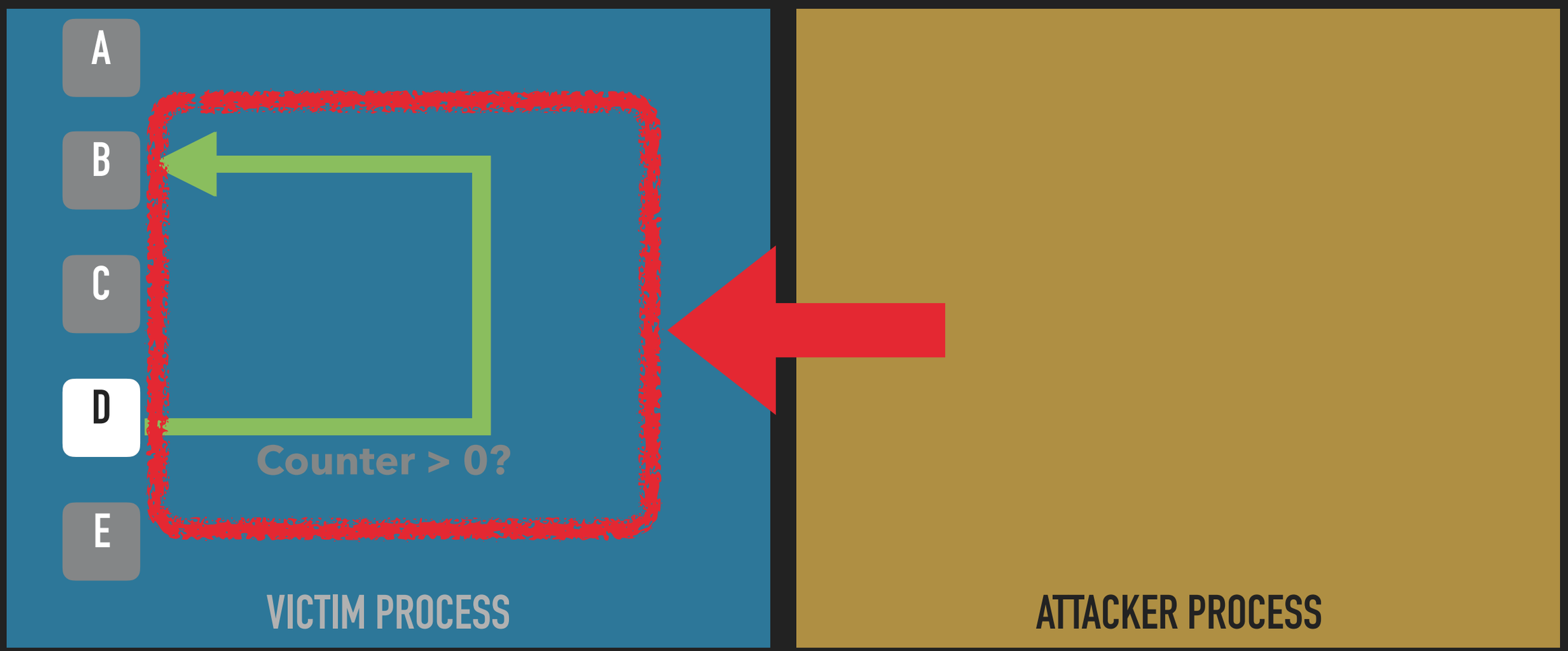
[1] Virtual vs. physical memory is a subject for another time [2] For fully virtualised VMs

SPECULATIVE
EXECUTION

SPECTRE

SPECTRE

Spectre attacks other processes by forcing them to *speculatively* run other code paths



SPECTRE

Spectre works like this:

- force victim to leak secret
- stash away secret
- retrieve secret

SPECTRE

Spectre works like this:

- force victim to leak secret
- stash away secret
- retrieve secret

- and *basically* work like in Meltdown

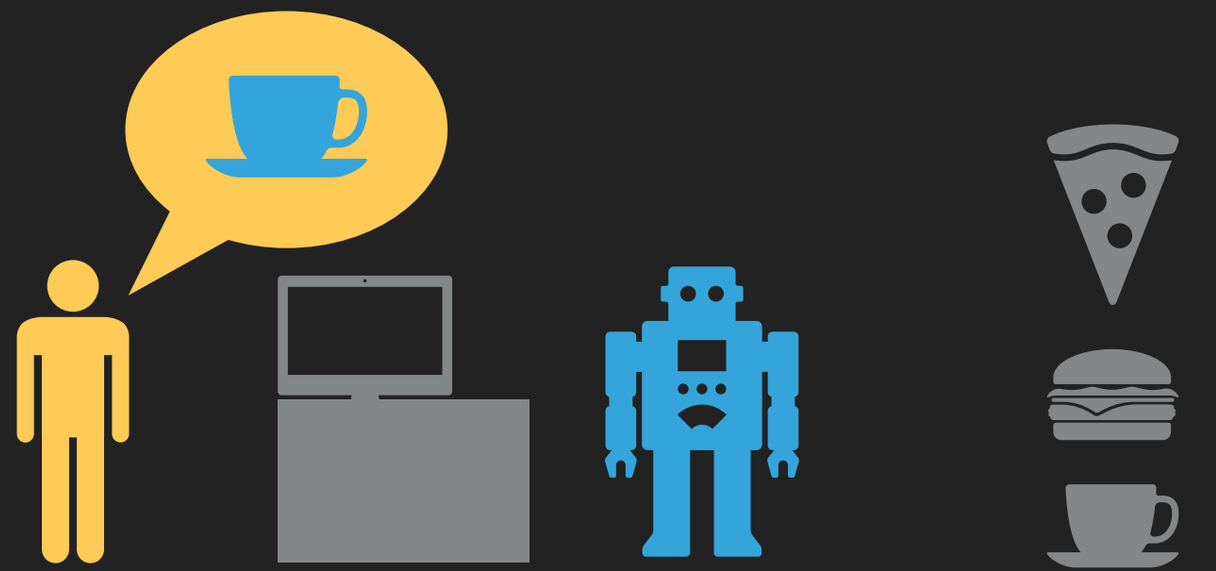
SPECTRE

Spectre works like this:

