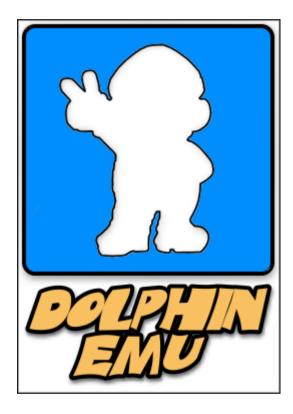
#### Inside an Emulator

キーホンノバ協び着の空母の日日で、
「キスアロ特にしたから」が出たしたまた。「この日本の日本では、「この日本の」であった。「「この日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の日本の
アドキとロるアくなのくる辺辺るアメタロロなるアキン井下山の年メオディ
※1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、1、
(学校の日本)をおまたす。「ない」のは、「ない」ので、「ない」ので、「ない」では、「ない」」では、「ない」」では、「ない」」では、「ない」」では、「ない」、「ない」」では、「ない」」の。」」」、「ない」、「ない」、「ない」、「ない」」」では、「ない」」」」、「ない」、「ない」、「ない」」」、「ない」、「ない」」」、「ない」、「ない
00004990190710000 000000000000000000000000000
○上谷本口日本アミデカデタデアアデルやヨデカの
2
- ○○号ものの今日人もからソコノへんかは作作りかり干日が入りのものというのでの人もかでした。
4年年「「たんがン協会上登録目式上部市「古ななな日本」など提供した。
269234151 20114640194
● 12717371735 ● デロ商用曲体ななながかアク
のさキャステレロ共同され上下を言のさり
「キナケゾ母ゾ8日の人上11日日ム」
いまい、「キャメモン」というのでのの名となりました。 「アージののの名となるの名と、「ないのうな」というのの名と、「「ない」」、「シージー」、
···・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・
▲根へ谷町米米ターよら上の身白さ山谷の町街台やアデアプロムで手
(04年により、14日本での人間の時代の4年の一年によります。)
A STALOOLSAASAOLA-OSSX-LSSA-
「ケロケトのアのキスゾゾアゾ目からか」下ノくケちムの手下くカアさんのおびはんケカア、 しゅうかんびゅのちょ アちさんおく いまいろ
〒山地1+母1×8ヶ市ワ60~市
#4为中心为87%以7并23日6%年2年人834,7为4%人1%为172次00%4%人2%7%以上87%1%1%400%1%1%2%1%2%1%2%1%2%1%2%1%2%1%2%1%2%1%2
●アドレンロのササポイズリント: なもののではない「上のワットメンター」 ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・ ・
Parse ALDACARTY ALTER A
11年人はほんパメメンドをなめったら近近後にものをみなどの日日 の人ものの日本を含む、11年のの人は近後に、してもようしてもなりにならならのためのだめがたがら、
1.2.2.4.1.4.2.4.4.1.4.1.4.1.4.1.4.1.4.1.
「中国」になるないかでのたままで分かれた日本を見かっつがまかかった」
二〇一〇日〇日で単手アやケムノタロ目を見なのがられるメタクはつくていたがない。
うともあるが、およれないになったから、「しついいである」をした。
Sessed and the sessed of the s
今日かんとうと、「」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」」
= r < r 180 r - 186 x X < #84 r 8
10.2111月1日の日本部の日本部の日本部の日本部の日本部の日本部の日本部の日本部の日本部の日本
8年年1842年18日 10日 10日 10日 10日 10日 10日 10日 10
19日本での日本に、日本では、19日本には、19日本に、1
14日へとのロケモデアジルボ しのういいへきにつきにもありまし
A BARBOVENERSKERDAGEAGEAGEAGEACHINAAAABEGVAENI A BARBOVENERSKERDAGEAGEAGEAGEACHINAAAABEGVAENI A BARBOVENERSKERDAGEAGEAGEAGEACHINAAAABEGVAENI
(キャンペントの参加ですとしていたが、ログルログになっていた。 またいし、ログントの参加でしたいがあるののなどのもには、「「「「「」」」」
●水油口口●出心ドメメド <b>び</b> 200以後の名 <b>び</b> (大和の2011)~ の土土をメロルト、
SFROStar STORTAN

