从mov指令到仙剑：通过NEMU构建简单完整的计算机系统

余子濠
中科院计算所

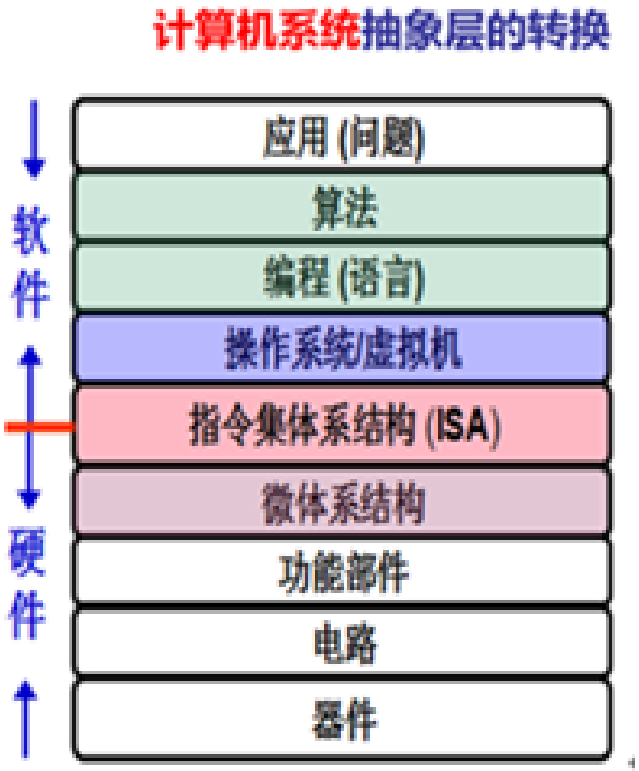
2019.11.22@杭州

提纲

- 引言
- 实验内容
- 设计原则
- 一些讨论

来自本科教育的灵魂拷问

- 系统方向的两个终极问题
  - 程序如何在计算机上运行?
    - How to make things work?
  - 程序如何在计算机上高效运行?
    - How to make things work better?



- 终极问题1:

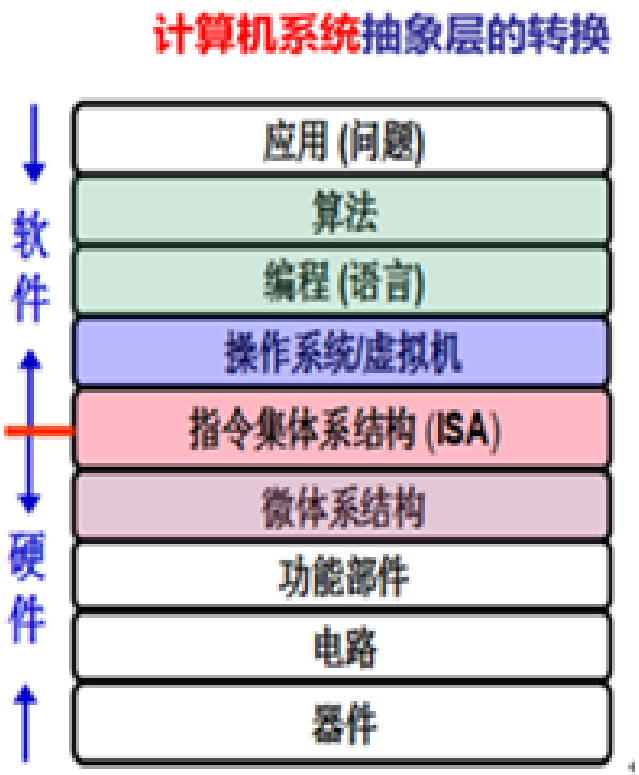
```
[23:46:56 ~]$ vim hello.c
[23:47:27 ~]$ gcc hello.c -o hello
[23:47:39 ~]$ ./hello
Hello World!
[23:47:42 ~]$
```

- 这个过程计算机都做了些什么?

需要一个打通全栈的实验

- 课程各讲各的知识，没有一门课把知识串起来
  - 哪怕是用很简单的例子，但覆盖从程序到机器
- 全靠学生学完后自己悟
  - 悟性不太好就不知道他们想什么了

- **只有亲自实现一个完整的计算机系统，并在其上运行真实的程序，才能明白系统栈每一个层次之间的关系，才会对“程序如何在计算机上运行”有深刻的认识**



PA简介

- 南京大学大二上“计算机系统基础”(袁春风)的课程项目
- Programming Assignment
  - 指导学生实现一个功能完备的模拟器NEMU(NJU EMUlator)
    - 支持x86(教学版子集)/mips32/riscv32
  - 最终在NEMU上运行真实游戏“仙剑奇侠传”
  - 揭示“程序在计算机上运行”的基本原理

- PA包括一个准备实验以及6部分连贯的实验内容:
  - PA0: 开发环境配置(准备实验)
  - PA1: 简易调试器
  - PA2: 冯诺依曼计算机系统
  - PA3: 批处理系统
  - PA4: 分时多任务
  - PA5: 程序性能优化(选做)

PA资源

- 实验平台与工具
  - GNU/Linux + gcc + C
  - 其它工具: gdb, make, git
- 实验讲义
  - https://nju-projectn.github.io/ics-pa-gitbook
- 框架代码
  - https://github.com/NJU-ProjectN/ics-pa

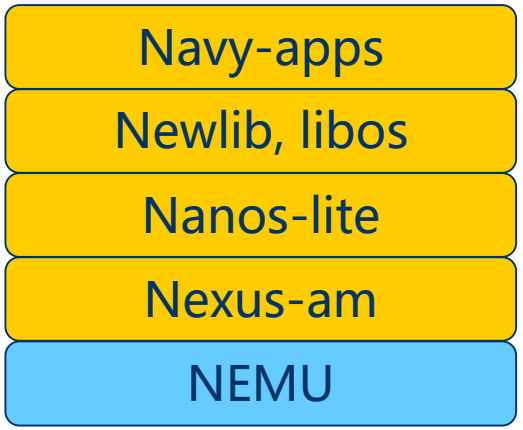
- 无需编写硬件代码
  - 因此无需FPGA开展实验

提纲

- 引言
- 实验内容
- 设计原则
- 一些讨论

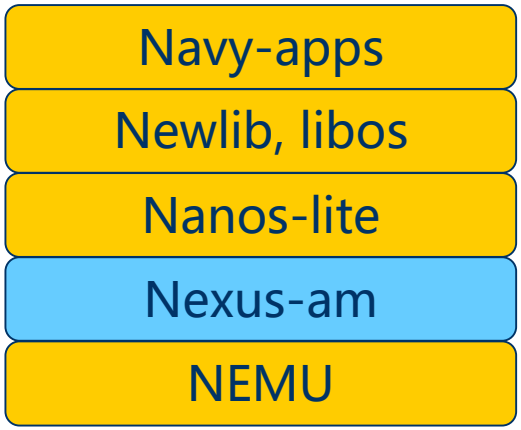
PA组成-NEMU全系统模拟器

- 简易调试器(Monitor) - 基础设施
  - 单步执行, 打印寄存器/内存, 表达式求值, 监视点
- CPU
  - 完整的指令周期: 取指, 译码, 执行
  - 异常处理模块
  - 支持分页的MMU
- 内存
- 设备
  - 时钟, 键盘, VGA, 串口(功能经过简化)



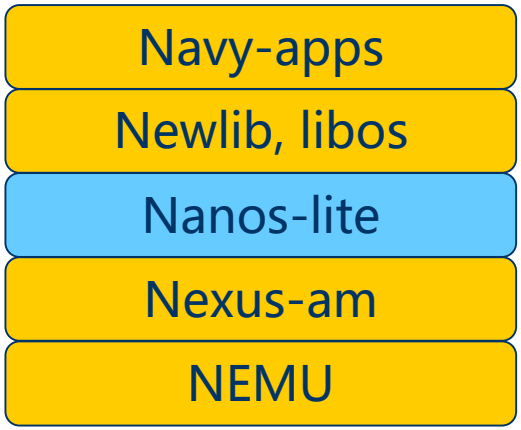
PA组成-Nexus-AM ISA抽象层

- 把机器功能抽象成C语言API, 向程序屏蔽ISA细节
- TRM 计算抽象
  - _putc() 输出一个字符
  - _halt() 终止程序运行
- IOE 输入输出抽象
  - uptime() 返回当前时间
  - read_key() 返回按键
  - draw_rect() 在屏幕上绘制矩形像素
- CTE 上下文管理抽象
  - 上下文保存/恢复
  - 事件处理回调函数
- VME 虚存抽象
  - _protect() 创建虚拟地址空间
  - _map() 添加va到pa的映射关系



PA组成-Nanos-lite简易多任务OS

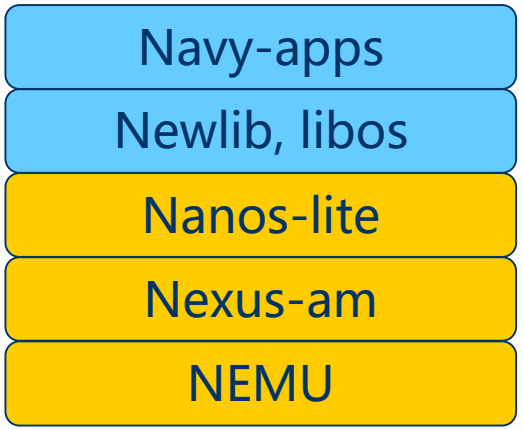
- Nanos(Nanjing U OS)简化版, 能支撑真实程序
  - ELF加载器
  - 中断/异常事件分发
  - 6+2个系统调用
    - open, read, write, lseek, close, brk
    - exit, exec
  - 简易VFS
    - 文件数量, 大小皆固定, 没有目录
    - 文件系统基于ramdisk, 无需持久化
    - 3个设备文件(映射到设备抽象层)
      - /dev/fb, /dev/events, /proc/dispinfo
  - 设备抽象层
  - 简易分页存储管理 (顺序分配, 不释放)
  - 两个进程的简易轮转调度



库和应用程序Navy-apps

- Newlib – C库, 支持POSIX标准
- libos – 系统调用接口

- Navy-apps – 应用程序集
  - hello, 仙剑奇侠传, LiteNES (PA用)
  - Lua解释器, busybox套件(shell, vi)
  - NWM窗口管理器, Nterm终端
  - NCC编译器



PA1 - 简易调试器
[TRM]

```
No image is given. Use the default build-in image.
Welcome to NEMU!
[src/monitor/monitor.c,23,welcome] Build time: 17:39:41, Jul 26 2017
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026
```

只有mov指令的程序

学生任务: 实现简易调试器

理解最简单计算机的基本构成

- 单步执行 - PC
- 打印寄存器 - Reg
- 扫描内存 - Mem

基础设施(帮助调试), 复习C语言程序设计

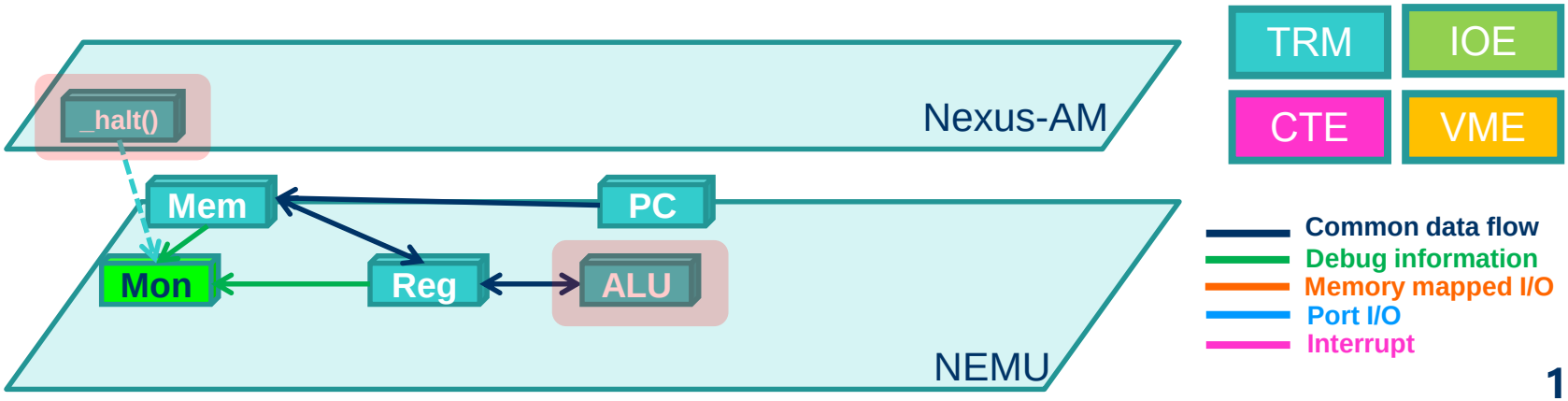
- 表达式求值(正则表达式+递归)
- 监视点(指针, 链表)

PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集)]

学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

矩阵乘法

```
[src/monitor/cross-check/cross-check.c,97,init_check] Connect to QEMU successfully
The image is /home/yuzihao/NJUCS-ComputerSystemLab/nexus-am/tests/cputest/build/matrix-mul-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,23,welcome] Build time: 17:39:41, Jul 26 2017
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x0010005e
```

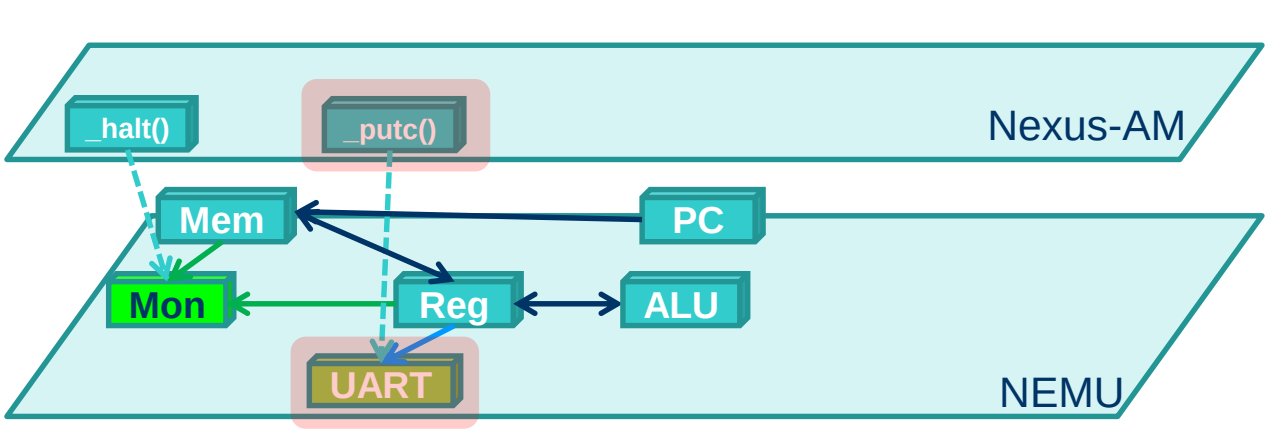


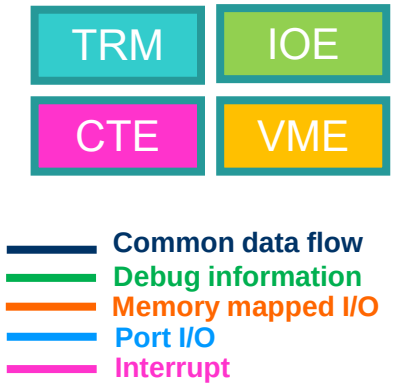
PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]

```
The image is /home/yuzihao/NJUCS-Computer
Welcome to NEMU!
[src/monitor/monitor.c,23,welcome] Build
For help, type "help"
(nemu) c
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
nemu: HIT GOOD TRAP at eip = 0x0010006e
```