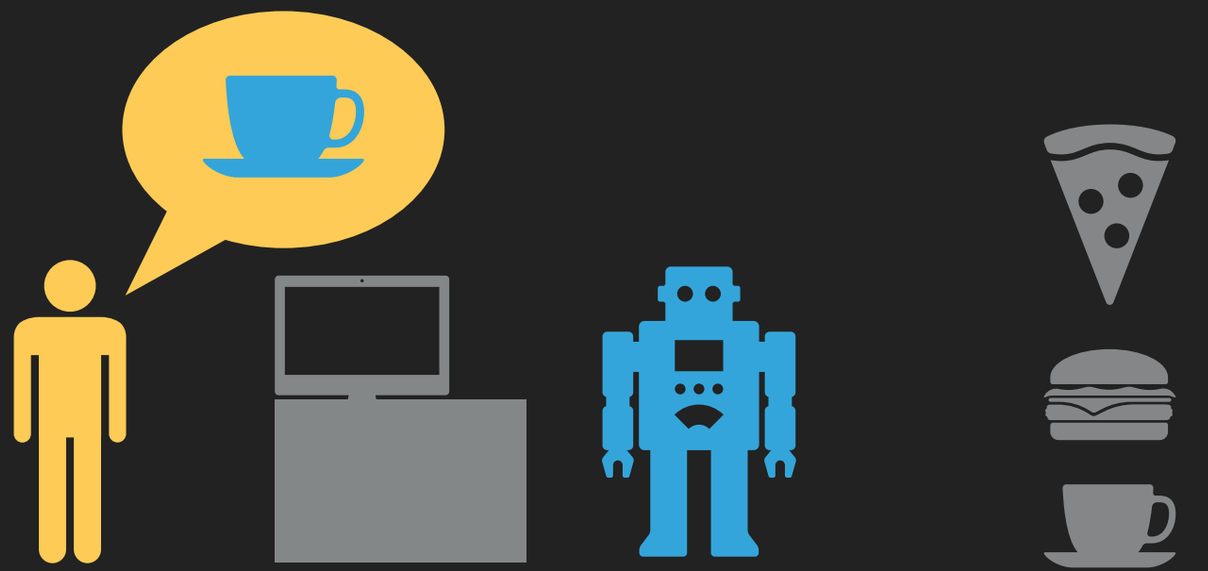
- force victim to leak secret
- stash away secret
- retrieve secret

- and *basically* work like in Meltdown
- works by manipulating the *branch prediction* of the CPU