Maarten ter Huurne T-DOSE 2014

#### About the speaker





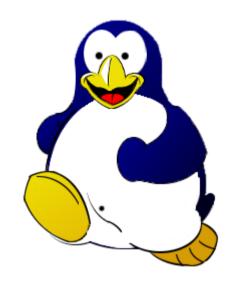
MSX

openmsx.org

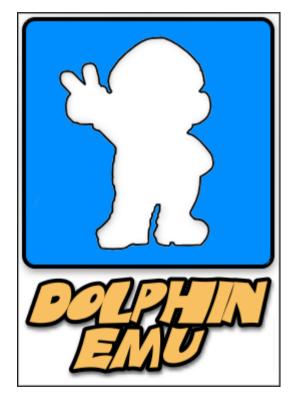
GameCube + Wii

dolphin-emu.org

#### About the speaker



#### openMSX



2001-2014

2992 commits

2008, 2011

235 commits

# Contents

- What is emulation?
- How does it work?
  - CPU emulation
  - Peripheral emulation
  - Synchronization
- Thinking non-linearly
  - Determinism
  - Tool-Assisted Speedruns
  - Debugger with single-step-back

## What is an emulator?

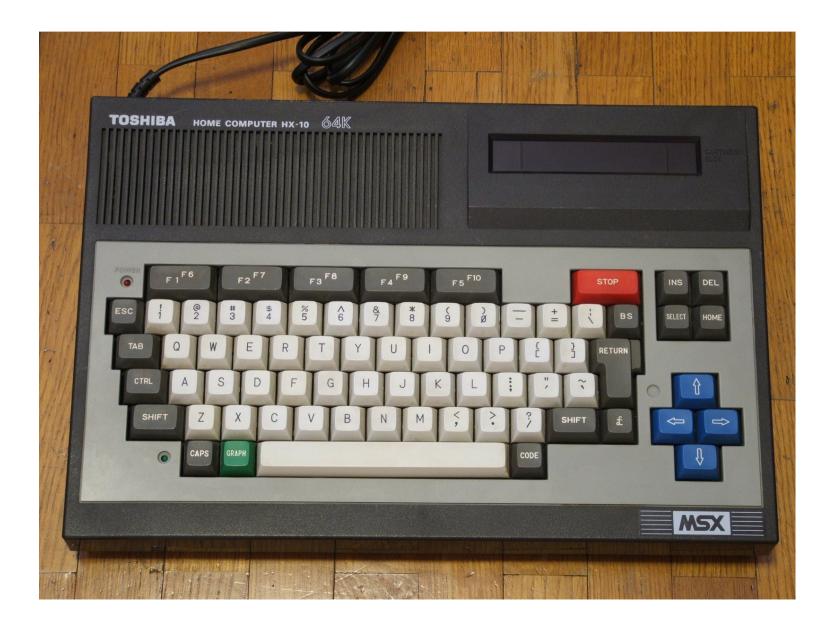
A program to duplicate the behavior of one machine on another machine.

Host: machine running the emulatorGuest: machine being emulated

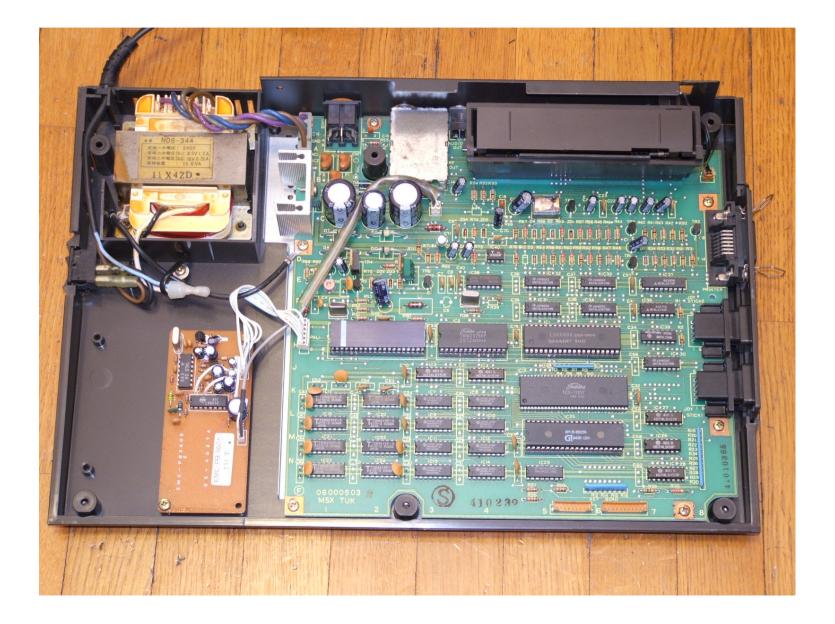
# Types of emulation

Туре:	Host:	Guest:
Full-system	machine A	machine B
Virtualization	machine A	machine A
Virtual machine	machine A	imaginary machine

