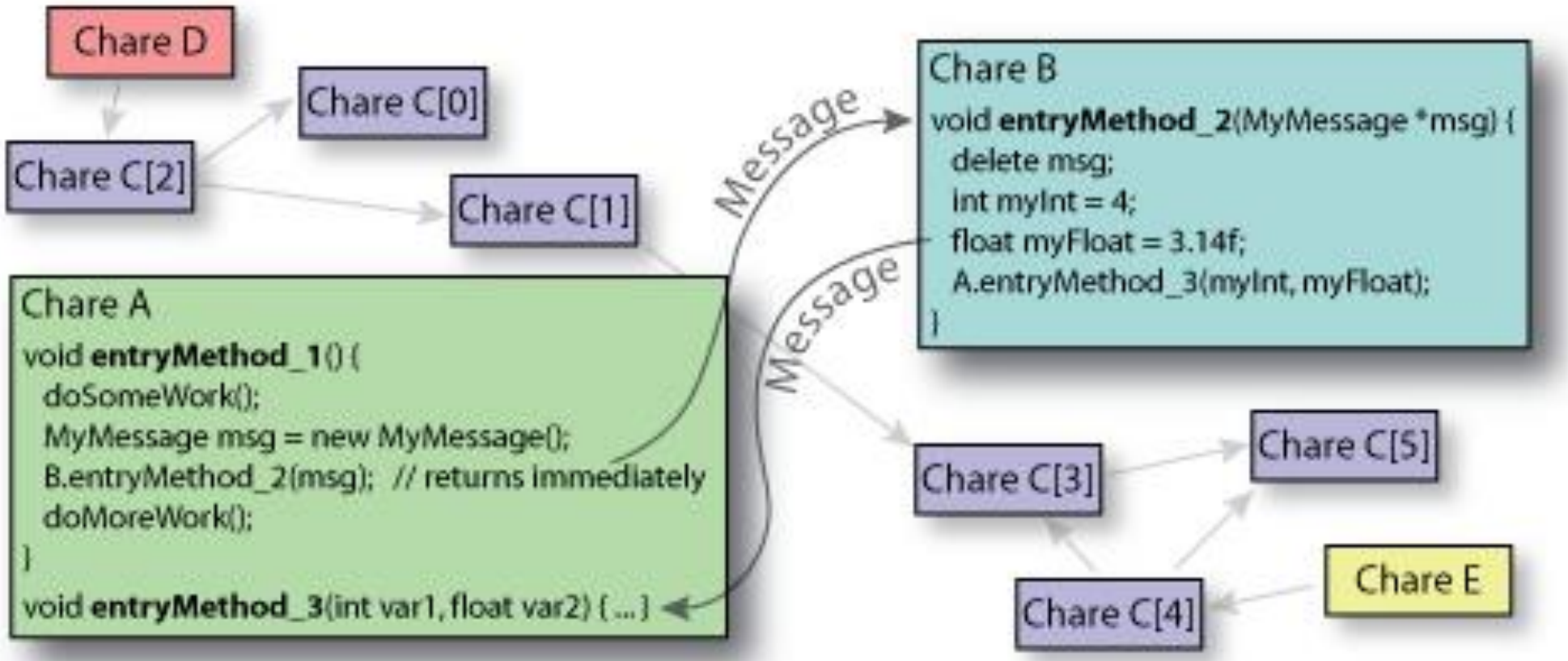
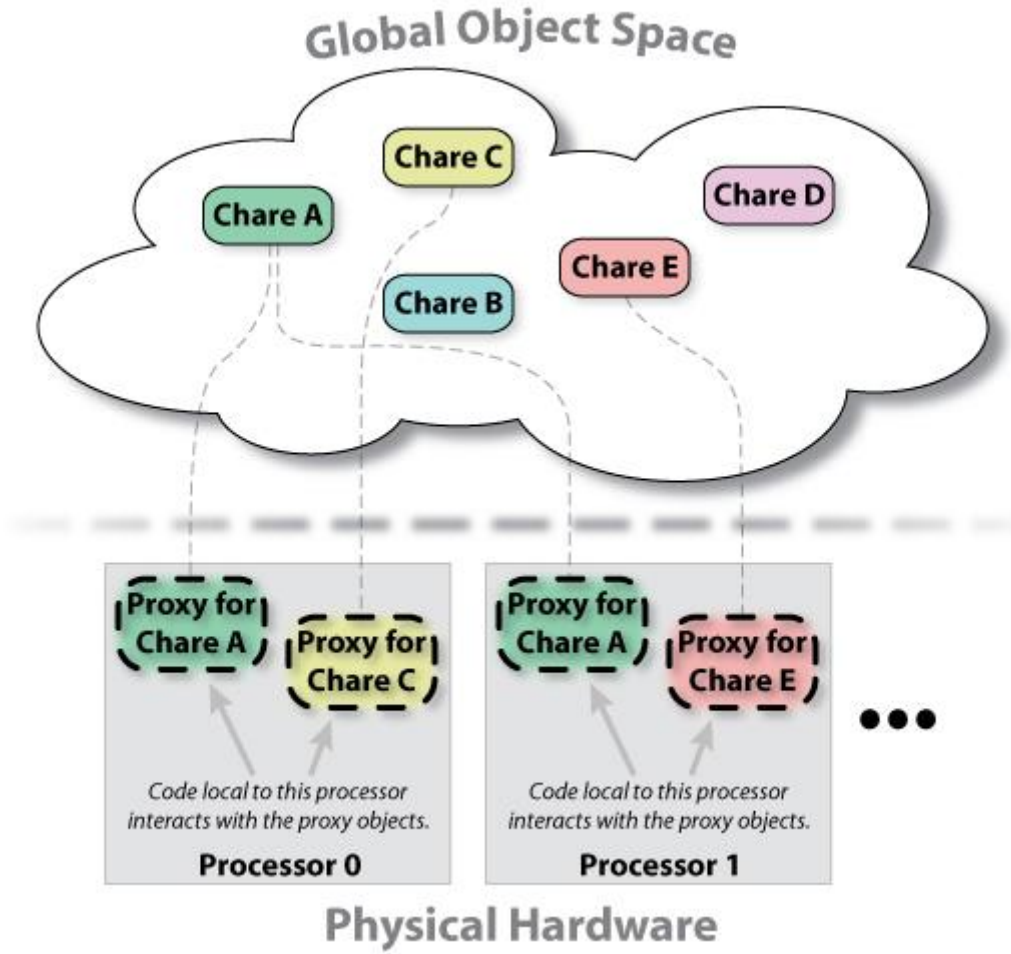
Designing Charm++ Program

