Tr

Haym @SalomonCrypto

Sep 26 • 26 tweets • SalomonCrypto/status/1574208089661747200

(1/25) Computer Science Basics: Virtual Machines

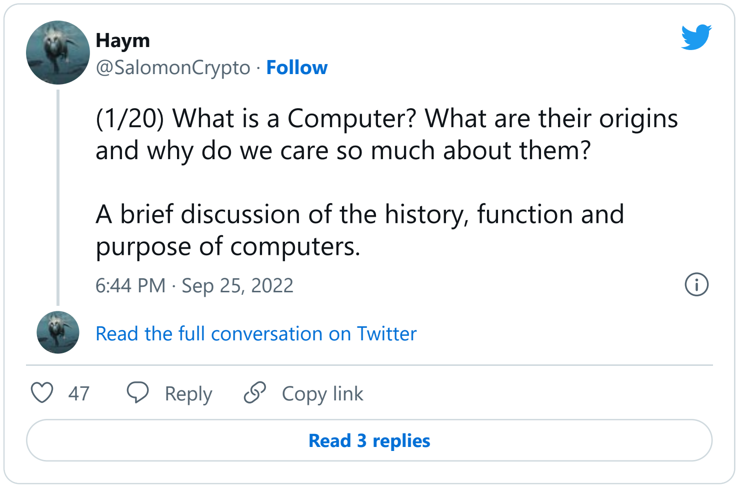
The secret to innovation - Abstraction

The final form of computational abstraction - Virtual Machines (VMs)

Your guide to the concepts and technologies at the core of computer science, blockchain tech and @ethereum.

(2/25) Computer: a digital electronic machine that can be programmed to carry out sequences of arithmetic or logical operations (computation) automatically. -Wikipedia

Wat?



Haym
@SalomonCrypto · **Follow**

(1/20) What is a Computer? What are their origins and why do we care so much about them?

A brief discussion of the history, function and purpose of computers.

6:44 PM · Sep 25, 2022

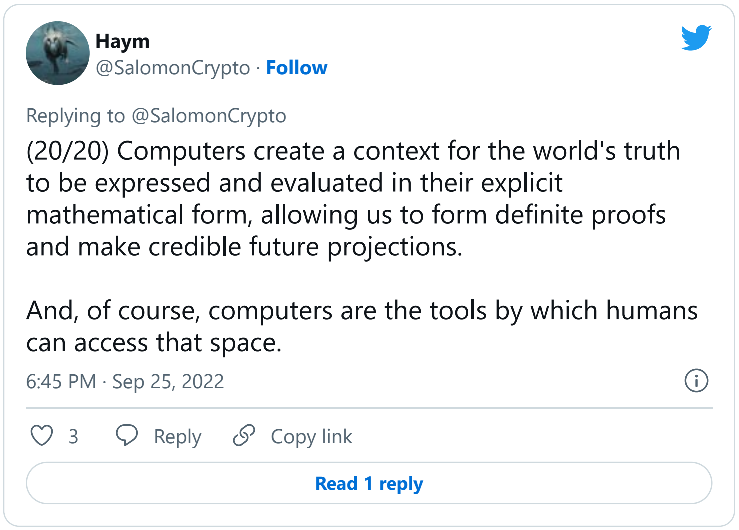
Read the full conversation on Twitter

47 Reply Copy link

Read 3 replies

(3/25) The way I think of a computer is it's a machine that provides a platform for the (incredibly) quick and efficient calculation of math.

By translating human-native thoughts and problems into math, computers provide real-world utility.



Haym
@SalomonCrypto · Follow

Replying to @SalomonCrypto

(20/20) Computers create a context for the world's truth to be expressed and evaluated in their explicit mathematical form, allowing us to form definite proofs and make credible future projections.

And, of course, computers are the tools by which humans can access that space.

6:45 PM · Sep 25, 2022

3 Reply Copy link

Read 1 reply

(4/25) A computer is a physical object; it is made up of real materials, it actually moves electricity through circuits, and makes real changes in its physical state as it functions.