学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

添加IOE
- UART(端口I/O) -> _putc() -> hello程序



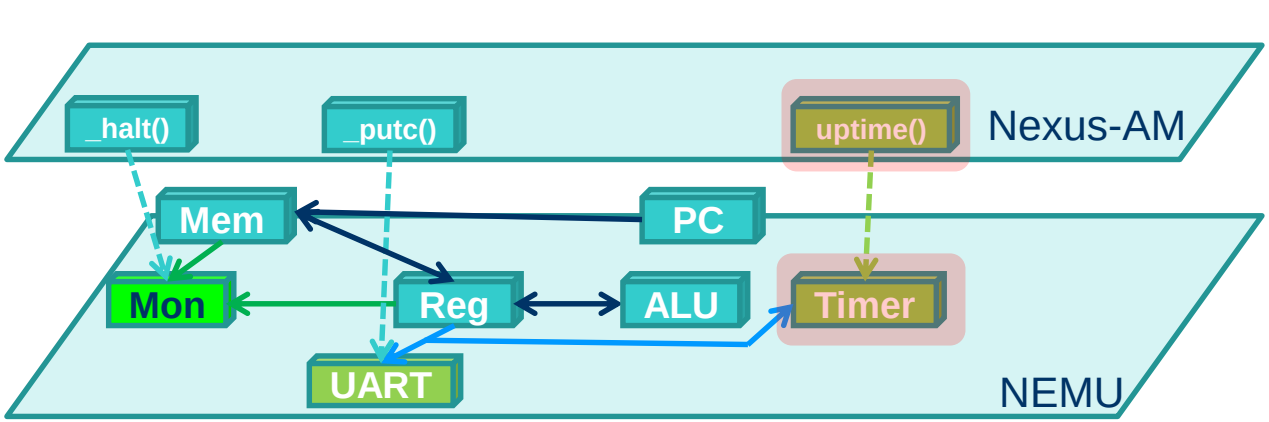


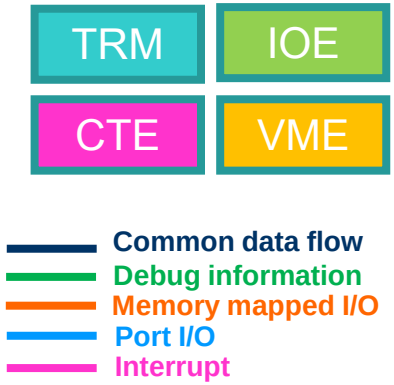
PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]

```
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time:
For help, type "help"
(nemu) c
1 second.
2 seconds.
3 seconds.
4 seconds.
5 seconds.
6 seconds.
7 seconds.
8 seconds.
9 seconds.
10 seconds.
11 seconds.
12 seconds.
13 seconds.
```

学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

添加IOE
- UART(端口I/O) -> _putc() -> hello程序
- Timer -> uptime() -> timertest程序





PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]

```
The image is /home/yuzihao/NJUCS-ComputerSystemLab/nexus
Welcome to NEMU!
[src/monitor/monitor.c,23,welcome] Build time: 16:34:25,
For help, type "help"
(nemu) c
[qsort] Quick sort: * Passed.
  min time: 2891 ms [190]
[queen] Queen placement: * Passed.
  min time: 5205 ms [99]
[bf] Brainf**k interpreter: * Passed.
  min time: 29526 ms [88]
[fib] Fibonacci number: * Passed.
  min time: 57489 ms [49]
[sieve] Eratosthenes sieve: * Passed.
  min time: 48906 ms [86]
[15pz] A* 15-puzzle search: * Passed.
  min time: 9257 ms [62]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 8477 ms [159]
[lzip] Lzip compression: * Passed.
  min time: 24113 ms [109]
[ssort] Suffix sort: * Passed.
  min time: 4714 ms [125]
[md5] MD5 digest: * Passed.
  min time: 45426 ms [43]
================================================
MicroBench PASS         101 Marks
                vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x00100032
```

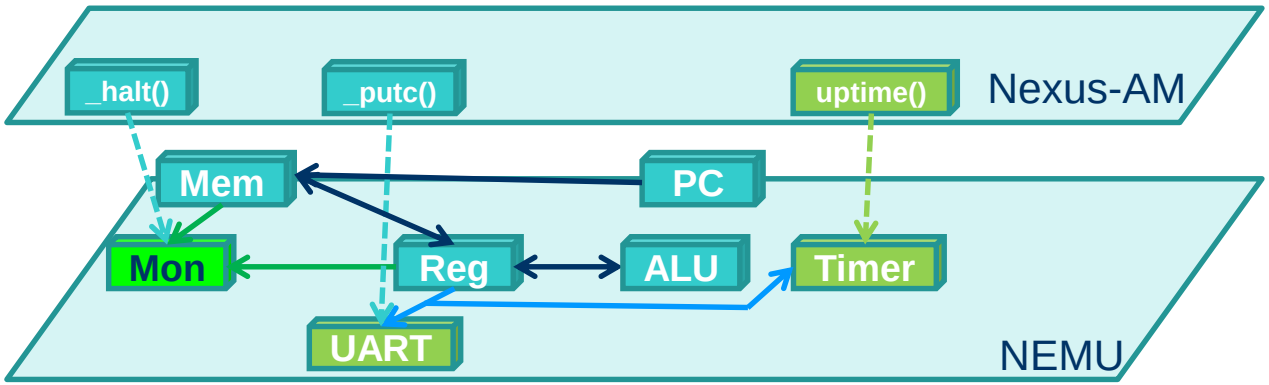
学生任务:

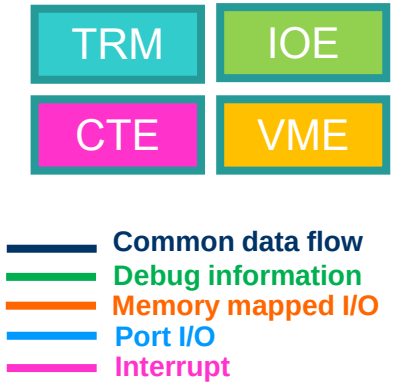
添加常用指令 -> cputest测试集

- 用RTL实现指令

添加IOE

- UART(端口I/O) -> _putc() -> hello程序
- Timer -> uptime() -> timertest程序
  - 运行microbench



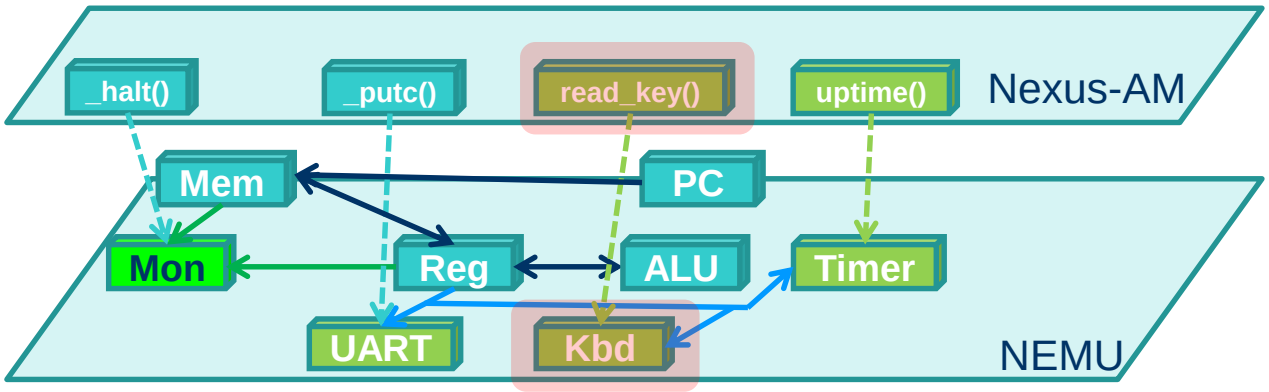


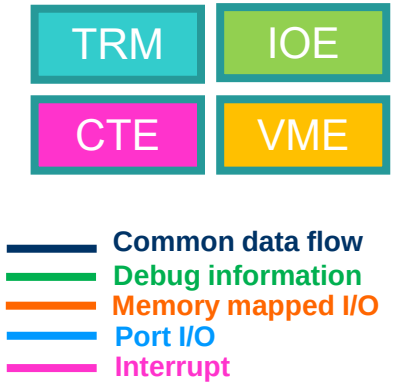
PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]

```
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome]
For help, type "help"
(nemu) c
Get key: 15 1 down
Get key: 15 1 up
Get key: 43 A down
Get key: 49 J down
Get key: 43 A up
Get key: 45 D down
Get key: 49 J up
Get key: 45 D up
Get key: 54 RETURN down
Get key: 54 RETURN up
Get key: 70 SPACE down
Get key: 70 SPACE up
```

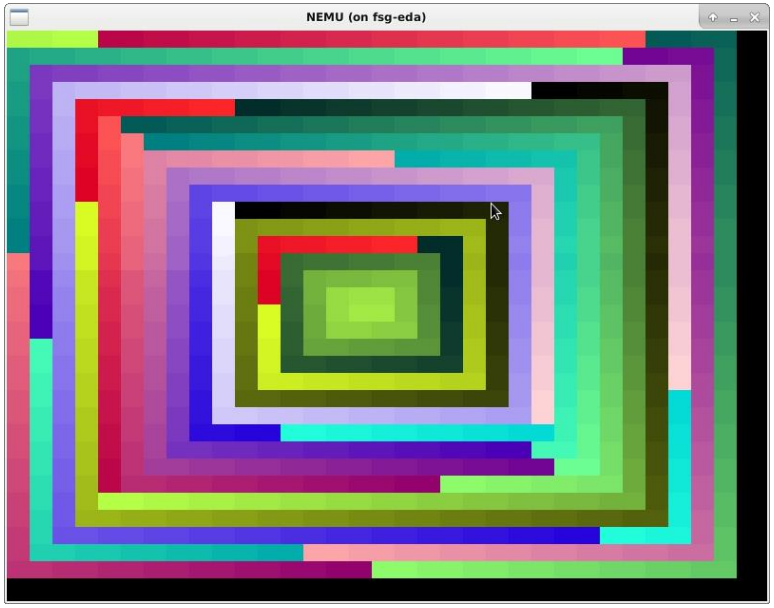
学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

添加IOE
- UART(端口I/O) -> _putc() -> hello程序
- Timer -> uptime() -> timertest程序
  - 运行microbench
- Kbd ->read_key() -> keytest程序



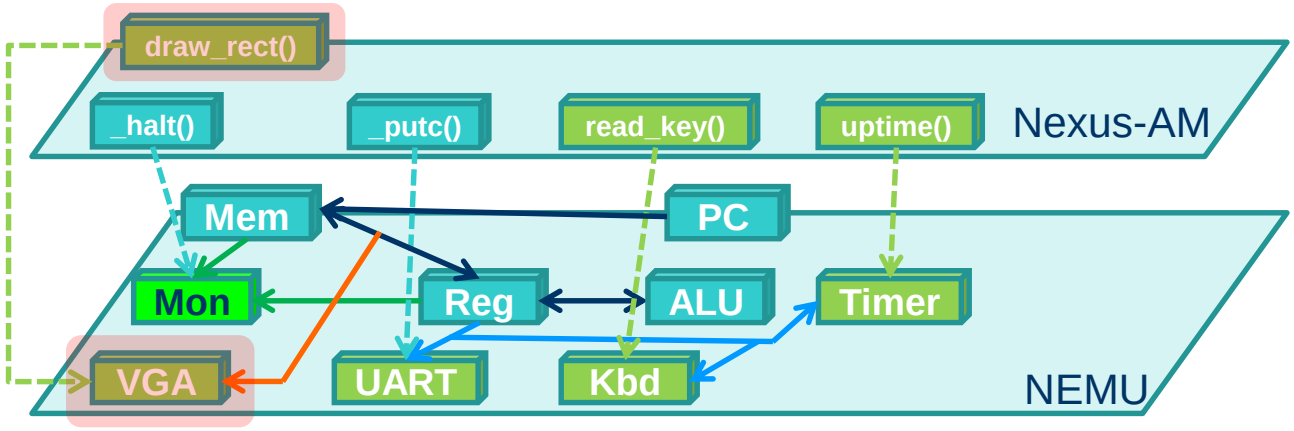


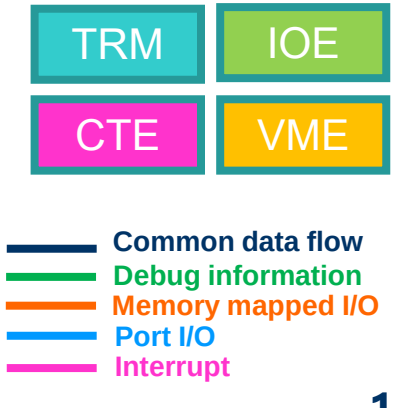
PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]



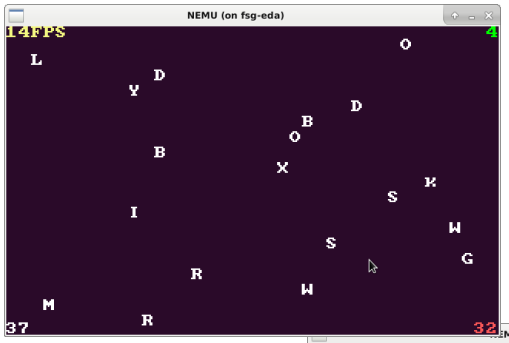
学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

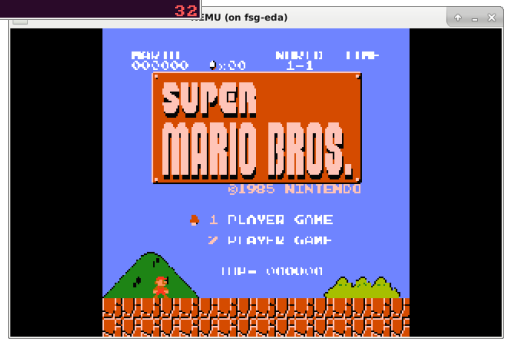
添加IOE
- UART(端口I/O) -> _putc() -> hello程序
- Timer -> uptime() -> timertest程序
  - 运行microbench
- Kbd ->read_key() -> keytest程序
- VGA(内存映射I/O) -> draw_rect() -> videotest程序





PA2 - 冯诺依曼计算机系统
[TRM(x86/riscv32/mips32指令集) + IOE]

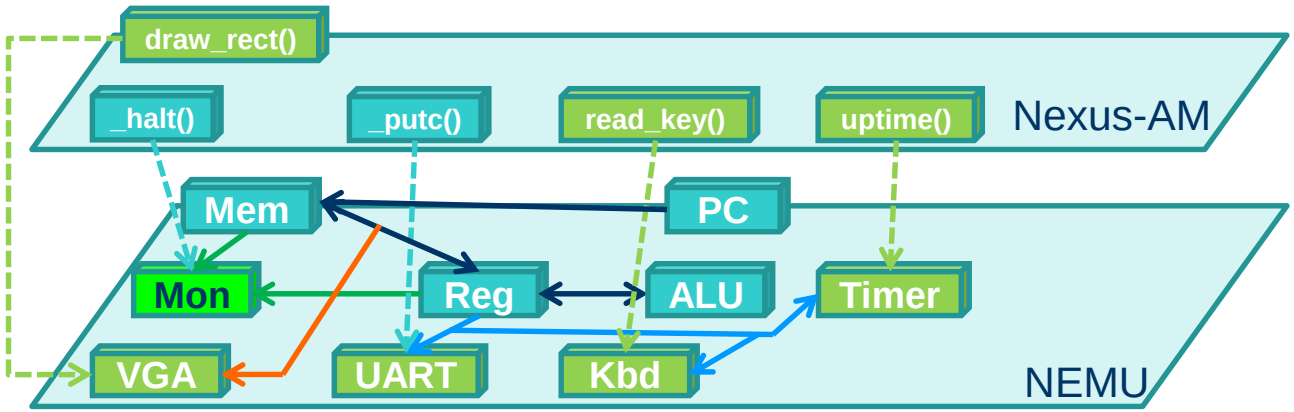


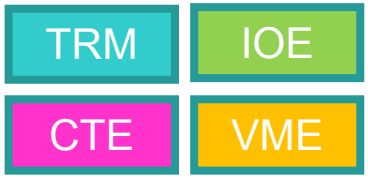


学生任务:
添加常用指令 -> cputest测试集
- 用RTL实现指令

添加IOE
- UART(端口I/O) -> _putc() -> hello程序
- Timer -> uptime() -> timertest程序
  - 运行microbench
- Kbd ->read_key() -> keytest程序
- VGA(内存映射I/O) -> draw_rect() -> videotest程序

