chsgcxy

outline

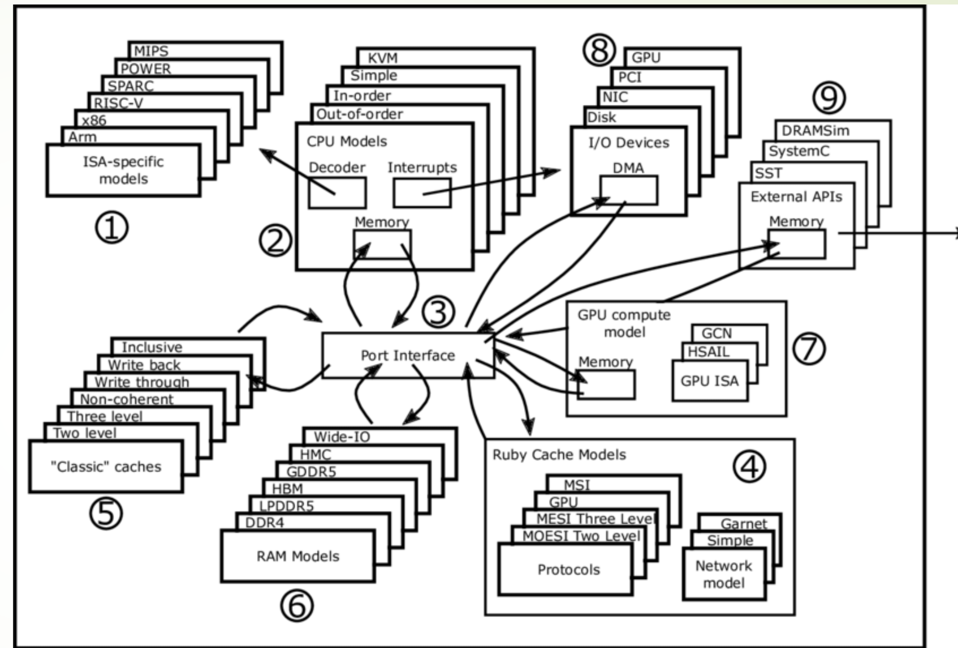
- Overview
- How to use

官网：[http://www.gem5.org]
学习指导：[http://www.gem5.org/documentation/learning_gem5/introduction]
相关资料：[http://daystrom.m5sim.org/Main_Page]

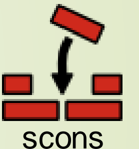
overview

one of the most popular academic-focused computer architecture simulation frameworks

- First publication in 2011
- Dynamically configurable
- Execute-in-execute
- Full system mode and sys-call mode
- Out-of-order, in-order, others
- X86, Arm, RISCV, MIPS, Power, GPU…
- KVM support(functional)
- Ruby memory sub-system
- DDR3,DDR4,GDDR,HBM,HMC,LPDDR4,LPDD-R5…(not cycle-accurate but nearly accurate and more flexibility)
- Support SystemC as components
- DRAM-Sim(cycle-accurate) has been integrated
- Integrate with Structural Simulation Toolkit(SST)
- Gem5-resources(linux-kernel, disk images, benchmark, tests-cases…) supported



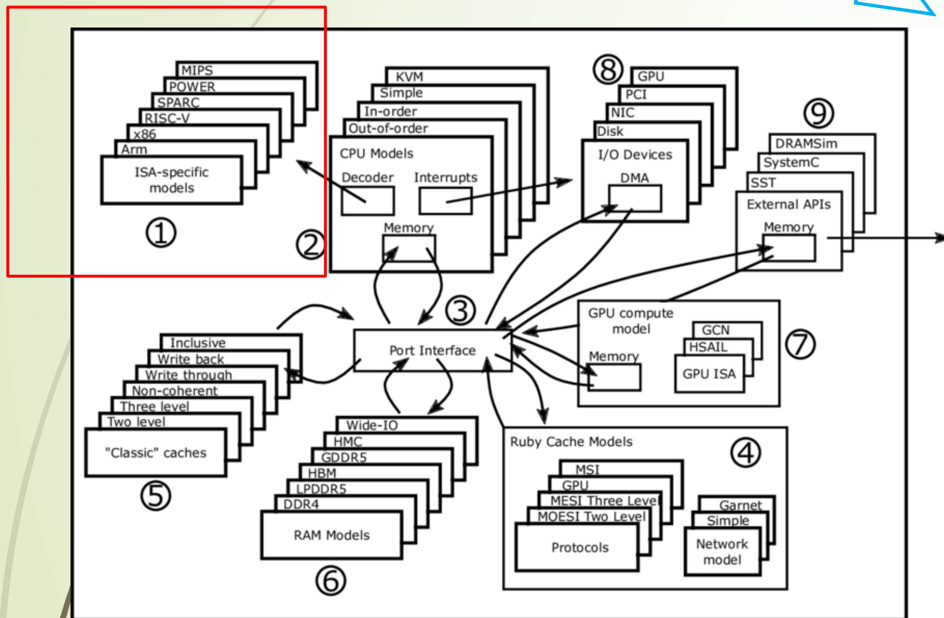
gem5是一款可以实现时钟精确仿真的SOC模拟器。它本身的目的并不是帮助产品开发，它更偏向于教育和架构探索



