gem5 快速上手

顾宇浩

关于我……

- 顾宇浩
- 2022 届学硕
- 超算中心 302 实验室
- 个人主页：https://yhgu2000.github.io
- 研究方向：编译器、编程语言、虚拟机

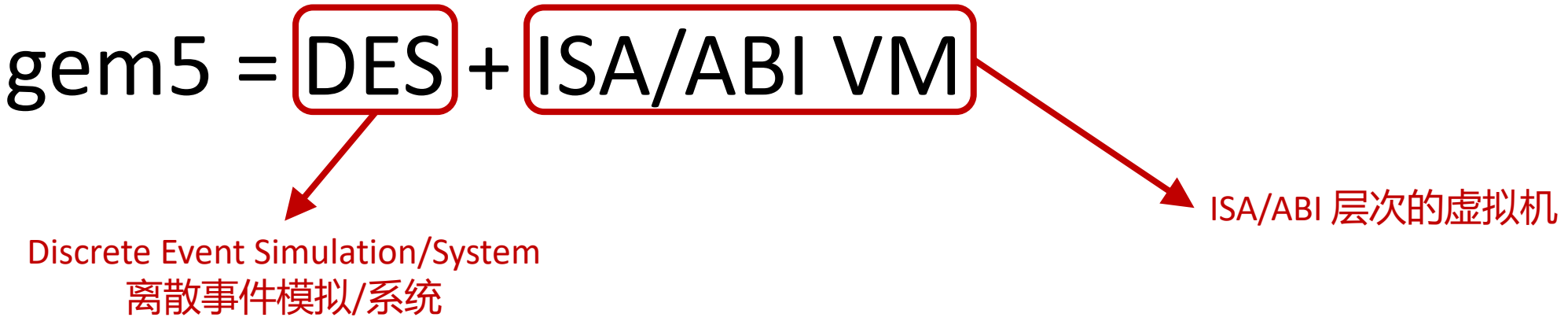
gem5 = DES + ISA/ABI VM

gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

gem5 是什么东西？

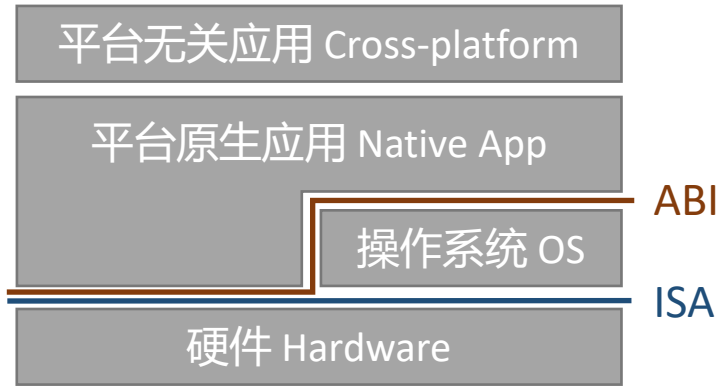

gem5 = DES + ISA/ABI VM
Discrete Event Simulation/System
离散事件模拟/系统
ISA/ABI 层次的虚拟机

gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

ISA/ABI 层次的虚拟机

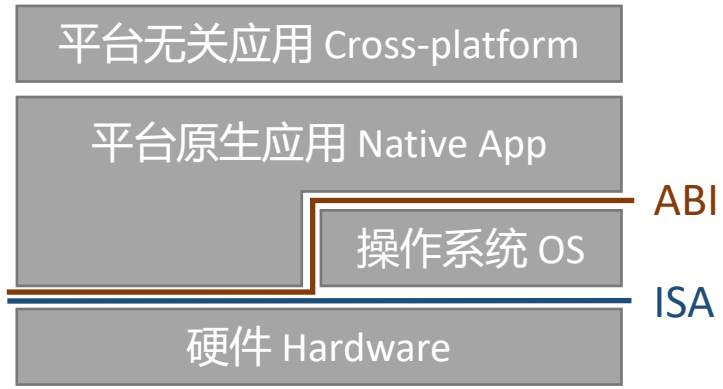


gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

ISA/ABI 层次的虚拟机



ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

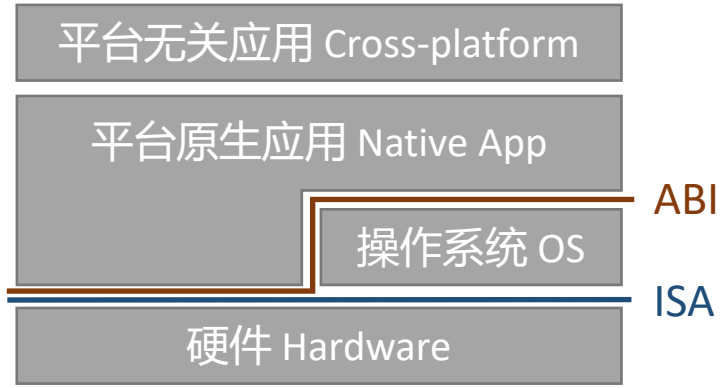
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

事件队列

ISA/ABI 层次的虚拟机



ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

gem5 是什么东西？

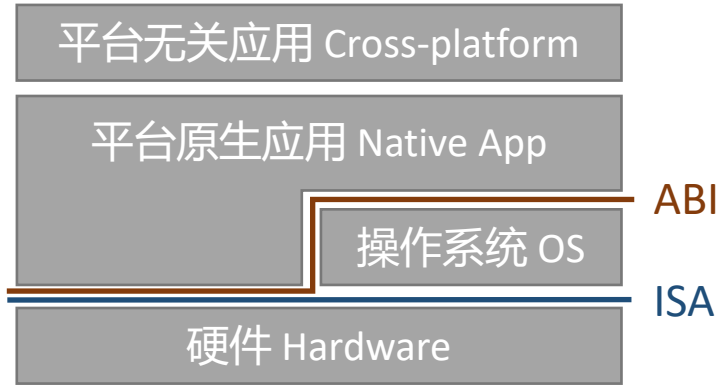
gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

事件队列

调度器

ISA/ABI 层次的虚拟机

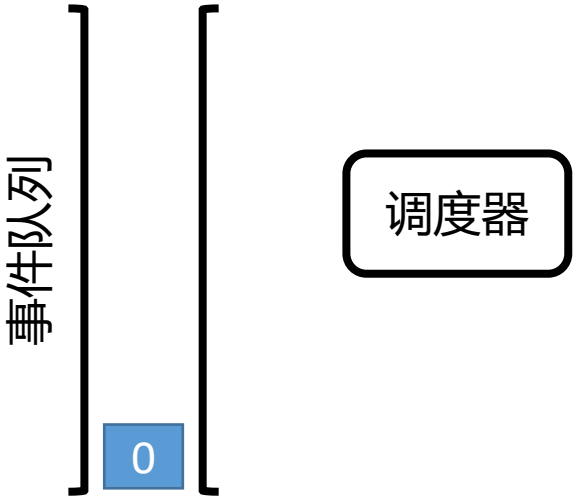


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

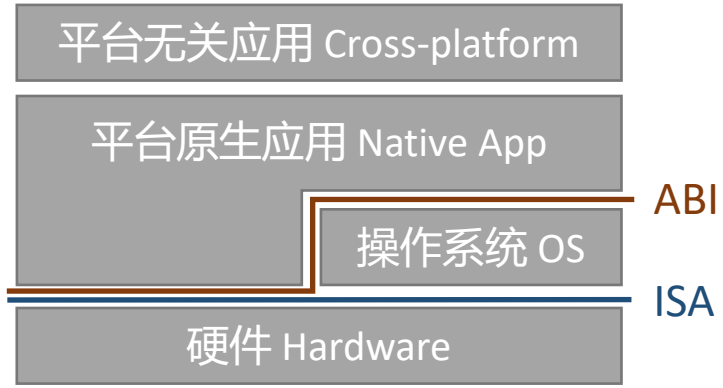
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



ISA/ABI 层次的虚拟机

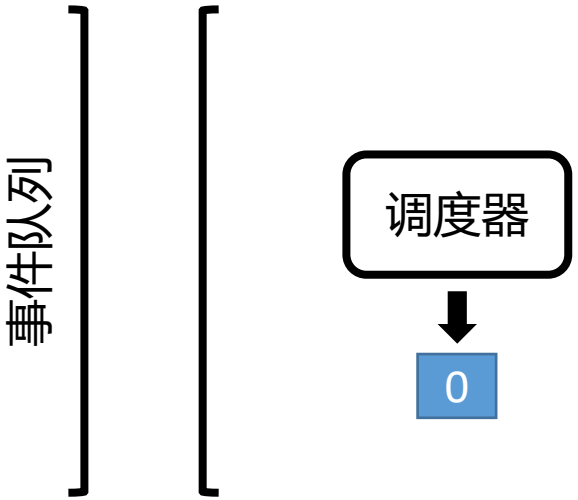


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

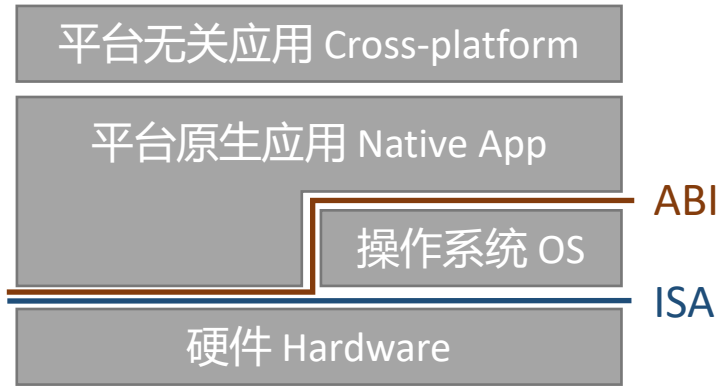
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



ISA/ABI 层次的虚拟机

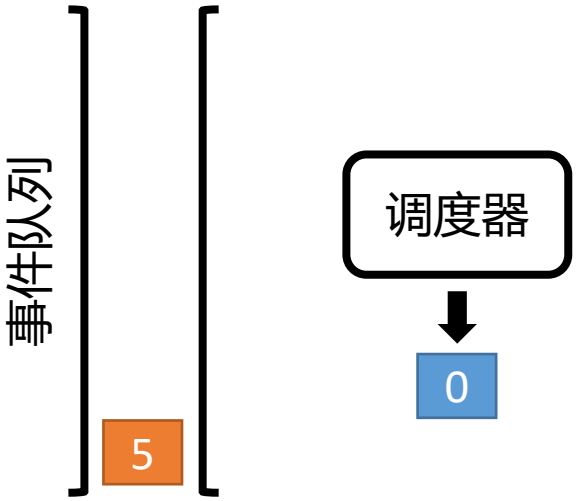


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

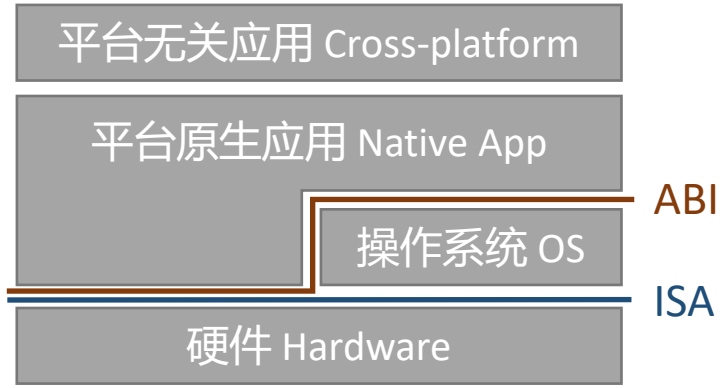
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



ISA/ABI 层次的虚拟机

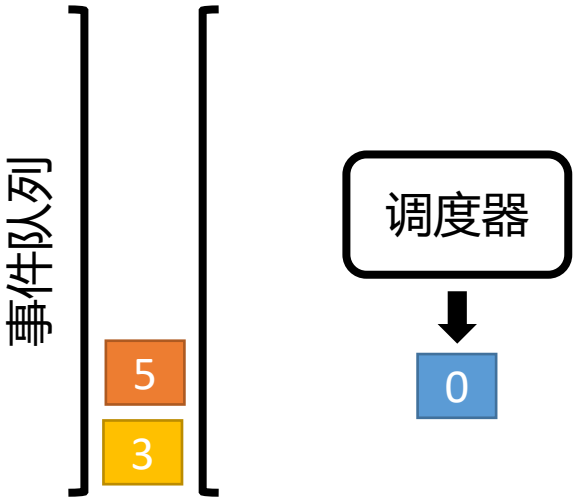


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

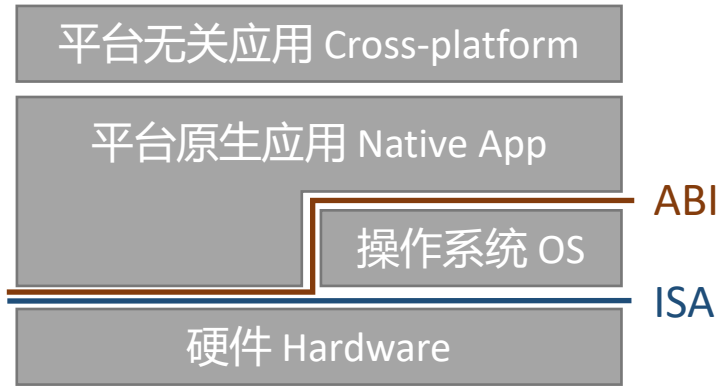
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