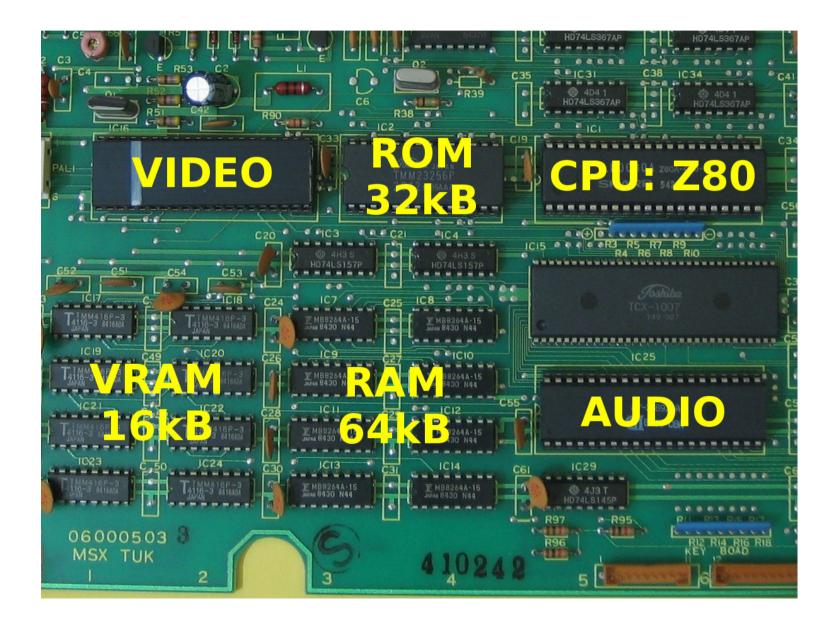
#### Guest machine: MSX



## Guest machine: inside



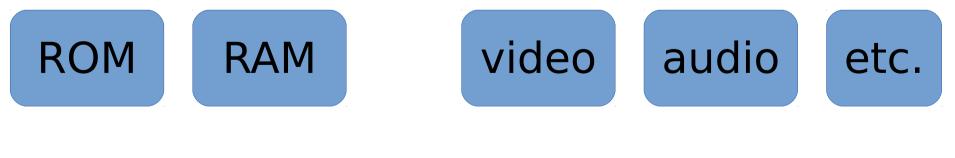
#### Guest machine: components



## Guest machine: components



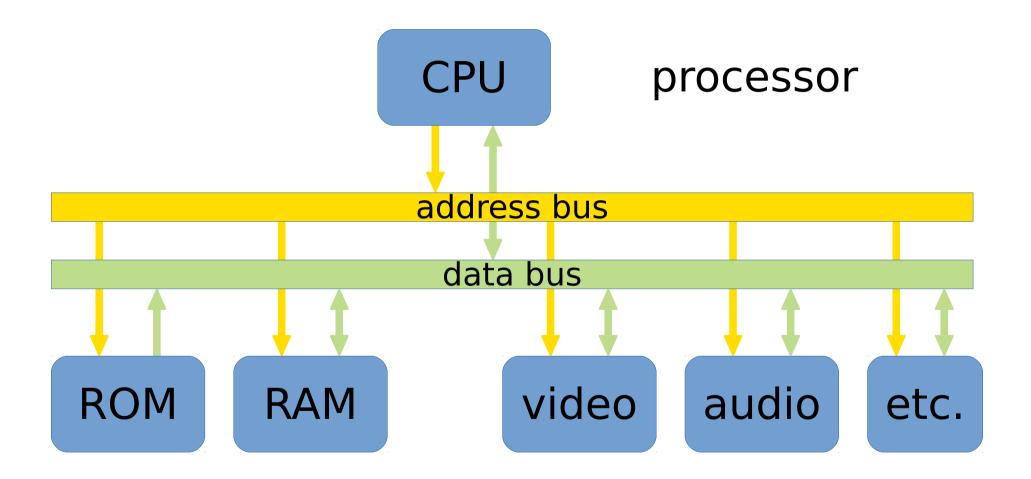
#### processor



memory

peripherals

## Guest machine: components



#### memory

#### peripherals

# CPU: executing an instruction

- Fetch
  - read instruction from memory
  - increase program counter
- Decode
  - figure out what operation to execute
- Execute
  - perform the actual operation