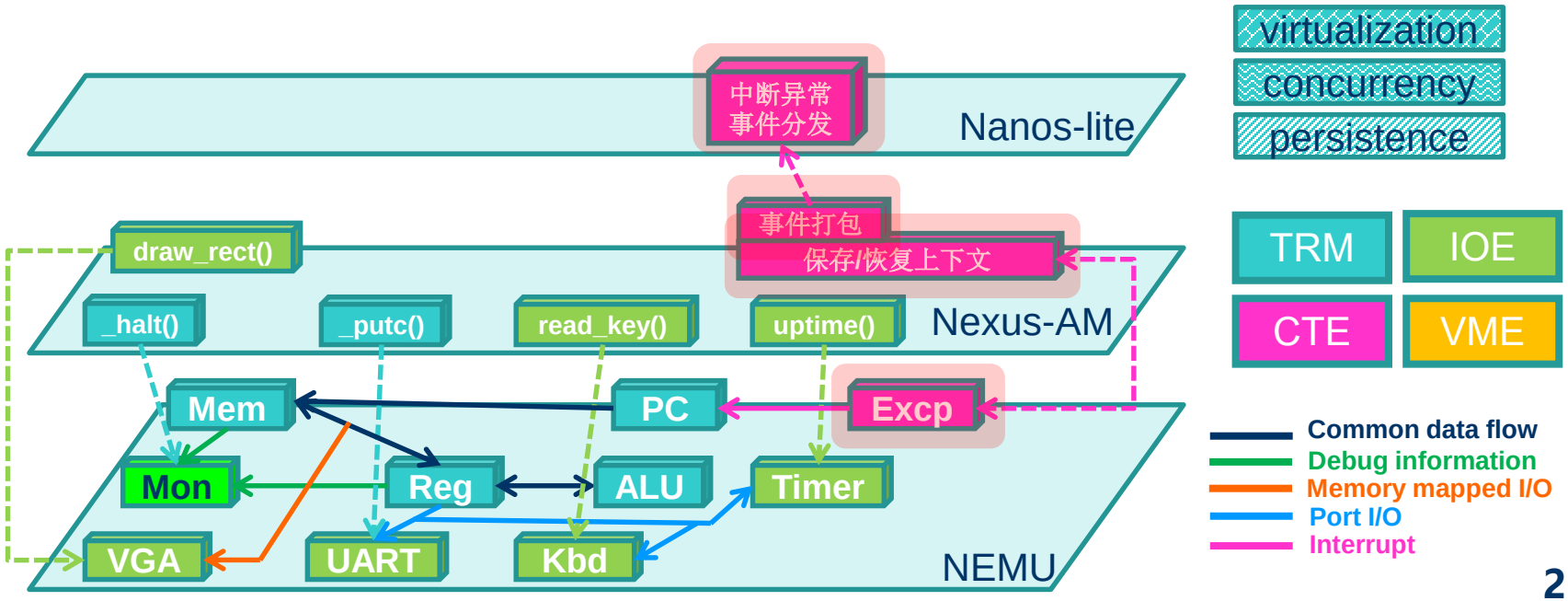
运行打字游戏, LiteNES





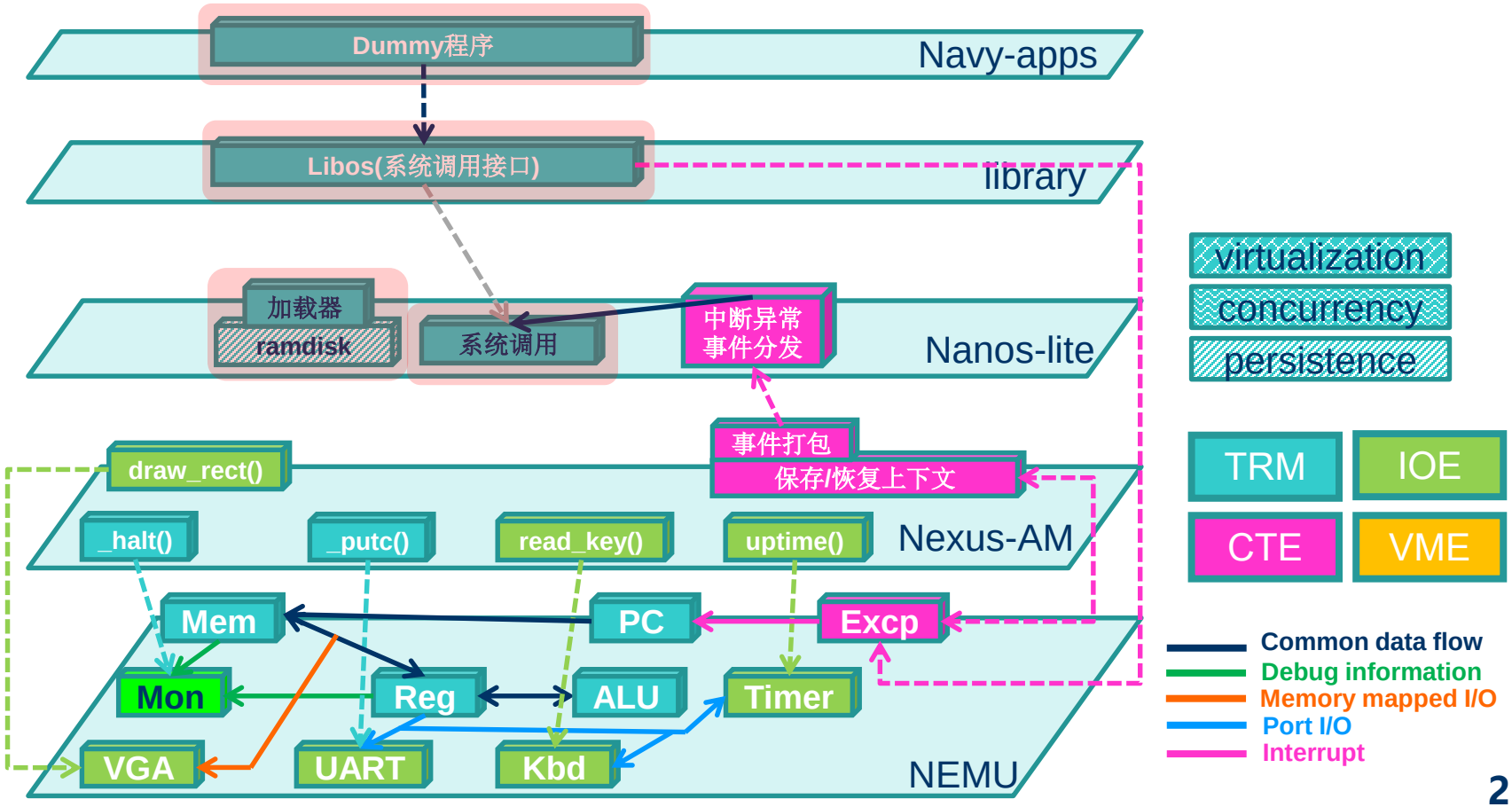
PA3 - 批处理系统
[TRM + IOE + CTE]

学生任务:
添加CTE -> _yield()



PA3 - 批处理系统
[TRM + IOE + CTE]

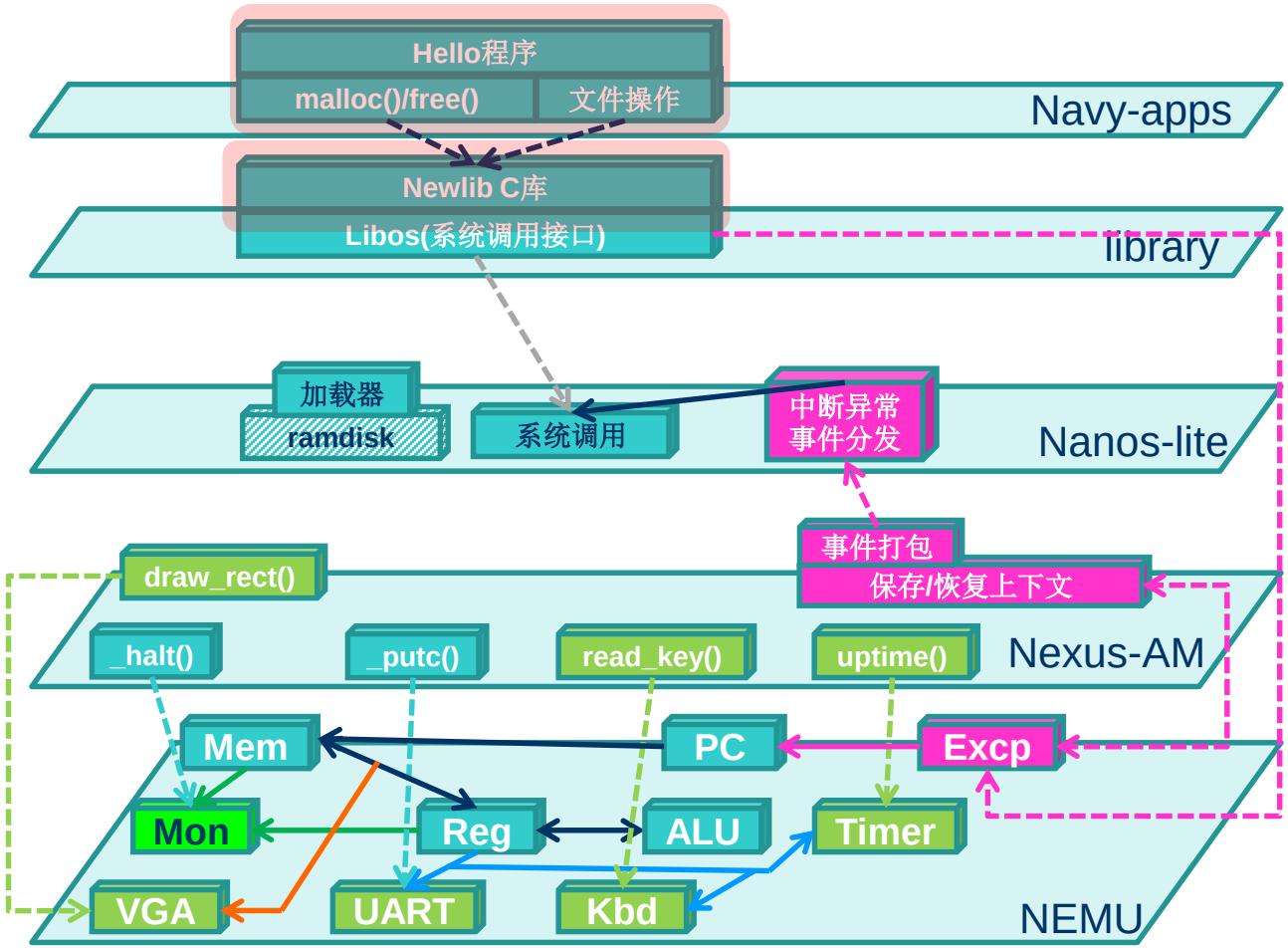
学生任务:
添加CTE -> _yield()
实现简易加载器
->系统调用SYS_yield

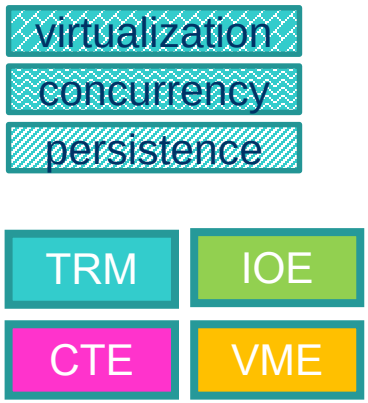


PA3 - 批处理系统
[TRM + IOE + CTE]

学生任务:
添加CTE -> _yield()
实现简易加载器
->系统调用SYS_yield
添加SYS_write
-> hello程序

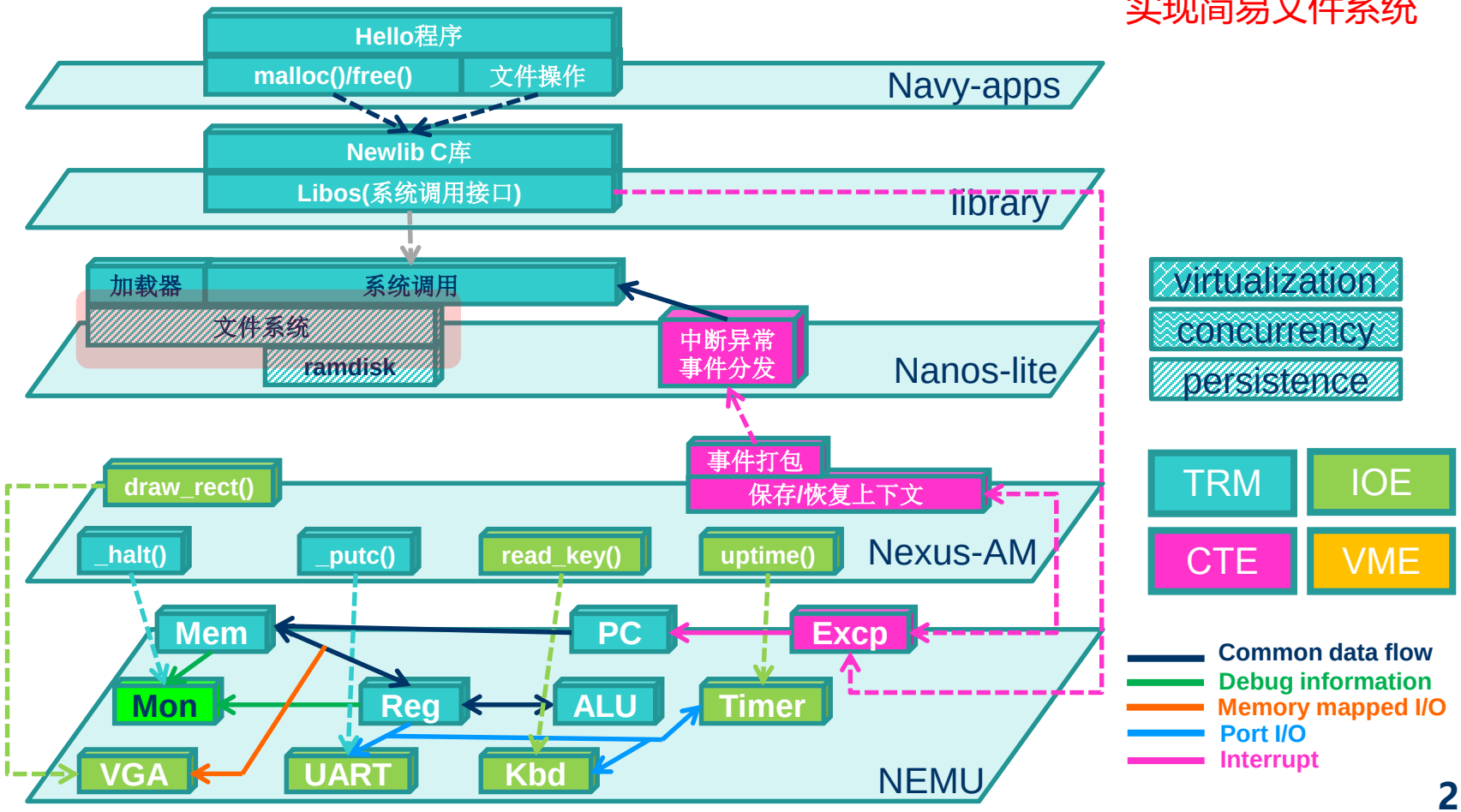
```
Hello World!
Hello World for the 2th time
Hello World for the 3th time
Hello World for the 4th time
Hello World for the 5th time
```





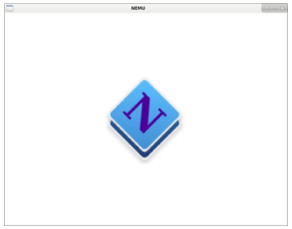
PA3 - 批处理系统
[TRM + IOE + CTE]