ISA/ABI 层次的虚拟机

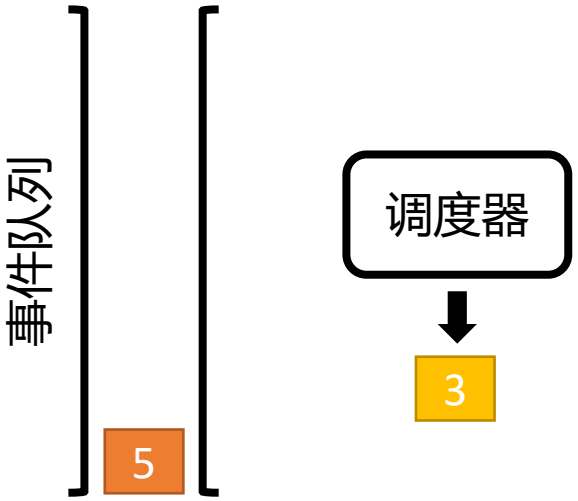


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

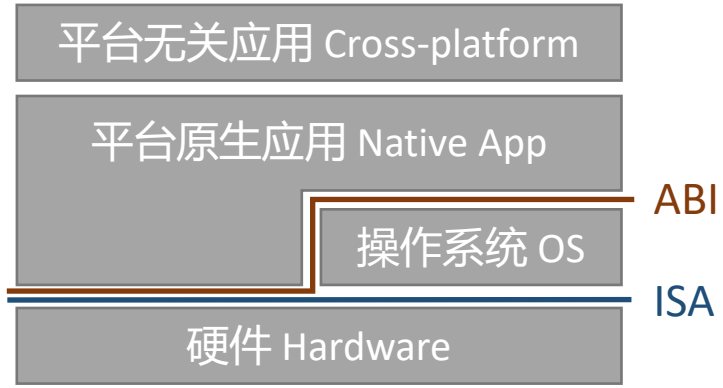
gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



ISA/ABI 层次的虚拟机

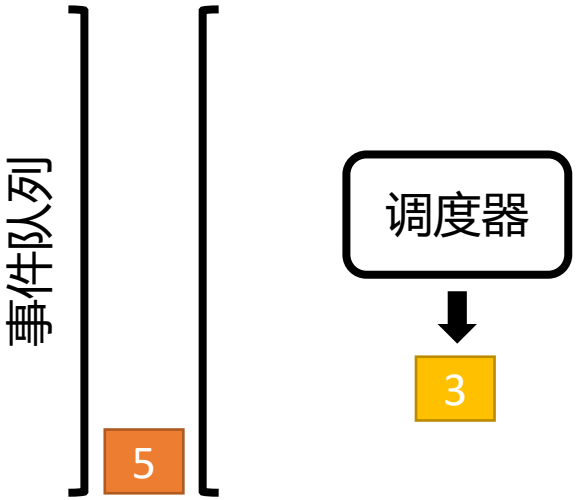


ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

gem5 是什么东西？

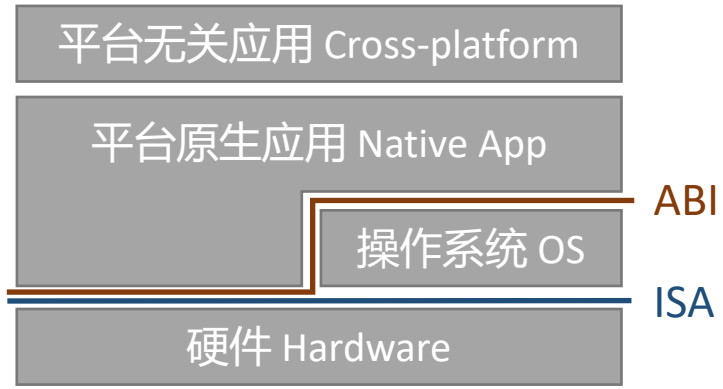
gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统



- 事件的发生在时间上是离散的
- 系统在事件发生之间没有状态变化
- 不同事件可能以任意的顺序出现

ISA/ABI 层次的虚拟机

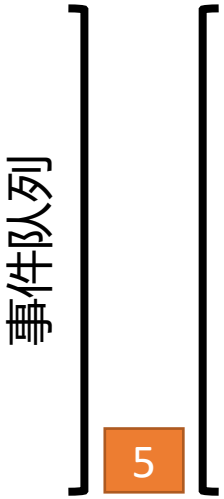


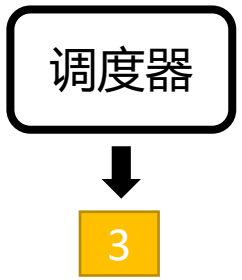
ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

gem5 是什么东西？

gem5 = DES + ISA/ABI VM

Discrete Event Simulation/System
离散事件模拟/系统

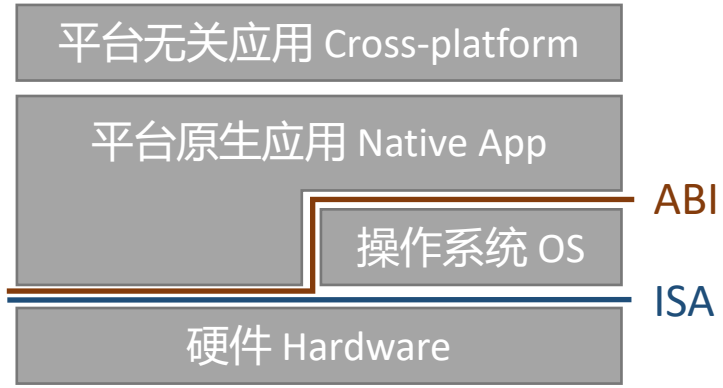




- 事件的发生在时间上是离散的
- 系统在事件发生之间没有状态变化
- 不同事件可能以任意的顺序出现

典型应用：GUI

ISA/ABI 层次的虚拟机



ISA层：VMware、VirtualBox
ABI层：Rosetta转译层
高级：JVM(Java)、V8(JavaScript)……

安装&配置开发环境

https://www.gem5.org/documentation/general_docs/building

安装&配置开发环境

https://www.gem5.org/documentation/general_docs/building

- 使用Docker：https://gty111.github.io/SYSU-ARCH/docs/LAB1/LAB1.html

安装&配置开发环境

https://www.gem5.org/documentation/general_docs/building

- 使用Docker：https://gty111.github.io/SYSU-ARCH/docs/LAB1/LAB1.html
- 手工配置
  - git clone https://gem5.googlesource.com/public/gem5
  - 使用apt安装依赖

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
    libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
    python3-dev python-is-python3 libboost-all-dev pkg-config
```

  - 在仓库根目录中执行：

```
scons build/X86/gem5.opt -j4
```

  - 运行：./build/X86/gem5.opt --help

安装&配置开发环境

https://www.gem5.org/documentation/general_docs/building

- 手工配置
  - git clone https://gem5.googlesource.com/public/gem5
  - 使用apt安装依赖

```
sudo apt install build-essential git m4 scons zlib1g zlib1g-dev \
    libprotobuf-dev protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
    python3-dev python-is-python3 libboost-all-dev pkg-config
```

  - 在仓库根目录中执行：

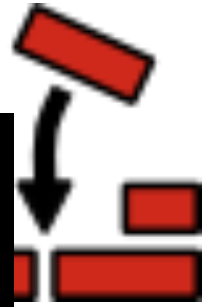
```
scons build/X86/gem5.opt -j4
```

  - 运行：./build/X86/gem5.opt --help

SCons是什么

- https://scons.org
- 一个构建系统，可用于C、 C++、 D、 Java、 Fortran……
  联想Linux中的make
- 特点：基于Python，图灵完备
- *好用吗？也许吧！*

SCon是什么



- https:/
- 一个构
  联想L
- 特点：
- *好用吗*

```
[     CXX] X86/python/gem5/components/memory/dram_interfaces/wideio.py.cc -> .pyo
[     CXX] X86/python/gem5/components/cachehierarchies/chi/nodes/private_l1_moesi_cache.py.cc -> .pyo
[     CXX] X86/mem/MemInterface.py.cc -> .pyo
[EMBED PY] X86/python/gem5/prebuilt/demo/x86_demo_board.py -> .cc
[     CXX] X86/python/gem5/prebuilt/demo/x86_demo_board.py.cc -> .pyo
[EMBED PY] X86/python/gem5/components/cachehierarchies/ruby/topologies/__init__.py -> .cc
[EMBED PY] X86/arch/generic/InstDecoder.py -> .cc
[EMBED PY] X86/mem/AddrMapper.py -> .cc
[     CXX] X86/python/gem5/components/cachehierarchies/ruby/topologies/__init__.py.cc -> .pyo
[     CXX] X86/arch/generic/InstDecoder.py.cc -> .o
[     CXX] X86/mem/AddrMapper.py.cc -> .pyo
[     CXX] X86/arch/generic/InstDecoder.py.cc -> .pyo
[EMBED PY] X86/arch/x86/AtomicSimpleCPU.py -> .cc
[EMBED PY] X86/cpu/simple/BaseSimpleCPU.py -> .cc
[EMBED PY] X86/mem/MemChecker.py -> .cc
[ TRACING]  -> X86/debug/Faults.cc
[EMBED PY] X86/dev/Device.py -> .cc
[EMBED PY] X86/python/gem5/utils/requires.py -> .cc
[     CXX] X86/arch/x86/AtomicSimpleCPU.py.cc -> .pyo
[     CXX] X86/cpu/simple/BaseSimpleCPU.py.cc -> .pyo
[     CXX] X86/arch/x86/AtomicSimpleCPU.py.cc -> .o
[     CXX] X86/dev/Device.py.cc -> .pyo
[     CXX] X86/mem/MemChecker.py.cc -> .pyo
[EMBED PY] X86/mem/Bridge.py -> .cc
[EMBED PY] X86/learning_gem5/part2/SimpleMemobj.py -> .cc
[     CXX] X86/python/gem5/utils/requires.py.cc -> .pyo
[EMBED PY] X86/mem/HMCController.py -> .cc
[EMBED PY] X86/python/gem5/prebuilt/__init__.py -> .cc
[EMBED PY] X86/python/m5/util/convert.py -> .cc
[EMBED PY] X86/python/gem5/components/cachehierarchies/ruby/caches/mesi_two_level/l2_cache.py -> .cc
[     CXX] X86/learning_gem5/part2/SimpleMemobj.py.cc -> .pyo
[     CXX] X86/mem/Bridge.py.cc -> .pyo
[     CXX] X86/mem/HMCController.py.cc -> .pyo
[     CXX] X86/python/gem5/prebuilt/__init__.py.cc -> .pyo
[     CXX] X86/python/gem5/components/cachehierarchies/ruby/caches/mesi_two_level/l2_cache.py.cc -> .pyo
[     CXX] X86/python/m5/util/convert.py.cc -> .pyo
```

使用gem5

- gem5的用户界面就是Python解释器

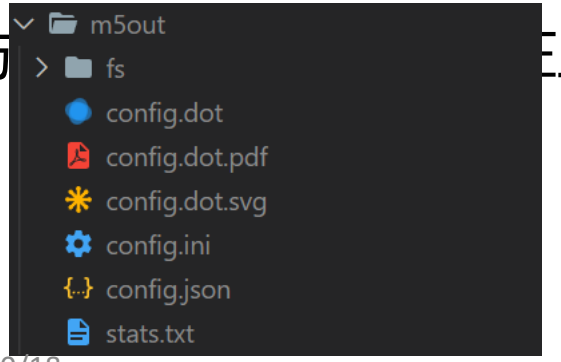
  通过编写Python脚本配置仿真的系统，在脚本里调用Python函数m5.simulate()执行仿真

  官方教程中的示例：configs/learning_gem5/part1/simple.py

```python
# 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()))
```

- 执行脚本：./build/X86/gem5.opt configs/learning_gem5/part1/simple.py

- 仿[illegible]工作目录中生成m5out目录



config.* 记录了仿真运行前的所有参数配置

stats.txt 仿真结果，所有的统计数据

使用gem5

- gem5的用户界面就是Python解释器

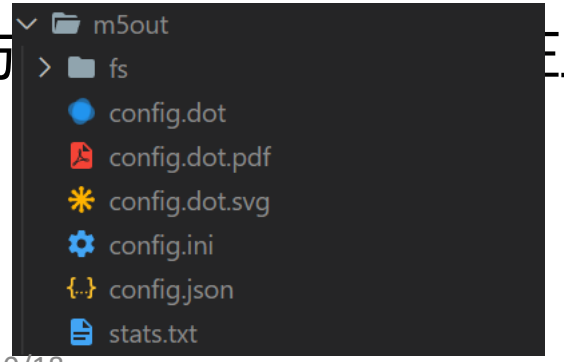
  通过编写Python脚本配置仿真的系统，在脚本里调用Python函数m5.simulate()执行仿真

  官方教程中的示例：configs/learning_gem5/part1/simple.py

```python
# 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()))
```

- 执行脚本：./build/X86/gem5.opt configs/learning_gem5/part1/simple.py

- 仿[illegible]工作目录中生成m5out目录



config.* 记录了仿真运行前的所有参数配置

stats.txt 仿真结果，所有的统计数据

Python脚本要怎么写呢？

```python
# create the system we are going to simulate
system = System()

# Set the clock frequency 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_mode = 'timing'               # Use timing accesses
system.mem_ranges = [AddrRange('512MB')] # Create an address range