# **CPU: interpreter**

Fetch, decode, execute instruction every time.

Advantage:

• Simple

Disadvantage:

Slow

# CPU: Just-In-Time compiler

Fetch and decode instructions once, generate host code for execution.

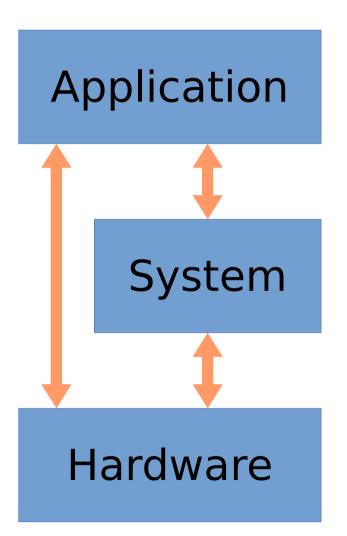
Advantage:

• Fast

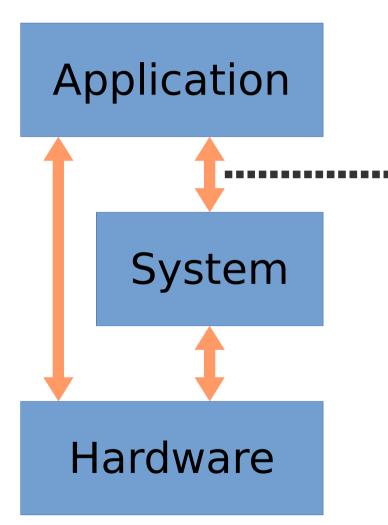
Disadvantages:

- Complex
- Host code generation is not portable

## Peripherals



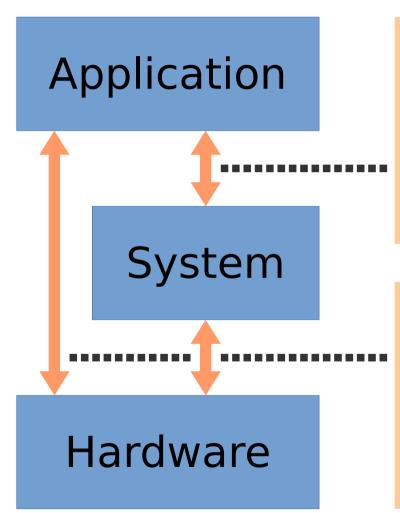
# Peripherals



#### High-level emulation

- fast
- more direct mapping to host system

# Peripherals



#### High-level emulation

- fast
- more direct mapping to host system

#### Low-level emulation

- accurate
- no assumptions about (use of) system layer

# High-level emulation

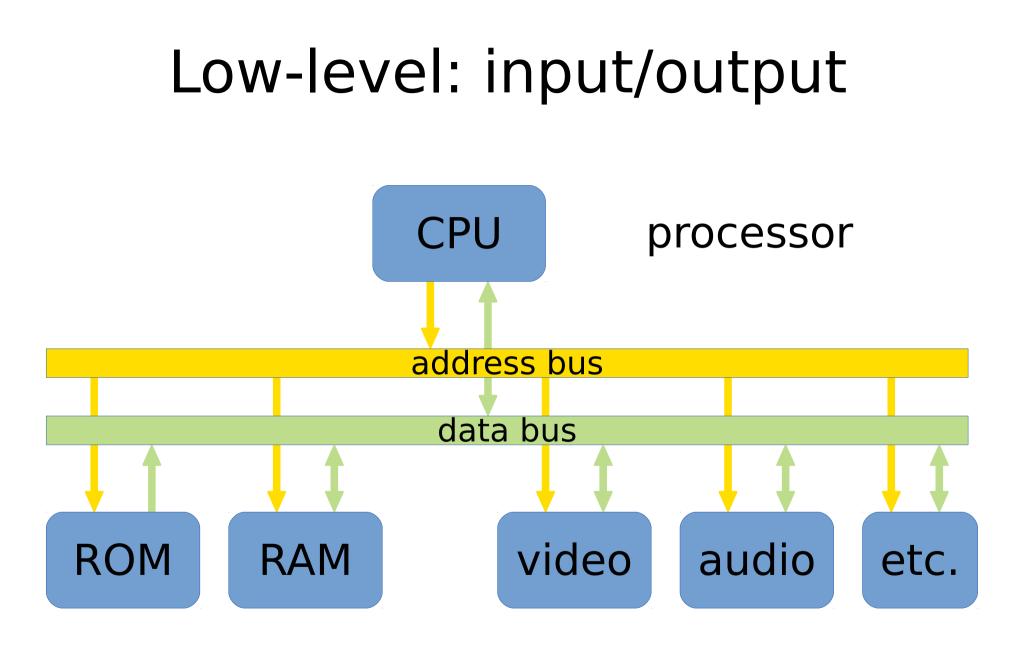
Interpreter: (Fetch; Decode; Execute)\*

- Intercept at fetch: high overhead
- Intercept at decode: patch system with illegal instruction

# High-level emulation

JIT Compiler: Fetch ; Decode ; Execute\*

 Intercept at fetch (code generation): instead of generating code for a system routine, jump to an emulation routine



#### memory

#### peripherals

# Low-level: input/output

Instruction reads from or writes to a peripheral

I/O mapped I/O:

- dedicated in/out instructions
- peripheral selected by I/O port number

Memory mapped I/O:

- general load/store instructions
- peripheral selected by special memory address

# I/O address mapping

<RTC id="Real time clock"> <io base="0xB4" num="2" type="0"/> <io base="0xB5" num="1" type="I"/> </RTC>

<WD2793 id="Memory Mapped FDC"> <mem base="0x7FF8" size="4"/> </WD2793>

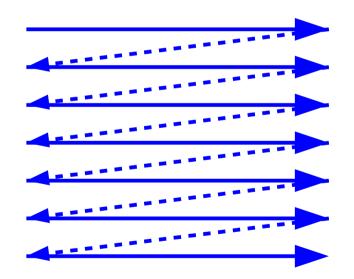
## Peripheral emulation

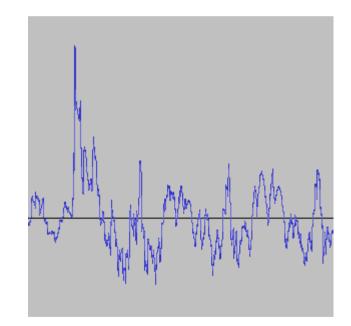
```
Data Device::read(Addr a) {
    return reg[a];
}
```

```
void Device::write(Addr a, Data d) {
   reg[a] = d;
}
```

# Active peripherals

# Change state or produce output in between I/O operations





#### Video

Audio

# Multi-threading

Emulate active peripheral in host thread

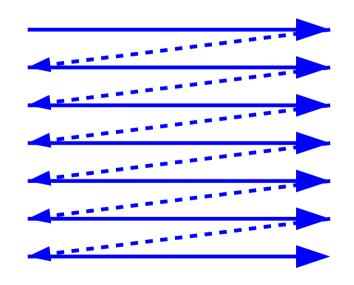
- Low timing accuracy
- Synchronization is expensive

# Static interleaved execution

Emulate active peripheral every N guest clock ticks

Emulate video chip

- once per frame,
- once per line,
- once per pixel?

