EXPLAINED:

MELTDOWN & SPECTRE

Jens Neuhalfen

https://github.com/neuhalje/presentation_meltdown_spectre

EXPLAINED:

MELTDOWN & SPECTRE

for normal people

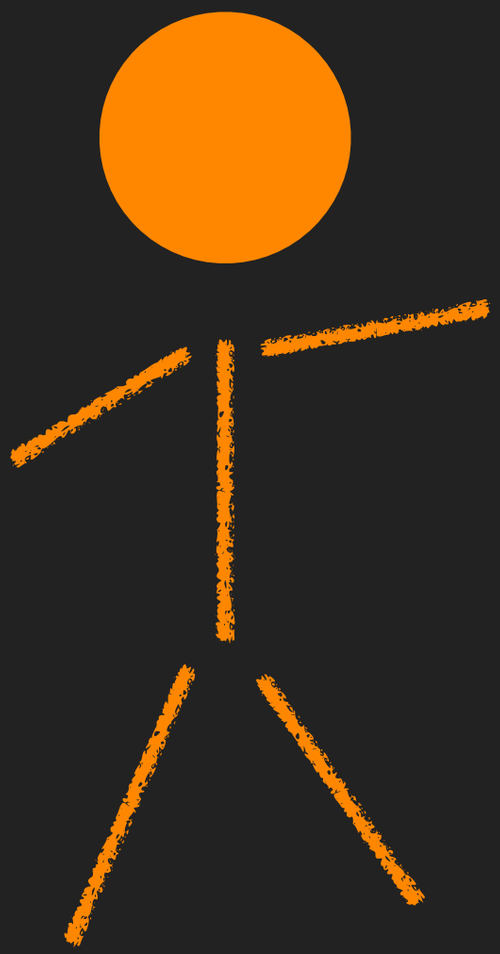
https://github.com/neuhalje/presentation_meltdown_spectre

Jens Neuhalfen

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

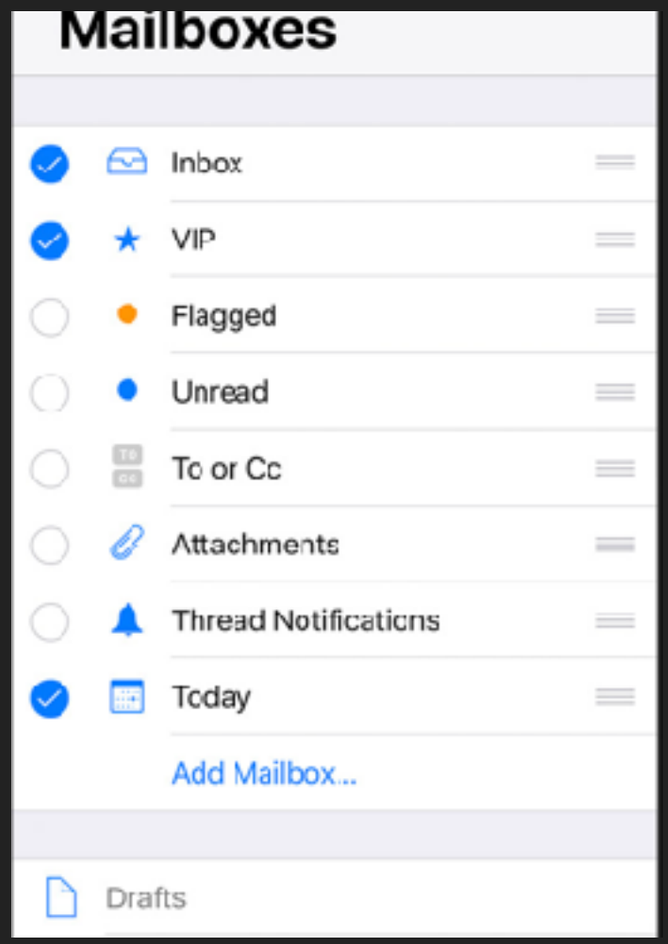
SECURITY GOALS: APP ISOLATION

SECURITY GOALS: APP ISOLATION

You don’t want
this

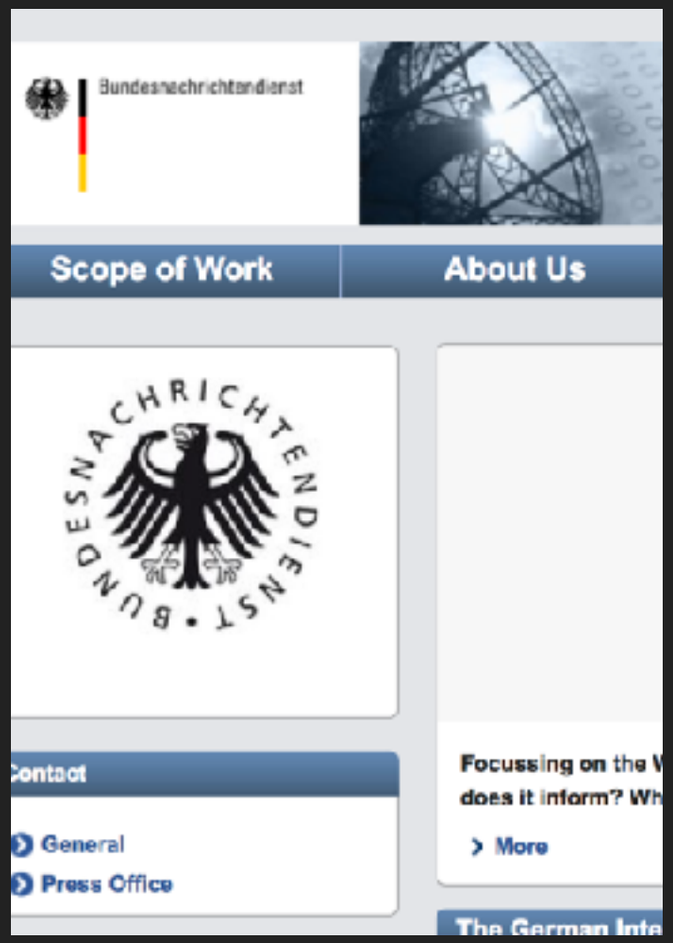
SECURITY GOALS: APP ISOLATION

You don't want
this



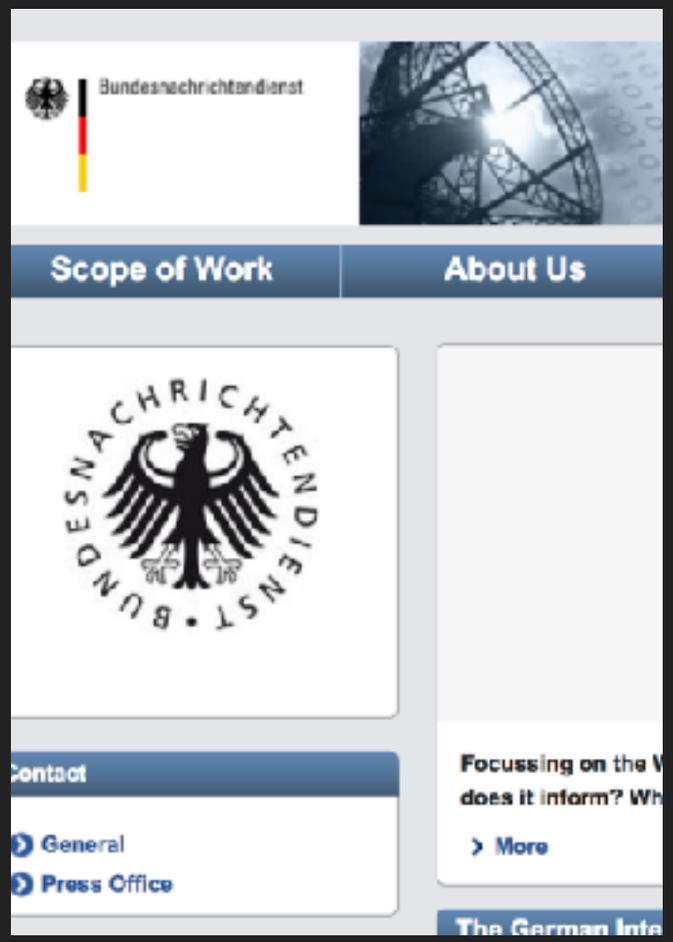
To read
that

SECURITY GOALS: BROWSER TAB ISOLATION

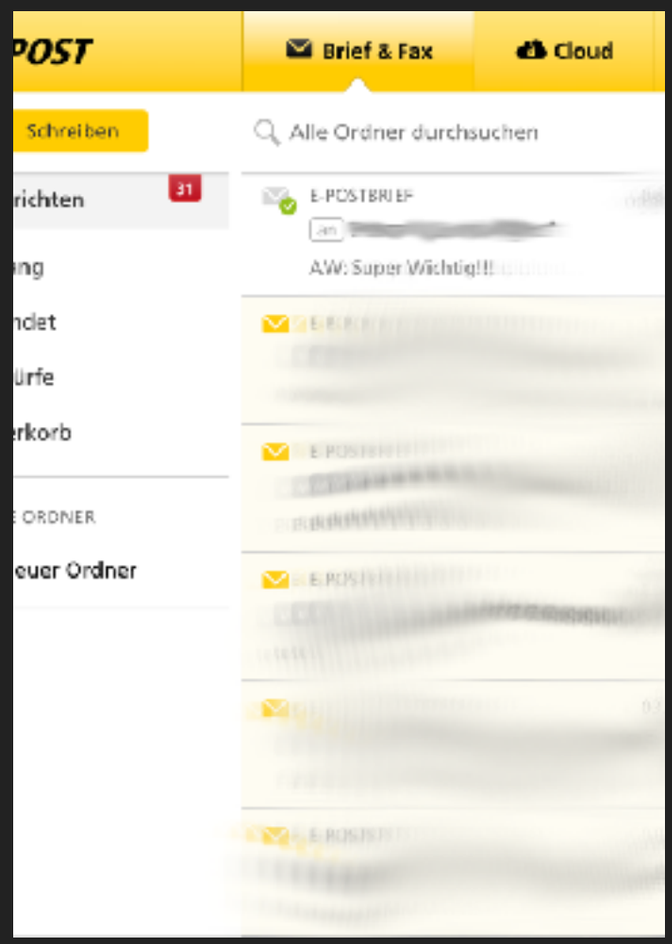


You don’t want *this*

SECURITY GOALS: BROWSER TAB ISOLATION



You don't want *this*



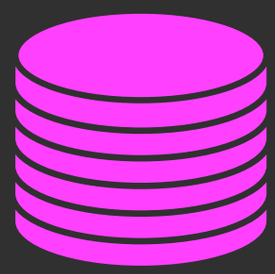
To read *that*

SECURITY GOALS: CLOUD ISOLATION

You don’t want

this

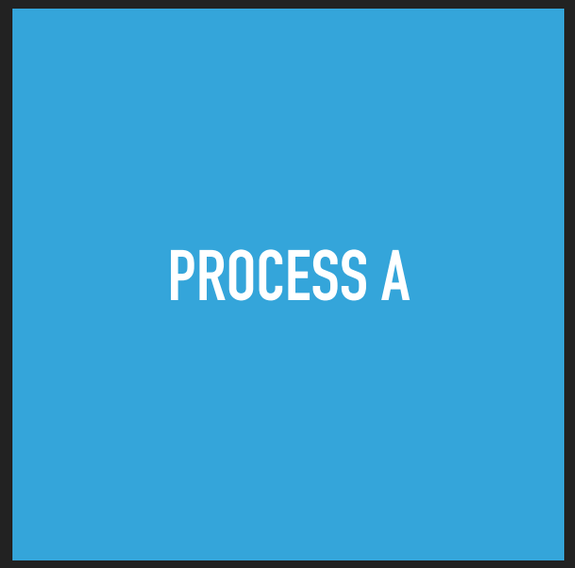
SECURITY GOALS: CLOUD ISOLATION



You don't want
this

To read
that

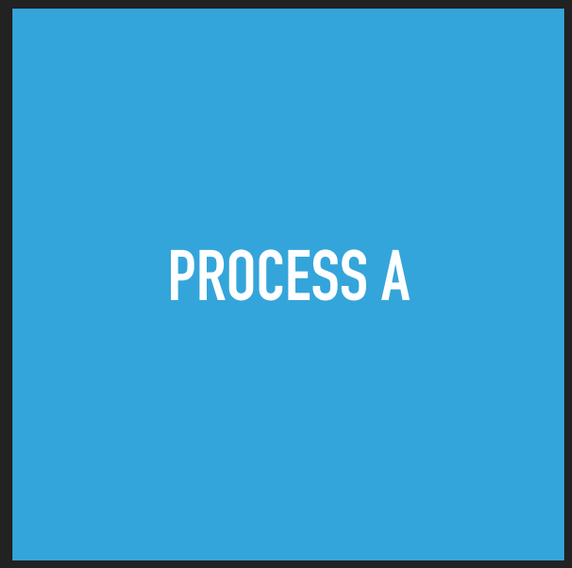
SECURITY GOALS: MEMORY ISOLATION



You don’t want

this

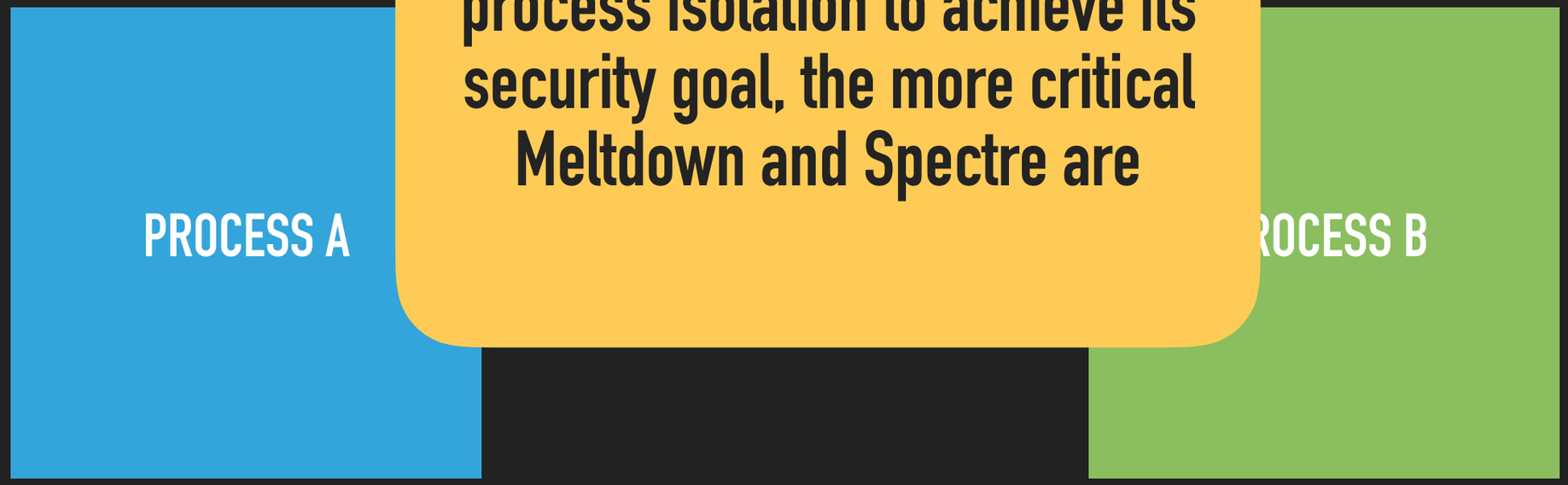
SECURITY GOALS: MEMORY ISOLATION



You don't want
this

To read
that

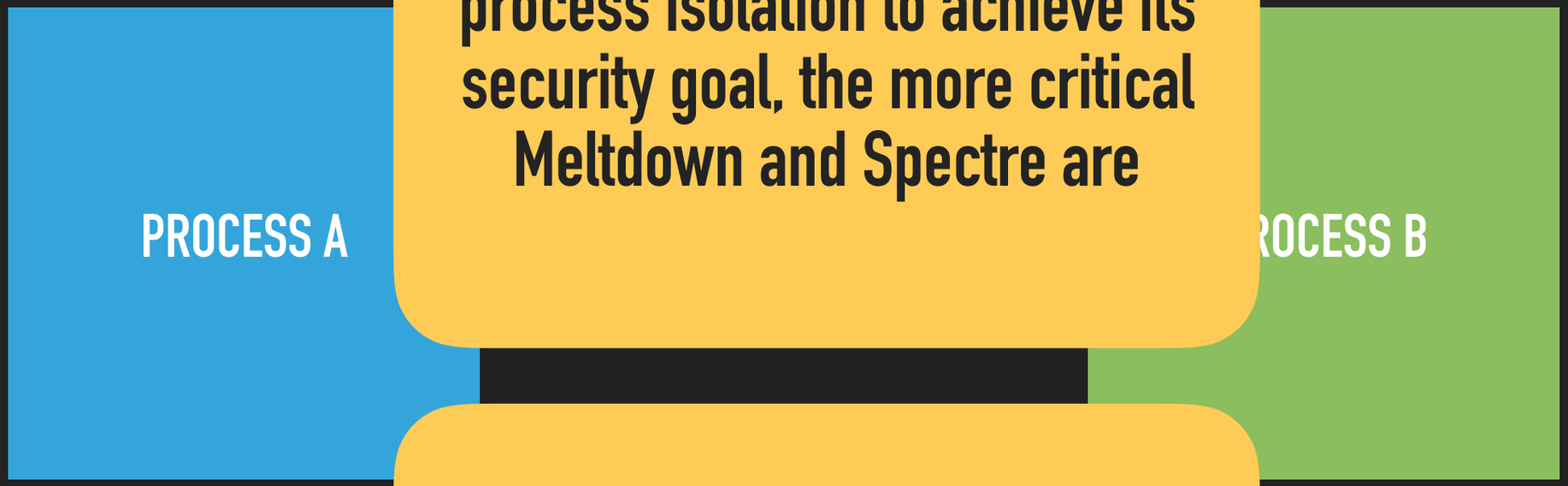
SECURITY GOALS: MEMORY ISOLATION



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 utilise the Meltdown and Spectre attacks

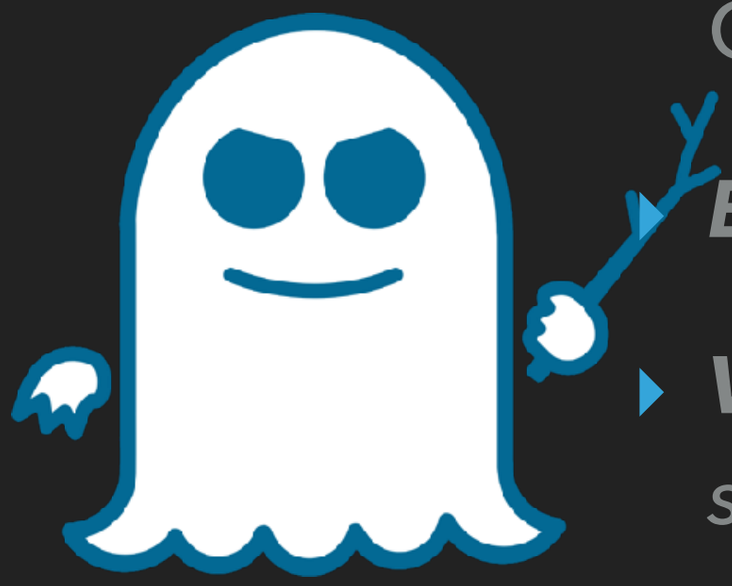
You don't read
this *hat*

MELTDOWN

- ***Result***:Programs can read memory it should not
- ***Affects***: All modern CPU/OS
- ***Vector***: Uses *speculative execution* to read forbidden memory and *cache timing* as side channel to exfiltrate data
- ***How bad***: Very bad
- ***Fixes***: Mainly OS patches. Negligible performance impact on modern CPU. High impact on older CPU

SPECTRE

TODO

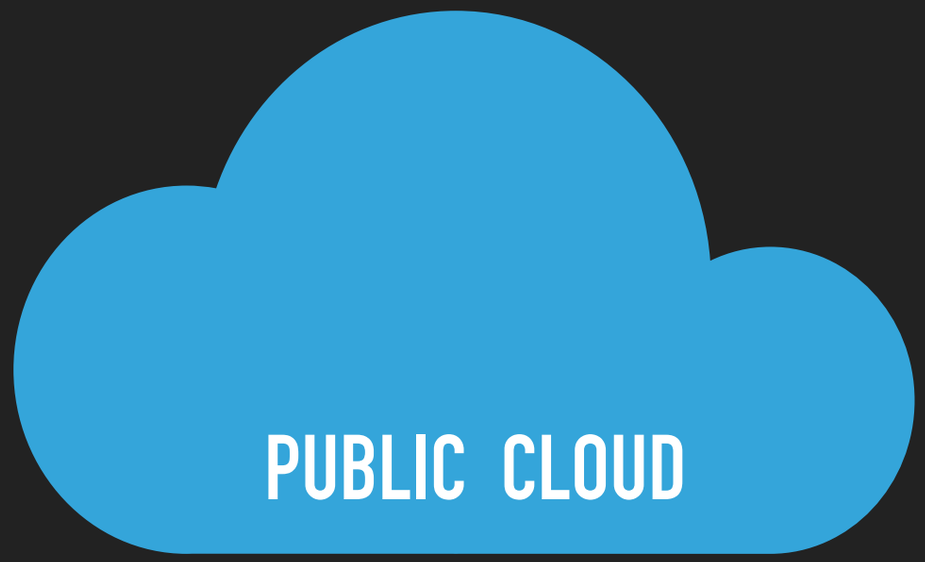


- ***Result***: Programs can read all memory
- ***Affects***: All modern CPU/OS
- ***Break out of VM***: No
- ***Vector***: Uses *speculative execution* to read forbidden memory and *cache timing* to exfiltrate data
- ***How bad***: Very bad
- ***Fixes***: Mainly OS patches. Negligible performance impact on modern CPU. High impact on older CPU

THREAD-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”

THREAD-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”

THREAD-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”

THREAD-O-METER

DATABASE SERVER

PUBLIC CLOUD

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”

THREAD-O-METER

LOW RISK

Exploit unlikely or running untrusted code already worst case

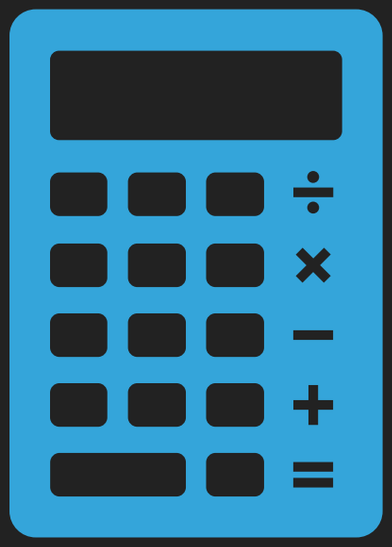
MEDIUM RISK

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.

THREAD-O-METER



MAILSERVER

LOW RISK

Exploit unlikely or running untrusted code already worst case

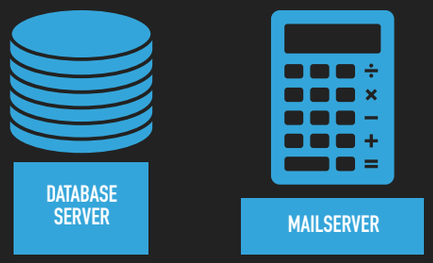
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-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

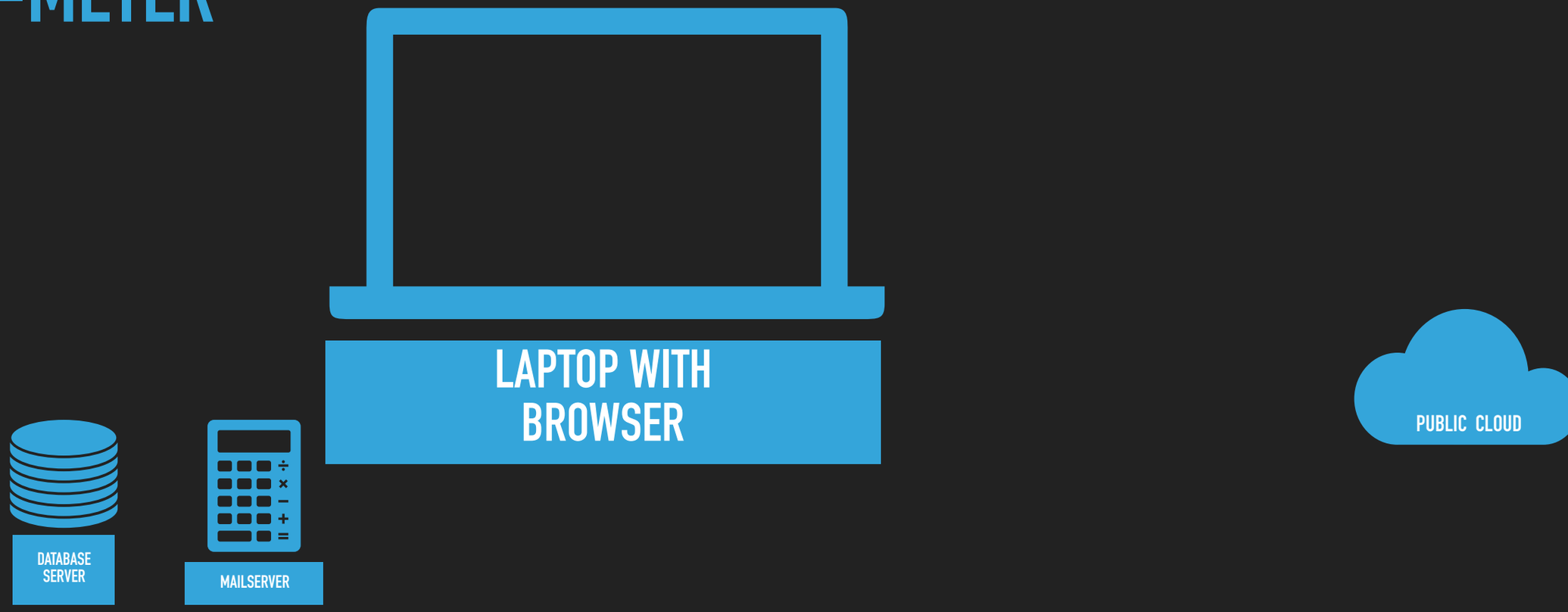
…design"

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.