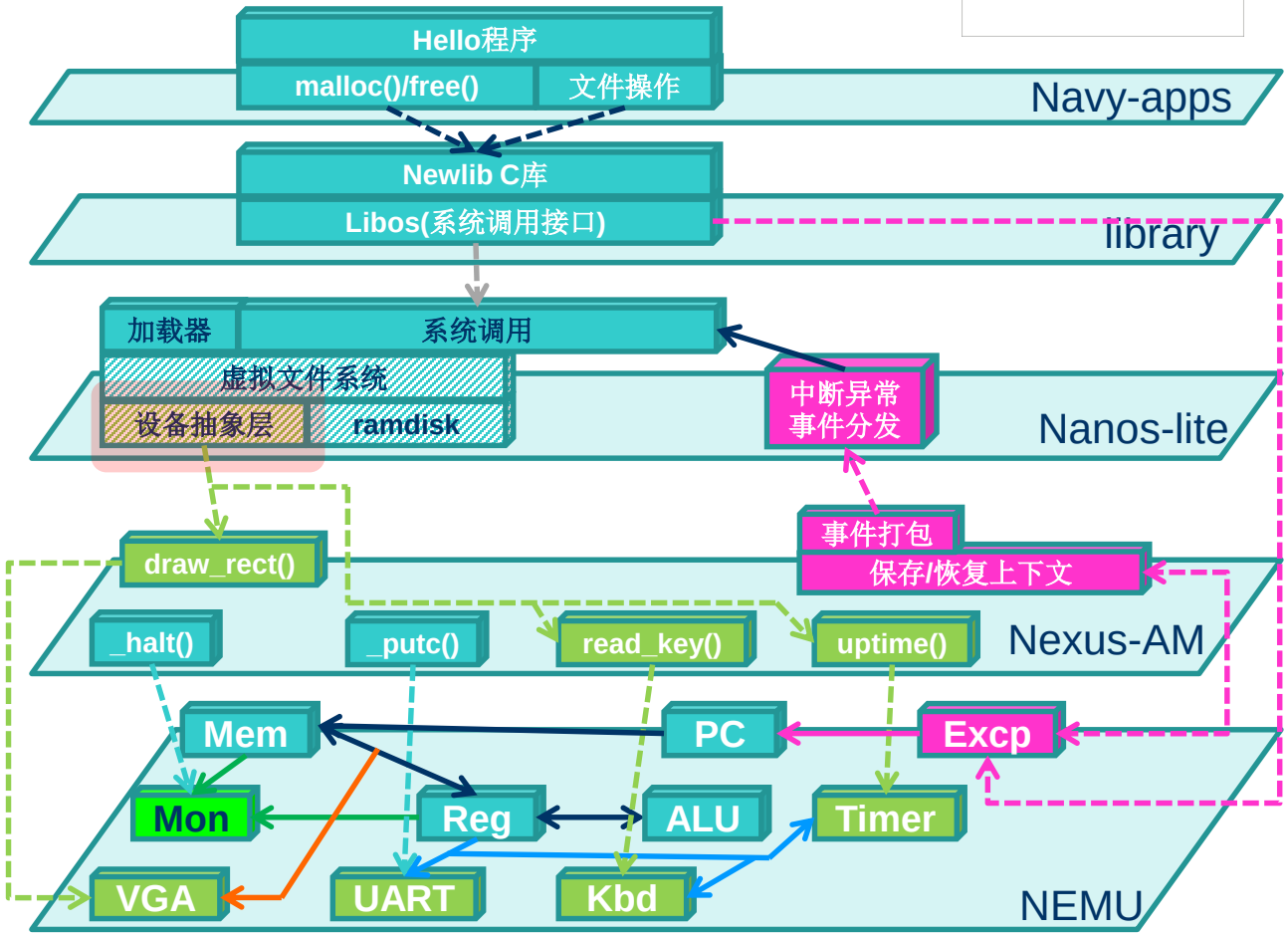
学生任务:
添加CTE -> _yield()
实现简易加载器
->系统调用SYS_yield
添加SYS_write
-> hello程序
实现简易文件系统



PA3 - 批处理系统
[TRM + IOE + CTE]

学生任务:
添加CTE -> _yield()
实现简易加载器
->系统调用SYS_yield
添加SYS_write
-> hello程序
实现简易文件系统
添加VFS和设备抽象层





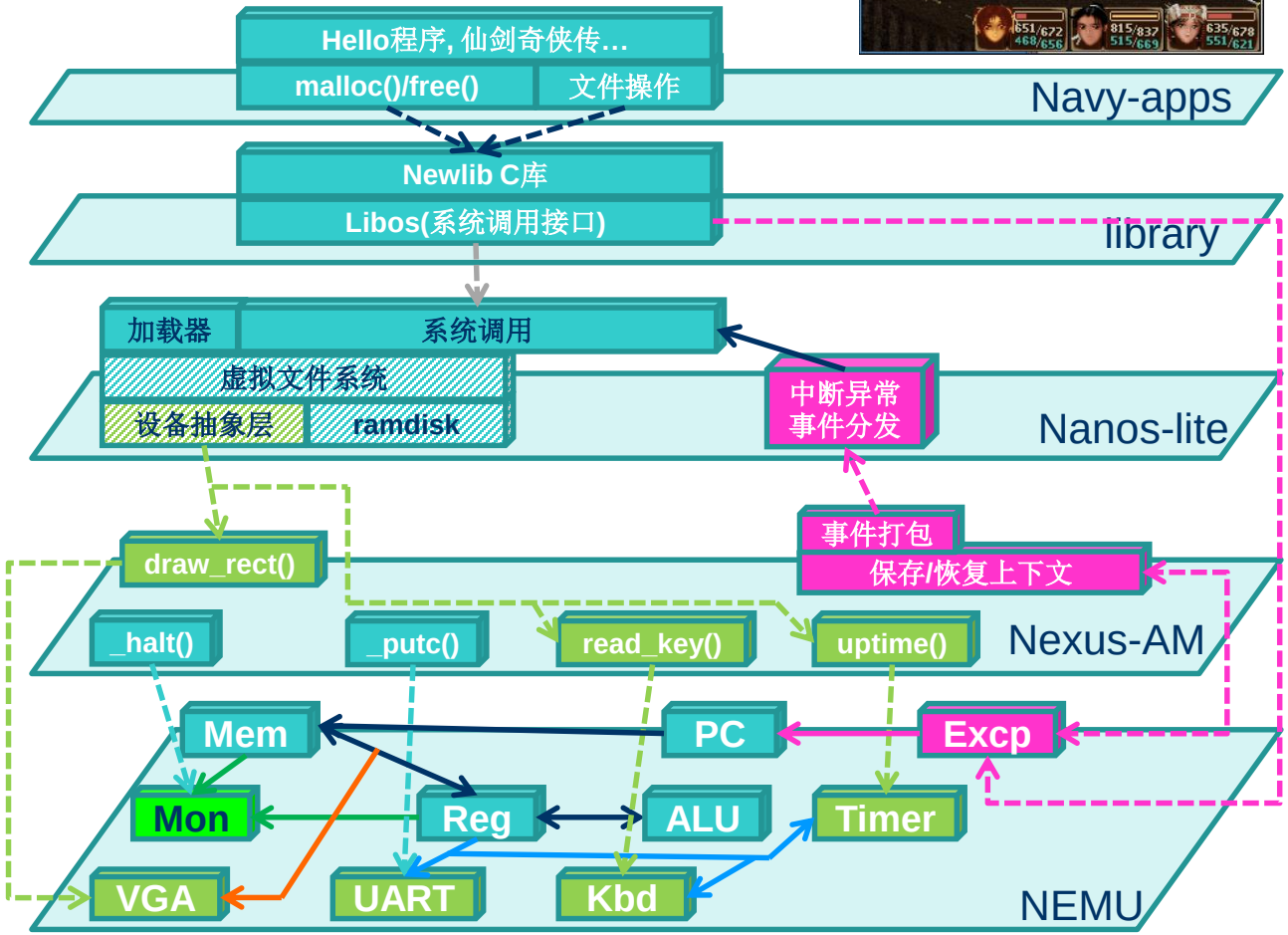
virtualization
concurrency
persistence

TRM
IOE
CTE
VME

Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt

PA3 - 批处理系统
[TRM + IOE + CTE]

学生任务:
添加CTE -> _yield()
实现简易加载器
->系统调用SYS_yield
添加SYS_write
-> hello程序
实现简易文件系统
添加VFS和设备抽象层
运行仙剑奇侠传



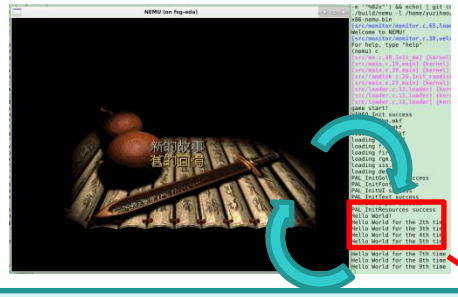
virtualization
concurrency
persistence

TRM
IOE
CTE
VME

Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt

PA4 - 分时多任务
[TRM + IOE + CTE + VME]

学生任务:
实现上下文切换
- 多道程序, 并发运行 hello程序和仙剑



Hello程序, 仙剑奇侠传...
malloc()/free()
文件操作
Navy-apps

Newlib C库
Libos(系统调用接口)
library

```
Hello World!
Hello World for the 2th time
Hello World for the 3th time
Hello World for the 4th time
Hello World for the 5th time
```

加载器
系统调用
虚拟文件系统
设备抽象层
ramdisk
进程调度
进程切换
中断异常
事件分发
Nanos-lite

virtualization
concurrency
persistence

事件打包
上下文切换
保存/恢复上下文
draw_rect()
_halt()
_putc()
read_key()
uptime()
Nexus-AM

TRM
IOE
CTE
VME

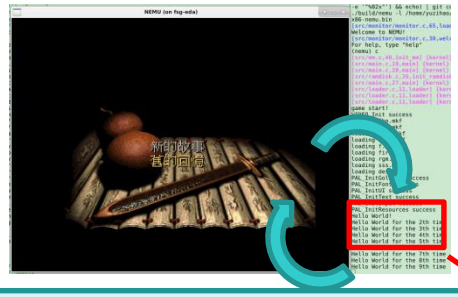
Mem
PC
Excp
Mon
Reg
ALU
Timer
VGA
UART
Kbd
NEMU

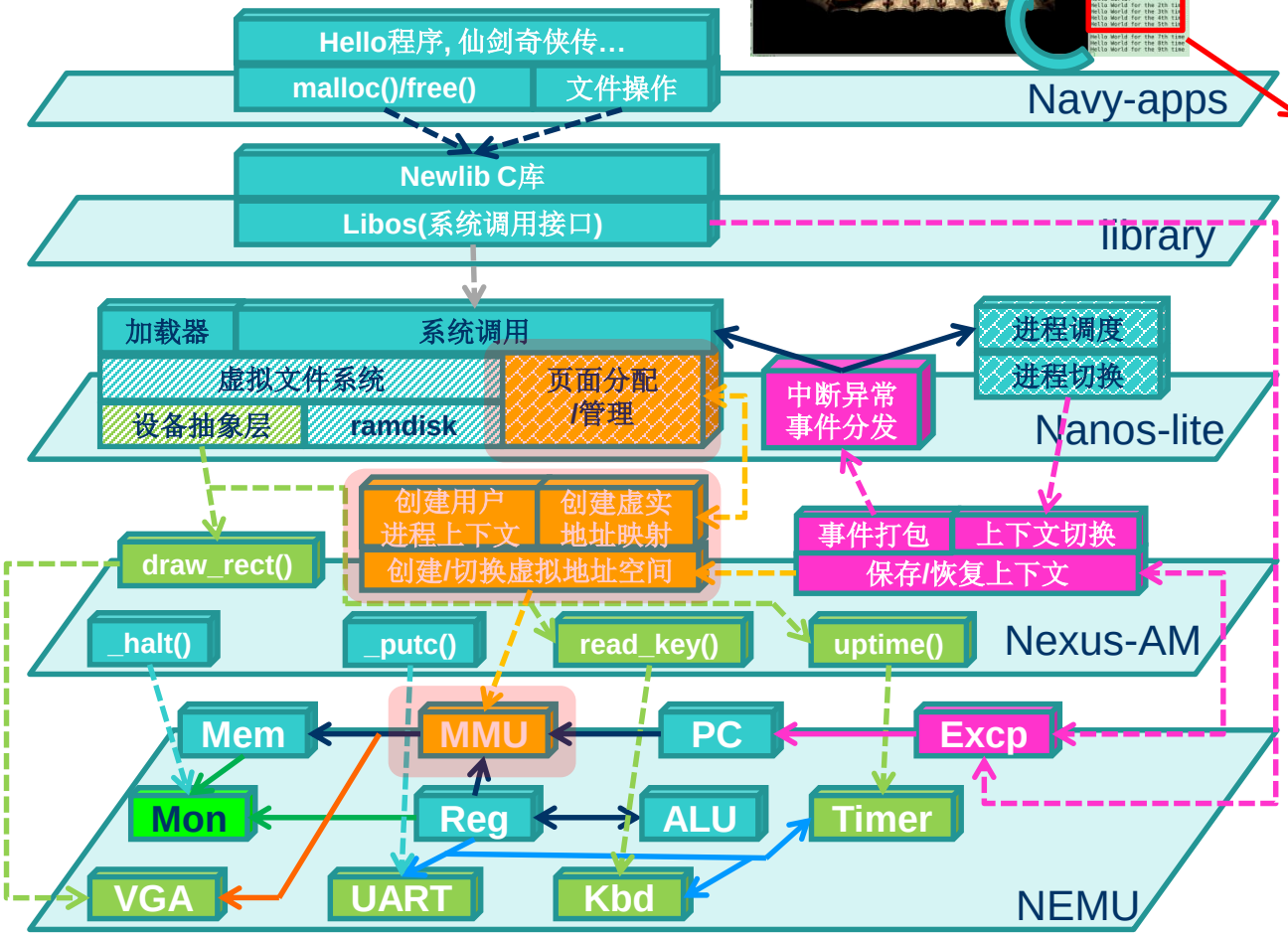
Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt

PA4 - 分时多任务
[TRM + IOE + CTE + VME]

学生任务:
实现上下文切换
- 多道程序, 并发运行hello程序和仙剑

实现VME -> 支持虚存管理的多道程序






```
Hello World!
Hello World for the 2th time
Hello World for the 3th time
Hello World for the 4th time
Hello World for the 5th time
```

virtualization
concurrency
persistence

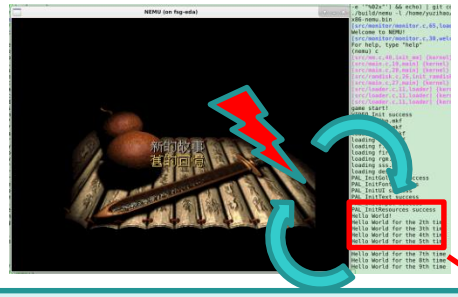
TRM IOE
CTE VME

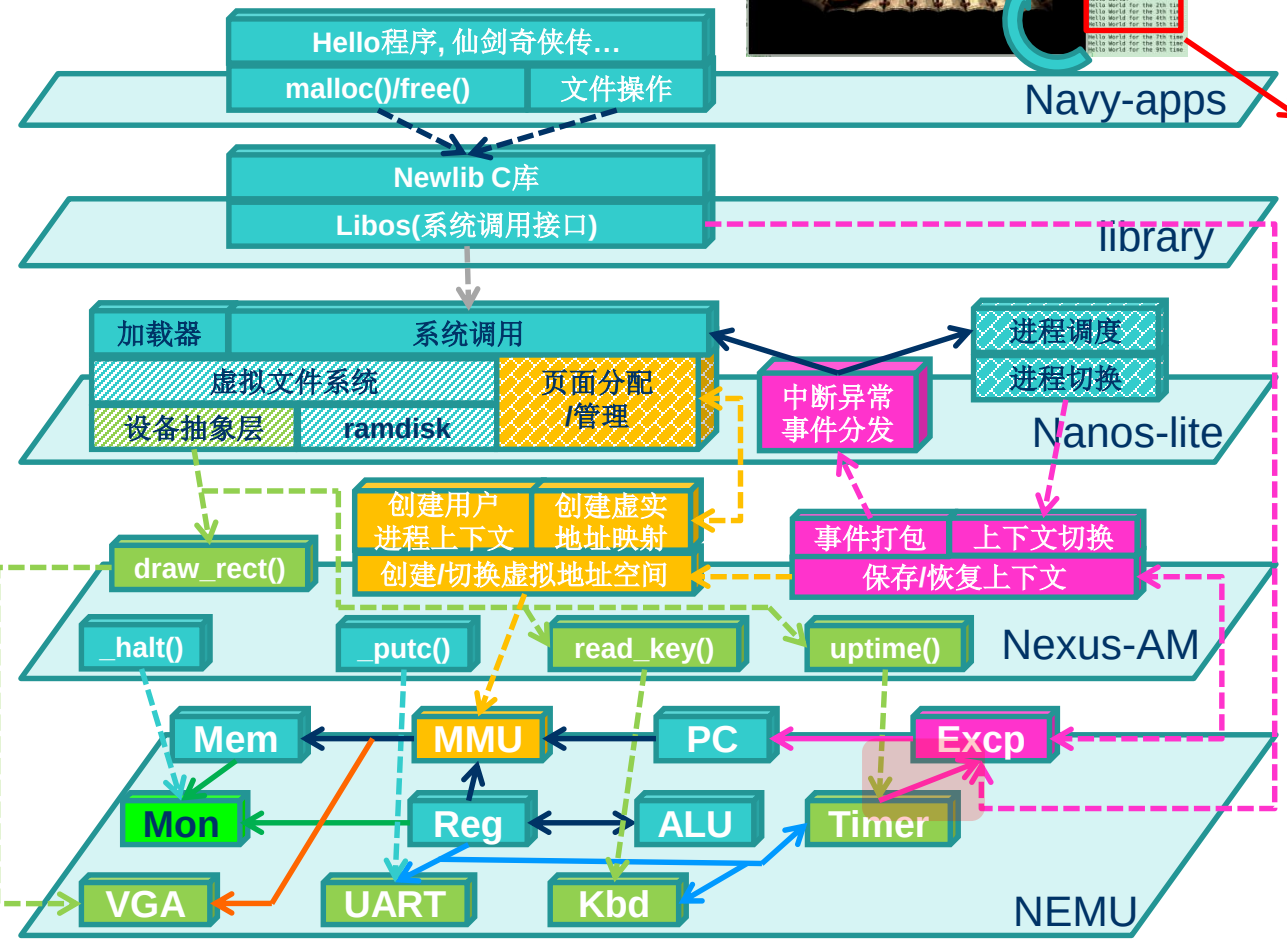
Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt