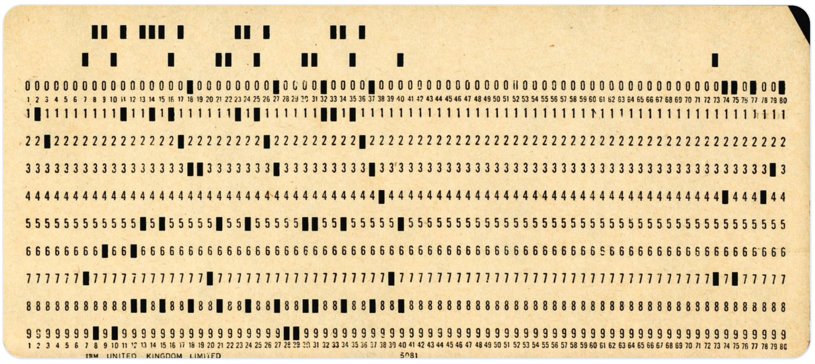
Development on physical computers must account for all of this.

(5/25) If you go back to VERY early computing, you can visibly see this.

The first computers used punched cards for input, output and storage. The physical location of a hole on a piece of paper would represent computational data.

(6/25) First, a developer would phrase a question that could be represented by math (in these early days, often it was just... simple math).

They would make a hole at the spot that marked the correct number or operation and insert the card into the computer.



(7/25) The computer would be designed with the specific card in mind, and so would know which number/operation that every hole corresponds to.

It processed the card, translating the expression punched into paper into machine language and building the model imprinted on the card.

(8/25) Once the model has been constructed, the computer can then evaluate it (much more quickly and precisely than a human could, even back in those times).

Finally, the computer communicates the results by punching new holes into the card (or a new card).

(9/25) This was the essence of early computing: humans wanted answers that computers had, but they didn't speak the same language.

Punched cards act as the interface between the two, providing a shared language by which both parties can communicate.

(10/25) While groundbreaking for its time, this system has some severe drawbacks. We'll skip passed most of them, we're here for one in particular:

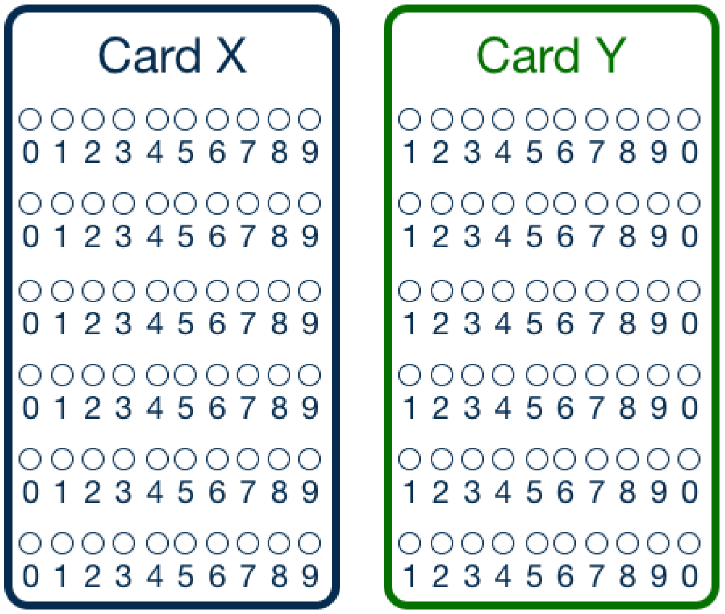
Programs created for punched card computers were INCREDIBLY specific to the individual machine they were created for.

(11/25) Here's an example:

Imagine you want to represent the number 22 for a punched card application. If the computer accepts cards in the format X, you will punch:

--X-------
--X-------

If you feed that card into a format Y machine, it will read the same input as 33.



(12/25) So let's recap: a developer was not only responsible for translating a real world problem into a mathematical statement, he also had to manage all the peculiarities of each machine he was working with.

Any innovation on one machine had to be recreated on another.

(13/25) To cut to the punchline, the solution was abstraction. Developers were removed further and further from the concerns of the hardware with the development of high-level languages like BASIC and C++.