THREAD-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

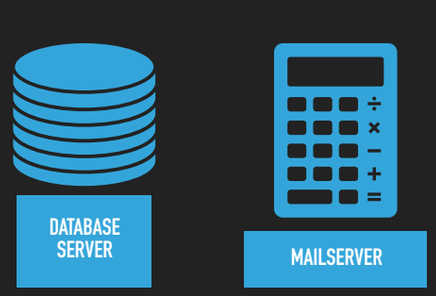
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-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

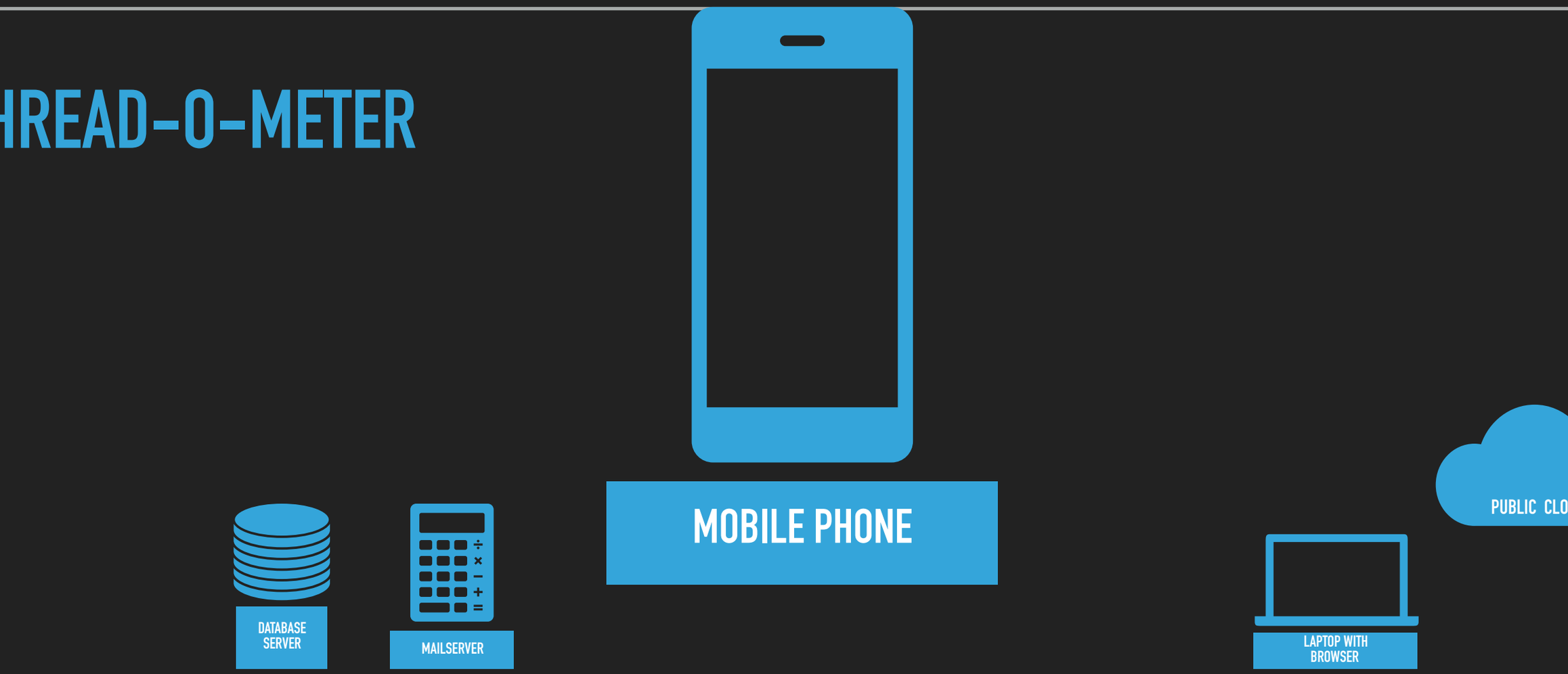
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

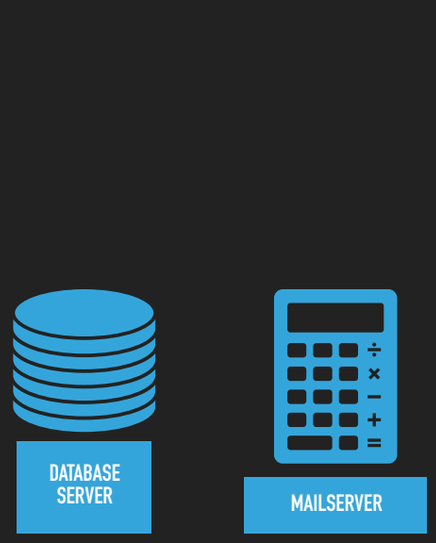
Exploit possible but needs another successful attack to run attackers code

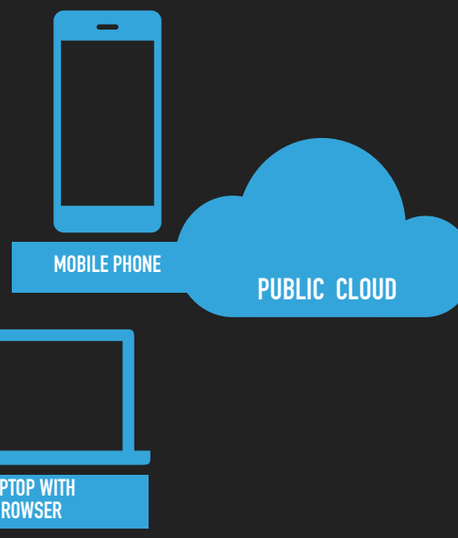
HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER

Mobile phones run apps and websites (JavaScript)





LOW RISK

Exploit unlikely or running untrusted code already worst case

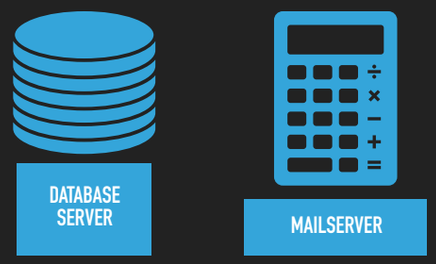
MEDIUM RISK

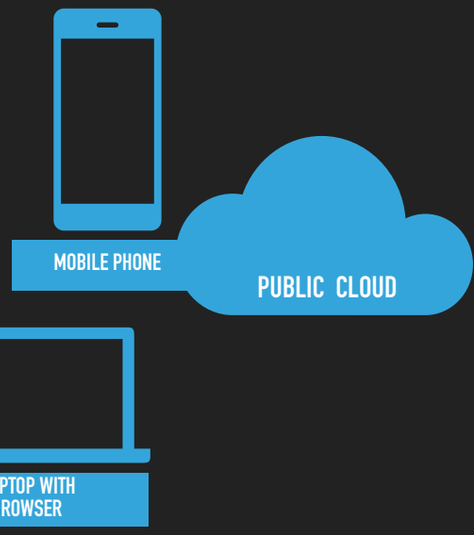
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER





LOW RISK

Exploit unlikely or running untrusted code already worst case

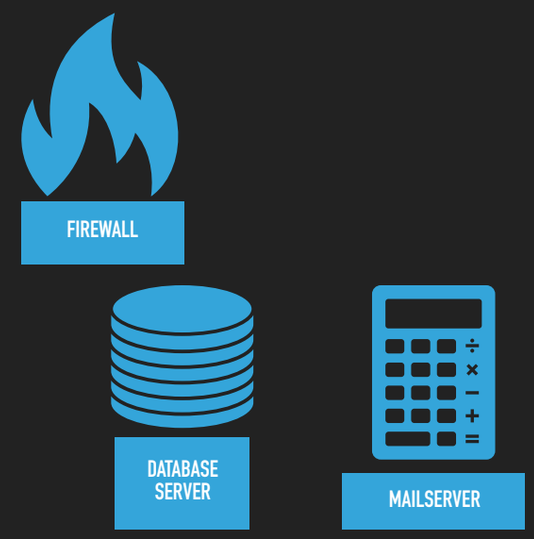
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

Exploit possibl needs anoth successful attack attackers co

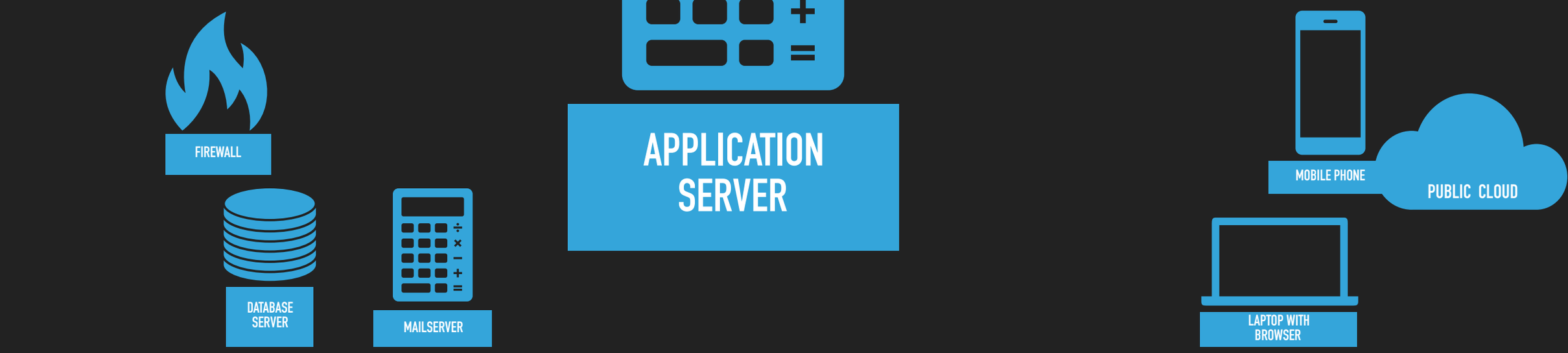
Firewalls and switches (normally) do not expose an attackable surface to the external network.

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.

THREAD-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

MEDIUM RISK

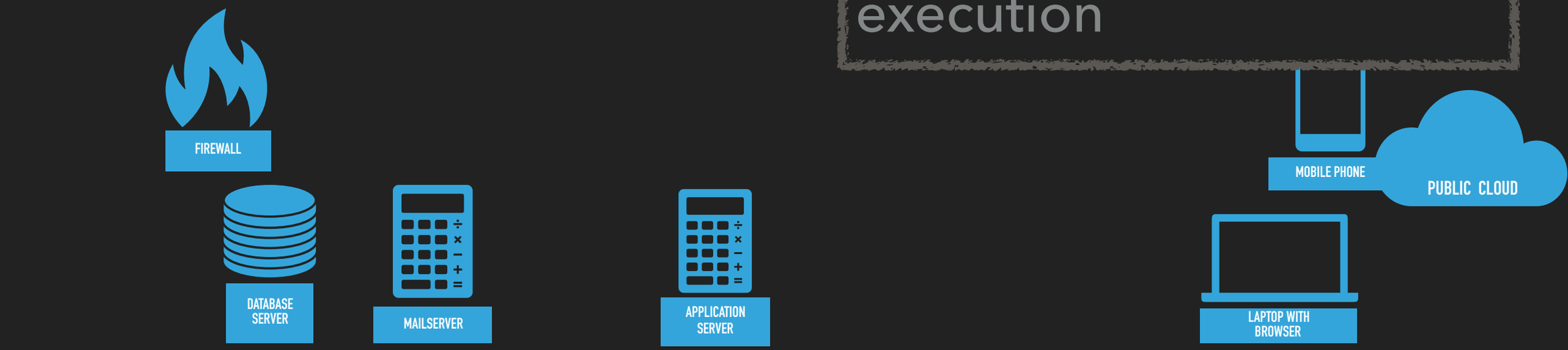
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER

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



LOW RISK

Exploit unlikely or running untrusted code already worst case

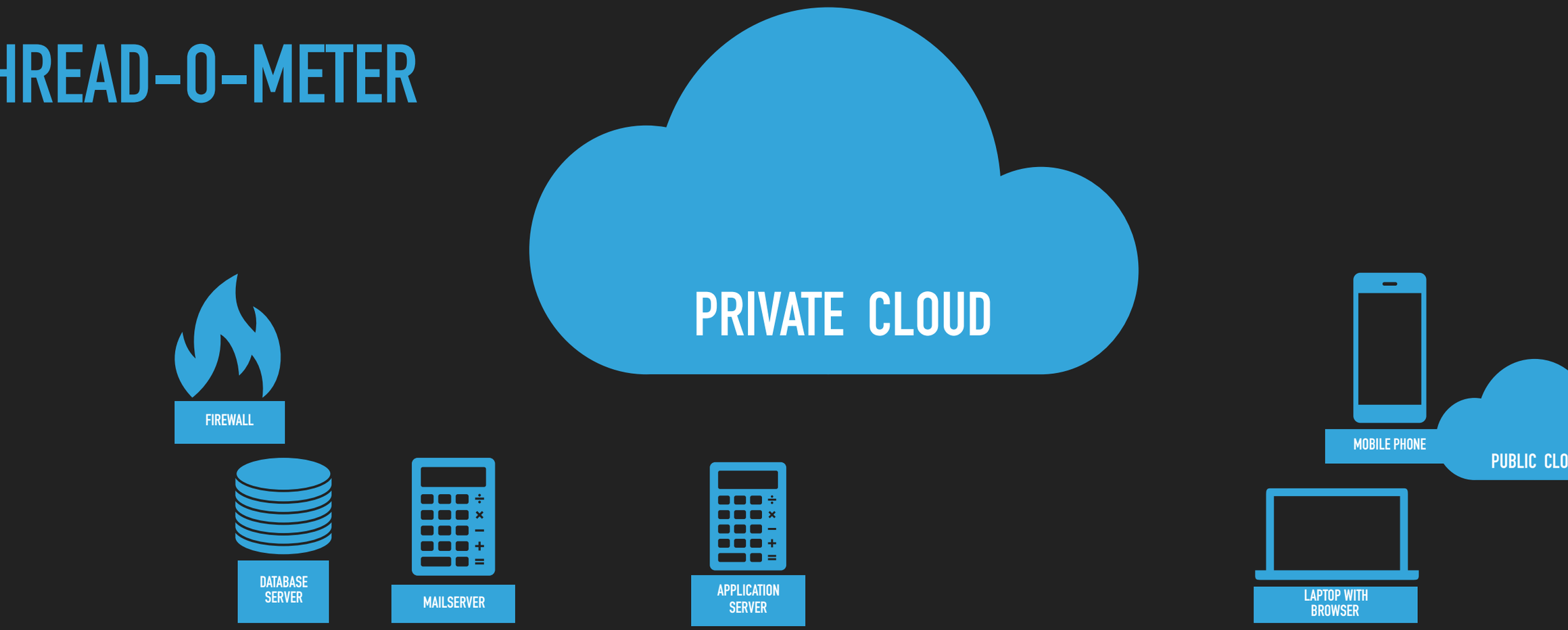
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-O-METER



LOW RISK

Exploit unlikely or running untrusted code already worst case

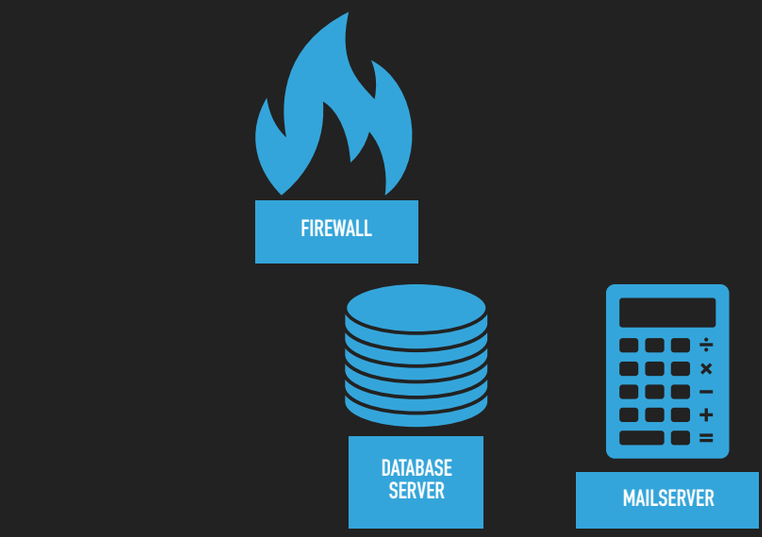
MEDIUM RISK

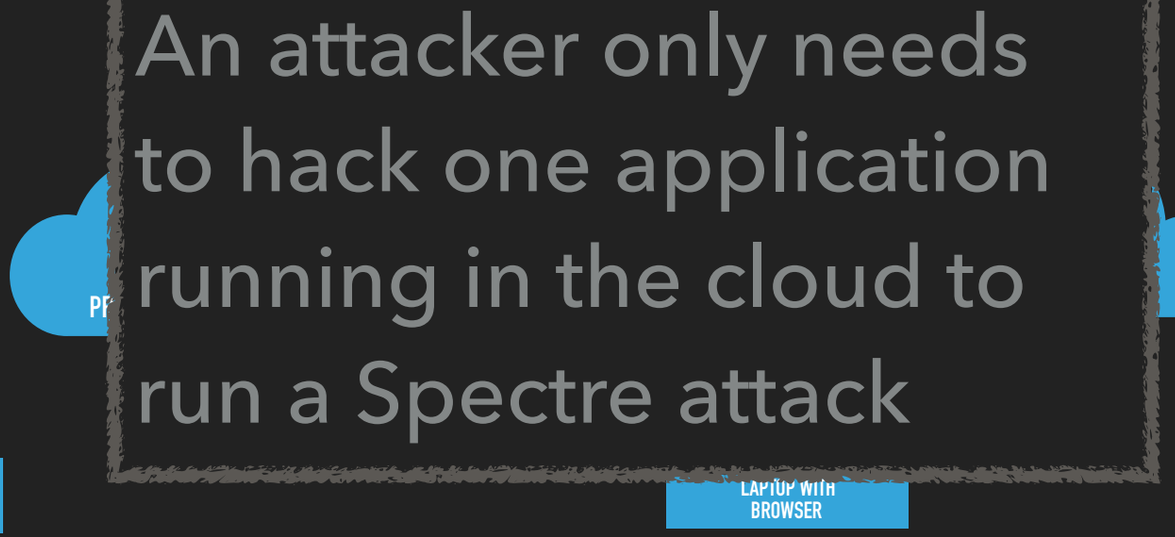
Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-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

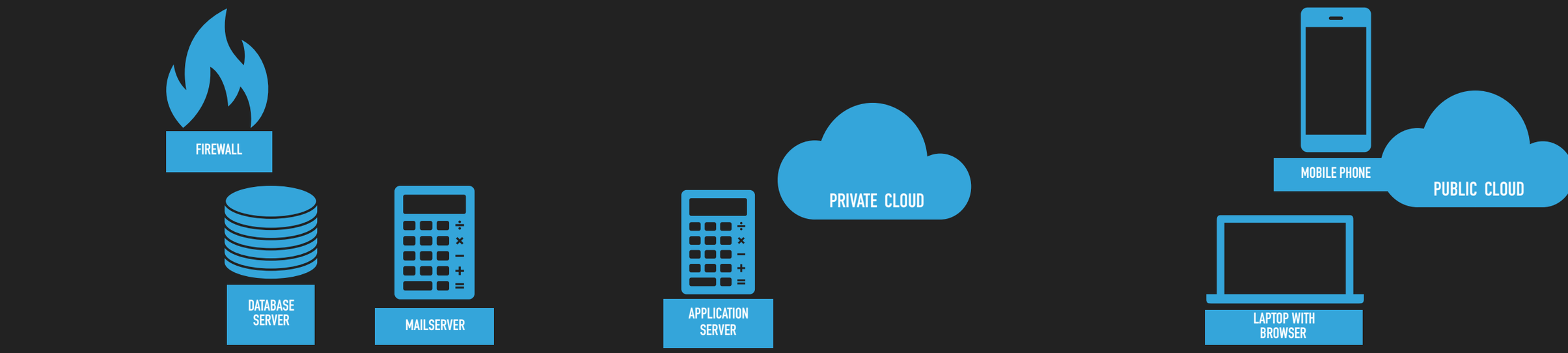
MEDIUM RISK

Exploit possible but needs another successful attack to run attackers code

HIGH RISK

Exploit possible and runs untrusted code “by design”

THREAD-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

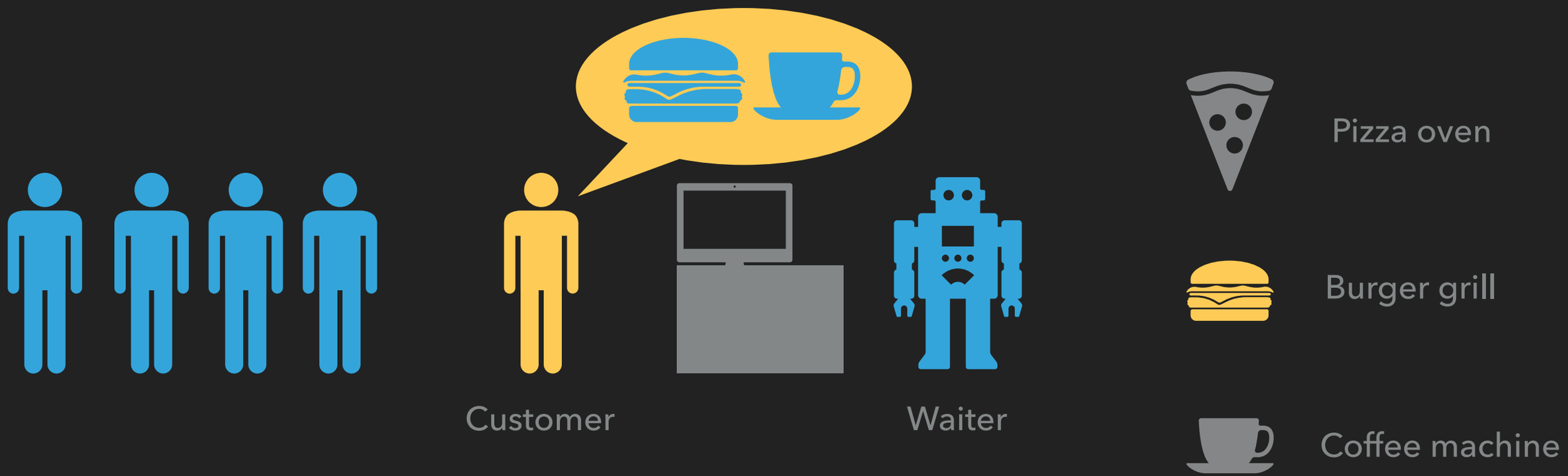
Exploit possible and runs untrusted code “by design”

ACCIDENT, MALICE, INCOMPETENCE?

WHY DID THIS HAPPEN?

CONFIDENTIAL BURGERS INC.

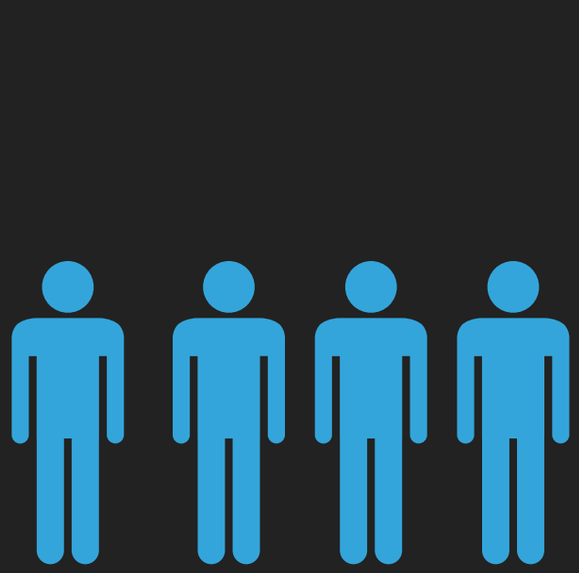
Confidential Burgers inc. sells burgers, pizza, and coffee.



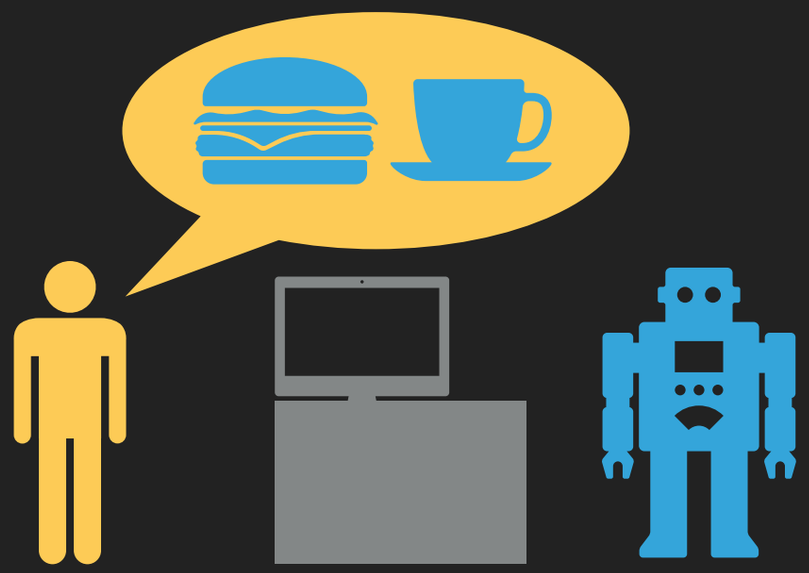
Grand Opening Today

CONFIDENTIAL BURGERS INC.

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



The waiter (CPU) will



Customer

Waiter

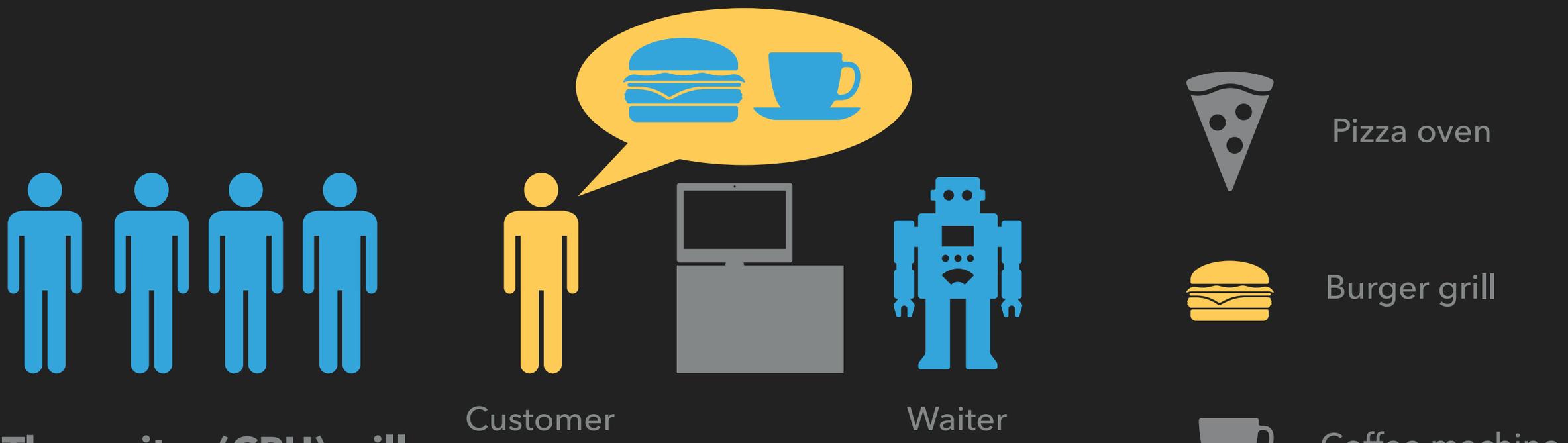
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC.