自带python解析器

overview

不同架构支持，通过编译不同的target实现（即一个编译好的gem5只能支持某个特定架构）



进行组合

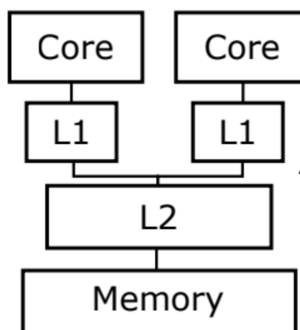
python

```python
system = System()

# Set the clock fequency of the system (and all of its children)
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()

# Set up the system
system.mem_ranges = [AddrRange('512MB')] # Create an address range

# Create a simple CPU
system.cpu = TimingSimpleCPU()

# Hook the CPU ports up to the membus
system.cpu.icache_port = system.membus.cpu_side_ports
system.cpu.dcache_port = system.membus.cpu_side_ports

# Create a DDR3 memory controller and connect it to the membus
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.port = system.membus.mem_side_ports

# Connect the system up to the membus
system.system_port = system.membus.cpu_side_ports

# set up the root SimObject and start the simulation
root = Root(full_system = False, system = system)
# instantiate all of the objects we've created above
m5.instantiate()

print("Beginning simulation!")
exit_event = m5.simulate()
print('Exiting @ tick %i because %s' % (m5.curTick(),
exit_event.getCause()))
```

How to use

到底怎么用？

微架构模型　　配置文件　　运行获取统计数据




```
system = System()
system.cpu = OOO_CPU()
system.cpu.width = 8
system.l1 = Cache()
...
system.l1.mem_side = \
 system.l2.cpu_side
...
system.workload = \
 'hello.exe'
simulate()
```

```
> ./gem5 script.py
```

```
hello world!
```

```
l1.misses 2836
l1.hits   10374
cpu.ipc     1.3
```

Use python script to describe the system
C++ works actually

不同的CPU架构通过不同的编译参数进行区分
同一CPU架构的不同微架构大部分可以通过配置文件进行修改

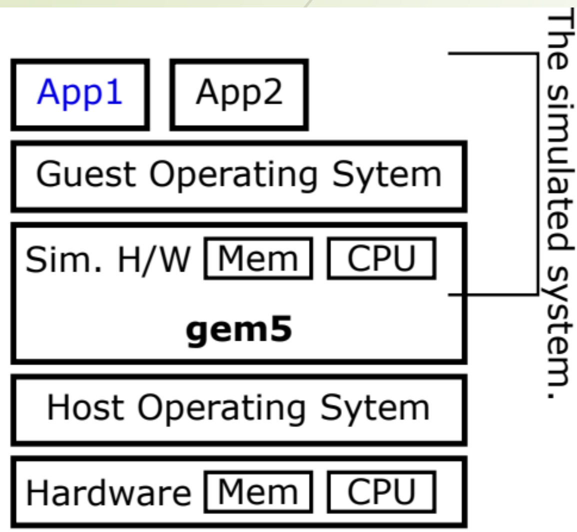
```
./build/NULL/gem5.debug --debug-flags=RxuPrefetchTester,RubyNetwork configs/example/rxu_noc.py --num-cpus=32 --num-dirs=1 --network=garnet --topology=RXUMesh_XY --mesh-rows=4 --synthetic=uniform_random --mem-type=DDR4_2400_16x4 --mem-size=4GB --num-packets-max=50 --sys-clock=500ps --ruby-clock=500ps
```