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

# Create a memory bus, a system crossbar, in this case
system.membus = SystemXBar()

# 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
```

第一个仿真：首次适应内存分配

First-Fit Memory Allocation

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

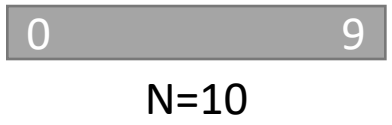
- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1



问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1

N=10　　malloc(3) -> 0

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1

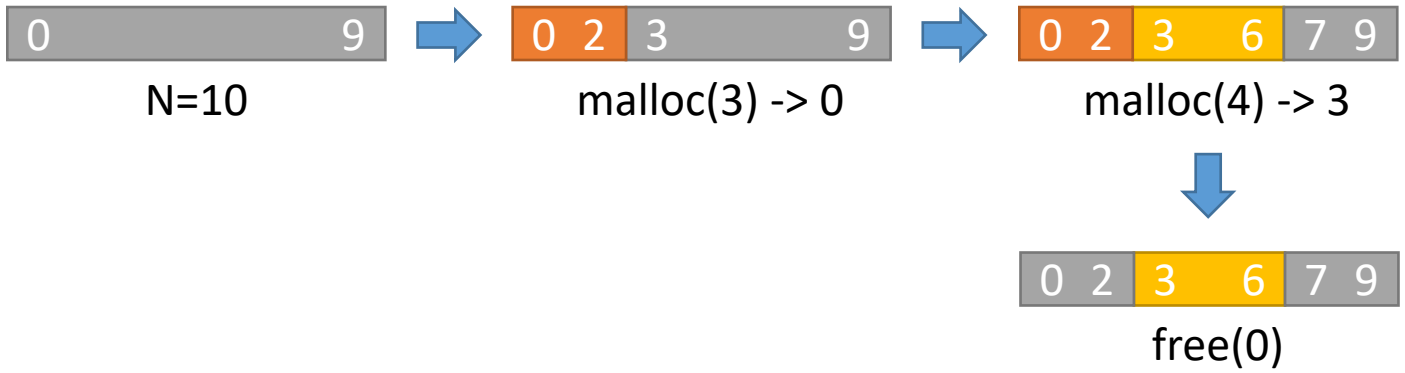
N=10　　malloc(3) -> 0　　malloc(4) -> 3

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1

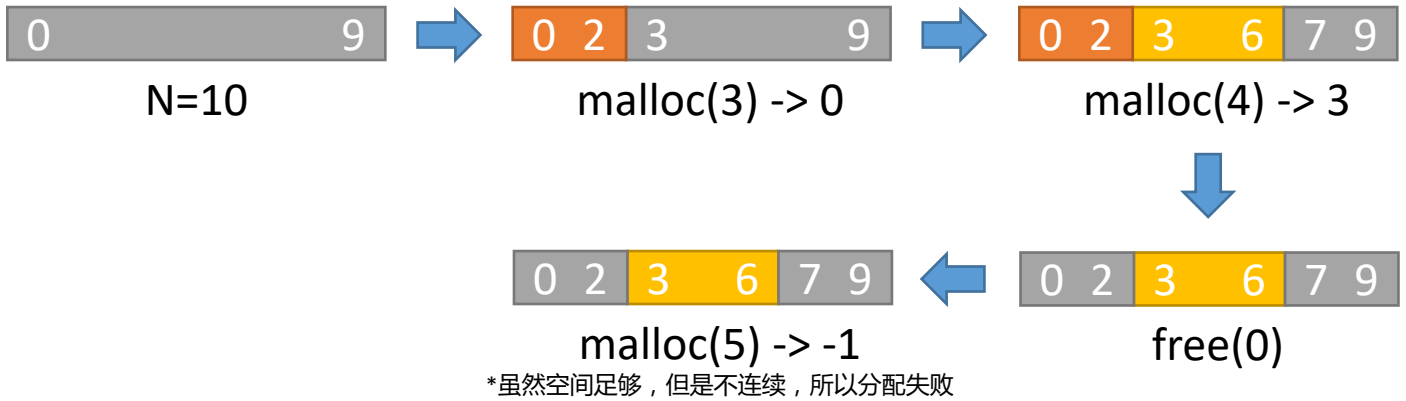


问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1



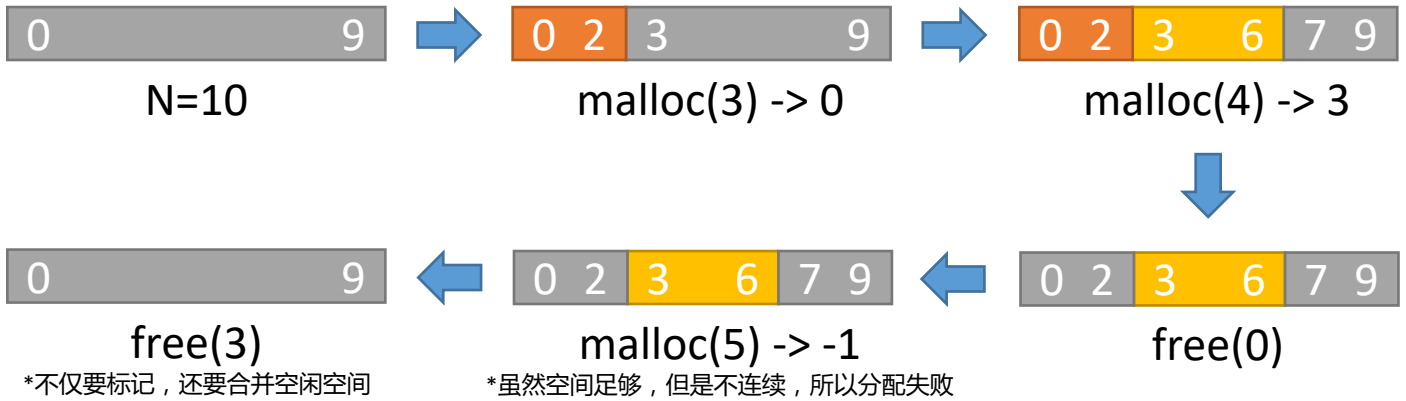
*虽然空间足够，但是不连续，所以分配失败

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1



*不仅要标记，还要合并空闲空间

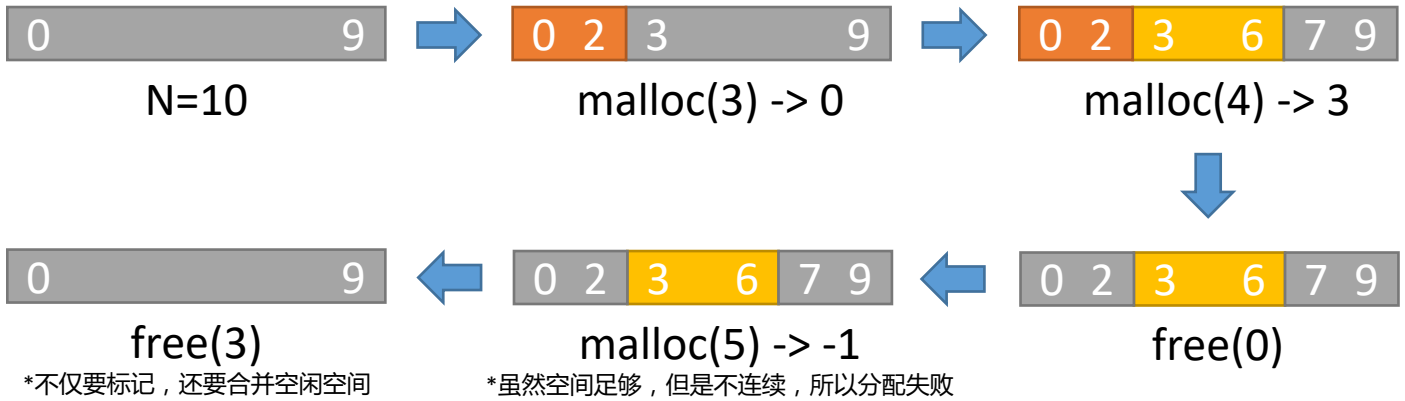
*虽然空间足够，但是不连续，所以分配失败