PA4 - 分时多任务
[TRM + IOE + CTE + VME]

学生任务:
实现上下文切换
- 多道程序, 并发运行hello程序和仙剑

实现VME -> 支持虚存管理的多道程序
添加中断
-> 抢占式分时多任务






```
Hello World!
Hello World for the 2th time
Hello World for the 3th time
Hello World for the 4th time
Hello World for the 5th time
```

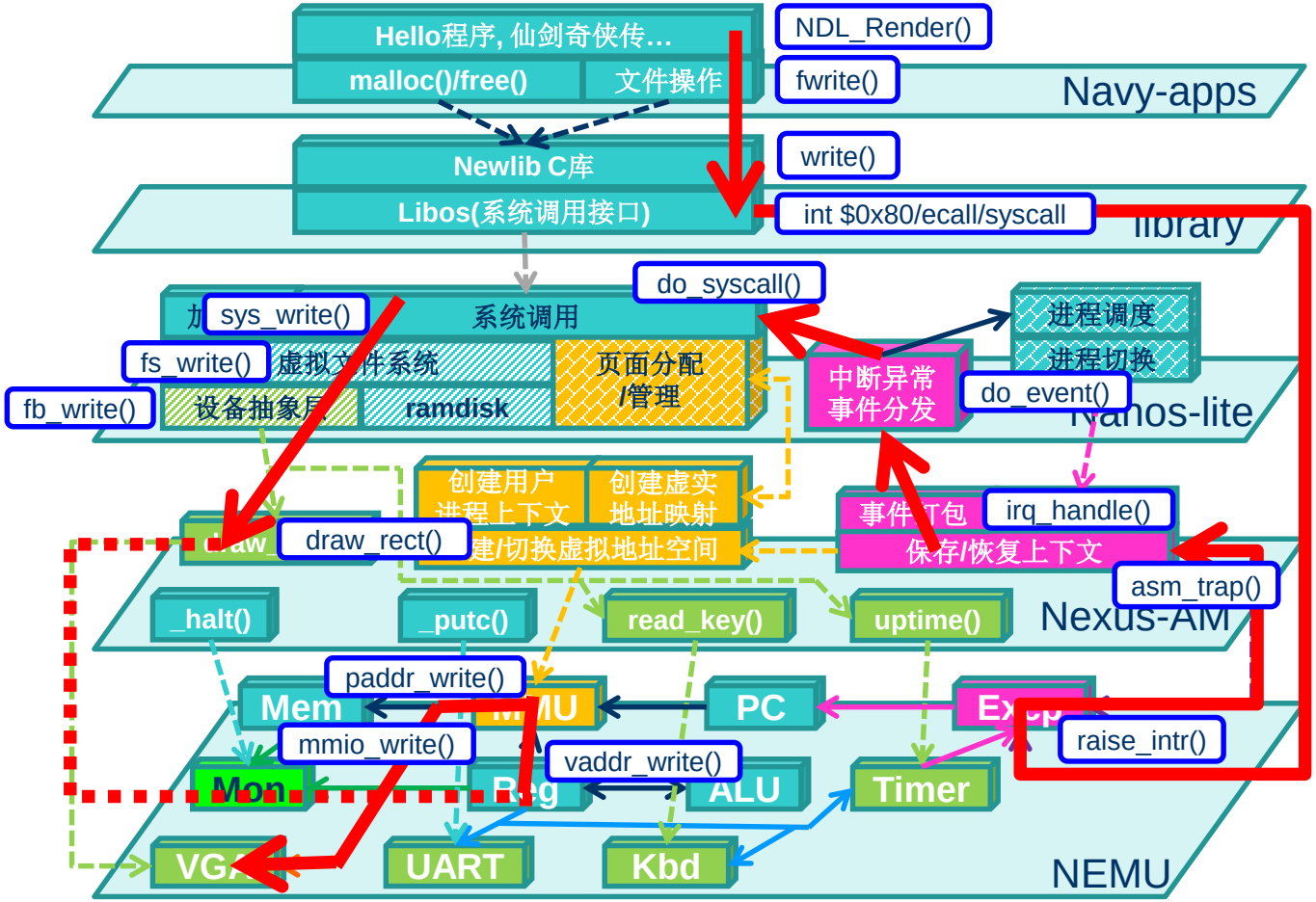
virtualization
concurrency
persistence

TRM IOE
CTE VME

Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt

例子 - 仙剑奇侠传更新屏幕

学生亲手构建这个计算机系统
可以白盒理解 **“程序如何运行”**



PA方案耗时与代码量

- 约50个小模块, 每个小模块平均20~30行代码
  - 需要厘清每个模块之间的关系, 从而建立基本的系统观

实验内容	持续时间/周	预计耗时/小时	代码量/行
PA0 – 开发环境配置	1	10	无
PA1 – 简易调试器	3	30	400
PA2 – 冯诺依曼计算机系统	4	30	300
课时不足可选择完成到PA2 [小计]	(8)	(70)	(700)
PA3 – 批处理系统	3	30	200
PA4 – 分时多任务	3	30	200
PA5 – 程序性能[选做]	-	-	-
总计	14	130	1100

- 预计耗时以中等水平学生为准
  - 中等水平: 编过规模约500行代码的单个程序, 懂得调试

提纲

- 引言
- 实验内容
- 设计原则
- 一些讨论

设计的挑战

- 如何提炼最核心的实验主线?
  - 实验的终极目标是什么?
  - 如何引导学生一步步达到这一目标?
- 如何在实验过程中突出这一主线?
  - 实验/课本中涉及的知识点很多, 如何安排它们的主次?
  - 在一个学期构建一个完整的计算机系统, 可能吗?
- 如何把握完整性与复杂度之间的平衡?
  - 实现一个完整的系统有很多工程挑战, 如何取舍?

挑战1 - 提炼实验主线

- 实验终极目标 - **从构建计算机的角度揭示程序运行的基本原理**
  - 做实验只能自底向上
- 按什么顺序构建?
- 课本顺序
  - 指令系统 -> 浮点数/栈帧链/符号表/ELF loader -> cache -> 虚存管理 -> 异常控制流 -> I/O与中断

- 学生问题:
  - 我最后已经能跑仙剑了, 但还是不知道它怎么跑起来的
- 没能达到"理解程序如何在计算机上运行"的终极目标!

问题分析

- 课本顺序
  - 指令系统 -> 浮点数/栈帧链/符号表/ELF loader -> cache -> 虚存管理 -> 异常控制流 -> I/O与中断

- 学生疑惑: 为什么要按照这个顺序来做?
  - 为何要在一个不能输入输出的机器中加入虚存管理?
    - 因为课上讲了到XXX, 所以在实验里面要做XXX
  - 最终结果 - 学生不理解这些知识点对构建整个系统的意义
- **"计算机如何构建"的主线不够突出**
  - 学生容易迷失在不必要的细节中
    - 不易明白正在做的事情和计算机系统整体的关系

重新审视主线

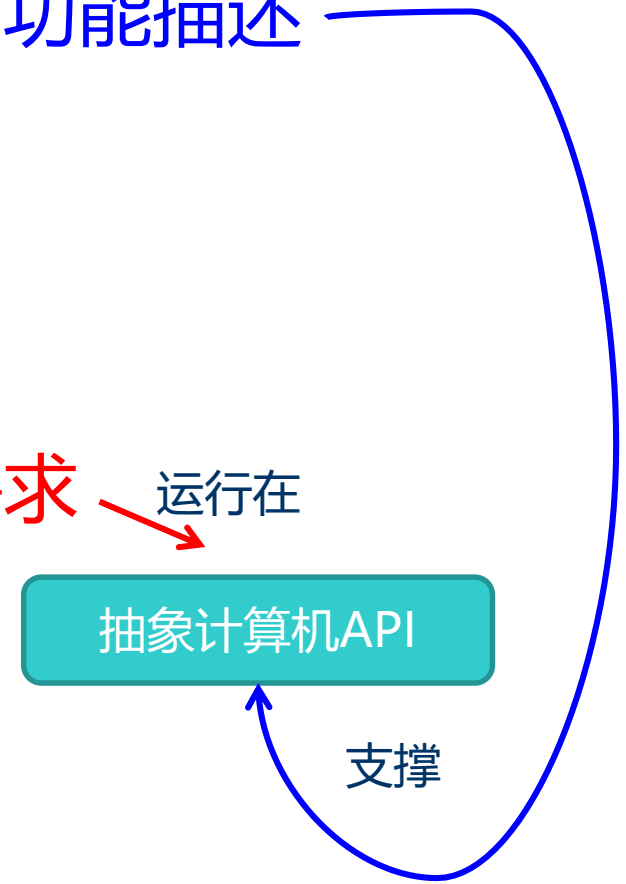
- 反思: 课本顺序对做实验来说合适吗?

- 理论课侧重的是程序的故事
  - 程序的表示->转换->执行
  - **区别 – 假设已经搭建出完整的计算机系统(IA-32 + Linux)**
- PA需要有自己的故事 - 计算机构建的故事
  - 数字电路(前导课程) -> 最小的计算机 -> 功能强大的计算机

- 按什么顺序构建?
  - 以史为鉴 - 向计算机发展史学习!

AM(Abstract Machine) - 计算机抽象模型

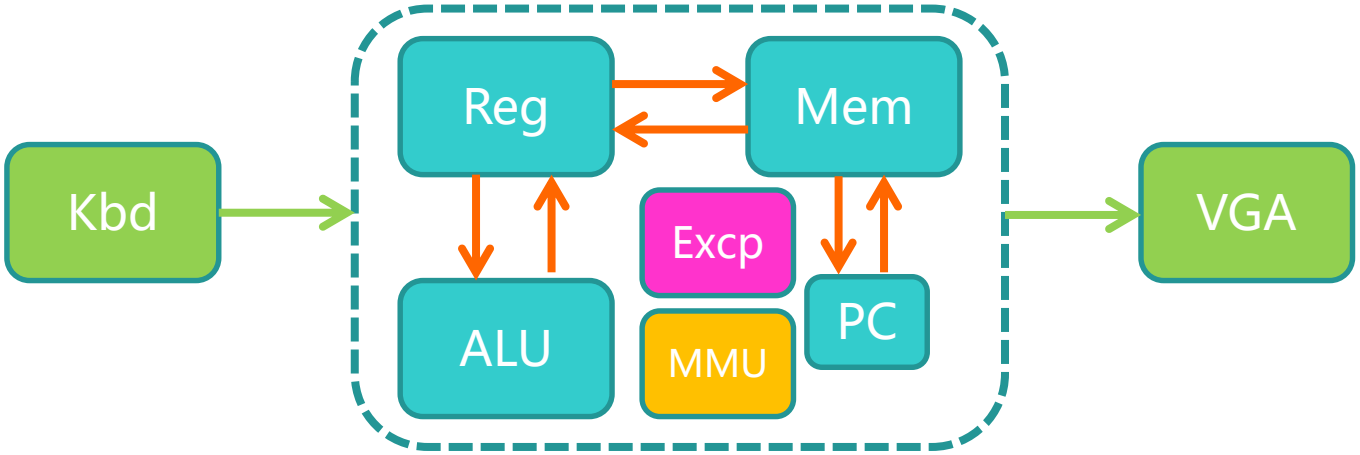
- 根据计算机发展史定义出机器的功能描述
  - 图灵机(1936)
  - 冯诺依曼机(1945)
  - 操作系统GM-NAA I/O(1956)
  - CTSS(1961)
- 抽象出相应机器上程序的运行需求
  - 算法 - 计算
  - 独占资源的任务 – 输入输出
  - 批处理系统 – 上下文管理
  - 分时多任务 – 虚存管理



Jiang and Yu, Building Computer Systems From Scratch Around AbstractMachine, CSEW'18
http://ics.nju.edu.cn/~jyy/teach/am-talk.pdf

AM(Abstract Machine) - 计算机抽象模型

- AM提出者 – 蒋炎岩, 南京大学
- AM = TRM + IOE + CTE + VME + MPE
  - TRM(TuRing Machine) - 图灵机
  - IOE(I/O Extension) - 输入输出扩展
  - CTE(ConText Extension) – 上下文扩展
  - VME(Virtual Memory Extension) - 保护扩展
  - MPE(Multi-Processor Extension) - 多处理器扩展 (PA中不涉及)



AM的意义

- AM = 一组按照计算机发展史来将计算机功能模块地抽象化的C语言API
  - 计算机发展史 – 从需求的角度更容易理解为什么
    - 程序要(高效地)计算 -> (支持指令集的)图灵机 [TRM]
    - 程序要输入输出 -> 冯诺依曼机 [IOE]
    - 想依次运行多个程序 -> 批处理系统 [CTE]
    - 想同时运行多个程序 -> 分时多任务 [VME]
  - **API – 软硬协同地揭示程序与计算机的关系**
    - (NEMU)实现硬件功能 -> (AM)提供API抽象 -> (APP层)运行程序
    - (在NEMU中)实现更强大的硬件功能 -> (在AM中)提供更丰富的API抽象 -> (在APP层)运行更复杂的程序
  - 抽象化 – 支持多种ISA-平台的组合
    - x86-nemu, riscv32-sodor, mips32-qemu, ??-fpga, Linux native
    - 在AM上开发的应用(包括OS)可以移植到各种平台
    - **打通系统方向各课程实验的关键**

AM的意义

- **按照计算机发展史的顺序软硬协同地揭示程序与计算机的关系**

图灵机(1936) -> 冯诺依曼机(1945) -> GM-NAA I/O(1956) -> CTSS(1961)

运行程序 ^ || 提供抽象 ^ || 实现功能

	TRM	IOE	CTE	VME
Apps	-	文件操作	-	malloc, free
库	-	Newlib C库		
		Libos(系统调用接口)		
Nanos-lite	系统调用			
	文件系统		进程调度	页面分配/管理
	加载器	设备抽象层	上下文切换	
Nexus-AM	指令, 堆 _putc() _halt()	_uptime() _read_key() _draw_rect()	保存/恢复现场 事件打包	创建/切换虚拟地址空间 创建虚实地址映射 创建用户进程上下文
NEMU	常用指令集	内存映射I/O	中断异常处理	分页机制

一个例子 – 软硬协同理解并实现MMU

- 传统的组原课只讲MMU的硬件行为
  - 不讲软件如何协同, 也不写MMU的RTL, 学生无法深入理解
- 操作系统课把底层硬件的具体行为当做黑盒
  - MMU变魔术般进行地址转换, 学生同样难以理解

- 在PA中, 学生
  - 在NEMU中实现MMU, 在AM中实现_map(va, pa), 在Nanos-lite的ELF loader中调用_map()
    - 完全明白与MMU相关的每一个细节
- 真正的**软"硬"协同理解计算机系统**
- 和组原课相比, PA更适合当OS的前导课

多种ISA的支持

- AM使得各种ISA可以共享同一套上层软件(OS+应用)
- 学生可以在PA中选择x86/mips32/riscv32

- 难度分布(1星-困难, 5星-容易)

	x86	mips32	riscv32
PA1 - 简易调试器	与ISA选择关系不大		
PA2 - 冯诺依曼计算机系统	*	***	*****
PA3 - 批处理系统	*****	***	*****
PA4 - 分时多任务	*****	*	****

- 二周目可选择不同的ISA, 从系统层次比较ISA的区别
  - 但由于理论课讲x86, 大部分学生还是选x86

挑战2 - 突出主线

- 如何在实验过程中突出这一主线?
  - 在一个学期构建一个完整的计算机系统, 可能吗?
  - 实验/课本中涉及的细节知识点很多, 如何安排它们的主次?