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

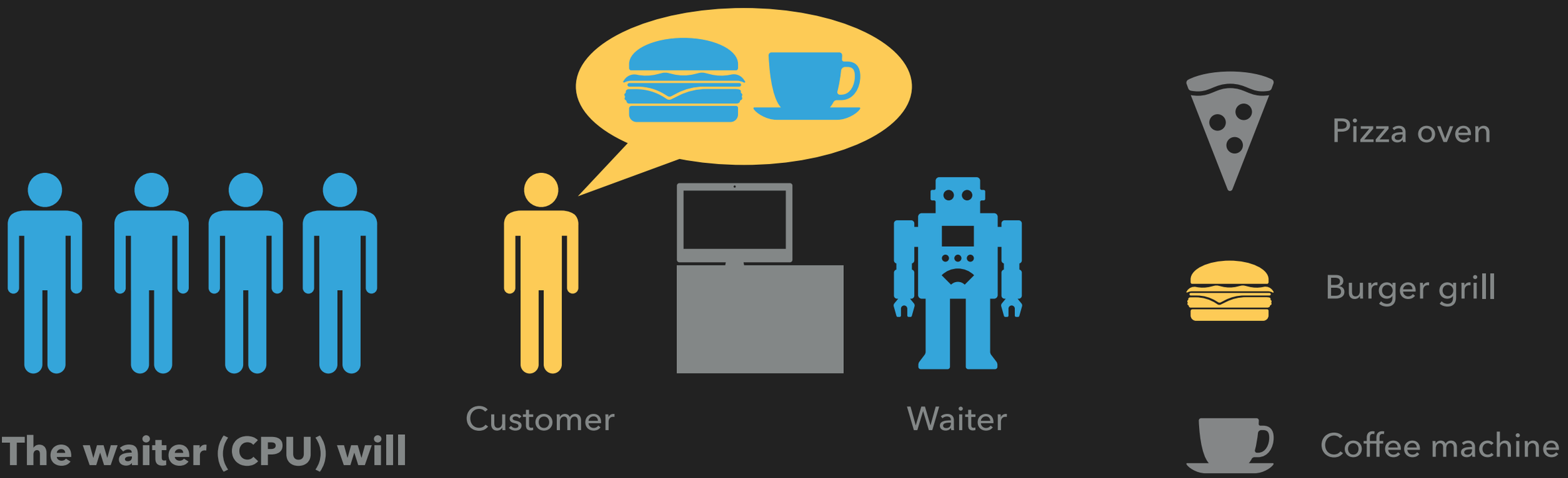


The waiter (CPU) will

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

CONFIDENTIAL BURGERS INC.

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

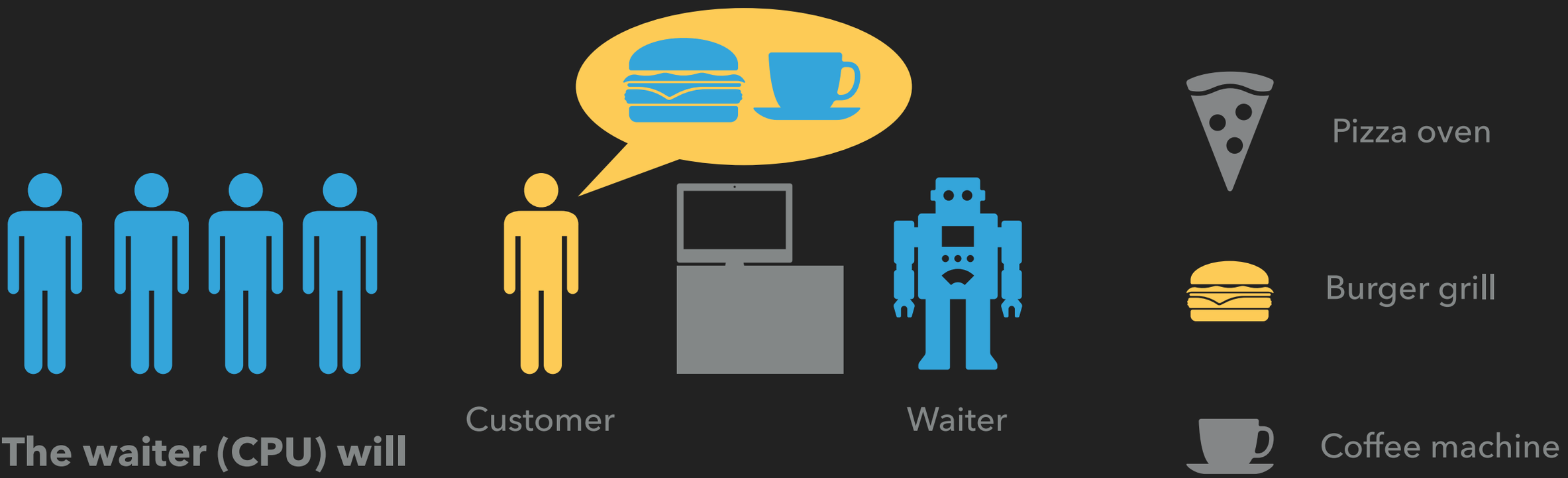


The waiter (CPU) will

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.

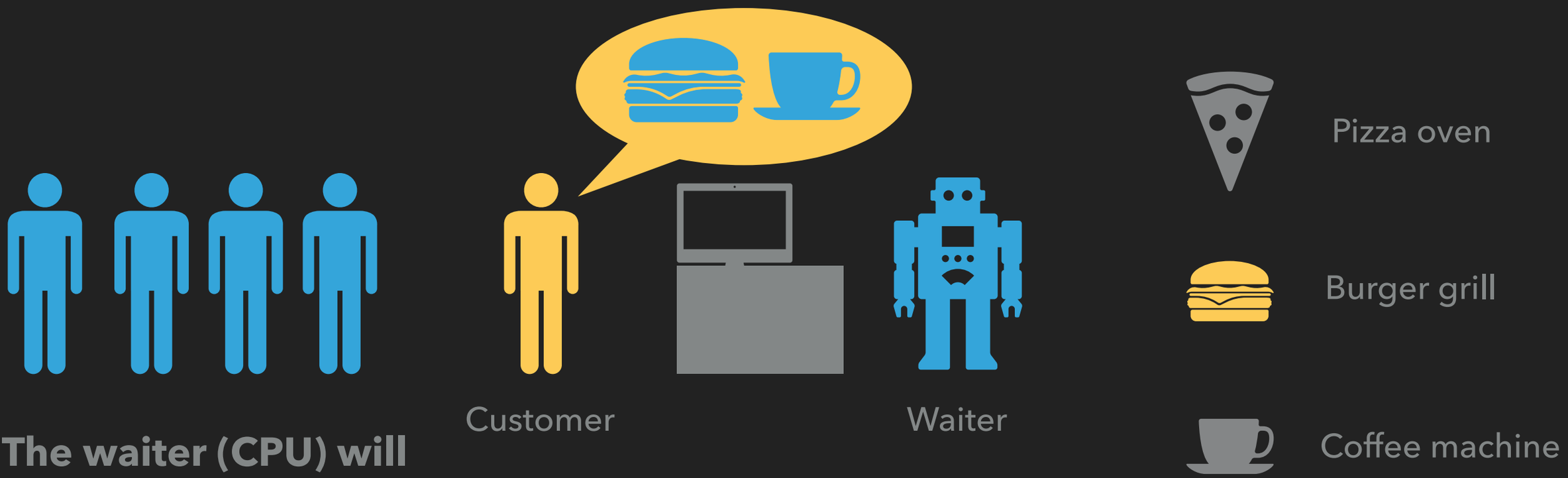


The waiter (CPU) will

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.

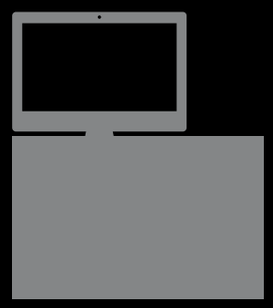


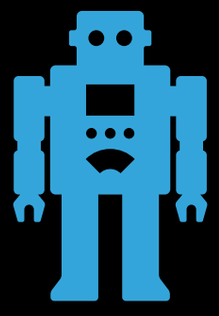
The waiter (CPU) will

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

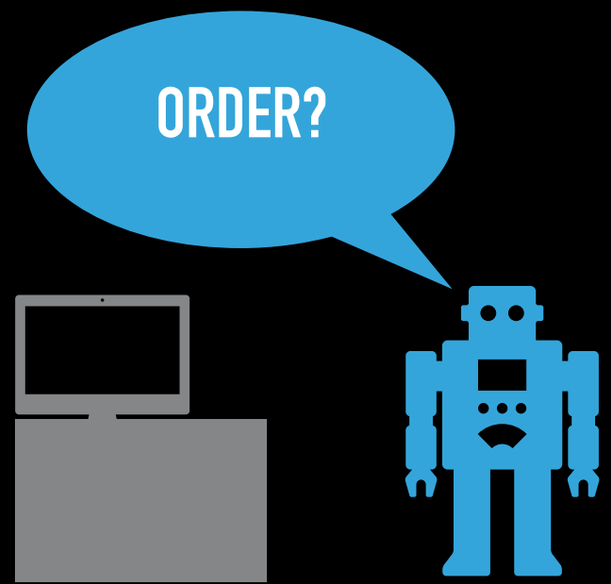
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION

Customer



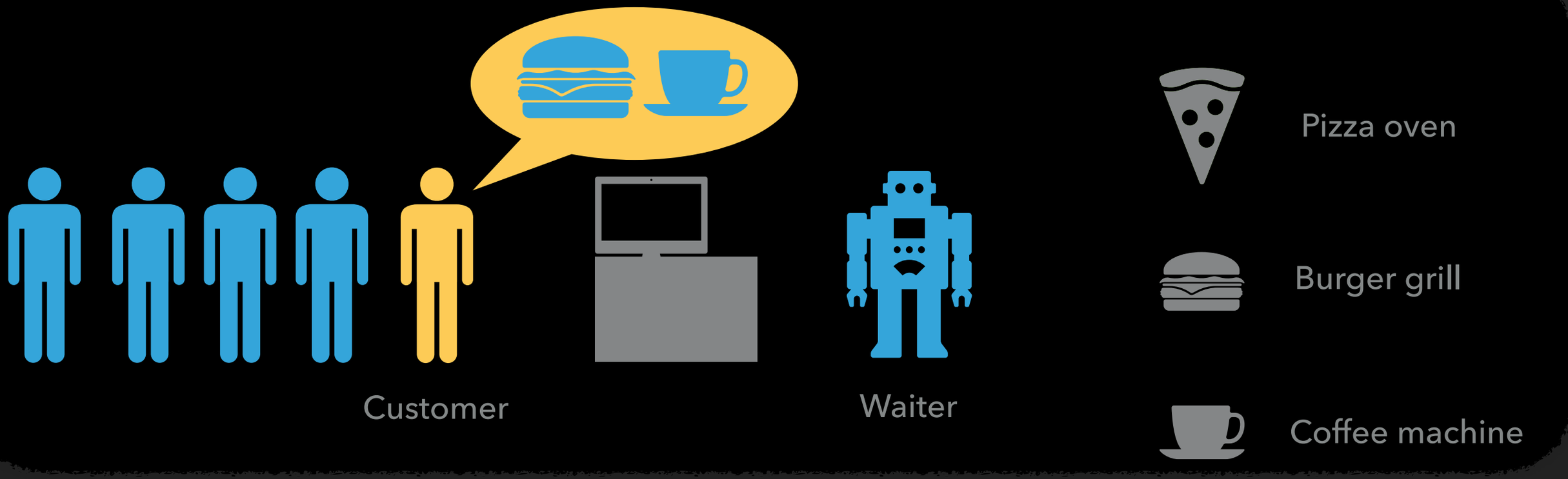
Waiter

Pizza oven

Burger grill

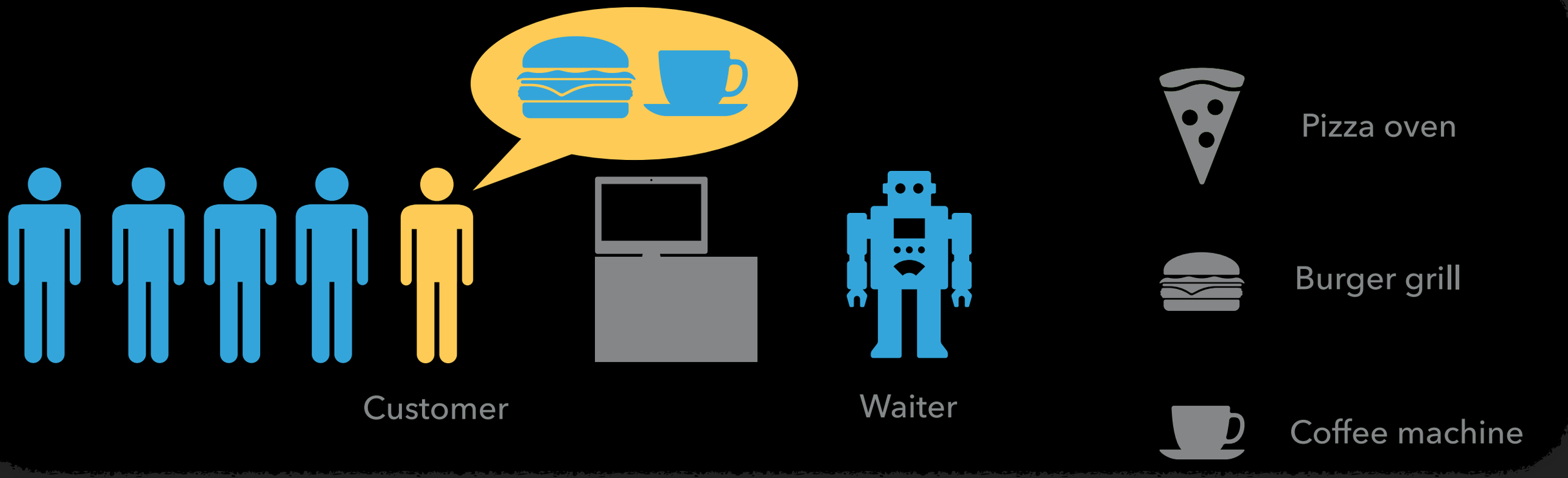
Coffee machine

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



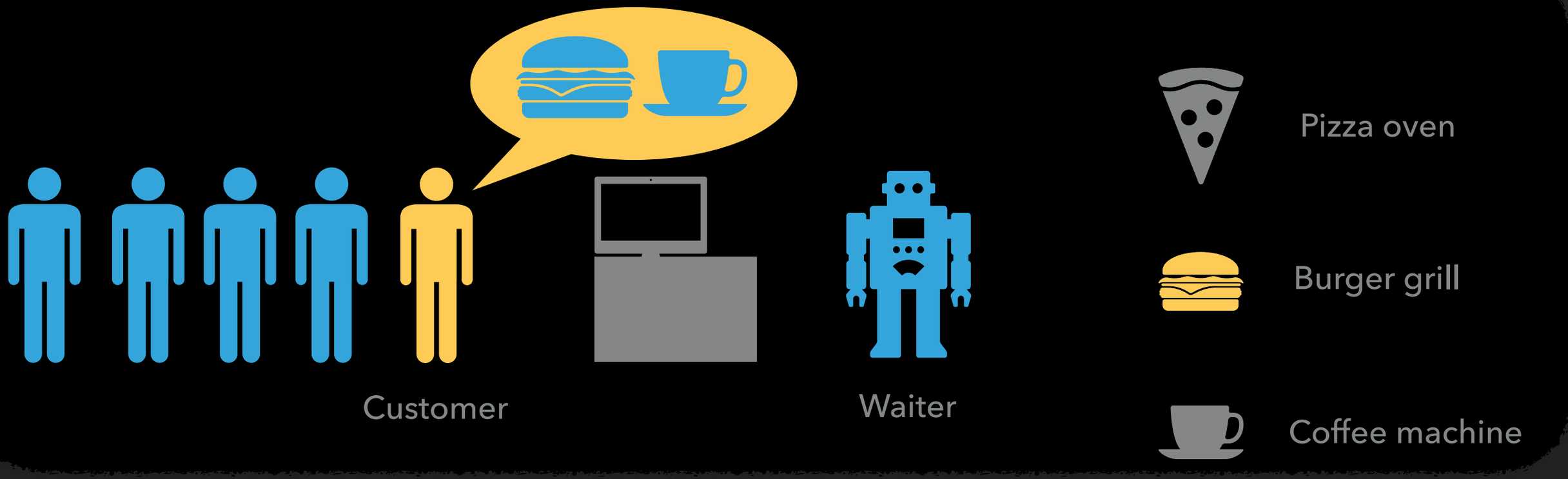
- Decode instruction into μOPs

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



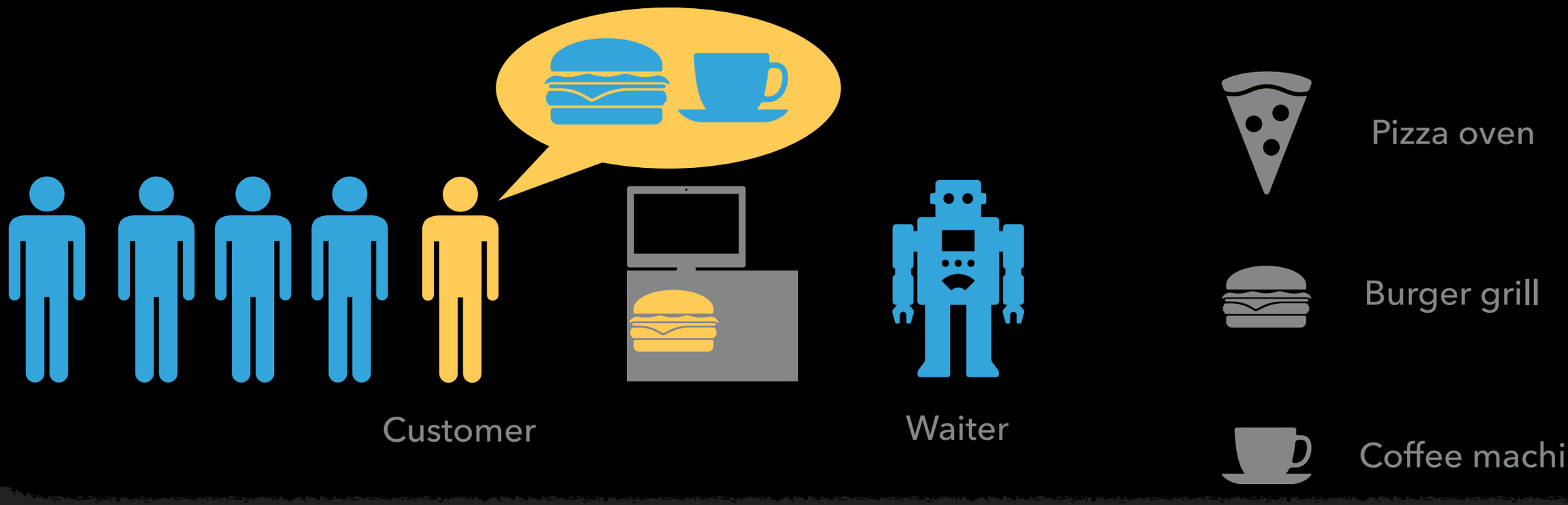
- Decode instruction into µOPs
- Schedule µOPs

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



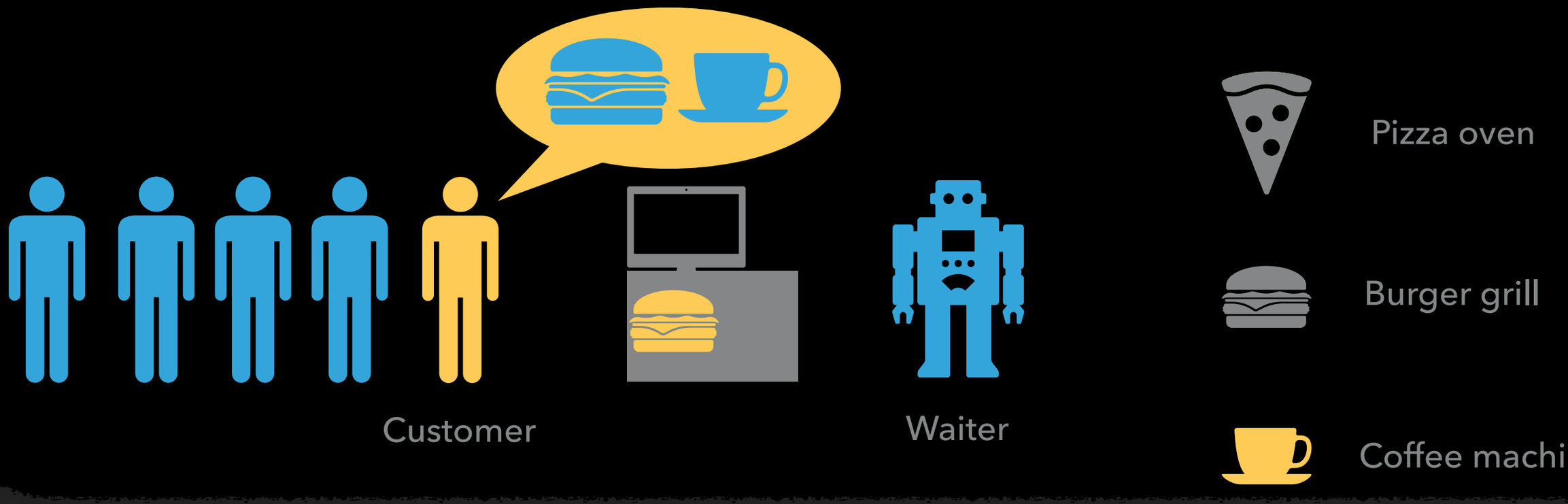
- Decode instruction into µOPs
- Schedule µOPs
  - run 1st µOP (grill the burger)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



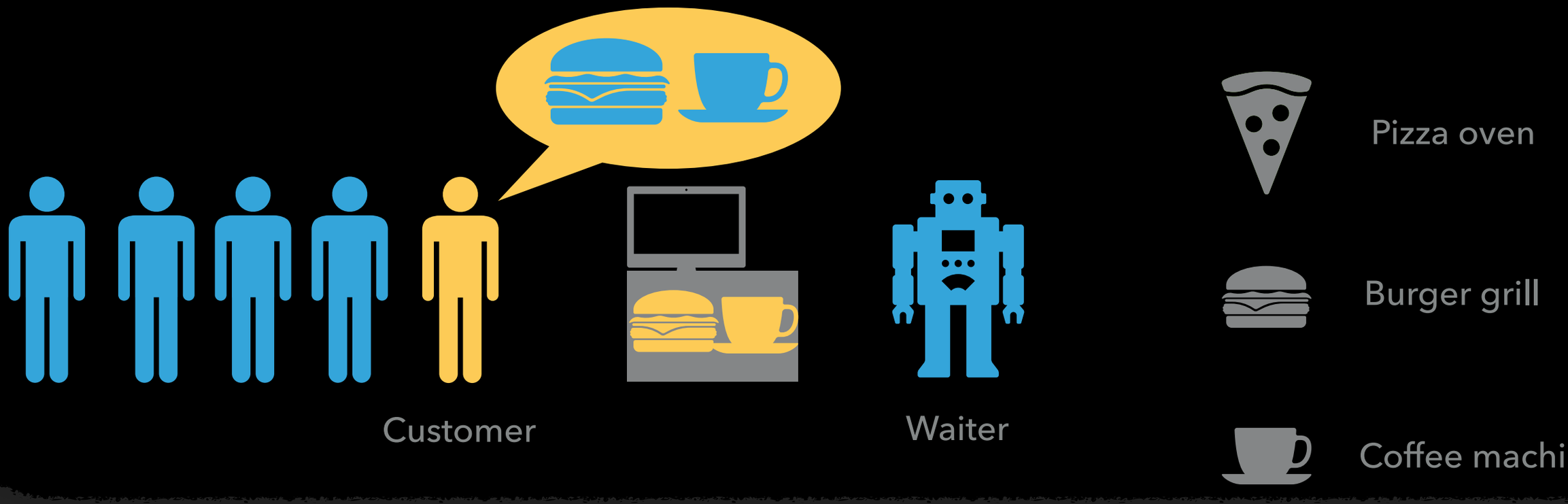
- Decode instruction into µOPs
- Schedule µOPs
  - run 1st µOP (grill the burger)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



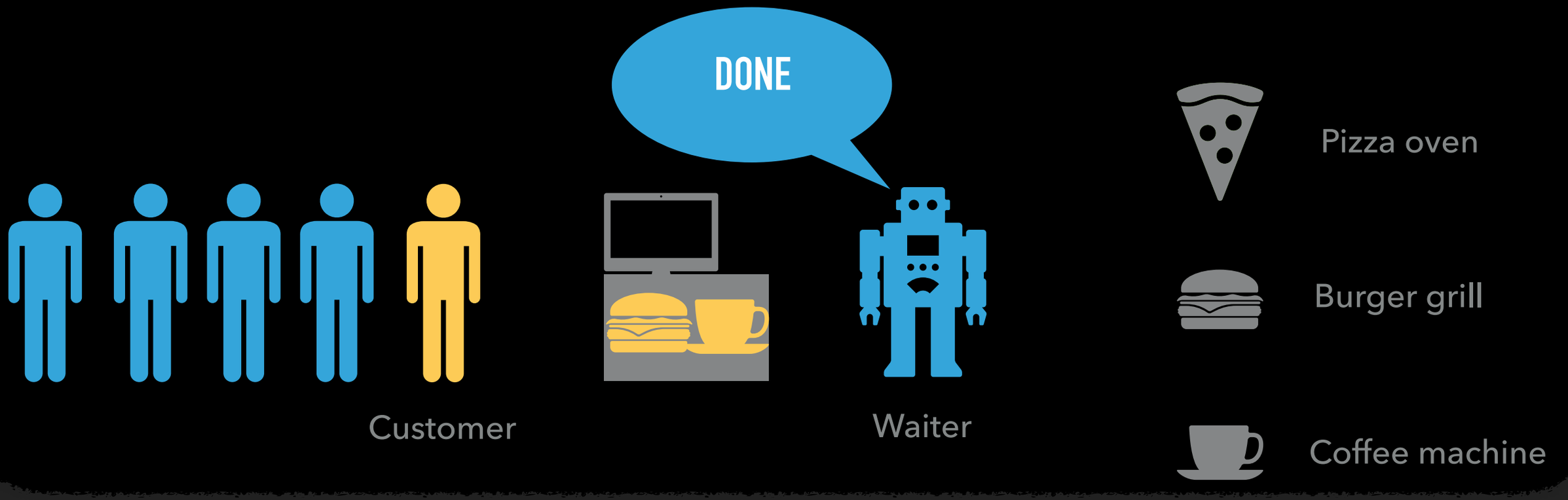
- Decode instruction into µOPs
- Schedule µOPs
  - run 1st µOP (grill the burger)
  - run 2nd µOP (brew coffee, serial execution)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