问题描述

- `void* malloc(size_t size);`
  在堆中分配不少于 `size` 大小的连续内存段，返回起始地址指针，失败则返回`nullptr`
- `void free(void* ptr);`
  释放之前由 `malloc` 返回的指针，回收指针指向的内存段以待下次分配

简化：

- 将内存资源看作是N个字节的数组，地址从0到N-1
- 每次内存分配以字节为单位进行，如果分配失败则返回-1



*不仅要标记，还要合并空闲空间

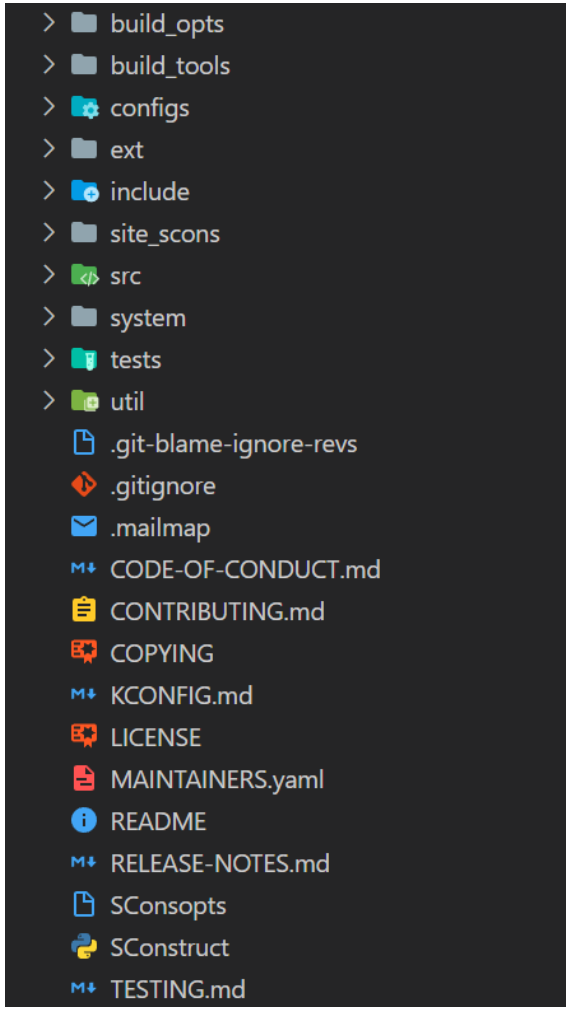
*虽然空间足够，但是不连续，所以分配失败

给定N和一张表：

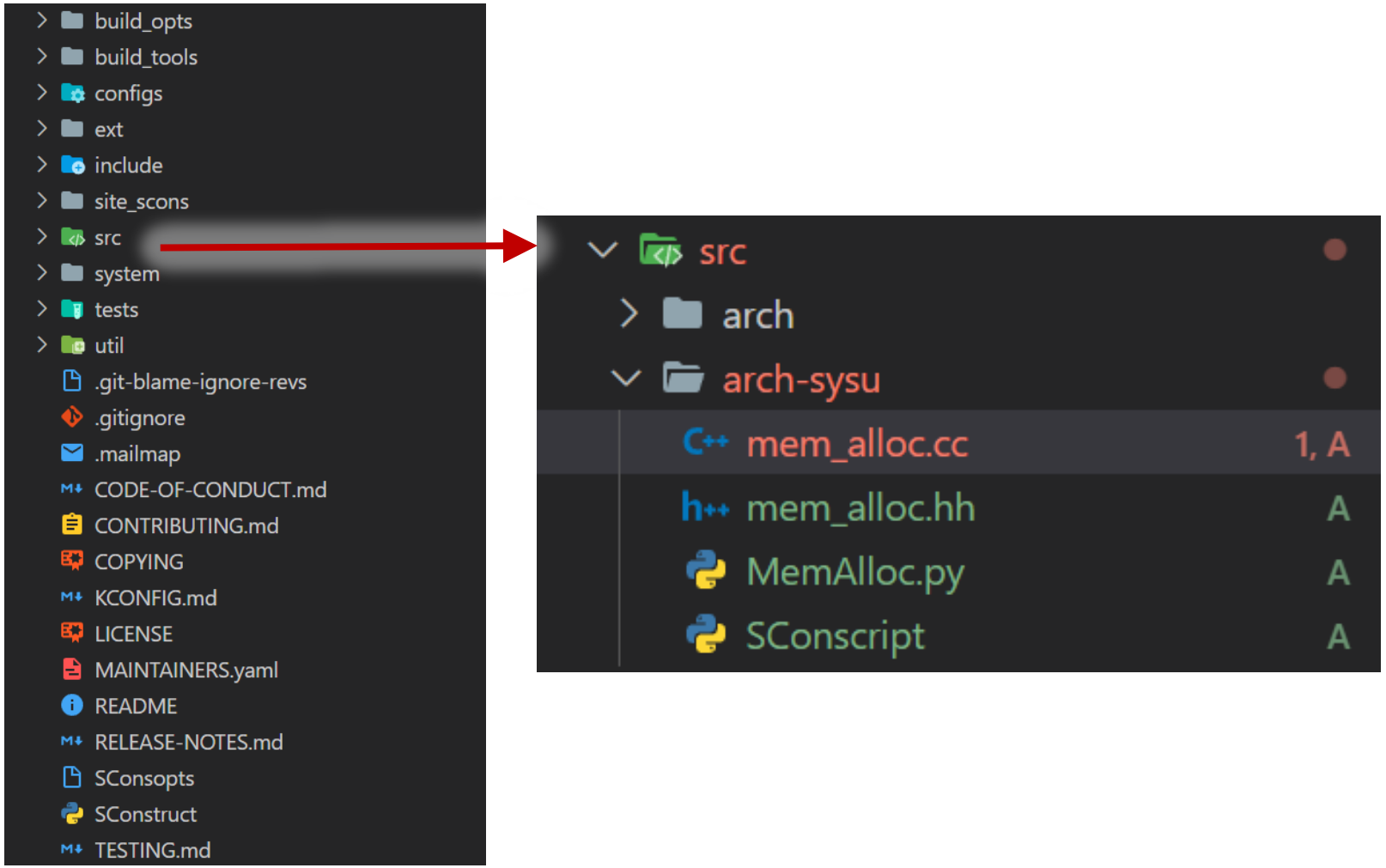
时刻	0	1	3	7	6	10	…
大小	1	5	3	4	7	2	…
生存期	9	4	5	1	2	9	…

是否所有的分配请求都能被满足？

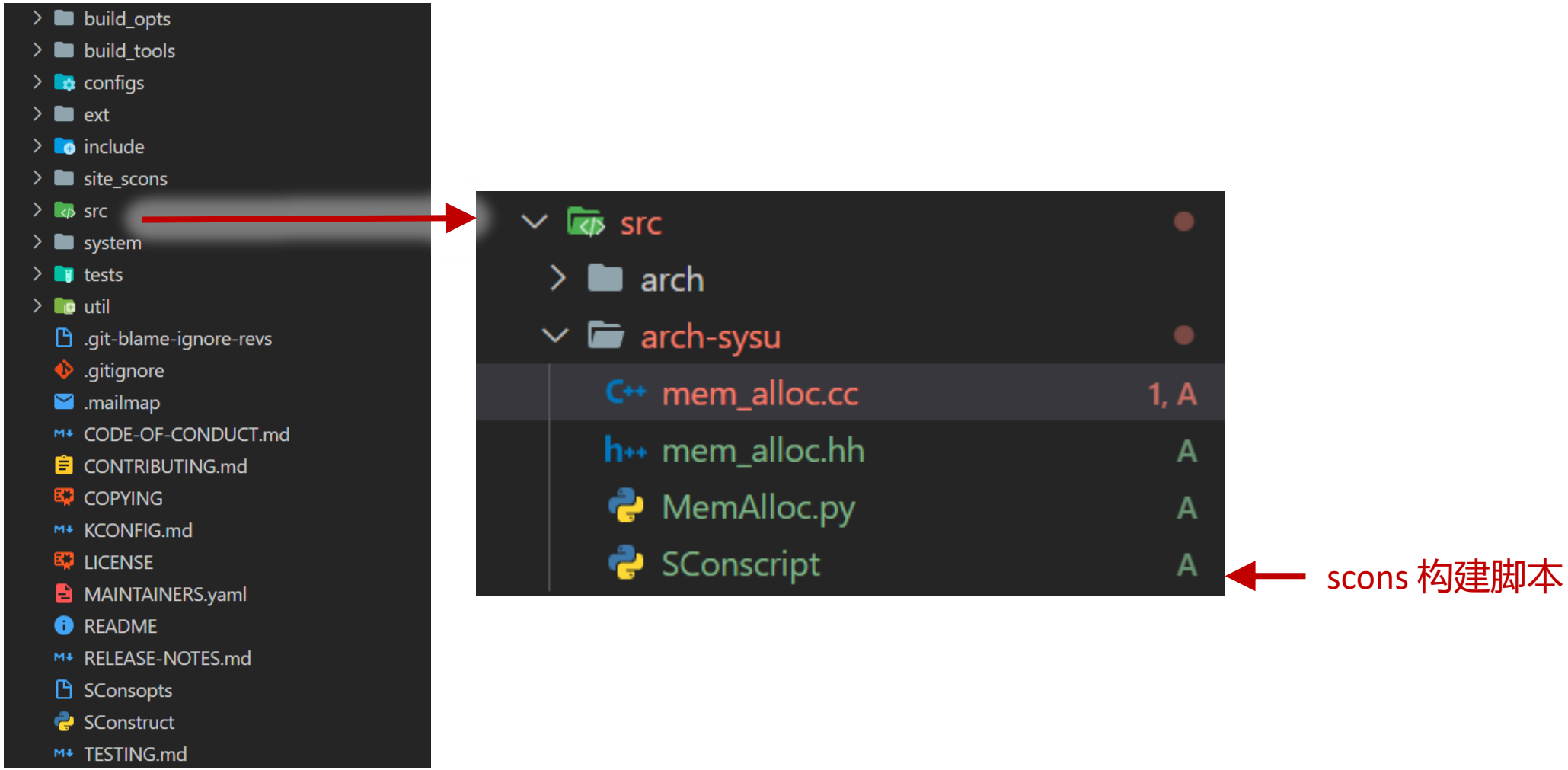
Let’s code!（创建文件）



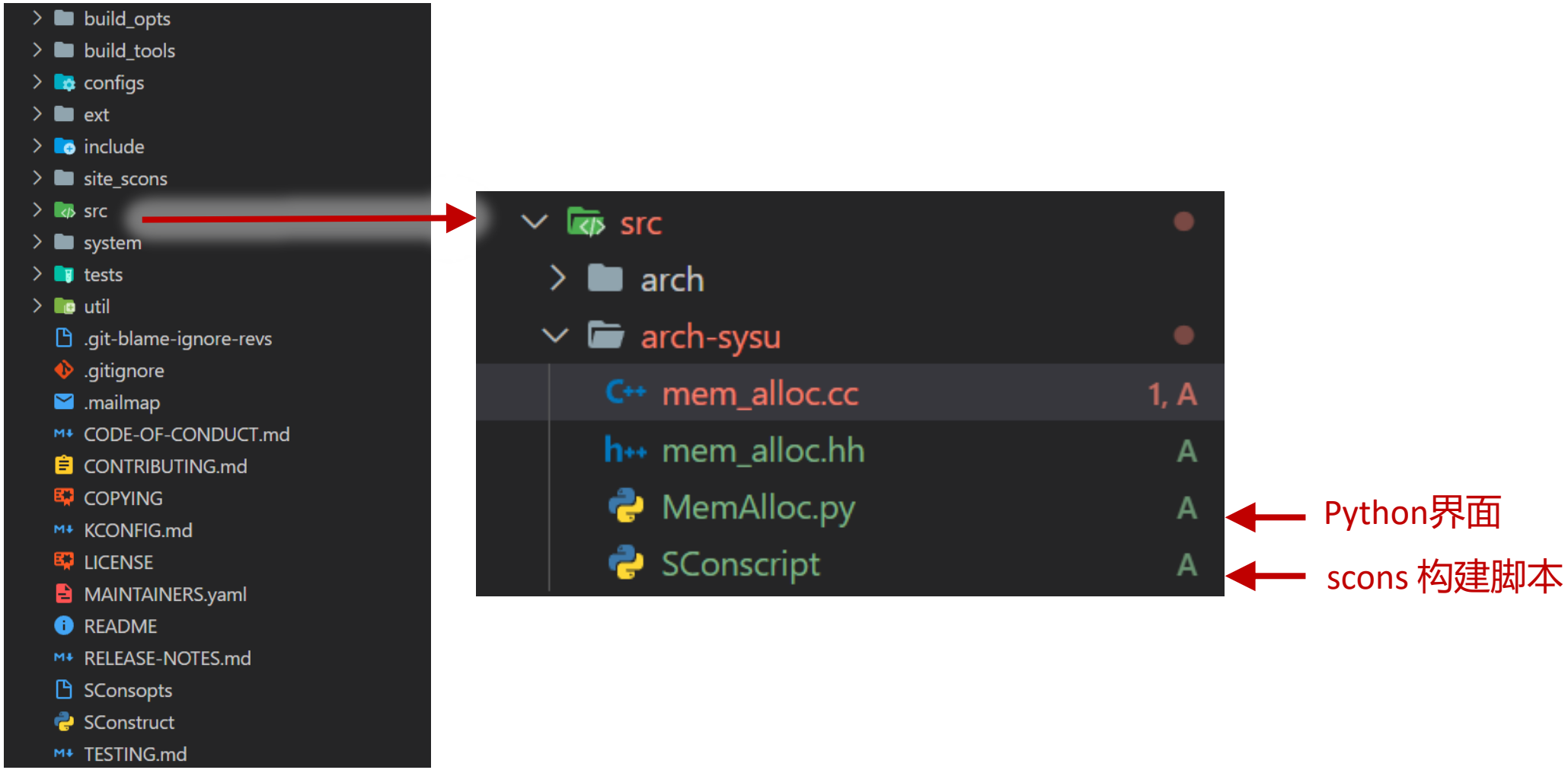
Let’s code!（创建文件）



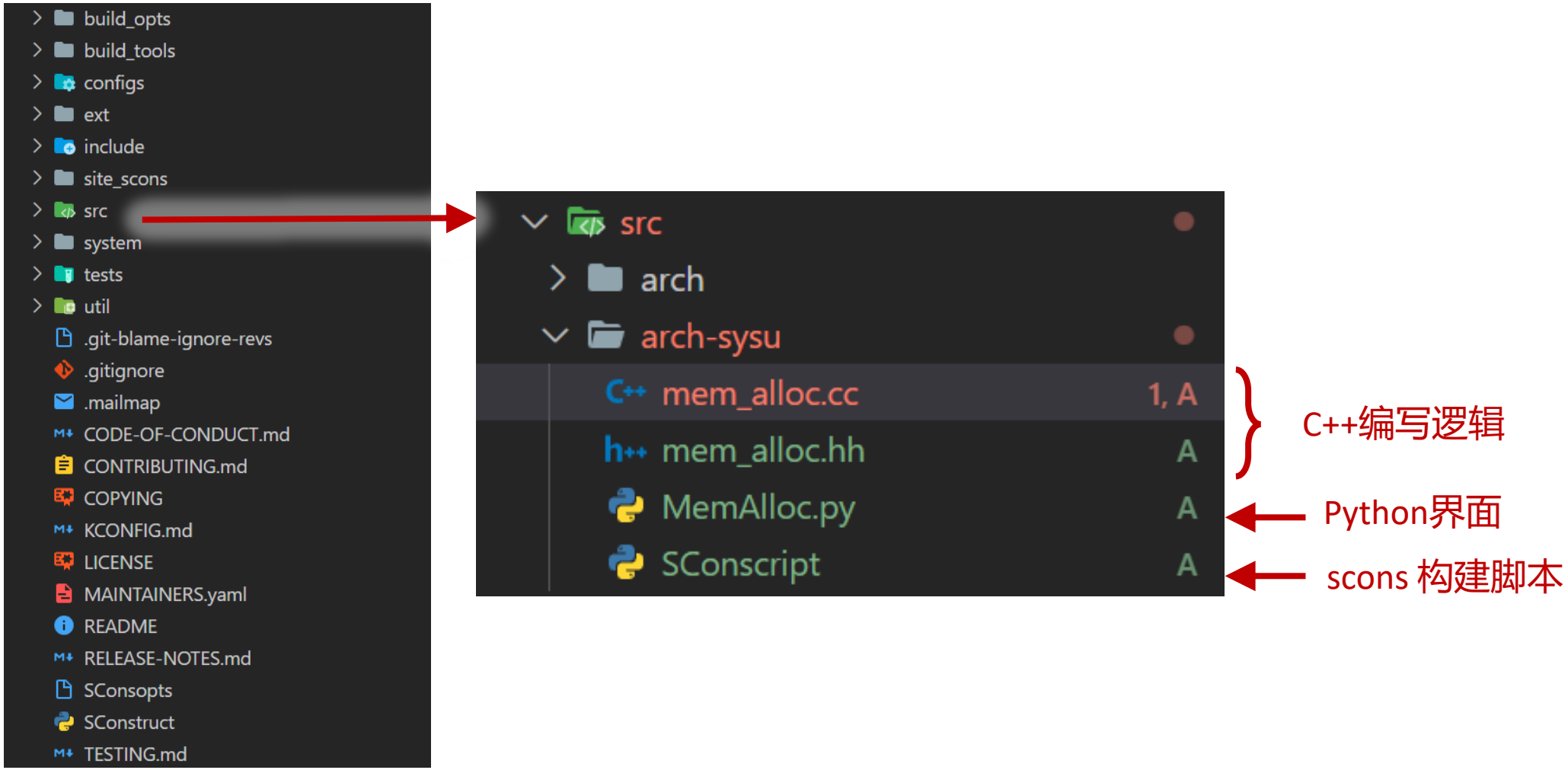
Let’s code!（创建文件）



Let’s code!（创建文件）

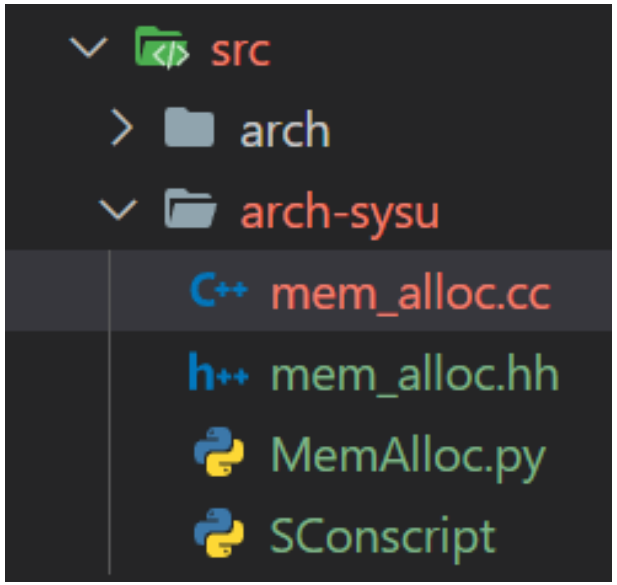


Let’s code!（创建文件）

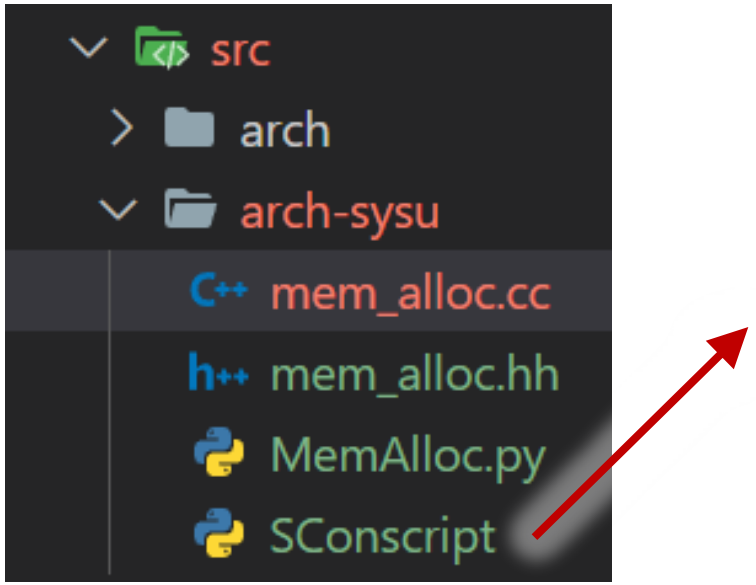


Let’s code!（SConscript）

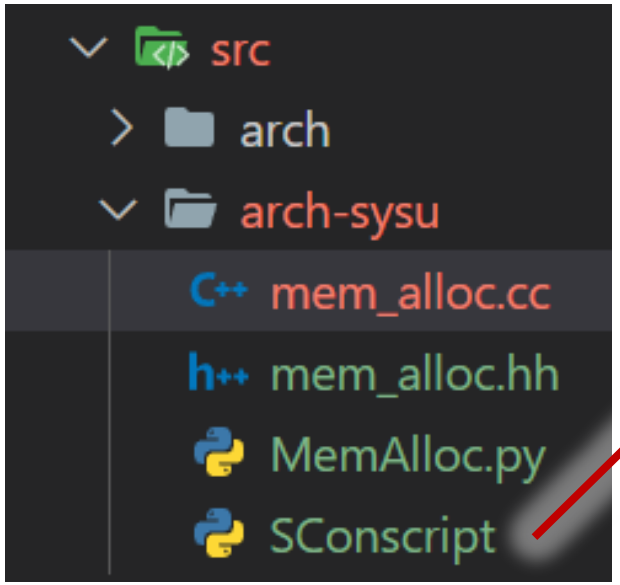
Let’s code!（ SConscript ）



Let’s code!（ SConscript ）

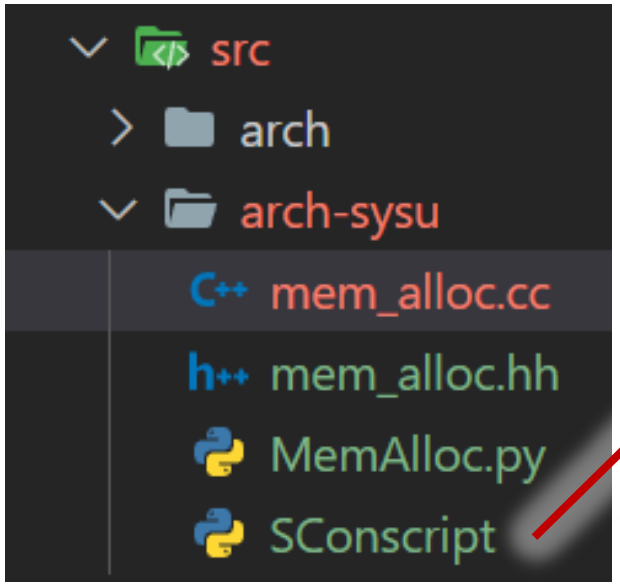


Let’s code!（ SConscript ）



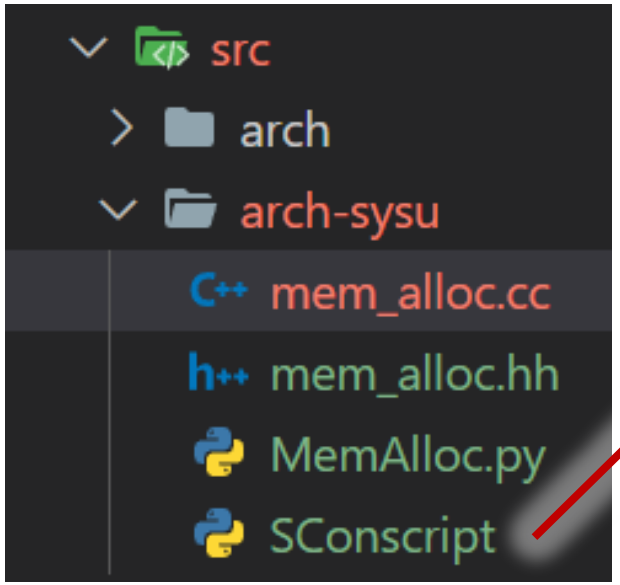