Full System & sys-call

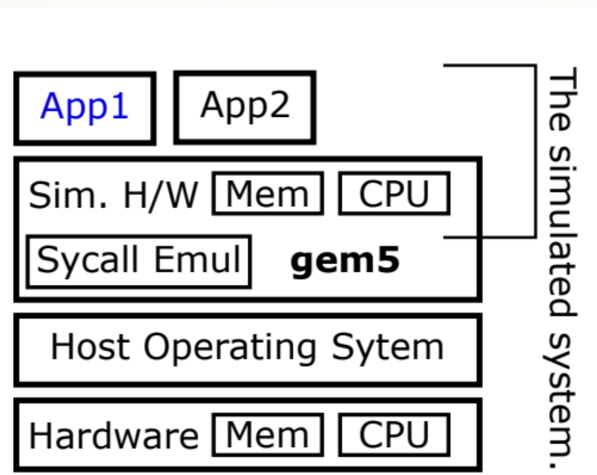
Full System mode

Sys-call mode

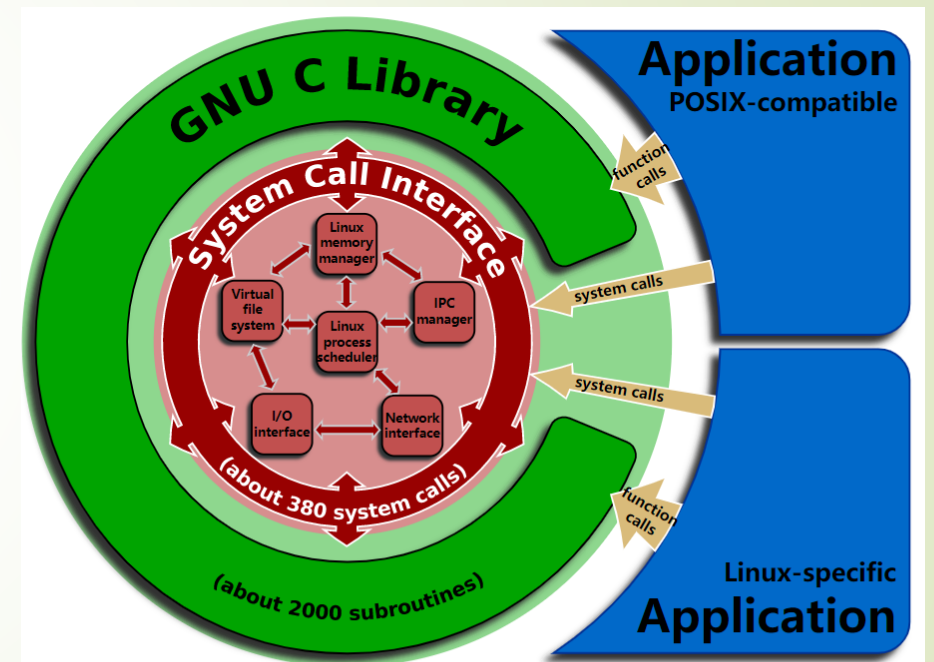
系统调用



(b) The hardware/software abstraction layers when using gem5 in full system simulation mode.



(c) The hardware/software abstraction layers when using gem5 in system call emulation mode.