SPECTRE: BRANCH PREDICTION

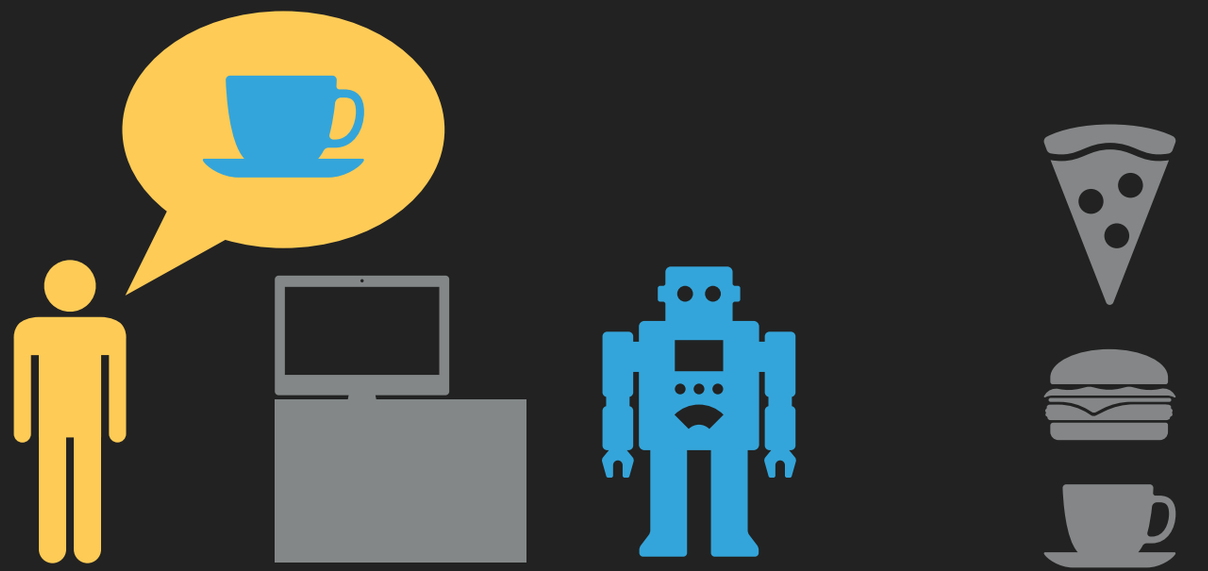


SPECTRE: BRANCH PREDICTION



Monday

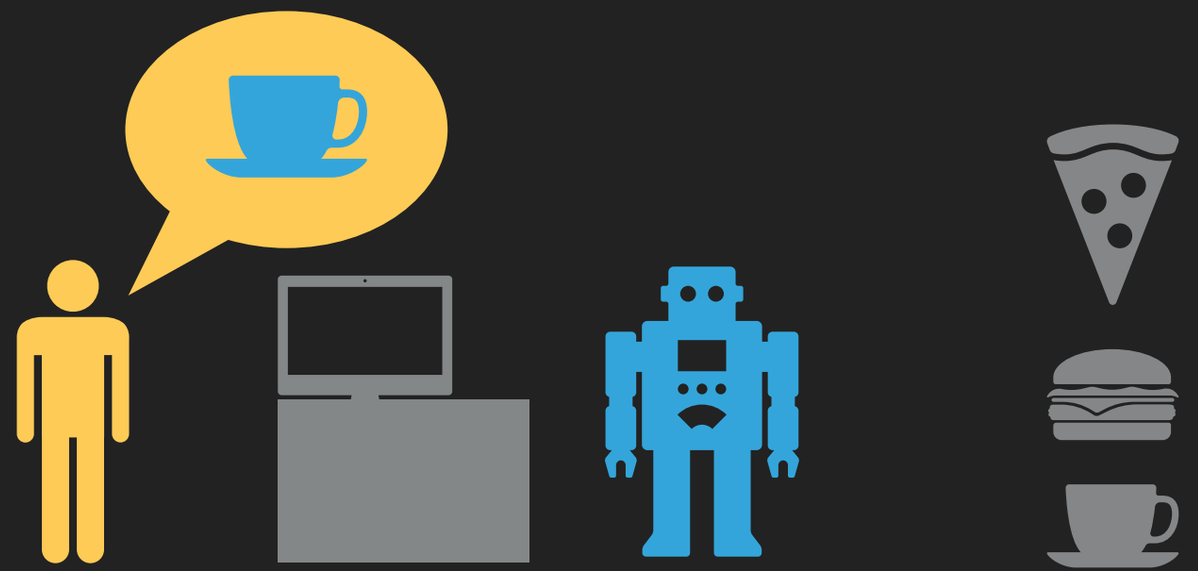
SPECTRE: BRANCH PREDICTION



Monday

Tuesday

SPECTRE: BRANCH PREDICTION

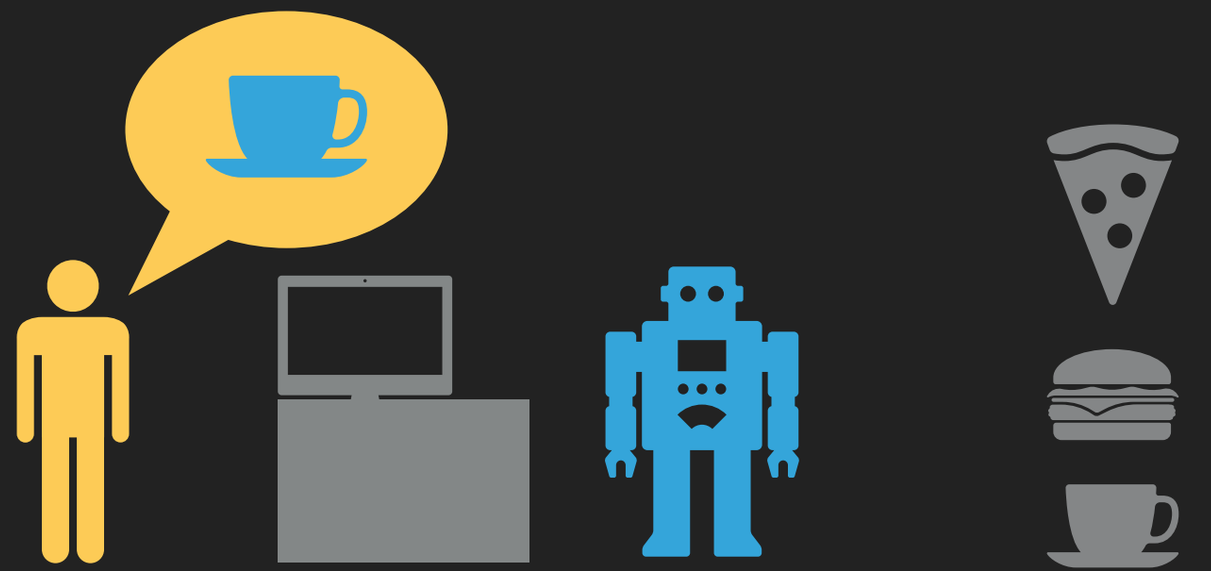


Monday

Wednesday

Tuesday

SPECTRE: BRANCH PREDICTION



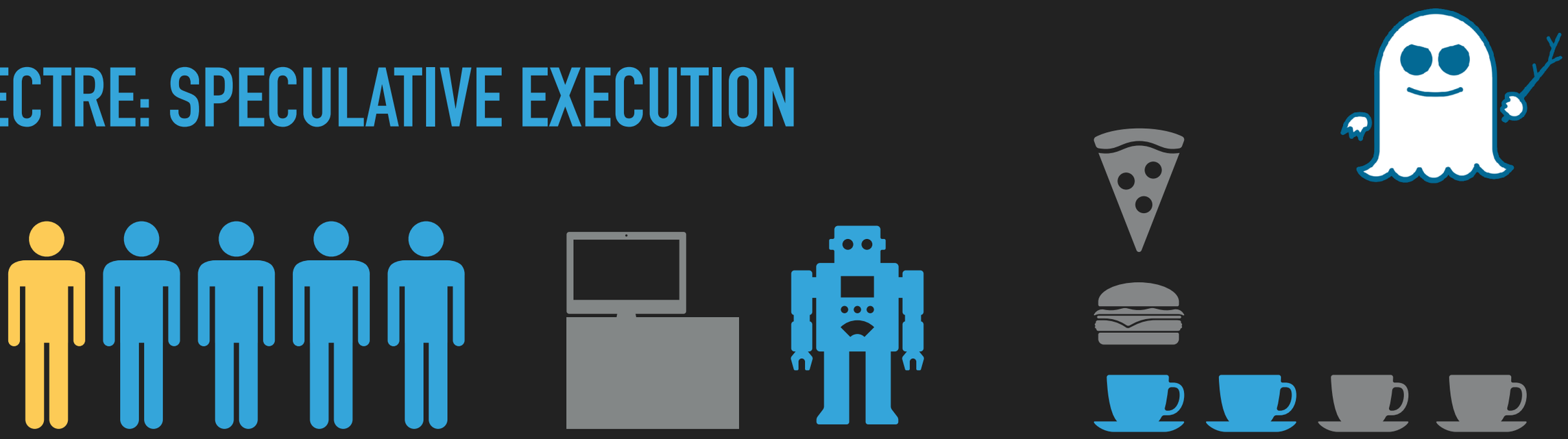
Monday

Wednesday

Tuesday

...

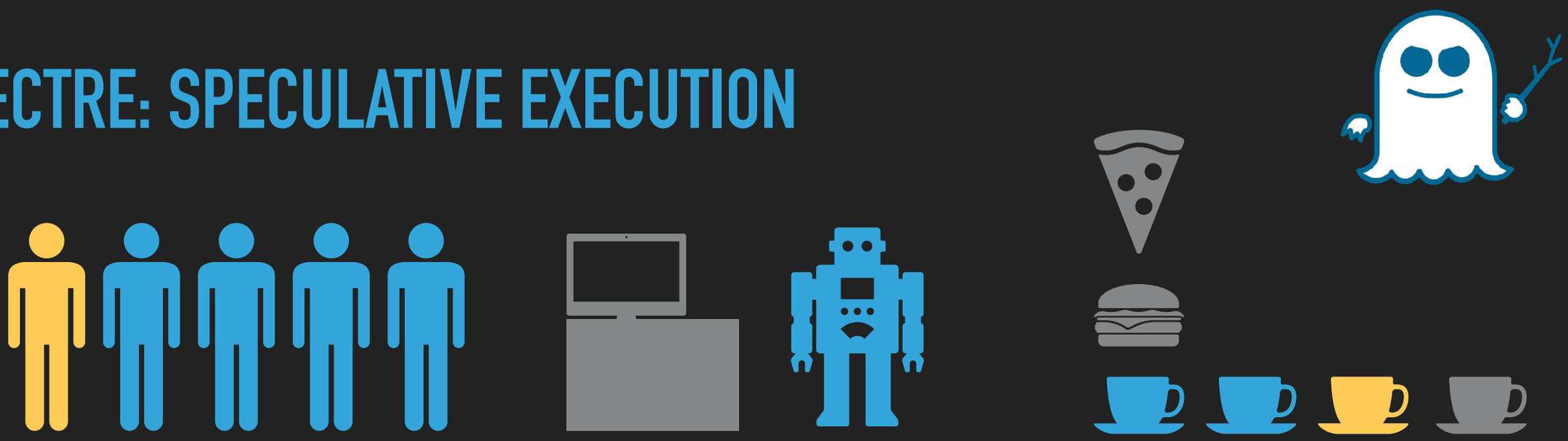
SPECTRE: SPECULATIVE EXECUTION



The CPU can improve the coffee machine utilisation by *speculatively* brewing the coffee for

This is very similar to the effect seen in Meltdown.