- 黄金法则 - KISS(Keep It Simple, Stupid)
  - 先完成, 后完美
  - (1) 先实现一个功能**完整**的**最小**计算机系统
    - TRM+ IOE + CTE + VME
  - (2) 在(1)的基础上, 增强计算机的功能/提升程序的性能
    - 浮点模拟, profiling, JIT

功能完整的最小系统

- 实现标准mips32/riscv32费时费力, 标准x86更难
- 程序如何运行 -> **正确的**程序如何运行
  - 假设用户程序不会有非法行为

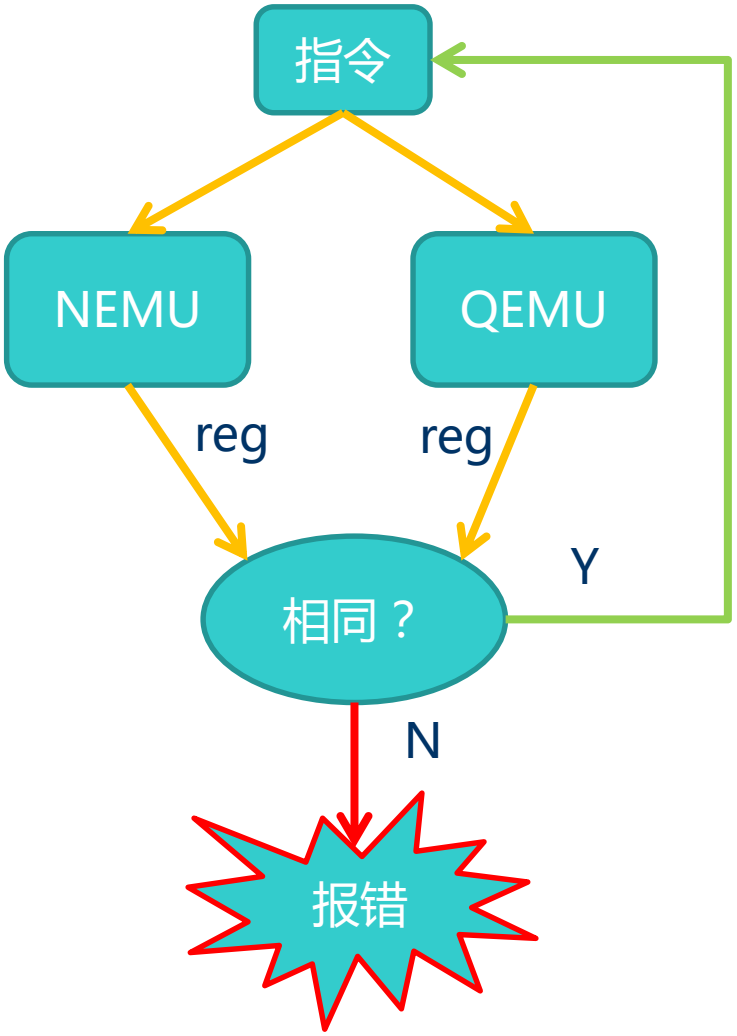
- 裁剪
  - 指令(TRM) -> riscv优势: 指令模块化方便裁剪
    - x86和mips32无法给出某个子集, 使得编译器仅生成该子集内的指令
  - 输入输出(IOE) -> 简化设备的功能
  - 中断异常(CTE) -> 去掉保护机制(特权级), 异常只保留系统调用, 中断只保留时钟中断
    - 其它外设的中断功能通过轮询实现
  - 虚存管理(VME) -> 没有权限检查的分页(page walk)

例子 – 教学简化版的x86

- 简化设计 - 延后, 减小, 删除非主线内容
- ISA无关的简化
  - 浮点数的支持延后到PA5
  - 去掉栈帧链, 符号表, cache
    - 移出PA, 独立成为Lab(小实验)
  - 简化串口, 键盘, 去掉磁盘, DMA, 中断控制器...
- 以教学裁剪版的x86为例
  - 去掉字符串处理指令(gcc的-mstringop-stragety=loop)
  - 去掉EFLAGS中的PF和DF
  - 简化IDT相关的结构设计
  - 去掉分段, TLB, 硬件保护机制
- 完整性: 仍然可以运行真实游戏仙剑奇侠传

挑战3 - 完整性与复杂度的平衡

- 强化基础设施 - 帮助学生快速调试
  - differential testing
    - 把QEMU作为参考实现
    - 逐条指令对比执行后的状态
  - 一键回归测试
  - 系统快照
  - AM的Linux native平台
    - 方便调试
    - OS也可以运行在native上



- 好处
  - 减轻工程压力
  - 理解科学的做事原则
  - 接触前沿技术

提纲

- 引言
- 实验内容
- 设计原则
- 一些讨论

系统能力和处理器设计能力

- 有人说: NEMU只是个C模拟器, 不写RTL, 太假了
- 但不写RTL恰恰是NEMU的优势
- RTL的问题: (相对软件)难写难调试
  - 分时多任务仙剑(PA一学期) vs 五级流水线跑???(组原一学期)
- 更多的问题: 中断异常和虚存对学生来说更是"迷之黑洞"

- 更本质地: 系统能力和处理器设计能力是两种不同的能力
  - 程序, 库, 运行时环境, ABI, syscall, VFS, HAL, ISA
  - 信号, 寄存器, 状态机, 流水线, buffer, 队列, cache, 握手
  - 这些概念是正交的
- PA的价值: 把处理器设计和系统能力解耦开来
  - 学生通过很小的代价(写C模拟器而不是硬件)来体会软硬件协同
- 和传统的组原课相比, PA更适合当OS的前导课

系统能力和处理器设计能力

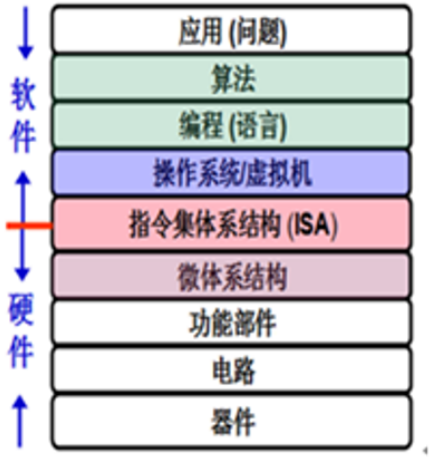
系统能力指什么？

能在系统层面进行分析、设计、调优、检错，站在系统高度解决应用问题。

- 对软、硬件功能进行合理划分
- 对系统不同层次进行抽象和封装
- 对系统整体性能进行分析和调优
- 对系统各层面的错误进行调试和修正
- 对系统进行准确的性能评估和优化
- 根据不同应用要求合理构建系统框架
- ……

计算机专业不仅要培养程序员，更要培养能够构建计算机系统的专门人才

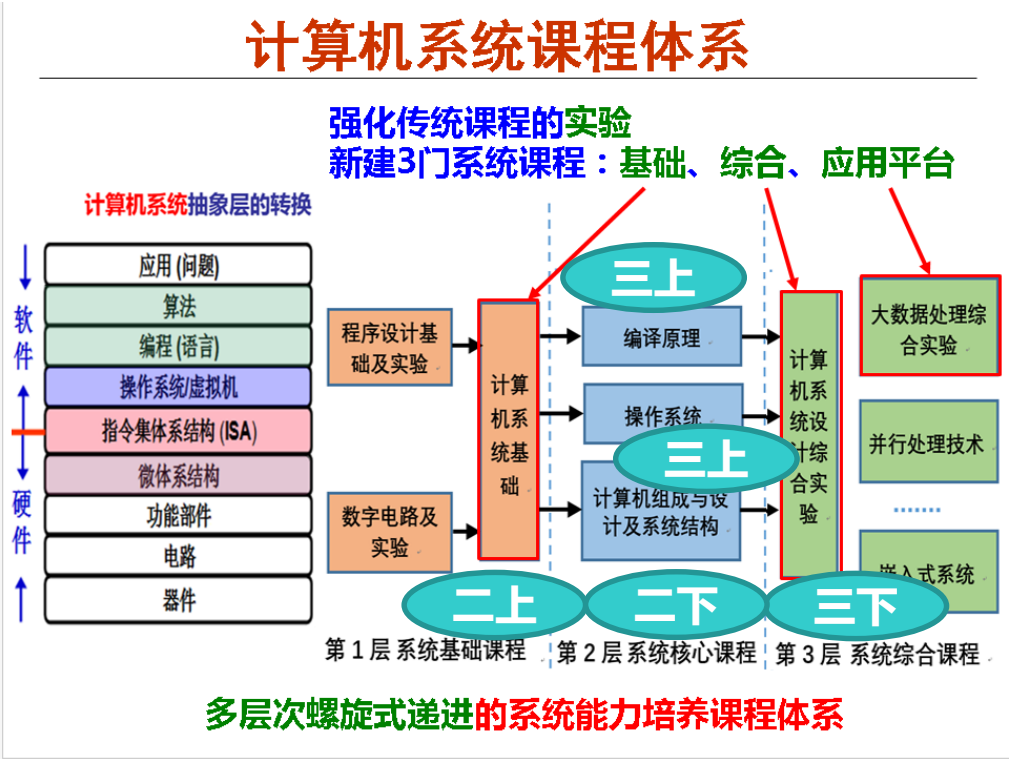
计算机系统抽象层的转换



袁春风, 南京大学

- 不设计处理器, 也可以培养(基本的)系统能力
- 仅设计处理器, 也培养不了(基本的)系统能力

南京大学的计算机系统课程体系



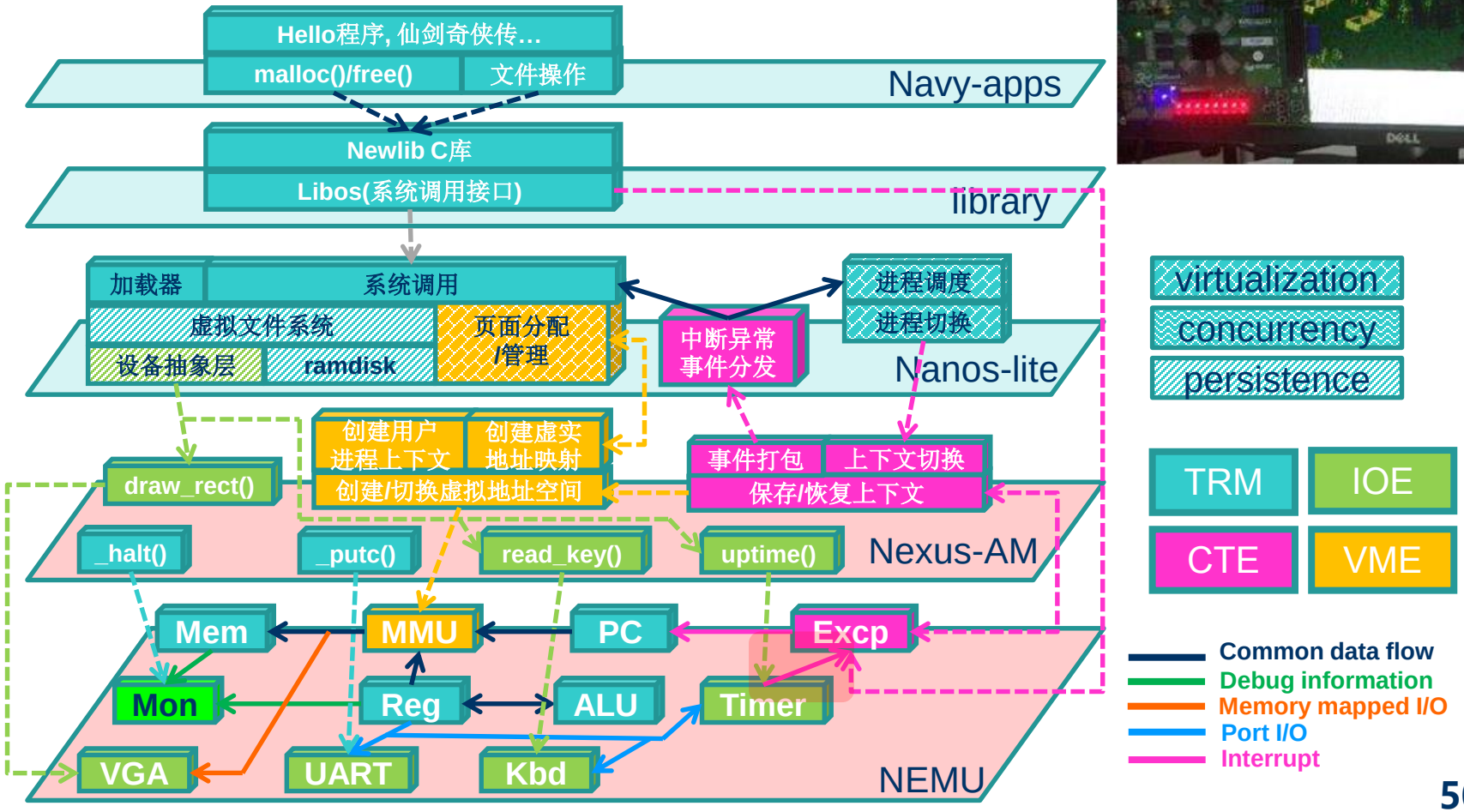
袁春风, 南京大学

- 不同课程的使命:
  - (二上)计算机系统基础 – 一个程序如何在计算机上运行
  - (二下)操作系统 – 多个程序如何在计算机上运行
  - (三上)编译原理 – 程序是怎么来的
  - (三上)组成原理 – 如何造一个正确的计算机

PA与系统方向其他课程实验的衔接

组成原理实验 = 硬件版的NEMU

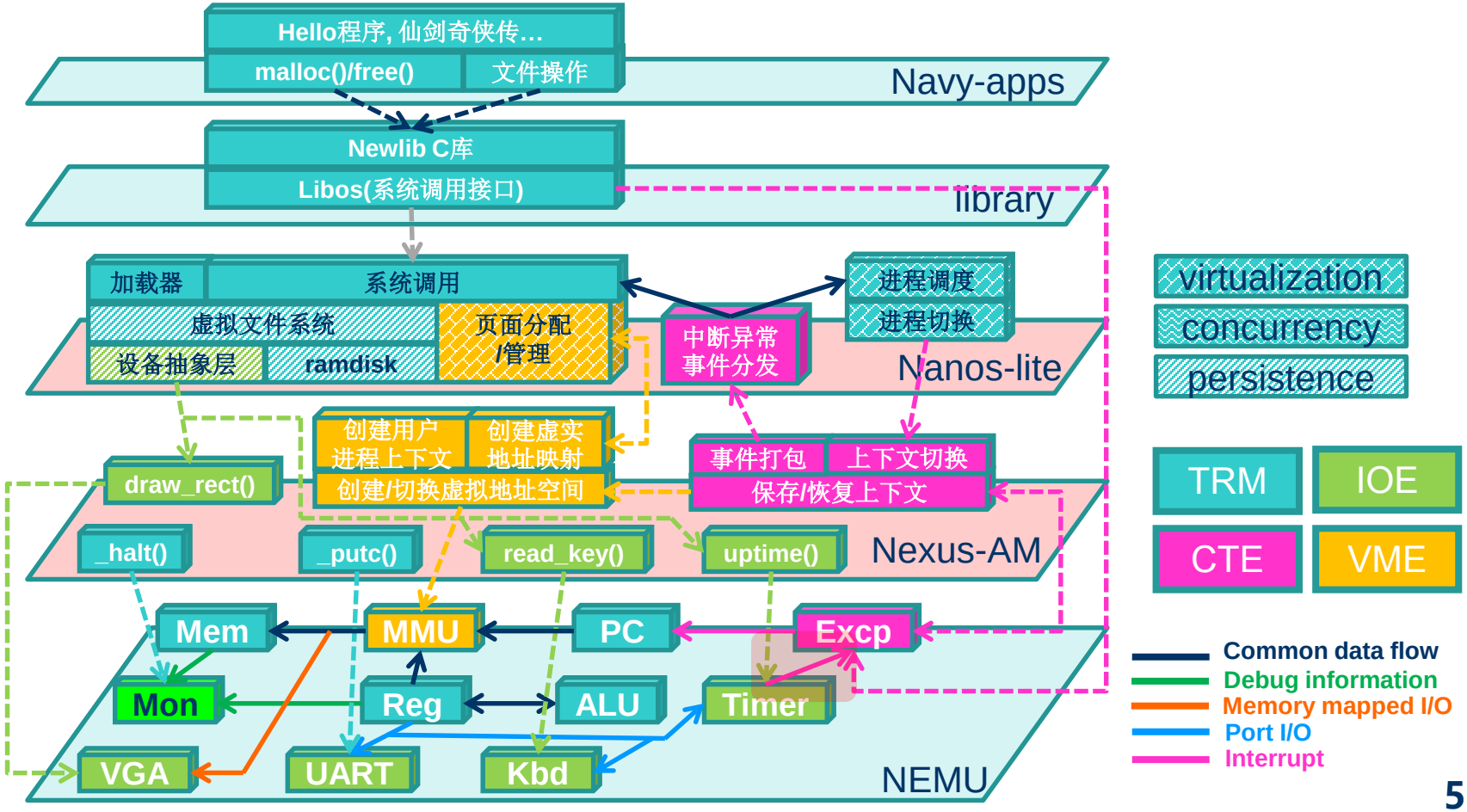
- 集中关注微结构的实现细节(PA已经打通全栈)