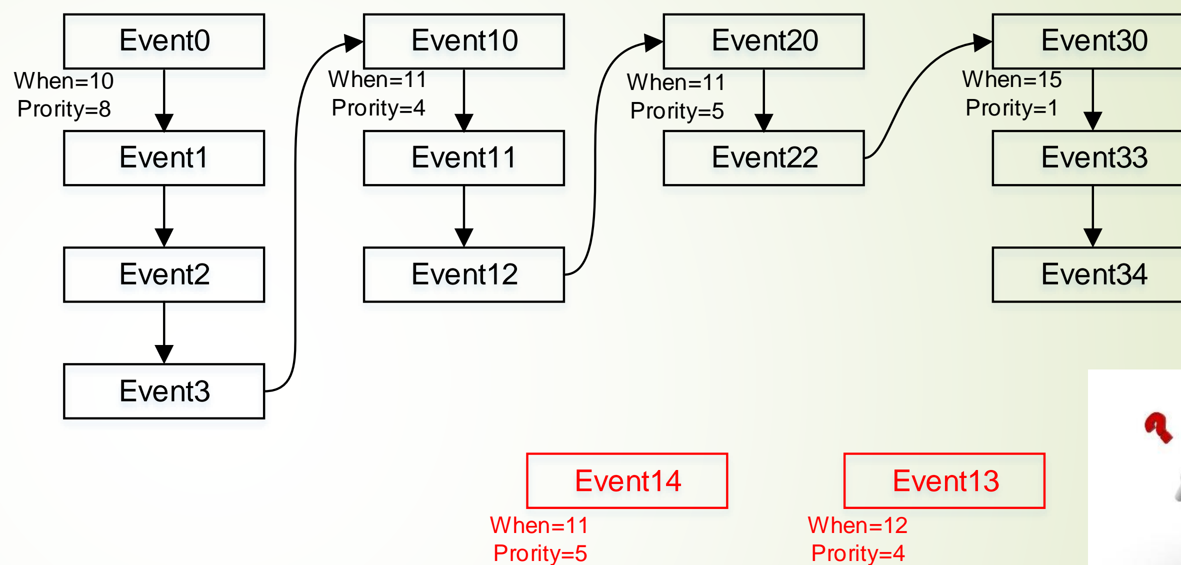


Gem5-resources support linux-kernel and disk images

要想跑一个简单的benchmark，应该怎么做？
想跑一个bare-metal的代码，应该怎么做？

事件机制

gem5的运行是靠事件机制来完成的，事件按照tick和优先级进行排序，同一tick和优先级的事件被称为“InBin”，按照栈的方式来管理 事件管理的核心代码在src/sim/eventq.cc及eventq.hh中

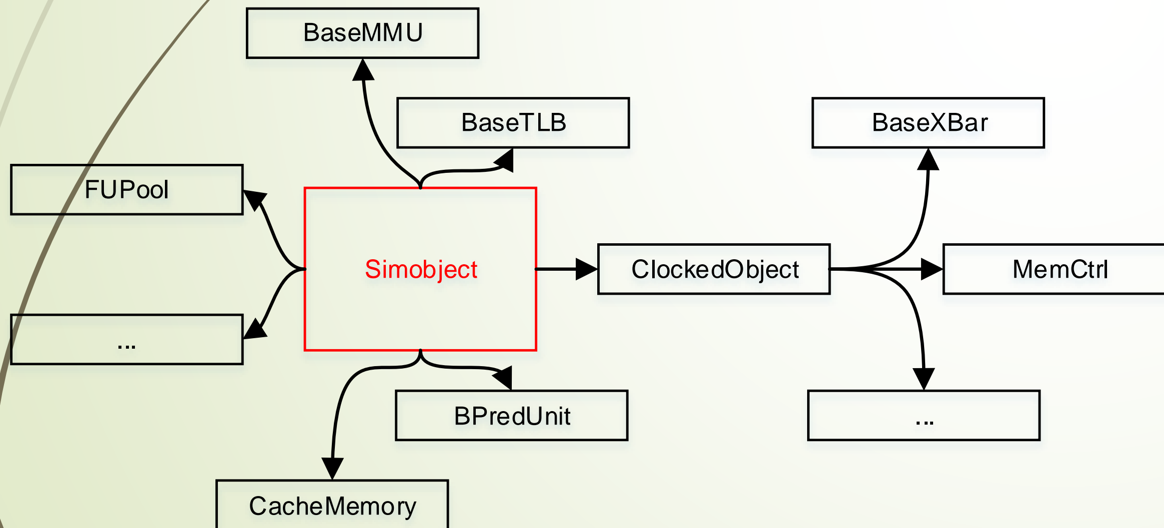


- serviceOne负责执行当前EventQueue中的一个事件
- insertBefore负责插入待执行的事件，一般由schedule触发
- 最终event->process()执行Event回调函数
- 从总入口上看来，整个事件机制的运行就是在一个大while循环中，不断调用eventQueue的serviceOne