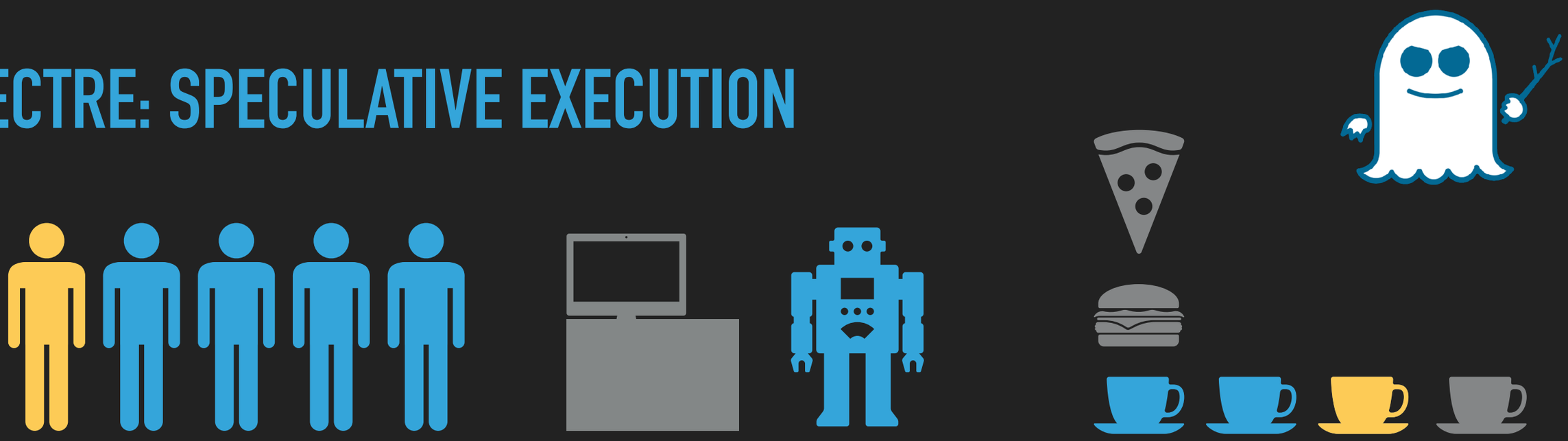
SPECTRE: SPECULATIVE EXECUTION



The CPU can improve the coffee machine utilisation by *speculatively* brewing the coffee for

This is very similar to the effect seen in Meltdown.

SPECTRE: SPECULATIVE EXECUTION

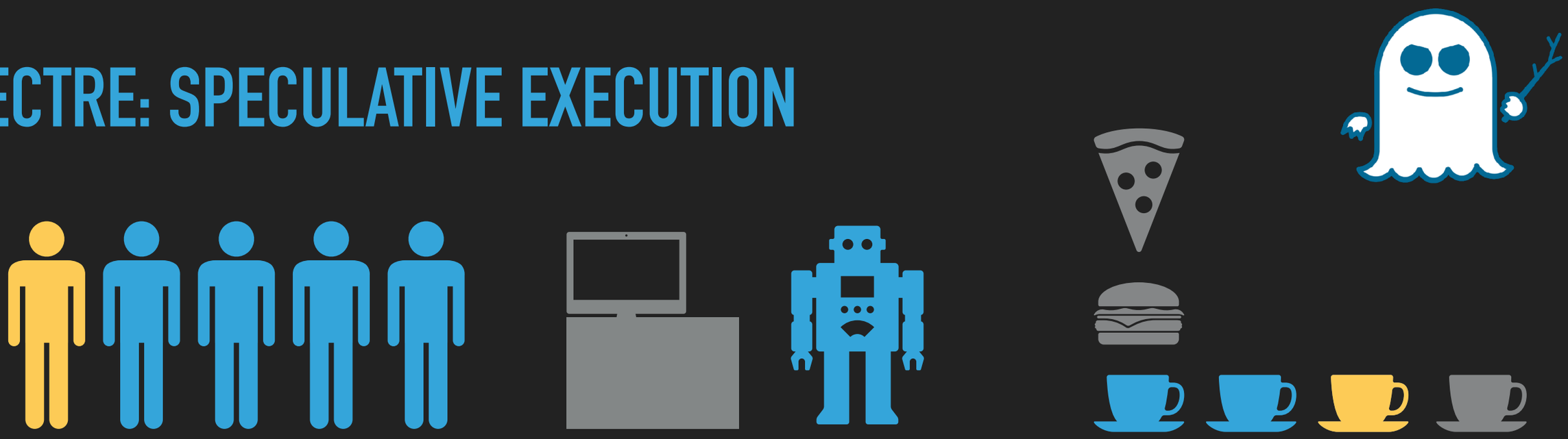


The CPU can improve the coffee machine utilisation by *speculatively* brewing the coffee for

This is very similar to the effect seen in Meltdown.

- In the **Meltdown** attack the CPU knows the next instruction (order) and asynchronously checks the permissions