PA与系统方向其他课程实验的衔接(2)

操作系统实验 = 完整版的Nanos-lite

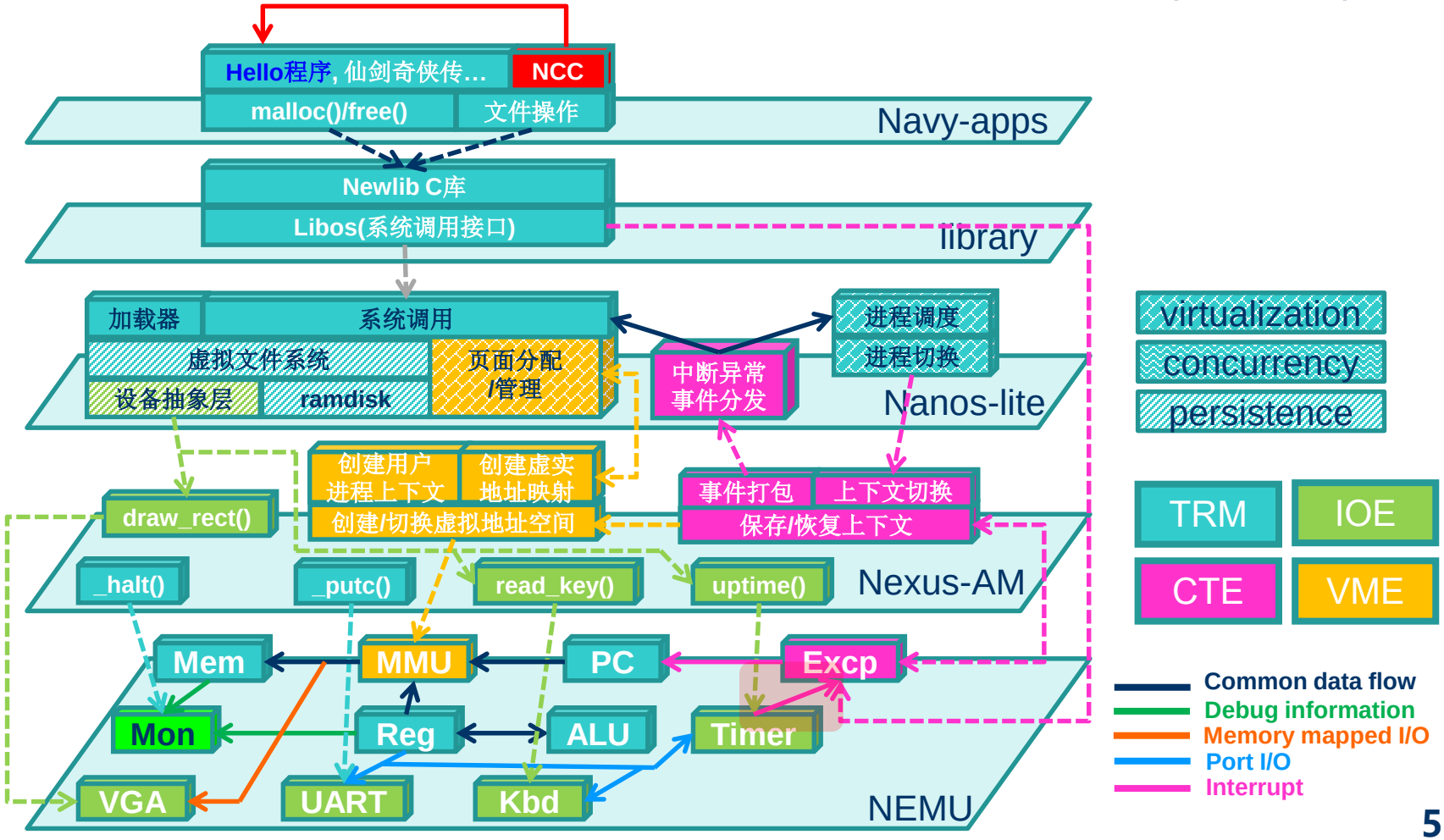
- 集中关注OS的核心概念 - 虚拟化, 并发, 持久化 (PA已经打通全栈)



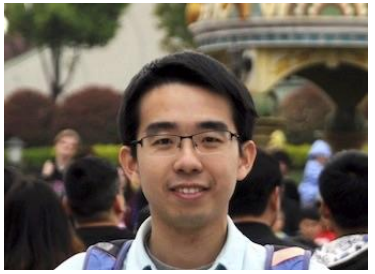
PA与系统方向其他课程实验的衔接(3)

编译原理实验可归到Navy-apps中(相对独立)

- 可考虑如何生成运行在教学生态系统栈上的可执行文件(简化ABI)



有ICS和PA支持下的OS课程安排



- 蒋炎岩的OS课
  - ICS和PA中省下的时间可以让OS课介绍更前沿的内容
  - 无硬件细节内容

1. 操作系统概述 (slides) | 应用眼中的操作系统 (slides) | [M1] pstree
2. 虚拟化：进程抽象 (slides) | 进程管理与Shell (slides) | [M2] perf
3. 虚拟化：虚存抽象 (slides) | 链接与加载 (slides) | [M3] crepl
4. 专题：操作系统中的资源管理和调度 (slides) | [L0] amgame (slides)
5. 习题选讲：操作系统中的程序设计 (slides)
6. 并发：进程与线程 (slides) | 互斥 (slides) | [M4] malloc/free
7. 并发：同步、条件变量 (slides) | 信号量 & POSIX线程编程 (slides) | [L1] kthread
8. 虚拟化与并发复习 (slides) | 期中测验 | 并发bug (slides)
9. 设备管理 (slides) | [M5] 内存修改器
10. 持久化：数据的故事 (slides) | 文件系统实现 (slides)
11. 持久化：崩溃一致性 (slides) | 持久数据的可靠性 (slides) | [M6] libkvdb
12. 复习：操作系统概观 (slides) | 系统软件的质量保障 (slides) | [L2] virtualfs
13. 系统虚拟化：虚拟机和容器 (slides) | Guest Lecture by Xinyu Feng
14. 网络与套接字 (slides) | 分布式系统 (slides) | [M7] httpd
15. 操作系统安全 (slides) | Java虚拟机 (slides); Guest Lecture by Rong Gu
16. 操作系统复习 (slides)

2018年春季
12周结束主要内容
7个前沿专题
L0~L2共3个大实验

1. 操作系统概述 | C 应用眼中的操作系统 | [M1] pstree | [L0] amgame
2. 并发 共享内存多线程 | C 硬件眼中的操作系统 | [M2] libco
3. 并发 互斥 | C 并发数据结构 | [L1] kalloc
4. 并发 同步 | C 并发bugs
5. 虚拟化 进程抽象 | C 终端和Shell | [M3] sperf
6. 虚拟化 处理器调度 by 杨已彪 | C 链接与加载 | [M4] crepl
7. 虚拟化 虚存抽象 | C 调试操作系统内核 | [L2] kthreads
8. 期中复习 | 随堂期中测验
9. 持久化 存储介质 | I/O设备与驱动
10. 持久化 文件系统概念 | C 文件系统API
11. 持久化 文件系统实现 | FAT和ext2 | [M5] frecov
12. 持久化 持久数据的可靠性 | 崩溃恢复与日志 | [M6] libkvdb | [L3] vfs
13. Putting Pieces Together | 微内核操作系统
14. 专题 多用户与访问控制 | 操作系统安全
15. 专题 网络与套接字 | 分布式系统 | [L4] uproc
16. 专题 容器与虚拟机 | 期末复习

4周时间介绍
前沿知识

2019年春季
12周结束主要内容
6个前沿专题
L0~L4共5个大实验

完整的ProjectN组件

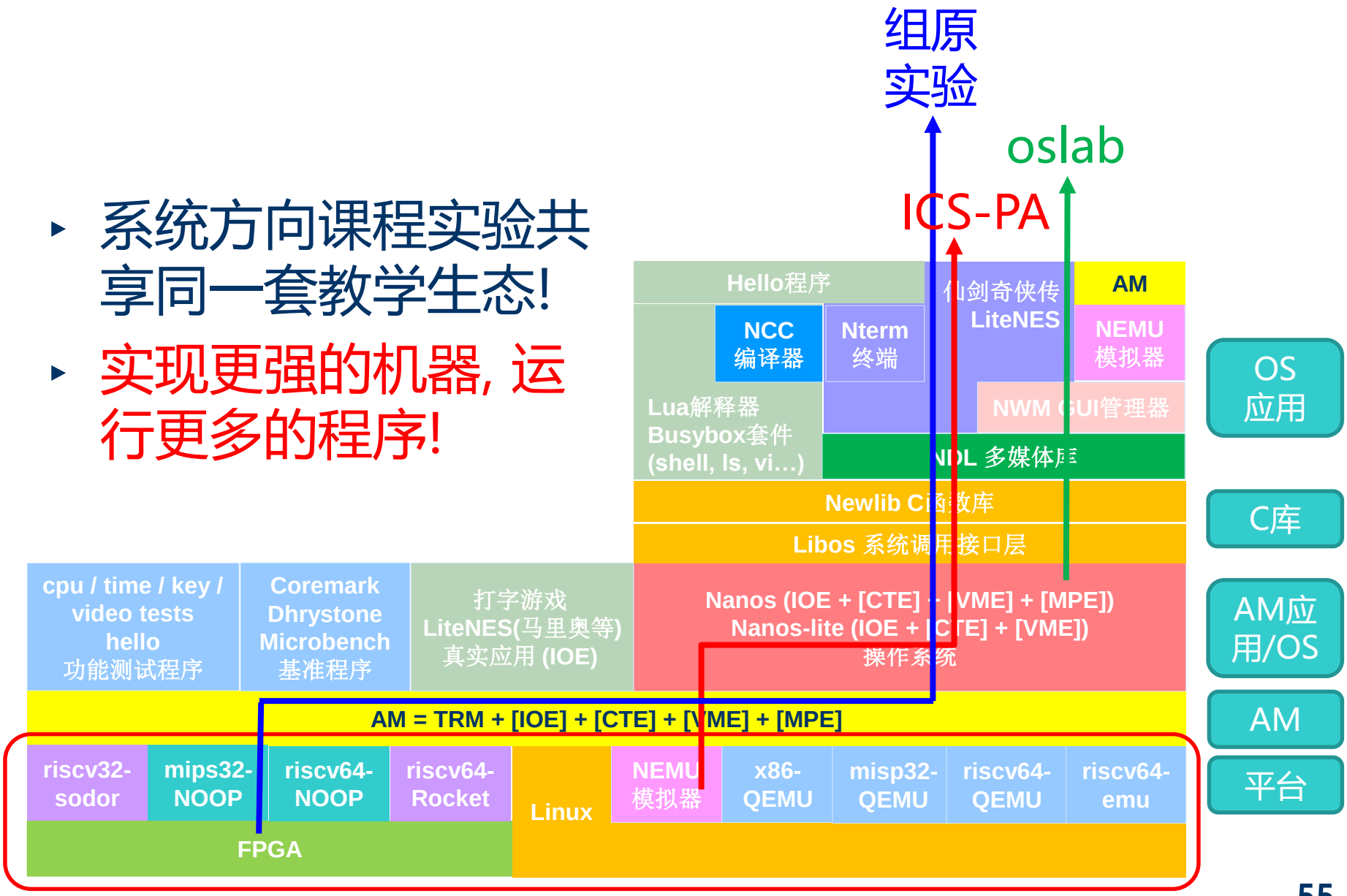
- 和蒋炎岩共同维护

ProjectN组件	说明
Navy-apps	应用程序集
NCC	NJU C Compiler, C编译器
Newlib	嵌入式C库(从官方版本移植)
Nanos/Nanos-lite	NanJU OS, 操作系统/简化版
Nexus-AM	抽象计算机, ISA抽象层
NEMU	NJU EMUlator, 全系统模拟器
NPC	NJU Personal Computer, SoC
NOOP	NJU Out-of-Order Processor, 处理器

- PA用到的组件构成mini-ProjectN

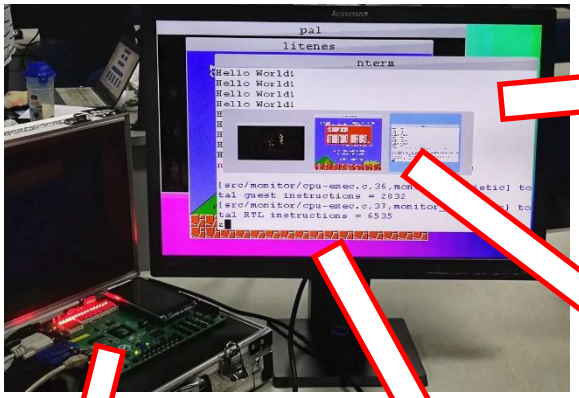
基于AM的ProjectN (南京大学系列实验)

- 系统方向课程实验共享同一套教学生态!
- 实现更强的机器, 运行更多的程序!