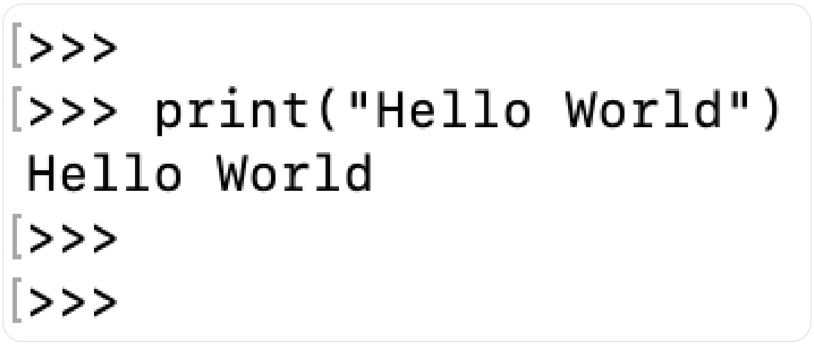
(14/25) The target of a high-level language is not a computer, it is a compiler/interpreter, programs that translates one language into another.

Developers can write code in a language that expresses information in a human-digestible format and interpreters will do the rest.

(15/25) Let's say a developer wants to write a program that prints "Hello World"

In a punched card system, the developer would have worry about specific measurements and machines.

In an abstracted system, the interpreter takes care of 99% of the work. This is all it takes:



```
[>>>
[>>> print("Hello World")
Hello World
[>>>
[>>>
```

(16/25) But here's the thing... there's an ENORMOUS difference between 99% and 100%.

We wont get in to specifics, but suffice to say that every dev is thinking about the machine specs and operating systems of their users.

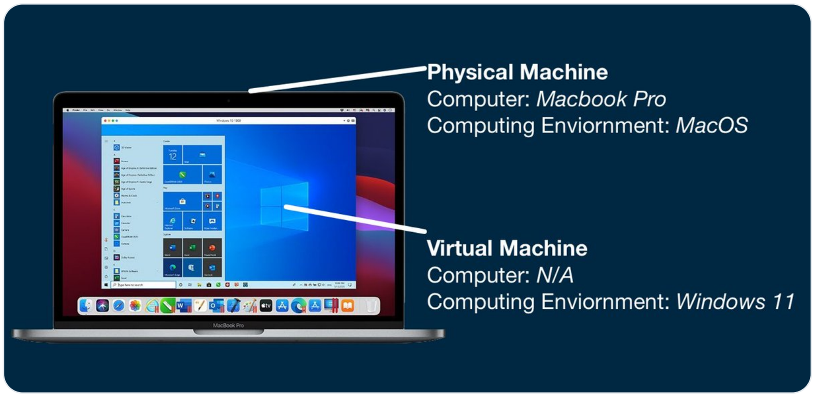
Fortunately, we don't have to stop at this layer.

(17/25) High-level code still gives developers exposure to the machine, so we'll abstract it away. We'll create a Virtual Machine (VM).

A VM is a piece of software that emulates a complete (and enclosed) computer system within another computing environment.

(18/25) The best example is the one you can see; take a look at the Parallels app.

Parallels runs on Apple computers in the MacOS environment. Once installed, it allows the user to run a copy of Windows entirely within MacOS/their Apple computer.



(19/25) The example above is a little bit of a red herring; when most people see Windows-on-Mac the reaction is "I guess you can play games on Mac."

The VMs we care about don't look like this, it's just a great illustration.

The VMs we care about don't look like anything.

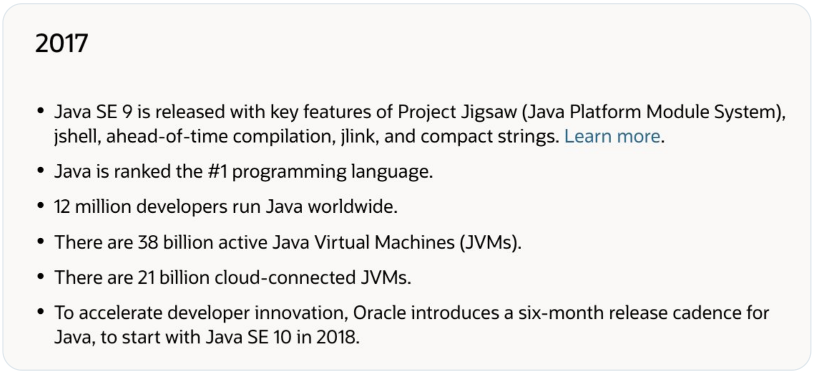
(20/25) While not the first, the most famous early VM is @java. Java provided a fully capable computing environment that anyone could develop on.