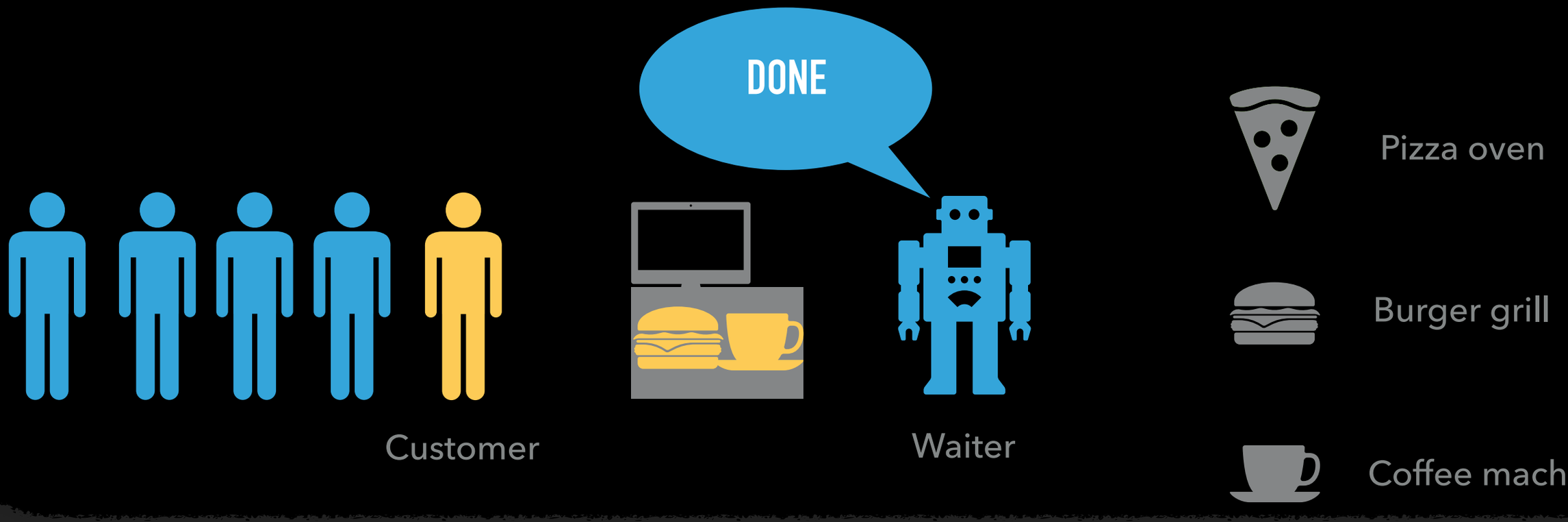
- Decode instruction into µOPs
- Schedule µOPs
  - run 1st µOP (grill the burger)
  - run 2nd µOP (brew coffee, serial execution)

CONFIDENTIAL BURGERS INC. : SERIAL, IN ORDER EXECUTION



- Decode instruction into µOPs
- 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



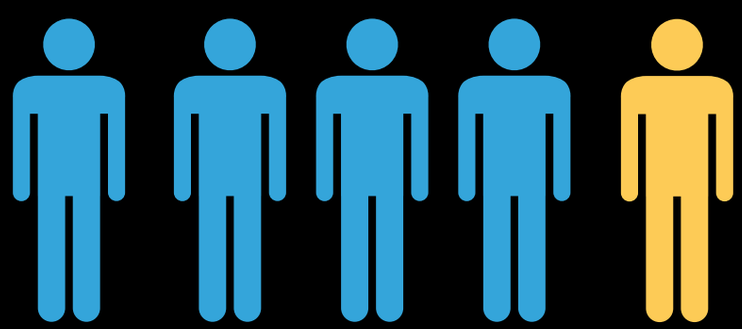
- One customer[1] after another (**in order**)
- Each part of the order [2] executed **serially**

I.e. first the burger, then the coffee

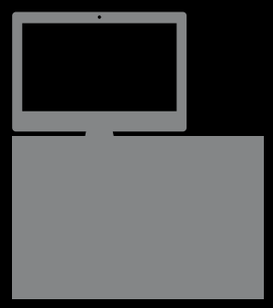
- PRO: Easy to implement and understand
- CON: Slow because resources[3] not utilised fully

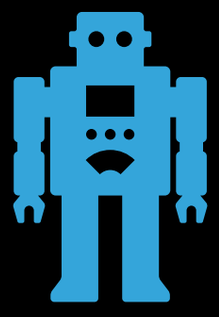
[1] customer == CPU instruction [2] part == µOP - micro operation [3] oven, grill, coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



Customer





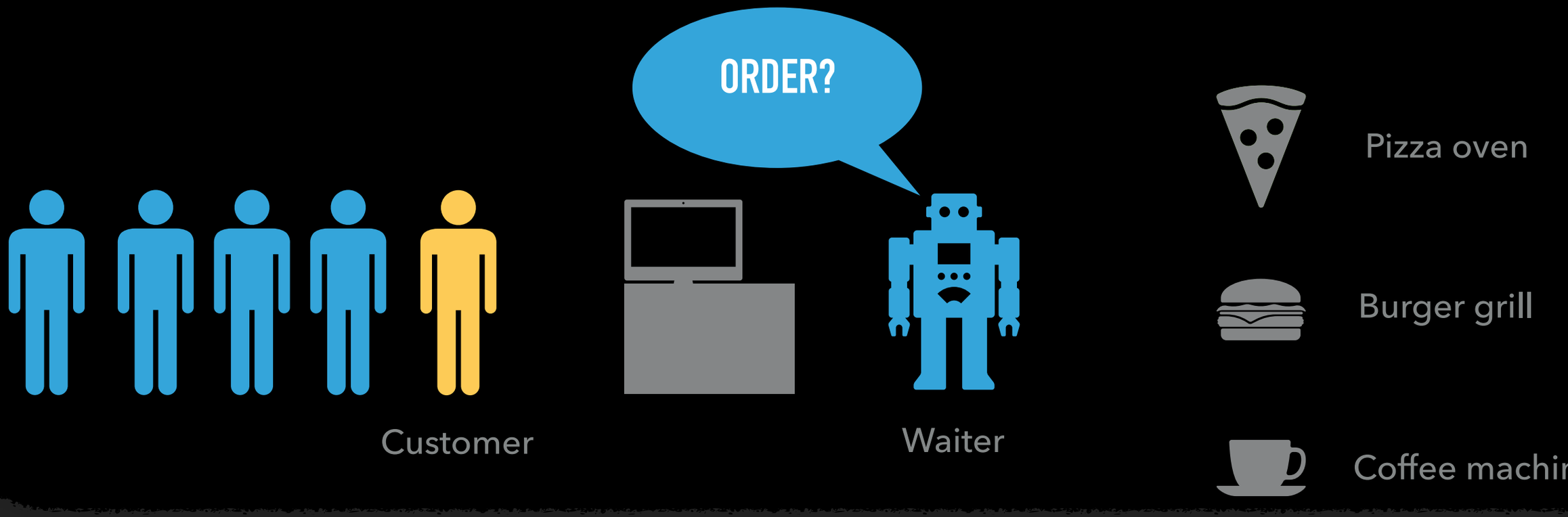
Waiter

Pizza oven

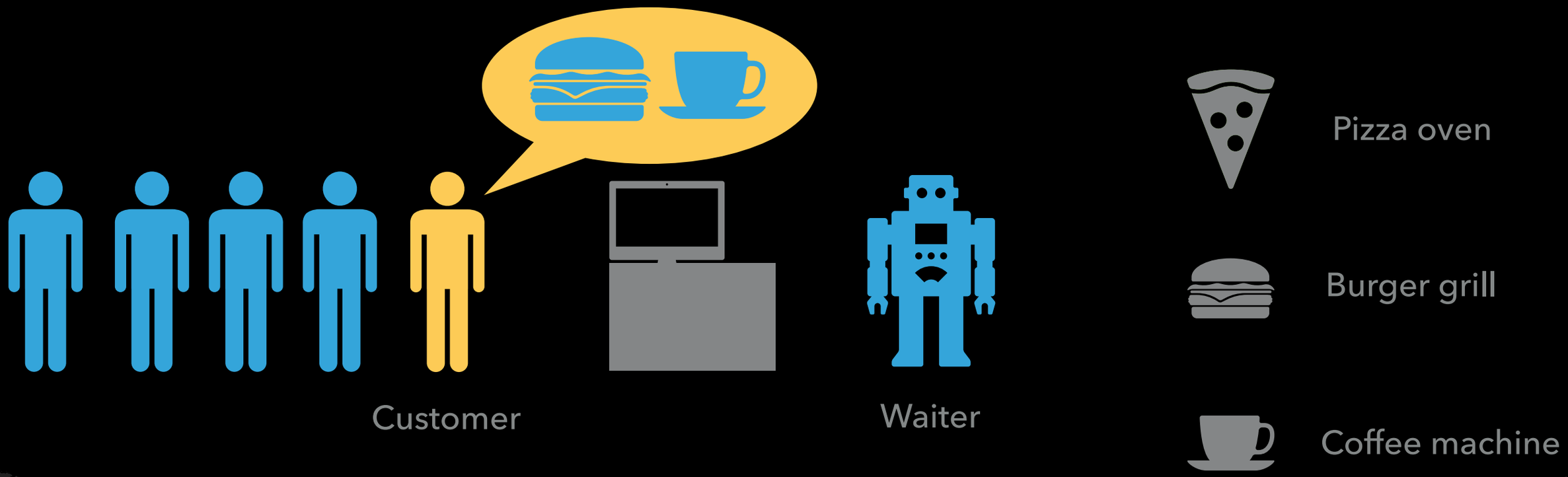
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION

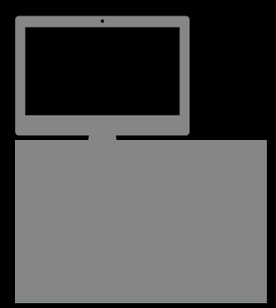


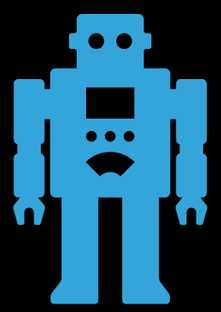
CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



- Decode instruction into μOPs

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION





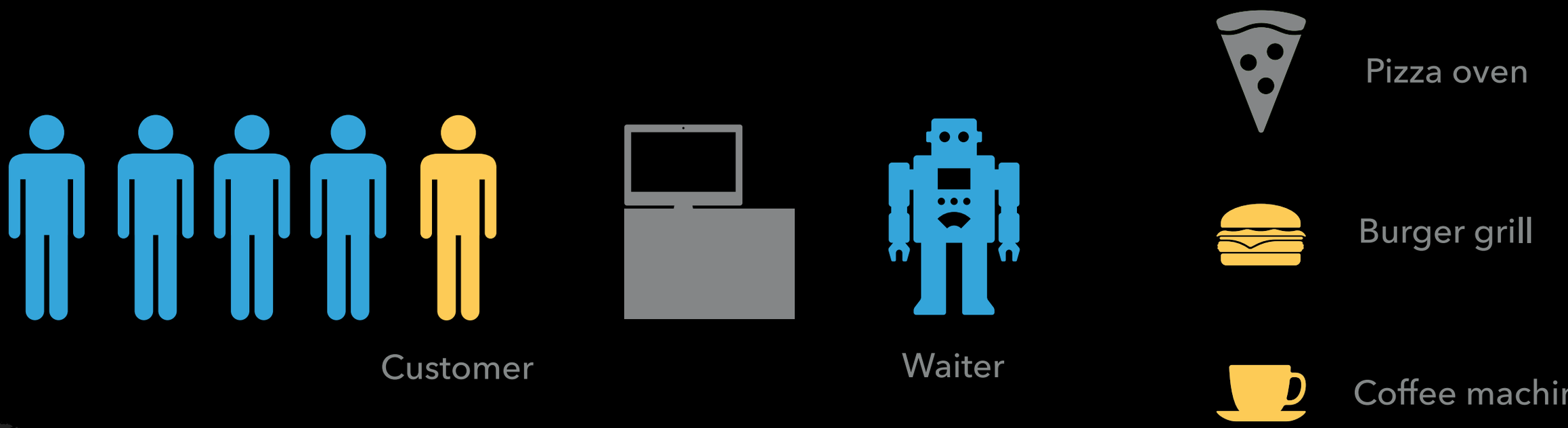
Pizza oven

Burger grill

Coffee machine

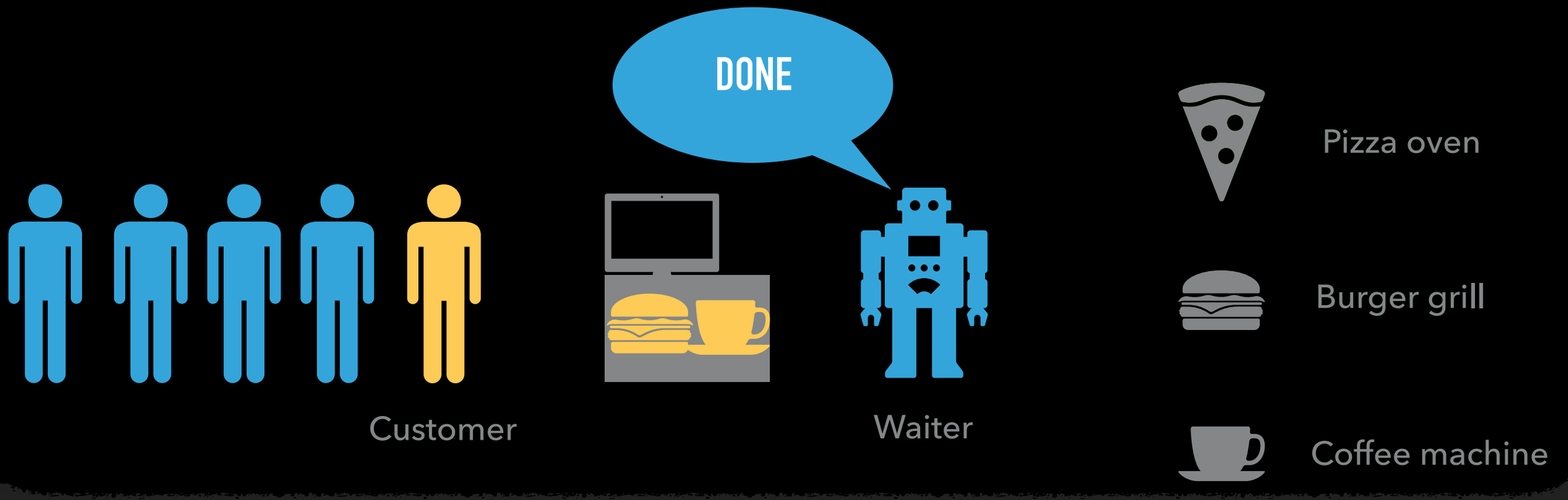
- Decode instruction into µOPs
- Schedule µOPs

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



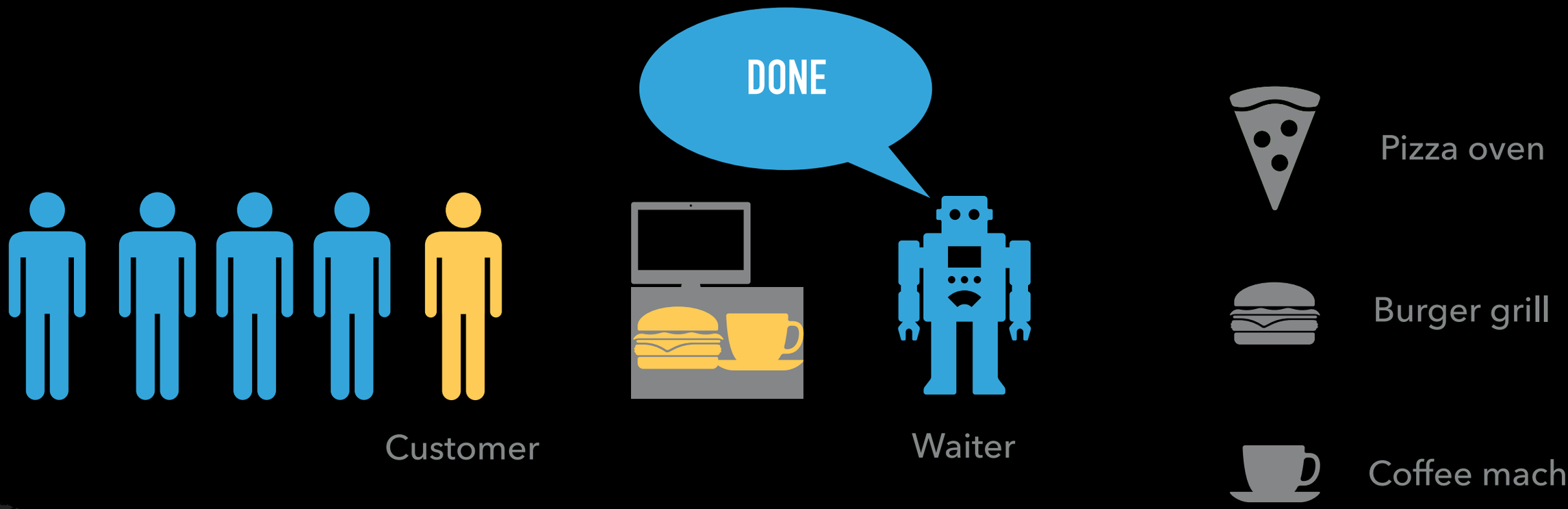
- Decode instruction into μOPs
- Schedule μOPs
  - run 1st μOP and 2nd μOP (parallel execution)

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION



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

CONFIDENTIAL BURGERS INC. : PARALLEL, IN ORDER EXECUTION

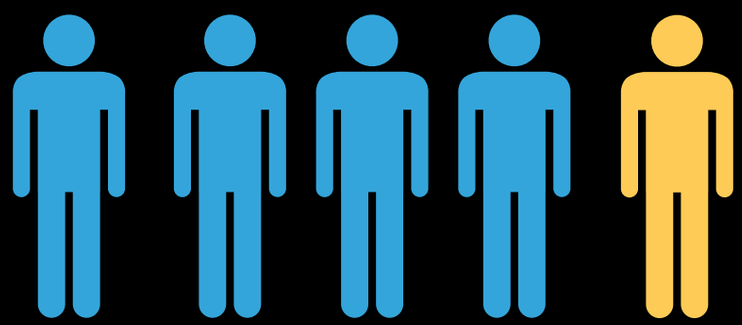


- One customer[1] after another (**in order**)
- Each part of the order [2] executed **in parallel**

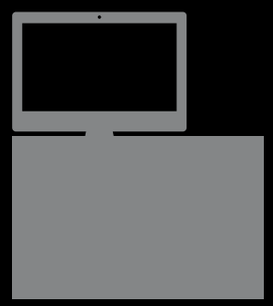
I.e. burger and coffee prepared at the same time

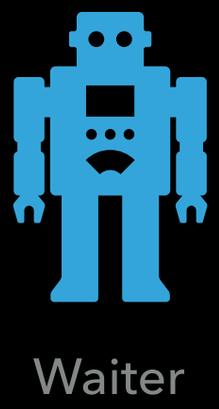
- PRO: **Faster bc. of better resource utilisation.**
- CON: Still not perfect, more complex

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer





Waiter

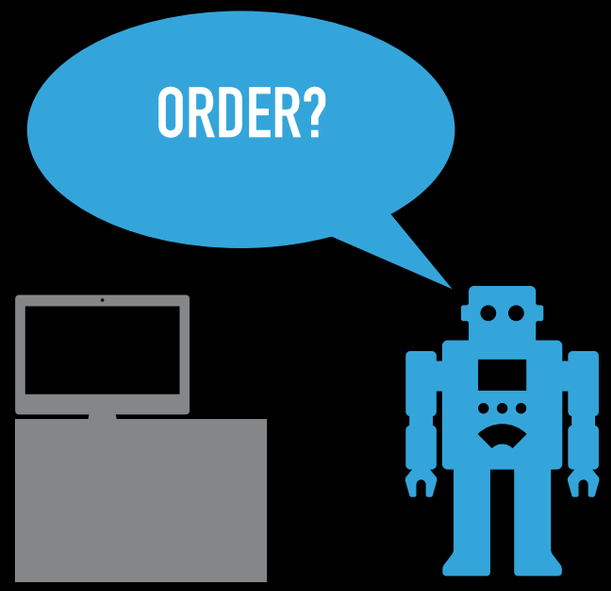
Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

Customer



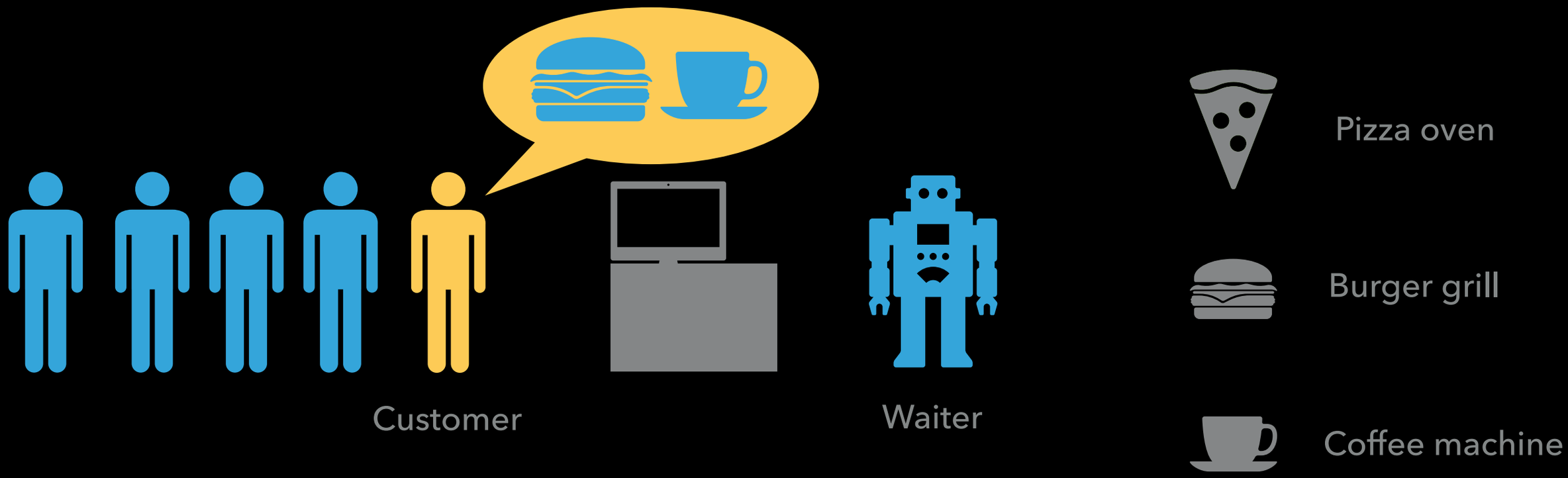
Waiter

Pizza oven

Burger grill

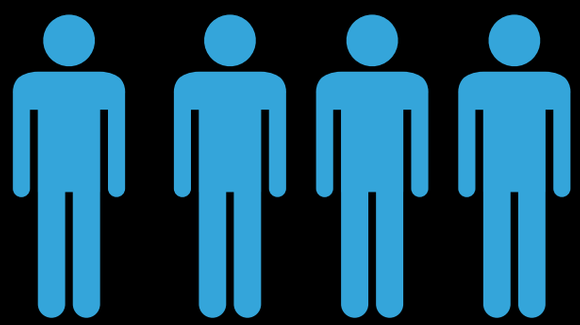
Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

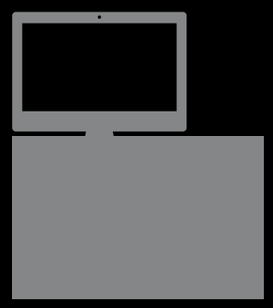


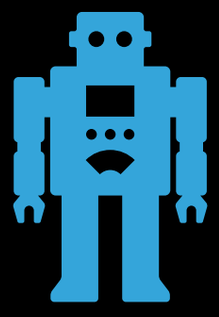
CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer





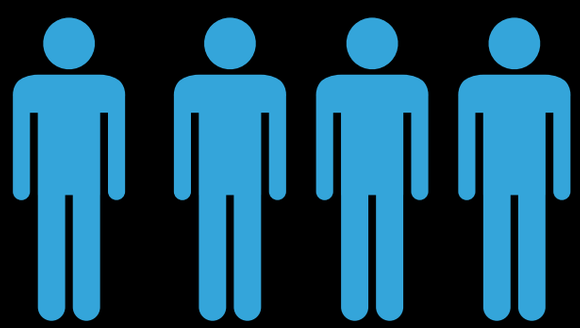
Waiter

Pizza oven

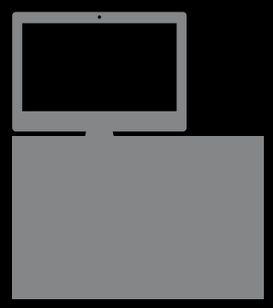
Burger grill

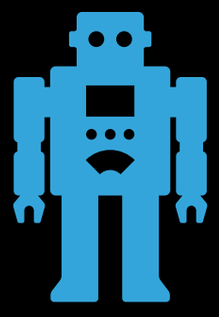
Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION



Customer





Waiter

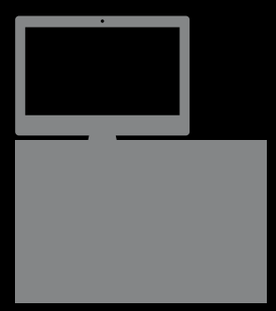
Pizza oven

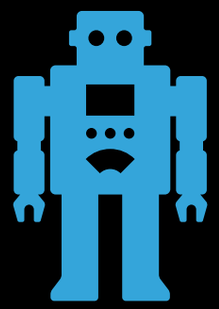
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