南京大学的龙芯杯作品 (系统综合实验)

- AM是打通系统方向各课程实验的关键



x86
模拟器

nterm
终端模拟器

图形应用程序
NES模拟器、仙剑

应用

NWM 窗口管理器

libos, libc, libndl (图形)

运行库

功能测试

基准程序

打字游戏

Nanos
操作系统

操作系统

AM (抽象计算机) = TRM + IOE + [CTE] + [VME]

抽象层

mips32

ISA

NOOP 乱序处理器

NEMU全系统模拟器

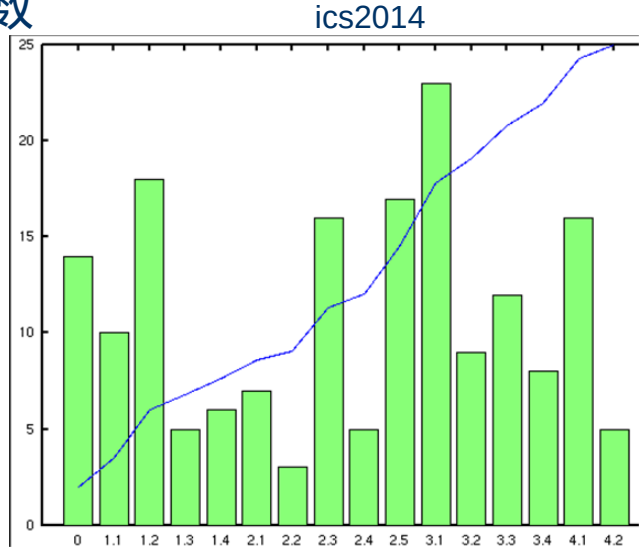
微结构

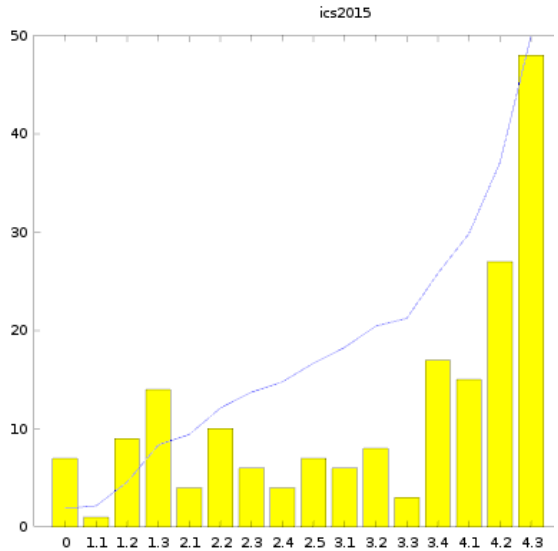
学生完成情况(2014-2016)

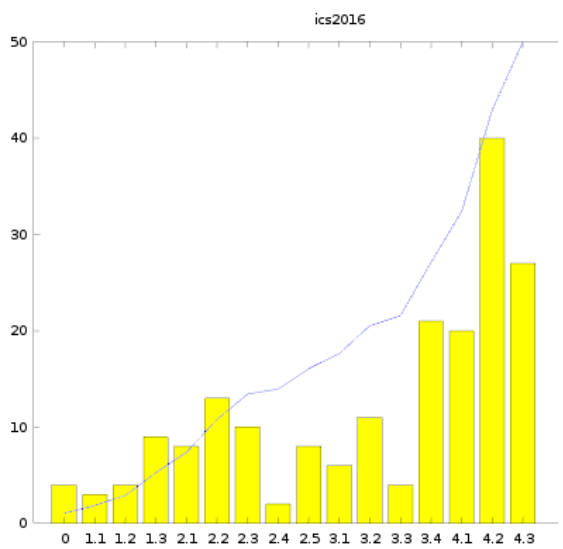
- 第1年缺少有效的指导, 第2, 3年情况好转

学期	人数	平均分/40	完成PA2	完成PA3	开始移植仙剑
2014秋	174	23.07	51.72%	10.92%	2.87%
2015秋	186	29.50	70.43%	57.53%	25.81%
2016秋	190	30.88	72.11%	56.84%	14.21%

人数



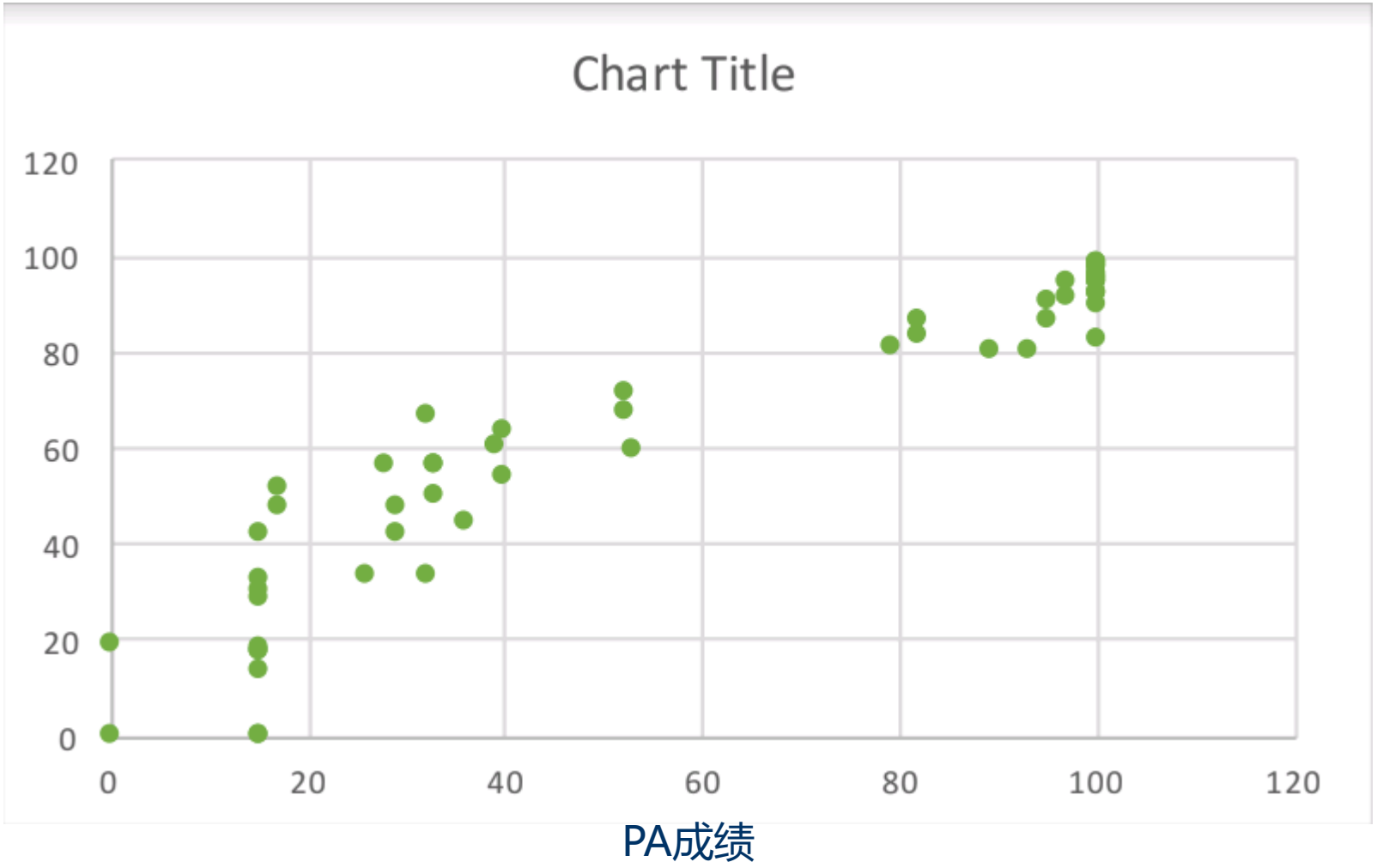




阶段

学生完成情况(2017年秋)

- 第4, 5年引入AM, 明确主线, 效果更好



学生匿名反馈(2018年秋季)

- 35份有效回答

问题	平均得分
课程开始前对"程序如何在计算机上运行"的理解	2.97 (10-完全理解)
课程结束后对"程序如何在计算机上运行"的理解	6.94 (10-完全理解)
PA对理解"程序如何在计算机上运行"的帮助	7.88 (10-非常有帮助)
做PA前的编码能力	3.68 (10-非常强)
做PA后的编码能力	6.06 (10-非常强)
完成PA后对处理器设计的兴趣	5.00 (10-非常感兴趣)
完成PA后对操作系统, 编译器等系统软件的兴趣	7.12 (10-非常感兴趣)

问题1 – 学生不适应PA的训练

基本原理	做事方案	正确性风险	代表例子
阐述	明确	基本正确	高中物理实验
阐述	明确	可能出错	程序设计作业
阐述	需要思考	基本正确	数学/算法题
阐述	需要思考	可能出错	PA, OSlab
需要探索	需要思考	可能出错	业界和科研的真实问题

- 学生反馈
  - 我觉得讲义写得不清楚
  - 我不想RTFM, 能不能直接告诉我具体要做什么
  - 我懂分页的工作原理, 但我写不出相应的代码
- PA = 最靠近真实问题的训练
  - 独立完成PA -> 具备了解决真实问题的基本素质
    - 开源社区, 科研问题, 业界需求
- **挑战 - 如何更好引导学生踏入探索性/开源项目?**

问题2 – 学生编程能力薄弱

- 学生面向OJ编程
  - 以通过测试为目标
  - 代码风格差
- 持续性的连贯实验
  - 程序设计实验/OJ作业交上去就可以撒手不管
  - 在PA中, 前期留下的bug就是后期的大坑
    - 想调试, 发现自己以前写的代码都看不懂了
- 没有绝对的正确答案
  - 通过测试不代表正确

- 学生需要面对事实: 如何写出可维护的高质量代码?

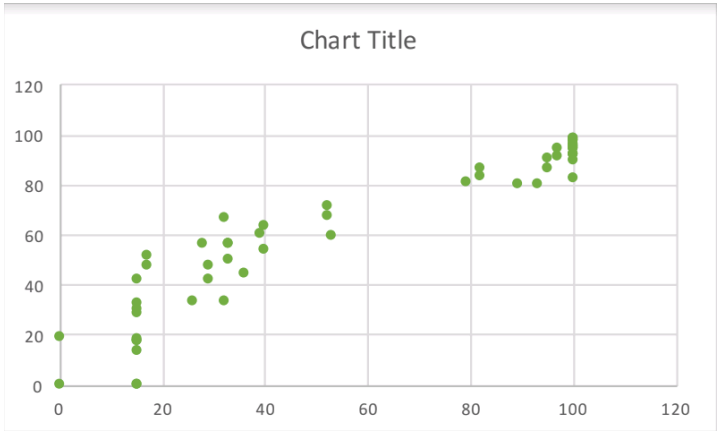
问题3 – 学生心态不端正

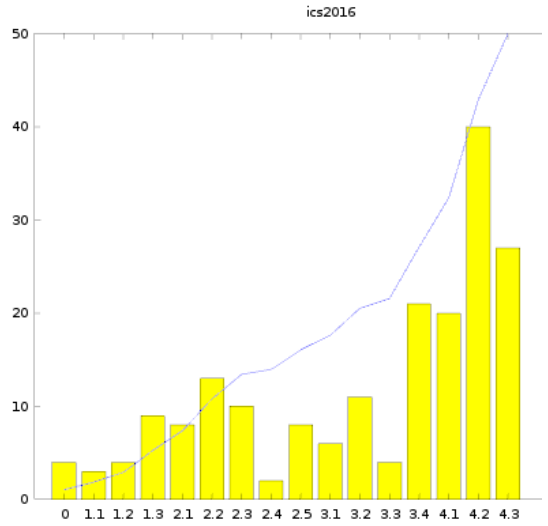
- 不想使用新工具(gdb的bt命令等)
  - 又要RTFM, STFW, 太麻烦了, 我还是直接看自己写的代码吧
- 不去尝试彻底理解代码的每一处细节
  - 这部分代码看不大懂/不是我写的, 随便吧, 反正和成绩没关系
- 遇到bug
  - 我盯着代码看了半个小时都看不出什么问题, 还是找个大腿吧
    - 也反映出程序设计课训练不足, 学生编程基础不扎实
  - 心态也容易崩

- 这样的学生错过了很多锻炼的机会
  - **调bug是厘清系统模块之间关系的好机会**(可惜学生不一定理解)
  - 随着实验的推进, 不同抽象层的交互越来越复杂, 学生的能力越来越跟不上

问题4 – 分数两极分化

- 分数两极分化, 但能力总体仍然呈正态分布
  - 大神型(10%) - 真正能吸收PA所传递的价值观
    - 有能力从设计者的角度去思考问题
  - 大众型(40%) - 分数至上, 仅仅是完成任务, 忽视方法的锻炼
    - 无法胜任新的探索性项目
  - 伸手党(30%) - 遇到问题就紧抱大神的大腿
    - 就算跑起了仙剑, 也没有得到该有的技能训练
    - 无法独立重新完成一遍PA
  - 学渣型(20%) - 做到PA1就直接放弃





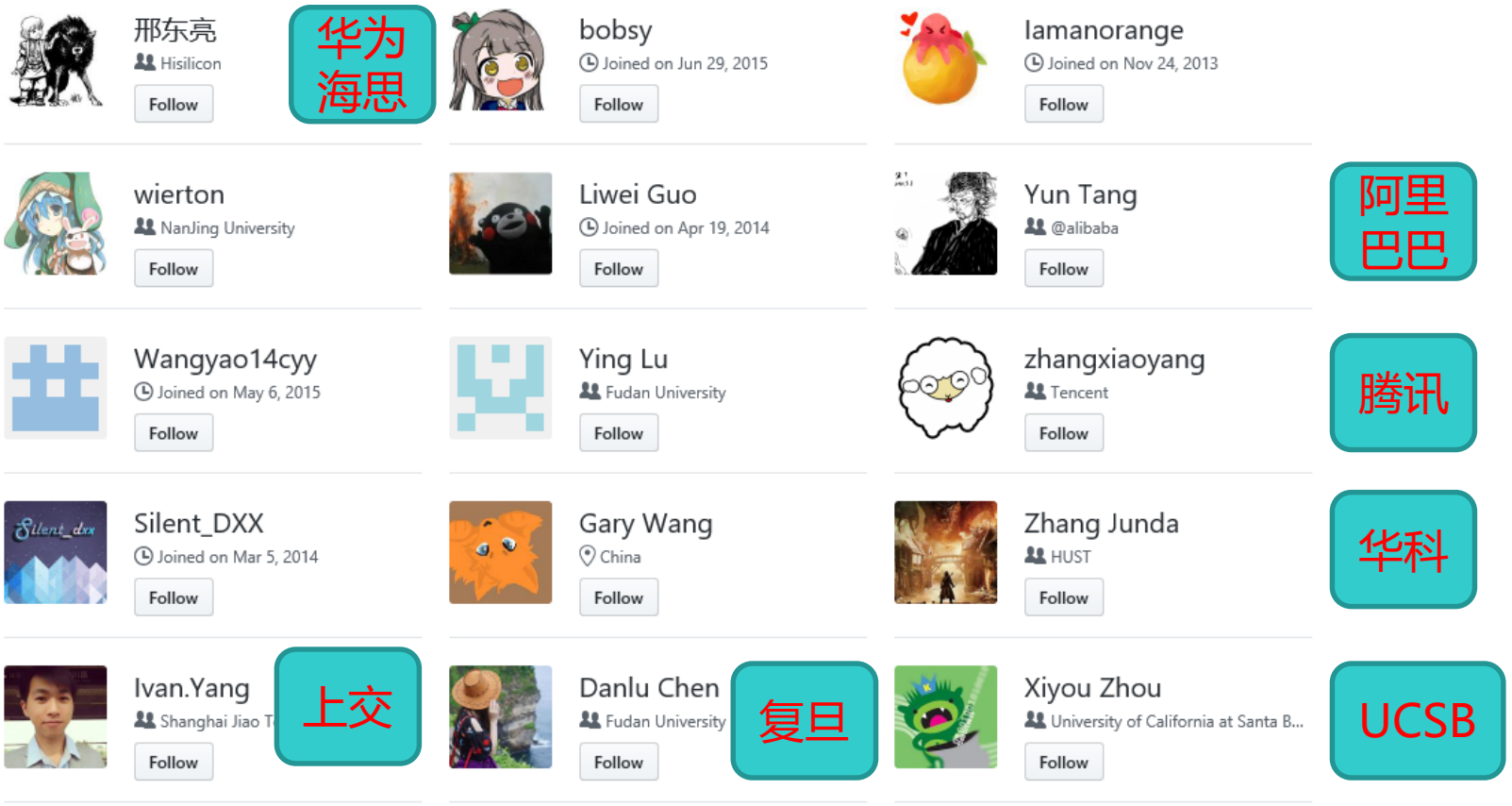
一些惊喜

- 学生对跑游戏还是很感兴趣, 实验初期都很有志气
  - 我一定要把仙剑跑起来
- 游戏里面也自己的"CPU"
  - LiteNES有6502, 仙剑有脚本指令系统
  - 学生能认识到NEMU并不仅仅是个玩具
- 抄袭率很低, 虽然CSDN有攻略, github有(非官方)答案
  - 学生真的愿意投入, 虽然很痛苦
  - 要么就诚实地放弃

- PA的天花板很高, 学生跳得越高, 收获就越大
  - 不少学生真的坚持下来, 很有成就感
  - 学生感触良多, 改变了他们很多(看学生报告感觉读了个博)

PA受到了各方人士的关注

- PA实验方案已被南航, 复旦, 华科, 天大, 南开等高校相继采用
- 一些工业界的程序员也感兴趣



小结

- 条理清晰的主线(完整)
  - AM = TRM + IOE + CTE + VME
    - (在NEMU中)实现硬件功能 -> (在AM中)提供API抽象 -> (在APP层)运行程序
  - 让学生按发展史亲手构建一个完整的计算机系统
    - 理解程序如何在计算机上运行
- 主次得当的设计(精简)
  - 优先实现完整的最小系统, 简化不必要的复杂设计
  - 向学生灌输正确的做事原则
    - 先完成, 后完美
- 丰富的基础设施(前沿)
  - 引导学生实现足够强大的工具帮助对系统进行调试
  - 带领学生走向前沿
    - Differential Testing, 一键回归测试, 快照, AM