```
1   # 导入构建过程中的所有其它符号（固定写法）
2   Import('*')
```

Let’s code!（ SConscript ）



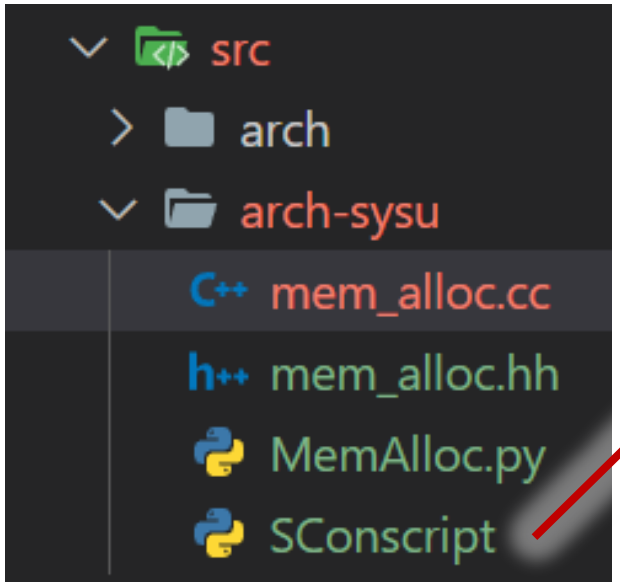

```
# 导入构建过程中的所有其它符号（固定写法）
Import('*')

# 声明要添加哪些 SimObject
SimObject('MemAlloc.py', sim_objects=['MemAlloc', 'MemRequest'])
```

Let’s code!（ SConscript ）



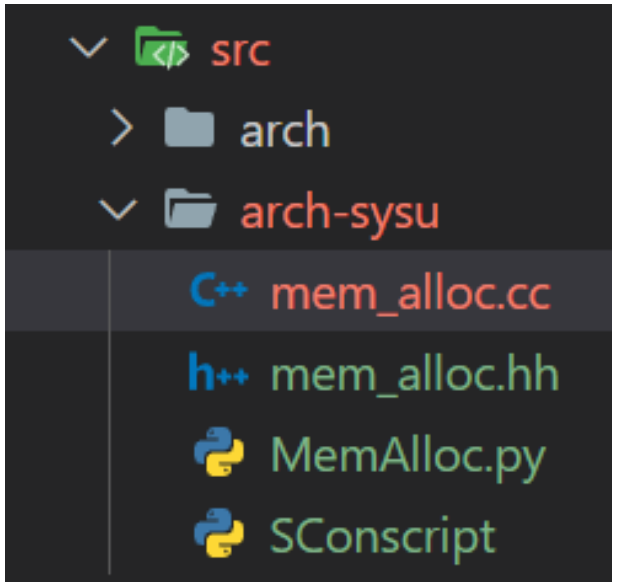

```
1  # 导入构建过程中的所有其它符号（固定写法）
2  Import('*')
3
4  # 声明要添加哪些 SimObject
5  SimObject('MemAlloc.py', sim_objects=['MemAlloc', 'MemRequest'])
6
7  # 添加 C++ 源文件
8  Source('mem_alloc.cc')
9
```

Let’s code!（ SConscript ）

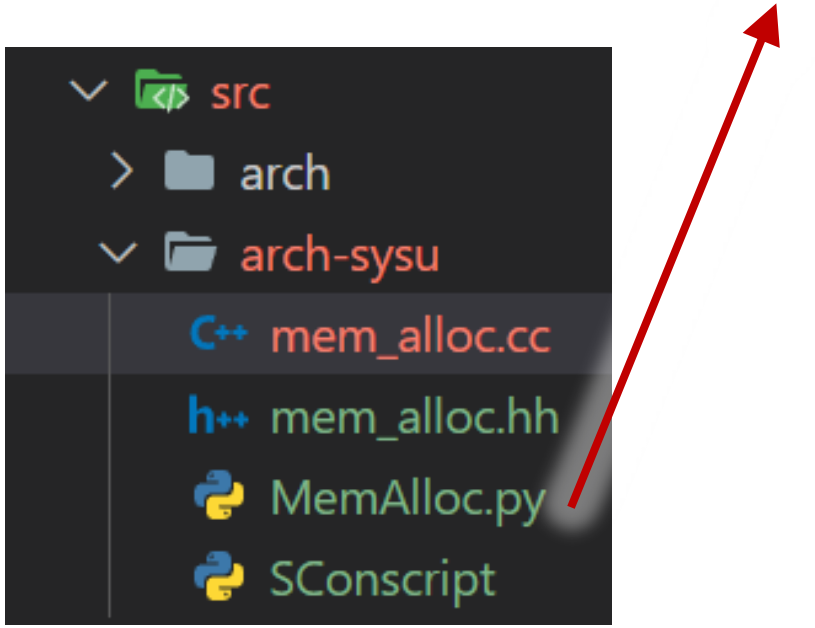



```
1   # 导入构建过程中的所有其它符号（固定写法）
2   Import('*')
3
4   # 声明要添加哪些 SimObject
5   SimObject('MemAlloc.py', sim_objects=['MemAlloc', 'MemRequest'])
6
7   # 添加 C++ 源文件
8   Source('mem_alloc.cc')
9
10  # 添加一个新的调试标志
11  DebugFlag('MemAlloc', "Some discription for this MemAlloc debug flag")
12
```

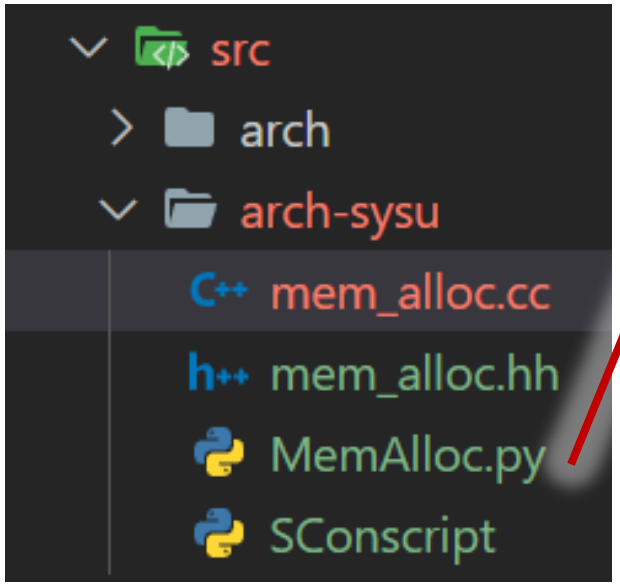
Let’s code!（ MemAlloc.py ）



Let’s code!（ MemAlloc.py ）

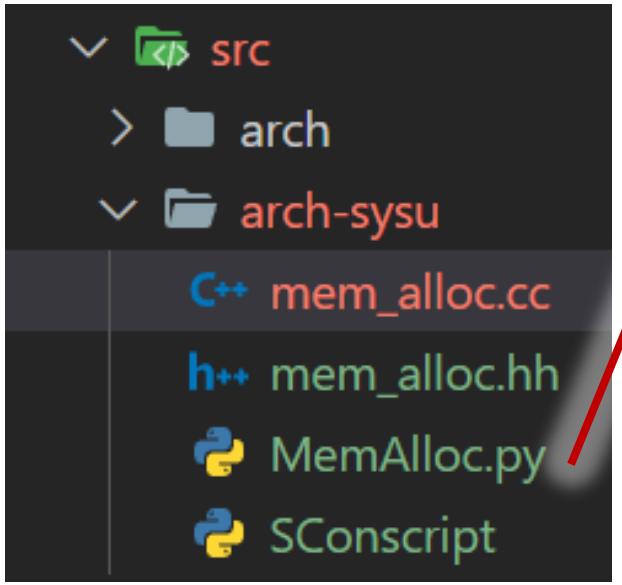


Let’s code!（ MemAlloc.py ）



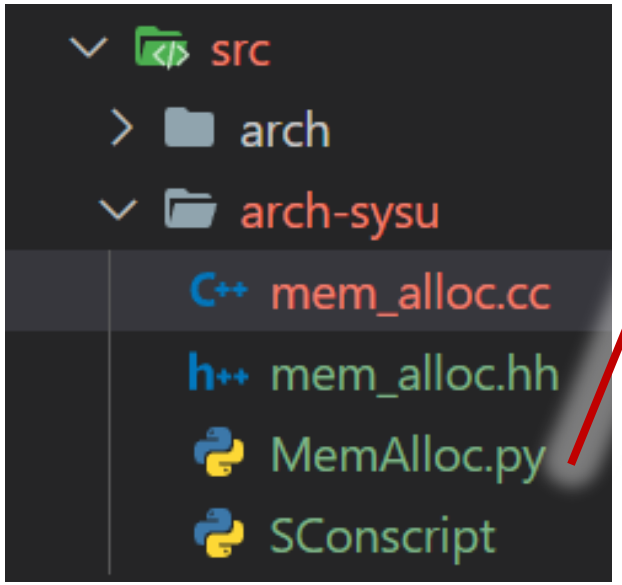

```
1   from m5.params import *
2   from m5.SimObject import SimObject
3
```

Let’s code!（ MemAlloc.py ）



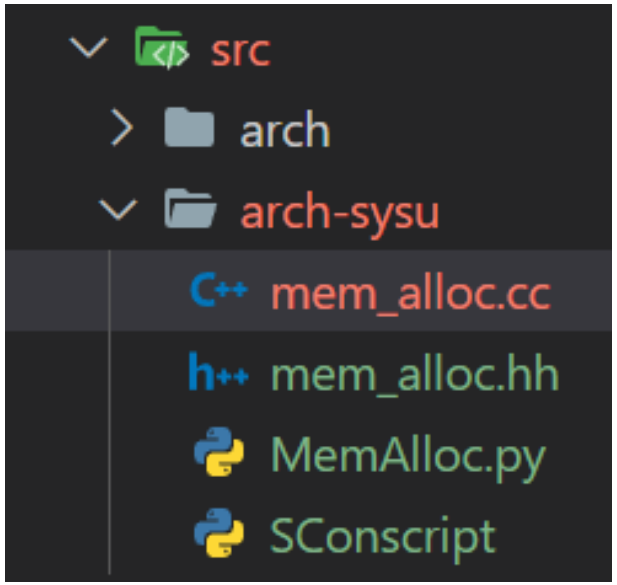