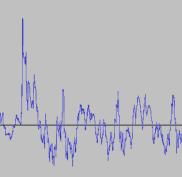


# The challenger

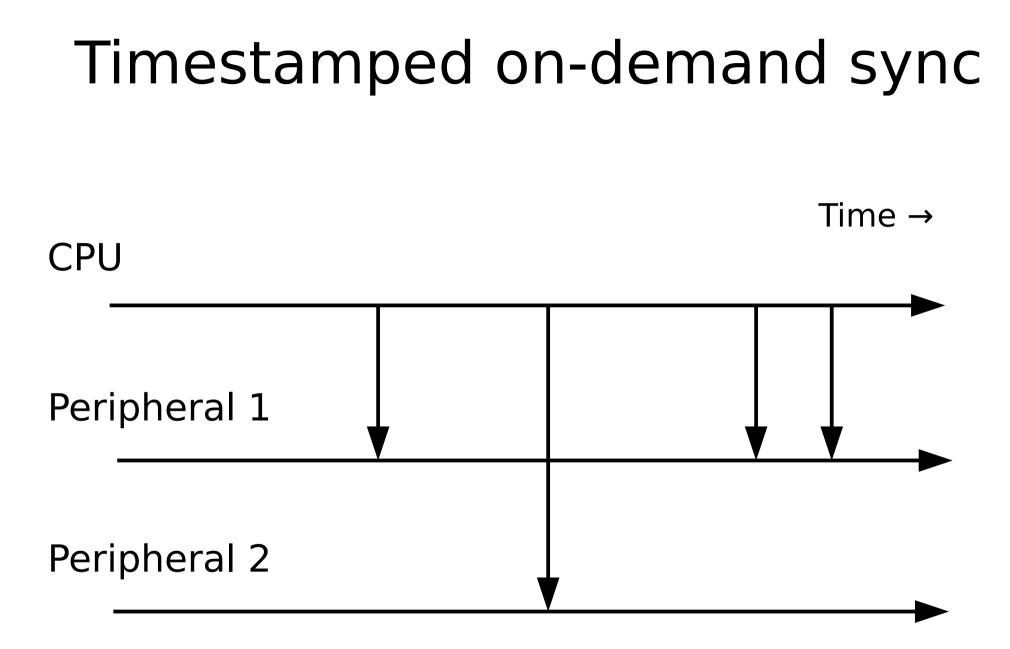
#### Unknown Reality by NOP

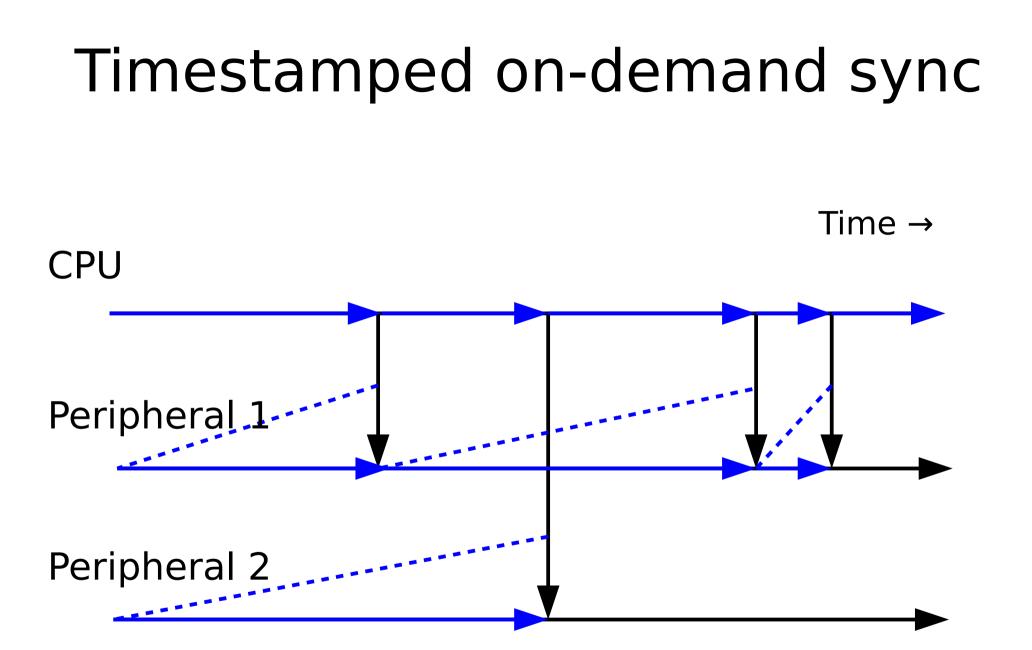


Video



Audio

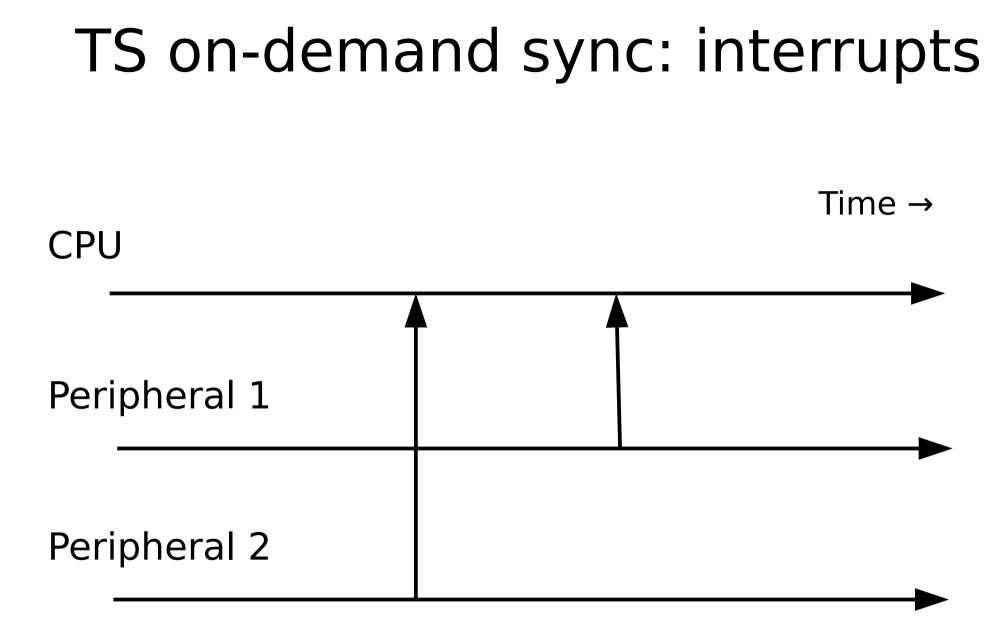




# Timestamped on-demand sync

```
Data Device::read(Addr a, Time t) {
    sync(t);
    return reg[a];
}
```

```
void Device::write(Addr a, Data d, Time t) {
    sync(t);
    reg[a] = d;
}
```



# TS on-demand sync: interrupts

Sync point:

Guest time stamp at which peripheral emulation needs to run

- Peripheral registers sync point at predicted time of interrupt request
- CPU will execute until first sync point, then corresponding peripheral runs and can raise interrupt request

#### Determinism

old state + input  $\rightarrow$  new state

Deterministic:

New state depends on **nothing else** 

# Determinism: why?

old state + input  $\rightarrow$  new state

Deterministic emulation is **reproducible**:

- helps with reproducing bugs
- required for accurate emulation
- enables emulator-only features

## Determinism: how?

old state + input  $\rightarrow$  new state

Non-determinism comes from host:

- if emulation result depends on host state
- if host timing is relevant

## Determinism: how?

old state + input  $\rightarrow$  new state

To eliminate non-determinism:

- model use of host state as an input
- do all timing using guest time stamps

## Determinism: random?

old state + input  $\rightarrow$  new state

Pseudo random: only looks random to casual observer

Real random: always comes from input

### Determinism: replay

#### old state + input $\rightarrow$ new state

state snapshot + input recording ↓ reproducible replay

## Speedruns

Complete a game as fast as possible:

- careful planning
- glitch abuse
- execution skill

Goals:

- challenge
- entertainment

Complete a game as fast as possible:

- careful planning
- glitch abuse
- execution skill use of emulation tools

Goals:

- challenge
- entertainment
- find limits

Emulation tools:

- per-frame recording
- re-recording
- disassembly
- luck manipulation
- algorithmic input generation

Demo

TASVideos – tasvideos.org

Video archive:

- rendered movies
- input recordings

Community:

- resources
- discussion

# Debugger with single-step-back

- Input recording in memory: replay to any point in history
- Regular snapshots: replay **quickly** to any point in history
- Replay until timestamp: replay quickly to any point in history and stop there
- Combined: replay until just before previous instruction

# Debugger with single-step-back

Demo

# Conclusions

- Many different implementation options
- Often trade-off between accuracy and execution speed – but not always!
- Determinism + input recording  $\rightarrow$  replay
- Emulation can provide unique ways to analyze system behavior