Customer





Waiter

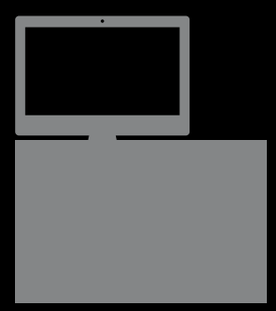
Pizza oven

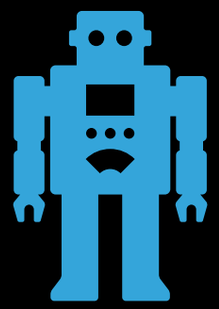
Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

Customer





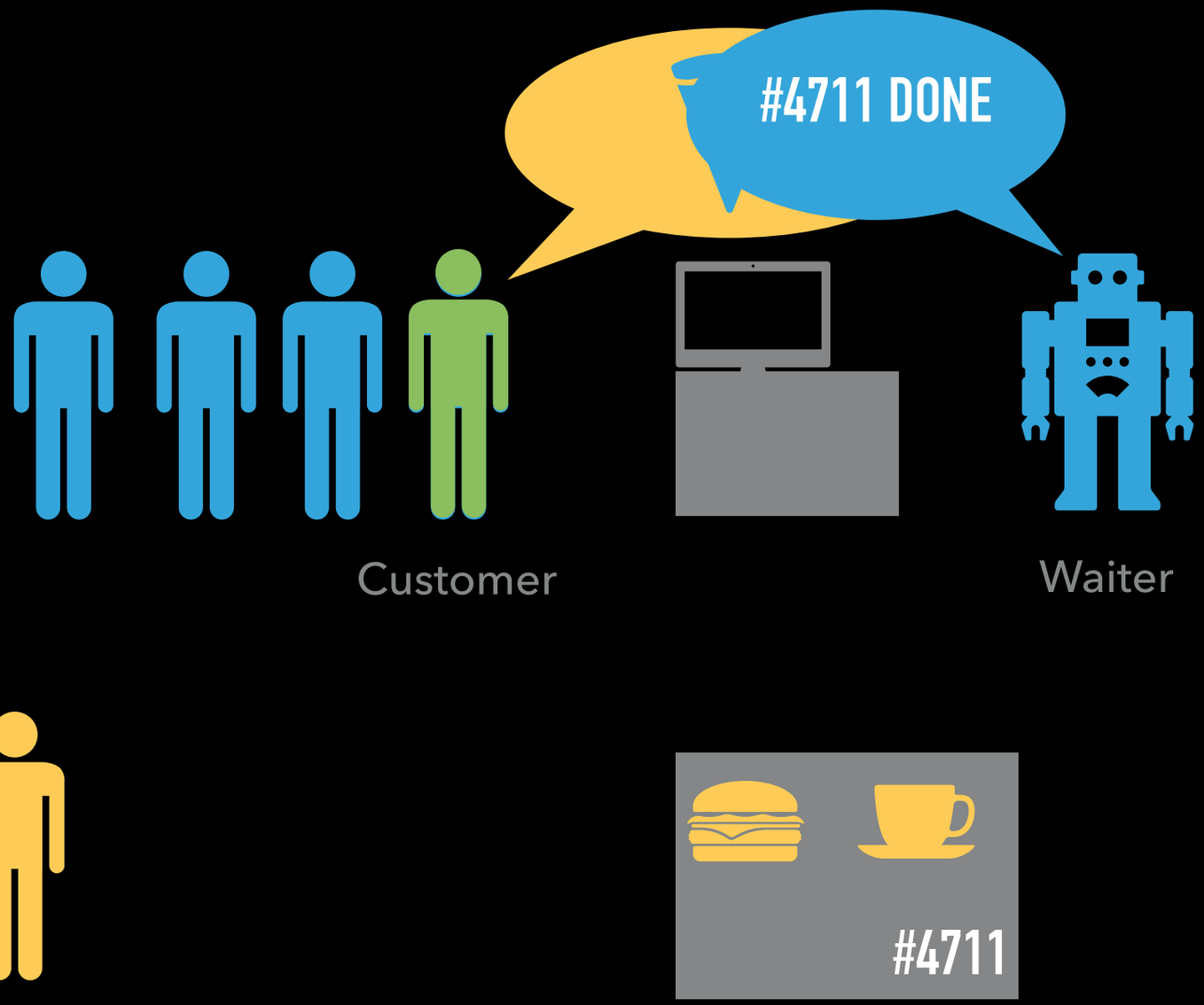
Waiter

Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION

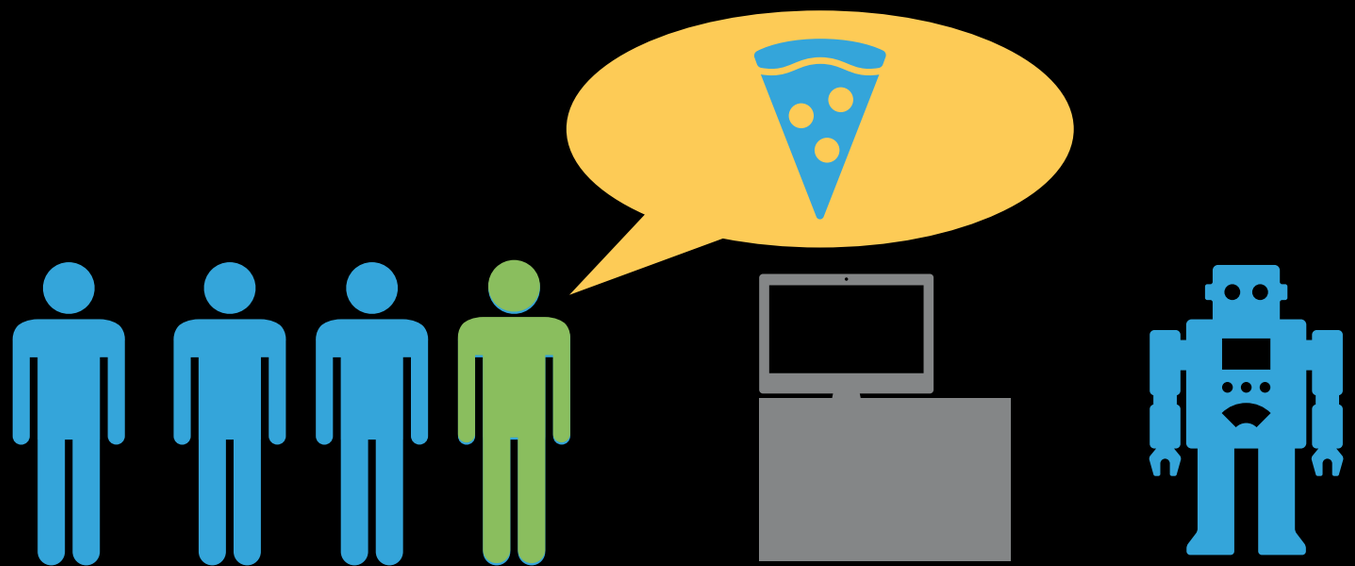


Pizza oven

Burger grill

Coffee machine

CONFIDENTIAL BURGERS INC. : PARALLEL, OUT OF ORDER EXECUTION




Pizza oven


Burger grill


Coffee machine

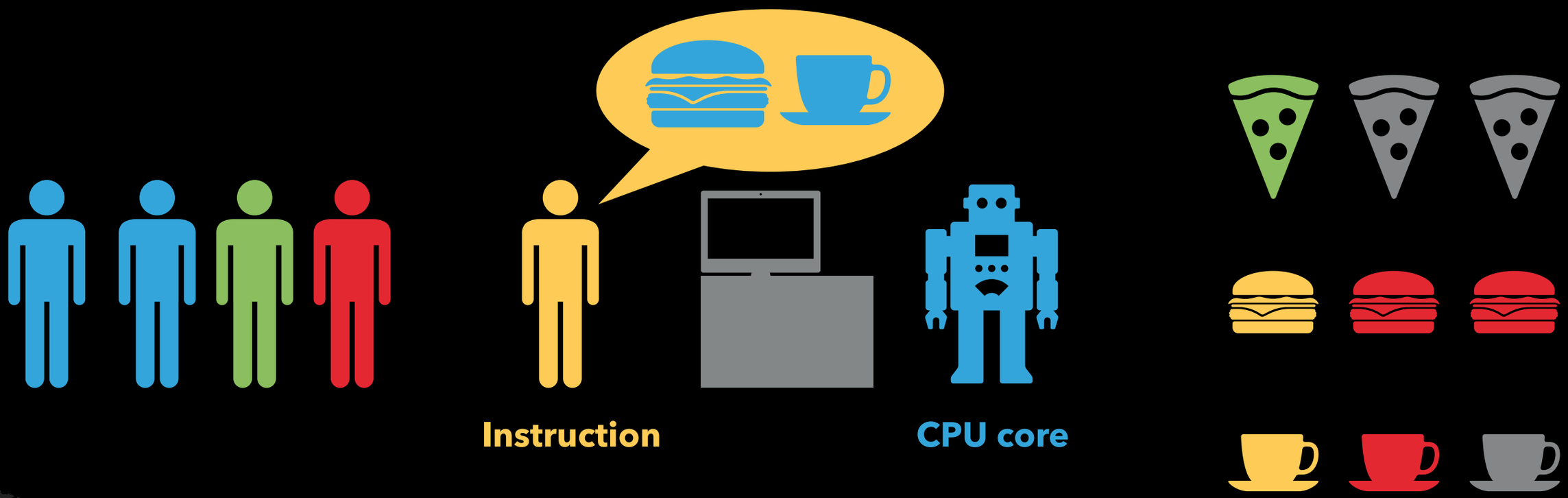
- Multiple customers' orders **executed in parallel**[1] and **delivered** (retired) **in order**

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*

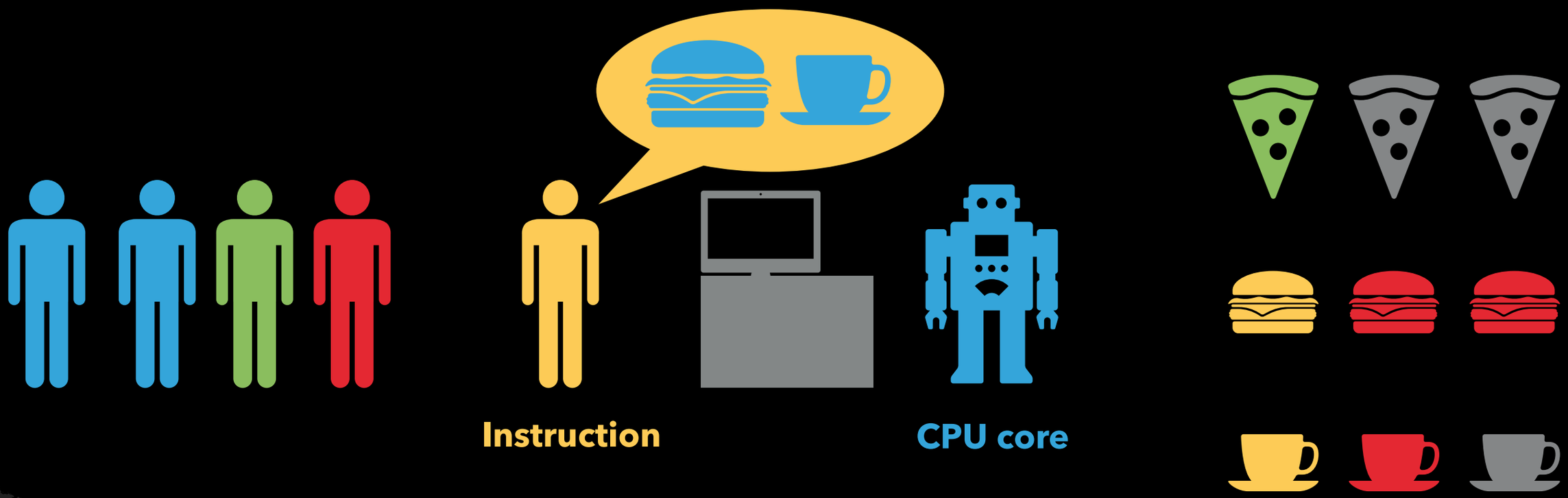
CONFIDENTIAL BURGERS INC.



Adding more resources increase parallelism & throughput

This is all on one CPU core

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

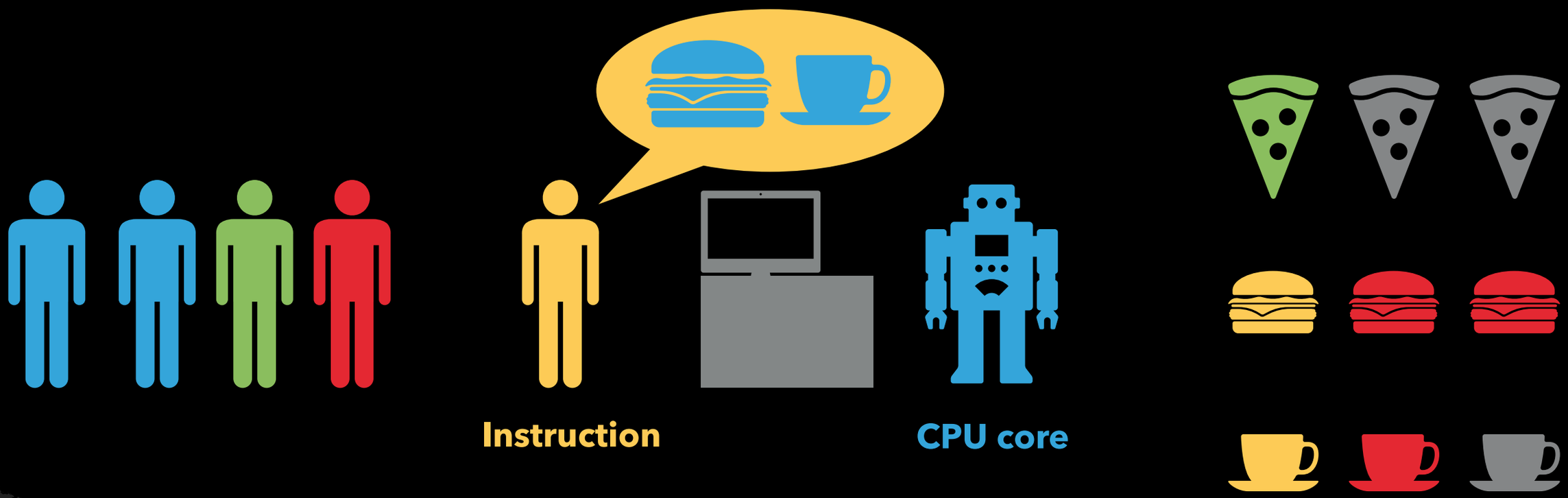


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

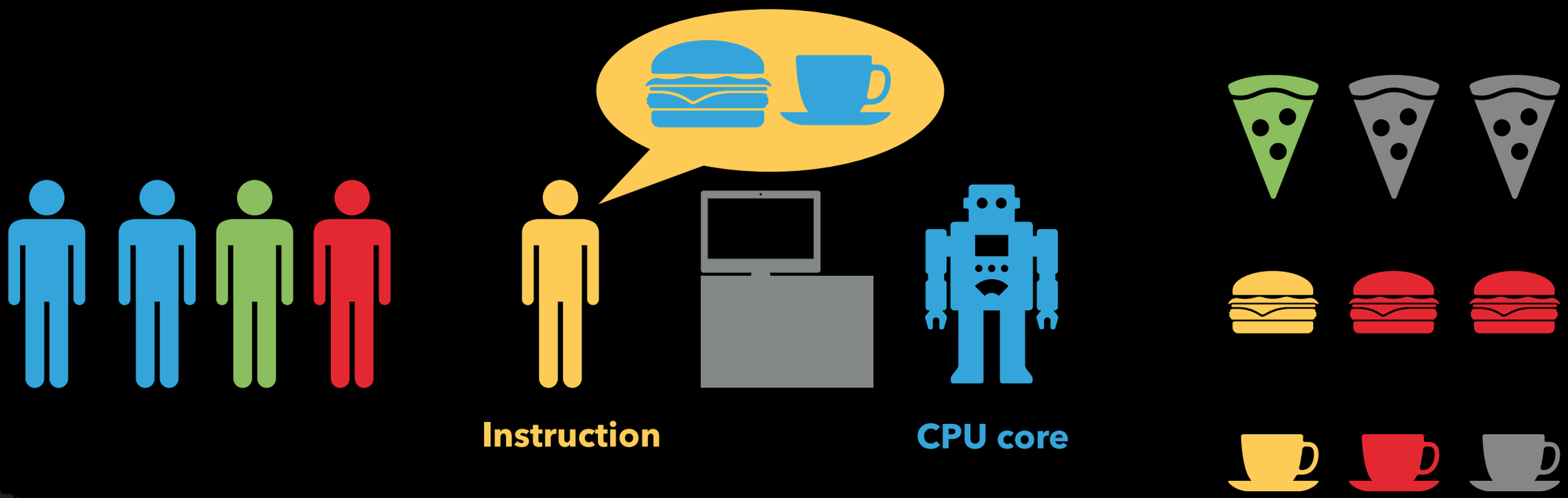


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

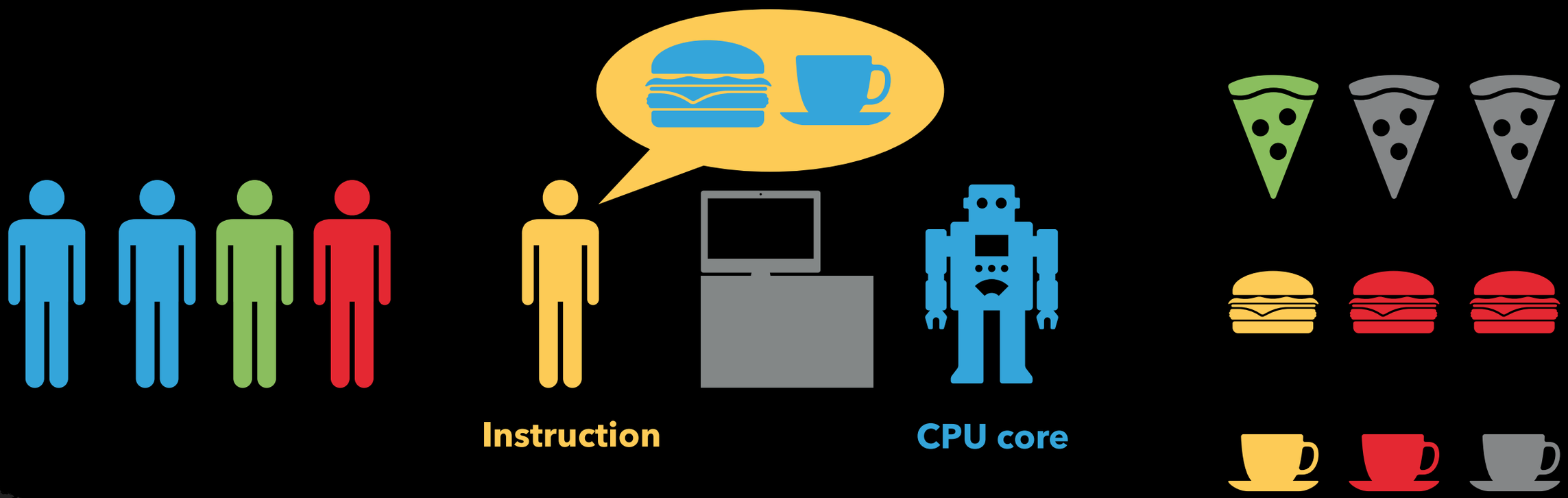


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

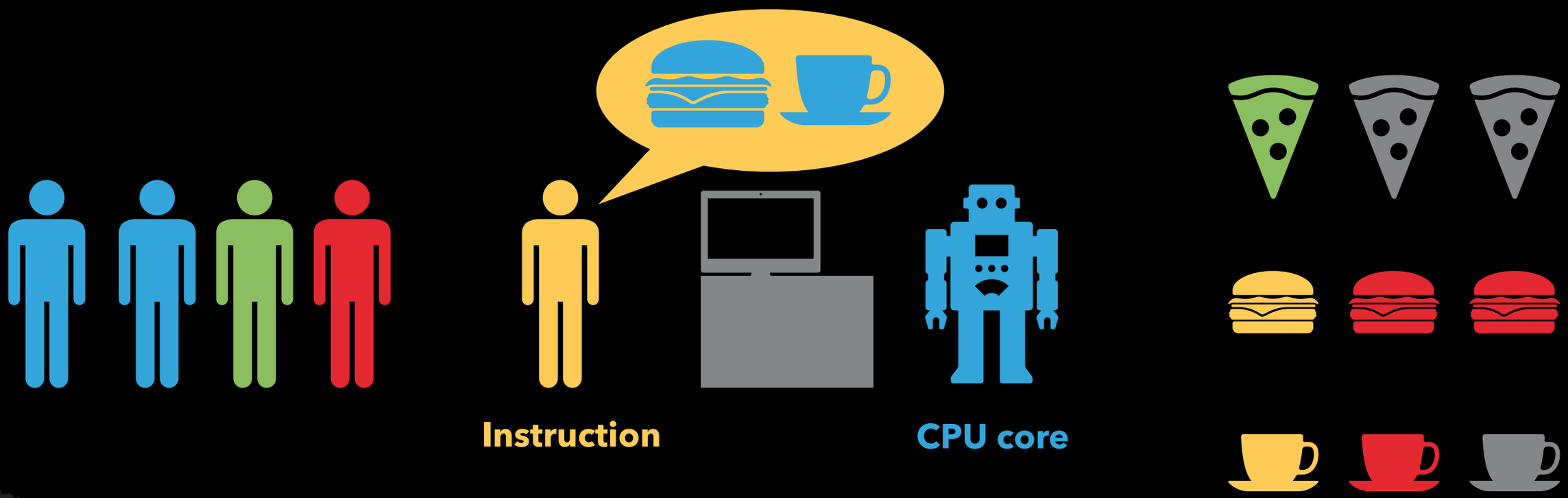


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

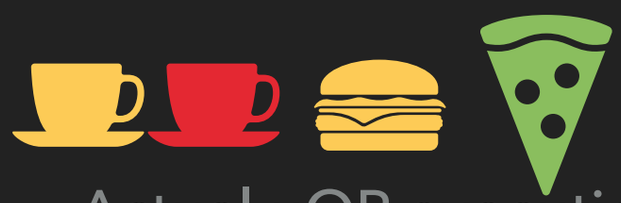
Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



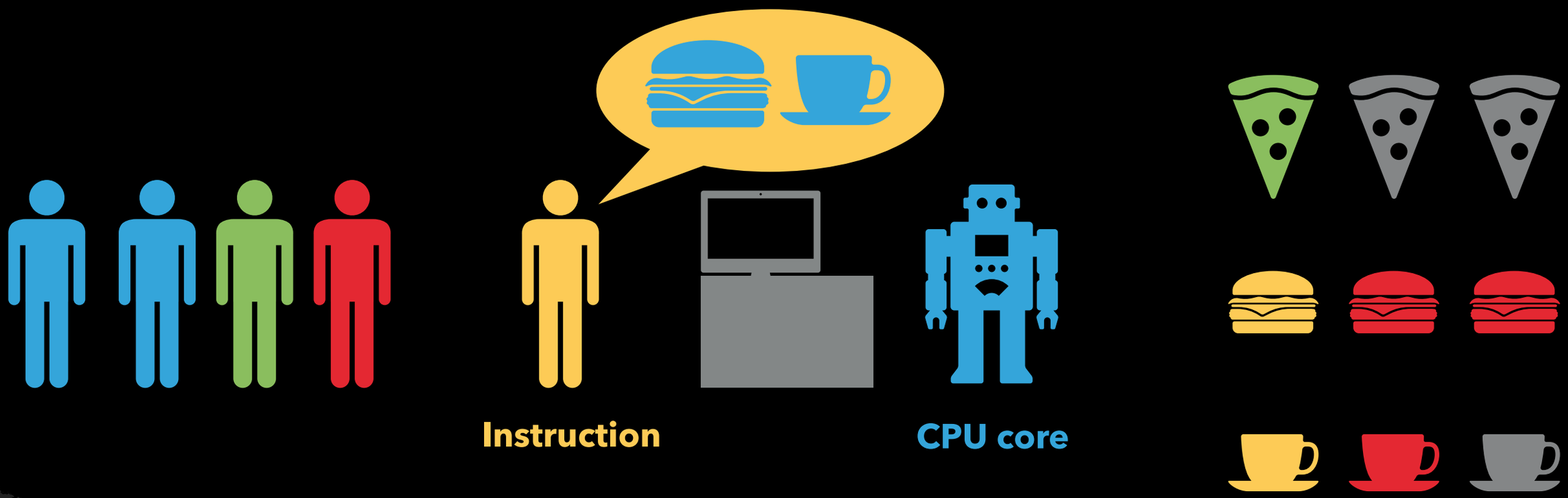
The green instruction will finish before the red instruction

