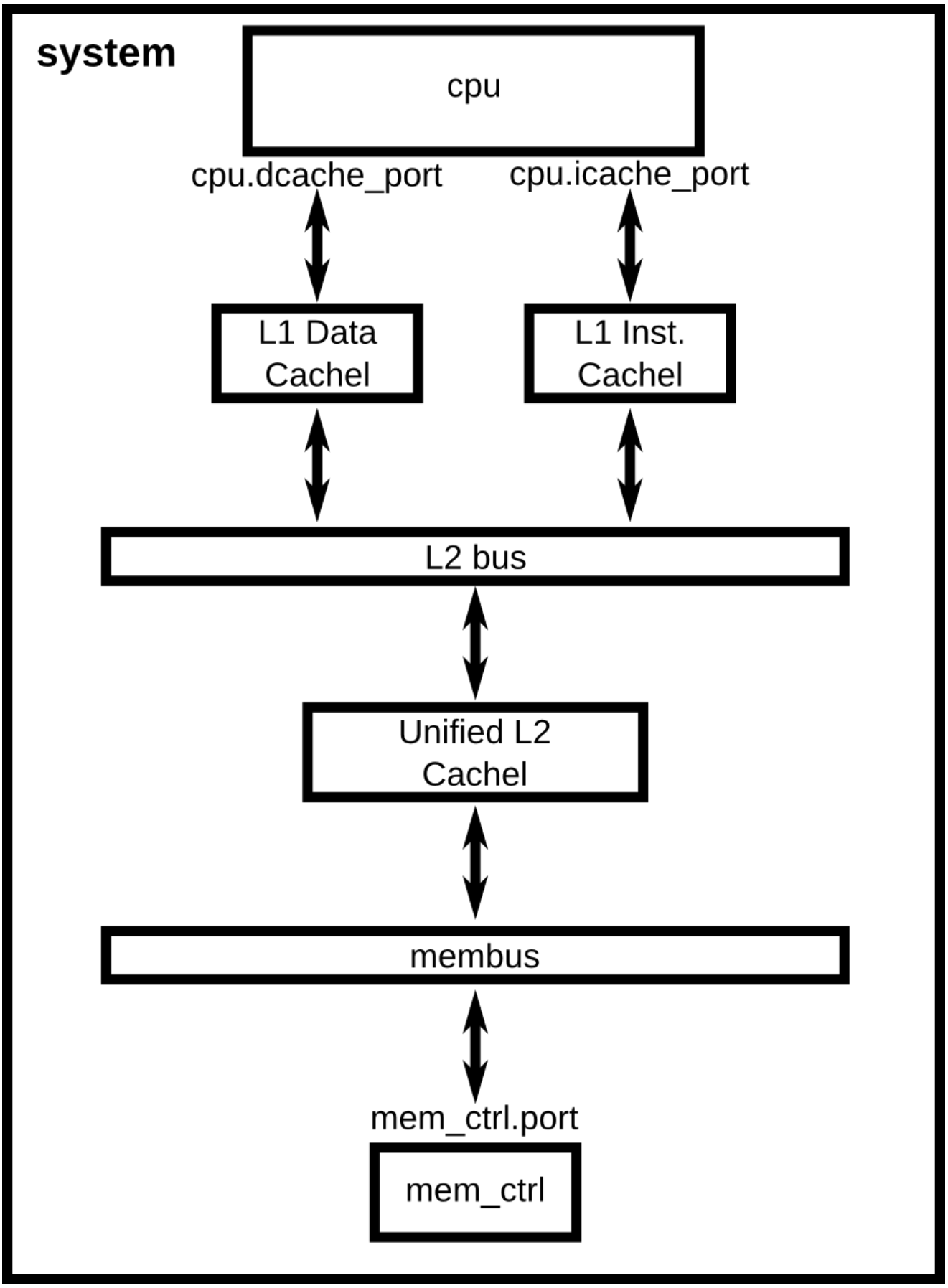
The gem5 simulator guide

Add Cache to the configuration

- We are in a single CPU mode, so we don't care about cache coherence
- Two models of cache:
  - Classic cache
  - Ruby(Good for coherence simulation)



Create Cache Objects

We should create a Cache object based on the `BaseCache` for each level.
All configs at: `configs/learning_gem5/part1`

``

For example:

```
class L1Cache(Cache):
        assoc = 2
        tag_latency = 2
        data_latency = 2
        response_latency = 2
        mshrs = 4
        tgts_per_mshr = 20
```

Setup The System

Step 1(Create L1 Caches)

```
system.cpu.icache = L1ICache(args)
system.cpu.dcache = L1DCache(args)
```

Step 2(Connect L1 Cache to CPU)

```
system.cpu.icache.connectCPU(system.cpu)
system.cpu.dcache.connectCPU(system.cpu)
```

Step 3(Create L2 Bus and Connect L1 Cache)

```
system.l2bus = L2XBar()

# Hook the CPU ports up to the l2bus
system.cpu.icache.connectBus(system.l2bus)
system.cpu.dcache.connectBus(system.l2bus)
```