SPECTRE: SPECULATIVE EXECUTION

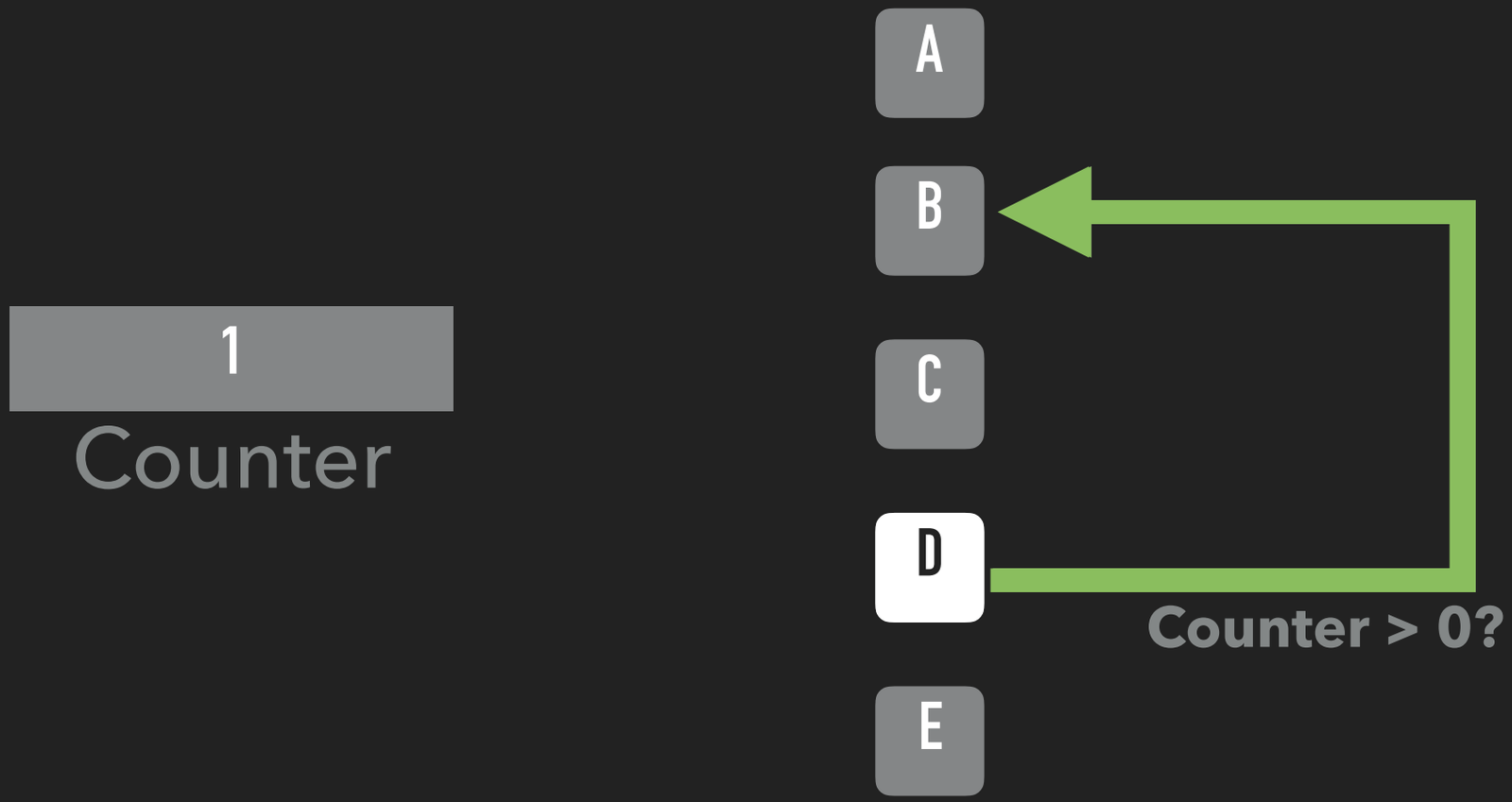


The CPU can improve the coffee machine utilisation by *speculatively* brewing the coffee for

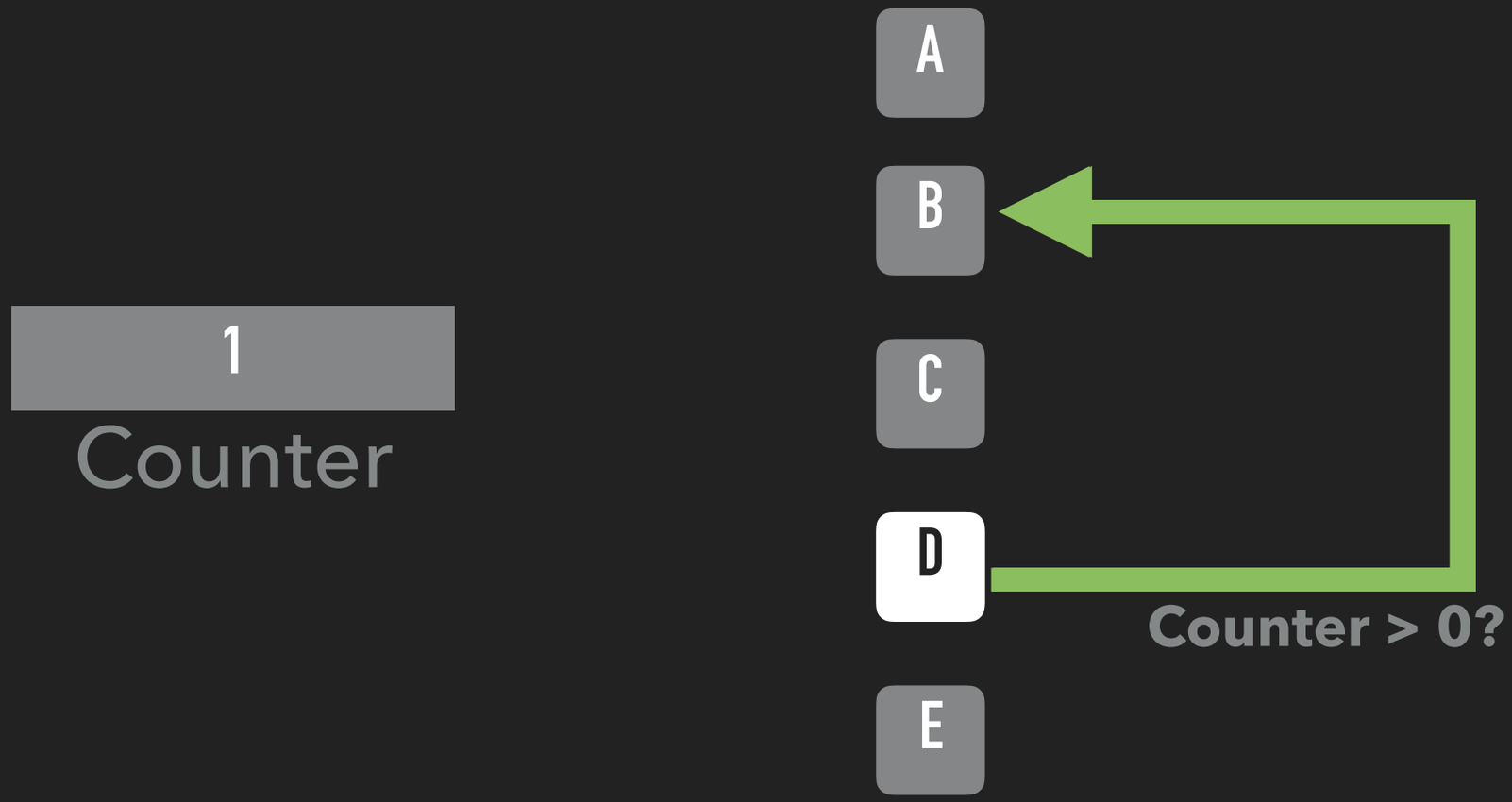
This is very similar to the effect seen in Meltdown.