很多框架，包括编程语言，都会抽象几个基础的类，作为整个框架构建的基础

Simobject

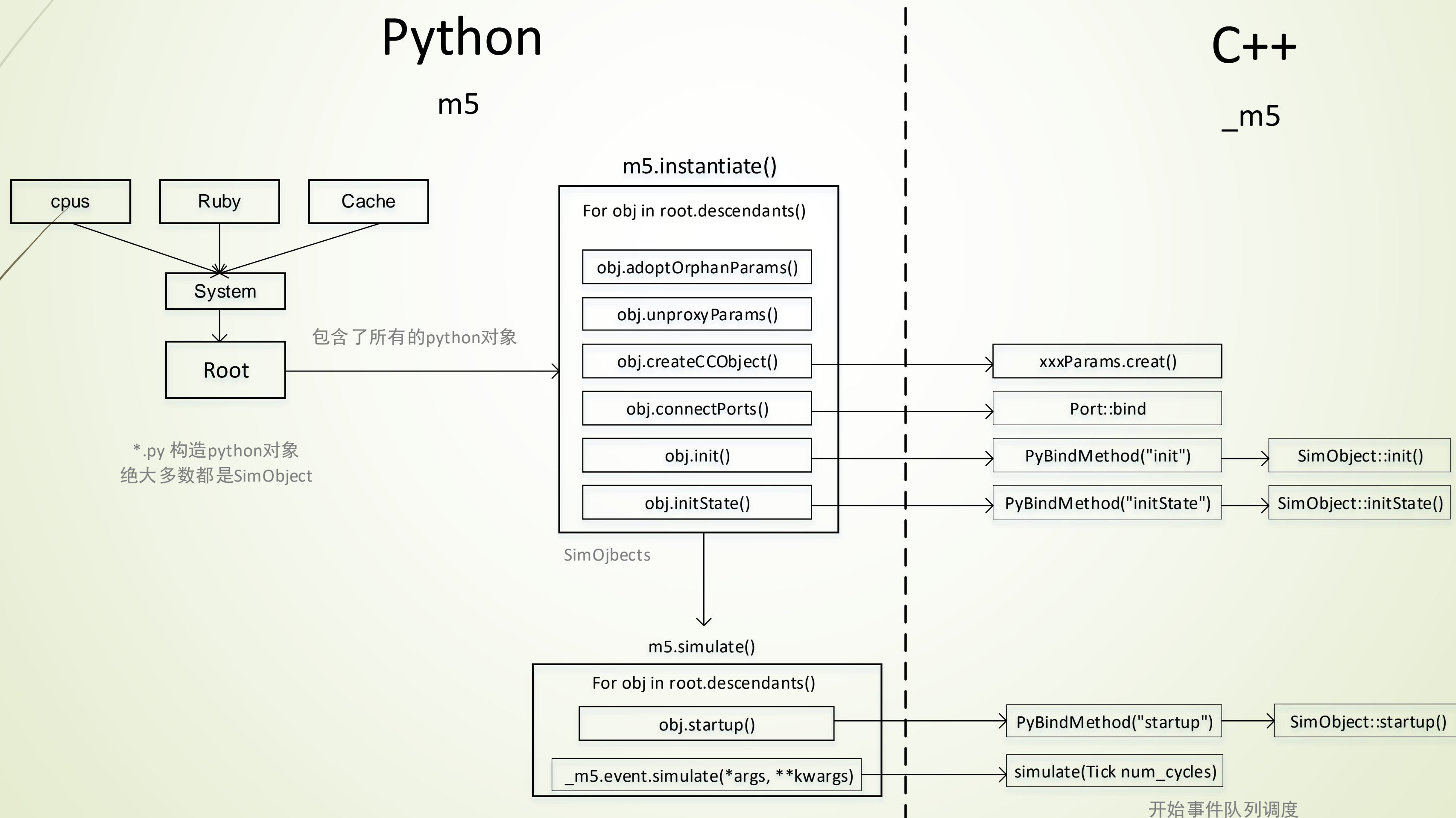
- Python层面能够对SimObject进行处理，python的功能之一就是对SimObject进行组合。实际上，gem5通过pybind11对SimObject进行了处理，使它能够在python侧进行操作
- 其中的param实现了数据从python到C++的传递
- 实现了一些基础接口，以便于在系统对这些实例同一管理




```cpp
class SimObject : public EventManager, public Serializable, public Drainable,
                  public statistics::Group, public Named
{
  private:
    typedef std::vector<SimObject *> SimObjectList;

    /** List of all instantiated simulation objects. */
    static SimObjectList simObjectList;

  protected:
    const SimObjectParams &_params;

  public:
    typedef SimObjectParams Params;
    const Params &params() const { return _params; }

    SimObject(const Params &p);

  public:
    virtual void init();

    virtual void loadState(CheckpointIn &cp);

    virtual void initState();

    virtual Port &getPort(const std::string &if_name,
                          PortID idx=InvalidPortID);

    virtual void startup();

    void serialize(CheckpointOut &cp) const override {};
    void unserialize(CheckpointIn &cp) override {};

    static SimObject *find(const char *name);
};
```

初始化过程

通过pybind11实现了c++的python封装，组成了gem5接口。Config目录下的python配置文件调用这些接口实现了特定的模拟器功能



内存系统的连接

不同的cpu, 不同的soc，都会有不一样的内存拓扑，作为一个通用模拟器，如何做到灵活可配？

- Ports are used to interface memory objects to each other
- They will always come in pairs, with a MasterPort and a SlavePort
- Every memory object has to have at least one port to be useful