Every instance of Java would support the exact same features; it is up to the computer to support Java (well... it's mutual).

(21/25) When a developer wants to deploy a feature, he only needs to worry if @java will support it. If it does, he can be confident that it will run on every computer that can run the Java VM.

If anyone's counting, there were 38 billion instances of the JVM...

...in 2017.



2017

- Java SE 9 is released with key features of Project Jigsaw (Java Platform Module System), jshell, ahead-of-time compilation, jlink, and compact strings. Learn more.
- Java is ranked the #1 programming language.
- 12 million developers run Java worldwide.
- There are 38 billion active Java Virtual Machines (JVMs).
- There are 21 billion cloud-connected JVMs.
- To accelerate developer innovation, Oracle introduces a six-month release cadence for Java, to start with Java SE 10 in 2018.

(22/25) In practice, VMs are not needed for most apps. Most consumers will barely touch a VM, those that do probably wont notice.

Their importance becomes much more clear behind the scenes, especially in infrastructure, corporate development and other large-scale projects.

(23/25) We began our story in the era of punched cards; back then there were like 500 computers TOTAL and even then the complexity was a nightmare.

In 2022, things are many orders of magnitude more complex with near endless variations in the machines we call computers.

(24/25) Abstraction is so powerful because it frees devs from worrying about the entire system; they can focus on the part that best allows them to express the question they are trying to ask.

VMs are the conclusion of computational abstraction; a computer within a computer.

(25/25) A parting thought:

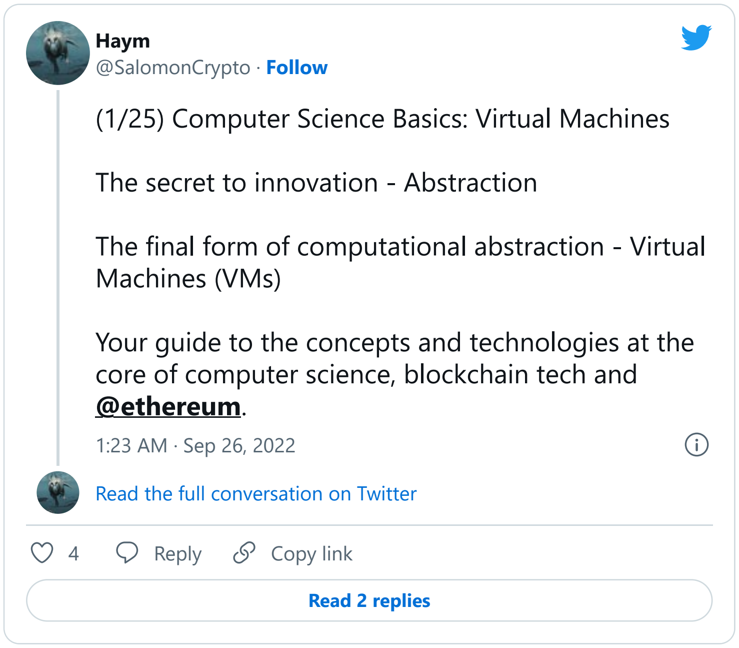
A computer within a computer means that we can all exist in the same computing environment, but (usually) just locally.

What if we wanted to exist in the same actual environment... like literally 1 VM?

What if we wanted to do it trustlessly?

Like what you read? Help me spread the word by retweeting the thread (linked below).

Follow me for more explainers and as much alpha as I can possibly serve.


Haym
@SalomonCrypto · Follow

(1/25) Computer Science Basics: Virtual Machines

The secret to innovation - Abstraction

The final form of computational abstraction - Virtual Machines (VMs)

Your guide to the concepts and technologies at the core of computer science, blockchain tech and **@ethereum**.

1:23 AM · Sep 26, 2022

Read the full conversation on Twitter

4 Reply Copy link

Read 2 replies

• • •