- In the **Meltdown** attack the CPU knows the next instruction (order) and asynchronously checks the permissions
- In **Spectre** the CPU guesses the next instructions based on heuristics (brew coffee without knowing the order)

SPECTRE: SPECULATIVE EXECUTION

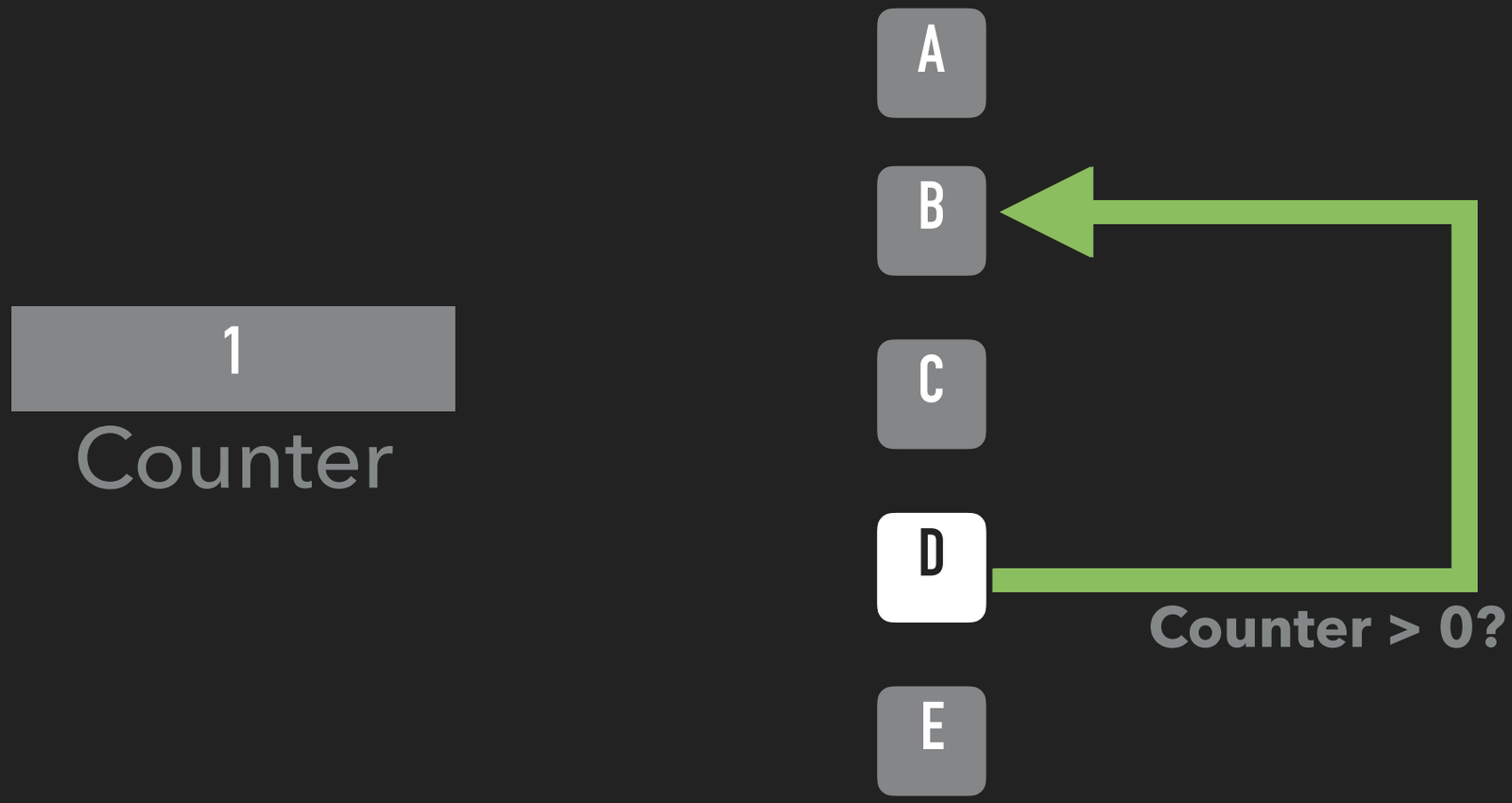


SPECTRE: SPECULATIVE EXECUTION



The CPU has learned that Counter *probably* is > 0

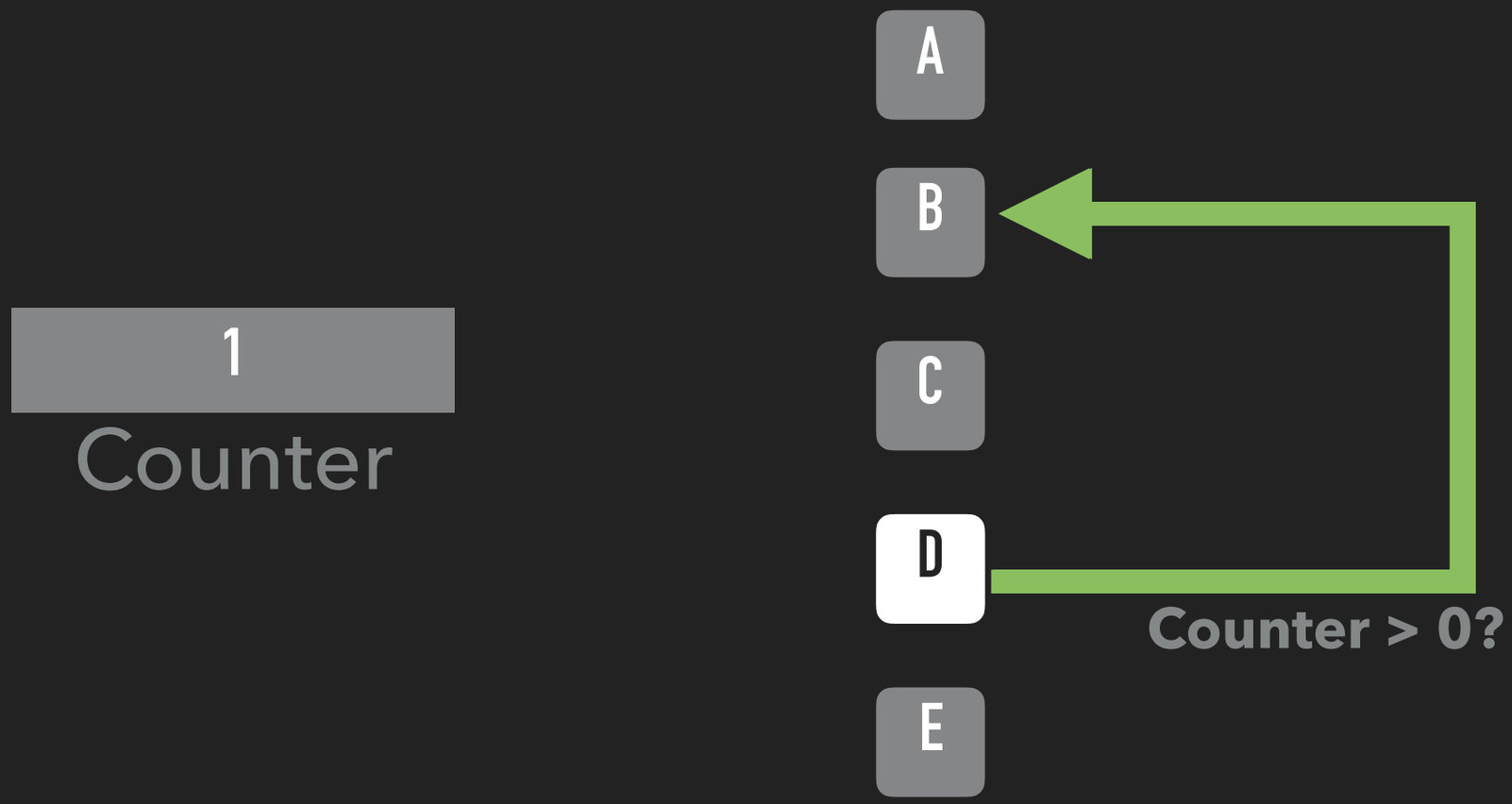
SPECTRE: SPECULATIVE EXECUTION



The CPU has learned that Counter *probably* is > 0