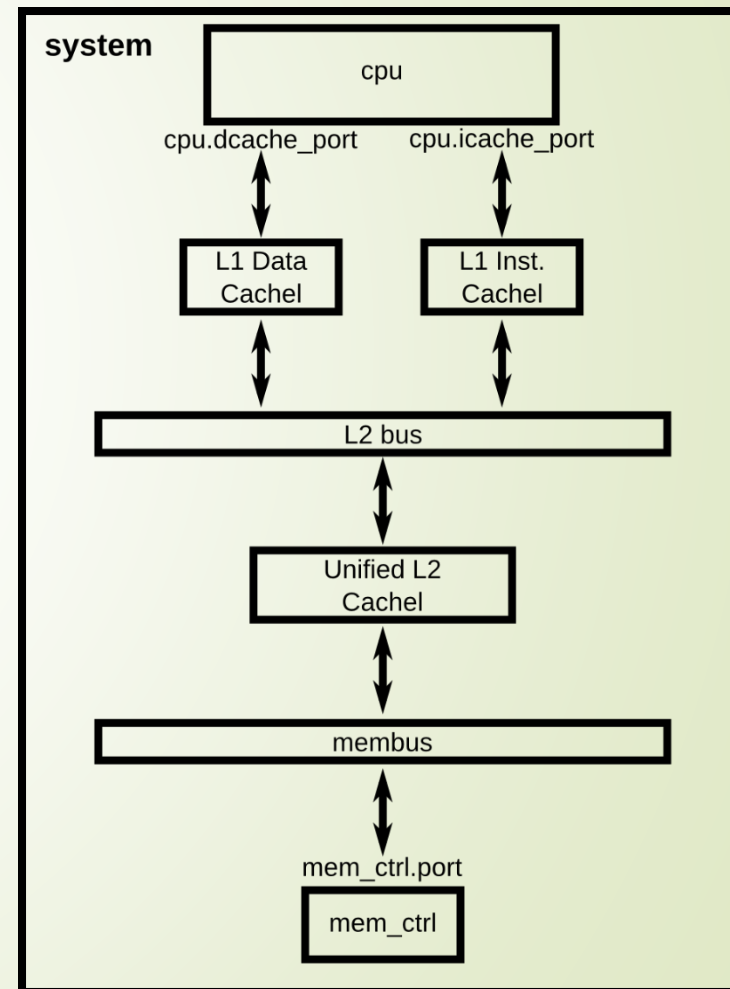
master

```
inline bool RequestPort::sendTimingReq(PacketPtr pkt)
{
    try {
        return TimingRequestProtocol::sendReq(_responsePort, pkt);
    } catch (UnboundPortException) {
        reportUnbound();
    }
}
```

```
Bool TimingRequestProtocol::sendReq(TimingResponseProtocol *peer, PacketPtr
pkt)
{
    assert(pkt->isRequest());
    return peer->recvTimingReq(pkt);
}
```

slave

```
Bool MemCtrl::recvTimingReq(PacketPtr pkt)
{
    // This is where we enter from the outside world
    DPRINTF(MemCtrl, "recvTimingReq: request %s addr %#x size %d\n",
            pkt->cmdString(), pkt->getAddr(), pkt->getSize());
```