Geng(Leo) Liu

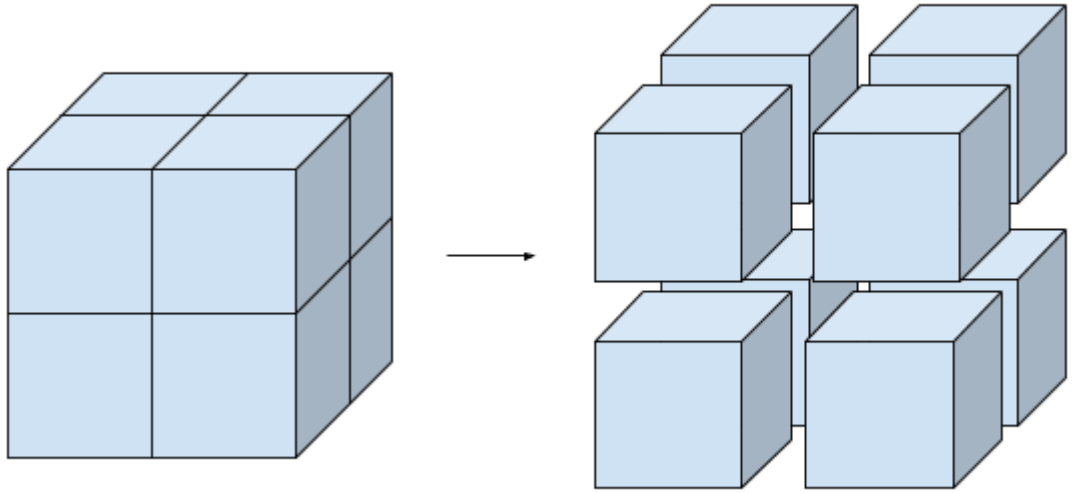
Chare


Chare D
Chare C[0]
Chare C[2]
Chare C[1]
Chare A
void entryMethod_1() {
doSomeWork();
MyMessage msg = new MyMessage();
B.entryMethod_2(msg); // returns immediately
doMoreWork();
}
void entryMethod_3(int var1, float var2) { ... }
Message
Message
Chare B
void entryMethod_2(MyMessage *msg) {
delete msg;
int myInt = 4;
float myFloat = 3.14f;
A.entryMethod_3(myInt, myFloat);
}
Chare C[3]
Chare C[5]
Chare C[4]
Chare E