Reading Counter from memory is very slow

SPECTRE: SPECULATIVE EXECUTION

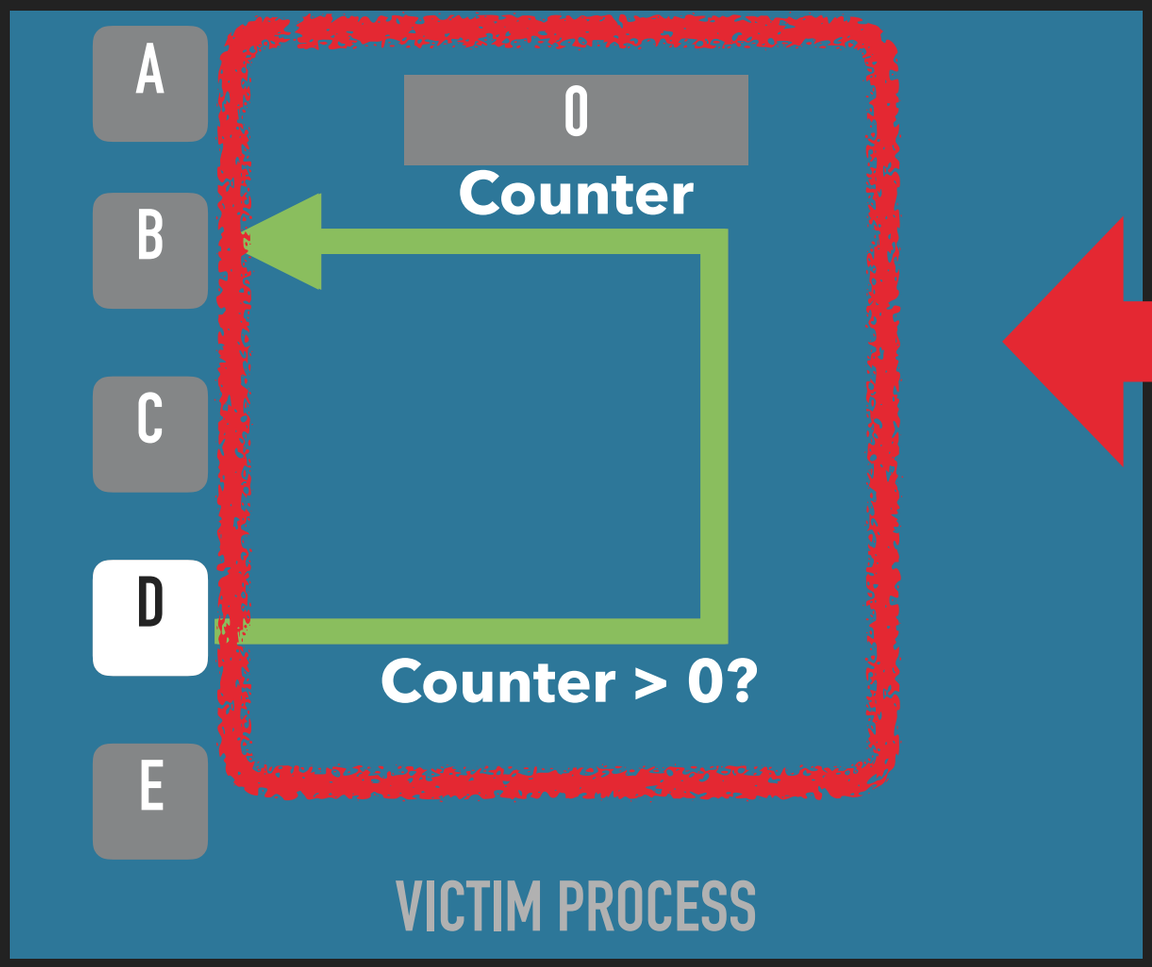


The CPU has learned that Counter *probably* is > 0

Reading Counter from memory is very slow

The CPU *speculatively* executes B C to improve performance

SPECTRE: SPECULATIVE EXECUTION



1. **Prime the branch prediction to expect a loop**
2. **Make sure *Counter* is not cached so the CPU is more likely to *speculatively* run the code**
3. **Find a way that victim leaks data when B & C are executed speculatively**