The CPU ensures that red is seen before green



Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

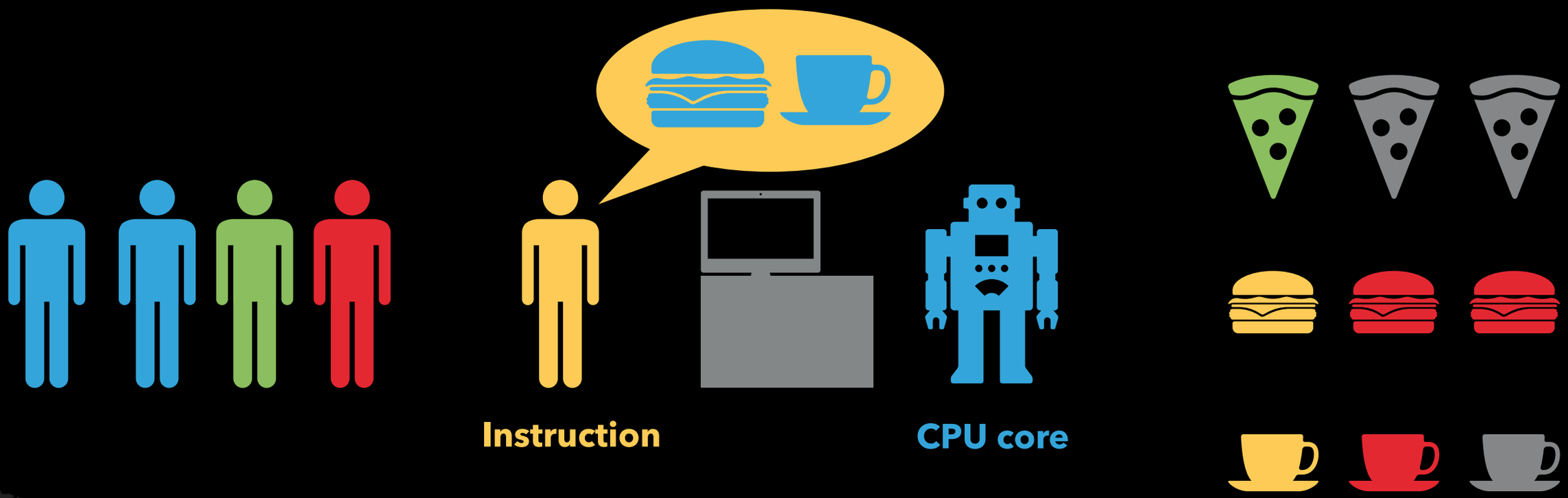


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT

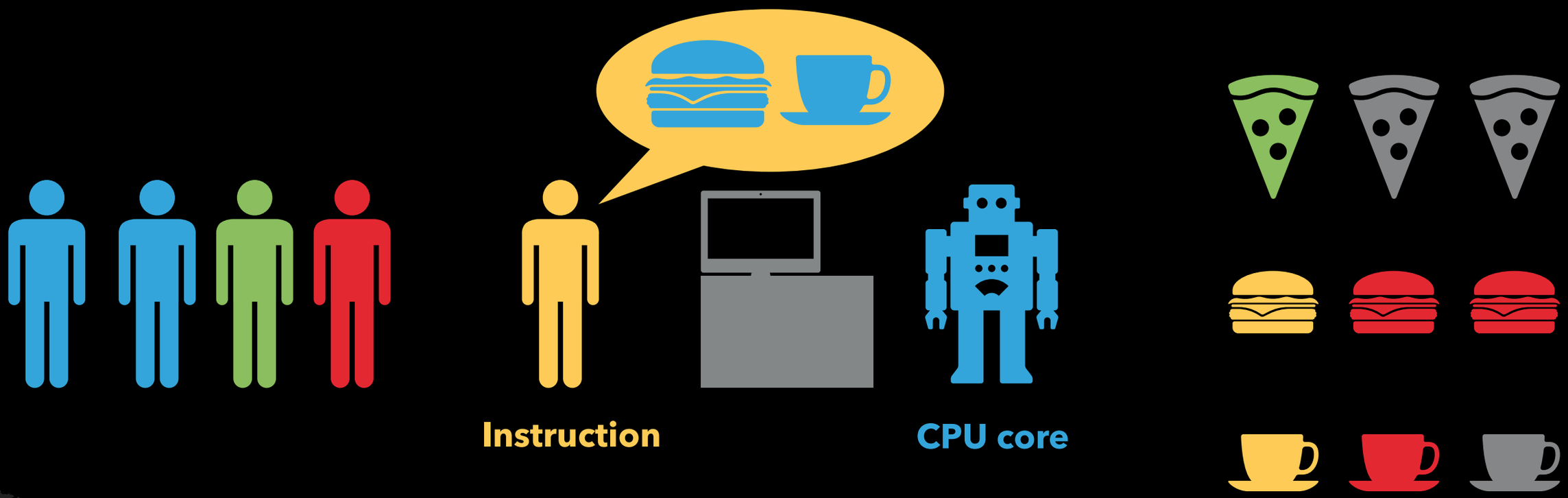


The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



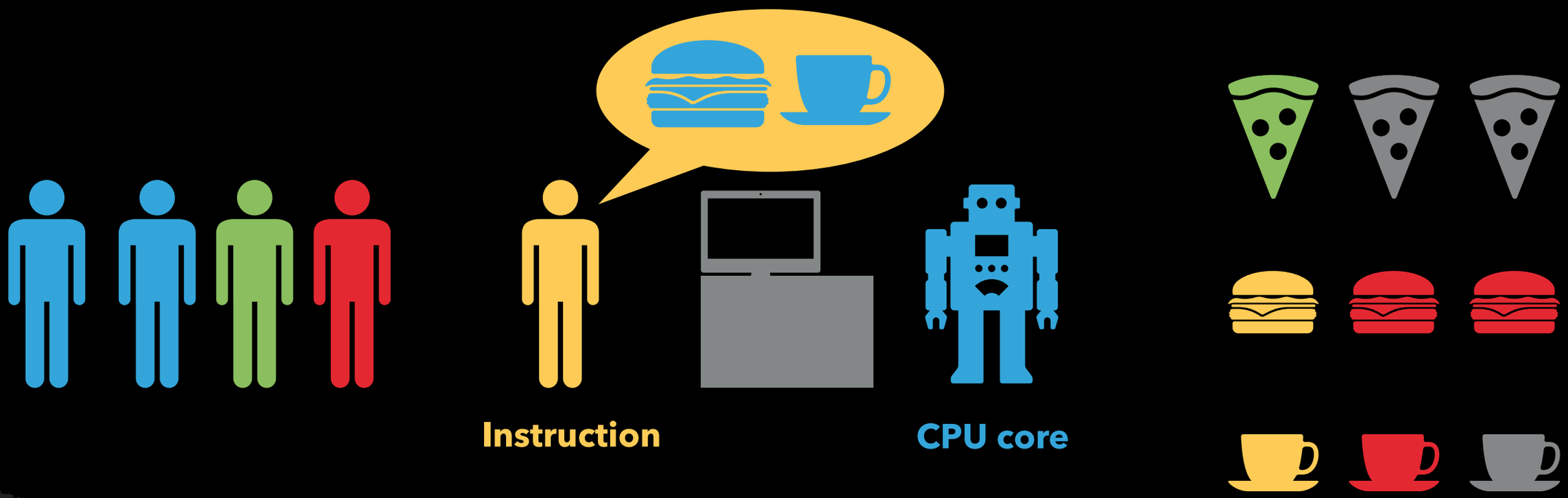
The green instruction will finish before the red instruction

The CPU ensures that red is seen before green

Actual µOP execution order

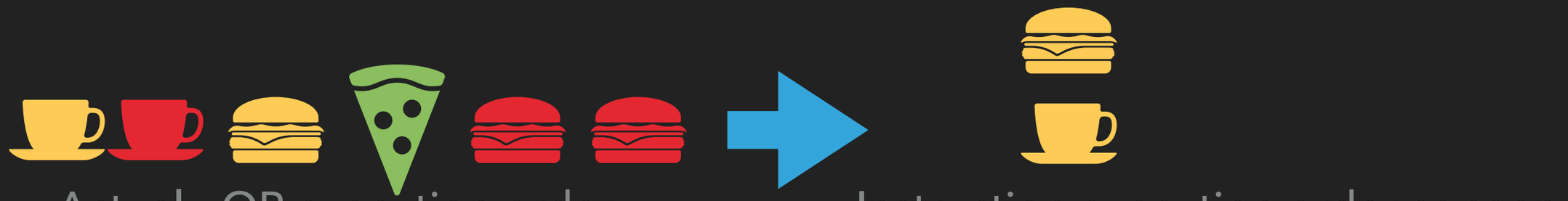
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction

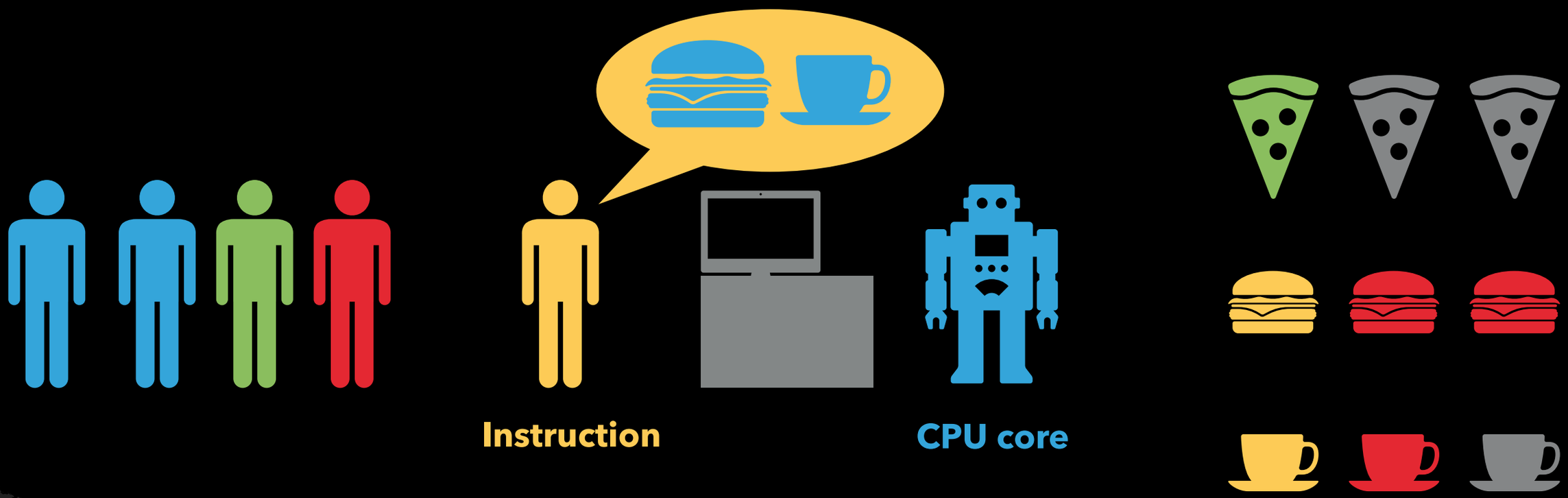
The CPU ensures that red is seen before green



Actual µOP execution order

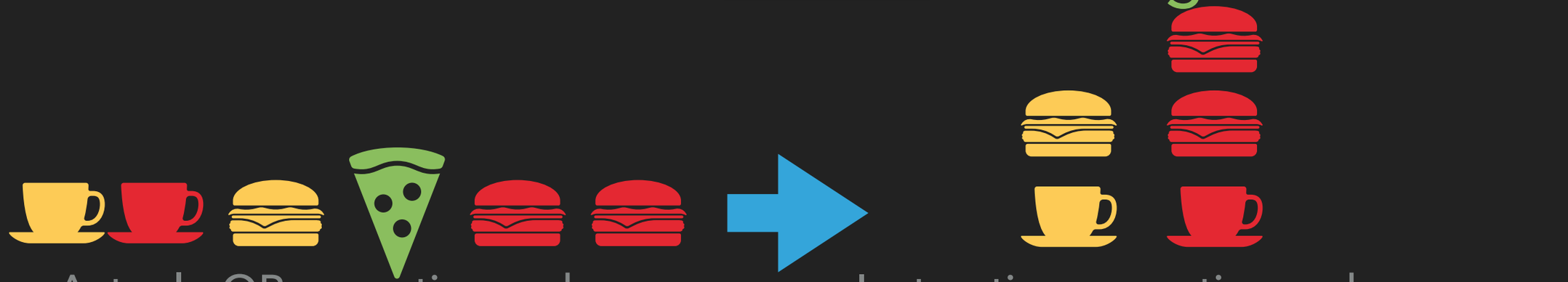
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction

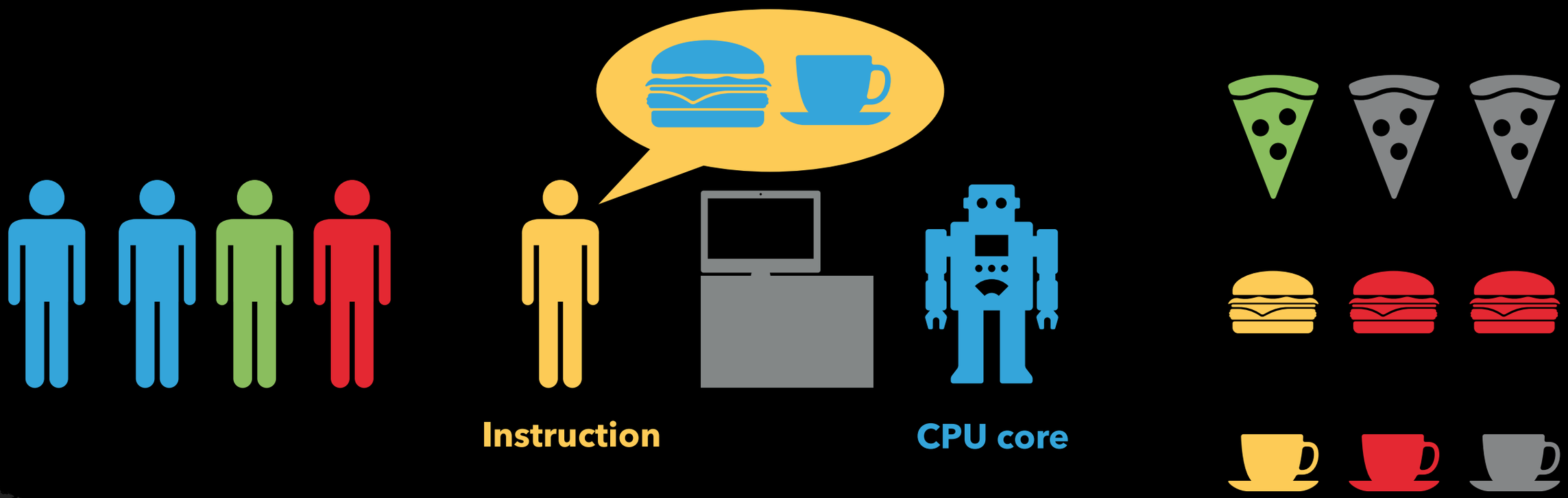
The CPU ensures that red is seen before green



Actual µOP execution order

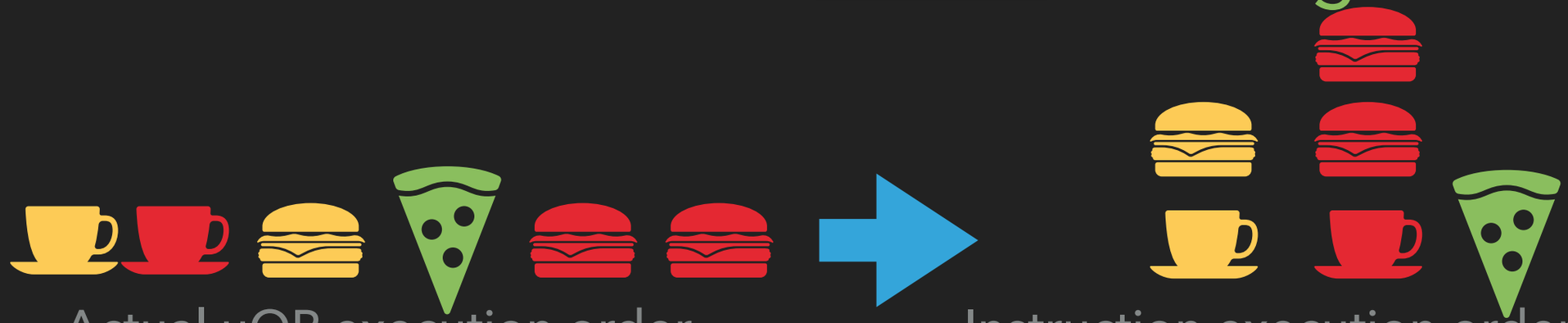
Instruction execution order as seen

CONFIDENTIAL BURGERS INC. : ORDER IS IMPORTANT



The green instruction will finish before the red instruction

The CPU ensures that red is seen before green



Actual µOP execution order

Instruction execution order as seen

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

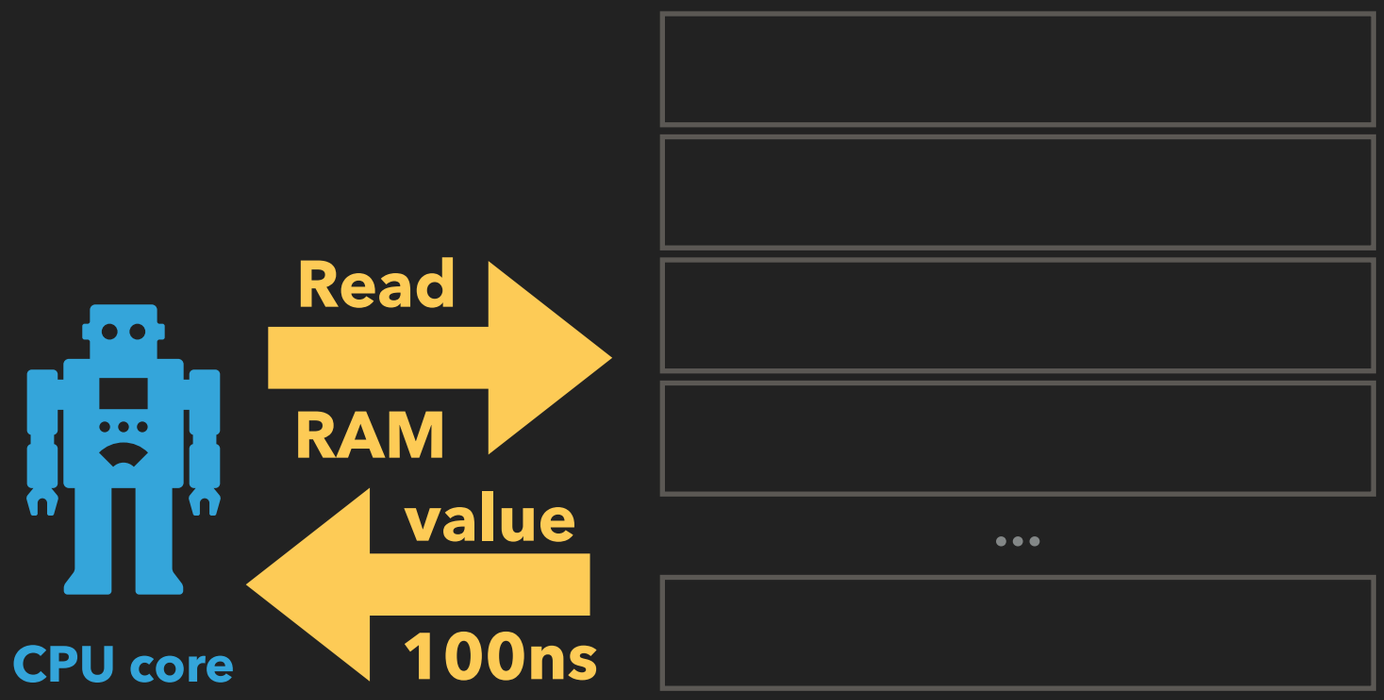
MELTDOWN

"Stashing" and "retrieving" the secret works via *side channels*.

Side channels are *observable side effects* of actions.

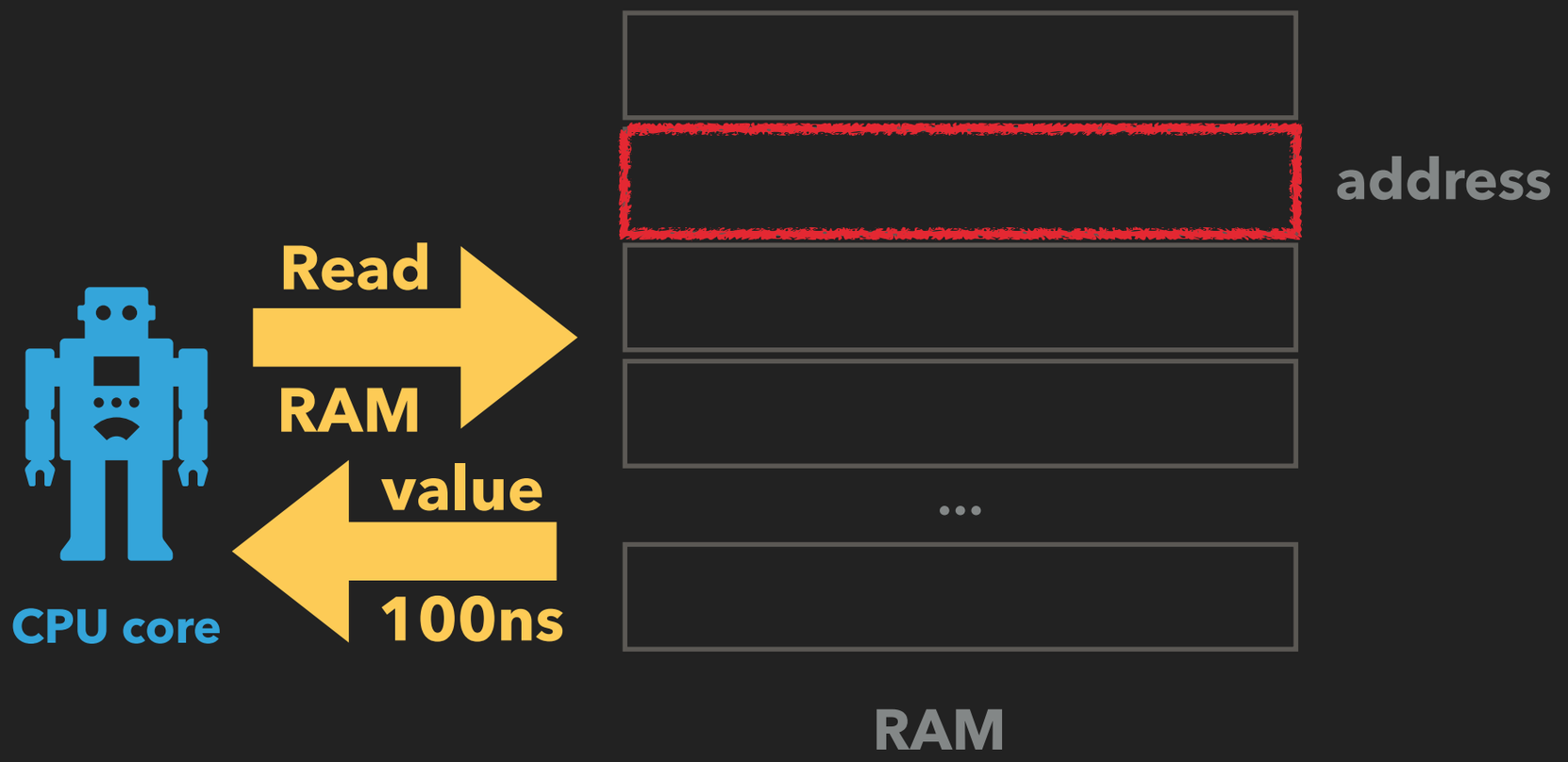
- READ secret from forbidden address
- Stash away secret by caching a memory location that depends on the secret
- Retrieve secret by finding which memory location is cached

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

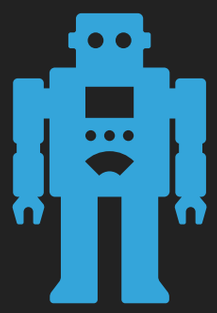


- Data is stored in RAM
- RAM is very slow
- Reading one byte stalls the CPU for hundreds of µOPs

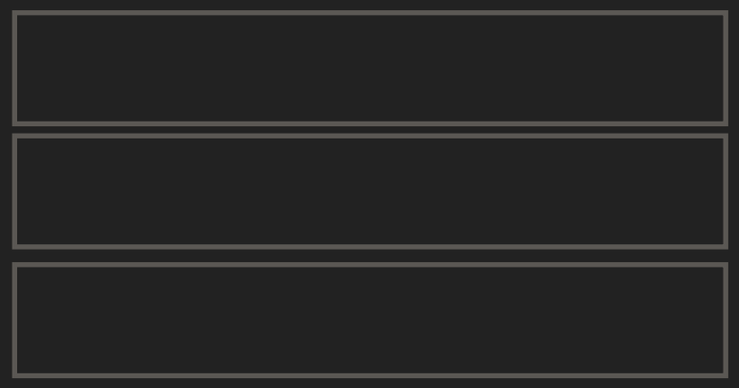
MELTDOWN: STASHING AWAY – SIDECHANNEL

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

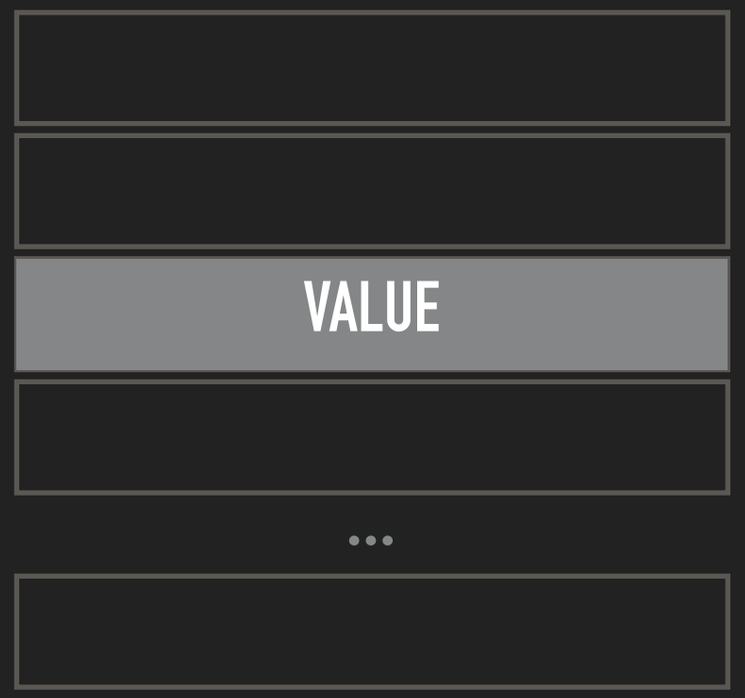
MELTDOWN: STASHING AWAY – SIDECHANNEL



CPU core



Cache

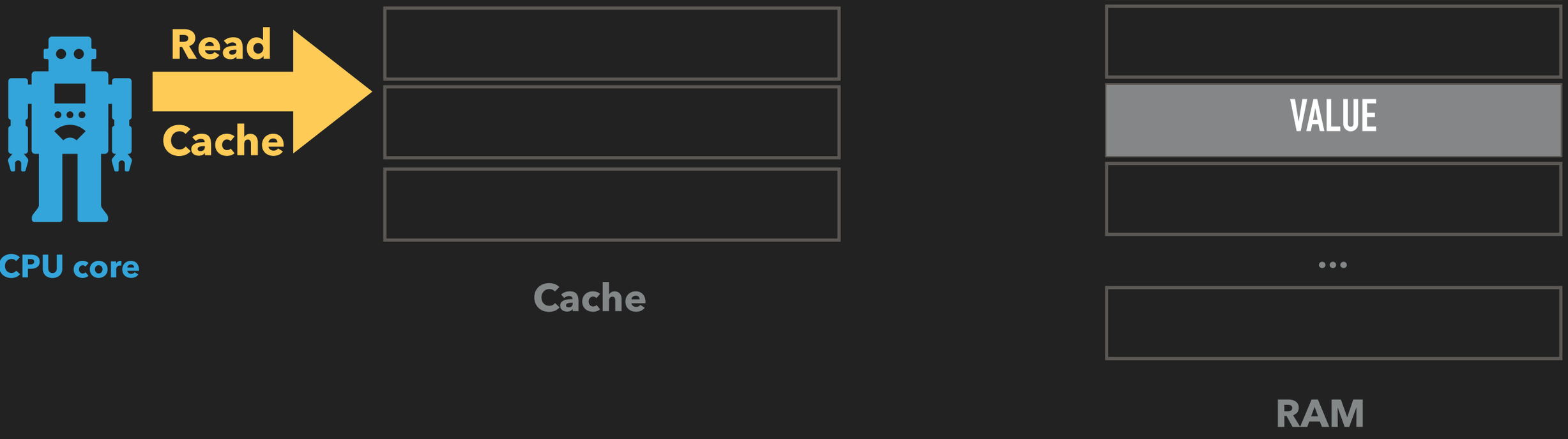


RAM

- 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"*

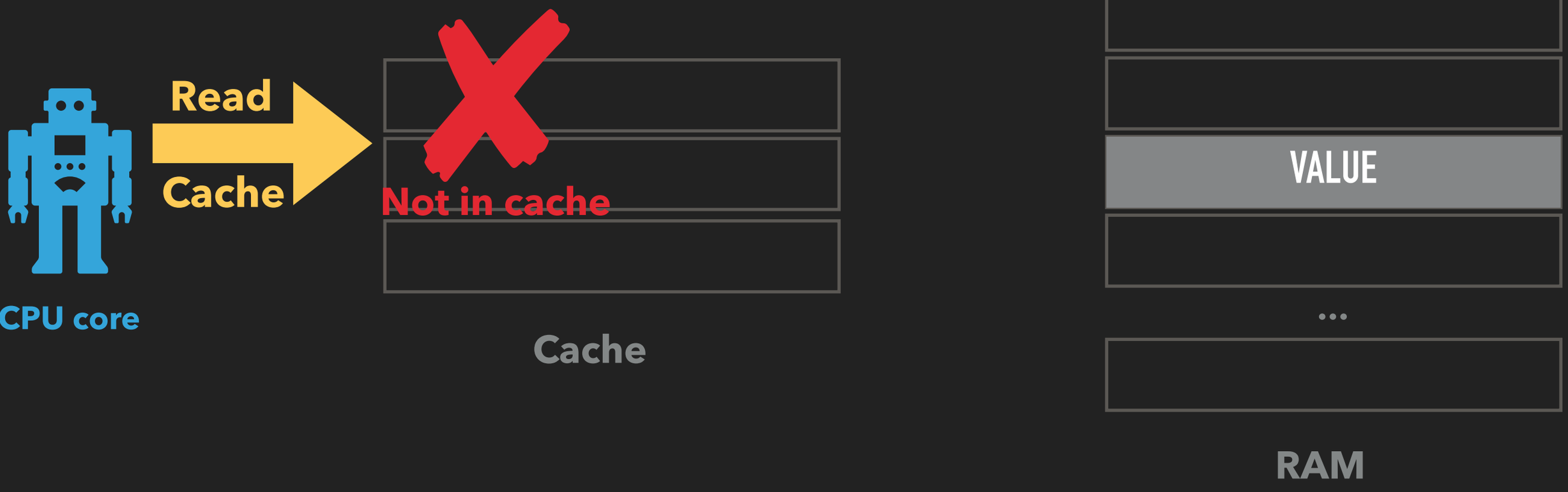
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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"*