Chare


Global Object Space
Chare C
Chare A
Chare D
Chare E
Chare B
Proxy for Chare A
Proxy for Chare C
Proxy for Chare A
Proxy for Chare E
Code local to this processor interacts with the proxy objects.
Processor 0
Code local to this processor interacts with the proxy objects.
Processor 1
Physical Hardware

Over-Decomposition



Distribution of Chares

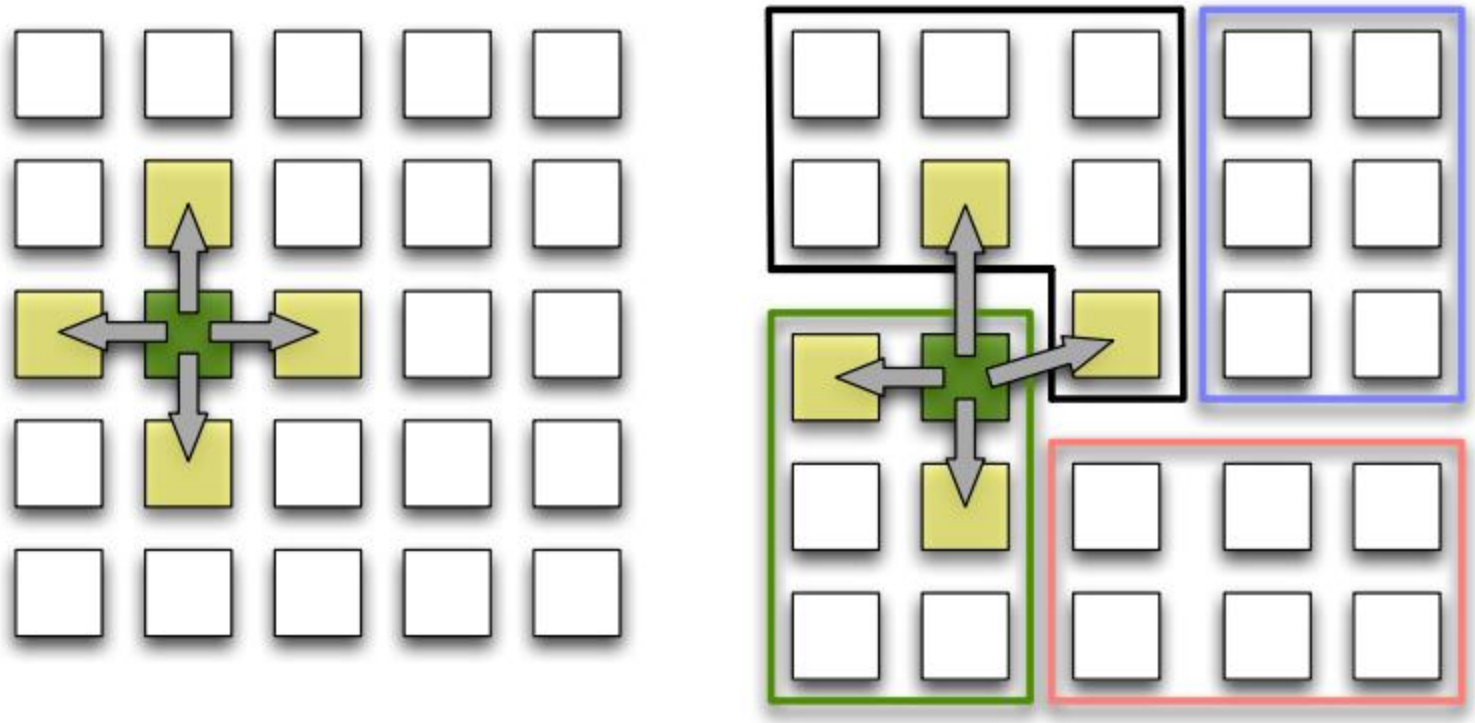


FIGURE 2.1: Chares for a 2D Stencil computation and their mapping to physical processors

