menjadi kunci penting dari perancangan sistem yang matang. Dalam menentukan perancangan ini digambarkan dalam bentuk diagram blok.

5. **Implementasi Sistem**

   Tahap ini dilakukan simulasi menggunakan *Gem5* dengan mode *full-system* sehingga dapat diketahui simulasi sistem tersebut dengan mempertimbangkan *OS* yang digunakan beserta aplikasi yang dijalankan di *OS* itu.

6. **Pengujian Sistem**

   Tahap ini memastikan sistem dapat dijalankan sesuai dengan hipotesa dan tujuan yang ingin dicapai. Pengujian ini dilakukan dengan skenario yang telah ditentukan sehingga memilki alur yang jelas

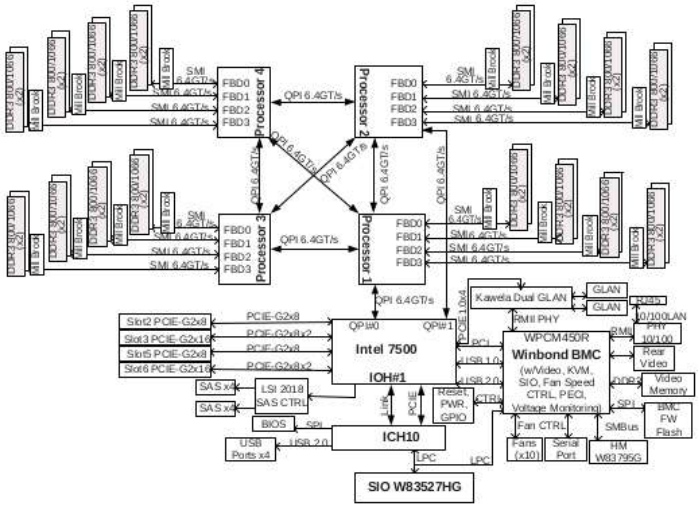
7. **Penyusunan Buku TA**

   Penyusunan buku ini sekaligus memaparkan hasil pengujian serta menarik kesimpulan dari sistem yang telah dirancang

2. Tinjauan Pustaka

2.1. NUMA

NUMA merupakan arsitektur komputer yang menerapkan memori local pada setiap *prosessor* sehingga *cache* mampu mengakses proses yang ada di dalam memori dengan waktu yang relatif lebih cepat. Adapun *prosessor* dapat mengakses memori di prosessor lain dengan bantuan protokol tertentu yaitu *cache coherence protocol*. NUMA yang memanfaatkan *cache coherence protocol* dinamakan ccNUMA[11].



Gambar 2-1 Arsitektur NUMA[15]

2.2. Cache Coherence

Cache coherence merupakan teknik yang digunakan oleh *multiprosessor* agar dapat menjaga keutuhan dan kebenaran data ketika *cache* saling berbagi. Adapun penerapannya pada model SMP *(Shared Memori Prosessor)* hingga NUMA *(Non-Uniform Memori Access).Cache coherence* diatur oleh sebuah aturan atau protokol yang sering disebut dengan *cache coherence protocol*[11].

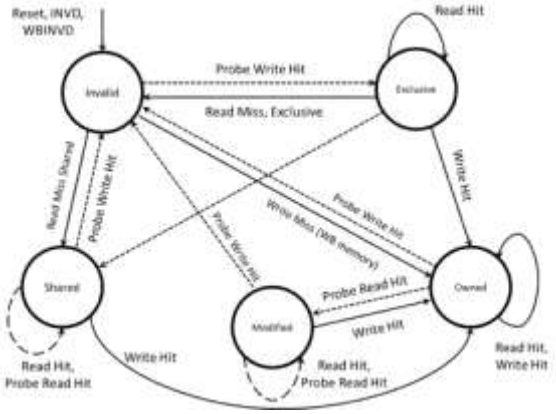
2.2.1. Cache Coherence Protocol

Cache coherence protocol berguna untuk membantu *cache coherence* tetap menjalankan fungsinya dengan baik. Protokol ini diimplementasikan sebagai tatat aturan bagi hardware komputer agar pengaksesan data yang dilakukan oleh *cache* menuju pada lokasi memori yang seharusnya. *Cache coherence protocol* dibagi menjadi dua macam *snooping* dan *directory-based*. Saat ini, *directory-based* yang sering digunakan, salah satu contohnya *MOESI*[11].