```python
from m5.params import *
from m5.SimObject import SimObject


class MemAlloc(SimObject):
    type = "MemAlloc"  # 习惯上与类名相同
    cxx_header = "arch-sysu/mem_alloc.hh"  # SimObject定义的头文件
    cxx_class = "gem5::MemAlloc"  # C++类名

    # 定义我们需要的参数，等号左边的名称就对应了C++类构造函数中params的成员名
    size = Param.Int("Total avaliable memory size")

```

Let’s code!（ MemAlloc.py ）

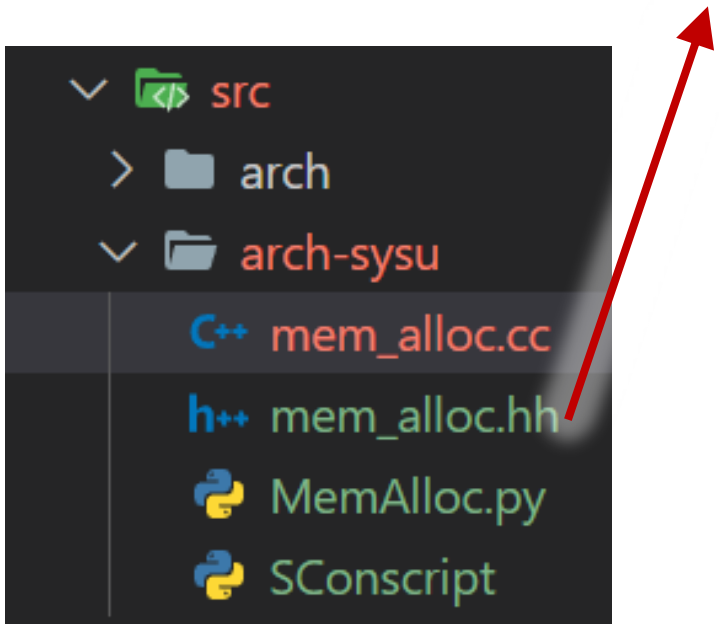



```python
from m5.params import *
from m5.SimObject import SimObject


class MemAlloc(SimObject):
    type = "MemAlloc"  # 习惯上与类名相同
    cxx_header = "arch-sysu/mem_alloc.hh"  # SimObject定义的头文件
    cxx_class = "gem5::MemAlloc"  # C++类名

    # 定义我们需要的参数，等号左边的名称就对应了C++类构造函数中params的成员名
    size = Param.Int("Total avaliable memory size")


class MemRequest(SimObject):
    type = "MemRequest"
    cxx_header = "arch-sysu/mem_alloc.hh"
    cxx_class = "gem5::MemRequest"

    # 参数可以为另一个SimObject对象
    mem_alloc = Param.MemAlloc("Reference to MemAlloc object")

    size = Param.Int("Number of bytes to allocate")
    when = Param.Tick("Time at which the memory allocation is requested")
    life = Param.Tick("Time after which the allocated memory is freed")

```

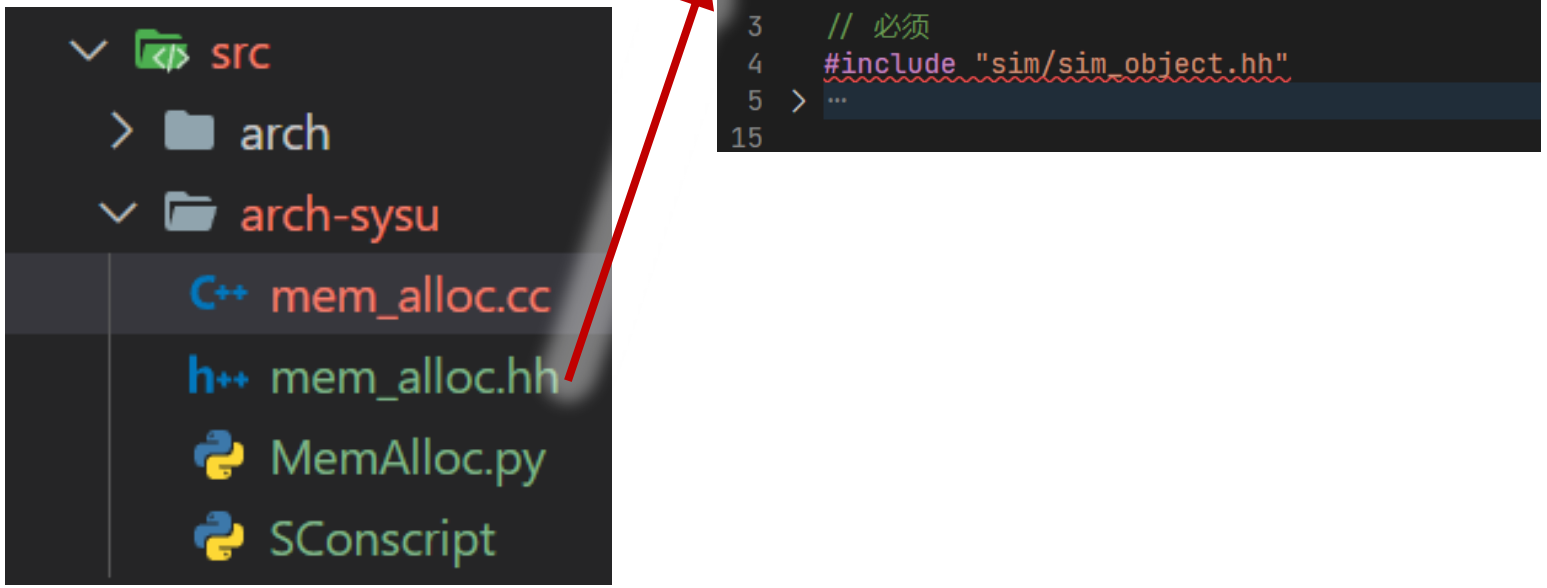
Let’s code!（ mem_alloc.hh mem_alloc.cc ）



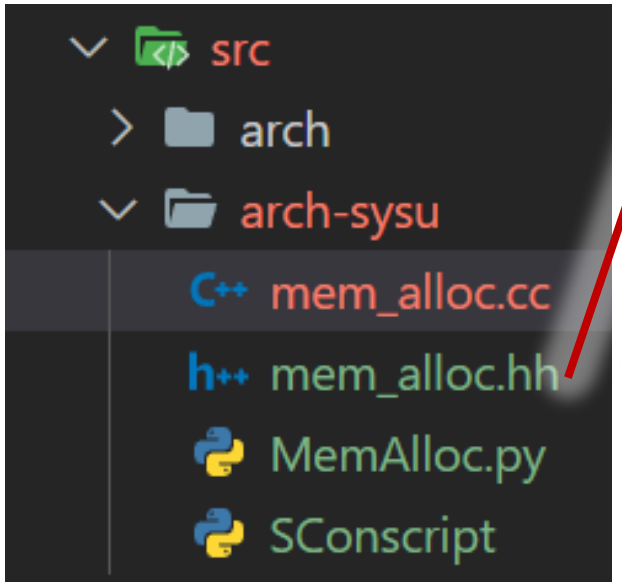
Let’s code!（ mem_alloc.hh mem_alloc.cc ）



Let’s code!（ mem_alloc.hh mem_alloc.cc ）

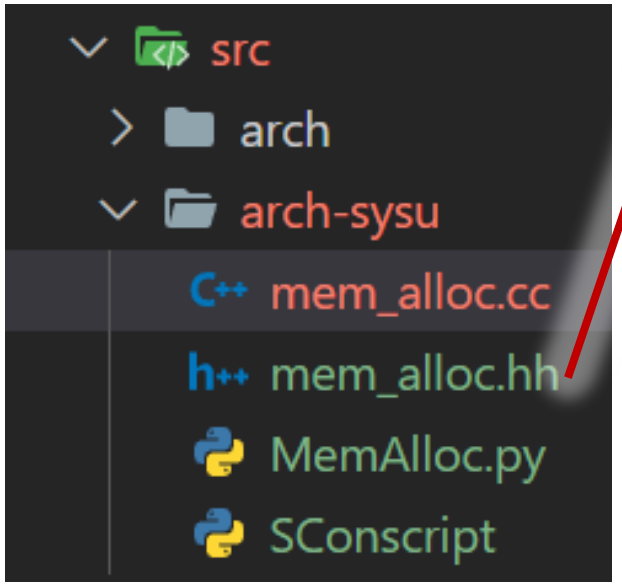


Let’s code!（ mem_alloc.hh mem_alloc.cc ）




```
1   #pragma once
2
3   // 必须
4   #include "sim/sim_object.hh"
5   …
15
16  namespace gem5 {
17
18  /**…
21  class MemAlloc : public SimObject
22  {
23  public:
24    MemAlloc(const MemAllocParams& params);
25
26  public:
27    /**…
30    int alloc(int size);
31
32    /**…
35    bool free(int ptr);
36
37  private:…
53  };
```

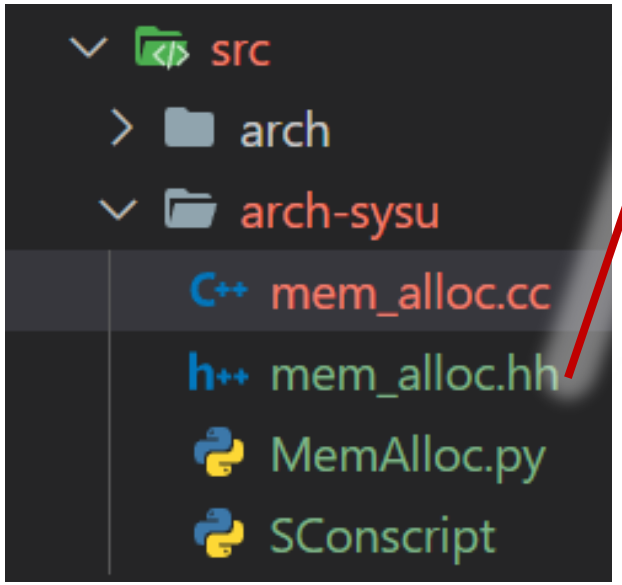
Let’s code!（ mem_alloc.hh mem_alloc.cc ）




```
1   #pragma once
2
3   // 必须
4   #include "sim/sim_object.hh"
5   …
15
16  namespace gem5 {
17
18  /**…
21  class MemAlloc : public SimObject
22  {
23  public:
24    MemAlloc(const MemAllocParams& params);
25
26  public:
27    /**…
30    int alloc(int size);
31
32    /**…
35    bool free(int ptr);
36
37  private:…
53  };
```

```
55  /**…
58  class MemRequest : public SimObject
59  {
60  public:
61    MemRequest(const MemRequestParams& params);
62
63  public:
64    /**…
67    void startup() override;
68
69  private:…
80  };
```

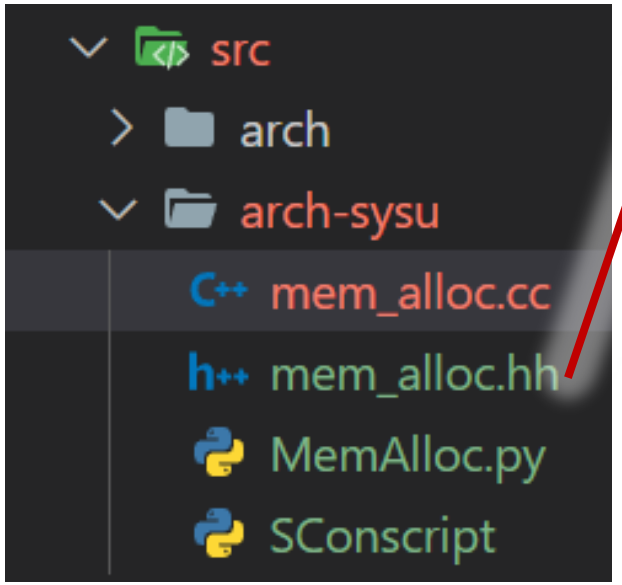
Let’s code!（ mem_alloc.hh mem_alloc.cc ）




```
1   #pragma once
2
3   // 必须
4   #include "sim/sim_object.hh"
5   …
15
16  namespace gem5 {
17
18  /**…
21  class MemAlloc : public SimObject
22  {
23  public:
24    MemAlloc(const MemAllocParams& params);
25
26  public:
27    /**…
30    int alloc(int size);
31
32    /**…
35    bool free(int ptr);
36
37  private:…
53  };
```

```
55  /**…
58  class MemRequest : public SimObject
59  {
60  public:
61    MemRequest(const MemRequestParams& params);
62
63  public:
64    /**…
67    void startup() override;
68
69  private:…
80  };
```

Let’s code!（ mem_alloc.hh mem_alloc.cc ）

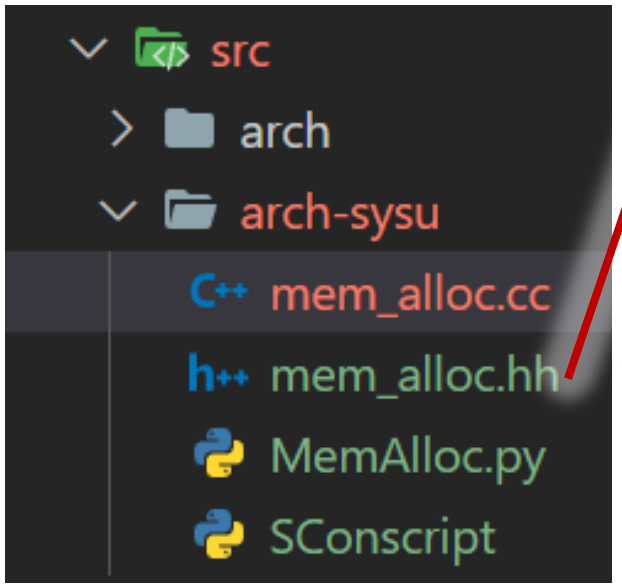