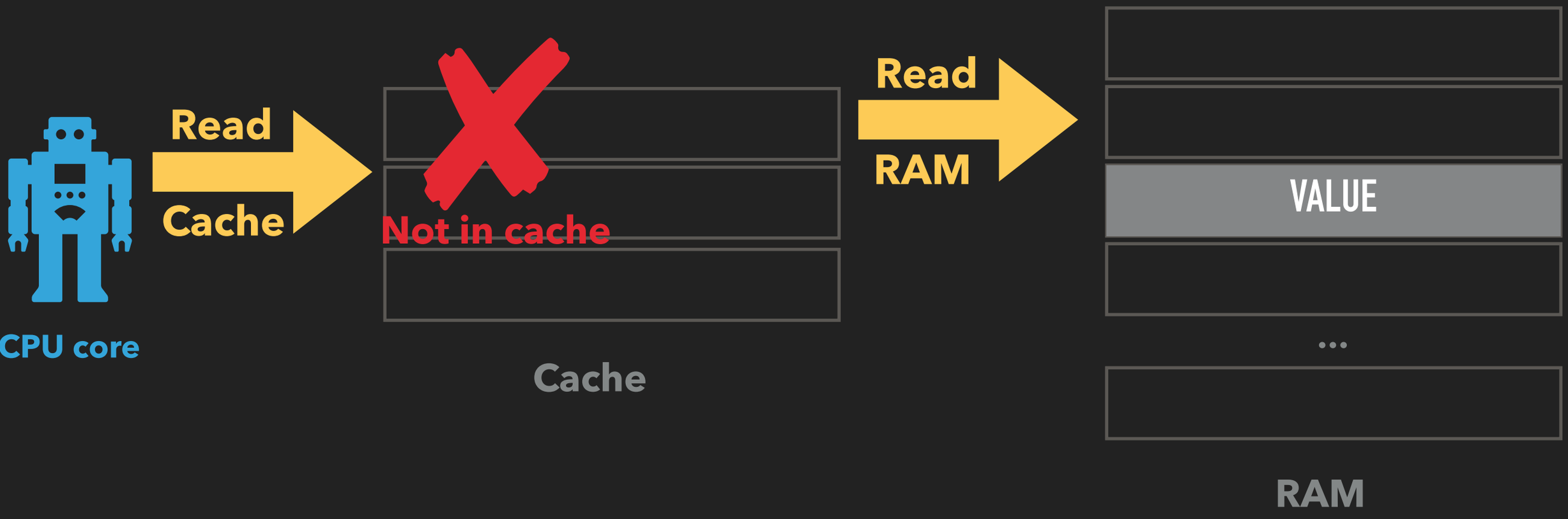
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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"*

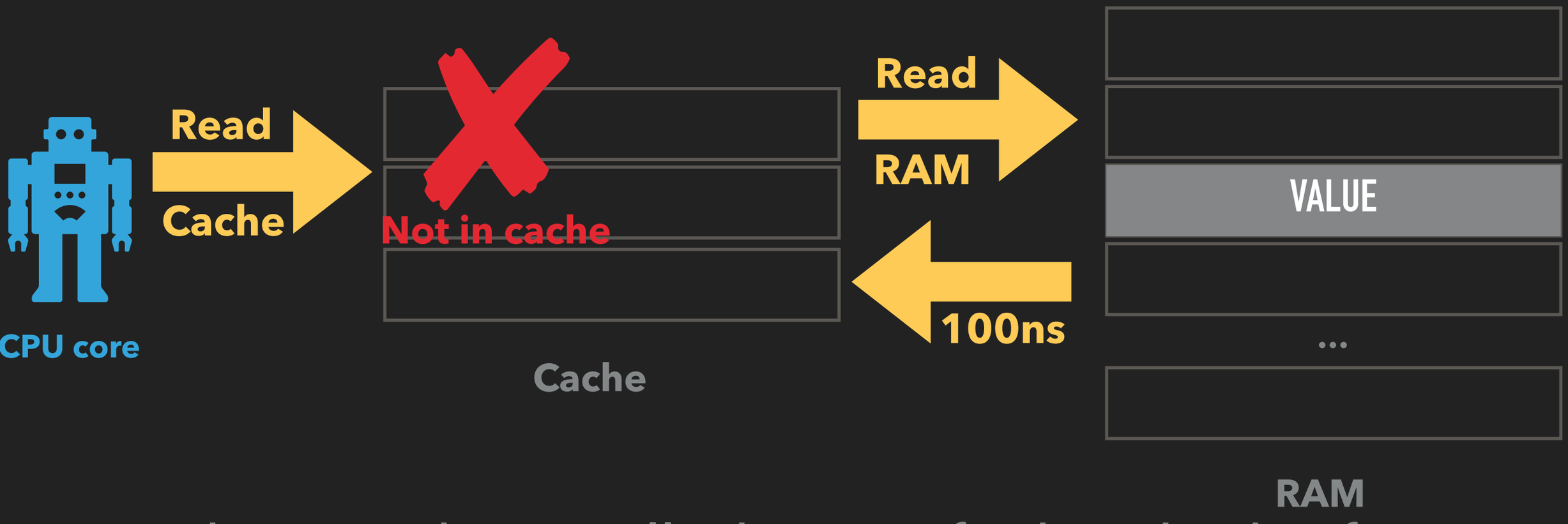
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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"*

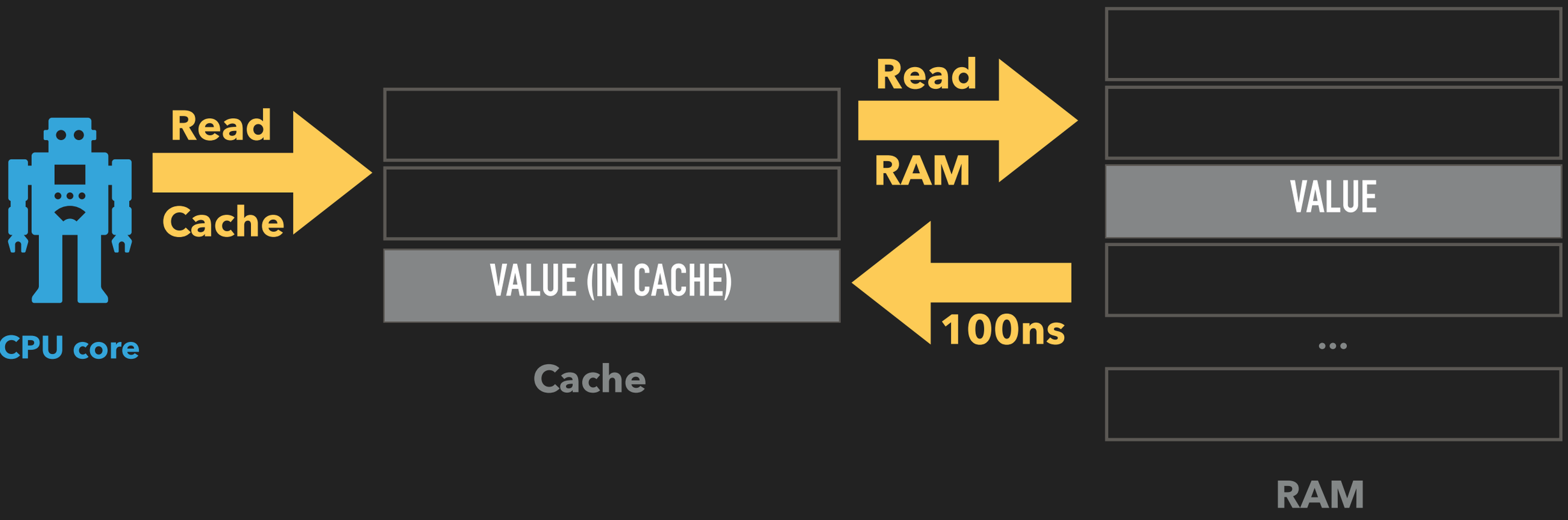
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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"*

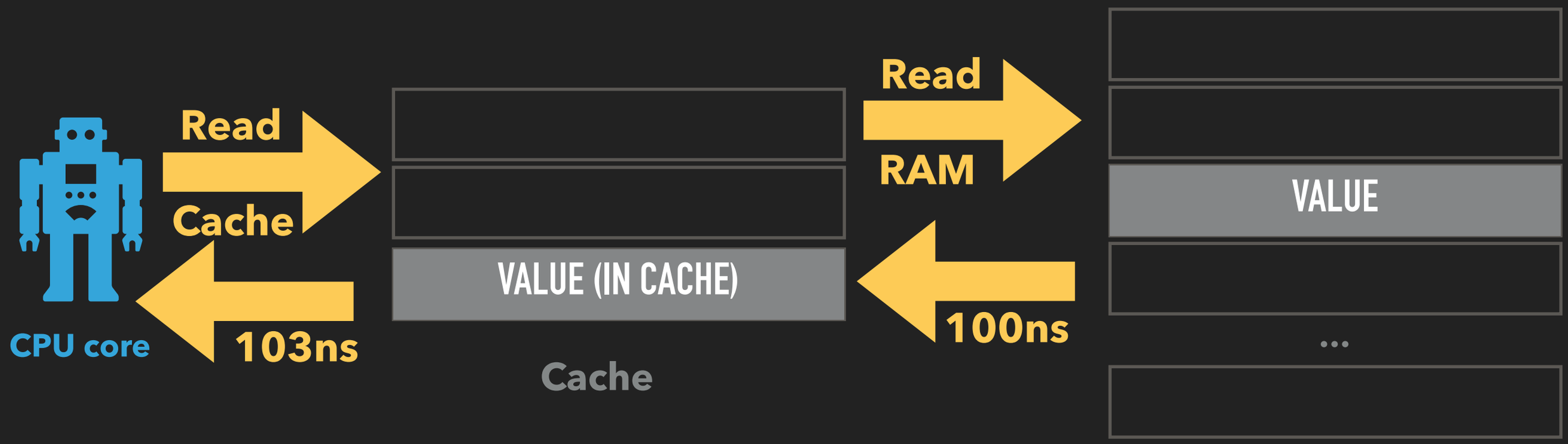
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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"*

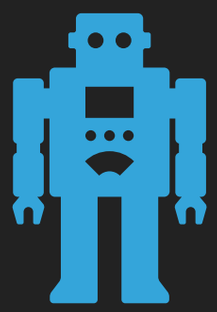
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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

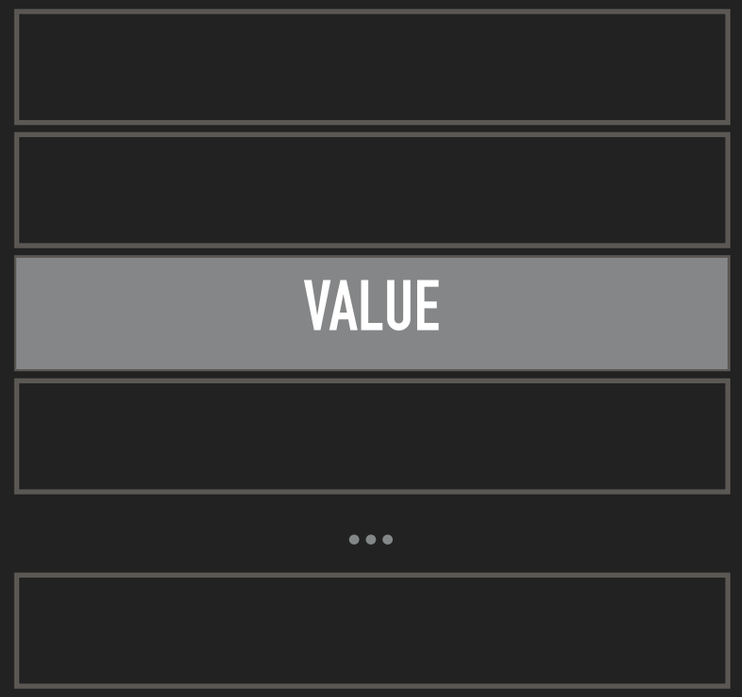
The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

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

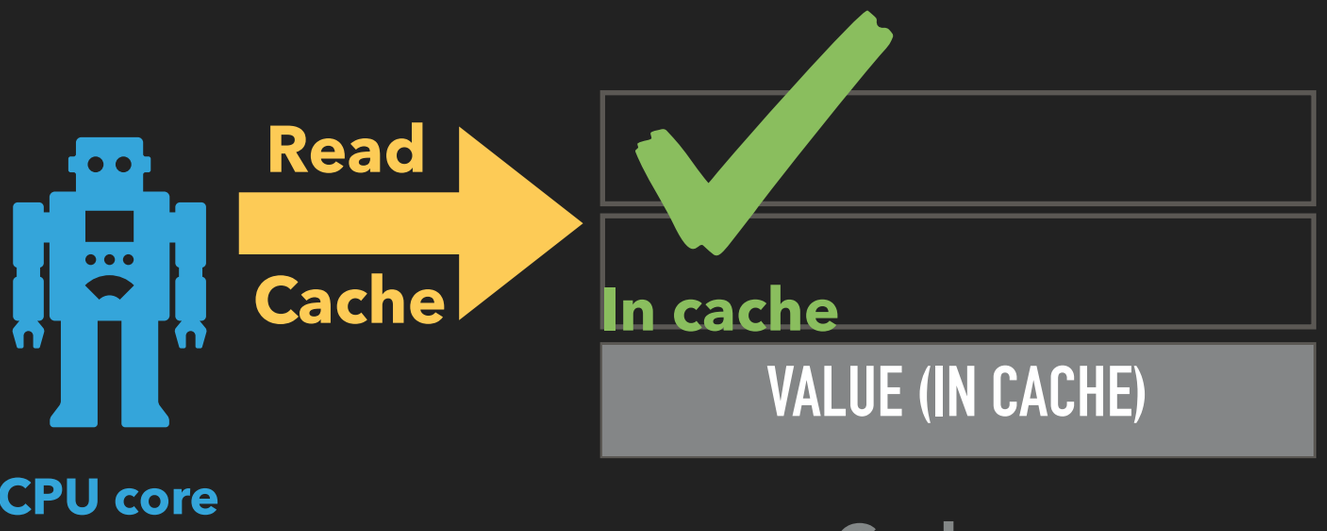
The cache speeds up *"what is the value at address X?"*. This is called *"(address) X is cached"*

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"*

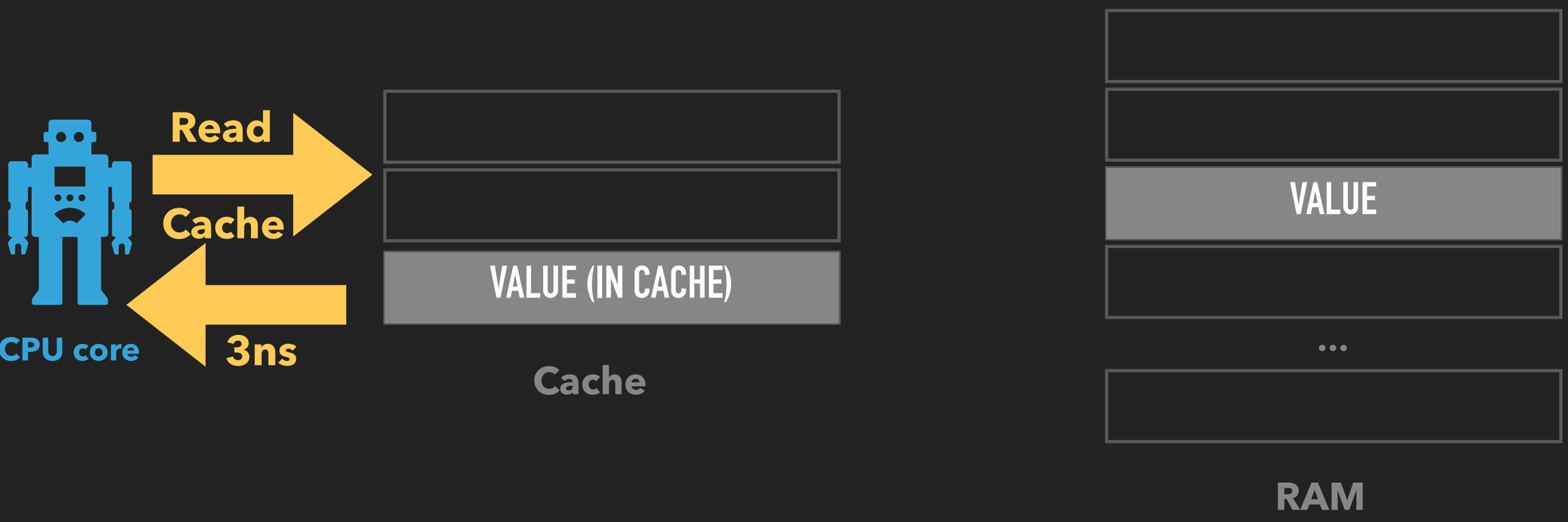
MELTDOWN: STASHING AWAY – SIDECHANNEL



- 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
2. Customer has not enough money
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 beca
that is only seen *iff* succe

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

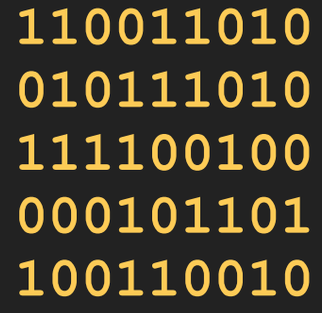
In our burger example:

1. **Customer orders a burger**
2. **Customer gets his burger**
3. **Customer has not enough money**
4. **Customer runs away with burger**

MELTDOWN

For Meltdown two actors are needed

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

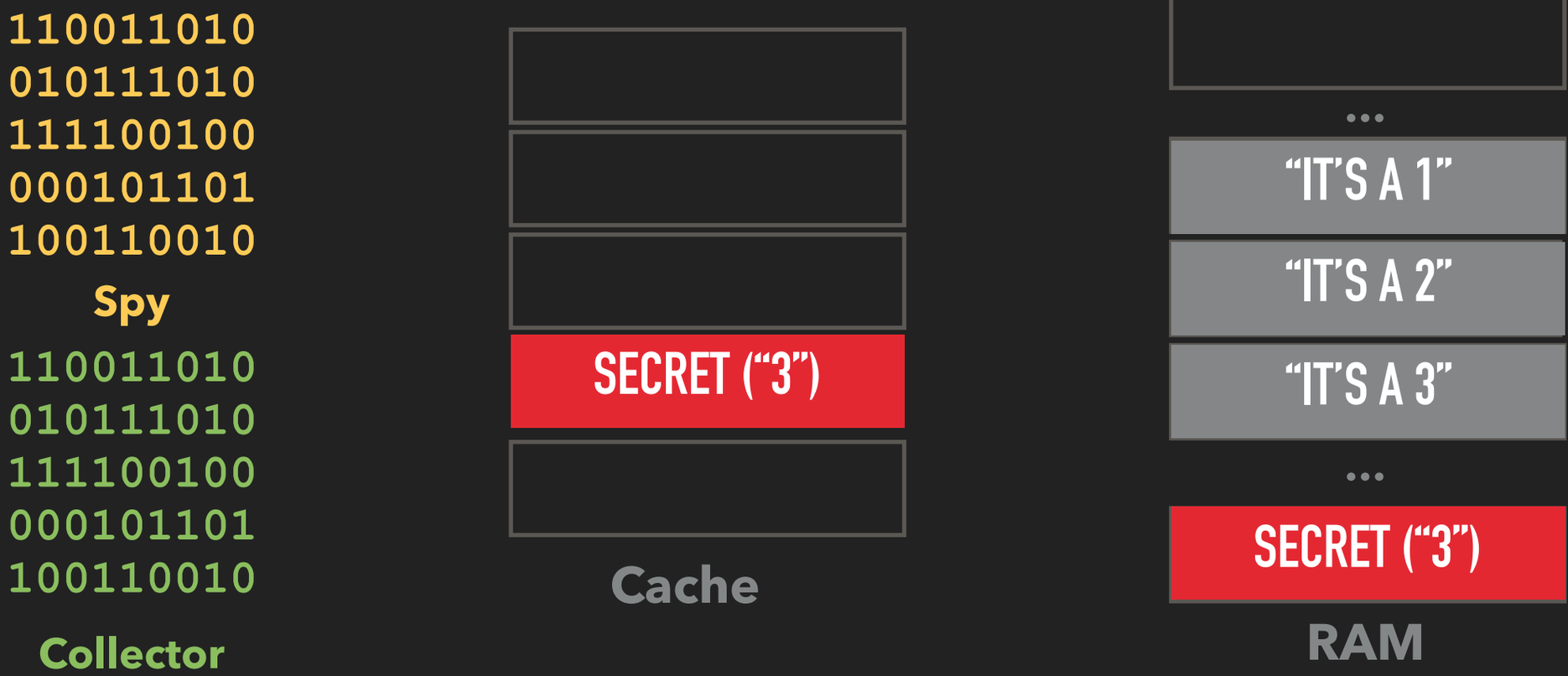


The **spy** will “steal” the secret and stash it away.

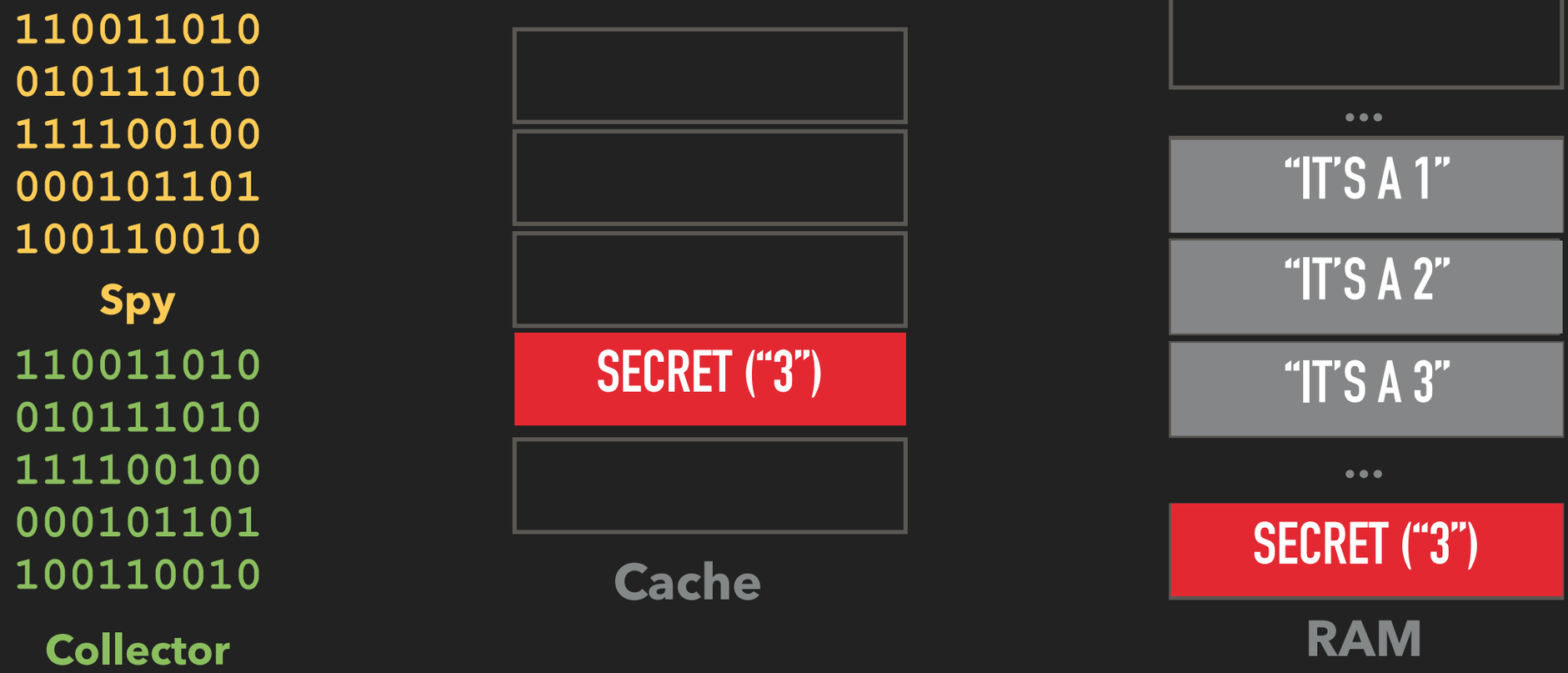
The CPU will kill him for accessing the secret information.