Step4(Create L2 Cache and Connect to L2 Bus)

```
system.l2cache = L2Cache(args)
system.l2cache.connectCPUSideBus(system.l2bus)
```

Step5(Create System Bus and connect L2 Cache)

```
system.membus = SystemXBar()

# Connect the L2 cache to the membus
system.l2cache.connectMemSideBus(system.membus)
```

Test The Cache

Compare the following codes from locality aspect. Which one is better? Which type of locality is violated in the worse one?(Spatial or Temporal)

```c
#define LIMIT 128

int a[LIMIT][LIMIT];

int main() {

    for (int i = 0; i < LIMIT; i++)
        for (int j = 0; j < LIMIT; j++)
            a[i][j] = i * j;

    long long sum = 0;
    for (int i = 0; i < LIMIT; i++)
        for (int j = 0; j < LIMIT; j++)
            sum += a[j][i];

    return 0;
}
```

```c
#define LIMIT 128

int a[LIMIT][LIMIT];

int main() {

    for (int i = 0; i < LIMIT; i++)
        for (int j = 0; j < LIMIT; j++)
            a[i][j] = i * j;

    long long sum = 0;
    for (int i = 0; i < LIMIT; i++)
        for (int j = 0; j < LIMIT; j++)
            sum += a[i][j];

    return 0;
}
```

Step1(Compile)

```
gcc binary/good_locality.c -o binary/good_locality
gcc binary/bad_locality.c -o binary/bad_locality
```

Step2(Simulate the Good and get miss rate)

```
./build/X86/gem5.opt configs/workshop/cache/two_level.py
binary/good_locality
grep 'system.cpu.dcache.overallMissRate::total' m5out/stats.txt
```

Step3(Simulate the Bad and get miss rate)

```
./build/X86/gem5.opt configs/workshop/cache/two_level.py
binary/bad_locality
grep 'system.cpu.dcache.overallMissRate::total' m5out/stats.txt
```

No tangible difference? Why?

Cache size matters. L1DCache size must have a significant difference against the array size.
To observe the difference we set cache size to `1kB`.

```
./build/X86/gem5.opt configs/workshop/cache/two_level.py
binary/bad_locality --l1d_size=1kB
grep 'system.cpu.dcache.overallMissRate::total' m5out/stats.txt
```

```
./build/X86/gem5.opt configs/workshop/cache/two_level.py
binary/good_locality --l1d_size=1kB
grep 'system.cpu.dcache.overallMissRate::total' m5out/stats.txt
```

Replacement Policy

For this section we use `configs/deprecated/example/se.py`. This config has a good set of options. Here we want to use cache related ones.

`./build/X86/gem5.opt configs/deprecated/example/se.py -h`

```
  --caches
  --l2cache
  --num-dirs NUM_DIRS
  --num-l2caches NUM_L2CACHES
  --num-l3caches NUM_L3CACHES
  --l1d_size L1D_SIZE
  --l1i_size L1I_SIZE
  --l2_size L2_SIZE
  --l3_size L3_SIZE
  --l1d_assoc L1D_ASSOC
  --l1i_assoc L1I_ASSOC
```

```
  --l2_assoc L2_ASSOC
  --l3_assoc L3_ASSOC
  --cacheline_size CACHELINE_SIZE
```

We want to make the cache replacement policy configurable.

Step 1

Add the following line to `Configs/common/ObjectList`

```
repl_list = ObjectList(getattr(m5.objects, 'BaseReplacementPolicy', None))
```

Step 2

Add some new options in `configs/common/Options.py`

```
    parser.add_argument("--l1d_repl", default="LRURP",
                        choices=ObjectList.repl_list.get_names(),
                        help="replacement policy for l1")

    parser.add_argument("--l2_repl", default="LRURP",
                       choices=ObjectList.repl_list.get_names(),
                       help = "replacement policy for l2")
```

Step 3

Change `configs/common/CacheConfig.py`

```
replacement_policy_attr = f"{level}_repl"
    if hasattr(options, replacement_policy_attr):
        opts["replacement_policy"] =
ObjectList.repl_list.get(getattr(options, replacement_policy_attr))()
```

Now we have the following arguments.

```
  --l1d_repl
{BIPRP,BRRIPRP,DRRIPRP,DuelingRP,FIFORP,LFURP,LIPRP,LRURP,MRURP,NRURP,RRIP
RP,RandomRP,SHiPMemRP,SHiPPCRP,SecondChanceRP,TreePLRURP,WeightedLRURP}
                        replacement policy for l1
  --l2_repl
{BIPRP,BRRIPRP,DRRIPRP,DuelingRP,FIFORP,LFURP,LIPRP,LRURP,MRURP,NRURP,RRIP
RP,RandomRP,SHiPMemRP,SHiPPCRP,SecondChanceRP,TreePLRURP,WeightedLRURP}
```

```
                        replacement policy for l2
```

Simulation

You can set cache related stuff using these commmands

```
./build/X86/gem5.opt configs/deprecated/example/se.py -c binary/mean --
caches --l1d_repl LRURP --l1d_size 1kB
```