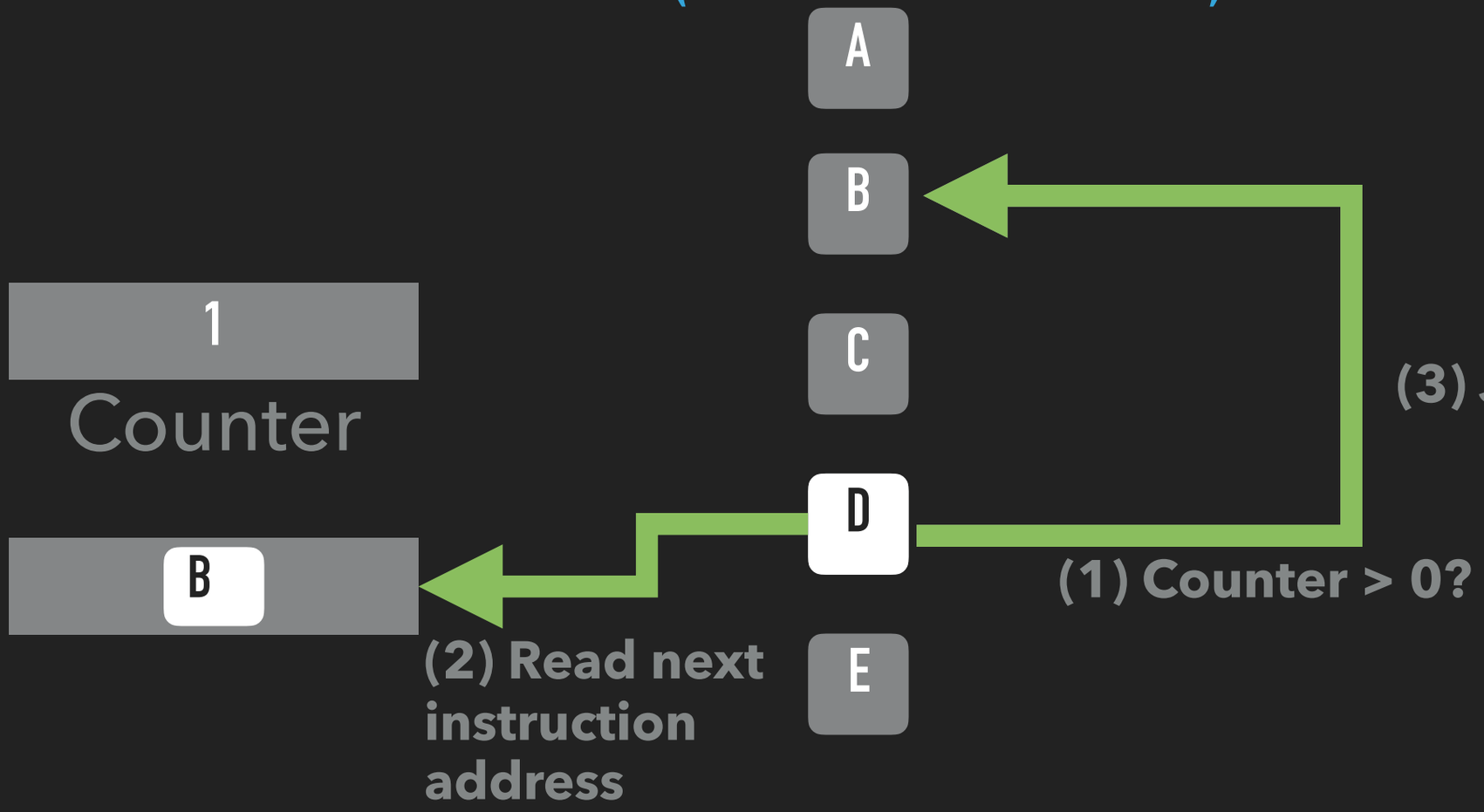
ATTACKER PROCESS

Attacker can influence the CPUs branch prediction of victim.

Making the victim *speculatively* execute “wrong” code.

E.g. loop even when Counter is == 0.

SPECTRE: VARIANT 2 (CVE-2017-5715)



- The conditional jump (branch) D now is an *indirect jump*.
- Indirect jumps use addresses stored "somewhere else".
- This can also be used to *speculatively* execute any code found in the target process (kernel).

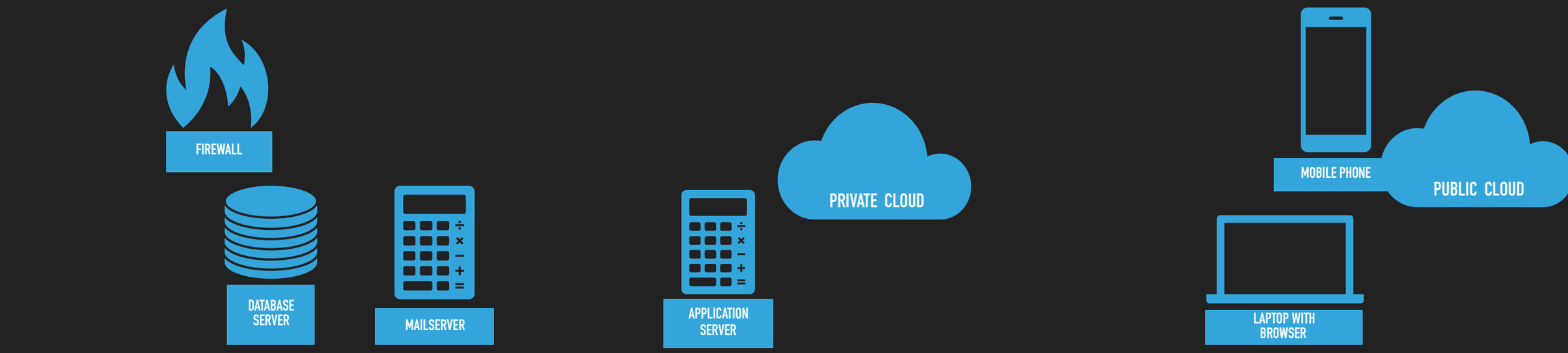
MELTDOWN AND SPECTRE

CONCLUSION

THREAT-O-METER

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Meltdown and Spectre exploit side effects of modern CPU architectures.

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Meltdown and Spectre exploit side effects of modern CPU architectures.

The *exploitability* very much depends on the field of application of the system.

BLIC CLOUD

LOW RISK	MEDIUM RISK	HIGH RISK
Exploit unlikely or running untrusted code already worst case	Exploit possible but needs another successful attack to run attackers code	Exploit possible and runs untrusted code “by design”

THREAT-O-METER

Meltdown and Spectre exploit side effects of modern CPU architectures.

The *exploitability* very much depends on the field of application of the system.

Expect new “bugs” of this type!

Exploit unlikely or running untrusted code already worst case

Exploit possible but needs another successful attack to run attackers code

Exploit possible and runs untrusted code “by design”

MELTDOWN

THREAT-

Melt ern

CPU

The of

appl

Expe

untrusted code already worst case

successful attack to run attackers code

Exp le and code “by design”

Exclusive: Spectre-NG - Multiple new Intel CPU flaws revealed, several serious

Jürgen Schmidt 03.05.2018

Intel Core i, Meltdown und Spectre, Prozessoren, Sicherheitslücken, Spectre

New flaws and even more patches - "Spectre Next Generation" is just around the corner. According to information exclusively available to c't, researchers have already found eight new security holes in Intel processors.

Knock knock

Branch prediction

Who's there?

Q & A

https://github.com/neuhalje/presentation_meltdown_spectre

Something missing?

Boring?

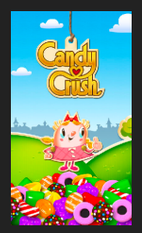
Awesome?

Feedback helps!

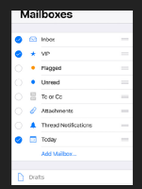
FEEDBACK

https://github.com/neuhalje/presentation_meltdown_spectre

BOTH THE MELTDOWN AND SPECTRE LOGO ARE FREE TO USE, RIGHTS WAIVED VIA CCO. LOGOS ARE DESIGNED BY NATASCHA EIBL.
HTTPS://SPECTREATTACK.COM/

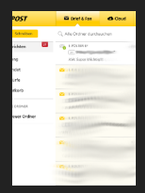


CANDY CRUSH LOGO FROM THE APP STORE © KING
HTTPS://DISCOVER.KING.COM/ABOUT/



APPLE MAIL SCREENSHOT © APPLE
HTTPS://SUPPORT.APPLE.COM/EN-GB/HT207213

SCREENSHOT © BUNDESNACHRICHTENDIENST
HTTPS://WWW.BND.BUND.DE/EN/_HOME/HOME_NODE.HTML



SCREENSHOT E-POST
HTTPS://PORTAL.E-POST.DE

HEISE.DE
https://www.heise.de/ct/artikel/Exclusive-Spectre-NG-Multiple-new-Intel-CPU-flaws-revealed-several-serious-4040648.html

ASSETS

ABANDONED SLIDES