```
1   #pragma once
2
3   // 必须
4   #include "sim/sim_object.hh"
5   …
15
16  namespace gem5 {
17
18  /**…
21  class MemAlloc : public SimObject
22  {
23  public:
24    MemAlloc(const MemAllocParams& params);
25
26  public:
27    /**…
30    int alloc(int size);
31
32    /**…
35    bool free(int ptr);
36
37  private:…
53  };
```

```
55  /**…
58  class MemRequest : public SimObject
59  {
60  public:
61    MemRequest(const MemRequestParams& params);
62
63  public:
64    /**…
67    void startup() override;
68
69  private:…
80  };
```

```
92  void
93  MemRequest::startup()
94  {
95    schedule(_alloc, _when);
96  }
```

Let’s code!（ mem_alloc.hh mem_alloc.cc ）



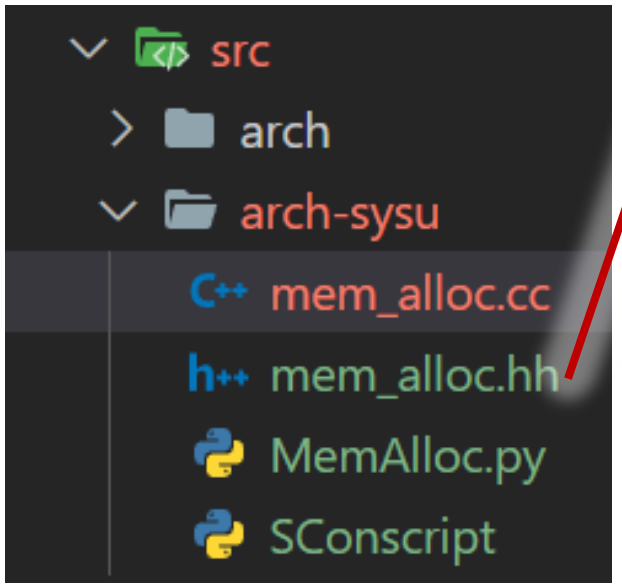

```
#pragma once

// 必须
#include "sim/sim_object.hh"
…

namespace gem5 {

/**…
class MemAlloc : public SimObject
{
public:
  MemAlloc(const MemAllocParams& params);

public:
  /**…
  int alloc(int size);

  /**…
  bool free(int ptr);

private:…
};
```

```
/**…
class MemRequest : public SimObject
{
public:
  MemRequest(const MemRequestParams& params);

public:
  /**…
  void startup() override;

private:…
};
```

```
void
MemRequest::startup()
{
  schedule(_alloc, _when);
}
```

事件
对象

Let’s code!（ mem_alloc.hh mem_alloc.cc ）



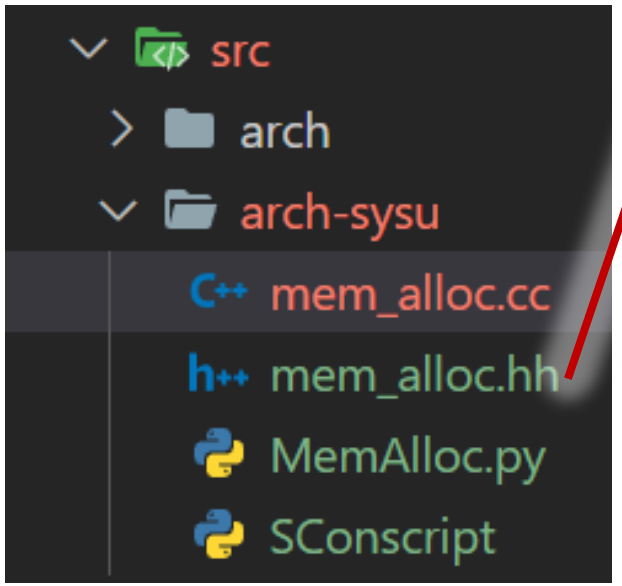

```cpp
#pragma once

// 必须
#include "sim/sim_object.hh"
…

namespace gem5 {

/**…
class MemAlloc : public SimObject
{
public:
  MemAlloc(const MemAllocParams& params);

public:
  /**…
  int alloc(int size);

  /**…
  bool free(int ptr);

private:…
};
```

```cpp
/**…
class MemRequest : public SimObject
{
public:
  MemRequest(const MemRequestParams& params);

public:
  /**…
  void startup() override;

private:…
};
```

```cpp
void
MemRequest::startup()
{
  schedule(_alloc, _when);
}
```

事件对象　发生时刻

Let’s code!（ mem_alloc.hh mem_alloc.cc ）



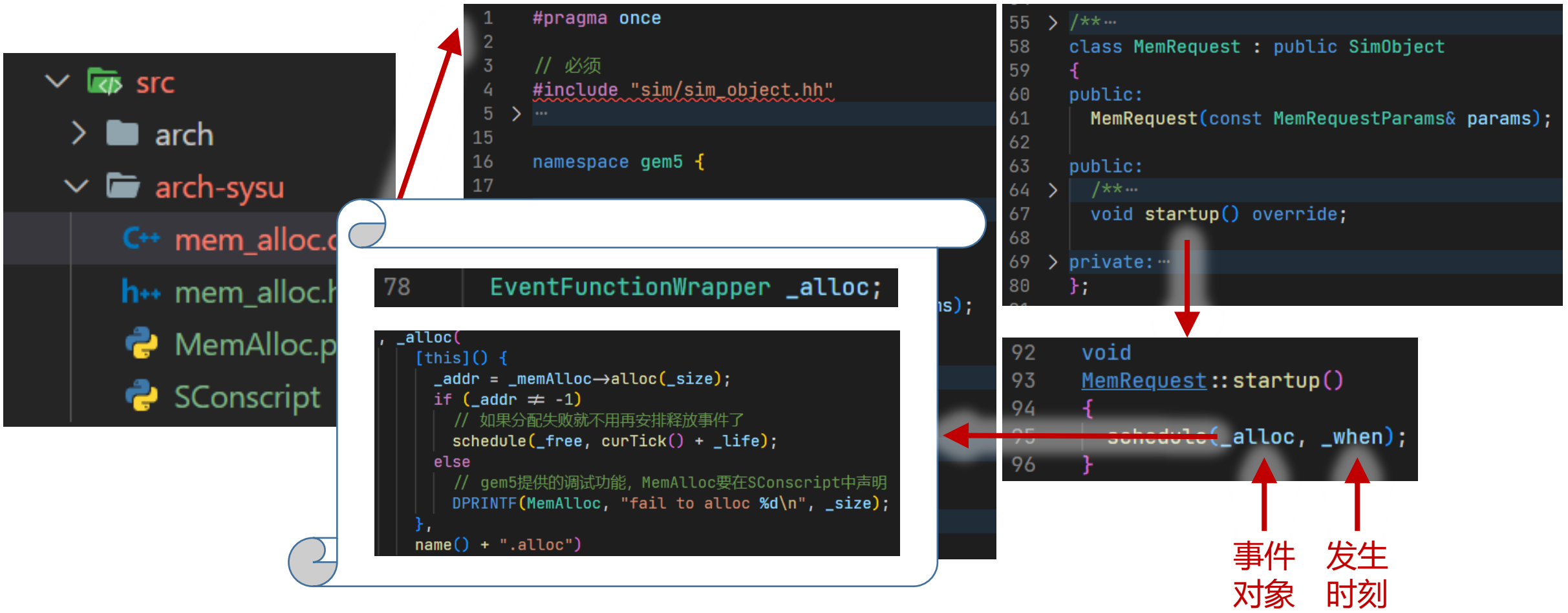

```cpp
#pragma once

// 必须
#include "sim/sim_object.hh"
…

namespace gem5 {

/**…
class MemAlloc : public SimObject
{
public:
  MemAlloc(const MemAllocParams& params);

public:
  /**…
  int alloc(int size);

  /**…
  bool free(int ptr);

private:…
};
```

```cpp
/**…
class MemRequest : public SimObject
{
public:
  MemRequest(const MemRequestParams& params);

public:
  /**…
  void startup() override;

private:…
};
```

```cpp
void
MemRequest::startup()
{
  schedule(_alloc, _when);
}
```

事件对象　发生时刻

Let’s code!（ mem_alloc.hh mem_alloc.cc ）



Let’s code!（仿真脚本）

Let’s code!（仿真脚本）

```python
import m5
from m5.objects import *

# 创建内存分配器
mem_alloc = MemAlloc()
mem_alloc.size = 10
```

Let’s code!（仿真脚本）

```python
import m5
from m5.objects import *

# 创建内存分配器
mem_alloc = MemAlloc()
mem_alloc.size = 10

reqdata = [
    (3, 0, 5),
    (4, 1, 8),
    (5, 7, 2),
]
```

Let’s code!（仿真脚本）

```python
import m5
from m5.objects import *

# 创建内存分配器
mem_alloc = MemAlloc()
mem_alloc.size = 10

reqdata = [
    (3, 0, 5),
    (4, 1, 8),
    (5, 7, 2),
]

# 所有创建的SimObject对象必须关联到一个root对象上
root = Root(full_system=False)
```

Let’s code!（仿真脚本）

```python
import m5
from m5.objects import *

# 创建内存分配器
mem_alloc = MemAlloc()
mem_alloc.size = 10

reqdata = [
    (3, 0, 5),
    (4, 1, 8),
    (5, 7, 2),
]

# 所有创建的SimObject对象必须关联到一个root对象上
root = Root(full_system=False)

# 此处只是一个示例：利用Python的分支、循环等语法，我们可以很容易地描述庞大与复杂的体系结构
for i, (size, when, life) in enumerate(reqdata):
    # 创建每个内存请求
    mem_request = MemRequest()
    mem_request.mem_alloc = mem_alloc
    mem_request.size = size
    mem_request.when = when
    mem_request.life = life

    # 关联到root对象上
    setattr(root, f"req_{i}", mem_request)
```

Let’s code!（仿真脚本）

```python
import m5
from m5.objects import *

# 创建内存分配器
mem_alloc = MemAlloc()
mem_alloc.size = 10

reqdata = [
    (3, 0, 5),
    (4, 1, 8),
    (5, 7, 2),
]

# 所有创建的SimObject对象必须关联到一个root对象上
root = Root(full_system=False)

# 此处只是一个示例：利用Python的分支、循环等语法，我们可以很容易地描述庞大与复杂的体系结构
for i, (size, when, life) in enumerate(reqdata):
    # 创建每个内存请求
    mem_request = MemRequest()
    mem_request.mem_alloc = mem_alloc
    mem_request.size = size
    mem_request.when = when
    mem_request.life = life

    # 关联到root对象上
    setattr(root, f"req_{i}", mem_request)


m5.instantiate()  # 前面所有创建的Python对象在这一行才会真正创建对应的C++类实例

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

Let’s run!

```
$ ./build/X86/gem5.opt --debug-flags=MemAlloc arch-sysu/mem_alloc_1.py
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 22.0.0.2
gem5 compiled Sep 19 2022 09:51:40
gem5 started Sep 19 2022 10:58:36
gem5 executing on GUYUHAO-UBUNTU, pid 175941
command line: ./build/X86/gem5.opt --debug-flags=MemAlloc arch-sysu/mem_alloc_1.py

Global frequency set at 1000000000000 ticks per second
build/X86/base/statistics.hh:278: warn: `allocFailureCounter` is a legacy stat. Legacy stat is a stat that does not belong to any st
atistics::Group. Legacy stat is deprecated.
Beginning simulation!
build/X86/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
      7: req_2: fail to alloc 5
Exiting @ tick 18446744073709551615 because simulate() limit reached
```

m5out/stats.txt:

```
---------- Begin Simulation Statistics ----------
simSeconds                                   18446744.073710                       # Number of seconds simulated (Second)
simTicks                                     18446744073709551616                       # Number of ticks simulated (Tick)
finalTick                                    18446744073709551616                       # Number of ticks from beginning of simulation
simFreq                                      1000000000000                       # The number of ticks per simulated second ((Tick/Sec
hostSeconds                                          0.00                       # Real time elapsed on the host (Second)
hostTickRate                                 4728418859009177131417                       # The number of ticks simulated per host se
hostMemory                                       114576                       # Number of bytes of host memory used (Byte)
allocFailureCounter                                   1                       # count of failures to allocate memory (Unspecified)

---------- End Simulation Statistics   ----------
```

使用gem5中已有的组件

gem5 Boot Camp 2022

https://www.gem5.org/events/boot-camp-2022

我怎么知道有哪些类，它们都叫什么，怎么用？

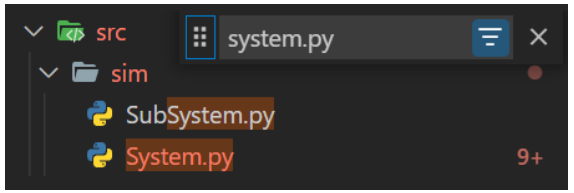
在v21.2之前，只能查源代码：

```
# import the m5 (gem5) library created when gem5
import m5
# import all of the SimObjects
from m5.objects import *

# create the system we are going to simulate
system = System()

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

# Set up the system
system.mem_mode = 'timing'               # Use ti
system.mem_ranges = [AddrRange('512MB')] # Create

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

我怎么知道有哪些类，它们都叫什么，怎么用？

在v21.2之前，只能查源代码：

```python
# import the m5 (gem5) library created when gem5
import m5
# import all of the SimObjects
from m5.objects import *