2.2.1.1. MOESI

Protokol ini berbasis directory-based yang merupakan pengembangan dari protokol MESI. Penambahan state O yang berarti Owned menjadi bukti dari protokol sebelumnya. Adapun detail dari masing-masing statenya [5]:

a. *Invalid* : *cache block* pada *state* ini tidak meyimpan salinan data. Salinan data terdapat pada memori atau *cache* yang lain[5].
b. *Exclusive*: *cache block* pada *state* ini menyimpan salinan data baru sekaligus benar. Salinan data terdapat pada memori utama dan tidak ada prosesor lain yang memegang salinan datanya[5].
c. *Shared*: *cache block* pada *state* ini menyimpan salinan data yang baru dan benar, tapi prosesor lain mungkin juga mempunyai salinan data .
d. *Modified* : *cache block* pada *state* ini menyimpan salinan data yang baru dan benar. Salinan data di memori utama yang kurang tepat dan hanya satu prosessor yang mempunyai salinan datanya[5].
e. *Owned* : *cache block* pada *state* ini menyimpan salinan data yang baru dan benar. Adapun pada state "*shared*"semua prosesor wajib memegang data[5]



Gambar 2-2 MOESI *Coherence Protocol State* [8]

2.2.1.2. MOESI CMP Token

Salah satu jenis dari *MOESI* yang menggunakan token sebagai cara untuk suatu cache blok menulis data. Adapun token ini telah ditetapkan pada masing-masing cache yang ada pada setiap *prosessor* sehingga tidak akan pernah berubah. Susunan *2 level cache* menjadi model utama dari protokol ini[5][8]

2.3. Gem5 *Simulator*

gem5 merupakan simulator yang mampu memodelkan arsitektur komputer layakya mikroarsitektur komputer[6]. Secara garis besar, simulator ini perpaduan dari M5 dan GEMS. Keduanya memiliki fitur masing-masing yang diharapkan dapat menjadi simulator yang bagus. M5 memiliki kelebihan di antaranya mampu dilakukan konfigurasi, menyediakan beberapa jenis *ISA (X86, ALPHA, MIPS,*

dsb), serta pemodelan yang berbeda dari *CPU (atomic, timing,* dan *detailed).* Sedangkan GEMS fitur yang mampu menggambarkan alur kerja cache coherence protocol dan interkoneksi.

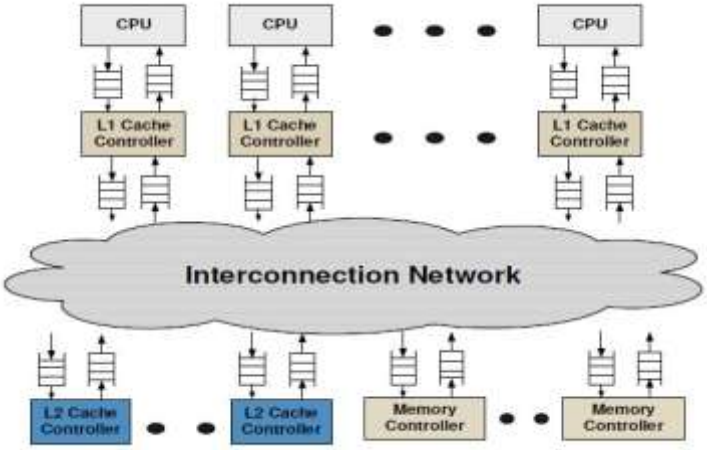
Gem5 menyediakan dua mode simulasi yaitu *System-call Emulation* dan *Full System*. Perbedaannya terletak pada penggunaan OS dan keseluruhan perangkat ketika simulasi dijalankan. Terkait dengan model memori yang ditawarkan terdapat dua macam yaitu *Classic dan Ruby.Classic* menawarkan cepat , kemudahan konfigurasi, tetapi akurasi belum bisa dijamin. Lain halnya, dengan *Ruby* yang lebih fleksibel karena mampu dikoneksikan dengan *cache coherence protocol* serta fitur interkoneksi[4]

2.3.1.1. SLICC

SLICC adalah bahasa khusus untuk menerjemahkan *cache coherence protocol*. SLICC dibangun dengan menggunakan *compiler* berbasis *C++* yang mampu dikombinasikan dengan *Ruby* dan*HTML*[6].

2.3.1.2. Interconnection Network

Fitur dalam simulator gem5 yang mampu menggambarkan simulasi dari alurpacket data terkirim pada topologi suatu jaringan. Dalam hal ini, gem5 menyediakan beberapa topologi yang memiliki karakteristik masing-masing, diantaranya *Pt2pt, MeshDir, Mesh, Torus,* dan *Crossbar*. Adapun penyusunan topologi interkoneksi dapat menggunakan simulator eksternal bernama *TOPAZ*[9].



Gambar 2-3 *Interconnection Network* pada Ruby[9]

2.4. TOPAZ

Simulator interkoneksi yang diadaptasi dari *Syscosis*. Simulator ini dapat memberikan mekanisme kerja paket ketika di-*inject* ke dalam jaringan dengan kecepatan dan presisi yang berbeda. TOPAZ dapat berjalan pada OS berbasis UNIX dengan menggunakan compiler C++. Penggabungan dengan gem5 akan membantu dalam melakukan simulasi jaringan dengan penataan yang lebih rapi berdasarkan parameter yang tersedia pada Ruby.py di Gem5[1].