Migrating Chares, Load Balancing

- Pack-and-unpack (PUP)
- Send to another node as message

Fault Tolerance

- Checkpoints (disk based)
- True “fault-tolerance” – in memory double checkpointing
  - Restart

Multiphysics Modules

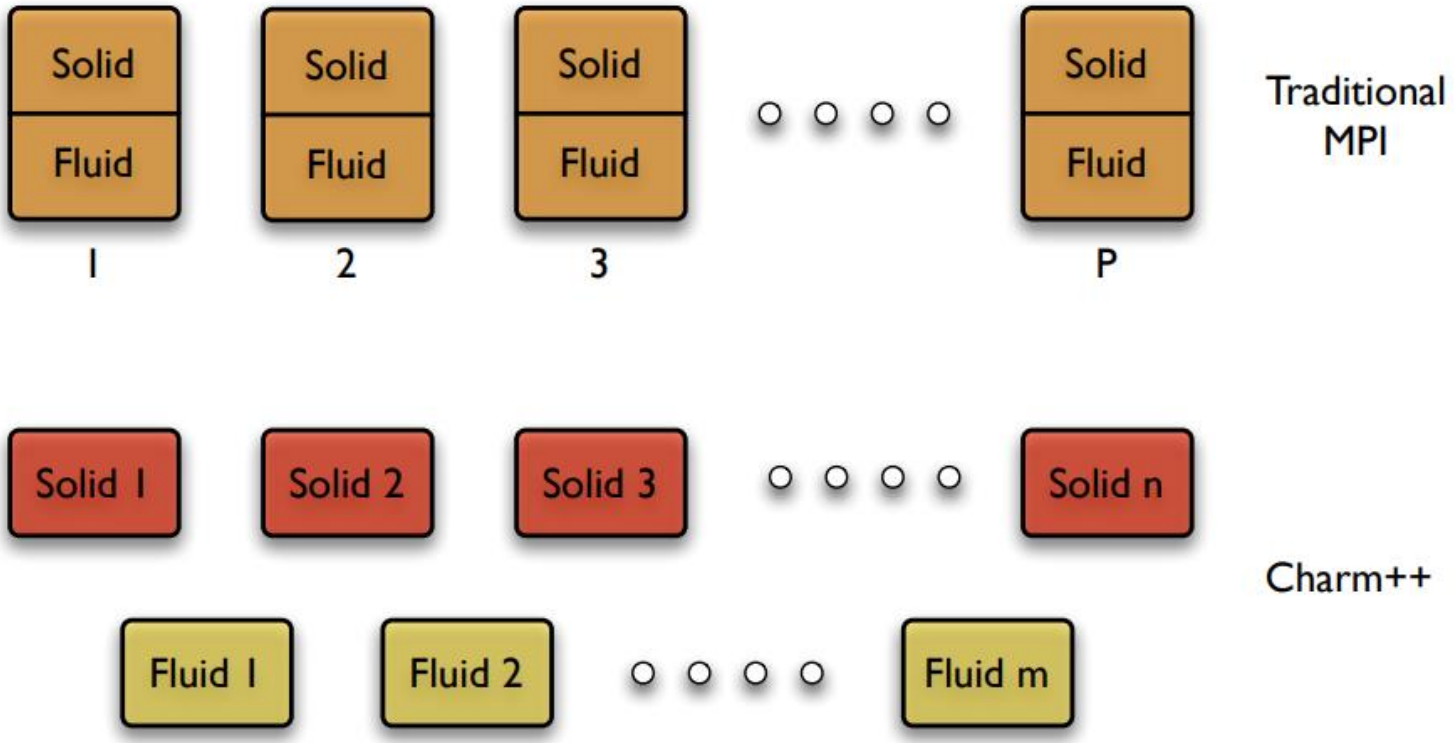
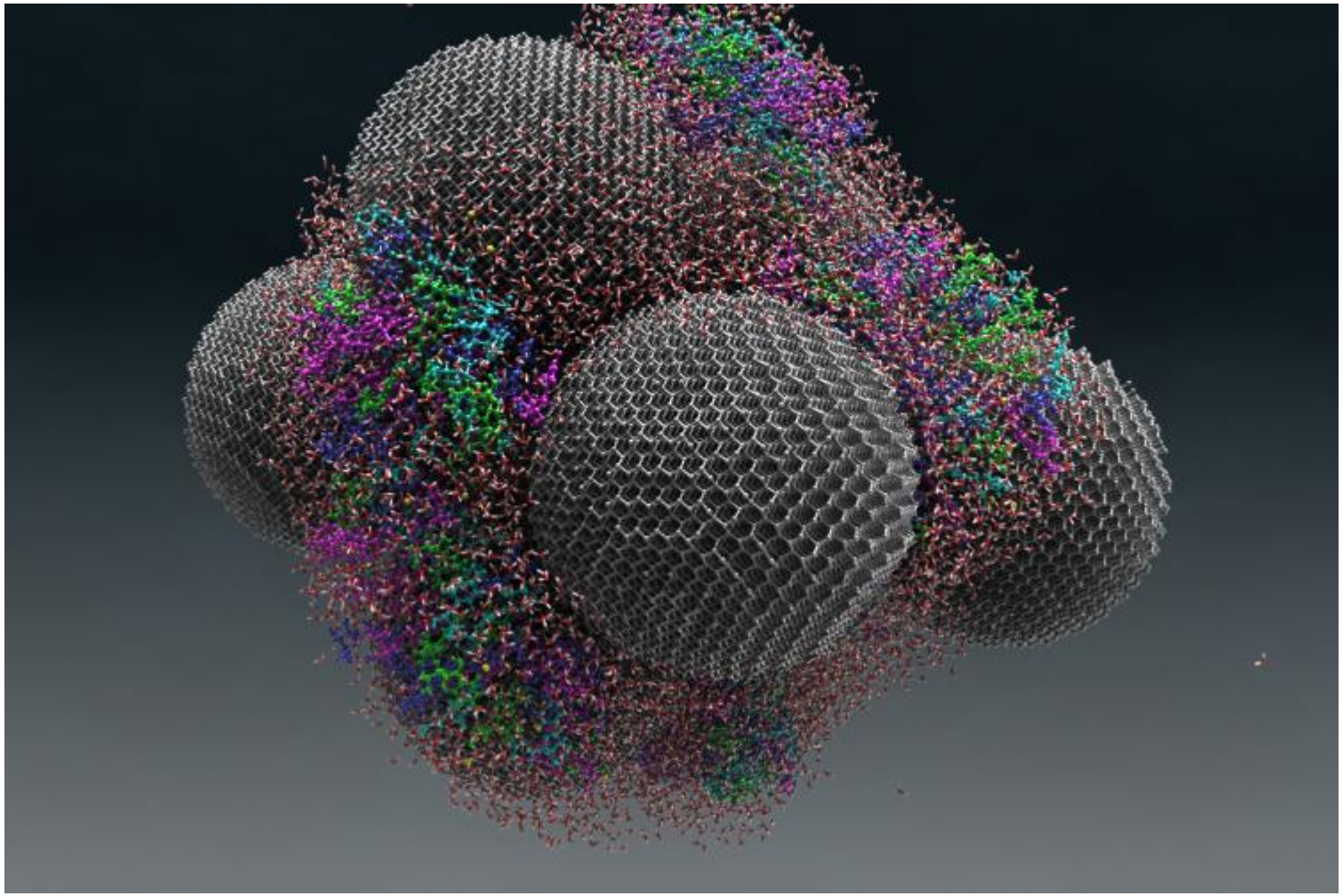