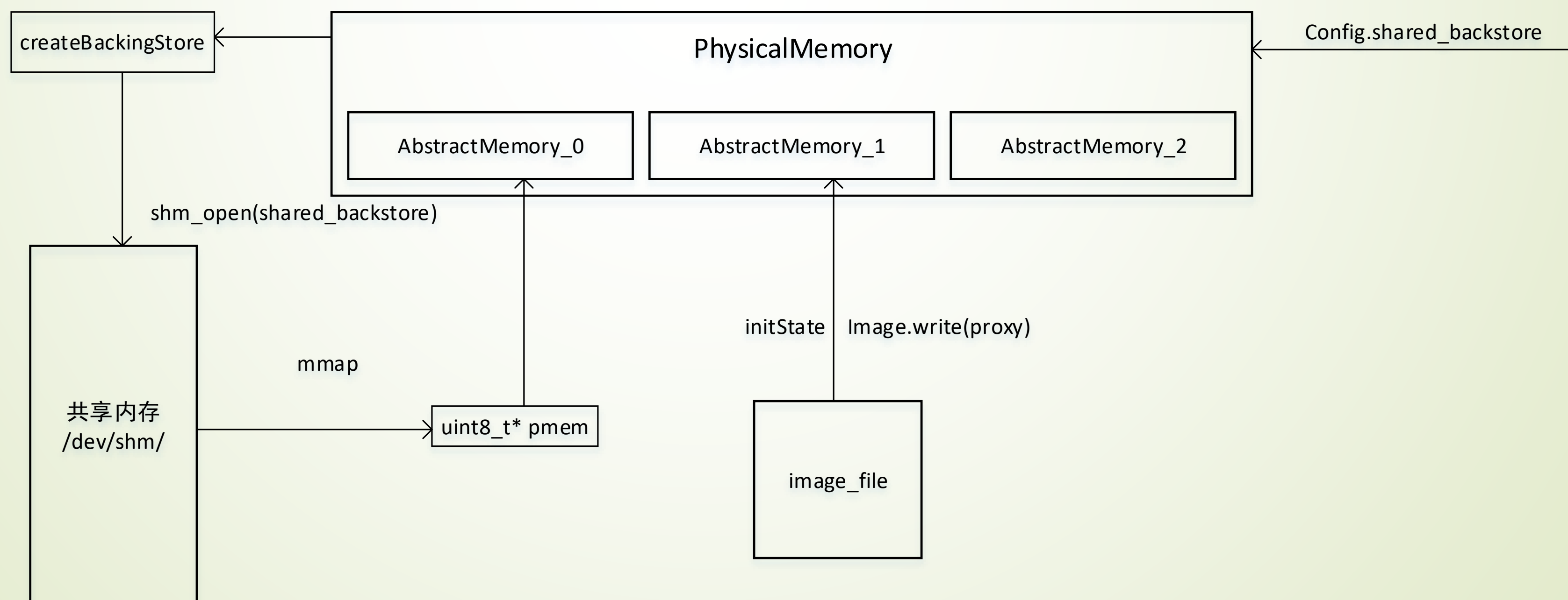


内存管理

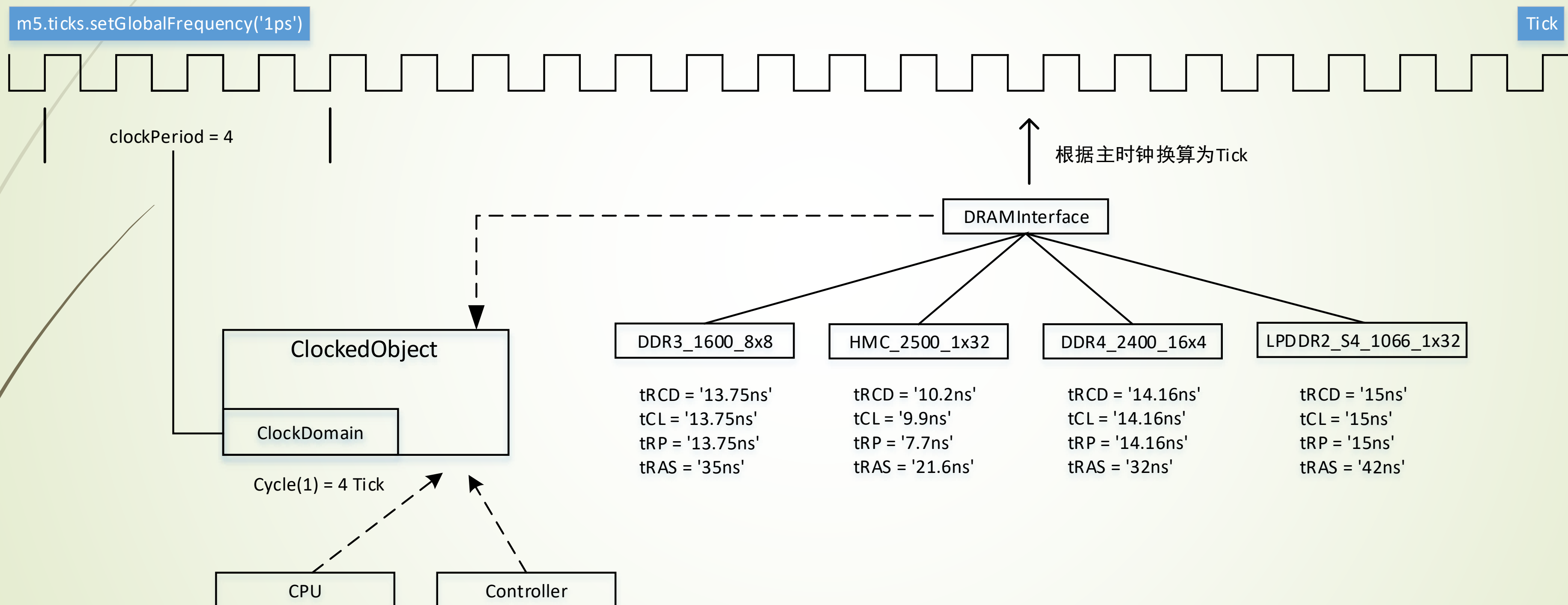
假如你要模拟一个CPU，它有一个2MB的SRAM，你会怎么实现？

模拟器在实现target内存单元的时候，往往都是在host上开辟一块内存，进行封装，抽象成target内存
比如SimpleScalar模拟器使用一级页表对host内存进行管理，从而实现target内存
Gem5中通过抽象内存和物理内存实现了内存管理，其中关键机制都在抽象内存中，物理内存相对简单