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

MELTDOWN: THE ATTACK

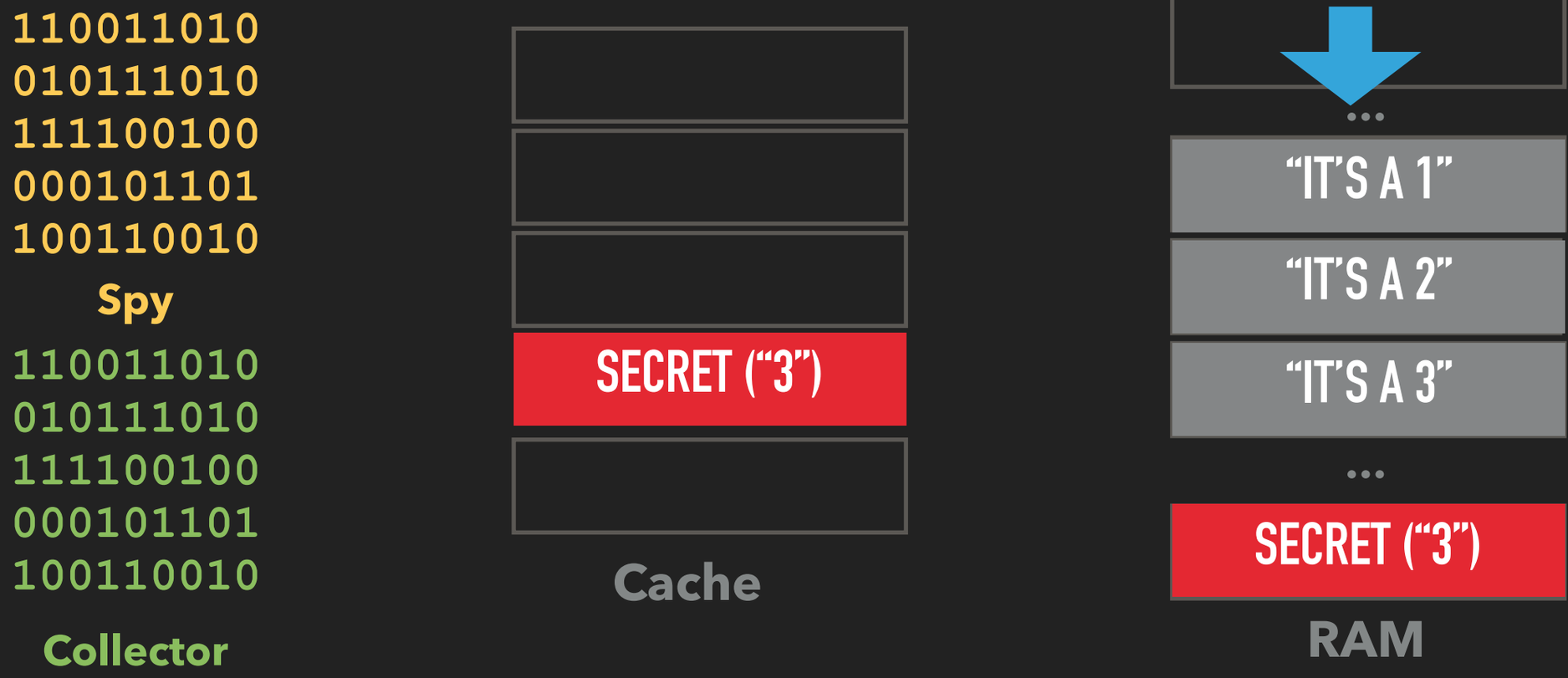


MELTDOWN: THE ATTACK



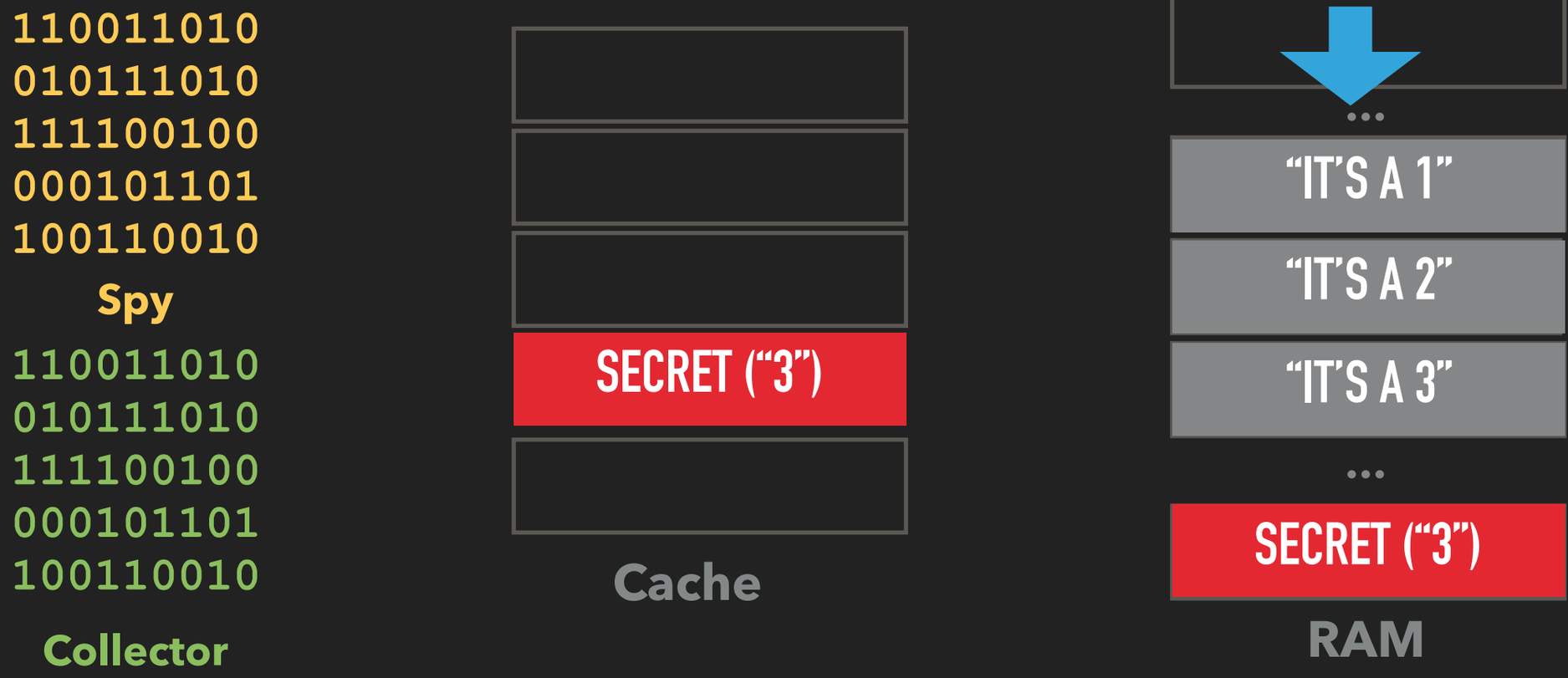
- Meltdown needs some preconditions

MELTDOWN: THE ATTACK



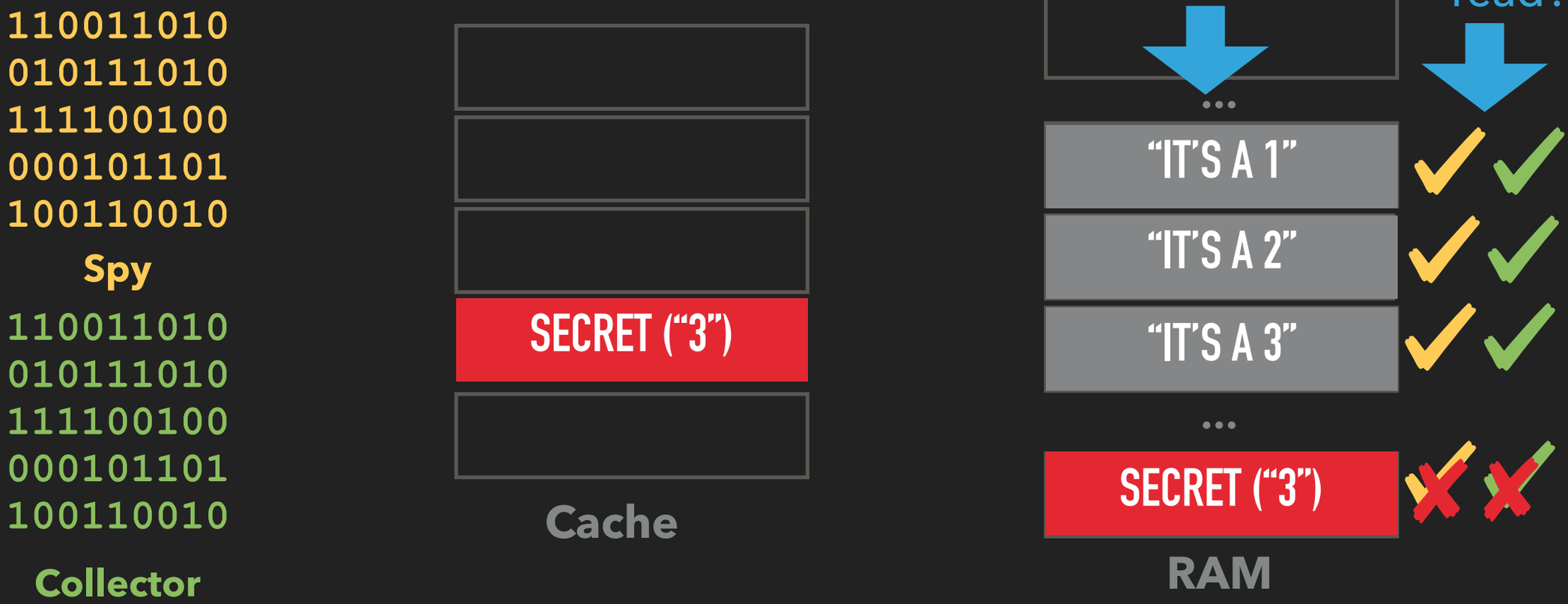
- Meltdown needs some preconditions

MELTDOWN: THE ATTACK



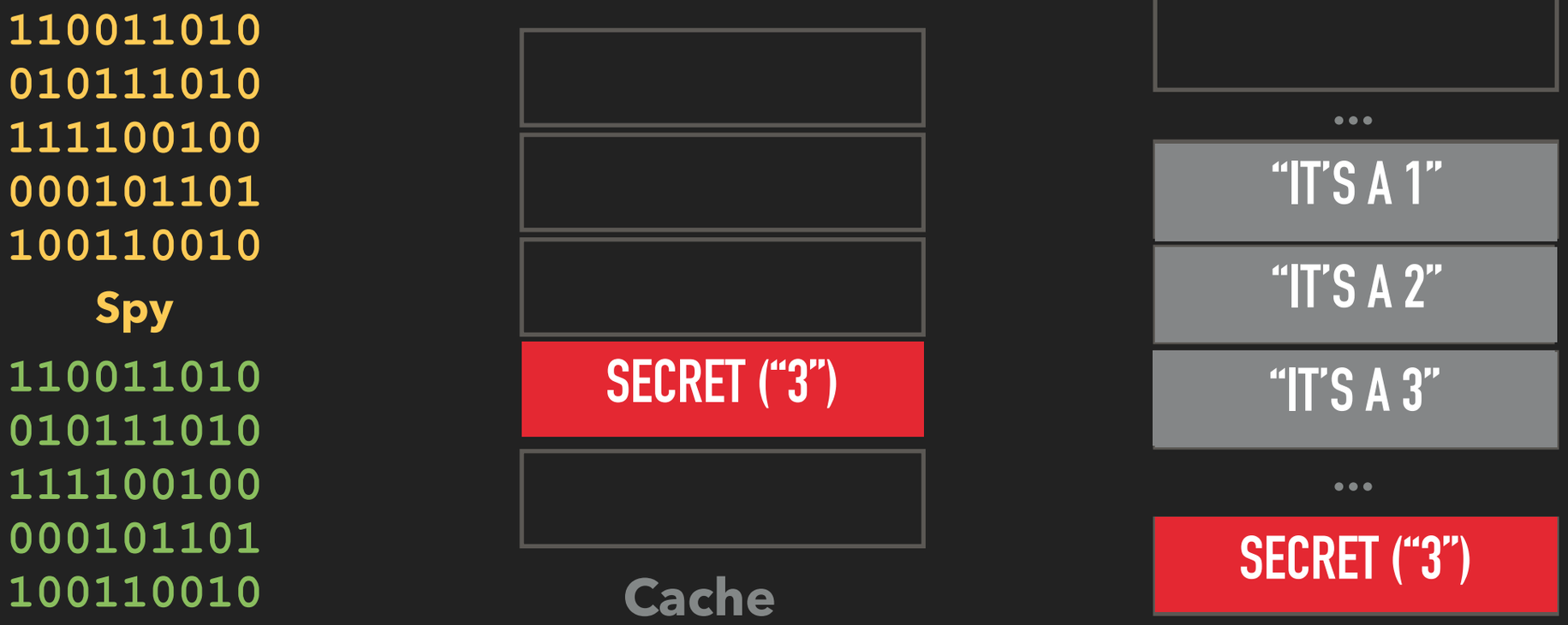
- Meltdown needs some preconditions
- The **secret** is in the cache (value: 3)

MELTDOWN: THE ATTACK



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

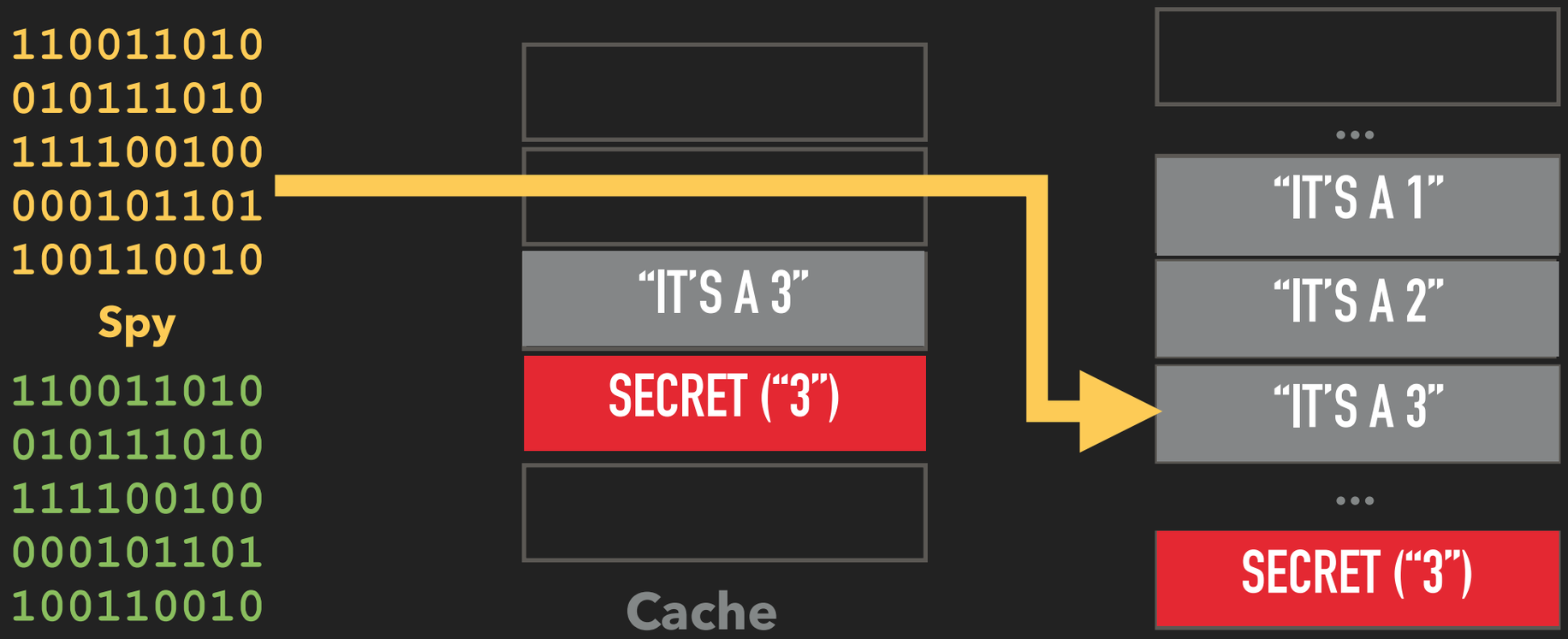
MELTDOWN: THE ATTACK



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**

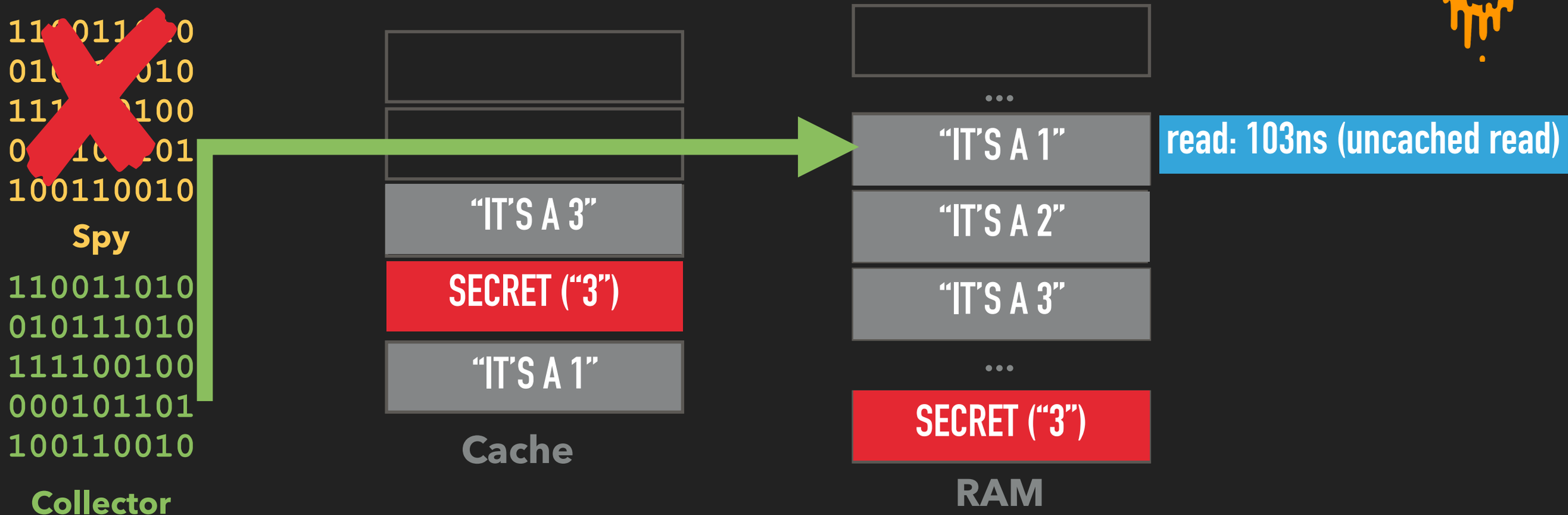
2. Depending on the **value**, **Spy** will cache a grey block[1]

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**

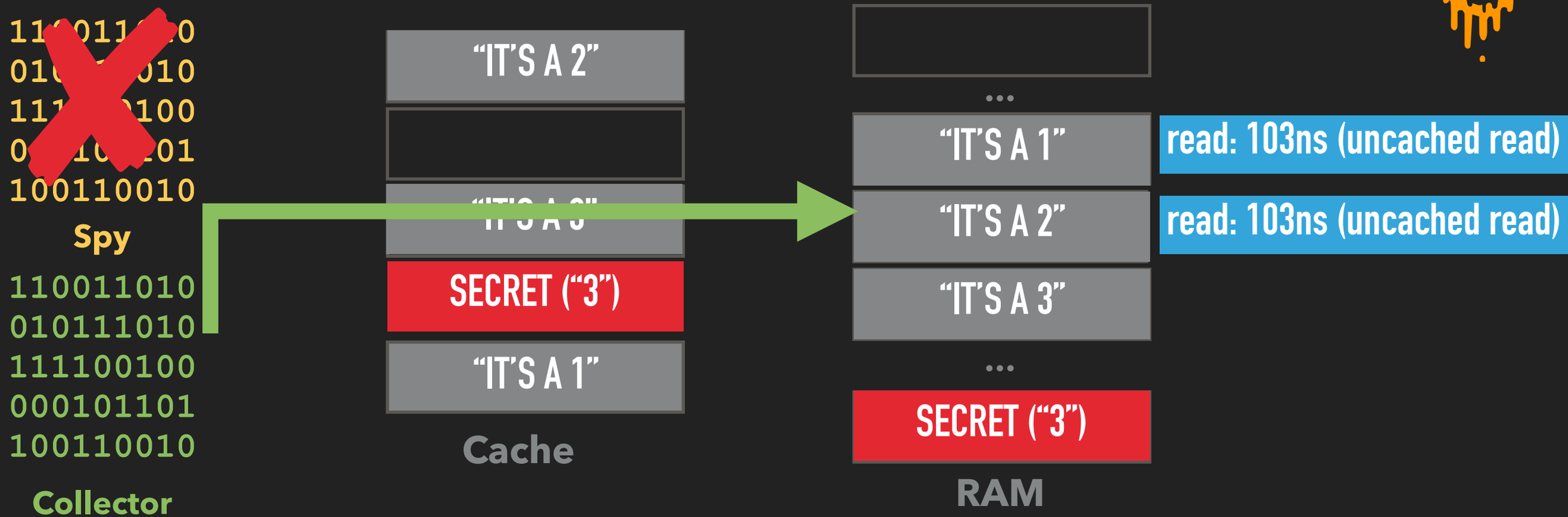
2. Depending on the **value**, **Spy** will cache a grey block[1]

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**

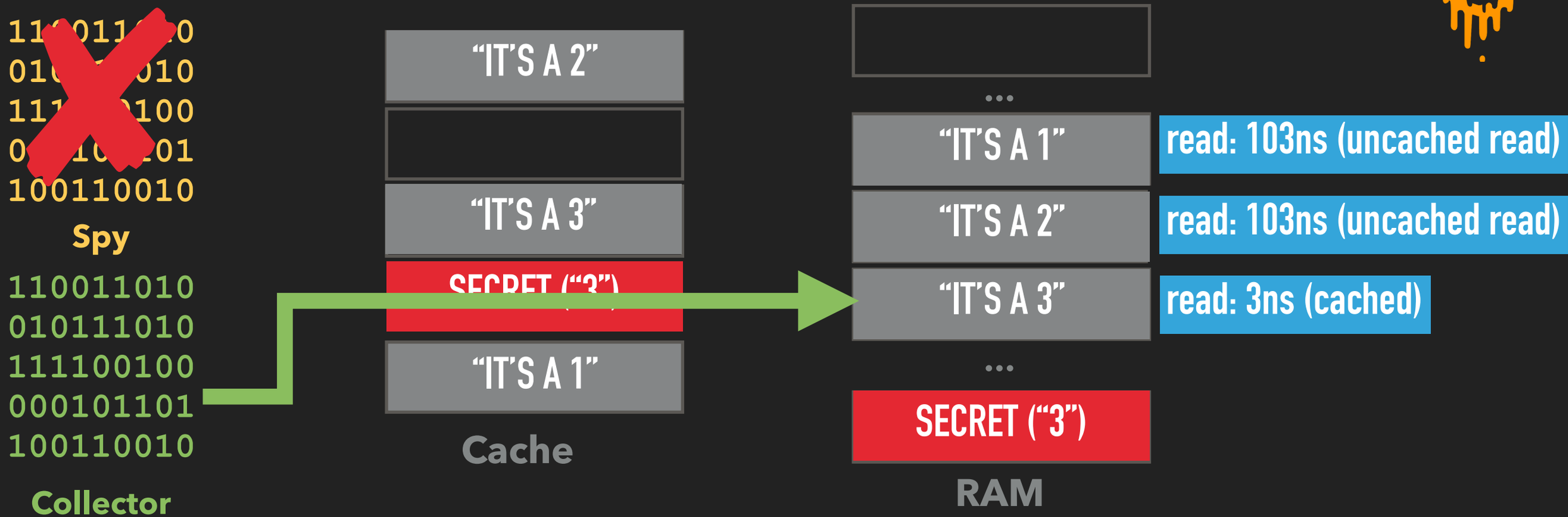
2. Depending on the **value**, **Spy** will cache a grey block[1]

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

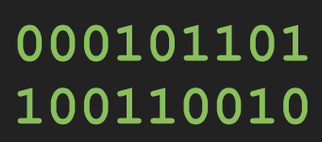
[1] Actually Spy will cache the *address* of block #3 and Collector will read the blocks *addresses*

MELTDOWN: THE ATTACK

read: 103ns (uncached read)

read: 103ns (uncached read)

read: 3ns (cached)



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 *speculative* execution of OPs and µOPs
- Timing side channels for *speculatively executed* OPs

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”

[1] Virtual vs. physical memory is a subject for another time

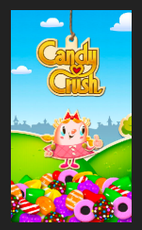
SPECULATIVE
EXECUTION

SPECTRE

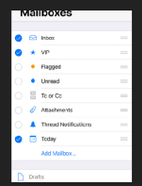
Q & A

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/

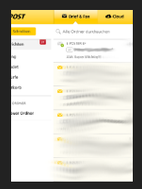


CANDY CRUSH LOGO FROM THE APP STORE © KING
HTTPS://DISCOVER.KING.COM/ABOUT/



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

ASSETS