FIGURE 2.2: Decomposition in traditional MPI versus CHARM++

NAMD



NAMD

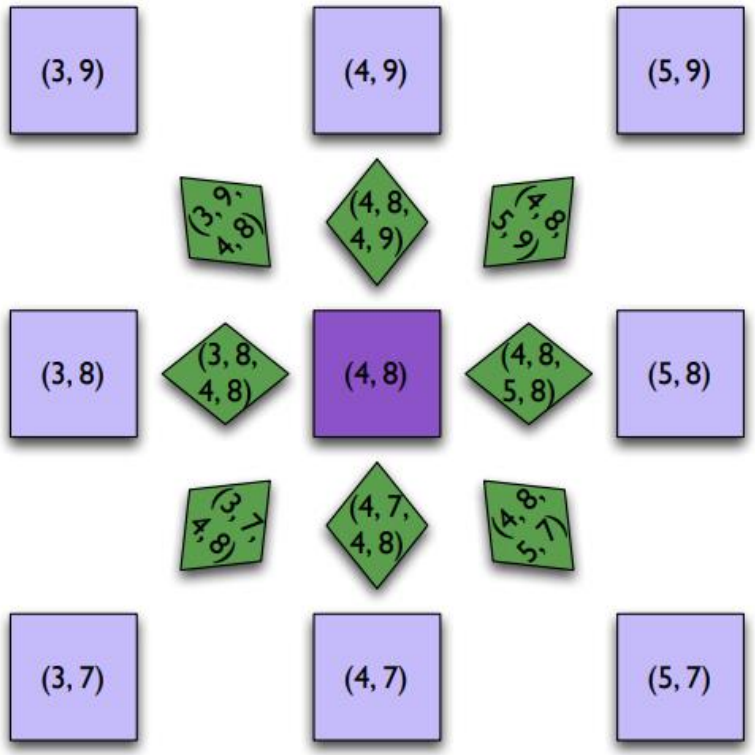


FIGURE 2.3: Use of interaction objects for force calculations in LeanMD

SAMR: Chare Arrays with Dynamic insertion and Flexible Indices

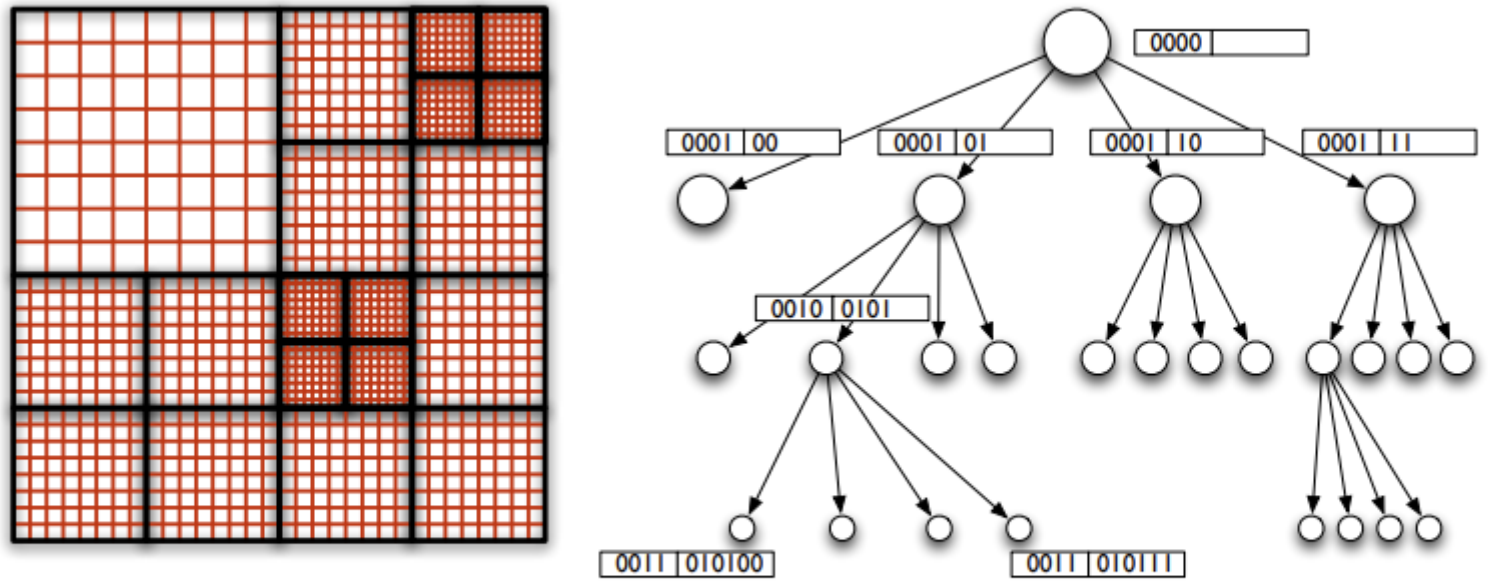


FIGURE 2.6: BitVector Indexing of Chares for AMR