# create the system we are going to simulate
system = System()

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

# Set up the system
system.mem_mode = 'timing'               # Use ti
system.mem_ranges = [AddrRange('512MB')] # Create

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

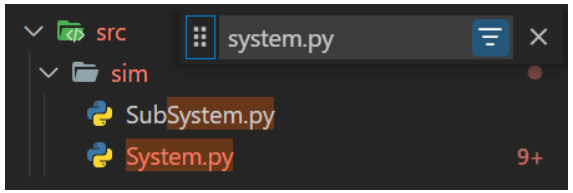



```python
class System(SimObject):
    type = 'System'
    cxx_header = "sim/system.hh"
    cxx_class = 'gem5::System'

    system_port = RequestPort("System port")

    cxx_exports = [
        PyBindMethod("getMemoryMode"),
        PyBindMethod("setMemoryMode"),
    ]

    memories = VectorParam.AbstractMemory(Self.all,
                                          "All memories in the sy
    mem_mode = Param.MemoryMode('atomic', "The mode the memory sy

    thermal_model = Param.ThermalModel(NULL, "Thermal model")
    thermal_components = VectorParam.SimObject([],
            "A collection of all thermal components in the system
```

我怎么知道有哪些类，它们都叫什么，怎么用？

在v21.2之前，只能查源代码：

```python
# import the m5 (gem5) library created when gem5
import m5
# import all of the SimObjects
from m5.objects import *

# create the system we are going to simulate
system = System()

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

# Set up the system
system.mem_mode = 'timing'               # Use ti
system.mem_ranges = [AddrRange('512MB')] # Create

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



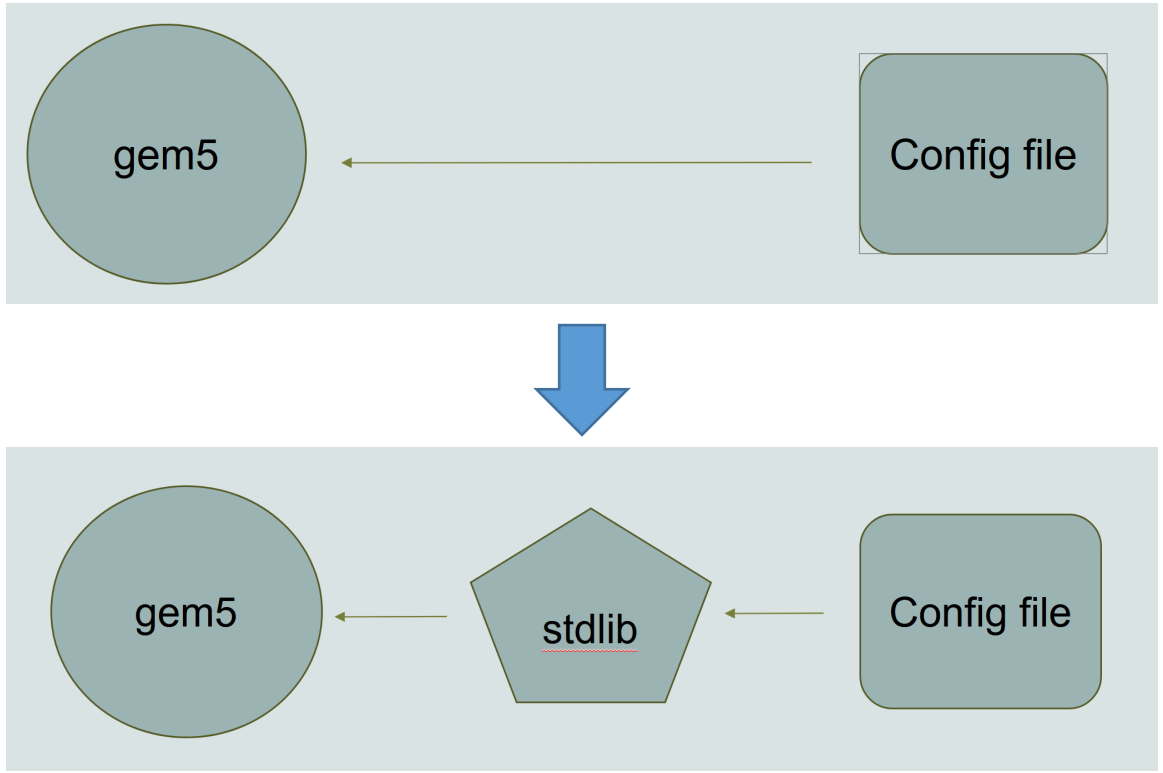

```python
class System(SimObject):
    type = 'System'
    cxx_header = "sim/system.hh"
    cxx_class = 'gem5::System'

    system_port = RequestPort("System port")

    cxx_exports = [
        PyBindMethod("getMemoryMode"),
        PyBindMethod("setMemoryMode"),
    ]

    memories = VectorParam.AbstractMemory(Self.all,
                                          "All memories in the sy
    mem_mode = Param.MemoryMode('atomic', "The mode the memory sy

    thermal_model = Param.ThermalModel(NULL, "Thermal model")
    thermal_components = VectorParam.SimObject([],
            "A collection of all thermal components in the system
```

```
$ ./build/X86/gem5.opt --list-sim-objects > help.txt
```

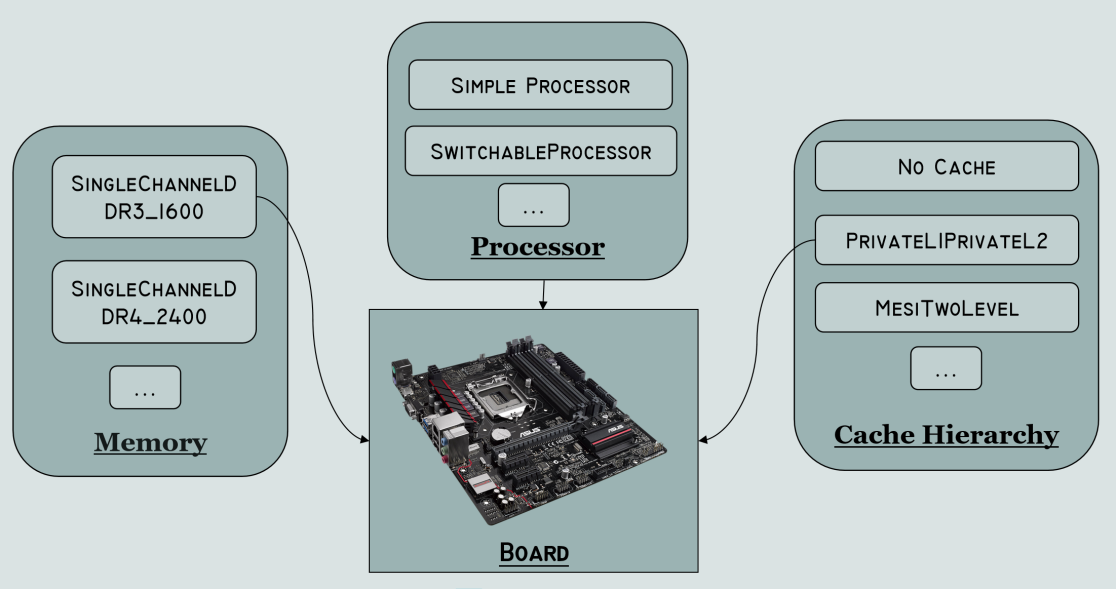
```
        desc: Source System

    target
        desc: Target System


System
    auto_unlink_shared_backstore
        desc: Automatically remove the shmem segment file upon
              destruction. This is used only if shared_backstore is non-
              empty.

    cache_line_size
        default: 64
        desc: Cache line size in bytes

    eventq_index
        default: Parent.eventq_index
        desc: Event Queue Index

    exit_on_work_items
        desc: Exit from the simulation loop when encountering work item
              annotations.

    init_param
        desc: numerical value to pass into simulator

    m5ops_base
        desc: Base of the 64KiB PA range used for memory-mapped m5ops.
              to 0 to disable.
```

gem5 standard library

- v21.2之后，gem5引入了标准库来简化配置文件的编写：

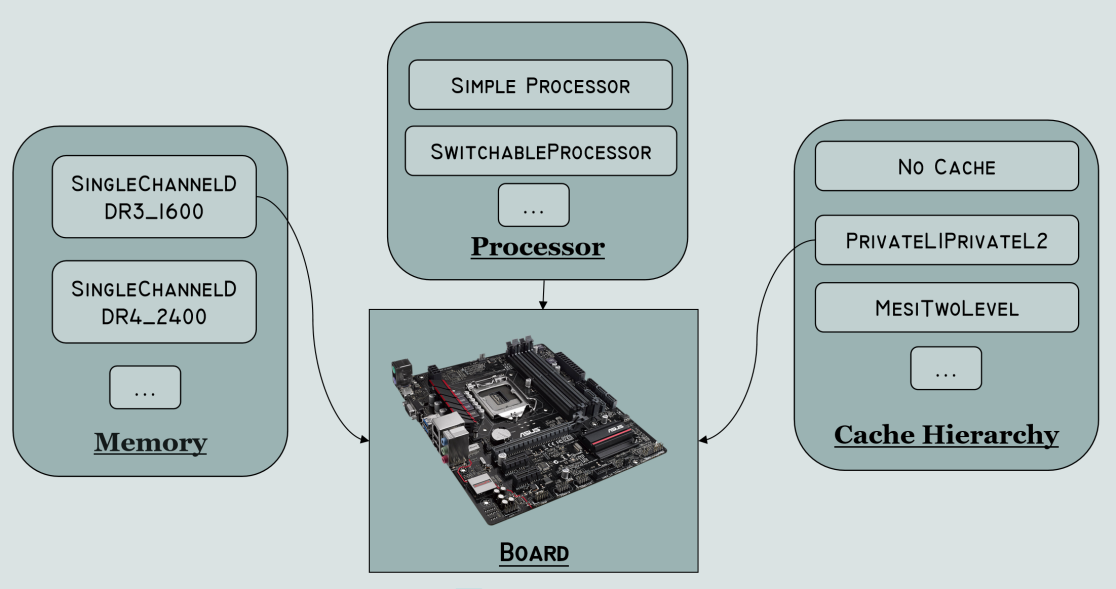


- 使用标准库后，写配置文件就像攒机一样简单：




```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator


# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)

#Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)

# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

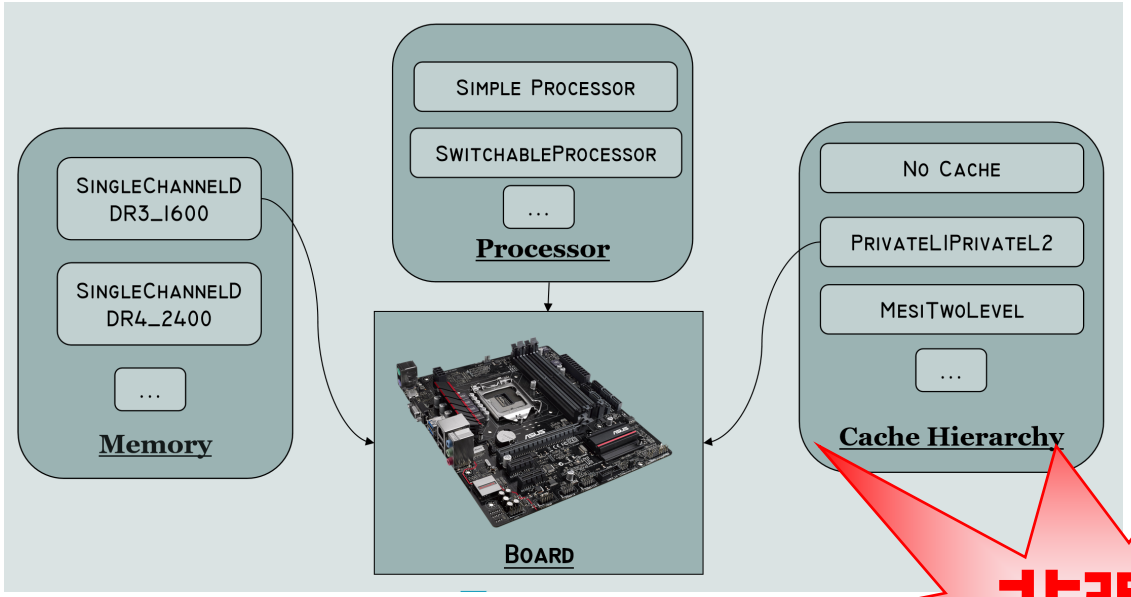
- 使用标准库后，写配置文件就像攒机一样简单：




```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator


# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)

#Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)

# Set the workload.
binary = Resource("x86-hello64-static")
board.set_se_binary_workload(binary)

# Setup the Simulator and run the simulation.
simulator = Simulator(board=board)
simulator.run()
```

https://www.gem5.org/documentation/gem5-stdlib/overview

- 使用标准库后，写配置文件就像攒机一样简单：




```python
from gem5.components.boards.simple_board import SimpleBoard
from gem5.components.cachehierarchies.classic.no_cache import NoCache
from gem5.components.memory.single_channel import SingleChannelDDR3_1600
from gem5.components.processors.simple_processor import SimpleProcessor
from gem5.components.processors.cpu_types import CPUTypes
from gem5.resources.resource import Resource
from gem5.simulate.simulator import Simulator


# Obtain the components.
cache_hierarchy = NoCache()
memory = SingleChannelDDR3_1600("1GiB")
processor = SimpleProcessor(cpu_type=CPUTypes.ATOMIC, num_cores=1)

#Add them to the board.
board = SimpleBoard(
    clk_freq="3GHz",
    processor=processor,
    memory=memory,
    cache_hierarchy=cache_hierarchy,
)

et the workload.
nary = Resource("x86-hello64-static")
set_se_binary_workload(binary)

up the Simulator and run the simulation.
lator = Simulator(board=board)
r.run()
```

非强制
仅推荐

https://www.gem5.org/documentation/gem5-stdlib/overview

实验一作业

- 向MemAlloc中添加功能：
  - 添加一个MemFreeReq类，这个类有一个addr和when参数，它会在when时刻释放addr指向的内存段
  - 添加调试输出：每当内存段被错误释放时，cout << 错误地址
  - 添加统计量：整个模拟过程中，内存碎片（不连续的空闲空间）最多时候的数量

- 添加并使用MinorCPU

  编写一个素数筛程序并模拟运行

- 给gem5添加X87 FSUBR指令

  只需要修改十几行代码即可