- Gem5的物理内存负责管理多个虚拟内存，可以通过配置输入内存backstore的路径，这个backstore是共享内存路径，可以将内存初始化文件丢入共享内存目录（/dev/shm）以达到初始化内存的目的，同时也可以使用共享内存完成进程间的数据传输
- 每个虚拟内存也提供一个image_file的配置参数，用来初始化内存
- 每一个system实例只有一个物理内存，所有的DDR，SRAM等都是虚拟内存



时钟系统



指令与架构分离

如何能最大化的做到同样的CPU微架构既能运行ARM指令，又能运行RISCV指令？

```
enum OpClass
    {
        No_OpClass = 0,
        IntAlu = 1,
        IntMult = 2,
        IntDiv = 3,
        FloatAdd = 4,
        FloatCmp = 5,
        FloatCvt = 6,
        FloatMult = 7,
        FloatMultAcc = 8,
        FloatDiv = 9,
```

```
enum Flags
    {
        IsNop = 0,
        IsInteger = 1,
        IsFloating = 2,
        IsVector = 3,
        IsVectorElem = 4,
        IsLoad = 5,
        IsStore = 6,
        IsAtomic = 7,
        IsStoreConditional = 8,
        IsInstPrefetch = 9,
        IsDataPrefetch = 10,
        IsControl = 11,
        IsDirectControl = 12,
        IsIndirectControl = 13,
```

- opClass
- Flags
- 编解码信息（包含了寄存器号）
- 结果计算（如果是计算类指令）
- 反汇编信息

只需要掌握这些信息，而不需要知道到底是哪条指令

```
    virtual Fault execute(ExecContext *xc,
            Trace::InstRecord *traceData) const = 0;
```

```
virtual std::string generateDisassembly(
            Addr pc, const loader::SymbolTable *symtab) const = 0;
```

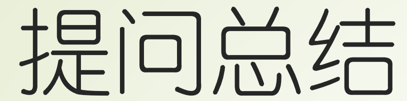
执行阶段如何做到准确

```python
class MinorDefaultIntFU(MinorFU):
    opClasses = minorMakeOpClassSet(['IntAlu'])
    timings = [MinorFUTiming(description="Int",
        srcRegsRelativeLats=[2])]
    opLat = 3

class MinorDefaultIntMulFU(MinorFU):
    opClasses = minorMakeOpClassSet(['IntMult'])
    timings = [MinorFUTiming(description='Mul',
        srcRegsRelativeLats=[0])]
    opLat = 3

class MinorDefaultIntDivFU(MinorFU):
    opClasses = minorMakeOpClassSet(['IntDiv'])
    issueLat = 9
    opLat = 9

class MinorDefaultFloatSimdFU(MinorFU):
    opClasses = minorMakeOpClassSet([
        'FloatAdd', 'FloatCmp', 'FloatCvt', 'FloatMisc', 'FloatMult',
        'FloatMultAcc', 'FloatDiv', 'FloatSqrt',
        'SimdAdd', 'SimdAddAcc', 'SimdAlu', 'SimdCmp', 'SimdCvt',
        'SimdAes', 'SimdAesMix',
        'SimdSha1Hash', 'SimdSha1Hash2', 'SimdSha256Hash',
        'SimdSha256Hash2', 'SimdShaSigma2', 'SimdShaSigma3'])

    timings = [MinorFUTiming(description='FloatSimd',
        srcRegsRelativeLats=[2])]
    opLat = 6
```

- 总体来讲，模拟器会采取和RTL类似的反压机制，只不过是把RTL的信号改为了变量
- 但为了更灵活，模拟器对一些过程进行了抽象，使得其更灵活、通用、高效
- issueLat参与决定发射间隔
- opLat决定指令执行时间
- 配合scoreBoard来控制寄存器的valid
- 额外的Lat参数来实现bypass之类的延迟控制，只需要设置这些参数就能拟合实际的电路延迟

执行周期不确定的指令如何处理？比如load指令遇到cache miss ？

mark_unpredictable



提问总结

- Gem5是开源软件，社区比较活跃，代码变动比较大。比如之前OOO CPU是用模板类实现的，现在舍弃了模板类。比如之前的TickedObject很多，现在几乎看不到了，主要是ClockedObject
- 但总体上，有一些核心思想是不变的。比如事件机制，比如如何做到cycle级模拟，这些都是通用思想，万变不离其宗